All Sciences Proceedings

2nd International Conference on Scientific and Academic Research

March 14-16, 2023 : Konya, Turkey

ICSAR

All Sciences Proceedings

http://as-proceeding.com/

SPECTRUM OCCUPANCY PREDICTION USING A MODIFIED XGBOOST MACHINE LEARNING ALGORITHM

Temitayo Ayodeji OYEWO[1*], Umar Suleiman DAUDA[1] and Abubakar Saddiq MOHAMMED[2]

[1]*Department of Electrical/Electronic Engineering, Federal University of Technology Minna, Nigeria.*
[2]*Department of Telecommunication Engineering, Federal University of Technology Minna, Nigeria.*

** (teteyeah4u@yahoo.com) Email of the corresponding author*

Abstract – There is an alarming rate of growth in the usage of spectrum, where some of the allocated spectra is fully engaged while others are sparsely utilized. This gives attention to the use of cognitive radios where the Primary users can maximize the available spectrum holes alongside the secondary users. The challenge of using cognitive radio technology is interference which is a factor that causes a delay in the handoff time. XGBoost alongside other regression Machine Learning (ML) Algorithm such as linear Regression, Lasso Regression, Ridge regression, and the random forest was used to train and predict the dataset gotten from sensing the spectrum at a location called Morris Fertilizer within the environs of Minna, Niger state. Linear regression, Random forest regression, XGBoost, Ridge, and Lasso have been used for the prediction of cognitive radio frequencies based on 10 power features.
The linear Regression, Ridge and Lasso gave the same level of accuracy of 6.39%, while Random forest gave an accuracy of 54.65%Xgboost gave the best performance with an accuracy level of 96.85%, thus boosting algorithm shows a high level of prediction ability.

*Keywords**:** Cognitive Radio, Spectrum Sensing Energy Detector, Machine Learning Algorithm.*

I. INTRODUCTION

The current exponential growth in technological advancements has led to an increased demand for wireless devices. This surge in demand, coupled with the static management of the radio spectrum, has resulted in a shortage of available spectrum. This shortage is due to the inefficiency of the static management of the spectrum, which is unable to accommodate the growing number of wireless devices. [1] Observed that most current wireless communication systems are based on the concept of fixed frequency allocation. The quality-of-service (QoS) criteria of the radio spectrum are met while consuming less energy thanks to cognitive radio (CR) technology [2]. Machine learning (ML) algorithms are used to mimic human intelligence and they can make decisions without explicit programming [3]. The use of ML algorithms in CR technology improves both the efficiency and effectiveness of spectrum utilization [4] & [5]. [6] Worked to minimize the delay that occurs during spectrum handoff. To increase the time and energy efficacy of Spectrum Sensing [7] reduced the number of channels used for sensing with spectrum prediction. [8] Shows that XGBoost outperforms a number of other well-known machine learning methods such as the linear Regression, Lasso Regression, Ridge regression and random forest on a range of datasets from a spectrum sensing of frequency range between 80MHz to 2GHz. The analysis carried out shows the XGBoost ML algorithm has the ability to work better with a large dataset, whereas the other regression ML algorithm seems to be limited.

II. MATERIALS AND METHODS

The system design consists briefly of two major stages: (a) the spectrum sensing stage. (b). the data analysis with the ML algorithm for the prediction of spectrum holes (XGBoost). To collect data to run analysis, a high-gain outdoor antenna with an acceptable capability frequency range of 80MHz to 2GHz was connected to the spectrum analyzer to capture electromagnetic signals.

The components used for the collection of data are as follows:

Antennae, which enables the spectrums analyzer to capture electromagnetic waves. Table 1 shows the frequency band and allocations that were analyzed

An Agilent spectrum Analyzer N9342C, A personal computer, and a backup power supply (generator) to sustain the sensing process for the period of the exercise. The Global Position System (GPS) was used to determine the location where the exercise was carried out. Figure 1 shows a block diagram of components. The dataset collected as CSV files at collated in a spreadsheet and imported into the Python 3.0 Jupyter.

The dataset was checked to know the type if it was a discreet or classification (yes or no) problem.

The dataset from the spectrum sensing exercise is continuous. Which means it is a regression problem. The correlation was visualized using the heat map representation as shown in figure 6. After this, the correlation between the dataset was checked. The dataset was divided into training and testing datasets. To train the dataset 80% of the dataset was used to train while the other 20% was used to test the functionality of the prediction exercise. The dataset was grouped into the predictor and the target (what is being predicted). The target is the frequencies of the analyzed spectrum in Hz and the predictors are the powers in -dB. Using powers to predict frequencies. The Linear regression, Random forest regression, Ridge and Lasso was trained and tested with the dataset until the XGBoost gave a high percentage accuracy level and prediction.

Table 1 Frequency band and allocations analyzed

FREQUENCY RANGE	SERVICE
88 - 108 MHz	FM Radio Broadcast
108 - 200 MHz	Navigation (VOR), Aeronautical Navigation, Military Land Mobile, Amateur Radio , Land Mobile (VHF)
200 - 400 MHz	Land Mobile (VHF), Fixed & Land Mobile Military Aviation
400 - 600 MHz	Land Mobile (UHF), Amateur Radio, Land Mobile (UHF), TV Channels 14 – 20, TV Channels 21 – 36
600 - 960 MHz	Land Mobile (UHF), TV Channels 38 – 51, Public Safety & Commercial Wireless
960 – 2000 MHz	Microwave Communication, GPS, Bluetooth, Wi-Fi

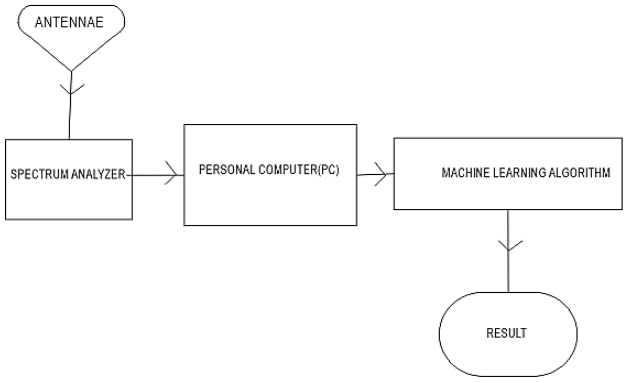

Fig 1 A block diagram of components

III. RESULT

The sensing and prediction of the spectrum were carried out in stages in which the first stage was the collection of the dataset in the field and stage two was to build a suitable model for the dataset acquired.

```
In [21]: Train_X, Test_X, Train_Y, Test_Y = train_test_split(X_std, y, train_size = 0.8, test_size = 0.2, random_state = 1)
         print('Original set --->',X.shape, y.shape,
               '\nTraining set --->', Train_X.shape, Train_Y.shape,
               '\nTesting set --->',Test_X.shape, Test_Y.shape )

         Original set ---> (461, 11) (461, 1)
         Training set ---> (368, 11) (368, 1)
         Testing set ---> (93, 11) (93, 1)

In [22]: linReg = LinearRegression()
         linReg.fit(Train_X, Train_Y)
         pred_train_lin = linReg.predict(Train_X)

In [23]: train_acc = linReg.score(Train_X, Train_Y)
         print("The train accuracy score of the LinearRegression is: {0:.2f}%".format(train_acc *100))
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_lin)))

         The train accuracy score of the LinearRegression is: 6.39%
         539168604.5971681
```

```
In [21]: Train_X, Test_X, Train_Y, Test_Y = train_test_split(X_std, y, train_size = 0.8, test_size = 0.2, random_state = 1)
         print('Original set --->',X.shape, y.shape,
               '\nTraining set --->', Train_X.shape, Train_Y.shape,
               '\nTesting set --->',Test_X.shape, Test_Y.shape )

         Original set ---> (461, 11) (461, 1)
         Training set ---> (368, 11) (368, 1)
         Testing set ---> (93, 11) (93, 1)

In [22]: linReg = LinearRegression()
         linReg.fit(Train_X, Train_Y)
         pred_train_lin = linReg.predict(Train_X)

In [23]: train_acc = linReg.score(Train_X, Train_Y)
         print("The train accuracy score of the LinearRegression is: {0:.2f}%".format(train_acc *100))
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_lin)))

         The train accuracy score of the LinearRegression is: 6.39%
         539168604.5971681
```

Figure 2 Linear Regression Test

```
In [24]: from sklearn.ensemble import RandomForestRegressor
         rf = RandomForestRegressor()
         #randomized search cv
         n_estimators = [int(x) for x in np.linspace(start = 100, stop =1500, num = 15)]
         #numbers of features to consider in every split
         print(n_estimators)
         max_features = ['auto', 'sqrt']
         #max nos. of levels in tree
         max_depth = [int(x) for x in np.linspace(5,20, num =5)]
         #min nos. of samples required to split a node
         min_samples_split = [2,5,10,15,100]
         ##min nos. of samples required at each leaf
         min_samples_leaf = [1, 2, 5, 10]

         [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500]

In [25]: from sklearn.model_selection import RandomizedSearchCV
         #various parameters that was taken
         random_grid = {'n_estimators': n_estimators,
                        'max_features': max_features,
                        'max_depth': max_depth,
                        'min_samples_split':min_samples_split,
                        'min_samples_leaf': min_samples_leaf}
         print(random_grid)

         {'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500], 'max_features': ['auto', 's
         qrt'], 'max_depth': [5, 8, 12, 16, 20], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

Figure 3 Random Forest

```
In [26]: rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, scoring = 'neg_mean_squared_error',
                                        n_iter = 100, cv = 5, random_state = 1)

In [27]: rf_random.fit(Train_X, Train_Y)

Out[27]: RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=100,
                            param_distributions={'max_depth': [5, 8, 12, 16, 20],
                                                 'max_features': ['auto', 'sqrt'],
                                                 'min_samples_leaf': [1, 2, 5, 10],
                                                 'min_samples_split': [2, 5, 10, 15,
                                                                       100],
                                                 'n_estimators': [100, 200, 300, 400,
                                                                  500, 600, 700, 800,
                                                                  900, 1000, 1100, 1200,
                                                                  1300, 1400, 1500]},
                            random_state=1, scoring='neg_mean_squared_error')

In [28]: pred_train_rf = rf_random.predict(Train_X)
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_rf)))
         train_score = r2_score(Train_Y, pred_train_rf)
         print("The train accuracy score (R2) of the Random Forest Regression is: {0:.2f}%".format(train_score *100))

         374461472.50481296
         The train accuracy score (R2) of the Random Forest Regression is: 54.85%

In [29]: from xgboost import XGBRegressor

In [30]: predictors = [x for x in Train_X.columns]
         xgb1 = XGBRegressor(
          learning_rate =0.1,
          n_estimators=1000,
          max_depth=5,
          min_child_weight=1,
          gamma=0,
          subsample=0.8,
          colsample_bytree=0.8,

          nthread=4,
          scale_pos_weight=1,
          seed=27)
         xgb1.fit(Train_X, Train_Y)

Out[30]: XGBRegressor(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=0.8, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.1, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=5, max_leaves=None,
                      min_child_weight=1, missing=nan, monotone_constraints=None,
                      n_estimators=1000, n_jobs=None, nthread=4, num_parallel_tree=None,
                      predictor=None, ...)
```

Figure 4 XGB Regressor

```
In [31]: pred_train_xg = xgb1.predict(Train_X)
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_xg)))
         train_score = r2_score(Train_Y, pred_train_xg)
         print("The train accuracy score (R2) of the xgboost Regression is: {0:.2f}%".format(train_score *100))

         98914030.93971506
         The train accuracy score (R2) of the xgboost Regression is: 96.85%

In [32]: from sklearn.linear_model import Ridge
         from sklearn.linear_model import Lasso

In [33]: rr = Ridge(alpha = 0.01)
         rr.fit(Train_X, Train_Y)

Out[33]: Ridge(alpha=0.01)

In [34]: pred_train_rr = rr.predict(Train_X)
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_rr)))
         train_score = r2_score(Train_Y, pred_train_rr)
         print("The train accuracy score (R2) of the Ridge Regression is: {0:.2f}%".format(train_score *100))

         539168621.5106899
         The train accuracy score (R2) of the Ridge Regression is: 6.39%

In [35]: las = Lasso()

In [36]: las.fit(Train_X, Train_Y)
         pred_train_las = las.predict(Train_X)
         print(np.sqrt(mean_squared_error(Train_Y, pred_train_las)))
         train_score = r2_score(Train_Y, pred_train_las)
         print("The train accuracy score (R2) of the Lasso Regression is: {0:.2f}%".format(train_score *100))

         539168604.5971684
         The train accuracy score (R2) of the Lasso Regression is: 6.39%
```

Figure 5 The lasso Regression

IV. DISCUSSION

The dataset was collected based on laboratory experiments and using the excel tool we did some data preprocessing, before importing the data into python where further data preprocessing, exploratory data analysis(EDA), and feature engineering was carried out. Thus, we proceed to model training using some machine learning algorithm.

In this machine learning model, five different algorithms namely; Linear regression, Random forest regression, XGBoost, Ridge, and Lasso has been used for the prediction of cognitive radio frequencies based on 10 power features.

The linear Regression, Ridge and Lasso gave the same level of accuracy of 6.39%, while Random forest gave an accuracy of 54.65%Xgboost gave the best performance with an accuracy level of 96.85%, thus boosting algorithm shows a high level of prediction ability and can be recommended for the prediction of cognitive radio frequencies.

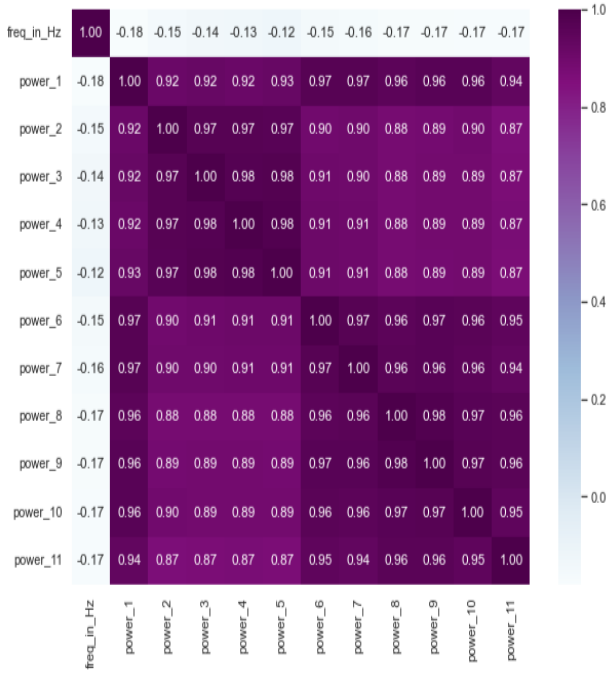

Figure 6 The Heat Map

V. CONCLUSION

Sensing the spectrum to know the level of occupancy, for a cognitive radio user to engage the spectrum holes,can have certain setback which the

handoff time leads to the interference of the secondary user with the primary user. This factor affects the quality of serviceof the spectra.

This brought about the idea to collate a set of data from the system which can be used to train a machine learning model to be able to predict the spectrum holes.

A well-built and tested model will increase the quality of service of the spectrum sensed, thereby giving better reliability for users when transmitting.

REFERENCES

[1] A. Nasser, H. A. H. Hassan, J. A. Chaaya, A. Mansour, and K. C. Yao, "Spectrum sensing for cognitive radio: Recent advances and future challenge," *Sensors*, vol. 21, no. 7, pp. 1–29, 2021, doi: 10.3390/s21072408.

[2] L. Chen, Y. Zhu, G. Papandreou, and F. Schroff, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation".

[3] I. J. Goodfellow, J. Pouget-abadie, M. Mirza, B. Xu, and D. Warde-farley, "Generative Adversarial Nets," pp. 1–9,2016.

[4] M. Saber, A. El Rharras, R. Saadane, A. Chehri, N. Hakem, and H. A. Kharraz, "Spectrum sensing for smart embedded devices in cognitive networks using machine learning algorithms," *Procedia Comput. Sci.*, vol. 176, pp. 2404–2413, 2020, doi: 10.1016/j.procs.2020.09.311.

[5] S. Solanki, V. Dehalwar, and J. Choudhary, "Deep learning for spectrum sensing in cognitive radio," *Symmetry (Basel).*, vol. 13, no. 1, pp. 1–15, 2021, doi: 10.3390/sym13010147.

[6] E. Alozie, N. Faruk, A. A. Oloyede, O. A. Sowande, and A. Lucky, "Intelligent Process of Spectrum Handoff in Cognitive Radio Networks," vol. 4, no. 1, pp. 205–218, 2022.

[7] A. Unadhye, P. Saravanan, S. S. Chandra, and S. Gurugopinath, "A Survey on Machine Learning Algorithms for Applications in Cognitive Radio Networks," *Proc. CONECCT 2021 7th IEEE Int. Conf. Electron. Comput. Commun. Technol.*, 2021, doi: 10.1109/CONECCT52877.2021.9622610.

[8] T. Chen and C. Guestrin, "XGBoost : A Scalable Tree Boosting System," pp. 785–794, 2016, doi: 10.1145/2939672.2939785.