Chapter 1

Outline

1.1 Introduction

Due to advancements in various technical sectors, the requirement for varied antenna design has increased exponentially. Considering IoT, for instance, different device dimensions demand antennas with precise parameters; thus these antennas must be fabricated with high precision keeping in account an optimized performance. These days optimization is carried out mostly with traditional methods (primarily using EM simulators) which have critical down points like high compute intensity, less accurate results. To optimize an antenna having several parameters which have to be optimized (eg. 3D Antennas) [1] these traditional methods are proved impractical. To tackle these design challenges Machine Learning Domain is getting considered a suitable tool for this use case. In this communication we are trying to do a thorough study of a CDRA (Cylindrical Dielectric Resonator Antenna) to obtain the most optimal Machine Learning technique applicable, and the study can then be extended to another antenna designs also.

1.2 Objective

The Antenna Design optimization process can leverage some of the positives offered by Machine Learning techniques, this could tackle several issues related to design optimization. The study throws some light on this particular use case of ML with Antennas and will try to put forward different methodologies through comparison. We will identify the effectiveness and accuracy of various Machine Learning Techniques on our antenna and once we have obtained the techniques and have the relational model, the output for any data model can be predicted while using the same data set for multiple goals.

Chapter 2

Literature Review

2.1 CDRA

CDRA stands for Cylindrical Dielectric Resonator Antenna were, as evident from the figure, cylindrical attributes to its shape. It is utilized more often than not at frequencies greater than or equal to microwave frequencies (> 1 GHz) and thus acts like a radio antenna. The Dielectric Resonator is made up of ceramic, which is the dielectric here, and is placed on a metal surface, ground plane. The choice of ceramic is due to the requirement of a material with high quality factor (20 < Q < 1000) and having low loss property. Ideally, we require the dielectric constant between 10 < εr < 100 so that we can trade off with other factors for various applications. DRA's can perform the transformation of guided waves into unguided waves (RF signals) using the radiating resonators. The size of DRA is inversely proportional to (εr)^½ and hence can be contained in a small space if the chosen material is of high dielectric constant. Courtesy of this , DRA's are now looked upon to be the possible solution to the requirement of smaller spacecraft equipment in future. Also the choice of low-loss material and the absence of any conducting material results in excellent radiation efficiency ,thus making it optimal for applications operating at high frequencies.

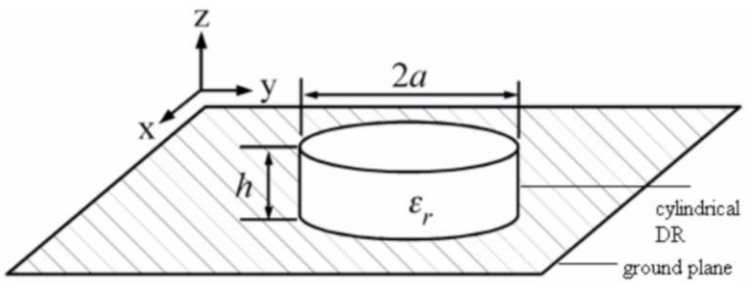


Fig 2.1: Antenna configuration of cylindrical DRA

2.2 Machine Learning

A domain that recently has proved very useful with decision making and analytics in a wide variety of applications. With techniques and optimization algorithms we can find the hidden mathematical relations in the input and output data considering which we can make future predictions. Other heuristic optimization techniques could also be used like genetic algorithms and particle swarm optimization but these algorithms search for the optimal solution by analyzing the output on individual data points and generating new and possibly better search directions until one global maxima or minima is identified. And once a model is trained the same data set gets beneficial for multiple output goals. Though some studies regarding ML implementation have been conducted as mentioned in [2], a comparative study could be required to get a clear picture. Our work tries to support ML as a promising choice to include in antenna design, and later on, this same idea can be extended to complicated designs even. Below mentioned 5 ML techniques have been studied, implemented, and analyzed thoroughly.

Broadly classifying there are two types of ML techniques - supervised learning and unsupervised learning. The major difference between the two techniques is the use of labeled data to train. Supervised techniques use labeled data and the model has the job of finding the mapping function. Whereas unsupervised learning uses unlabeled data and the model here itself has to infer patterns from the data set. Unsupervised learning models find the similarities and kind of cluster or group them together . It can be used to gather important insights from the dataset . However, since here we are trying to build a model which can predict the output for any data provided , we had to go forward with supervised learning models. Unsupervised learning is more close to Artificial Intelligence compared to supervised learning as it learns similarly as a child by observing its surroundings. But supervised learning is optimal for applications requiring an accurate response and therefore we can justify the use of it in our scenario.

2.2.1 ANN (Artificial Neural Network)

The technique comes under Deep Learning, a subset of Machine Learning which is inspired by biological brain functionality. It consists of a group of artificial neurons which process info over intercommunication. Drawing an analogy with our biological neural network - dendrites represent inputs , cell nucleus represents nodes , synapse represents weights and axon represents output. This tries to follow up our brain striving for a similar ability to understand things and carry out decision making.

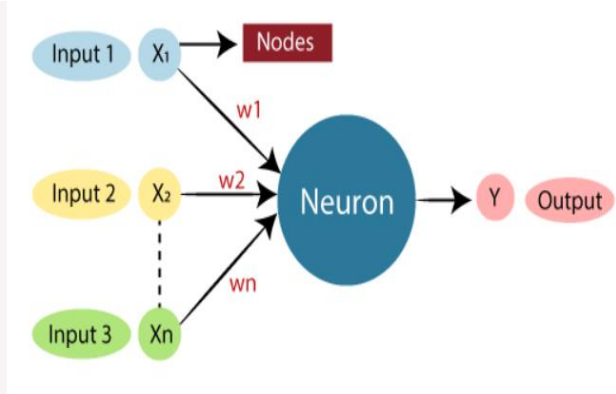


Fig 2.2: The typical Artificial Neural Network

Certain common ANN structures include MLP (Multi-Layer Perceptrons) which consists of 3 layers: Input, Output, and Hidden layer. Training for small and medium-sized patterns can be done with algorithms like LM (Levenberg Marquardt)[3], back-propagation, and delta-bar delta. The prowess to learn and develop complex relationship models gives MLP's an edge over others in the preference order.

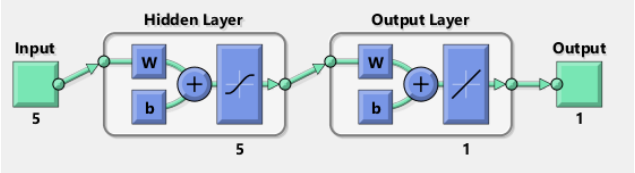


Fig 2.3: ANN Architecture based on MLP

2.2.2 Decision Tree

A decision-making technique that uses a tree-based model, the node, branch, and leaves respectively represent attribute test, outcome, and class label. It is generally represented as a flow chart and is a popular tool for classification and prediction applications. Decision Trees can be classified on the basis of type of target variable - Categorical or Continuous. The nodes here also allow splitting into two or more sub nodes. In the decision tree each node is a classifier and decides which edge the entity will descend to based on the attribute of the node. This is a recursive process and is performed all the way from the root node till the terminal node - which is the final classification of the entity.

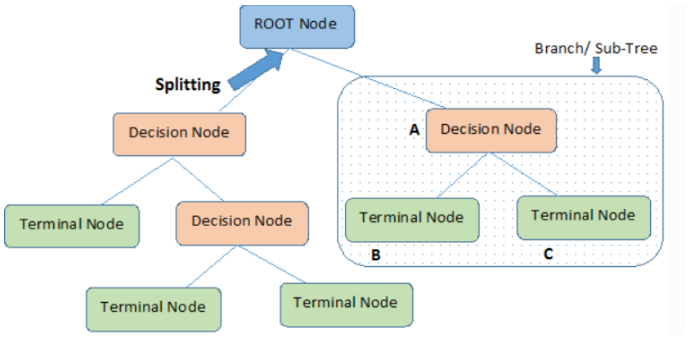


Fig 2.4: Decision Tree structure

However, this model can have prediction drawbacks in case of high noise in the training data. Moreover, a small change in the training data could change the structure of the optimal tree which can result in inaccuracy.

2.2.3 Random Forest

Based on the decision tree Random Forest algorithm is widely used for classification and regression. It basically consists of a number of decision trees made on different data subsets, an average is taken. It's an extension of the Bagging (bootstrapping + aggregation) technique and is based on the concept of ensemble learning.

2.2.4 KNN (K-Nearest Neighbour) [4]

A supervised and insurance-based machine learning technique that makes predictions considering the similarities with the training data set. It takes a new data

point and then using its algo places it into the category which has the most similarity out of all available categories. It is not an immediate learner from the training sets , rather it conserves all the data and takes action at the time of classification. There is no definite way for the determination of K - it can be perfected using hit and trial only. After selecting K , we need to figure out the euclidean distance of its neighbors and proceed to picking out the K closest ones to the new data point. Once we have these K neighbors , make a count of the number of data points falling in each category. The category with the maximum number of closest data points will have the new data point assigned to it.

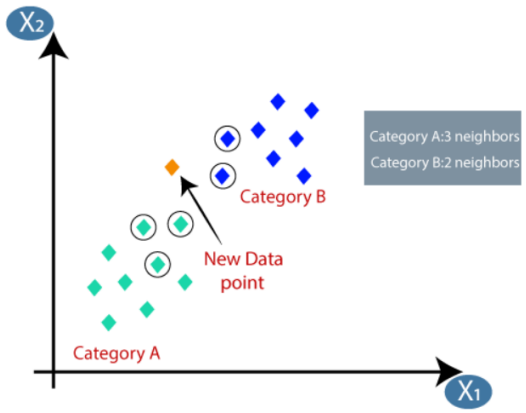


Fig 2.5: Categorising a new data point by KNN

The value of k is considered 5 in our antenna's case as it gives the minimum cross-validation error.

2.2.5 XGBoost (Extreme Gradient Boost) [5]

Decision tree-based algorithms work on ensemble learning and gradient boosting. It is an optimized distributed gradient boosting library that is highly efficient, flexible, portable, and provides a parallel tree boosting that solves many data science problems in a fast and accurate way. It is widely used as an alternative to ANN in applications involving small/medium-sized structured/tabular data.

Chapter 3

Implementation

3.1 HFSS Model

High-Frequency Structure Simulator, is an industry-standard software tool distributed by Ansys that is used for antenna design, complex radio frequency electronic circuit elements including transmission lines, filters, etc. [6] The reference cylindrical DRA is designed and tested in an HFSS environment. The performance of a CDRA depends on several design parameters, for our study we have considered the *h* and *r* (height and the radius) of the cylindrical substrate as variable parameters. The frequency band is 1-5 GHz, and we have considered the Figure of merit as S11 value generated with variable *h*, *r*, and *f* values (*f* = operation frequency). S11 is the reflection coefficient which determines the power which is reflected from the antenna. This parameter could determine the antennas performance in major cases and thus has been selected as an FOM.

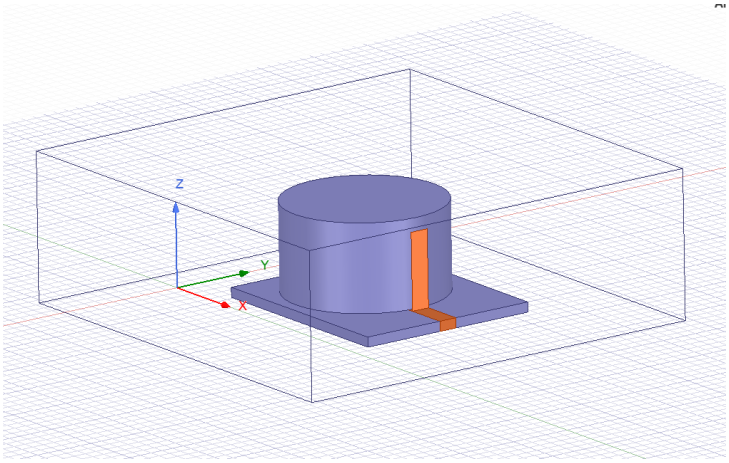
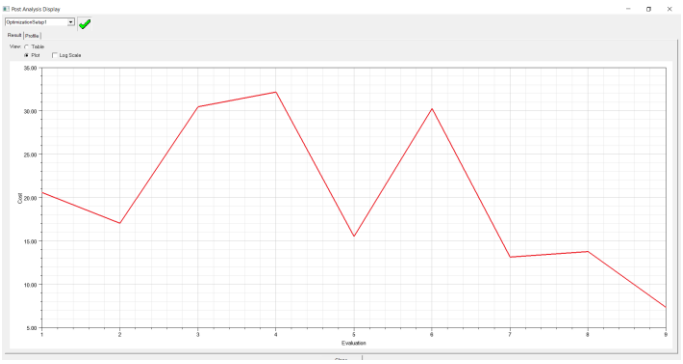


Fig 3.1: Designed HFSS model of CDRA

3.2 Dataset

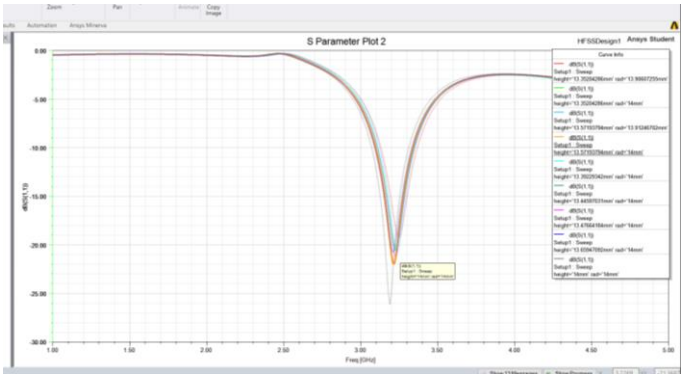
For the training purpose, the model requires proper data to get an accurate result. On increasing the training samples the model prediction gets better. For the dataset generation, HFSS Parametric is used. But we need a range and step size for *h,r* first, and for this firstly, we will try to optimize the design with an HFSS

optimizer. Here our optimization goal is to get the dB(S11) value <= -20 dB in the frequency band of 1-5Ghz. There are options for several optimizers, but for this particular use case, we have used the Quasi-Newton *(Gradient)* optimizer. On optimization with 9 iterations, we got the optimized value of *h, r*.



The graph shows the values of cost for all 9 iterations. The parameter values of the iteration with the least value of cost are considered optimal.

Fig 3.2: Optimization iterations cost values.



9 S11 sweep graphs are overlapped representing the dB S11 values for different frequencies. Color

Fig 3.3: S11 sweep for each iteration

With the optimized values, we can proceed with HFSS parametric. By setting a suitable range and step size of *h* and *r*, we can generate a dataset that can be exported in any suitable format. We have considered the .csv format as the data can be visualized on any spreadsheet and can be easily parsed. The dataset .csv file has 4 columns, *h*, *r*, *freq*, *S11* with around 50K rows of samples.

3.3 Training

The generated dataset in .csv format is first parsed and input/output data frames are made with it. Dataframes are then distributed into training and test sets.

The training set will be used for training each model, which will then be making predictions for test sets. The accuracy of the test-set predictions can then be analyzed to determine the feasibility of the model. This distribution will be random with test-data size being 30% of the total set.

```
# Reading data from csv file

dataset = pd.read_csv('../DataSets/data_set.csv')

x = pd.DataFrame(dataset.iloc[:, 0:3].values)
y = dataset.iloc[:, 3].values

# Generating training and test data sets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 123)
```

The python code of each model's definition and training is mentioned in the following section.

KNN

```
# KNN

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train, y_train)
```

Used the KNeighborsRegressor class from the *sklearn* library, k= 5 is set for the regressor.

ANN

```
# Artificial NN

classifier = Sequential()

# Input Layer
classifier.add(Dense(units = 6, activation = 'relu', input_dim = 3))

# 1st Hidden Layer
classifier.add(Dense(units = 8, activation = 'relu'))

# 2nd Hidden Layer
classifier.add(Dense(units = 3, activation = 'relu'))

# Output Layer
classifier.add(Dense(units = 1, activation = 'linear'))
```

The ANN model is made using the *Keras* library, there is one input layer with the dimension of 3 as we have three input features for the model with a 'relu'

activation function. 2 hidden layers are added also having ‘relu’ activation function. The output layer of unit=1 provides the final result.

```
# Training

classifier.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics = ['accuracy'])

history=classifier.fit(x_train, y_train, batch_size = 10, epochs = 100, validation_data=(x_test,y_test))
```

Used the *Adam* optimizer and *mean_squared_error* as the loss for the model. The training is carried on for 10 epochs with a batch size of 10. Training of an ANN model is the time-taking process for machines with less compute ability.
The training process was also analyzed to have a clear picture of how the model trains itself with the input training data set.

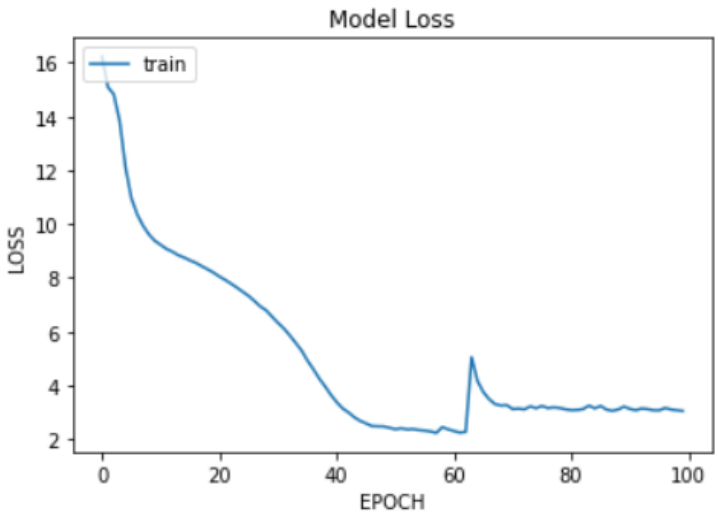


Fig 3.4: Loss vs Epoch graph for ANN

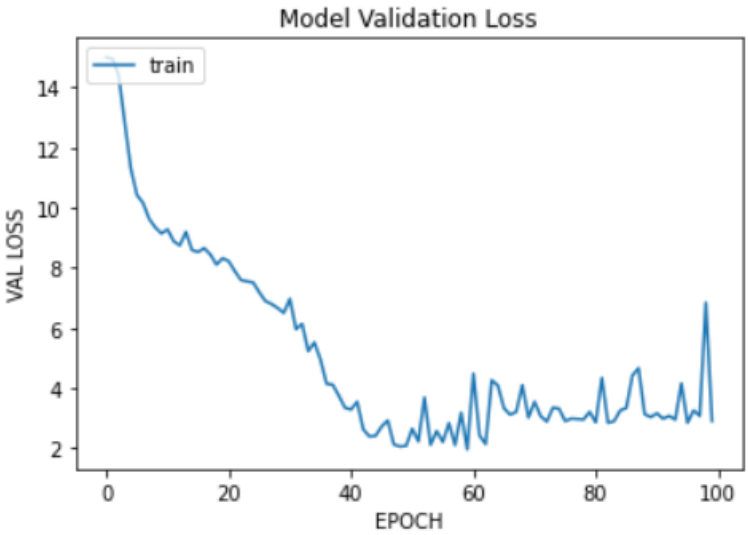


Fig 3.5: ANN Model validation loss vs Epoch graph

Decision Tree

```
# Decision Tree

from sklearn.tree import DecisionTreeRegressor
DTReg = DecisionTreeRegressor()
DTReg.fit(x_train,y_train)
```

Used *sklearn* DecisionTreeRegressor class.

Random Forest

```
# Random Forest

from sklearn.ensemble import RandomForestRegressor
RFreg = RandomForestRegressor(n_estimators = 10, random_state = 0)
RFreg.fit(x_train,y_train)
```

Used *sklearn* RandomForestRegressor class, the value of the estimators is set to 10.

XGBoost

```
# XGBoost
import xgboost as xgb
xgb_reg = xgb.XGBRegressor(n_estimators=100, objective='reg:linear', seed = 123)

# Training
xgb_reg.fit(X_train, y_train)
```

The xgb package is imported and used to create the XGBRegressor.

Chapter 4
Analysis

4.1 Prediction

By now all the models which we are considering are trained with the generated dataset. Each of these models can now predict the values of S11 for a given antenna parameter in a particular frequency band. As we discussed earlier, it is better to test the model with a data-set with which it is not trained, to understand it's precision better and analyze the prediction accuracy. If the predictions made are not up to the mark then any of the following could be the case.

1. The training data could be inappropriate or of poor quality.
2. Model might require hyperparameter tuning or some cross-validation.
3. Choice of the input features could be wrong.

There might be other reasons even, but for our use case, we tried working on the above-mentioned issues in case of the poor model prediction.

4.2 Accuracy

On comparing the predicted outputs with the actual outputs we obtained the *r2 score, mean absolute error, mean squared error, max error.* Table 3.1 contains these accuracy metrics for predictions made by each ML technique. To obtain these values we used the methods provided by *sklearn.metrics* . These metrics provide us a picture of the prediction effectiveness and helps in improving the training dataset or tuning the model.

	R2 Score	Mean Absolute Error	Mean Squared Error	Maximum Error
ANN	0.45	1.70	11.04	40.90
KNN	0.98	0.12	0.33	23.53
Decision Tree	0.97	0.18	0.59	22.61
Random Forest	0.99	0.12	0.24	24.21
XGBoost	0.98	0.20	0.37	20.26

Table 4.1: Prediction metrics for each technique

Following are a few graphs that visualize the relations between the predicted and the actual values.

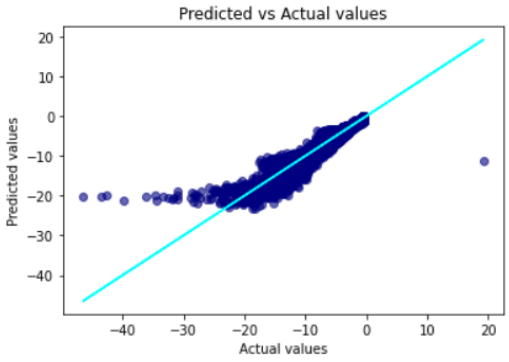


Fig 4.1 Predicted vs Actual values for *ANN*

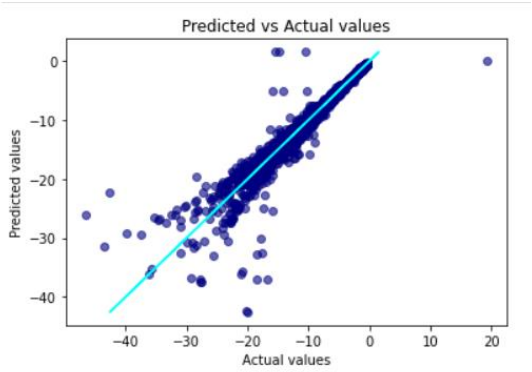


Fig 4.2 Predicted vs Actual values for *Decision Tree*

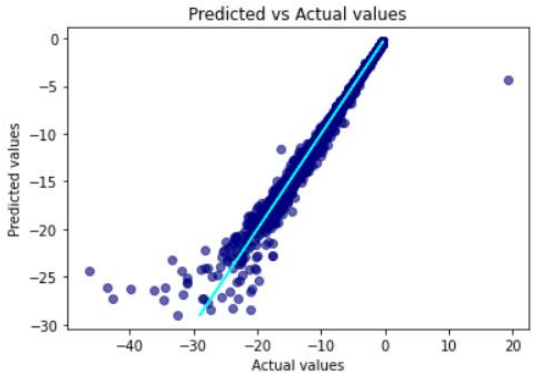


Fig 4.3 Predicted vs Actual values for *KNN*

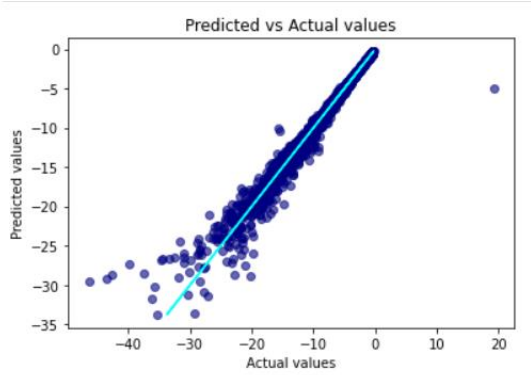


Fig 4.4 Predicted vs Actual values for *Random Forest*

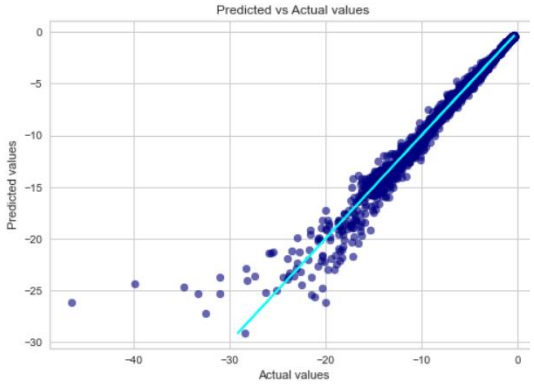


Fig 4.5 Predicted vs Actual values for *XGBoost*

4.3 Comparison

In the accuracy analysis we took a fraction of the data-set (the test-set) to compare the predictions, but now we will carry on the comparison of the HFSS results with the model predicted values. The data which we will use in this section will be randomly generated by uniform random distribution using the *NumPy* library in python. The range and step size values that we used for *h* and *r* in HFSS parametric will only be used here to generate a random data frame with 5 samples. The values of antenna parameters for a given frequency are set for the HFSS model and a simulation is carried out to get the value of S11 which can then be compared with the predicted values. For each data sample, an error is calculated to get an idea of the prediction accuracy considering the HFSS value as the actual value.

Data Sample	Height (mm)	Radius(mm)	Frequency (GHz)	HFSS S11
D1	14.10	14.55	2.42	-0.39
D2	13.04	13.30	1.84	-0.39
D3	13.19	13.74	4.70	-5.44
D4	14.42	13.12	2.41	-0.78
D5	13.86	14.57	4.37	-4.83

Table 4.2 5 Random data samples.

	D1	D2	D3	D4	D5
Prediction	-0.42	-0.40	-5.55	-0.66	-4.83
Error	0.08	0.03	0.02	0.15	0

Table 4.3 Prediction of KNN

	D1	D2	D3	D4	D5
Prediction	-0.63	-0.52	-5.90	-0.75	-5.55
Error	0.61	0.33	0.08	0.03	0.14

Table 4.4 Prediction of ANN

	D1	D2	D3	D4	D5
Prediction	-0.42	-0.40	-5.48	-0.70	-4.83
Error	0.08	0.03	0.01	0.10	0

Table 4.5 Prediction of Decision Tree

	D1	D2	D3	D4	D5
Prediction	-0.41	-0.39	-5.62	-0.73	-4.80
Error	0.05	0	0.03	0.06	0.01

Table 4.6 Prediction of Random Forest

	D1	D2	D3	D4	D5
Prediction	-0.43	-0.40	-5.58	-0.69	-4.80
Error	0.10	0.02	0.03	0.11	0.01

Table 4.7 Prediction of XGBoost

Conclusion

In this study we went through different Machine Learning techniques which can be leveraged into antenna design optimization. A brief description of each technique was presented followed by their use cases and application. The reference CDRA antenna was then explained with its design characteristics. Later with the HFSS design we tried to optimize the parameters for a specific goal with optimetrics and then with parametrics we generated the training and test data set.

Above mentioned 5 ML techniques were used to firstly train the model followed by tuning each model by analyzing the prediction metrics. Finally for few data samples the predicted values were compared with the HFSS calculated values. This brought us to infer that the prediction performance of all these techniques were quite similar except ANN, despite being from DL domain it's model didn't show satisfactory outputs due the small-sized data set and less optimized hyperparameter tuning. With this our study shows the potential and versatility of ML techniques in automated antenna design.

Future Scope

We carried on our study on a CDRA reference antenna, but the same techniques can be further leveraged by the automated design process of more complicated antenna designs like 3D antennas and a versatile for a versatile range of antennas required in different applications like IoT. As of now just one antenna parameter which is S11 is considered for optimization but probably for better results and optimization other parameters and aspects like the radiation pattern and beam tilting should also be taken into account. A detailed study on several other antenna parameters and their effect in its optimization is something which can be carried on in further iterations by considering and leveraging the results of our study and analysis. Also with automated antenna optimization the fabrication of antenna for different applications could also be done in an automated fashion. These fabricated physical antennas can be used to carry out practical observations to further study the performance of these antennas in real application rather than in HFSS simulation. Training and prediction of these ML models with physical antennas may produce better prediction results as several practical phenomena are taken into account in this case.