MACHINE LEARNING FOR TARGET DETECTION USING UWB RADAR SENSOR NETWORKS

by

DHEERAL BHOLE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

Dec 2022

To my parents and sister who have always believed in me.

ACKNOWLEDGEMENTS

I sincerely express my gratitude to Dr. Qilian Liang and Dr. Chenyun Pan for guiding me throughout my research. Their guidance helped induce independent thinking and logical reasoning that a good researcher must imbibe. Dr. Qilian Liang stood by me during tough times and has guided me to achieve success. His enthusiasm and attitude towards research has deeply influenced me and helped me do better research. Also, I want to thank Dr. Stephen Gibbs and Dr. Yuze (Alice) Sun for their interests in my research and for being in my dissertation defense committee.

I would also like to thank all the teachers who taught me during my school, undergraduate and Ph.D. studies, first in India, then in the United States.

A big thank you to my lab mates Longwei Wang, Ganlin Zhao, Fangqi Zhu, Zhangliang Chen and Chengchen Mao for their interest in my research, helpful discussions and invaluable comments.

I would like to thank all my friends who have helped me in my journey. I am one of the lucky ones to have many great people in my life. To name a few would be unfair, so I'd rather not.

Finally, I would like to express my deep gratitude to my family, my parents Naresh Bhole and Jyoti Bhole, my sister Dr. Nehali Bhole who have encouraged me, inspired me and sponsored my undergraduate and graduate studies. I couldn't have done it without you. Love always!

This work was supported by U.S. National Science Foundation (NSF) under Grant CCF-2219753.

Nov 17, 2022

ABSTRACT

MACHINE LEARNING FOR TARGET DETECTION USING UWB RADAR SENSOR NETWORKS

DHEERAL BHOLE, Ph.D.

The University of Texas at Arlington, 2022

Supervising Professors: Qilian Liang and Chenyun Pan

Machine learning (ML) has recently been used to solve critical problems. This dissertation focuses on developing systems using Ultra-Wideband (UWB) wireless sensor networks and machine learning to solve critical tasks such as target detection in various challenging scenarios. These tasks have been researched for several years and efforts have been made to achieve universal solutions.

In the first part of this dissertation, we have proposed a system to detect metallic targets in foliage environment. Mission critical systems need to be ready for the harsh working environment such as dense foliage, water bodies, rain, heavy winds and other natural challenges. Extreme engineering excellence is needed to achieve a faultless system during such critical tasks and routines. To solve this problem, we come up with a Machine Learning system trained on wireless sensor network dataset. Our work consists of four main parts: First, we clean and standardize the dataset. Second, we transform the dataset using IFFT and PCA. Third, we train a XGBoost model on this dataset and make predictions on the test dataset. Finally, we calculate errors by comparing the predicted and actual values and obtain high accuracy with our method.

In the second part of this dissertation, we have proposed a system to detect humans through walls. Human detection through walls, doors and corridors is critical in applications such as hostage rescue situation, surveillance, activity recognition, etc. Our work consists of three main parts: First, we clean and standardize the dataset. Second, we train a Neural Network model on this dataset and make predictions on the test dataset. Finally, we calculate errors by comparing the predicted and actual values and obtain high accuracy with our method.

In the last part of this dissertation, we have proposed an ensemble ML system to optimize the first task of target detection in foliage environment. We apply generalized stacked machine learning system to harness the power of different ML models and we achieve the best accuracy with this method.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

In the summer of 1956, a group of scientists gathered at Dartmouth to form a new branch of science which is Artificially Intelligent [1]. They had the ambition to make machines that have awareness and that can learn specific tasks. They did make some progress since carefully programmed computers can do simple arithmetic, play chess and perform many human-like tasks. While humans evolve and write code for machine, machines learned very little. Later, in 1959, scientists realized that instead of teaching computers everything, it might be better to teach them to learn by themselves. One of the pioneering scientist Arthur Samuel's checker-playing program [2] was among the world's first successful self-learning programs that led to the term "Machine Learning". Machine learning (ML) is defined as a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty [3]. There are several ML methods including neural networks (NN), support vector machines [4] and k-nearest neighbor classifiers [5].

Support vector machine (SVM) [4] was a popular and dominant classification method before the arrival of deep learning. Even though it is still used in applications where less data is available. SVM solves the problem of maximizing the distance or margin separating two classes in input space. The problem turns out to be convex, and any local solution is also a global solution [6]. Using kernel methods such as Radial Basis Function (RBF) can boost the performance of SVM [7]. Boosting is an effective method of putting together multiple base classifiers to produce a strong classifier whose performance can be sufficiently better than that of any of the base classifier [6]. The most widely used boosting approach is AdaBoost [8] which stands for adaptive boosting. Adaboost collects all base classifiers' decisions, then uses training to adjust the weights given to each of them. The idea is that, better base classifiers should have stronger weights while worse base classifiers should have smaller weights. Adaboost can have much better performance than the best of the base classifiers. Deep learning is a set of ML models which allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts [9]. The graph of such model is deep with many layers, leading to the name "deep learning". The layers are neural networks that can be trained individually or simultaneously. The RBF kernel SVM on the MNIST benchmark was outperformed by Hinton's neural network [10] which led to the beginning of current neural network renaissance [9]. Deep learning has been very successful applications such as natural language processing [11], speech recognition [12] and image recognition [13].

NN were first proposed by Rosenblatt in the form of the perceptron [14]. NN work well on classification problems such as pattern recognition [15, 16, 17, 18, 19, 20], remote sensing [21, 22, 23], image processing [24, 25, 26], power load forecasting [27, 28, 29], nonlinear estimation [30, 31, 32, 33], limitation of inverse analysis [34, 35] and real time temperature prediction using neural networks [36, 37]. There are primarily 3 types of neural network - Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). NN with enough hidden units can approximate any continuous function giving it universal approximation property [38]. Even though there have been many successes, neural nets still have many problems. First, training is still heavily based on first order backpropagation which is slow, easily falls into local minima, and is not affine invariant. Second, current neural nets and deep learning models have an excessive number of parameters that need to be manually adjusted. Third, deep learning works well, but there is still no convincing theory for it.

In the past decade, ML has been able to solve incredibly difficult tasks where traditional programming has failed. ML has done very well on 1-D, 2-D and 3-D data in the supervised learning paradigm [9]. All this has been possible due to the availability of incredible computation resources, parallel & distributed implementation of algorithms and the hugely supportive Artificial Intelligence (AI) community [39].

Mission critical systems need to be ready for the harsh working environment such as dense foliage, water bodies, rain, heavy winds and other natural challenges. Extreme engineering excellence is needed to achieve a faultless system during such critical tasks and routines. We have considered one such scenario which is target detection in foliage environment. Many research efforts have been invested in studying the dense cluttered foliage environment. Multipath fading, scattering from trees, ground reflections, moving components of the forest, adverse weather conditions and non-stationary nature of the environment make it a challenging but interesting Engineering problem.

In previous work, [40] applied a differential based approach for foliage target detection. Whereas [41] used information theory for the same. Previous work used DSP extensively which require expertise needed to be tuned for different scenarios. A universal approach that does not require extensively expert knowledge and can work everywhere is badly needed. Our system is robust and only data dependent. [42] provides a radar sensor wireless channel model in foliage environment. In outdoor UWB channel, the multipath contributions that arrive at the receiver are grouped into clusters. The time of arrival of clusters can be modeled as a Poisson arrival process, while within each cluster, subsequent multipath contributions for rays also arrive according to a Poisson process. The amplitude of channel coefficient at each path follows Rician distribution for medium and far distance, and it is non-stationary for paths from short distance (one of two Rician distributions), and these observations are quite different with the IEEE indoor UWB channel model and S-V model. [43] proposed a Discrete-Cosine Transform (DCT)-based approach for sense-through-foliage target detection when the echo signal quality is good, and a Radar Sensor Network (RSN) and DCT-based approach when the echo signal quality is poor.

Human detection through walls, doors and corridors is critical in applications such as hostage rescue situation, surveillance, activity recognition, etc. Such a critical task also comes with a lot of diverse engineering problems. One of the most important problem being detecting humans through walls, doors and other construction material. Good penetration and range resolution have enabled Ultra wideband (UWB) to be primary choice for target detection through most construction materials and other indoor situations [44]. Large bandwidth of UWB signals enable it to achieve high range resolution and thus detection of multiple nearby objects. In this study we have considered a tougher problem with stationary Human being. Movement can make the problem relatively simpler as the radar would receive different echoes for different timestamps, making detection easier. We have considered typically occurring brick wall, gypsum wall and wooden door structures in our study.

Internet of things (IoT) development is strongly based on effective and energy efficient wireless sensor network deployment [45]. Accurate and efficient Human detection/localization will improve the performance of IoT ecosystem.

CHAPTER 2

Ultra-wideband (UWB): A Review

2.1 Ultra-wideband

Ultra-wideband (UWB) is a wireless communication protocol that uses very low energy for short-range, high-bandwidth communications over a spectrum of frequencies ranging from 3.1 to 10.6 GHz. First definition for UWB signal was based on the fractional bandwidth (B_f) of the signal having a value greater than 0.25 which was defined by The Defense Advanced Research Projects Agency (DARPA) [46]. Later the fractional bandwidth requirement was reduced to 0.2. The fractional bandwidth is defined in 2.1, where f_L is the lower frequency of the -10dB emission input and f_H is the upper frequency of the -10dB emission point, respectively [47]. As per FCC, the signal is considered as UWB if the signal bandwidth is 500MHz or more and the transmitter cannot have transmit power more than 0.5mW. This limitation significantly limits the applications of UWB to short range-high data rate or long range-low data rate applications. But this enables UWB devices to not interfere with other narrowband device signals and it can co-exist with other signals.

$$B_f = 2\frac{f_H - f_L}{f_H + f_L} \tag{2.1}$$

The transmission center frequency f_c is determined as the average of these cut-off frequencies as indicated in 2.2 and depicted in Figure 2.1 [48].

$$f_c = \frac{(f_H - f_L)}{f_H + f_L} \tag{2.2}$$

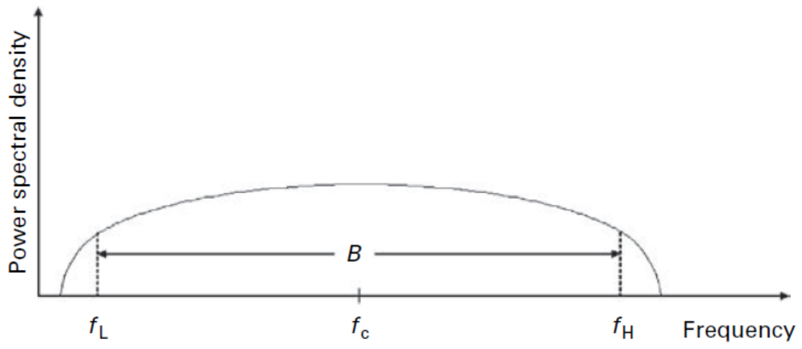


Figure 2.1: A UWB signal in frequency domain

UWB communications avail many advantages over narrowband communications technology, an important one being improved channel capacity. By Shannon's channel capacity formula, the capacity increases with bandwidth (B) as indicated in 2.3.

$$C = B \log_2(1 + SNR) \tag{2.3}$$

2.1.1 The UWB Radar Range Equation

The UWB range equation is a non-stationary quantity because the directivity factor of a transmitting antenna, G, the effective cross section of a receiving antenna, A, and the effective scattering cross section of a target, σ_{UWB}, become dependent on time and signal waveform [49].

$$R(s,t) \leq \sqrt[4]{\frac{EG(\theta,\phi,S,t)\sigma_{UWB}(t)A(\theta,\phi,S,t)}{(4\pi)^2 \rho q N_0}} \tag{2.4}$$

where:

E	= the energy of the radiated signal
G	= the directivity factor of a transmitting antenna
S	= the signal waveform
A	= the effective cross section of a receiving antenna
σ_{UWB}	= the effective scattering cross section of a target

ρ = the losses in all the systems of a radar

q = the threshold signal-to-noise ratio

N_0 = the spectral density of noise power

2.1.2 IEEE 802.15.4a Standard

IEEE 802.15 low-rate alternative PHY task group (TG4a) was formed in 2004 to design an alternate PHY specification for the already existing IEEE 802.15.4 standard for WPANs [50]. High-precision ranging with low-power and low-cost devices was the main aim of the TG4a. In 2007, the TG4a defined IEEE 802.15.4a standard supporting new applications.

2.1.3 UWB Transmission

Conventional communication systems transmit coded information using amplitude, phase and/or frequency modulation of a sinusoidal signal. UWB is very different to this as it transmits information by sending radio signals at specific time intervals which occupy a large bandwidth. The information can also be modulated on UWB signals (pulses) by encoding the amplitude of the pulse, its polarity and/or by using orthogonal pulses [46, 49]. Generally, UWB antennas directly radiate impulse UWB signals without the need of a carrier signal compared to traditional narrowband transceivers, hence the transceiver complexity is reduced. Data transmission can be conducted from one of the two approaches:

- Orthogonal Frequency Division Multiplexing (OFDM) channels are created by subdividing the total available UWB bandwidth.
- Impulse radios (Ultra short pulses in the picosecond range) which span all frequencies.

The first approach utilizes the spectrum more efficiently and facilitates improved performance and data throughput. This comes at the cost of increased signal processing effort (complexity) and power consumption. The second approach is cheaper at the expense of reduced signal to noise ratio (SNR). The choice between the two approaches is application specific.

2.1.3.1 Channel Allocations

As per IEEE 802.15.4 standard, a UWB equipment could transmit at one or more of the following bands:

1. Sub-GHz: 250-750 MHz
2. Low band: 3.244-4.742 GHz
3. High band: 5.944-10.234 GHz

The Sub-GHz band has 1 channel, Low band has 4 channels and High band has 11 channels. These channels and their bandwidths and center frequencies are mentioned in [50].

2.1.3.2 System Parameters

The IEEE 802.15.4a standard supports multiple data rates 0.11, 0.85, 1.7, 6.81, 27.24 Mbps.The standard supports these variable data rates through the use of variable-length bursts. The channels can transmit pulses with various mean pulse repetition frequency (PRF) options, which are 3.90, 15.6 and 62.4 MHz.

2.1.3.3 Ranging and Location Awareness

Ranging capability for the UWB PHY option is supported by IEEE 802.15.4a. The ranging estimation is obtained from time-delay estimates which are obtained via certain two-way protocols between two devices [50].

CHAPTER 3

Machine Learning (ML): A Review

3.1 Machine Learning

ML is the amalgamation of Mathematics and Computer Science aided by strong computing resources [51]. It aims to optimize the performance of specific task by using data samples and/or past experience [52]. With recent developments and huge interest in the field, there are many ML methods. In this chapter, we review a few of those. ML can be divided into many types based on different parameters. The most common being the use of known labels in the training process and thus yielding supervised and unsupervised learning. New sub fields like semi-supervised, generative networks are also coming up with the amount of research interest in AI [53, 54, 55].

3.1.1 Supervised Learning

Supervised learning is the most common and applied field of ML. Learning the underlying data distribution using labeled data and mathematical models is the crux of supervised ML. It attempts to learn a function mapping known inputs to unknown outputs based on learning of patterns found in label training data. Common supervised ML algorithms are listed below:

- Support Vector Machines (SVM)
- Naive Bayes
- k-Nearest Neighbors (KNN)
- Family of Regression
- Decision Trees

- Random Forest (RF)
- Ensemble Methods
- Neural Networks (NN)

The received waveform is time domain 1-D data. After exploratory data analysis, we conclude that the data we considered is not enough for deep learning models. So, we employ algorithms like k-nearest neighbors, support vector machines, random forest, XGBoost and shallow neural networks which work very well with less data. We have explained XGBoost and NN in details in the following chapters as we propose systems based on these models.

KNN [56] is a simple non-parametric supervised learning algorithm where the output decision for a test sample is based on its k neighbors. K in KNN is based on feature similarity and finding the correct value of k is called parameter tuning. Smaller k value can produce noisy output and have larger influence on the result. Larger k value will be computationally expensive and might result in lower variance but increased bias. If X is matrix of features from an observation and Y is a class label, KNN estimates the conditional distribution of Y given X and classifies an observation with the highest probability. Given a positive integer k , k-nearest neighbors looks at the k observations closest to a test observation x_0 and estimates the conditional probability that it belongs to class j using equation 3.1.

$$Pr(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in N_0} I(y_i = j) \tag{3.1}$$

where N_0 is the set of k-nearest observations and $I(y_i = j)$ is an indicator variable that evaluates to 1 if a given observation (x_i, y_i) in N_0 is a member of class j, and 0 if otherwise. After estimating these probabilities, k -nearest neighbors assigns the observation x_0 to the class which the previous probability is the greatest.

SVM [57] is a popular classification method used in applications where less data is available. SVM solves the problem of maximizing the distance or margin separating two classes in input space. The problem turns out to be convex and any local solution is also a global solution [6]. Different kernel methods such as linear, radial basis function (RBF), polynomial, sigmoid can be used to train the SVM model. The SVM finds the maximum margin separating hyperplane. We consider a linear classifier: $h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b)$ with a binary classification setting with labels {+1, -1}. So the best hyperplane is the one which maximizes the distance to the closest data points. SVM optimization problem can be represented as equation 3.2, where optimized parameters (**w**,b) can minimize the equation.

$$\min_{\mathbf{w},b} \quad \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \max[1 - y_i(\mathbf{w}^T\mathbf{x} + b), 0] \tag{3.2}$$

Algorithm 1 Random Forest for Regression or Classification

```
1: for $b = 1, 2, ..., B$ do
2:     (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
3:     (b) Grow a random-forest tree $T_b$ to the bootstrapped data:
4:             i. Select $m$ variables at random from the $p$ variables.
5:             ii. Pick the best variable/split-point among the $m$.
6:             iii. Split the node into two daughter nodes.
7: end for
8: Output the ensemble of trees $\{T_b\}_1^B$.
   To make a prediction at a new point $x$:
   Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
   Classification: $\hat{C}_{rf}^B(x) = majority\ vote\ \{\hat{C}_b(x)\}_1^B$.
```

RF [58] is an ML method consisting of multiple decision trees. Each tree receives different subset of data obtained by bootstrapping and random feature selection thus creating an uncorrelated forest of trees. This leads to a more accurate ensemble than any of the individual tree. Typically, the more the number of trees in the forest the better the performance of the ensemble. We tried 5, 25, 50, 100, 200, 500, 750 and 1000 number of trees. The performance plateaus after 500 trees and is the optimal choice for all cases which are shown in section 4.

3.1.1.1 Supervised Learning Training Process

High quality data acquisition is the first step as seen in Figure 3.1 [51]. Majority of time is spent in getting the data ready for the training process. Raw data generated/collected needs to be studied, cleaned and prepared for the next stage so that researchers can decide on appropriate learning algorithms, metrics, loss functions and parameters [59]. Data preprocessing techniques like Fourier Transform, Dimensionality Reduction and Normalization are used in this work. Data is split into Training, Test and Evaluation sets and the splitting ratio is experimentally setup [60]. The aim of obtaining a generalized model depends on sensible data splitting ratio.

Training is the next phase where ML algorithms learn the relationship between input features and target variable. A loss function is used to determine the difference between the true and predicted value. The role of the ML algorithm is to update the predicted values such that the loss is minimized. At this stage, training and validation sets are used to refine the model (Optimize hyperparameters, change features, etc). In this work a variety of algorithms are used to learn the aforementioned relationship. A set of Individually trained classifiers whose predictions are combined when classifying new instances is known as an ensemble learner. Ensemble is often more accurate than any of the individual classifiers in it [61].

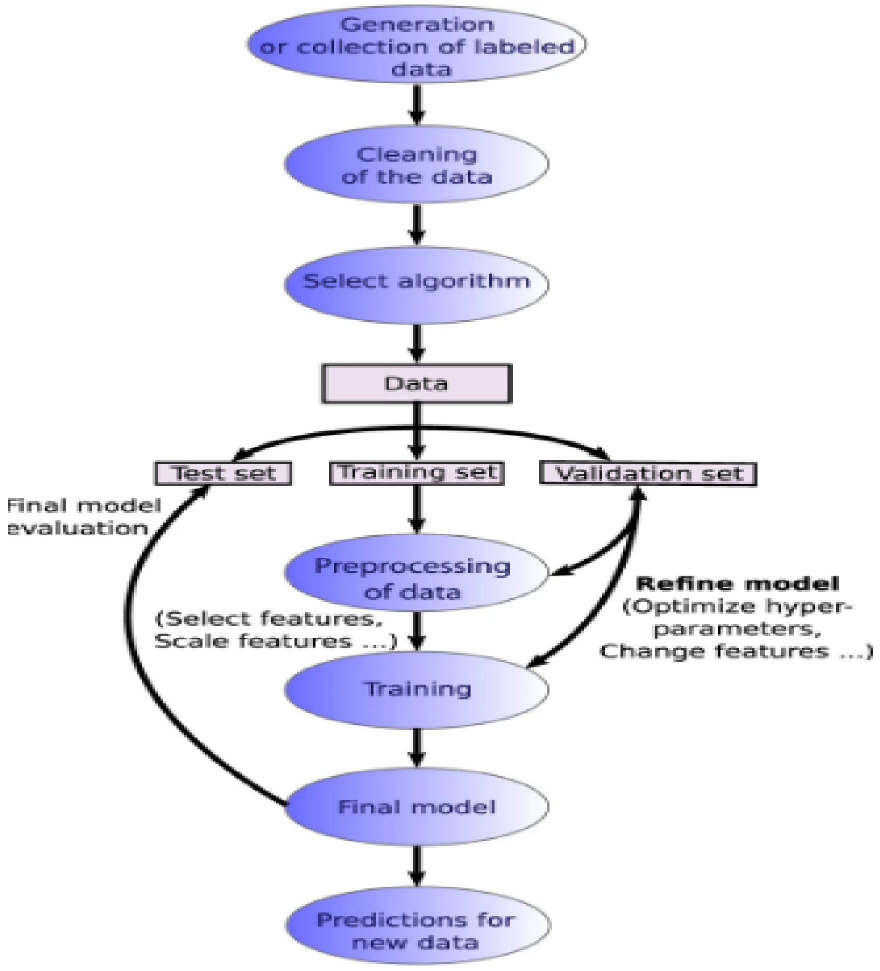


Figure 3.1: Supervised learning flowchart

Model evaluation is the process of utilizing various evaluation metrics to understand a ML model's performance. It is used to compare the performance of different algorithms to make up an empirical study. Predicted labels are compared to true labels at this stage.

Model Testing is the final stage of predicting the outputs for new data. The aim for a model is to generalize well and not to under or over fit the training data.

3.1.2 Unsupervised Learning

Unsupervised learning is another field of ML which works with abundantly available unlabeled data. Learning the underlying data distribution, assemble that data based on similarities and express that dataset in a precise format is the crux of unsupervised ML [62]. Common unsupervised ML algorithms are listed below:

- Clustering
- Anomaly Detection
- Principal Component Analysis (PCA)
- Neural Networks (NN)

3.1.3 Reinforcement Learning

Reinforcement learning is a ML training method based on maximizing reward for actions taken by an agent in an environment. Simply stating Reinforcement learning rewards an agent for choosing from a correct set of actions and punishes for choosing otherwise. The goal is to obtain an action model that maximizes total reward of the agent. A reinforcement learning agent interacts with its environment, takes action and learns through trial and error. [63, 64]. Common reinforcement learning algorithms are listed below:

- State-action-reward-state-action (SARSA)
- Q-learning
- Deep-learning
- Markov decision process

3.2 Preprocessing Techniques

ML performance and efficacy depends on data preprocessing techniques yielding high quality Data. Data quality includes accuracy, completeness, consistency, believability and interpretability [65]. The techniques used in this work are discussed briefly:

3.2.1 Dimensionality Reduction

Dimensionality reduction is a method of decreasing the number of random variables or attributes in the dataset. Wavelet transforms and principal component analysis (PCA) transforms original data onto a smaller space. Attribute subset selection detects and removes irrelevant, weakly relevant or redundant attributes or dimensions [65].

3.2.1.1 Principal Component Analysis (PCA)

Principal component analysis is the multivariate statistical technique of finding the principal components and using them to perform a change of basis on the data [66]. PCA analyzes a dataset's inter-correlated dependent variables with the goal of mapping these features to a set of new orthogonal variables called principal components. PCA also represents the pattern of similarity of the observations and the variables by displaying them as points in maps [67]. The new projections have highest variance along the direction in space known as first principal component. The second principal component is orthogonal to the first principal component and maximizes variance among all directions orthogonal to the first. In this work PCA is utilized to reduce the high dimensional data to low dimensional data and only utilizing components carrying most variance of the data. This results in low computation effort without any accuracy trade-off.

3.2.2 Fourier Transform

Fourier transform is not a commonly used preprocessing technique in ML but it is often used in signal processing tasks. A Fourier Transform is a mathematical transform that decomposes functions depending on time or space into functions depending on temporal frequency or spatial frequency. Fourier and inverse Fourier transform are shown in 3.3 and 3.4 respectively.

$$X(w) = \int_{-\infty}^{\infty} x(t)e^{-jwt}dt \tag{3.3}$$

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(w)e^{jwt}dw \tag{3.4}$$

When handling recurrent time domain sometimes classification becomes tough due to lack of easy decision boundaries. So a natural option arises to convert time domain data to frequency domain data. Frequency domain representation gives how much signal is encompassed within each frequency band over a range of frequencies. So particular frequencies will be more populated and providing natural decision boundaries to help ML models classify easily as compared to its time domain representation [68]. In this work Discrete Fourier Transform is applied using the NumPy built-in Fast Fourier Transform (FFT) algorithm (CT) [69, 70].

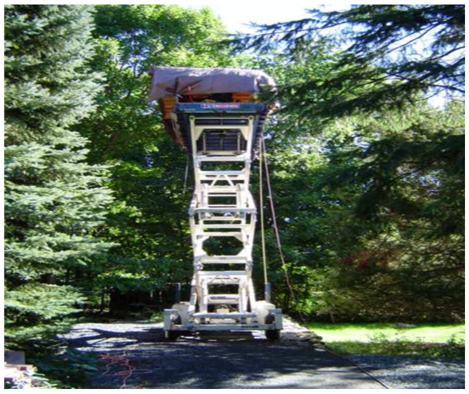
CHAPTER 4

Target Detection in Foliage Environment

Mission critical systems need to be ready for the harsh working environment such as dense foliage, water bodies, rain, heavy winds and other natural challenges. Extreme engineering excellence is needed to achieve a faultless system during such critical tasks and routines. We have considered one such scenario which is target detection in foliage environment. Many research efforts have been invested in studying the dense cluttered foliage environment. Multipath fading, scattering from trees, ground reflections, moving components of the forest, adverse weather conditions and non-stationary nature of the environment make it a challenging but interesting Engineering problem.

4.1 Data Collection

The foliage penetration measurement task was conducted from August 2005 to December 2005 [71]. Barth pulse source was used to collect the data in this experimental setup. For this UWB data, each sample is spaced at 50 picosecond intervals and 16,000 samples were collected for each collection for a total duration of 0.8 microseconds at a rate of approximately 20 Hz. The Barth pulse source was operated at low amplitude and 35 pulses reflected signal were averaged for each collection.


(a) The lift in the experiment with all the equipment [71]

(b) The target (a trihedral reflector) is shown on the stand at 300 feet from the lift [71]

Figure 4.1: Foliage data collection setup

Figure 4.1 depicts the experimental setup. Figure 4.1a shows the lift used in the experiment which supported the entire measuring apparatus. The man lift was a 4-wheel drive diesel platform that was driven up and down a graded track 25 meters long with an experimental length of 20 meters. The lift was placed at 11 different positions along the track resulting in 11 data collections for target and no target scenario. Figure 4.1b shows the trihedral reflective target. Foliage-target data were taken from the track firing into the foliage with and without the target.

Figure 4.2 displays the received echoes with (a) no target and (b) with target on range. On visual inspection it is difficult to conclude the presence of target in these figures. To give a better understanding of these received echoes to the reader, expanded views from sample 13001–15000 are shown in Figure 4.3a and 4.3b. The

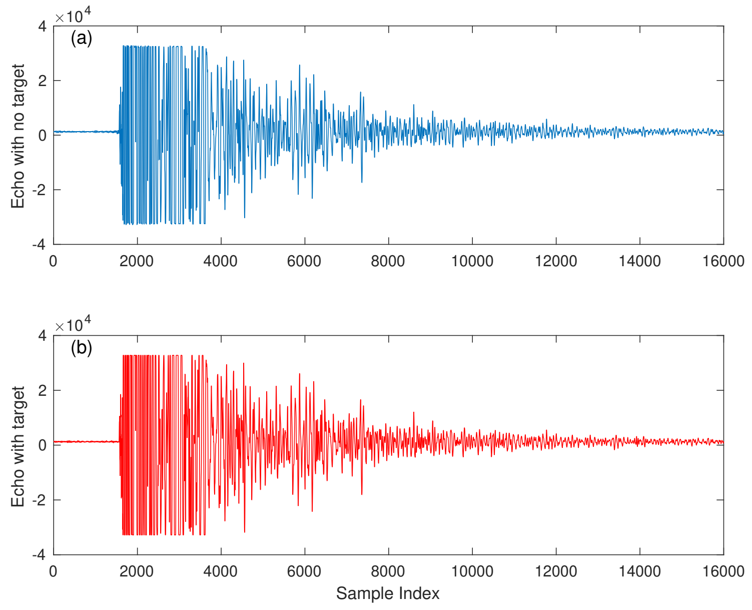


Figure 4.2: Reflected echoes for two different cases with: (a) no target (b) target

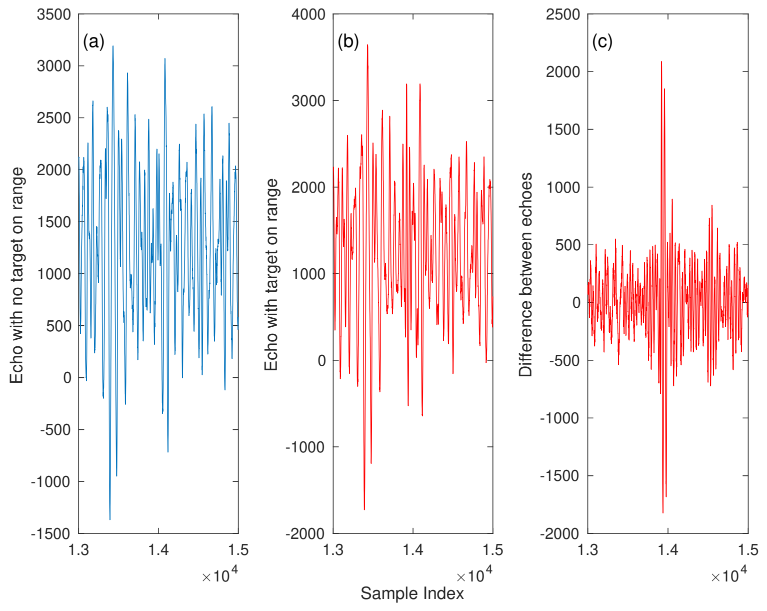


Figure 4.3: Expanded view of reflected echoes from sample index 13001 to 15000 for two different cases with: (a) no target (b) target (c) their difference

received echo in Figure 4.3a can be treated as response from the foliage clutter since there is no target in range. Therefore, the difference between Figure 4.3a (echo with no target) and Figure 4.3b (echo with target) is the target response which is shown in 4.3c. In practical scenario, we either receive echo as shown in Figure 4.2a or Figure 4.2a. The main challenge is to train an Artificial Intelligence to make target detection decision based only on the received echo.

4.2 Dataset Preparation

One of the most important steps to get good results with ML is to have a clean dataset. The dataset we handled had consistent received echoes with 16000 samples for multiple locations with target and no target cases. As per our knowledge the target is detected around sample 13900 of the received echo for this experimental setup. So, the ML model does not need to be trained for the entire sequence of 16000 samples. We chose a window of 2000 samples from 14001 to 16000 from each received echo and reduced the size of the dataset to be considered. So each signal had 2000 samples and 35 such signals were recorded for each target and no target case was repeated over 11 different positions. The entire dataset had $11 * 2 * 35 = 770$ signals with 2000 samples each.

4.3 XGBoost Model for Target Detection

This section presents the work published in [72] and the next section presents a later on carried out comparative analysis with different ML models for the same dataset.

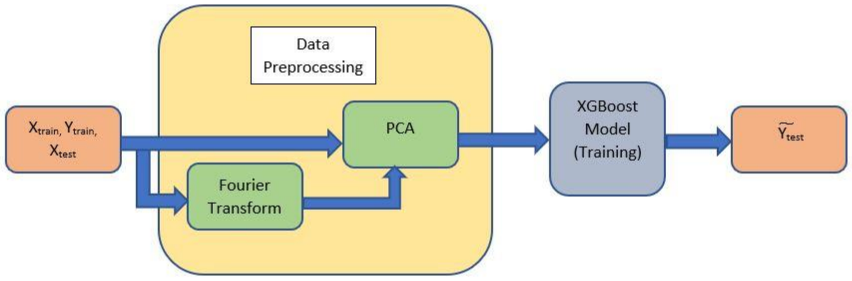


Figure 4.4: ML pipeline for foliage dataset

4.3.1 ML Pipeline

Figure 4.4 shows the flow of the data through the ML pipeline. The received raw UWB data is split into training (X_{train}, Y_{train}) and testing (X_{test}, Y_{test}) datasets. These datasets are preprocessed in the next phase. We transform the sequential data from time to Frequency domain using FFT, so that we could do analysis on two domain representations of data. Then PCA is applied to reduce the dimensionality of dataset while retaining maximum variance present in the dataset along new dimension called principal components. This preprocessed data is then fed to the ML model (Eg. XGBoost) which trains on (X_{train}, Y_{train}) processed dataset. After training and validation the model is used for testing with X_{test} which returns predicted labels $\widehat{Y}_{test}$. These predicted labels $\widehat{Y}_{test}$ and true test labels Y_{test} are compared to determine different performance metrics like area under the curve (AUC) and accuracy.

4.3.2 XGBoost Algorithm

The received echoes are time domain single dimensional data, and its Fourier transform makes it a complex valued two-dimensional data. XGBoost [73] is a supervised, scalable end-to-end tree boosting algorithm for sparse data and weighted quantile sketch for approximate tree learning. Simply stated it is an open-source implementation of gradient boosted trees (Gradient Boosting on decision trees). Firstly, decision trees are just a sequence of simple decision rules that combined produce a prediction of the desired variable. Secondly, Gradient Boosting is a ML algorithm which creates a prediction model in the form of an ensemble of weak prediction models, usually decision trees. If the weak learner is a decision tree as shown in Figure 4.5, the final algorithm is called gradient boosted tree which typically outperforms random forests [40, 74]. Quicker model exploration is enabled due to parallel and distributed computing used in XGBoost.

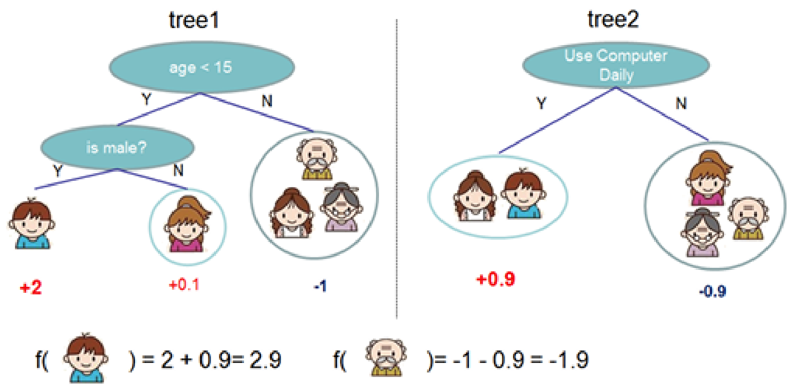


Figure 4.5: Tree ensemble method

Figure 4.5 [73] shows an ensemble of 2 decision trees. The tree starts with a bigger problem at the root then breaking up the bigger problem into simple binary decision problems at each level. The process repeats until the tree reaches problems which cannot be broken up further known as leaf nodes. The final prediction of a given example is calculated by summing up the score in the corresponding leaves. The prediction for example 1 is calculated by aggregating its score from each tree, hence prediction = 2+0.9 = 2.9. For consistent performance, we trained 50 XGBoost models and averaged the performance rather than using the best accuracy achieved for a lucky random XGBoost model. For each model, the number of trees and its depth were different. This enables the model to achieve high accuracy.

Boosting fits an ensemble of models with the final Equation 4.1 and can be written as adaptive basis function models Equation 4.2

$$f(x) = \sum_{m=0}^{M} f_m(x) \tag{4.1}$$

$$f(x) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi_m(x) \tag{4.2}$$

where $f_0(x) = \theta_0$, $f_m(x) = \theta_m \phi_m(x) for\ m = 1, \ldots, M$, and ϕ_m is a sequentially add base functions to improve the fitness of the current model.

Booting algorithms thus solve Equation 4.3 either exactly or approximately at each iteration.

$$\{\hat{\theta}_m, \hat{\phi}_m\} = \underset{\{\theta_m, \phi_m\}}{\arg\min} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \theta_m \phi_m(x_i)) \tag{4.3}$$

For tree based learning, in the first step the tree considers every split parallel to the coordinate axes and chooses the split that minimizes the objective. In the second step, the tree considers every split parallel within each region and chooses the split which maximizes Gain. These steps are repeated till the stopping criterion is met. For a region R_j, weights w_j are learnt which minimize the loss function Equation 4.4. Region R_j includes the set of indices I_j such that $x_i \in R_j$ for $i \in I_j$. The empirical loss is given in Equation 4.5 where $\hat{L}_j$ is the aggregated loss at node j.

$$\hat{w_j} = \underset{w}{\arg\min} \sum_{i \in I_j} L(y_i, w) \tag{4.4}$$

$$\tilde{L}(\hat{f}) = \sum_{j=1}^{T} \sum_{i \in I_j} L(y_i, \hat{w}_j) \equiv \sum_{j=1}^{T} \hat{L}_j \tag{4.5}$$

XGBoost learns a new tree at iteration m in Equation 4.6

$$\{\hat{w}_{jm}, \hat{R}_{jm}\}_{j=1}^{T_m} = \underset{\{w_{jm}, R_{jm}\}_{j=1}^{T_m}}{\arg\min} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \sum_{j=1}^{T_m} w_{jm} I(x_i \in R_{jm})) \tag{4.6}$$

XGBoost tries to find the optimal tree structure $\hat{R}_{jm}$ and optimal weights $\hat{w}_{jm}$ that minimize the empirical loss shown in Equation 4.7

$$\{\{\hat{w}_{jm}, \hat{R}_{jm}\}_{j=1}^{T_m}\}_{m=1}^{M} = \underset{\{\{w_{jm}, R_{jm}\}_{j=1}^{T_m}\}_{m=1}^{M}}{\arg\min} \sum_{i=1}^{n} L(y_i, \sum_{m=1}^{M}\sum_{j=1}^{T_m} w_{jm} I(x_i \in R_{jm})) \quad (4.7)$$

Hessian matrix plays an important role to determine the tree structure as shown in Equation 4.8. The optimal leaf weights are determined by Equation 4.9.

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{jm}^2}{H_{jm}}\right] \quad (4.8)$$

$$\hat{w}_{jm} = -\frac{G_{jm}}{H_{jm}} \quad (4.9)$$

For Dataset $D = \{(\mathbf{x}_i, y_i)\}$ where $(|D| = n, \mathrm{x}_i \in R^m, y_i \in R)$, loss function L, number of iterations M, learning rate η and number of terminal nodes T, the pseudo-code for XGBoost with exact greedy algorithm can be described as follows:

Algorithm 2 XGBoost with exact greedy algorithm for split finding for small dataset

```
1: Input: $\mathbf{D}, L, M, \eta, T$
2: Initialize: $\hat{f}^{(0)}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg\min_\theta \sum_{i=1}^{n} L(y_i, \theta)$
3: for $m = 1, 2, ..., M$ do
4:     $\hat{g}_m(x_i) = [\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=\hat{f}^{(m-1)}(x)}$
5:     $\hat{h}_m(x_i) = [\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}]_{f(x)=\hat{f}^{(m-1)}(x)}$
6:     Find structure $\{\hat{R}_{jm}\}_{j=1}^{T}$ for splits which maximize Gain $= \frac{1}{2}[\frac{\mathbf{G}_L^2}{\mathbf{H}_L} + \frac{\mathbf{G}_R^2}{\mathbf{H}_R} - \frac{\mathbf{G}_{jm}^2}{\mathbf{H}_{jm}}]$
7:     Find leaf weights $\{\hat{\omega}_{jm}\}_{j=1}^{T}$ for learnt structure by $\hat{\omega}_{jm} = -\frac{\mathbf{G}_{jm}}{\mathbf{H}_{jm}}$
8:     $\hat{f}_m(x) = \eta \sum_{j=1}^{T} \hat{\omega}_{jm} I(x \in \hat{R}_{jm})$
9:     $\hat{f}^{(m)}(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x);$
10: end for
11: Output: $\hat{f}(x) \equiv \hat{f}^{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$
```

4.4 Results

We experimented with multiple window sizes like 500, 1000, 2000 and 3000 samples around the target sample. We found best predictive results for 2000 samples as it makes sense to only concentrate on relevant samples and ignore irrelevant samples. We considered different ML models on the rich foliage environment Dataset. Since we get best results for XGBoost for full dataset (Time and Frequency domain), we consider full dataset for all models.

4.4.1 XGBoost

Table 4.1: XGBoost performance for foliage dataset

Domain		Mean Accuracy with std dev (%)
Time		88.1±0.83
	Real	92.3±0.58
Frequency	Imaginary	88.3±0.79
	Both	93.6±0.89
All Data		**93.7**±0.59

- The target detection accuracy for time domain data is the least while it is far exceeded by frequency domain data. This corresponds to the ML model being able to learn better boundaries across different frequencies.
- For frequency domain data, accuracy is much better for Real part data than for Imaginary part data.
- For frequency domain data having both Real part and Imaginary part, it almost achieves maximum accuracy.
- As per ML rules, more relevant data means better model predictive power. We get the best accuracy of **93.7**% for entire data (Time and Frequency domain).

4.4.2 K Nearest Neighbors (KNN)

Table 4.2: K nearest neighbor performance for foliage dataset

No. of Neighbors	Mean Accuracy with std dev (%)
5 neighbors	**88.64**±0.81
10 neighbors	88.61±0.79
25 neighbors	87.66±0.21
50 neighbors	81.81±0.44
75 neighbors	81.11±0.19
100 neighbors	77.27±0.39
200 neighbors	72.72±0.46

From Table 4.2, K Nearest Neighbor [56] model performs best for K=5. As the value of K increases, accuracy drops. Being a majority voting algorithm, a small value of K means less number of neighboring points are important in this dataset classification problem. Compared to other algorithms used for this task as seen in Figure 4.8, KNN has the third best accuracy of **88.64%**.

4.4.3 Random Forest (RF)

Table 4.3: Random forest performance for foliage dataset

No. of Decision Trees in Random Forest	Mean Accuracy with std dev (%)
5 decision trees	68.18±2.87
10 decision trees	74.67±2.04
25 decision trees	72.00±1.57
50 decision trees	79.22±2.06
75 decision trees	79.22±1.03
100 decision trees	79.87±1.08
200 decision trees	81.16±0.98
500 decision trees	**83.81**±0.82

From Table 4.3, Random Forests Classifier [58] model performs best for Number of Trees = 500. Being a Decision Tree combining algorithm, a large number of Decision Trees means it depends on output of many weak learner trees. Thus increasing its training time too. So there should be a trade off between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 4.8, RF has the lowest accuracy of **83.81%**.

4.4.4 Neural Networks (NN)

Table 4.4: Neural network performance for foliage dataset

No. of Neurons in each hidden Layer	Number of Hidden Layers	Mean Accuracy with std dev (%)
	1	87.85±0.14
50	2	85.96±0.68
	3	85.96±0.07
	1	85.96±0.41
100	2	83.36±0.07
	3	84.31±0.00
	1	**90.45**±0.41
128	2	88.96±0.60
	3	89.61±0.63

From Table 4.4, Neural Network [75] model performs best for Number of Neurons=128 in 1 hidden layer. Being universal function approximator [38] algorithm, a large number of Neurons means it depends on output of many activation paths. Thus increasing its training time. So there should be a trade off between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 4.8, NN has the second highest accuracy of **90.45%**.

4.4.5 Confusion Matrix Analysis of All Models for Foliage Dataset

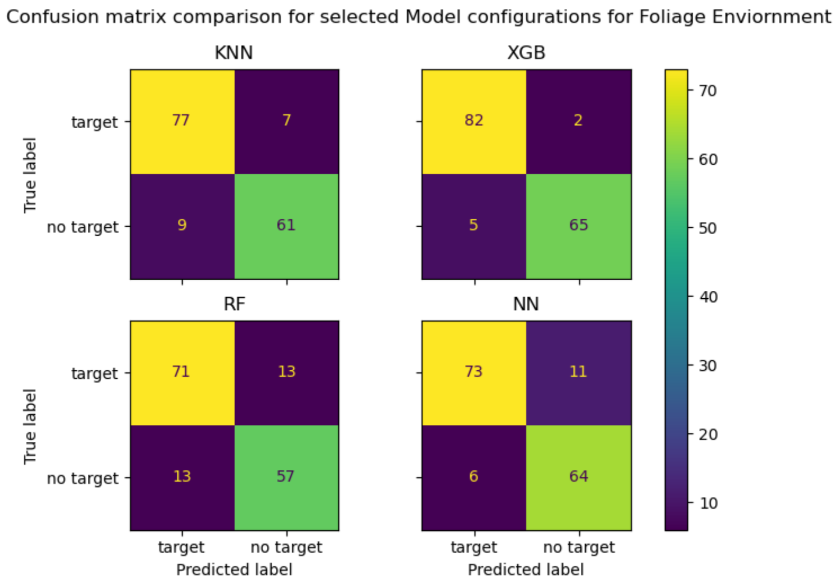


Figure 4.6: Confusion matrix of ML models for foliage dataset

The confusion matrix in Figure 4.6 shows label wise model performance. False positive and false negative can be extracted from this matrix. As a foliage target detection problem is very security sensitive, false negatives are not tolerable. XGBoost performs the best for this metric.

4.4.6 ROC Curve Analysis of All Models for Foliage Dataset

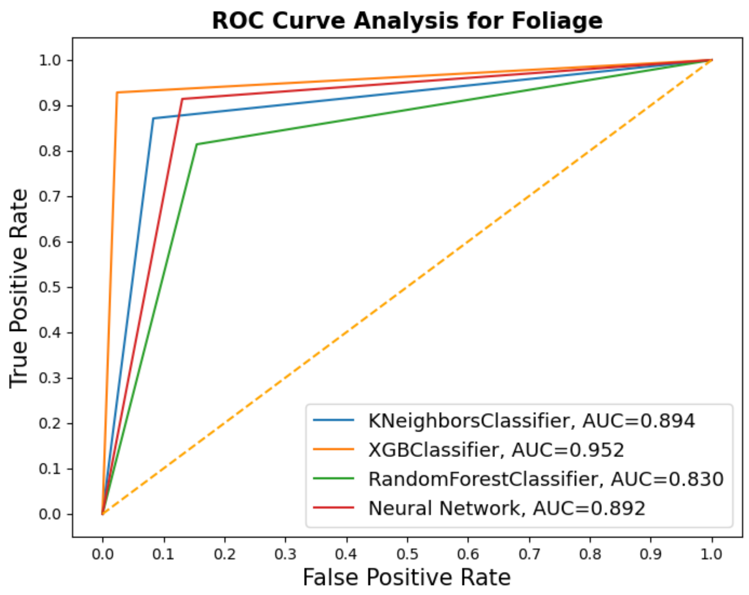


Figure 4.7: ROC curve of ML models for foliage dataset

ROC is a probability curve and AUC indicates the capability of the model to distinguish between classes. Higher AUC means that the model is better at classifying target. As shown in Figure 4.7, XGBoost receives a high AUC of 0.95 and have plotted it along with other model's AUC and random guessing AUC = 0.5, where the model has no predictive power. Since the dataset is well balanced, TPR and TNR performances are very close for all sets of data.

4.4.7 Accuracy Comparison of All Models for Foliage Dataset

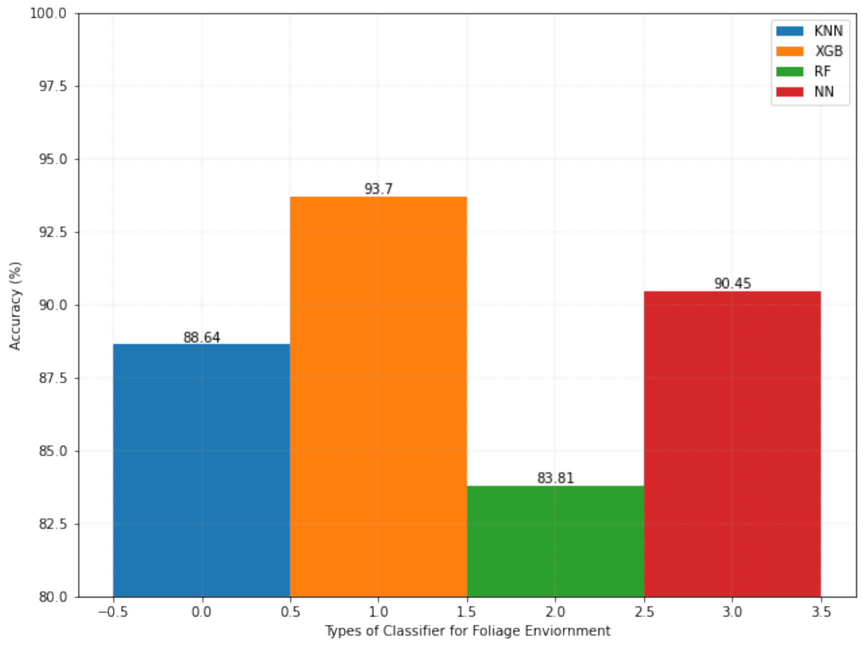


Figure 4.8: Accuracy of ML models for foliage dataset

The accuracy bar chart in Figure 4.8 shows accuracy performance for considered models. Random Forest has the lowest accuracy of **83.81%** while XGBoost has the best accuracy of **93.7**% [72].

After extensive experimentation and applying data transformation using our domain expertise we were able to achieve significant performance improvement over other methods. Our XGBoost based ML system is able to detect metallic target in foliage situation with an accuracy of **93.7**%.

CHAPTER 5

Human Detection through Wooden Door, Gypsum Wall & Brick Wall

Human detection through walls, doors and corridors is critical in applications such as hostage rescue situation, surveillance, activity recognition, etc. Such a critical task also comes with a lot of diverse engineering problems. One of the most important problem being detecting humans through walls, doors and other construction material. Good penetration and range resolution have enabled Ultra Wideband (UWB) to be primary choice for target detection through most construction materials and other indoor situations [44]. Large bandwidth of UWB signals enable it to achieve high range resolution and thus detection of multiple nearby objects. In this study we have considered a tougher problem with stationary human being. Movement can make the problem relatively simpler as the radar would receive different echoes for different timestamps, making detection easier. We have considered typically occurring brick wall, gypsum wall and wooden door structures in our study.

5.1 Data Collection

The Human detection through wall measurement was conducted at the University of Texas at Arlington [76]. 3 different locations were used to sense human through brick wall, gypsum wall and wooden door. P220 UWB radar in monostatic mode (shown in Figure 1) where waveform pulses are transmitted from a single Omni-directional antenna and the scattered waveform is received by a collocated Omni-directional antenna were used [77, 78]. The radar parameters in each of the cases given below were Integration: Hardware Integration = 512, Software Integration = 2, Pulse Repetition Frequency: 9.6 MHz Step Size: 13 bin, Window Size (ft): 10 ft

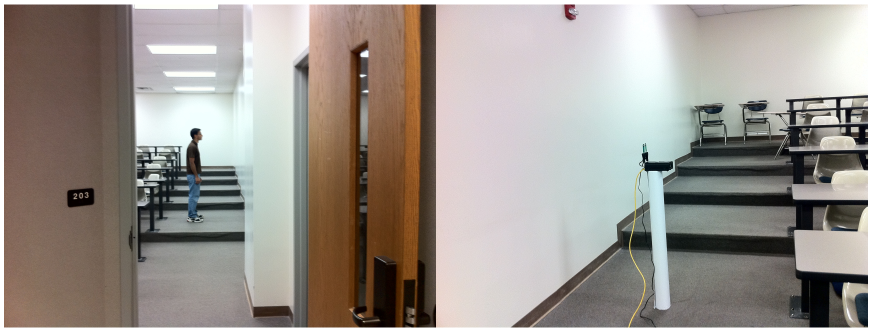


Figure 5.1: Human target (Left), UWB radar (Right) for gypsum wall

Figure 5.1 shows the location of the radar and Human target on different sides of a 1-ft thick gypsum partition wall. Person is at a distance of 6.5 ft from the radar on the other side of the wall and the height of the antennas from ground is 3'4".

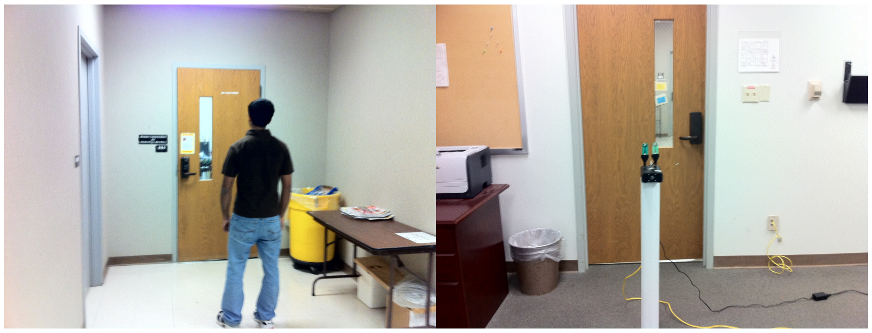


Figure 5.2: Human target (Left), UWB radar (Right) for wooden door

Figure 5.2 shows the location of the radar and Human target on different sides of a 1.57" wooden door. Person is standing at a distance of 7'6" from the radar on the other side of the door and the height of the antennas from ground is 3'4".

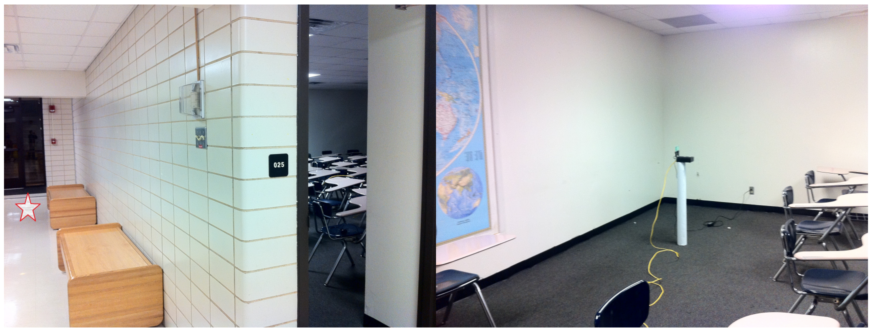


Figure 5.3: Human target position (Left), UWB radar (Right) for brick wall

Figure 5.3 shows the location of the radar and Human target on different sides of a 4.75" brick wall. Person is standing at a distance of 8' from the radar on the other side of the door and the height of the antennas from ground is 3'4".

Figure 5.4 displays the received echoes with (a) no person and (b) with person through wall or door. On visual inspection it is difficult to conclude the presence of target in these figures. The received echo in Figure 5.4a can be treated as response from the surroundings since there is no person in range. Therefore, the difference between Figure 5.4a (echo with no person) and Figure 5.4b (echo with person) is the target response which is shown in 5.4c. In practical scenario, we either receive echo as shown in Figure 5.4a or Figure 5.4a. The main challenge is to train an Artificial Intelligence to make target detection decision based only on the received echo.

5.2 Dataset Preparation

Preparing a good, clean, noiseless, outlier free, model friendly dataset is at the heart of the ML process. The dataset we handled had inconsistent received echoes with different numbers of samples per signal for multiple locations (gypsum wall, brick wall & wooden door) with person and no person cases. We used 512 samples for each signal (Figure 5.4) to have uniform amount of data for all cases. So, each

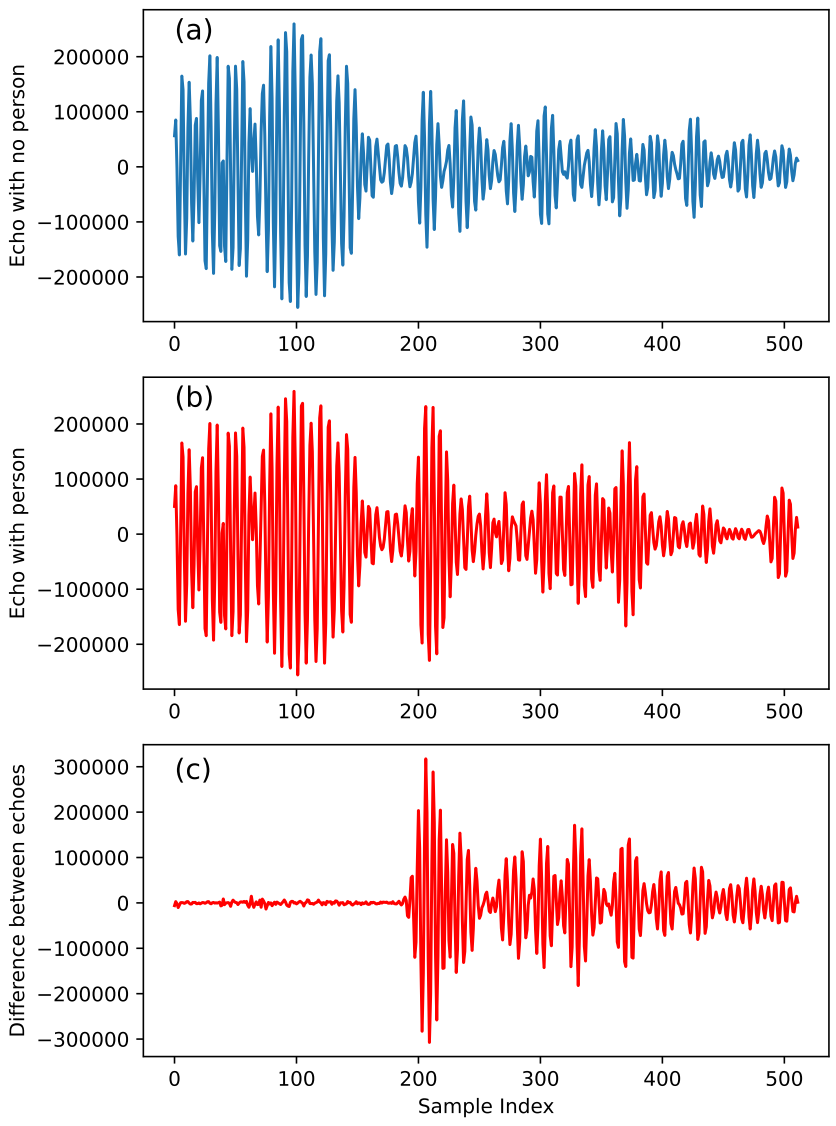


Figure 5.4: Reflected echoes through wall with: (a) no person (b) person (c) their difference

signal had 512 samples and 766 such signals were recorded for both person and no person case repeated over 3 different locations. The entire dataset had 3*2*766 = 4596 signals with 512 samples each.

5.3 Neural Network for Human Detection

This section presents the work submitted in [79].We carried out comparative analysis with different ML models for the same dataset.

5.3.1 ML Pipeline

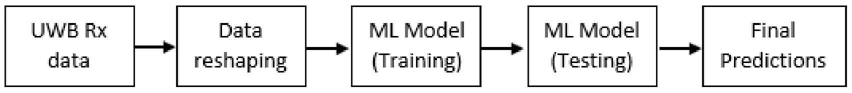


Figure 5.5: ML pipeline for wall dataset

Figure 5.5 shows the flow of data through the ML pipeline. Received UWB data is reshaped as explained in 5.2, then sent to the training and testing phase. We used 80-20 split for training and testing on the entire data. The training data is used to train multiple ML models. The model classifies new data in the testing phase and gives final predictions.

5.3.2 Neural Networks Algorithm

Artificial Neural Networks derive motivation from biological neural networks [75]. They are a set of ML models which allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts [9].

Figure 5.6 shows a shallow neural network. NN are made up of neurons arranged as input layer, one or more hidden layers, and an output layer. The artificial neurons from one layer are connected to the next layer with an associated weight and threshold. The aim of training is to learn weights of the edges connecting the nodes in the hidden layers. The node is activated if its output is above the specified threshold. The input

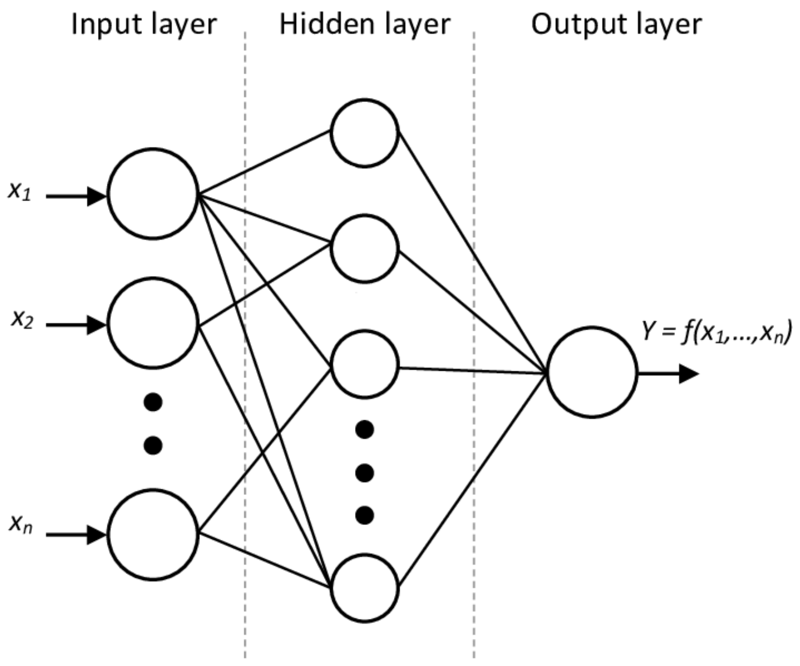


Figure 5.6: Shallow neural network

layer receives input which passes through the activated hidden layer nodes and final prediction is outputted by the output layer. NN can adapt to varying input such that it generates the best possible result without the need of changing the output criteria. The steps involved in the neural network algorithm are as follows:

- Step 1: Assign random weights to all edges in the network.
- Step 2: Using the inputs and the edges find the activation rate of hidden nodes.
- Step 3: Using the activation rate of hidden nodes and edges to output, find the activation rate of output nodes.
- Step 4: Find the error rate at the output node and recalibrate all the edges between hidden nodes and output nodes.
- Step 5: Using the weights and error found at the output node, cascade down the error to hidden nodes.
- Step 6: Recalibrate the weights between hidden node and input nodes.

- Step 7: Repeat the process till convergence criterion is met.
- Step 8: Using the final edge weights score the activation rate at the output nodes.

With $\hat{y}^{(i)}$ as the prediction of the i^{th} example, the differentiable negative log likelihood loss function for classification can be defined as:

$$L = -\sum_{i=1}^{N} [y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)})] \tag{5.1}$$

In the hidden layer, relu function is used to make the activation outputs non-negative and is defined in Equation 5.2.

$$relu(x) = max(0, x) \tag{5.2}$$

In the output layer, since it is a binary classification task, sigmoid function is used to limit output between 0 and 1 representing probability and is defined in Equation 5.3.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

For Dataset $D = \{(\mathbf{x}_i, y_i)\}$ where $(|D| = n, \mathrm{x}_i \in R^m, y_i \in R)$, loss function L, number of epochs E, learning rate η, hidden layer weights W_h & biases b_h and input layer weights $\mathbf{W_i}$ & biases b_i; the pseudo-code for neural network training process can be described as follows:

Algorithm 3 Neural network training pseudocode

```
1: Input: $\mathbf{D}, L, E, \eta, \mathbf{W_i}, b_i, W_h, b_h$
2: Initialize: $\mathbf{W_i}, b_i, W_h, b_h$ (random)
3: for $e = 1, 2, ..., E$ do
4:     # Feedforward: Pass data samples to neural network
5:     $h^{(m)} = relu(\mathbf{W_i} x^{(m)} + b_i)$
6:     $\hat{y}^{(m)} = \sigma(W_h h^{(m)} + b_h)$
7:     # Feedbackward: Update weights and bases such that L is minimized
8:     $W_h = W_h - \eta\ \partial L/\partial W_h$
9:     $\mathbf{W_i} = \mathbf{W_i} - \eta\ \partial L/\partial \mathbf{W_i}$
10:    $b_h = b_h - \eta\ \partial L/\partial b_h$
11:    $b_i = b_i - \eta\ \partial L/\partial b_i$
12: end for
13: Output: $\mathbf{W_i}, b_i, W_h, b_h$ (optimal)
```

5.4 Results

We conducted multiple experiments to test the performance of different ML models on the indoor environment wall Dataset. We recorded Mean Accuracy with Standard Deviation, Mean Training Time per Sample (ms) and Mean Testing Time per sample (ms).

5.4.1 Wooden Door Scenario

5.4.1.1 Neural Network (NN)

From Table 5.1, Neural Network [75] model performs best for Number of Neurons=50 in 1 hidden layer. Being universal function approximator [38] algorithm, a

Table 5.1: Neural network performance for wooden door dataset

No. of Neurons in each hidden Layer	Number of Hidden Layers	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
50	**1**	**97.36**±0.14	**16.70**±2.47	**0.30**±0.02
	2	97.16±0.68	17.42±3.22	0.31±0.02
	3	96.68±0.07	18.95±7.07	0.31±0.02
100	1	97.17±0.41	17.41±3.50	0.33±0.01
	2	97.03±0.07	17.63±3.63	0.30±0.02
	3	97.10±0.00	18.88±6.24	0.31±0.03
128	1	97.35±0.41	18.12±4.58	0.32±0.02
	2	97.16±0.60	19.52±7.80	0.31±0.03
	3	97.21±0.63	21.77±8.83	0.31±0.01

small number of Neurons means it depends on output of few activation paths. Thus having a relatively short training time. Compared to other algorithms used for this task as seen in Figure 5.13, NN achieves second best accuracy of **97.36%**.

5.4.1.2 K Nearest Neighbor (KNN)

Table 5.2: K nearest neighbor performance for wooden door dataset

No. of Neighbors	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 neighbors	78.89±0.15	0.28±0.10	0.008±0.000
10 neighbors	82.05±0.20	0.26±0.02	0.006±0.002
25 neighbors	85.86±0.22	0.22±0.10	0.007±0.008
50 neighbors	86.87±0.37	0.27±0.22	0.004±0.003
75 neighbors	87.62±0.57	0.21±0.20	0.005±0.001
100 neighbors	**87.71**±0.87	**0.21**±0.37	**0.002**±0.017
200 neighbors	86.94±0.70	0.23±0.16	0.002±0.001

From Table 5.2, K Nearest Neighbor [56] model performs best for K=100. As the value of K increases, accuracy rises. Compared to other algorithms used for this task as seen in Figure 5.13, KNN achieves third best accuracy of **87.71%**.

5.4.1.3 Support Vector Machine (SVM)

Table 5.3: SVM performance for wooden door dataset

Kernel Types	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
Radial Basis Function (RBF)	**99.00**±0.60	0.50±0.06	0.38±0.18
Linear function	97.52±0.62	**0.21**±0.02	**0.10**±0.01
3rd Order polynomial function	98.96±0.52	0.54±0.08	0.47±0.05
Sigmoid function	97.94±0.56	0.24±0.10	0.10±0.01

From Table 5.3, Support Vector Machine [57] model performs best for Radial Basis function (RBF) kernel Type. Being a non-probabilistic binary linear classification algorithm, SVM maps training samples to point in space so as to maximize the width of gap between the two categories. Compared to other algorithms used for this task as seen in Figure 5.13, SVM achieves best accuracy of **99.00%** after NN but it has a considerably lower training time than NN. It is an ideal candidate for this task.

5.4.1.4 Random Forest (RF)

Table 5.4: Random forest performance for wooden door dataset

No. of Decision Trees in Random Forest	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 decision trees	72.87±2.87	**0.12**±0.01	**0.01**±0.000
10 decision trees	78.79±2.04	0.20±0.02	0.01±0.002
25 decision trees	88.33±1.57	0.49±0.10	0.04±0.008
50 decision trees	93.73±2.06	0.87±0.22	0.04±0.003
75 decision trees	94.77±1.03	1.19±0.20	0.05±0.001
100 decision trees	96.20±1.08	1.50±0.13	0.07±0.017
200 decision trees	96.44±0.98	3.24±0.67	0.15±0.010
500 decision trees	**97.12**±0.82	7.00±0.44	0.30±0.004

From Table 5.4, Random Forests Classifier [58] model performs best for Number of Trees = 500. Being a Decision Tree combining algorithm, a large number of Decision Trees means it depends on output of many weak learner trees. This increasing its training time too. So there should be a trade off between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 5.13, RF achieves third best accuracy of **97.12%**.

5.4.1.5 Confusion Matrix Analysis of All Models for wooden door dataset

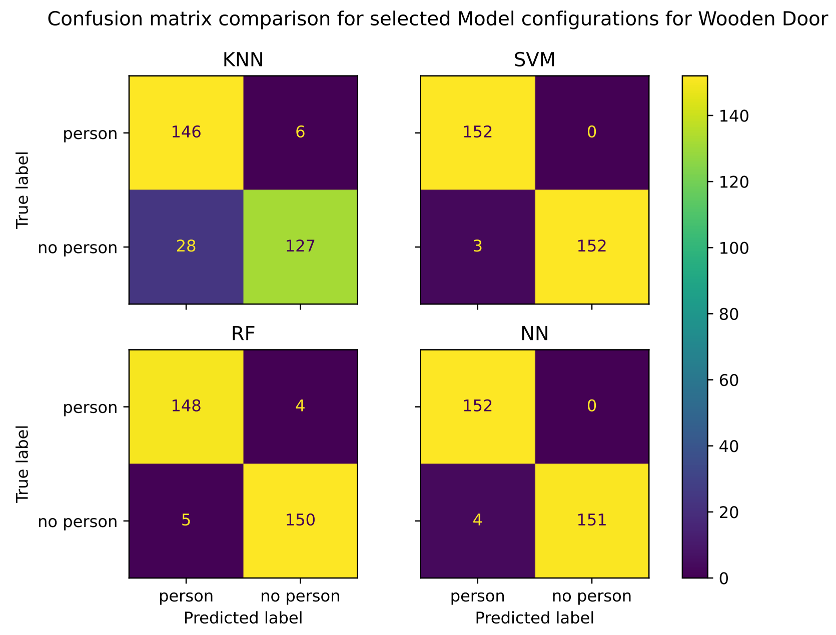


Figure 5.7: Confusion matrix of ML models for wooden door dataset

The confusion matrix in Figure 5.7 shows label wise model performance. False positive and false negative can be extracted from this matrix. As a person detection problem is very security sensitive, false negatives are not tolerable. SVM and NN perform the best for this metric.

5.4.1.6 ROC Curve Analysis of All Models for wooden door dataset

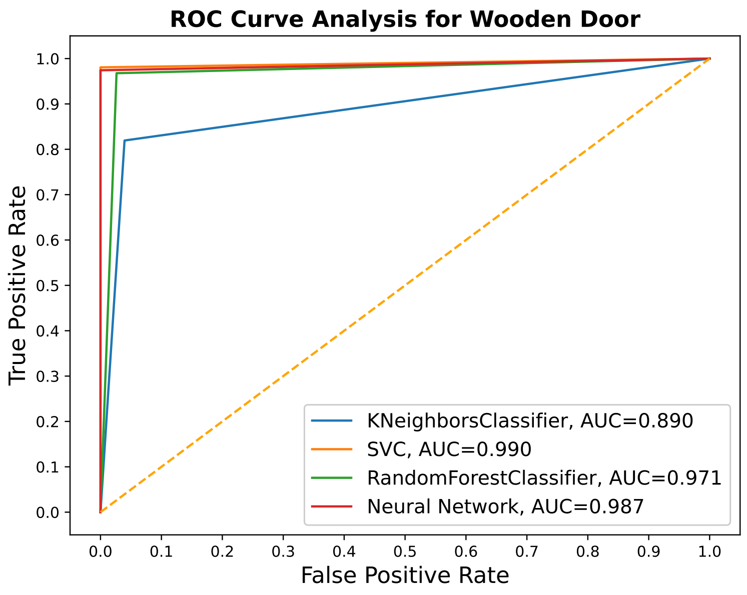


Figure 5.8: ROC curve of ML models for wooden door dataset

The ROC curve in Figure 5.8 shows AUC metric for considered models. Again, SVM and NN perform the best for this metric.

5.4.2 Gypsum Wall Scenario

5.4.2.1 Neural Networks (NN)

From Table 5.5, Neural Network [75] model performs best for Number of Neurons=128 in 3 hidden layers but for the best. Being universal function approximator [38] algorithm, a large number of Neurons means it depends on output of many activation paths. Thus increasing its training time too. So there should be a trade off

Table 5.5: Neural network performance for gypsum wall dataset

No. of Neurons in each hidden Layer	Number of Hidden Layers	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
50	1	94.54±0.05	**16.23**±2.53	0.29±0.02
	2	94.67±0.12	16.85±3.28	0.30±0.03
	3	94.22±0.18	16.35±2.48	0.30±0.02
100	1	94.89±0.04	16.87±3.07	0.29±0.02
	2	94.79±0.08	17.44±4.25	0.29±0.03
	3	93.71±014	17.38±4.06	0.30±0.04
128	1	94.76±0.07	19.87±3.45	**0.24**±0.01
	2	94.86±0.09	19.72±3.01	0.24±0.02
	3	**95.03**±0.63	19.82±3.03	0.25±0.01

between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 5.13, NN achieves highest accuracy of **95.03%**.

5.4.2.2 K Nearest Neighbor (KNN)

Table 5.6: K nearest neighbor performance for gypsum wall dataset

No. of Neighbors	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 neighbors	68.22±0.02	0.29±0.09	0.006±0.001
10 neighbors	73.23±0.43	**0.25**±0.18	**0.004**±0.002
25 neighbors	77.49±0.72	0.31±0.12	0.005±0.008
50 neighbors	80.92±0.11	0.30±0.09	0.005±0.012
75 neighbors	82.22±0.07	0.30±0.20	0.006±0.006
100 neighbors	84.68±0.29	0.27±0.04	0.005±0.013
200 neighbors	**85.68**±0.70	0.29±0.33	0.007±0.014

From Table 5.6, K Nearest Neighbor [56] model performs best for K=200. As the value of K increases, accuracy rises. Being a majority voting algorithm, a large value of K means more number of neighboring points are important in this dataset classification problem. Compared to other algorithms used for this task as seen in Figure 5.13, KNN achieves least accuracy of **85.68%**.

5.4.2.3 Support Vector Machine (SVM)

Table 5.7: SVM performance for gypsum wall dataset

Kernel Types	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
Radial Basis Function (RBF)	**93.41**±0.01	0.68±0.08	0.42±0.28
Linear function	89.84±0.62	**0.24**±0.04	**0.09**±0.01
3rd Order polynomial function	88.28±0.52	0.59±0.07	0.52±0.03
Sigmoid function	93.39±0.00	0.25±0.01	0.13±0.00

From Table 5.7, Support Vector Machine [57] model performs best for 'Radial Basis Function (RBF)' kernel Type. Being a non-probabilistic binary linear classification algorithm, SVM maps training samples to point in space so as to maximize the width of gap between the two categories. Compared to other algorithms used for this task as seen in Figure 5.13, SVM achieves second best accuracy of **93.41%** after NN but it has a considerably lower training time than NN. It is an ideal candidate for this task.

5.4.2.4 Random Forest (RF)

Table 5.8: Random forest performance for gypsum wall dataset

No. of Decision Trees in Random Forest	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 decision trees	69.39±0.03	**0.13**±0.01	**0.01**±0.000
10 decision trees	70.85±0.02	0.23±0.04	0.01±0.002
25 decision trees	81.30±0.01	0.52±0.08	0.02±0.001
50 decision trees	85.90±2.06	0.79±0.11	0.04±0.003
75 decision trees	86.98±1.42	1.11±0.12	0.04±0.000
100 decision trees	89.80±1.28	1.48±0.13	0.06±0.002
200 decision trees	90.25±0.37	2.91±0.37	0.12±0.003
500 decision trees	**92.22**±0.82	7.25±0.45	0.27±0.010

From Table 5.8, Random Forests Classifier [58] model performs best for Number of Trees = 500. Being a Decision Tree combining algorithm, a large number of Decision Trees means it depends on output of many weak learner trees. Thus increasing its training time too. So there should be a trade off between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 5.13, RF achieves third best accuracy of **92.22%**.

5.4.2.5 Confusion Matrix Analysis of All Models for gypsum wall dataset

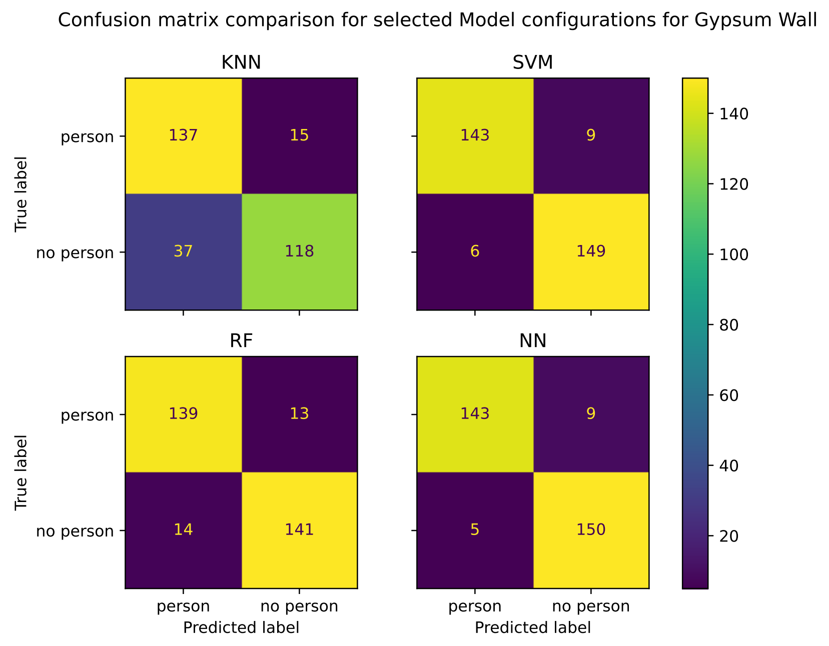


Figure 5.9: Confusion matrix of ML models for gypsum wall dataset

The confusion matrix in Figure 5.9 shows label wise model performance. SVM and NN perform the best for this metric.

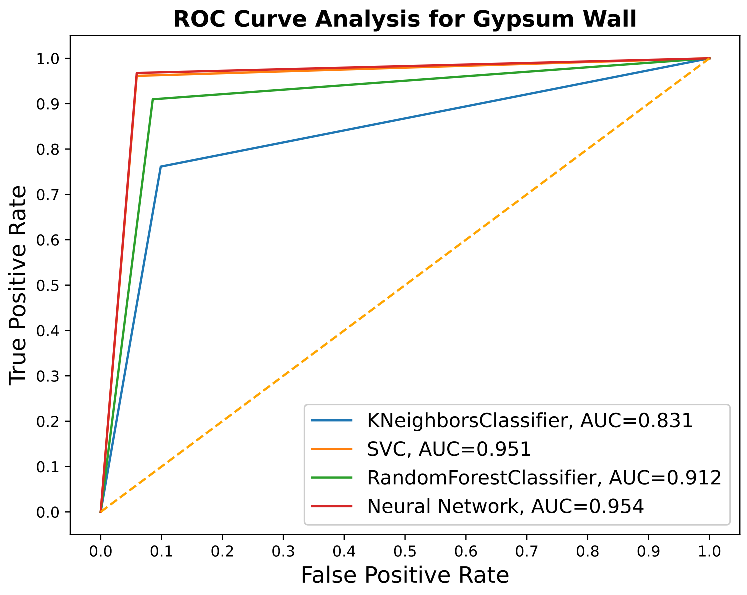


Figure 5.10: ROC curve of ML models for gypsum wall dataset

5.4.2.6 ROC Curve Analysis of All Models for gypsum wall dataset

The ROC curve in Figure 5.10 shows AUC metric for considered models. Again, SVM and NN perform the best for this metric.

5.4.3 Brick Wall Scenario

5.4.3.1 Neural Networks (NN)

Table 5.9: Neural network performance for brick wall dataset

No. of Neurons in each hidden Layer	Number of Hidden Layers	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
	1	**85.18**±0.12	**16.85**±2.83	0.23±0.01
50	2	84.88±0.32	17.06±2.11	0.22±0.00
	3	84.53±0.02	16.89±1.36	0.23±0.05
	1	85.14±0.04	17.05±1.75	**0.21**±0.01
100	2	84.76±0.09	17.91±0.63	0.21±0.02
	3	84.57±0.32	17.99±3.40	0.22±0.02
	1	84.95±0.01	17.82±3.36	0.23±0.01
128	2	84.73±0.18	17.85±3.08	0.22±0.01
	3	84.60±0.03	18.62±4.90	0.23±0.02

From Table 5.9, Neural Network [75] model performs best for Number of Neurons=50 in 1 hidden layer. Being universal function approximator [38] algorithm, a small number of Neurons means it depends on output of few activation paths. Thus having a relatively short training time. Compared to other algorithms used for this task as seen in Figure 5.13, NN achieves highest accuracy of **85.18%**.

5.4.3.2 K Nearest Neighbor (KNN)

From Table 5.10, K Nearest Neighbor [56] model performs best for K=200. As the value of K increases, accuracy rises. Being a majority voting algorithm, a large value of K means more number of neighboring points are important in this dataset classification problem. Compared to other algorithms used for this task as seen in Figure 5.13, KNN achieves third best accuracy of **81.10%**.

Table 5.10: K nearest neighbor performance for brick wall dataset

No. of Neighbors	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 neighbors	64.00±0.02	0.32±0.27	0.008±0.000
10 neighbors	66.63±0.01	0.36±0.01	0.006±0.002
25 neighbors	71.58±0.03	0.32±0.15	0.007±0.008
50 neighbors	75.77±0.02	**0.29**±0.49	0.004±0.003
75 neighbors	77.27±0.01	0.37±0.15	0.005±0.001
100 neighbors	79.01±0.03	0.31±0.18	**0.002**±0.017
200 neighbors	**81.10**±0.01	0.33±0.23	0.002±0.001

5.4.3.3 Support Vector Machine (SVM)

From Table 5.11, Support Vector Machine [57] model performs best for Radial Basis Function (RBF) kernel Type. Being a non-probabilistic binary linear classification algorithm, SVM maps training samples to point in space so as to maximize the width of gap between the two categories. Compared to other algorithms used for this task as seen in Figure 5.13, SVM achieves second best accuracy of **84.38%** after NN but it has a considerably lower training time than NN. It is an ideal candidate for this task.

Table 5.11: SVM performance for brick wall dataset

Kernel Types	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
Radial Basis Function (RBF)	**84.38**±0.00	0.72±0.12	0.48±0.09
Linear function	76.88±0.04	0.46±0.11	**0.10**±0.01
3rd Order polynomial function	74.03±0.09	0.49±0.04	0.45±0.05
Sigmoid function	82.15±0.02	**0.36**±0.04	0.22±0.01

5.4.3.4 Random Forest (RF)

From Table 5.12, Random Forests Classifier [58] model performs best for Number of Trees = 500. Being a Decision Tree combining algorithm, a large number of Decision Trees means it depends on output of many weak learner trees. Thus increasing its training time too. So there should be a trade off between accuracy and training time. Compared to other algorithms used for this task as seen in Figure 5.13, RF achieves lowest accuracy of **80.73%**.

Table 5.12: Random forest performance for brick wall dataset

No. of Decision Trees in Random Forest	Mean Accuracy with std dev (%)	Mean Training Time per Sample (ms)	Mean Testing Time per Sample (ms)
5 decision trees	59.49±0.02	**0.10**±0.01	**0.01**±0.000
10 decision trees	59.97±0.02	0.20±0.03	0.01±0.001
25 decision trees	65.14±0.37	0.49±0.04	0.02±0.004
50 decision trees	71.14±0.02	0.80±0.11	0.03±0.002
75 decision trees	73.33±0.02	1.13±0.14	0.05±0.001
100 decision trees	73.11±0.33	1.42±0.07	0.06±0.002
200 decision trees	76.79±0.15	3.02±0.32	0.13±0.009
500 decision trees	**80.73**±0.14	7.20±0.18	0.30±0.006

5.4.3.5 Confusion Matrix Analysis of All Models for brick wall dataset

The confusion matrix in Figure 5.11 shows label wise model performance. SVM and NN perform the best for this metric.

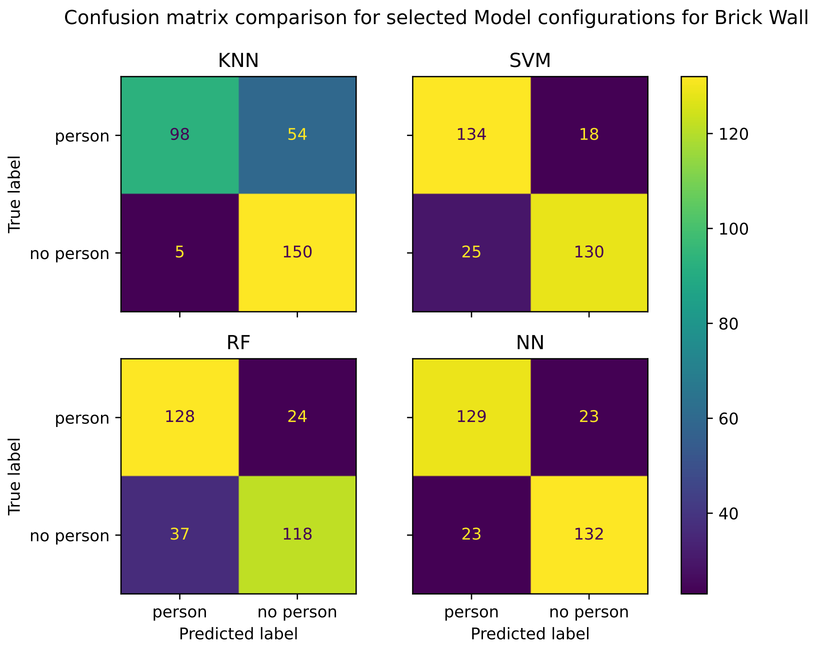


Figure 5.11: Confusion matrix of ML models for brick wall dataset

5.4.3.6 ROC Curve Analysis of All Models for brick wall dataset

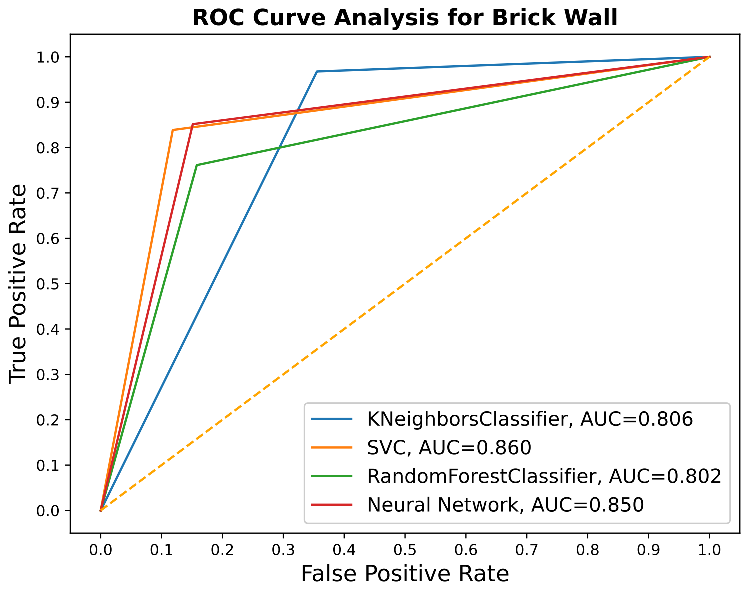


Figure 5.12: ROC curve of ML models for brick wall dataset

The ROC curve in Figure 5.12 shows AUC metric for considered models. Again, SVM and NN perform the best for this metric.

5.4.4 Accuracy Comparison of All Models for Different Scenarios

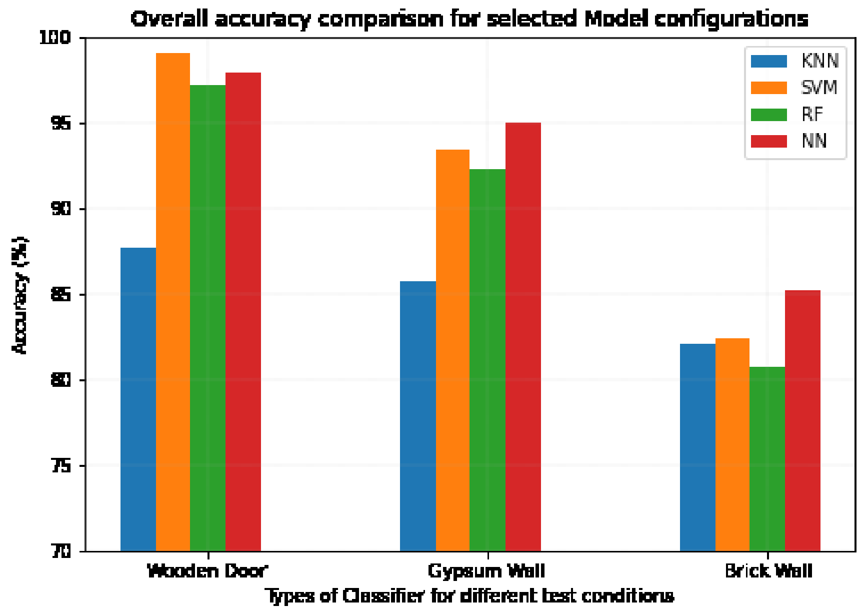


Figure 5.13: Accuracy of ML models for different scenarios

The Accuracy bar chart in Figure 5.13 shows Accuracy performance for considered models in different situations. The models are trained for individuals datasets and not the entire datasets. Sense through wall human detection seems easiest in the wooden door case followed by gypsum wall and is toughest in the brick wall case. All algorithms suffer in the brick wall case. Again, SVM and NN have the best accuracy for all the scenarios with SVM having very low training time.

5.5 Confusion Matrix Analysis of All Models for Entire Wall Dataset

The confusion matrix in Figure 5.14 shows label wise model performance. As the models are trained on more data, the performance improves with less false positives and false negatives. NN perform the best for this metric.

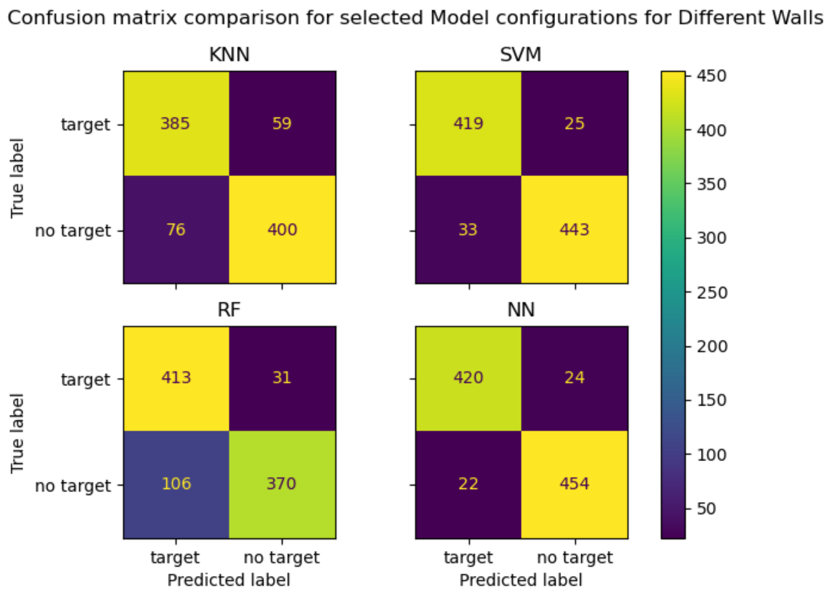


Figure 5.14: Confusion matrix of ML models for entire wall dataset

5.6 ROC Curve Analysis of All Models for Entire Wall Dataset

The ROC curve in Figure 5.15 shows AUC metric for considered models trained on the entire dataset. As the models are trained on more data, the performance improves. NN perform the best for this metric with AUC=**0.966**.

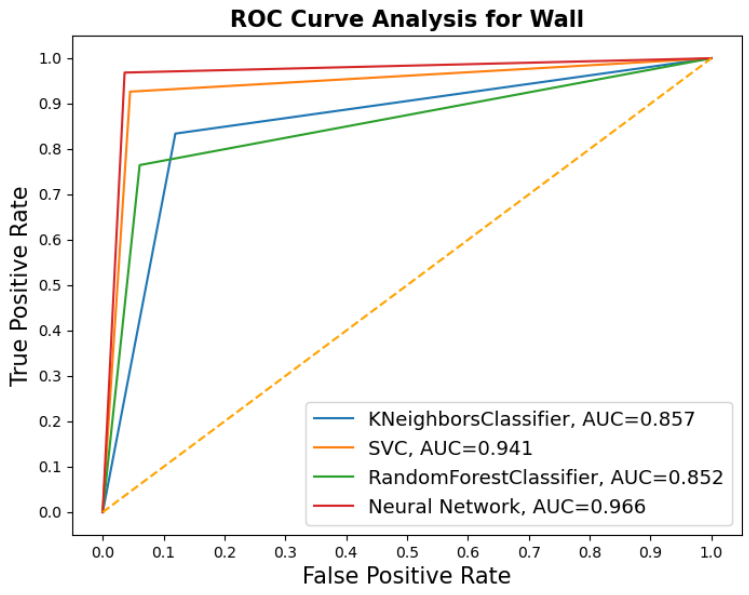


Figure 5.15: ROC curve of ML models for entire wall dataset

5.7 Accuracy Performance of All Models for Entire Wall Dataset

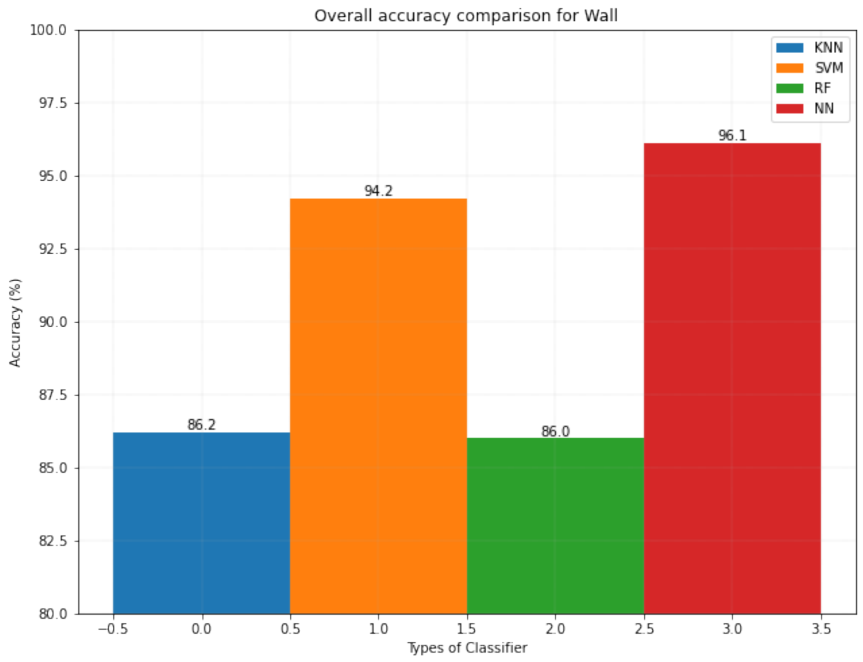


Figure 5.16: Accuracy of ML models for entire wall dataset

The accuracy bar chart in Figure 5.16 shows accuracy performance for considered models trained on entire dataset. KNN and RF perform the worst while NN performs the best with accuracy of **96.1%** with 1 hidden layer and 50 nodes [79].

After extensive experimentation and applying data transformation using our domain expertise we were able to achieve significant performance improvement over other methods. Our Neural Network based ML system is able to detect humans through walls with an accuracy of **96.1**%.

CHAPTER 6

Stacked Machine Learning for Target Detection

In this chapter we use stacking to solve the problem of target detection. Stacking is an ensemble machine learning algorithm where multiple algorithms are applied at multiple levels. Stacking can harness the power of a range of decent-performing models on a regression or classification task which have better performance than any individual model in the ensemble. Typical ensemble methods include bagging and boosting. Bagging [80] has parallel execution while boosting [8] has sequential execution. Stacking is a bit different from bagging and boosting [81]. Unlike bagging, in stacking, the models are typically different for the same dataset. Unlike boosting, in stacking, a single model is used to learn how to best combine the predictions from contributing models. Longer training time is needed since a set of models are considered. For N models, it takes N times more training time in case of sequential algorithms like boosting and stacking. Non-sequential models can be parallelized and do not increase training time. A trade-off between training time, compute power requirement and performance enhancement must be considered.

Reasons to consider ensemble learning:

1. Training a robust, less noisy and more stable model with linear and non-linear relationship learning capability.
2. Increased classification accuracy compared to individual base classifiers.
3. Reduced bias/variance resulting in not overfitted/underfitted models.

6.1 Bagging

Algorithm 4 Bagging

```
1: If $n$ is the number of bootstrap samples
2: for $i = 1, 2, ..., n$ do
3:     Draw bootstrap sample of size $m, \mathbf{D_i}$
4:     Train classifier $h_i$ on $\mathbf{D_i}$
5: end for
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$
```

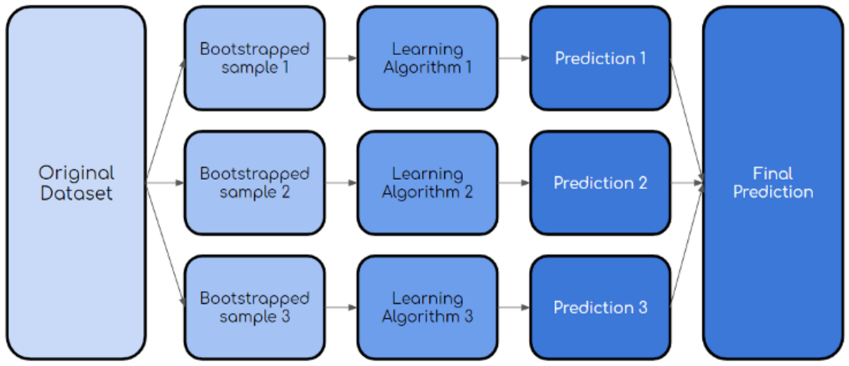


Figure 6.1: Bagging

In bagging [61], we create n bootstrapped samples from the entire dataset which are chosen randomly with replacement. Typically for a dataset with R rows and C columns, $0.67 * R$ rows and $\sqrt{C}$ columns are selected randomly in each bootstrapped sample. These samples are fed to ML models. Mode is used in classification tasks whereas mean is used in regression tasks to make the final prediction. Bagging reduces the bias of any individual model or feature in a dataset as it gives equal importance to all models. As seen in Fig. 6.1 the 3 bootstrapped samples are generated from the original dataset. 3 same or different ML models are trained on these samples. These models train on this data and learn from different features. Final prediction is made with input coming from all the models.

6.2 Boosting

In boosting [61], we create a sequence of weak learners which emphasize on learning on data for which the previous models predicted incorrect. When these models are added, they are weighted in a way that is related to weak learner's accuracy. "Re-weighting" is performed to readjust the data weights. Correctly classified data inputs have higher weights and wrongly classified data inputs have lower weights. There are many types of boosting algorithms like AdaBoost [8], Gradient Boost [82], XGBoost [73], Light GBM [83] and CatBoost [84].

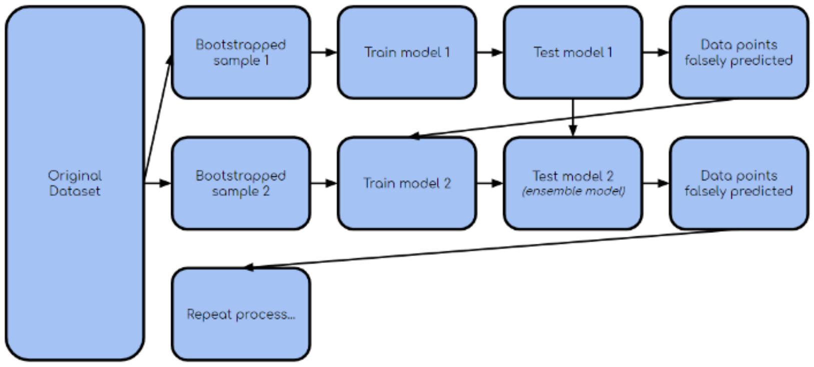


Figure 6.2: Boosting

Algorithm 5 Gradient Boosting

```
1: Input data: ${\langle \mathbf{x}^{(i)}, y^{(i)} \rangle}_{i=1}^{n}$
   Differentiable Loss function: $\mathbf{L}(y^{(i)}, h(\mathbf{x}^{(i)}))$
2: Initialize model
   $$h_0(\mathbf{x}) = \arg\min_{\hat{y}} \sum_{i=1}^{n} \mathbf{L}(y^{(i)}, \hat{y})$$
3: for $t = 1, 2, ..., T$ do
4:     Compute pseudo residual $r_{i,t} = -\left[\frac{\partial \mathbf{L}(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$; for i = 1 to $n$
5:     Fit tree to $r_{i,t}$ values and create terminal nodes $R_{j,t}$ for $j = 1, ..., J_t$
6:     for $j = 1, 2, ..., J_t$ do
           $$\hat{y}_{j,t} = \arg\min_{\hat{y}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} \mathbf{L}(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y})$$
7:     end for
8:     Update
       $$h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} | (\mathbf{x} \in R_{j,t})$$
9: end for
10: Return $h_t(\mathbf{x})$
```

6.2.1 AdaBoost

AdaBoost stands for adaptive boosting, where the miss-classified data samples are weighted more in the new data sample. The model adapts the weights of the data points, hence the name Adaptive boosting. So progressively, models keep becoming better than previous models. The models are trained in sequential model.

6.2.2 Gradient Boosting

Gradient Boosting used loss functions instead of penalizing miss-classified data samples as in AdaBoost. The loss function can be mean squared error for regression or log loss for classification tasks. This boosting algorithm, utilizes gradient descent method [85] to continuously minimize loss function. This enables the algorithm to reach the optimal point with lowest error. Theoretically, it performs better than AdaBoost. It is more susceptible to problems like overfitting and long run-time.

6.2.3 XGBoost

XGBoost stands for Extreme Gradient Boosting. It utilizes the engineering skills to improve speed and performance for gradient boosting methods by pushing the limits of computation resources. XGBoost can be considered as a framework and it has recently won most of the competitions on Kaggle. It uses regularization to overcome the problem of overfitting and speeds up performance by parallelization, cache optimization and out-of-memory computation [73]. It also prunes the decision trees beyond a certain depth which reduces run-time significantly.

6.2.4 Light GBM

Light GBM reduces the run-time of the boosting algorithm by making the computational workload light. It still maintains higher level of model performance compared to other similar algorithms. LightGBM implements a highly optimized histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption. It uses leaf-wise tree growth method instead of level-wise tree growth method.

6.2.5 CatBoost

CatBoost stands for Categorical Boosting. Categorical features can be directly handled in CatBoost without the need of encoding. Its performance is optimal when the features are categorical and it uses a permutation driven alternative.

6.3 Stacking

Stacking utilizes the power of different ML models as different models can learn different patterns from the data. The architecture of a stacked model includes more than one base level (level-0) models and a meta level (level-1) model that combines the predictions of base models. The level-1 model is trained on the predictions made by the contributing level-0 models on unseen data. The base model outputs may be real values in case of regression and class labels or probabilities in case of classification tasks.

Base models can be chosen wisely to make varied assumptions about how to solve predictive modeling task. Some common models of choice are neural networks, linear regression, logistic regression, support vector machines and decision trees. Instead of individual models, ensemble models like random forest can be used as a level-0 model. The performance of base models is better if their predictions are sufficiently uncorrelated.

Meta level models are usually linear models with linear regression for regression and logistic regression for classification tasks. The linear models almost adapts outputs from all base level models as a weighted combination.

Algorithm 6 Stacking

```
 1: Input: training data, $D = \{x_i, y_i\}_{i=1}^{m}$
 2: Output: ensemble classifier H
 3: *Step 1: learn base-level classifiers*
 4: for $t = 1, 2, ..., T$ do
 5:     learn $h_t$ based on $D$
 6: end for
 7: *Step 2: construct new data set of predictions*
 8: for $i = 1, 2, ..., m$ do
 9:     $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
10: end for
11: *Step 3: learn a meta-classifier*
12: learn $H$ based on $D_h$
13: return $H$
```

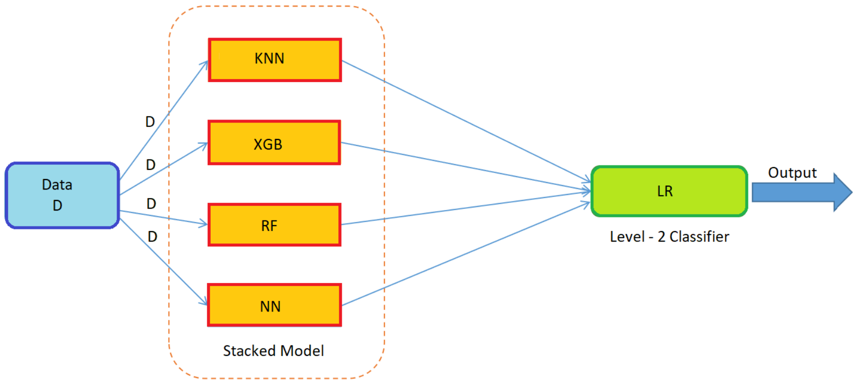


Figure 6.3: Stacking

Fig. 6.3 shows a configuration of stacked machine learning method. We have considered K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Random

Forest (RF) and Neural Networks (NN) as the base models. The meta level model is logistic regression in our case. Same training data D is fed to all base level models which use on different strengths to make predictions on the test data. These predictions make the input to the meta level model which in turn gives the final prediction.

6.4 Results

We conducted multiple experiments to test the performance of different ML models on the foliage target detection dataset. We tried different model configurations but we get the best performance for the stack with KNN (5 neighbors), RF(500 decision trees), XGBoost(100 decision trees) and NN(1 hidden layer with 128 nodes).

6.4.1 Confusion Matrix Analysis of all models for foliage target detection

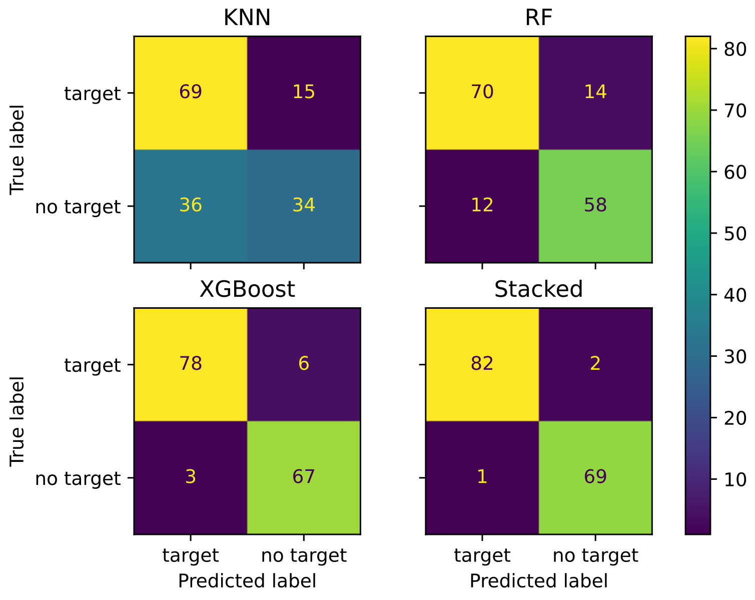


Figure 6.4: Confusion matrix of all ML models for Foliage dataset

The confusion matrix in Fig. 6.4 shows label wise model performance. The stacked ML model performs the best as compared to KNN, RF and XGBoost. The miss classifications are very low in the stacked ML case. The reason for this being the base layer models are able to classify correctly in most test cases and working well as an ensemble.

6.4.2 ROC Curve Analysis of All Models for Foliage Dataset

As shown in Fig. 6.4, Stacked ML model receives the highest AUC of 0.982. It is significantly higher from the next best AUC for XGBoost. Stacked ML model is able to perform well on true positive rate for all thresholds of classification decision.

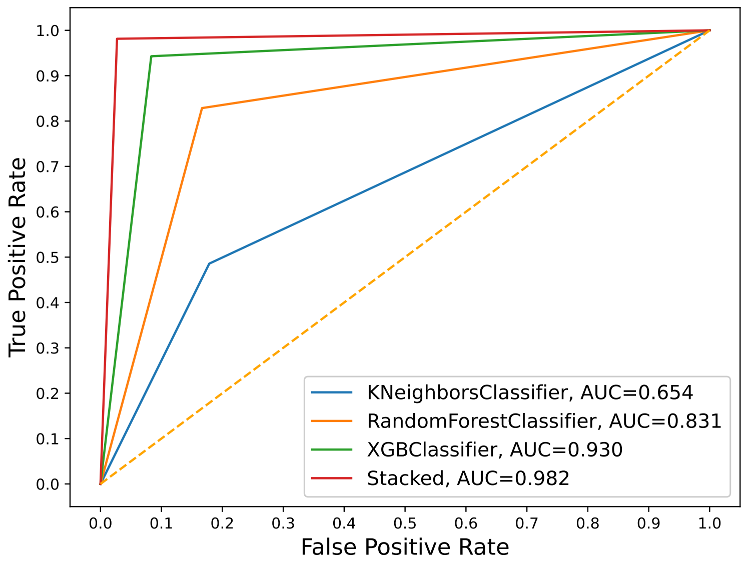


Figure 6.5: ROC curve of all ML models for Foliage dataset

6.4.3 Accuracy Performance of All Models for Entire Wall Dataset

The accuracy bar chart in Figure 6.6 shows accuracy performance for considered models trained on foliage dataset. KNN and RF perform the worst while stacked model performs the best with accuracy of **98.75%**.

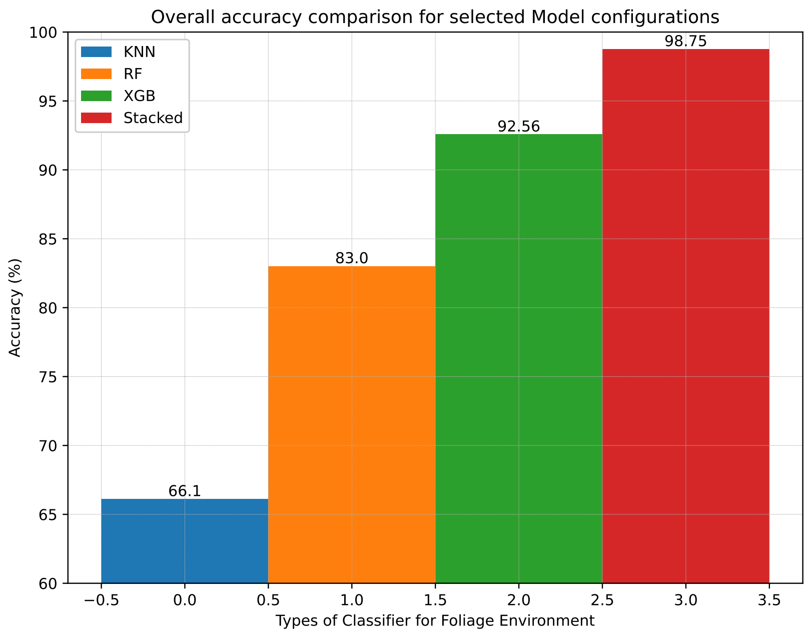


Figure 6.6: Accuracy of all ML models for Foliage dataset

After extensive experimentation and applying data transformation using our domain expertise we were able to achieve significant performance improvement over other methods. Our stacked ML system is able to detect metallic target in foliage situation with an accuracy of **98.75**%.

CHAPTER 7

Conclusion and Future Research

7.1 Conclusions

In the first part of this dissertation, we have proposed a system to detect metallic targets in foliage environment. Mission critical systems need to be ready for the harsh working environment such as dense foliage, water bodies, rain, heavy winds and other natural challenges. Extreme engineering excellence is needed to achieve a faultless system during such critical tasks and routines. To solve this problem, we come up with a Machine Learning system trained on wireless sensor network dataset. Our work consists of four main parts: First, we clean and standardize the dataset. Second, we transform the dataset using IFFT and PCA. Third, we train an XGBoost model on this dataset and make predictions on the test dataset. Finally, we calculate errors by comparing the predicted and actual values and obtain high accuracy with our method [72]. Experiments and plots in Chapter 4 validate these claims. Chapter 4.1 discusses the entire data collection process in the foliage environment. Chapter 4.2 shows the dataset preparation phase of the experiment. Chapter 4.3 details the XGBoost algorithm and its intricacies. Finally, Chapter 4.4 displays the confusion matrix, ROC and accuracy metrics performance for the ML models.

In the second part of this dissertation, we have proposed a system to detect humans through walls. Human detection through walls, doors and corridors is critical in applications such as hostage rescue situation, surveillance, activity recognition, etc. Our work consists of three main parts: First, we clean and standardize the dataset. Second, we train a Neural Network model on this dataset and make predictions on the test dataset. Finally, we calculate errors by comparing the predicted and actual values and obtain high accuracy with our method [79]. Experiments and plots in Chapter 5 validate these claims. Chapter 5.1 discusses the entire data collection process in the foliage environment. Chapter 5.2 shows the dataset preparation phase of the experiment. Chapter 5.3 details the neural network algorithm and its intricacies. Finally, Chapter 5.4 displays the confusion matrix, ROC and accuracy metrics performance for the ML models.

In the last part of this dissertation, we have proposed an ensemble ML system to optimize the first task of target detection in foliage environment. We apply generalized stacked machine learning system to harness the power of different ML models. We get the best accuracy with this method. Chapter 6.1 discusses the first approach of ensemble learning. Chapter 6.2 discusses the second approach of ensemble learning. Chapter 6.3 discusses the last approach of stacked learning. Finally Chapter 6.4 displays the confusion matrix, ROC and accuracy metrics performance for the ML models.

7.2 Future Research

7.2.1 Infuse Multi-Modality Data in the System

Typically, ML does well with more data availability [9]. But more data comes with high compute need so it is a trade off between performance and speed. An amount of data should be selected which gives satisfactory results with optimum resources. Varied data types can provide unique information pertaining to independent dimensionality which can help improve the model's performance.

Reasons to consider multi-modality data are as follows:

1. Modalities have different quantitative influence over the prediction output.

2. As it is well known, ML models are chosen based on the type of data available. Different models are able to extract different patterns from the data. Therefore, it is natural to consider multiple modalities of the data if available.

Limitations of multi-modality data are as follows:

1. The problem with considering multi-modality multi classifier system is that it starts with equal importance to all the sub-networks / modalities which is highly unlikely in real-life situations. A weighted combination of the sub-networks needs to be found so that each input modality can have a learned contribution towards the output prediction.
2. N modalities of data are considered for the task and therefore it requires a much longer time for training. Additionally 2-D data takes more time to train, so trade-off between training time, compute power requirement and performance enhancement must be considered.

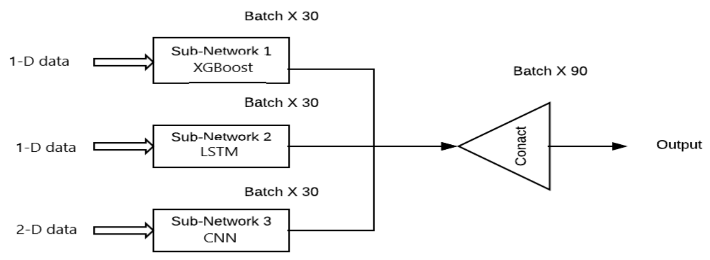


Figure 7.1: Multimodality ML system

7.2.2 Uncertainty with Probabilistic Bayesian Neural Network (BNN)

Feedforward neural networks learn weights as numbers. It does not take into account the uncertainty which can be found in the data or the model. Taking a probabilistic approach to deep learning allows to account for uncertainty, so that models can assign fewer level of confidence to incorrect predictions. Sources of uncertainty can be found in the data, due to measurement error or noise in the labels, or the model, due to insufficient data availability for the model to learn effectively [86].

Reasons to consider BNN are as follows:

1. BNN finds the distribution of weight instead of a single set of weights making it a robust model. By catering to the probability distributions, it can avoid the overfitting problem by addressing the regularization properties.
2. BNN model provides the whole picture towards the prediction which allows to automatically calculate the uncertainties associated with prediction when dealing with unknown targets.

Limitations of BNN are as follows:

1. Since the model architecture is much more sophisticated, BNN trains for the distribution parameters and it requires a much longer time to converge for training.
2. It is challenging to understand all the theories and formulae behind BNN. A solid understanding of statistical distributions is needed so as to apply the appropriate prior and posterior functions.

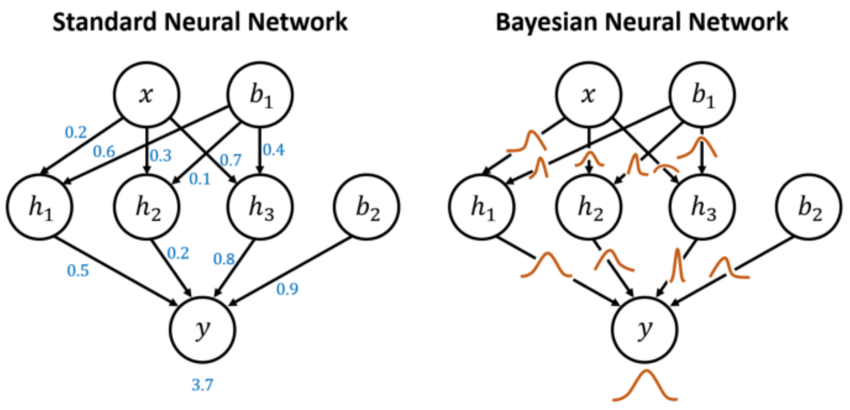


Figure 7.2: Standard & Bayesian neural network

Therefore, considering multi-modality data for target detection in foliage and sense-through-wall human detection will be beneficial and next on the agenda. Uncertainty prediction using Bayesian Neural Network in the sense-through-wall human detection will be one of the focus of next work.

REFERENCES

[1] J. Moor, "The dartmouth college artificial intelligence conference: The next fifty years," *Ai Magazine*, vol. 27, no. 4, pp. 87–87, 2006.

[2] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[3] K. P. Murphy, *Machine learning: a probabilistic perspective.* MIT press, 2012.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[6] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning.* Springer, 2006, vol. 4, no. 4.

[7] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[8] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm," in *icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[10] G. E. Hinton, "To recognize shapes, first learn to generate images," *Progress in brain research*, vol. 165, pp. 535–547, 2007.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[12] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2011, pp. 196–201.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[14] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[15] W. G. Baxt, "Use of an artificial neural network for the diagnosis of myocardial infarction," *Annals of internal medicine*, vol. 115, no. 11, pp. 843–848, 1991.

[16] R. W. Brause, "Medical analysis and diagnosis by neural networks," in *International symposium on medical data analysis*. Springer, 2001, pp. 1–13.

[17] G.-P. Economou, C. Spiropoulos, N. Economopoulos, N. Charokopos, D. Lymberopoulos, M. Spiliopoulou, E. Haralambopulu, and C. Goutis, "Medical diagnosis and artificial neural networks: a medical expert system applied to pulmonary diseases," in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*. IEEE, 1994, pp. 482–489.

[18] S. Kulluk, L. Ozbakir, and A. Baykasoglu, "Training neural networks with harmony search algorithms for classification problems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 1, pp. 11–19, 2012.

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[20] N. Morgan and H. A. Bourlard, "Neural networks for statistical recognition of continuous speech," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 742–772, 1995.

[21] P. M. Atkinson and A. R. Tatnall, "Introduction neural networks in remote sensing," *International Journal of remote sensing*, vol. 18, no. 4, pp. 699–709, 1997.

[22] T. Kavzoglu and P. M. Mather, "Pruning artificial neural networks: an example using land cover classification of multi-sensor images," *International Journal of Remote Sensing*, vol. 20, no. 14, pp. 2787–2803, 1999.

[23] M. Manry, M. Dawson, A. Fung, S. Apollo, L. Allen, W. Lyle, and W. Gong, "Fast training of neural networks for remote sensing," *Remote sensing reviews*, vol. 9, no. 1-2, pp. 77–96, 1994.

[24] M. U. Akram and A. Usman, "Computer aided system for brain tumor detection and segmentation," in *International conference on Computer networks and information technology.* IEEE, 2011, pp. 299–302.

[25] S. M. Bhandarkar, J. Koh, and M. Suk, "Multiscale image segmentation using a hierarchical self-organizing map," *Neurocomputing*, vol. 14, no. 3, pp. 241–272, 1997.

[26] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networks—a review," *Pattern recognition*, vol. 35, no. 10, pp. 2279–2301, 2002.

[27] K. Lee, Y. Cha, and J. Park, "Short-term load forecasting using an artificial neural network," *IEEE transactions on power systems*, vol. 7, no. 1, pp. 124–132, 1992.

[28] K. Liu, S. Subbarayan, R. Shoults, M. Manry, C. Kwan, F. Lewis, and J. Naccarino, "Comparison of very short-term load forecasting techniques," *IEEE Transactions on power systems*, vol. 11, no. 2, pp. 877–882, 1996.

[29] Y. Rui and A. El-Keib, "A review of ann-based short-term load forecasting models," in *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*. IEEE, 1995, pp. 78–82.

[30] J. T. Luxhøj, "An artificial neural network for nonlinear estimation of the turbine flow-meter coefficient," *Engineering Applications of Artificial Intelligence*, vol. 11, no. 6, pp. 723–734, 1998.

[31] T. Parisini and R. Zoppoli, "Neural networks for nonlinear state estimation," *International Journal of Robust and Nonlinear Control*, vol. 4, no. 2, pp. 231–248, 1994.

[32] J. C. Patra, G. Panda, and R. Baliarsingh, "Artificial neural network-based nonlinearity estimation of pressure sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 43, no. 6, pp. 874–881, 1994.

[33] J. Wang and J. Huang, "Neural network enhanced output regulation in nonlinear systems," *Automatica*, vol. 37, no. 8, pp. 1189–1200, 2001.

[34] S. Patil, S. Chintamani, J. Grisham, R. Kumar, and B. H. Dennis, "Inverse determination of temperature distribution in partially cooled heat generating cylinder," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 57502. American Society of Mechanical Engineers, 2015, p. V08BT10A024.

[35] O. Fabela, S. Patil, S. Chintamani, and B. H. Dennis, "Estimation of effective thermal conductivity of porous media utilizing inverse heat transfer analysis on cylindrical configuration," in *ASME International Mechanical Engineering Congress and Exposition*, vol. 58431. American Society of Mechanical Engineers, 2017, p. V008T10A089.

[36] S. Patil, S. Chintamani, B. H. Dennis, and R. Kumar, "Real time prediction of internal temperature of heat generating bodies using neural network," *Thermal Science and Engineering Progress*, vol. 23, p. 100910, 2021.

[37] S. Patil, "Inverse analysis of heat generating body for safety applications," Ph.D. dissertation, The University of Texas at Arlington, 2018.

[38] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[39] H. Guo, S. Li, B. Li, Y. Ma, and X. Ren, "A new learning automata-based pruning method to train deep neural networks," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3263–3269, 2017.

[40] J. Liang, Q. Liang, and S. W. Samn, "A differential based approach for sense-through-foliage target detection using uwb radar sensor networks," in *2008 IEEE International Conference on Communications*, 2008, pp. 1952–1956.

[41] I. Maherin and Q. Liang, "A mutual information based approach for target detection through foliage using uwb radar," in *2012 IEEE International Conference on Communications (ICC)*. IEEE, 2012, pp. 6406–6410.

[42] Q. Liang, "Radar sensor wireless channel modeling in foliage environment: Uwb versus narrowband," *IEEE Sensors Journal*, vol. 11, no. 6, pp. 1448–1457, 2010.

[43] Q. Liang, S. W. Samn, and X. Cheng, "Uwb radar sensor networks for sense-through-foliage target detection," in *2008 IEEE International Conference on Communications*. IEEE, 2008, pp. 2228–2232.

[44] M. Ghavami, L. Michael, and R. Kohno, *Ultra wideband signals and systems in communication engineering*. John Wiley & Sons, 2007.

[45] C. Di, F. Li, and S. Li, "Sensor deployment for wireless sensor networks: A conjugate learning automata-based energy-efficient approach," *IEEE Wireless Communications*, vol. 27, no. 5, pp. 80–87, 2020.

[46] I. Oppermann, M. Hämäläinen, and J. Iinatti, *UWB: theory and applications*. John Wiley & Sons, 2004.

[47] F. C. Commission *et al.*, “First report and order 02-48,” *Washington, DC, USA*, 2002.

[48] Z. Sahinoglu, S. Gezici, and I. Guvenc, “Ultra-wideband positioning systems,” *Cambridge, New York*, 2008.

[49] J. D. Taylor, *Ultra-wideband radar technology.* CRC press, 2000.

[50] “Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans): Amendment 1: Add alternate phys,” *IEEE Std 802.15.4a-2007 (Amendment to IEEE Std 802.15.4-2006)*, pp. 1–210, 2007.

[51] J. Schmidt, M. R. Marques, S. Botti, and M. A. Marques, “Recent advances and applications of machine learning in solid-state materials science,” *npj Computational Materials*, vol. 5, no. 1, pp. 1–36, 2019.

[52] E. Alpaydin, “Introduction to machine learning. 3rd,” 2014.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.

[54] Z. Yan, G. Li, Y. TIan, J. Wu, S. Li, M. Chen, and H. V. Poor, “Dehib: Deep hidden backdoor attack on semi-supervised learning via adversarial perturbation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 585–10 593.

[55] Y. Zhao, S. Li, and F. Jin, “Identification of influential nodes in social networks with community structure based on label propagation,” *Neurocomputing*, vol. 210, pp. 34–44, 2016.

[56] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review/Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989.

[57] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[58] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[59] S. C. Weller and A. K. Romney, *Systematic data collection.* Sage publications, 1988, vol. 10.

[60] Y. Xu and R. Goodacre, "On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning," *Journal of analysis and testing*, vol. 2, no. 3, pp. 249–262, 2018.

[61] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999.

[62] T. Hastie, R. Tibshirani, and J. Friedman, "Unsupervised learning," in *The elements of statistical learning.* Springer, 2009, pp. 485–585.

[63] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[64] C. Di, Q. Liang, F. Li, S. Li, and F. Luo, "An efficient parameter-free learning automaton scheme," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 11, pp. 4849–4863, 2020.

[65] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining.* Springer, 2015, vol. 72.

[66] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality.* John Wiley & Sons, 2007, vol. 703.

[67] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[68] K. Xu, M. Qin, F. Sun, Y. Wang, Y.-K. Chen, and F. Ren, "Learning in the frequency domain," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1740–1749.

[69] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[70] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[71] C. Dill, "Foliage penetration (phase ii) field test: Narrowband versus wideband foliage penetration," *Final report of contract*, no. F41624-03, 2005.

[72] D. Bhole and Q. Liang, "Machine learning enabled sense-through-foliage target detection using uwb radar sensor network," in *Communications, Signal Processing, and Systems*, Q. Liang, W. Wang, X. Liu, Z. Na, and B. Zhang, Eds. Singapore: Springer Singapore, 2022, pp. 1239–1246.

[73] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[74] S. Madeh Piryonesi and T. E. El-Diraby, "Using machine learning to examine impact of type of performance indicator on flexible pavement deterioration modeling," *Journal of Infrastructure Systems*, vol. 27, no. 2, p. 04021005, 2021.

[75] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[76] S. Singh, Q. Liang, D. Chen, and L. Sheng, "Sense through wall human detection using uwb radar," *EURASIP Journal on Wireless Communications and Networking*, vol. 2011, no. 1, pp. 1–11, 2011.

[77] L. E. Miller, "Why uwb? a review of ultrawideband technology," *Report to NETEX Project Office, DARPA. Wireless Communication Technologies Group National Institute of Standards and Technology. Gaithersburg, Maryland*, 2003.

[78] "The gaussian monocycle pulses, fractional derivatives, uwb wireless, and the fcc part 15 spectral masks," 2007.

[79] "Neural network for uwb radar sensor network-based sense-through-wall human detection," in *Communications, Signal Processing, and Systems*, In Publication.

[80] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[81] G. Hackeling, *Mastering Machine Learning with scikit-learn.* Packt Publishing Ltd, 2017.

[82] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[83] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[84] A. V. Dorogush, V. Ershov, and A. Gulin, “Catboost: gradient boosting with categorical features support,” *arXiv preprint arXiv:1810.11363*, 2018.

[85] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[86] D. F. Specht, “Probabilistic neural networks,” *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

BIOGRAPHICAL STATEMENT

Dheeral Bhole was born in Bhusawal, Maharashtra, India in 1990. He received his B.E. degree from University of Pune, India, in 2012. He received his M.S. and Ph.D. degree from The University of Texas at Arlington in 2014 and 2022 respectively, all in Electrical Engineering. From May 2018 to Jan 2019, he was with Rockwell Collins as a software engineer co-op. His current research interests include Machine Learning, Wireless Sensor Networks, Signal Processing and Internet of Things (IoT).