ADHD Classification Significance using Neural Networks

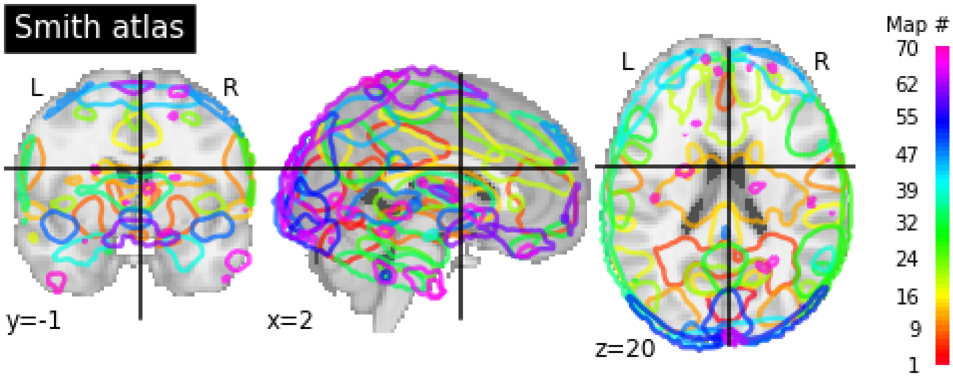
```
from nilearn import datasets
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
plt.style.use('ggplot')


from nilearn.input_data import NiftiMapsMasker
from nilearn import plotting

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn import metrics

from keras.models import Model, Sequential
from keras.layers import Input, Dense
from keras.layers import LSTM
from keras import optimizers
from keras.utils import plot_model
from keras import utils
from sklearn.metrics import roc_curve


from scipy.stats import ttest_1samp
from scipy import interp
```

Data Preperation

Getting the Masker first.

```
smith_atlas = datasets.fetch_atlas_smith_2009()
smith_atlas_rs_networks = smith_atlas.rsn70
```

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



ADHD Dataset

```
adhd_data=datasets.fetch_adhd(n_subjects=100)
```

```
Downloaded 67925261 of 67925261 bytes (100.0%,    0.0s remaining) ...done. (71 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7803/adhd40_3154996
Downloaded 32919780 of 32919780 bytes (100.0%,    0.0s remaining) ...done. (31 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7804/adhd40_3205761
Downloaded 59835286 of 59835286 bytes (100.0%,    0.0s remaining) ...done. (57 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7805/adhd40_3520880
Downloaded 61857076 of 61857076 bytes (100.0%,    0.0s remaining) ...done. (80 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7806/adhd40_3624598
Downloaded 59385206 of 59385206 bytes (100.0%,    0.0s remaining) ...done. (46 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7807/adhd40_3699991
Downloaded 41518251 of 41518251 bytes (100.0%,    0.0s remaining) ...done. (50 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7808/adhd40_3884955
Downloaded 32108848 of 32108848 bytes (100.0%,    0.0s remaining) ...done. (25 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7809/adhd40_3902469
Downloaded 39415752 of 39415752 bytes (100.0%,    0.0s remaining) ...done. (32 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7810/adhd40_3994098
Downloaded 59297020 of 59297020 bytes (100.0%,    0.0s remaining) ...done. (42 s

Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7811/adhd40_4016887
Downloaded 20157314 of 20157314 bytes (100.0%,    0.0s remaining) ...done. (22 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7812/adhd40_4046678
Downloaded 21375806 of 21375806 bytes (100.0%,    0.0s remaining) ...done. (24 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7813/adhd40_4134561
Downloaded 69914913 of 69914913 bytes (100.0%,    0.0s remaining) ...done. (51 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
```

```
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7814/adhd40_4164316
Downloaded 45506732 of 45506732 bytes (100.0%,    0.0s remaining) ...done. (30 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7815/adhd40_4275075
Downloaded 32363673 of 32363673 bytes (100.0%,    0.0s remaining) ...done. (24 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7816/adhd40_6115230
Downloaded 73484949 of 73484949 bytes (100.0%,    0.0s remaining) ...done. (87 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7817/adhd40_7774305
Downloaded 42188959 of 42188959 bytes (100.0%,    0.0s remaining) ...done. (46 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7818/adhd40_8409791
Downloaded 70396354 of 70396354 bytes (100.0%,    0.0s remaining) ...done. (55 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7819/adhd40_8697774
Downloaded 45075978 of 45075978 bytes (100.0%,    0.0s remaining) ...done. (52 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7820/adhd40_9744150
Downloaded 63380505 of 63380505 bytes (100.0%,    0.0s remaining) ...done. (53 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
Downloading data from https://www.nitrc.org/frs/download.php/7821/adhd40_9750701
Downloaded 46607053 of 46607053 bytes (100.0%,    0.0s remaining) ...done. (46 s
Extracting data from /Users/gilikarni/nilearn_data/adhd/166bfb3ae13f7c60c012aa21
```

The dimensions of the first adhd_data['func'][0] image

```
import nibabel as nib
img=nib.load(adhd_data['func'][0])

print(img.header['dim'])
print(img.header['pixdim'])
```

```
[  4  61  73  61 176   1   1   1]
[-1.  3.  3.  3.  2.  0.  0.  0.]
```

```
# masker

masker = NiftiMapsMasker(maps_img=smith_atlas_rs_networks,  # Smith stals
                         standardize=True, # centers and norms the time-series
                         memory='nilearn_cache', # cache
                         verbose=0) #do not print verbose
```

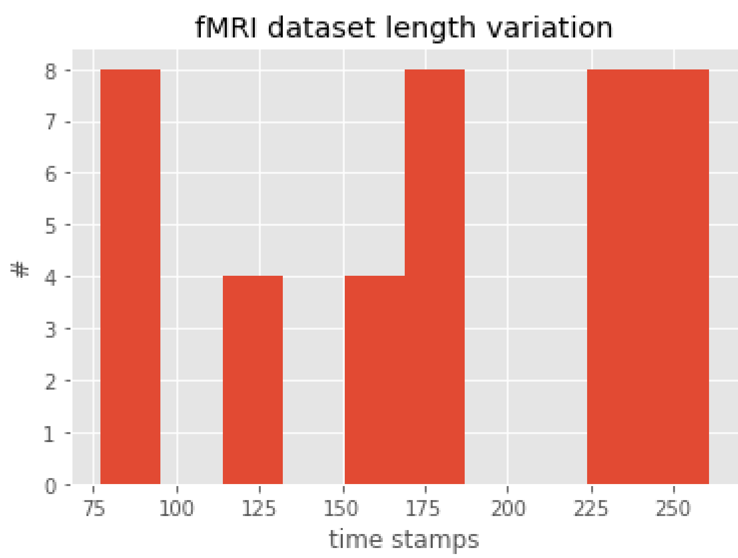
```
all_subjects_data=[]
labels=[]  # 1 if ADHD, 0 if control

for func_file, confound_file, phenotypic in zip(
        adhd_data.func, adhd_data.confounds, adhd_data.phenotypic):

    time_series = masker.fit_transform(func_file, confounds=confound_file)
```

```
    all_subjects_data.append(time_series)
    labels.append(phenotypic['adhd'])
```

```
print('N control:' ,labels.count(0))
print('N adhd:' ,labels.count(1))
```

```
N control: 20
N adhd: 20
```

```
plt.hist([len(i) for i in all_subjects_data])
plt.title('fMRI dataset length variation')
plt.xlabel('time stamps')
plt.ylabel('Count')
plt.show()
```



Finding the longest image in the obtained data

```
max_len_image=np.max([len(i) for i in all_subjects_data])
```

reshaping the data into uniform shape

```
all_subjects_data_reshaped=[]
for subject_data in all_subjects_data:
  # Padding
  N= max_len_image-len(subject_data)
  padded_array=np.pad(subject_data, ((0, N), (0,0)),
                      'constant', constant_values=(0))
  subject_data=padded_array
```

```
  subject_data=np.array(subject_data)
  subject_data.reshape(subject_data.shape[0],subject_data.shape[1],1)
  all_subjects_data_reshaped.append(subject_data)
```

shape of data

40 subjects 261 time stamps 10 netwroks values

```
np.array(all_subjects_data_reshaped).shape
```

```
(40, 261, 70)
```

```
# The data, split between train and test sets.

def get_train_test(X, y, i, verbrose=False):
  X_train, X_test, y_train, y_test = train_test_split(X,
                                        y, test_size=0.2, random_state=i)

  # Reshapes data to 4D 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))

  X_train = X_train.astype('float32')
  X_test = X_test.astype('float32')

  if verbrose:
    print(X_train.shape[0], 'train samples')
    print(X_test.shape[0], 'test samples')

  # Converts class vectors to binary class matrices.
  y_train = utils.to_categorical(y_train, 2)
  y_test = utils.to_categorical(y_test, 2)

  return X_train, X_test, y_train, y_test
```

```
32 train samples
8 test samples
```

Build the LSTM model

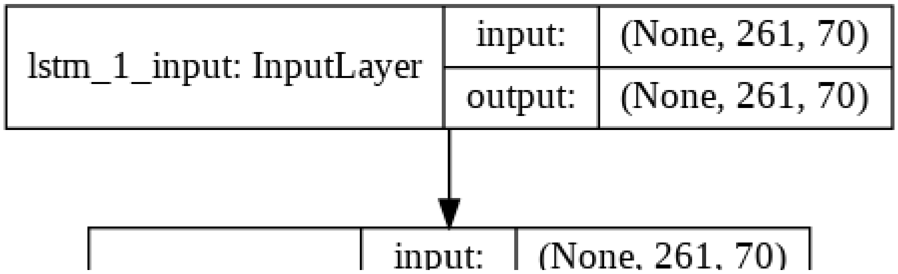
```
# create the model
```

```
# Create the model

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
               dropout=0.4, # drop (inputs)
               recurrent_dropout=0.15, # drop (recurent state)
               return_sequences=True, # return the last state
               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"))

model.compile(loss='binary_crossentropy',
                optimizer=optimizers.Adam(lr=0.001),
                metrics=['binary_accuracy'])

print(model.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tens

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tens

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tens

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tens
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tens

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/py
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 261, 70)           39480     
_________________________________________________________________
lstm_2 (LSTM)                (None, 261, 60)           31440     
_________________________________________________________________
lstm_3 (LSTM)                (None, 261, 50)           22200     
_________________________________________________________________
lstm_4 (LSTM)                (None, 40)                14560     
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 82        
=================================================================
Total params: 107,762
Trainable params: 107,762
Non-trainable params: 0
_________________________________________________________________
None
```

```
plot_model(model, show_shapes=True, show_layer_names=True)
```

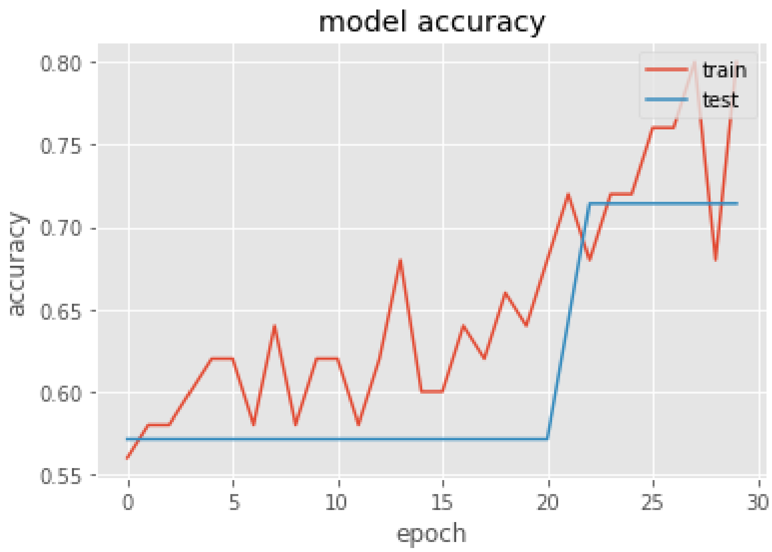


Train the LSTM model

```
X_train, X_test, y_train, y_test = get_train_test(all_subjects_data_reshaped,
                                                   labels, i=8, verbrose=True)

history = model.fit(X_train, y_train, validation_split=0.2, epochs=30)

# summarize history for accuracy
plt.plot(history.history['binary_accuracy'])
plt.plot(history.history['val_binary_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()
```

```
32 train samples
8 test samples
Train on 25 samples, validate on 7 samples
Epoch 1/30
25/25 [==============================] - 2s 100ms/step - loss: 0.6885 - binary_a
Epoch 2/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6929 - binary_ac
Epoch 3/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6929 - binary_ac
Epoch 4/30
25/25 [==============================] - 1s 23ms/step - loss: 0.6881 - binary_ac
Epoch 5/30
25/25 [==============================] - 1s 25ms/step - loss: 0.6813 - binary_ac
Epoch 6/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6774 - binary_ac
Epoch 7/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6806 - binary_ac
Epoch 8/30
25/25 [==============================] - 1s 23ms/step - loss: 0.6786 - binary_ac
Epoch 9/30
25/25 [==============================] - 1s 23ms/step - loss: 0.6837 - binary_ac
Epoch 10/30
25/25 [==============================] - 1s 23ms/step - loss: 0.6724 - binary_ac
Epoch 11/30
25/25 [==============================] - 1s 25ms/step - loss: 0.6632 - binary_ac
Epoch 12/30
25/25 [==============================] - 1s 29ms/step - loss: 0.6698 - binary_ac
Epoch 13/30
25/25 [==============================] - 1s 23ms/step - loss: 0.6626 - binary_ac
Epoch 14/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6610 - binary_ac
Epoch 15/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6657 - binary_ac
Epoch 16/30
25/25 [==============================] - 1s 27ms/step - loss: 0.6879 - binary_ac
Epoch 17/30
25/25 [==============================] - 1s 24ms/step - loss: 0.6694 - binary_ac
Epoch 18/30
25/25 [==============================] - 1s 26ms/step - loss: 0.6423 - binary_ac
Epoch 19/30
25/25 [==============================] - 1s 29ms/step - loss: 0.6476 - binary_ac
Epoch 20/30
25/25 [==============================] - 1s 29ms/step - loss: 0.6510 - binary_ac
Epoch 21/30
25/25 [==============================] - 1s 27ms/step - loss: 0.6163 - binary_ac
Epoch 22/30
25/25 [==============================] - 1s 26ms/step - loss: 0.5932 - binary_ac
Epoch 23/30
25/25 [==============================] - 1s 25ms/step - loss: 0.5996 - binary_ac
Epoch 24/30
25/25 [==============================] - 1s 29ms/step - loss: 0.5883 - binary_ac
Epoch 25/30
25/25 [==============================] - 1s 28ms/step - loss: 0.5739 - binary_ac
Epoch 26/30
25/25 [==============================] - 1s 26ms/step - loss: 0.5343 - binary_ac
Epoch 27/30
25/25 [==============================] - 1s 26ms/step - loss: 0.5521 - binary_ac
Epoch 28/30
```

```
Epoch 28/30
25/25 [==============================] - 1s 30ms/step - loss: 0.5184 - binary_ac
Epoch 29/30
25/25 [==============================] - 1s 29ms/step - loss: 0.5613 - binary_ac
Epoch 30/30
25/25 [==============================] - 1s 26ms/step - loss: 0.4429 - binary_ac
```



Evaluate the LSTM model


```
from sklearn.metrics import accuracy_score

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, validation_split=0.2, epochs=n_epochs)

    # evaluate model
    print('evaluating..')
```

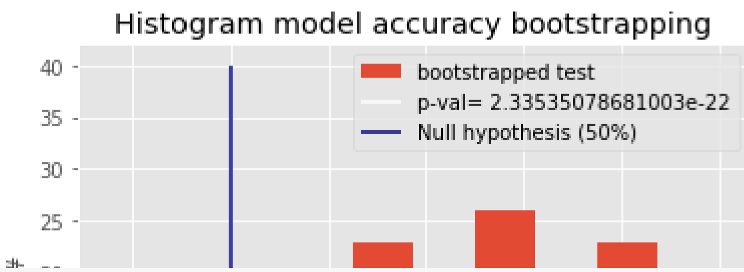
```
    y_pred=model.predict(X_test)
    y_test_1d=[i[0] for i in y_test]
    y_pred_1d=[1.0 if i[0]>.5 else 0.0 for i in y_pred]

    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



accuracy, roc_msrmnts_fpr, roc_msrmnts_tpr  = boostrapping_hypothesis_testing(X_train,
```

```
fitting..
Train on 25 samples, validate on 7 samples
Epoch 1/50
25/25 [==============================] - 1s 31ms/step - loss: 0.6721 - binary_ac
Epoch 2/50
25/25 [==============================] - 1s 27ms/step - loss: 0.6737 - binary_ac
Epoch 3/50
25/25 [==============================] - 1s 23ms/step - loss: 0.6482 - binary_ac
Epoch 4/50
25/25 [==============================] - 1s 30ms/step - loss: 0.6489 - binary_ac
Epoch 5/50
25/25 [==============================] - 1s 36ms/step - loss: 0.6650 - binary_ac
Epoch 6/50
25/25 [==============================] - 1s 22ms/step - loss: 0.6448 - binary_ac
Epoch 7/50
25/25 [==============================] - 1s 21ms/step - loss: 0.6397 - binary_ac
Epoch 8/50
25/25 [==============================] - 1s 32ms/step - loss: 0.6474 - binary_ac
Epoch 9/50
25/25 [==============================] - 1s 23ms/step - loss: 0.6426 - binary_ac
Epoch 10/50
25/25 [==============================] - 1s 22ms/step - loss: 0.6206 - binary_ac
Epoch 11/50
25/25 [==============================] - 1s 35ms/step - loss: 0.6298 - binary_ac
Epoch 12/50
25/25 [==============================] - 1s 20ms/step - loss: 0.6279 - binary_ac
Epoch 13/50
25/25 [==============================] - 1s 24ms/step - loss: 0.6534 - binary_ac
Epoch 14/50
25/25 [==============================] - 1s 26ms/step - loss: 0.6415 - binary_ac
Epoch 15/50
25/25 [==============================] - 1s 33ms/step - loss: 0.5858 - binary_ac
Epoch 16/50
25/25 [==============================] - 1s 29ms/step - loss: 0.6461 - binary_ac
Epoch 17/50
25/25 [==============================] - 1s 31ms/step - loss: 0.6134 - binary_ac
Epoch 18/50
```

```
25/25 [==============================] - 1s 30ms/step - loss: 0.5976 - binary_ac
Epoch 19/50
25/25 [==============================] - 1s 28ms/step - loss: 0.6028 - binary_ac
Epoch 20/50
25/25 [==============================] - 1s 25ms/step - loss: 0.6071 - binary_ac
Epoch 21/50
25/25 [==============================] - 1s 22ms/step - loss: 0.5996 - binary_ac
Epoch 22/50
25/25 [==============================] - 1s 28ms/step - loss: 0.5807 - binary_ac
Epoch 23/50
25/25 [==============================] - 1s 38ms/step - loss: 0.5752 - binary_ac
Epoch 24/50
25/25 [==============================] - 1s 26ms/step - loss: 0.5751 - binary_ac
Epoch 25/50
25/25 [==============================] - 1s 23ms/step - loss: 0.5740 - binary_ac
Epoch 26/50
25/25 [==============================] - 1s 20ms/step - loss: 0.6090 - binary_ac
Epoch 27/50
25/25 [==============================] - 0s 19ms/step - loss: 0.5622 - binary_ac
Epoch 28/50
25/25 [==============================] - 1s 22ms/step - loss: 0.5377 - binary_ac
Epoch 29/50
```

```
def calc_p_val(stats, h0, n_iterations):

  tset, pval = ttest_1samp(stats, h0)

  return pval

p_val=calc_p_val(accuracy, .5)
```

```
def plot_p_value(stats, p_val):

  plt.hist(stats, label='bootstrapped test')
  plt.vlines(.5, 0, 40, color='white', label='p-val= {}'.format(p_val))
  plt.vlines(.5, 0, 40, color='navy', label='Null hypothesis (50%)')

  plt.title('Histogram model accuracy bootstrapping')
  plt.xlabel('Model accuracy')
  plt.ylabel('#')
  plt.legend()
  plt.plot()

plot_p_value(accuracy, p_val)
```




```
def plot_roc_curve(fpr_vals, tpr_vals, roc_auc, p_val):

  ## 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_2 = np.minimum(mean_tpr + 2*std_tpr, 1)
  tprs_lower_2 = np.maximum(mean_tpr - 2*std_tpr, 0)

  tprs_upper_1 = np.minimum(mean_tpr + std_tpr, 1)
  tprs_lower_1 = np.maximum(mean_tpr - std_tpr, 0)

  median_auc_roc=np.median(roc_auc)


  ## plot
  if p_val<0.05:
    p_val=0.05
  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,2)))
  plt.fill_between(median_fpr, tprs_lower_2, tprs_upper_2, color='grey', alpha=.2,
                 label=r'$\pm$ 1 std. dev.')

  plt.fill_between(median_fpr, tprs_lower_1, tprs_upper_1, color='cadetblue', alpha=.2
                 label=r'$\pm$ 2 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")

  plt.show()
```

```
plot_roc_curve(roc_msrmnts_fpr, roc_msrmnts_tpr, accuracy,p_val)
```