

**MACHINE LEARNING TECHNIQUES FOR AUTOMATIC MODULATION
CLASSIFICATION**

A Thesis
Presented to the
Faculty of
San Diego State University

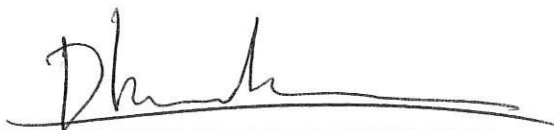
In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Electrical Engineering

by
Isha Choubey
Fall 2017

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the
Thesis of Isha Choubey:

Machine Learning Techniques for Automatic Modulation Classification



Duy Nguyen, Chair
Department of Electrical and Computer Engineering



Ke Huang
Department of Electrical and Computer Engineering



Xiaobai Liu
Department of Computer Science

10/27/2017

Approval Date

Copyright © 2017
by
Isha Choubey
All Rights Reserved

ABSTRACT OF THE THESIS

Machine Learning Techniques for Automatic Modulation Classification

by

Isha Choubey

Master of Science in Electrical Engineering

San Diego State University, 2017

Automatic Modulation Classification (AMC) is concerned with automatically identifying the modulation type of communication signals. AMC is the fundamental component of signal recovery systems and is also employed in jammers in military electronic warfare. Its potential to solve serious issues such as spectral congestion encourages one to develop systems that can quickly and efficiently identify the modulation class of intercepted signals.

This thesis is dedicated to classifying digital signals into one of the eight classes: 8-Pulse shift keying (8-PSK), Binary pulse shift keying (BPSK), Continuous-phase frequency-shift keying (CPFSK), Gaussian frequency-shift keying (GFSK), 4-Pulse amplitude modulation (4-PAM), 16-Quadrature amplitude modulation (16-QAM), 64-QAM and Quadrature phase shift keying (QPSK). The classification task has been accomplished via machine learning techniques. The objective is to study and compare various classifiers for identifying the class of a digitally modulated signal.

Machine learning classifiers k-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forests and Artificial Neural Networks were implemented. The classifiers were trained to perform the task of AMC and their performances were examined and compared with each other. Manual feature engineering was done to train the classifiers. An alternate solution to feature engineering was presented in the form of feature learning from raw data.

TABLE OF CONTENTS

	PAGE
ABSTRACT	iv
LIST OF TABLES	viii
LIST OF FIGURES	ix
GLOSSARY	xiii
ACKNOWLEDGMENTS	xv
CHAPTER	
1 INTRODUCTION	1
1.1 Motivation.....	2
1.2 Thesis Contribution and Outline	4
1.3 Summary	6
2 BACKGROUND ON SIGNAL CLASSIFICATION	7
2.1 Approaches to Signal Classification	8
2.1.1 Likelihood Based Approach.....	8
2.1.2 Feature Based Approach	10
2.1.2.1 Instantaneous Time Domain Features	11
2.1.2.2 Constellation shape.....	11
2.1.2.3 Transformation features	12
2.1.2.4 Statistical features	12
2.1.2.5 Zero-crossing	13
2.2 Contributions In This Field	13
2.3 Summary	17
3 BACKGROUND ON MACHINE LEARNING	18
3.1 Nearest Neighbors Classification	21
3.1.1 K-Nearest Neighbors.....	21
3.1.2 Example of K-Nearest Neighbors Implementation	22
3.2 Support Vector Machine	23
3.2.1 Margin Maximization	23
3.2.2 Kernels	25

3.2.3	Example of Support Vector Machine Implementation	27
3.3	Decision Trees and Random Forests	27
3.3.1	Decision Tree Algorithm	28
3.3.2	Random Forests	30
3.3.3	Bagging and Pasting	31
3.3.4	Boosting.....	31
3.3.5	Example of Decision Tree and Random Forest Implementation	33
3.4	Artificial Neural Network.....	34
3.4.1	Perceptron	34
3.4.2	Multi Layer Perceptron	35
3.4.3	Training a Neural Network- Backpropagation Algorithm	36
3.4.4	The Modern Neural Network.....	37
3.4.5	Deep Neural Network	39
3.4.6	Convolutional Neural Network.....	41
3.5	Summary	45
4	IMPLEMENTATION OF MACHINE LEARNING TECHNIQUES FOR AU- TOMATIC MODULATION CLASSIFICATION	46
4.1	Feature Extraction.....	46
4.2	Classification.....	49
4.2.1	K-Nearest Neighbors.....	50
4.2.1.1	Hyperparameters	50
4.2.1.2	Experimental setup and results	50
4.2.1.3	Summary	53
4.2.2	Support Vector Machine.....	53
4.2.2.1	Hyperparameters	54
4.2.2.2	Experimental setup and results	55
4.2.2.3	Summary	56
4.2.3	Decision Trees and Random Forests	56
4.2.3.1	Decision tree	57
4.2.3.2	Random Forest	60
4.2.3.3	Bagging and pasting.....	61
4.2.3.4	Boosted trees	61

4.2.3.5	Summary	63
4.2.4	Multi Layer Perceptron	64
4.2.4.1	Hyperparameters	64
4.2.4.2	Experimental setup and results	65
4.2.4.3	Summary	66
4.2.5	Artificial Neural Networks	67
4.2.5.1	Hyperparameters	67
4.2.5.2	Summary	74
4.2.6	Convolutional Neural Networks.....	75
4.2.6.1	Summary	80
4.3	Summary	80
4.4	Conclusion and Future Work.....	80
REFERENCES	83

LIST OF TABLES

	PAGE
Table 4.1. KNN Classifiers Performance.....	51
Table 4.2. SVM Classifiers Performance.....	55
Table 4.3. Decision Tree Classifiers Performance	58
Table 4.4. Random Forest Classifiers Performance	60
Table 4.5. Bagging and Pasting Classifiers Performance	61
Table 4.6. Boosted Trees Performance	62
Table 4.7. Multi Layer Perceptron Performance.....	65
Table 4.8. DNN Performance	67
Table 4.9. ANNs - Single Hidden Layer	68
Table 4.10. Deep and Narrow ANN	69
Table 4.11. Funnel Shape for Hidden Layers.....	69
Table 4.12. Same Sized Hidden Layers	70
Table 4.13. DNN in Plain TensorFlow- Feature Set 1	71
Table 4.14. DNN in Plain TensorFlow- Feature Set 2	72
Table 4.15. DNN using Dropout- Feature Set 2	73
Table 4.16. Regularization Applied to DNNs	74
Table 4.17. Single Convolutional and Dense Layer CNNs	76
Table 4.18. CNNs- Varying Width.....	76
Table 4.19. CNNs- Uniform Width.....	76
Table 4.20. Narrow and Deep CNNs	77
Table 4.21. Effect of Various Activation Functions and Weight Initializers	79
Table 4.22. Dropout Applied to Single Convolutional and Dense Layer CNNs	79
Table 4.23. Regularization Applied to Deep CNNs	79

LIST OF FIGURES

	PAGE
Figure 1.1. AMC block diagram: AMC is the stage prior to the demodulator. It comprises of two subsystems, namely preprocessor and classification algorithm. Source: [1].....	1
Figure 1.2. AMC in Military Electronic warfare: Electronic Support system employs AMC to collect information from radio frequency emissions. Source: [4].....	3
Figure 1.3. US frequency allocation chart: A crowded electromagnetic spectrum underlines the need for efficient utilization of the available spectrum. Source: [5].	4
Figure 2.1. Communication signal classifications: Types of classification for communication signals from modulation recognition point of view.....	7
Figure 3.1. Types of machine learning. Source: [50].	18
Figure 3.2. Machine learning model: The predictive model trains on labeled data; then it can be used to make predictions. Source: [49].	19
Figure 3.3. Automation in machine learning: The process of training a machine learning algorithm can be automated. Source: [49].....	20
Figure 3.4. Example of KNN with $k = 5$: The classifier takes majority vote from the data points surrounding the query point, to decide the class of the query point. Source: [50].	22
Figure 3.5. Digits dataset consisting of ten classes. Source: [52].....	22
Figure 3.6. SVM hyperplane and margin: SVM selects the optimal hyperplane by margin maximization. Source: [50].....	23
Figure 3.7. Linear kernel is unfit for complex data: Linear decision boundary fails to separate the two classes. A nonlinear function is required for this classification problem. Source: [50].	25
Figure 3.8. Kernel method: Feature function transforms the feature space into a higher dimension where the classes can be separated by a linear hyperplane. The feature function then transforms the feature space back to the original dimensions. Source: [50].....	26
Figure 3.9. Effect of varying SVM hyperparameters: plots showing the effect of training SVM with three different values of C and three different values of γ . Source: [52].....	28

Figure 3.10. Decision tree: An instance presented at the root node traverses the tree until it reaches one of the leaf nodes- where it finally gets classified. Source: [53].	29
Figure 3.11. Ensemble of decision trees: Bagging involves random sampling with replacement, while pasting is done without replacement. Source: [49].	32
Figure 3.12. Weight update in Adaboost classifier: Each model in the ensemble is trained, after which its weights are boosted for the samples that it misclassified. Source: [49].	33
Figure 3.13. Linear Threshold Unit: Inputs are multiplied by weights to form the net input. Activation function is applied to the net input to generate an output. Source: [50].	35
Figure 3.14. Perceptron decision boundary: A Perceptron can perform only simple binary classification. Source: [50].	36
Figure 3.15. Multi Layer Perceptron: Contains several Perceptrons to enable multi-class classification. Source: [49].	37
Figure 3.16. Activation functions and their derivatives: ReLU activation is the most preferred since its derivative avoids the problem of vanishing and exploding gradients. Source: [49].	38
Figure 3.17. ReLU activated neural network: ReLU is the most commonly used activation function in neural networks. Source: [49].	39
Figure 3.18. Deep Neural Network for object classification: The first hidden layer forms simple features such as edges. The layers that follow, form more complex features. Source: [53].	40
Figure 3.19. Local receptive fields of neurons in visual cortex: Each neuron in a convolutional layer is connected to a few neurons of the previous layer. Source: [49].	41
Figure 3.20. Local receptive fields of neurons in CNN layers: Each neuron has a limited receptive field. Source: [49].	42
Figure 3.21. Convolutional layer structure: Convolutional layers have three dimensions- length, width and height. Source: [54].	43
Figure 3.22. Connections between layers: Computing receptive field of neurons in convolutional layer. Source: [49].	43
Figure 3.23. Convolutional layers with multiple feature maps: A convolutional layer is comprised of several feature maps. All neurons in a feature map share the same weights and bias. Source: [49].	44
Figure 3.24. Example of CNN: A deep CNN used for image classification. Source: [54].	45

Figure 4.1. Performance of KNN Classifier given by randomized search on feature Set 1: The plot shows that the classification accuracy increases as the SNR rises.	52
Figure 4.2. Confusion matrix: Performance of KNN classifier on each modulation class.	53
Figure 4.3. Confusion matrix: Precision values of KNN classifier are computed and confusion matrix visualized for all modulation classes.....	54
Figure 4.4. Performance of SVM classifier on feature Set 2: The plot shows that the classification accuracy increases as the SNR rises.	56
Figure 4.5. Confusion matrix: Precision values of SVM Classifier are computed and confusion matrix visualized for all modulation classes.....	57
Figure 4.6. Performance of Decision Tree classifier from grid search on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.	59
Figure 4.7. Performance of Random Forest classifier From randomized search on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.....	60
Figure 4.8. Performance of Pasting ensemble classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.	62
Figure 4.9. Performance of Gradient Boosted ensemble classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.	63
Figure 4.10. Confusion matrix: Precision values of Gradient Boosted classifier are computed and confusion matrix visualized for all modulation classes.	64
Figure 4.11. Performance of three hidden layer [16,256,128] MLP classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.	66
Figure 4.12. Performance of five hidden layer [640,320,160,80,16] DNN on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.	68
Figure 4.13. Performance of [16,16] hidden layer DNN on feature set 1: The DNN uses Leaky ReLU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.	71
Figure 4.14. Performance of [32,32] hidden layer DNN on feature set 2: The DNN uses ELU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.	72

Figure 4.15. Performance of regularized [256,256] hidden layer DNN on feature set 2: The DNN uses ReLU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.	73
Figure 4.16. Confusion matrix: Precision values of DNN are computed and confusion matrix visualized for all modulation classes.	75
Figure 4.17. Convolutional Neural Network with [256,256,256] convolutional layers and [256,256,256] fully connected layers: The plot shows that the classification accuracy increases as the SNR rises.	77
Figure 4.18. Confusion matrix: Precision values of CNN are computed and confusion matrix visualized for all modulation classes.	78
Figure 4.19. Classifier comparison: Comparison of feature learning vs. feature engineering approach. CNN is based on feature learning, while rest of the classifiers are based on feature engineering.	81

GLOSSARY

ALRT	Average Likelihood Ratio Test
AM-DSB	Amplitude Modulation Double-sideband
AMC	Automatic Modulation Classification
AM-SSB	Amplitude Modulation Single-sideband
ANN	Artificial Neural Network
AWGN	Additive White Gaussian Noise
BPSK	Binary pulse shift keying
CC	Cyclostationarity
CNN	Convolutional Neural Network
CNR	Carrier to Noise Ratio
COMINT	Communication Intelligence
CPFSK	Continuous-phase frequency-shift keying
CR	Cognitive Radio
CW	Continuous Wave
dB	Decibel
DNN	Deep Neural Network
DT	Decision Tree
ES	Electronic Support
EW	Electronic Warfare
FB	Feature Based
FCC	Federal Communications Commission
FSK	Frequency-shift keying
FT	Fourier Transform
GFSK	Gaussian frequency-shift keying
GLRT	Generalized Likelihood Ratio Test
HLRT	Hybrid Likelihood Ratio Test
HOM	Higher Order Moment
HOS	Higher Order Statistics
KNN	k-Nearest Neighbors
LB	Likelihood Based
MFSK	Multiple frequency-shift keying
MLP	Multi Layer Perceptron
NTN	Neural Tree Network

OFDM	Orthogonal Frequency Division Multiplexing
OQPSK	Orthogonal quadrature phase shift keying
PAM	Pulse amplitude modulation
PDF	Probability Density Function
PSD	Power Spectral Density
PSK	Pulse shift keying
QAM	Quadrature amplitude modulation
QHLR	Quasi Hybrid Likelihood Ratio
QPSK	Quadrature phase shift keying
QLLR	Quasi-Log Likelihood Ratio
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RML	Radio Machine Learning
RNN	Recurrent Neural Network
RF	Random Forest
SD	Sphere Decoding
SDR	Software Defined Radio
SNR	Signal to Noise Ratio
SVM	Support Vector Machine
WBFM	Wideband Frequency Modulation
WT	Wavelet Transform
ZC	Zero-crossing

ACKNOWLEDGMENTS

I am deeply grateful to Prof. Duy Nguyen, who has made my research at SDSU an invaluable experience. His knowledge and guidance have wonderfully shaped my research and made my thesis a success. I truly appreciate the inputs that he offered to help me tackle the challenges in this research. I am glad to have been receiving his support throughout my academic career, in the form of all the resources I required- I certainly could not have succeeded without it. But most worthwhile have been his expert views for finding efficient solutions to the problems I faced. His immense knowledge in the area of communication technology was a vital element for the accomplishment of this project.

I would like to express my sincere gratitude to Prof. Ke Huang for funding my research. He has been truly supportive and a wonderful research guide as well. I am thankful to him for the valuable research opportunities he offered.

I would also like to thank Prof. Xiaobai Liu for his interest in my research, for his time and commitment to my thesis and for being a member of my thesis committee.

CHAPTER 1

INTRODUCTION

Communication signals differ from each other by frequency and modulation type. It is required to monitor these signals with the objective of recognizing the signal characteristics- one of the most essential one being the modulation type. A signal's modulation type signifies how the signal carries the information. Identifying the type of modulation is what forms the problem of modulation classification [1]. An Automatic Modulation Classification (AMC) system is composed of two subsystems- a preprocessor and classification algorithm.

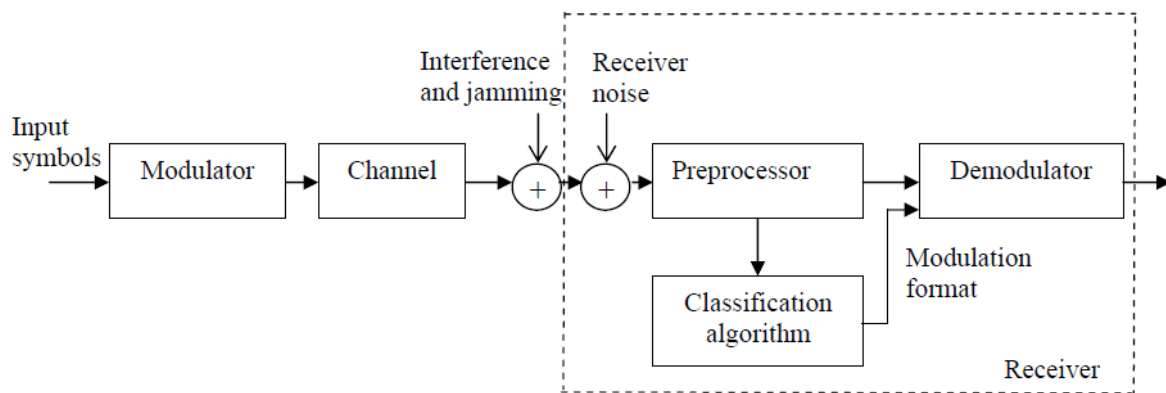


Figure 1.1. AMC block diagram: AMC is the stage prior to the demodulator. It comprises of two subsystems, namely preprocessor and classification algorithm. Source: [1]

Figure 1.1 shows the block diagram of a modulation classifier, which can be broken down into two processes: signal preprocessing and selection of classification algorithm.

1. **Preprocessor:** This stage deals with extracting features from the received signal. Examples are: features based on amplitude and phase information of the received signal. These features thus convert the raw data into patterns that the classifier must learn to identify.
2. **Classification Algorithm:** For each signal that the preprocessor receives, the classification algorithm takes the features formed by the previous stage as the input, and outputs the modulation type of that signal. The approaches to solve the problem of AMC are mainly of two kinds. The first one is decision theoretic approach that uses probabilistic and hypothesis testing arguments. The second one, and the one

employed in this thesis, is statistical pattern recognition- that relies on the structure in patterns of signals, in order to identify their modulation type. In this technique, the samples or the instances belonging to the same modulation class form a pattern. We train the pattern recognizer to determine the modulation type of a new instance or signal. The challenge then, lies in building and training this classifier.

1.1 MOTIVATION

AMC is of great importance due to its several applications. It is a fundamental part of jammers, threat analysis systems and surveillance used in the military. It has essential civilian applications too- including intelligent communication systems such as Software Defined Radio (SDR) and Cognitive Radio (CR) [2].

If the modulation type of a signal is unknown, and it is applied to an improper demodulator, the information content of the signal could be damaged- hence rendering the communication signal useless. This has a negative effect on the deciphering process [2].

AMC is the stage that converts the demodulated signal, which is in its non-intelligible form to intelligible form [2]. Hence, modulation classification is the key to selecting the appropriate demodulator. Once the classifier correctly identifies the modulation type, demodulation of the signal can be performed, followed by other operations such as information extraction. When AMC is used to identify the signal's modulation class at the receiver, we can create generic receivers so that there is only one receiver design for several types of transmitters [3].

The earliest motivation behind AMC was its application in the military [4]. Identification of adversary transmitting units was required in communication jammers in order to generate energy efficient jamming signals [3] and then recover the intercepted signal.

Electronic Support (ES) is one of the subsystems in Electronic Warfare (EW) [4]. Figure 1.2 shows the structure of the system typically employed in ES. The objective of this system is to collect information from radio frequency emissions. The modulation type detected by AMC used in ES is then used to demodulate the intercepted signal, so that the transmitted message can be recovered among the adversary units. The modulation information also serves an essential purpose to electronic mapping system for identifying adversary units along with their possible locations [4].

Modulation classification is the step prior to signal detection and demodulation. It is the task that an intelligent receiver is supposed to perform. This step is vital for efficient implementation of information services in a crowded electromagnetic spectrum. The radio spectrum is considered a precious resource. As the number of wireless applications

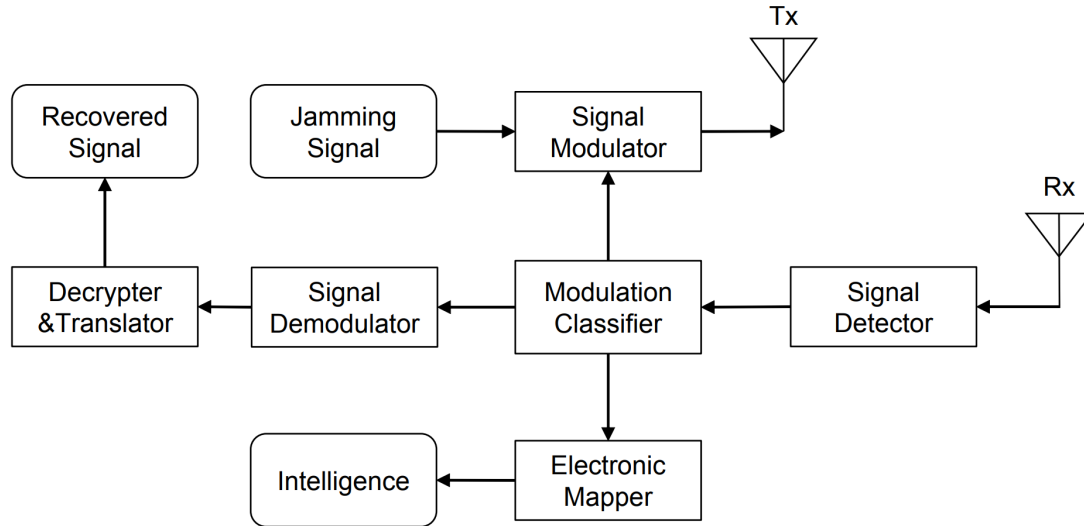


Figure 1.2. AMC in Military Electronic warfare: Electronic Support system employs AMC to collect information from radio frequency emissions. Source: [4].

and services increase, the need to find the spectrum for them also rises [2]. Such spectral congestion emphasizes the need for utilizing the available spectrum in the most efficient way possible.

Figure 1.3 illustrates the spectrum occupancy for different wireless application [5]. Given how congested the spectrum is, one solution to this problem is AMC. Communication engineers are faced with the challenge of securely transmitting and receiving friendly signals. Moreover, hostile signals have to be located, identified and jammed [2]. A need therefore arises for advanced techniques that are capable of real-time signal interception and processing.

SDR [6] deals with various kinds of communication systems. Radio systems are incorporating dynamically configurable software into SDR design. As discussed above, spectrum congestion is a serious issue. Federal Communications Commission (FCC) encourages the use of CR as a solution. A CR has the ability to modify its transmitter parameters depending upon interactions in the environment in which it is operating. FCC considers adaptive modulation as a desired capability of CR. Adaptive modulation exploits opportunities and tries to use the spectrum efficiently, by changing transmission waveforms and characteristics. Commercial systems like SDR use blind techniques along with an intelligent receiver. This reduces the overhead, thus increasing the transmission efficiency. SDR needs quick processing for seamless demodulation. AMC is required in SDR, in order to enable adaptive modulation.

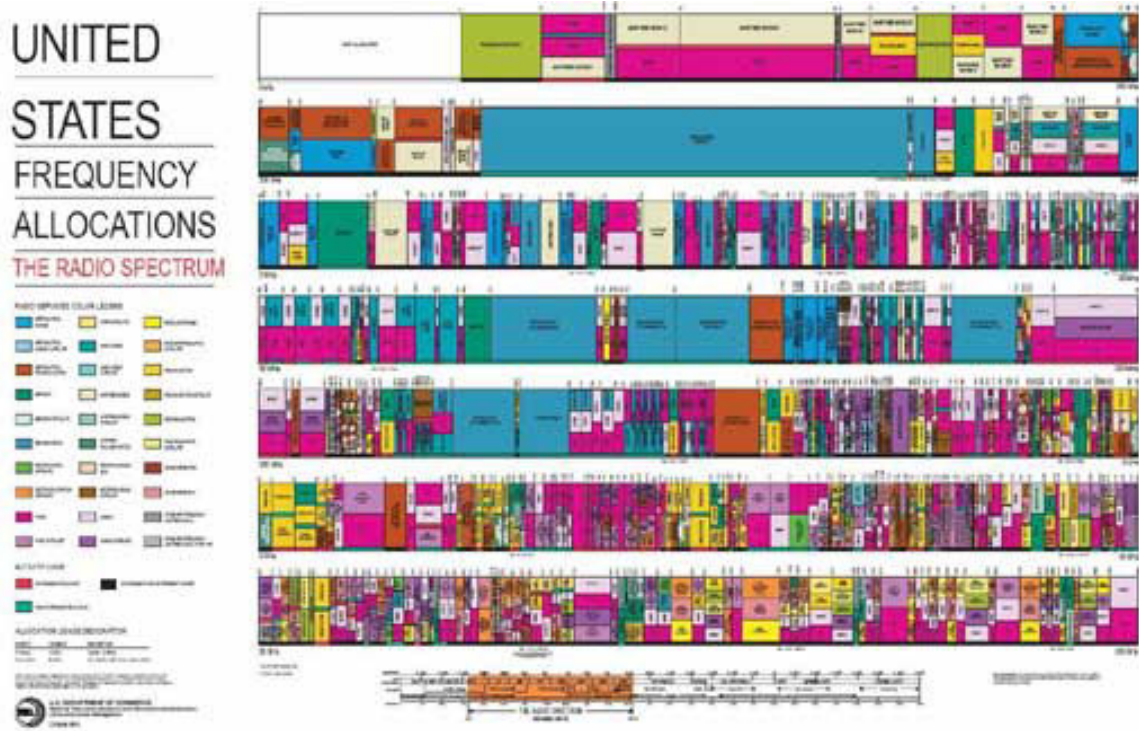


Figure 1.3. US frequency allocation chart: A crowded electromagnetic spectrum underlines the need for efficient utilization of the available spectrum. Source: [5].

1.2 THESIS CONTRIBUTION AND OUTLINE

The thesis is divided into four chapters and is organized as follows.

Chapter 2 is concerned with the techniques for signal classification. The first section of this chapter presents a detailed description of the two major approaches utilized for AMC, namely Likelihood Based (LB) and Feature Based (FB) approach. The second section mentions significant contributions and results for each of these approaches.

Chapter 3 considers various machine learning techniques for the problem of classification. It starts with a nearest neighbors classifier called k-Nearest Neighbors. It then introduces other classification algorithms, namely Support Vector Machine, Decision Trees, Random Forests and Artificial Neural Networks. Each of these sections ends with an example of the classifier's implementation on digits dataset, which is a dataset containing several images of digits that are required to be classified as one of the digits ranging from 0 to 9.

Chapter 4 is concerned with all the experiments conducted in this research. It starts by describing the dataset used to solve the problem of signal classification. The problem at hand is a supervised learning problem, which implies that the dataset consists of signals labeled by their modulation types. The task is to train a classifier to identify the

modulation type of a given signal. Each classifier is trained on 50% of the samples- called training set and tested on the remaining samples- called test set.

The first section considers the first stage of AMC, namely feature extraction. It explains all the cyclic-moment based features designed for the purpose of training a modulation classifier. The second section is concerned with the second stage of AMC, namely classification. It mentions the experimental setup for each machine learning classifier, followed by a summary of the results of implementation.

The thesis studies the classification of signals based on modulation types, by using both carefully engineered features and raw data as well. Eight digital modulation types: 8-PSK, BPSK, CPFSK, GFSK, 4-PAM, 16-QAM, 64-QAM and QPSK have been considered. Feature engineering was done to create a total of 32 features to train various classifiers to perform the task of AMC. The first 16 features are based on 0 cyclic time lag. Each raw signal was therefore transformed to 16 dimensional feature vector and the classifiers were trained on these 16 features.

The feature set was then expanded to improve the performance of the predictive models. 16 features based on 8 cyclic time lag were designed and the classifiers were trained on 32 features. The classifiers k-Nearest Neighbors, Support Vector Machine, Decision Trees, Random Forests, Adaboost, Gradient Boosted trees and Artificial Neural Networks were trained. These models were implemented in the machine learning framework scikit-learn. Each classifier was trained on the first 16 features and later trained on all the 32 features. Hyperparameter optimization was performed via grid search and randomized search for each of these classifiers.

Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) were implemented in the machine learning library TensorFlow. The DNN models were trained on the same feature sets as the models trained in scikit-learn. Artificial Neural Networks (ANNs) comprising of a single hidden layer were initially implemented, followed by DNNs consisting of multiple hidden layers. Various DNN architectures were implemented by modifying the width and depth of the network, along with different weight initializers and activation functions. Techniques such as regularization and early stopping were used to improve the performance of the classifier.

The CNNs were trained on raw data. CNNs were implemented with the objective of eliminating the need for feature engineering. Handcrafted features may not prove best for the task at hand. The time and effort spent in designing features can be saved by constructing models that learn from raw data. Unlike the previous modulation classifiers, CNN classifiers do not rely on designed features, but rather learn to recognize patterns in the raw data itself. CNNs differing in width, depth, activation functions and weight

initializers were trained. Deep CNNs consisting of multiple convolutional and fully connected layers were extensively investigated.

Performance of the various classifiers was evaluated and compared via metrics such as accuracy on the test set, training duration and confusion matrix. All the classifiers were tested on samples belonging to 20 Signal to Noise Ratio (SNR) values ranging from -20dB to 18dB, in steps of 2dB. The accuracy is highly dependent on the SNR. Hence, a separate classification accuracy was computed for every SNR value. A confusion matrix was computed and visualized for every classifier in order to examine the classifier's performance on the individual classes.

1.3 SUMMARY

This chapter has presented an overview of AMC, the motivation behind the thesis and summary of the work done in the thesis. The introduction to the chapter described the problem of signal classification and the major steps that AMC comprises of. This was followed by the essential applications of AMC in the military, spectrum management, SDR and signal recovery. The second section presented an outline of the chapters in the thesis. The last part of this section gave a short description of the features designed, followed by an overview of the classifiers implemented.

CHAPTER 2

BACKGROUND ON SIGNAL CLASSIFICATION

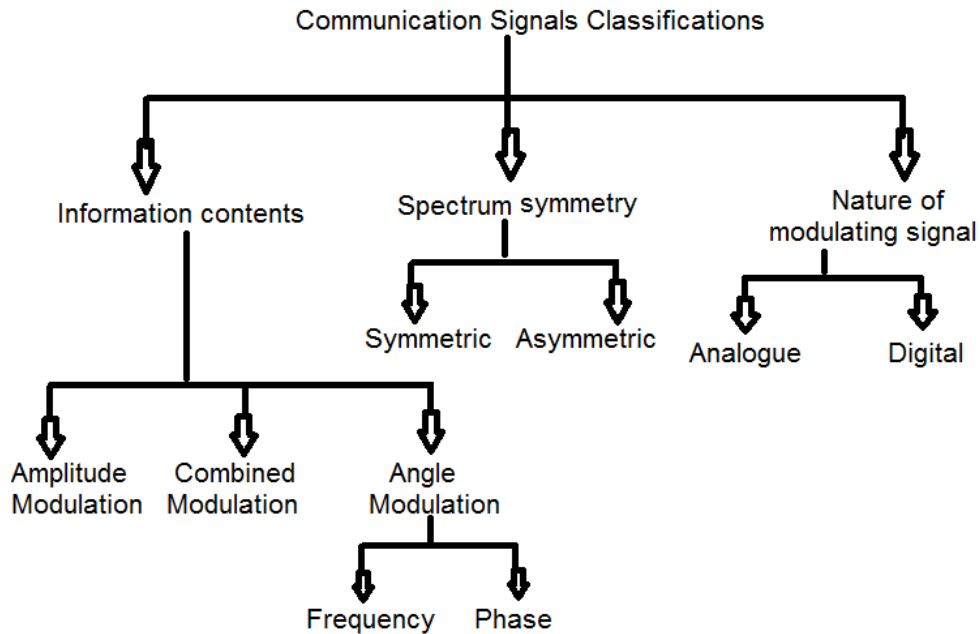


Figure 2.1. Communication signal classifications: Types of classification for communication signals from modulation recognition point of view.

Figure 2.1 shows that many kinds of classifications of communication signals are possible when we need to identify the modulation type of a signal. The first kind of classification is according to information content of the signal [7]. The second type of classification is according to spectrum efficiency of the signal around its carrier frequency. Finally, the third one is according to the modulating signal's nature. This thesis carries out signal classification of the first type due to its several essential applications. Identifying the modulation type of signals is required to implement jammers used for surveillance in the military [4]. Knowledge of the modulation type facilitates in the selection of the appropriate demodulator, so that the information can be safely recovered from a modulated signal [7]. Knowledge of the modulation type is also used in CR to mitigate the problem of spectral congestion. [2]

The objective of this research is to identify where the intercepted signal carries the useful information. We identify whether the communication signal contains information in instantaneous phase, instantaneous amplitude or instantaneous frequency.

Modulation identification plays a key role in Communication Intelligence (COMINT) applications. For the monitoring of intercepted signals, one of the characteristics that need to be identified is the modulation type. Earlier, COMINT systems relied on operators that manually identified the signal modulation type. But later, modulation identifiers were developed that automatically identified the modulation types of an incoming RF signal [8]. AMC is much more efficient than manual identification because it increases the operator's efficiency to monitor activities in the frequency band of interest.

2.1 APPROACHES TO SIGNAL CLASSIFICATION

More than two decades of research on AMC has given rise to a large number of algorithms [2]. The techniques developed for signal classification are different from each other, but they can be broadly classified into two categories. The two general classes of AMC algorithms are:

- Likelihood Based approach
- Feature Based approach

2.1.1 Likelihood Based Approach

LB approach utilizes the likelihood function of the received signal in order to identify its modulation type. Signal classification is done by forming likelihood ratios and then deciding the modulation type by comparing the ratio against a threshold. LB methods involve two steps [2]:

- Computing the likelihood function of the received signal for each of the candidate modulations.
- Deciding the modulation class by maximum likelihood ratio test.

The received signal is of the form [9]

$$r(t) = s(t) + n(t), \quad (2.1)$$

where $s(t)$ is the signal radiated by a transmitter, while $n(t)$ is Additive White Gaussian Noise (AWGN) with two sided Power Spectral Density (PSD) of $\frac{N_0}{2}$. The transmitted signal $s(t)$ is typically modelled as [9]

$$s(t) = A(t) \cos(\omega_c t + \theta_c + \phi(t)) = \text{Re}\{A(t)e^{j\phi(t)}e^{j(\omega_c t + \theta_c)}\}, \quad (2.2)$$

where $A(t)$ is the modified amplitude, $\phi(t)$ is the modified phase, ω_c is the carrier frequency and θ_c is unknown phase offset. While the complex envelope of $s(t)$ is given by [9]

$$\tilde{s}(t) = A(t)e^{j\phi(t)}. \quad (2.3)$$

The received signal belongs to one of the N modulation types. Let integers $i = 0, 1, \dots, N-1$ denote the candidate modulation classes. The event that the received signal falls into modulation class i is denoted by m_i . Also, we assume equal a priori probability $P[m_i]$ that the modulation i is used.

LB methods involve estimating the unknown parameters to identify the signal's modulation type. We assume that the parameters carrier frequency ω_c , symbol rate R_s , timing offset t_d , carrier phase θ_c , pulse shape $p(t)$ and SNR γ have been estimated. The demodulated matched filter output can be expressed as [9]

$$\tilde{r}_n = \int_{-\infty}^{\infty} r(t)e^{-j(\omega_c t + \theta_c)} \sqrt{\frac{2}{E_p}} p(t - nT_s - t_d) dt = r_{I,n} + jr_{Q,n}, \quad (2.4)$$

where $R_s = \frac{1}{T_s}$ is the symbol rate, t_d is an unknown timing offset, $r_{I,n}$ and $r_{Q,n}$ are the inphase and quadrature components respectively, $p(t)$ is the pulse shape and E_p is the pulse energy.

The Probability Density Function (PDF) for the demodulated symbols r_n , given that the modulation class is m_i [9], can be expressed as

$$p_{\tilde{r}}(r_{I,n}, r_{Q,n} | m = m_i) = \frac{1}{M(i)} \sum_{k=1}^{M(i)} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{I,n} - \sqrt{E_s}\mu_{I,k})^2 + (r_{Q,n} - \sqrt{E_s}\mu_{Q,k})^2}{2\sigma^2}}, \quad (2.5)$$

where $\mu_{I,k} = \text{Re}\{\mu_k\}$, $\mu_{Q,k} = \text{Im}\{\mu_k\}$ and $\sigma = \frac{N_0}{2}$.

The N demodulated symbols can be expressed in vector form as follows [9]

$$\tilde{r} = r_I + jr_Q = [\tilde{r}_1, \tilde{r}_2, \dots, \tilde{r}_N]. \quad (2.6)$$

The PDF governing the received signal \tilde{r} is [9]

$$p_{\tilde{r}}(r_I, r_Q | m = m_i) = \prod_{n=1}^N p_{\tilde{r}}(r_{I,n}, r_{Q,n} | m = m_i). \quad (2.7)$$

The maximum likelihood rule then uses the PDF given by equation 2.7 to make a decision about what modulation type the received signal belongs to. It selects modulation $m = m_k$ only if $p_{\tilde{r}}(r_I, r_Q | m = m_i)$ is maximum for $i = k$.

Three types of LB techniques have been presented in the literature: Average Likelihood Ratio Test (ALRT) [10]–[14], Generalized Likelihood Ratio Test (GLRT) [15]–[17] and Hybrid Likelihood Ratio Test (HLRT) [15], [18].

In ALRT approach, the unknown quantities are treated as random variables with certain PDFs. For the complex valued received signal $\mathbf{r} = [r(1), r(2), \dots, r(N)]$ observed over N samples, the likelihood function of this signal is expressed as [14]

$$\Lambda_i^{ALRT}(\mathbf{r}) = \mathbb{E}_{\{x\}_{i=1}^M} \mathbb{E}_{\Theta}[p(\mathbf{r}|\{x\}_{i=1}^M, \Theta)], \quad (2.8)$$

where Θ is the vector of unknown parameters, $\mathbb{E}_x[.]$ and $\mathbb{E}_{\Theta}[.]$ are expectations with respect to unknown constellation points of the signal.

In GLRT based approach, the unknown parameters are considered as unknown deterministic [2]. The likelihood function under hypothesis H_i is given by

$$\Lambda_G^{(i)}[r(t)] = \max_{\mathbf{v}_i} \Lambda[r(t)|\mathbf{v}_i, H_i], \quad (2.9)$$

where \mathbf{v}_i is the vector of unknown quantities. ALRT required multidimensional integration, which is difficult to compute when a large number of quantities are present. This disadvantage is overcome by GLRT, that uses multidimensional maximization instead.

HLRT is a combination of ALRT and GLRT. This technique benefits from the merits of ALRT and GLRT, and simultaneously overcomes the demerits of both methods. HLRT computes the average over unknown data symbols that removes nested constellations- a problem in GLRT based AMC. The likelihood function under H_i is given by [2]

$$\Lambda_H^{(i)}[r(t)] = \max_{\mathbf{v}_{i_1}} \int \Lambda[r(t)|\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, H_i] p(\mathbf{v}_{i_2}|H_i) d\mathbf{v}_{i_2}, \quad (2.10)$$

where \mathbf{v}_{i_1} and \mathbf{v}_{i_2} are vectors of unknown quantities.

2.1.2 Feature Based Approach

FB approach extracts features from the received signal and inputs these features to a classification algorithm. Research in the area of modulation classification is now dominated by FB methods. This is due to their higher performance and lower processing complexity as compared to the LB methods. FB algorithms require a set of features for data representation, that they use for decision making in order to identify the modulation type of a signal. The following reasons describe in detail why FB is the preferred modulation recognition algorithm [2]:

- LB algorithms offer optimal solution in Bayesian sense- they minimize the probability of false classification. Though this solution is optimal, it is computationally complex at the same time- resulting in suboptimal classifiers. FB methods may not be optimal, but they are simpler to implement. Their performance is near optimal when they are designed properly. The computational complexity is lower as compared to LB methods.

- LB methods are not robust in terms of residual channel effects, non-Gaussian distributions, timing errors, as well as phase and frequency offsets.
- In a two hypothesis problem such as 8-QAM vs. 16-QAM, the LB approach will be inclined towards the latter constellation since it is denser.

FB algorithms utilize some features of the signal for decision making. Examples of these features are [2]: information extracted from phase, amplitude, frequency information of the signal [7], phase PDF [19], statistical moments [20], cumulants, moments, cyclic cumulants of the signal [21]–[24] and several others. The most commonly used features in the application of AMC can be classified as one of the following five types [3]:

- Instantaneous time domain features
- Constellation shape
- Transformation features
- Statistical features
- Zero-crossing

2.1.2.1 INSTANTANEOUS TIME DOMAIN FEATURES

Such features are based on the amplitude, phase or frequency of the signal. Instantaneous version of these parameters have been used quite often to implement AMC [1], [25], [26]. The instantaneous amplitude for a received signal $r(t)$ can be expressed as [3]

$$a(t) = \sqrt{r^2(t) + \hat{r}^2(t)} = |z(t)|, \quad (2.11)$$

where \hat{r} is the Hilbert transform of $r(t)$. The instantaneous phase is [3]

$$\phi(t) = \text{unwrap}(\text{angle}(z(t))) - 2\pi t f_c t, \quad (2.12)$$

where $\text{unwrap}(\text{angle}(z(t)))$ corrects the radian phase angles in the vector $z(t)$ given by equation 2.11.

The instantaneous frequency is given by [3]

$$f_N = \frac{1}{2\pi} \frac{d(\text{arg}(z(t)))}{dt}. \quad (2.13)$$

2.1.2.2 CONSTELLATION SHAPE

The geometric shape of constellation diagrams of QAM and PSK signals differ from each other [3] due to the number of points and location of these points in the diagrams. Features such as Euclidean distance between constellation points and number of sets of equal amplitude have been utilized for AMC [27].

2.1.2.3 TRANSFORMATION FEATURES

Transformation features are usually designed by transforming the signal via Fourier Transform (FT), Wavelet Transform (WT), or combination of the two [3]. The transformation is accompanied by various pre- and post-processing operations like smoothing, median filtering and normalization.

Maximum Value of PSD of Normalized-Centered Instantaneous Amplitude has been used several times as one of the features for AMC [28], [29]. The PSD can be expressed as [3]

$$\gamma_{\max} = \frac{\max |DFT(a_{cn}(n))^2|}{N_s}, \quad (2.14)$$

where γ_{\max} reflects the variations in amplitude that enables the classification algorithm to distinguish between amplitude and non-amplitude modulations.

WT is also used as a technique for feature extraction. It offers the advantage of reducing the effects of noise [3], [29], [30]. The features designed using WT consist of time domain and frequency domain information [3]. Hence, these features have a greater resilience to noise as compared to the features based on FT.

2.1.2.4 STATISTICAL FEATURES

The commonly used statistical features for AMC are Higher Order Statistics (HOS), Cyclostationarity (CC), Constellation shape features and Zero-crossing (ZC). HOS offer significant advantages [3], such as resistance to noise, robustness to phase rotation and also represent higher order statistical characteristic of the signal.

The first kind of HOS are Higher Order Moments (HOMs). A particular HOM for a received signal $x(n)$ can be expressed as [3]

$$M_{p+q,p} = E[x(n)^p (x(n)^*)^q], \quad (2.15)$$

which is the expectation of the p^{th} power of the signal, multiplied by the q^{th} power of the conjugate of the signal. The combination of two or more HOMs gives HOC. For example, the fourth-order cumulant is expressed as [3]

$$C_{42} = \text{cum}[x(n)x(n)x(n)x(n)^*] = E[|x(n)|^4] - |E[x(n)^2]|^2 - 2E^2[|x(n)|^2]. \quad (2.16)$$

The received signal $x(n)$ is considered a zero mean random variable. Modulation classifiers using cumulants as features are based on the idea that the shape of the distribution of a random variable (which in equation 2.16 is the received signal), can be characterized by higher-order cumulants [31]. The shape of the distribution can then be used to distinguish modulation types from each other. Fourth-order cumulants are widely used as features in modulation classifiers.

2.1.2.5 ZERO-CROSSING

The number of ZCs of the received signal has been employed for AMC along with some post-processing techniques. The rate of ZC is fixed in case of PSK, but varies in Frequency-shift keying (FSK) signals. Hence, ZC can be used as one of the features for distinguishing PSK from FSK modulation type [32].

2.2 CONTRIBUTIONS IN THIS FIELD

This section gives a literature survey of the work accomplished in the area of AMC. An overview of modulation classification done using LB as well as FB approach is presented.

With all the parameters perfectly known, multiple antennas at the receiver and AWGN, ALRT based AMC was implemented by Abdi *et al.* in [33] for linear as well as nonlinear modulations. The classifier was complex to implement, hence a Quasi Hybrid Likelihood Ratio (QHLR) based classifier was also implemented. They distinguished between three modulation types, namely 16-QAM, 32-QAM and 64-QAM. They added a second antenna to significantly boost the performance of classification algorithm, as compared to single antenna ALRT method. At 10dB, the accuracy rises from 82% in single antenna ALRT, to 94% after adding a second antenna.

Low-complexity techniques were used to estimate unknown parameters. ALRT based classifier was derived with the assumption of per-symbol phase-incoherence in [34]. Beidas and Weber classified various Multiple frequency-shift keying (MFSK) signals corrupted by AWGN. The classifier performance for 32-FSK vs 64-FSK case (10 symbols) ranges from 65% to 99% for SNRs from 4dB to 12dB [34].

Literature shows that M-ary Frequency Shift Keying (MFSK) signals are often classified by utilizing Higher-order Correlation (HOC) operations. An optimal performance was achieved in [35], by utilizing measurements made in HOC domain. Combination of HOC domain and statistical decision theory gave rise to classifiers that delivered satisfactory performance, even under complete lack of knowledge of the incoming epoch.

An intrinsically wideband HOC method was proposed in [35], that is resistant to imperfect knowledge of exact frequency locations. The probability of correct classification with synchronous classifier ranges from 0.77 to 0.99 for SNR range 4dB to 10dB in 32-FSK vs 64-FSK case. Using the same classifier, the probability ranges from 0.77 to 0.99. Using an asynchronous classifier offers the benefit of reducing the gap of sensitivity in performance as we increase the quantization level. A good performance can then be achieved even with low quantization level [35].

Panagiotou *et al.* used a combination of ALRT and GLRT in [15] to achieve greater performance and also overcome analytical difficulties when using ALRT alone. They distinguished between the modulation pairs V.29 and 16-PSK, 16-QAM and V.29, 16-QAM and 16-PSK. The performance loss for 16-QAM and V.29 was greater as compared to the other two pairs. Further, they developed GLRT and HLRT based algorithms that exhibited significantly better performance as compared to ARLT based algorithms.

Shim and Kang [36] modified GLRT to approximate optimal classifier in the Bayesian sense. Their Sphere Decoding (SD) algorithm enabled maximum-likelihood multiuser detection, along with modulation classification to distinguish between 16-QAM and QPSK. The gain was significantly better than conventional receiver algorithms.

A new powerful and computationally non-expensive algorithm for AMC introduced by Tadaion *et al.* [37] combined ALRT and GLRT, resulting in suboptimal implementations with almost no performance loss. They concluded that GLRT based algorithms could not identify nested constellations. HLRT was found to be computationally expensive when the number of symbols is large. They developed an algorithm that classified modulation signals without requiring any knowledge of the signal's complex amplitude or noise variance. The accuracy for SNR ranging from -10dB to 10dB varies from 0.5% to 100% [37].

Leinonen and Juntti [38] investigated AMC algorithms in Orthogonal Frequency Division Multiplexing (OFDM) systems with adaptive modulation, that provided acceptable performance with low number of observed symbols. Quasi-Log Likelihood Ratio (QLLR) algorithm was found to deliver good performance, provided SNR was high and at least 40 symbols were allowed to be buffered.

Chugg *et al.* [39] utilized HLRT based AMC to perform autonomous power estimation of PSK signal corrupted by AWGN. They identified modulation types BPSK, QPSK and OQPSK, while including autonomous power estimation as well as threshold setting. They achieved 100% classification accuracy at just 4dB SNR for 500 symbols of the three modulation types. They concluded that the hardest aspect of modulation classifier design was acquiring reliable estimation of power values [39], with the availability of just the in-band measurements.

Hong and Ho [18] proposed an HLRT based approach to distinguish BPSK signals from QPSK signals, without a priori knowledge of the received signal. Its classification accuracy outperformed that of a classifier which assumes a known and constant signal level. The probability of correct classification for SNR values -9dB and -1dB were 77% and 74% respectively. While the accuracy of the classifier that assumes known and

constant signal level is no more than 92% [18]. The best result of 94% was obtained by the classifier that uses true value of the signal.

Derakhtian *et al.* [40] proposed an HLRT solution to classify ASK, PSK and QAM signals, followed by quasi-HLRT classifier to reduce the computational cost in unknown slow flat fading channel. For symbol length of 20, their algorithm was able to classify modulation types with accuracy values ranging from 30% to 100% for SNRs in range -10dB to 10dB [40].

The HLRT approach was used in [41] to develop a QHLR based classifier that can be applied to any fading distribution, including Rayleigh and Rice. QHLR based multi-antenna classifier showed performance improvement over ALRT and GLRT based classifiers in various channel and fading conditions. They demonstrated that the classifier performance improves with increase in number of antennas. They distinguished between 16-QAM, 32-QAM and 64-QAM (500 symbols) with accuracies ranging from 65% to 90% for SNRs 1dB to 5dB [41].

Amplitude and phase information contents were utilized by Azzouz and Nandi in [42] to classify twelve analogue and nine digital modulation types, without a priori knowledge of whether the signal is analogue or digital. For digitally modulated signals, they achieved 83% and 100% classification accuracies at SNRs 10dB and 20dB respectively. The accuracy was 90% at 10dB and 100% at 20dB [42].

The same authors proposed a new classification technique in [1], comprising of three different structures of ANNs. Such features are based on instantaneous amplitude, phase and frequency of the intercepted signal. The overall classification accuracy of decision theoretic algorithm was 94% for classes 2-FSK, 4-FSK, 2-ASK and 4-ASK at SNRs 15dB and 20dB. The ANN algorithm improved the results to 96% [1].

A novel hierarchical structure for AMC classifier was proposed in [43], where Farrel and Memmone implemented a Neural Tree Network (NTN) that utilizes features obtained from autoregressive model of the signal. These features are the instantaneous frequency, bandwidth and derivative of the instantaneous frequency. For Carrier to Noise Ratio (CNR) ranging from 5dB to 20dB, the accuracy of decision tree was found to be 86%. The new NTN algorithm beats the decision tree with 97% accuracy. The NTN classifier overcomes an issue in classifiers such as decision tree- lower accuracy for problems that need diagonal discriminant. This is because decision tree requires discriminant boundaries to be perpendicular to features axes [43].

Wong and Nandi [44] proposed a statistical based feature set (second, third and fourth order cumulants) along with adaptive varying weights and early stopping, that is

able to distinguish between eight digital and two analogue modulation types. The overall accuracies at SNRs from -5dB to 20dB range from 89% to 100% [44].

WT offers the utility of localizing modifications in instantaneous amplitude, phase and frequency of the intercepted signal. Ho *et al.* [45] apply WT on the intercepted signal to extract distinctive patterns without the need of any a priori knowledge. The transient information given by the WT was then utilized for simple processing in order to distinguish between ten digital modulation types. The overall accuracy was around 96% at CNR 13dB [45].

Liu and Lao [46] designed a feature set using HOS moments of fractional Fourier Transform (FT) along with WT. The special properties of fractional WT and fractional FT extend the range of modulation recognition, and also give a high classification accuracy at low SNRs through Rayleigh fading channels. The modulation types 2-ASK, 2-FSK, 16-QAM and BPSK were classified with accuracy of around 85% at -5dB SNR [46].

Swami and Sadler [21] proposed fourth-order cumulants that prove to be efficient when being used in a hierarchical scheme. This technique offers the advantage of classifying modulation types into lower subclasses with small sample size and at low SNR. In addition, it can be implemented recursively and is robust in the presence of carrier frequency and phase offsets. They demonstrate that HOS based statistical features give better results with lower complexity, when compare to LB methods. The performance in AWGN channel was found to be a bit better [21].

Dai *et al.* [22] eliminate the need for knowledge of signal and noise power to accomplish AMC, by introducing a novel technique of Joint Power Estimation and Modulation Classification (JPEMC). The technique comprises of using the relationship between second- and higher-moments of the intercepted signal, along with signal and noise power. They distinguish the modulation types 16-QAM, 4-ASK, 8-PSK and BPSK with accuracies ranging from 25% to 100% at SNR values ranging from 0dB to 20dB [22].

Hatzichristos and Fargues employed a hierarchical approach [23] consisting of a tree based model where each node is a neural network decision unit. They use a combination of power-normalized moments and cumulants that deliver good performance for signals corrupted by additive noise. QAM, FSK and PSK signals were identified in multipath environments. The classification is done in three kinds of environments: rural, small town and urban type. The best performance achieved was 97% at 20dB SNR in rural. The results at the same SNR were 53% and 65% in small town and urban type respectively [23].

Signal constellation shape employed as a feature for AMC in [47] has been found to be a stable modulation signature. This feature exhibits significant stability in weakly

synchronous and high noise environments. It offers the benefit of resistance to receiver imperfections and channel effects. The authors distinguish between modulation types V.29_fallback, V.29, 16-QAM and 8-PSK. The probability for correct classification in V.29_fallback vs. 8-PSK case ranges from 0.5 to 1.0 for SNRs -5dB to 20dB. While the probability for the V.29 vs. 16-QAM case ranges from 0.35 to 0.99 in the same SNR range [47].

Hsue and Soliman selected ZC as the feature for AMC in [48]. This signal conditioner provided the merit of accuracy in phase transition information over a wide dynamic range of frequency. Simulations for classification of modulation types BPSK, QPSK, 8-PSK, BFSK, 4-FSK, 8-FSK and Continuous Wave (CW) are carried out at 15dB CNR. An overall accuracy of approximately 98% is achieved [48].

2.3 SUMMARY

This chapter has studied signal classification in detail. It starts by mentioning the types of signal classification and stating the objective of this research. The two major approaches to signal classification- LB and FB were described in detail. The subtypes of LB and FB were considered as well. The last section summarized the significant contributions and results achieved in the area of AMC, for both LB and FB approaches.

CHAPTER 3

BACKGROUND ON MACHINE LEARNING

Machine learning techniques can accomplish the task of AMC by using a data driven methodology. The classification of digitally modulated signals is performed by utilizing supervised learning, which is a type of machine learning. Machine learning is the science of programming computers, so they are able to learn from data [49]. This practice aims at giving computers the ability to learn without being explicitly programmed. The objective is to design self-learning algorithms that can make sense of data.

Machine learning can be divided into three types, as Figure 3.1 illustrates.

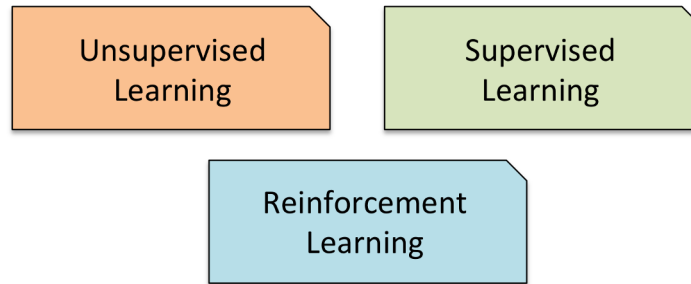


Figure 3.1. Types of machine learning. Source: [50].

Out of the three kinds, we have used the supervised learning approach to solve the problem of AMC. Figure 3.2 depicts a supervised learning model. The objective of the model is to learn a function from the labeled data given to it. Every instance presented to the model during the training phase is a pair consisting of a vector and its corresponding label or class [50]. The function learned by the model can then be used to make predictions on unlabeled data. In our case, the model accepts a vector as input and predicts the modulation type of this vector.

The data used in the thesis is a large set of N samples $\{x_1, x_2, \dots, x_N\}$. All of these samples are labeled- meaning that we know the modulation type to which each of these N samples belong. The data is then separated into training set and test set. The samples that our machine learning model uses to learn, comprise the training set. Whereas, the samples (unseen by the model) that the model will be tested on, comprise the test set. We input the training set to the machine learning algorithm, that learns from this data and

forms a predictive model. We can then use this model to predict the category of new samples.

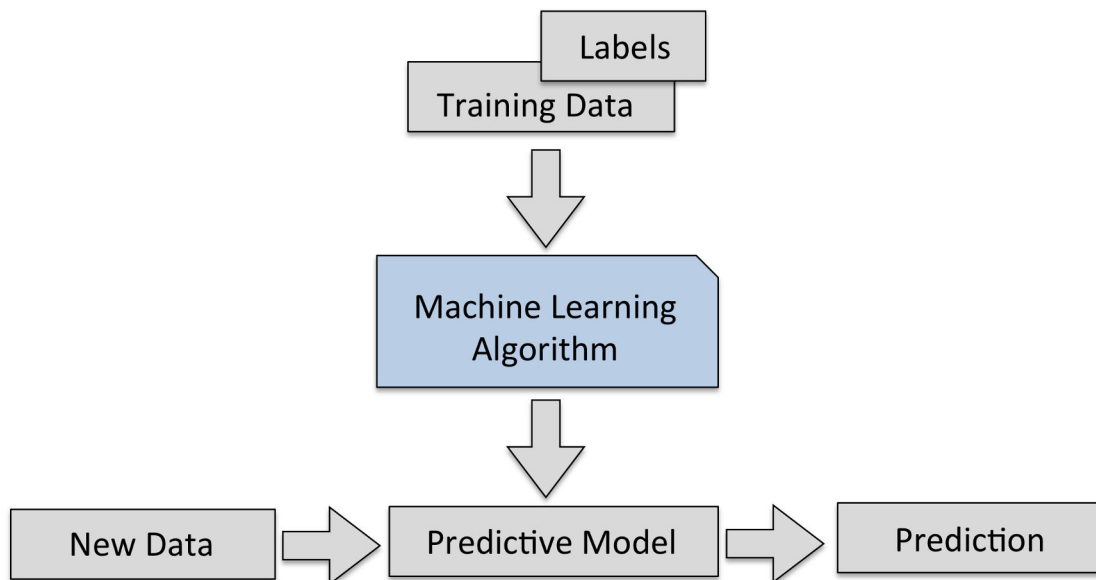


Figure 3.2. Machine learning model: The predictive model trains on labeled data; then it can be used to make predictions. Source: [49].

The following reasons elaborate why machine learning is an excellent choice for the problem at hand:

1. As explained in the previous sections, the first step to AMC is feature extraction. If features are to be used for classifying a sample into one of the given modulation types, then we need an algorithm that is capable of spotting patterns in the data. This ability of the algorithm is required since AMC is a pattern recognition problem. Our goal is to train a model to learn patterns in the given data, and then use it to classify an input vector into one of the modulation classes. This is the task of classification- one of the subtypes of supervised machine learning.
2. Machine learning is known for its ability to handle large amounts of data. The Radio Machine Learning dataset [51] used in this research contains 160,000 samples in total. Each sample is a (2, 128) vector that belongs to one of the eight digital modulation types. Machine learning shines where such great amount of data is present. It can dig into massive amounts of data to discover patterns and therefore solve complex problems such as AMC. The various machine learning algorithms discussed in this thesis can effectively analyze the several thousand samples in the dataset used, and gain valuable insights about the patterns occurring in the data. The classifiers detect the different patterns occurring in the samples belonging to different modulation types. Using such instance based learning, they

build a predictive model that can identify the modulation type of an unknown signal presented to it.

3. Machine learning algorithms can adapt to new data as shown in Figure 3.3. This can be more clearly understood by considering traditional versus machine learning approach. Suppose, our classification algorithm needs to have knowledge of certain characteristics of BPSK modulated signals, so that it can identify BPSK signals, from among the samples belonging to other modulation types. Consider that we give three rules to our classifier in order for it to identify BPSK samples. What if the classifier encounters a sample that follows none of these three rules- but it belongs to the BPSK modulation type? There would arise a need to add a fourth rule to the traditional programming technique. We might even need to add more rules in the future. In contrast, a machine learning classifier does not require such human intervention. It automatically notices the new pattern occurring in the sample(s) labeled BPSK. Machine learning algorithms are therefore capable of adapting to change.

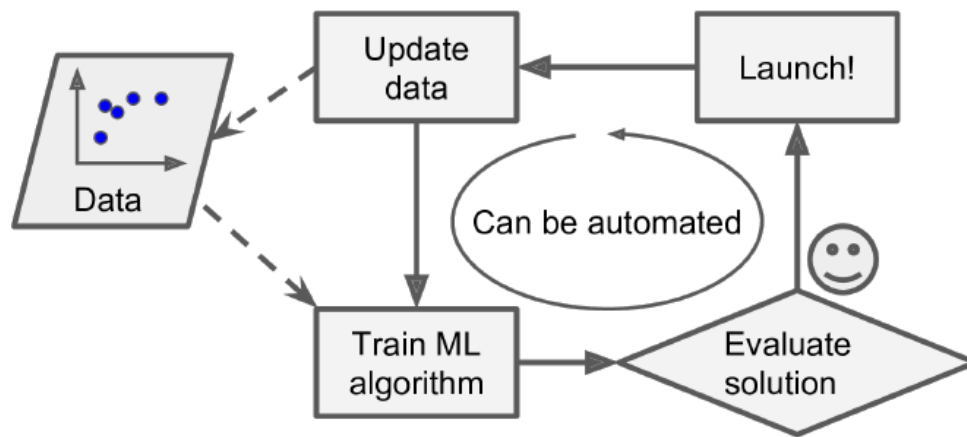


Figure 3.3. Automation in machine learning: The process of training a machine learning algorithm can be automated. Source: [49].

The problem of AMC in this research is a supervised learning problem, since the training data fed to our learning algorithm includes the correct solutions or labels. We have implemented the following supervised learning techniques:

1. k-Nearest Neighbors
2. Support Vector Machine
3. Decision Tree
4. Random Forest and boosted ensemble of trees
5. Deep Neural Network
6. Convolutional Neural Network

3.1 NEAREST NEIGHBORS CLASSIFICATION

Nearest Neighbors Classification is instance-based or non-generalizing learning [52]. This means that its objective is not to construct a general internal model; it rather stores samples of the training data. For every query point/sample x_1 , a certain number of surrounding points is considered. The classifier then observes the labels of these ‘neighbors’ and collects majority vote- the label that occurs the most. This is the label that the classifier predicts for query point x_1 .

3.1.1 K-Nearest Neighbors

In k-Nearest Neighbors (KNN), learning is based on the k closest neighbors of every query point, where k is an integer. The optimal value of k highly depends on the data. A very low value, such as $k = 1$ implies that the KNN algorithm considered only one data point which is right next to the query data point for prediction. If this prediction is wrong, the algorithm is said to have captured noise in the data. A small value of k results in a highly complex model, leading to overfitting. This causes the model to poorly generalize to the data, thus resulting in low accuracy on the test data.

Now consider the opposite scenario - suppose we choose a very large value $k = 100$. The model becomes over-generalized. The effect of noise is suppressed in this case, but it ends up making the classification boundary less distinct. The right choice of k is therefore in between these two extremes, such as $k = 16$.

To recap, KNN iterates through the following steps [50]:

1. Select the value of k and distance metric- euclidean distance is most commonly used.
2. Find the k nearest neighbors of the query data point- the sample we need to classify.
3. Take the majority vote to decide the class.

Figure 3.4 depicts a 3-class classification problem. Since we have set $k = 5$ for this problem, the KNN algorithm considers the five closest neighbors of the query point ‘?’. It then takes the majority vote to decide the class of this query point. The algorithm observes that most of the neighbors belong to class 3. It therefore assigns the label class 3 to this query point.

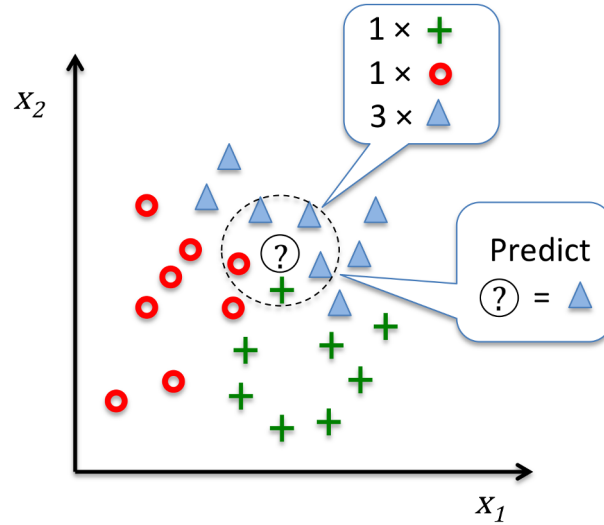


Figure 3.4. Example of KNN with $k = 5$: The classifier takes majority vote from the data points surrounding the query point, to decide the class of the query point. Source: [50].

3.1.2 Example of K-Nearest Neighbors Implementation

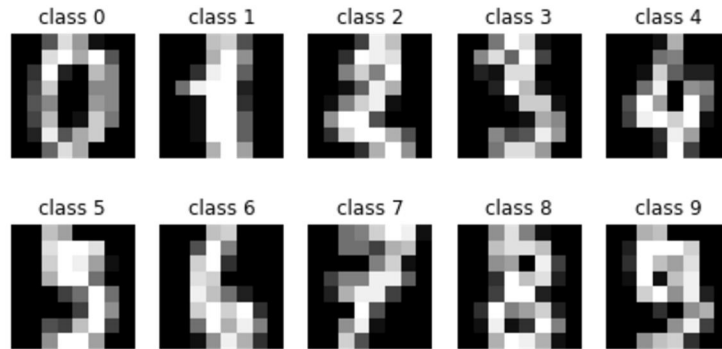


Figure 3.5. Digits dataset consisting of ten classes. Source: [52].

This section illustrates the implementation of KNN classifier on the digits dataset. This dataset consists of 1797 samples and 64 features. Each sample is an 8×8 image of a handwritten digit. There are ten classes- for digits 0 through 9, as depicted in Figure 3.5.

The task of the classifier is to recognize a digit, given its image. The 1797 samples are divided into training and test sets in the ratio of 50:50. Prior to feeding the training

data to the classifier, it is required to standardize the features. Feature scaling is required for optimal performance. This involves using the training data to compute the mean and standard deviation for every feature dimension. These estimated parameters are then used as the scaling parameters to standardize the test set. The model is trained and tested on the standardized training and test data respectively. The KNN classifier trained on this dataset gives classification accuracy of 97%.

3.2 SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) is a powerful learning algorithm capable of learning complex nonlinear functions [50]. It is similar to logistic regression in that, it is driven by a linear function [50]

$$w_0 + \mathbf{w}^T \mathbf{x}, \quad (3.1)$$

where w_0 is a constant called bias, \mathbf{w} is the set of SVM weights and \mathbf{x} is the set of feature vectors. But unlike logistic regression, SVM outputs a class identity, rather than class probabilities. This machine learning model is capable of performing linear classification, nonlinear classification, regression and outlier detection.

3.2.1 Margin Maximization

The optimization objective of SVM is to maximize the margin. Margin is the distance between the decision boundary or hyperplane, and the training samples that are nearest to this hyperplane. Such samples are the support vectors [50]. Figure 3.6 illustrates the concepts of hyperplane and margin.

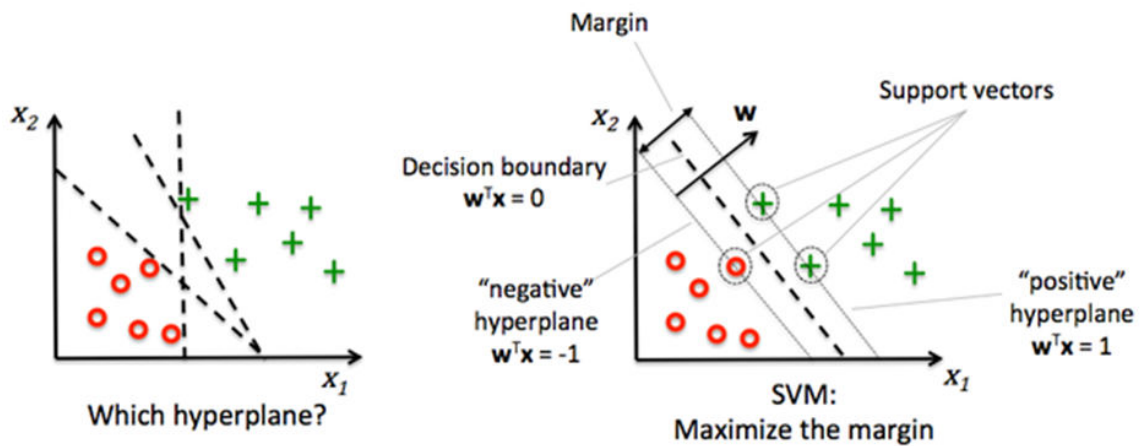


Figure 3.6. SVM hyperplane and margin: SVM selects the optimal hyperplane by margin maximization. Source: [50].

Small margins are prone to overfitting, whereas large margins result in a lower generalization error. Hence, it is desired to have decision boundaries with large margins. Margin maximization can be clearly understood by deriving the equation for margin [50]. The equations for the hyperplanes seen in Figure 3.6 can be expressed as follows

$$\text{Equation for positive hyperplane: } w_0 + \mathbf{w}^T \mathbf{x}_{pos} = 1. \quad (3.2)$$

$$\text{Equation for negative hyperplane: } w_0 + \mathbf{w}^T \mathbf{x}_{neg} = -1. \quad (3.3)$$

$$\text{Subtracting the above equations: } \mathbf{w}^T (\mathbf{x}_{pos} - \mathbf{x}_{neg}) = 2. \quad (3.4)$$

Normalizing equation 3.4 by length of a vector \mathbf{w} [50]

$$\frac{\mathbf{w}^T (\mathbf{x}_{pos} - \mathbf{x}_{neg})}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}. \quad (3.5)$$

Where the vector \mathbf{w} is defined as:

$$\|\mathbf{w}\| = \sqrt{\sum_{j=1}^m w_j^2}. \quad (3.6)$$

The left hand side of equation 3.4 is the margin, or the distance between positive and negative hyperplanes as shown in Figure 3.6. The objective of SVM is margin maximization, which is equal to maximizing $\frac{2}{\|\mathbf{w}\|}$. This is to be achieved under the constraint that the samples are correctly classified. The correct classification of samples can be expressed as:

$$w_0 + \mathbf{w}^T \mathbf{x}^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1, \quad (3.7)$$

which implies that all the positive samples must fall behind the positive hyperplane, and

$$w_0 + \mathbf{w}^T \mathbf{x}^{(i)} < -1 \quad \text{if } y^{(i)} = -1, \quad (3.8)$$

implies that all the negative samples must fall on one side of the negative hyperplane. Equations 3.7 and 3.8 can be combined as [50]

$$y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) \geq 1 \quad \forall_i. \quad (3.9)$$

The SVM algorithm must satisfy equation 3.9, so that all the positive samples fall on the right of the positive hyperplane and all negative samples fall on the left of the negative hyperplane, as shown in Figure 3.6. This condition ensures margin maximization, so that the SVM correctly classifies a new instance into one of the classes shown.

3.2.2 Kernels

Figure 3.6 illustrated a case where the classes were conveniently separated using a linear decision boundary. The problem was solved using a linear kernel, but SVMs have gained popularity due to their ability to learn nonlinear decision boundaries. Kernel is the trick behind SVMs and makes them powerful complex nonlinear classifiers. For example, a linear decision boundary would be a poor fit to separate the two classes shown in Figure 3.7.

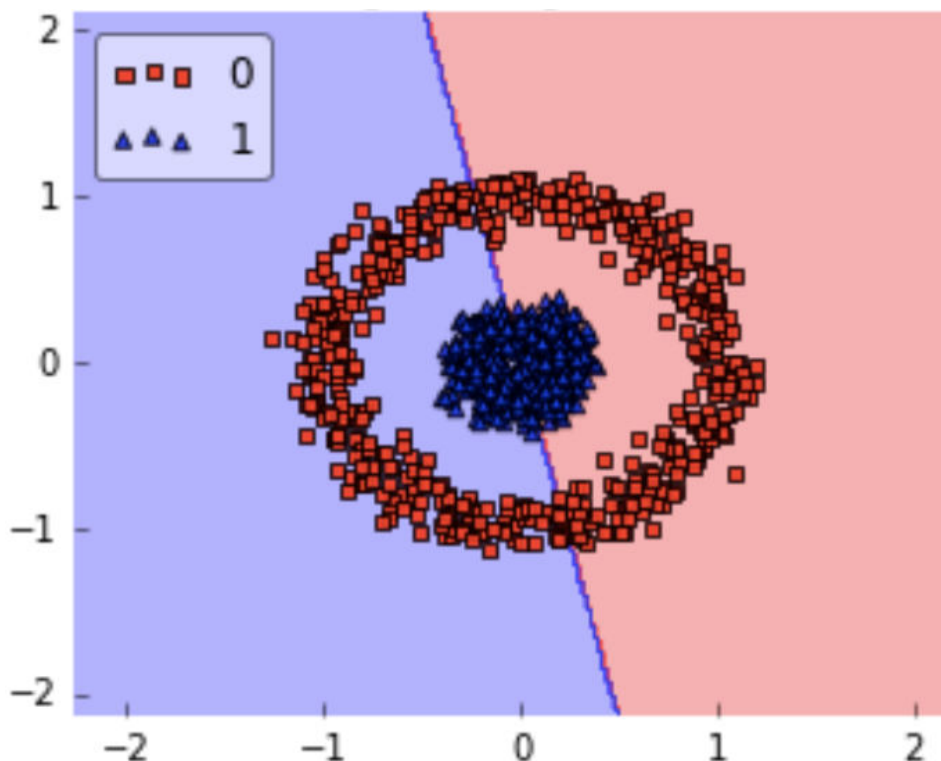


Figure 3.7. Linear kernel is unfit for complex data: Linear decision boundary fails to separate the two classes. A nonlinear function is required for this classification problem. Source: [50].

Kernel methods deal with such data that cannot be separated via a linear decision boundary. Kernels create nonlinear combinations of the original features, and project them onto a higher dimensional mapping space via a mapping function $\phi(x)$.

Kernel SVM performs this transformation by creating non-linear combinations of features, thus uplifting the samples onto a higher-dimensional feature space. In the case shown in Figure 3.8, $\phi(x)$ transforms a two-dimensional space into a three-dimensional one. It does so by creating three non-linear combinations of the two features x_1 and x_2

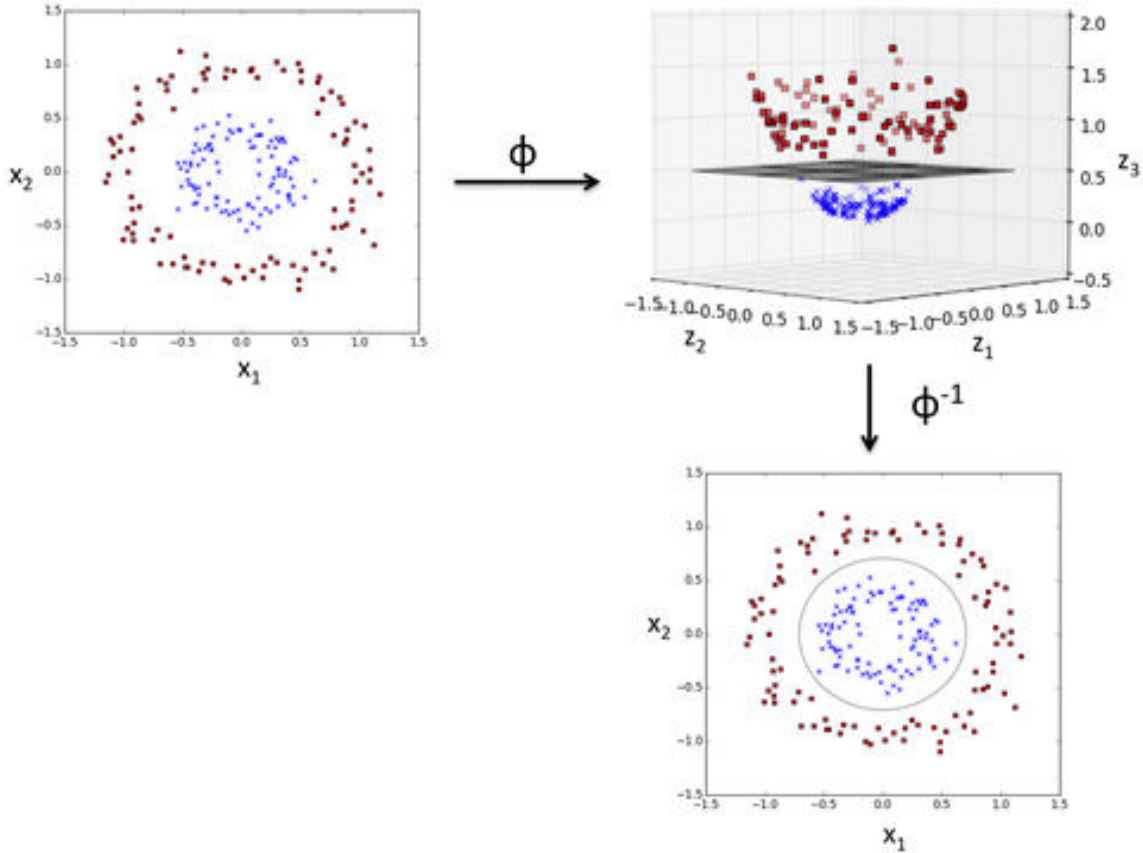


Figure 3.8. Kernel method: Feature function transforms the feature space into a higher dimension where the classes can be separated by a linear hyperplane. The feature function then transforms the feature space back to the original dimensions. Source: [50].

present in the original feature space. This process can be expressed as [50]

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2). \quad (3.10)$$

A linear decision boundary can then be used to separate the classes in the new higher dimensional feature space. Such transformation therefore allows the SVM algorithm to separate the two classes via a linear hyperplane. Then we project the hyperplane back onto the original two-dimensional feature space- the plane thus gets transformed into a nonlinear decision boundary.

Given a dataset, we might not know right away which kernel works best for it. The solution is to start with a simple hypothesis space and then advance towards more complex hypothesis space. Kernel based functions are used by SVM to make predictions in cases where the data is not linearly separable. Applying the mapping function $\phi(\cdot)$ to a classification problem means preprocessing the data by applying $\phi(\cdot)$ to all the inputs. A

linear model learned in the new space gets transformed into a complex model in the original feature space.

The Gaussian Radial Basis Function (RBF) kernel is one of the most widely used kernels. RBF kernel separates the nonlinear data by adding features. These features are computed using a similarity function that measures the similarity between a pair of samples [50]:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2). \quad (3.11)$$

The last plot in Figure 3.8 shows that the nonlinear RBF kernel is able to separate the two classes, unlike the linear kernel in Figure 3.7.

In RBF SVM, training entails optimization of two hyperparameters γ and C . γ defines the influence or reach of the training samples. Increasing its value makes each sample's range of influence smaller, resulting in a tighter decision boundary around the individual samples. A smaller value of γ results in a softer or smoother decision boundary, meaning that the samples have a larger range of influence. This is illustrated in the three plots on the left of Figure 3.9. The model here is quite constrained and not able to capture the complexity of the data. As the value of γ increases (left to right), the radius of the area of influence becomes increasingly constrained and tightly bound around the support vectors.

The other parameter is C , that controls the penalty for misclassified training samples. Large value of C implies that the SVM algorithm tries hard to classify all the samples correctly. Whereas a smaller C means that the classifier could misclassify a few samples. As the three plots on the first row of Figure 3.9 depict, a small value of C makes the decision surface smooth. But as we increase its value (top to bottom row), the SVM algorithm tries harder to classify all the samples correctly. The optimum values of both these parameters are computed by using grid search and randomized search.

3.2.3 Example of Support Vector Machine Implementation

We introduced the digits dataset in section 1.1. Here we train SVM on the same dataset. As was done in case of KNN, mean and standard deviation are computed from the training data. SVM is trained on this standardized data and tested on the standardized test data. We set the hyperparameters of SVM and use RBF as the kernel. SVM classifier gives a classification accuracy of 97% on the digits dataset.

3.3 DECISION TREES AND RANDOM FORESTS

Decision Tree (DT) is a learning algorithm that breaks up the input sequence into regions. Each of these regions have separate parameters. DTs form a fundamental

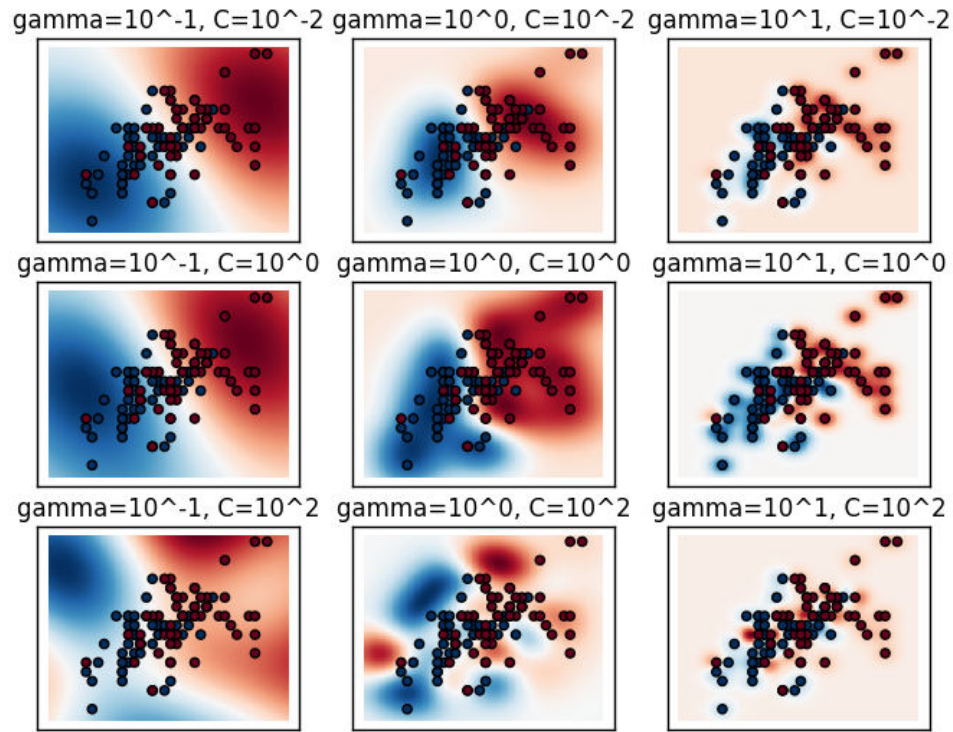


Figure 3.9. Effect of varying SVM hyperparameters: plots showing the effect of training SVM with three different values of C and three different values of γ . Source: [52].

component of Random Forests, one of the most powerful machine learning algorithms [50]. DT performs classification based on a set of questions. It takes the features in the training set and learns a series of questions, which helps it to infer the classes present in the data.

3.3.1 Decision Tree Algorithm

DT learning algorithm can be clearly understood by Figure 3.10. It shows a tree composed of multiple nodes, where each node contains all of, or most of data belonging to one category [53]. The algorithm has been designed in a way such that, when an instance is presented to the decision tree, it easily falls into one of the nodes. This solves the classification problem by indicating what category this instance belongs to.

Each of the nodes is associated with a region in the input space. There are internal nodes present, that break this region into a subregion for every child of the node [53]. This means that the input space gets subdivided into non-overlapping regions. Leaf node

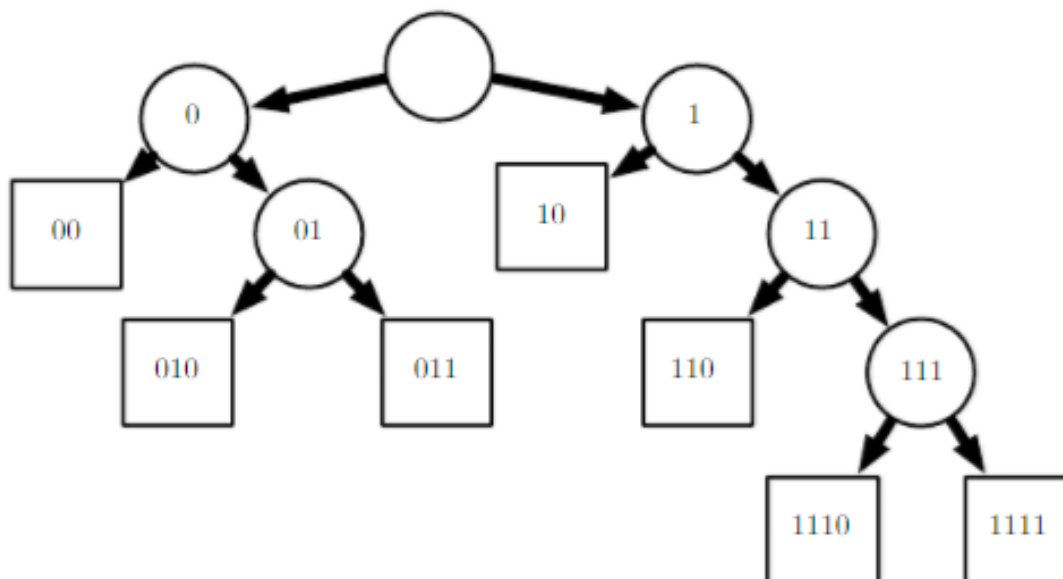


Figure 3.10. Decision tree: An instance presented at the root node traverses the tree until it reaches one of the leaf nodes- where it finally gets classified. Source: [53].

is the terminal node in the DT- this is the node that predicts the outcome. All circular nodes in Figure 3.10 are internal nodes. The nodes represented by square - such as '00' and '010' are terminal nodes. This tree therefore contains data belonging to seven classes- as indicated by the seven terminal nodes.

A greedy algorithm is used for training DTs. This algorithm uses a single feature k and a threshold t_k to split the dataset into two subsets. The values of k and t_k are chosen by finding the optimal pair (k, t_k) - the pair that creates the purest subset. Here, pure means a subset containing data from only one class. The dataset is split into two subsets and these subsets are split further using the same concept [53]. The splitting of space into subregions results in a one-to-one correspondence between leaf nodes and input regions. Hence, the leaf node performs the task of mapping each point in its input region to an output region, thus classifying each incoming instance. This means that the process of splitting stops when the tree reaches maximum depth. At this point, the tree is unable to find a split which will further decrease the impurity.

The root node in Figure 3.10 is the topmost node of the tree and contains the whole dataset for that tree. This is the node at which the instances enter the tree and point where the algorithm begins. As stated earlier, the tree performs classification by asking questions. The job of every node in the tree is to receive an instance and ask a question. Based on the answer, the node sends this instance to either its left child node or

right child node [53]. So the root node observes the features of every instance presented, and asks if according to these features, should it send the instance to node '0' or node '1'?

Suppose the root node decides to send the instance to the left node '0'. A similar procedure is followed and node '0' will decide whether to send the instance presented to it, to either node '00' or node '01'. If it observes the instance's attributes and discovers that they match with the dataset contained in the left node '00', then node '0' will send this particular instance to node '00'. Node '00' is the leaf node - it does not have any child nodes. So node '00' does not ask any questions. The DT simply predicts the category of this instance as '00', thus completing the classification task for this instance [53].

The greedy algorithm starts by looking for an optimum split at the top level- the root node. It then repeats the process of splitting at each level. But the disadvantage of greedy algorithm is that it does not ensure that the split which it is performing, will result in the minimum possible impurity level [53]. It often gives a good solution, but not an optimal one. This problem is solved by a more powerful machine learning algorithm called Random Forest.

3.3.2 Random Forests

A Random Forest (RF) is an ensemble of DT classifiers. Such a technique of combining multiple classifiers is called ensemble learning. Each DT is trained on a different random subset of the training set [49]. RFs will then make predictions by using the predictions made by each tree. It selects the class (of the instance presented to RF algorithm) that gets the maximum number of votes. Training RFs involves ensemble methods such as bagging and boosting.

The importance of RFs can be understood from the fact that training a DT would become increasingly complex as the number of instances and features in the dataset increase. Doing binary splits at each node of the tree would create a very large tree. If the small tree shown in Figure 3.10 took a few minutes for building and training, then a larger tree could take hours. A single tree is thus ill suited to large datasets - the tree becomes quite deep and significantly more time is needed by the tree to classify an instance. The solution therefore, is to combine several decision trees and grow a forest.

A RF consists of N DTs and the dataset is divided into N number of smaller datasets. The trees are built in parallel in order to minimize the runtime. Every tree is built on a different subset of the large dataset.

The effectiveness of RFs can be attributed to randomness. The randomness is essential to make sure that each of the trees in the forest is trained on distinct subsets of the training set- there is no overlap in the data. Firstly, there is randomness in the split of the training set. Secondly, there is randomness due to bootstrapping- randomly selecting

data points with replacement. This implies that some points could be selected more than once, and others may not be chosen at all. Hence, randomness is present in the data and in the feature splits as well.

RF algorithm containing N trees iterates through the following steps [50]. For tree $n = 1$ to N :

1. Pick a bootstrap sample from the training data.
2. Grow a tree T_b to the bootstrapped data by iterating through the following steps for every leaf node of the tree, till the point when the minimum node of size n_{min} is reached.
 - a) From the p features in training set, select a subset of size m .
 - b) Select the best split point or variable among the m features.
 - c) Split the node into two child nodes.
3. After all the trees have been built, output the ensemble of trees: $\{T_b\}_1^N$.

The randomness in the RFs makes sure that the trees are unique [49]. A decision tree that is very deep tends to overfit the data, thus resulting in high variance. This issue too is solved by RFs. Each tree in the forest has high variance because it is based on a subset of the data. The overall variance is then reduced by averaging the trees. Following sections discuss the most commonly used ensemble methods: bagging, pasting, boosting and stacking [49].

3.3.3 Bagging and Pasting

As stated earlier, the trees in the forest are trained on different random subsets of the training set. The technique is called bagging when the sampling is done with replacement, and pasting when it is done without replacement. This means that the instances from the training set are sampled several times for the same tree, in case of bagging. But in both bagging and pasting, the training instances are sampled multiple times across the trees. Pasting and bagging are illustrated in Figure 3.11.

After all the trees have been trained, the RF can classify a new incoming instance by aggregating the predictions made by all the trees. The popularity of bagging and pasting is due to the fact that both these methods scale quite well. This can be seen from Figure 3.11, where several models are being trained in parallel.

3.3.4 Boosting

Boosting is an ensemble method that combines several weak models to build a powerful ensemble. Boosting aims at training the models one after the other, where every model attempts to correct the preceding model. AdaBoost and Gradient Boosting are the commonly used boosting methods.

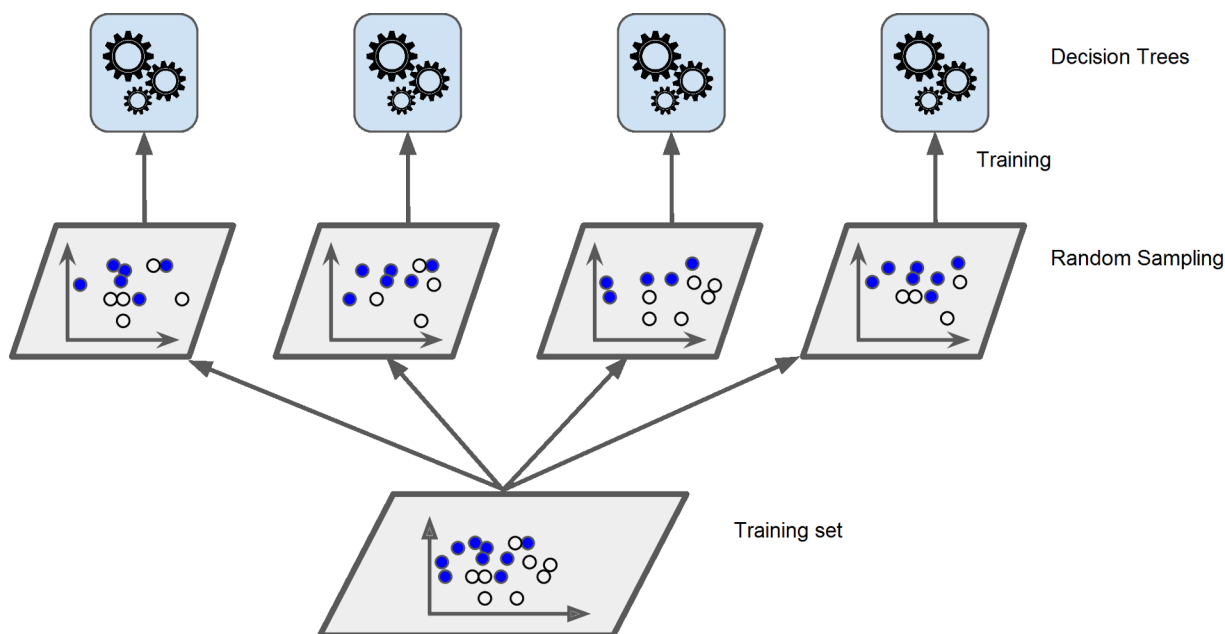


Figure 3.11. Ensemble of decision trees: Bagging involves random sampling with replacement, while pasting is done without replacement. Source: [49].

Boosting involves an ensemble of simple base classifiers that are weak learners, for example a decision tree stump- decision tree having only the root node connected to the leaves. The power of boosting lies in the fact that this technique focuses on the samples that are hard to classify. The ensemble's classification accuracy is improved by making the weak learners learn from the misclassified training samples.

Adaboost or adaptive boosting trains an ensemble of machine learning models that pay special attention to hard cases. A model m tries to correct the previous model $m - 1$ by focusing more on the samples that model $m - 1$ underfitted.

An instance first arrives at the left most classifier shown in Figure 3.12. The first classifier could underfit the samples in the training set when the training starts. The second classifier will pay more attention to the samples that its predecessor had underfitted. As shown in Figure 3.12, the weights of the first classifier get adjusted after it is trained. AdaBoost will boost or increase the relative value of weights only for those training samples that the first classifier misclassified.

The second classifier then uses these updated weight values to make predictions on the training set. It tries to classify all the samples correctly, but may end up misclassifying a few or several samples. The predictions made by it are passed on to the third classifier, that tries to spot the samples that the second classifier misclassified. Just like the algorithm adjusted the weights for the first classifier, it does the same for the second classifier as well- giving greater weight to the misclassified samples. This process

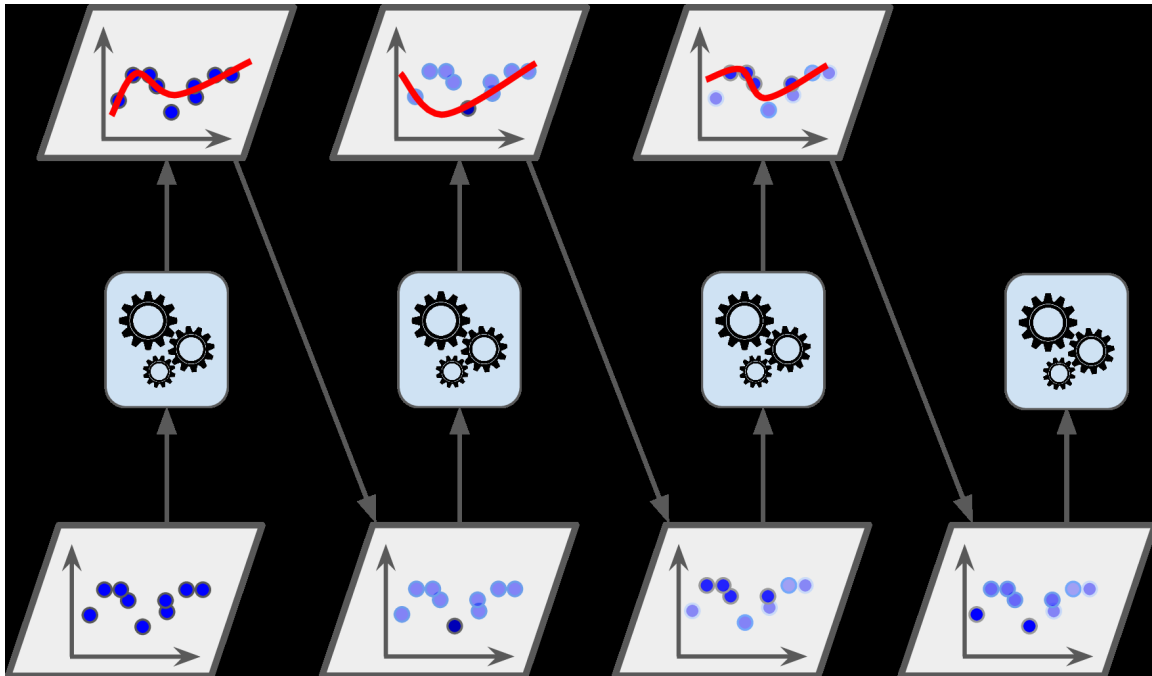


Figure 3.12. Weight update in Adaboost classifier: Each model in the ensemble is trained, after which its weights are boosted for the samples that it misclassified. Source: [49].

of prediction and weight updation continues throughout all classifiers in the ensemble. All the predictors get trained in this manner and the ensemble is able to make predictions similar to a bagging or pasting classifier. The only difference is that, the predictors now have weights that are different from each other.

Gradient Boosting technique works similar to AdaBoost, in that the predictors are sequentially trained. But the difference here is that Gradient Boosting aims at fitting the next predictor to residual errors that the previous predictor made. This kind of boosting is used various applications such as ecology and web search ranking [52]. Gradient Boosted ensemble offers significant advantages over AdaBoost. It is more robust to outliers, has more predictive power and is able to conveniently handle heterogeneous features [52].

3.3.5 Example of Decision Tree and Random Forest Implementation

DT classifier fitted to the digits dataset ends up getting classification accuracy of 82%. The performance is improved by RF- ensemble of 400 decision trees, that enhances the accuracy to 97%.

3.4 ARTIFICIAL NEURAL NETWORK

Artificial Neural Networks are systems that were inspired by the biological brain. Researchers and scientists turned towards the brain's architecture, so they could understand its working, in order to make intelligent systems or artificial intelligence work. They then came up with the idea of simplified brain cells - neurons, that are interconnected nerve cells in the brain [49]. This gave them the idea on how to build intelligent machines, leading to the creation of ANNs. ANNs have since been at the very core of Deep Learning and have been utilized to solve highly complex problems- from product recommendation in Amazon, to speech recognition in Google's Cloud Speech API that recognizes more than 80 languages [49].

3.4.1 Perceptron

The very first model based on biological neuron was Perceptron learning rule proposed by F. Rosenblatt [50]. This simple ANN architecture is based on Linear Threshold Unit (LTU)- wherein every input connection is associated with a weight. Perceptron was designed to automatically learn optimal weight coefficients. These optimized parameters are then multiplied with input features. The following steps provide an outline of Perceptron Learning Algorithm:

1. Weights are initialized to some random values or zeros.
2. Two steps are performed for each sample $\mathbf{x}^{(i)}$:
 1. Compute the output \hat{y} .
 2. Update the weights.

Mathematically, this involves computation of a weighted sum of the inputs, resulting in the net input $z = w_1x_1 + w_2x_2 + \dots + w_mx_m$. The LTU receives the input values, or vector $\mathbf{x} = [x_1x_2\dots x_m]$, that is combined or multiplied with weight vector $\mathbf{w} = [w_1w_2\dots w_m]$, resulting in the net input \mathbf{z} . Activation function (which in case of LTU is unit step function) is then applied to produce the final output $\phi(z)$ that corresponds to either of two classes- depending on whether $\phi(z)$ is greater than or less than a defined threshold θ . Perceptron therefore is used for simple binary classification [50]

$$\hat{y} = \phi(\mathbf{x}) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}.$$

The output value \hat{y} given by equation 3.4.1 is the class label predicted by the LTU. Post this prediction (for each incoming input or sample $\mathbf{x}^{(i)}$), the weight vector \mathbf{w} gets updated. This involves simultaneous update of each weight w_j [50]

$$w_j := w_j + \Delta w_j, \tag{3.12}$$

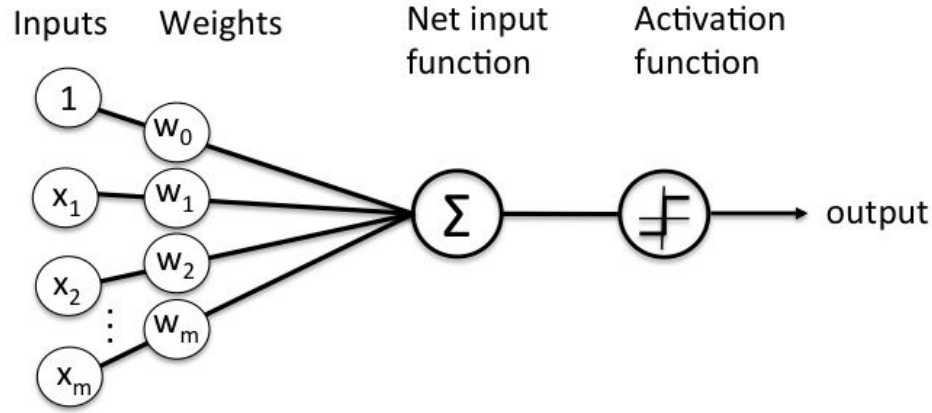


Figure 3.13. Linear Threshold Unit: Inputs are multiplied by weights to form the net input. Activation function is applied to the net input to generate an output. Source: [50].

where w_j is the current weight value and Δw_j is the amount by which the weight value gets updated. The weight updation occurs according to the following rule [50]

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}. \quad (3.13)$$

A Perceptron or LTU is capable of only simple binary classification. Also, the two classes must be linearly separable in order for the Perceptron to be able to discriminate between them, as illustrated in Figure 3.14.

3.4.2 Multi Layer Perceptron

Samples can be classified into more than two classes by using Multi Layer Perceptron (MLP)- a network consisting of one or more layers of LTUs.

The outline of the learning algorithm is similar to the Perceptron learning algorithm, as mentioned in the previous section. The rule for weight updation changes as follows [50]

$$w_{i,j}^{(nextstep)} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i, \quad (3.14)$$

where x_i is i^{th} input value of current training sample, y_j is target output from j^{th} output neuron for current training sample, \hat{y}_j is actual output from j^{th} output neuron for current training sample, η is learning rate and $w_{i,j}$ is weight value on the connection from i^{th} input neuron to j^{th} output neuron.

As described in the previous section, Perceptron is able to classify instances if they are linearly separable. This happens due to its decision boundary being linear. This problem is overcome by MLPs that are able to learn complex patterns. MLP is trained using backpropagation algorithm as explained in the next section.

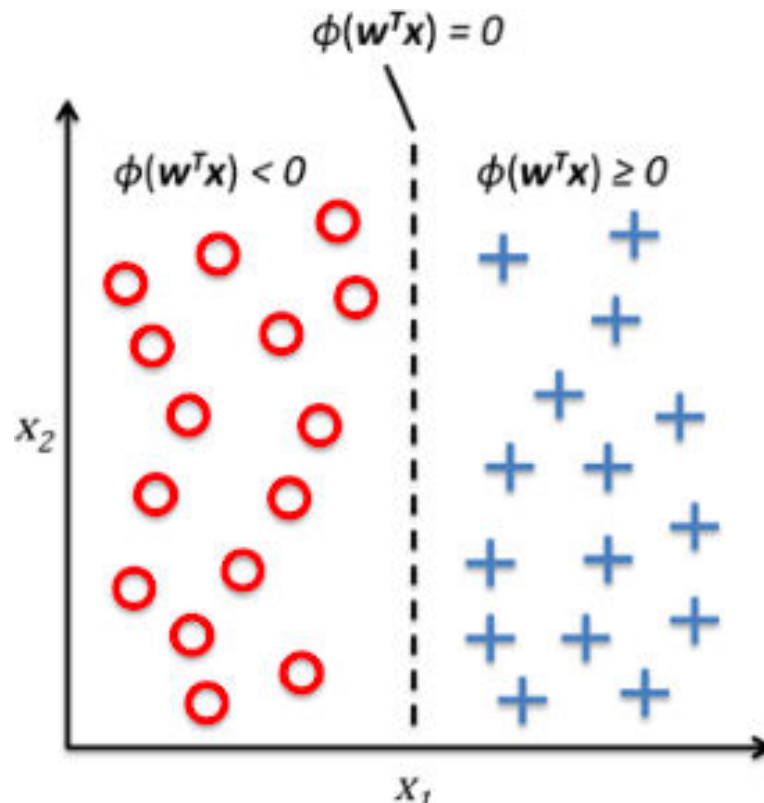


Figure 3.14. Perceptron decision boundary: A Perceptron can perform only simple binary classification. Source: [50].

3.4.3 Training a Neural Network- Backpropagation Algorithm

In MLPs or neural networks, also known as feedforward neural networks, the information flows from the input layer, passes through intermediate computations (layers) and reaches the output layer. It flows in the forward direction- feedback paths from output layer to other layers of the model are absent. The output of each neuron in every consecutive layer is computed [49]. This is called the forward pass. Then the network's output error is measured- difference between the desired output (targets) and the network's actual output. This process happens for each sample presented to the neural network [49].

The above process is followed by the backward pass. The algorithm calculates how much every neuron present in the last hidden layer contributed to the error generated by every neuron in the output layer. Then it computes the error contributions from each neuron in the previous layer [49]. The computations are carried on this way until the input layer is reached. In this way, the backward pass efficiently computes the error

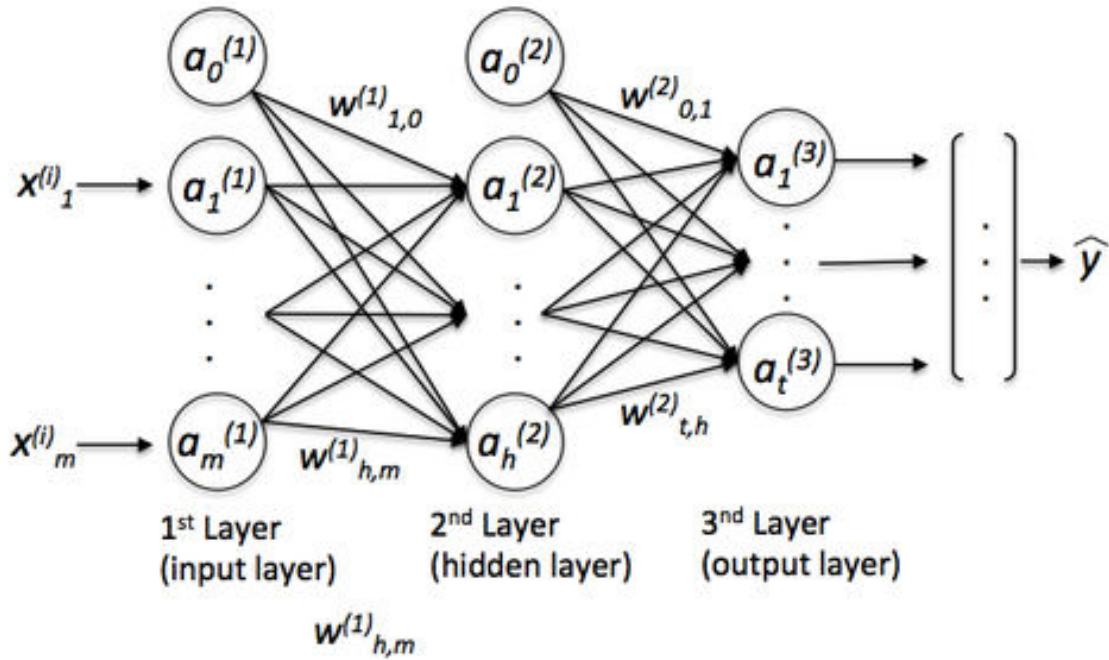


Figure 3.15. Multi Layer Perceptron: Contains several Perceptrons to enable multi-class classification. Source: [49].

gradient. The gradients are computed across each of the connection weights present in the neural network. The error is said to have been propagated backward in the network [49].

To summarize the training procedure of the neural network, the process comprises of two stages- forward pass and backward pass. When a sample is presented to the network, the network makes a prediction. This is followed by computation of error at the last layer. Then the error contributions at each layer are computed. This calculation is used to make a weight update- the weights are modified with the objective of reducing the error.

3.4.4 The Modern Neural Network

As seen in Figure 3.13, the Perceptron used step function as the activation function. An MLP that uses step function has a linear decision boundary- just like Perceptron. This issue was solved by scientists by replacing the step function [49]

$$\hat{y} = \phi(x) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases},$$

by the logistic function: $\sigma(z) = 1/(1 + \exp(-z))$.

As illustrated in Figure 3.16, the step function contains flat segments. Training of a neural network is done using backpropagation, which involves updating the weights to

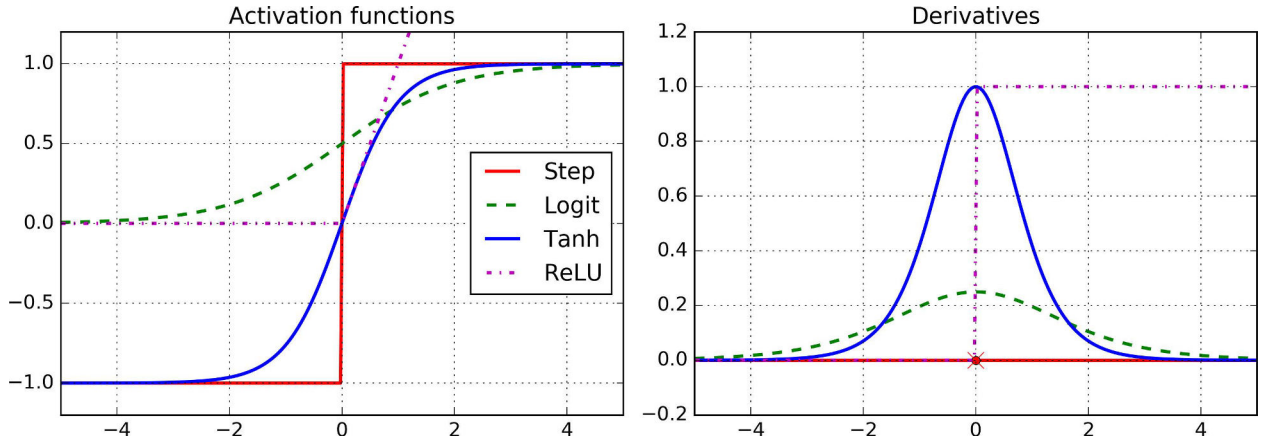


Figure 3.16. Activation functions and their derivatives: ReLU activation is the most preferred since its derivative avoids the problem of vanishing and exploding gradients. Source: [49].

reduce errors. Changing the weights needs computing the gradients of activation functions. The gradients will be zero if we use step function, hence making no improvement to weights and therefore performance of the network. The step function's derivative will be zero- gradient descent is unable to move on a flat surface. A need for activation function with suitable derivatives therefore arises.

Rectified Linear Unit or ReLU activation function is used more than logistic function. $\text{ReLU}(z) = \max(0, z)$. This activation function is most commonly used with feedforward neural networks. Applying ReLU to output of a linear transformation generates nonlinear transformation. ReLUs are almost linear, thus restoring many of the properties which make optimization (with gradient based methods) of linear models easy. Practically, ReLU is efficient to work with and fast to compute. Figure 3.17 illustrates a neural network with ReLU as the activation function.

The last layer has been modified, as compared to the MLP in 3.15, in that it uses softmax function now. When a sample x is presented to the softmax function, it computes the score as follows:

$$s_k(x) = \theta_k^T \cdot x. \quad (3.15)$$

This score is calculated for every class k . This is followed by computing the probability of every class using the softmax function [49]

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp(s_k(x))}{\sum_{j=1}^K \exp(s_j(x))}, \quad (3.16)$$

where $s(x)$ contains scores of every class for instance x , $\sigma(s(x))_k$ is the probability estimated that instance x belongs to class k , when scores of every class for that instance are given and K is the number of classes.

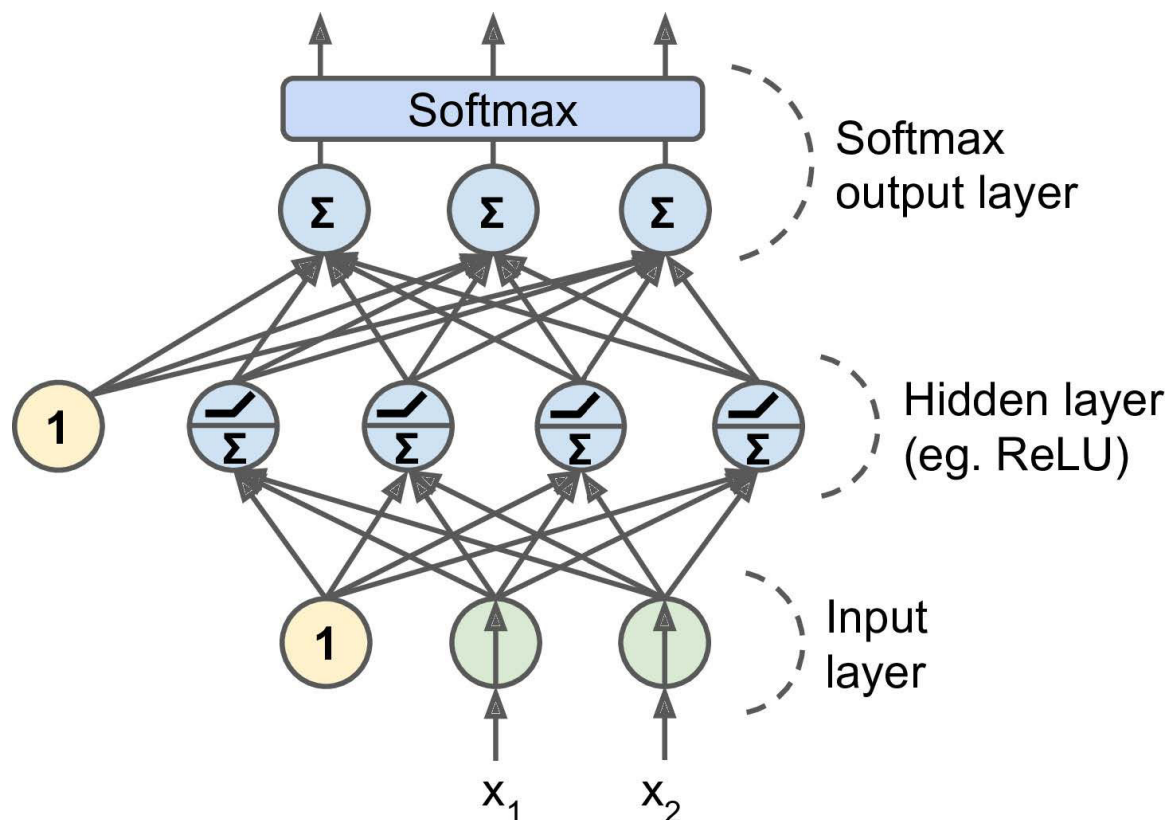


Figure 3.17. ReLU activated neural network: ReLU is the most commonly used activation function in neural networks. Source: [49].

The output of every neuron in softmax layer of Figure 3.17 is the score of one class. It is the probability that the neural network assigns to a class. Hence the number of neurons in this layer is equal to the number of classes present.

3.4.5 Deep Neural Network

A neural network with more than one hidden layer is called a Deep Neural Network. Neural networks were designed to make computers learn from experience [53]. The objective was to make them understand a given problem (such as image recognition) in the form of a hierarchy of concepts. Each concept is defined in terms of its relation to simpler concepts.

The idea behind deep learning can be clearly understood through the DNN in Figure 3.18, that attempts to classify a given image into one of three classes [53]. The aim of this DNN is to map the set of pixels in the sample image to an object identity- car, person or animal. Deep learning solves this complicated problem by breaking it down to several simpler tasks. Each of these tasks involves some kind of mapping, that is supposed to be performed by a layer of the DNN [53].

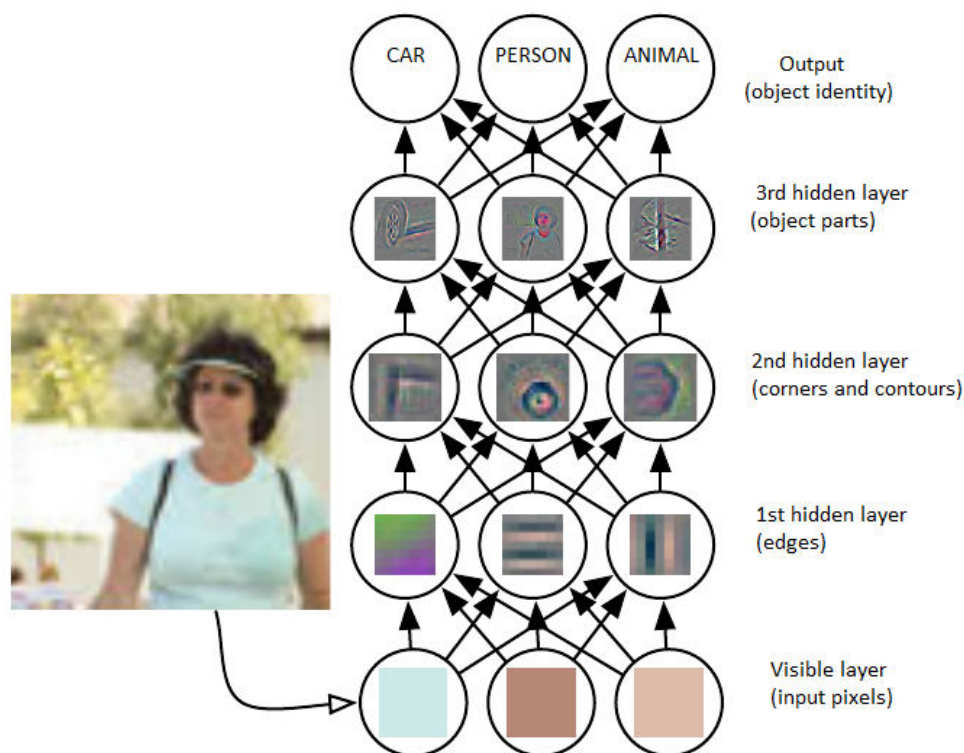


Figure 3.18. Deep Neural Network for object classification: The first hidden layer forms simple features such as edges. The layers that follow, form more complex features. Source: [53].

The very first layer is the visible layer- consisting of variables, here pixels that we can observe. The following layers are called hidden layers, since the given data- the images, do not contain the values that these layers must learn [53]. It is the job of the DNN to determine these values in order to accomplish the task of classifying the image. These layers are supposed to learn concepts that are useful for understanding the relationships in the given data.

The first hidden layer identifies very simple features such as edges [53]. It passes on this feature that it learns, on to the next layer- the second hidden layer. The second layer uses this collection of edges to identify more complicated features, such as corners and contours. Now that the second hidden layer has recognized more complicated set of features, it passes on these features to the third hidden layer [53]. This layer is able to recognize the whole parts of objects in the image, because these parts are a collection of corners and contours- features already described by the second hidden layer.

Deep learning makes the computer learn- by first building simple concepts, such as edges, and building more complicated concepts such as corners and contours on top of the simple concepts. This process continues until the model is able to accomplish the assigned task, which in this case is classifying a given image. The graph of the DNN shown in Figure 3.18 is deep- the concepts have been built on top of each other; it has several layers. This approach is therefore termed as deep learning [53].

3.4.6 Convolutional Neural Network

Convolutional Neural Networks explicitly assume that the inputs are images. The evolution of CNNs was inspired by the brain's visual cortex. These networks are used in the most complex visual tasks such as [49]

- Image search engines
- Image and video classification systems
- Self-driving cars
- Reading zip codes and digits

The neurons in the visual cortex of our brain have a small local receptive field [49]. Every neuron reacts to visual stimuli located in a limited region of the visual field. This can be clearly seen in Figure 3.19.

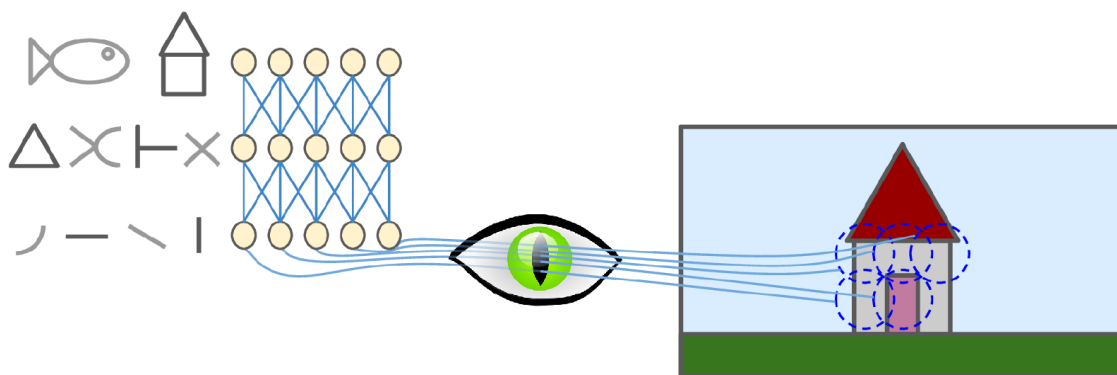


Figure 3.19. Local receptive fields of neurons in visual cortex: Each neuron in a convolutional layer is connected to a few neurons of the previous layer. Source: [49].

Figure 3.19 shows five neurons, where each neuron is able to see only a specific part of the image. The part that the neuron is able to see is called its receptive field. All the receptive fields together form the total visual field. Based on this concept, a CNN was designed such that each neuron in a CNN is connected to only some neurons of the previous layer. The architecture of CNN differs from the architecture of the fully

connected neural network in that, each neuron of the latter is connected to every neuron in the previous layer.

A limited receptive field for a neuron in a CNN implies that every neuron will observe only a part of the input image rather than the whole, as shown in 3.19. As a result, neurons in the first layer will observe simple features such as lines and curves [49]. The neurons in higher layers will observe more complex features as we go deeper. Such complex architecture of the network is capable of detecting very complex patterns in the visual field.

CNNs offer a significant advantage over DNNs when it comes to handling images [49]. A DNN might successfully detect small images, but it breaks down in case of larger images. This is due to the fact that the number of parameters rises when the input image is larger. Observing a larger image would require a greater number of neurons in the first layer- a size comparable with the number of pixels in the image. Restricting the width of the first layer would place a restriction on the number of features that the layer is able to observe. But a wider layer would give rise to a lot of connections between the first layer and the following layer. CNNs solve this problem by using partially connected layers [49].

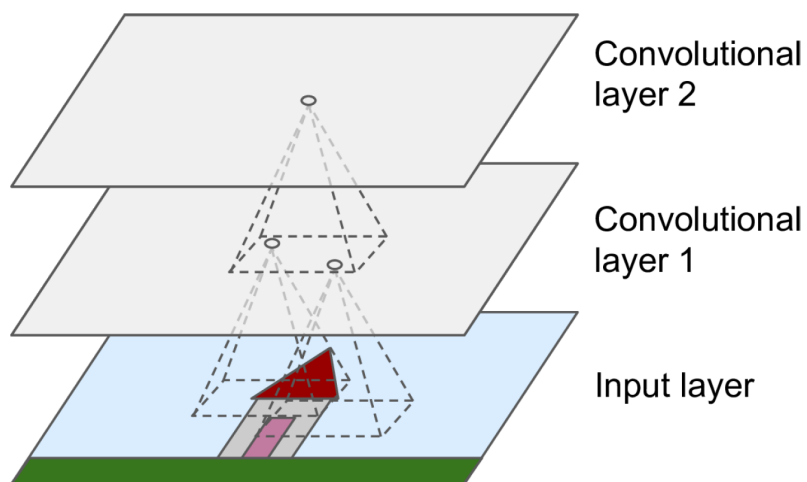


Figure 3.20. Local receptive fields of neurons in CNN layers: Each neuron has a limited receptive field. Source: [49].

Figure 3.20 shows a CNN consisting of three layers: an input layer and two convolutional layers. Each neuron in the first convolutional layer is not connected to every single pixel of the input image [49]. It is rather connected to pixels in a small rectangle of the input layer or image. Similarly, each neuron in the second convolutional layer is connected to neurons present in a small rectangle of the previous layer. Hence,

the first hidden layer concentrates on low level features. The second layer assembles these low level features to form high level features, and so on.

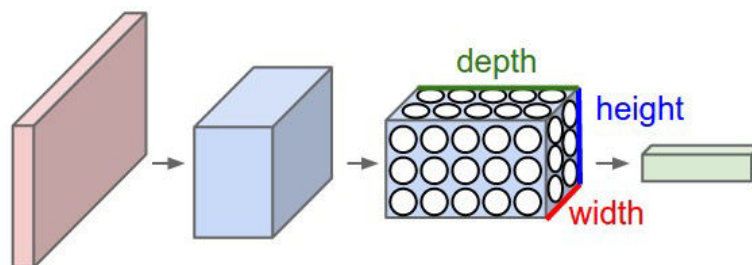


Figure 3.21. Convolutional layer structure: Convolutional layers have three dimensions- length, width and height. Source: [54].

Figure 3.21 shows the arrangement of neurons in a CNN. In a convolutional layer, neurons are arranged in three dimensions: width, height and depth. A CNN perceives the width and height of an image, as well as the number of channels in the image. It treats every image as a 3D volume and transforms this 3D input volume to a 3D output volume of neuron activations [54]. The first layer in Figure 3.21 has width, height and depth equal to that of the image, since it holds the input image. The depth is equal to there, corresponding to red, green and blue channels in the image. The layers that follow convert this 3D input volume to 3D volumes of different width, height and depth as shown in the figure. The way that these transformations take place can be understood by observing the connections between the layers in a CNN, as shown in Figure 3.22.

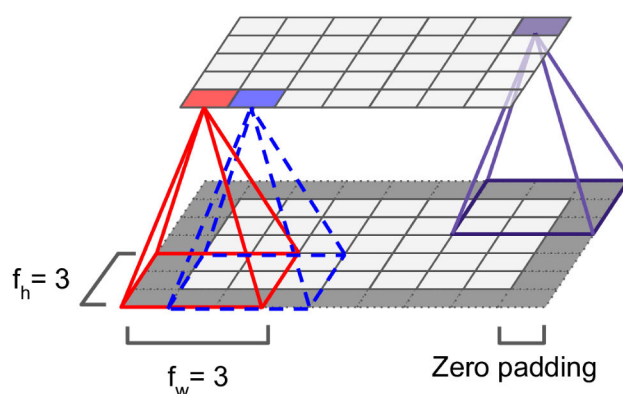


Figure 3.22. Connections between layers: Computing receptive field of neurons in convolutional layer. Source: [49].

A neuron $n_l[i, j]$ located at row i , column j of a layer l is connected to outputs of neurons located in the region: rows $\in [i : i + f_h - 1]$ and columns $\in [j : j + f_w - 1]$ of the previous layer $l - 1$, where f_h and f_w are the height and width of the receptive field [49]. Zero padding is added around the inputs, as shown in Figure 3.22, so that the next layer has the same height and width as the previous (here input) layer. The neuron marked in red has a receptive field of 3×4 since it is connected to a region of only three rows and four columns in the previous layer. Similarly, the blue neuron has a receptive field of 3×3 .

The sets of weights for a neuron are called filters or kernels. The convolution operation slides one function over another function and computes the integral of the resulting pointwise multiplication [49]. The convolutional layer parameters are a set of learnable filters. Each filter is small along width and height, but it penetrates along the full depth of the image.

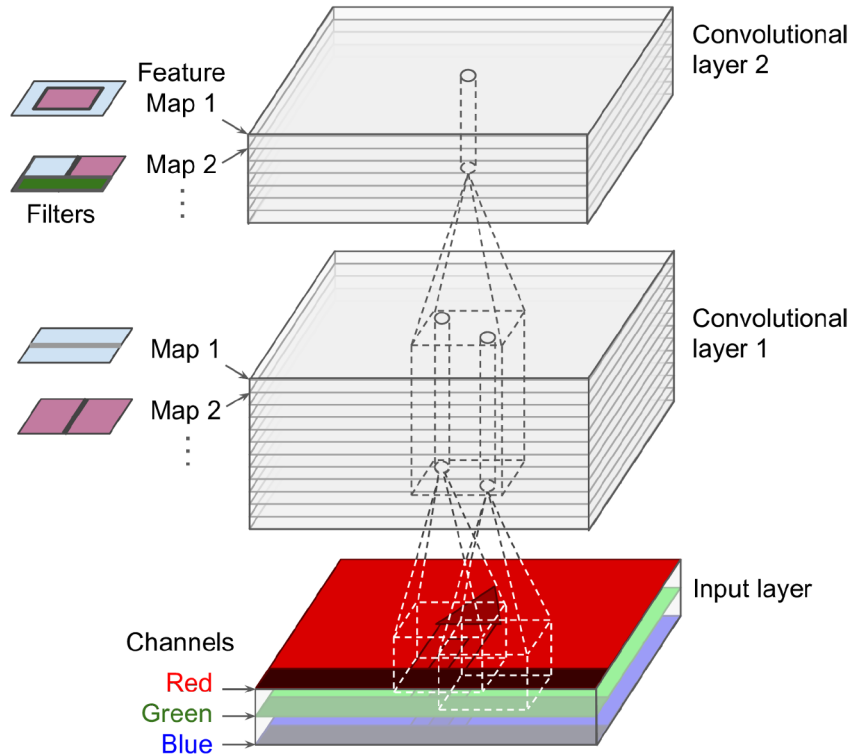


Figure 3.23. Convolutional layers with multiple feature maps: A convolutional layer is comprised of several feature maps. All neurons in a feature map share the same weights and bias. Source: [49].

Figure 3.23 shows two convolutional layers. Each of these layers consists of multiple feature maps stacked one on top of the other [49]. A feature map is comprised of

a layer of neurons that use the same filter. All neurons in a feature map share the same parameters- weights and bias; different feature maps can have different parameters. Such sharing of the same parameters between several neurons greatly reduces the total number of parameters in the network [49].

The input image here is composed of three channels: red, green and blue. Each neuron in convolutional layer 1 extends through the full depth of the image at the input layer. But it covers only a small height and width across the image. A single map contains multiple neurons and several maps combine to form a convolutional layer [49].

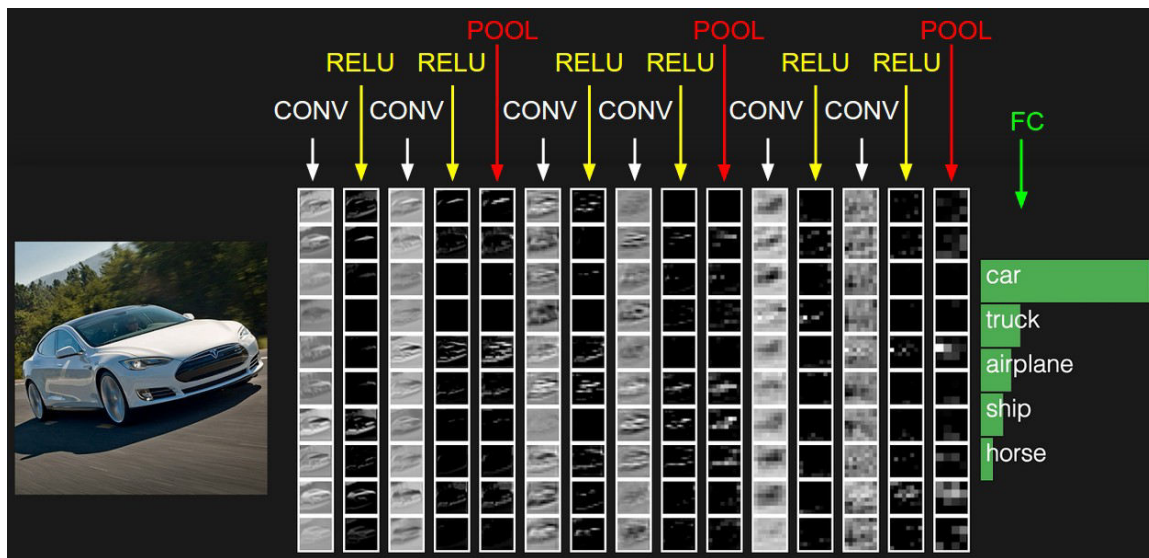


Figure 3.24. Example of CNN: A deep CNN used for image classification. Source: [54].

Figure 3.24 shows an example application of CNN [54]. The input layer stores the image presented to the CNN. Each convolutional layer is a collection of feature maps stacked one on top of the other. Also, each convolutional layer is followed by a ReLU layer- that applies element wise activation function to the outputs of the convolutional layer. The last layer is a fully connected layer that stores the class scores (only the top five scores are shown here) [54].

3.5 SUMMARY

This chapter has presented a background on machine learning. The first part introduces machine learning and its types. This is followed by reasons that state why machine learning was found to be a good choice for solving the problem of AMC in this research. The chapter then gives the theoretical background of classifiers KNN, SVM, DTs, RFs, boosted trees and ANNs.

CHAPTER 4

IMPLEMENTATION OF MACHINE LEARNING TECHNIQUES FOR AUTOMATIC MODULATION CLASSIFICATION

Machine Learning techniques were utilized to solve the problem of AMC in the thesis. The dataset used for this purpose is Radio Machine Learning (RML) dataset [51]. The RML dataset consists of 11 modulation classes- 8 digital and 3 analogue, at SNR values ranging from -20dB to 18dB, at steps of 2dB. Modulation types in the dataset are:

- 8-PSK
- AM-DSB
- AM-SSB
- BPSK
- CPFSK
- GFSK
- 4-PAM
- 16-QAM
- 64-QAM
- QPSK
- WBFM

This research is concerned with the digital modulation schemes only. The dataset is in the form of a Python dictionary. After picking only the digital modulation types, we have 160 (modulation, SNR) pairs and each pair is assigned a (1000, 2, 128) array- meaning 1000 samples, with each sample a (2, 128) array. This results in a total of 160,000 samples. The given dataset is therefore a (160000, 2, 128) array.

4.1 FEATURE EXTRACTION

Cyclic-moment based features are widely used for the purpose of modulation recognition [55]. A total of 32 features have been designed and the classifiers have been trained on these features, rather than raw data. The first 16 features are based on 0 cyclic time lag and the next 16 features are based on 8 cyclic time lag. The features are of the

form [55]

$$s_{mn} = f_m(\{x_i^n x_{i+T}^n\}). \quad (4.1)$$

which is the the m th order statistic on the n th power of the instantaneous or time delayed received signal x_i .

The 16 expert features based on 0 cyclic time lag are:

1. First moment of the first power of the complex signal

$$s_1 = f_1(\{x_i\}) = \frac{1}{N} \sum_{i=0}^{N-1} x_i. \quad (4.2)$$

2. First moment of the first power of the amplitude

$$s_2 = f_1(\{|x_i|\}) = \frac{1}{N} \sum_{i=0}^{N-1} |x_i|. \quad (4.3)$$

3. First moment of the first power of the phase

$$s_3 = f_1(\{\angle x_i\}) = \frac{1}{N} \sum_{i=0}^{N-1} \angle x_i, \quad (4.4)$$

where the phase of a complex number a is defined as $\angle a = \tan^{-1}\left(\frac{I(a)}{R(a)}\right)$.

4. First moment of the first power of the absolute value of the phase

$$s_4 = f_1(\{|\angle x_i|\}) = \frac{1}{N} \sum_{i=0}^{N-1} |\angle x_i|. \quad (4.5)$$

5. First moment of the second power of the complex signal

$$s_5 = f_1(\{x_i^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} x_i^2. \quad (4.6)$$

6. First moment of the second power of the amplitude

$$s_6 = f_1(\{|x_i|^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} |x_i|^2. \quad (4.7)$$

7. First moment of the second power of the phase

$$s_7 = f_1(\{(\angle x_i)^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} (\angle x_i)^2. \quad (4.8)$$

8. First moment of the second power of the absolute value of the phase

$$s_8 = f_1(\{|\angle x_i|^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} |\angle x_i|^2. \quad (4.9)$$

Features (7) and (8) should give the same result.

9. Second moment of the first power of the complex signal

$$s_9 = f_2(\{x_i\}) = \frac{1}{N} \sum_{i=0}^{N-1} x_i^2 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} x_i \right)^2. \quad (4.10)$$

10. Second moment of the first power of the amplitude

$$s_{10} = f_2(\{|x_i|\}) = \frac{1}{N} \sum_{i=0}^{N-1} |x_i|^2 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} |x_i| \right)^2. \quad (4.11)$$

11. Second moment of the first power of the phase

$$s_{11} = f_2(\{\angle x_i\}) = \frac{1}{N} \sum_{i=0}^{N-1} (\angle x_i)^2 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} \angle x_i \right)^2. \quad (4.12)$$

12. Second moment of the first power of the absolute value of the phase

$$s_{12} = f_2(\{|\angle x_i|\}) = \frac{1}{N} \sum_{i=0}^{N-1} |\angle x_i|^2 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} |\angle x_i| \right)^2. \quad (4.13)$$

13. Second moment of the second power of the complex signal

$$s_{13} = f_2(\{x_i^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} x_i^4 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} x_i^2 \right)^2. \quad (4.14)$$

14. Second moment of the second power of the amplitude

$$s_{14} = f_2(\{|x_i|^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} |x_i|^4 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} |x_i|^2 \right)^2. \quad (4.15)$$

15. Second moment of the second power of the phase

$$s_{15} = f_2(\{(\angle x_i)^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} (\angle x_i)^4 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} (\angle x_i)^2 \right)^2. \quad (4.16)$$

16. Second moment of the second power of the absolute value of the phase

$$s_{16} = f_2(\{|\angle x_i|^2\}) = \frac{1}{N} \sum_{i=0}^{N-1} |\angle x_i|^4 - \frac{1}{N^2} \left(\sum_{i=0}^{N-1} |\angle x_i|^2 \right)^2. \quad (4.17)$$

The next 16 expert features based on 8 cyclic time lag are computed by replacing $x_i = x_i x_{i+8}$. These features are designed using the same 16 formulae as above.

The feature extraction stage therefore transforms the raw data into a set of designed features. Each (2, 128) sample is transformed into a 32 dimensional vector. This results in a (160000, 32) dataset.

4.2 CLASSIFICATION

Half of the samples belonging to each (modulation, SNR) pair were randomly picked and included in the training set, and the other half were saved for the test set. Each classifier is therefore trained on 80,000 and tested on 80,000 samples. We first train every classifier on the first 16 features- feature set 1, based on 0 cyclic time lag. Hence the classifier is trained on (80000, 16) data and tested on (80000, 16) data.

The feature set is then expanded to include the next 16 features, resulting in a total of 32 features- feature set 2, based on 8 cyclic time lag. Now the classifier is trained on (80000, 32) data and tested on (80000, 32) data.

The following machine learning classifiers were trained:

1. k-Nearest Neighbors
2. Support Vector Machine
3. Decision Tree and Random Forests
4. Adaboost and Gradient Boosted ensemble of trees
5. Artificial Neural Network: Deep Neural Network and Convolutional Neural Network

The KNN, SVM, DT, RF, boosted ensemble and ANN classifiers were implemented in the machine learning framework scikit-learn, while DNN and CNN were implemented in the deep learning framework TensorFlow. Each of the following sections gives a detailed description of a model's implementation. The following are discussed for each machine learning model:

1. Hyperparameters: Hyperparameters are the parameters of the learning algorithm that are required to be set before the training begins, and they remain constant during the training phase. This section gives an explanation of the hyperparameters that we have tuned in the experiments.
2. Experimental setup and results: Classifiers KNN, SVM, DT, RF and MLP in were first trained using the default values for hyperparameters, followed by grid search and randomized search for hyperparameter tuning. Grid and randomized search are performed over the same hyperparameter space. Classifier accuracy and confusion matrix were used as the performance metrics to quantify the quality of each classifier's predictions. The test accuracy reported in the tables is the best accuracy obtained out of all the SNR values. The classifiers' accuracy on the test set is plotted. The performance is strongly dependent on the SNR value. The RML dataset consists of 20 SNR values, so a separate test accuracy was computed for each of these SNR values. Confusion matrix was computed for 18dB samples in the test set, for the model giving the highest accuracy in case of each type of classifier. Other essential evaluation metrics, namely precision and recall can then be conveniently inferred from the matrix.

4.2.1 K-Nearest Neighbors

We trained KNN classifier with:

1. Default parameters
2. Grid search
3. Randomized search

4.2.1.1 HYPERPARAMETERS

KNN classifier accepts the following hyperparameters

- *n_neighbors*: The number of neighbors implies that the classifier considers *n_neighbors* nearest points to the query data point, in order to predict the label of this data point.
- *algorithm*: This is the algorithm used to compute the nearest neighbors. The mode 'auto' selects the best algorithm out of ball-tree and kd-tree, to be used for fitting KNN to the data. The kd-tree or k-dimensional tree is a space-partitioning data structure that organizes points in a k-dimensional space. They are utilized in problems that involve search a multidimensional search key. Ball-tree starts with creating two clusters that resemble balls in an n-dimensional space. Any point in this space will belong to the cluster to which the distance between this point and the cluster's centroid is the least. Each cluster is then broken down into two smaller clusters. This process of breaking down continues till some specified depth is reached. The parameter *algorithm* decides the indexing data structure to be utilized for speeding up the KNN algorithm.
- *weights*: When set to 'uniform', every one of the k neighbors has an equal vote, no matter what its distance from the query point. Another option is 'distance', where the vote of points closer to the query point is given greater priority as compared to the points further away.
- *leaf_size*: Leaf size is the size of leaf in the selected *algorithm* - ball-tree or kd-tree. Larger value of this parameter leads to faster initial indexing structure information. The price paid is slower classification of the query point.
- *metric*: It governs the calculation of distance in space. The metric 'minkowski' distance is euclidean distance or Manhattan distance, depending on whether *p* is equal to 2 or 1 respectively.

4.2.1.2 EXPERIMENTAL SETUP AND RESULTS

- Default parameters setting: With no hyperparameter tuning, the following classifier gives an accuracy of 64.775% and 63.3% on feature sets 1 and 2 respectively on the test set.
`KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski')`

Table 4.1. KNN Classifiers Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Default parameters	set 1	64.775%	0.15 seconds
Default parameters	set 2	63.3%	0.25 seconds
$n_neighbors = 20$	set 1	66.9%	0.14 seconds
$n_neighbors = 20$	set 2	65.125%	0.27 seconds
Grid search	set 1	67.625%	7 minutes
Grid search	set 2	65.125%	44 minutes
Randomized search	set 1	68.625%	10 minutes
Randomized search	set 2	68.15%	88 minutes

, *metric_params* = None, *n_jobs* = 1, *n_neighbors* = 5, *p* = 2, *weights* = 'uniform')

The accuracy with all default parameter settings increases slightly when the number of neighbors is set to 20, with all other parameter values unchanged.

- Grid search: The search was performed over the hyperparameter space:
`params = {'n_neighbors': np.arange(1, 31, 2), 'metric': ['euclidean', 'cityblock']}`
As seen in Table 4.1, grid search takes longer, but gives a better classifier. The best estimator is found to be:
`KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=19, p=2, weights='uniform')`
- Randomized search: Randomized search gives the best results amongst all. The best estimator is found to be
`KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='cityblock', metric_params=None, n_jobs=1, n_neighbors=27, p=2, weights='uniform')`
when the classifier is trained on feature set 1.

Figure 4.1 depicts the performance of the second last classifier in Table 4.1. The classification accuracy on the test set rises with SNR, to a maximum of 68.62%. Figure 4.2 depicts the confusion matrix computed on 18dB samples in the test set. Confusion matrix is a much better metric for evaluating a classifier's performance. The true labels are along the rows and the predicted labels are along the columns.

The matrix can be interpreted as follows. The total number of points in the first class 8-PSK was 523 (obtained by adding the first row of the matrix). KNN was successfully able to identify 260 of these as 8-PSK. But it identified 2 of the points incorrectly as BPSK, 2 of the points as CPFSK and so on. 260 is the number of true positives- the data points that belonged to class 8-PSK, and were correctly identified by

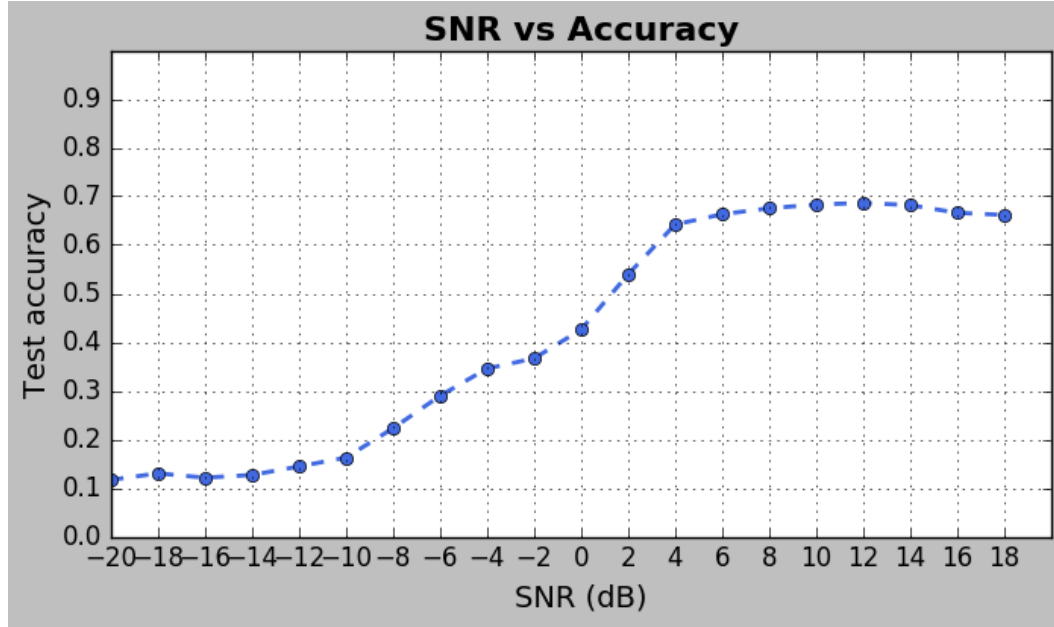


Figure 4.1. Performance of KNN Classifier given by randomized search on feature Set 1: The plot shows that the classification accuracy increases as the SNR rises.

the classifier. While the values other than 260 in the first row are false positives- the data points that actually belonged to 8-PSK, but were incorrectly classified into other classes.

Looking at the first column now, the classifier identified 500 data points as the points belonging to class 8-PSK. Out of these, 260 points (called true positives) actually belonged to class 8-PSK. 1 point belongs to BPSK, 56 points belong to CPFSK and so on. The values other than 260 in the first column are called false negatives- the data points that were detected as those belonging to 8PSK, but actually belonged to other classes as indicated by the rows. Such interpretation of the confusion matrix leads us to two essential metrics widely used to evaluate machine learning models, namely precision and recall. Precision is given by the formula

$$\text{precision} = \text{true positives} / (\text{true positives} + \text{false positives}) \quad (4.18)$$

While recall is given by

$$\text{recall} = \text{true positives} / (\text{true positives} + \text{false negatives}) \quad (4.19)$$

Confusion matrix gives the complete information about the performance of the classifier on every class present in our dataset. The values precisely indicate the correct and wrong classifications, but the matrix can be visualized more concisely after normalizing it. The matrix shown in Figure 4.3 has been normalized by the number of

Confusion Matrix Without Normalization								
	8PSK	BPSK	CPFSK	GFSK	PAM4	QAM16	QAM64	QPSK
8PSK	260	2	2	2	2	66	54	135
BPSK	1	485	0	0	21	9	11	4
CPFSK	56	0	451	119	1	17	9	63
GFSK	16	1	46	377	0	10	15	16
PAM4	1	12	0	0	474	6	8	1
QAM16	24	0	0	0	1	159	188	19
QAM64	13	0	0	0	1	193	190	12
QPSK	129	0	1	2	0	40	25	250

Figure 4.2. Confusion matrix: Performance of KNN classifier on each modulation class.

elements in every class, also known as the class support size. Each value of a row was divided by the total number of data points in that row. This implies that the diagonal elements are the precision values for the respective classes indicated by the rows- the fraction of data points that the classifier correctly classified for that class.

For example, 91% of the samples belonging to class BPSK were correctly classified by our KNN classifier. While the off-diagonal elements indicate the fraction of data points misclassified by the classifier. A much better visualization is obtained by color map, as shown in the second plot of Figure 4.3. A quick look at the diagonal indicates which modulation classes were easy to classifier for the KNN classifier, and which were the classes that the classifier confused with the other classes. For instance, it was able to identify most of the BPSK and 4-PAM samples correctly, while it was thoroughly confused between 16-QAM and 64-QAM.

4.2.1.3 SUMMARY

KNN classifier was trained on feature sets 1 and 2. Initially, the hyperparameters were set to default values- which gave an accuracy of 64.77% on the test set. Approximately 2% increase in accuracy was observed when the number of neighbors was increased to 20. Grid search and randomized search were used for hyperparameter tuning. Grid search resulted in 67.62% accuracy, while randomized search further improved it to 68.62% in a significantly lower time duration as compared to grid search.

4.2.2 Support Vector Machine

Support Vector Machine (SVM) was trained with:

1. Default parameters
2. Grid search
3. Randomized search

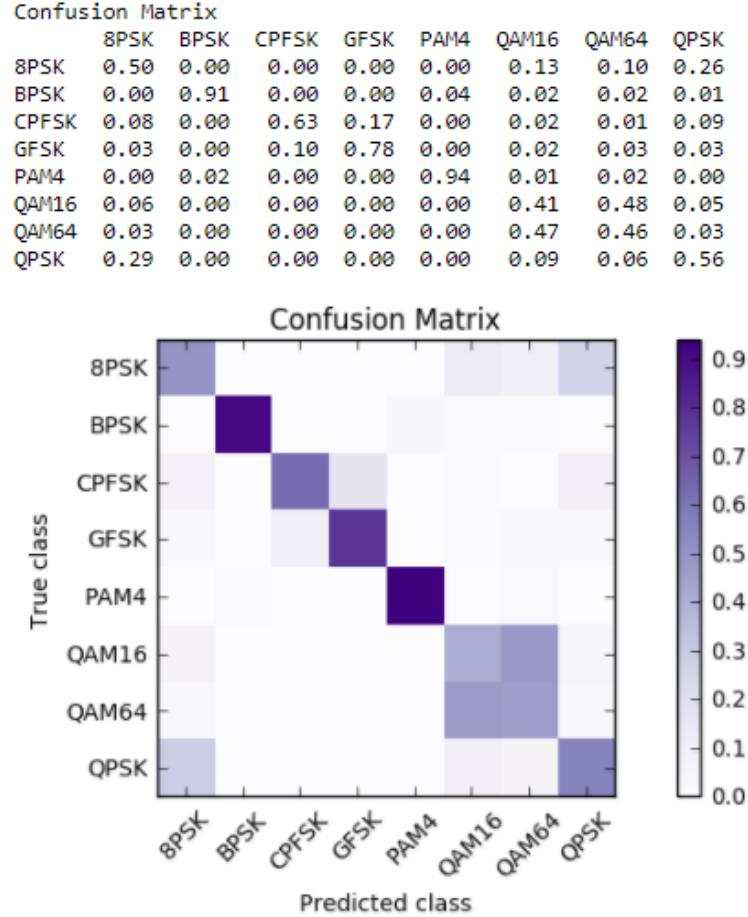


Figure 4.3. Confusion matrix: Precision values of KNN classifier are computed and confusion matrix visualized for all modulation classes.

4.2.2.1 HYPERPARAMETERS

- *C*: This is a penalty parameter for misclassifying a sample. A smaller value, for example 1, implying high bias and low variance means that the classifier may misclassify samples. The classifier is tolerant of misclassified samples. Whereas a large value, such as 100, implying low bias and high variance means that the classifier gets heavily penalized for misclassifying the data points. The decision boundary in this case is more severe- the classifier tries hard to get all the predictions right. A very high value, such as 1000, will lead to overfitting. This happens because the classifier tries so hard to fit to the training data, that it fails to generalize well to the data. It might end up misclassifying the unseen samples or test data.
- *gamma*: It decides the spread or stretch of the decision region. A low value like 0.01 makes the decision boundary a simple one. It is not much curvy and the decision region is quite broad. A higher value such as 1, makes the decision

boundary curvy and it fits better around the data points. A further increase makes the boundary tighter around the points.

- *kernel*: It can be set to 'linear', 'rbf', 'poly', 'sigmoid' or 'precomputed'. The recommended way to select one of these is to start with a simple one- 'linear' and see if it fits the data, followed by more complicated kernels. We have used RBF kernel in all the experiments.
- *cache_size*: This is the kernel cache size (in MB). As the number of storage vectors increases, SVM's computational and storage requirements increase rapidly. The size of the kernel cache strongly impacts the run time for large problems. Hence, it is recommended to set the cache size parameter to a value higher than the default value of 200 MB.

4.2.2.2 EXPERIMENTAL SETUP AND RESULTS

Table 4.2. SVM Classifiers Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Default parameters	set 1	72.35%	5.20 minutes
Default parameters	set 2	73.15%	8.90 minutes
Grid search	set 1	79.05%	15 hours
Grid search	set 2	81.62%	22 hours
Randomized search	set 1	78.675%	5.9 hours
Randomized search	set 2	77.675%	5.4 hours

The cache size was changed in each case, in order to speed up the computations.

- Default parameters setting:
`SVC(C = 1.0, cache_size = 5000, class_weight = None, coef0 = 0.0, decision_function_shape = None, degree = 3, gamma = 'auto', kernel = 'rbf', max_iter = -1, probability = False, random_state = None, shrinking = True, tol = 0.001, verbose = False)`
resulted in a classification accuracy of about 73.15% within a few minutes. The results can be further improved by hyperparameter tuning as discussed below.
- Grid search: Grid search was performed over the following hyperparameter space:
`params = 'C' : np.logspace(-3, 2, 6), 'gamma' : np.logspace(-3, 2, 6)`
It takes significantly more time than the previous case, but gives a better classification accuracy at the same time. The best estimator was found to be:
`SVC(C = 100.0, cache_size = 5000, class_weight = None, coef0 = 0.0, decision_function_shape = None, degree = 3, gamma = 0.01, kernel = 'rbf', max_iter = -1, probability = False, random_state = None, shrinking = True, tol = 0.001, verbose = False)`

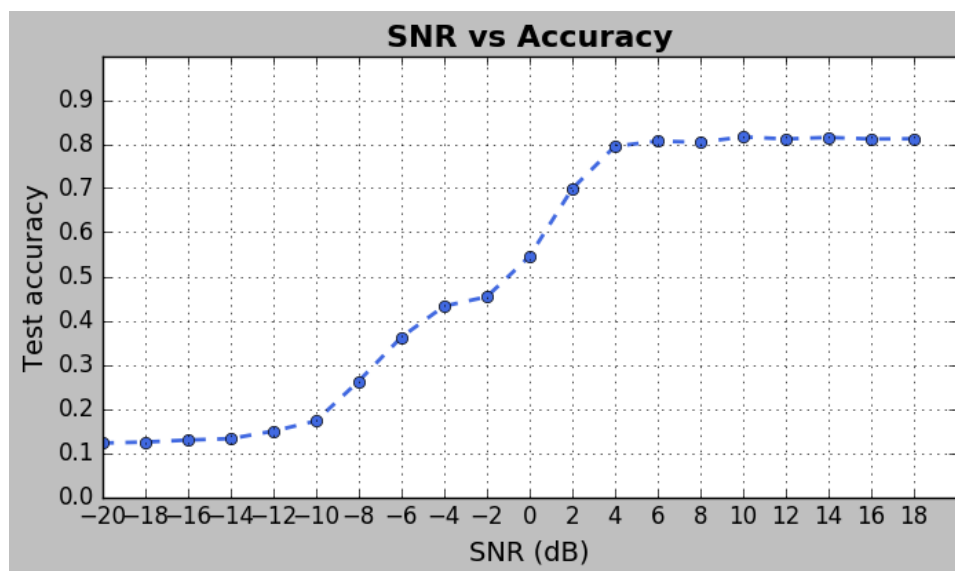


Figure 4.4. Performance of SVM classifier on feature Set 2: The plot shows that the classification accuracy increases as the SNR rises.

Figure 4.4 depicts the performance of the SVM classifier for various SNR values. An accuracy of 81.62% is achieved, which is a significant improvement over 68.62% accuracy given by KNN classifier. Figure 4.5 shows the confusion matrix computed, normalized and plotted for this classifier.

- Randomized search: Randomized search was performed on the same hyperparameter space as grid search. The best estimator found was: `SVC(C = 10.0, cache_size = 7000, class_weight = None, coef0 = 0.0, decision_function_shape = None, degree = 3, gamma = 0.10000000000000001, kernel = 'rbf', max_iter = -1, probability = False, random_state = None, shrinking = True, tol = 0.001, verbose = False)`. The run time was significantly less- 2.5 of the time taken by grid search, but at the cost of lower classification accuracy of 78.675%.

4.2.2.3 SUMMARY

All the SVM classifiers implemented outperformed the KNN classifiers. The accuracy on test set was 73.15% with all the default parameters settings. It improved to 78.67% with randomized search. Grid search resulted in the best classifier amongst all the SVMs trained- 81.62% test accuracy.

4.2.3 Decision Trees and Random Forests

A single decision tree as well as ensemble of decision trees was trained on the data. The following experiments were performed:

1. Decision tree and random forest with default parameters.

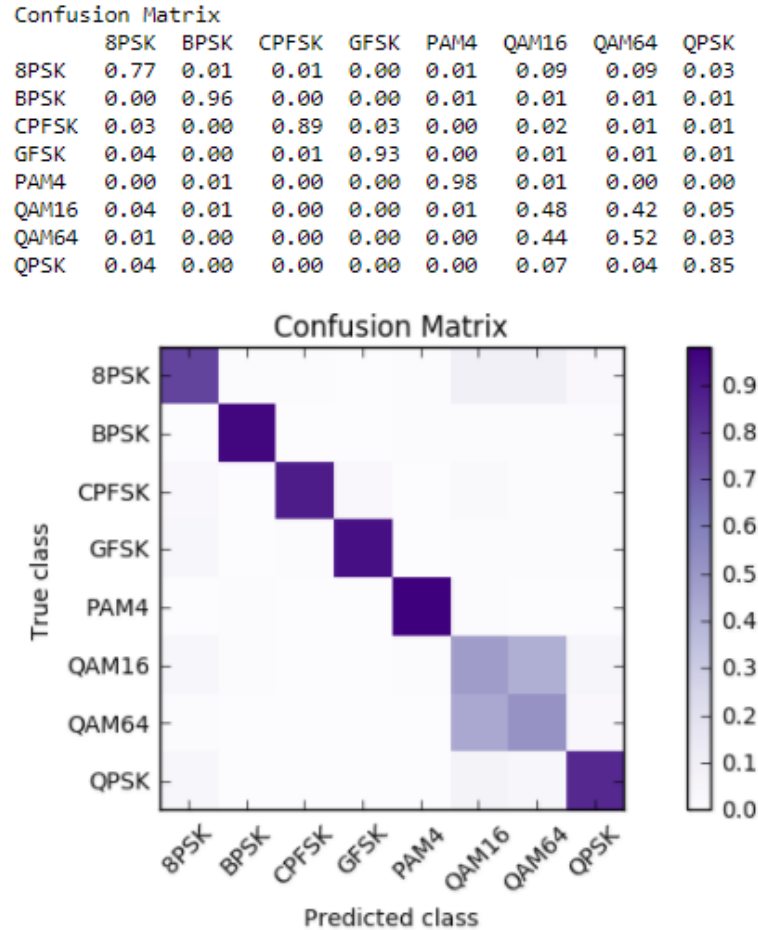


Figure 4.5. Confusion matrix: Precision values of SVM Classifier are computed and confusion matrix visualized for all modulation classes.

2. Hyperparameter optimization of decision tree by using grid search and randomized search.
3. Hyperparameter optimization of random forest by using grid search and randomized search.
4. Bagging and pasting.
5. Boosted ensemble of decision trees: Adaptive boosting, gradient boosting.

4.2.3.1 DECISION TREE

Decision tree classifier takes the following hyperparameters:

- *criterion*: The impurity measure can be ‘gini’ or ‘entropy’. A node’s gini score is 0, i.e. the node is pure, if all the samples at this node belong to the same category. A set’s entropy is 0 when all its samples belong to the same class. Both these modes produce almost the same result. But ‘gini’ makes the training a bit faster; ‘entropy’ produces more balanced trees.

- *max_features*: This is the number of features that the algorithm considers while looking for the best split. This can be an integer if one wants to specify a particular number of features for the split. If this parameter is a float, then this is the fraction of features to be considered. Setting it to 'auto' or 'sqrt' makes it equal to the square root of the number of features. Other options are 'log2' that is equal to $\log_2(\text{number of features})$. When left to None, it is equal to the number of features.
- *max_depth*: Depth of a tree is the longest path from the root node to a terminal node (leaf). If no value is specified, the tree is expanded until the leaves become pure- instances at each node belong to the same class. This can lead to a very deep tree and result in over fitting. It is therefore recommended to limit the *max_depth*. This is called pruning the tree. A deeper decision tree leads to a more complex decision boundary.
- *min_samples_split*: This is the minimum number of samples needed to split an internal node of the tree. This can be an integer or float if one wants to specify what fraction of samples to use for this purpose.
- *min_samples_leaf*: The minimum number of samples that a leaf must have.
- *min_weight_fraction_leaf*: Same as the previous parameter, but expressed as weighted fraction of total number of weighted samples.
- *max_leaf_nodes*: Maximum number of leaf nodes in the tree.

Regularization is done by increasing the value of *min_* hyperparameters and decreasing the value of *max_* ones. Following is a summary of the experimental runs.

Table 4.3. Decision Tree Classifiers Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Default parameters	set 1	80.4%	2.69 seconds
Default parameters	set 2	82.00%	5.31 seconds
Grid search	set 1	82.7%	12.18 hours
Grid search	set 2	83.72%	19.63 hours
Randomized search	set 1	80.5%	13.60 seconds
Randomized search	set 2	83.525%	62.84 seconds

- Default parameters setting:
`DecisionTreeClassifier(class_weight = None, criterion = 'gini', max_depth = None, max_features = None, max_leaf_nodes = None, min_impurity_split = 1e-07, min_samples_leaf = 1, min_samples_split = 2, min_weight_fraction_leaf = 0.0, presort = False, random_state = None, splitter = 'best')`
 Table 4.3 shows that DT classifier performs better (82.7%) than SVM that took more than 20 hours for a lower classification accuracy of 81.62%. Moreover, a

randomized search of just a few seconds gives near about the same accuracy (83.525%) as several hours of grid search (83.72%).

- Grid search:

Grid search was performed over a vast hyperparameter space specified by:

```
params = {'max_depth': list(range(9, 18)), 'max_leaf_nodes':  
list(range(50, 150)), 'min_samples_split': [2, 3, 4], 'max_features':  
[None, 'auto', 'sqrt', 'log2'], 'criterion': ['gini', 'entropy']}
```

With a run time of 12 hours, the best estimator was found to be:

```
DecisionTreeClassifier(class_weight = None, criterion = 'gini', max_depth =  
13, max_features = None, max_leaf_nodes = 149, min_impurity_split =  
1e-07, min_samples_leaf = 1, min_samples_split =  
2, min_weight_fraction_leaf = 0.0, presort = False, random_state =  
42, splitter = 'best').
```

that gave a classification accuracy of 83.72% in 19 hours. The results are shown in Figure 4.6. The DT classifier outperforms the SVM classifier that was able to achieve accuracy up to 81.62%.

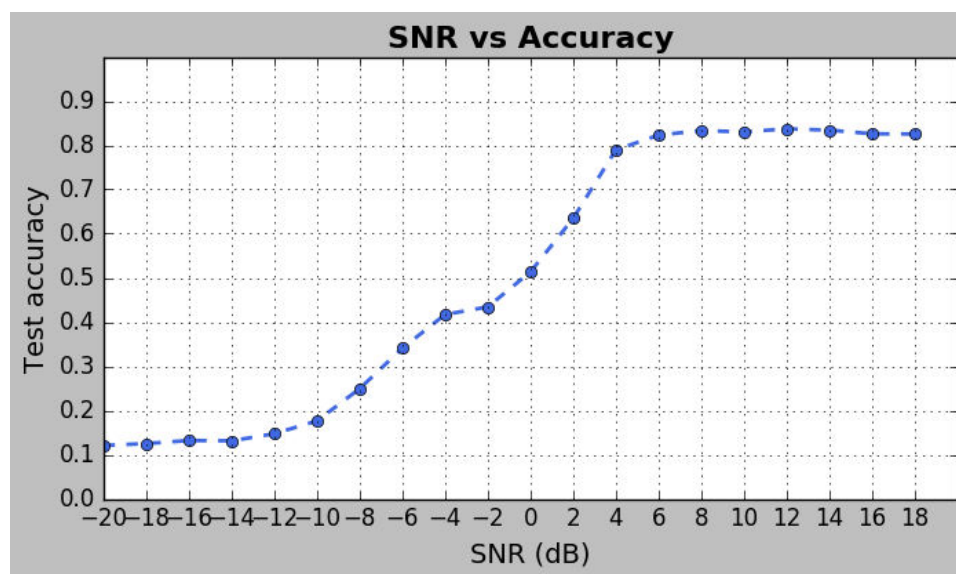


Figure 4.6. Performance of Decision Tree classifier from grid search on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

- Randomized search: Randomized search performed over the same hyperparameter space takes a just a few seconds to complete and gives a classifier nearly as good as grid search.

4.2.3.2 RANDOM FOREST

Random forest classifier takes the following hyperparameters:

- *n_estimators*: number of trees in the ensemble.
- Other parameters are the various *max_* and *min_* parameters of DT, that control the growth of the tree.

Table 4.4. Random Forest Classifiers Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Default parameters (10 trees)	set 1	83.3%	4.59 seconds
Default parameters (10 trees)	set 2	83.675%	5.39 seconds
Grid search (230 trees)	set 1	83.05%	9.68 hours
Grid search (230 trees)	set 2	84.425%	11.95 hours
Randomized search (230 trees)	set 1	83.025%	26 minutes
Randomized search (250 trees)	set 2	84.425%	29 minutes

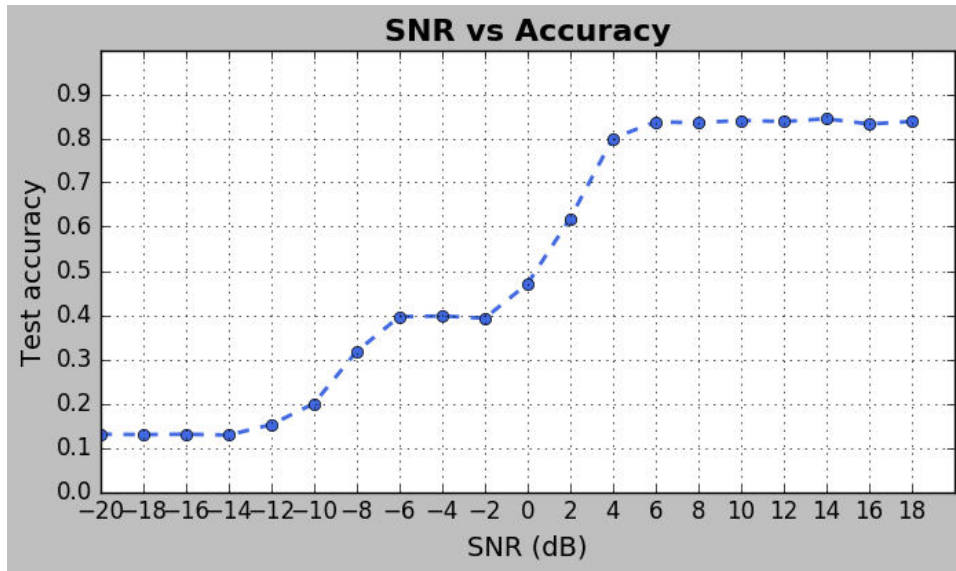


Figure 4.7. Performance of Random Forest classifier From randomized search on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

Grid and randomized search were performed on the following hyperparameter space:

params = { 'n_estimators' = [200, 230, 250], 'max_depth' = *list(range(13, 16))*,

```

‘max_leaf_nodes’ : list(range(145, 150)), ‘min_samples_split’ : [2, 3, 4],
‘bootstrap’ : [True, False]}

```

Ensemble learning and randomized search further raise the classification accuracy to 84.425%. Once again, randomized search takes considerably lower time- just 29 minutes, as compared to grid search- 11.95 hours, with both giving almost the same performance. Figure 4.7 shows the increase in the classification accuracy with the SNR. The RF’s performance is an improvement over 83.72% accuracy of DT classifier.

4.2.3.3 BAGGING AND PASTING

Bagging and pasting classifier has the following hyperparameters:

- *base_estimator*: The base estimator to fit on random subsets of the data; we used decision tree for all of the experiments.
- *n_estimators*: Number of base estimators in the ensemble.
- *max_samples*: Number of samples to be drawn from the dataset, to train each base estimator. This can be an integer or fraction of the number of samples.
- *max_features*: Number of features to be drawn from the dataset, to train each base estimator. This can be an integer or fraction of the number of features.
- *bootstrap*: When set to True, the samples are drawn with replacement to implement bagging. It should be set to False for pasting.

Table 4.5. Bagging and Pasting Classifiers Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Bagging classifier (400 trees)	set 1	79.25%	7.99 seconds
Pasting classifier (400 trees)	set 2	79.525%	7.47 seconds
Bagging classifier (400 trees)	set 1	81.475%	18.12 seconds
Pasting classifier (400 trees)	set 2	82.55%	18.26 seconds

Figure 4.8 illustrates the performance of pasting classifier, comprising of an ensemble of 400 DTs. The classifier trained on feature set 2 gives the best result (82.55%) among all the bagging and pasting classifiers implemented.

4.2.3.4 BOOSTED TREES

Adaboost and Gradient Boosting algorithms were used to train an ensemble of trees. The Adaboost classifier accepts the following hyperparameters:

- *learning_rate*: This parameter scales the contribution of each tree. The classifier shrinks the contribution of each tree by this value. More trees are required in the ensemble if learning rate is low, for example 0.1. A lower value means better

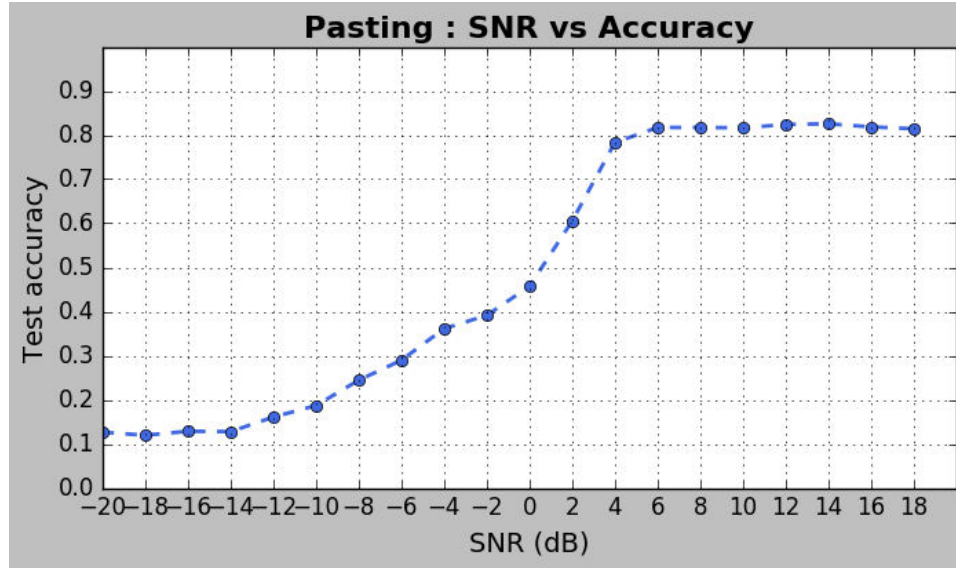


Figure 4.8. Performance of Pasting ensemble classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

generalization - hence it is a regularization parameter. This regularization technique is called shrinkage.

- *n_estimators*: number of trees in the ensemble.
- *algorithm*: One option is SAMME which is a discrete boosting algorithm. A better option is SAMME.R that converges faster since it takes less iterations and gives a lower test error as well.

Gradient boosting classifier takes the following hyperparameters:

- *learning_rate*: It is the same as that described for Adaboost.
- *n_estimators*: number of trees in the ensemble.
- Other parameters are the various *max_* and *min_* hyperparameters previously explained, that control the growth of the decision tree.

Table 4.6. Boosted Trees Performance

Experimental Setup	Feature Set	Accuracy	Training Time
Adaboost classifier (400 trees)	set 1	80.3%	2.72 seconds
Adaboost classifier (400 trees)	set 2	82.05%	5.45 seconds
Gradient boosted classifier (400 trees)	set 1	84.175%	1.80 minutes
Gradient boosted classifier (400 trees)	set 2	85.575%	3.48 minutes
Gradient boosted, grid search (500 trees)	set 1	84.725%	48.53 minutes
Gradient boosted, grid search (500 trees)	set 2	86.525%	85.86 minutes

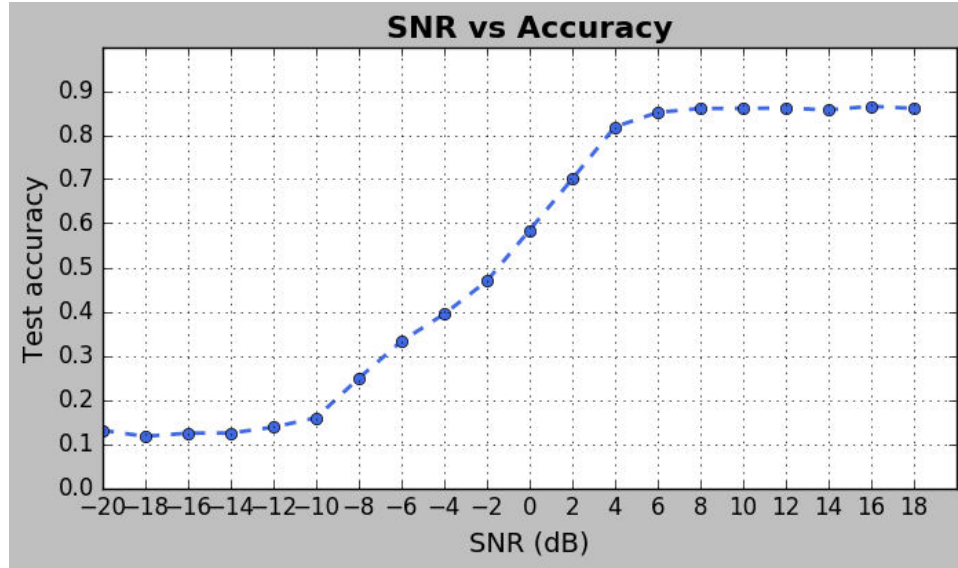


Figure 4.9. Performance of Gradient Boosted ensemble classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

Gradient boosted classifier resulted in a test accuracy of 86.525%.

GradientBoostingClassifier(n_estimators = 500, max_depth = 14, min_samples_split = 4, max_features = 'auto', max_leaf_nodes = 149)

The improvement in accuracy of this classifier, with rise in SNR is depicted in Figure 4.9. Results of grid search from random forest were used to set the hyperparameters and train an ensemble of 500 DTs. Figure 4.10 illustrates the confusion matrix produced by the same classifier. It can be seen that the classifier is still confused between 16-QAM and 64-QAM, but it is able to detect all the other classes with precision of around 90%.

4.2.3.5 SUMMARY

A single decision tree with default hyperparameter settings, grid search and randomized search gave test accuracies of 82%, 83.72% and 83.52% respectively. Randomized and grid search tuned the hyperparameters of RF, which improved the test accuracy to 84.42% in both the cases. The performance of Bagging and Pasting classifiers and Adaboost was found to be similar to that of a single decision tree. With test accuracy of 86.52%, Gradient Boosted ensemble of trees outperformed all the classifiers trained until now.

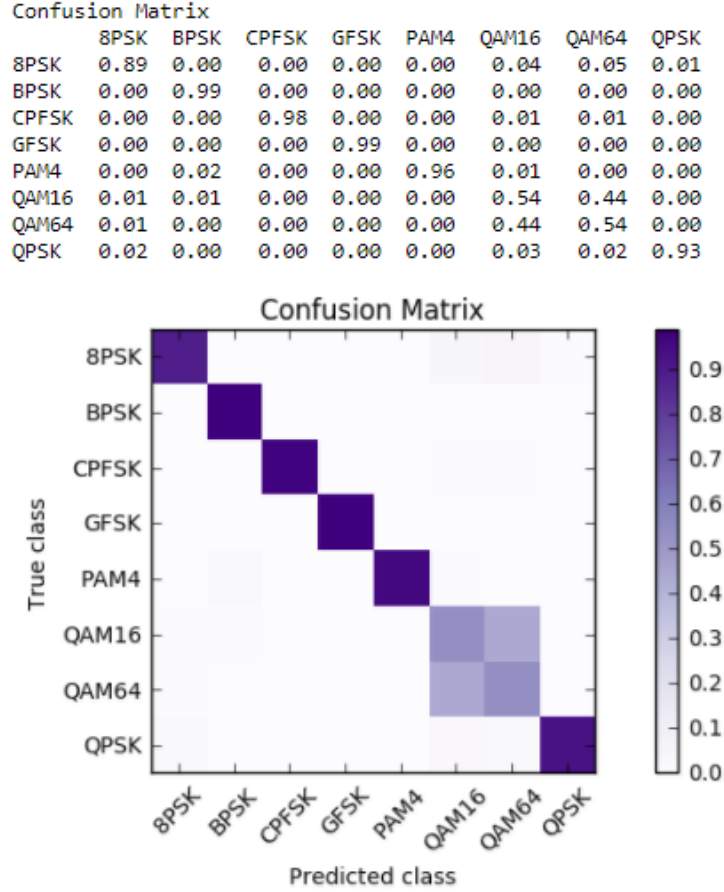


Figure 4.10. Confusion matrix: Precision values of Gradient Boosted classifier are computed and confusion matrix visualized for all modulation classes.

4.2.4 Multi Layer Perceptron

4.2.4.1 HYPERPARAMETERS

The MLP classifier has the following hyperparameters:

- *hidden_layer_sizes*: A tuple containing the number of neurons in each hidden layer.
- *activation*: the activation function for the hidden layer(s); options are 'relu', 'logistic' and 'tanh'.
- *solver*: the optimizer. The solver 'adam' (adaptive moment) is suitable for large datasets - thousands of samples. Other choices are 'sgd' (stochastic gradient descent) and 'lbfgs' that works better for smaller datasets.
- *batch_size*: the minibatch size used by the optimizers.
- *tol*: tolerance for optimization, used for early stopping. The training stops when loss does not improve by this value for two consecutive iterations.

4.2.4.2 EXPERIMENTAL SETUP AND RESULTS

Table 4.7. Multi Layer Perceptron Performance

Hidden Layers and Experimental Setup	Feature Set	Accuracy	Training Time
100 (default parameters)	set 1	80.675%	25.71 seconds
100 (default parameters)	set 2	84.1%	29.02 seconds
16,200 (grid search)	set 1	79.8%	101 minutes
16,300 (grid search)	set 2	82.75%	115 minutes
16,200 (randomized search)	set 1	79.875%	4 minutes
16,200 (randomized search)	set 2	80.55%	4 minutes
16, 256, 128	set 1	81.325%	13.81 seconds
16, 256, 128	set 2	83.05%	14.81 seconds

- Default parameters setting:
We started with default settings for MLPClassifier- just one hidden layer with 100 neurons. A classification accuracy of 80.675% was achieved within 25 seconds. The results were further improved with hyperparameter tuning as follows.
- Grid search: Grid search was performed over hyperparameter space:
`params = {'hidden_layer_sizes': [(16, 100), (16, 200), (16, 300)],`
`'activation': ['logistic', 'tanh', 'relu'], 'batch_size': [64, 128, 256, 512, 1024],`
`'solver': ['sgd', 'adam'], 'learning_rate': ['constant', 'invscaling', 'adaptive']}`
The best estimator was found to be:
`MLPClassifier(activation='relu', alpha=0.0001, batch_size=64, beta_1=`
`0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,`
`hidden_layer_sizes=(16, 300), learning_rate='adaptive', learning_rate_init=`
`0.001, max_iter=200, momentum=0.9, nesterovs_momentum=`
`True, power_t=0.5, random_state=None, shuffle=True, solver=`
`'adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=`
`False)`
resulting in a performance of 82.75% in 115 minutes.
- Randomized search: Randomized search performed over the same hyperparameter space computed the best estimator:
`MLPClassifier(activation='tanh', alpha=0.0001, batch_size=128, beta_1=`
`0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,`
`hidden_layer_sizes=(16, 200), learning_rate='invscaling',`
`learning_rate_init=0.001, max_iter=200, momentum=0.9,`
`nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=`
`True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,`

warm_start = False)

that delivers classification accuracy of 80.55% in 4 minutes.

- Deeper neural network: A deeper neural network was trained:
`MLPClassifier(activation='relu', batch_size=128, early_stopping = True, hidden_layer_sizes=(16,256,128), learning_rate='adaptive', solver='adam')`
 that resulted in classification accuracy of 83.05% on feature set 2, within 30 seconds. The accuracy of this classifier is better than those obtained earlier by grid and randomized search and the training time is lower as well.

The best of all the MLP classifiers was found to be the one with just one hidden layer of 100 neurons. A plot of SNR and accuracy is shown in Figure 4.11. All parameters other than *hidden_layer_sizes* were set to default values. A classification accuracy up to 84.1% is achieved.

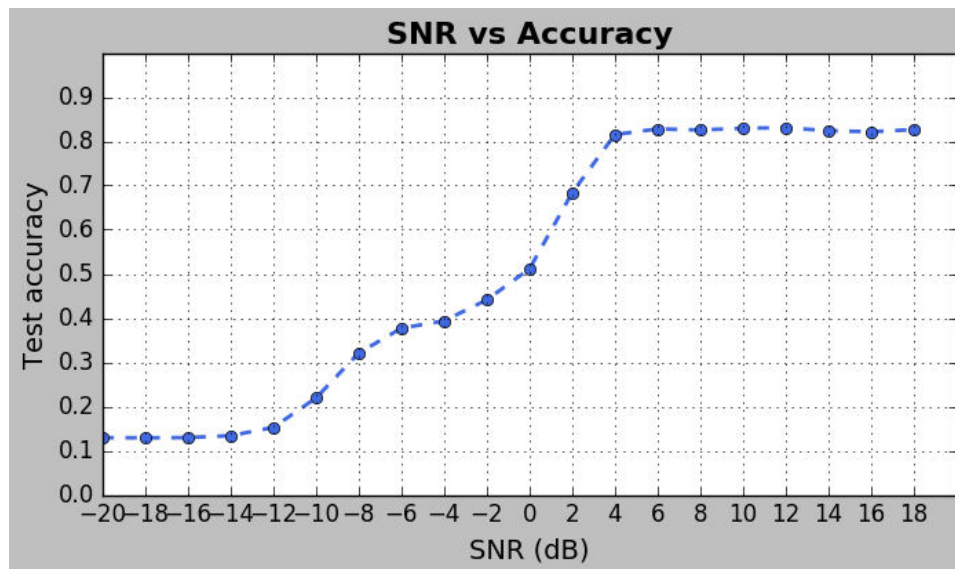


Figure 4.11. Performance of three hidden layer [16,256,128] MLP classifier on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

4.2.4.3 SUMMARY

MLPs of varying depth and width were trained on both the feature sets. MLP with default parameter settings (1-hidden layer) resulted in a better classifier- 84.1%, as compared to the 2-hidden layer MLPs given by grid search and randomized search- 82.75% and 80.55% respectively. A deeper 3-hidden layer MLP outperformed the 2-hidden layer MLPs with test accuracy of 83.05%.

4.2.5 Artificial Neural Networks

TensorFlow- an open-source software library for Machine Intelligence was used to train Artificial Neural Networks and Deep Neural Networks. A variety of ANN architectures were implemented, with and without early stopping. We used TensorFlow's high level API TF Learn and plain TensorFlow as well.

TF Learn allows users to quickly build models with very few settings of hyperparameters.

4.2.5.1 HYPERPARAMETERS

- *hidden_units* is the list of neurons in network's hidden layers. These were modified in each experiment, in order to train ANNs of different architectures.
- The parameter *every_n_steps* is set to 50 for each experiment, meaning that the model will be evaluated on the test set every 50 steps.
- The *early_stopping_metric* was set to *loss*- the metric whose value is monitored while the model trains.
- *early_stopping_rounds* was set to 200, which implies that TensorFlow will stop the training if the loss does not decrease during 200 steps.

Table 4.8. DNN Performance

Hidden Layers	Feature Set	Accuracy	Training Time (minutes)
300,100	set 1	80.375%	2.25
300,100	set 2	83.75%	2.61
640,320	set 1	79.8%	5.01
640,320	set 2	82.75%	4.61
640,320,160	set 1	79.875%	4.42
640,320,160	set 2	80.55%	5.07
640,320,160,80,16	set 1	81.325%	5.83
640,320,160,80,16	set 2	83.05%	5.73

As shown in Table 4.8, the DNN takes between 2 to 6 minutes depending upon the depth and width (number of neurons in hidden layer) of the network. But the training continues for the specified number of steps, which in our case is 40,000. A better performance can be achieved by implementing early stopping.

DNNs of various architectures were then trained by implementing early stopping, as summarized in Tables 4.9 through 4.12. The number of steps specified was 40,000 for each model, but less number of steps were required with early stopping. Regularizing by

early stopping ceases the training when the performance of the ANN does not improve any further. This is accomplished via TensorFlow's validation monitor.

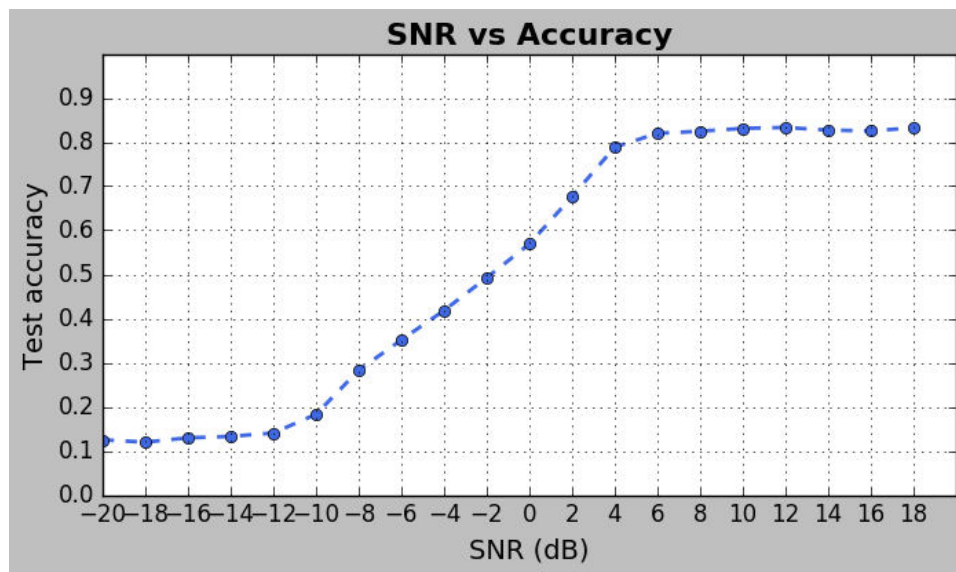


Figure 4.12. Performance of five hidden layer [640,320,160,80,16] DNN on feature set 2: The plot shows that the classification accuracy increases as the SNR rises.

Figure 4.12 shows the performance of a five hidden layer [640,320,160,80,16] DNN that was trained on feature set 2, without the use of validation monitor in TF Learn.

Following is a description of the ANN architectures trained on feature sets 1 and 2. Validation monitor was used for early stopping in each experiment. Different approaches were employed to design the architecture of neural networks.

1. One hidden layer: We started with a single hidden layer and width equal to the number of features and then increased the width of the network to be a multiple of the number of features. As seen in Table 4.9, a wider neural network doesn't necessarily mean a better performing model.

Table 4.9. ANNs - Single Hidden Layer

Hidden Layers	Feature Set	Accuracy	Training Time
16	set 1	77.9%	50.74 seconds
1024	set 1	78.75%	3.95 minutes
4096	set 1	71.92%	4.23 minutes
32	set 2	82.75%	60.83 seconds
1024	set 2	79.875%	3.87 minutes
4096	set 2	80.55%	4.91 minutes

2. Deep and narrow ANN: Next, the number of hidden layers were increased to make the network deeper. The width of all the layers was set equal to the number of features.

Table 4.10. Deep and Narrow ANN

Hidden Layers	Feature Set	Accuracy	Training Time (minutes)
2 layers, 16 neurons per layer	set 1	79.72%	1.74
3 layers, 16 neurons per layer	set 1	80.025%	2.81
4 layers, 16 neurons per layer	set 1	80.925%	3.63
5 layers, 16 neurons per layer	set 1	69.57%	4.73
6 layers, 16 neurons per layer	set 1	69.375%	4.89
2 layers, 32 neurons per layer	set 2	84.325%	2.64
3 layers, 32 neurons per layer	set 2	80.55%	3.53
4 layers, 32 neurons per layer	set 2	80.31%	3.83
5 layers, 32 neurons per layer	set 2	74.83%	5.82
6 layers, 32 neurons per layer	set 2	71.74%	6.73

A deeper ANN with fewer neurons takes less time to train as compared to a model having much greater number of neurons and a single layer (case 1). This is due to that fact the parameter efficiency of deep ANNs is better than that of shallow ones. They are able to model complex functions using much less neurons than shallow ANNs. Some of these deeper and narrow networks perform better when compared to the single layer wider ones (compare Tables 4.9 and 4.10). But much deeper ANNs- five and six layers perform quite poorly as seen in Table 4.10. The model benefits from greater number of features, as the accuracy of [32,32] DNN trained on feature set 2 goes up to 84.325%. But the network starts performing poorly after the fourth hidden layer.

3. The hidden layers were then designed to form a funnel- fewer neurons in the subsequent hidden layers. The number of neurons in the first hidden layer was always set to a multiple of the number of features. The number of neurons in the each of the subsequent hidden layers, were half of those in the previous layer.

Table 4.11. Funnel Shape for Hidden Layers

Hidden Layers	Feature Set	Accuracy	Training Time (minutes)
256,128	set 1	74.525%	2.82
256,128,64	set 1	72.2%	2.24
256,128,64,32	set 1	70.05%	4.72
256,128	set 2	79.725%	3.31
256,128,64	set 2	76.025%	4.26

4. Another approach utilized was to maintain the width of the network, while increasing the depth.

Table 4.12. Same Sized Hidden Layers

Hidden Layers	Feature Set	Accuracy	Training Time (minutes)
256,256	set 1	78.10%	2.16
256,256,256	set 1	70.00%	2.26
256,256,256,256	set 1	71.95%	2.94
128,128	set 1	78.475%	1.54
128,128,128	set 1	74.6	3.11
64,64	set 2	82.625	0.64
128,128	set 2	81.975	3.46
256,256	set 2	76.725	3.51

A bigger feature set of 32 features enhances the model's performance. But it declines when the network is made wider in case of both feature sets 1 and 2.

Hence, the results reveal that a bigger neural network does not necessarily mean a better classifier. A simpler network might surpass the wider and/or deeper ones. A common approach for training ANNs is to start with a very simple one and gradually increase the size until it starts overfitting. The same was done in this research- we started with an ANN of just one hidden layer [16] and gradually expanded the network to more complicated layers. A DNN ultimately starts overfitting and the performance degrades as seen in Tables 4.8 through 4.12.

The best architecture for the problem of modulation classification was found to be intermediate between these two extremes i.e., a DNN with [32,32] hidden layers that gave classification accuracy of 84.325% on feature set 2.

Neural networks implemented in plain TensorFlow offer greater flexibility in terms of several parameters such as weight initializer, activation function and performance measure. ANNs with various settings were implemented and evaluated. Early stopping was used in each case. In Table 4.13, ANN of two hidden layers of 16 neurons each was trained on feature set 1 and ANN consisting of two hidden layers of 32 neurons each was trained on feature set 2.

ReLU activation function is the most commonly used activation function in neural networks. It is faster when compared to other activation functions and it does not suffer from the problem of saturation for large input values. But ReLU neurons may die during

training, which is solved by leaky ReLU- a variant of ReLU. ELU activation may further reduce the training time and give a better performance as well.

Also, Xavier and He initialization were experimented with. Xavier initialization of weights randomly initializes the weight values. It overcomes the problem of vanishing and exploding gradients and makes the training of neural networks significantly faster.

Table 4.13. DNN in Plain TensorFlow- Feature Set 1

Activation	Initializer	Accuracy	Training Time (minutes)
ReLU	Xavier	82.8%	2.70
ReLU	He	82.85%	3.21
Leaky ReLU	Xavier	83.9%	3.21
Leaky ReLU	He	83.15%	3.16
ELU	Xavier	83.75%	3.01
ELU	He	83.05%	3.10

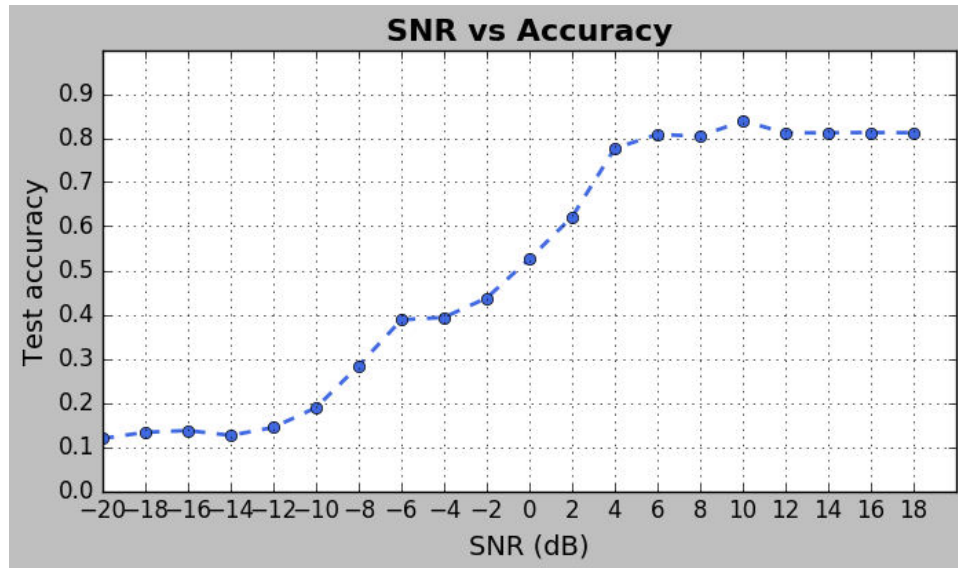


Figure 4.13. Performance of [16,16] hidden layer DNN on feature set 1: The DNN uses Leaky ReLU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.

Figure 4.13 shows the performance of the best classifier from amongst the ones shown in Table 4.13. The DNN consisting of [16,16] hidden layers and using Leaky ReLU and Xavier achieves classification accuracy up to 83.75%.

Next, DNNs with hidden layers [32,32], different activations and weight initializers were trained on feature set 2. The training time was found to be approximately the same for all of the cases shown in Table 4.14.

Table 4.14. DNN in Plain TensorFlow- Feature Set 2

Activation Function	Weight Initializer	Accuracy	Training Time (minutes)
ReLU	Xavier	85.05%	2.94
ReLU	He	84.325%	2.26
Leaky ReLU	Xavier	84.825%	2.52
Leaky ReLU	He	84.45%	1.55
ELU	Xavier	85.2%	2.35
ELU	He	84.4%	2.26

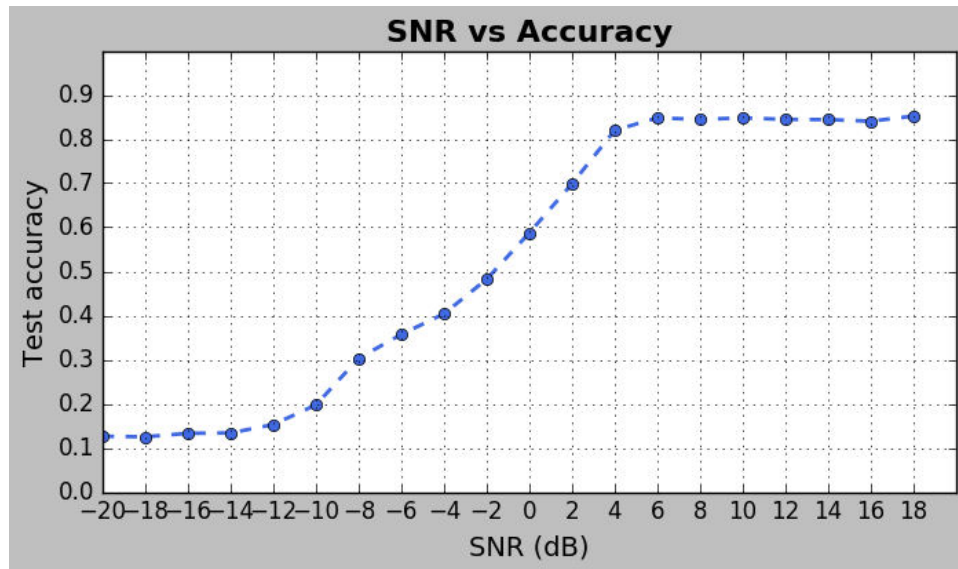


Figure 4.14. Performance of [32,32] hidden layer DNN on feature set 2: The DNN uses ELU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.

Expanding the feature set results in a better classifier, as seen in the previous machine learning models as well. Table 4.14 shows that the DNN using ELU activation function and Xavier initialization is found to be the best (85.2%) amongst all the DNNs tested so far, using TF Learn and plain TensorFlow. The classifier's performance is shown in Figure 4.14.

DNNs can be made better at performing tasks by regularizing them. Dropout is a popular regularization technique aimed at preventing overfitting in DNNs. A dropout layer was applied after each hidden layer in Table 4.15. All the DNNs were trained on feature set 2. The dropout rate specifies what percentage of neurons are dropped out from the layer to which dropout is applied. As in all the previous DNNs implemented in plain TensorFlow, early stopping has been used in the following cases as well. Table 4.15 summarizes the performance of DNNs of various architectures.

Table 4.15. DNN using Dropout- Feature Set 2

Hidden Layers	Dropout Rate	Accuracy	Training Time (minutes)
256,256	0.2	83.75%	3.11
256,256	0.3	83.9%	4.27
256,256	0.4	84.54%	4.45
246,256	0.5	84.325%	5.65
1024,1024	0.2	83.525%	6.33
1024,1024	0.3	83.675%	6.73
1024,1024	0.4	83.87%	7.35
1024,1024	0.5	83.45%	8.59

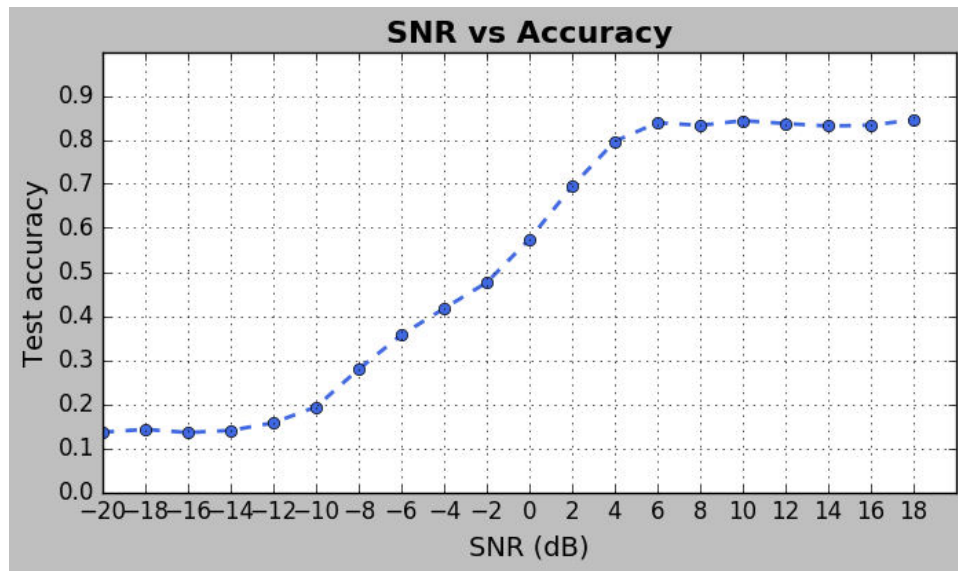


Figure 4.15. Performance of regularized [256,256] hidden layer DNN on feature set 2: The DNN uses ReLU as activation function and Xavier for weight initialization. The plot shows that the classification accuracy increases as the SNR rises.

Table 4.15 shows that we start by dropping out a small fraction (20%) of neurons from each of the hidden layers and progress towards dropping a greater number of neurons. The network's performance is improved as the dropout rate rises. The DNN starts underfitting at a dropout rate of 50%. The performance achieved by the regularized DNNs was found to be better when compared to the [256,256] hidden layer DNN that achieved accuracy of only 76.725% as shown in Table 4.12. Figure 4.15 shows the performance of a [256,256] hidden layer DNN regularized with dropout of 50%. Accuracy up to 84.54% is achieved on the test set.

But the true potential of regularization can be seen by applying it to deep networks, rather than shallow ones. Next, a very deep network [256,256,256,256] was regularized by using dropout, *l1_regularization*, *l2_regularization* and max-norm regularization techniques. A comparison of the results obtained is presented in Table 4.16. Each of the models use ELU as activation function and Xavier as weight initializer.

Table 4.16. Regularization Applied to DNNs

Regularization Technique	Accuracy	Training Time (minutes)
25% dropout	84.7%	11.71
<i>l1_regularization</i>	84.50%	2.01
<i>l2_regularization</i>	84.25%	1.73
Max-norm regularization	83.80%	2.08

Figure 4.16 depicts the confusion matrix for the first classifier in Table 4.16. The elements along the diagonal are the precision values for the respective modulation classes indicated by the rows.

4.2.5.2 SUMMARY

ANNs of various depths and widths were trained on both the feature sets. Narrow and 1-hidden layer ANNs were often found to surpass wider 1-hidden layer ANNs. DNNs with varying width of hidden layers did not perform as good as narrow DNNs with the width set to the number of features, the test accuracies being 79.72% and 84.325% respectively. Different activation functions- namely ReLU, Leaky ReLU and ELU as well as weight initializers Xavier and He were investigated. The combination of Leaky ReLU and Xavier resulted in the best performance (83.9%) in case of feature set 1, while ELU and Xavier was found to be the best choice (85.2%) in case of feature set 2.

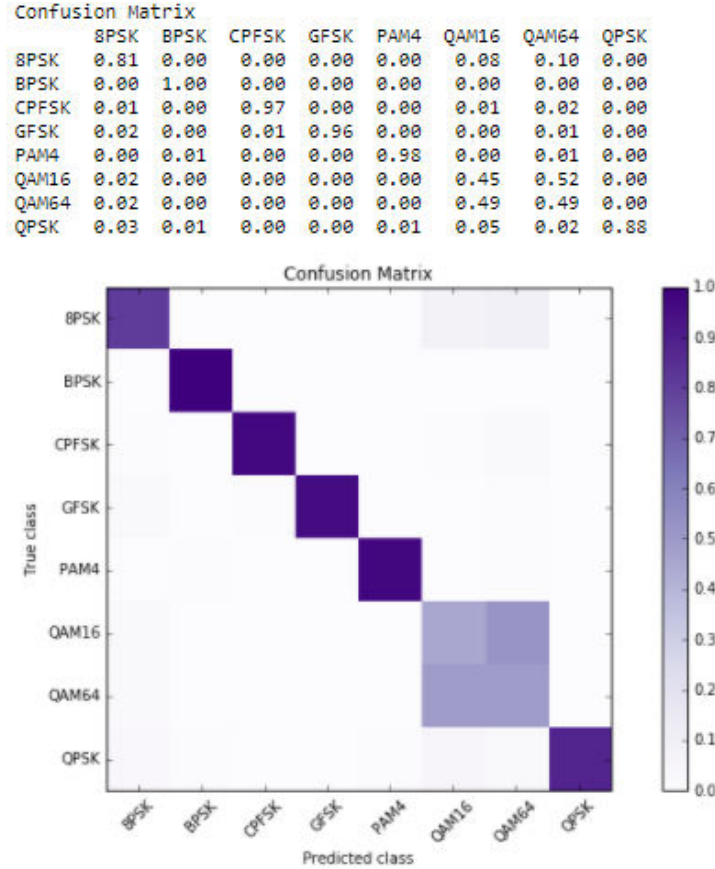


Figure 4.16. Confusion matrix: Precision values of DNN are computed and confusion matrix visualized for all modulation classes.

4.2.6 Convolutional Neural Networks

Convolutional Neural Networks were trained on raw time series data, rather than the designed features. The classifiers were trained on 80,000 and tested on 80,000 samples. The data is first standardized to bring all the features on the same scale. CNN explicitly assumes that the input provided to it is an image. Each sample is a (2, 128) array. So the input is treated as an image of height equal to 2 and width of 128.

A typical CNN architecture consists of one or more convolutional layers, followed by pooling and fully connected layers. Pooling is done to shrink or subsample the input image to reduce the number of parameters, so that the computational load and memory usage are reduced. Limiting the number of parameters avoids overfitting as well.

We first trained CNNs consisting of a single convolutional layer, followed by pooling and a fully connected layer. The number of feature maps in convolutional and pooling layer and the number of neurons in fully connected layer were all set to a multiple of the number of features (256). The results are summarized in Table 4.17.

Table 4.17. Single Convolutional and Dense Layer CNNs

Convolutional Layers	Fully Connected Layers	Accuracy	Training Time (minutes)
256	256	64.37%	13.86
512	512	72.8%	30.27
1024	1024	76.84%	73.65

The CNN architecture was then expanded to include more than one convolutional layer and fully connected layer. A pooling layer is added after the last convolutional layer. Adding another convolutional layer in case of the second CNN in Table 4.18 showed

Table 4.18. CNNs- Varying Width

Convolutional Layers	Fully Connected Layers	Accuracy	Training Time (minutes)
512,256	128	71.74%	92.47
256,128	256	79.65%	43.33
256,128	128	78.64%	37.40
256,64	64	76.99%	25.07

significant gain in accuracy, when compared to the CNNs in Tables 4.17 and 4.22. But a wider CNN consisting of the same number of layers was found to perform poorly when compared to all three of the other CNNs that are narrow. This leads to the conclusion that deeper networks might be a better choice than narrower ones. The last three CNNs outperform the first wider CNN in terms of training duration as well. The performance then degrades as we decrease the width of layers.

Table 4.19 shows that adding more convolutional layers significantly improves the classifier's performance. We started with a single convolutional layer that resulted in an

Table 4.19. CNNs- Uniform Width

Convolutional Layers	Fully Connected Layers	Accuracy	Training Time (minutes)
256	256	64.37%	13.86
512,512	512	72.54%	157.15
256,256	256	80.17%	62.26
256,256,256	256	82.72%	91.52
256,256,256	256,256	83.07%	101.08
256,256,256	256,256,256	84.07%	107.87

accuracy of just 64%. The model was improved by adding more convolutional layers,

while keeping the number of fully connected layers constant. The best among the first three CNNs was found to be the one with three convolutional layers. Hence, more fully connected layers were added to this architecture and the performance was further improved to 84.07%.

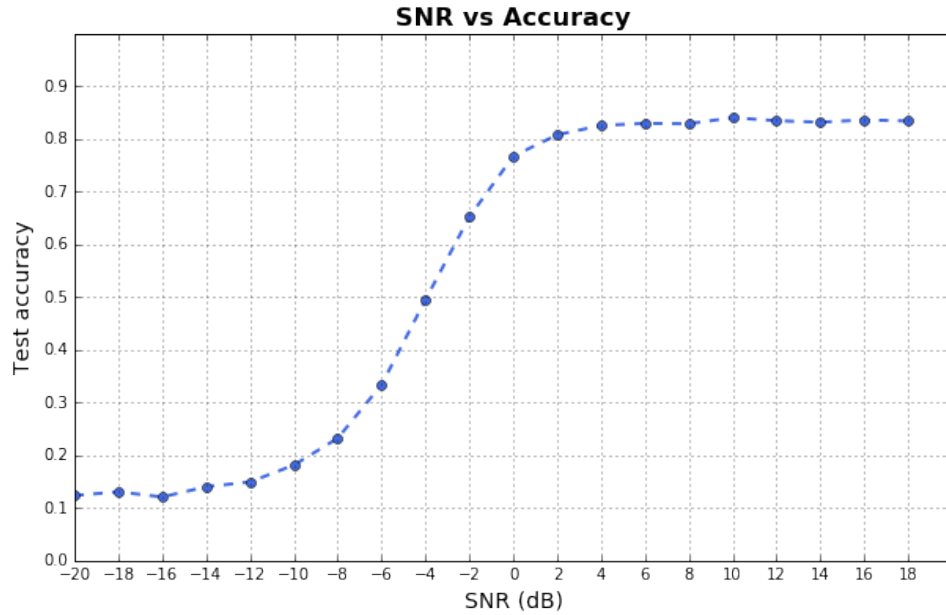


Figure 4.17. Convolutional Neural Network with [256,256,256] convolutional layers and [256,256,256] fully connected layers: The plot shows that the classification accuracy increases as the SNR rises.

Figure 4.17 illustrates the performance of the last CNN in Table 4.19. Figure 4.18 depicts the precision values in the confusion matrix, as well as a visualization of the matrix.

Narrow and deep CNNs were experimented with as well, as shown in Table 4.20. Narrow CNNs with 128 neurons in each of the layers were implemented. Similar to Table

Table 4.20. Narrow and Deep CNNs

Convolutional Layers	Fully Connected Layers	Accuracy	Training Time (minutes)
128	128	55.87%	6.47
128,128	128	77.54%	20.67
128,128,128	128	80.92%	35.13
128,128	128,128	81.69%	28.53
128,128,128	128,128	82.4%	41.50
128,128,128	128,128,128	82.17%	39.43

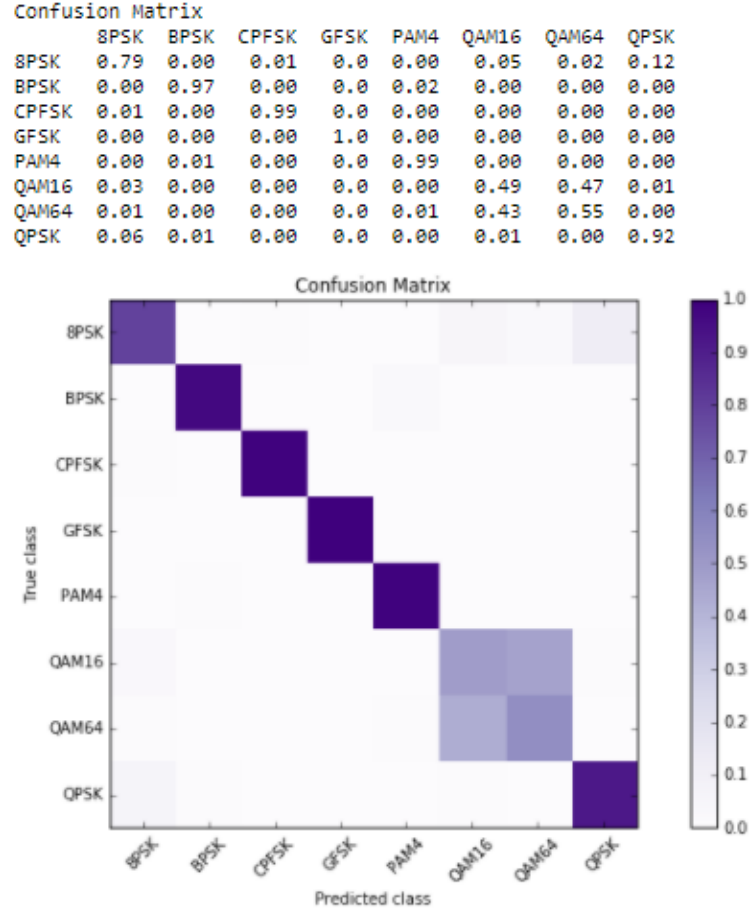


Figure 4.18. Confusion matrix: Precision values of CNN are computed and confusion matrix visualized for all modulation classes.

4.19, the effect of adding more convolutional layers was observed in Table 4.20. It was found that the classifier improves as more convolutional layers are added, and further improves as more fully connected layers are added. Narrow and deep CNNs in Table 4.20 outperform the wider and shallower CNNs shown in Table 4.17 by a large margin.

ReLU activation and Xavier initialization have been used for all of the above CNNs. Other activation functions, namely Leaky ReLU and ELU were investigated along with He weight initializer. Table 4.21 summarizes the performance of a [128,128] convolutional layer and [128,128] dense layer CNN with various combinations of activations and weight initializers.

Similar to DNNs in the previous section, various regularization techniques were applied to deep CNNs as well. First, the effect of dropout on shallow and wide networks was studied. A dropout layer is applied after convolutional, as well as dense layers. Table 4.22 shows the effect of varying dropout rates on two CNN architectures. Such

Table 4.21. Effect of Various Activation Functions and Weight Initializers

Activation Function	Weight Initializer	Accuracy	Training Time (minutes)
ReLU	Xavier	81.69%	28.53
ReLU	He	81.82%	29.39
Leaky ReLU	Xavier	81.65%	30.94
Leaky ReLU	He	69.37%	30.48
ELU	Xavier	74.50%	20.80
ELU	He	78.324%	20.81

Table 4.22. Dropout Applied to Single Convolutional and Dense Layer CNNs

Convolutional Layers	Dropout Rate	Fully Connected Layers	Accuracy	Training Time (minutes)
256	0.2	256	60.87%	21.63
256	0.3	256	62.25%	22.17
256	0.4	256	58.95%	21.17
512	0.2	512	72.32%	45.14
512	0.3	512	72.42%	43.06
512	0.4	512	66.90%	44.41

regularization did not show any improvements when compared to the results in Table 4.17. The performance of all the [256,256] CNNs is worse than the [256,256] CNNs shown in 4.17. A similar observation can be made for the cases of [512,512] CNN. This is because regularization works well on deeper, rather than shallow and wide neural networks.

Various regularization techniques were then applied to deeper CNNs. As seen in Table 4.20, the performance of CNN improves with the addition of layers, until we arrive at the last CNN consisting of six layers in total. The underfitting seen might be due to the presence of a large number of parameters. Hence, the need for regularization arises. CNN consisting of convolutional layers [128,128,128] and dense layers [128,128,128] was regularized and the results are summarized in Table 4.23. In the first CNN, a dropout of

Table 4.23. Regularization Applied to Deep CNNs

Regularization Technique	Accuracy	Training Time (minutes)
Dropout	81.15%	37.63
l_1 regularization	43.07%	37.41
l_2 regularization	81.90%	38.13
Max-norm regularization	82.89%	38.89

25% is applied after the third convolutional layer and dropout of 50% is applied after the last dense layer. The second and third CNN's dense layers were regularized with l_1 and l_2 regularization respectively, where l_1 regularized CNN performed poorly, while l_2 regularization was found to be even better than dropout. Max-norm regularization was found to be the best of all. It was the only regularization technique that could improve the performance of unregularized CNN- the last CNN presented in Table 4.20, that gave an accuracy value of 82.17%.

4.2.6.1 SUMMARY

We started with CNNs comprising of single convolutional and dense layers. Adding one more convolutional layers significantly boosted the accuracy, raising it from 64.7% to 80.17%. The performance could be improved further by the addition of each convolutional layer, while the number of dense layers was set to one in each case. The addition of dense layers too produced increasingly better classifiers. Narrow CNNs were studied as well- deep and narrow CNNs (width of 128) resulted in better accuracies than their wider counterparts (width of 256) of the same depth. Regularization was investigated to make the models more efficient.

4.3 SUMMARY

This chapter presented the results of all the experiments performed. It was organized into two parts, namely feature extraction and classification. The first part presented a detailed description of the features engineered to transform raw data into features that are useful for the task of AMC. The second part presented all the results of each classifier's implementation in a tabular form. The performance of the best model was visualized for each classifier- KNN, SVM, DT, RF, AdaBoost, Gradient Boosted trees, ANN and DNN. An alternative to feature design was presented in the form of CNN that was trained on raw time series, rather than the feature sets.

4.4 CONCLUSION AND FUTURE WORK

This thesis studied several machine learning classifiers and applied them to the task of AMC. Some classifiers were trained on features engineered by using cyclic-moments, while some were trained on raw data. Classifiers k-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forests, Adaboost, Gradient Boosted trees and Artificial Neural Networks were trained on a total of 32 features. For each of these models, hyperparameter tuning was performed in order to select the optimal parameters for the algorithm. The classifiers were evaluated and compared on the basis of their accuracy and precision values on the test set.

Deep Neural Networks were thoroughly studied and trained on the designed features. An attempt to improve the learning algorithm was made, by varying the model architecture and by techniques such as early stopping and regularization. It was observed that networks with simple architecture frequently outperformed those with relatively complex architecture. The effect of increasing the depth and width of the network was particularly studied. It was demonstrated that deeper neural networks often turned out to be better learners as compared to shallower and/or wider networks.

Convolutional Neural Networks were trained on raw data. The models were made to learn from raw waveforms, rather than carefully designed features. It was shown that a CNN that learns features from raw data might be a stronger candidate when compared to the models that learn from expert features. CNNs were found to be better at the task of modulation classification of digital signals, when compared to KNN, SVM, and Decision Tree, as shown in Figure 4.19. The plot shows, that at very low SNRs- ranging from -4dB

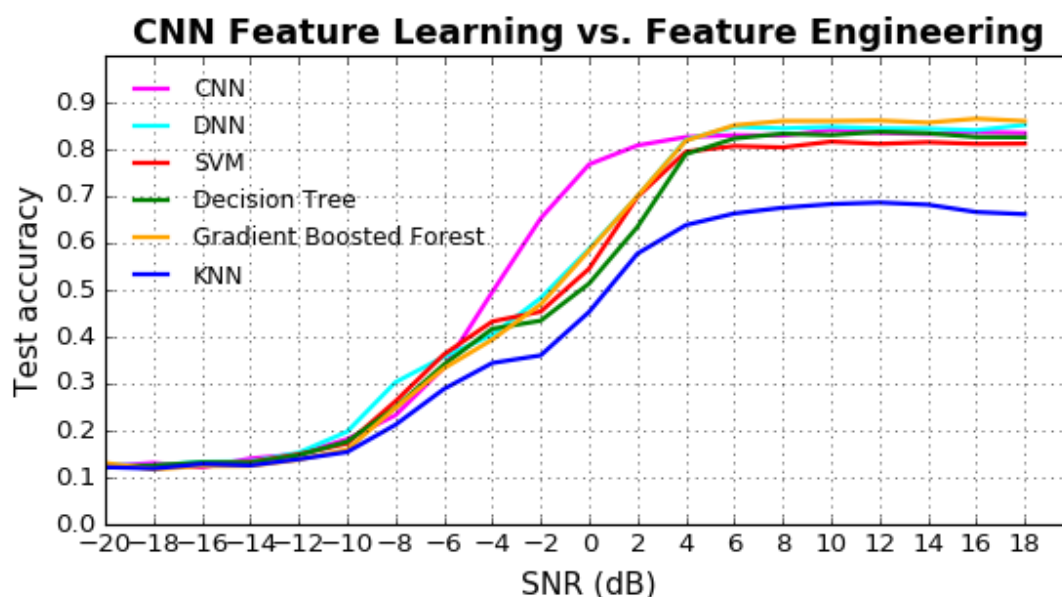


Figure 4.19. Classifier comparison: Comparison of feature learning vs. feature engineering approach. CNN is based on feature learning, while rest of the classifiers are based on feature engineering.

to 4dB, CNN outperformed all the other classifiers. It should be noted that CNN was trained on raw time series data, while all the other classifiers were trained on the expert features. This implies that CNNs have the ability to recognize patterns in raw signals, without understanding the signals' underlying waveform. Moreover, this observation leads to the conclusion that classifiers such as CNNs trained on radio waveforms might be

better at the task of modulation recognition, when compared to other machine learning classifiers based on expert features. Hence, the time and efforts spent on carefully designing features can instead be spent on constructing models such as CNN that can learn from the raw data itself.

Efficient machine learning models such as CNN can therefore eliminate the need for feature engineering. The idea is that humans may not have knowledge of what features are best suited to a problem. A valuable alternative to handcrafted features is to enable the model to create salient features of its own, from raw data. This highlights the potential of CNNs as they can be made independent of human effort and expert knowledge of feature design.

This research has examined a feature based vs. feature learning approach to tackle the problem of AMC of digital signals. Potential lies in further research that could investigate other kinds of neural networks such as Recurrent Neural Network (RNN) for modulation recognition. RNNs are well suited to sequential data, but the time-series data used in this thesis is non-sequential. RNNs are widely used for speech recognition and machine translation where the data is sequential in nature. But they have been successfully applied to classification tasks such as the classification of handwritten digits. It will be interesting to explore more complex architectures such as deeper CNNs and deep RNNs as well. Combining CNNs and RNNs to create more complex models could be a worthwhile research direction.

REFERENCES

- [1] A. K. Nandi and E. E. Azzouz, "Algorithms for automatic modulation recognition of communication signals," *IEEE Trans. Commun.*, vol. 46, no. 4, pp. 431–436, Apr. 1998.
- [2] O. A. Dobre, A. Abdi, Y. Bar-Ness, and W. Su, "Survey of automatic modulation classification techniques: classical approaches and new trends," *J. IET Commun.*, vol. 1, no. 2, pp. 137–156, Apr. 2007.
- [3] A. Hazza, M. Shoaib, S. A. Alshebeili, and A. Fahad, "An overview of feature-based methods for digital modulation classification," in *Proc. Conf. Commun. Signal Process. Appl.*, 2013, pp. 1–6.
- [4] Z. Zhu, "Automatic classification of digital communication signal modulations," Ph.D. dissertation, Dept. Elect. Eng., Brunel Univ. London, Uxbridge, United Kingdom, 2014.
- [5] National Telecommunications and Information Administration, "United States frequency allocations chart," 2016. [Online]. Available: <https://www.ntia.doc.gov>
- [6] A. Shaha, D. H. N. Nguyen, and S. Kumar, "Implementing Directional Tx-Rx of High Modulation QAM Signaling with SDR Testbed," in *Proc. IEEE Ubiquitous Comput. Elect. Mobile Commun. Conf.*, 2017.
- [7] E. E. Azzouz and A. K. Nandi, *Automatic Modulation Recognition of Communication Signals*. Norwell, MA, USA: Kluwer Academic, 1996.
- [8] A. K. Nandi and E. E. Azzouz, "Automatic identification of digital modulation types," *J. Signal Process.*, vol. 47, no. 1, pp. 55–69, Nov. 1995.
- [9] J. A. Sills, "Maximum-likelihood modulation classification for PSK/QAM," in *Proc. IEEE Military Commun. Conf.*, 1999, pp. 217–220.
- [10] A. Hazza, M. Shoaib, A. Saleh, and A. Fahd, "Classification of digitally modulated signals in presence of non-Gaussian HF noise," in *Proc. Int. Symp. Wireless Commun. Syst.*, 2010, pp. 815–819.
- [11] W. Wei and J. M. Mendel, "Maximum-likelihood classification for digital amplitude-phase modulations," *IEEE Trans. Commun.*, vol. 48, no. 2, pp. 189–193, Feb. 2000.

- [12] K. Kim and A. Polydoros, "Digital modulation classification: the BPSK versus QPSK case," in *Proc. IEEE Military Commun. Conf.*, 1988, pp. 431–436.
- [13] A. Polydoros and K. Kim, "On the detection and classification of quadrature digital modulations in broad-band noise," *IEEE Trans. Commun.*, vol. 38, no. 8, pp. 1199–1211, Aug. 1990.
- [14] A. O. A. Salam, R. E. Sheriff, S. R. Al-Araji, K. Mezher, and Q. Nasir, "A unified practical approach to modulation classification in cognitive radio using likelihood-based techniques," in *Proc. Can. Conf. Elect. Comput. Eng.*, 2015, pp. 1024–1029.
- [15] P. Panagiotou, A. Anastasopoulos, and A. Polydoros, "Likelihood ratio tests for modulation classification," in *Proc. IEEE Military Commun. Conf.*, 2000, pp. 670–674.
- [16] L. Hong and K. C. Ho, "Modulation classification of BPSK and QPSK signals using a two element antenna array receiver," in *Proc. IEEE Military Commun. Conf.*, 2001, pp. 118–122.
- [17] K. C. Ho and L. Hong, "Antenna array likelihood modulation classifier for BPSK and QPSK signals," in *Proc. IEEE Military Commun. Conf.*, 2002, pp. 647–651.
- [18] L. Hong and K. C. Ho, "BPSK and QPSK modulation classification with unknown signal level," in *Proc. IEEE Military Commun. Conf.*, 2000, pp. 976–980.
- [19] Y. Yang and S. S. Soliman, "Optimum classifier for M-ary PSK signals," in *Proc. Int. Commun. Conf.*, 1991, pp. 1693–1697.
- [20] S. S. Soliman and Y. Yang, "Statistical moments based classifier for MPSK signals," in *Proc. Global Telecommun. Conf.*, 1991, pp. 72–76.
- [21] A. Swami and B. M. Sadler, "Hierarchical digital modulation classification using cumulants," *IEEE Trans. Commun.*, vol. 48, no. 3, pp. 416–429, Mar. 2000.
- [22] W. Dai, Y. Wang, and J. Wang, "Joint power estimation and modulation classification using second- and higher statistics," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2002, pp. 155–158.
- [23] G. Hatzichristos and M. P. Fargues, "A hierarchical approach to the classification of digital modulation types in multipath environments," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2001, pp. 1494–1498.

- [24] A. Swami, S. Barbarossa, and B. M. Sadler, "Blind source separation and signal classification," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2000, pp. 1187–1191.
- [25] J. J. Popoola and R. V. Olst, "A novel modulation-sensing method," *IEEE Veh. Technol. Mag.*, vol. 6, no. 3, pp. 60–69, Sep. 2011.
- [26] Y. Liu, G. Liang, X. Xu, and X. Li, "The methods of recognition for common used M-ary digital modulations," in *Proc. Int. Conf. Wireless Commun. Netw. Mobile Comput.*, 2008, pp. 1–4.
- [27] S. Gulati and R. Bhattacharjee, "Automatic blind recognition of noisy and faded digitally modulated MQAM signals," in *Proc. Annu. IEEE India Conf.*, 2006, pp. 1–6.
- [28] L. D. Vito, S. Rapuano, and M. Villanacci, "Prototype of an automatic digital modulation classifier embedded in a real-time spectrum analyzer," *IEEE Trans. Instrum.*, vol. 59, no. 10, pp. 2639–2651, Oct. 2010.
- [29] J. Yuan, Z. Zhao-Yang, and Q. Pei-Liang, "Modulation classification of communication signals," in *Proc. IEEE Military Commun. Conf.*, 2004, pp. 1470–1476.
- [30] K. Maliatsos, S. Vassaki, and P. Constantinou, "Interclass and intraclass modulation recognition using the wavelet transform," in *Proc. IEEE Int. Symp. Pers. Ind. Mobile Radio Commun.*, 2007, pp. 1–5.
- [31] P. Liu, X. J. Ji, and G. Q. Zhao, "Asynchronous classification of digital modulations in multipath flat-fading channel using fourth-order cumulant," in *Proc. Int. Conf. Signal Process.*, 2016, pp. 1435–1439.
- [32] L. D. Vito and S. Rapuano, "Validating a method for classifying digitally modulated signals," *J. Meas.*, vol. 42, no. 3, pp. 427–435, 2009.
- [33] A. Abdi, O. A. Dobre, R. Choudhry, Y. Bar-Ness, and W. Su, "Modulation classification in fading channels using antenna arrays," in *Proc. IEEE Military Commun. Conf.*, 2004, pp. 211–217.
- [34] B. F. Beidas and C. L. Weber, "Higher-order correlation-based approach to modulation classification of digitally frequency-modulated signals," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 1, pp. 89–101, Jan. 1995.

- [35] C. L. Weber and B. F. Beidas, "Asynchronous classification of MFSK signals using the higher order correlation domain," *IEEE Trans. Commun.*, vol. 46, no. 4, pp. 480–493, Apr. 1998.
- [36] B. Shim and I. Kang, "Joint modulation classification and detection using sphere decoding," *IEEE Signal Process. Lett.*, vol. 16, no. 9, pp. 778–781, Sep. 2009.
- [37] A. A. Tadaion, M. Derakhtian, S. Gazor, and M. R. Aref, "Likelihood ratio tests for PSK modulation classification in unknown noise environment," in *Proc. Can. Conf. Elect. Comput. Eng.*, 2005, pp. 151–154.
- [38] J. Leinonen and M. Juntti, "Modulation classification in adaptive OFDM systems," in *Proc. IEEE Veh. Technol. Conf.*, 2004, pp. 1554–1558.
- [39] K. M. Chugg, C.-S. Long, and A. Polydoros, "Combined likelihood power estimation and multiple hypothesis modulation classification," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 1995, pp. 1137–1141.
- [40] M. Derakhtian, A. A. Tadaion, and S. Gazor, "Modulation classification of linearly modulated signals in slow flat fading channels," *IET Signal Process.*, vol. 5, no. 5, pp. 443–450, Aug. 2011.
- [41] A. Abdi, O. A. Dobre, R. Choudhry, Y. Bar-Ness, and W. Su, "Modulation classification in fading channels using antenna arrays," in *Proc. IEEE Military Commun. Conf.*, 2004, pp. 211–217.
- [42] E. E. Azzouz and A. K. Nandi, "Procedure for automatic recognition of analogue and digital modulations," in *Proc. Inst. Elect. Eng. Commun.*, 1996, pp. 259–266.
- [43] K. Farrell and R. Mammone, "Modulation classification using a neural tree network," in *Proc. IEEE Military Commun. Conf.*, 1993, pp. 1028–1032.
- [44] M. L. D. Wong and A. K. Nandi, "Automatic digital modulation recognition using spectral and statistical features with multi-layer perceptrons," in *Proc. Int. Symp. Signal Process. Appl.*, 2001, pp. 390–393.
- [45] K. C. Ho, W. Prokopiw, and Y. T. Chan, "Modulation identification by the wavelet transform," in *Proc. IEEE Military Commun. Conf.*, 1995, pp. 886–890.
- [46] J. Liu and Q. Luo, "A novel modulation classification algorithm based on wavelet and fractional fourier transform in cognitive radio," in *Proc. IEEE Int. Conf. Commun. Technol.*, 2012, pp. 115–120.

- [47] B. G. Mobasseri, “Constellation shape as a robust signature for digital modulation recognition,” in *Proc. IEEE Military Commun. Conf.*, 1999, pp. 442–446.
- [48] S. Z. Hsue and S. S. Soliman, “Automatic modulation classification using zero crossing,” in *Proc. Inst. Elect. Eng. Radar Signal Process.*, 1990, pp. 459–464.
- [49] A. Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 1st ed. Sebastopol, CA: O’Reilly, 2017.
- [50] S. Raschka, *Python Machine Learning*, 1st ed. Birmingham, United Kingdom: Packt, 2015.
- [51] T. J. O’Shea and N. West, “Radio machine learning dataset generation with GNU radio,” in *Proc. 6th GNU Radio Conf.*, 2016, pp. 1–6.
- [52] F. Pedregosa, “Scikit-Learn: Machine learning in Python,” *J. of Mach. Learning Res.*, vol. 12, pp. 2825–2830, 2011.
- [53] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
- [54] A. Karpathy, “CS231n convolutional neural networks for visual recognition,” 2017. [Online]. Available: <https://cs231n.github.io/convolutional-networks>
- [55] T. J. O’Shea, J. Corgan, and T. C. Clancy, “Convolutional radio modulation recognition networks,” 2016. [Online]. Available: <https://arxiv.org/pdf/1602.04105.pdf>