WIRELESS SIGNAL CLASSIFICATION VIA MACHINE LEARNING

Jacob Ramey, Paras Goda, Curtis Zgoda, Anas Qutah

ECE 5424 Advanced Machine Learning

Dec 13th, 2024

1. ABSTRACT

This study aims to classify radio signals based on their time series values or features. We will compare the performance of different models to determine the most effective approach. We use publicly available datasets from DeepSig and extract key features, which will be discussed later. The goal is to classify the modulation type of signals and compare the performance of several models to existing classifiers, such as those proposed by (Flowers & Headly, 2024) and Roy (2020). We will evaluate the performance of Support Vector Machines (SVM), Decision Trees (DTrees), Recurrent Neural Networks (RNNs), and Gradient-Boosted Trees (XGBoost). By comparing these models, we aim to find the most effective approach for modulation classification in wireless signals.

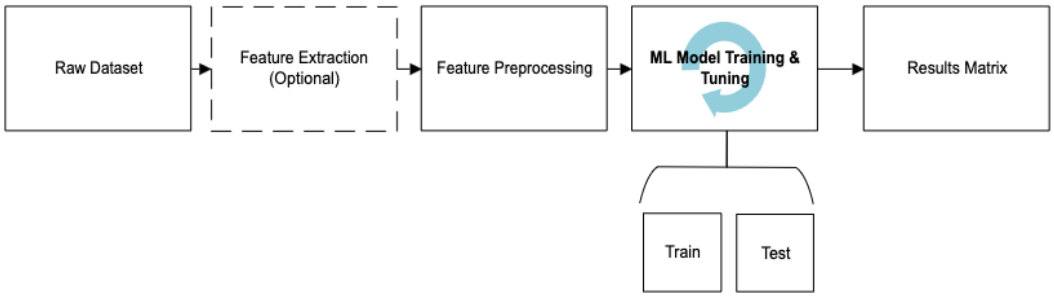


Figure 1: ML Model Pipeline

2. BACKGROUND AND MOTIVATION

Digital wireless signals can be uniquely identified by their modulation scheme. Labeled I/Q Data will be used to classify a signal based on a set of features (i.e., frequency, bandwidth, power, etc.) using supervised machine-learning techniques. Different models will be used and evaluated on a zero-one loss error rate. Data stored in I/Q format can be split, trained with, and then tested to validate machine learning models. The raw I/Q data may be directly inputted into a neural network. However, some other models, like SVM and decision trees, require engineered features to be created. The RadioML data from DeepSig is provided with eleven modulation classifications, including FM, AM x 2, and 8 digitally modulated signals (CPFSK, GFSK, BPSK, QPSK, 8PSK, PAM4, QAM16, QAM64).

In the future, we can expand this classification using unsupervised models to identify new or unknown signals, as the industry is rapidly growing, and to create a future-proof algorithm. Clustering or using deep learning could show some advantages here. Another future improvement is to collect live over-the-air (OTA) data and feed it through the trained algorithm. Lastly, an expansion of this research could be to use computer vision to collect the data from a spectrum analyzer waveform vs digital I/Q samples. These future thoughts are outside this proposal's scope but are presented here for reader consideration.

The RadioML dataset (DeepSig, 2024), consisting of 11 modulation classes, is labeled, with each modulation consisting of samples at 20 different SNR levels (-20dB to 18dB, in 2dB steps). There are one thousand samples for each SNR/Modulation combination. This gives us 220,000 samples to train/test (70/30 split). Each sample consists of a 2 x 128-time series vector of I/Q data points. We assume our data is independent and identically distributed (i.i.d); all our data come from the same distribution. The data requires no cleanup; duplicates do not exist since DeepSig provides it clean.

In RF applications, a two-dimensional sinusoidal signal is sampled with a complex component, adding a third dimension. With time on one axis, the in-phase and quadrature-phase components are known as the I/Q data points. The formulas below give them, where A is the amplitude, f is the frequency, and ϕ phase is the function of time, *t*.

$$I(t) = A * \cos(2\pi f t + \phi) \qquad Q(t) = A * \sin(2\pi f t + \phi)$$

Using multiple supervised machine learning algorithms described in 3 below, we can compare the performance of various models. The permutations of each model, such as data set sizes or hyperparameters, will be adjusted to achieve the best accuracy.

3. METHODS

FEATURES

Using DSP techniques, we engineered the features below to allow for classification. The ***scipy*** and ***PyWavelets*** Python libraries were used to derive the necessary features. Two sets of features were created to test the impact of feature selection on accuracy.

Feature Set 1

FEATURE	DEFINITION	USAGE
Signal Magnitude (mean, skew, kurtosis, std. Dev.)	[Magnitude=root(I2+Q2)]. The absolute value of signal amplitude.	Used in conjunction with signal phase to represent symbols on a constellation plot in digital signals. Different modulation schemes use distinct positions on the constellation to represent symbols. Thus, statistics on magnitude can help classify modulations.
Signal Phase (mean, skew, kurtosis, std. Dev.)	[Phase=tan−1(IQ)] Position of a point in time on a waveform cycle.	[Used with signal magnitude to represent symbols on a constellation plot in digital signals. Modulation schemes vary in the different phase values used for representing symbols. Phase statistics can help classify signal modulation.
Spectral Entropy	The measure of randomness in a signal's frequency spectrum	Used for classification based on modulation complexity

FEATURE	DEFINITION	USAGE
Peak Frequency	The frequency component with the highest amplitude	Identifies dominant frequencies for signal analysis
Signal Power (Average)	Mean of squared amplitudes of a signal	Evaluate signal strength for classification
Signal To Noise Ratio	The ratio of signal power to background noise	The usefulness of other statistics varies based on the amount of noise in the data.

Feature Set 2

FEATURE	DEFINITION	USAGE
Bandwidth	The range of frequencies occupied by a signal. Bandwidth helps distinguish between different modulation types, as modulated signals have characteristic bandwidths.	Some modulation schemes occupy a more comprehensive range of frequencies (e.g., OFDM), while others are narrower (e.g., AM, FM). Bandwidth can also help filter out unwanted noise or adjacent signals.
Center Frequency	The frequency at the center of the signal's bandwidth is critical in wireless communications and signal classification.	Different signals are transmitted at specific frequency bands (e.g., Wi-Fi, Bluetooth). Knowing the center frequency helps identify and classify modulation types based on expected frequency ranges.
Modulated	Indicates whether the signal is modulated and, if so, specifies the modulation type (e.g., AM, FM, PSK, QAM).	Modulation type provides essential information about how the signal is transformed, allowing models to differentiate between amplitude-modulated, phase-modulated, and frequency-modulated signals.
Signal Power (peak, average, std. Dev.)	Measures the strength of the signal. Peak power captures the highest value; average power gives the mean, and standard deviation measures variability.	Power characteristics help distinguish between modulation schemes, which may show varying power levels across time or frequency. For instance, PSK has consistent power, while QAM shows significant variation in power.
Higher-Order Statistics (HOS)	Skewness, kurtosis, and other moments of the signal's statistical distribution.	It captures non-linearities and subtle variations in the signal that first-order statistics might miss, helping to classify modulation types accurately.

FEATURE	DEFINITION	USAGE
Time-Frequency Representations	Is frequency content using STFT, Wavelet Transform, or Wigner-Ville distribution over time?	It offers a detailed view of how the signal frequency changes over time, aiding in classifying non-stationary signals, such as modulated ones.
Hilbert Transform Features	Extracts the analytic signal, providing instantaneous amplitude, phase, and frequency.	It is helpful for signals where instantaneous characteristics, such as amplitude or frequency, are critical for correct classification.

MODELS

SVM (PARAS GODA)

Support Vector Machines are a binary model that maps the data to a higher dimension and defines the type of separation boundary or hyperplane desired. When used as a classifier, as in our case, we use the Support Vector Classifier (SVC) versus a Support Vector Regressor (SVR). Common boundaries or hyperplanes studied here include linear, second-degree polynomial, and a radial basis function. Our fundamental problem is a multi-class classification for different modulation types. Conventional strategies to implement multi-class in SVCs include one-vs-one (OvO) and one-vs-rest (OvR). We use the ***scikit-learn (sklearn)*** Python SVC implementation, which utilizes the OvO strategy. This creates $\frac{classes*(classes-1)}{2} = 55$ Separate classifies under the hood that trains on data from two classifications. Under the hood, this uses LIBSVM, which uses the MSE as our loss function. We assume we will need a non-linear SVM and will prove this. The input support vectors and the decision boundary are defined by:

$$\omega = \sum_{i \to m} \alpha_i t_i K(x_i, x) + b$$

where x_i Are the input support vectors, and m is the number of total Lagrange multipliers, α_i. $K(x_i, x)$ Represents our kernel function, of which we test three variations:

$$Linear = \omega^T x + b$$

$$Polynomial = (\gamma \omega^T x + b)^N$$

$$Gaussian\ RBF = e^{-\gamma \|x-y\|^2}, \gamma = \frac{1}{2\sigma^2}$$

PREPROCESSING

SVMs are sensitive to pre-processing, so some simple tricks can help us achieve optimal training once we extract our different feature sets. One is to convert our label from a text string to an enumerated value. The second step is to scale our data, knowing that the I/Q samples are all numeric and have varying ranges. I use the ***StandardScaler***

class to rescale the data with a mean of 0 and a standard deviation of 1. Other scaling methods are available, but in the interest of time, I tested them to understand the impact of scaling on training time and accuracy.

Model	Permutation	Features	Convergence Time	Accuracy (Peak) %	HyperParameters
SVM	Linear – Unscaled Features	28	1620 minutes	39% (58%)	n/a
SVM	Linear – Scaled Features	28	18 minutes	44% (69%)	n/a

Table 1: SVM Permutations

From the , we reduced training time by 90-fold and increased overall accuracy by 5%. All future tests will use the scaled features as the time to converge could be more manageable and beneficial.

Linear hyperplanes are the most basic, but our accuracy score still needs to improve. This is unsurprising because half of our data has an SNR < 0, which causes much of the feature space to overlap. This can be observed in Figure 2 & **Error! Reference source not found. Error! Reference source not found.**, showing an increase in accuracy trend at 2db. The max accuracy was ~69% at an SNR of

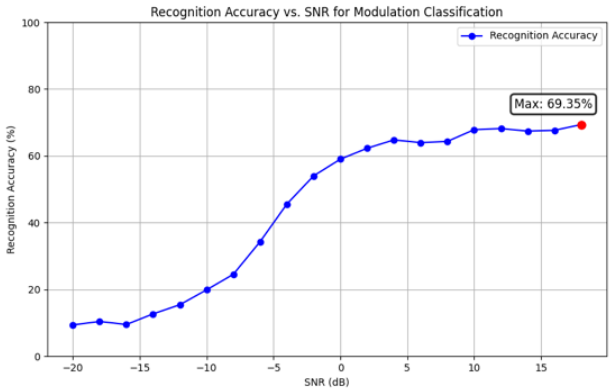


Figure 2: SVC(Linear) w/ Features Scaled

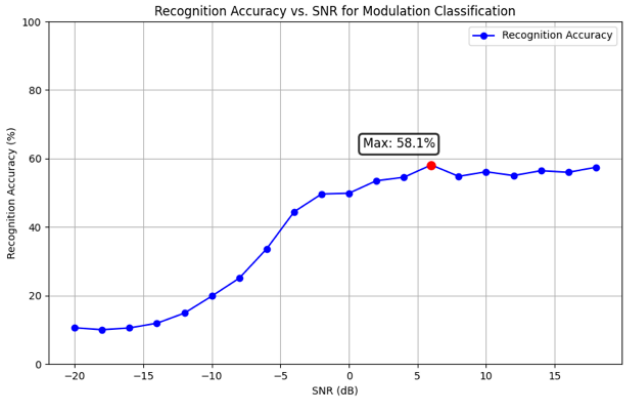


Figure 3: SVC(Linear) w/ Features Unscaled

18dB (an increase of ~10%).

MODEL EXPERIMENTATION

The other levers we can turn are various kernel functions. We tried a polynomial and radial basis function with multiple hyperparameters. As with everything else in SVM, trial and error are the methods for performing a "GridSearch" across various permutations. I used the standard built-in function for the linear kernel and did not do any

hyperparameter tuning. For the Polynomial kernel, I used a 2nd-degree polynomial with a gamma of 1/features and coefficient, b = 0.5.

For the RBF kernel, I did a Grid Search with gamma between $1e^{-9}$ to $1e^{3}$, and the C parameter between $1e^{-2}$ and $1e^{5}$, exponentially spaced apart into five increments. A higher regularization term (C) increases the penalty for the cost function but results in substantial computational numbers and convergence time. This allows for a soft margin minimization as we are confident a perfect hyperplane will not be discovered. I used a 1-fold cross-validation technique, with a 20% split, to do the search, which took almost 4 hours.

The data in Table 1 suggest that once mapped to a higher dimensional space, the features show a non-linear boundary as accuracy increased on average ~10%. However, the difference between adding more features showed a marginal improvement because they are derived from the same I/Q samples.

Model	Permutation	Features	Convergence Time	Accuracy (Peak) %	HyperParameters
SVM	Linear	12	12 minutes	41% (66%)	n/a
SVM	Polynomial	12	11 minutes	48% (78%)	degree=2, gamma='auto', coef0=1, C=5
SVM	RBF	12	140 minutes	50% (80%)	gamma=0.001, C=100000.0
SVM	Linear	28	18 minutes	44% (69%)	n/a
SVM	Polynomial	28	15 minutes	52% (75%)	degree=2, gamma='auto', coef0=1, C=5
SVM	RBF	28	546 minutes	53% (78%)	gamma=0.001, C=100000.0

Table 2: SVM Permutation Matrix

RESULTS & DISCUSSION

Further inspection of the results again reassures us that sub-zero SNRs have too much feature space overlap to distinguish well due to the noise overshadowing the signal. Above 0db, we see a plateau of accuracy.

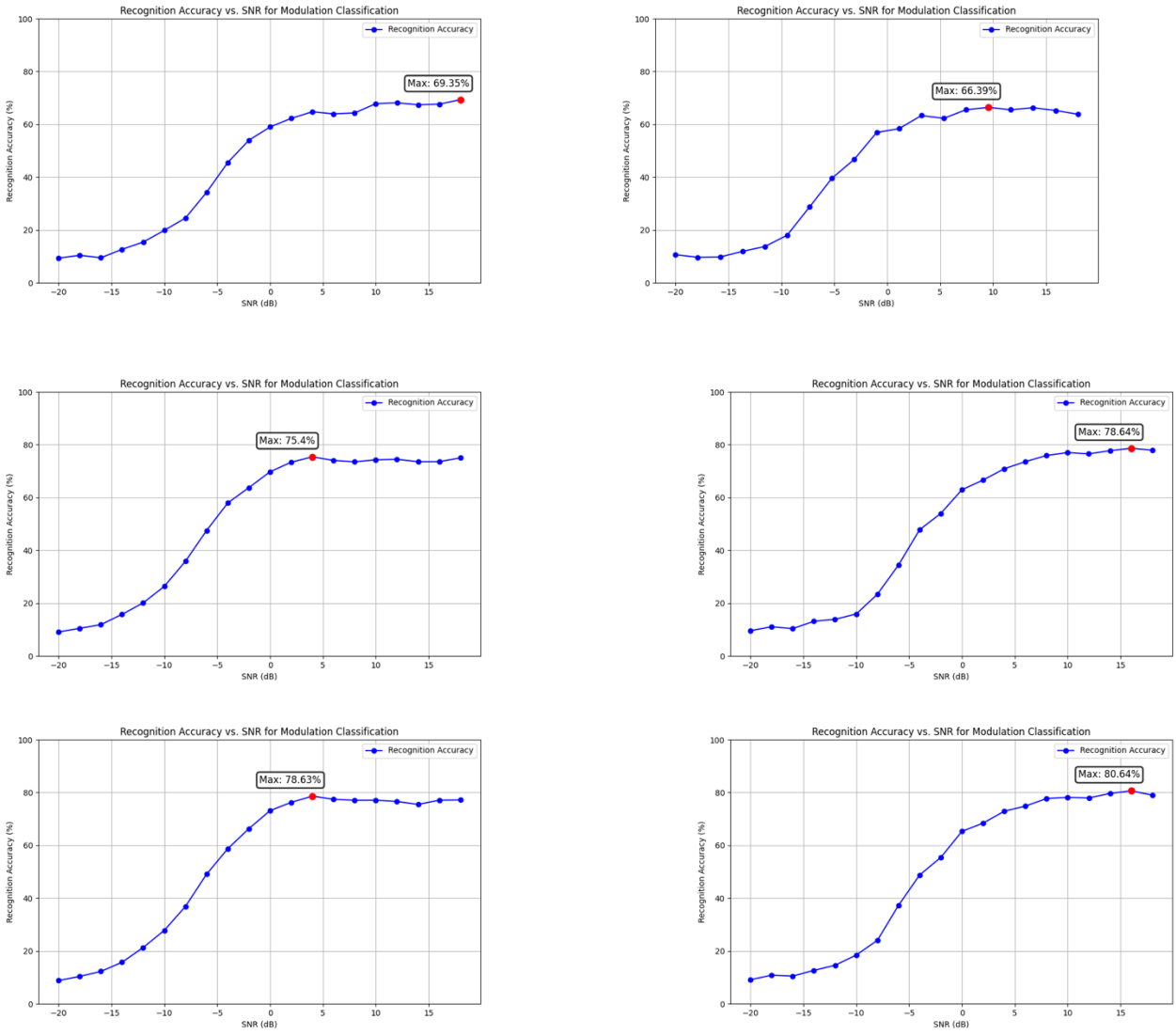


Figure 4: SNR Accuracy Curves

The accuracy scores of the training set vs the testing set are all within 1-2% of each other, indicating this is a manageable problem. While accuracy for QPSK and WBFM are relatively in line, the precision score is significantly lower, indicating many misclassifications. The confusion matrix reinforces this issue. This is expected as WBFM and AM-DSB share characteristics, including spectral density, multiple sidebands, and a wide bandwidth. There was also noticeable confusion with QPSK, 8PSK, QAN16, and QAM64 signals for similar reasons. One of the most significant findings and likely issues with these signals is that they are a time series with features that resemble each other throughout the signal. SVMs are sampling support vectors of engineered features and do not consider the signal's time domain change and histogram.

Classification Report for Modulation Types:								
Train Set Result:					**Test Set Result:**			
	precision	*recall*	*f1-score*	*support*	*precision*	*recall*	*f1-score*	*support*
8PSK	0.42	0.49	0.45	14014	0.4	0.47	0.43	5986
AM-DSB	0.49	0.62	0.55	13969	0.46	0.58	0.51	6031
AM-SSB	0.5	0.77	0.6	14031	0.48	0.77	0.59	5969
BPSK	0.74	0.55	0.63	13946	0.71	0.53	0.61	6054
CPFSK	0.59	0.58	0.58	13946	0.58	0.55	0.56	6054
GFSK	0.63	0.63	0.63	14053	0.6	0.59	0.6	5947
PAM4	0.71	0.64	0.68	13974	0.7	0.62	0.66	6026
QAM16	0.55	0.51	0.53	14103	0.53	0.5	0.51	5897
QAM64	0.56	0.69	0.62	14035	0.55	0.69	0.61	5965
QPSK	0.53	0.35	0.43	13877	0.5	0.33	0.39	6123
WBFM	0.51	0.29	0.37	14052	0.44	0.26	0.32	5948
accuracy			0.56	154000			0.53	66000
macro avg	0.57	0.56	0.55	154000	0.54	0.53	0.53	66000
weighted avg	0.57	0.56	0.55	154000	0.54	0.53	0.53	66000

Table 3: SVM RBF KERNEL w/ 28 Features (All SNRs)

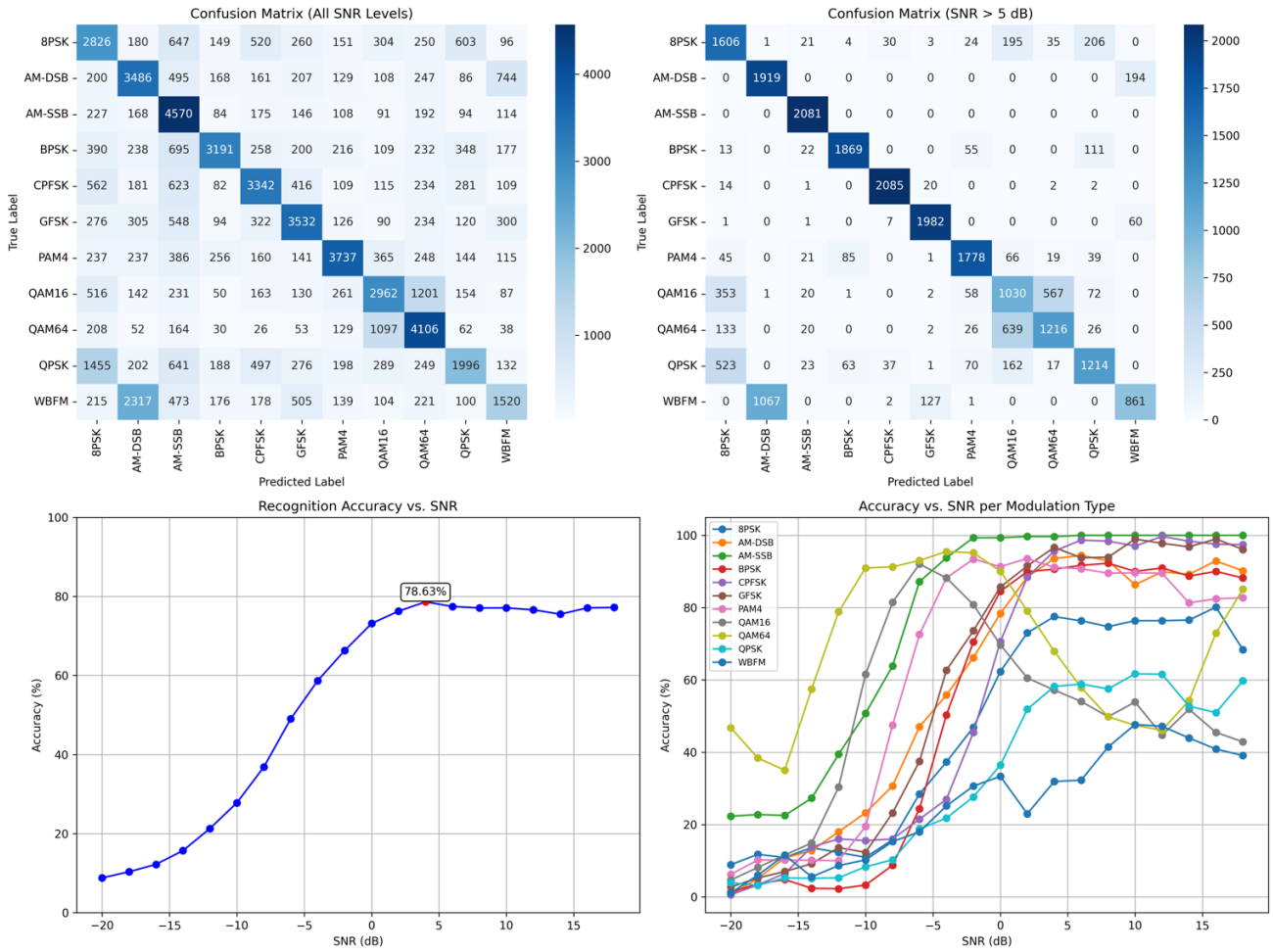


Figure 5: SVM RBF KERNEL w/ 28 Features

SOURCE CODE LOCATION

This repository link contains all the code to generate, train, and evaluate the models and the results used in git for source control.

https://github.com/gawdygoda/Wireless-Signal-Classification-ML

EXPERIMENTAL DESIGN

A decision tree is a nonparametric supervised learning algorithm. Its goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The algorithm has a hierarchical tree structure consisting of decision and leaf nodes, as shown below.

The algorithm has a hierarchical tree structure consisting of decision and leaf nodes, as shown in the Figure 6.

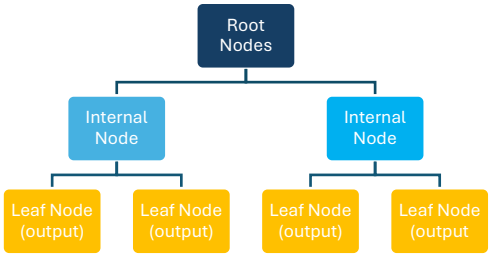


Figure 6: Decision Trees

Decision nodes contain conditions (from data features) to split the data, and the latter helps to decide the class of the data point. At each Node, our model needs to learn which features to take and the corresponding threshold values to split the data optimally.

How do you choose the best attribute at each node?

The decision tree Information Gain (IG) and Gini Impurity (GI) methods commonly use two criterion functions. These criterion functions measure the "impurity" or randomness within a data set, helping the algorithm choose the best feature to split on at each node. The algorithm aims to create more homogeneous sub-groups with lower impurity levels.

Information Gain (IG):

The Decision Tree model compares each possible split and takes the one that maximizes IG.

$$IG(A.S) = H(S) - \sum_{j=1}^{v} \frac{|S_j|}{|S|} * H(S_j)$$

Gini Impurity (GI):

A low Gini impurity score indicates a better split. The Gini impurity is at its minimum (zero) when all cases in a node are in a single target category.

$$GI = 1 - \sum_{i=1}^{n} (p_i)^2$$

DATA PREPROCESSING

The dataset is open source in a dictionary format, including the modulation type, I/Q points, and SNR levels. We assume our data is independent, identically distributed, and cleaned up. The data from the original file in "pickle format" was transformed into a 2-dimensional array in table format with rows (samples or singles) and columns (data features). For data preparation, we need to convert categorical data (modulation classes) to an enumerated value using a "*label encoder" function from the* sklearn library. This involves finding thresholds that optimize a particular criterion (Information Gain or Gini). In addition, to scale our I/Q raw data with a "0" mean and "1" standard deviation using "*StandardScaler"* from the sklearn library. However, data scaling doesn't adequately improve the model's performance so that the model can be trained in both scaled and unscaled data. Finally, before training the model, create two separate data frame tables, one for the data features and the other for the encoded label classes.

TRAINING PROCEDURE

After developing the two data frames of input feature and output classes, the model is ready for training now. Using the function "train_test_split" from the sklearn library, the data is split 80% for training and 20% for testing.

The decision tree classification function "DecisionTreeClassifier" is from the sklearn library without hyperparameter tuning. Due to the hardware limitations of my PC, I'm running the models only with 5 modulation classes (BPSK, QPSK, QAM16, WBFM, and GFSK). A total of 11 features are selected as follows:

- In-phase (I) and Quadrature (Q) Statistics (Mean, Variance, Skewness, and kurtosis) for the magnitude and similar features of the phase. A total of 8 features
- Signal Power (Peak and Average). A total of two features
- Spectral entropy. One feature
- Signal-to-noise ratio (SNR). One feature

HYPERPARAMETERS TUNING

Decision trees are powerful models used in machine learning. However, their performance relies highly on the hyperparameters, which are selected optimally. (How to tune a Decision Tree in Hyperparameter tuning, 2024) Hyperparameters can significantly impact the model's accuracy, generalization ability, and robustness. Generally, there are two types of model parameters:

1. Learnable parameters: These are updated iteratively during training, and external assistance is required to tune them.
2. Hyperparameter: a parameter defined before the learning process begins and helps control aspects of the learning process. Examples of hyperparameters include the maximum depth, minimum samples split, the cross-validation splitting strategy, the criterion function (Gini or Entropy), and maximum leaf nodes.

Using the *"GridSearchCV"* function from the sklearn python library (GridSearchCV, n.d.), the best decision tree for the five Modulation Classification is as follows:

- The best criterion function is Gini
- The best maximum depth is 20
- The best minimum sample leaf is 10

- The best cross-validation k-fold is 5

With optimizing the hyperparameters**, the best accuracy is 64.07%,** with training samples from all SNR levels.

TRAINING MODELS FOR DECISION TREE CLASSIFIER

The Decision Tree classification function is trained for different models as follows:

Model 1: The dataset was filtered to select only positive SNR levels. The decision tree classifier was run without any hyperparameter tuning.

Model 2: The dataset includes all SNR levels. The decision tree classifier is run without any hyperparameter tuning.

Model 3: The dataset includes all SNR levels. The decision tree classifier is run with hyperparameters tuning.

Model 4: The dataset includes all SNR levels. The decision tree classifier is run with hyperparameters tuning. The input features are scaled with "0" mean and "1" variance.

Model	Permutation	Features	Convergence Time	Accuracy (Peak) %	Hyperparameters
1	Unscaled Features	12	15 minutes	90.4% (95.2%)	n/a
2	Unscaled Features	12	25 minutes	62.6% (95.3%)	n/a
3	Unscaled Features	12	30 minutes	64.07% (94.9)	Criterion: Gini Max depth:20 Min samples leaf:10 Cross-validation k-fold:5
4	scaled Features (zero mean and one variance)	12	25 minutes	64.25%(94.9%)	Criterion: Gini Max depth:20 Min samples leaf:10 Cross-validation k-fold:5

Table 4: Decision Tree Models

RESULTS AND DISCUSSION

The decision tree classification model revealed its power and ability to classify the different modulation signals at varying levels of SNR with an overall accuracy of 64%. Moreover, there is no need for further scaling and normalization (just tiny improvements) of the dataset if it is independent, identically distributed, and cleaned up. However, the models are significantly affected by the high noise level in the signal, which eventually reduces the model prediction and accuracy due to the noise overshadowing the signal. As the noise ratio increases (noise decreases), classification accuracy increases. The accuracy starts overshooting when the SNR level is above -10 db. In addition, tuning the hyperparameters slightly contributes to the overall accuracy (3% improvement). Due to the large number of data samples trained, hyperparameter tuning would be more sufficient if we added more features to the data. The modulation types most accurately classified at high SNRs (0 dB or above) were WBFM and QPSK. However, throughout all possible SNR levels, the most accurately classified were WBFM and QAM16.

SOURCE CODE LOCATION

This repository link contains all the code to generate, train, and evaluate the models and the results used in git for source control.

https://github.com/ANAS2119/ML_Wireless_Classification.git

FIGURES AND TABLES

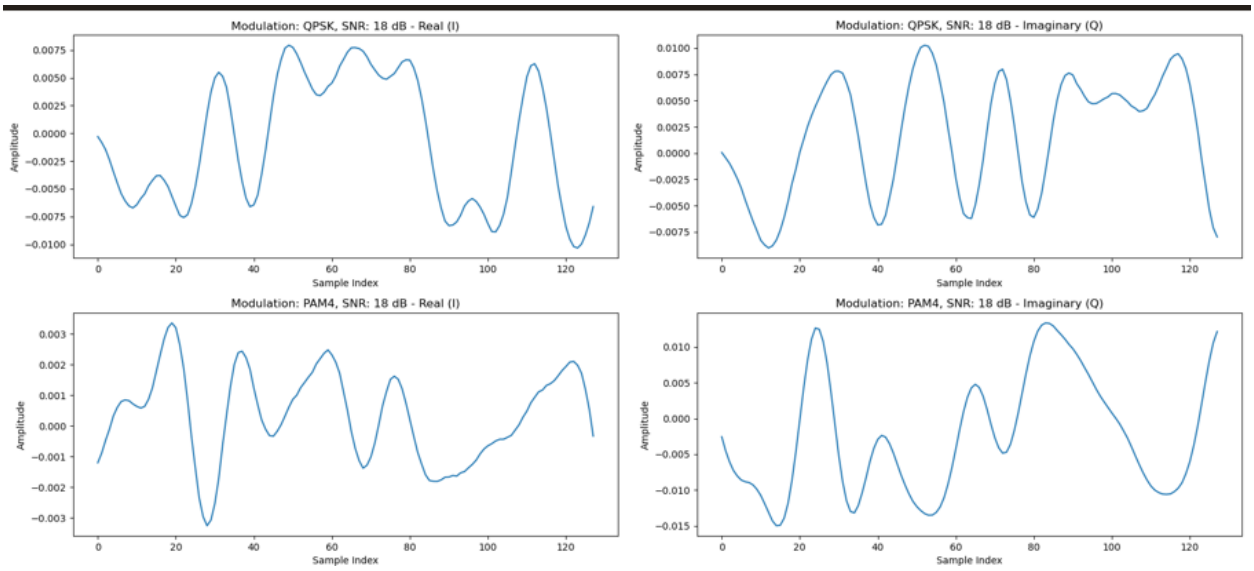


Figure 7: Modulation type's highest SNR signal

signal_type	snr	magnitude	magnitude_std	magnitude_skew	magnitude_kurtosis	phase_mean	phase_std	phase_skew	phase_kurtosis	spectral_entropy	peak_frequency	average_power
QPSK	2	0.007812	0.002739581	-0.332200528	0.558704971	-0.011607189	1.77863	-0.094661469	-1.368190543	3.2917287	0.125	1.70E-05
QPSK	2	0.007813	0.002668424	0.025987751	-0.042563169	-0.1481909	1.6842675	-0.089194763	-1.160112904	3.7682638	0.09375	1.48E-05
QPSK	2	0.007813	0.002873162	0.115224646	-0.526413789	0.15516579	1.7097895	-0.106532738	-1.115362117	3.3847487	0.0625	1.87E-05
QPSK	2	0.007813	0.002991104	0.079100769	-0.680334657	-0.007725559	1.7436442	-0.052037171	-1.21997759	3.2234788	0.078125	2.20E-05
QPSK	2	0.007813	0.002577783	-0.056228434	-0.535420397	-0.07072617	1.5851145	0.353147319	-0.938411505	3.7234604	0.0546875	1.30E-05
QPSK	2	0.007813	0.003065837	-0.011615485	-0.639522614	0.5164175	1.815129	-0.199052571	-1.401104858	3.6585352	0.1015625	1.74E-05
QPSK	2	0.007813	0.002637571	0.242168074	-0.037016022	0.61936724	1.647029	-0.38082765	-0.93228309	3.5607316	0.125	1.08E-05
QPSK	2	0.007812	0.002752842	0.046444236	-0.24018806	0.71449846	1.6252294	-0.52809472	-0.655194566	3.6336532	0.046875	1.34E-05
QPSK	2	0.007813	0.002914212	-0.138513081	-0.845584429	0.60954857	2.1372845	-0.441222514	-1.503120727	3.57051	0.0859375	1.42E-05
QPSK	2	0.007813	0.002576619	0.176206745	0.661328257	-0.27124935	1.8471993	0.237297943	-1.242769082	3.4245727	0.0703125	1.03E-05
QPSK	2	0.007812	0.002528884	-0.36329668	-0.347799745	0.06670363	1.8387709	-0.193844549	-1.424158462	3.420837	0.0703125	1.05E-05
QPSK	2	0.007813	0.00234345	-0.315921444	-0.01796356	-0.09318476	1.4513317	0.13240322	-0.315610332	3.5803957	0.125	1.23E-05
QPSK	2	0.007812	0.003072118	-0.098209103	-0.437400098	0.18604551	1.9860312	-0.20437738	-1.431641684	3.1765854	0.1015625	1.72E-05
QPSK	2	0.007813	0.002631833	-0.20260421	0.169667551	0.3316447	1.924003	-0.158010381	-1.451291531	3.6544836	0.078125	1.10E-05

Figure 8: Dataset in a table format, 2-dimensional array

CONFUSION MATRIX

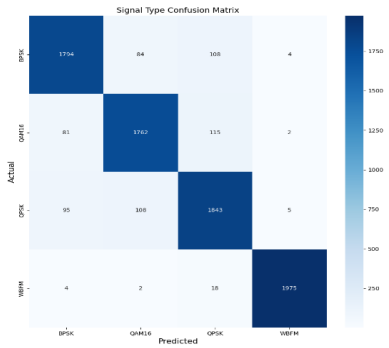


Figure 9: DT Model 1 Confusion Matrix

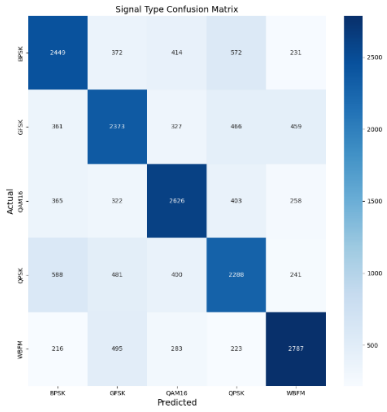


Figure 10: DT Model 2 Confusion Matrix

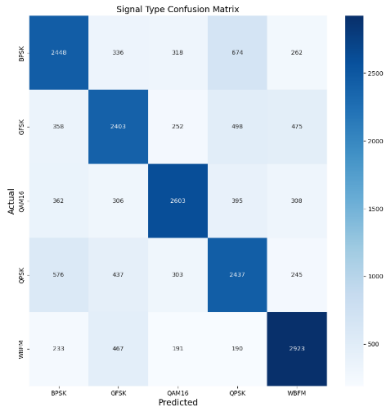


Figure 11: DT Model 3 Confusion Matrix

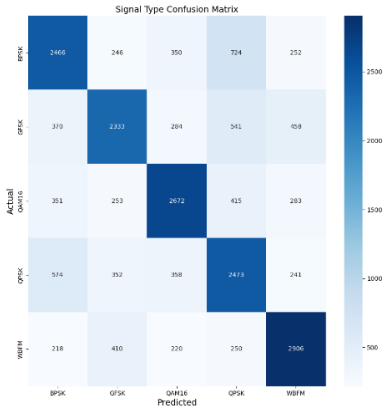


Figure 12: DT Model 4 Confusion Matrix

ACCURACY VS SNR

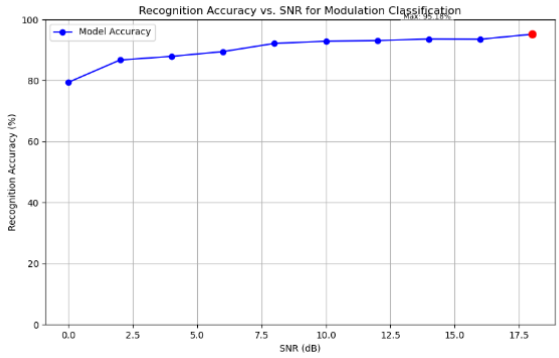


Figure 13: DT Model 1 Accuracy vs SNR

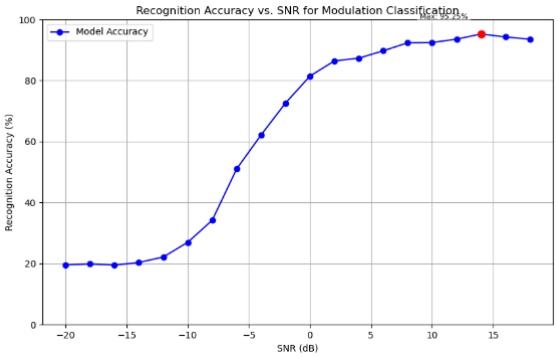


Figure 14: DT Model 2 Accuracy vs SNR

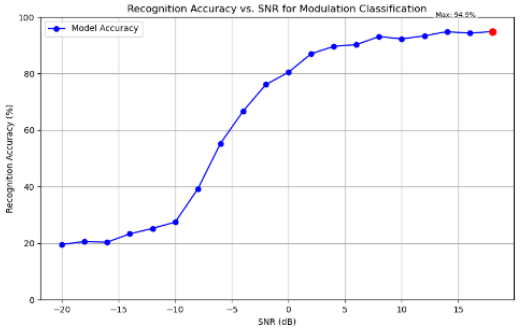


Figure 15: DT Model 3 Accuracy vs SNR

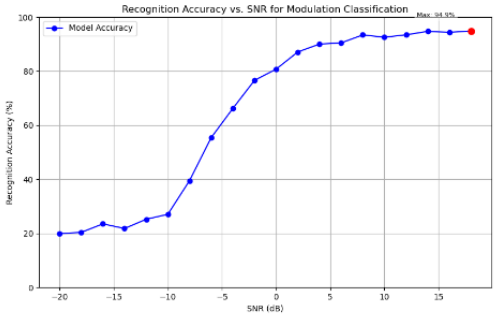


Figure 16: DT Model 4 Accuracy vs SNR

GRADIENT BOOSTED TREES (CURTIS ZGODA)

Gradient Boosted Trees (GBT) represent a powerful ensemble machine learning technique that sequentially constructs decision trees to create robust predictive models. Unlike single decision trees, GBT builds an ensemble where each subsequent tree is trained to correct the errors of the previous trees, effectively minimizing the residual prediction errors through a gradient descent-like optimization process. This approach allows the model to capture complex non-linear relationships in data with remarkable precision. The fundamental strength of Gradient Boosted Trees lies in their ability to iteratively improve predictive performance by learning from previous mistakes. Combining multiple weak learners (decision trees) into a strong predictive model allows GBT to handle intricate feature interactions without requiring extensive manual feature engineering.

EXPERIMENTAL DESIGN

MODEL ARCHITECTURE

A Gradient Boosted Tree model in Python can be implemented using XGBoost, with key hyperparameters including tree depth, number of estimators, and learning rate tuned to optimize performance. The tree depth was systematically tested from 1 to 10, exploring how model complexity impacts predictive capabilities. The number of estimators was investigated from 50 to 750, incrementing by 50 to comprehensively evaluate the model's performance as the ensemble size changes.

The learning rate was methodically examined from 0.05 to 0.25 with step increments 0.05. By exploring these hyperparameters systematically, the goal was to identify the optimal configuration that balances model complexity, generalization, and computational efficiency.

PREPROCESSING

Feature extraction and preprocessing were critical steps in ensuring the dataset was suitable for use with gradient-boosted trees (GBTs). Gradient-boosted trees utilize structured numerical data, making it essential to extract features that represent the underlying statistical characteristics of the signal data. Derived features such as signal amplitude, frequency domain characteristics, and time-domain patterns to capture distinctions between modulation types while reducing raw data complexity from IQ data. The statistical features extracted from the training data were explicitly chosen for signal data patterns that emerge for modulation schemes, such as phase and magnitude variation.

TRAINING PROCEDURE

After preparing the data, the model is initialized to run with three primary hyperparameters that control the model's training: tree depth, number of estimators, and learning rate. Tree depth defines the maximum depth of the decision trees in the model. Increasing depth increases the complexity of the model and can improve the model's accuracy. However, increasing the tree depth too high results in overtraining and long training times. The number of estimators is the number of decision trees the model will generate. This goes hand in hand with the maximum tree depth, as it can lead to over-training and long training times. The learning rate is the weight each decision in backpropagation has when modifying the decision weights of the decision trees. The data is then split into training and

testing data, and the model is called to train on the given data. This trained model can then predict the testing data's labels.

MODEL EXPERIMENTATION

Most of the model experimentation came from tuning the hyperparameters of the model. Table 5 shows six of the model experiments. These experimental results were chosen to demonstrate the difference in hyperparameters. These were the most important results for learning the ideal hyperparameters for the model. Tree depth increases the model's accuracy until the depth reaches 5, then the accuracy decreases slightly as the model is over-trained. The number of estimators was less crucial, as shown in the difference between the trial with a depth of 9 and 500 estimators versus a depth of 10 with 300 estimators. Lastly, the learning rate had an accuracy peak when it was selected to be 0.1. Higher values lead to over-training, and lower values lead to under-training, resulting in lower accuracy during testing.

Model	Features	Training Time	Accuracy (Peak) %	Tree Depth	Number of Estimators	Learning Rate
XGBOOST	12	12 minutes	45.19% (74%)	1	300	0.1
XGBOOST	12	34 minutes	50.54% (79%)	2	500	0.1
XGBOOST	12	47 minutes	51.99% (81%)	5	300	0.1
XGBOOST	12	47 minutes	51.46% (80%)	5	300	0.2
XGBOOST	12	131 minutes	51.07% (81%)	9	500	0.1
XGBOOST	12	87 minutes	51.20% (81%)	10	300	0.1

Table 5: XGBoostedTrees Training Matrix

RESULTS & DISCUSSION

Examination of the recognition accuracy graphs reveals key takeaways from the experiments run with this model. As the signal-to-noise ratio increases, classification accuracy increases. This trend starts to plateau at an SNR of 0 dB and remains consistent when the SNR is more significant than 10 dB. Examining the recognition accuracy by signal modulation over SNR shows that the gradient-boosted tree model selected performed over 15% better at classifying specific signals given the selected statistical features. The modulation types most accurately classified at high SNRs (10 dB or above) were CPFSK, GFSK, PAM4, AM-SSB, and BPSK. The modulation types that were the least

accurately classified were QAM64, QAM16, QPSK, 8PSK, and WBFM. Two modulation types were outliers as they did not have their highest classification accuracy at high SNR values. QAM64 and QAM16 had their highest classification rate at an SNR of –4 dB.

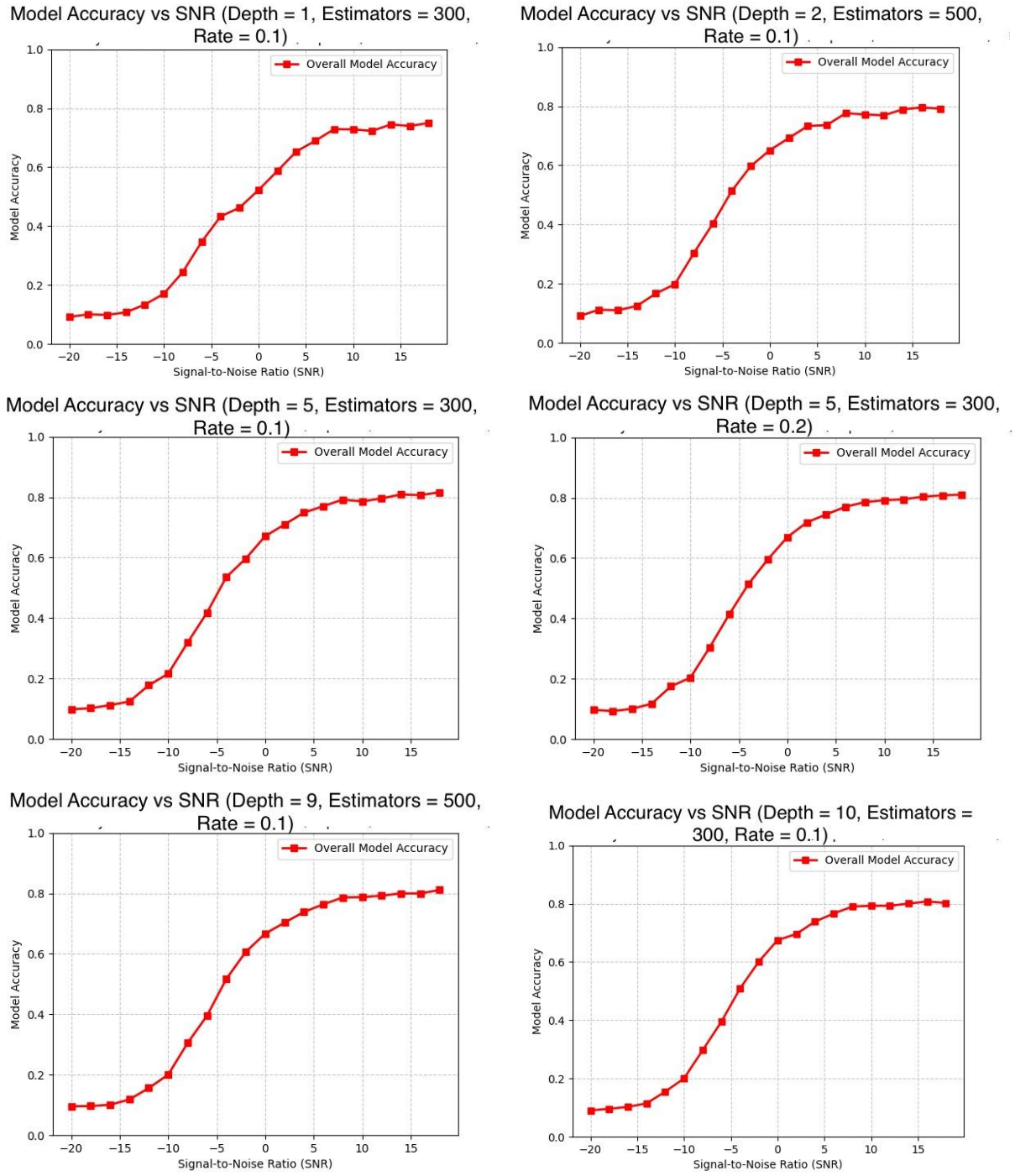


Figure 17: SNR Accuracy Curves

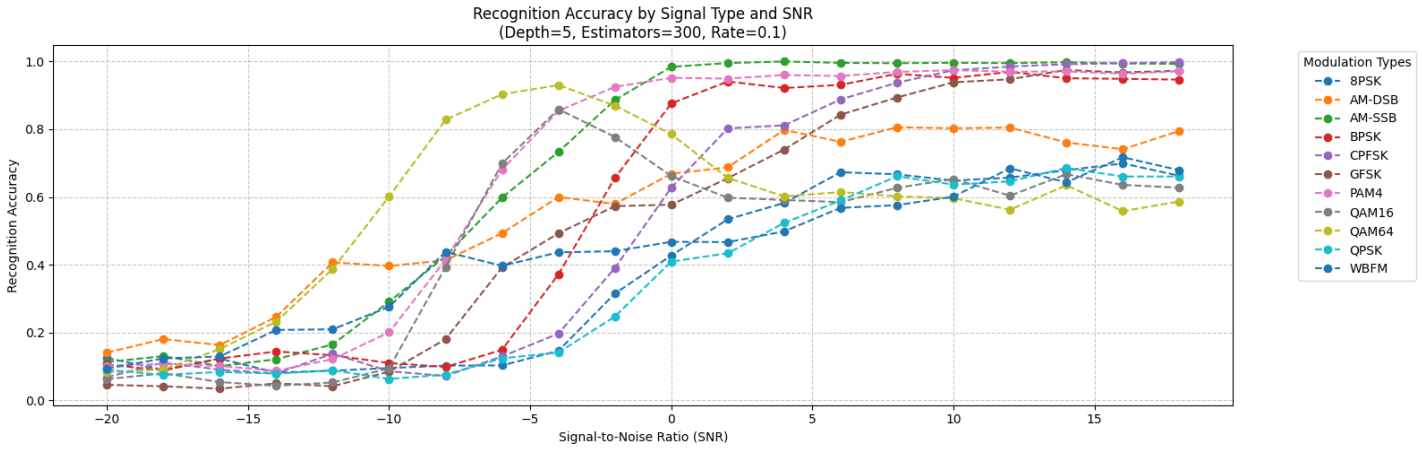


Figure 18: SNR Accuracy Curve by Modulation

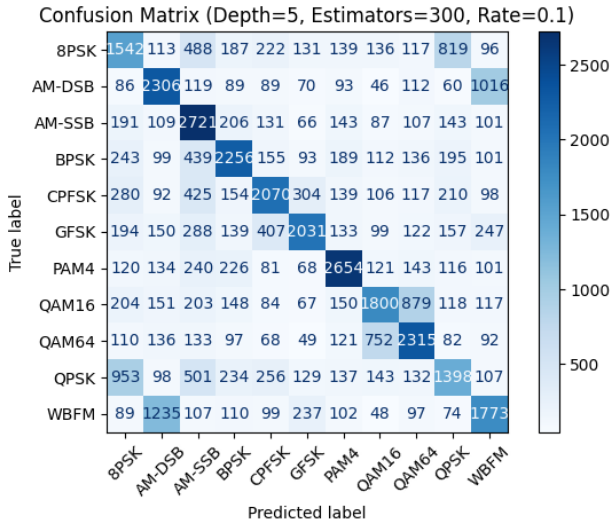


Figure 19: Confusion Matrix (All SNRs)

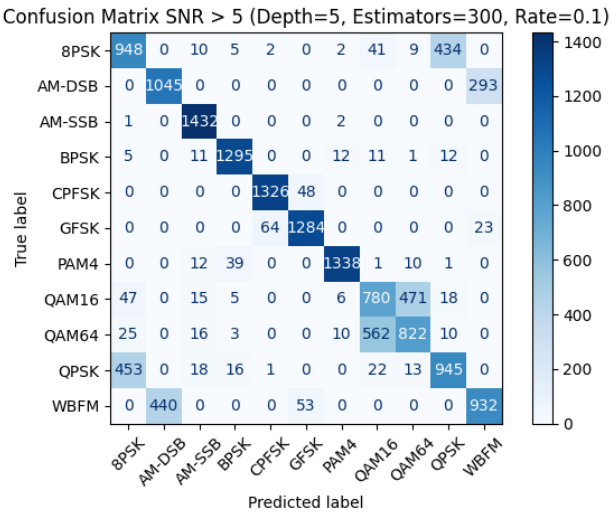


Figure 20: Confusion Matrix (SNR > 5)

CLASS	PRECISION	RECALL	F1-SCORE	SUPPORT
8PSK	0.38	0.39	0.39	3990
AM-DSB	0.49	0.56	0.53	4086
AM-SSB	0.49	0.68	0.57	4005

BPSK	0.57	0.56	0.57	4018
CPFSK	0.57	0.52	0.54	3995
GFSK	0.62	0.51	0.56	3967
PAM4	0.66	0.67	0.66	4004
QAM16	0.52	0.46	0.49	3921
QAM64	0.54	0.58	0.56	3955
QPSK	0.42	0.34	0.37	4088
WBFM	0.46	0.44	0.45	3971
accuracy			0.52	44000
Macro avg	0.52	0.52	0.52	44000
Weighted avg	0.52	0.52	0.52	44000

Table 6: XGBoost Classification Matrix

An overall accuracy of 52% could be better for most applications. However, many everyday signals using these modulation schemes are received with an SNR of 10dB or higher. Examining the results for SNR of 10dB or higher shows an accuracy of approximately 80%. This is significantly better than the overall accuracy. The confusion matrix of classification with signals of SNR > 5 indicates that many modulation types can be narrowed down to a smaller subset of types. In addition to this, the extracted features could be explicitly modified to classify between a smaller subset of signals. This data set encapsulates many classification labels, and many include similar phase or magnitude properties that lead to misclassifications, reducing accuracy. An example is 8PSK and QPSK, where magnitude remains identical across the two modulations, and there is significant overlap in phase values. This led to misclassification even at SNRs greater than 5dB.

SOURCE CODE LOCATION

All the code to generate, train, and evaluate the models and the results for the GBT section used git for source control and can be accessed at this repository link.

https://github.com/cujo612/gradient-boosted-trees

LSTM RNN (JACOB RAMEY)

A Recurrent Neural Network (RNN) is helpful in patterns in time series data and is instrumental in signal processing. In this case, we directly use the IQ samples as inputs to a neural network, then train and let the model learn what it needs. This is a massive benefit since no features need to be extracted, making it suitable for real-time interference applications and systems with limited resources (i.e., ARM, NVIDIA Jetson, Raspberry Pi, etc.). Using an RNN, we achieved superior accuracy over 0dB, around 94.5%. See the figures in the following sections to better understand how accuracy is determined with this dataset and model. More than specifying the model's overall accuracy is needed; by showing how accurate it is over SNR ranges and different classification types, we can understand where the model falls short and then target that area for improvement.

EXPERIMENTAL DESIGN

MODEL ARCHITECTURE

An RNN can be implemented using TensorFlow in Python. The input takes the shape of the data and passes it through the layers. The LSTM layers all had a dropout layer between them, which randomly dropped some of the features during training to prevent overfitting. The values (0.5, 0.2, 0.1) were adjusted from (0.2, 0.2, 0.2) and a slight increase in accuracy was achieved. After the training, the weights are updated if the accuracy improves, but the original features are kept in the model. Size 128 was chosen because the IQ data was 128 samples wide. In a system with more than 128 samples, we could increase that to match (i.e., 256, 512, 1024) or downsample and see if necessary. A smaller model should make inferences faster.

```python
def build_model(self, input_shape, num_classes):
    if os.path.exists(self.model_path):
        print(f"Loading existing model from {self.model_path}")
        self.model = load_model(self.model_path)
    else:
        print(f"Building new model")
        self.model = Sequential()
        self.model.add(LSTM(128, input_shape=input_shape, return_sequences=True))
        self.model.add(Dropout(0.5))
        self.model.add(LSTM(128, return_sequences=False))
        self.model.add(Dropout(0.2))
        self.model.add(Dense(128, activation="relu"))
        self.model.add(Dropout(0.1))
        self.model.add(Dense(num_classes, activation="softmax"))

        optimizer = Adam(learning_rate=self.learning_rate)
        self.model.compile(
            loss="sparse_categorical_crossentropy",
            optimizer=optimizer,
            metrics=["accuracy"],
        )
```

The model's learning rate started at 0.0001 and was adjusted throughout training to overcome what seemed to be overfitting. The hyperparameter tuning section below provides more information about this.

PREPROCESSING

The IQ and SNR data were extracted from the dataset and used as features. The SNR value was extended to 128 samples wide to match the IQ data. The dataset was in a pickle format, so Python was used to load and process it. The figure to the right shows the routine used to prepare the training and testing data from the RML2016a dataset.

```python
def prepare_data(self):
    X, y = [], []

    for (mod_type, snr), signals in self.data.items():
        for signal in signals:
            # # Stack the normalized real and imaginary parts to form a (128, 2) array
            iq_signal = np.vstack([signal[0], signal[1]]).T
            snr_signal = np.full((128, 1), snr)
            combined_signal = np.hstack([iq_signal, snr_signal])
            X.append(combined_signal)
            y.append(mod_type)

    X = np.array(X)
    y = np.array(y)
    self.label_encoder = LabelEncoder()
    y_encoded = self.label_encoder.fit_transform(y)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y_encoded, test_size=0.2, random_state=42
    )
    X_train = X_train.reshape(-1, X_train.shape[1], X_train.shape[2])
    X_test = X_test.reshape(-1, X_test.shape[1], X_test.shape[2])

    return X_train, X_test, y_train, y_test
```

TRAINING PROCEDURE

After preparing the data, the model was trained using a simple routine, as shown to the right. The training routine used the train/test data produced from the previous routine as the data. Some callbacks were used to monitor training progress and change the learning rate throughout the process to find the best learning rate. I experimented with using an alternative class weight dictionary to give more weight to WBFM/AM-DSB since the model struggled the most with those. This didn't help improve the model's accuracy. However, it still had difficulty with WBFM/AM-DSB and degraded performance. More experimentation here could yield better results.

The training was done on a PC with an RTX2060 GPU and an NVIDIA Jetson Xavier AGX. Training in both was used to test different models in parallel.

```python
def train(
    self,
    X_train,
    y_train,
    X_test,
    y_test,
    epochs=20,
    batch_size=64,
    use_clr=True,
    clr_step_size=10,
):
    callbacks = []

    if use_clr:
        clr_scheduler = LearningRateScheduler(
            lambda epoch: self.cyclical_lr(epoch, step_size=clr_step_size)
        )
        callbacks.append(clr_scheduler)

    stats_interval = 20
    for epoch in range(epochs // stats_interval):
        history = self.model.fit(
            np.nan_to_num(X_train, nan=0.0),
            y_train,
            epochs=stats_interval,
            batch_size=batch_size,
            validation_data=(X_test, y_test),
            callbacks=callbacks,
            class_weight=self.class_weights_dict
        )

        self.update_epoch_stats(epochs)
        current_accuracy = max(history.history["val_accuracy"])
        self.update_and_save_stats(current_accuracy)

    return history
```

HYPERPARAMETER TUNING

These are the hyperparameters used and how they adjusted:

- **Learning Rate**: This changed from anywhere between 0.01 to 1e-7. The best results I had were in the 1e-4 to 1e-6 range. A learning rate scheduler was used to cycle between different learning rates during training to find the best rates and prevent getting stuck.
- **Dropout values**: The original dropout values of 0.2, 0.2, and 0.2 were adjusted to 0.5, 0.2, and 0.1 to reduce overfitting. These are more difficult to modify, and the model needed to be retrained entirely when I did this. I tried to transfer the weights but still had difficulty with that. Eventually, I retrained it, which was marginally superior (67% to 68% overall accuracy). The main improvement was in >0dB SNR, and the WBFM classification rate increased from 34% to approximately 45%.
- **Batch Size and Epochs**: I initially used 20 epochs and a batch size of 64. Most of the training was done using those values. I eventually tried adjusting the number of epochs and batch size, but after training my original model, it did not improve very much. I tried to train a model from the start using five epochs, and this model struggled to learn. I'd suggest starting with more epochs, then implementing early stopping to prevent overfitting and giving the model the epochs it needs to learn usually.

I modified some other hyperparameters but needed more time to train thoroughly.

- **Number of units in the LSTM/Dense layers**: The 128 in the LSTM layers could be increased or decreased to change the complexity. The dense layers can be adjusted similarly.
- **Number of LSTM layers**: Adding more LSTM layers with more gradual dropout rates
- **LSTM type**: It could be adjusted to be a bidirectional LSTM to capture patterns in both directions

RESULTS AND DISCUSSION

CLASS	PRECISION	RECALL	F1-SCORE	SUPPORT
8PSK	0.77	0.60	0.68	3990
AM-DSB	0.50	0.77	0.61	4086
AM-SSB	0.51	0.80	0.62	4005
BPSK	0.84	0.66	0.74	4018
CPFSK	0.83	0.64	0.72	3995
GFSK	0.77	0.68	0.73	3967
PAM4	0.79	0.77	0.78	4004
QAM16	0.77	0.75	0.76	3921

QAM64	0.69	0.86	0.77	3955
QPSK	0.74	0.60	0.66	4088
WBFM	0.47	0.32	0.38	3971
Overall Accuracy	68%			
Accuracy Over 0dB	94.5%			
Accuracy At -4dB	80%			

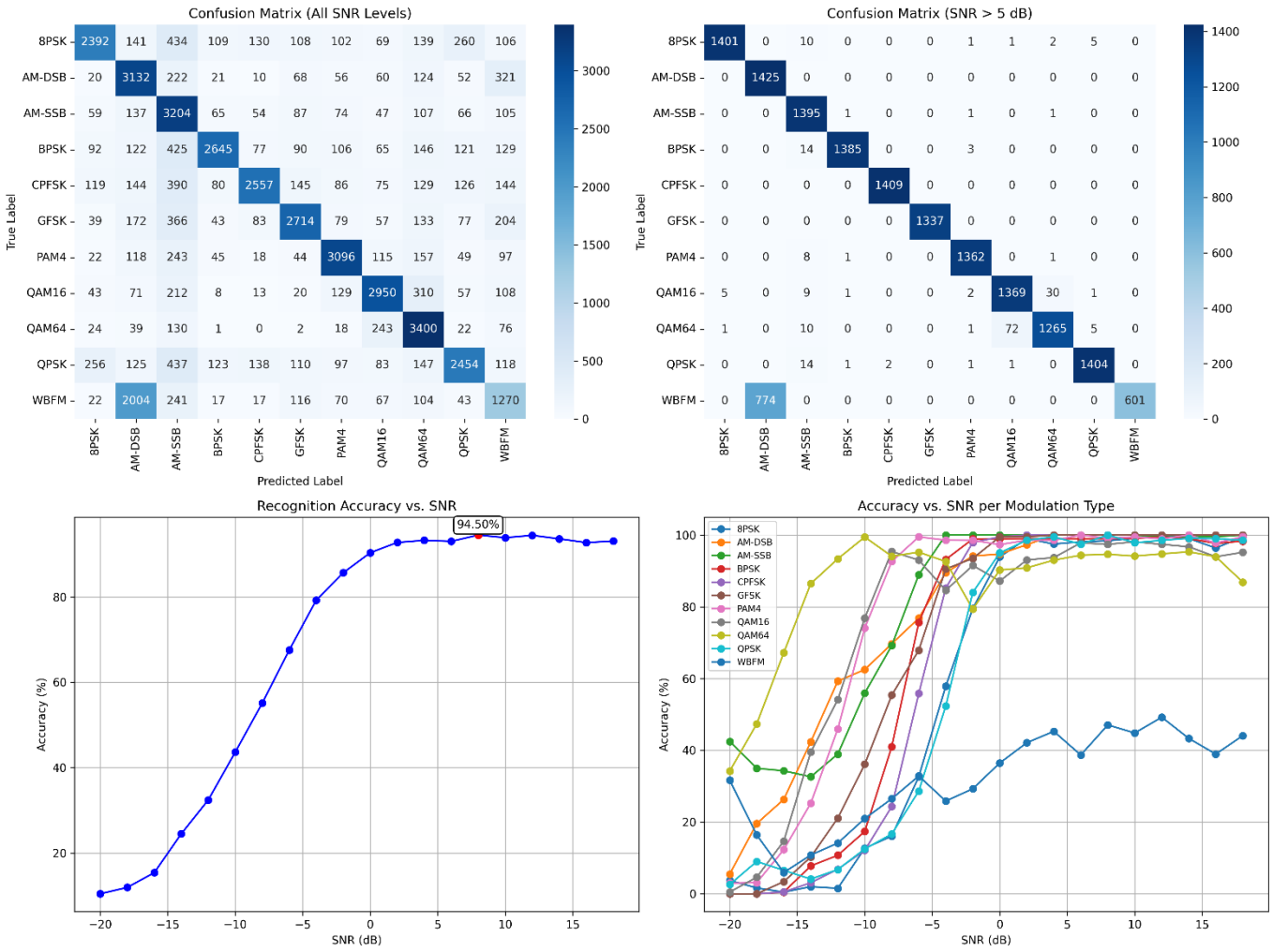


Figure 21: Confusion Matrices and Accuracy plots for SNR RNN LSTM

The overall accuracy of 68% needs to be more accurate. It's helpful to see how the accuracy changes over changes in SNR and for the different classification types to understand where the model falls short. We can see that

the model has significant confusion at the lower SNR, which increases dramatically with the SNR up to an approximate accuracy of 94.5%. The accuracy at higher SNR (over 5dB) was very high in most areas, but the model struggled to differentiate WBFM from AM-DSB. This was consistent with any hyperparameters I tried. The final plot shows accuracy versus SNR for each classification type, showing that low accuracy for WBFM is the main issue with achieving higher accuracy. The next most challenging pair were QAM16 and QAM64, but it was not as much an issue as the WBFM/AM-DSB.

My next step in improving the model is to create an ensemble between the original model and another that will differentiate WBFM/AM-DSB from others.

SOURCE CODE LOCATION

All the code to generate, train, and evaluate the models and the results for the RNN section used git for source control and can be accessed at this repository link.

https://github.com/rameyjm7/ML-wireless-signal-classification

4. CONCLUSION

All models fell short in SNRs below 0 dBm, which is expected as the noise saturates the receiver. Several modulation types, such as AM-SSB and QAM64, were better at lower SNR detection but had poor classification overall at the lower SNRs. A decision tree model showed ~10% better accuracy at the lower SNRs[2].

Once we crossed the 0 dBm threshold, the true power of ML shone through, as seen in the Peak Accuracy scores in Table 7. Training time (indicating the model's computational complexity) did not indicate better accuracy. This shows that simple models can present appealing results when tuned to the right problem space. However, the best overall and peak accuracy was achieved with RNNs but was the most computationally costly. This is likely because of the memory aspect of the network. Sharp feature changes, such as phase change in modulated signals, would throw off any averaging but can be singled out in memory-type models.

An ensemble of multiple models would provide the best result. Results indicate that using decision trees[2] for SNRs < 0 and RNN for SNRs > 0 would create a state-of-the-art prediction model for wireless RF signals. If a model could decide between WBFM/AM-DSB/Others, using it with the LSTM RNN would provide a superior result since this area had the most confusion. See more details in section 5 below.

Model	Training Time[1]	Accuracy (Peak) %
SVM	140 minutes	50% (80%)
XGBOOST	87 minutes	51.2% (81%)
DECISION TREE [2]	25 minutes	64% (94%)

RNN	960 minutes	68% (94.5%)

Table 7: Model Accuracy Comparison

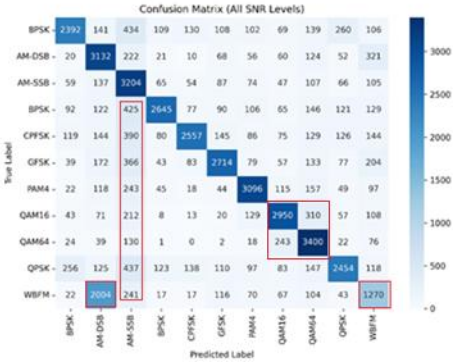
NOTES:

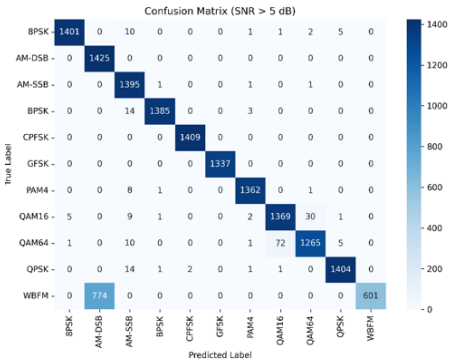
[1] *The models were all trained using separate and different hardware. The relative time between models is meaningless; however, the absolute time can indicate the computational power required.*

[2] *The Decision tree model only used 5 modulation types instead of the total 11 types.*

5. WHAT WE WOULD DO NEXT…

In all the models, the areas in which they had the most difficulty were WMFM/AM-DSB, AM-SSB, and QAM16/QAM64. In the LSTM RNN model, WBFM was classified incorrectly as AM-DSB more than it was classified as WBFM. The AM-SSB column shows that many signals are confused with AM-SSB, mainly when there is a lower SNR. The second figure shows how much AM-SSB confusion improves as the SNR exceeds 5 dB. Even when the SNR is high, the WBFM signals often is incorrectly classified as AM-DSB.





We should address the deficiency with WBFM/AM-DSB to improve the model's accuracy. Here are our suggested next steps:

- Create an ensemble model that uses the original LSTM RNN and a second model to help decide between WBFM/AM-DSB. A prediction can be made using the original model, and if it predicts AM-DSB, we will use the second model to determine if it's AM-DSB or WBFM, leading to a superior result over the original model
- Train the original model with more data on AM-DSB and WBFM, especially using actual data collected from SDRs that have noise in them
- Implement denoising algorithms to improve accuracy at lower SNRs; see ANR in this paper (Bai, Huang, & Yang, 2023)
- Add more layers to the original model and transfer the weights. A more complex model could capture the subtle patterns necessary to discern AM-DSB from WBFM. Also, we could consider using Bidirectional LSTM layers instead to capture information in both directions in time

If given more time, we would train a model using the newer DeepSig datasets, such as RADIOML 2018.01A, which contains "24 digital and analog modulation types." (DeepSig, 2024).

The last nice to have is a comparison of how long it takes for each of these models to make an inference on the same hardware, so we can make a comparison to see if any of the models can be used in a real-time scenario on live data.

6. REFERENCES

Bai, H., Huang, M., & Yang, J. (2023, October 1). An efficient Automatic Modulation Classification method based on the Convolution Adaptive Noise Reduction network. *ICT Express*, pp. 834-840. Retrieved from https://www.sciencedirect.com/science/article/pii/S2405959522001515

Bhuiya, S. (2020, October 31). *Disadvantages of CNN models*. Retrieved from Medium: https://sandeep-bhuiya01.medium.com/disadvantages-of-cnn-models-95395fe9ae40

Choudhary, J. (2023, September 20). *Mastering XGBoost: A Technical Guide for Machine Learning Practitioners*. Retrieved from Medium: https://medium.com/@jyotsna.a.choudhary/mastering-xgboost-a-technical-guide-for-intermediate-machine-learning-practitioners-f7ad167c6865

DeepSig. (2024, October 1). *Datasets*. Retrieved from DeepSig AI: https://www.deepsig.ai/datasets/

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification (2nd ed.).* New York: John Wiley & Sons, Inc.

Flowers, B., & Headly, W. C. (2024, September 24). *Radio Frequency Machine Learning (RFML) in PyTorch*. Retrieved from Github: https://github.com/brysef/rfml

GeeksForGeeks. (2024, October 10). *Support Vector Machine (SVM) Algorithm*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Gish, H. (2006, August 06). A probabilistic approach to the understanding and training of neural network classifiers. *International Conference on Acoustics, Speech, and Signal Processing.* Albuquerque: IEEE Xplore. Retrieved from IEEE Explore: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=115636

Google Developers. (2022, September 28). *Gradient Boosted Decision Trees*. Retrieved from Google Developers.: https://developers.google.com/machine-learning/decision-forests/intro-to-gbdt

GridSearchCV. (n.d.). Retrieved from Scikit Learn: https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning.* Springer.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation.* Prentice-Hall, Inc.

How to tune a Decision Tree in Hyperparameter tuning. (2024, April). Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/how-to-tune-a-decision-tree-in-hyperparameter-tuning/

Hutomo, I. S. (2021, October 8). *https://github.com/alexivaner/Deep-Learning-Based-Radio-Signal-Classification*. Retrieved from Github: https://github.com/alexivaner/Deep-Learning-Based-Radio-Signal-Classification

IBM. (2024, October 9). *What is a decision tree?* Retrieved from IBM: https://www.ibm.com/topics/decision-trees

IBM. (2024, October 9). *What is random forest?* Retrieved from IBM: https://www.ibm.com/topics/random-forest

Kızılırmak, S. (2023, February 11). *Rectified Linear Unit (ReLU) Function: Understanding the Basics*. Retrieved from Medium: https://medium.com/@serkankizilirmak/rectified-linear-unit-relu-function-in-machine-learning-understanding-the-basics-3770bb31c2a8

Keylabs AI. (2024, September 13). *K-Nearest Neighbors (KNN): Real-World Applications*. Retrieved from Keylabs AI: https://keylabs.ai/blog/k-nearest-neighbors-knn-real-world-applications/

O'Shea, T. J., Roy, T., & Clancy, T. C. (2018). Over-the-Air Deep Learning Based Radio Signal Classification. *IEEE Journal of Selected Topics in Signal Processing*, 168-179.

O'Shea, T., & West, N. (2016, September 6). *Radio Machine Learning Dataset Generation with GNU Radio.* Retrieved from Gnuradio: https://pubs.gnuradio.org/index.php/grcon/article/view/11/10

Qiu, Y., Zhang, J., Chen, Y., & Zhang, J. (2023, April 20). *Radar2: Passive Spy Radar Detection and Localization Using COTS mmWave Radar*. Retrieved from IEEE Explore: https://ieeexplore.ieee.org/document/10105863

Roy, D. (2020). *MACHINE LEARNING BASED RF TRANSMITTER CHARATERIZATION IN THE*. Retrieved from Northeastern University College of Engineering: https://www1.coe.neu.edu/~droy/Doctoral_Dissertation_Debashri_Roy.pdf

Virginia Tech. (2024, October 9). ECE5424 LN12.pdf. Blackburg, VA, United Stated of America.

Viso.AI. (2024, October 9). *Ensemble Learning: A Combined Prediction Model (2024 Guide)*. Retrieved from Viso.AI: https://viso.ai/deep-learning/ensemble-learning/