After preprocessing the data we used support vector machine (SVM) and long-short term memory (LSTM) neural network to classify the non-seizure and seizure EEG signals.

For feature extraction in case of SVM model, we used discrete wavelet transform and statistical features are extracted from the wavelet coefficients in each sub-band.

Since the dataset is highly imbalanced in which majority of the samples are from non-seizure, we used SMOTE technique to balanced the dataset.

In [2]:

```
# we imported all the modules and pacakages
import glob
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import os
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
import seaborn as sn
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow.keras import Sequential
from keras.optimizers import SGD
from keras_tuner import RandomSearch

from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import LSTM
tf.keras.backend.clear_session()

import keras_tuner as kt
import warnings; warnings.simplefilter('ignore')
import warnings
warnings.filterwarnings('ignore')
```

Dataset A, B, C and D contain the datapoints of non-seizure and dataset E contains the datapoints of seizure. We labeled the non-seizure data as class '0' and seizure data as class'1'

In [3]:

```
 # Data Loading
# we used the path of final preprocessed dataset.Each dataset contains 100 files
# read all the files from sets and concates the files

# DataA
pathA = r'H:/Final_project/Data/merged/A'
all_filesA = glob.glob(pathA + "/*.csv")

tempA = []

for filename in all_filesA:
    df1 = pd.read_csv(filename, index_col=None, header=0)
```

```
    tempA.append(df1)

dfA = pd.concat(tempA, axis=0, ignore_index=True)
dfA.insert(loc=len(dfA.columns), column='Y', value='0', allow_duplicates=True)


# DataB
pathB = r'H:/Final_project/Data/merged/B'
all_filesB = glob.glob(pathB + "/*.csv")

tempB = []

for filename in all_filesB:
    df2 = pd.read_csv(filename, index_col=None, header=0)
    tempB.append(df2)

dfB = pd.concat(tempB, axis=0, ignore_index=True)
dfB.insert(loc=len(dfB.columns), column='Y', value='0', allow_duplicates=True)

# DataC
pathC = r'H:/Final_project/Data/merged/C'
all_filesC = glob.glob(pathC + "/*.csv")

tempC = []

for filename in all_filesC:
    df3 = pd.read_csv(filename, index_col=None, header=0)
    tempC.append(df3)

dfC = pd.concat(tempC, axis=0, ignore_index=True)
dfC.insert(loc=len(dfC.columns), column='Y', value='0', allow_duplicates=True)


## Data D
pathD = r'H:/Final_project/Data/merged/D'
all_filesD = glob.glob(pathD + "/*.csv")

tempD = []

for filename in all_filesD:
    df4 = pd.read_csv(filename, index_col=None, header=0)
    tempD.append(df4)

dfD = pd.concat(tempD, axis=0, ignore_index=True)
dfD.insert(loc=len(dfD.columns), column='Y', value='0', allow_duplicates=True)


## Data E
pathE = r'H:/Final_project/Data/merged/E'
all_filesE = glob.glob(pathE + "/*.csv")

tempE = []

for filename in all_filesE:
    df5 = pd.read_csv(filename, index_col=None, header=0)
    tempE.append(df5)

dfE = pd.concat(tempE, axis=0, ignore_index=True)
dfE.insert(loc=len(dfE.columns), column='Y', value='1', allow_duplicates=True)
```

In [4]:
```
# merged all the dataset
df = pd.concat([dfA, dfB,dfC,dfD,dfE], ignore_index=True)
# displaying only frist 10 entries
df.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	169	170	171	172	173	174	175	176	177
0	45	69	74	79	78	66	43	33	36	34	...	-53	-48	-40	-17	-23	-32	-41	-50	-53
1	-33	-31	-28	-18	-20	-16	-3	19	33	24	...	39	54	54	34	29	31	33	36	35
2	22	9	15	20	18	3	-16	-34	-50	-30	...	33	42	46	56	59	42	29	16	14
3	6	-1	-14	-26	-23	-18	-10	-10	-3	-5	...	-33	-40	-36	-28	-33	-38	-44	-37	-25
4	0	29	50	72	90	102	92	75	44	20	...	-50	-38	-28	-31	-33	-29	-21	-7	-1

5 rows × 179 columns

In [5]:
```
# shape of the dataset
df.shape
```

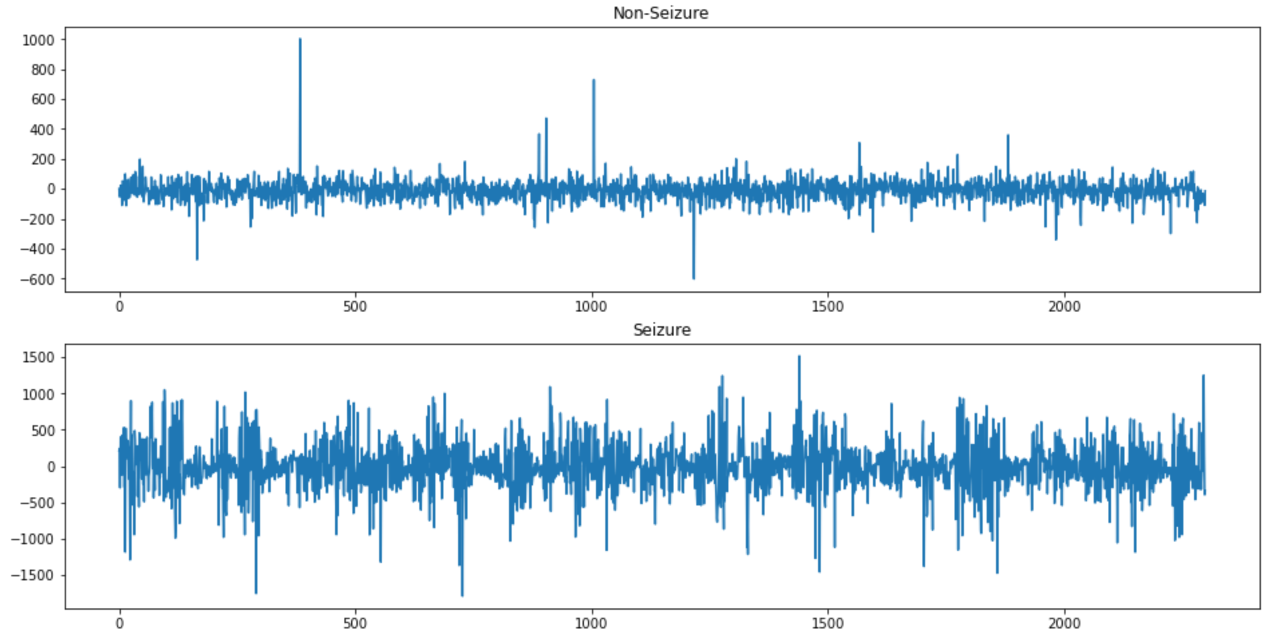
Out[5]:
```
(11500, 179)
```

In [6]:
```
# size of the dataset
df.size
```

Out[6]:
```
2058500
```

In [7]:
```
# The dataset consists of 11,500 samples with 178 features
```

In [8]:
```
# Visualizing the data points

df_nseiz=pd.concat([dfA, dfB,dfC,dfD], ignore_index=True) # non-seizure dataset (files 

df_nseiz = shuffle(df_nseiz) # seizure dataset (file E)
df_seiz=dfE

X_ns = df_nseiz.iloc[:,1:177].values
X_s =  df_seiz.iloc[:,1:177].values
samples = np.arange(0,2300,1)

#let us plot the siezure and non-seizure data
plt.figure(figsize=(16,8))
plt.subplot(2,1,1)
plt.title('Non-Seizure')
plt.plot(samples,X_ns[:2300,0], label = 'Non-Seizure')
plt.subplot(2,1,2)
plt.title('Seizure')
plt.plot(samples,X_s[:,0],label= 'Seizure')
plt.savefig('ns.png')
plt.show()
```



feature extraction

In [9]:

```
# feature extraction using wavelet transform
# The EEG signal is decomposed using Daubechies wavelet of order 4 (db4)
# and statistical features are extracted from the wavelet coefficients in each sub-band
import pywt

target=df["Y"]
target = target.astype('int')
df2=df.drop(["Y"],axis=1)



# let us define function to calcualte the statiscal features
def statisticsForWavelet(coefs):
    median = np.nanpercentile(coefs, 50)
    mean = np.nanmean(coefs)
    std = np.nanstd(coefs)
    var = np.nanvar(coefs)
    rms = np.nanmean(np.sqrt(coefs**2))
    return [median, mean, std, var, rms]

#let us define function to calculate the wavelet coeficient using using Daubechies wave
def getWaveletFeatures(data,target):
    list_features = []
    for signal in range(len(data)):
        list_coeff = pywt.wavedec(data.iloc[signal], "db4")
        features = []
        for coeff in list_coeff:
            features += statisticsForWavelet(coeff)
        list_features.append(features)
    return createDfWavelet(list_features,target)

# let us define the final function to combine the results of these function
def createDfWavelet(data,target):
    for i in range(len(data)):
        data[i].append(target[i])
```

```
    return pd.DataFrame(data)


# visualize the features
df2_fea=getWaveletFeatures(df2,target)
df2_fea = shuffle(df2_fea)
df2_fea.head()
```

Out[9]:

	0	1	2	3	4	5	6	7
5783	-26.282653	-26.875206	66.132637	4373.525694	57.603995	-4.231536	-1.390345	66.599979
6895	-184.206449	-123.637340	198.832822	39534.490914	196.536038	-10.399480	-7.865153	65.177462
589	71.169943	46.094077	80.420413	6467.442816	78.748800	0.044817	-10.294989	40.637858
3864	-457.575360	-386.524956	154.284271	23803.636224	386.524956	3.523396	5.927985	37.289195
7658	-36.393986	2.389391	172.276913	29679.334749	157.566954	3.256909	9.042643	78.589691

5 rows × 26 columns

In [11]:

```
# let us use the input and target
X = df2_fea.iloc[:,0:24].values
Y = df2_fea.iloc[:,25].values

# to train the SVM model we normalized the dataset using standardscalar
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# split the dataset into training (80%) and testing dataset(20%)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2)
```

SVM model

In [12]:

```
# now we used the support vector machine (SVM) model for seizure detection

from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_predict
svmclf = SVC(kernel='rbf', probability=True)



# let us define the hyperparameters of SVM model
param_grid = {
    'C':   [0.001, 0.01, 0.1, 1, 10],
    'kernel': ['linear', 'poly','rbf'],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'degree': [1, 2, 3, 4, 5]}

## we used randomizedserch  to find the best hyperparameters

searchSVM = RandomizedSearchCV(svmclf, param_grid, cv = 5,     #5-fold cross-validation
```

```
                                n_iter=5, scoring ='accuracy',
                                refit='precision',
                                return_train_score=False,
                                n_jobs=-1,
                                verbose=1)
searchSVM.fit(X_train,Y_train)

# search the optimal hypermeters
SVM = searchSVM.best_params_

# extract the best hyperparameters
clf_svm = SVC(
    C = SVM['C'],
    kernel = SVM['kernel'],
    gamma = SVM['gamma'],
    degree = SVM['degree'],probability=True)

# after getting the best hyperparameter,we trained the SVM model using the fit method
clf_svm.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[12]:

```
SVC(C=1, degree=1, gamma=0.0001, kernel='linear', probability=True)
```

In [13]:

```
# let us check whether the model is overfitting or not
# we calculated the performance of model on training dataset
y_predtrain = cross_val_predict(clf_svm, X_train, Y_train, cv=5)

# print the performance metrics (precison, recall and f1 score)
print("Training metrics\n",classification_report(Y_train, y_predtrain,target_names=['No
```

```
Training metrics
               precision    recall  f1-score   support

 Non-Seizure       0.98      0.98      0.98      7354
     Seizure       0.93      0.91      0.92      1846

    accuracy                           0.97      9200
   macro avg       0.95      0.94      0.95      9200
weighted avg       0.97      0.97      0.97      9200
```

In [14]:

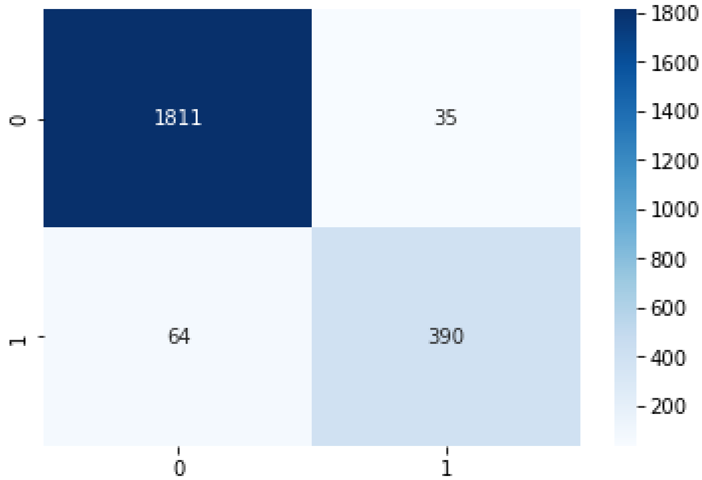
```
# and then we calculated the performance of model on testing dataset

y_pred = cross_val_predict(clf_svm, X_test, Y_test, cv=5)

# print the performance metrics (precison, recall and f1 score) on testing dataset

print("Testing metrics \n",classification_report(Y_test, y_pred,target_names=['Non-Seiz

# let us plot the confusion matrix of trained model on test data
cm = confusion_matrix(Y_test, y_pred)
df_cm = pd.DataFrame(cm, range(2), range(2))
sn.heatmap(df_cm, annot=True,fmt='g',cmap ='Blues')
```

```
Testing metrics
               precision    recall  f1-score   support

 Non-Seizure       0.97      0.98      0.97      1846
```

```
     Seizure       0.92      0.86      0.89       454

    accuracy                           0.96      2300
   macro avg       0.94      0.92      0.93      2300
weighted avg       0.96      0.96      0.96      2300
```

Out[14]:

```
<AxesSubplot:>
```



balance the dataset

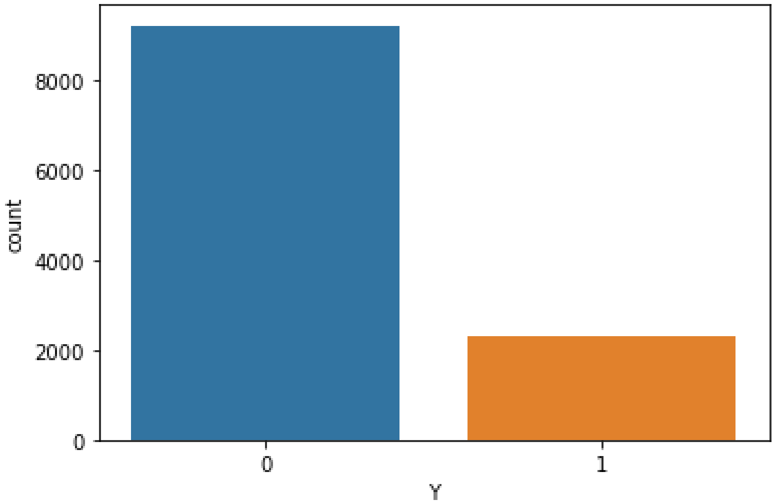
In [15]:

```
# since there is not much difference in training and testing metrics,we can say that ou
```

In [16]:

```
# visualize the percentage of classes in dataset

import seaborn as sn
cols = df.columns
tgt = df.Y
ax = sn.countplot(tgt,label="Count")
non_seizure, seizure = tgt.value_counts()
# p
print('The number of samples for the non-seizure class is:', non_seizure)
print('The number of samples for the seizure class is:', seizure)
```

```
The number of samples for the non-seizure class is: 9200
The number of samples for the seizure class is: 2300
```



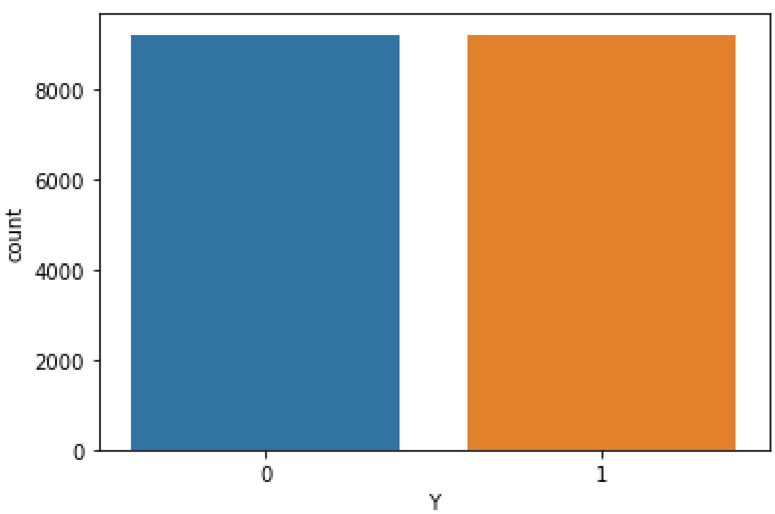
In [ ]:

```
# Since most of the data are from non-seizure, the dataset is highly imbalanced.So we u
#which consists of synthesizing elements for the minority class, based on those that al
```

In [17]:

```
from imblearn.over_sampling import SMOTE

Yd=df["Y"]
Yd = Yd.astype('int')
Xd=df.drop(["Y"],axis=1)
Xd.head()

sm = SMOTE(random_state=40)

X_sm, Y_sm = sm.fit_resample(Xd, Yd)
ax = sn.countplot(Y_sm, label="Count")
non_seizure = np.count_nonzero(Y_sm == 0)
seizure = np.count_nonzero(Y_sm == 1)
print('The number of samples for the non-seizure class is:', non_seizure)
print('The number of samples for the seizure class is:', seizure)
```

```
The number of samples for the non-seizure class is: 9200
The number of samples for the seizure class is: 9200
```



In [19]:

```
# extract the features of balanced dataset using wavelet tranform
```

```
df3_fea=getWaveletFeatures(X_sm,Y_sm)
df3_fea = shuffle(df3_fea)
df3_fea.head()
```

Out[19]:

	0	1	2	3	4	5	6	
2937	213.131509	142.741244	148.539011	22063.837657	182.589342	28.881455	47.336157	120.86106
16928	-326.504329	-64.905372	718.334301	516004.168035	622.067940	-5.040416	-41.472954	657.71840
7919	124.913097	69.560904	199.480252	39792.370747	186.115948	1.667044	4.224123	73.49374
14407	589.646546	312.770828	705.534187	497778.488396	712.115525	49.298905	-18.629135	283.23188
4765	11.116920	10.781207	221.471949	49049.824169	198.946153	-0.987599	-15.592580	163.86717

5 rows × 26 columns

In [20]:

```
Xb = df3_fea.iloc[:,0:24].values
Yb = df3_fea.iloc[:,25].values

# to train the SVM model we normalized the dataset using standardscalar
scaler = StandardScaler()
scaler.fit(Xb)
Xb = scaler.transform(Xb)

# split the balanced dataset in training (80%) and testing (20%) dataset
Xb_train, Xb_test, Yb_train, Yb_test = train_test_split(Xb, Yb, test_size = 0.2)
```

In [22]:

```
# we trained the SVM model on balanced dataset


svmclf2 = SVC(kernel='rbf', probability=True)

# let us define the hyperparameters of SVM model
param_grid2 = {
    'C':   [0.001, 0.01, 0.1, 1, 10],
    'kernel': ['linear', 'poly','rbf'],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'degree': [1, 2, 3, 4, 5]}

searchSVM2 = RandomizedSearchCV(svmclf2, param_grid2, cv = 5,
                              n_iter=5, scoring ='accuracy',
                              refit='precision',
                              return_train_score=False,
                              n_jobs=-1,
                              verbose=1)
searchSVM2.fit(Xb_train,Yb_train)

SVM2 = searchSVM2.best_params_

clf_svm2 = SVC(
    C = SVM['C'],
    kernel = SVM['kernel'],
    gamma = SVM['gamma'],
    degree = SVM['degree'],probability=True)
```

```
clf_svm2.fit(Xb_train,Yb_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[22]:

```
SVC(C=1, degree=1, gamma=0.0001, kernel='linear', probability=True)
```

In [23]:

```
# let us check whether the model is overfitting or not
# we calculated the performance of model on training dataset
y_predtrain2 = cross_val_predict(clf_svm2, Xb_train, Yb_train, cv=5)
print("Training metrics\n",classification_report(Yb_train, y_predtrain2,target_names=['
```

```
Training metrics
               precision    recall  f1-score   support

 Non-Seizure       0.95      0.95      0.95      7374
     Seizure       0.95      0.95      0.95      7346

    accuracy                           0.95     14720
   macro avg       0.95      0.95      0.95     14720
weighted avg       0.95      0.95      0.95     14720
```

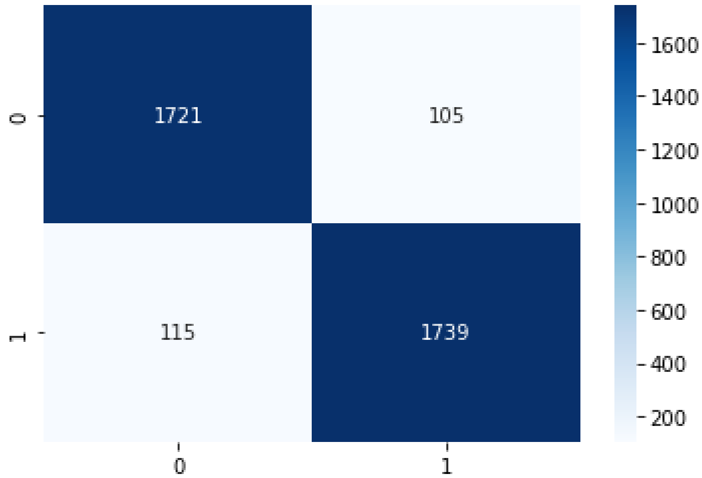
In [24]:

```
# let us evaluate the performance of SVM model on balanced dataset

y_predb = cross_val_predict(clf_svm2, Xb_test, Yb_test, cv=5)
print("Testing metrics\n",classification_report(Yb_test, y_predb,target_names=['Non-Sei

# confusion matrix
cmb = confusion_matrix(Yb_test, y_predb)
df_cmb = pd.DataFrame(cmb, range(2), range(2))
sn.heatmap(df_cmb, annot=True,fmt='g',cmap ='Blues')
```

```
Testing metrics
               precision    recall  f1-score   support

 Non-Seizure       0.94      0.94      0.94      1826
     Seizure       0.94      0.94      0.94      1854

    accuracy                           0.94      3680
   macro avg       0.94      0.94      0.94      3680
weighted avg       0.94      0.94      0.94      3680
```

Out[24]:

```
<AxesSubplot:>
```



LSTM model

In [ ]:

In [25]:

```
## Now the second model used in this work to for seizure detection is LSTM
```

In [26]:

```
# we used the original dataset
from keras.utils import to_categorical
df = shuffle(df)
X1 = df.iloc[:,1:177].values
Y1 = df.iloc[:,178].values

# scaling the data using standard scalar
scaler = StandardScaler()
scaler.fit(X1)
X1 = scaler.transform(X1)

Y1 = to_categorical(Y1)

# divide the dataset into training (60%) validation (20%) and testing dataset(20%)
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size = 0.2)
X1_train, X1_val, Y1_train, Y1_val = train_test_split(X1_train, Y1_train, test_size=0.2

X1_train = np.reshape(X1_train, (X1_train.shape[0],1,X1.shape[1]))
X1_test = np.reshape(X1_test, (X1_test.shape[0],1,X1.shape[1]))
X1_val = np.reshape(X1_val, (X1_val.shape[0],1,X1.shape[1]))
```

In [27]:

```
# import all the library neccesary for LSTM

import tensorflow as tf
from tensorflow.keras import Sequential
from keras.optimizers import SGD
from keras_tuner import RandomSearch

from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import LSTM
tf.keras.backend.clear_session()
```

In [28]:

```
#  to find the the optimal number of LSTM layers we used hp.int() in a for loop, which
# To avoid overfitting the neural network, we add a dropout layer and to find the right
# and defined the the final layer as dense layer
#After defining the hyper-parameters we compiled the model with RMSprop optimizer,
# binary cross-Entropy loss function, and metric and return that model

# we used early stopping to Stop the training when a monitored metric has stopped impro

stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)


def build_model(hp):
    model = Sequential()
    model.add(LSTM(hp.Int('input_unit',min_value=16,max_value=512,step=16),return_seque
    for i in range(hp.Int('n_layers', 1, 4)):
        model.add(LSTM(hp.Int(f'lstm_{i}_units',min_value=16,max_value=512,step=16),ret
    model.add(LSTM(hp.Int('layer_2_neurons',min_value=16,max_value=512,step=16)))
    model.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.9,step=0.1)))
    model.add(Dense(Y1_train.shape[1], activation=hp.Choice('dense_activation',values=[
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=hp_learning_rate),
                  loss = 'binary_crossentropy', metrics = ['accuracy'])
    return model
```

In [29]:

```
#To find the best model we used  hyperband to optimize the hyperparameters
tuner = kt.Hyperband(build_model,
                     objective='val_accuracy',
                     max_epochs=50,
                     factor=3,
                     directory=r'H:\Final_project\Save_model\binaryclass', # path of di
                     project_name='Best_model')
```

```
INFO:tensorflow:Reloading Oracle from existing project H:\Final_project\Save_model\binar
yclass\Best_model\oracle.json
INFO:tensorflow:Reloading Tuner from H:\Final_project\Save_model\binaryclass\Best_model
\tuner0.json
```

In [30]:

```
# after defining the tuner we used search method to find the best model and save the mo
tuner.search(
        x=X1_train,
        y=Y1_train,
        epochs=50,
        validation_data=(X1_val,Y1_val),
        callbacks=[stop_early]
        )
```

```
INFO:tensorflow:Oracle triggered exit
```

In [31]:

```
# hyperparameters of best models
tuner.results_summary()
```

```
Results summary
Results in H:\Final_project\Save_model\binaryclass\Best_model
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x000001E9D1A5BB20>
```

```
Trial summary
Hyperparameters:
input_unit: 480
n_layers: 1
lstm_0_units: 400
layer_2_neurons: 64
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 464
lstm_2_units: 32
lstm_3_units: 32
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 3
tuner/round: 2
tuner/trial_id: 0037
Score: 0.9743478298187256
Trial summary
Hyperparameters:
input_unit: 240
n_layers: 2
lstm_0_units: 288
layer_2_neurons: 144
Dropout_rate: 0.2
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 272
lstm_2_units: 432
lstm_3_units: 32
tuner/epochs: 50
tuner/initial_epoch: 17
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 0079
Score: 0.9730435013771057
Trial summary
Hyperparameters:
input_unit: 416
n_layers: 2
lstm_0_units: 112
layer_2_neurons: 432
Dropout_rate: 0.6000000000000001
dense_activation: sigmoid
learning_rate: 0.0001
lstm_1_units: 192
lstm_2_units: 496
lstm_3_units: 96
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 2
tuner/round: 1
tuner/trial_id: 0066
Score: 0.9726086854934692
Trial summary
Hyperparameters:
input_unit: 240
n_layers: 2
lstm_0_units: 288
layer_2_neurons: 144
```

```
Dropout_rate: 0.2
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 272
lstm_2_units: 432
lstm_3_units: 32
tuner/epochs: 17
tuner/initial_epoch: 0
tuner/bracket: 1
tuner/round: 0
Score: 0.9726086854934692
Trial summary
Hyperparameters:
input_unit: 352
n_layers: 1
lstm_0_units: 96
layer_2_neurons: 192
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 416
lstm_2_units: 144
lstm_3_units: 64
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 3
tuner/round: 2
tuner/trial_id: 0034
Score: 0.9717391133308411
Trial summary
Hyperparameters:
input_unit: 416
n_layers: 2
lstm_0_units: 112
layer_2_neurons: 432
Dropout_rate: 0.6000000000000001
dense_activation: sigmoid
learning_rate: 0.0001
lstm_1_units: 192
lstm_2_units: 496
lstm_3_units: 96
tuner/epochs: 6
tuner/initial_epoch: 0
tuner/bracket: 2
tuner/round: 0
Score: 0.9713043570518494
Trial summary
Hyperparameters:
input_unit: 368
n_layers: 2
lstm_0_units: 64
layer_2_neurons: 368
Dropout_rate: 0.5
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 272
lstm_2_units: 224
lstm_3_units: 288
tuner/epochs: 6
tuner/initial_epoch: 0
```

```
tuner/bracket: 2
tuner/round: 0
Score: 0.9708695411682129
Trial summary
Hyperparameters:
input_unit: 368
n_layers: 2
lstm_0_units: 64
layer_2_neurons: 368
Dropout_rate: 0.5
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 272
lstm_2_units: 224
lstm_3_units: 288
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 2
tuner/round: 1
tuner/trial_id: 0054
Score: 0.9704347848892212
Trial summary
Hyperparameters:
input_unit: 128
n_layers: 1
lstm_0_units: 480
layer_2_neurons: 352
Dropout_rate: 0.6000000000000001
dense_activation: sigmoid
learning_rate: 0.0001
lstm_1_units: 144
lstm_2_units: 112
lstm_3_units: 352
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 2
tuner/round: 1
tuner/trial_id: 0057
Score: 0.9704347848892212
Trial summary
Hyperparameters:
input_unit: 352
n_layers: 1
lstm_0_units: 96
layer_2_neurons: 192
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 416
lstm_2_units: 144
lstm_3_units: 64
tuner/epochs: 6
tuner/initial_epoch: 2
tuner/bracket: 3
tuner/round: 1
tuner/trial_id: 0022
Score: 0.9700000286102295
```

In [32]:

```
# we extract the best model using the get_best_models method of the tuner instance
```

```
best_model = tuner.get_best_models()[0]
best_model.build(X1_train.shape)
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 1, 480)            1261440   
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 400)            1409600   
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                119040    
_________________________________________________________________
dropout (Dropout)            (None, 64)                0         
_________________________________________________________________
dense (Dense)                (None, 2)                 130       
=================================================================
Total params: 2,790,210
Trainable params: 2,790,210
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```
# now we trained the best model using the fit method
best_model.fit(X1_train,Y1_train,epochs=50,
        validation_data=(X1_val,Y1_val),
        batch_size = 16,
        callbacks=[stop_early])
```

```
Epoch 1/50
432/432 [==============================] - 10s 24ms/step - loss: 0.0724 - accuracy: 0.98
06 - val_loss: 0.0529 - val_accuracy: 0.9848
Epoch 2/50
432/432 [==============================] - 9s 21ms/step - loss: 0.0592 - accuracy: 0.983
5 - val_loss: 0.0624 - val_accuracy: 0.9791
Epoch 3/50
432/432 [==============================] - 9s 22ms/step - loss: 0.0444 - accuracy: 0.986
5 - val_loss: 0.0673 - val_accuracy: 0.9835
Epoch 4/50
432/432 [==============================] - 9s 21ms/step - loss: 0.0413 - accuracy: 0.988
4 - val_loss: 0.0779 - val_accuracy: 0.9787
Epoch 5/50
432/432 [==============================] - 9s 22ms/step - loss: 0.0312 - accuracy: 0.990
3 - val_loss: 0.0879 - val_accuracy: 0.9783
Epoch 6/50
432/432 [==============================] - 10s 22ms/step - loss: 0.0292 - accuracy: 0.99
14 - val_loss: 0.0752 - val_accuracy: 0.9835
```

Out[33]:

```
<tensorflow.python.keras.callbacks.History at 0x1e9e0cb8d90>
```

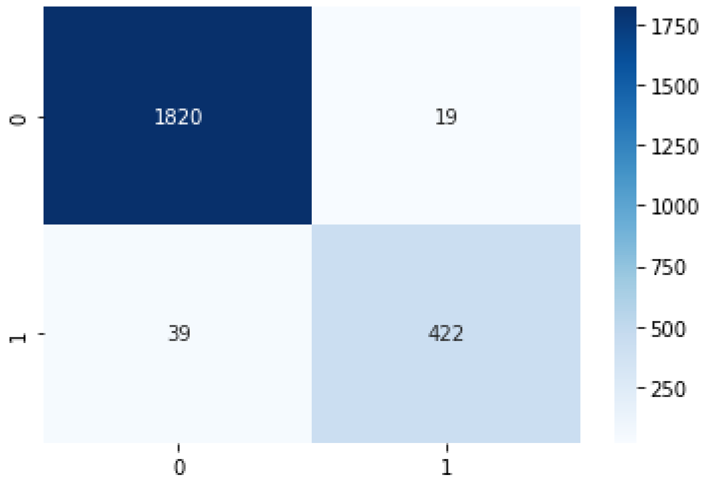
In [34]:

```
# evaluate the performance of best model on test data
Y_pred = best_model.predict(X1_test)
print(classification_report(Y1_test.argmax(axis=1), Y_pred.argmax(axis=1), target_names

#confusion matrix

cm = confusion_matrix(Y1_test.argmax(axis=1), Y_pred.argmax(axis=1))
df_cm = pd.DataFrame(cm, range(2), range(2))
sn.heatmap(df_cm, annot=True,fmt='g',cmap ='Blues')
```

```
              precision    recall  f1-score   support

 Non-seizure       0.98      0.99      0.98      1839
     Seizure       0.96      0.92      0.94       461

    accuracy                           0.97      2300
   macro avg       0.97      0.95      0.96      2300
weighted avg       0.97      0.97      0.97      2300
```

Out[34]:

```
<AxesSubplot:>
```



In [ ]:

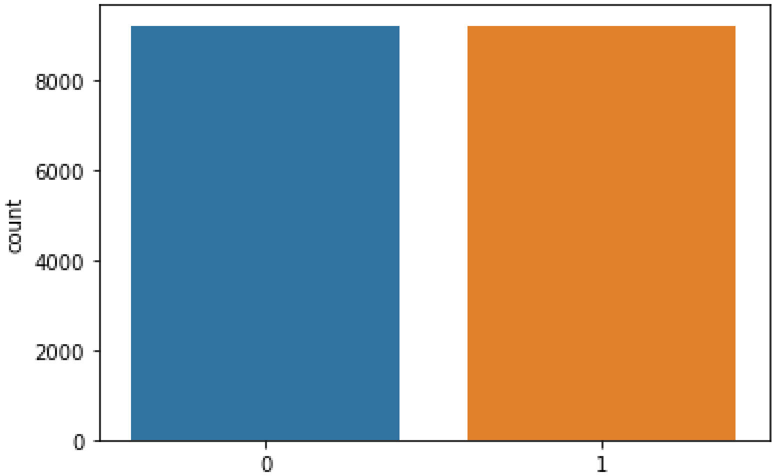
LSTM model on balanced dataset

In [35]:

```
# Now test the perforamnce of LSTM model on balanced dataset
```

In [36]:

```
df_sm = shuffle(df)
X2 = df_sm.iloc[:,1:177].values
Y2 = df_sm.iloc[:,178].values
# define SMOTE
sm = SMOTE(random_state=40)

X1_sm, Y1_sm = sm.fit_resample(X2, Y2)
ax = sn.countplot(Y1_sm,label="Count")
```



In [37]:

```
# scaling the data using standard scalar
scaler = StandardScaler()
scaler.fit(X1_sm)
X1_sm = scaler.transform(X1_sm)

Y1_sm = to_categorical(Y1_sm)

# divide the dataset into training (60%) validation (20%) and testing dataset(20%)
X1sm_train, X1sm_test, Y1sm_train, Y1sm_test = train_test_split(X1_sm, Y1_sm, test_size
X1sm_train, X1sm_val, Y1sm_train, Y1sm_val = train_test_split(X1sm_train, Y1sm_train, t

X1sm_train = np.reshape(X1sm_train, (X1sm_train.shape[0],1,X1_sm.shape[1]))
X1sm_test = np.reshape(X1sm_test, (X1sm_test.shape[0],1,X1_sm.shape[1]))
X1sm_val = np.reshape(X1sm_val, (X1sm_val.shape[0],1,X1_sm.shape[1]))
```

In [38]:

```
# define the LSTM model for balanced dataset

stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

def build_model2(hp):
    model2 = Sequential()
    model2.add(LSTM(hp.Int('input_unit',min_value=16,max_value=512,step=16),return_sequ
    for i in range(hp.Int('n_layers', 1, 4)):
        model2.add(LSTM(hp.Int(f'lstm_{i}_units',min_value=16,max_value=512,step=16),re
    model2.add(LSTM(hp.Int('layer_2_neurons',min_value=16,max_value=512,step=16)))
    model2.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.9,step=0.1)))
    model2.add(Dense(Y1_train.shape[1], activation=hp.Choice('dense_activation',values=
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
    model2.compile(optimizer=keras.optimizers.RMSprop(learning_rate=hp_learning_rate),
                   loss = 'binary_crossentropy', metrics = ['accuracy'])
    return model2
```

In [39]:

```
#To find the best model we used  hyperband to optimize the hyperparameters
tuner2 = kt.Hyperband(build_model2,
                      objective='val_accuracy',
                      max_epochs=50,
                      factor=3,
                      directory=r'H:\Final_project\Save_model\binaryclass', # path of di
                      project_name='Best_modelsm')
```

```
INFO:tensorflow:Reloading Oracle from existing project H:\Final_project\Save_model\binar
yclass\Best_modelsm\oracle.json
INFO:tensorflow:Reloading Tuner from H:\Final_project\Save_model\binaryclass\Best_models
m\tuner0.json
```

In [40]:

```
# after defining the tuner we used search method to find the best model and save the mo
tuner2.search(
        x=X1sm_train,
        y=Y1sm_train,
        epochs=50,
        validation_data=(X1sm_val,Y1sm_val),
        callbacks=[stop_early]
        )
```

```
INFO:tensorflow:Oracle triggered exit
```

In [43]:

```
# hyperparameters of best models
tuner2.results_summary()
```

```
Results summary
Results in H:\Final_project\Save_model\binaryclass\Best_modelsm
Showing 10 best trials
<keras_tuner.engine.objective.Objective object at 0x000001E9D12CBCA0>
Trial summary
Hyperparameters:
input_unit: 368
n_layers: 3
lstm_0_units: 224
layer_2_neurons: 208
Dropout_rate: 0.5
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 368
lstm_2_units: 448
lstm_3_units: 48
tuner/epochs: 50
tuner/initial_epoch: 17
tuner/bracket: 3
tuner/round: 3
tuner/trial_id: 0049
Score: 0.990217387676239
Trial summary
Hyperparameters:
input_unit: 272
n_layers: 4
lstm_0_units: 192
layer_2_neurons: 192
Dropout_rate: 0.1
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 128
lstm_2_units: 512
lstm_3_units: 192
tuner/epochs: 17
tuner/initial_epoch: 0
tuner/bracket: 1
tuner/round: 0
Score: 0.988043487071991
Trial summary
```

```
Hyperparameters:
input_unit: 400
n_layers: 2
lstm_0_units: 192
layer_2_neurons: 32
Dropout_rate: 0.0
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 80
lstm_2_units: 224
lstm_3_units: 32
tuner/epochs: 50
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.98777174949646
Trial summary
Hyperparameters:
input_unit: 368
n_layers: 3
lstm_0_units: 224
layer_2_neurons: 208
Dropout_rate: 0.5
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 368
lstm_2_units: 448
lstm_3_units: 48
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 3
tuner/round: 2
tuner/trial_id: 0036
Score: 0.987500011920929
Trial summary
Hyperparameters:
input_unit: 272
n_layers: 4
lstm_0_units: 192
layer_2_neurons: 192
Dropout_rate: 0.1
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 128
lstm_2_units: 512
lstm_3_units: 192
tuner/epochs: 50
tuner/initial_epoch: 17
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 0080
Score: 0.987228274345398
Trial summary
Hyperparameters:
input_unit: 384
n_layers: 2
lstm_0_units: 144
layer_2_neurons: 160
Dropout_rate: 0.0
dense_activation: sigmoid
```

```
learning_rate: 0.01
lstm_1_units: 384
lstm_2_units: 320
lstm_3_units: 352
tuner/epochs: 50
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.9864130616188049
Trial summary
Hyperparameters:
input_unit: 320
n_layers: 2
lstm_0_units: 368
layer_2_neurons: 192
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.01
lstm_1_units: 304
lstm_2_units: 512
lstm_3_units: 480
tuner/epochs: 50
tuner/initial_epoch: 17
tuner/bracket: 3
tuner/round: 3
tuner/trial_id: 0047
Score: 0.9850543737411499
Trial summary
Hyperparameters:
input_unit: 416
n_layers: 1
lstm_0_units: 48
layer_2_neurons: 512
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 304
lstm_2_units: 288
lstm_3_units: 480
tuner/epochs: 50
tuner/initial_epoch: 17
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: 0067
Score: 0.9850543737411499
Trial summary
Hyperparameters:
input_unit: 336
n_layers: 1
lstm_0_units: 352
layer_2_neurons: 432
Dropout_rate: 0.6000000000000001
dense_activation: sigmoid
learning_rate: 0.01
lstm_1_units: 128
lstm_2_units: 304
lstm_3_units: 416
tuner/epochs: 17
tuner/initial_epoch: 0
tuner/bracket: 1
```

```
tuner/round: 0
Score: 0.9847826361656189
Trial summary
Hyperparameters:
input_unit: 416
n_layers: 1
lstm_0_units: 48
layer_2_neurons: 512
Dropout_rate: 0.4
dense_activation: sigmoid
learning_rate: 0.001
lstm_1_units: 304
lstm_2_units: 288
lstm_3_units: 480
tuner/epochs: 17
tuner/initial_epoch: 6
tuner/bracket: 2
tuner/round: 1
tuner/trial_id: 0065
Score: 0.9842391014099121
```

In [44]:

```
# let us choose the best model
best_model2 = tuner2.get_best_models()[0]
best_model2.build(X1sm_train.shape)
best_model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 1, 368)            802240
_________________________________________________________________
lstm_1 (LSTM)                (None, 1, 224)            531328
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 368)            872896
_________________________________________________________________
lstm_3 (LSTM)                (None, 1, 448)            1464064
_________________________________________________________________
lstm_4 (LSTM)                (None, 208)               546624
_________________________________________________________________
dropout (Dropout)            (None, 208)               0
_________________________________________________________________
dense (Dense)                (None, 2)                 418
=================================================================
Total params: 4,217,570
Trainable params: 4,217,570
Non-trainable params: 0
_________________________________________________________________
```

In [45]:

```
# now we trained the best model using the fit method
best_model2.fit(X1sm_train,Y1sm_train,epochs=50,
        validation_data=(X1sm_val,Y1sm_val),
        batch_size = 16,
        callbacks=[stop_early])
```

```
Epoch 1/50
690/690 [==============================] - 23s 33ms/step - loss: 0.0446 - accuracy: 0.98
73 - val_loss: 0.0405 - val_accuracy: 0.9867
Epoch 2/50
```

```
690/690 [==============================] - 21s 30ms/step - loss: 0.0277 - accuracy: 0.99
21 - val_loss: 0.0934 - val_accuracy: 0.9783
Epoch 3/50
690/690 [==============================] - 21s 30ms/step - loss: 0.0221 - accuracy: 0.99
42 - val_loss: 0.0634 - val_accuracy: 0.9889
Epoch 4/50
690/690 [==============================] - 21s 30ms/step - loss: 0.0188 - accuracy: 0.99
38 - val_loss: 0.0513 - val_accuracy: 0.9875
Epoch 5/50
690/690 [==============================] - 21s 30ms/step - loss: 0.0193 - accuracy: 0.99
56 - val_loss: 0.0639 - val_accuracy: 0.9880
Epoch 6/50
690/690 [==============================] - 21s 30ms/step - loss: 0.0151 - accuracy: 0.99
55 - val_loss: 0.0686 - val_accuracy: 0.9902
```

Out[45]:

```
<tensorflow.python.keras.callbacks.History at 0x1e9ef9dd070>
```

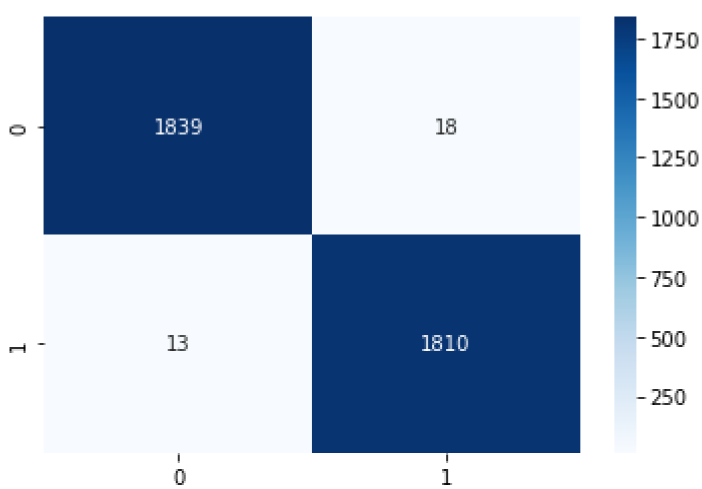
In [46]:

```
# predict the performance of best model on test
Y2_pred = best_model2.predict(X1sm_test)

# print the performance metrics (precison, recall and f1 score) on testing dataset
print(classification_report(Y1sm_test.argmax(axis=1), Y2_pred.argmax(axis=1), target_na

# plot the confusion matrix
cm2 = confusion_matrix(Y1sm_test.argmax(axis=1), Y2_pred.argmax(axis=1))
df_cm2 = pd.DataFrame(cm2, range(2), range(2))
sn.heatmap(df_cm2, annot=True,fmt='g',cmap ='Blues')
```

```
              precision    recall  f1-score   support

 Non-seizure       0.99      0.99      0.99      1857
     Seizure       0.99      0.99      0.99      1823

    accuracy                           0.99      3680
   macro avg       0.99      0.99      0.99      3680
weighted avg       0.99      0.99      0.99      3680
```

Out[46]:

```
<AxesSubplot:>
```



ROC curves of all models

In [47]:

```
pred_prob1 = clf_svm.predict_proba(X_test) # SVM model on unbalanced dataset
```

```
pred_prob2 = clf_svm2.predict_proba(Xb_test) #SVM model on balanced dataset
pred_prob3 = best_model.predict_proba(X1_test) # LSTM model on unbalanced dataset
pred_prob4 = best_model2.predict_proba(X1sm_test) #LSTM model on balanced dataset
```

```
WARNING:tensorflow:From C:\Users\kau19001\AppData\Local\Temp\ipykernel_23768\3805367940.
py:3: Sequential.predict_proba (from tensorflow.python.keras.engine.sequential) is depre
cated and will be removed after 2021-01-01.
Instructions for updating:
Please use `model.predict()` instead.
```

In [48]:

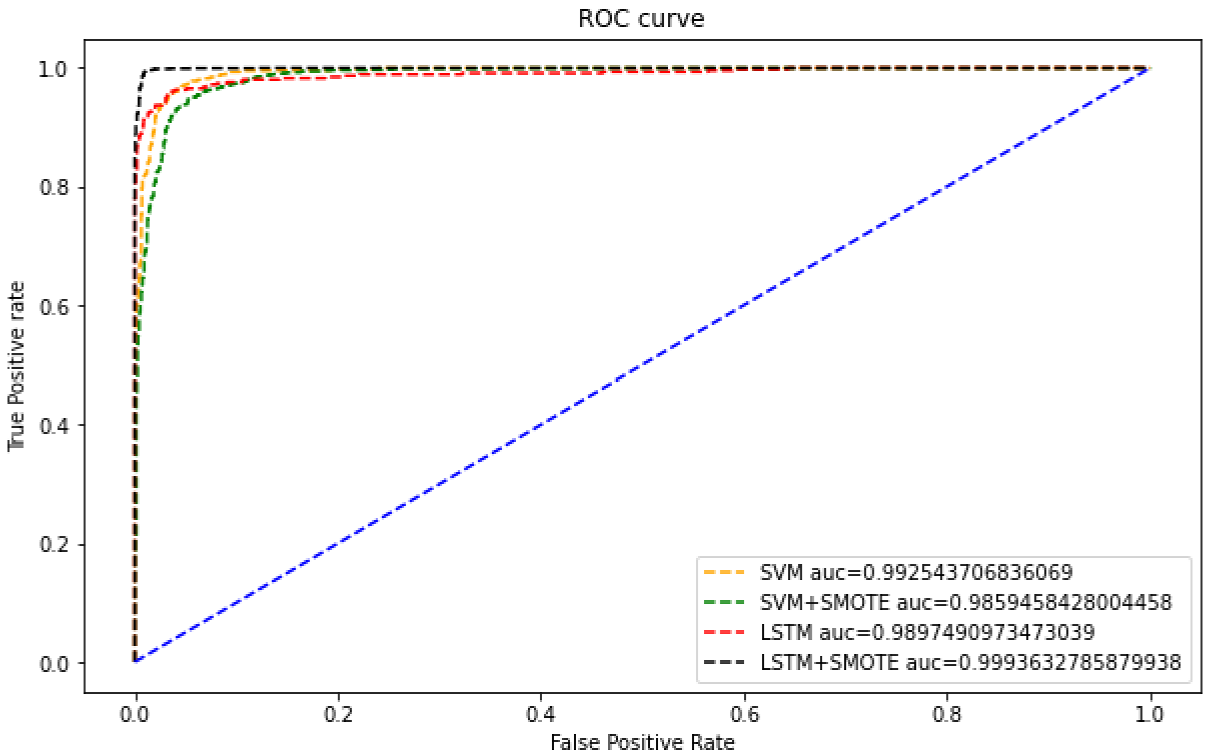
```
from sklearn.metrics import roc_curve, roc_auc_score, auc
fpr1, tpr1, thresh1 = roc_curve(Y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(Yb_test, pred_prob2[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(Y1_test.argmax(axis=1), pred_prob3[:,1], pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(Y1sm_test.argmax(axis=1), pred_prob4[:,1], pos_label=1)

random_probs = [0 for i in range(len(Y_test))]
p_fpr, p_tpr, _ = roc_curve(Y_test, random_probs, pos_label=1)
```

In [49]:

```
from sklearn import metrics
auc1 = metrics.roc_auc_score(Y_test, pred_prob1[:,1])
auc2 = metrics.roc_auc_score(Yb_test, pred_prob2[:,1])
auc3 = metrics.roc_auc_score(Y1_test.argmax(axis=1), pred_prob3[:,1])
auc4 = metrics.roc_auc_score(Y1sm_test.argmax(axis=1), pred_prob4[:,1])
```

In [50]:

```
import matplotlib.pyplot as plt
plt.figure(1, figsize=(10, 6))
plt.clf()

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label="SVM auc="+str(auc1))
plt.plot(fpr2, tpr2, linestyle='--',color='green', label="SVM+SMOTE auc="+str(auc2))
plt.plot(fpr3, tpr3, linestyle='--',color='red', label="LSTM auc="+str(auc3))
plt.plot(fpr4, tpr4, linestyle='--',color='black', label="LSTM+SMOTE auc="+str(auc4))
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```


ROC curve
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
SVM auc=0.992543706836069
SVM+SMOTE auc=0.9859458428004458
LSTM auc=0.9897490973473039
LSTM+SMOTE auc=0.9993632785879938

In [ ]:

In [ ]: