Final Project

Xinyi(Louise) Li

1282080

1.1 Introduction

The dataset for the project is the Animal Noise Exposure Dataset, which is provided. There are many interesting details that are worth digging into, and there is little research related to the problem. We can do as much analysis as we want on the track using diverse techniques and algorithms. Moreover, the dataset is very clear, most of the variables(columns) have meanings and can be treated as useful features. As for the question itself, auditory is one of the most important abilities for creatures, especially for human beings. Using animal experiments can give people ideas on the influence of noise exposure and generate more research on protecting human being's hearing in the future.

The most important and interesting thing to analyze in this dataset should be whether animals under the noise exposure will influence their ABR metrics?

This question can be analyzed by another way, can we use the metrics data to classify whether the animal is under any noise exposure? If so, then the noise exposure will influence the performance of animals. That's the problem we will solve in this project report.

1.2 Data Description

The dataset has 183 rows and 104 columns, each row indicating one subject's experiment result under different conditions. Each column records each experiment's specific condition setting or result. The columns can be treated as four different parts, including the noise exposure groups (including control, 91db, and 96db), stimulus levels(including 70~70, 70~90, and 90~90), Auditory Brainstem Response(ABR) metrics before the experiment (column names ended with T0), and Auditory Brainstem Response(ABR) metrics after the experiment(column names ended with T2).

There is only one column for the groups and stimulus levels, and 50 columns for ABR metrics before the experiments and 50 columns for ABR metrics after the experiments. More importantly, these columns can be used for contrast because they have the same time scale.

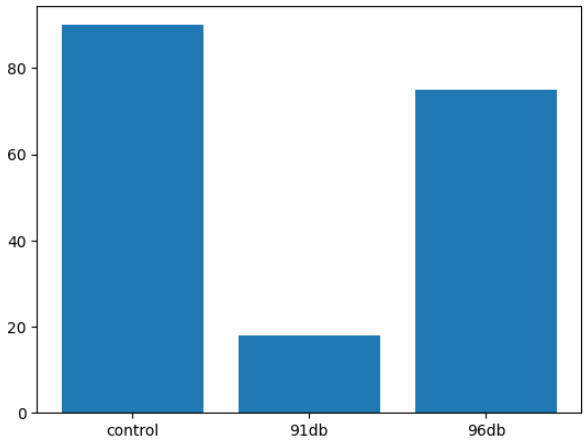
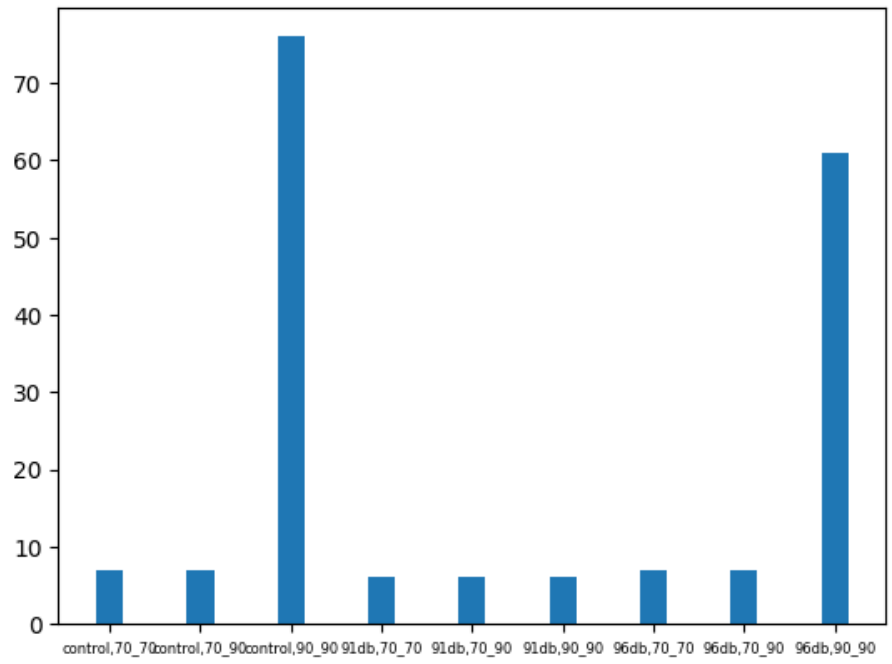
Going back to the question we have raised in Introduction, we can separate it into two questions. Can we use the metrics data before the experiments and after the experiments to classify the noise exposure group? Can we use the metrics data after the experiments to classify the noise exposure group? These two questions are both multi-class classification problem in machine learning.

However, before doing so, we need to do some data cleaning.

SubjectID and Run are two features(columns) that make less sense to this question, so we will simply drop them. These two variables are related to the experiment's progress, instead of the experiment's content itself.

To visualize the dataset, I use histograms to display the number of objects within each group(control, 91db and 96db) and the number of objects within each pair of group and stimulus

level. Accordingly, we can see the data is imbalanced, there are much more objects in the control and 96db groups than 91db.

Class Distribution	
Class Distribution Pair	

At the same time, we can find there are some missing values. There are 76 rows that contain at least one missing value. Moreover, 'A_C2W5T2', 'A_C2W5T0', 'rC_C2W5T2',

'lC_C2W5T2, 'cC_C2W5T2', 'L_C2W5T2' are the top six variables that lose the most. These are very important variables because many of them have recorded the results after the experiments. We cannot simply drop these columns.

It is neither a good way to drop the rows with missing values. We only have 183 rows which is not a large dataset, so we should keep every experiment result and try to impute them.

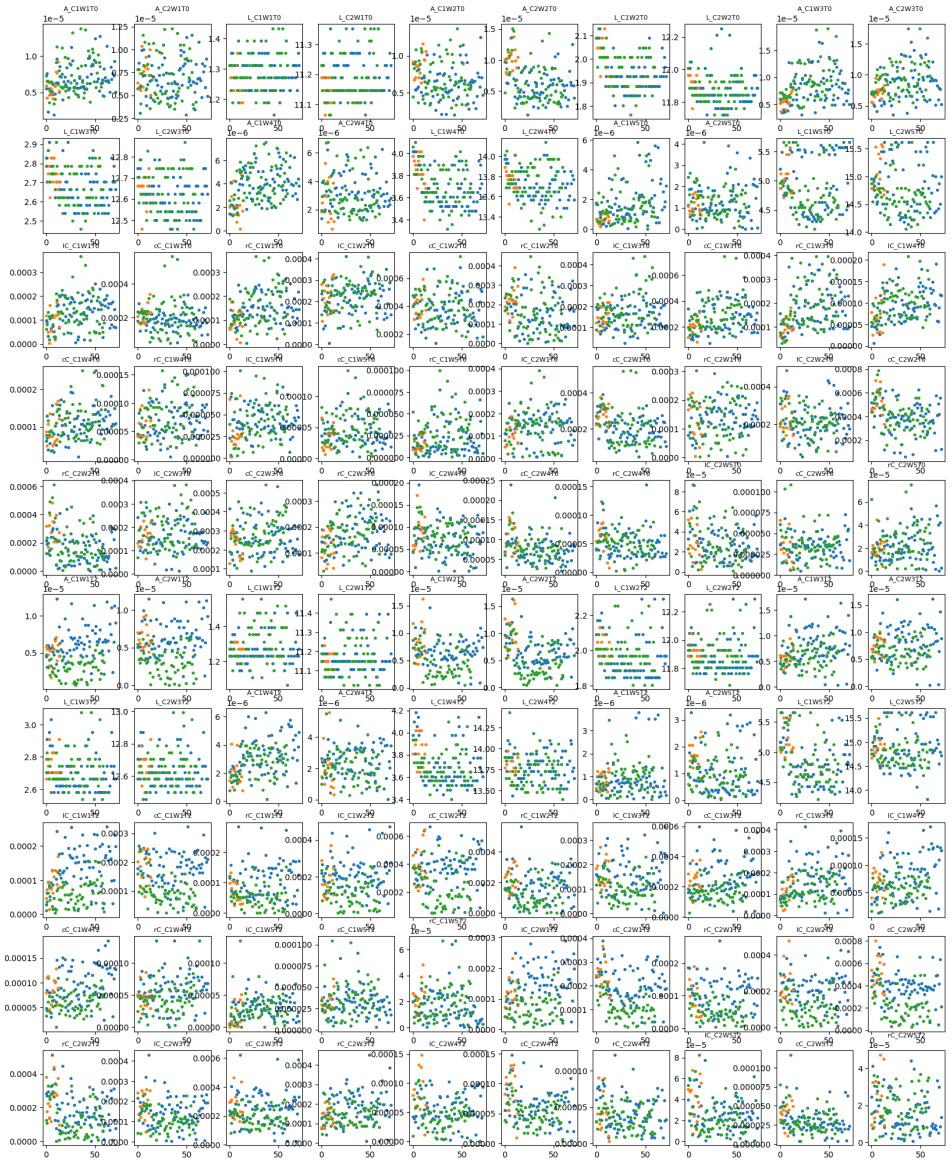
To solve the two problems raised above, we should separate the current data to train set and test set at first, to prevent any possible data leak.

The train set and test set should be kept consistent in the following analysis, so we can set a seed to maintain it. We will not lose the original data if any accident code breaks the datasets. The algorithm I choose is train_test_split from sklearn.model_selection, which we have already used multiple times in class, a safe function to separate data into train and test.

Before considering the problem of imbalance, I want to impute the missing values first. The model I choose here is KNNImputer from sklearn.impute. It will use the mean value from its 6 closed neighbors. The dataset is not that large, so six should be appropriate to handle outliers. The model will impute the missing values from the train set and test set separately, so it should not have any data leak. However, even though stimulus level should be an independent variable as metrics in this problem, I narrow down the neighbor choices within the pair of noise exposure groups and the stimulus level because level is also categorical and it does not contain any missing value.

After handling the missing values, we can make scatter plots to see the dirstibutions of each feature. From the scatter plots of the variables, lots of the values of T2 features from group 96db are dirstibuted under those of the values from group control. Therefore, we can make

assumption that if subjects are under 96db's noise exposure experiment, they will have lower metrics in some of the T2 variables.



Train dataset distribution for each feature

To deal with the imbalanced data, we can only handle it on the train dataset. There are three ways to do that, including adding weight to different classes, in this question, adding weight on the 91db group, oversampling or undersampling by copying or deleting objects, and using the SMOTE function from imblearn.over_sampling.

Each of them should be valid, and I use the SMOTE function in this question because it is easy and efficient. It is an oversampling method that increases objects in the smaller size class.

We also need to make some modifications to the categorical features, as they are strings, which will be inappropriate to make classifications in some algorithms models.

There are two different ways to do that, one is to use the LabelEncoder, which will convert the array into numbers, or use the get_dummies to change the feature into different binary variables. In different algorithm models, I use either of them.

If using the LabelEncoder, we will get 0,1,2 to represent different classes. 0 represents the group of 91db, which has the smallest size in the train dataset; 1 represents the group of 96db; 2 represents the group of control.

2. Methods

External Libraries:

```
   1. XGboost
from xgboost import XGBClassifier
from xgboost import plot_importance
   2. Adaboost
from sklearn.ensemble import AdaBoostClassifier
   3. RandomForest
from sklearn.ensemble import RandomForestRegressor
   4. Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import ComplementNB
   5. SVM
from sklearn import svm
   6. Neural Network
!pip install scikeras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.optimizers import SGD
from scikeras.wrappers import KerasClassifier
   7. Logistic Regression
from sklearn.linear_model import LogisticRegression
```

Algorithm 1: XGBoost

"A connection is made between stagewise additive expansions and steepest descent minimization. A general gradient descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least absolute deviation, and Huber-M loss functions for regression, and multiclass logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression" (Friedman).

XGBoost is a gradient boosting decision tree model that will use the previous tree model to make improvements that can raise the accuracy. It will be built sequentially. It is a good method to deal with an imbalanced dataset, because it can raise a class's weight if it has failed to predict one class. In this problem, which we have an imbalanced dataset, we should use this algorithm to make classifications. Moreover, it is good at dealing with overfitting. It will stop adding new leaves if the gain difference obtained by future split is small enough. It also uses L2 regularization which is useful for preventing overfitting. It is also faster due to the greedy optimization algorithm. It is also available to use L1.

Besides these benefits that the algorithm itself brings us, it is easy to import the package and make hyperparameters tuning. It is useful in multi-class classification problems.
For this algorithm, I have tuned the learning rate, max_depth, lambda and n_estimators, which are the common and influential parameters in the XGBClassifier.
In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame

is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 2: AdaBoost

"AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1 \ldots T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted $D_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set" (Freund and Schapire #771).

Adaboost is also a boosting decision tree model, but it will only use one level of nodes, which is called stump. Therefore, it will form a stump forest. It is a weak learner and it is good at dealing with overfitting problems. Same as the XGBoost, it is a sequential model that will update the weight if it misclassifies one sample, but focusing on the misclassified sample more. Some stumps have more weight than others. Moreover, it is also convenient to import and apply as it is from the sklearn package.

For this algorithm, I have tuned the learning rate, algorithm, and n_estimators, which are the common and influential parameters in the AdaBoostClassifier.

In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 3: Random Forest

"Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them" (Breiman #5).

Unlike the previous two algorithms, though it is also based on decision trees, random forest is built by parallel decision trees that will update weight from previous models. The final result will be decided by the majority vote of the trees, which is useful in accuracy classifications. It is an ensemble learning model, and each tree should be built independently and the errors should also be independent. However, it may have an overfitting problem, so I want to

use this result to compare with the previous two algorithms to compare its performance and the other two. It is easy to apply too, which is also from the sklearn package.

In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 4:Naive Bayes

"Naive Bayes is an algorithm that learns the probability of every object, its features, and which groups they belong to. It is also known as a probabilistic classifier. The Naive Bayes Algorithm comes under supervised learning and is mainly used to solve classification problems" ("Naive Bayes Algorithm in ML: Simplifying Classification Problems"). There are many sub tracks in Naive Bayes methods and can be used in different situations when distributions change. Therefore, I have considered two of them, Gaussian Naive Bayes which is the most common one for classification problems, and Complement Naive Bayes which can be treated for an imbalanced dataset. We will get the probability for each of the classes and then choose the class with the largest probability. It is common for many situations without limitations, but the

prediction for the test case may not be that accurate. However, it is also easy to implement through the sklearn package.

In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 5: SVM

"Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are: Effective in high dimensional spaces; Still effective in cases where number of dimensions is greater than the number of samples; Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient ; Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels." (scikit-learn)

It is a deterministic supervised learning algorithm for classification and regression problems. We can maximize the boundary to make classification. However, it cannot directly do the multi-class classification, it should create multiple kernels to make classifications between each of the two classes, therefore, in order to identify the three classes, we need 2*3/2 = 3 models to do the classification, where I am using the one versus one approach.

Even though it is easy to import and apply the method from sklearn, it is still important to use the GridSearchCV to tune the parameters for the function.

In this algorithm, I choose to tune the parameters including the kernel type (rbf and linear), Gemma for rbf kernel, C for each of the kernels. Large C and low C can give different boundaries and we don't know which is better in this question, so we can tune each of them and compare the result. C is very important to match with Gemma as sklearn's official package suggests.

In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 6: Neural Network

"An artificial neural network (ANN) is a machine learning algorithm inspired by biological neural networks.8,9,21 Each ANN contains nodes (analogous to cell bodies) that communicate with other nodes via connections (analogous to axons and dendrites). Much in the way synapses between neurons are strengthened when their neurons have correlated outputs in a biological neural network (the Hebbian theory postulates that "nerves that fire together, wire together"), connections between nodes in an ANN are weighted based upon their ability to provide a desired outcome" (Choi #14).

Neural Network is also an important and strong method for multi-class classification. We can import it from the Keras package, which can also provide us with a function called SGD, which is an optimizer that can be used in gradient descent, and update the model. However, as we do not need to make auditory or image classification, we can simply add the dense layer with neurons and let the model run itself in different epochs. If the dataset is nice and the relationship is meaningful, we can expect to have a high accuracy for both the train dataset and the test dataset.

By using Neural Network, it is important to use data normalization and change imbalanced data to balanced, because the issue will be serious when the dataset is quite imbalanced like this dataset.

In this algorithm, we can tune the activation function, number of neurons in each layer, and the optimizer function. The activation function can lead to quite different results, and increasing the neurons can raise the accuracy of the training dataset but also may lead to an overfitting issue.

In Neural Network, we cannot simply use the GridSearchCV method as previous algorithms, but we can use another method called `KerasClassifier` and adding several parameters like epochs, batchsize to fit the model cross validation paramters, while we still need to manually calculate the cross validation score of each model. The model with the highest mean cross validation score will be chosen.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

Algorithm 7: Logistic

"Logistic regression is one of the most important analytic tools in the social and natural sciences. In natural language processing, logistic regression is the baseline supervised machine learning algorithm for classification, and also has a very close relationship with neural networks. The most important difference between naive Bayes and logistic regression is that logistic regression is a discriminative classifier while naive Bayes is a generative classifier" (Jurafsky and Martin #Ch5).

Logistic Regression also has strong power to make classifications in multi-class regressions which are based on probabilistic . It is high-efficient, easy to import from class sklearn, and has parameters including L1 and L2 regularization terms to deal with overfitting. I also use scaling and oversampling to do the training that will make the dataset more consistent and balanced.

Therefore, there are also many parameters that are worth tuning. For example, I am using penalty, C and tol to tune. However, I also want to tune the solver and multi_class parameters, but with the suggestion of the official page of this function, in a relatively small dataset solve can choose liblinear. Using liblinear also constrains the use of penalty to be either l1 or l2, and cannot use them at the same time, and the multi_class will automatically use ovr when the solve is liblinear.

In order to make the Parameters selection valid and accurate, I am using GridSearchCV method to help me with parameters selection. The metrics for choosing a model should be its mean score for cross validation, higher score will be better. As for the cross validation split, as my data frame is built by target value in order, shuffle should be important in order to get valid results. Therefore, I am using the KFold method to evenly divide the train dataset into 5 folds and choose random seed 1 to keep them consistent.

The metrics I used to evaluate the performance is the accuracy, F1 score, and the classification report of the test dataset, accuracy score is the most common way to reflect its performance; F1 score is better for an imbalanced dataset; classification report can show the details of each class's classification results.

3.1 Results

Algorithm 1: XGBoost

After the parameters tuning, I find the model with

```
{'learning_rate': 0.5,
 'max_depth': 4,
 'n_estimators': 50,
 'reg_alpha': 0,
 'reg_lambda': 1}
```

In the train dataset, we can find the accuracy and F1 score reaches 100%. After doing the test metrics evaluation, the accuracy and F1 score are close to 92% and 90%.

By using the default parameters, the accuracy and F1 score are also 100%, but as for the test case, its accuracy and F1 score are close to 95% and 94%, which out-performs the chosen parameters function after tuning. Therefore, I think this is because XGBoost can perform really well, some kind of overfitting. In the training dataset, it will cover some problems under the model, and it cannot compare which model is better when they get the same performance of the training dataset. However, it is really rare to get identical results.

Algorithm 2: Adaboost

After the parameters tuning, I find the model with

```
{'algorithm': 'SAMME', 'learning_rate': 1, 'n_estimators': 100}
```

Then we apply the tuned parameters to the AdaBoostClassifier Function, we can get 99.3% in both accuracy score and F1 score. Moreover, we can get 87% in the test dataset's accuracy and 86% in the F1 score.

Algorithm 3: Random Forest

After the parameters tuning, I find the model with

```
{'criterion': 'squared_error',
 'max_depth': 10,
 'max_features': 10,
 'n_estimators': 150}
```

By applying these parameters, we can set some parameters to each decision tree in the forest. Then, we get 99.5% accuracy and F1 score in the train dataset.

As we get the result of about 73% accuracy and 69% F1 score in test dataset, which are lower than the results from previous two algorithms, we can say random forest regressor cannot handle overfitting problem as well as the sequential learning models using random forest.

Algorithm 4: Naive Bayes

As for the Naive Bayes method, I get `[0.34, 0.33, 0.33]` priors which has the best score in tuning. I have tried to use SMOTE to deal with the imbalance issue and compare with the result of not using it. It gets the best score if using `[0.0958904109589041, 0.410958904109589, 0.4931506849315068]`. However, I find they do not make a big difference, while using the complement naive bayes method, the accuracy and F1-score are really low. By using the Gassuain Naive Bayes, and not dealing with imbalance issues, I get 80% accuracy and 80% in F1 score in the train dataset. After using SMOTE to deal with the imbalance issue, I get 88% accuracy and 88% in F1 score. The accuracy and F1 score in the test dataset should be both 73% and 72% which do not differ a lot by handling with imbalance data.

Moreover, we can find the classification rate of class 91db is low, which is about 40%. However, we can see the classification rates of class control and 96db are fine. F1 score can be dealt with an imbalanced dataset, but adding more samples to 91db in the train dataset to train the model, it does not affect the test dataset's metrics values.

Algorithm 5: SVM

After the parameters tuning, I find the model with

```
Best C: 1
Best Kernel: linear
Best Gamma: scale
```

Have the best performance, which can have 100% accuracy score and 100% F1 score in . At the same time, its accuracy and F1 score in the test dataset is both 95%.

By applying the chosen parameters model to the test dataset, we get a high accuracy and F1 score in the training dataset which is 100%, while the test dataset's performance is similar to the XGBoost, where the scores are both close to 95%. It can classify the group of 96db perfectly and has good predictive results to classify the control group and the 91db.

Algorithm 6: Neural Network

After changing its activation function and the optimization function, we can find when the activation function is Relu and the optimization function is Adam we can get the best result in the training dataset which is 96% for the mean cross validation score. If we build a new model and fit it to the training dataset, we can get 100% accuracy even before 10 epochs. While appling the neural network model to the test dataset, its accuracy is 95% and the F1 score is 94%.

Algorithm 7: Logistic

```
{'C': 1, 'penalty': 'l1', 'tol': 0.01}
```

These are the choen parameters after tuning in the constraints of using solve = liblinear and multi_class = ovr. In the training dataset, we can 100% in both accuracy and F1 score, while in the test dataset, accuracy is 92% and the F1 score is 93%.

Overall, I think XGBoost, SVM, Logistic do a really good performance in this dataset. Moreover, SVM and Logistic can also run faster than XGBoost, so SVM and Logistic are slightly better.

3.2 HyperParameter Selection

Algorithm 1: XGBoost

By using these parameters, I want to see whether using the L2 regularization term(reg_lambda) can help us prevent overfitting. I also want to know the effect of L1 regularization, so I am also tuning the L1 regularization term(reg_alpha). To find some features related to each tree inside the forest, I use max_depth parameter to find the optimized max depth in each tree and use n_estimators to find to do that. We can find reg_lamdba in this case is high, which is 1, so L2 regularization should be important to reach optimization.

However, after comparing the result of train accuracy and the test accuracy, I raised a question: is there an overfitting problem? Although the test accuracy and test F1 score are desired and good-standing results, but comparing to 100%, it also leads to some overfitting value, and I can also prove it by comparing the result of using the default XGBClassifier Function parameters. Besides, using cross validation can help reduce the overfitting problem.

As I have mentioned above, I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

Algorithm 2: Adaboost

By using these parameters, I want to see how the learning rate influences the accuracy of the model, because Adaboost is a sequential model and it can add weight by the misclassified

class, which is determined by the learning rate. Moreover, to find some features related to the number of trees in the forest, I use n_estimators to do that. Besides, I am also interested in the algorithm of SAMME and SAMME.R, which both focus on boosting but one is discrete boosting while the other is real boosting.

However, after comparing the result of train accuracy and the test accuracy, I raised a question: is there an overfitting problem? Similar to the XGBoost algorithm, its accuracy in the train dataset is close to perfect, but the test dataset's accuracy and F1 score is lower. Although they are lower than XGBoost results, they can still be considered as good-performance results. However, it also leads to some overfitting, where the gap between train and test is larger. In the training dataset, it will cover some problems under the model, and it cannot compare which model is better when they get the same performance of the training dataset.

As I have mentioned above, I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

Algorithm 3: Random Forest

As for the Random Forest, similar to Adaboost which is also based on ensemble decision trees, I have also tuned some parameters for each tree. However, Adaboost also requires the learning rate to reevaluate the model, while the original Random Forest does not. I have used n_estimators to find the number of trees in the forest, max_depth for each tree, max_features for selection to consider, and criterion to reduce the L1 and L2 loss.

As for the criterion, absolute error can reduce the L1 loss while the squared error and reduce the L2 loss, as we cannot apply it on the same time, so I am wondering which one should be better in this dataset.

As I have mentioned above, I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

Algorithm 4: Gaussian Naive Bayes

Gaussian Naive Bayse has few parameters that can be tuned so I am only tuning priors in this algorithm. I have tried to use the None, evenly divided possibility of three classes, and the possibility of each class in the train set as the priors' parameters. However, I find the result that do not differ greatly so it is not important to do that in this case.

As I have mentioned above, I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

Algorithm 5: SVM

There are many different methods in SVM so I want to tune the most important features. First of all, the kernel. There are two major types of kernel, linear and nonlinear. There are several branches inside of nonlinear, in order to save the time, I choose the common one rbf. Therefore, I will train to use linear and rbf as two options for the kernel. As for both of the

kernels, we have to train C, which can tune for L2 penalty. Moreover, for the nonlinear kernel, there is another parameter called gamma, which is the coefficient for the kernel which will take effect on the boundaries.

As I have mentioned above, I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

Algorithm 6: Neural Network

There are also many parameters thet can be tuned for Neural Network, including the number of neurons each layer, number of layers, activation function each layer, optimizer function, batch size, epochs, and so on. In order to save time and solve it in efficiency, I only choose to tune the number of neurons in each layer, activation function and the optimizer function. Therefore, there are some options. For each layer, excluding the last layer, I only choose 50 and 100 neurons to tune, because I want to get a high score and also not receive overfitting problems, so I am avoiding having too many neurons, even though there are many features that are inputted. Moreover, I only picked three layers because only three classes needed to be classified. As for the activation function, I have two options, one is sigmoid and one is relu. Relu is really common for multi-class classification problems, while sigmoid is a good method for binary classification. The last choice is the optimizer function, Adam and SGD. We have used SGD at homework but Adam is also a nice optimizer for multi-class classification problems. Therefore, I write a function to create a Neural Network model with different parameters. I also directly use epochs = 30 and batch size = 16 to do the tuning, and cross

validation folds are consistent with other algorithms. The model with the highest mean score will be chosen.

I find though sigmoid and SGD are both algorithms that have nice performance using relu and Adam, they do not have a good performance while using the same time in the same model.

There is some overfitting problem but optimizer function and cross validation for parameters tuning have already reduced with it. Moreover, the loss is small, so it should be fine.

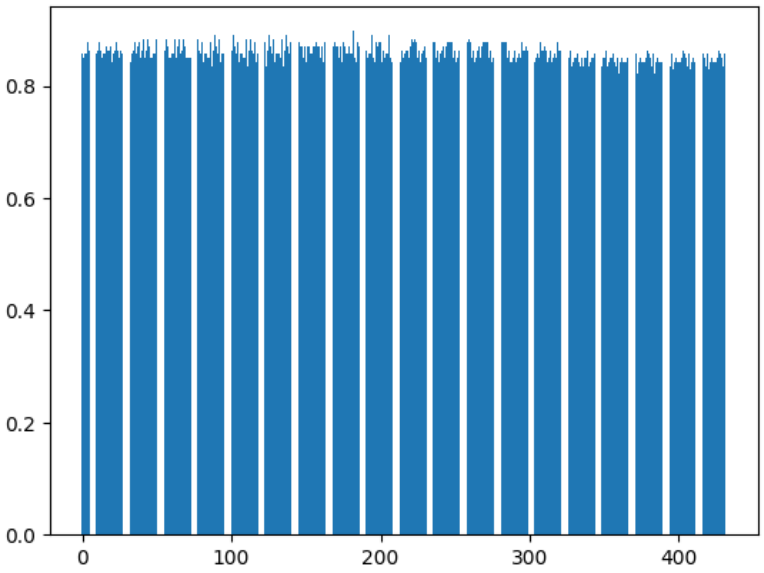
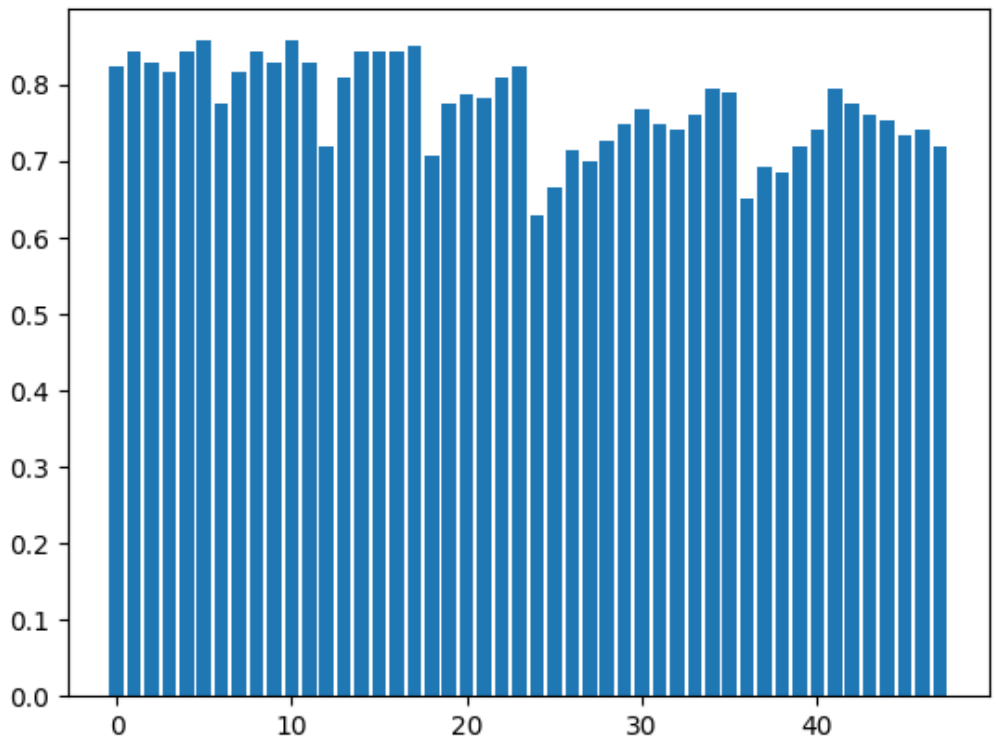
Algorithm 7: Logistic

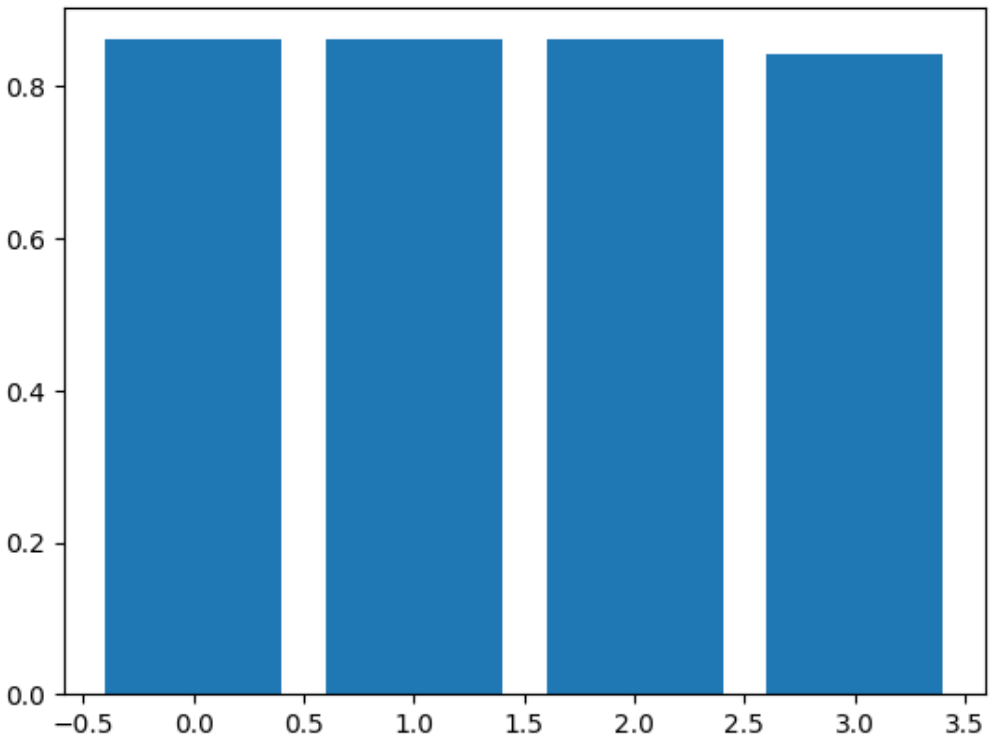
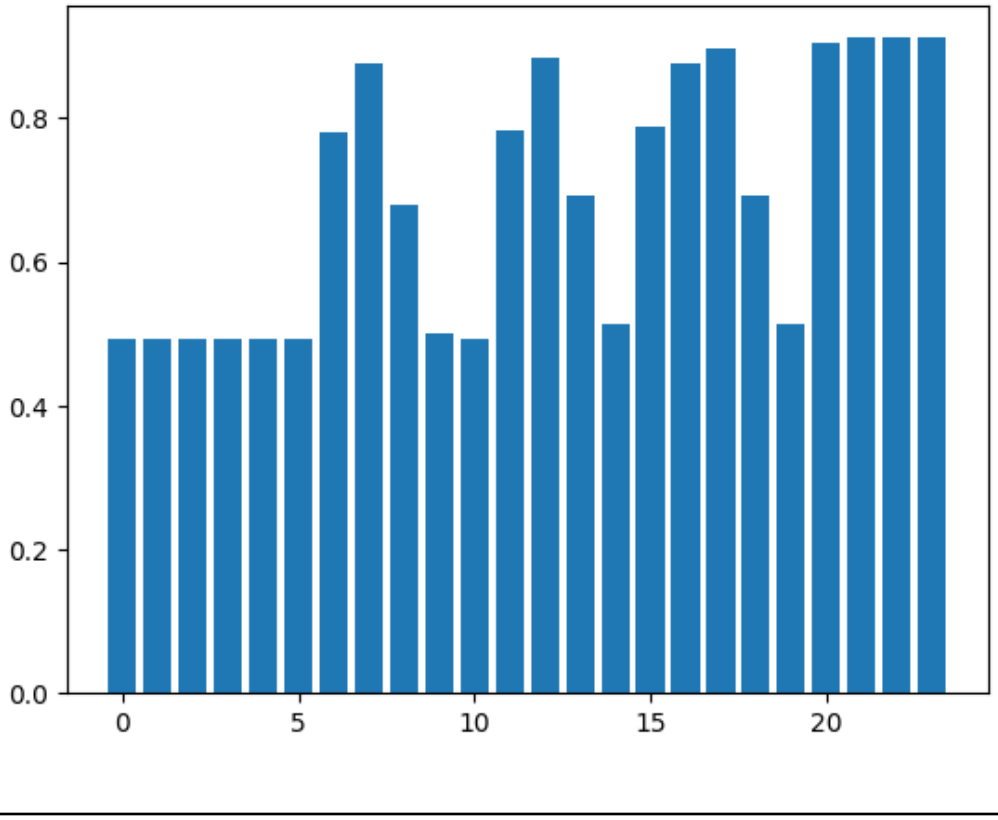
As I have mentioned above in Methods, I tune the penalty, C, and tol to train the model and select the best parameters. C is the inverse of regularization strength, as it gets smaller, it will have stronger regularization. Tol is the tolerance for the stopping criteria, which can be used to deal with the overfitting problem. I find the Logistic Rregression function has some constraints on each of the parameters, and lots of them cannot be used at the same time, C and tol are two general parameters that will not impact other parameters's selection.

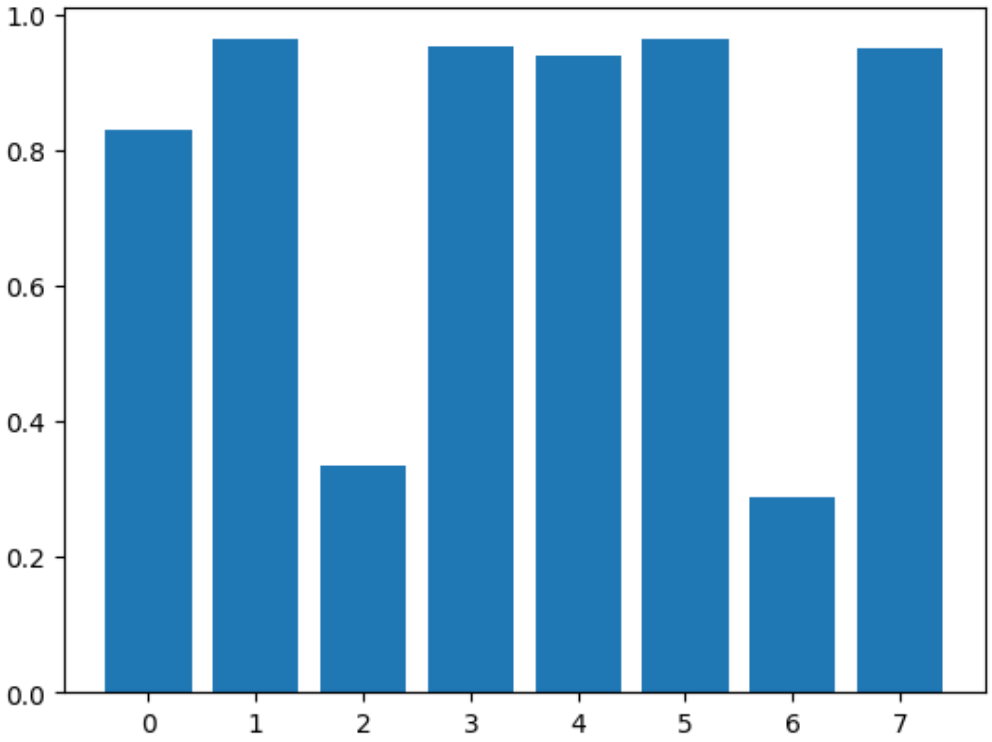
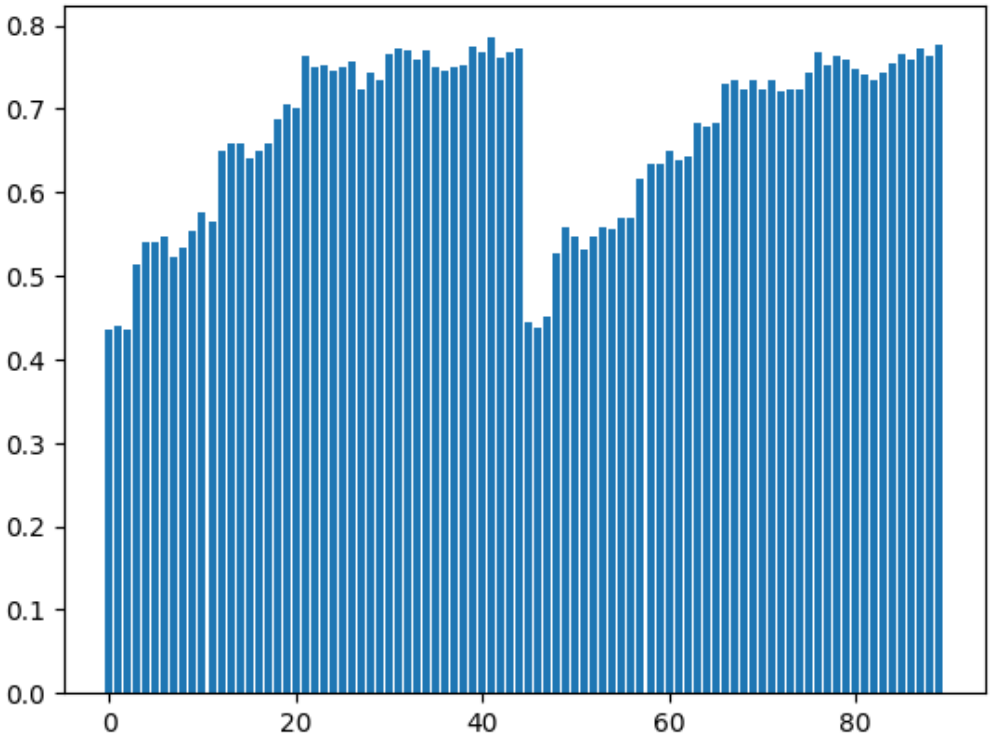
I am splitting the training dataset into five folds randomly in parameters tuning in GridSearchCV, but using the same random seed so the data can be consistent. The model with the highest mean test scores among the five validation groups will be selected.

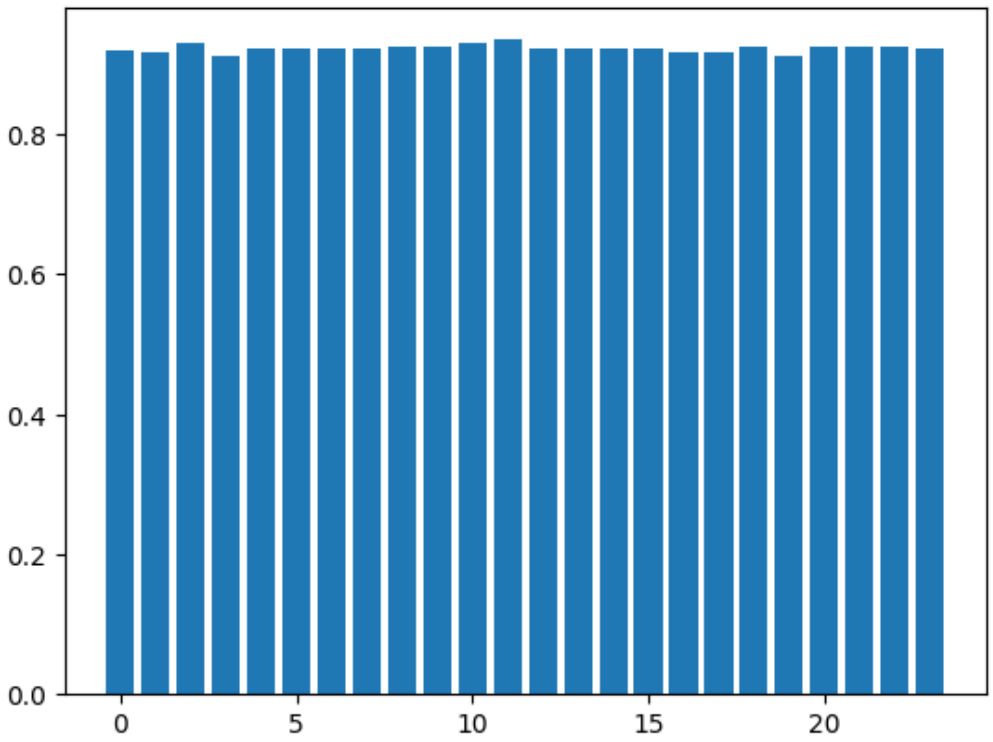
The training accuracy and F1 score are really high, which are similar to the previous algorithms that a small dataset may be the reason. Although the test accuracy and F1 scores are really high, the overfiiting problem still exists. However, with the cross validation, penalty and tol of the model, the overfiiting problem should have already been reduced. It can classify the group of 96db perfectly and has good predictive results to classify groups of control.

3.3 visualization

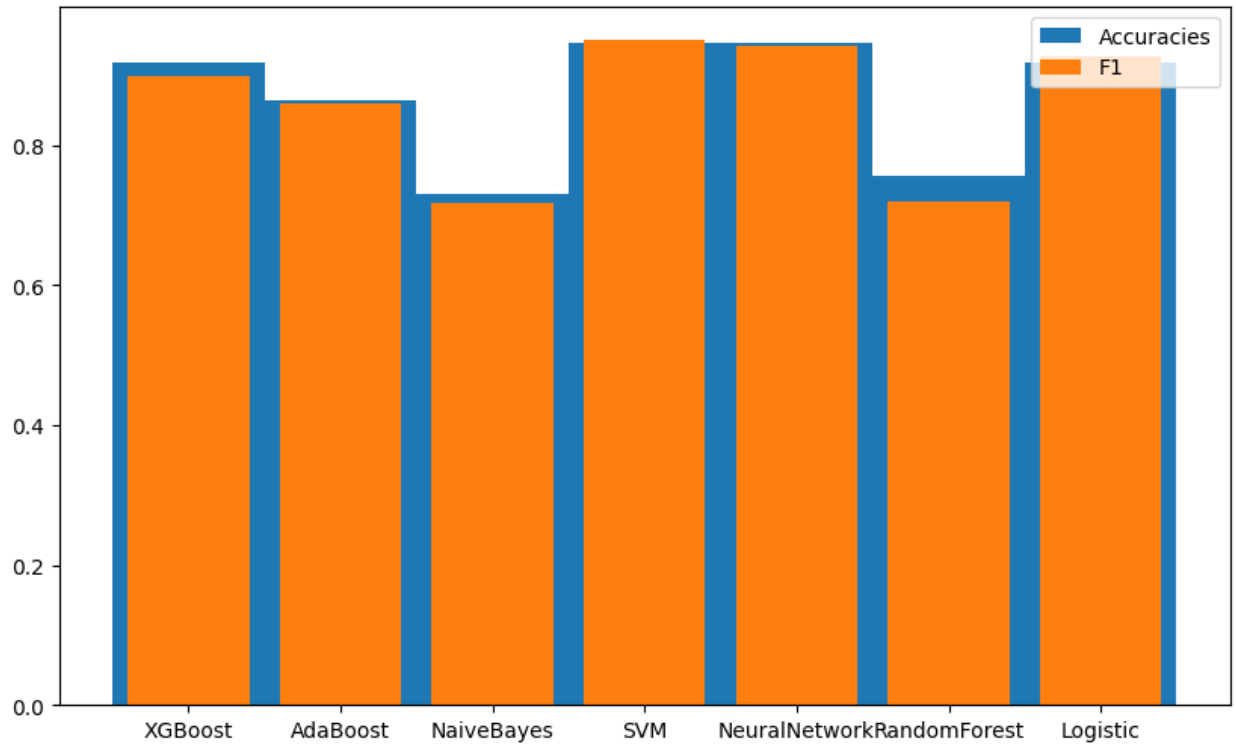
XGBoost Accuracy Training score	
Adaboost Accuracy Training score	

Naive Bayes Accuracy Training score	
SVM Accuracy Training score	

Neural Network Accuracy Training score	
Random Forest Accuracy Training score	

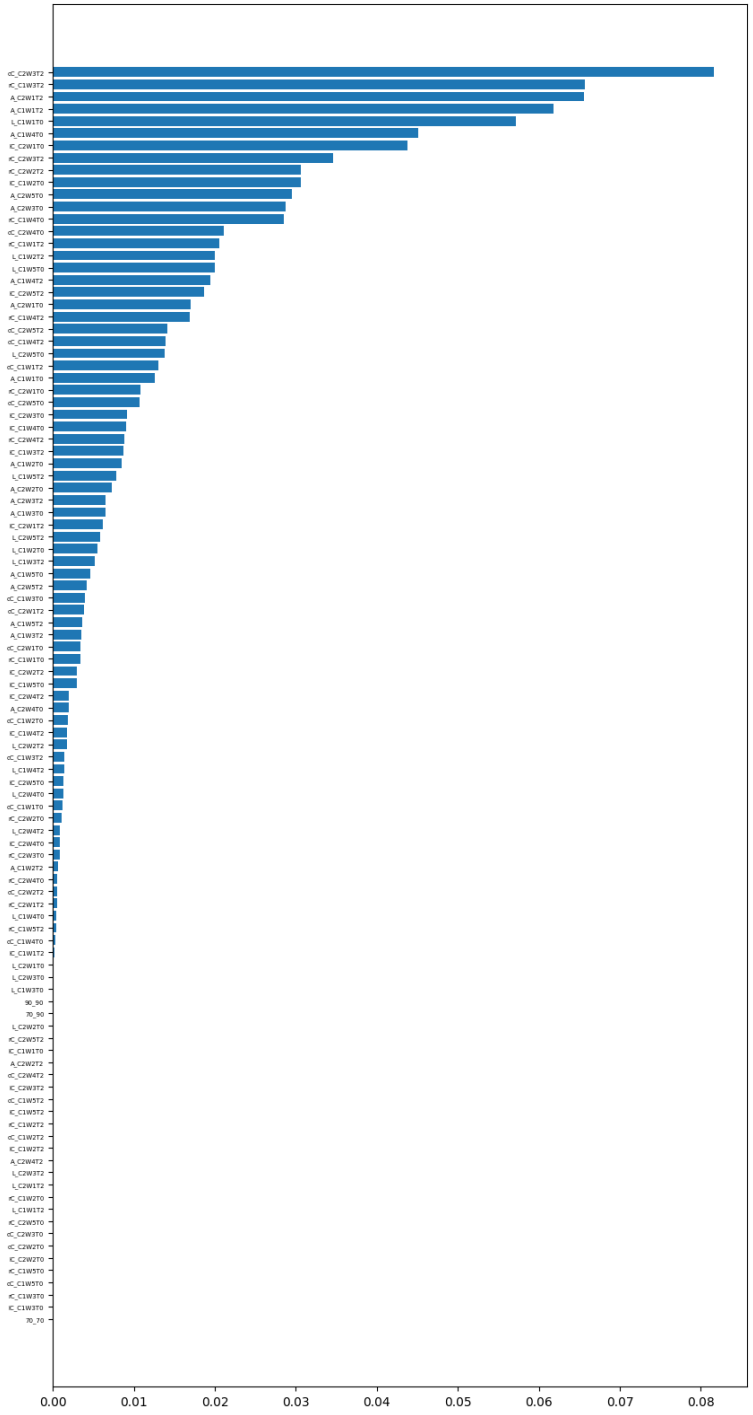
Logistic Accuracy Training score	

For all algorithms, Accuracies and F1 score bars:

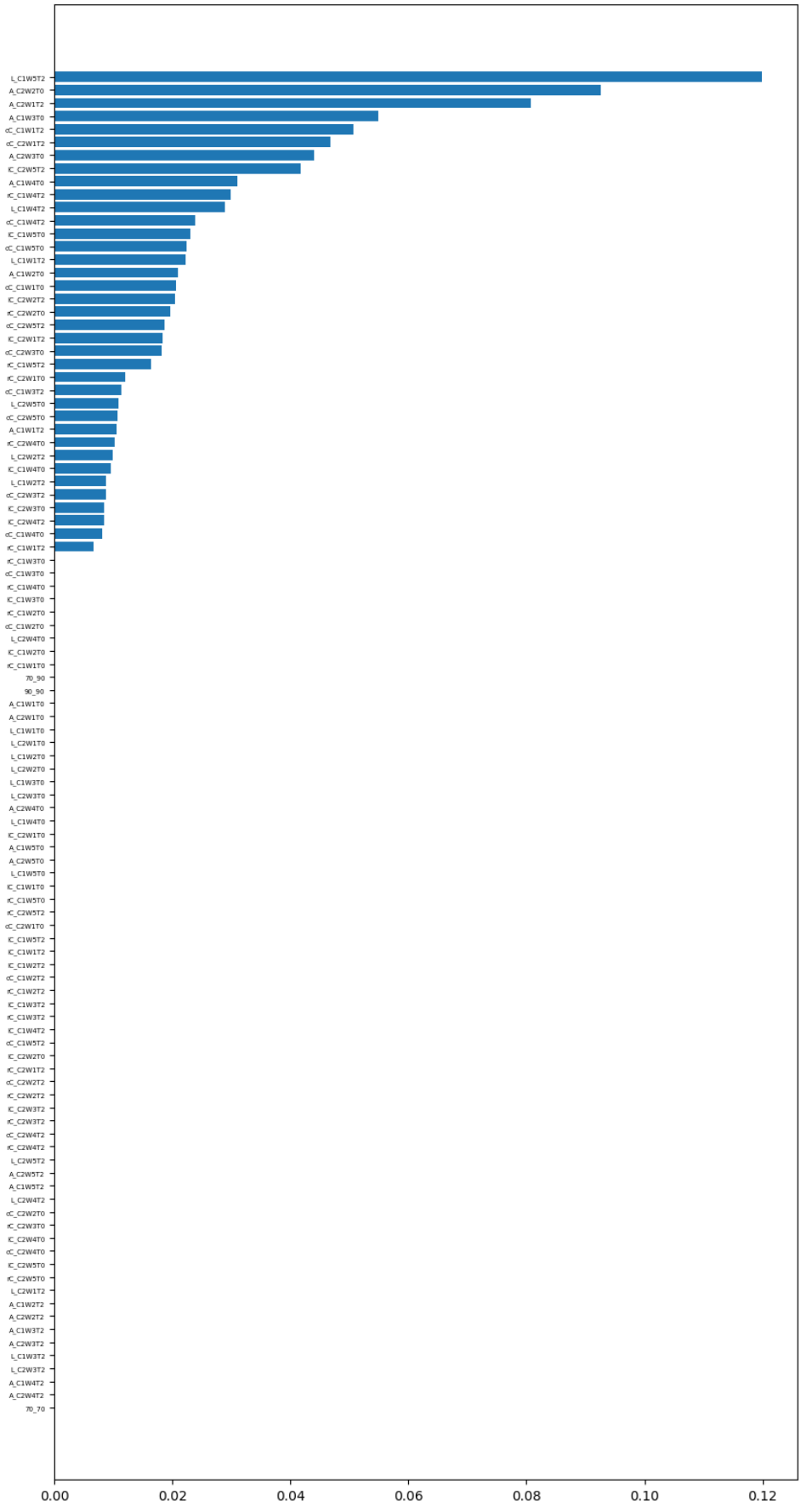


3.4 Variable Importance Analysis

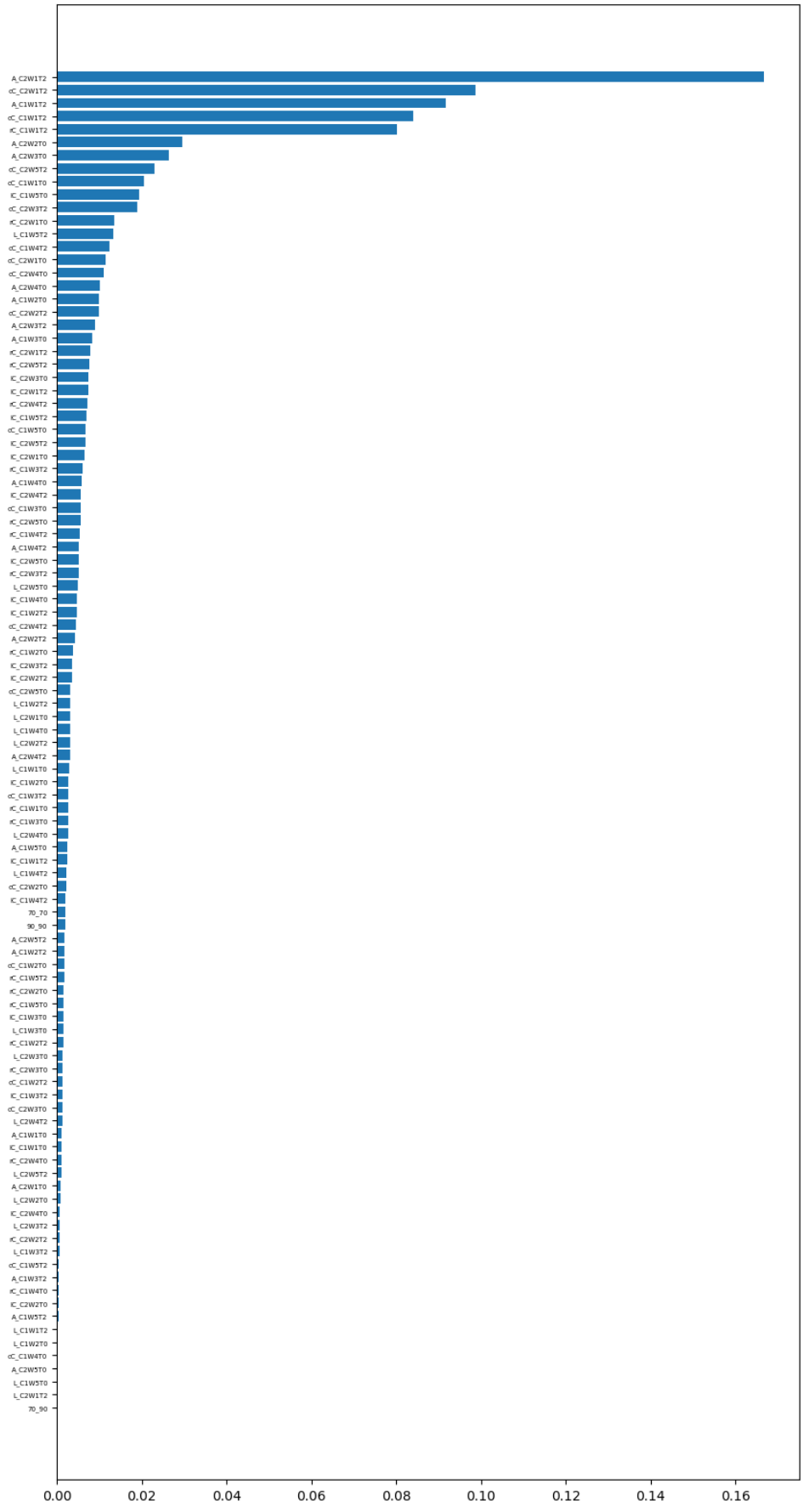
Part one: Variable Importance Analysis with the previous models using all variables

XGBoost	

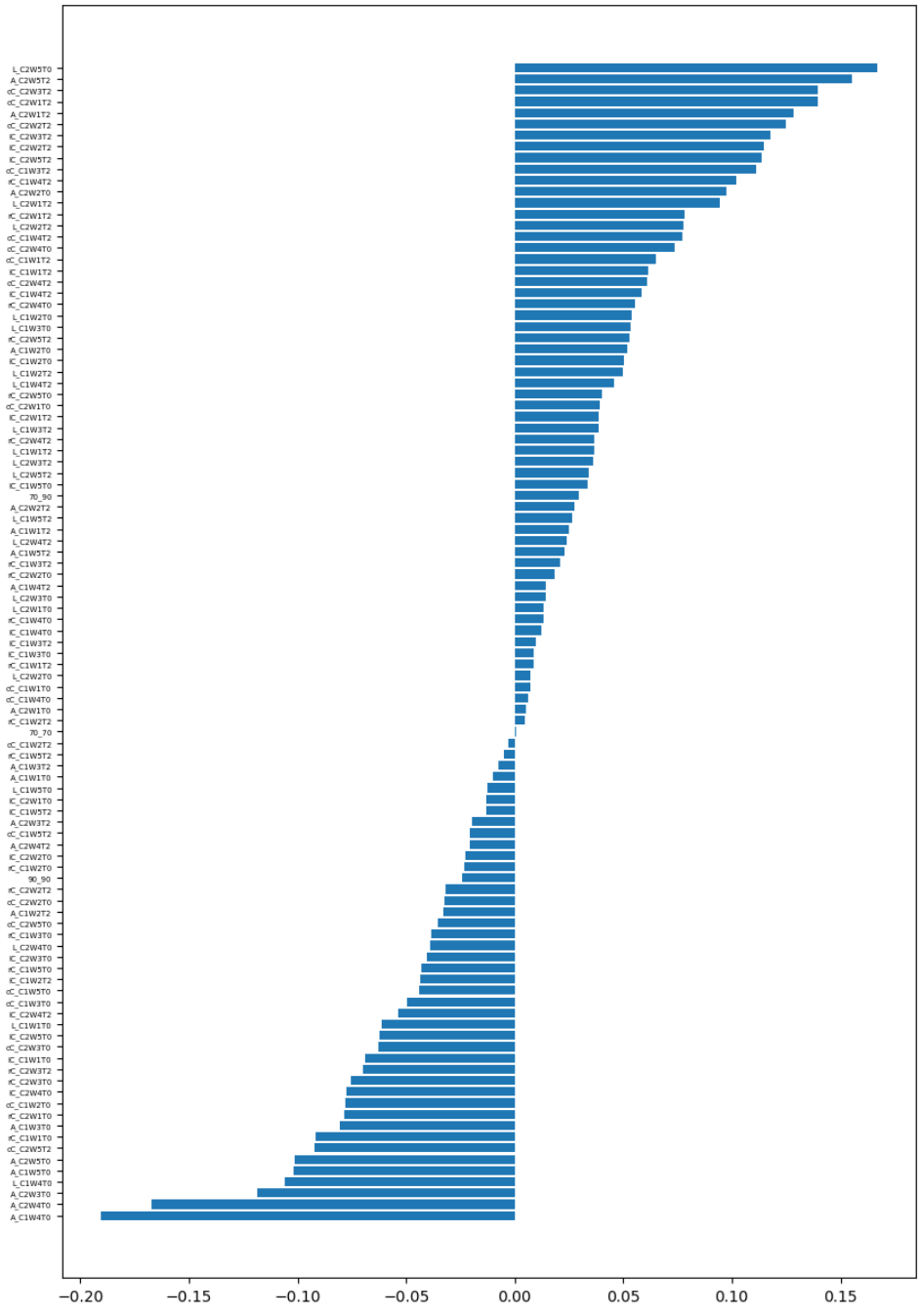
AdaBoost


I_C1W5T2
A_C2W2T0
A_C2W1T2
A_C1W3T0
oC_C1W1T2
oC_C2W1T2
A_C2W3T0
IC_C2W5T2
A_C1W4T0
oC_C1W4T2
I_C1W4T2
oC_C1W4T2
IC_C1W5T0
oC_C1W5T0
I_C1W1T2
A_C1W2T0
oC_C1W1T0
IC_C2W2T2
oC_C2W2T0
oC_C2W5T2
IC_C2W1T2
oC_C2W3T0
oC_C1W5T2
oC_C2W1T0
oC_C1W3T2
I_C2W5T0
oC_C2W5T0
A_C1W1T2
oC_C2W4T0
I_C2W2T2
IC_C1W4T0
I_C1W2T2
oC_C2W3T2
IC_C2W3T0
IC_C2W4T2
oC_C1W4T0
oC_C1W1T2
oC_C1W3T0
oC_C1W3T0
oC_C1W4T0
IC_C1W3T0
oC_C1W2T0
oC_C1W2T0
I_C2W4T0
IC_C1W2T0
oC_C1W1T0
70_90
90_90
A_C1W1T0
A_C2W1T0
I_C1W1T0
I_C2W1T0
I_C1W2T0
I_C2W2T0
I_C1W3T0
I_C2W3T0
A_C2W4T0
I_C1W4T0
IC_C2W1T0
A_C1W5T0
A_C2W5T0
I_C1W5T0
IC_C1W1T0
oC_C1W5T0
oC_C2W5T2
oC_C2W1T0
IC_C1W5T2
IC_C1W1T2
IC_C1W2T2
oC_C1W2T2
oC_C1W2T2
IC_C1W3T2
oC_C1W3T2
IC_C1W4T2
oC_C1W5T2
IC_C2W2T0
oC_C2W1T2
oC_C2W2T2
oC_C2W2T2
IC_C2W3T2
oC_C2W3T2
oC_C2W4T2
oC_C2W4T2
I_C2W5T2
A_C2W5T2
A_C1W5T2
I_C2W4T2
oC_C2W2T0
oC_C2W3T0
IC_C2W4T0
oC_C2W4T0
IC_C2W5T0
oC_C2W5T0
I_C2W1T2
A_C1W2T2
A_C2W2T2
A_C1W3T2
A_C2W3T2
I_C1W3T2
I_C2W3T2
A_C1W4T2
A_C2W4T2
70_70
0.00
0.02
0.04
0.06
0.08
0.10
0.12

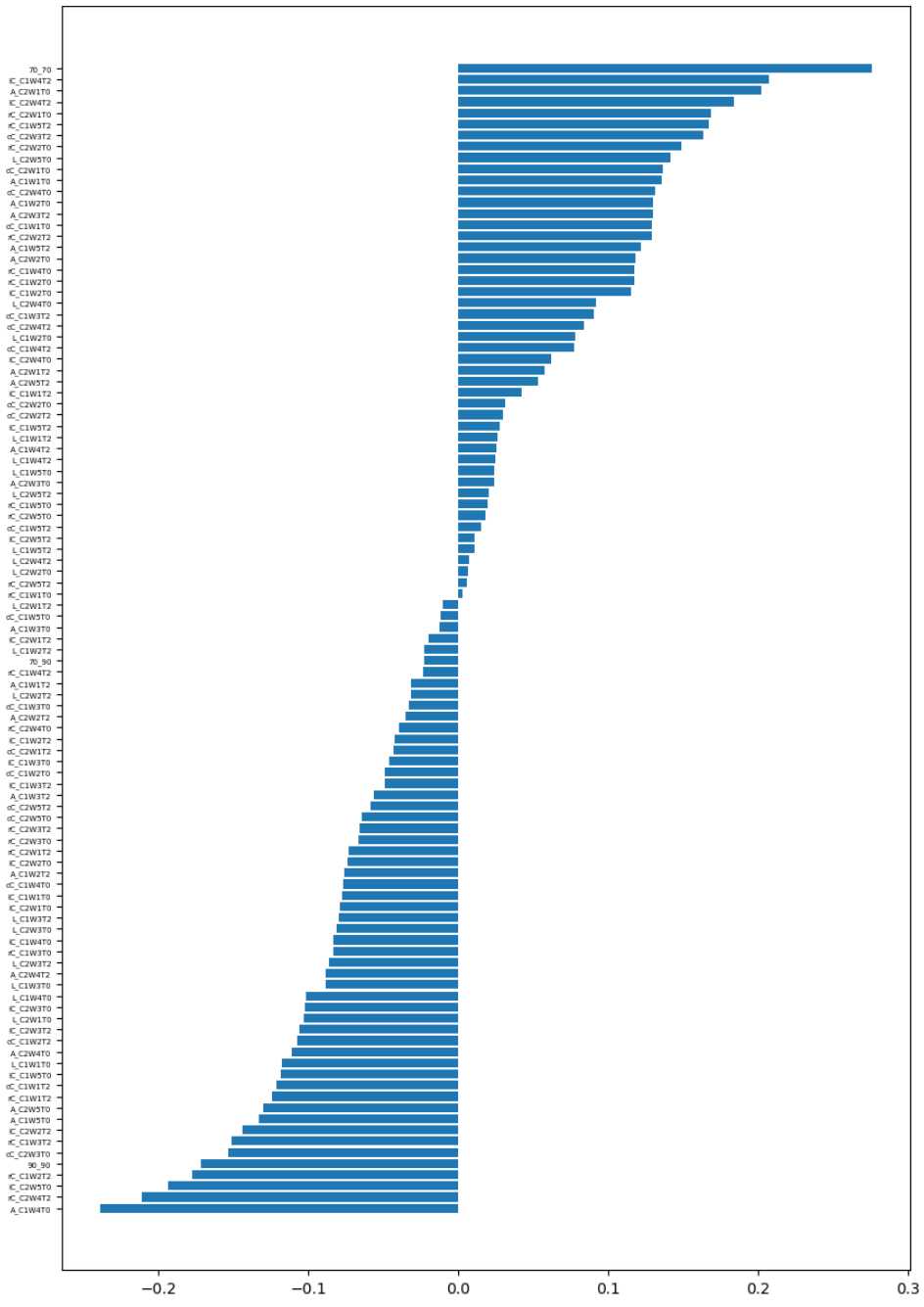
Random Forest



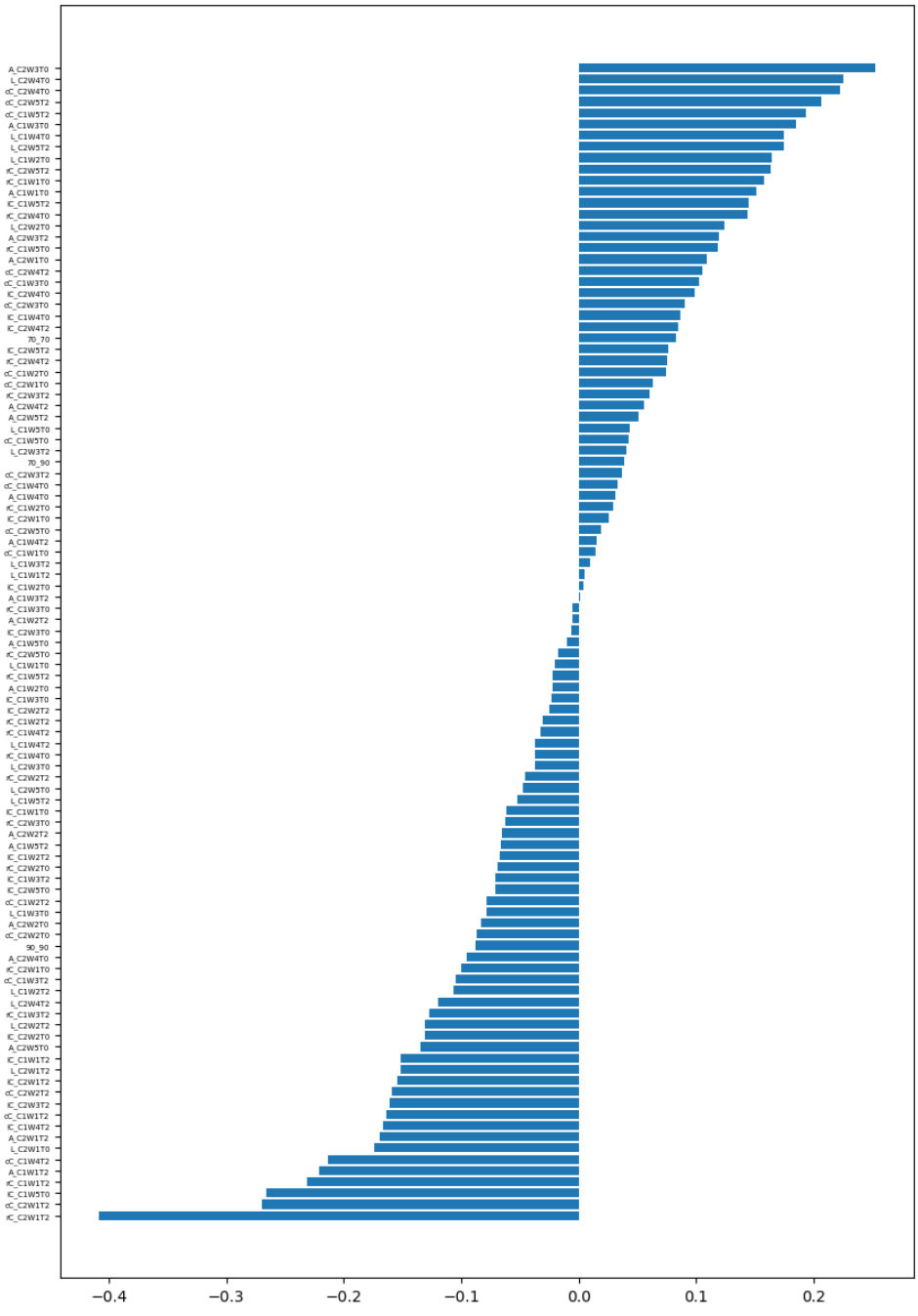
SVM-1

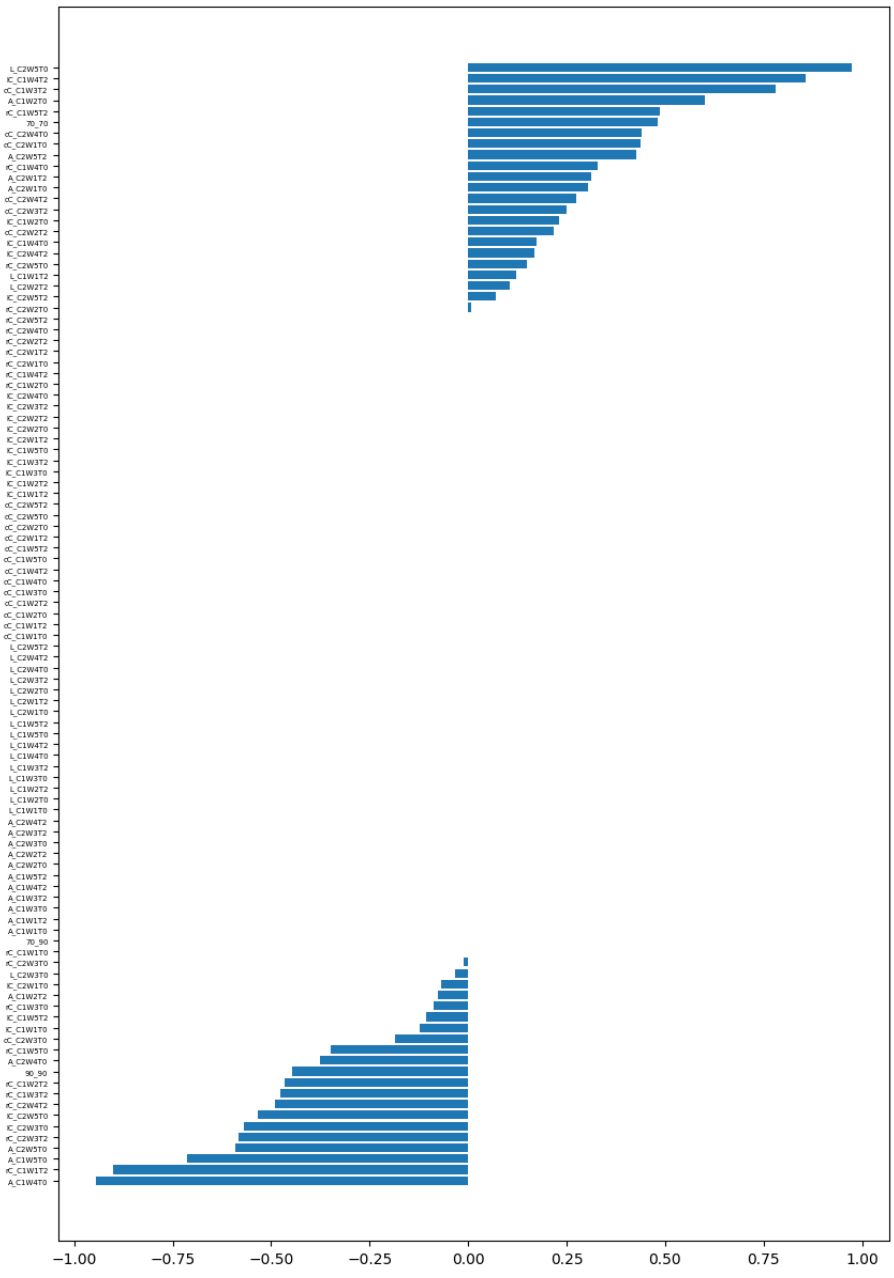

L_C2W5T0
A_C2W5T2
dC_C2W3T2
dC_C2W1T2
A_C2W1T2
dC_C2W2T2
iC_C2W3T2
iC_C2W2T2
iC_C2W5T2
dC_C1W3T2
dC_C1W4T2
A_C2W2T0
L_C2W1T2
dC_C2W1T2
L_C2W2T2
dC_C1W4T2
dC_C2W4T0
dC_C1W1T2
iC_C1W1T2
dC_C2W4T2
iC_C1W4T2
dC_C2W4T0
L_C1W2T0
L_C1W3T0
dC_C2W5T2
A_C1W2T0
iC_C1W2T0
L_C1W2T2
L_C1W4T2
dC_C2W5T0
dC_C2W1T0
iC_C2W1T2
L_C1W3T2
dC_C2W4T2
L_C1W1T2
L_C2W3T2
L_C2W5T2
iC_C1W5T0
70_90
A_C2W2T2
L_C1W5T2
A_C1W1T2
L_C2W4T2
A_C1W5T2
dC_C1W3T2
dC_C2W2T0
A_C1W4T2
L_C2W3T0
L_C2W1T0
dC_C1W4T0
iC_C1W4T0
iC_C1W3T2
iC_C1W3T0
dC_C1W1T2
L_C2W2T0
dC_C1W1T0
dC_C1W4T0
A_C2W1T0
dC_C1W2T2
70_70
dC_C1W2T2
dC_C1W5T2
A_C1W3T2
A_C1W1T0
L_C1W5T0
iC_C2W1T0
iC_C1W5T2
A_C2W3T2
dC_C1W5T2
A_C2W4T2
iC_C2W2T0
dC_C1W2T0
90_90
dC_C2W2T2
dC_C2W2T0
A_C1W2T2
dC_C2W5T0
dC_C1W3T0
L_C2W4T0
iC_C2W3T0
dC_C1W5T0
iC_C1W2T2
dC_C1W5T0
dC_C1W3T0
iC_C2W4T2
L_C1W1T0
iC_C2W5T0
dC_C2W3T0
iC_C1W1T0
dC_C2W3T2
dC_C2W3T0
iC_C2W4T0
dC_C1W2T0
dC_C2W1T0
A_C1W3T0
dC_C1W1T0
dC_C2W5T2
A_C2W5T0
A_C1W5T0
L_C1W4T0
A_C2W3T0
A_C2W4T0
A_C1W4T0
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15

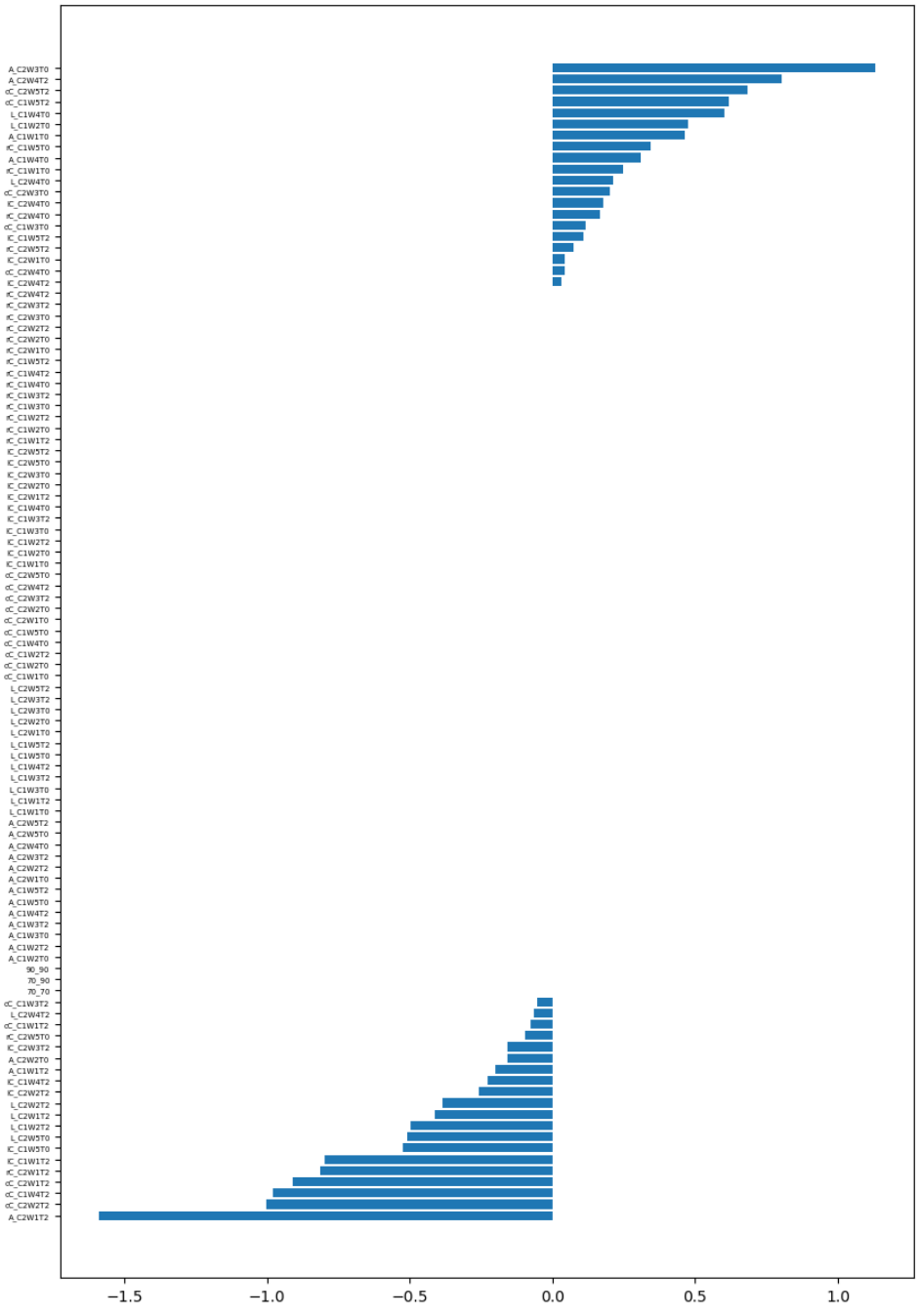
SVM-2

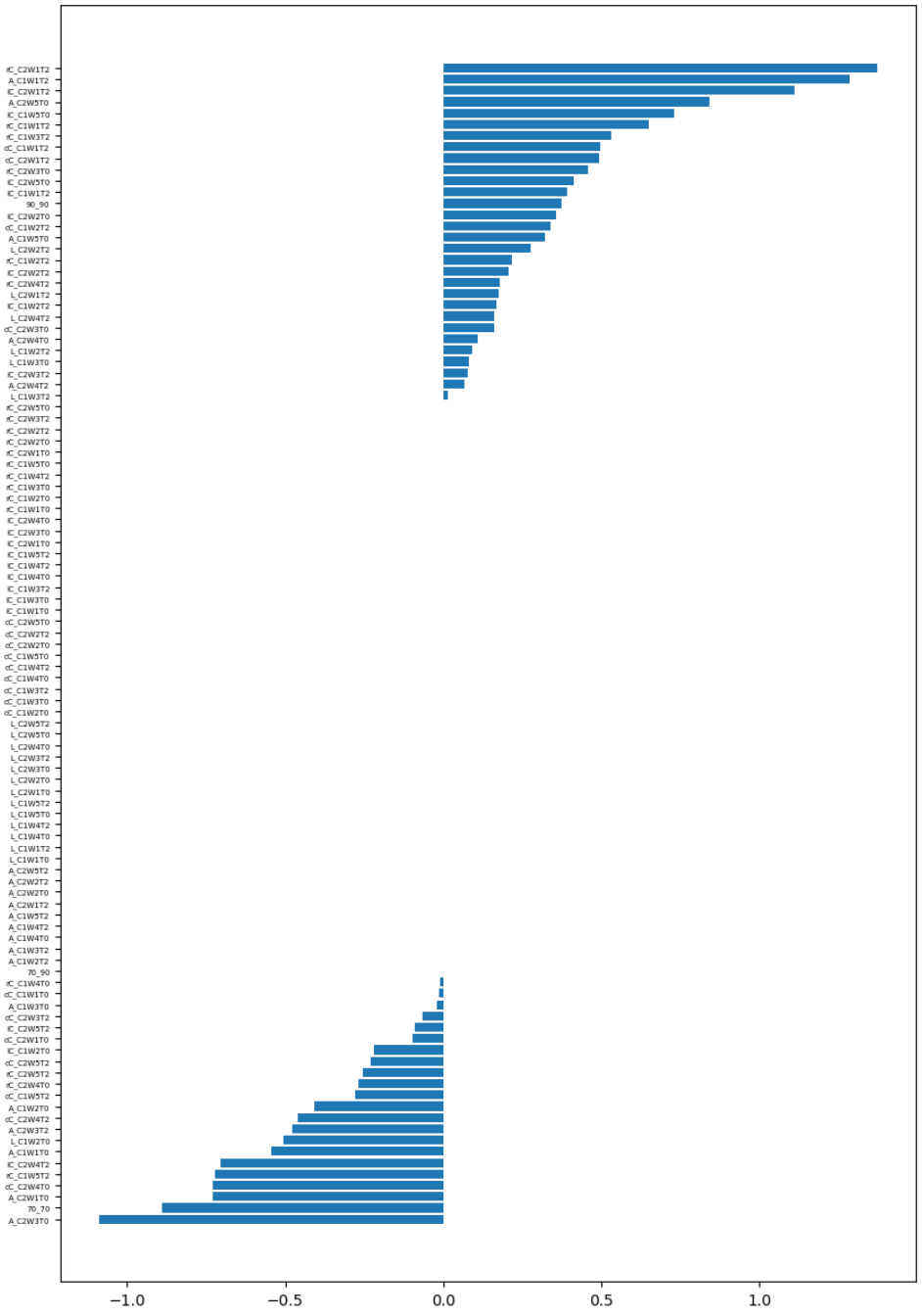

70_70
iC_C1W4T2
A_C2W1T0
iC_C2W4T2
iC_C2W1T0
iC_C1W5T2
oC_C2W3T2
iC_C2W2T0
L_C2W5T0
oC_C2W1T0
A_C1W1T0
oC_C2W4T0
A_C1W2T0
A_C2W3T2
oC_C1W1T0
iC_C2W2T2
A_C1W5T2
A_C2W2T0
iC_C1W4T0
iC_C1W2T0
iC_C1W2T0
L_C2W4T0
oC_C1W3T2
oC_C2W4T2
L_C1W2T0
oC_C1W4T2
iC_C2W4T0
A_C2W1T2
A_C2W5T2
iC_C1W1T2
oC_C2W2T0
oC_C2W2T2
iC_C1W5T2
L_C1W1T2
A_C1W4T2
L_C1W4T2
L_C1W5T0
A_C2W3T0
L_C2W5T2
iC_C1W5T0
iC_C2W5T0
oC_C1W5T2
iC_C2W5T2
L_C1W5T2
L_C2W4T2
L_C2W2T0
iC_C2W5T2
iC_C1W1T0
L_C2W1T2
oC_C1W5T0
A_C1W3T0
iC_C2W1T2
L_C1W2T2
70_90
iC_C1W4T2
A_C1W1T2
L_C2W2T2
oC_C1W3T0
A_C2W2T2
iC_C2W4T0
iC_C1W2T2
oC_C2W1T2
iC_C1W3T0
oC_C1W2T0
iC_C1W3T2
A_C1W3T2
oC_C2W5T2
oC_C2W5T0
iC_C2W3T2
iC_C2W3T0
iC_C2W1T2
iC_C2W2T0
A_C1W2T2
oC_C1W4T0
iC_C1W1T0
iC_C2W1T0
L_C1W3T2
L_C2W3T0
iC_C1W4T0
iC_C1W3T0
L_C2W3T2
A_C2W4T2
L_C1W3T0
L_C1W4T0
iC_C2W3T0
L_C2W1T0
iC_C2W3T2
oC_C1W2T2
A_C2W4T0
L_C1W1T0
iC_C1W5T0
oC_C1W1T2
iC_C1W1T2
A_C2W5T0
A_C1W5T0
iC_C2W2T2
iC_C1W3T2
oC_C2W3T0
90_90
iC_C1W2T2
iC_C2W5T0
iC_C2W4T2
A_C1W4T0
−0.2
−0.1
0.0
0.1
0.2
0.3

SVM-3


A_C2W3T0
L_C2W4T0
cC_C2W4T0
cC_C2W5T2
cC_C1W5T2
A_C1W3T0
L_C1W4T0
L_C2W5T2
L_C1W2T0
iC_C2W5T2
iC_C1W1T0
A_C1W1T0
iC_C1W5T2
iC_C2W4T0
L_C2W2T0
A_C2W3T2
iC_C1W5T0
A_C2W1T0
cC_C2W4T2
cC_C1W3T0
iC_C2W4T0
cC_C2W3T0
iC_C1W4T0
iC_C2W4T2
70_70
iC_C2W5T2
iC_C2W4T2
cC_C1W2T0
cC_C2W1T0
iC_C2W3T2
A_C2W4T2
A_C2W5T2
L_C1W5T0
cC_C1W5T0
L_C2W3T2
70_90
cC_C2W3T2
cC_C1W4T0
A_C1W4T0
iC_C1W2T0
iC_C2W1T0
cC_C2W5T0
A_C1W4T2
cC_C1W1T0
L_C1W3T2
L_C1W1T2
iC_C1W2T0
A_C1W3T2
iC_C1W3T0
A_C1W2T2
iC_C2W3T0
A_C1W5T0
iC_C2W5T0
L_C1W1T0
iC_C1W5T2
A_C1W2T0
iC_C1W3T0
iC_C2W2T2
iC_C1W2T2
iC_C1W4T2
L_C1W4T2
iC_C1W4T0
L_C2W3T0
iC_C2W2T2
L_C2W5T0
L_C1W5T2
iC_C1W1T0
iC_C2W3T0
A_C2W2T2
A_C1W5T2
iC_C1W2T2
iC_C2W2T0
iC_C1W3T2
iC_C2W5T0
cC_C1W2T2
L_C1W3T0
A_C2W2T0
cC_C2W2T0
90_90
A_C2W4T0
iC_C2W1T0
cC_C1W3T2
L_C1W2T2
L_C2W4T2
iC_C1W3T2
L_C2W2T2
iC_C2W2T0
A_C2W5T0
iC_C1W1T2
L_C2W1T2
iC_C2W1T2
cC_C2W2T2
iC_C2W3T2
cC_C1W1T2
iC_C1W4T2
A_C2W1T2
L_C2W1T0
cC_C1W4T2
A_C1W1T2
iC_C1W1T2
iC_C1W5T0
cC_C2W1T2
iC_C2W1T2
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2

Logistic-1	

Logistic-2	 A_C2W3T0 A_C2W4T2 cC_C2W5T2 cC_C1W5T2 L_C1W4T0 L_C1W2T0 A_C1W1T0 rC_C1W5T0 A_C1W4T0 rC_C1W1T0 L_C2W4T0 cC_C2W3T0 lC_C2W4T0 rC_C2W4T0 cC_C1W3T0 lC_C1W5T2 rC_C2W5T2 lC_C2W1T0 cC_C2W4T0 lC_C2W4T2 rC_C2W4T2 rC_C2W3T2 rC_C2W3T0 rC_C2W2T2 rC_C2W2T0 rC_C2W1T0 rC_C1W5T2 rC_C1W4T2 rC_C1W4T0 rC_C1W3T2 rC_C1W3T0 rC_C1W2T2 rC_C1W2T0 rC_C1W1T2 lC_C2W5T2 lC_C2W5T0 lC_C2W3T0 lC_C2W2T0 lC_C2W1T2 lC_C1W4T0 lC_C1W3T2 lC_C1W3T0 lC_C1W2T2 lC_C1W2T0 lC_C1W1T0 cC_C2W5T0 cC_C2W4T2 cC_C2W3T2 cC_C2W2T0 cC_C2W1T0 cC_C1W5T0 cC_C1W4T0 cC_C1W2T2 cC_C1W2T0 cC_C1W1T0 L_C2W5T2 L_C2W3T2 L_C2W3T0 L_C2W2T0 L_C2W1T0 L_C1W5T2 L_C1W5T0 L_C1W4T2 L_C1W3T2 L_C1W3T0 L_C1W1T2 L_C1W1T0 A_C2W5T2 A_C2W5T0 A_C2W4T0 A_C2W3T2 A_C2W2T2 A_C2W1T0 A_C1W5T2 A_C1W5T0 A_C1W4T2 A_C1W3T2 A_C1W3T0 A_C1W2T2 A_C1W2T0 90_90 70_90 70_70 cC_C1W3T2 L_C2W4T2 cC_C1W1T2 rC_C2W5T0 lC_C2W3T2 A_C2W2T0 A_C1W1T2 lC_C1W4T2 lC_C2W2T2 L_C2W2T2 L_C2W1T2 L_C1W2T2 L_C2W5T0 lC_C1W5T0 lC_C1W1T2 rC_C2W1T2 cC_C2W1T2 cC_C1W4T2 cC_C2W2T2 A_C2W1T2 −1.5 −1.0 −0.5 0.0 0.5 1.0

Logistic-3	

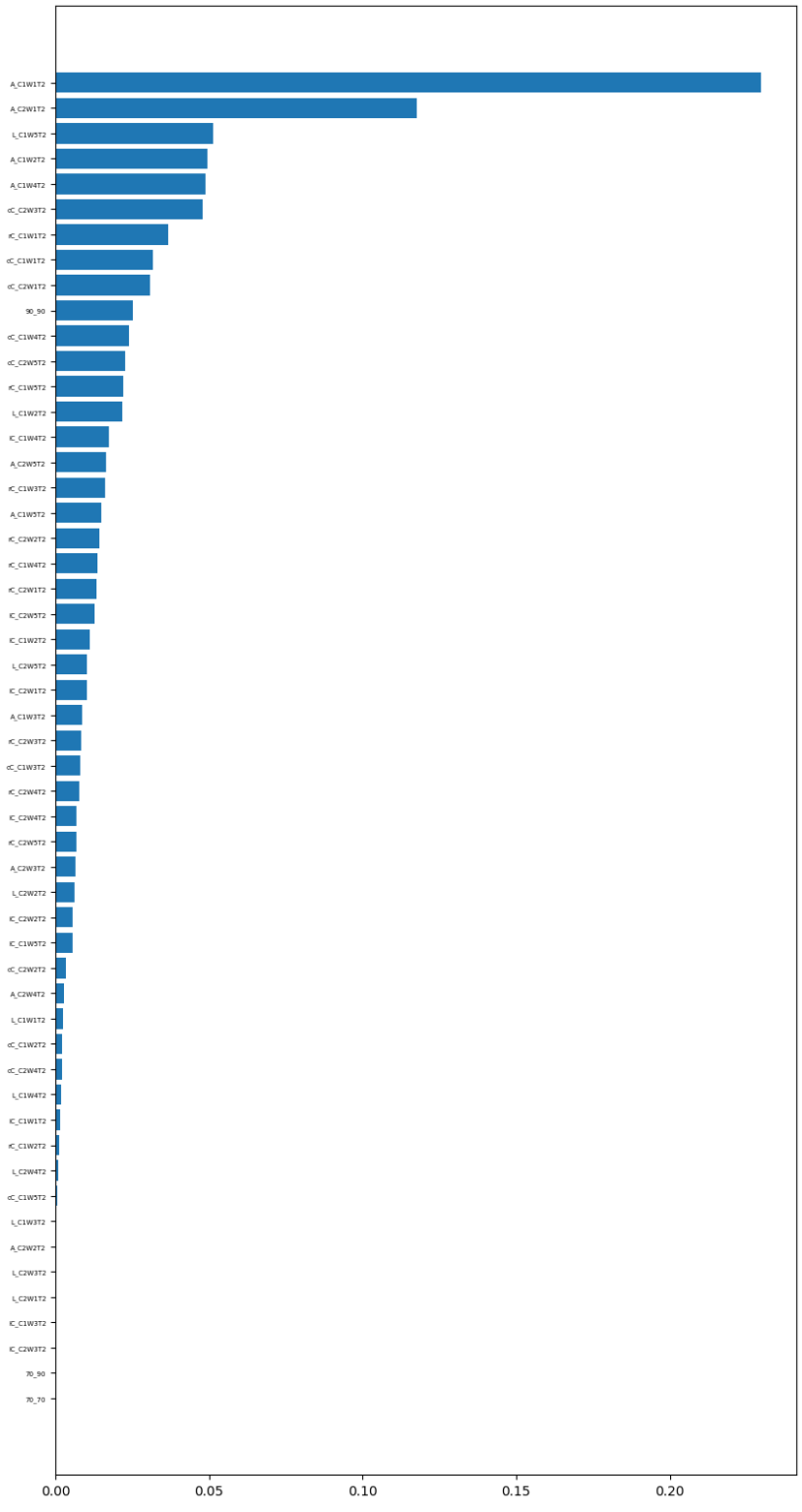
These are the variance importance plots I get from the algorithms, we can see A_C2W1T0 is a really important feature, and lC_C1W4T2, rC_C1W4T2, cC_C1W4T2 are also important which are consistent of the model.

However, ranking variables by these variance importance are not that accurate, correlation between variables, ignorance of variables will impact the answer. Therefore, I compare the results with the same algorithms, using almost the same tuning parameters methods to compare the accuracy and F1 score for a dataset that only uses T2 and T0 variables separately.

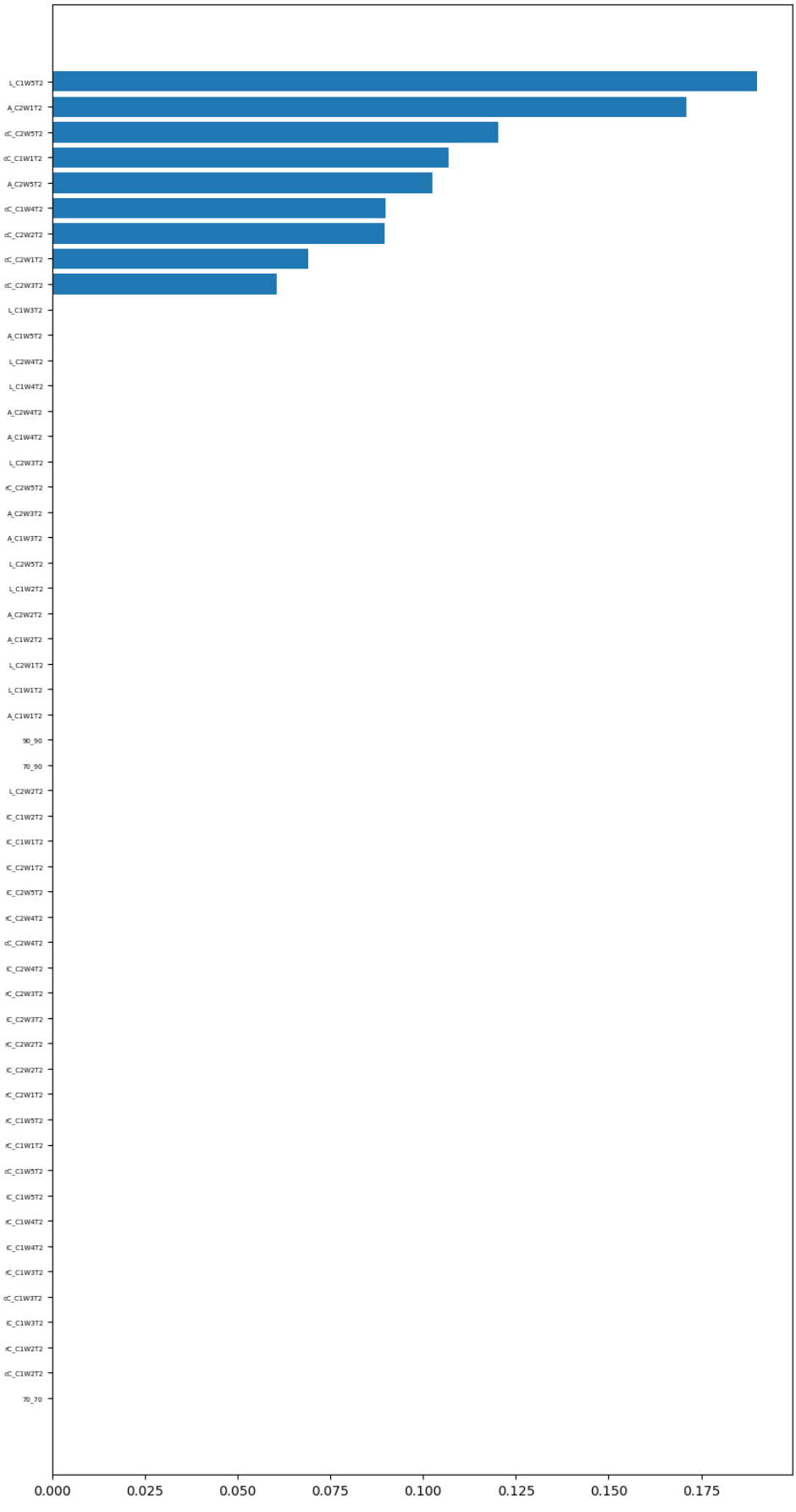
Part two: models only dealing with T2 variables and stimulus level

As we should care more about T2, so I post the variance importance graph below:

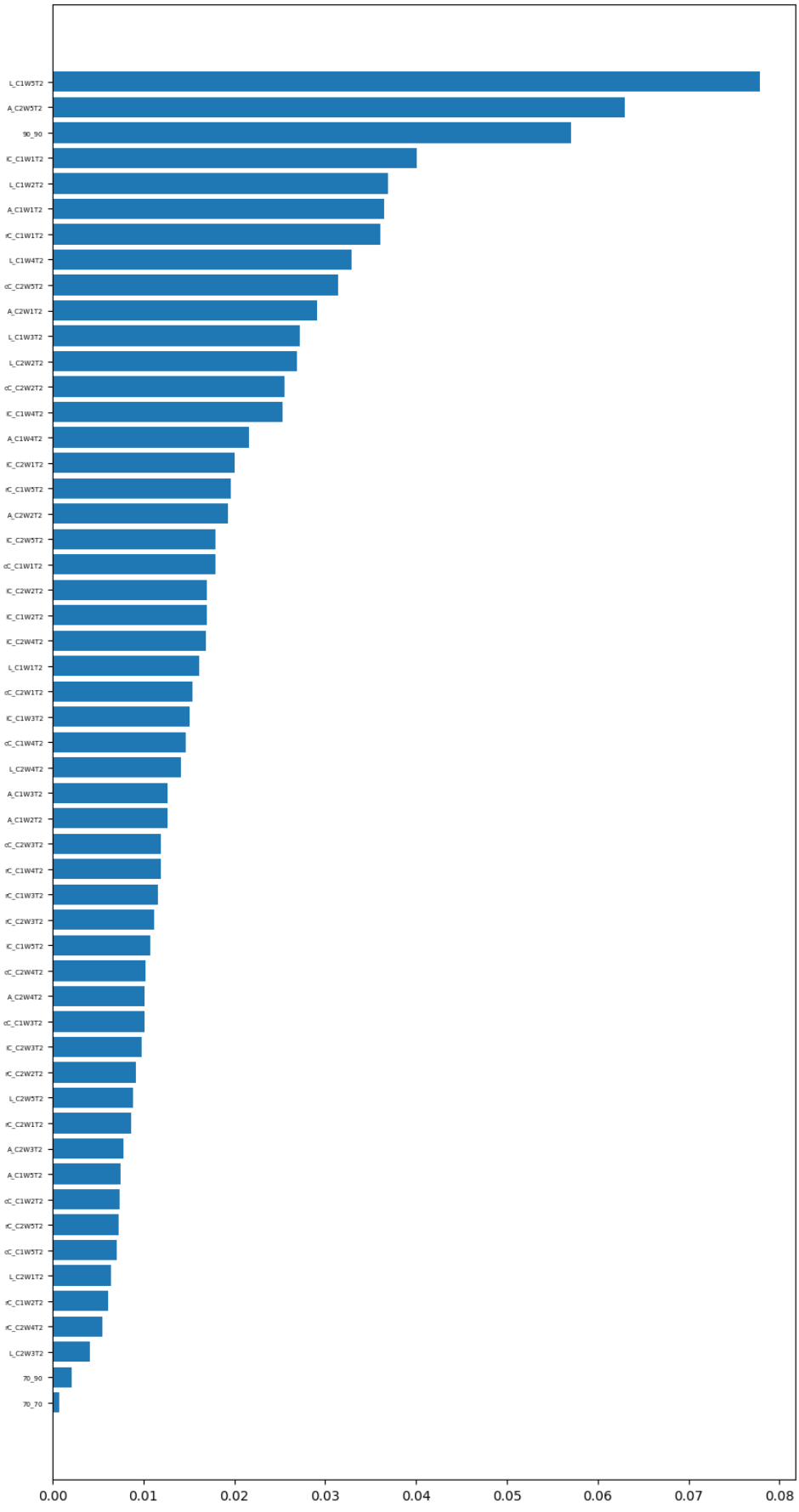
XGBoost

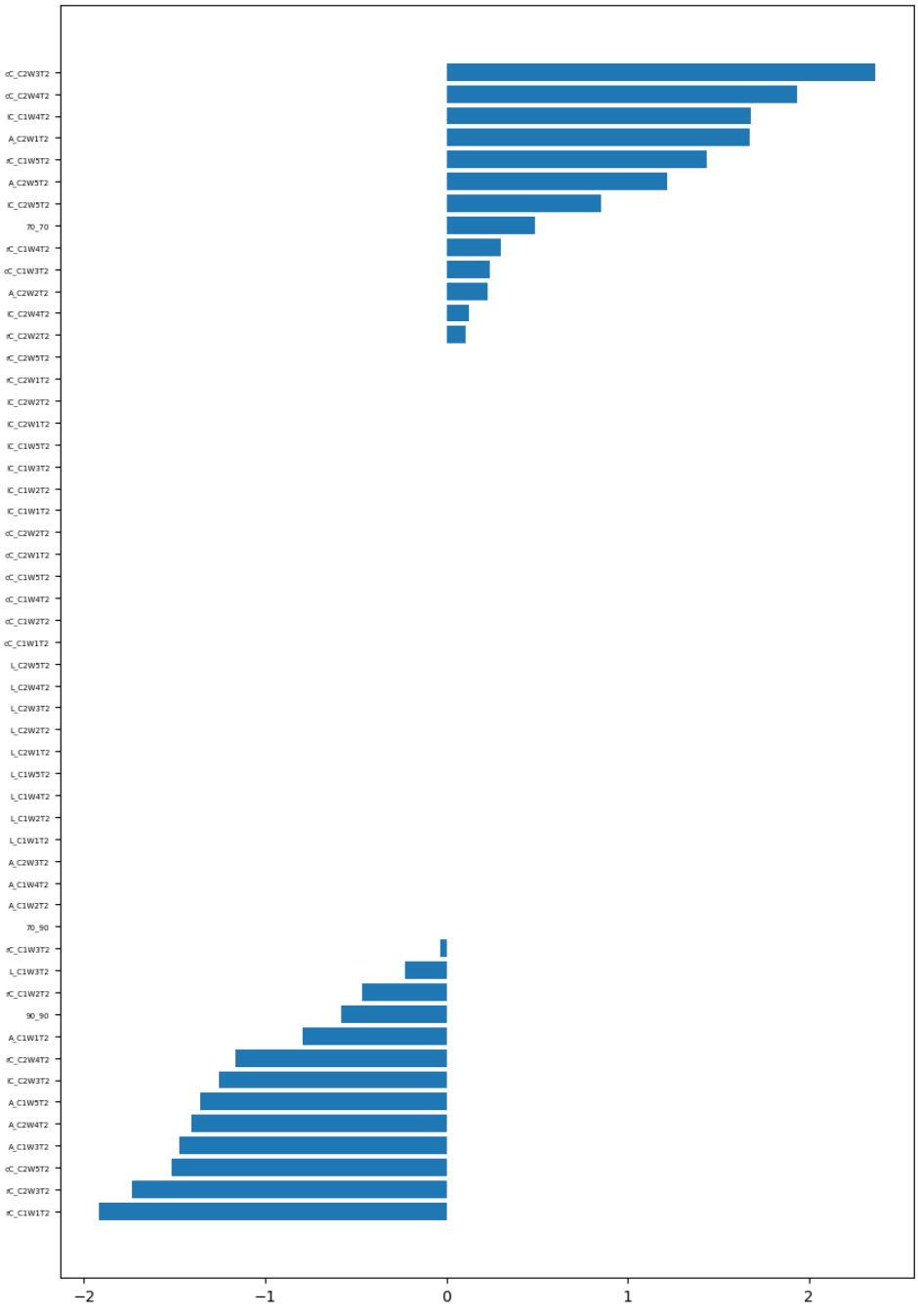

A_C1W1T2
A_C2W1T2
L_C1W5T2
A_C1W2T2
A_C1W4T2
oC_C2W3T2
nC_C1W1T2
oC_C1W1T2
oC_C2W1T2
90_90
oC_C1W4T2
oC_C2W5T2
nC_C1W5T2
L_C1W2T2
IC_C1W4T2
A_C2W5T2
nC_C1W3T2
A_C1W5T2
nC_C2W2T2
nC_C1W4T2
nC_C2W1T2
IC_C2W5T2
IC_C1W2T2
L_C2W5T2
IC_C2W1T2
A_C1W3T2
nC_C2W3T2
oC_C1W3T2
nC_C2W4T2
IC_C2W4T2
nC_C2W5T2
A_C2W3T2
L_C2W2T2
IC_C2W2T2
IC_C1W5T2
oC_C2W2T2
A_C2W4T2
L_C1W1T2
oC_C1W2T2
oC_C2W4T2
L_C1W4T2
IC_C1W1T2
nC_C1W2T2
L_C2W4T2
oC_C1W5T2
L_C1W3T2
A_C2W2T2
L_C2W3T2
L_C2W1T2
IC_C1W3T2
IC_C2W3T2
70_90
70_70
0.00
0.05
0.10
0.15
0.20

AdaBoost

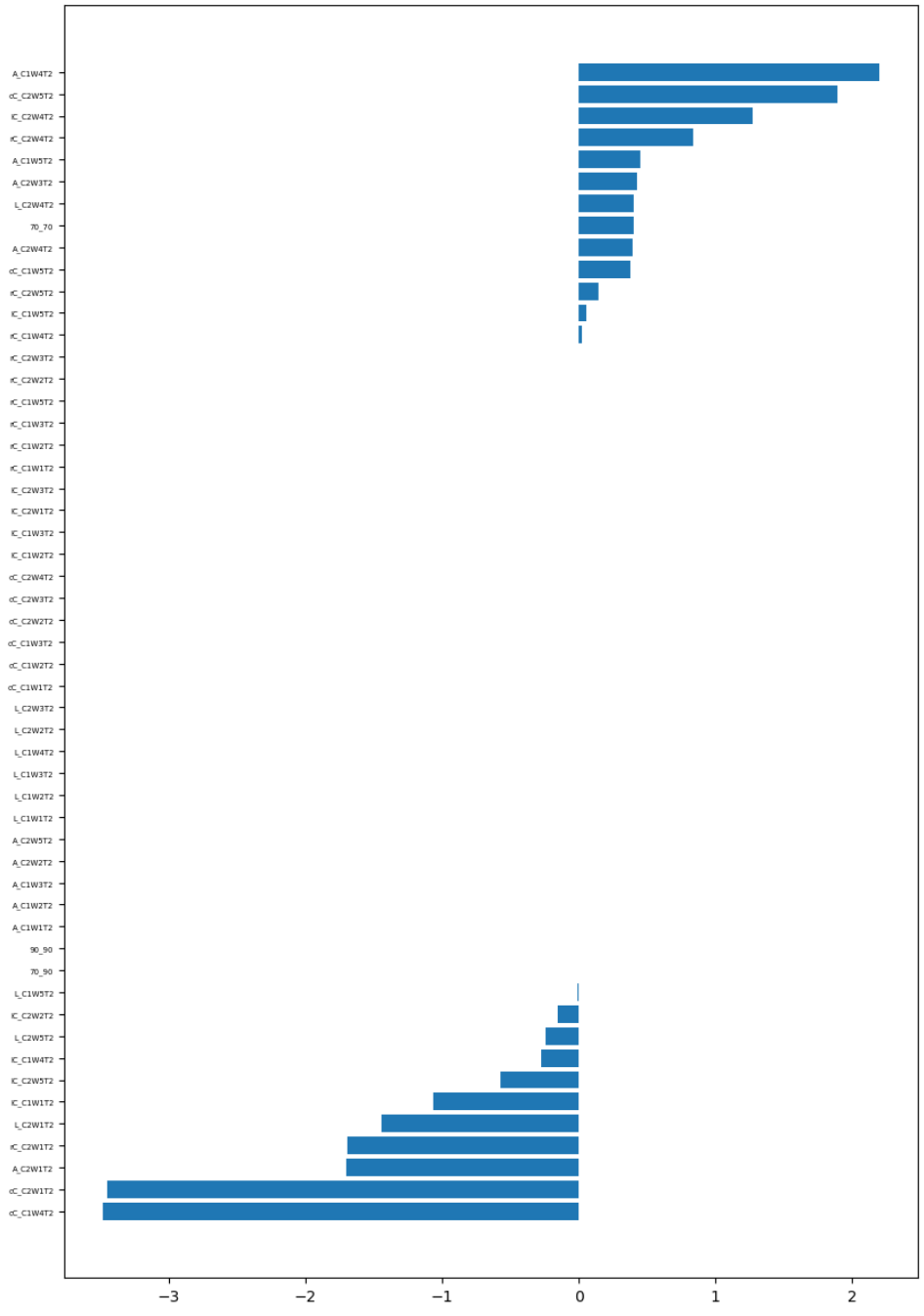

L_C1W5T2
A_C2W1T2
oC_C2W5T2
oC_C1W1T2
A_C2W5T2
oC_C1W4T2
oC_C2W2T2
oC_C2W1T2
oC_C2W3T2
L_C1W3T2
A_C1W5T2
L_C2W4T2
L_C1W4T2
A_C2W4T2
A_C1W4T2
L_C2W3T2
xC_C2W5T2
A_C2W3T2
A_C1W3T2
L_C2W5T2
L_C1W2T2
A_C2W2T2
A_C1W2T2
L_C2W1T2
L_C1W1T2
A_C1W1T2
90_90
70_90
L_C2W2T2
IC_C1W2T2
IC_C1W1T2
IC_C2W1T2
IC_C2W5T2
xC_C2W4T2
oC_C2W4T2
IC_C2W4T2
xC_C2W3T2
IC_C2W3T2
xC_C2W2T2
IC_C2W2T2
xC_C2W1T2
xC_C1W5T2
xC_C1W1T2
oC_C1W5T2
IC_C1W5T2
xC_C1W4T2
IC_C1W4T2
xC_C1W3T2
oC_C1W3T2
IC_C1W3T2
xC_C1W2T2
oC_C1W2T2
70_70
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

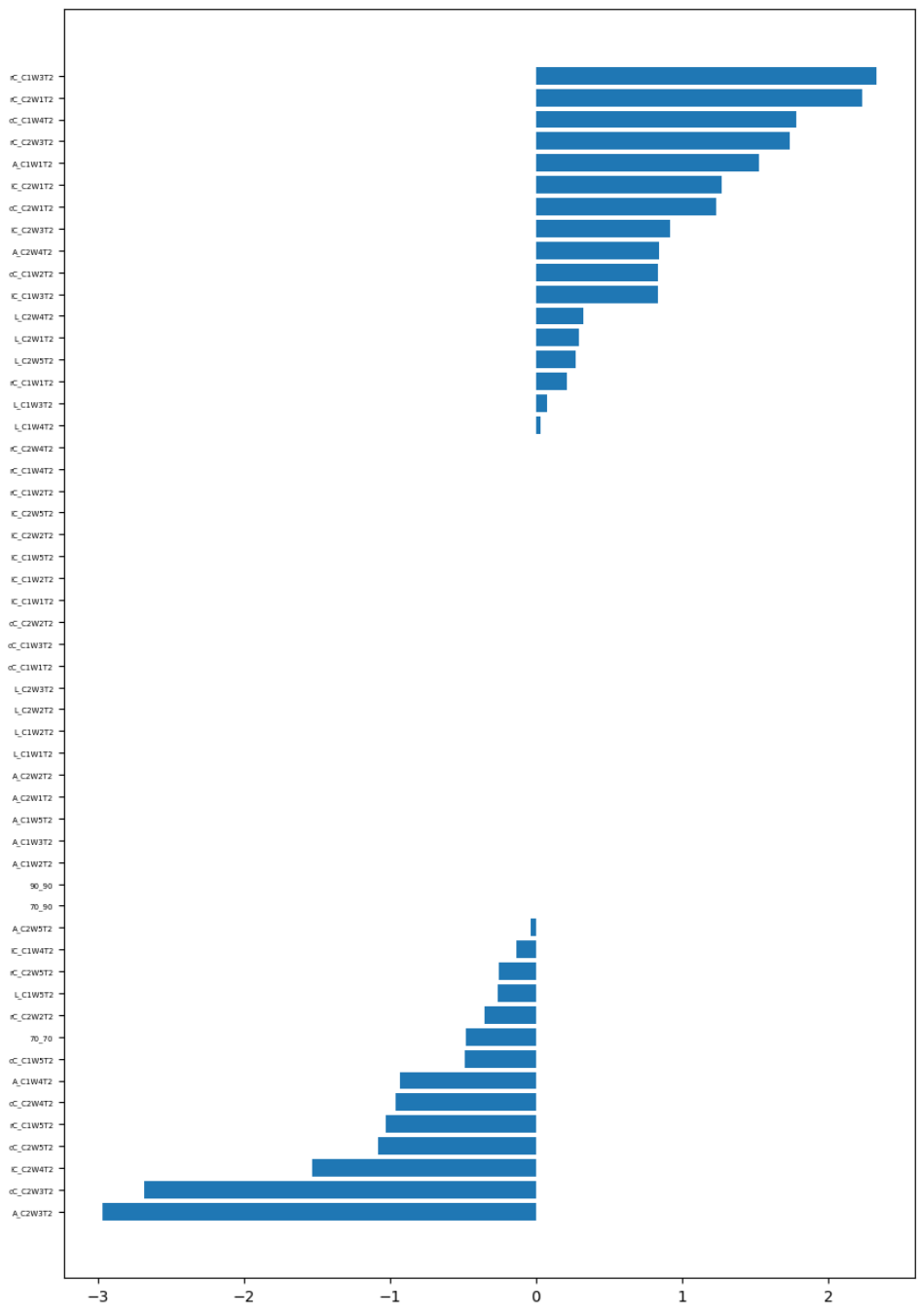
Random Forest



Logistic-1	 cC_C2W3T2 cC_C2W4T2 tC_C1W4T2 A_C2W1T2 tC_C1W5T2 A_C2W5T2 tC_C2W5T2 70_70 tC_C1W4T2 cC_C1W3T2 A_C2W2T2 tC_C2W4T2 tC_C2W2T2 tC_C2W5T2 tC_C2W1T2 tC_C2W2T2 tC_C2W1T2 tC_C1W5T2 tC_C1W3T2 tC_C1W2T2 tC_C1W1T2 cC_C2W2T2 cC_C2W1T2 cC_C1W5T2 cC_C1W4T2 cC_C1W2T2 cC_C1W1T2 L_C2W5T2 L_C2W4T2 L_C2W3T2 L_C2W2T2 L_C2W1T2 L_C1W5T2 L_C1W4T2 L_C1W2T2 L_C1W1T2 A_C2W3T2 A_C1W4T2 A_C1W2T2 70_90 tC_C1W3T2 L_C1W3T2 tC_C1W2T2 90_90 A_C1W1T2 tC_C2W4T2 tC_C2W3T2 A_C1W5T2 A_C2W4T2 A_C1W3T2 cC_C2W5T2 tC_C2W3T2 tC_C1W1T2 −2 −1 0 1 2

Logistic-2


A_C1W4T2
cC_C2W5T2
IC_C2W4T2
iC_C2W4T2
A_C1W5T2
A_C2W3T2
L_C2W4T2
70_70
A_C2W4T2
cC_C1W5T2
iC_C2W5T2
IC_C1W5T2
iC_C1W4T2
iC_C2W3T2
iC_C2W2T2
iC_C1W5T2
iC_C1W3T2
iC_C1W2T2
iC_C1W1T2
IC_C2W3T2
IC_C2W1T2
IC_C1W3T2
IC_C1W2T2
cC_C2W4T2
cC_C2W3T2
cC_C2W2T2
cC_C1W3T2
cC_C1W2T2
cC_C1W1T2
L_C2W3T2
L_C2W2T2
L_C1W4T2
L_C1W3T2
L_C1W2T2
L_C1W1T2
A_C2W5T2
A_C2W2T2
A_C1W3T2
A_C1W2T2
A_C1W1T2
90_90
70_90
L_C1W5T2
IC_C2W2T2
L_C2W5T2
IC_C1W4T2
IC_C2W5T2
IC_C1W1T2
L_C2W1T2
iC_C2W1T2
A_C2W1T2
cC_C2W1T2
cC_C1W4T2
−3
−2
−1
0
1
2

Logistic-3	

As for SVM, I find the rbf kernel is better than the linear in the tuning process so I cannot simply depict the plot. However, I will compare the results of these algorithms:

Methods/ Metrics	Accuracy for all Vars	F1-score for all Vars	Accuracy for T2 only	F1-score for T2 only	Accuracy for T0 only	F1-score for T0 only
XGBoost	92%	90%	84%	84%	84%	82%
AdaBoost	86%	86%	86%	85%	59%	58%
RForest	73%	69%	78%	77%	76%	72%
Naive Bayes	73%	72%	70%	69%	43%	46%
SVM	95%	95%	89%	90%	73%	68%
Neural Network	95%	94%	84%	82%	70%	69%
Logistic Regression	92%	93%	92%	93%	68%	68%

We can see that if we use only T2 variables to make classifications, we can have a close accuracy/F1 score as using the whole variables, and for some algorithms, the performance is even better. As for only using T0 variables, we can see most of them drop significantly, except for Random Forest. Therefore, we can claim that T2 variables make a huge influence on making the classification of groups.

However, I also generate another new question: does any variable in T2 can provide some great unique information, even though they are dependent variables?

Now I am considering testing it with ‘A_C2W1T2’ in XGBoost and Adaboost, using scramble and OOB methods to test its importance.

	Accuracy Scrambled	F1-score Scarmbled	Accuracy OOB	F1-score OOB
XGBoost	89%	88%	89%	88%
AdaBoost	95%	94%	95%	94%

We can see that the two models generate the exact same result no matter using scrambled or OOB to test the variance importance, so we can say that the scrambled ‘A_C2W1T2’ does not influence the model. However, we can see that the model still has good performance, AdaBoost’s result is even better.

That may explain how the dependent variables influence the model.

4. Insights

From the scatter plots of the variables, lots of the values of T2 features from group 96db are distributed under those of the values from group control.

By using each powerful Machine Learning method, we can almost classify each group of the noise exposure by using stimulus levels and ABR metrics, especially to classify the control group and the group under 96db.

In order to know the importance of T2 variables, I calculate the variance importances for most of the methods and then compare the results using the same algorithms using the only T2 variables and T0 variables, accordingly, we can see that T2 variables provide great importance to the classifications, while T0 variables do not perform as well as T2 variables.

However, because these variables in these dataset are dependent variables, losing one variable will not affect the result, and none of these variables can dominate the classification, because other features can influence the result, too.

Moreover, in this dataset, SVM and Logistic Regression should be the best algorithm to do the classification, not only because of its high accuracy and F1 score, but also the huge difference that different variables can bring us.

5. Errors and Mistakes

There are many missing values that may affect the results. Although using imputing methods can deal with the issue, it may affect the result.

The dataset is kind of imbalanced, so it may be hard to make classification to group 91db.

Variance importance should use OOB, scramble variables, Model Reliance, Conditional Model Reliance, and Conditional Model Class Reliance to make more specific analysis with every feature and apply to more algorithms.

Libraries Appendix References/Citations

XGBoost Documentation — xgboost 2.0.2 documentation, https://xgboost.readthedocs.io/en/stable/. Accessed 5 December 2023.

"Code examples." *Keras*, https://keras.io/examples/. Accessed 5 December 2023.

"Multiclass classification using neural network." *Kaggle*, https://www.kaggle.com/code/asamad06/multiclass-classification-using-neural-network. Accessed 5 December 2023.

"sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. Accessed 5 December 2023.

"sklearn.ensemble.RandomForestRegressor — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Accessed 5 December 2023.

"sklearn.impute.KNNImputer — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html. Accessed 5 December 2023.

"sklearn.linear_model.LogisticRegression — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed 5 December 2023.

"sklearn.model_selection.cross_val_score — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html. Accessed 5 December 2023.

"sklearn.model_selection.GridSearchCV — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed 5 December 2023.

"sklearn.model_selection.KFold — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html. Accessed 5 December 2023.

"sklearn.naive_bayes.ComplementNB — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB. Accessed 5 December 2023.

"sklearn.naive_bayes.GaussianNB — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB. Accessed 5 December 2023.

"sklearn.preprocessing.StandardScaler — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed 5 December 2023.

"sklearn.svm.SVC — scikit-learn 1.3.2 documentation." *Scikit-learn*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed 5 December 2023.

Other References

Friedman, Jerome H. “Greedy Function Approximation: A Gradient Boosting Machine.” The Annals of Statistics, vol. 29, no. 5, 2001, pp. 1189–232. JSTOR, http://www.jstor.org/stable/2699986. Accessed 5 Dec. 2023.

Freund, Yoav, and Robert E. Schapire. “A Short Introduction to Boosting.” Journal of Japanese Society for Artificial Intelligence, vol. 14, no. 5, 1999, pp. 771-780.

Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

“Naive Bayes Algorithm in ML: Simplifying Classification Problems.” Turing, https://www.turing.com/kb/an-introduction-to-naive-bayes-algorithm-for-beginners#what-is-the-naive-bayes-algorithm? Accessed 5 December 2023.

Choi, Rene. “Introduction to Machine Learning, Neural Networks, and Deep Learning.” Transl Vis Sci Technol, vol. 9, no. 2, 2020, p. 14.

Jurafsky, Dan, and James H. Martin. Speech and Language Processing, 2nd Edition. Pearson Prentice Hall, 2008. Accessed 5 December 2023.

Brownlee, Jason. “Multi-Class Classification Tutorial with the Keras Deep Learning Library - MachineLearningMastery.com.” *Machine Learning Mastery*, 7 August 2022, https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/. Accessed 5 December 2023.

Brownlee, Jason. “One-vs-Rest and One-vs-One for Multi-Class Classification - MachineLearningMastery.com.” *Machine Learning Mastery*, 27 April 2021,

https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/. Accessed 5 December 2023.

Brownlee, Jason. “Understand the Impact of Learning Rate on Neural Network Performance - MachineLearningMastery.com.” *Machine Learning Mastery*, 12 September 2020, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. Accessed 5 December 2023.

McCaffrey, James. “Multi-Class Classification Using a scikit Neural Network.” *Visual Studio Magazine*, 3 April 2023, https://visualstudiomagazine.com/articles/2023/04/03/scikit-neural-network.aspx. Accessed 5 December 2023.

Macina, Jakub. “Determining the most contributing features for SVM classifier in sklearn.” *Stack Overflow*, 11 January 2017, https://stackoverflow.com/questions/41592661/determining-the-most-contributing-features-for-svm-classifier-in-sklearn. Accessed 5 December 2023.

Code

```
# -*- coding: utf-8 -*-
"""project.ipynb

Automatically generated by Colaboratory.

Original file is located at
   https://colab.research.google.com/drive/1yM-F5Bplz13lss7BeMjlncbibYnW20Qg
"""

import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from matplotlib import pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import ComplementNB
from sklearn import svm
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import KNNImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor

!pip install scikeras

from xgboost import XGBClassifier
from xgboost import plot_importance

import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.optimizers import SGD
from scikeras.wrappers import KerasClassifier

from imblearn.over_sampling import SMOTE

from google.colab import files
uploaded = files.upload()

import io
data = pd.read_csv(io.BytesIO(uploaded["FINAL Animal Data 2022.csv"]))

#drop unnecessary columns
```

```
data.drop(["SubjectID","Run"],axis=1,inplace=True)
oversample = SMOTE()

#plot class distributions
groups = ['control','91db','96db']
group_len  =
[len(data[(data["Group"]=="control")]),len(data[(data["Group"]=="91db")]),len(data[(data["Gr
oup"]=="96db")])]
plt.bar(x= groups,height = group_len)

#plot class distributions
groups =
['control,70_70','control,70_90','control,90_90','91db,70_70','91db,70_90','91db,90_90','96d
b,70_70','96db,70_90','96db,90_90']
group_len  = [len(data[(data["Group"]=="control") &
(data["Levels"]=="70_70")]),len(data[(data["Group"]=="control") &
(data["Levels"]=="70_90")]),len(data[(data["Group"]=="control") &
(data["Levels"]=="90_90")]),len(data[(data["Group"]=="91db") &
(data["Levels"]=="70_70")]),len(data[(data["Group"]=="91db") &
(data["Levels"]=="70_90")]),len(data[(data["Group"]=="91db") & (data["Levels"]=="90_90")]),
            len(data[(data["Group"]=="96db") &
(data["Levels"]=="70_70")]),len(data[(data["Group"]=="96db") &
(data["Levels"]=="70_90")]),len(data[(data["Group"]=="96db") & (data["Levels"]=="90_90")])]
plt.xticks(fontsize = 6)
plt.bar(x= groups,height = group_len,width = 0.3)

#find rows with missing values
data[data.isnull().any(axis=1)]

#order the variables with the most missing values
order = []
for i in data.columns[data.isna().any()]:
   order.append((data[i].isna().sum(),i))
order.sort(reverse=True)
print(order)

"""#impute for train and test separately.

"""

#imputting process using KNN
train, test = train_test_split(data, test_size=0.2, random_state=42)
data_add_train = pd.DataFrame(columns=data.columns)
data_add_test = pd.DataFrame(columns=data.columns)
for i in ["control","91db","96db"]:
   for j in ["70_70","70_90","90_90"]:
       group_data = train[(train["Group"]==i) & (train["Levels"]==j)].iloc[:,2:]
       imputer = KNNImputer(n_neighbors=6)
       temp = train[(train["Group"]==i) & (train["Levels"]==j)].iloc[:,:2]
       temp = temp.reset_index(drop=True)
       temp = pd.concat([temp,pd.DataFrame(imputer.fit_transform(group_data),columns =
train.columns[2:])],axis=1)
```

```
       data_add_train = pd.concat([data_add_train,temp],axis=0)
       group_data = test[(test["Group"]==i) & (test["Levels"]==j)].iloc[:,2:]
       if(len(group_data)>0):
         imputer = KNNImputer(n_neighbors=6)
         temp = test[(test["Group"]==i) & (test["Levels"]==j)].iloc[:,:2]
         temp = temp.reset_index(drop=True)
         temp = pd.concat([temp,pd.DataFrame(imputer.fit_transform(group_data),columns =
test.columns[2:])],axis=1)
         data_add_test = pd.concat([data_add_test,temp],axis=0)

#plot class distributions for train dataset
groups =
['control,70_70','control,70_90','control,90_90','91db,70_70','91db,70_90','91db,90_90','96d
b,70_70','96db,70_90','96db,90_90']
group_len  = [len(train[(train["Group"]=="control") &
(train["Levels"]=="70_70")]),len(train[(train["Group"]=="control") &
(train["Levels"]=="70_90")]),len(train[(train["Group"]=="control") &
(train["Levels"]=="90_90")]),len(train[(train["Group"]=="91db") &
(train["Levels"]=="70_70")]),len(train[(train["Group"]=="91db") &
(train["Levels"]=="70_90")]),len(train[(train["Group"]=="91db") &
(train["Levels"]=="90_90")]),
           len(train[(train["Group"]=="96db") &
(train["Levels"]=="70_70")]),len(train[(train["Group"]=="96db") &
(train["Levels"]=="70_90")]),len(train[(train["Group"]=="96db") &
(train["Levels"]=="90_90")])]
plt.xticks(fontsize = 6)
plt.bar(x= groups,height = group_len,width = 0.3)

#plot class distributions for test dataset
groups =
['control,70_70','control,70_90','control,90_90','91db,70_70','91db,70_90','91db,90_90','96d
b,70_70','96db,70_90','96db,90_90']
group_len  = [len(test[(test["Group"]=="control") &
(test["Levels"]=="70_70")]),len(test[(test["Group"]=="control") &
(test["Levels"]=="70_90")]),len(test[(test["Group"]=="control") &
(test["Levels"]=="90_90")]),len(test[(test["Group"]=="91db") &
(test["Levels"]=="70_70")]),len(test[(test["Group"]=="91db") &
(test["Levels"]=="70_90")]),len(test[(test["Group"]=="91db") & (test["Levels"]=="90_90")]),
           len(test[(test["Group"]=="96db") &
(test["Levels"]=="70_70")]),len(test[(test["Group"]=="96db") &
(test["Levels"]=="70_90")]),len(test[(test["Group"]=="96db") & (test["Levels"]=="90_90")])]
plt.xticks(fontsize = 6)
plt.bar(x= groups,height = group_len,width = 0.3)

"""Blue: control group

Orange: 91db group

Green: 96db group
"""

#plot values distributions for each vars in train dataset
```

```
g_c = data_add_train[(data_add_train["Group"]=="control")]
g_91 = data_add_train[(data_add_train["Group"]=="91db")]
g_96 = data_add_train[(data_add_train["Group"]=="96db")]
fig, axes = plt.subplots(nrows=10, ncols=10,figsize=(20,25))
for i in range(100):
 axes[int(i/10),i%10].scatter(x=range(len(g_c.iloc[:,i+2])),y = g_c.iloc[:,i+2],s=8,color =
'tab:blue')
 axes[int(i/10),i%10].scatter(x=range(len(g_91.iloc[:,i+2])),y = g_91.iloc[:,i+2],s=8,color
= 'tab:orange')
 axes[int(i/10),i%10].scatter(x=range(len(g_96.iloc[:,i+2])),y = g_96.iloc[:,i+2],s=8,color
= 'tab:green')
 axes[int(i/10),i%10].set_title(data_add_train.columns[i+2],fontsize = 8)

#plot values distributions for each vars in test dataset
g_c = data_add_test[(data_add_test["Group"]=="control")]
g_91 = data_add_test[(data_add_test["Group"]=="91db")]
g_96 = data_add_test[(data_add_test["Group"]=="96db")]
fig, axes = plt.subplots(nrows=10, ncols=10,figsize=(20,25))
for i in range(100):
 axes[int(i/10),i%10].scatter(x=range(len(g_c.iloc[:,i+2])),y = g_c.iloc[:,i+2],s=8,color =
'tab:blue')
 axes[int(i/10),i%10].scatter(x=range(len(g_91.iloc[:,i+2])),y = g_91.iloc[:,i+2],s=8,color
= 'tab:orange')
 axes[int(i/10),i%10].scatter(x=range(len(g_96.iloc[:,i+2])),y = g_96.iloc[:,i+2],s=8,color
= 'tab:green')
 axes[int(i/10),i%10].set_title(data_add_test.columns[i+2],fontsize = 8)

le = LabelEncoder()
#y_train: 0-91db, 1-96db, 2-control
X_train,y_train,X_test,y_test =
data_add_train.iloc[:,1:],le.fit_transform(data_add_train.iloc[:,0]),data_add_test.iloc[:,1:
],le.fit_transform(data_add_test.iloc[:,0])

#get dummies for the categorical values stimulus level
X_train = pd.concat([pd.get_dummies(X_train.iloc[:,0]),X_train.iloc[:,1:]],axis=1)
X_test = pd.concat([pd.get_dummies(X_test.iloc[:,0]),X_test.iloc[:,1:]],axis=1)

#define Kfolds
kfolds = KFold(5,shuffle=True,random_state=1)

#store accuracies and f1 scores
Accuracies_all = []
F1_all = []

"""For each algorithm, following these steps:

1. tune the paramters with grids

2. list each model's paramters

3. plot each model's performance(score)
```

```
4. fit the best model with train dataset, and then train the X_test

5. print out the accuracy and F1 score for train dataset and test dataset

(6.) some models can plot the variances importances. If possible,plot them

There are many duplicated codes for scaling and oversampling:

scaling the data: scaler = preprocessing.StandardScaler()

oversampling the train set : X, y = oversample.fit_resample(X_train_scaled,y_train)

def f_importances function shows up several times in order to plot the coefficients of
features in svm and logistic algorithm.

## **XGBoost Algorithm**
"""

#tune the paramters with grids
grid = {
   'max_depth': [2,4,6,9],
   'n_estimators':[50,100,150],
   'learning_rate':[0.3,0.5,1],
   'reg_alpha':[0,0.01,0.1,0.5],
   'reg_lambda':[0,0.1,1]
}
models = GridSearchCV(estimator=XGBClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train, y_train)
models.best_params_

#list every model's parameters
models.cv_results_['params']

#plot the scores(performance) for each model
plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

#fit the best model with train dataset, and then train the X_test
model = XGBClassifier(max_depth = 4, learning_rate = 0.5, n_estimators= 50,reg_lambda =
1,reg_alpha = 0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train, model.predict(X_train)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train, model.predict(X_train),average
= 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
Accuracies_all.append(accuracy)
F1_all.append(f1)
```

```
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])

"""Compare our model with default parameter"""

model = XGBClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train, model.predict(X_train)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train, model.predict(X_train),average
= 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **Adaboost**
same steps as XGBoost
"""

grid = {
   'algorithm': ["SAMME","SAMME.R"],
   'n_estimators':[10,30,50,70,100,150],
   'learning_rate':[0.5,1,1.5,2]
}
models = GridSearchCV(estimator = AdaBoostClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train, y_train)
print(models.best_params_,models.best_score_)

models.cv_results_['params']

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 1, n_estimators= 100)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
# print("Accuracy: ",accuracy)
# print("f1 score: ",f1)
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train, model.predict(X_train)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train, model.predict(X_train),average
= 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
```

```
print(metrics.classification_report(y_test,y_pred))
Accuracies_all.append(accuracy)
F1_all.append(f1)
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])

"""## **Naive Bayes**

Naive Bayes, including three sub models:

1. Not dealing with imbalance issue using GaussianNB

2. dealing with imbalance issue using GaussianNB

3. ComplementNB
"""

grid = {
   'priors':
[None,[0.34,0.33,0.33],[len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/l
en(y_train),len(y_train[y_train==[2]])/len(y_train)],[len(y_train[y_train==[2]])/len(y_train
),len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/len(y_train)]]
}
models = GridSearchCV(estimator = GaussianNB(), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
models.fit(X_train_scaled, y_train)
print(models.best_params_,models.best_score_)

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

#not using SMOTE
model = GaussianNB(priors = [0.34, 0.33, 0.33])
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_scaled)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_scaled),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
```

```
#Using Smote to deal with imbalance issue
grid = {
   'priors':
[None,[0.34,0.33,0.33],[len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/l
en(y_train),len(y_train[y_train==[2]])/len(y_train)],[len(y_train[y_train==[2]])/len(y_train
),len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/len(y_train)]]
}
models = GridSearchCV(estimator = GaussianNB(), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X, y = oversample.fit_resample(X_train_scaled,y_train)
models.fit(X, y)
print(models.best_params_,models.best_score_)

models.cv_results_['params']

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = GaussianNB(priors = [0.0958904109589041, 0.410958904109589, 0.4931506849315068])
model.fit(X, y)
y_pred = model.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
Accuracies_all.append(accuracy)
F1_all.append(f1)
print(metrics.classification_report(y_test,y_pred))

"""Compare Gaussian NB with ComplementNB, Complement NB performs worse than Gaussian"""

model = ComplementNB()
X, y = oversample.fit_resample(X_train,y_train)
model.fit(X, y)
y_pred = model.predict(X_test_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **SVM**"""

scaler = preprocessing.StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-4,1e-3,0.01,0.1,0.5],'C': [0.1,1,1.5,10]},
{'kernel': ['linear'], 'C': [0.1,1,1.5,10]}]
svm_model = GridSearchCV(svm.SVC(), params_grid, cv=kfolds)
svm_model.fit(X_train_scaled, y_train)

print('Best score for training data:', svm_model.best_score_)
# View the best parameters for the model found using grid search
print('Best C:',svm_model.best_estimator_.C)
print('Best Kernel:',svm_model.best_estimator_.kernel)
print('Best Gamma:',svm_model.best_estimator_.gamma)
model = svm_model.best_estimator_
y_pred = model.predict(X_test_scaled)

svm_model.cv_results_['params']

plt.bar(range(len(svm_model.cv_results_['mean_test_score'])),svm_model.cv_results_['mean_tes
t_score'])

accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred, average='weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_scaled)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_scaled),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
Accuracies_all.append(accuracy)
F1_all.append(f1)
print(metrics.classification_report(y_test,y_pred))

def f_importances(coef, names):
   imp = coef
   imp,names = zip(*sorted(zip(imp,names)))
   plt.figure(figsize=(10,15))
   plt.barh(range(len(names)), imp, align='center')
   plt.tick_params(axis='y', labelsize=5)
   plt.yticks(range(len(names)), names)
   plt.show()
f_importances(model.coef_[0], X_train.columns)
f_importances(model.coef_[1], X_train.columns)
f_importances(model.coef_[2], X_train.columns)

"""## **Neural Network**"""

in_dim = len(X_train.columns)
oversample = SMOTE()
X, y = oversample.fit_resample(X_train,data_add_train.iloc[:,0])
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test)
```

```
"""Build model because we cannot tune the NNmodel directly with paramters"""

def NNModel(nodes,layer,actf,optimz):
 model = Sequential()
 for i in range(layer):
   model.add(Dense(100, input_dim = in_dim, activation = actf))
 model.add(Dense(3, activation = 'softmax'))
 if(optimz=="SGD"):
   opt = SGD(learning_rate = 0.01)
 else:
   opt = 'Adam'
 model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
 return model

"""use Neurons(50/100),activation function(relu/sigmoid),optimization function(SGD/Adam) to
compare the performance manually"""

accuracies = []
model = NNModel(50,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
```

```
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())

print('Accuracies in order:\n',
     '50 Neurons/layer,relu,SGD\n','50 Neurons/layer,relu,Adam,\n','50
Neurons/layer,sigmoid,SGD\n','50 Neurons/layer,sigmoid,Adam,\n','50
Neurons/layer,relu,SGD\n','50 Neurons/layer,relu,Adam,\n','50
Neurons/layer,sigmoid,SGD\n','50 Neurons/layer,sigmoid,Adam,\n')

plt.bar(range(len(accuracies)),accuracies)

in_dim = len(X_train.columns)
oversample = SMOTE()
X, y = oversample.fit_resample(X_train,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test)
#build the best model from previous selection
model = Sequential()
model.add(Dense(100, input_dim = in_dim, activation = 'relu'))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
model.fit(train_features, pd.get_dummies(y), epochs = 30, batch_size = 16,verbose = 2)
scores = model.evaluate(test_features, pd.get_dummies(y_test))
for i, m in enumerate(model.metrics_names): #print loss and accuracy
   print("\n%s: %.3f"% (m, scores[i]))
y_pred = model.predict(test_features).round() #to get the predicted class for X_test after
scaling
f1 = metrics.f1_score(y_pred,pd.get_dummies(y_test),average = 'weighted')
print(f1)
print(metrics.classification_report(pd.DataFrame(y_pred),pd.get_dummies(y_test)))
Accuracies_all.append(scores[1])
F1_all.append(f1)

"""## **Random Forest**"""

grid = {
   'criterion':["squared_error","absolute_error"],
   'max_depth': [2,3,5,7,10],
   'max_features': [3,10,"sqrt"],
   'n_estimators':[50,100,150]
}
model = GridSearchCV(estimator=RandomForestRegressor(), param_grid=grid, cv= kfolds)
X, y = oversample.fit_resample(X_train,y_train)
scaler = preprocessing.StandardScaler()
```

```
train_features = scaler.fit_transform(X)
model.fit(train_features, y)
model.best_params_

model.cv_results_['params']

plt.bar(range(len(model.cv_results_['mean_test_score'])),model.cv_results_['mean_test_score'
])

model = RandomForestRegressor(criterion = 'squared_error',max_depth = 10,max_features =
10,n_estimators= 150)
X, y = oversample.fit_resample(X_train,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test)
model.fit(train_features, y)
y_pred = model.predict(test_features)
accuracy = metrics.accuracy_score(y_test, [round(i) for i in y_pred])
f1 = metrics.f1_score(y_test,[round(i) for i in y_pred], average='weighted')
print("Accuracy on Train dataset:", metrics.accuracy_score(y, [round(i) for i in
model.predict(train_features)]))
print("F1 on Train dataset:", metrics.f1_score(y, [round(i) for i in
model.predict(train_features)],average = 'weighted'))
print("Accuracy on Test dataset:", accuracy)
print("F1 on Test dataset:", f1)
print(metrics.classification_report(y_test,  [round(i) for i in y_pred]))
Accuracies_all.append(accuracy)
F1_all.append(f1)

importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train.columns)[indices])

"""## **Logistic**"""

grid = {
   'penalty': ['l1', 'l2'],
   'C':[0.5,1,1.5],
   'tol':[0.0001,0.001,0.01,0.1]
}
models = GridSearchCV(estimator=LogisticRegression(solver = 'liblinear',multi_class =
'ovr'), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X, y = oversample.fit_resample(X_train_scaled,y_train)
models.fit(X, y)
models.best_params_
```

```
models.cv_results_['params']

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = LogisticRegression(solver = 'liblinear',multi_class = 'ovr', C =  1, penalty = 'l1',
tol = 0.01)
model.fit(X, y)
y_pred = model.predict(X_test_scaled)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
Accuracies_all.append(accuracy)
F1_all.append(f1)
def f_importances(coef, names):
   imp = coef
   imp,names = zip(*sorted(zip(imp,names)))
   plt.figure(figsize=(10,15))
   plt.barh(range(len(names)), imp, align='center')
   plt.tick_params(axis='y', labelsize=5)
   plt.yticks(range(len(names)), names)
   plt.show()
f_importances(model.coef_[0], X_train.columns)
f_importances(model.coef_[1], X_train.columns)
f_importances(model.coef_[2], X_train.columns)

"""## **Accuracies and F1 score for each algorithm**"""

algos = ['XGBoost','AdaBoost','NaiveBayes','SVM','NeuralNetwork','RandomForest','Logistic']
plt.figure(figsize=(10,6))
plt.bar(algos,Accuracies_all,width =1,label = 'Accuracies')
plt.bar(algos,F1_all,label = 'F1')
plt.legend()

"""## **What happens if we only use the T2 variables? **
## **Repeat above steps/algorithms with T2 variables(+stimulus level) olnly **
"""

X_train_T2 = pd.concat([X_train.iloc[:,:3],X_train.iloc[:,53:]],axis=1)
X_test_T2 = pd.concat([X_test.iloc[:,:3],X_test.iloc[:,53:]],axis=1)

"""## **XGBoost for T2 only**


"""
```

```
grid = {
    'max_depth': [2,4,6,9],
    'n_estimators':[100,150],
    'learning_rate':[0.3,0.5,1],
    'reg_alpha':[0,0.01,0.1,0.5],
    'reg_lambda':[0,0.1,1]
}
models = GridSearchCV(estimator=XGBClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_T2, y_train)
models.best_params_

model = XGBClassifier(max_depth = 2, learning_rate = 0.3, n_estimators= 100,reg_lambda
=0,reg_alpha = 0.5)
model.fit(X_train_T2, y_train)
y_pred = model.predict(X_test_T2)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T2)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T2),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_T2.columns)[indices])

#Now we compare with default parametrs, our chosen model performs better than de the default
model.
model = XGBClassifier()
model.fit(X_train_T2, y_train)
y_pred = model.predict(X_test_T2)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T2)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T2),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
```

```
_ = ax.set_yticklabels(np.array(X_train_T2.columns)[indices])

"""## **Adaboost for T2 only**"""

grid = {
   'algorithm': ["SAMME","SAMME.R"],
   'n_estimators':[10,30,50,70,100,150],
   'learning_rate':[0.5,1,1.5,2]
}
models = GridSearchCV(estimator = AdaBoostClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_T2, y_train)
print(models.best_params_,models.best_score_)

models.cv_results_['params']

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 1, n_estimators= 10)
model.fit(X_train_T2, y_train)
y_pred = model.predict(X_test_T2)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T2)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T2),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_T2.columns)[indices])

"""## **GaussianNB for T2 only**"""

grid = {
   'priors':
[None,[0.34,0.33,0.33],[len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/l
en(y_train),len(y_train[y_train==[2]])/len(y_train)],[len(y_train[y_train==[2]])/len(y_train
),len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/len(y_train)]]
}
models = GridSearchCV(estimator = GaussianNB(), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_T2_scaled = scaler.fit_transform(X_train_T2)
X_test_T2_scaled = scaler.transform(X_test_T2)
models.fit(X_train_T2_scaled, y_train)
print(models.best_params_,models.best_score_)
```

```
print(models.cv_results_['params'])
plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

#not using SMOTE
model = GaussianNB()
model.fit(X_train_T2_scaled, y_train)
y_pred = model.predict(X_test_T2_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T2_scaled)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T2_scaled),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

#using SMOTE
grid = {
   'priors':
[None,[0.34,0.33,0.33],[len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/l
en(y_train),len(y_train[y_train==[2]])/len(y_train)],[len(y_train[y_train==[2]])/len(y_train
),len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/len(y_train)]]
}
models = GridSearchCV(estimator = GaussianNB(), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_T2_scaled = scaler.fit_transform(X_train_T2)
X_test_T2_scaled = scaler.transform(X_test_T2)
oversample = SMOTE()
X, y = oversample.fit_resample(X_train_T2_scaled,y_train)
models.fit(X, y)
print(models.best_params_,models.best_score_)

print(models.cv_results_['params'])
plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = GaussianNB(priors =  [0.4931506849315068, 0.0958904109589041, 0.410958904109589])
model.fit(X, y)
y_pred = model.predict(X_test_T2_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **SVM for T2 only**"""
```

```
scaler = preprocessing.StandardScaler()
X_train_T2_scaled = scaler.fit_transform(X_train_T2)
X_test_T2_scaled = scaler.transform(X_test_T2)
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-4,1e-3,0.01,0.1,0.5],'C': [0.1,1,1.5,10]},
{'kernel': ['linear'], 'C': [0.1,1,1.5,10]}]
svm_model = GridSearchCV(svm.SVC(), params_grid, cv=kfolds)
svm_model.fit(X_train_T2_scaled, y_train)

svm_model.cv_results_['params']

plt.bar(range(len(svm_model.cv_results_['mean_test_score'])),svm_model.cv_results_['mean_tes
t_score'])

print('Best score for training data:', svm_model.best_score_)
# View the best parameters for the model found using grid search
print('Best C:',svm_model.best_estimator_.C)
print('Best Kernel:',svm_model.best_estimator_.kernel)
print('Best Gamma:',svm_model.best_estimator_.gamma)
model = svm_model.best_estimator_
y_pred = model.predict(X_test_T2_scaled)

accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred, average='weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T2_scaled)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T2_scaled),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **Neural Network for T2 only**"""

in_dim = len(X_train_T2.columns)
oversample = SMOTE()
X, y = oversample.fit_resample(X_train_T2,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test_T2)

accuracies = []
model = NNModel(50,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
```

```
model = NNModel(50,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())

print('Accuracies in order:\n',
     '50 Neurons/layer,relu,SGD\n','50 Neurons/layer,relu,Adam,\n','50
Neurons/layer,sigmoid,SGD\n','50 Neurons/layer,sigmoid,Adam,\n','50
Neurons/layer,relu,SGD\n','50 Neurons/layer,relu,Adam,\n','50
Neurons/layer,sigmoid,SGD\n','50 Neurons/layer,sigmoid,Adam,\n')

plt.bar(range(len(accuracies)),accuracies)

model = Sequential()
model.add(Dense(100, input_dim = in_dim, activation = 'relu'))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
model.fit(train_features, pd.get_dummies(y), epochs = 30, batch_size = 16,verbose = 2)
scores = model.evaluate(test_features, pd.get_dummies(data_add_test.iloc[:,0]))
for i, m in enumerate(model.metrics_names):
   print("\n%s: %.3f"% (m, scores[i]))
y_pred = model.predict(test_features).round()
print(metrics.f1_score(y_pred,pd.get_dummies(y_test),average = 'weighted'))
print(metrics.classification_report(pd.DataFrame(y_pred),pd.get_dummies(y_test)))
```

```
"""## **Random Forest for T2 only**"""

grid = {
    'criterion':["squared_error","absolute_error"],
    'max_depth': [2,3,5,7,10],
    'max_features': [3,10,"sqrt"],
    'n_estimators':[50,100,150]
}
model = GridSearchCV(estimator=RandomForestRegressor(), param_grid=grid, cv= kfolds)
X, y = oversample.fit_resample(X_train_T2,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
model.fit(train_features, y)
model.best_params_

model.cv_results_['params']

plt.bar(range(len(model.cv_results_['mean_test_score'])),model.cv_results_['mean_test_score'
])

model = RandomForestRegressor(criterion = 'squared_error',max_depth = 10,max_features =
"sqrt",n_estimators = 150)
X, y = oversample.fit_resample(X_train_T2,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test_T2)
model.fit(train_features, y)
y_pred = model.predict(test_features)
accuracy = metrics.accuracy_score(y_test, [round(i) for i in y_pred])
f1 = metrics.f1_score(y_test,[round(i) for i in y_pred], average='weighted')
print("Accuracy on Train dataset:", metrics.accuracy_score(y, [round(i) for i in
model.predict(train_features)]))
print("F1 on Train dataset:", metrics.f1_score(y, [round(i) for i in
model.predict(train_features)],average = 'weighted'))
print("Accuracy on Test dataset:", accuracy)
print("F1 on Test dataset:", f1)
print(metrics.classification_report(y_test,  [round(i) for i in y_pred]))

importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_T2.columns)[indices])

"""## **Logsitic for T2 only**"""

grid = {
    'penalty': ['l1', 'l2'],
    'C':[0.5,1,1.5],
```

```
    'tol':[0.0001,0.001,0.01,0.1]
}
models = GridSearchCV(estimator=LogisticRegression(solver = 'liblinear',multi_class =
'ovr'), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_T2_scaled = scaler.fit_transform(X_train_T2)
X_test_T2_scaled = scaler.transform(X_test_T2)
X, y = oversample.fit_resample(X_train_T2_scaled,y_train)
models.fit(X, y)
models.best_params_

models.cv_results_['params']

plt.bar(range(len(models.cv_results_['mean_test_score'])),models.cv_results_['mean_test_scor
e'])

model = LogisticRegression(solver = 'liblinear',multi_class = 'ovr', C = 1.5, penalty =
'l1', tol = 0.0001)
model.fit(X, y)
y_pred = model.predict(X_test_T2_scaled)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
def f_importances(coef, names):
    imp = coef
    imp,names = zip(*sorted(zip(imp,names)))
    plt.figure(figsize=(10,15))
    plt.barh(range(len(names)), imp, align='center')
    plt.tick_params(axis='y', labelsize=5)
    plt.yticks(range(len(names)), names)
    plt.show()
f_importances(model.coef_[0], X_train_T2.columns)
f_importances(model.coef_[1], X_train_T2.columns)
f_importances(model.coef_[2], X_train_T2.columns)

"""## *What happens if only using T0?*
## *do the same steps/algorithms as above using T0 variables only*
"""

X_train_T0 = X_train.iloc[:,:53]
X_test_T0 = X_test.iloc[:,:53]

"""## **XGboost for T0 only**"""

grid = {
    'max_depth': [2,4,6,9],
    'n_estimators':[50,100,150],
```

```
   'learning_rate':[0.3,0.5,1],
   'reg_alpha':[0,0.01,0.1,0.5],
   'reg_lambda':[0,0.1,1]
}
models = GridSearchCV(estimator=XGBClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_T0, y_train)
models.best_params_

model = XGBClassifier(max_depth = 6,learning_rate = 0.5, n_estimators= 100,reg_alpha =
0,reg_lambda = 1)
model.fit(X_train_T0, y_train)
y_pred = model.predict(X_test_T0)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T0)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T0),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_T0.columns)[indices])

"""## **Adaboost for T0 only**"""

grid = {
   'algorithm': ["SAMME","SAMME.R"],
   'n_estimators':[10,30,50,70,100,150],
   'learning_rate':[0.5,1,1.5,2]
}
models = GridSearchCV(estimator = AdaBoostClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_T0, y_train)
print(models.best_params_,models.best_score_)

model = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 0.5, n_estimators= 100)
model.fit(X_train_T0, y_train)
y_pred = model.predict(X_test_T0)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
# print("Accuracy: ",accuracy)
# print("f1 score: ",f1)
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T0)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T0),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
```

```
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_T0.columns)[indices])

"""## **Gaussian NB for T0 only**"""

grid = {
   'priors':
[None,[0.34,0.33,0.33],[len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/l
en(y_train),len(y_train[y_train==[2]])/len(y_train)],[len(y_train[y_train==[2]])/len(y_train
),len(y_train[y_train==[0]])/len(y_train),len(y_train[y_train==[1]])/len(y_train)]]
}
models = GridSearchCV(estimator = GaussianNB(), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_T0_scaled = scaler.fit_transform(X_train_T0)
X_test_T0_scaled = scaler.transform(X_test_T0)
X, y = oversample.fit_resample(X_train_T0_scaled,y_train)
models.fit(X, y)
print(models.best_params_,models.best_score_)

model = GaussianNB(priors =  [0.4931506849315068, 0.0958904109589041, 0.410958904109589])
model.fit(X, y)
y_pred = model.predict(X_test_T0_scaled)
accuracy = metrics.accuracy_score(y_pred, y_test)
f1 = metrics.f1_score(y_pred, y_test,average="weighted")
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **SVM for T0 only**"""

scaler = preprocessing.StandardScaler()
X_train_T0_scaled = scaler.fit_transform(X_train_T0)
X_test_T0_scaled = scaler.transform(X_test_T0)
params_grid = [{'kernel': ['rbf'], 'gamma': [1e-4,1e-3,0.01,0.1,0.5],'C': [0.1,1,1.5,10]},
{'kernel': ['linear'], 'C': [0.1,1,1.5,10]}]
svm_model = GridSearchCV(svm.SVC(), params_grid, cv=kfolds)
svm_model.fit(X_train_T0_scaled, y_train)

print('Best score for training data:', svm_model.best_score_)
# View the best parameters for the model found using grid search
print('Best C:',svm_model.best_estimator_.C)
print('Best Kernel:',svm_model.best_estimator_.kernel)
```

```
print('Best Gamma:',svm_model.best_estimator_.gamma)
model = svm_model.best_estimator_
y_pred = model.predict(X_test_T0_scaled)

accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test, y_pred, average='weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_T0_scaled)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_T0_scaled),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))

"""## **Neural Network for T0 only**"""

in_dim = len(X_train_T0.columns)
oversample = SMOTE()
X, y = oversample.fit_resample(X_train_T0,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test_T0)

accuracies = []
model = NNModel(50,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(50,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'relu','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
```

```
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','SGD')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())
model = NNModel(100,3,'sigmoid','Adam')
estimator = KerasClassifier(model, epochs=30, batch_size=16, verbose=0)
results = cross_val_score(estimator, train_features, pd.get_dummies(y), cv=kfolds)
print("*****Model Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
accuracies.append(results.mean())

model = Sequential()
#show the prunning by change #nodes, activ function,optimizer(opt/adam)
model.add(Dense(50, input_dim = in_dim, activation = 'relu'))
model.add(Dense(50, activation = 'relu'))
model.add(Dense(50, activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
model.fit(train_features, pd.get_dummies(y), epochs = 30, batch_size = 16,verbose = 2)
scores = model.evaluate(test_features, pd.get_dummies(y_test))
for i, m in enumerate(model.metrics_names):
   print("\n%s: %.3f"% (m, scores[i]))
y_pred = model.predict(test_features).round()
metrics.f1_score(y_pred,pd.get_dummies(y_test),average = 'weighted')
print(metrics.classification_report(pd.DataFrame(y_pred),pd.get_dummies(y_test)))

"""## **Random Forest for T0 only**"""

grid = {
   'criterion':["squared_error","absolute_error"],
   'max_depth': [2,3,5,7,10],
   'max_features': [3,10,"sqrt"],
   'n_estimators':[50,100,150]
}
model = GridSearchCV(estimator=RandomForestRegressor(), param_grid=grid, cv= kfolds)
X, y = oversample.fit_resample(X_train_T0,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
model.fit(train_features, y)
model.best_params_

model = RandomForestRegressor(criterion = 'squared_error',max_depth = 7,max_features =
"sqrt",n_estimators = 100)
X, y = oversample.fit_resample(X_train_T0,y_train)
scaler = preprocessing.StandardScaler()
train_features = scaler.fit_transform(X)
test_features = scaler.transform(X_test_T0)
model.fit(train_features, y)
y_pred = model.predict(test_features)
accuracy = metrics.accuracy_score(y_test, [round(i) for i in y_pred])
```

```
f1 = metrics.f1_score(y_test,[round(i) for i in y_pred], average='weighted')
print("Accuracy on Train dataset:", metrics.accuracy_score(y, [round(i) for i in
model.predict(train_features)]))
print("F1 on Train dataset:", metrics.f1_score(y, [round(i) for i in
model.predict(train_features)],average = 'weighted'))
print("Accuracy on Test dataset:", accuracy)
print("F1 on Test dataset:", f1)
print(metrics.classification_report(y_test,  [round(i) for i in y_pred]))

"""## **Logistic for T0 only**"""

grid = {
   'penalty': ['l1', 'l2'],
   'C':[0.5,1,1.5],
   'tol':[0.0001,0.001,0.01,0.1]
}
models = GridSearchCV(estimator=LogisticRegression(solver = 'liblinear',multi_class =
'ovr'), param_grid=grid, cv= kfolds)
scaler = preprocessing.StandardScaler()
X_train_T0_scaled = scaler.fit_transform(X_train_T0)
X_test_T0_scaled = scaler.transform(X_test_T0)
X, y = oversample.fit_resample(X_train_T0_scaled,y_train)
models.fit(X, y)
models.best_params_

model = LogisticRegression(solver = 'liblinear',multi_class = 'ovr', C = 1, penalty = 'l2',
tol = 0.0001)
model.fit(X, y)
y_pred = model.predict(X_test_T0_scaled)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y, model.predict(X)))
print("f1 score on train Dataset: ",metrics.f1_score(y, model.predict(X),average =
'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
def f_importances(coef, names):
   imp = coef
   imp,names = zip(*sorted(zip(imp,names)))
   plt.figure(figsize=(10,15))
   plt.barh(range(len(names)), imp, align='center')
   plt.tick_params(axis='y', labelsize=5)
   plt.yticks(range(len(names)), names)
   plt.show()
f_importances(model.coef_[0], X_train_T0.columns)
f_importances(model.coef_[1], X_train_T0.columns)
f_importances(model.coef_[2], X_train_T0.columns)

"""# *What happens to scramble/oob some important variables shown in the previous plot?*
# *Scramble and OOB A_C2W1T2 and compare results for XGBoost and Adaboost*
"""
```

```
X_train_scramble =
pd.concat([pd.DataFrame(X_train.iloc[:,:54]).reset_index(drop=True),pd.DataFrame(X_train['A_
C2W1T2'].sample(frac=1)).reset_index(drop=True),pd.DataFrame(X_train.iloc[:,55:]).reset_inde
x(drop=True)],axis=1)
X_train_oob = X_train.loc[:, X_train.columns != 'A_C2W1T2']
X_test_oob = X_test.loc[:, X_test.columns != 'A_C2W1T2']

"""## **XGboost for Scramble(using all other variables)**"""

model = XGBClassifier()
model.fit(X_train_scramble, y_train)
y_pred = model.predict(X_test)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_scramble)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_scramble),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_scramble.columns)[indices])

"""## **XGboost for OOB(using all other variables)**"""

model = XGBClassifier()
model.fit(X_train_oob, y_train)
y_pred = model.predict(X_test_oob)
accuracy  = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_oob)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_oob),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_oob.columns)[indices])
```

```
"""## **Adaboost for Scramble(using all other variables)**"""

grid = {
   'algorithm': ["SAMME","SAMME.R"],
   'n_estimators':[10,30,50,70,100,150],
   'learning_rate':[0.5,1,1.5,2]
}
models = GridSearchCV(estimator = AdaBoostClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_scramble, y_train)
print(models.best_params_,models.best_score_)

model = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 1, n_estimators= 70)
model.fit(X_train_scramble, y_train)
y_pred = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_scramble)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_scramble),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_scramble.columns)[indices])

"""## **Adaboost for OOB(using all other variables)**"""

grid = {
   'algorithm': ["SAMME","SAMME.R"],
   'n_estimators':[10,30,50,70,100,150],
   'learning_rate':[0.5,1,1.5,2]
}
models = GridSearchCV(estimator = AdaBoostClassifier(), param_grid=grid, cv= kfolds)
models.fit(X_train_oob, y_train)
print(models.best_params_,models.best_score_)

model = AdaBoostClassifier(algorithm = 'SAMME', learning_rate = 1, n_estimators= 70)
model.fit(X_train_oob, y_train)
y_pred = model.predict(X_test_oob)
accuracy = metrics.accuracy_score(y_test, y_pred)
f1 = metrics.f1_score(y_test,y_pred,average = 'weighted')
print("Accuracy on train Dataset: ",metrics.accuracy_score(y_train,
model.predict(X_train_oob)))
print("f1 score on train Dataset: ",metrics.f1_score(y_train,
model.predict(X_train_oob),average = 'weighted'))
print("Accuracy on test Dataset: ",accuracy)
```

```
print("f1 score on test Dataset: ",f1)
print(metrics.classification_report(y_test,y_pred))
importances = model.feature_importances_
indices = np.argsort(importances)
fig, ax = plt.subplots(figsize=(10,20))
ax.barh(range(len(importances)), importances[indices])
ax.set_yticks(range(len(importances)))
ax.tick_params(axis='y', labelsize=5)
_ = ax.set_yticklabels(np.array(X_train_oob.columns)[indices])
```