# Epileptic Seizure Prediction using EEG signals

A Project Report By

Rushil Shivade -201080053
Chirag Kathoye - 201080045

For the Academic year 2022-2023
Bachelors of Technology
(Information Technology)

Veermata Jijabai Technological Institute

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our course instructor Dr. Sandeep Sambhaji Udmale for giving us an opportunity and guidance regarding our Machine Learning project of Epileptic Seizure Detection. He has been a constant source of inspiration and motivation for us throughout this course.

We also appreciate his valuable feedback and suggestions on our project work, which helped us improve our understanding and skills in machine learning. He has been very supportive and encouraging in every aspect of our learning process. We are grateful for his time and effort in teaching us this course.

# Table of Contents

**AIM:** The Objective of the project is to solve a current world problem with the help of Machine Learning.

**Problem Statement:** Epileptic Seizure Prediction using Machine learning to detect and avoid sleep anomalies and convulsion threats.

**Analysis:**

The project involves comparison of various results obtained via different types of machine learning models. These models are developed and implemented using the famous libraries ‘keras’ and ‘Scikit-learn’. The models used over in the projects are:

1. Decision Tree
2. KNN classifier
3. Naive Bayes
4. Random Forest
5. SVM
6. ANN
7. RNN
8. LSTM
9. CNN

**1. Decision Tree:** A decision tree is a supervised machine learning algorithm that builds a tree-like model to make decisions or predictions by learning from input data. It splits the data based on different attributes and creates a tree structure where each internal node represents a decision rule and each leaf node represents an outcome.

**2. KNN classifier:** K-nearest neighbors (KNN) classifier is a supervised learning algorithm that assigns a class label to an input sample based on its K nearest neighbors in the training data. It measures the similarity between instances using distance metrics and assigns the most common class label among the K nearest neighbors to the input sample.

**3. Naive Bayes:** Naive Bayes is a probabilistic machine learning algorithm that is based on Bayes' theorem. It assumes that the features in the data are conditionally independent of each other given the class label. It calculates the probability of a sample belonging to a particular class based on the feature values and assigns the class label with the highest probability.

**4. Random Forest:** Random Forest is an ensemble learning method that combines multiple decision trees to create a more robust and accurate model. It generates a set of decision trees by bootstrapping the training data and selecting random subsets of features for each tree. The final prediction is made by aggregating the predictions of all individual trees.

**5. SVM (Support Vector Machine):** Support Vector Machine is a powerful supervised learning algorithm used for classification and regression tasks. It finds an optimal hyperplane that separates the data points of different classes with the maximum margin. SVM can handle both linear and non-linear data by using different kernel functions.

**6. ANN (Artificial Neural Network):** Artificial Neural Network is a computational model inspired by the structure and functionality of the human brain. It consists of interconnected nodes or "neurons" organized in layers. ANN can learn and generalize from input data to make predictions or decisions. It is commonly used for various machine learning tasks including classification, regression, and pattern recognition.

**7. RNN (Recurrent Neural Network):** Recurrent Neural Network is a type of neural network that is designed to process sequential data. It has connections between the nodes that form a directed cycle, allowing information to be stored and propagated across multiple time steps. RNNs are suitable for tasks such as natural language processing, speech recognition, and time series analysis.

**8. CNN (Convolutional Neural Network):** Convolutional Neural Network is a deep learning architecture primarily used for image recognition and computer vision tasks. It consists of multiple layers, including convolutional layers that apply filters to input images, pooling layers that downsample the features, and fully connected layers that make predictions. CNNs are capable of automatically learning hierarchical representations of visual data.

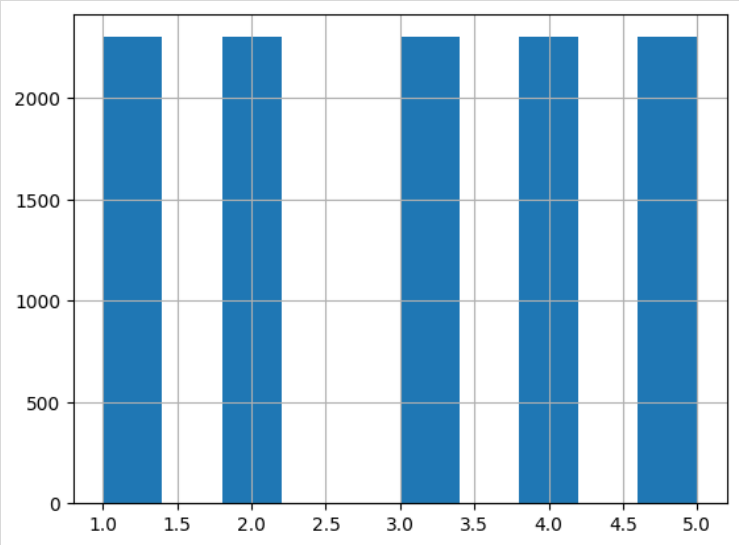
**9. LSTM (Long Short-Term Memory):** Long Short-Term Memory is a specialized type of recurrent neural network architecture. It is designed to overcome the limitation of traditional RNNs in capturing long-term dependencies. LSTM introduces memory cells and gating mechanisms that allow it to selectively store and retrieve information over extended sequences, making it particularly effective in tasks involving long-range dependencies.

## Code:

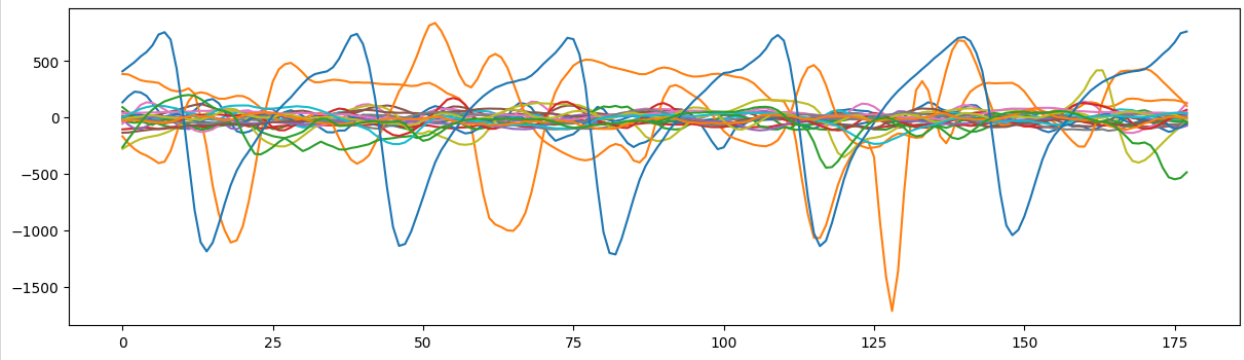
```
# %%
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import normalize, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_validate,
RepeatedStratifiedKFold, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
import plotly
import plotly.figure_factory as ff
from plotly import tools
from plotly.offline import init_notebook_mode, iplot

import imblearn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from keras.layers import Dense, Convolution1D, MaxPool1D, Flatten, Dropout
from keras.layers import Input, LSTM
from keras.models import Model
from tensorflow import keras
from tensorflow.keras import layers
# from keras.layers.normalization import BatchNormalization
from keras.layers import BatchNormalization
import keras
from matplotlib.pyplot import figure
```

```
from keras.callbacks import EarlyStopping, ModelCheckpoint

init_notebook_mode(connected=True)
pd.set_option('display.max_columns', 100)

# %%
# import mne
# import pandas as pd
# import numpy as np

# # Load the .edf file into Python
# raw = mne.io.read_raw_edf('rsvp_10Hz_02a.edf')

# # Extract the EEG signals
# eeg_data = raw.get_data()

# # Convert the EEG signals into a pandas DataFrame
# df = pd.DataFrame(np.transpose(eeg_data))

# # Save the DataFrame as a .csv file
# df.to_csv('sample.csv', index=False)


# %%
data = pd.read_csv("sample.csv")

# %%
data.head()

# %%
data.tail()

# %%
data.shape

# %% [markdown]
# This dataset is a pre processed form of Electroencephalogram (EEG)
signals specifically collected for the sake of predicting Epileptic
seizures. This dataset contains 11500 samples and 180 features, each
feature representing the EEG signal data.
```

```
# %% [markdown]
# ROWS: 11500
# COLUMNS: 180

# %% [markdown]
# The last column contains the category of the 178-dimensional input
vector. y can take values: {1, 2, 3, 4, 5}
#
# 5 - eyes open, means when they were recording the EEG signal of the
brain the patient had their eyes open
# 4 - eyes closed, means when they were recording the EEG signal the
patient had their eyes closed
# 3 - Yes they identify where the region of the tumor was in the brain and
recording the EEG activity from the healthy brain area
# 2 - They recorder the EEG from the area where the tumor was located
# 1 - Recording of seizure activity All subjects falling in classes 2, 3,
4, and 5 are subjects who did not have epileptic seizure. Only subjects in
class 1 have epileptic seizure.

# %%
data['y'].value_counts()

# %%
data.y.hist()
```



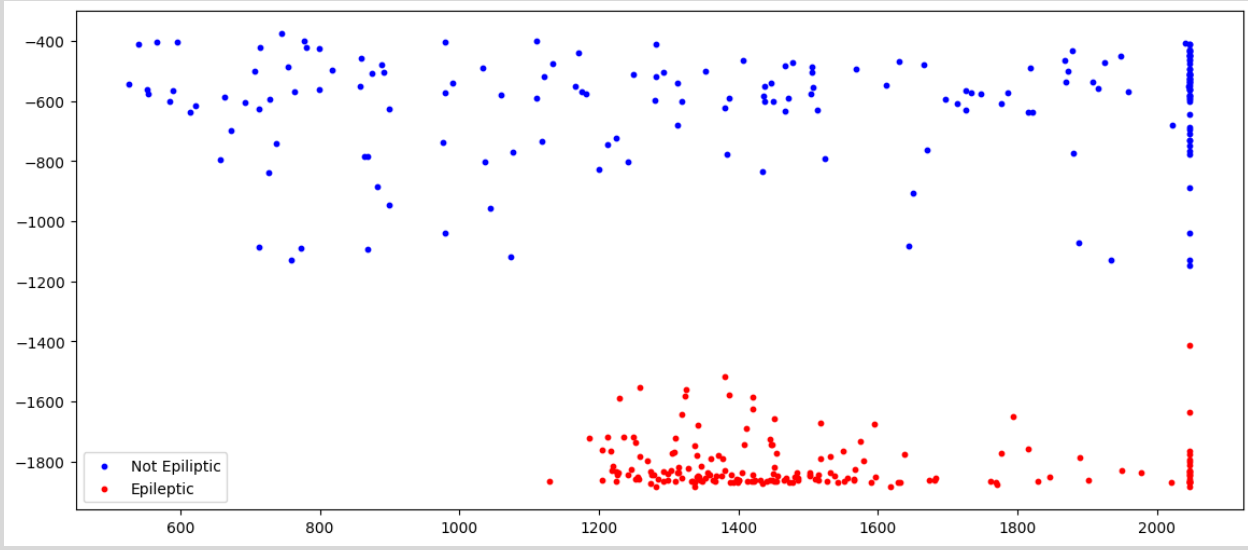
```
# %%
plt.figure(figsize=(50,4))
plt.subplot(131)
[plt.plot(data.values[i][1:-1]) for i in range(23)];
```



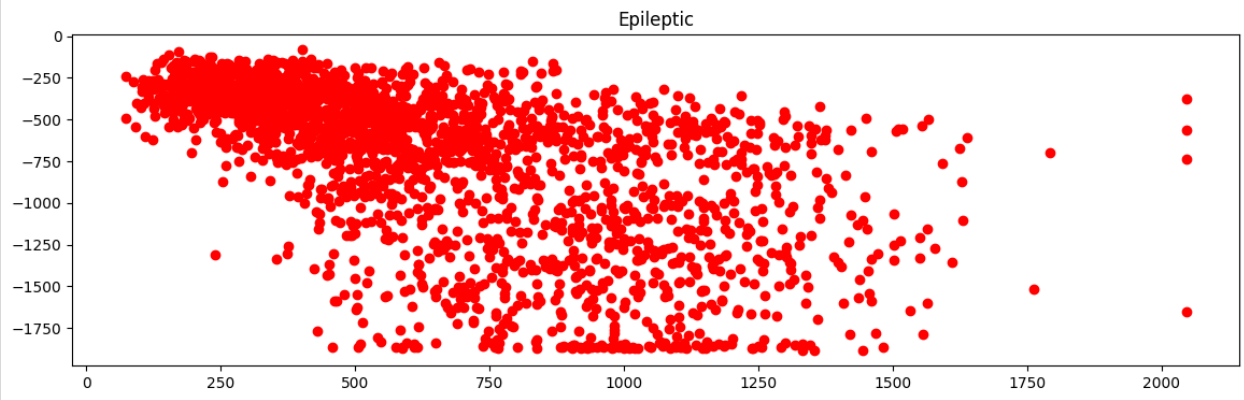

```
# %% [markdown]
# As we can see, samples with y label {1} have seizures whereas all other
{2,3,4,5} do not have seizures. So this multi-classification task can be
converted to a binary classification task. The first part will cover
binary classification. Later we'll train the models for
multi-classification.

# %%
dic = {5: 0, 4: 0, 3: 0, 2: 0, 1: 1}
data['y'] = data['y'].map(dic)
#Converting this to a binary classification task

# %%
print(data['y'].value_counts())

data.head()

# %%
data = data.drop('Unnamed', axis = 1)

# %%
data = shuffle(data)

# %%
data.describe()
```

```
# %%
data.info()

# %%
print('Number of records of Non Epileptic {0} VS Epilepttic
{1}'.format(len(data[data['y'] == 0]), len(data[data['y'] == 1])))

# %%
#Description of Non Epileptic
data[data['y'] == 0].describe().T

# %%
#Description of Epileptic

data[data['y'] == 1].describe().T

# %%
print('Mean VALUE for Epiletic: {}'.format((data[data['y'] ==
1].describe().mean()).mean()))

print('Std VALUE for Epiletic: {}'.format((data[data['y'] ==
1].describe().std()).std()))

print('Totall Mean VALUE for NON Epiletic: {}'.format((data[data['y'] ==
0].describe().mean()).mean()))

print('Totall Std VALUE for NON Epiletic: {}'.format((data[data['y'] ==
0].describe().std()).std()))

# %%
#lists of arrays containing all data without y column
not_epileptic = [data[data['y']==0].iloc[:, range(0,
len(data.columns)-1)].values]
epileptic = [data[data['y']==1].iloc[:, range(0,
len(data.columns)-1)].values]

#We will create and calculate 2d indicators in order plot data in 2
dimensions;

def indic(data):
```

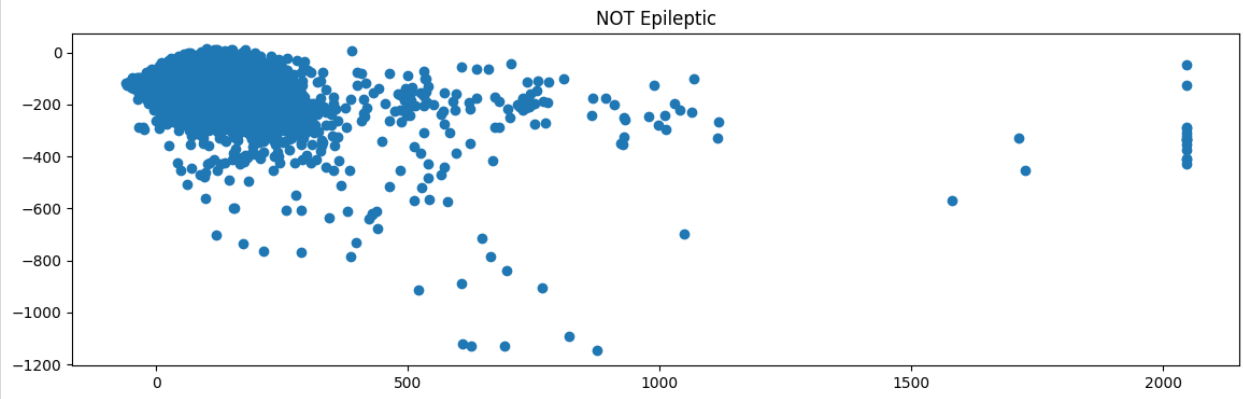
```
    """Indicators can be different. In our case we use just min and max
values
    Additionally, it can be mean and std or another combination of
indicators"""
    max = np.max(data, axis=1)
    min = np.min(data, axis=1)
    return max, min

x1,y1 = indic(not_epileptic)
x2,y2 = indic(epileptic)

fig = plt.figure(figsize=(14,6))
ax1 = fig.add_subplot(111)

ax1.scatter(x1, y1, s=10, c='b', label='Not Epiliptic')
ax1.scatter(x2, y2, s=10, c='r', label='Epileptic')
plt.legend(loc='lower left');
plt.show()
```




```
# %%
#Epileptic
x,y = indic(data[data['y']==1].iloc[:, range(0,
len(data.columns)-1)].values)
plt.figure(figsize=(14,4))
plt.title('Epileptic')
plt.scatter(x, y, c='r');
```




```
# %%
#Not Epileptic
x,y = indic(data[data['y']==0].iloc[:, range(0,
len(data.columns)-1)].values)
plt.figure(figsize=(14,4))
plt.title('NOT Epileptic')
plt.scatter(x, y);
```




```
# %%
print('Number of records of Non Epileptic {0} VS Epilepttic
{1}'.format(len(data[data['y'] == 0]), len(data[data['y'] == 1])))

# %% [markdown]
# ### Removing Outliers

# %%
X = data.drop('y', axis=1)
y = data['y']
df = pd.DataFrame(normalize(X))
```

```
# Initialize the counters for detected and managed outliers
detected_outliers = 0
managed_outliers = 0

# Loop through each of the 178 explanatory variables and calculate the IQR
and bounds
for col in df.columns[:-1]:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify any data points that fall outside the bounds and either
remove or adjust them
    outliers = (df[col] < lower_bound) | (df[col] > upper_bound)
    if outliers.any():
        detected_outliers += outliers.sum()


        df.loc[outliers, col] = np.nanmedian(df[col])
        managed_outliers += outliers.sum()

print(f"Detected {detected_outliers} outliers and managed
{managed_outliers} outliers.")

# %% [markdown]
# ### Eliminating Imbalance

# %% [markdown]
# As we can see, this dataset is quite imbalanced, so we'll need to
perform balancing techniques. This can be done using the imblearn library
in Python that can create synthetic samples in order to balance out the
data and thus provide us with more accurate results.

# %%
# define oversampling strategy
```

```
oversample =
imblearn.over_sampling.RandomOverSampler(sampling_strategy='minority')
# fit and apply the transform
X, y = oversample.fit_resample(data.drop('y', axis=1), data['y'])

X.shape, y.shape

# %%
print('Number of records of Non Epileptic {0} VS Epilepttic
{1}'.format(len(y == True), len(y == False)))

# %% [markdown]
# Now the data is balanced as number of records for epileptic as well as
non epileptic are equal (18400) Before it was 2300 by 9200

# %% [markdown]
# ### Normalizing

# %%
# X = data.drop('y', axis=1)
# y = data['y']

normalized_df = pd.DataFrame(normalize(X))
normalized_df



# %%
normalized_df['y'] = y

print('Normalized Totall Mean VALUE for Epiletic:
{}'.format((normalized_df[normalized_df['y'] ==
1].describe().mean()).mean()))
print('Normalized Totall Std VALUE for Epiletic:
{}'.format((normalized_df[normalized_df['y'] ==
1].describe().std()).std()))

print('Normalized Totall Mean VALUE for NOT Epiletic:
{}'.format((normalized_df[normalized_df['y'] ==
0].describe().mean()).mean()))
```

```
print('Normalized Totall Std VALUE for NOT Epiletic:
{}'.format((normalized_df[normalized_df['y'] ==
0].describe().std()).std()))

# %% [markdown]
# #### Splitting data

# %%
X = normalized_df.drop('y', axis=1)
y = normalized_df['y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,
random_state=42)

# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

# %%
#Check the shapes after splitting
he = X_train, X_test, y_train, y_test
[arr.shape for arr in he]

# %% [markdown]
# ### Testing on Models

# %% [markdown]
# 1. Decision Tree

# %%
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


dtree_model = DecisionTreeClassifier(max_depth =
None,min_samples_split=2,criterion='entropy').fit(X_train,
y_train.values.ravel())
```

```
dtree_predictions = dtree_model.predict(X_test)

#accuracy = DecisionTreeClassifier.score(X_test, y_test)
#print(accuracy)

# creating a confusion matrix
cm = confusion_matrix(y_test, dtree_predictions)
cm
print(classification_report(y_test,dtree_predictions))

accuracy = accuracy_score(y_test, dtree_predictions)
print("Accuracy:", accuracy)

# %% [markdown]
# 2. KNN classifier

# %%
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3).fit(X_train,
y_train.values.ravel())

knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
cm
print(classification_report(y_test,knn_predictions))
accuracy_knn = accuracy_score(y_test, knn_predictions)
print(accuracy_knn)

# %% [markdown]
# 3. Naive Bayes Classifier

# %%
# training a Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB(var_smoothing=1e-09).fit(X_train, y_train.values.ravel())
gnb_predictions = gnb.predict(X_test)

# accuracy on X_test
accuracy = gnb.score(X_test, y_test)
print(accuracy)
```

```
# creating a confusion matrix
cm = confusion_matrix(y_test, gnb_predictions)
print('Accuracy: ${cm}')
print(classification_report(y_test,gnb_predictions))

# %% [markdown]
# 4. Random Forest Classifier

# %%
from sklearn.ensemble import RandomForestClassifier
import sklearn.metrics as metrics
clf4 = RandomForestClassifier(random_state=42,n_estimators=10)

clf4.fit(X_train, y_train.values.ravel())
y_pred4 = clf4.predict(X_test)
Acc2 = metrics.accuracy_score(y_test,y_pred4)
print(Acc2)
cm = confusion_matrix(y_test, y_pred4)
print('Accuracy: ${cm}')
print(classification_report(y_test,y_pred4))

# %%
from sklearn.svm import SVC

# %%
model = SVC(kernel= 'rbf')
model.fit(X_train, y_train)

# %%
predictions = model.predict(X_test)

# %%
print("Accuracy:",metrics.accuracy_score(y_test, predictions))
print("Precision:",metrics.precision_score(y_test, predictions))
print("Recall:",metrics.recall_score(y_test, predictions))

# %% [markdown]
# ### Creating an Artificial Neural Network
```

```
# %%
import tensorflow as tf
from keras.models import Sequential
from keras.utils import np_utils
from keras.layers import Dense, Activation

# %% [markdown]
# Defining the model

# %%
model = Sequential()

# %%
# Add a dense layer with 10 neurons and ReLU activation function as the
input layer
model.add(Dense(256, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(128, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(128, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(64, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(32, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(32, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(32, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(16, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(16, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(8, input_dim=X_train.shape[1], activation='LeakyReLU'))
model.add(Dense(5, input_dim=X_train.shape[1], activation='LeakyReLU'))

# Add a dense layer with 1 neuron and sigmoid activation function as the
output layer
model.add(Dense(1, activation='softmax'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='SGD',
metrics=['accuracy'])

# Train the model on the training data
model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1)

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
```

```
print("Test loss:", test_loss)
print("Test accuracy:", test_accuracy)

# %%
model.summary()

# %% [markdown]
# ### Parallel Implementation of Support Vector Machine

# %% [markdown]
# This parallel implementation will be done using the JobLIB library in
python and sklearn for SVM. This will speed up the training of the model.

# %%
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from joblib import Parallel, delayed

# %%
def train_svm(X_train_subset, y_train_subset, X_test, y_test):
    clf = SVC(kernel='rbf')
    clf.fit(X_train_subset, y_train_subset)
    y_pred = clf.predict(X_test)
    return accuracy_score(y_test, y_pred)

# %%
n_cores = 2  # number of CPU cores to use

accuracies = Parallel(n_jobs=n_cores)(
    delayed(train_svm)(X_train_subset, y_train_subset, X_test, y_test)
    for X_train_subset, y_train_subset in zip(np.array_split(X_train,
n_cores), np.array_split(y_train, n_cores))
)

# %%
mean_accuracy = np.mean(accuracies)
print(f"Mean accuracy: {mean_accuracy}")
```

```
# %% [markdown]
# ### ANN

# %%
from keras.models import Sequential
from keras.layers import Dense, Dropout

num_features = X_train.shape[1]
# Define the model
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(num_features,)))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=500, batch_size=32,
validation_data=(X_val, y_val))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print('Test accuracy:', accuracy)



# %% [markdown]
# ## Recurrent Neural Network

# %%
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN

# Define hyperparameters
seq_length = 10
```

```
input_dim = 32
hidden_units = 64

# Generate some sample data
x_train = np.random.rand(100, seq_length, input_dim)
y_train = np.random.randint(0, 2, size=(100,))

# Define RNN model
model = Sequential()
model.add(SimpleRNN(units=hidden_units, input_shape=(seq_length,
input_dim)))
model.add(Dense(units=1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train model
model.fit(x_train, y_train, epochs=10, batch_size=32)

# Generate some sample test data
x_test = np.random.rand(50, seq_length, input_dim)
y_test = np.random.randint(0, 2, size=(50,))

# Evaluate model on test data
loss, accuracy = model.evaluate(x_test, y_test)
print(accuracy)


# %%
X_train.shape, X_test.shape

# %%
y_train.shape

# %% [markdown]
# ### Defining a Convolutional Network

# %%
def evaluate_model(history,X_test,y_test,model):
```

```
    scores = model.evaluate((X_test),y_test, verbose=0)
    print("Accuracy: %.2f%%" % (scores[1]*100))

    print(history)
    fig1, ax_acc = plt.subplots()
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Model - Accuracy')
    plt.legend(['Training', 'Validation'], loc='lower right')
    plt.show()

    fig2, ax_loss = plt.subplots()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Model - Loss')
    plt.legend(['Training', 'Validation'], loc='upper right')
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.show()
    target_names=['1','2', '3']

    y_true=[]
    for element in y_test:
        y_true.append(np.argmax(element))
    prediction_proba=model.predict(X_test)
    prediction=np.argmax(prediction_proba,axis=1)
    cnf_matrix = confusion_matrix(y_true, prediction)

# %%
def network_CNN(X_train,y_train):
    im_shape=(X_train.shape[1],1)
    inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
    conv1d_1 = layers.Conv1D(filters=32, kernel_size=6)(inputs_cnn)
    batch_normalization = BatchNormalization()(conv1d_1)
    max_pooling1d = layers.MaxPooling1D( 2,
padding='same')(batch_normalization)
    conv1d_2 = layers.Conv1D(filters=64, kernel_size=3)(max_pooling1d)
    batch_normalization_1 = BatchNormalization()(conv1d_2)
```

```
    max_pooling1d_1 = layers.MaxPooling1D(2,
padding='same')(batch_normalization_1)
    flatten = Flatten()(max_pooling1d_1)
    dense = Dense(32)(flatten)
    dense_1 = Dense(16)(dense)
    main_output = Dense(2)(dense_1)
    model1 = Model(inputs= inputs_cnn, outputs=main_output)
    model1.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',metrics = ['accuracy'])

    return(model1)

# %%
model1 = network_CNN(X_train,y_train)
print(model1.summary())

# %%
save_path = './tmp/checkpoint_1'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath=save_path,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

history = model1.fit(X_train, y_train,epochs=100, batch_size=32,
validation_data=(X_val,y_val), callbacks=[model_checkpoint_callback])

# %%
X_train.shape , y_train.shape

# %%
evaluate_model(history,X_test,y_test,model1)
y_pred=model1.predict(X_test)

# %%
model1.load_weights(save_path)
evaluate_model(history,X_test,y_test,model1)

# %% [markdown]
```

```
# ### Bidirectional LSTM

# %%
def network_LSTM(X_train,y_train):
    im_shape=(X_train.shape[1],1)
    inputs_lstm=Input(shape=(im_shape), name='inputs_lstm')

    dense = Dense(units=32, activation='relu', name='dense')(inputs_lstm)
    lstm = layers.Bidirectional(LSTM(units=128, name='lstm'))(dense)
    dropout = Dropout(0.3)(lstm)
    batch_normalization =
BatchNormalization(name='batch_normalization')(dropout)
    dense_1 = Dense(units=64, activation='relu',
name='dense_1')(batch_normalization)
    dropout_2 = Dropout(0.3, name='dropout_2')(dense_1)
    batch_normalization_1 =
BatchNormalization(name='batch_normalization_1')(dropout_2)
    main_output = Dense(units=2,
activation='softmax')(batch_normalization_1)

    model = Model(inputs= inputs_lstm, outputs=main_output)
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',metrics = ['accuracy'])

    return(model)

# %%
model2 = network_LSTM(X_train, y_train)
print(model2.summary())

# %%
# Training Bidirectional LSTM model
# Saving model at highest validation accuracy
save_path = './tmp/checkpoint_2'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath=save_path,
    save_weights_only=True,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)
```

```
history2 = model2.fit(X_train, y_train,epochs=50, batch_size=32,
validation_data=(X_val,y_val), callbacks=[model_checkpoint_callback])

# %%
evaluate_model(history2,X_test,y_test,model2)

# %%
model2.load_weights(save_path)
evaluate_model(history2,X_test,y_test,model2)
```

**CONTRIBUTIONS:**

1. Rushil Shivade
2. Chirag Kathoye

**OUTPUT:**

On a comparative study of these models, we get the following results:

| Model | Accuracy | Error |
|---|---|---|
| 1. Decision Tree | 87.71 % | 12.29 % |
| 2. KNN Classifier | 91.83 % | 8.17 % |
| 3. Naive Bayes | 60.52 % | 39.48 % |
| 4. Random Forest | 94.14 % | 5.86 % |
| 5. SVM | 86.64 % | 13.36 % |
| 6. ANN | 49.58 % | 50.42 % |
| 7. RNN | 43.99 % | 56.01 % |
| 8. LSTM | 96.85 % | 3.15 % |
| 9. CNN | 53.42 % | 46.58 % |

1. Decision Tree

```
[ ]
...                precision    recall  f1-score   support

             0       0.94      0.80      0.86      2754
             1       0.82      0.95      0.88      2766

      accuracy                           0.87      5520
     macro avg       0.88      0.87      0.87      5520
  weighted avg       0.88      0.87      0.87      5520

  Accuracy: 0.8739130434782608
```

2. KNN

```
[ ]
...                  precision    recall  f1-score   support

               0       0.92      0.88      0.90      2754
               1       0.88      0.93      0.90      2766

        accuracy                           0.90      5520
       macro avg       0.90      0.90      0.90      5520
    weighted avg       0.90      0.90      0.90      5520

    0.9021739130434783
```

3. Naive Bayes

```
[ ]
...  0.5907608695652173
     Accuracy: ${cm}
                  precision    recall  f1-score   support

               0       0.60      0.54      0.57      2754
               1       0.58      0.65      0.61      2766

        accuracy                           0.59      5520
       macro avg       0.59      0.59      0.59      5520
    weighted avg       0.59      0.59      0.59      5520
```

4. Random Forest

```
[ ]
...   0.9380434782608695
      Accuracy: ${cm}
                    precision    recall  f1-score   support

                 0       0.94      0.94      0.94      2754
                 1       0.94      0.94      0.94      2766

          accuracy                           0.94      5520
         macro avg       0.94      0.94      0.94      5520
      weighted avg       0.94      0.94      0.94      5520
```

## 5. SVM

```
    mean_accuracy = np.mean(accuracies)
    print(f"Mean accuracy: {mean_accuracy}")
[ ]
...   Mean accuracy: 0.8603260869565217
```

## 6. ANN

```
Epoch 13/500
...
Epoch 500/500
322/322 [==============================] - 1s 2ms/step - loss: 0.1931 - accuracy: 0.9236 - val_loss: 0.3889 - val_accuracy: 0.8901
173/173 [==============================] - 0s 1ms/step - loss: 0.4431 - accuracy: 0.8842
Test accuracy: 0.884239137172699
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

## 7. RNN

```
Epoch 10/10
4/4 [==============================] -
2/2 [==============================] -
0.5
```

8. CNN

```
model1.load_weights(save_path)
evaluate_model(history,X_test,y_test,model1)
```

```
Accuracy: 53.42%
<keras.callbacks.History object at 0x000001EC52AA1AB0>
```

Model - Accuracy

9. LSTM

```
model2.load_weights(save_path)
evaluate_model(history2,X_test,y_test,model2)
```

```
Accuracy: 96.85%
<keras.callbacks.History object at 0x000001EC5CA24640>
```

Model - Accuracy

**GRAPHS:**

Graphs of Accuracy and Loss for LSTM (giving highest accuracy)