ADHD Classification Significance using Neural Networks

Varaprasad Rao Kurra
vkurra1@student.gsu.edu

Yesaswini Kandru
ykandru1@student.gsu.edu

***Abstract*—The aim is to do a systematic study on the neural networks and find an optimal neural network that will give us the best accuracy in classifying the ADHD rs-fMRI data. We obtain the dataset from the Nilearn learn package. Nilearn is a Python module for fast and easy statistical learning on neuroimaging data. Our work involves practical steps of using machine learning to analyze rs-fMRI data and, more specifically to discriminate Attention Deficit Hyperactivity Disorder (ADHD) from healthy controls. We will discuss (1) feature extraction using masks (2) The benefits and drawbacks of recurrent neural networks (RNN) and particularly long short-term memory networks (LSTM) for classifying fMRI data (3) hypothesis testing and its use in model evaluation.**

Keywords – LSTM, ADHD, Masking, ICA, Neural Networks

Introduction:

Conventional Machine Learning models neglect to comprehend the examples in the fMRI information. We need AI models that will actually want to gain from the information. Distinguish the examples with the goal that the precision of the model can be expanded during the preparation stage itself. What is acceptable about the neural networks is that we have two stages to gain from the information: forward proliferation and in reverse spread. The neural networks learn and change the neural organization's loads and help get the outcomes with sensible precision. Our trial includes the Data Preparation state, where we veil our information to take out the undesirable highlights. Then, we train our neural organizations and contrast the models' exactness with realize which model is the best fit for the fMRI information.

I. Data Preparation

A. Feature Extraction

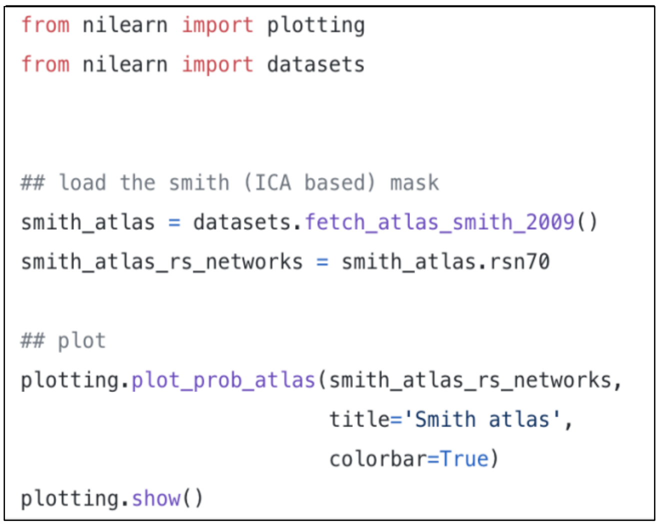
FMRI pictures are 4D lattices mirroring the actuation level of each voxel at the three-dimensional reality. Nonetheless, frequently, the important data is depicted by a subset of this information. For instance, here-we are just interested about the resting-state organizations. To exclude the unimportant information, we apply covers. Covers are basically channels that pass the ideal subset of the information while dropping the rest. For all intents and purposes, covers supplant the initiation worth of undesirable voxels to 0.

There are numerous styles to cover fMRI information, which is for the most part controlled by the investigation's objective. This instructional exercise centers around arranging ADHD patients from controls through their resting-state organizations. Hence, we apply Smith's rs-fMRI segments chart book (Smith et al.,2009). Smith map book reflects seventy resting-state organizations (RSN) obtained utilizing a free segment examination (ICA) on great many sound patients. We incline toward Smith map book over the utilization of dataset-explicit ICA segments since it abstains from going in for seconds (i.e., utilizing the information twice), prompting overfitting.

B. Feature Extraction Implementation

The layout is utilized to design the paper and style the content. All edges, section widths, line spaces, and text textual styles are recommended; kindly don't change them. You may note eccentricities. For instance, the head edge in this layout gauges proportionately more than is standard. This estimation and others are intentional, utilizing determinations that expect your paper as one piece of the whole procedures and not as an autonomous archive. Kindly don't overhaul any of the current assignments.

The Smith's atlas is available via Ni-Learn dataset:

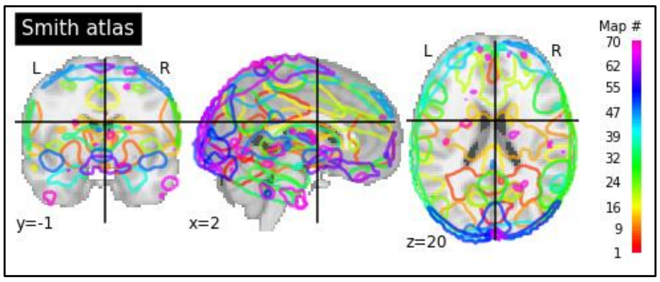


```
from nilearn import plotting
from nilearn import datasets


## load the smith (ICA based) mask
smith_atlas = datasets.fetch_atlas_smith_2009()
smith_atlas_rs_networks = smith_atlas.rsn70

## plot
plotting.plot_prob_atlas(smith_atlas_rs_networks,
                         title='Smith atlas',
                         colorbar=True)
plotting.show()
```

The code snippet gives us a resting state network of 70 regions. We have three options in choosing the number of networks. Firstly, we can select 10 regions at a time. Secondly, we can select the top 20 regions and 70 regions of the resting state network. To increase the accuracy and get a good picture of the data, we select 70 rs-networks as a masker to drop the unnecessary features.

In the underneath picture, we can see the three perspectives on the picture that we will use to extricate the highlights from the given fMRI examine. We can see the guide close to the pictures that give an image of the quantity of resting-state organizations. The ADHD dataset is likewise accessible on NiLearn. We see that the picture contains 73*61*61 voxels more than 176 timestamps from the principal picture's header.

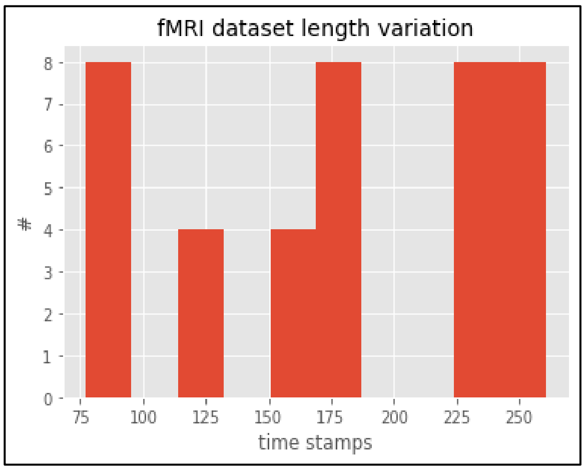
Additionally, each voxel is about 3mm³. It is important to note that this information may not be uniform across the whole dataset.



Why masking. Applying a standardization could contribute to the features' robustness. In masking, it can help enhance the signal by centering and normalizing the slices for each time-series. Considering the data confounds as part of the transformation process also enhances the signal by removing confounding noise.

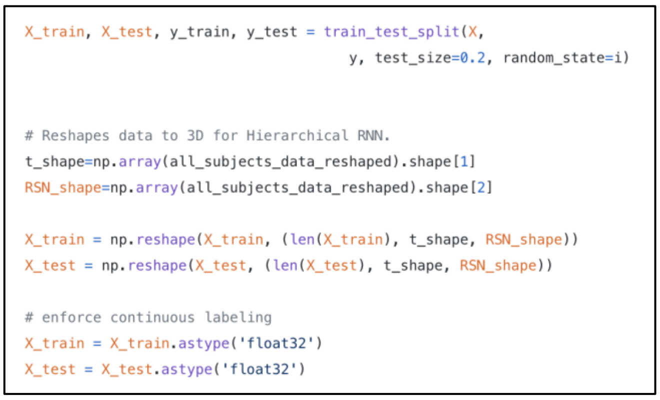
C. *Insights of the Data*

As referenced previously, a dataset probably won't hold a homogenous examining length - these 40 subjects present a serious extensive variety. In any case, most AI calculations (Keras included) require a uniform shape across all subjects. To advance for keeping information, we can utilize cushioning; attach each subject with zeros after the finish of its output to coordinate with the longest sweep's length. As well as cushioning, we reshape the information to accommodate Keras' necessities; (40,261,10) suggests that we have 40 subjects with 261 timestamps long example more than 10 areas.



D. *Data Preparation*

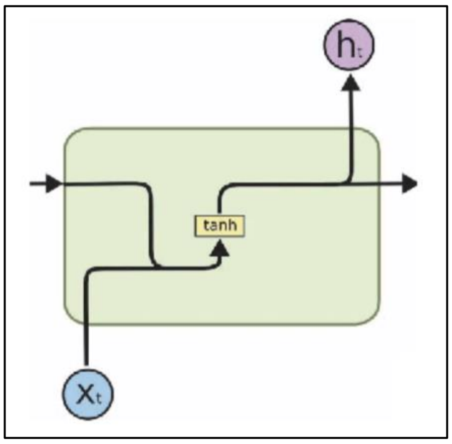
We utilize the train/test split worldview to ensure the model is tried on altogether new information. The capacity underneath arbitrarily parts the information into train and test and reshapes each part as per the model's necessities. We will see the code bit about how we played out the train test split in the impending advances. How could we reshape the information?



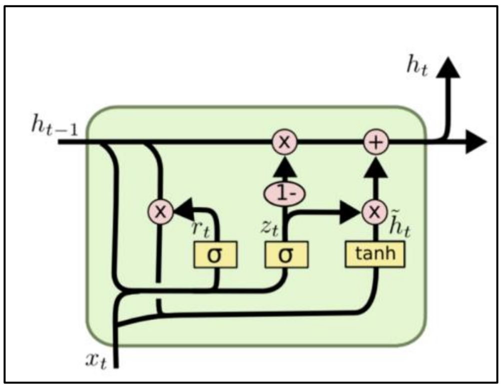
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                        y, test_size=0.2, random_state=i)


# Reshapes data to 3D for Hierarchical RNN.
t_shape=np.array(all_subjects_data_reshaped).shape[1]
RSN_shape=np.array(all_subjects_data_reshaped).shape[2]

X_train = np.reshape(X_train, (len(X_train), t_shape, RSN_shape))
X_test = np.reshape(X_test, (len(X_test), t_shape, RSN_shape))

# enforce continuous labeling
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

II. Neural Network Models

Simple RNN: Here, there is a simple multiplication of Input (Xt) and Previous Output (ht-1). Passed through Tanh activation function. No Gates present.



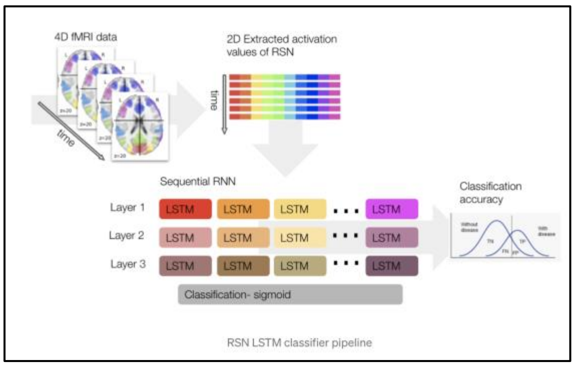
Gated Recurrent Unit (GRU): Here, an **Update gate** is introduced to decide whether to pass Previous O/P (ht-1)to the next Cell (as ht) or not. Forget gate is nothing but additional Mathematical Operations with a new set of Weights (Wt).



Long Short Term Memory Unit (LSTM):- Here, 2 additional Gates are presented (Forget and Output) notwithstanding the Update entryway of GRU. What's more, once more, as over, these are extra Mathematical Operations on similar sources of info (Xt and ht-1). So generally, LSTM has presented 2 Math tasks having 2 new arrangements of Weights.

Long transient memory (LSTM) models additionally give a few advantages in learning fMRI information. The primary explanation is that, not normal for most AI or profound learning strategies, they figure out how to keep the logical data of the information sources — in this manner fuse subtleties from past pieces of the information arrangement while handling a current one. All things considered, being profoundly relevant isn't in every case something worth being thankful for. There are situations where LSTMs are not the most ideal decision; being logical could prompt an over-translation of the information. Moreover, LSTMs could take more time to run than a straightforward NN, and they may have undeniably more boundaries to tune.

fMRI data represents dynamic brain activity over time, thus using LSTMs enables taking advantage of the temporal information (that otherwise would have been lost) when analyzing functional connectivity.



The above pipeline shows our step-by-step approach to find our experiment. Firstly, we get the raw ready to go data from the nilearn defined package. Next, we will transform these 4D matrices into meaningful data, i.e., 2D extracted RSN values.

A common enhancement for LSTM is convolutional neural networks (CNN), which supports analyzing the spatial structure. However, here, we extracted seventy discrete values reflecting independent components of the activation of whole networks — and thus gave up on the spatial property of the data. Therefore, CNN would be unlikely to be useful.

An issue with LSTMs is that they can undoubtedly overfit preparing information, diminishing their prescient limit. An answer for this issue is through regularization, which upsets the model's overfitting inclination. A traditional regularization in LSTM networks is the dropout, which probabilistically prohibits units from the layer association. There are two sorts of dropouts-input dropout and intermittent dropout. A dropout on the info implies that for a given likelihood, the information on the information association with each LSTM unit will be avoided from hub actuation and weight refreshes (see the dropout contention). The dropout on the intermittent info acts a similar path however on the repetitive association (see the reccurent_dropout contention). In any case, it is significant not to over-regularize as it will frustrate the model from realizing (which could be recognized by exacting non-demonstrative expectation).

The model we present here is a sequential model with three stacked LSTM layers and one dense layer with sigmoid activation. To be frank, there is no one right answer to how one chooses their hyperparameters. There is a lot of trial and error and rules of thumb. We present a few that we have collected.

- Generally, we might want to have at any rate two secret layers (excluding the last layer) since the force of NNs originates from their profundity (i.e., a zero-layer can just address direct capacities.).

- We might want to begin with various units more modest or equivalent to the info size and diminishing it (by about a half at that point) until arriving at the last layer.

- In the instance of characterization, the quantity of units at the yield layer ought to be equivalent to the quantity of classes. Normally, a paired grouping issue has one yield.

- The yield layer initiation is generally sigmoid for twofold grouping, SoftMax for a multi-class classifier and direct for relapse.

- The characterization misfortune capacity ought to be coordinated to the quantity of names and their sort.

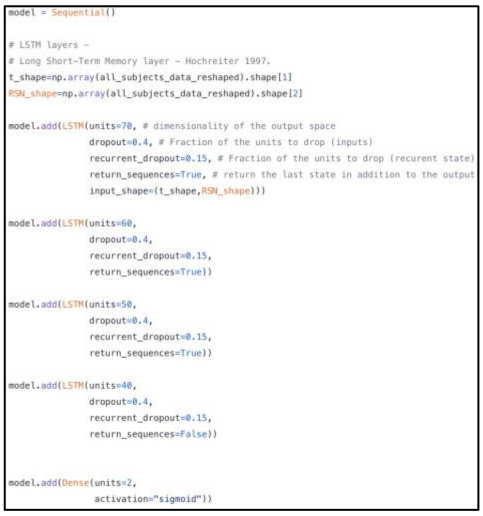
- The precision metric is superb to show the percent of right groupings. You can utilize mutiple!

- The more ages, the better; start with 30 and follow your approval set to diminish misfortune and improve exactness. On the off chance that you see improvement, increment the quantity of ages; in any case, return to the planning phase.

- Choosing the analyzer dependent on your insight into your information's properties and the profundity of the organization.

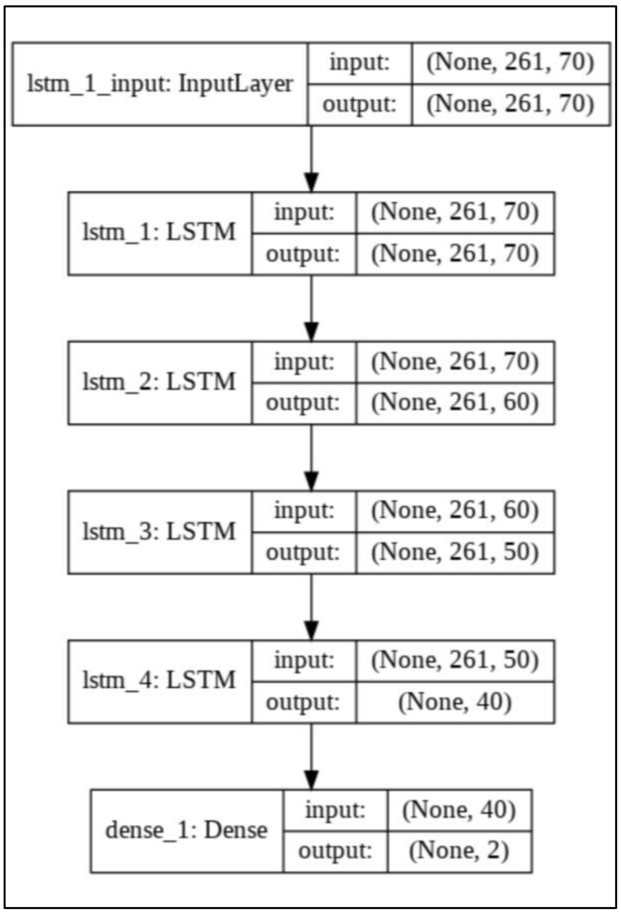
LSTM Model

The code snippet to build the LSTM model is as follows:



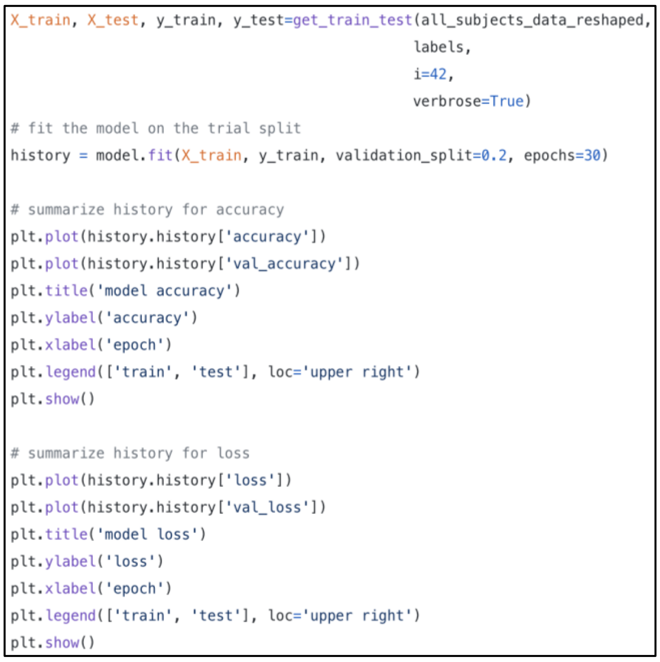
```
model = Sequential()

# LSTM layers -
# Long Short-Term Memory layer - Hochreiter 1997.
t_shape=np.array(all_subjects_data_reshaped).shape[1]
RSN_shape=np.array(all_subjects_data_reshaped).shape[2]

model.add(LSTM(units=70, # dimensionality of the output space
               dropout=0.4, # Fraction of the units to drop (inputs)
               recurrent_dropout=0.15, # Fraction of the units to drop (recurent state)
               return_sequences=True, # return the last state in addition to the output
               input_shape=(t_shape,RSN_shape)))

model.add(LSTM(units=60,
               dropout=0.4,
               recurrent_dropout=0.15,
               return_sequences=True))

model.add(LSTM(units=50,
               dropout=0.4,
               recurrent_dropout=0.15,
               return_sequences=True))

model.add(LSTM(units=40,
               dropout=0.4,
               recurrent_dropout=0.15,
               return_sequences=False))


model.add(Dense(units=2,
                activation="sigmoid"))
```

LSTM model which we have built looks as shown in the below screenshot.



III. Training the Model

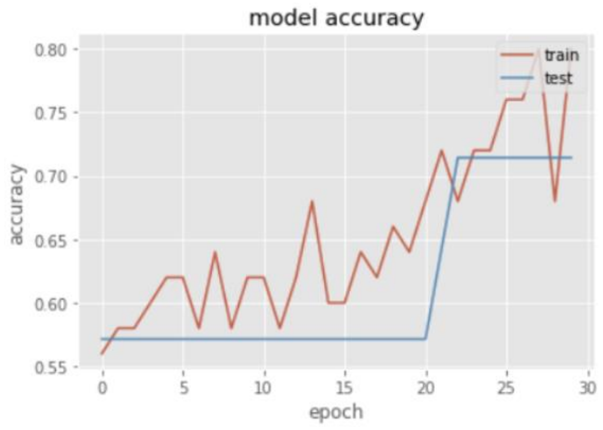
After building the model, we train the model and see if it manages to learn. To train the model, we split the dataset by using epochs.

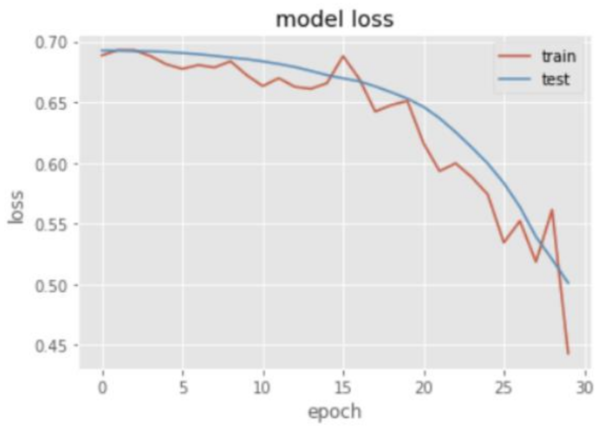


```
X_train, X_test, y_train, y_test=get_train_test(all_subjects_data_reshaped,
                                               labels,
                                               i=42,
                                               verbrose=True)
# fit the model on the trial split
history = model.fit(X_train, y_train, validation_split=0.2, epochs=30)

# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```

Here, one epoch is when an entire dataset is passed forward and backward through the neural network model only once. Since one epoch is too big to feed to the computer at once, we divide it into several batches. Accuracy is the number of correctly predicted data points out of all the data points, and loss indicates how bad the model's prediction is. If the model’s prediction is perfect, the loss is zero; otherwise, the loss is more significant.

The above code snippet is to train the model on the split dataset, and the graph shows the results of accuracy and loss. From the graph results, we see some increase in the model’s accuracy and some decrease in the model’s loss as the number of epochs increases —additionally, both the training and validation sets showing similar trends. Thus, we are (1) having a working model and (2) probably not overfitting.





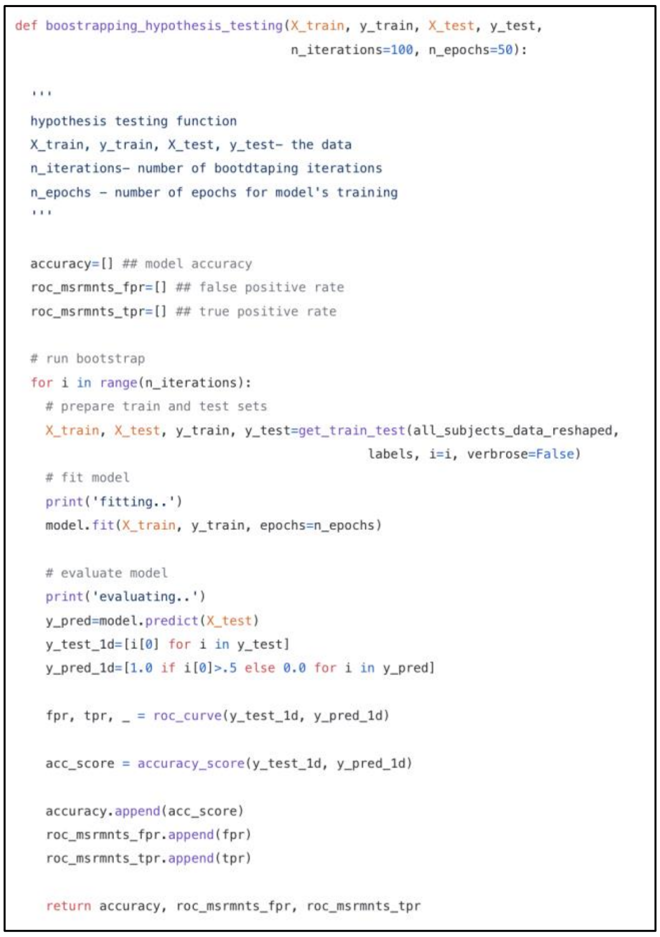
Hypothesis Testing

We use a hypothesis testing framework to evaluate the model since it helps obtain both the model's accuracy and its significance. To calculate the model’s significance, we use bootstrapping. Bootstrapping is a computer-based method for statistical inference without relying on too many assumptions. It helps us approximate the properties of the population by estimates from small data samples.

Our null hypothesis, to be tested, states that there are no differences between ADHD patients to controls; thus, the accuracy of comparing the two using our model should be about 0.5.

To evaluate this hypothesis, we will repeatedly resample with replacement from the data, splitting it into a train and test parts, fitting the model, predicting the labels of test data, and calculating the accuracy. Such repetition will create a distribution of accuracies. Such resampling that follows the central limit theorem is likely to approach a Gaussian shape the larger the number of iterations we use. Thus, we could employ statistical tests that require normal distribution, like t-test.

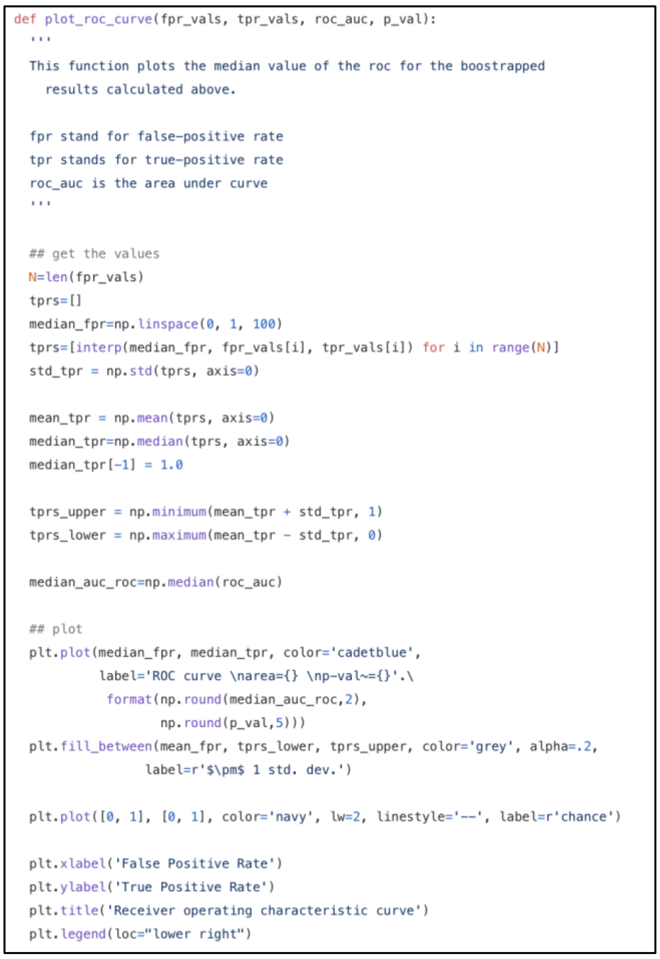
If 0.5 is placed at the distribution tails, we can conclude that it is very unlikely that the distribution is centered around 0.5 and reject the null hypothesis. Otherwise, we would fail to reject it. The next code snippet runs the bootstrapped experiment.



```
def boostrapping_hypothesis_testing(X_train, y_train, X_test, y_test,
                                    n_iterations=100, n_epochs=50):

  '''
  hypothesis testing function
  X_train, y_train, X_test, y_test- the data
  n_iterations- number of bootdtaping iterations
  n_epochs - number of epochs for model's training
  '''

  accuracy=[] ## model accuracy
  roc_msrmnts_fpr=[] ## false positive rate
  roc_msrmnts_tpr=[] ## true positive rate

  # run bootstrap
  for i in range(n_iterations):
    # prepare train and test sets
    X_train, X_test, y_train, y_test=get_train_test(all_subjects_data_reshaped,
                                                    labels, i=i, verbrose=False)
    # fit model
    print('fitting..')
    model.fit(X_train, y_train, epochs=n_epochs)

    # evaluate model
    print('evaluating..')
    y_pred=model.predict(X_test)
    y_test_1d=[i[0] for i in y_test]
    y_pred_1d=[1.0 if i[0]>.5 else 0.0 for i in y_pred]

    fpr, tpr, _ = roc_curve(y_test_1d, y_pred_1d)

    acc_score = accuracy_score(y_test_1d, y_pred_1d)

    accuracy.append(acc_score)
    roc_msrmnts_fpr.append(fpr)
    roc_msrmnts_tpr.append(tpr)

    return accuracy, roc_msrmnts_fpr, roc_msrmnts_tpr
```

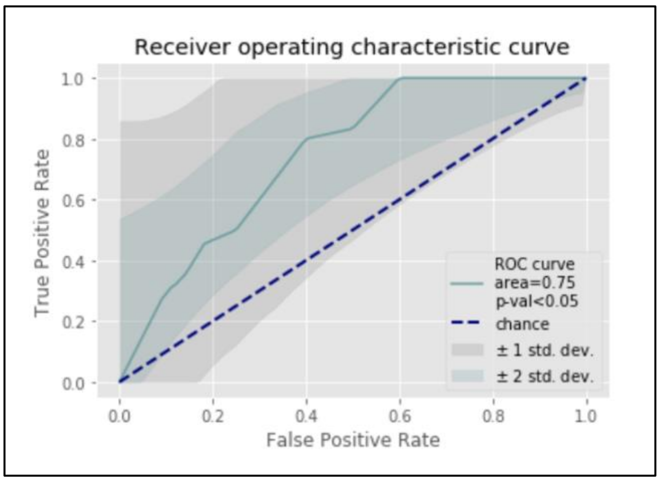
In conclusion, we introduced the outcomes utilizing the Receiver working trademark (ROC) bend. The ROC bend mirrors the model's affectability and explicitness by plotting the genuine positive rate against the bogus positive rate. We decided to introduce the ROC bend's middle worth since it presents a more strong estimation on the off chance that the information is slanted.

Here, Sensitivity is a proportion of the extent of real sure cases that got anticipated as sure, and it is additionally called Recall. Particularity is a proportion of the extent of individuals not experiencing the illness who got anticipated accurately as the individuals who are not experiencing the infection. The beneath code bit plots the ROC bend for the bootstrapped results determined previously.



```
def plot_roc_curve(fpr_vals, tpr_vals, roc_auc, p_val):
  '''
  This function plots the median value of the roc for the boostrapped
    results calculated above.

  fpr stand for false-positive rate
  tpr stands for true-positive rate
  roc_auc is the area under curve
  '''

  ## get the values
  N=len(fpr_vals)
  tprs=[]
  median_fpr=np.linspace(0, 1, 100)
  tprs=[interp(median_fpr, fpr_vals[i], tpr_vals[i]) for i in range(N)]
  std_tpr = np.std(tprs, axis=0)

  mean_tpr = np.mean(tprs, axis=0)
  median_tpr=np.median(tprs, axis=0)
  median_tpr[-1] = 1.0

  tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
  tprs_lower = np.maximum(mean_tpr - std_tpr, 0)

  median_auc_roc=np.median(roc_auc)

  ## plot
  plt.plot(median_fpr, median_tpr, color='cadetblue',
           label='ROC curve \narea={} \np-val~={}'.\
            format(np.round(median_auc_roc,2),
                   np.round(p_val,5)))
  plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                   label=r'$\pm$ 1 std. dev.')

  plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label=r'chance')

  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver operating characteristic curve')
  plt.legend(loc="lower right")
```

The below ROC curve shows a significant classification. We see that the median value and one SD around it is entirely placed above the chance diagonal. The 95% confidence intervals presented by 2 SD around the ROC curve expands mostly at the bottom left point. This concludes that the model is likely to express higher sensitivity but lower specificity.



The model's affectability is the extent of patients recognized effectively to have the infection upon the all out number of patients with the illness. The model's explicitness portrays the extent of patients distinguished effectively to not have the illness upon the all out number of patients who don't have the infection. Normally, these two presentation a converse connection. Notwithstanding, it is basic in demonstrative tests to support affectability hence not missing any evil patients, and such a choice is exceptionally identified with the idea of the speculation.

IV. Conclusion

We have built an LSTM model to classify ADHD patients from healthy controls via rs-fMRI. We have discussed hypothesis testing and demonstrated its benefits in a diagnostic experiment. Our Future work includes comparing the neural work models in classifying the ADHD rs-fMRI data.

References

[1] Borlase,N., Melzer, T.R., Eggleston, M.J., Darling, K.A., & Rucklidge, J.J, “Resting-state networks and neurometabolites in children with ADHD after 10 weeks of treatment with micronutrients: results of a randomised placebo-controlled trial, 2019.

[2] Dvornek, N.C., Ventola, P., Pelphery, K.A., & Duncan, J.S.(2017,September). Identifying austism from resting state fMRI using long short-term memory networks. In International Workshop on Machine Learning in Medical Imaging(pp. 362-370) . Springer, Cham.

[3] Parikh, R., Mathai, A., Parikh, S., Sekhar, G. C., & Thomas, R. (2008). Understanding and using sensitivity, specificity and predictive values. Indian journal of ophthalmology.

[4] https://nilearn.github.io/modules/generated/nilearn.datasets.fetch_adhd.html