ANIMAL NOISE EXPOSURE ANALYSIS

COMPSCI671: FINAL PROJECT

NETID: MS1112

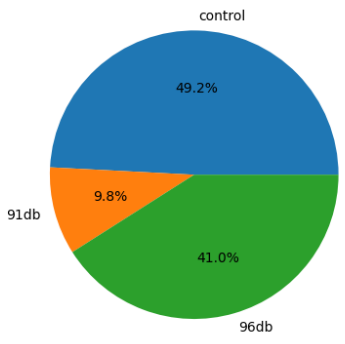
1. BACKGROUND

1.1. INTRODUCTION

Auditory Brainstem Response (ABR) is an action potential generated by brainstem in response to an auditory stimulus [1] in other words they are called evoked potentials. ABR testing is widely used in diagnosing various forms of deafness [2]. ABR readings can also be used to assess the neural integrity of auditory brainstem [3]. Analysing and interpreting ABRs for animals can right now be done only by veterinarians with expert training in diagnostic electrophysiology or board-certified veterinary neurologists. The motivation is to automate the ABR analysis to help identify the presence of noise and its type, specially in terms of decibel level. Inexperienced audiologists could give wrong diagnosis at times, and it would be helpful to have a tool to automate and validate the results. Being able to identify key ABR metrics needed to identify the presence of noise and its type could also give some insight into brainstem readings. This automation need not only be limited to animals, the same can be applied to detecting the hearing thresholds in infants which again is currently done only by a select few.

1.2. DATA DESCRIPTION

The dataset had a total of 108 columns and 183 datapoints. The target column contained 3 classes – control, 96db and 91db. The subjects under the control class were not exposed to any noise and the data was taken before (indicated by T0) and after 2 weeks (T2). The subjects under the 96db class were exposed to 96db noise for two hours and the data was collected before and 2 weeks after the noise exposure. The same with subjects under the class 91db. The concentration of these classes in the dataset can be seen in figure [1]

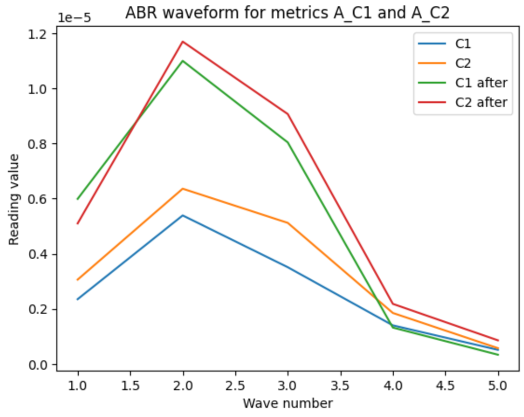


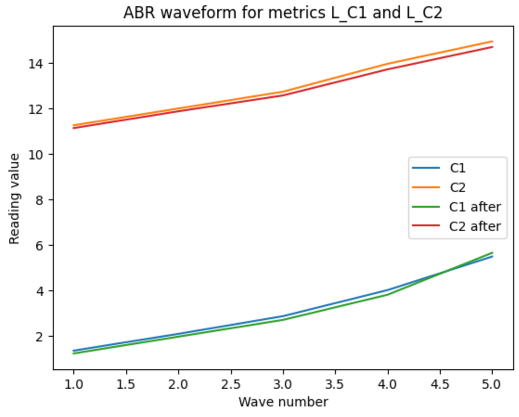
Experiment Related Features:
SubjectID and Run are features that are related to the specific experiment. SubjectID does not reveal anything about the subject and hence has no useful information and since there are 61 unique subjects in the dataset, one hot encoding them would increase the feature space and would not benefit the classification task. For these reasons I have dropped both these features from the dataset. The same subject is subjected to 3 different stimulus levels – 70_70, 70_90 and 90_90. Since the stimulus

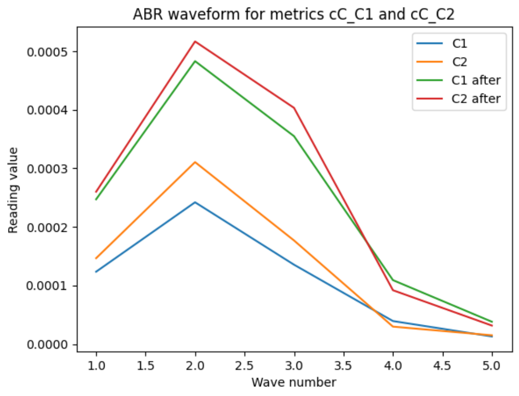
level is an important measure , this feature was one hot encoded and added to the dataset as

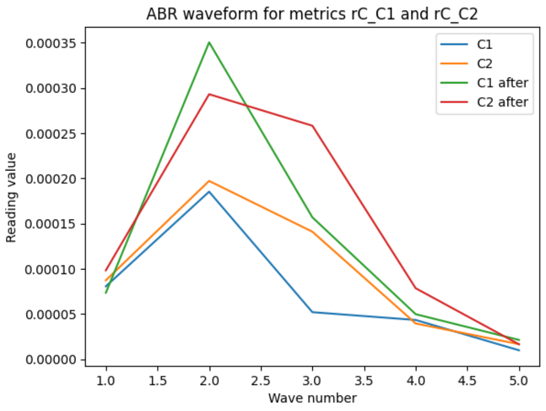
ABR metrics and Waveforms:

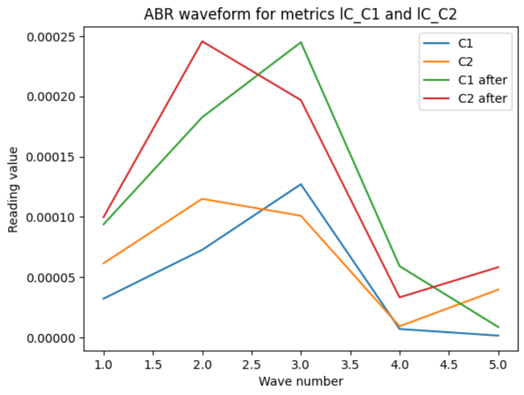
There are mainly 5 metrics that we can identify from the dataset – A, L, rC, lC and cC. These metrics are taken for C1 and C2 presumably left and right ear. For all the 5 ABR metrics, there are features with W1, W2, W3, W4 and W5. These are the main five waveforms in the ABR readings, and these are recorded for the 5 different metrics. A sample waveform for the 5 metrics for both C1 and C2 values are shown in figure []. These belong to the control group.







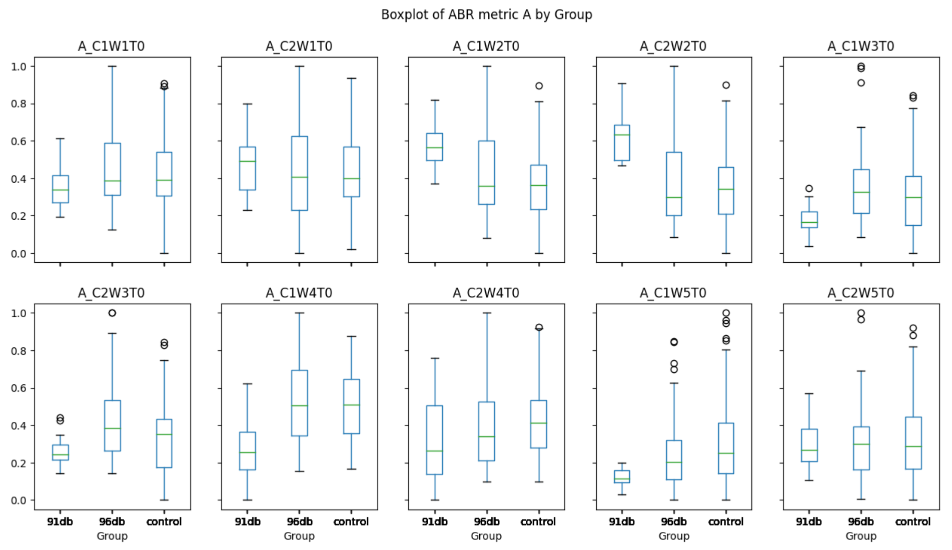


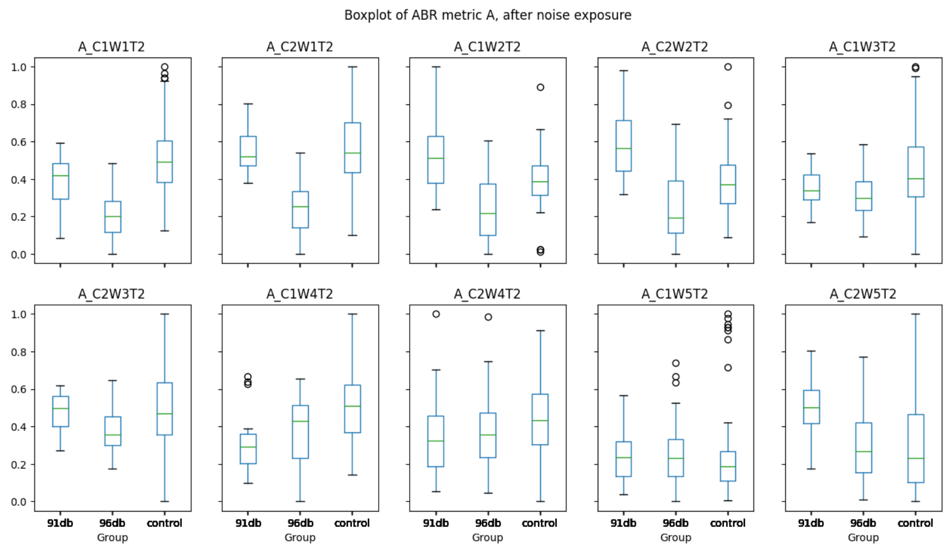


Here we can see that the metric cC and A have similar waveforms but different readings with Cc metric having a larger reading. L metric does not seem to measure the amplitude, this could be the latency readings of the ABR waves. The changes in L metric before and after the noise exposure is also minimal hence probably not good for classification purposes. Since C1 and C2 readings are very similar this could mean that these are the right and left ear readings as they are required to give similar readings for animals with unimpaired hearing.

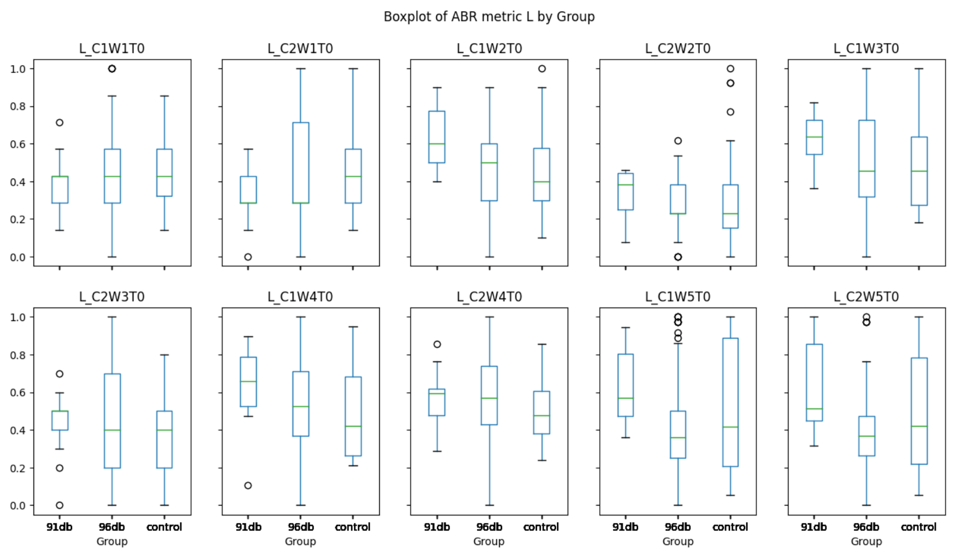
Features before and after Noise exposure:
The features before noise exposure are indicated by the T0 at the end and the features after noise exposure are indicated by T2. There are 50 features before and after amounting to a total of 100 different ABR metrics.

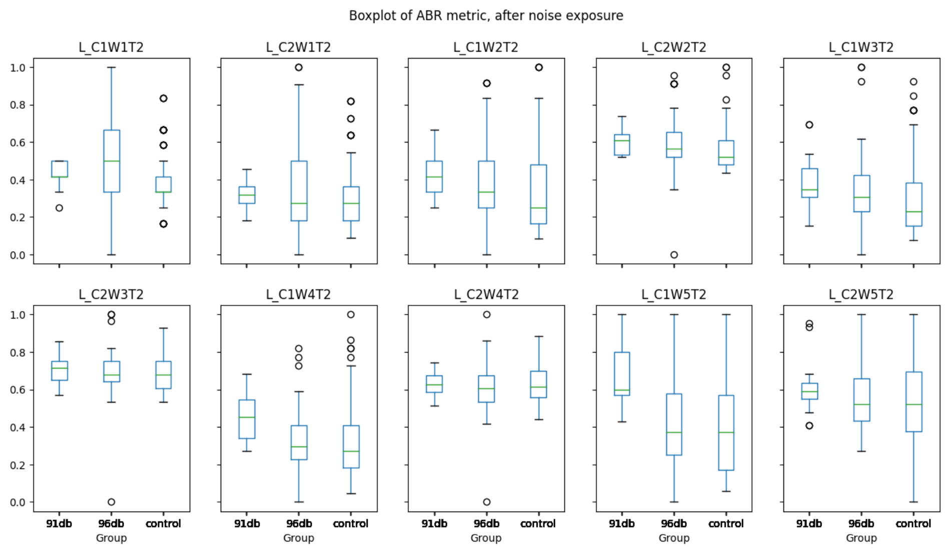
Understanding the Data Distribution:
Since the values of the features varied a lot, the features were normalized using a min max method to be able to plot all features on the same figure. T
ABR readings through metric A before and after noise exposure



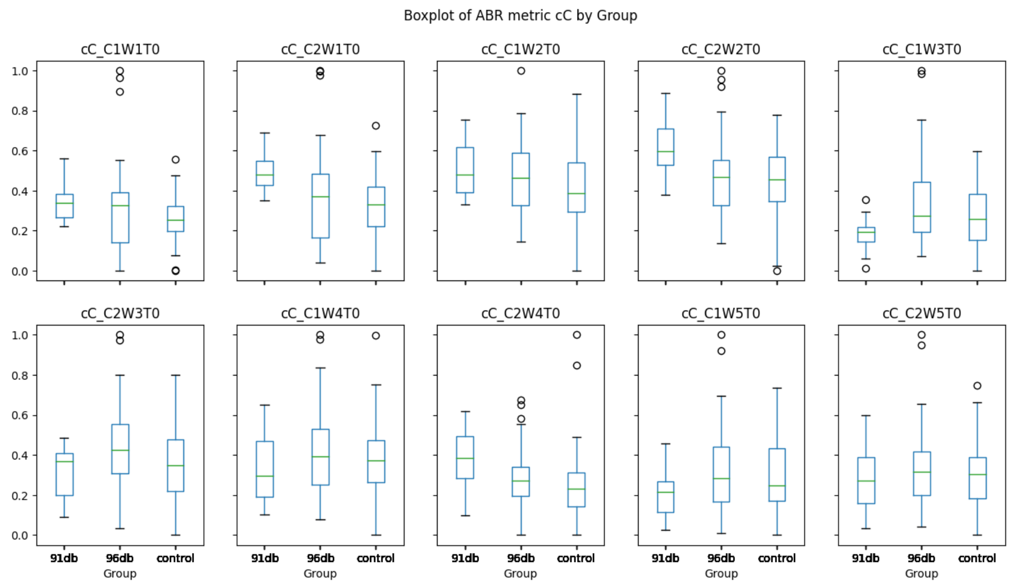


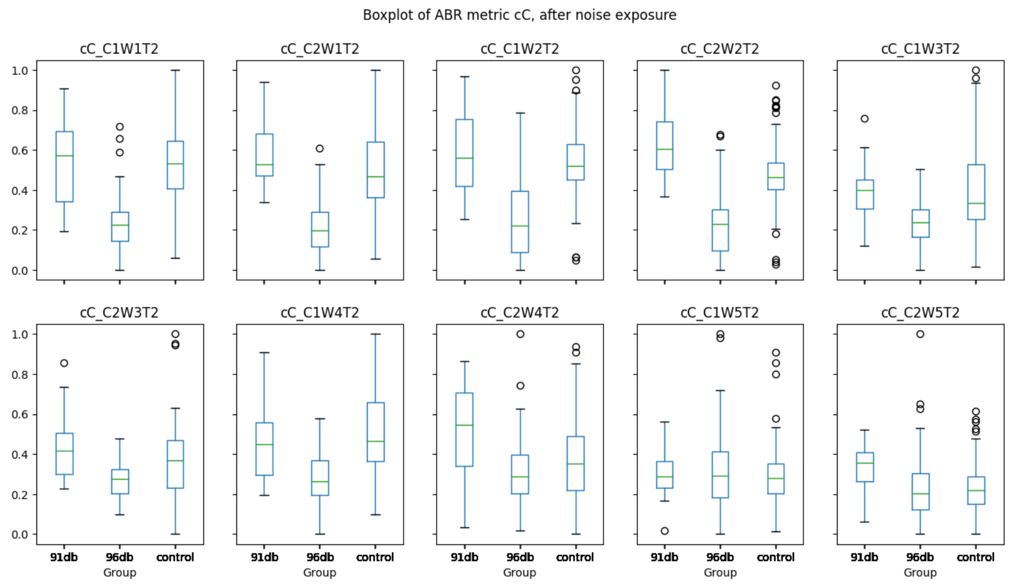
When we compare the boxplots of ABR metric after noise exposure we can see that there are some clear distinctions in the distributions for control and 96db class for some of the features like A_C1W1T2, A_C2W1T2. For feature A_C2W5T2 there is a clear difference in the distribution of 91db group and the rest. These features that show a difference in the distribution for the different groups could be of importance to the models.



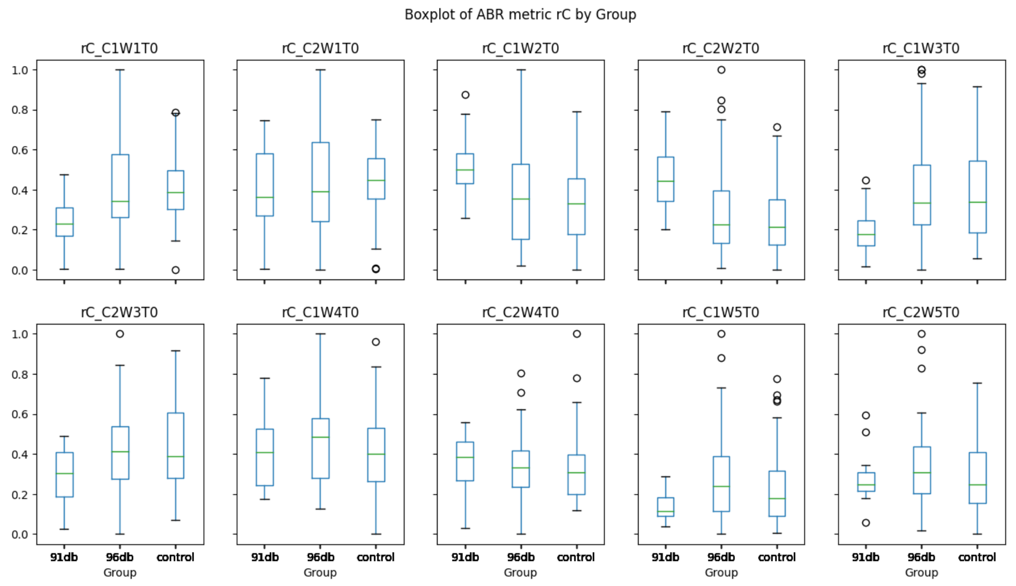


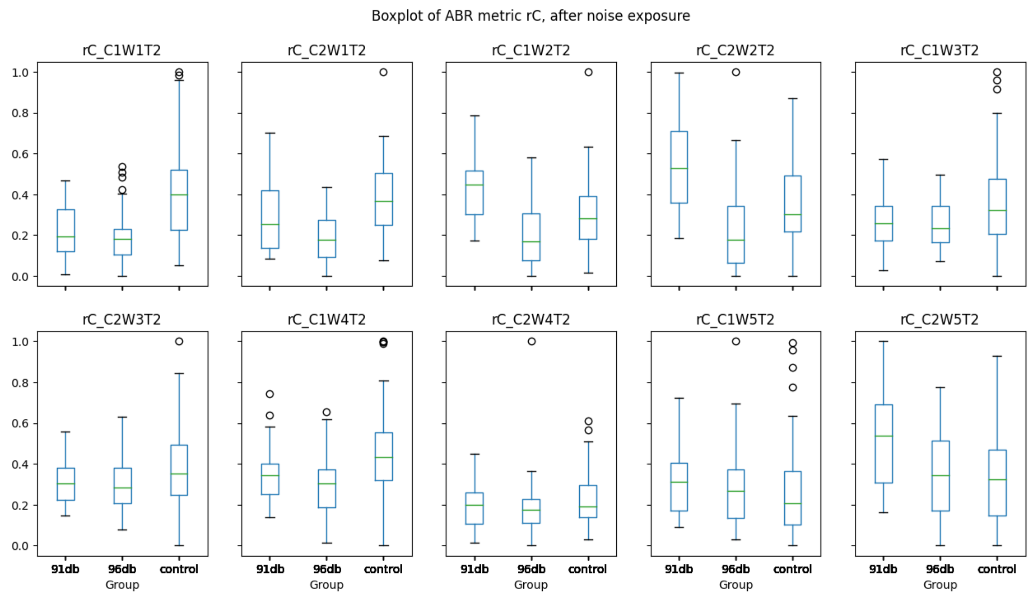
Here we can see that the feature L_C1W5T2 shows a different distribution for the group 91db but very similar distribution for the groups 96db and control. The other features after noise exposure seems to have a similar distribution for all the groups which means this metric might not give us the needed information as seen earlier in the waveform analysis of before and after noise exposure.


Boxplot of ABR metric cC by Group

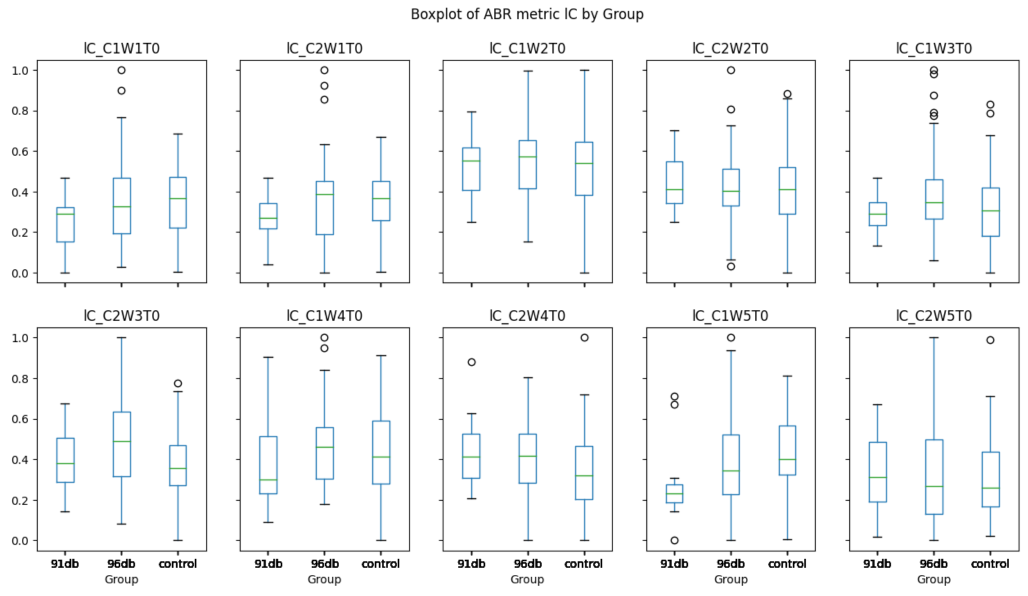

Boxplot of ABR metric cC, after noise exposure

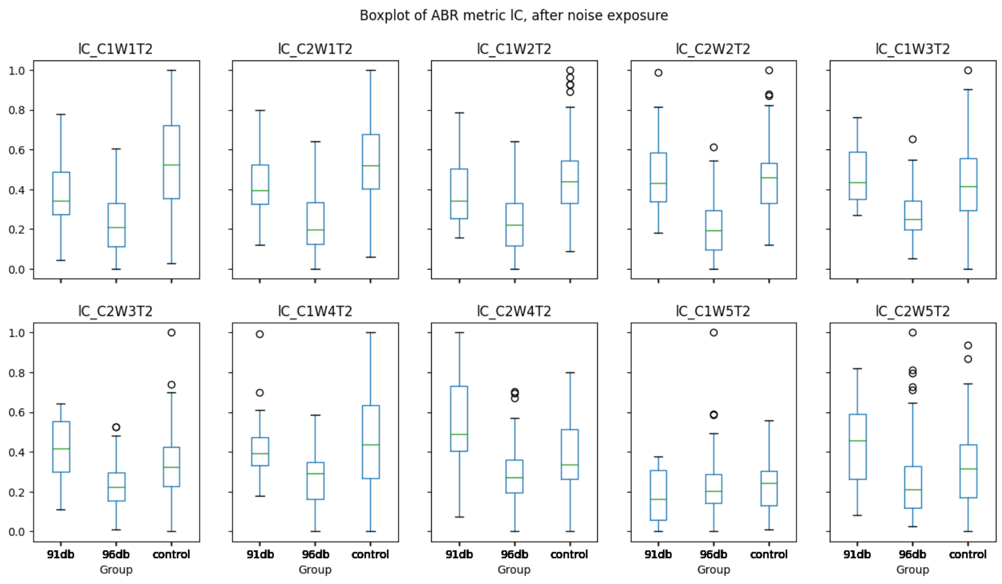
The waves W1,W2 and W3 has a different distribution for 96db and control class which could be useful for the classification models.


Boxplot of ABR metric rC by Group



The waves W1 and W2 has a different distribution for 96db and control class which could be useful for the classification models. W2 also shows a different distribution for the group 91db which could help classify the 91db class.





Most features here show a different distribution for the different groups which can be useful for classification.

Data Scaling:

Although there were a few outliers present in the dataset as seen in the boxplots they were not removed due to the limited number of datapoints available for training. There were no missing data points but the features had values of different range so they were normalized using a minmax scaler. The normalization was done before training without any data leakage. The minmax scaler was fit on the train dataset and this was used to normalize the test dataset ensuring no data leakage. The minmax scaler works as shown below:

$$X_{std} = \frac{X - X_{min}}{X\_\max - X\_min}$$

$$X_{scaled} = X_{std} * (\max - min) + min$$

2. METHODS

The four algorithms I have chosen for this project are logistic regression, decision tree, adaboost and explainable boosting machine (EBM).

Logistic Regression:

Why?

Logistic regression model gives coefficients which directly indicate the importance of the variable in the model. The higher the coefficient value, the higher the importance. Apart from this, logistic regression is a simpler, less complex and easily understandable model with a probabilistic interpretation as well.

Library Used:

I have utilized the Logistic Regression library from sklearn for this[1]. LogisticRegression is part of the linear_model library in sklearn. For this task the classification problem becomes a multinomial classification.

Training Algorithm and Optimization:

Class weights are given to the 3 classes based on the parameter class_weight = 'balanced' or None. Since the classes are imbalanced this parameter has been added to hyperparameter tuning to visualize the difference in the results with balanced and imbalanced data classification.

The loss of the model depends on the solver being used. If the solver is lbfgs () and newton-cg then the loss is defined as:

$$loss = \sum_{i} s_i loss_{base}(y_i, X_i@W + intercept) + 0.5 * regularizer * \|w\|_2^2$$

Where s_i is the sample weight (ones if the class_weight is None and calculated based on the equation before if given balanced). The first term is the loss of the model from misclassifications and the second term is the l2 regularizer for the coefficients where the regularizer determines the strength of regularization. The regularizer parameter C is used The base loss is the cross-entropy loss defined as:

$$loss_{base} = \log\left(\sum_{k=0}^{nclasses-1} \exp\left(raw\_pred_{i,k}\right)\right) - \sum_{k=0}^{nclasses-1} y_{true} * raw\_pred$$

Newton-cg Optimization: This is an iterative numerical optimization method for finding the minimum of a scalar function. In each iteration, it computes the Newton direction by solving a linear system approximately using the conjugate gradient (CG) method – utilizes both the gradient and the hessian product. The step size is determined through line search, and the algorithm iterates until the gradient of the objective function is sufficiently small or a maximum number of iterations is reached.

L-BFGS optimization: We do not need to calculate the hessian in this case. The gradient alone is used to minimize the scaler function.

Saga optimization: This optimization was chosen because this allows stochastic average gradient optimization with L1 loss, while other solvers do not support this for multinomial classifications.

Decision Tree:
Why?
Decision Trees are inherently intuitive because of their tree structure. The way the splits are created can also be explained. This makes the model extremely interpretable. Since decision trees have much fewer features in the final tree, I believe it also introduces some level of sparsity to the model. Decision Tree classifier libraries are readily available with sklearn package.

Library Used:

I am using the DecisionTreeClassifier from sklearn which can be used for binary and multiclass classification problems. It recursively partitions the feature space based on the most informative features, allowing it to create intuitive decision rules while offering parameters for controlling tree depth and impurity criteria [2].

Training Algorithm and Optimization:

The decision tree is a recursive algorithm continuously building nodes till the max depth criteria is met. At each node selection the algorithm chooses a feature at random and computes the feature importance or the impurity. The impurity is tracked to find the feature that gives the best measure to be later chosen as the best split. Then checks are done to determine whether the node should be a leaf or should continue splitting based on the minimum number of samples. The default value for this in the sklearn library is 2 which was not chosen as a hyperparameter to tune due to the low number of datapoints available for class 91db. If the node criteria is met then the node is split and the value for the node is evaluated and stored in the tree for interpretability.
The best split is tracked using impurity measures. The different impurity measure calculations are gini for the gini impurity calculations and log_loss or entropy for Shannon information gain. Log_loss and entropy are essentially the same where log_loss is preferred for binary classification and cross entropy for more generalized multiclass classification. If the classification problem has k classes, then the probability for this node for class k can be given as:

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

Where n_m is the number of datapoints in the node m, and Q_m is the data represented at node m. With this probability distribution we can compute the rest.

$$gini\ impurity\ measure = \sum_k p_{mk}(1 - p_{mk})$$

$$entropy\ or\ \ log\ loss = -\sum_k p_{mk} \log p_{mk}$$

ADABOOST ALGORITHM:
Why?
Apart from being known for boosting performance, adaboost is an ensemble learning algorithm that provides interpretability. The readily available sklearn package has the Adaboostclassifier which has a feature importance measure which could give more interpretability. Since Adaboost is using weak algorithms as the base classifier, the model transparency also increases. Adaboost combines different weak classifiers together, so it can capture nonlinearity of the models while still being interpretable.

Library Used:
sklearn.ensemble.AdaBoostClassifier [3]. The AdaBoost classifier is an ensemble learning algorithm that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of

incorrectly classified instances are adjusted such that subsequent classifiers focus more on the instances which were misclassified.

Training Algorithm and Optimization:

The ensemble learning process begins with the initialization of the weights for the estimators. For each estimator (the base estimator is a decision tree with max depth = 1, essentially a decision stump with one feature) the estimator predictions and errors are calculated. To determine the weight assigned to the estimator a y coding process is performed as shown below.

$$y\_coding_k = \begin{cases} 1, & c == k \\ \dfrac{-1}{K-1}, & c\,! = k \end{cases}$$

Where c is the true label and K is the total number of classes. So, in our case it would become.

$$y\_coding_k = \begin{cases} 1, & c == k \\ -0.5, & c\,! = k \end{cases}$$

The y coding is done for each target class as shown. This would give out an array of shape (n_samples, n_classes) after compute. For multiclass problems the y coding is done in a one vs all scheme. Single estimator weight calculation is given by:

$$estimator\ weight = \ -1 * (learning\ rate) * \frac{n_{classes-1}}{n_{classes}} * \textstyle\sum_n y_{coding_n} \log\left(y_{pred_n}\right) \qquad [1]$$

Sample weights are then boosted only if either the estimator weights are negative (give more importance to the misclassified classes) or sample weights are positive (from previous iterations). The sample weight update is given as:

$$\begin{aligned} &sample\ weight \\ &\quad = sample\ weight * exp(estimator\ weight \\ &\quad * ((sample\ weight > 0)\ or\ (estimator\ weight < 0))\ ) \end{aligned}$$

These estimator values are then stored, and these sample weights are carried forward to compute the estimator weights for the subsequent estimator till the number of estimators has reached or the estimation error has become zero or if the sum of the sample weights is non zero. The iterative nature of this process, guided by the boosting algorithm, contributes to the refinement of the model over multiple rounds, enhancing its overall predictive performance.

EXPLAINABLE BOOSTING MACHINE:

Why ?

EBM is an interpretable boosting algorithm which also has interaction features. EBM are often as accurate as the blackbox models. So I have used EBM to get a high predictive accuracy while maintaining interpretability.

Library Used:

interpret.glassbox. ExplainableBoostingClassifier [4] Explainable Boosting Machine (EBM) is a tree-based, cyclic gradient boosting Generalized Additive Model with automatic interaction detection. This is a glassbox model.

Training and Optimization:

EBM is a generalized additive model of the form:

$$g(E[y]) = \beta_0 + \sum f_j(x_j)$$

Where g is the function that adapts the model to regression or classification tasks.

If there are n features in the dataset, then the training happens only on one feature at a time. So, for each feature a small tree is trained looking at only that feature and the boosting is performed on this with the residual carried forward to the next feature where the small tree is trained on only that feature. This technique is called the round robin pass through all the features. The learning rate for the boosting will be small so the order of the features does not matter. During the second iteration, another small tree will be trained on all the features (round robin fashion). If the total number of iterations is m, then at the end of all the iterations you will get m small trees trained only on feature1, m trees trained only on feature2 and so on. We can summarize these trees into graphs by getting their predictions. The graphs are obtained for each feature. The final model is the addition of these graphs together. The loss function for optimization for classification problems in EBM is the logistic loss given as:

METRICS FOR PERFORMANCE EVALUATION:

1. Accuracy: This is a measure of correctness of the model. Higher the accuracy, higher the model performance. But this only tells us about the number of samples correctly identified.

$$accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ samples}$$

2. F1-Score: If the data is imbalanced (which in this case it is), it is better to use F1-score to determine the model's performance better. The F1 score for a multiclass problem is the average of the individual f1-scores. This way the imbalanced data will not affect the performance measure. If there are n testing samples, then:

$$f1score_{control} = 2 * \frac{precision_{control} * recall_{control}}{precision_{control} + recall_{control}}$$

$$f1score_{96db} = 2 * \frac{precision_{96db} * recall_{96db}}{precision_{96db} + recall_{96db}}$$

$$f1score_{91db} = 2 * \frac{precision_{91db} * recall_{91db}}{precision_{91db} + recall_{91db}}$$

$$f1score_{macro} = \frac{f1score_{control} + f1score_{96db} + f1score_{91db}}{3}$$

$$f1score_{weighted} = \frac{T_{11} * (f1score_{control}) + T_{22} * (f1score_{96db}) + T_{33} * (f1score_{91db})}{n}$$

CONFUSION MATRIX USED FOR COMPUTING PRECISION AND RECALL:

Actual	Predicted		
	Control	91db	96db
Control	True control (T_{11})	False 91db (F_{21})	False 96db (F_{31})
91db	False control (F_{12})	True 91db (T_{22})	False 96db (F_{32})
96db	False control (F_{13})	False 91db (F_{23})	True 96db (TP_{33})

Where T_{11}, T_{22}, T_{33} are the true positives for the classes control, 91db and 96db respectively.

Class	False Positives	False Negatives
Control	F_{12}, F_{13}	F_{21}, F_{31}
91db	F_{21}, F_{23}	F_{12}, F_{32}
96db	F_{31}, F_{32}	F_{13}, F_{23}

3. Sensitivity/recall: measure of how well a machine learning model can detect positive instances. This is the ratio of true positives and the total number of samples that are true (for that class). The higher the better.

$$Sensitivity_{control} = \frac{T_{11}}{T_{11} + F_{21} + F_{31}}$$
$$Sensitivity_{91db} = \frac{T_{22}}{T_{22} + F_{12} + F_{32}}$$
$$Sensitivity_{96db} = \frac{T_{33}}{T_{33} + F_{13} + F_{23}}$$

4. Specificity/precision: Precision is a measure of the ratio of true positive cases to all cases with positive predictive results. This measure tells how precise the model is in its results.

$$Specificity_{control} = \frac{T_{11}}{T_{11} + F_{12} + F_{13}}$$
$$Specificity_{91db} = \frac{T_{22}}{T_{22} + F_{21} + F_{23}}$$
$$Specificity_{96db} = \frac{T_{33}}{T_{33} + F_{31} + F_{32}}$$

5. Permutation importance scores: Impurity based feature importance scores can be misleading for high cardinality features (many unique values are possible). So, we can compute the scores using the sklearn library permutation importance scores. The feature we are calculating the score will be randomly permuted n number of times and then the score is evaluated.

$$importance = score_{reference} - \sum_{k=1}^{K} s_{k,j}$$

Where K is the number of times the random shuffling is done. j is the feature we are computing the importance score for and $s_{k,j}$ is the score of the model on corrupted data

3. EXPERIMENTS

3.1. RESULTS

Model Specifications:

Model	Hyperparameters
Logistic Regression	C=0.1, solver = 'saga'
Decision Tree	Max_depth = 3, criteria = 'log_loss'
Adaboost	n_estimators = 150, learning_rate = 1.5
Explainable Boosting Machine	Learning rate = 0.05

Combined Results: Accuracy, Weighted f1 score

Model	Accuracy	Weighted_F1_Score
LR	0.89189	0.82354
DT	0.86486	0.75928
Adaboost	0.86486	0.79249
EBM	0.945945	0.90015

The highest accuracy is for EBM as expected as EBM is an ensemble method designed to give results close to neural networks and boosting models while being interpretable. If we were to rank the models based on accuracy and weighted f1 scores the ranking would be EBM > LR > Adaboost > DT. Ensemble algorithms are tend to give better performance than a single decision tree.

Class wise Results: precision, recall and f1 score

Model	Precision (control)	Precision (96db)	Precision (91db)
Logistic Regression	0.89189	0.9375	0.5
DT	0.88889	0.86667	0.75
Adaboost	1.0	0	0.93334
EBM	0.9444	1.0	0.75

Model	recall (Control)	Recall (96_db)	recall (91_db)
LR	0.94445	1.0	0.25
DT	0.88889	1.0	0.5
Adaboost	0.78261	0	1.0
EBM	0.94444	1.0	0.75

Model	Macro_F1	F1_score (Control)	F1_score (96_db)	F1_score(91_db)
LR	0.73999	0.91892	0.96774	0.33334
DT	0.80582	0.88889	0.92857	0.6
Adaboost	0.61452	0.87805	0.96552	0
EBM	0.89814	0.94444	1.0	0.75

When we compare the individual results of all the classes we can see the best model is EBM as it has the highest individual scores for precision, recall and f1 scores. This means that the imbalance in the data has not affected the model. This could also suggest a strong correlation between some of the features and the classes, especially 91db as EBM creates an ensemble of classifiers using a single feature, such relations can also be visualized using the local explainability feature of EBM. Adaboost has performed the worst among the four models as the precision, recall and f1 score was zero. This means it could not get a single datapoint in 91db class correct. Unlike EBM, Adaboost has multiple estimators trained on different features (selecting the best among different features) so it is possible that the split is biased towards the other two classes with more number of datapoints.

3.2. CROSS VALIDATION STRUCTURE

The cross-validation structure is the same for all the four algorithms. The dataset had a total of 183 data points. 80% of dataset was used to training and 20% of the dataset was used for testing. The training dataset was further divided into validation sets following a 5-fold cross validation structure. The data was divided based on the class labels to ensure the same proportion in all the folds.

Dataset	#control	#96db	#91db
Train	72	60	14
Test	18	15	4

Fold	#control class - train	#96db class – train	# 91db class - train	#control class - val	#96db class – val	# 91db class - val
1	57	48	11	15	12	3
2	57	48	12	15	12	2
3	58	48	11	14	12	3
4	58	48	11	14	12	3
5	58	48	11	14	12	3

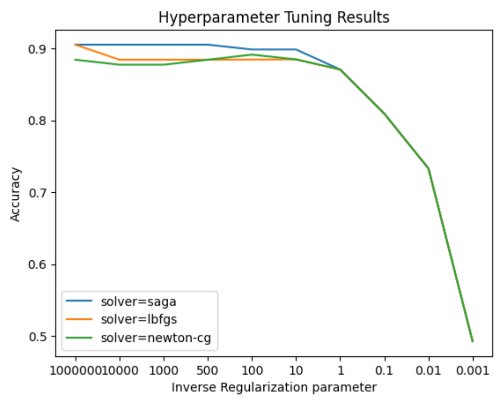
3.3. HYPERPARAMETER TUNING AND VISUALIZATIONS

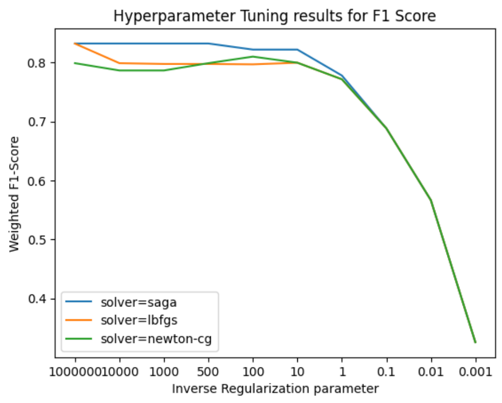
1. LOGISTIC REGRESSION

The hyperparameters tuned were the regularization parameter C and the solver. A grid search was done across these parameters. The best model is selected based on the highest accuracy and the highest f1-score. The regularization constant C is inversely proportional to the regularization strength. The solver determines the optimization algorithm.

Hyperparameter	Values
C	1000000,10000,1000,500,100,10,1,0.1,0.01,0.001
solver	Newton-cg, saga, lbfgs

Result:

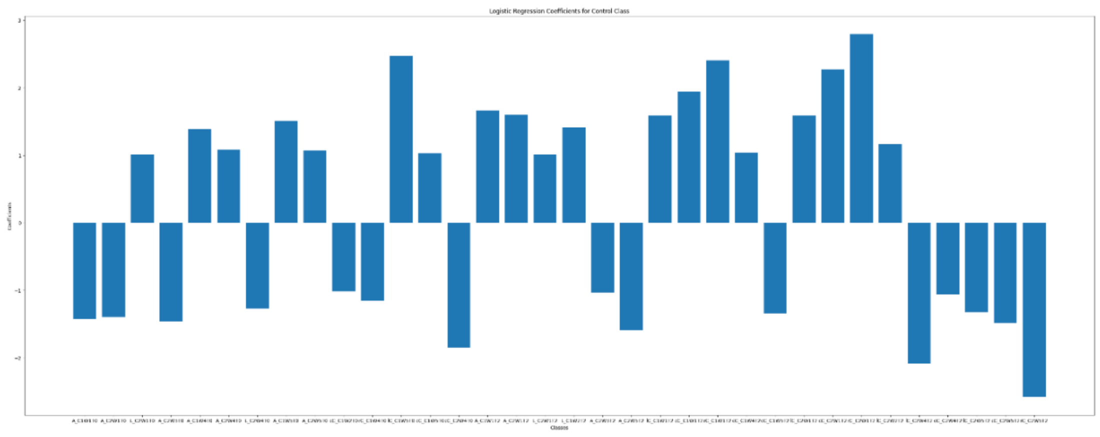




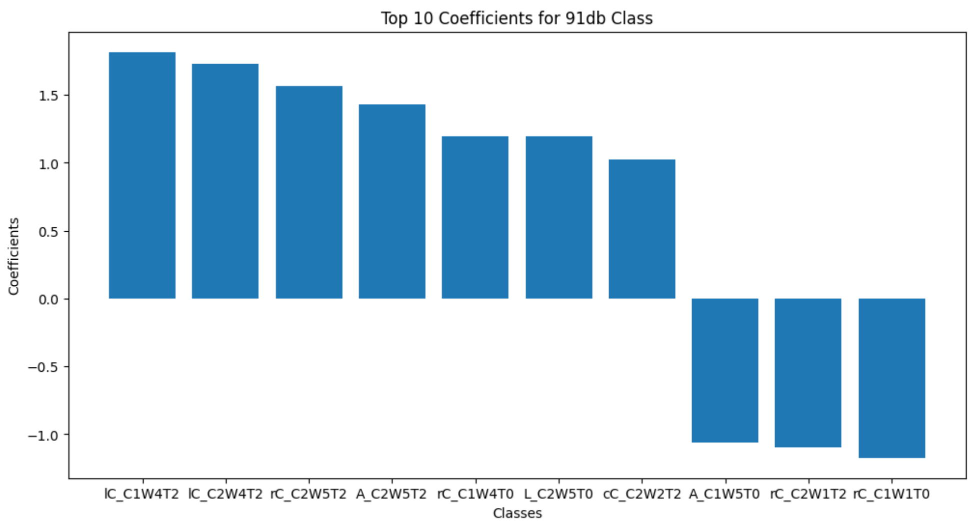
For very low regularization values (C= 0.001 to 1) all 3 solvers performed equally. But when the regularization starts getting stronger, the stochastic average gradient solver has outperformed the other two. The same accuracy is achieved by both the lbfgs and saga solvers but at different regularization values. Saga was able to achieve it with a much stronger regularization (C=500).

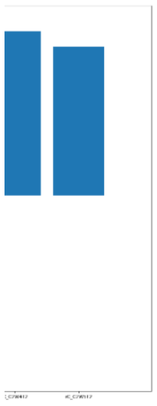
INTERPRETABILITY OF THE MODEL:

Logistic regression model for multiclass problems will have coefficients for each class. Since we have utilized l2 penalty this does not bring the coefficients down to zero, but it does reduce the coefficient strength of these features. Here we can see for the final model only 34 features have absolute coefficient value above 1 for the class control, which is a lot easier to interpret than the 103 features

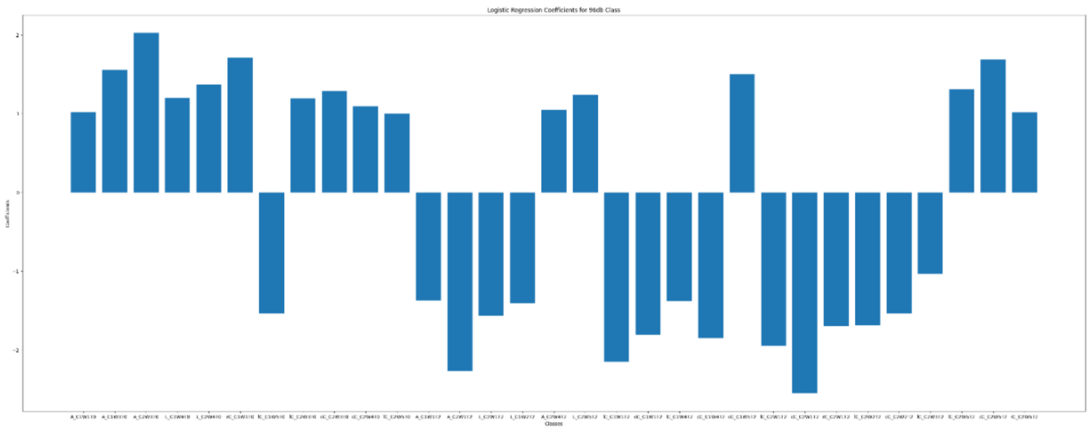


For the class 91db only 15 features have an absolute value of 1 and above.

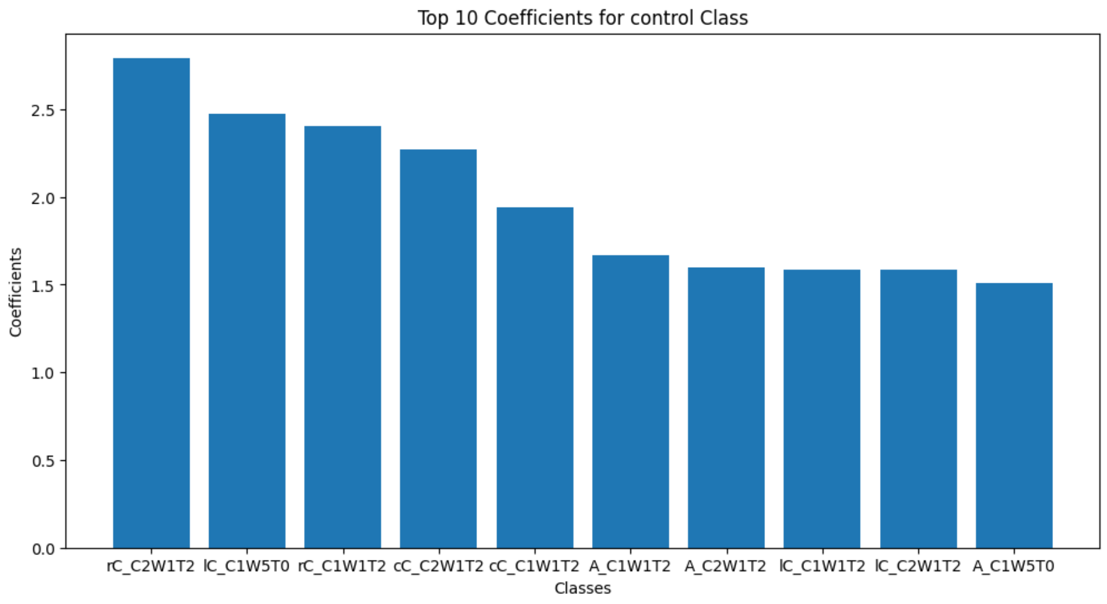




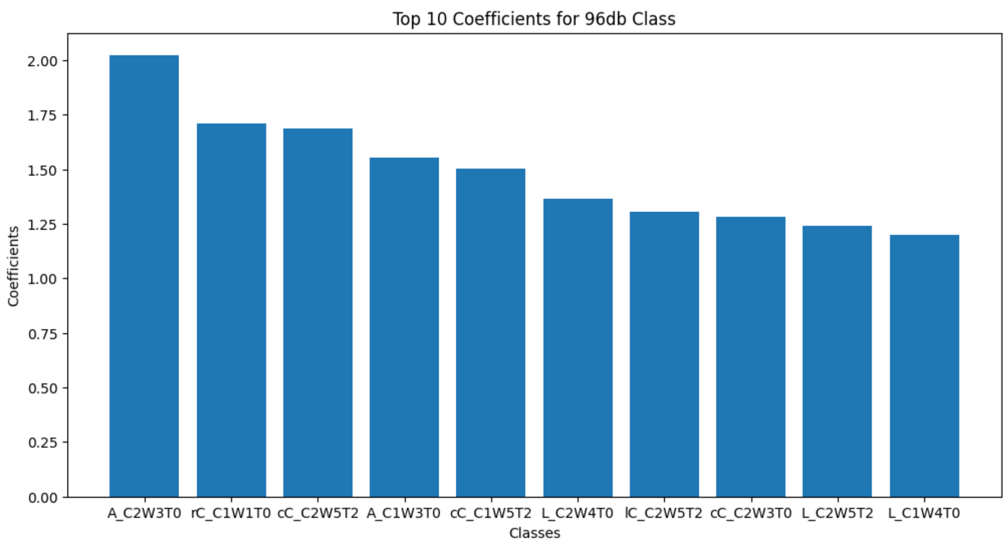
For class 96db there were 31 features with coefficient value of 1 and above.



We can also look at the top 10 features with the highest coefficients in each class.



\

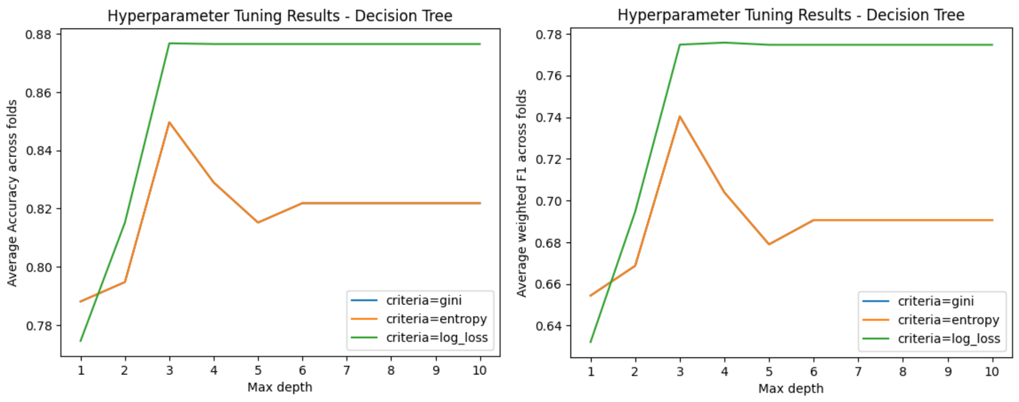


2. DECISION TREES

The two hyperparameters chosen for tuning are max_depth and criterion. Max_depth determines the maximum depth the tree can have. Trees with high depth can tend to get less interpretable and can sometimes overfit to the data so the values are taken from 1 to 10 to ensure lower depth trees are chosen. The criterion decides the splitting criteria for nodes as discussed earlier in methods.

Hyperparameter	Values
max_depth	1,2,3,4,5,6,7,8,9,10
criterion	Gini, log_loss , entropy.

Results:

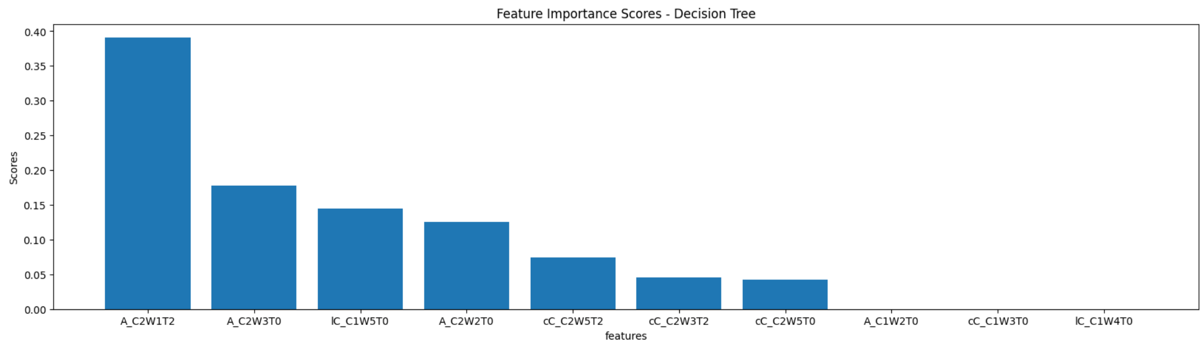


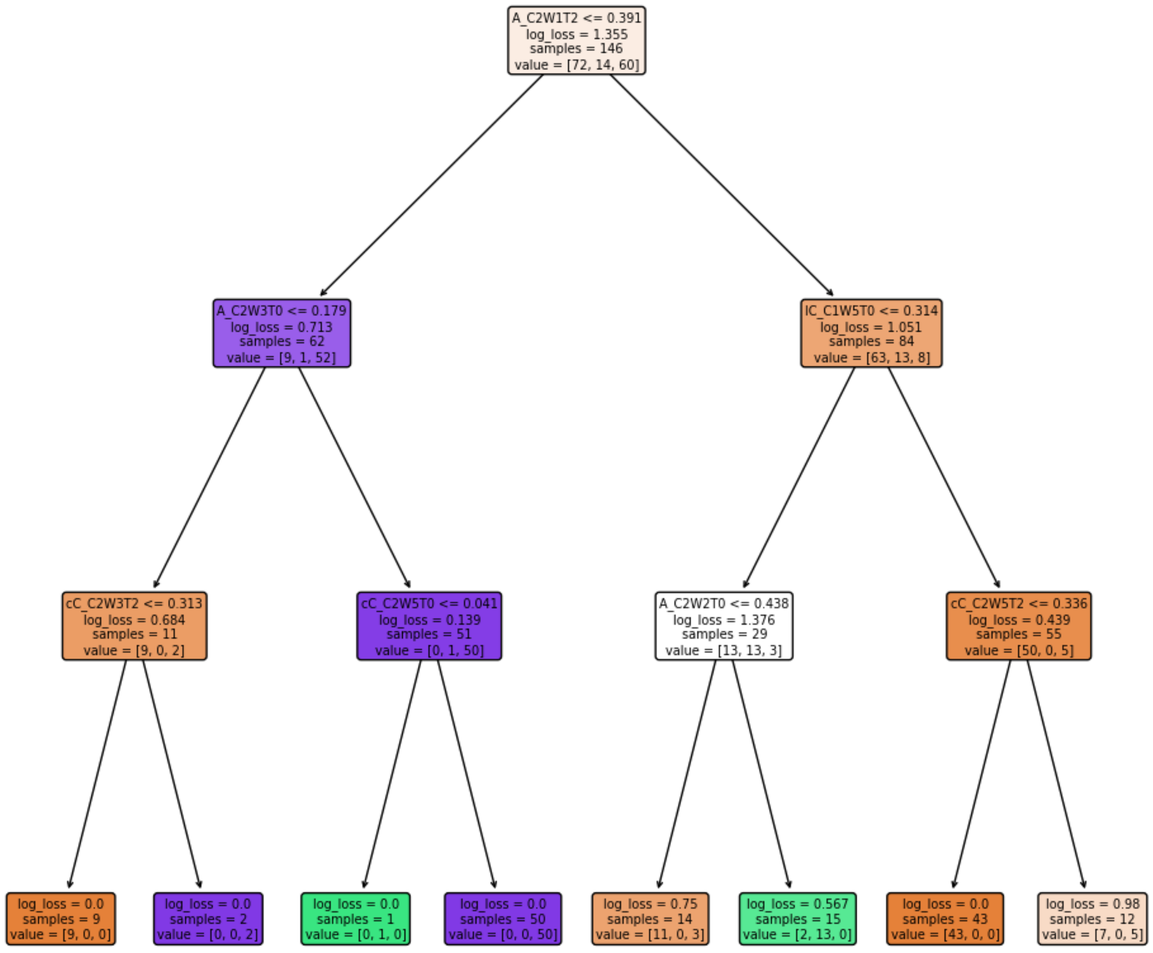
Both the graphs showing predictive accuracy and weighted f1 scores are similar to each other where the performance is better with the log_loss criteria for splitting. The best performance is also seen when the max depth is 3 which is small enough to be interpretable.

INTERPRETABILITY OF THE MODEL:

The advantage of decision tree models (especially trees with a lower depth) is that it can easily be visualized and explained using the tree structure. The model translates to an if else structure. From just observing the tree structure we can

make conclusions for example one way the model predicts control class if the feature A_C2W1T2 <= 0.391 and A_C2W3T0 <= 0.179 and cC_C2W3T2 <=0.313. The final model only uses 7 features for making predictions. This is a reduction from 103 to 7! Even though the overall accuracy did go down a little since it is a newer dataset with limited understanding or information regarding it, a more interpretable method would be ideal.
The decision tree classifier also gives a feature importance score for the features:

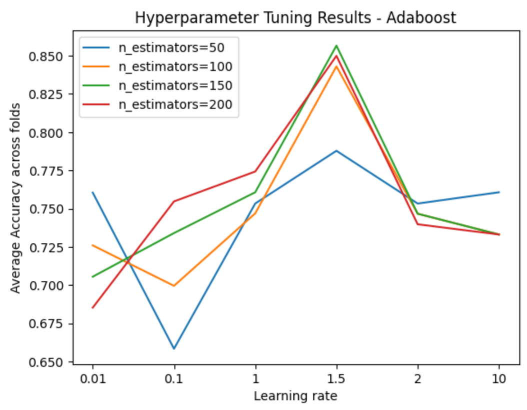


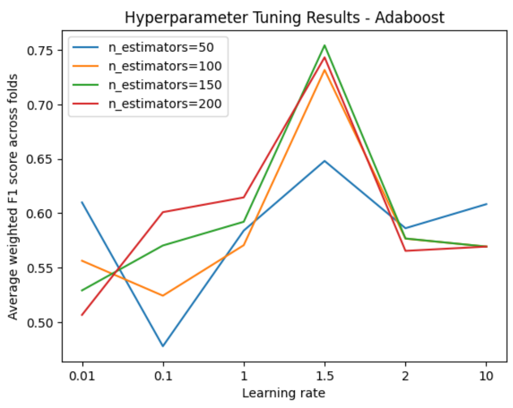


3. ADABOOST

The two hyperparameters chosen are the learning rate which determines how fast the model learns new weights (estimator weight update as shown in equation (1)). The second parameter is how many estimators or weak learners we need to get good predictive accuracy.

Hyperparameter	Values
Learning_rate	0.01, 0.1, 1, 1.5, 2, 10
n_estimators	50, 100, 150, 200

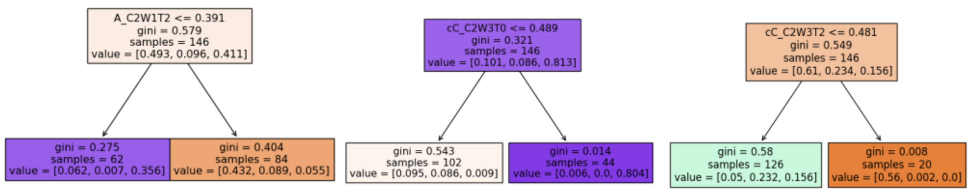




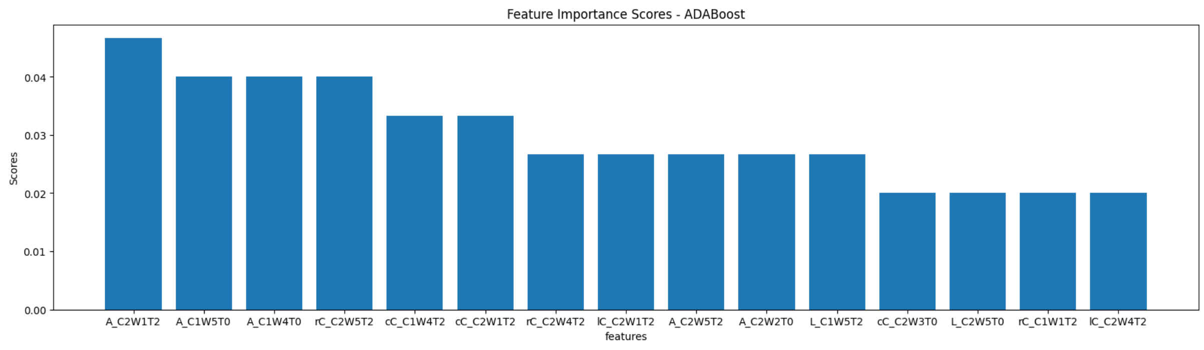
From comparing the plots, we can see that the best predictive accuracy and the best weighted f1 score is achieved when the learning rate is 1.5 and the number of estimators is 150. We can also see that if we bring the number of estimators down to 50 the accuracy is quite low, this is probably because its not enough estimators to learn about all the features.

INTERPRETABILITY OF THE MODEL:

Although in this example 150 estimators might seem like a lot, each of these is a decision stump. We can visualize a few of these decision stumps.



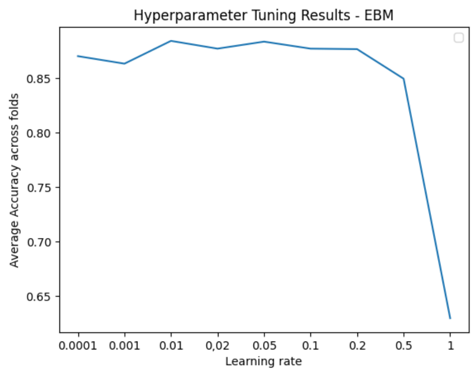
Just by looking at the distribution we can see that some estimators can classify one type of class. For example, the 3rd estimator which splits on cC_C2W3T2 has managed to bring most of control class data points to the same group. The second estimator can group most of class 96db datapoints. The predictive accuracy of this model increases when we add up these different estimators. When we take the number of non-zero features from the feature importance function, we get a total of 65 which is more than what we saw in the decision tree example and hence does increase the complexity of the model a little. This plot depicts the top 15 features chosen by the model.

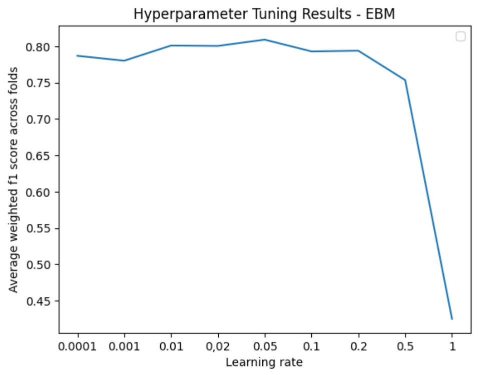


4. EXPLAINABLE BOOSTING MACHINE

The only hyperparameter that I have used for tuning the EBM model is the learning rate. The learning rate of the EBM model should be very small for the algorithm to work properly. So we are expecting to see a decrease in the accuracy with an increase in the learning rate.

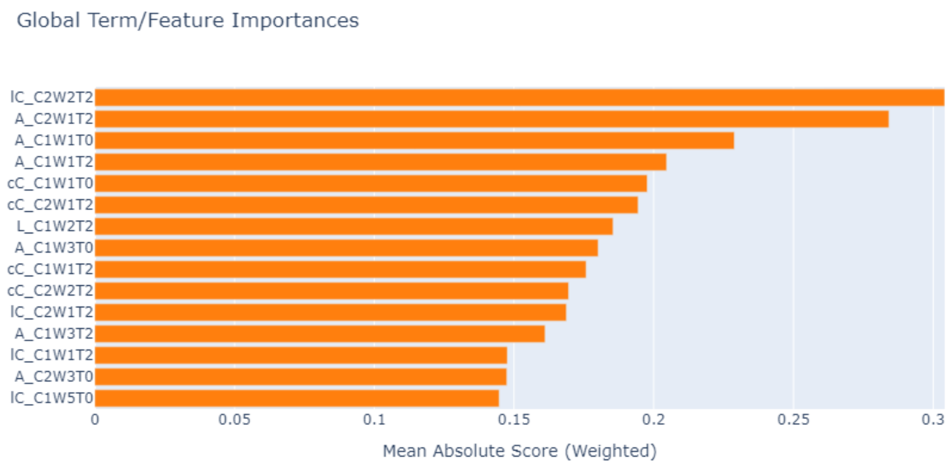
Hyperparameter	Values
Learning_rate	0.0001,0.001,0.01,0.02,0.05,0.1,0.2,0.5,1



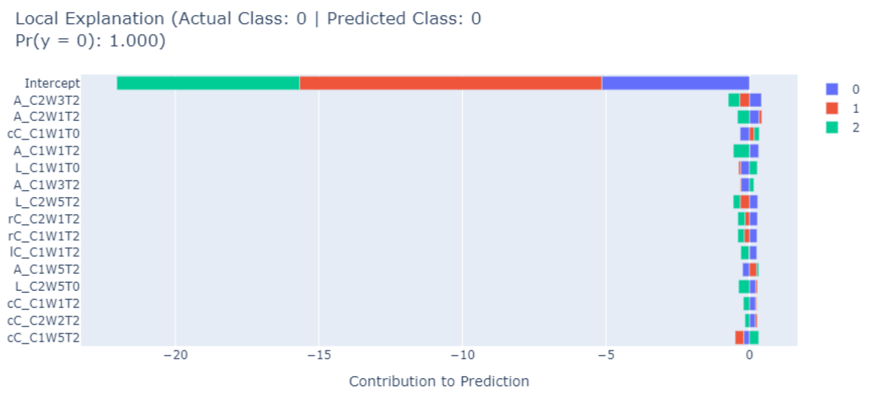


INTERPRETABILITY OF THE MODEL:

EBM comes with the ability to visualize the model globally and locally. The global summary gives the feature importance scores which are the mean absolute contribution each feature makes to the predictions averaged across the training dataset. The contributions are weighted by the number of samples in each bin.



Local explanations help understand the breakdown of how much each sample contributed to the prediction of a single sample. The intercept is the average y value of the train set, for classification problems it is the log of the base rate.



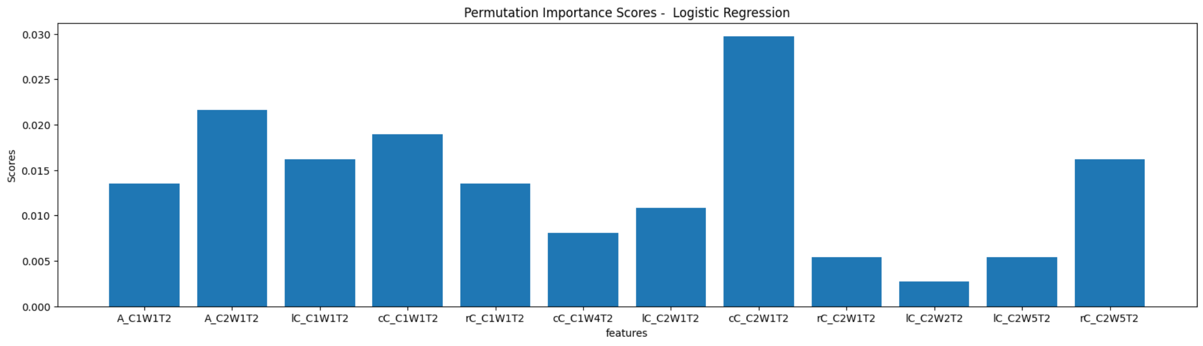
3.4 VARIABLE IMPORTANCE ANALYSIS

Library Used: sklearn.permutation_importance [5]

Features that are deemed of low importance for a bad model (low cross-validation score) could be very important for a good model. Therefore, it is always important to evaluate the predictive power of a model using a held-out set (or better with cross-validation) prior to computing importance. Permutation importance does not reflect to the intrinsic predictive value of a feature by itself but how important this feature is for a particular model, so because of this the features in this section may vary from the features we obtained through feature importance or term importance score.

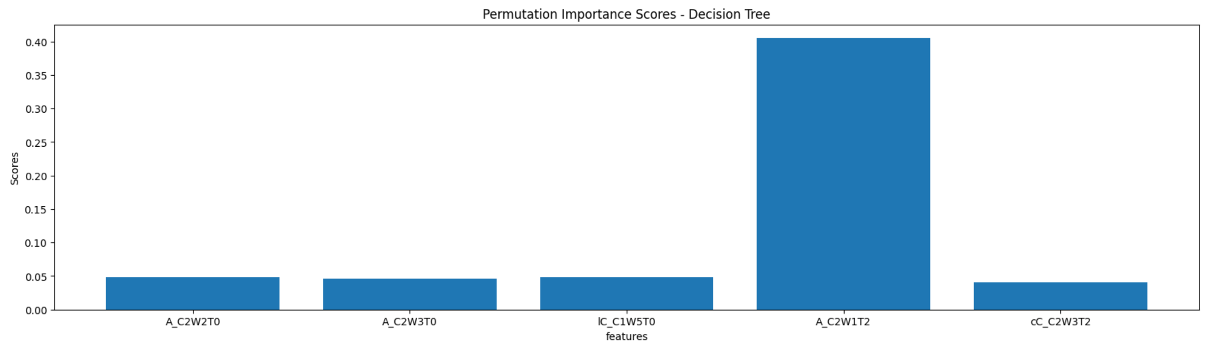
LOGISTIC REGRESSION

A total of 12 scores were given as the output of the permutation importance calculation. Most important feature is cC_C2W1T2 and the second most important feature is A_C2W1T2



DECISION TREE

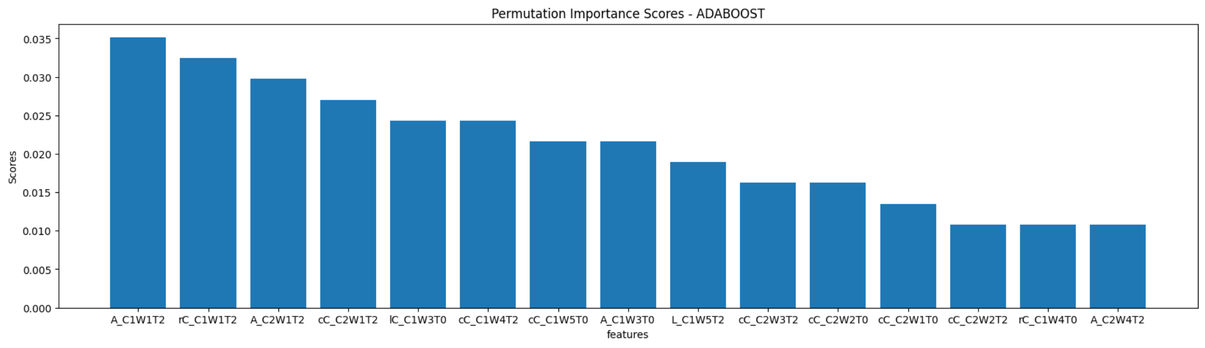
The permutation importance scores are computed for the best decision tree model, and it gave a total of 5 important features making decision tree model the sparsest among the four models.



The most important feature here is A_C2W1T2, this is also the parent node for the best tree which is intuitive.

ADABOOST

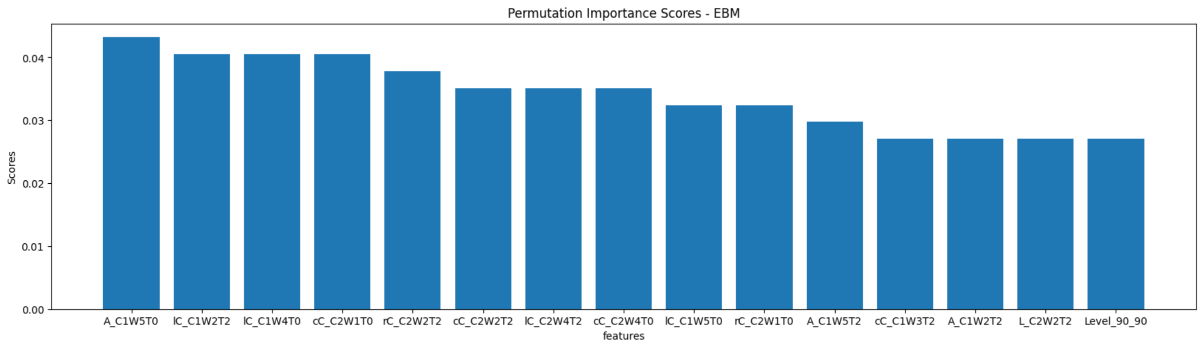
The permutation importance score is computed for the best adaboost model and it gave a total of 24 importance scores. This is a high number to plot so we can reduce it down to the top 15.



EXPLAINABLE BOOSTING ALGORITHM

Apart from the term importance scores seen earlier under interpretability, we can also compute the permutation importance score for EBM.

When we compute the permutation importance scores, there are 67 features out of 103 features that have a mean importance score. This number is probably higher because it is an ensemble model of classifiers trained on only on feature at a time and might need the culmination of more features to give good results. We can list out the top 15 important features.



Here we can notice that the most important feature of the model's logistic regression, decision tree and adaboost is only ranked 13th in the variable importance scores for EBM model. Maybe one of the reasons why EBM is able to perform much better is its ability to understand the features and their relationships to the target classes better. Based on the common features among the four models we can see that wave 1 is very frequently identified as important across the different metrics like cC, rC, lC and A.

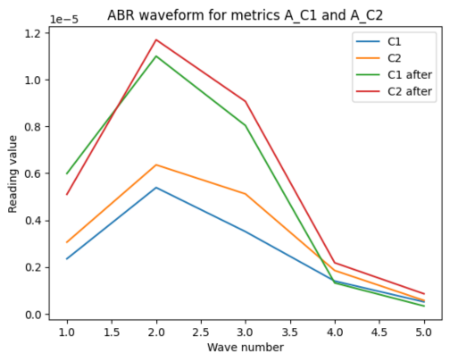
DT (common with LR)	Adaboost (common with LR)	EBM (common with LR)
A_C2W1T2	A_C2W1T2	A_C2W1T2
	A_C1W1T2	A_C1W1T2
	cC_C1W1T2	cC_C1W1T2
	cC_C1W4T2	cC_C1W4T2
	cC_C2W1T2	cC_C2W1T2
	lC_C2W1T2	
		lC_C2W2T2
		lC_C2W5T2
	rC_C1W1T2	rC_C1W1T2
	rC_C2W1T2	
	rC_C2W5T2	

Adaboost (common with DT)	EBM (common with DT)
A_C2W1T2	A_C2W1T2
	A_C2W2T0
	A_C2W3T0
cC_C2W3T2	
	lC_C1W5T0

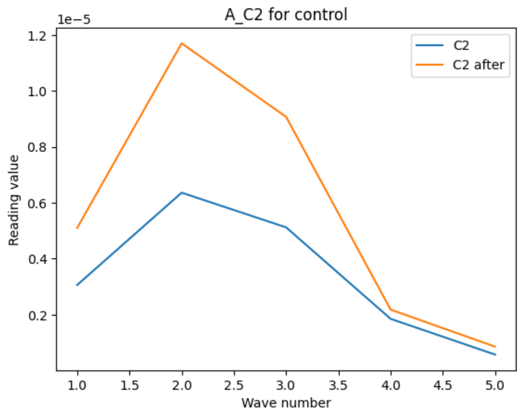
Features common between Adaboost and EBM
'A_C1W1T2','A_C2W1T2','A_C2W4T2','L_C1W4T2','L_C1W5T2','L_C2W5T2','cC_C1W1T2', 'cC_C1W3T2','cC_C1W4T2','cC_C1W5T0','cC_C2W1T0','cC_C2W1T2','cC_C2W2T2', 'lC_C1W2T2','lC_C1W3T0','rC_C1W1T2','rC_C1W4T0','rC_C2W3T2'

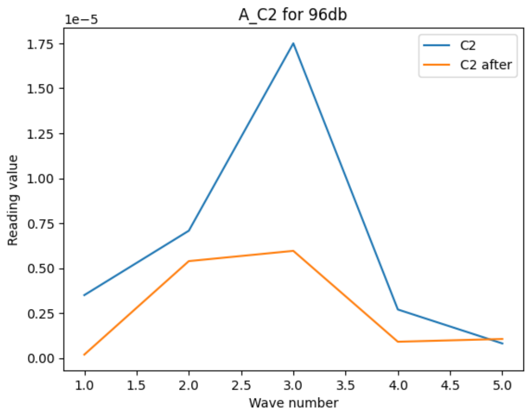
5. INSIGHT

The feature that was there in the importance scores for all the four models was A_C2W1T2.



From looking at this graph we can see that before the noise exposure the value of A_C2W1T0 is around 0.3 and after the noise exposure the value has increased to 0.5. But the same increase can be seen in wave 2 also, so what makes this feature so important for classification purposes?





These two waveforms represent the feature for control and for 96db class. We can see that for control the value after noise exposure increases but for the animals exposed to 96db, this feature after noise exposure decreases. So based on analysis so far, the most important ABR metric is A_C2W1T2.
Waves I and II are generated by the vestibulocochlear nerve. Wave III is generated by the neurons in or near the cochlear nucleus. Wave IV is generated by neurons in the superior olivary complex in the pons and the cochlear nucleus and nuclei of the lateral lemniscus. Wave V is generated by neurons in the contralateral caudal colliculus of the brainstem. Wave V is considered to be the most important while predicting the presence of ABR. But in our case most of the important features were associated with wave 1 and this could be because in our case we need to not only detect presence but also classify the type of noise based on the decibel. Although EBM model gives wave 5 as the top feature.

Another observation is that except for EBM, the other models could not predict the results of class 91db that well due to the huge imbalance in the data. In the future it would be beneficial for the model to be trained on balanced data.

6. ERRORS AND MISTAKES

The hardest part of this project was understanding the dataset and the features. I had referenced a couple of videos and papers for this project but none of them could exactly explain the metrics like A,L,rC,lC,cC. But at the same time, I was too curious to not give up on this dataset and continued working on it hoping the results would provide some insight into the features. Most of the papers mentioned how the left ear and right ear readings should be similar based on these and initial analysis I have concluded that left and right ear could correspond to the C1 and C2 readings. I also wanted to use sparse decision trees for this project but kept running to ram issues on colab which is surprising as it is a very small dataset but has large number of features.

7. CITATIONS

[1] Varoquaux, G., & Pedregosa, F. (n.d.). *sklearn.linear_model.LogisticRegression*. scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[2] Louppe, G., & Prettenhofer, P. (n.d.). *sklearn.tree.DecisionTreeClassifier*. scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[3] Dawe, N., & Louppe, G. (n.d.). *sklearn.ensemble.AdaBoostClassifier*. scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[4] Caruana, R., & Lou, Y. (n.d.). *ExplainableBoostingClassifier — InterpretML documentation*. InterpretML. https://interpret.ml/docs/ExplainableBoostingClassifier.html

[5] *4.2. Permutation feature importance*. (n.d.). scikit-learn. https://scikit-learn.org/stable/modules/permutation_importance.html