Artificial Intelligence & Expert Systems (CT-361)
Lab 08: Real/Fake Currency note classification

Objectives

The objective of this experiment to get familiar with following machine learning algorithms

- Support Vector Machine
- Perceptron
- The k-nearest neighbors (KNN) algorithm

Tools Required

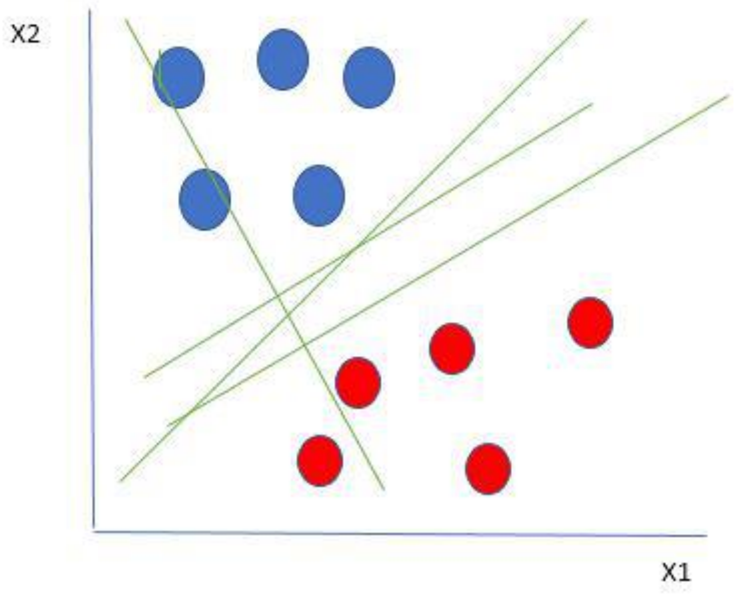
Jupyter IDE

Course Coordinator
– Course Instructor
– Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Introduction

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. While it can be applied to regression problems, SVM is best suited for classification tasks. The primary objective of the SVM algorithm is to identify the optimal hyperplane in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as support vectors, is maximized.

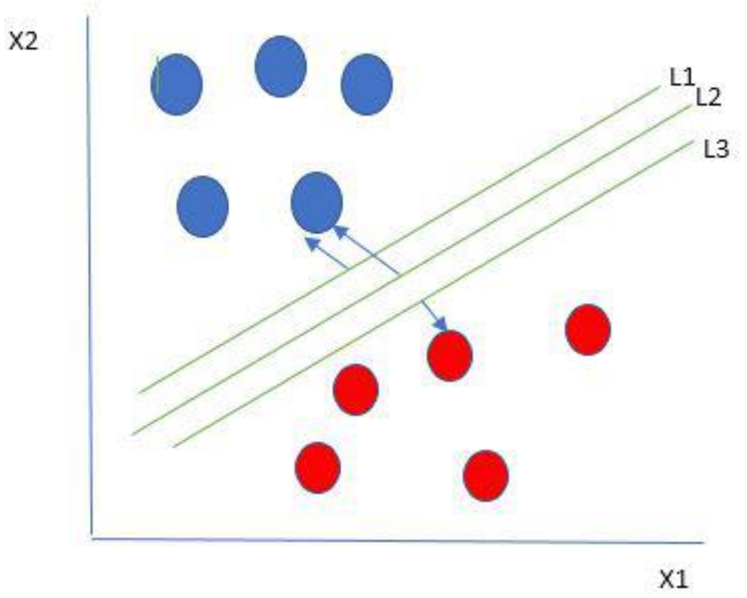
The dimension of the hyperplane depends on the number of features. For instance, if there are two input features, the hyperplane is simply a line, and if there are three input features, the hyperplane becomes a 2-D plane. As the number of features increases beyond three, the complexity of visualizing the hyperplane also increases.
Consider two independent variables, x1 and x2, and one dependent variable represented as either a blue circle or a red circle.In this scenario, the hyperplane is a line because we are working with two features (x1 and x2).There are multiple lines (or hyperplanes) that can separate the data points. The challenge is to determine the best hyperplane that maximizes the separation margin between the red and blue circles.



Linearly Separable Data points

From the figure above it’s very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or do a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points?

One reasonable choice for the best hyperplane in a Support Vector Machine (SVM) is the one that maximizes the separation margin between the two classes. The maximum-margin hyperplane, also referred to as the hard margin, is selected based on maximizing the distance between the hyperplane and the nearest data point on each side.



Multiple hyperplanes separate the data from two classes

The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning method employed to tackle classification and regression problems. Evelyn Fix and Joseph Hodges developed this algorithm in 1951, which was subsequently expanded by Thomas Cover. The article explores the fundamentals, workings, and implementation of the KNN algorithm.

What is Perceptron?

The Perceptron is one of the simplest artificial neural network architectures, introduced by Frank Rosenblatt in 1957. It is primarily used for binary classification. Perceptron is a type of neural network that performs binary classification that maps input features to an output decision, usually classifying data into one of two categories, such as 0 or 1.

A Perceptron is composed of key components that work together to process information and make predictions.

- **Input Features:** The perceptron takes multiple input features, each representing a characteristic of the input data.
- **Weights:** Each input feature is assigned a weight that determines its influence on the output. These weights are adjusted during training to find the optimal values.
- **Summation Function:** The perceptron calculates the weighted sum of its inputs, combining them with their respective weights.

- **Activation Function:** The weighted sum is passed through the **Heaviside step function**, comparing it to a threshold to produce a binary output (0 or 1).
- **Output:** The final output is determined by the activation function, often used for **binary classification** tasks.
- **Bias:** The bias term helps the perceptron make adjustments independent of the input, improving its flexibility in learning.
- **Learning Algorithm:** The perceptron adjusts its weights and bias using a learning algorithm, such as the Perceptron Learning Rule, to minimize prediction errors.

These components enable the perceptron to learn from data and make predictions. While a single perceptron can handle simple binary classification, complex tasks require multiple perceptrons organized into layers, forming a neural network.

How does Perceptron work?

A weight is assigned to each input node of a perceptron, indicating the importance of that input in determining the output. The Perceptron's output is calculated as a weighted sum of the inputs, which is then passed through an activation function to decide whether the Perceptron will fire.

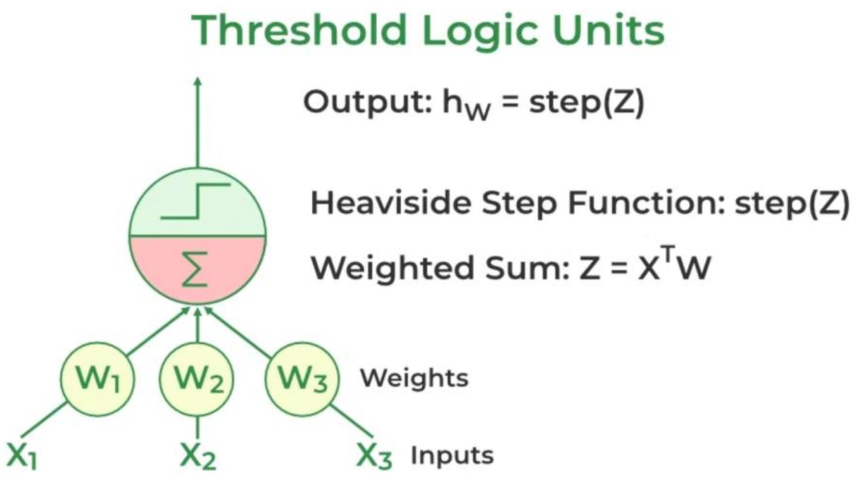
The weighted sum is computed as:

$$z=w_1x_1+w_2x_2+\ldots+w_nx_n=X^TW$$

The step function compares this weighted sum to a threshold. If the input is larger than the threshold value, the output is 1; otherwise, it's 0. This is the most common activation function used in Perceptrons are represented by the Heaviside step function:

$$h(z)=\begin{cases}0 & \text{if } z<\text{Threshold}\\ 1 & \text{if } z\geq\text{Threshold}\end{cases}$$

A perceptron consists of a single layer of Threshold Logic Units (TLU), with each TLU fully connected to all input nodes.



Threshold Logic units

In a fully connected layer, also known as a dense layer, all neurons in one layer are connected to every neuron in the previous layer.

The output of the fully connected layer is computed as:

$$f_{W,b}(X)=h(XW+b)$$

where X*X* is the input W*W* is the weight for each inputs neurons and b*b* is the bias and h*h* is the step function.
During training, the Perceptron's weights are adjusted to minimize the difference between the predicted output and the actual output. This is achieved using supervised learning algorithms like the delta rule or the Perceptron learning rule.
The weight update formula is:

wi,j=wi,j+η(yj−y^j)xi*wi,j*=*wi,j*+*η*(*yj*−*y*^*j*)*xi*

Where:

- wi,j*wi,j* is the weight between the ith*ith* input and jth*jth* output neuron,
- xi*xi* is the ith*ith* input value,
- yj*yj* is the actual value, and y^j*y*^*j* is the predicted value,
- η*η* is the **learning rate**, controlling how much the weights are adjusted.

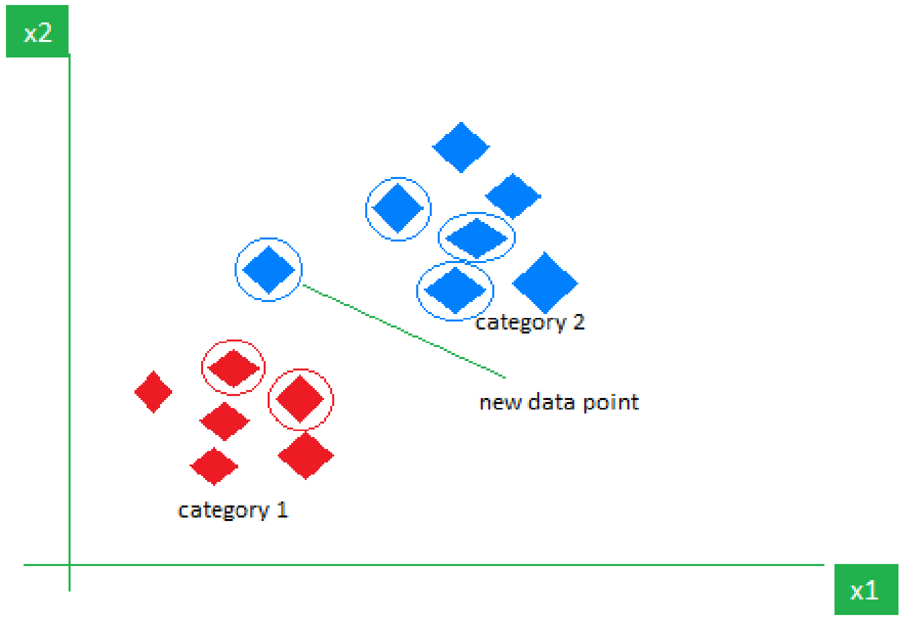
This process enables the perceptron to learn from data and improve its prediction accuracy over time.

What is the K-Nearest Neighbors Algorithm?

KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

As an example, consider the following table of data points containing two features:



.

KNN Algorithm working visualization

Now, given another set of data points (also called testing data), allocate these points to a group by analyzing the training set. Note that the unclassified points are marked as ‘White’.

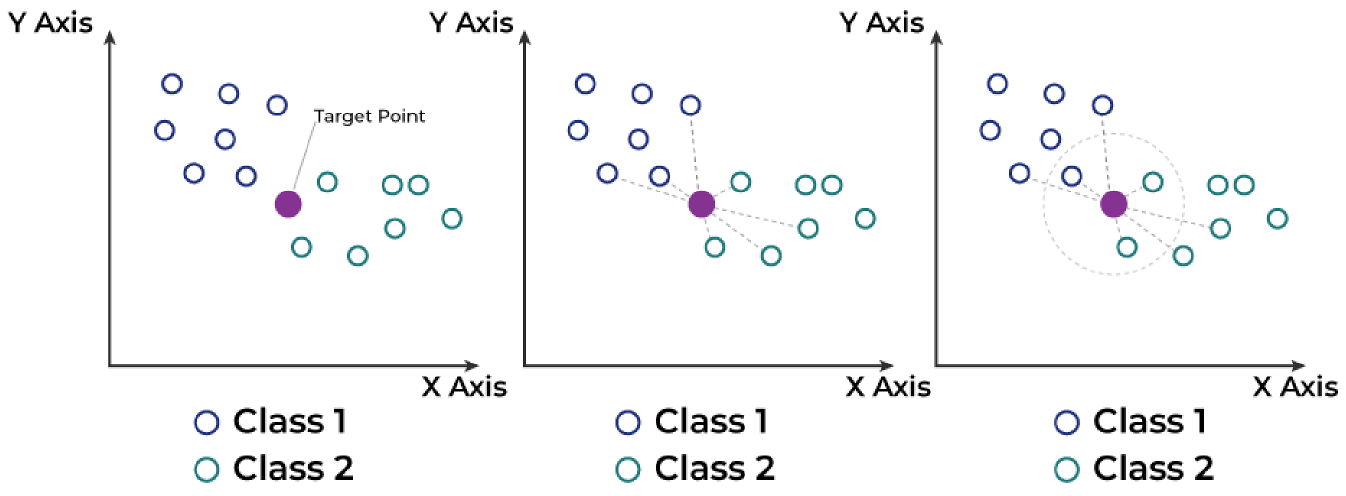
Intuition Behind KNN Algorithm

If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as ‘Red’ has a higher probability of getting classified as ‘Red’.

Intuitively, we can see that the first point (2.5, 7) should be classified as ‘Green’, and the second point (5.5, 4.5) should be classified as ‘Red’.

Workings of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity, where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K

K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

To measure the similarity between target and training data points, Euclidean distance is used. Distance is calculated between each of the data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

The k data points with the smallest distances to the target point are the nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

In the classification problem, the class labels of K-nearest neighbors are determined by performing majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point. In the regression problem, the class label is calculated by taking average of the target values of K nearest neighbors. The calculated average value becomes the predicted output for the target data point.

Let X be the training dataset with n data points, where each data point is represented by a d-dimensional feature vector XiXi and Y be the corresponding labels or values for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point XiXi in X using a distance metric, such as Euclidean distance:distance(x,Xi)=∑j=1d(xj–Xij)2]distance(x,Xi)=∑j=1d(xj–Xij)2]

The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K nearest neighbors and assigns it as the predicted value for x.

Exercise:

1. Write code to solve the Real/fake currency note classification KNN, SVM and Perceptron.
2. Create a confusion metrix using scikit-learn and searborn libraries
3. Design a user interface to get input features values from user and display the classification results on GUI.