EEE 498/591 Machine Learning Basics with Deployment to FPGAs

Final Project Report

Emre Lewis
Mishel Jyothis Paul
Sangeetha Thekkattuserry Ramaswami
Zachary Kalinowski

Table of Contents

Introduction

The project discusses a classification problem for water potability. The availability of safe drinking water is critical for humanity. Good drinking water keeps people healthy. The goal was to analyze a Kaggle dataset with nine features that were elements of water, and deploy to an FPGA. A stretch goal was to implement an ADC. Through this project, the team went through the process of machine learning. The team explored data analysis of a dataset that was not balanced or accurate. This project was an opportunity for the team to put the lessons learned in the class to the test for something important.

Data Analysis

Statistical analysis was performed on the Kaggle dataset to verify the quality of the inputs. Univariate analysis showed that the features were very well distributed. Results of bivariate analysis displayed low correlation between the input columns, and there were no columns fairly correlated with the required potability values. We also observed that the dataset had significant anomalies compared to real datasets, as values of many water quality metrics were beyond normal limits of ground water sources. It was concluded that the dataset is synthetic, and can be used primarily for analysis and demonstration purposes. We also found that the potability scores were not aligned with standards of usable water set by National Primary Drinking Water Regulations (NPDWR).
The imbalance of classes and high number of null values in the dataset made this problem challenging.

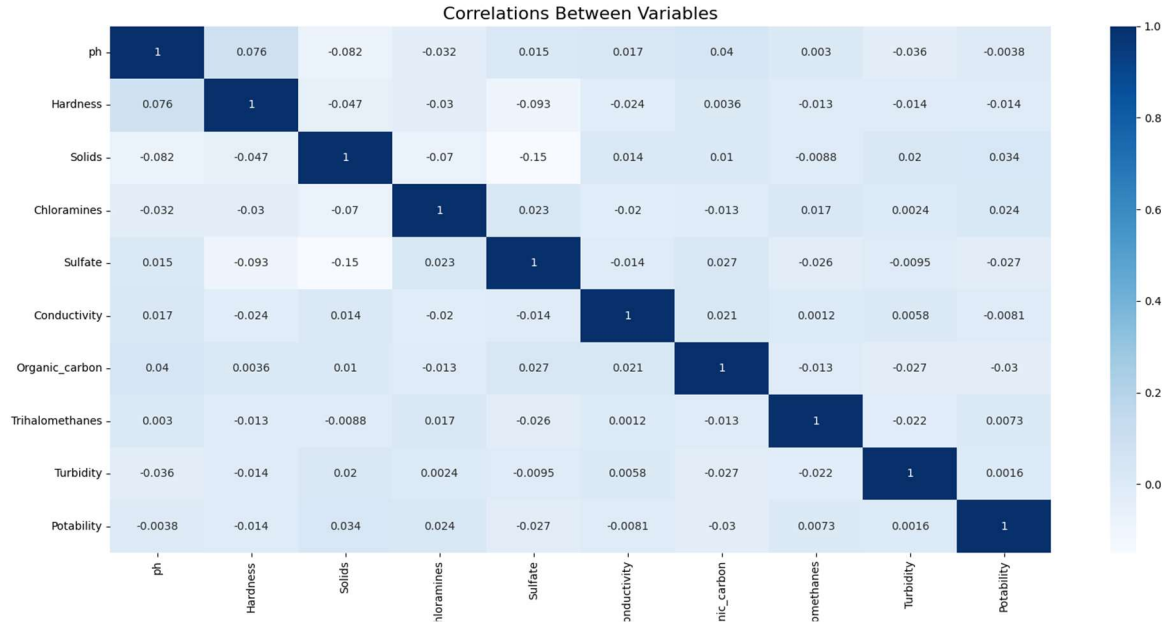


Figure 1: *Correlation between input data columns*

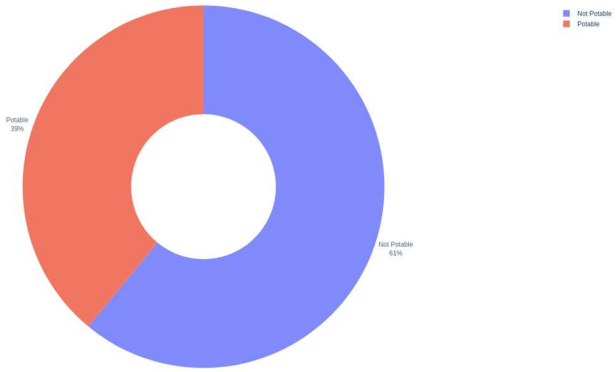


Figure 2: *Distribution of classes in Kaggle dataset*

Software Implementation

Stage 1: *Removing Null Values*

Initially the dataset contained 1434 null values. The first step in manipulating the data was to drop all rows containing null values using the python function ‘dropna’. Models were then trained initially on this dataset. Refer to Figure 3 for verification of null values being dropped.

Treating the NaN values by removing

```
In [9]:  water_dataset= water_dataset.dropna()

In [10]: water_dataset.isnull().sum()

Out[10]: ph                0
         Hardness          0
         Solids            0
         Chloramines       0
         Sulfate           0
         Conductivity      0
         Organic_carbon    0
         Trihalomethanes   0
         Turbidity         0
         Potability        0
         dtype: int64
```

Figure 3: *Removing null values from the dataset*

Next the team trained multiple models based on this modified dataset to gain an understanding of which models have the best accuracy. The models we chose for comparative study were:

1) Logistic Regression
2) Random Forest
3) Decision Tree
4) LightGBM
5) Support Vector Machines
6) XGBoost
7) Gaussian Naive Bayes

8) KNN

The best performing model was the support vector machine with a 67.8% accuracy. Due to the train accuracy being greater than the test accuracy, it was concluded that the Random Forest and Decision Tree models were overfit. Figure 4 details the accuracies found for this initial evaluation.

```
In [21]: models.sort_values(by="Test Accuracy Score", ascending=False)
```

Out[21]:

	Model	Train Accuracy Score	Test Accuracy Score	Misclassified_train	Misclassified_test
3	Support Vector Machines	0.752665	0.678808	348	194
1	Random Forest	1.000000	0.677152	0	195
2	LightGBM	0.999289	0.657285	1	207
4	XGBoost	1.000000	0.657285	0	207
6	KNN	0.786070	0.627483	301	225
7	Decision Tree	1.000000	0.610927	0	235
5	Gaussian Naive Bayes	0.628998	0.605960	522	238
0	Logistic Regression	0.611230	0.596026	547	244

Figure 4:*Initial Model Accuracies*

Stage 2: *Imputing Mean Values*

Imputation of values was used to further improve this dataset. In this stage, the mean value of each feature column was imputed where null values exist. Null values were concentrated in the following features: pH, Sulfates and Trihalomethanes. See Figure 5 for a code snippet to impute the mean values.

```
In [11]: water_dataset["ph"].fillna(value = water_dataset["ph"].mean(), inplace = True)

In [12]: water_dataset["Sulfate"].fillna(value = water_dataset["Sulfate"].mean(), inplace = True)

In [13]: water_dataset["Trihalomethanes"].fillna(value = water_dataset["Trihalomethanes"].mean(), inplace = True)
```

Figure 5: *Imputation of Mean Values*

As seen in Figure 6, mean imputation did not improve model accuracies and misclassifications.

```
In [21]: models.sort_values(by="Test Accuracy Score", ascending=False)
```

Out[21]:

	Model	Train Accuracy Score	Test Accuracy Score	Misclassified_train	Misclassified_test
3	Support Vector Machines	0.752665	0.678808	348	194
1	Random Forest	1.000000	0.677152	0	195
2	LightGBM	0.999289	0.657285	1	207
4	XGBoost	1.000000	0.657285	0	207
6	KNN	0.786070	0.627483	301	225
7	Decision Tree	1.000000	0.610927	0	235
5	Gaussian Naive Bayes	0.628998	0.605960	522	238
0	Logistic Regression	0.611230	0.596026	547	244

Figure 6: *Mean Imputation - Model Accuracies*

Stage 3: *Mean Imputation via Class-Based Grouping*

To further improve this dataset, the team tried a different method of mean imputation. The dataset was initially sorted into groups by potability classification. The mean of each column in the corresponding group was then imputed. Figure 7 details this imputation method.

```
water_dataset['ph'] = water_dataset['ph'].fillna(water_dataset.groupby(['Potability'])['ph'].transform('mean'))
water_dataset['Sulfate'] = water_dataset['Sulfate'].fillna(water_dataset.groupby(['Potability'])['Sulfate'].transform('mean')
water_dataset['Trihalomethanes'] = water_dataset['Trihalomethanes'].fillna(water_dataset.groupby(['Potability'])['Trihalometh
```

Figure 7: *Mean Imputation by Potability Group*

Models were then retrained based on this dataset. Overall model accuracies improved with the top performer being Random Forest at 82.4%. Other high-performing models were LightGBM, XGBoost and Decision Tree. The train accuracy score, test accuracy score, and misclassification metrics are shown in Figure 8.

```
In [10]: models.sort_values(by="Test Accuracy Score", ascending=False)
```

Out[10]:

	Model	Train Accuracy Score	Test Accuracy Score	Misclassified_train	Misclassified_test
1	Random Forest	1.000000	0.824695	0	115
2	LightGBM	0.996183	0.807927	10	126
4	XGBoost	1.000000	0.801829	0	130
7	Decision Tree	1.000000	0.746951	0	166
3	Support Vector Machines	0.735878	0.699695	692	197
6	KNN	0.786641	0.663110	559	221
5	Gaussian Naive Bayes	0.629008	0.631098	972	242
0	Logistic Regression	0.605725	0.628049	1033	244

Figure 8: *Grouped Mean Imputation - Model Accuracies*

After imputing mean values by group, the dataset was found to be imbalanced as shown by the confusion matrix in Figure 9.

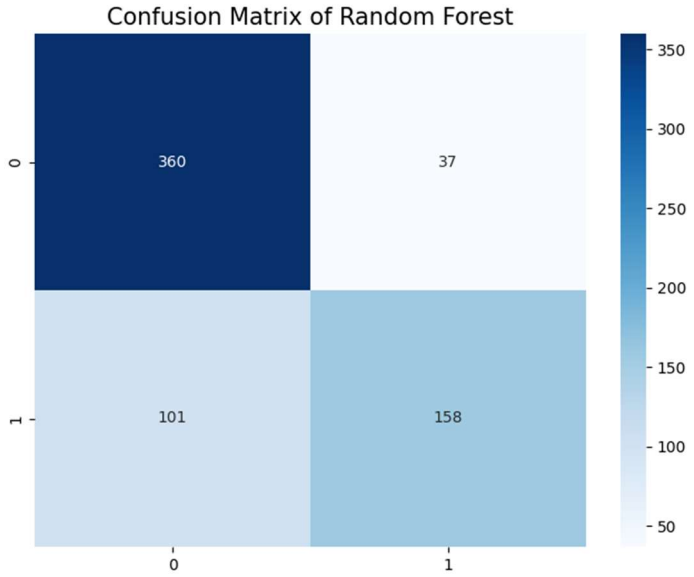


Figure 9: *Imbalance Dataset*

In order to balance the dataset, methods like oversampling, undersampling, ensemble learning techniques or cost-sensitive learning techniques can be used. In this case the best performing model was obtained using oversampling. The balanced dataset is shown by the confusion matrix in Figure 10.

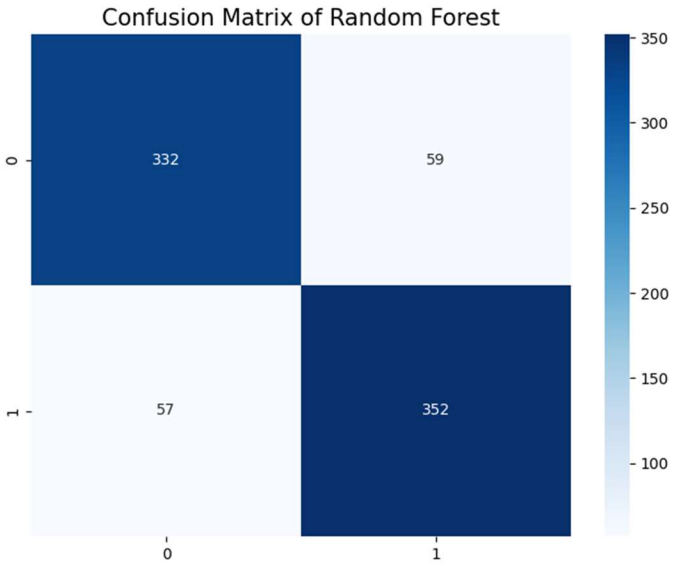


Figure 10: *Balanced Dataset After Oversampling*

Finally, the models were trained again on this balanced dataset. The top performing model was Random Forest with an accuracy score of 86.2%. Other top-performing models were XGBoost,LightGBM, and Decision Tree as shown in Figure 11 and Figure 12.

```
In [18]: models1.sort_values(by="Test Accuracy Score", ascending=False)
```

Out[18]:

	Model	Train Accuracy Score	Test Accuracy Score	Misclassified_train	Misclassified_test
1	Random Forest	1.000000	0.86250	0	110
4	XGBoost	1.000000	0.86000	0	112
2	LightGBM	0.996558	0.85625	11	115
7	Decision Tree	1.000000	0.83250	0	134
6	KNN	0.883292	0.67250	373	262
3	Support Vector Machines	0.744055	0.67000	818	264
5	Gaussian Naive Bayes	0.575407	0.56500	1357	348
0	Logistic Regression	0.524406	0.50875	1520	393

Figure 11: *Final Trained Model Accuracies*

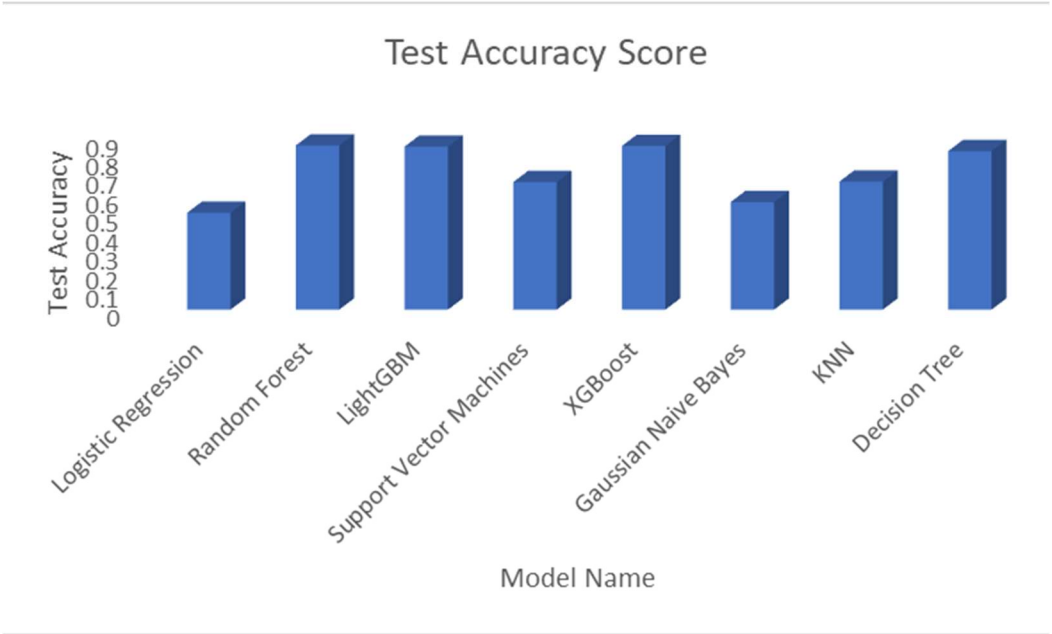


Figure 12: *Final Model Accuracies*

Model Optimization for Hardware

The decision tree model was selected for hardware implementation, as this provided similar accuracies compared to random forest, while still being resource optimal.

Using a grid search to determine the optimal parameters for the decision tree indicated that a depth of 24 levels was required to obtain an accuracy of 84%. However, this results in 2^{24} total nodes, each requiring memory for storing the corresponding feature index, threshold value and child nodes. The design size also grows as each node requires a floating point comparator.

As a result, it was decided to limit the tree height to 5 levels, resulting in a maximum of 32 nodes. Accuracy of the tree dropped from the earlier observed 83% to 75% due to this optimization but this enables the design to fit in FPGA hardware with lower number of resources.

Hardware Implementation

The initial intention of this project was to create a deployable system that could measure water potability in the field. Referring to Figure 13, the overall system was intended to use the onboard FPGA button to trigger the reading of a water sample. Output classification was to be displayed on the seven-segment displays onboard the FPGA. After training several models and ranking based on accuracy, our team decided that a decision tree model would be easiest to implement on the FPGA.

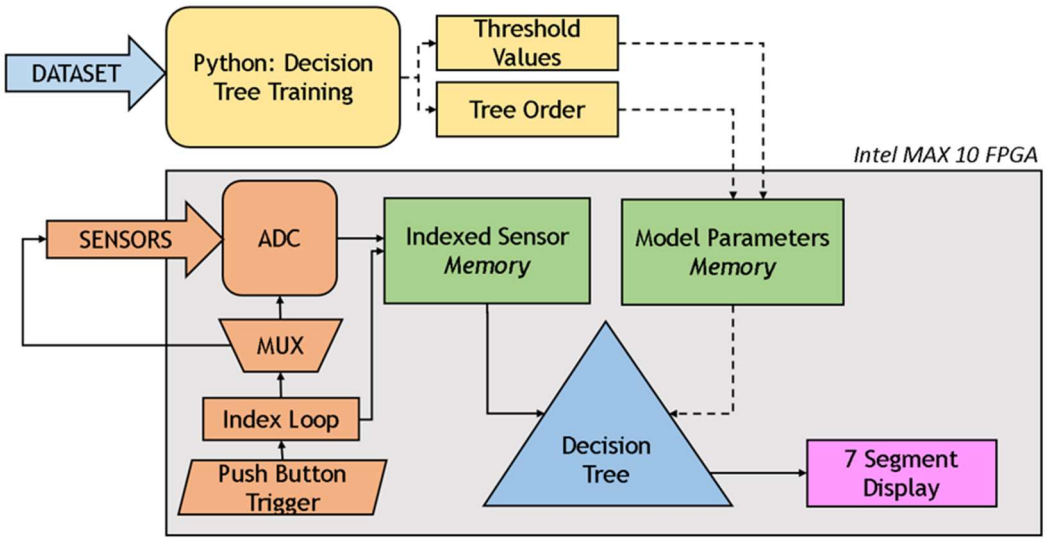


Figure 13: Hardware Implementation Overview

After tuning the model to a feasible 5-layer decision tree, the team began defining how to implement this tree in Verilog. First, the memory structure to hold the trained model parameters was defined as shown in Figure 14. It is important to consider the information the model needs at each node. First, the model must know if the node is a leaf. If the

node is a leaf, then the model will make a classification. In row zero of the memory array shown in Figure 14, the leaf data is held. Row 1 then indicates the classification. Held in Row 2 is the sensor value that will be compared to a threshold held in Row 3. Finally, Row 4 holds a comparison variable. The model followed a fixed decision making structure, where if the comparison is true, the node index was indexed one node right. If the comparison is false, the node index stays in the same column. To simplify the memory storage scheme, each branch contains its own parameter array.

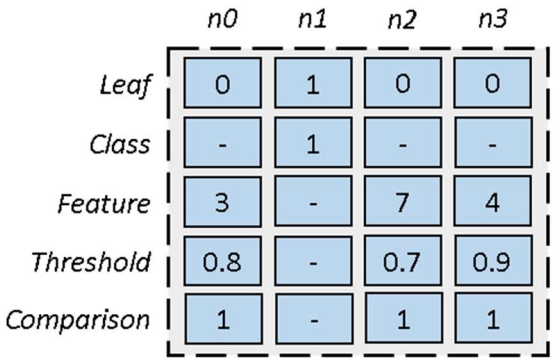
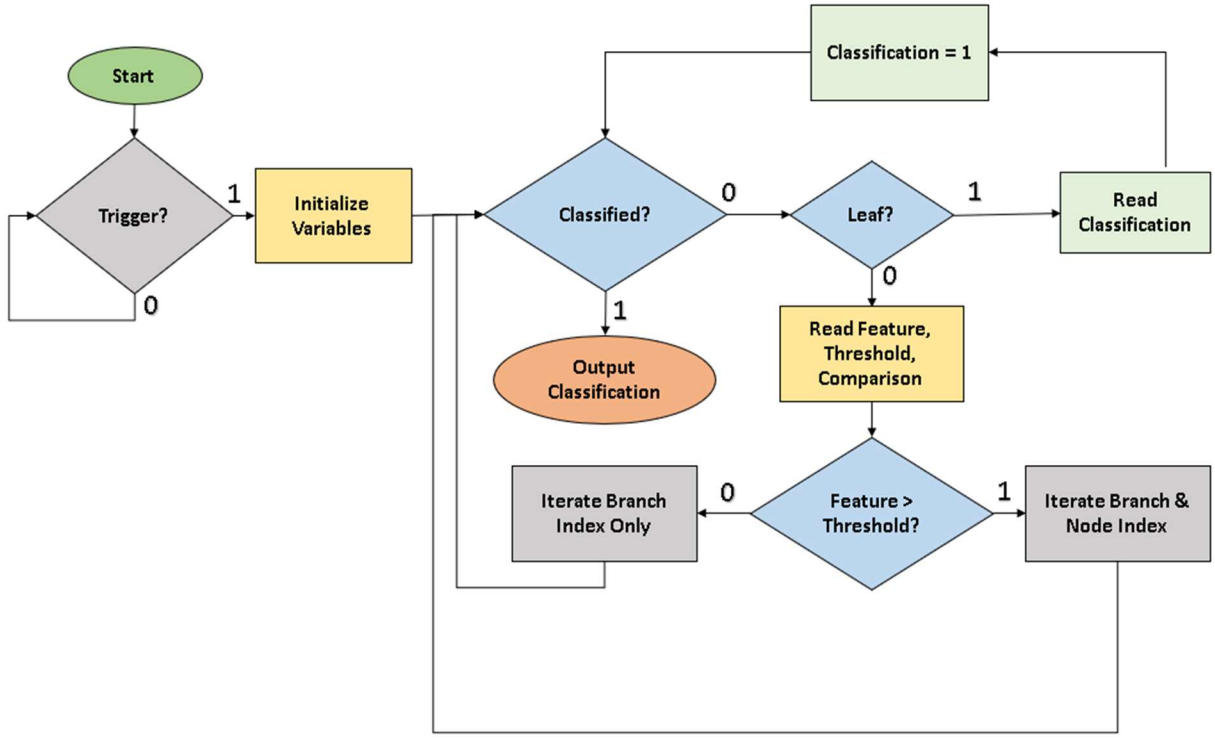


Figure 14: *Model Parameter Storage Structure*

Once the memory structure was clearly defined, our team wrote code to access this memory in the structure detailed in Figure 15. A flag variable was used to detect if a classification was made. If no classification has been made, the model moves downward by one branch and to the corresponding node. Then the memory array is accessed to pull the feature and threshold values from the memory array. This structure is repeated until an output classification has been made as indicated in the model parameter array.

Figure 15: *Decision Tree Code Structure*



Results

Preliminary validation of the model was done using a test bench run on a 2-layer decision tree. Parameters were set and the known classification was met. This verified that the memory access scheme and code structure worked correctly. Next, the decision tree model was expanded to 5 layers. To test the 5-layer decision tree, model parameters were hardcoded into the test bench. After inputting the model parameters, it was run over the training data set to determine the training accuracy of the model. Based on three trials, the model successfully classified each observation correctly. This corresponds with the 100% training accuracy seen in the Software Implementation. Finally, 10 test dataset observations were passed to the model. This test resulted in a 70% test accuracy for the model which closely resembled the 75% accuracy attained in by the model trained. The test bench results seen in Figure 16 are summarized in Table 1 and Table 2.

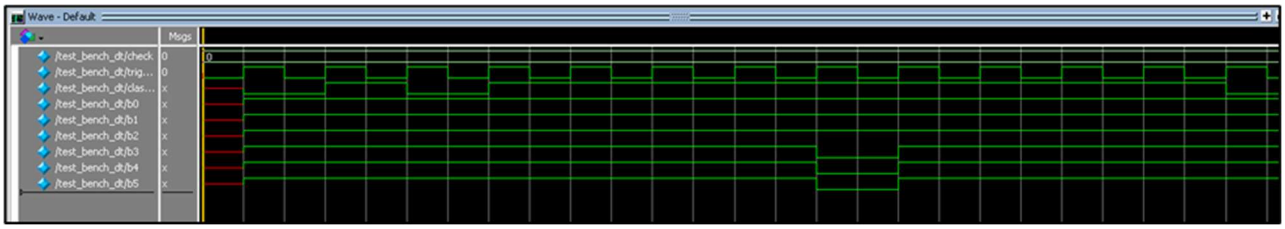


Figure 16: *Synthesizable Code Test Bench*

Test Bench: Training Accuracy		
Observation Number	**Classification**	**Test Bench Classification**
0	0	0
1	1	1
2	0	0
Training Accuracy		**100%**

Table 1: *Training Data Test-Bench Results*

Test Bench: Test Accuracy		
Observation Number	**Classification**	**Test Bench Classification**
0	0	1
1	1	1
2	1	1
3	1	1
4	0	1
5	1	1
6	0	1
7	1	1
8	1	1
9	0	0
Test Accuracy		**70%**

Table 2: *Test Data Test-Bench Results*

Scalable Implementation

A separate implementation was completed to verify the accuracy of Verilog implementations in simulation. Model files were generated from Python as left & right child nodes, thresholds, feature indices and predictions at nodes. These text files were then read back into the testbench and the accuracy of the classifier was measured.

For a maximum tree depth of 22 levels, we achieved a training accuracy of 100% and a test accuracy of 82.88%.

```
VSIM 69> run -all
# Total dataset size:        3196
# Number of mispredicts:           0
# Accuracy: 100.000000
# ** Note: $finish    : C:/Users/mishe/Documents/ASU Courses/EEE498_
#    Time: 38402 ps  Iteration: 0  Instance: /decisionTreee_tb
# 1
```

Figure 17: *Accuracy for Train dataset*

```
VSIM 65> run -all
# Total dataset size:         800
# Number of mispredicts:         137
# Accuracy: 82.875000
# ** Note: $finish    : C:/Users/mishe/Documents/ASU Courses/EEE
#    Time: 9650 ps  Iteration: 0  Instance: /decisionTreee_tb
# 1
```

Figure 18: *Accuracy for Test dataset*

Conclusion

The goal of this project was to train a machine learning model to correctly classify water potability based on nine features. This trained model was then aimed to be implemented in hardware using Verilog. This project first started by analyzing the dataset pulled from Kaggle. After cleaning the dataset and imputing mean values, multiple models were trained. The top performing model in this case was Random Forest with the Decision Tree model following behind it. To simplify the Verilog implementation, the Decision tree model was selected. First a simplified 5 layer decision tree model was hardcoded into Verilog. This code tested successfully in a test bench, matching the expected values. After verification of the 5 layer model, a scalable decision tree model was programmed in Verilog. Again this was verified via a test bench where the accuracies matched the expected values. The team achieved all of the intended goals except for the implementation of an ADC in Verilog. Future work for this project includes implementing an ADC to collect real data values. A stretch goal of this project is to implement a trainable model that can be deployed to an FPGA.

References

[1] Kadiwal, A. (2021) Water Quality: Drinking water potability, Version 3. Retrieved April 19, 2022 from https://www.kaggle.com/datasets/adityakadiwal/water-potability.

[2] “Verilog arrays and Memories,” *ChipVerify*. [Online]. Available: https://www.chipverify.com/verilog/verilog-arrays. [Accessed: 28-Apr-2022].

[3] D. K. Tala, “Verilog in one day part,” *I*, 01-Feb-1970. [Online]. Available: https://www.asic-world.com/verilog/verilog_one_day1.html. [Accessed: 28-Apr-2022].