Anjali Ramesh Urmi Suresh

Ashna Sood Sarvottam Jalsingh

Professor Virginia de Sa

COGS 189: Brain Computer Interfaces

17 March 2021

Machine Learning Analysis on EEG Confusion Data

Introduction and Motivation

Understanding EEG data is critical to understanding BCI research. From what we have learned in this class, EEG data is a large part of how scientists are able to analyze and extrapolate brain imaging data in an easy, efficient, and accurate way. Since so many brain imaging studies have been conducted through the use of EEG signals and waves, we decided to find a dataset that allowed us to explore the format of EEG data, through delta, sigma, theta, and gamma waves. To do this, we use various classification models in order to figure out if there exists certain kinds of models that are most appropriate for EEG data.

BCI research has gained a lot of attention in recent years and continues to grow in popularity, so we thought that it would be important to perform more model analyses on these kinds of datasets. It was also interesting to find an EEG dataset that contained information that was useful and relevant to ourselves. As stated in *Brain-Computer Interfaces: A Gentle Introduction*, BCI research has been developing quite quickly, and "every year, there are more BCI-related papers, conference talks, products, and media articles… Furthermore, BCIs are moving beyond communication tools for people who cannot otherwise communicate" (Graimann et al. 2). Generally, BCI research has become even more useful in recent years, so knowing how

to effectively explore, analyze, and compare results using BCI research was important to us in the big picture.

Since EEG data is a huge part of BCI research, we chose to perform machine learning analyses on an EEG dataset to further contribute to the exploration of what models are best suited for EEG data. The dataset we chose was "Confused Student EEG Brainwave Data" from Kaggle. In this dataset, EEG signal data was collected from 10 college students who were shown a total of 10 MOOC (Massive Open Online Course) videos. Half of these videos consisted of subjects that college students should be familiar with, and half were more complicated subjects. Each video was split so the most confusing parts would be shown. EEG data was only collected during these parts of the clips. After watching the video, the student rated their confusion level on a scale of 1-7, where one corresponded to the least confusing and seven corresponded to the most confusing. These labels were further normalized into binary labels of 1 or 0, or whether the student was confused or not (Wang).

We chose this dataset because it measured confusion, a cognitive phenomenon that affects everyone and is applicable in almost every field. In this specific dataset, the measured confusion could be analyzed in order to determine confusion levels for other videos, such as college lectures or online classes. Since this measured something that was almost guaranteed to affect many lives, we thought it was relevant to analyze.

Related work

There are numerous researchers and papers that utilize various machine learning algorithms to analyze EEG datasets. The dataset that we used was also analyzed in the article *Confused or Not Confused*, in which the problem was identified as a binary classification

problem. This research also had the same prediction task, which was to predict whether the student was confused or not. To solve this problem, the researchers started with dimensionality analysis and reduction, and then went on to use different machine learning techniques to analyze and predict the data (Ni et al). After reading this article, we thought that dimensionality reduction would be a good technique to utilize on this data to better visualize the effect of the easy and difficult video clips on the students' level of confusion. From the article we read in class, "Brain Computer Interfaces, a Review," we learned that Principle Component Analysis was a method of dimensionality reduction, and we attempt to do PCA in the analysis section of the report (Nicolas-Alonso and Gome-Gil 1229-1231). Also, although we used different machine learning models on this data than the ones specifically used in this research paper, we are able to compare our error metrics and model accuracies to theirs, which is one way of validating our results and comparing them to existing research done on this EEG data.

Other similar research was done in *Classifying Confusion: Autodetection of Communicative Misunderstandings using Facial Action Units*. While the dataset we are using classifies participants as confused or not confused based on their reaction to a series of videos, these researchers aimed to classify and detect confusion states based on various facial expressions (Borges et al). This would be useful in fields that regularly use human-computer interaction, such as websites that would like to know when a user is confused based on their facial expressions in order to direct them or help them. The Confused Student EEG Brainwave Data has a similar use, where predicting these confusion states could help classify how confused students are during class lectures, or when a topic is slightly too hard to understand, or when a student is having difficulty keeping up, and ultimately how to further improve the lecture style to

minimize the confusion. The real world applications of this dataset are even more reason for us to understand and analyze EEG data.

Methods

For our analysis we started by cleaning our dataset and making it optimal for our research task, exploratory data analysis (EDA), and lastly finally running and comparing the metrics of our four different models.

Data Collection and Cleaning: The dataset we are using is the "Confused Student EEG Brainwave Data" and within this data we had access to a demographics file and an EEG file, and we cleaned them individually. The EEG data contains 15 attributes, SubjectID, VideoID, Attention, Mediation, Raw, Delta, Theta, Alpha1, Alpha2, Beta1, Beta2, Gamma1, Gamma2, predefined label, and user-definedlabeln. The demographic data contains 4 attributes: subject ID, age, ethnicity, and gender. We dropped unnecessary columns as well as renaming some. In the demographic csv file we turned the 'Gender' column into a binary value column by making the Female variables 1 and the Male variables 0. This was to ensure that there was uniformity among the variables which made it easy to analyze later. We then merged the two datasets on the SubjectID column that was identical in both datasets. Next, we checked if there were any null/missing values in the dataframe that would affect our analysis. Since there were none, we finally had one complete and cleaned dataframe we would be performing our analysis on.

EDA: First we printed out the descriptive statistics of the dataframe including data such as the mean, standard deviation, minimum, and maximum of each variable. Since we wanted to see the correlation between relevant variables, we created a features array that held only the variables SubjectID, VideoID, Attention, Mediation, Raw, Delta, Theta, Alpha1, Alpha2, Beta1, Beta2,

Gamma1, and Gamma2. After printing out a correlation matrix, shown in Figure 1, we turned it into a heatmap that distinctly shows which variables are highly correlated with each other in a concise and readable visualization (*Appendix, Figure 1)*. Some notably high correlations were between Beta2 and Gamma1 at 81%, Gamma1 and Gamma2 at 74%, and between Beta2 and Gamma2 at 74%. We also printed out the counts of the number of confused and not confused samples (6567 confused and 6244 not confused) to verify the roughly even split of the data. Similarly, we looked at the distribution of samples for each video 1-10 (~1200-1400 per video) and also how many samples were for each participant (~1300 per participant). Another step we took to explore the data was to perform univariate analysis for the continuous features of our dataframe. This was a plot of individual boxplots for the variables Attention, Mediation, Raw, Delta, Theta, Alpha1, Alpha2, Beta1, Beta2, Gamma1, and Gamma2. From this plot, we saw the distributions of each variable and could observe if there were any outliers in the data (*Figure 2)*.

Analysis: The first step in our data analysis was defining X to be the features array which includes the important continuous values, and the Y to be the label for confused or not confused. We then split our X and Y datasets into training and testing sets with a 80-20 training/testing split. The classification methods we chose to use were Logistic Regression, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forest. For each model, to help visualize the classification results, we calculated and plotted a confusion matrix as well as printed out a classification report that summarized the main classification metrics (sensitivity/recall, specificity, precision, F1 score). We also calculate the regressor metrics Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) for each model to conduct cross validation in the Results section. Finally we perform K-folds cross validation with a varying number of folds to see the overall generalization error and average accuracy of each model.

1. Logistic regression: Logistic Regression is a classification model that calculates the probability of belonging to one of two classes. We started with logistic regression to see if the variables in our features array could predict the confused or not confused label accurately. To explore different parameter options we started with L2 regularization both with and without standardized data, as well as L1 regularization both with and without standardized data. In the end the combination of L1 regularization with the raw data performed the best. Our training set's accuracy was around 60.52% and our test set's accuracy was around 58.83% (*Figure 3*). Something interesting we noticed was that standardizing the X_train values did not generate a higher accuracy like we had predicted it would have, so we left them unscaled.

2. SVM: Support Vector Machines is another supervised algorithm that uses kernels to transform the data and then classify it. First we attempted to use GridSearchCV to optimize the C, gamma, and kernel parameters, however it took too long for the algorithm to run. Because of this we hand tested different combinations and came to the conclusion that gamma should be left as auto and the rbf kernel performed the best. Then, using gamma as 1/n_features (auto) and the kernel as rbf, we tried different C values and found that C=1000 gave the best accuracy (*Figure 4*). We did try higher C values that had slightly higher accuracies but they took a long time to run and thus did not have any added benefit of being the optimal value. Overall, the SVM model performed the best out of the four models. The training set's accuracy was about 90.16% and the testing set's accuracy was around 76.08% (*Figure 5*).

3. KNN: Our third model was the KNN classification that serves as another simple classification model which classifies data points based on the overall class of the k nearest data points. We performed KNN with 7 different k values (3, 5, 7, 9, 11, 13, and 15) to determine the optimal k value. From the results we see that k=11 produced the highest testing set accuracy with 57.04%.

Even though all the k values are performing a testing accuracy in the same range of 55% to 57%, having a lower k value means having more variance. Additionally, the k values higher than 11 did not produce any higher testing set accuracies. In the end, k=11 produced a training set accuracy of 65.97% and a testing set accuracy of 57.04% (*Figure 6*). We graphed the testing set accuracies for the seven k values we used and decided that a k value of 11 provided the best results (*Figure 7*).

4. Random Forest: Our last model was the Random Forest ensemble learning classifier that is a large collection of individual decision trees working together to output a classification. We made our n_estimators (the number of individual decision trees) 1000. We tested out n_estimators values of 200 and 500, and the testing set accuracy of 1000 _estimators was slightly higher than both 200 and 500. We also tested out 2000 n_estimators and the value was minutely larger but it took a long time to run, so there was no added benefit of increasing the n_estimators past 1000 (*Figure 8)*. Overall, the training accuracy came out to 60.52% and the testing accuracy came out to be 62.15% (*Figure 9*). This was the second best performing model out of the four we ran.

PCA: Although this model was not particularly useful for the overall analysis of the EEG data and predicting the confusion labels, we conducted Principle Component Analysis to reduce the dimensionality of the data and find the most important components (*Figure 10)* While the PCA model was able to do this, we did not find that the reduced dimensionality furthered our analyses.

Results

Throughout these models we discovered that the SVM model performed best with this EEG dataset (*Figure 11*). We had not expected this outcome as we assumed the Random Forest Classifier would execute better as it operates on the idea that a large number of individual

classifications is going to output a more correct label versus an individual classification. However, with the optimization of the parameters that we did with SVM, it resulted in a 76.09% testing accuracy. One major benefit of SVM is that it does perform well with large numbers of variables, which is what we had in this dataset. In order from best performing to worst, our models ranked at SVM, Random Forest, Logistic Regression, and KNN. The error metrics we ran and graphed can confirm this ordering as the SVM model's MSE and RMSE were 0.239 and 0.489 respectively (*Figure 12*).

Overall we were expecting higher testing set accuracies for our models and were unpleasantly surprised at how low they were. We are attributing the low accuracies of the models to the binary classification of the labels. We were also mildly confused about some of the dataset documentation and the variables in the data itself. With more time we could have reached out to the researchers who collected the dataset to clarify about the nominal classification of 1-7 and why it was not present in the dataset even though it was mentioned in the description. Additionally with more time and under different circumstances (due to the COVID-19 pandemic), we would have loved to play a part in the data collection as well as the analysis, allowing ourselves to experience the full spectrum of a BCI research project.

Something that went really well during our project was that after many parameter adjustments, we were able to optimize the SVM model to produce a decent testing accuracy that we were not expecting to achieve. Due to the limited nature of our dataset we assumed that the accuracies in classification may not increase past the 55% threshold that we were originally working with. Fortunately, the SVM classifier did a better job at the classification than expected and validated our results.

Discussion

There were several improvements that could be applied to the current scope of our project. First, we could have split the data differently by taking each subject individually and having the first 80% of the subject's samples become part of the training dataset and the last 20% of the subject's samples become part of the testing dataset, since we originally did a random train/test split. Time could have a major impact on the accuracy of the algorithm since some of the training data points could have been right before the test data points, which would have skewed the results since the model is being trained to correctly predict those test points that are right next in time. Because of this, at the end of our project we redid the train test split using the more accurate method described here and applied the same four models as we did above (*Figure 13*). From these, the KNN algorithm with k=11 performed the best and gave a training accuracy of 66.26% and a testing accuracy of 62.05%. Compared to the randomly split data, the newly split data produced lower accuracies as we had expected it to since we now ensured that data that is located next to each other in time is not in both the train and test sets (*Figure 14*).

We also could have possibly chosen a different dataset (or modified our own) to contain unlabeled data, where data for the User-Defined column would be removed. The purpose of this would be to use unsupervised machine learning algorithms, such as K-Means Clustering, that would separate the dataset into the groups of Confused and Not Confused and give us an idea of the requirements for a piece of data to be a member of either group. Thirdly, we could have used a PCA-like approach where we ourselves determine which features had the most impact on model accuracy by iteratively removing each feature one at a time and training the model. At the end, we would be able to see how the accuracy of the model fluctuated, and choose the most

important features for the final model. We could have also graphed up to 3 variables at a time in a 3-Dimensional graph in order to determine how the features changed in respect to each other.

Fourth, we could have also trained a bidirectional Long Short Term Memory (LSTM) neural network since this type of model yielded the best results in the article *Confused or Not Confused*. By training the neural network, we would have been further able to compare our results of our models with the results stated in the article. Fourth, we could have applied the GridSearchCV function from the Sk-learn's model selection package in order to ascertain the optimal C, gamma, and kernel parameters for the SVM model. We did run the function, however upon noticing that it took much longer to run than anticipated, we decided to drop it and instead learn them manually. As a result, we are still not entirely confident that we chose the optimal parameters. Lastly, we could have done further analysis with the k-folds cross validation method and save the best hyperparameters from all of the folds. We could have then taken these values to retrain the final model.

Appendix

Figure 1:

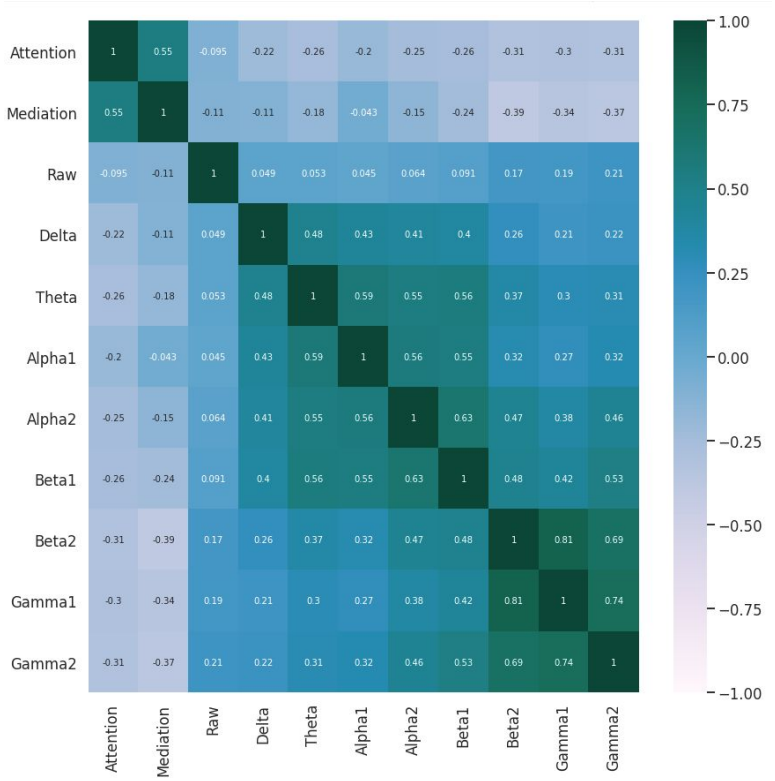


Figure 2:

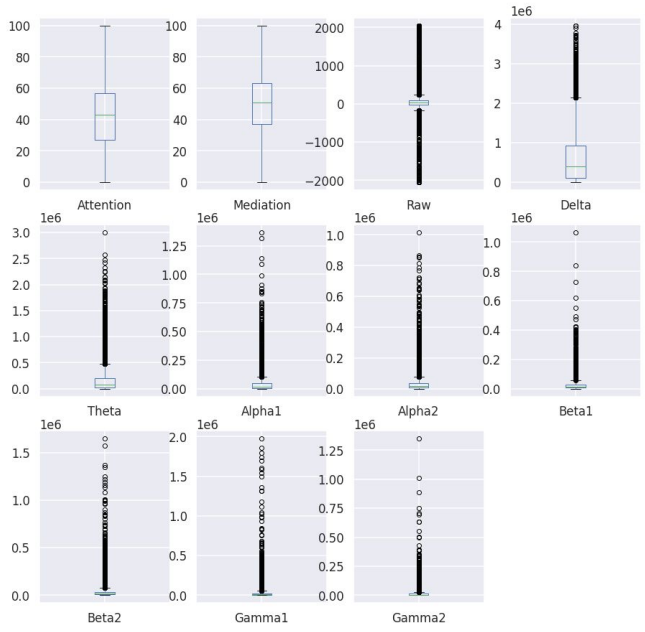
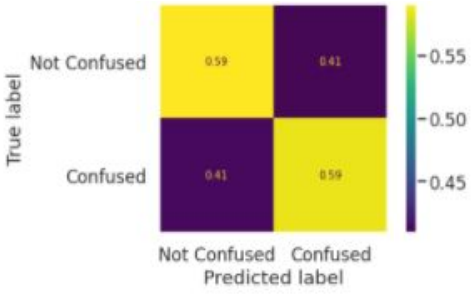


Figure 3:

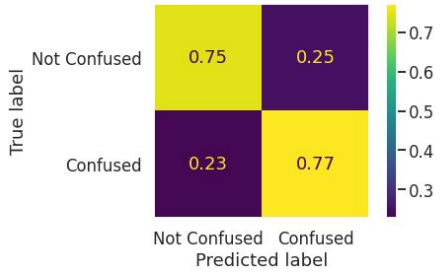



```
Training Set accuracy: 0.6052888368462139
Test Set accuracy: 0.5879828326180258
```

Figure 4:

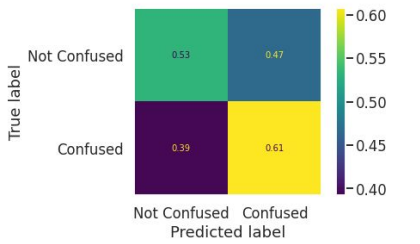
```python
# train SVM model using 5 different C values to determine the optimal parameter
train_acc = []
test_acc = []
C = [1, 10, 100, 1000, 10000, 100000]
for ind, Cval in enumerate(C):
  # create SVM model and train the model
  svm_model = make_pipeline(StandardScaler(), SVC(C=Cval, gamma='auto', kernel="rbf"))
  svm_model.fit(X_train, y_train)
  # store training accuracy for each model
  train_acc.append(svm_model.score(X_train, y_train))
  # calculate predictions and store testing accuracy for each model
  svm_preds = svm_model.predict(X_test)
  test_acc.append(accuracy_score(y_test, svm_preds))
  print(Cval, "C: Train Set acc:", train_acc[ind], " Test Set acc:", test_acc[ind])
```

Figure 5:




```
Training Set accuracy 0.9016393442622951
Test Set accuracy 0.760827155676941
```

Figure 6:




```
Training Set accuracy for k = 11: 0.6597384855581577
Test Set accuracy for k = 11: 0.5704252828716349
```

Figure 7:

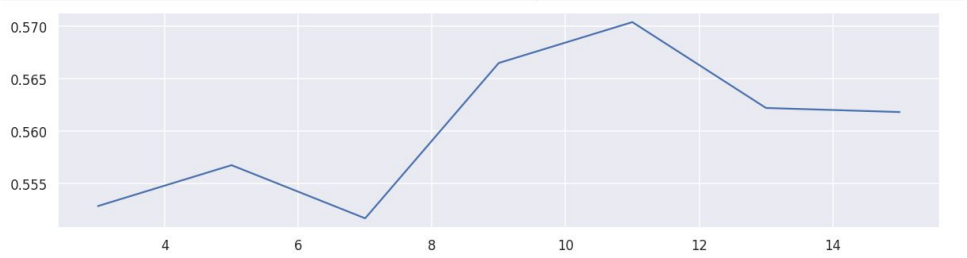
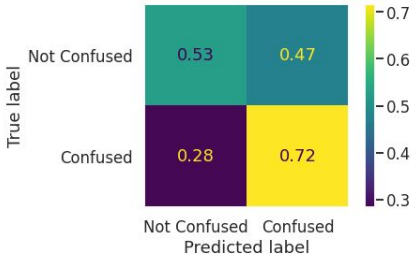


Figure 8:

```
# create and train the Random Forests model
# also trained with n_estimators 200, 500, and 2000, but 1000 was the optimal value
rf_model = RandomForestClassifier(n_estimators=1000, max_depth=2, random_state=13)
rf_model.fit(X_train,y_train)
```

Figure 9:




```
Training Set accuracy: 0.6052888368462139
Test Set accuracy: 0.6215372610222396
```

Figure 10:

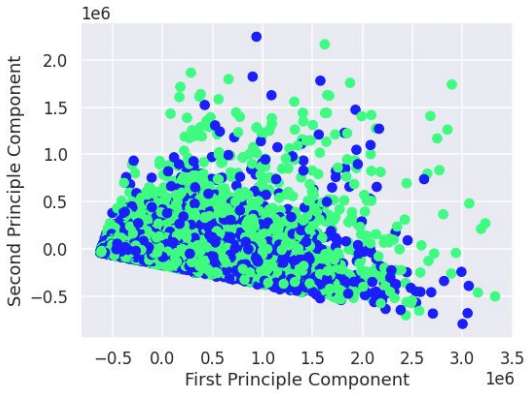


Figure 11:

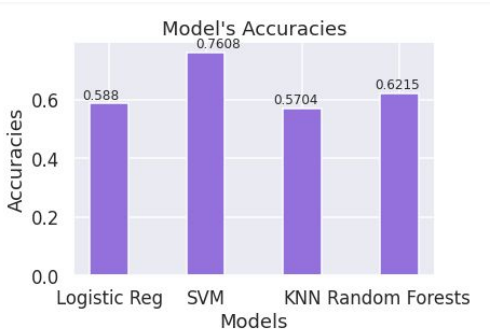


Figure 12:

Figure 13:

```
# train model on only one subject's data
df_subject1 = df.loc[df['SubjectID'].isin(range(0,1))]
df_subject1.shape
```

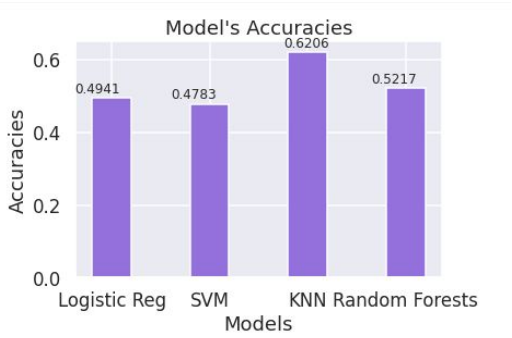
```
(1261, 16)
```

```
# 80/20 train test split -- 1008 train, 253 test
df_train = df_subject1.iloc[:1008, :]
print("Train data shape:", df_train.shape)

df_test = df_subject1.iloc[1008:, :]
print("Test data shape:", df_test.shape)
```

```
Train data shape: (1008, 16)
Test data shape: (253, 16)
```

```
# split the data into training and testing sets
X_train = df_train[features]
y_train = df_train["Label"]

X_test = df_test[features]
y_test = df_test["Label"]
```

Figure 14:



Works Cited

Borges, Niklas, et al. "Classifying Confusion: Autodetection of Communicative Misunderstandings Using Facial Action Units." *2019 8th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW)*, 2019, doi:10.1109/aciiw.2019.8925037.

Graimann, Bernhard, et al. *Brain–Computer Interfaces: A Gentle Introduction*. 2009, link.springer.com/book/10.1007/978-3-642-02091-9.

Nicolas-Alonso, Luis Fernando, and Jaime Gomez-Gil. "Brain computer interfaces, a review." *Sensors (Basel, Switzerland)* vol. 12,2 (2012): 1211-79. doi:10.3390/s120201211

Ni, Zhaoheng et al. "Confused or not Confused?: Disentangling Brain Activity from EEG Data Using Bidirectional LSTM Recurrent Neural Networks." *ACM-BCB ... ... : the ... ACM Conference on Bioinformatics, Computational Biology and Biomedicine. ACM Conference on Bioinformatics, Computational Biology and Biomedicine* vol. 2017 (2017): 241-246. doi:10.1145/3107411.3107513

Wang, H. (2018). Confused student EEG brainwave data, Version 6. Retrieved February 27, 2021 from https://www.kaggle.com/wanghaohan/confused-eeg.

Final Project Notebook Github link:https://github.com/ashnasood/Confusion-EEG-Data-Analysis