Epileptic Seizure Detection with EEG Signals

Sandra Anna Joshy

Epileptic seizure is a disease that affects a large number of people and it is caused by abnormal electrical activity in the brain [1]. Most seizures are very brief and rarely life threatening but critical seizures have to be recognized immediately.Also, according to studies[6], it is possible to control the attacks with early diagnosis.In addition Epileptic seizure patients require special treatments and continuous observation for the disease symptoms. Therefore several methods for epileptic seizure recognition have been developed in the past for early diagnosis. One of them is recognition with EEG signals. Even though EEG signals are a strong tool for epileptic seizure recognition, visual inspect for discriminating EEGs is a time consuming and high costly process [2].Therefore this project is aimed to make epileptic seizure detection using EEG signals that are transformed by wavelet transform, with machine learning and deep learning techniques.

Table of Content

Problem

Epilepsy is a group of neurological disorders characterized by recurrent epileptic seizures.[3]This seizures can cause severe shaking attacks therefore in these seizures patients can hurt her/hisself very badly.Luckily epilepsy is a curable diasease that 70% of cases are controlable with early diagnosis and medication[4][5].Therefore in order to make an early diagnosis, many epilepsy diagnostic methods have been developed over time. One of them is diagnosis with the help of EEG signals.However the visual inspect process for discriminating EEGs is a time consuming and high costly process.Therefore the project is aimed to make epileptic seizure detection very quickly and accurately by using EEG signal of the brain data with machine learning and deep learning techniques.

Data Understanding

For this task,I will use Bonn University's restructured EEG time series dataset that consists of 5 different target classes , 179 attributes and it contains 11500 samples.The classes 2, 3, 4, and 5 are subjects who did not have epileptic seizure. Only subjects in class 1 have epileptic seizure.I provied more details about dataset below.

Data Set Characteristics:	Multivariate, Time-Series
Number of Instances:	11500
Attribute Characteristics:	Integer, Real
Number of Attributes:	179

Target variable	Content	Epileptic seizure
1	Recording of seizure activity	YES
2	They recorder the EEG from the area where the tumor was located	NO
3	Yes they identify where the region of the tumor was in the brain and recording the EEG activity from the healthy brain area	NO
4	eyes closed, means when they were recording the EEG signal the patient had their eyes closed	NO
5	eyes open, means when they were recording the EEG signal of the brain the patient had their eyes open	NO

detection

```
from google.colab import files
uploaded = files.upload() #choose data.csv file from provided files
```

```
Choose files data.csv
• data.csv(text/csv) - 7635689 bytes, last modified: 07/04/2024 - 100% done
Saving data.csv to data.csv
```

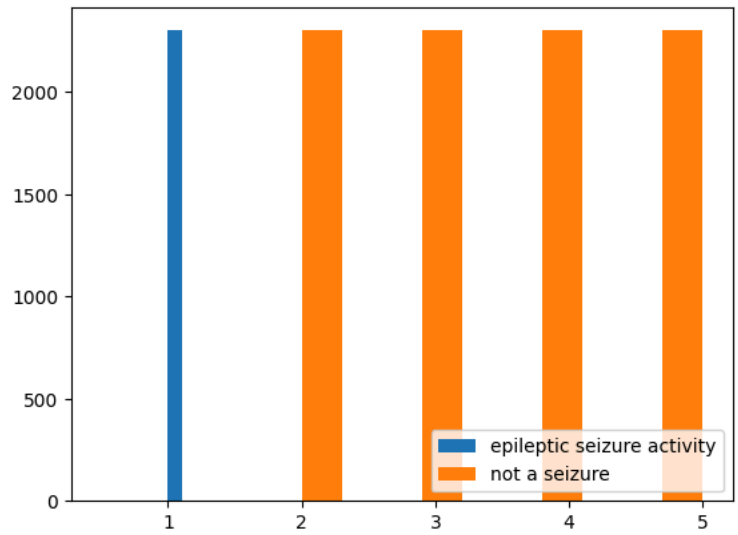
```
import pandas as pd
import numpy as np
df=pd.read_csv("data.csv")
df.head()
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12

```
print("General info about colums,rows etc.")
df.info()
print("\nTarget variables value counts\n",df["y"].value_counts())
```

```
General info about colums,rows etc.
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11500 entries, 0 to 11499
Columns: 180 entries, Unnamed: 0 to y
dtypes: int64(179), object(1)
memory usage: 15.8+ MB

Target variables value counts
 y
4    2300
1    2300
5    2300
2    2300
3    2300
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt
def hist(df,plt):
  plt.hist(df[df["y"]==1]["y"],label="epileptic seizure activity")
  plt.hist(df[df["y"]!=1]["y"],label="not a seizure")
  plt.legend(loc='lower right')
  plt.show()

hist(df,plt)
```



All class have same amount of instances and dataset looks balance however i will transform target variable into seizure or not so dataset will no longer be balanced.

Data Preparation

In data preparation process, firstly i cleaned the data from unnecessary columns then I transformed the target variable into binary classes (1 for seizure, 0 for not seizure).Then I created new variables with some mathematical transformations that called Hurst exponent and Discrete wavelet transformation (do not worry, I will explain all these transformations below.). Then i normalize data.After i restructured the unbalanced data into more balanced data.Lastly I inspected whether feature selection improve accuracies.

Cleaning the data and transforming the target variable

```
df.head(2) #just a quick look to data again. x1...x178 columns are a part of our time-series data (EEG Signals) but Unnamed:
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X1
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	1

```
df["Unnamed: 0"].value_counts #As you can see this column is exlusive for all instance
                              #So it means, the column has no effect on classification, it is unnecessary
```

```
pandas.core.base.IndexOpsMixin.value_counts
def value_counts(normalize: bool=False, sort: bool=True, ascending:
bool=False, bins=None, dropna: bool=True) -> Series

/usr/local/lib/python3.10/dist-packages/pandas/core/base.py
Return a Series containing counts of unique values.

The resulting object will be in descending order so that the
first element is the most frequently-occurring element.
Excludes NA values by default.
```

```
df["y"].value_counts() # I will transform 2,3,4,5 classes to 0, 1 class to 1
```

```
y
4    2300
1    2300
5    2300
2    2300
3    2300
Name: count, dtype: int64
```

```
#This method drop the unnecessary column (Unnamed: 0) and transform the target variable
def prepareData(df):
  df["y"]=[1 if df["y"][i]==1 else 0 for i in range(len(df["y"]))]
  target=df["y"]
  df_copy=df.drop(["Unnamed: 0","y"],axis=1)
  return df_copy,target

df_copy,target=prepareData(df)
```

Data transformation with hurst exponent and discrete wavelet transform

Before i started coding the project, I had some literature review.According to my review, I decided on not using raw time series data but using transformed data.I created new features with these transformations called hurst exponent and discrete wavelet transformation.

Hurst Exponent : The Hurst exponent is used as a measure of long-term memory of time series. It relates to the autocorrelations of the time series, and the rate at which these decrease as the lag between pairs of values increases[7].Basically the main reason why I use this variable is that I think there may be certain fluctuation differences between the classes of time series data.

Hurst exponent formula:

$$\mathbb{E}\left[\frac{R(n)}{S(n)}\right] = Cn^H \text{ as } n \to \infty,$$

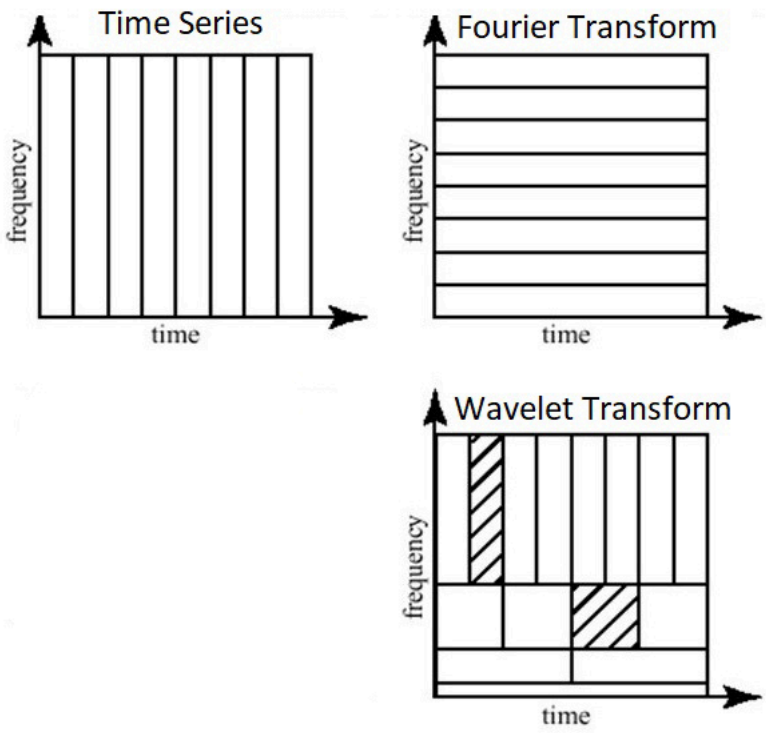
Discrete Wavelet Transform :

Discrete Wavelet Transform (DWT) is a kind of transform that any wavelet transform for which the wavelets are discretely sampled[8].(Not an open explanation right? What is a wavelet transform anyway?).

Note: I used db4 type DWT which is very popular in literature, for this task and also provided a table that shows which types are used in literature.(see figure 2)

Wavelet Transform :

Actually if you already familiar with Fourier Transform, Wavelet transform can be explained more concretely.In Fourier Transform, the transform can understand which frequencies (frequency domain) exist in the signal without any time information(time domain) but Wavelet transform can give more information about both time and frequencies domains.Wavelet transform give more information on frequency domain on small frequencies.On the other hand it give more information about time domain in large frequency values[9].



(Picture is modified version of given link picture taken from [10]]

(For more detail about wavelet transform, you can also check [9] reference.)

Table 4. Summary of the research in terms of the type of wavelet used.

Wavelet	DWT	CWT	Total
db4	19	0	19
db/=4	7	1	8
Morlet	0	7	7
ocs	5	1	6
bior	0	4	4
nr	4	1	5
MH	0	3	3
Morse	0	2	2
coif	1	0	1
Haar	1	0	1
Biphasic (bip)	1	0	1
rbio6.8	1	0	1
Symlet 5	0	1	1
CG	0	1	1
ocs	0	1	1

(Table 2, This table taken from [13] research paper)

```
pip install hurst #installing hurst module to get hurst values
```

```
Collecting hurst
  Downloading hurst-0.0.5-py3-none-any.whl (5.9 kB)
Requirement already satisfied: pandas>=0.18 in /usr/local/lib/python3.10/dist-packages (from hurst) (2.0.3)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/dist-packages (from hurst) (1.25.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->hur
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->hurst) (2023.
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.18->hurst) (202
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>
Installing collected packages: hurst
Successfully installed hurst-0.0.5
```

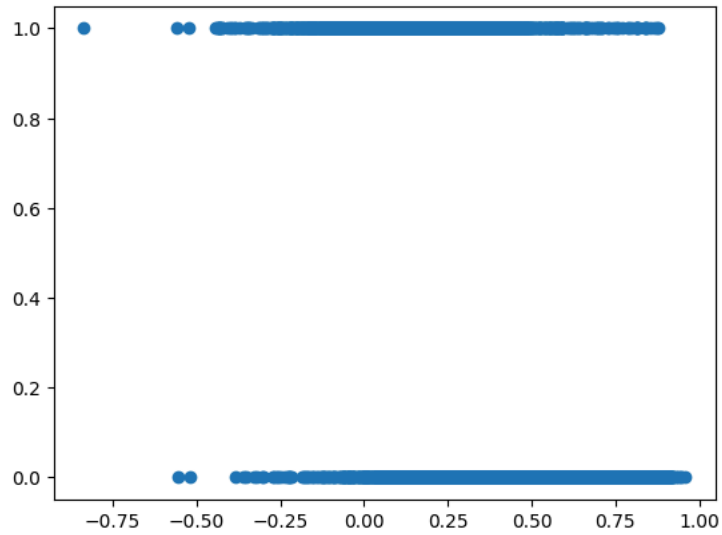
```
import pywt #importing pywt for getting wavelet transform features
from hurst import compute_Hc
```

```
def getHurst(df_copy):
  df_copy["hurst_ex"]=[compute_Hc(df_copy.iloc[i], kind="change", simplified=True)[0] for i in range(len(df_copy))]
  df_copy["hurst_c"]=[compute_Hc(df_copy.iloc[i], kind="change", simplified=True)[1] for i in range(len(df_copy))]
  return df_copy
```

```
def getStatsForHurst(df_copy):
  plt.scatter(df_copy["hurst_ex"],target)
  print("mean value of hurst exponent for class 1:",np.mean(df_copy.iloc[target[target==1].index]["hurst_ex"]))
  print("mean value of hurst exponent for class 0:",np.mean(df_copy.iloc[target[target==0].index]["hurst_ex"]))
  print("mean value of hurst constant for class 1:",np.mean(df_copy.iloc[target[target==1].index]["hurst_c"]))
  print("mean value of hurst constant for class 0:",np.mean(df_copy.iloc[target[target==0].index]["hurst_c"]))
  print("median value of hurst exponent for class 1:",np.median(df_copy.iloc[target[target==1].index]["hurst_ex"]))
  print("median value of hurst exponent for class 0:",np.median(df_copy.iloc[target[target==0].index]["hurst_ex"]))
  print("median value of hurst constant for class 1:",np.median(df_copy.iloc[target[target==1].index]["hurst_c"]))
  print("median value of hurst constant for class 0:",np.median(df_copy.iloc[target[target==0].index]["hurst_c"]))
```

```
#These methods create a new dataset with wavelet transform
#In getWaveletFeatures method, i get a group of wavelet coeffient and hurst exponent and the constant for all instance
#give these values to statisticsForWavelet function to get coeffients quartiles,mean,median,standart deviation,variance,root
#Lastly createDfWavelet method give all these values and return a new dataframe
def getWaveletFeatures(data,target):
    list_features = []
    for signal in range(len(data)):
        list_coeff = pywt.wavedec(data.iloc[signal], "db4")
        features = []
        features.append(data.iloc[signal]["hurst_ex"])
        features.append(data.iloc[signal]["hurst_c"])
        for coeff in list_coeff:
            features += statisticsForWavelet(coeff)
        list_features.append(features)
    return createDfWavelet(list_features,target)
#This method taken from [9]
def statisticsForWavelet(coefs):
    n5 = np.nanpercentile(coefs, 5)
    n25 = np.nanpercentile(coefs, 25)
    n75 = np.nanpercentile(coefs, 75)
    n95 = np.nanpercentile(coefs, 95)
    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 [n5, n25, n75, n95, median, mean, std, var, rms]

def createDfWavelet(data,target):
  for i in range(len(data)):
    data[i].append(target[i])
  return pd.DataFrame(data)


df_copy=getHurst(df_copy)
getStatsForHurst(df_copy)
```

```
mean value of hurst exponent for class 1: 0.17426759520249877
mean value of hurst exponent for class 0: 0.5044276844723696
mean value of hurst constant for class 1: 22.53775958904113
mean value of hurst constant for class 0: 9.765791813075161
median value of hurst exponent for class 1: 0.16788038442091016
median value of hurst exponent for class 0: 0.5082046350724865
median value of hurst constant for class 1: 10.294529749768916
median value of hurst constant for class 0: 5.2513747999821
```




```
df_copy_fea=getWaveletFeatures(df_copy,target)


df_copy_fea.head()#our new dataset is ready
```

	0	1	2	3	4	5	6
0	0.546585	2.601079	-305.176164	-151.216527	723.449162	776.917001	-3.295061
1	-0.254452	75.697950	-1242.993850	-488.612159	1410.559480	1774.820173	603.491844
2	0.742152	3.615098	-309.103358	-210.090841	-117.128242	-3.579475	-153.466544
3	0.767329	12.232493	-392.667105	-384.158663	-274.167936	-168.574097	-295.871642
4	0.647870	2.538971	-276.585954	-255.283369	5.788142	87.801277	-79.855665

5 rows × 48 columns

Resampling data

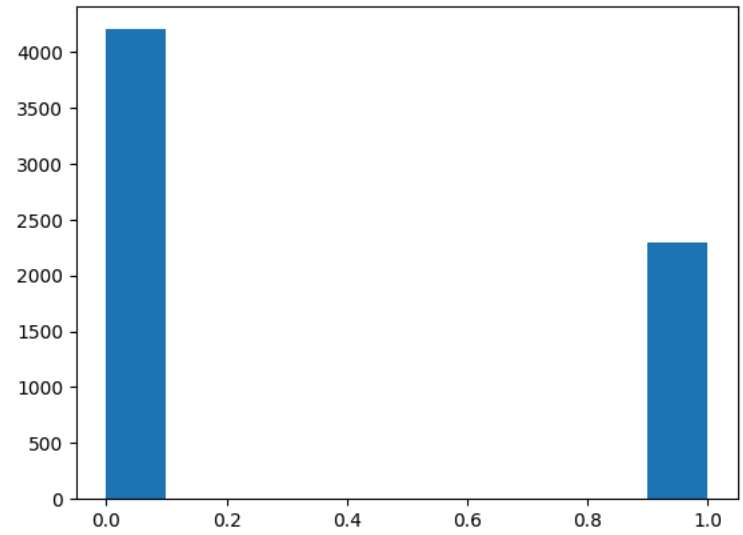
Before the resampling, the 0 class was too prominent.I tried to balance the data by shuffling and resampling the data, In order get a more balanced dataset for machine learning models.

```
from sklearn.utils import shuffle
def createBalancedDataset(data,random_state):
  #shuffling for random sampling
  X = shuffle(data,random_state=random_state)
  #getting first 6500 value
  return X.sort_values(by=47, ascending=False).iloc[:6500].index
```

```
v=createBalancedDataset(df_copy_fea,42)
```

```
plt.hist((df_copy_fea.iloc[v])[47])
(df_copy_fea.iloc[v][47]).value_counts() #more balanced dataset
```

```
47
0    4200
1    2300
Name: count, dtype: int64
```



Normalizing Data

Before the feature selection I normalized data.Normalizing data is necessary for Anova test.Also normalizing is a prerequisite for a lot of machine learning algorithms[11].

```
#normalizing dataset
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df_copy_fea.drop([47],axis=1))
n_df_fea=pd.DataFrame(scaler.transform(df_copy_fea.drop([47],axis=1)))
```

Feature Selection with Anova and Random Forest

Before giving inputs to models.I inspected whether feature selection improve the accuracies.In order to understand this situation, first i gave all features to model and then i gave only the selected features as input to model and i compared the accuries and f-scores.

For feature selection, I got help by below diagram.Since the dataset's input values are numeric but target variable is categoric, I choosed the Anova test for feature selection.Furthermore i did one more feature selection with random forest in order to compare them together.Eventhough feature selection can be very helpful on improving accuracies in many case, In this situation, it did not help so I decided to continue with all features as input.

```
#First the all features are given as a input to SVM model (I will explain why i used SVM later)


import numpy as np
from sklearn.model_selection import train_test_split
X_trainr, X_testr, y_trainr, y_testr = train_test_split(n_df_fea.iloc[v], target.iloc[v], test_size=0.33, random_state=42)


#Reference point for svm
from sklearn import svm
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report

#I will explain this model in model part in the notebook
clf = svm.SVC(kernel="linear")
clf.fit(X_trainr, y_trainr)
#cross validation is 10
y_pred = cross_val_predict(clf,X_testr,y_testr,cv=10)
print("All features are inclueded\n",classification_report(y_testr, y_pred))
```

```
     All features are inclueded
                   precision    recall  f1-score   support

                0       0.96      0.99      0.97      1373
                1       0.98      0.93      0.95       772

            accuracy                           0.97      2145
           macro avg       0.97      0.96      0.96      2145
        weighted avg       0.97      0.97      0.97      2145
```

Selecting most important 20 features with Anova

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
#Selection most important 20 feature by using Anova test
def selectFeature(X_trainr,y_trainr,X_testr):
  sel_f = SelectKBest(f_classif, k=20)
  X_train_f = sel_f.fit_transform(X_trainr, y_trainr)
  mySelectedFeatures=[i for i in range(len(sel_f.get_support())) if sel_f.get_support()[i]==True]
  j=0
  unseable_columns=[]
  #Creating a new dataset with these 20 features
  for i in X_trainr.columns:
    if(j not in mySelectedFeatures):
      unseable_columns.append(i)
    j+=1
  X_train_arranged=X_trainr.drop(columns=unseable_columns)
  X_test_arranged=X_testr.drop(columns=unseable_columns)
  return  X_train_arranged,X_test_arranged

X_train_arranged,X_test_arranged=selectFeature(X_trainr,y_trainr,X_testr)


X_train_arranged.columns #The most important columns according to Anova
```

```
     Index([2, 5, 8, 10, 11, 14, 17, 19, 20, 21, 22, 23, 26, 28, 29, 32, 35, 37, 44,
            46],
           dtype='int64')
```

```
#Overall accuracy is decreased
from sklearn import svm
from sklearn.metrics import classification_report
clf = svm.SVC(kernel="linear")
clf.fit(X_train_arranged, y_trainr)
y_pred = cross_val_predict(clf,X_test_arranged,y_testr,cv=10)
print("Only Anova test's Features are used\n",classification_report(y_testr, y_pred))
```

```
     Only Anova test's Features are used
                   precision    recall  f1-score   support

                0       0.96      0.98      0.97      1373
                1       0.97      0.92      0.95       772

            accuracy                           0.96      2145
           macro avg       0.96      0.95      0.96      2145
```

```
weighted avg       0.96      0.96      0.96      2145
```

Selecting most important 20 features with Random Forest

```
#Firstly I used grid Search for getting best hyperparameter for random-forest
np.random.seed(42)
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_jobs=-1,max_features= 'sqrt' ,n_estimators=50, oob_score = True)

param_grid = {
    'max_depth': [2,5],
    'min_samples_split':[2,5,10],
    'n_estimators': [100,150],
    'max_features': ['sqrt', 'log2']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5)
CV_rfc.fit(X_trainr, y_trainr)
print (CV_rfc.best_params_)
```

```
{'max_depth': 5, 'max_features': 'sqrt', 'min_samples_split': 10, 'n_estimators': 100}
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=42,max_depth=5,max_features='sqrt',min_samples_split=2,n_estimators=150)
clf.fit(X_trainr, y_trainr)
#I also get the importance rates and sort in a desending order and create a dataframe
zipped=pd.DataFrame(zip(X_trainr.columns,clf.feature_importances_),columns=["column","importance"]).sort_values(by="importar
y_pred2 = cross_val_predict(clf,X_testr,y_testr,cv=10)
print("All featuares are included\n",classification_report(y_testr, y_pred2))
```

```
All featuares are included
               precision    recall  f1-score   support

            0       0.98      0.98      0.98      1373
            1       0.96      0.96      0.96       772

     accuracy                           0.97      2145
    macro avg       0.97      0.97      0.97      2145
 weighted avg       0.97      0.97      0.97      2145
```

```
zipped.head(20) #The most important 20 values according to random forest
```

	column	importance
18	18	0.122870
17	17	0.102400
8	8	0.086825
9	9	0.083827
19	19	0.082742
36	36	0.075209
35	35	0.065976
14	14	0.047009
11	11	0.045067
26	26	0.036911
29	29	0.035794
32	32	0.032185
20	20	0.024624
28	28	0.017470
10	10	0.016753
27	27	0.015615
23	23	0.012727
37	37	0.012705
2	2	0.011719
13	13	0.009039

Next steps: Generate code with zipped View recommended plots

```
#F-scores are decreased
clf = svm.SVC(kernel="linear")
clf.fit(X_trainr[zipped.iloc[:20].index], y_trainr)
y_pred = cross_val_predict(clf,X_testr[zipped.iloc[:20].index],y_testr,cv=10)
print("Only random forest's features are inclueded\n",classification_report(y_testr, y_pred))
```

```
     Only random forest's features are inclueded
                   precision    recall  f1-score   support

                0       0.96      0.98      0.97      1373
                1       0.96      0.92      0.94       772

         accuracy                           0.96      2145
        macro avg       0.96      0.95      0.96      2145
     weighted avg       0.96      0.96      0.96      2145
```

Models	Accuracy	Macro avg	Weighted avg
SVM with all Features	97	96	97
SVM with Anova Features	96	96	96
SVM with Random Forest Features	96	96	96

As you can see, The feature selection process did not increase accuracies.Therefore I decided to continue with all features as input

Modeling

In the model selection process, I analysed the most used algoriths in the literature.I encounter that the most used algorithms are ANNs/RNNs, SVMs and Desion Trees.Therefore I decided to use SVM since this task is a binary classification problem , Random Forest for desion trees are pretty popular in the literature and lastly LSTM for RNN's are pretty popular in the literature and it is one of my favorites.

```
#SVM, for kernel, I used some kernels and get the most accurate one
clf = svm.SVC(kernel="linear",probability=True)
clf.fit(X_trainr, y_trainr)
#cross validation is 10
y_pred = cross_val_predict(clf,X_testr,y_testr,cv=10)
print("All features are included\n",classification_report(y_testr, y_pred))
```

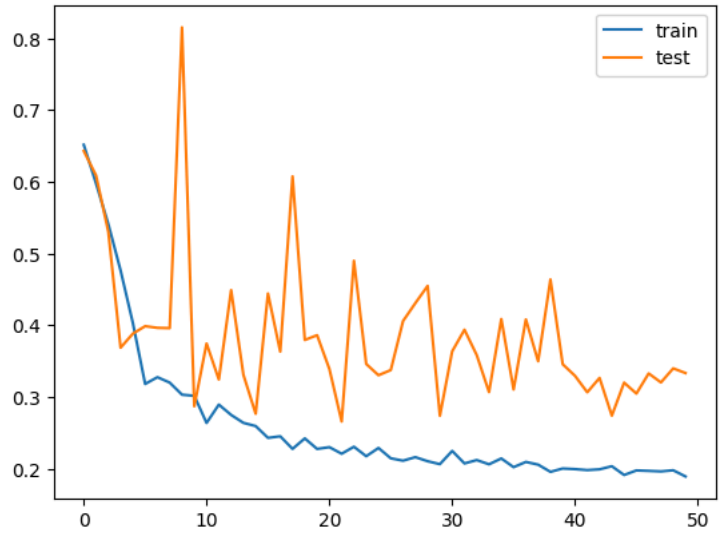
```
     All features are included
                   precision    recall  f1-score   support

                0       0.96      0.99      0.97      1373
                1       0.98      0.93      0.95       772

         accuracy                           0.97      2145
        macro avg       0.97      0.96      0.96      2145
     weighted avg       0.97      0.97      0.97      2145
```

```
#Random forest, I got hyperparameters from above grid-search
clf1 = RandomForestClassifier(random_state=42, max_depth=5, max_features='sqrt', min_samples_split=5, n_estimators=150)
clf1.fit(X_trainr, y_trainr)
y_pred2 = cross_val_predict(clf1,X_testr,y_testr,cv=10)
print("All featuares are included\n",classification_report(y_testr, y_pred2))
```

```
     All featuares are included
                   precision    recall  f1-score   support

                0       0.98      0.98      0.98      1373
                1       0.96      0.96      0.96       772

         accuracy                           0.97      2145
        macro avg       0.97      0.97      0.97      2145
     weighted avg       0.97      0.97      0.97      2145
```

```
#LSTM
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import LSTM
# When i designed the network, Basically i used heuristic approach
X_trainrr=np.array(X_trainr).reshape(X_trainr.shape[0],X_trainr.shape[1],1)
X_testrr=np.array(X_testr).reshape(X_testr.shape[0],X_testr.shape[1],1)
model = Sequential()
model.add(LSTM(50, input_shape=(X_trainrr.shape[1], X_trainrr.shape[2])))
model.add(Dropout(0.1))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
# fit network
history = model.fit(X_trainrr, y_trainr, epochs=50, batch_size=72, validation_data=(X_testrr, y_testr), verbose=2, shuffle=F
# plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```

```
61/61 - 2s - loss: 0.2062 - accuracy: 0.9284 - val_loss: 0.3069 - val_accura
Epoch 35/50
61/61 - 2s - loss: 0.2144 - accuracy: 0.9215 - val_loss: 0.4089 - val_accura
Epoch 36/50
61/61 - 2s - loss: 0.2023 - accuracy: 0.9290 - val_loss: 0.3105 - val_accura
Epoch 37/50
61/61 - 2s - loss: 0.2095 - accuracy: 0.9235 - val_loss: 0.4082 - val_accura
Epoch 38/50
61/61 - 2s - loss: 0.2058 - accuracy: 0.9224 - val_loss: 0.3499 - val_accura
Epoch 39/50
61/61 - 2s - loss: 0.1957 - accuracy: 0.9286 - val_loss: 0.4639 - val_accura
Epoch 40/50
61/61 - 2s - loss: 0.2004 - accuracy: 0.9249 - val_loss: 0.3458 - val_accura
Epoch 41/50
61/61 - 2s - loss: 0.1995 - accuracy: 0.9240 - val_loss: 0.3296 - val_accura
Epoch 42/50
61/61 - 3s - loss: 0.1982 - accuracy: 0.9302 - val_loss: 0.3067 - val_accura
Epoch 43/50
61/61 - 2s - loss: 0.1992 - accuracy: 0.9309 - val_loss: 0.3267 - val_accura
Epoch 44/50
61/61 - 2s - loss: 0.2036 - accuracy: 0.9249 - val_loss: 0.2741 - val_accura
Epoch 45/50
61/61 - 2s - loss: 0.1913 - accuracy: 0.9281 - val_loss: 0.3201 - val_accura
Epoch 46/50
61/61 - 2s - loss: 0.1977 - accuracy: 0.9263 - val_loss: 0.3049 - val_accura
Epoch 47/50
61/61 - 3s - loss: 0.1971 - accuracy: 0.9247 - val_loss: 0.3329 - val_accura
Epoch 48/50
61/61 - 2s - loss: 0.1963 - accuracy: 0.9272 - val_loss: 0.3204 - val_accura
Epoch 49/50
61/61 - 2s - loss: 0.1979 - accuracy: 0.9320 - val_loss: 0.3400 - val_accura
Epoch 50/50
61/61 - 2s - loss: 0.1893 - accuracy: 0.9336 - val_loss: 0.3335 - val_accura
```



Evaluation

For evalaute the model, I used f-scores, ROC curves and overall accuracies.I provided the models accuracies and f-scores as table below.I also provided ROC curves for SVM and Random Forest.

Models	Accuracy	F-score (1)	F-score(0)
SVM with all Features	97	95	97
Random Forest	97	96	98
LSTM	93	N/A	N/A

As you see the most accurate algorithm is Random Forest and all algorithms are more successful at classifiying 0 class than 1 class.

ROC Curves

```
pip install scikit-plot
```

```
Collecting scikit-plot
  Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Requirement already satisfied: matplotlib>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from scikit-plot) (3.7.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.2.2)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.11.4)
Requirement already satisfied: joblib>=0.10 in /usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.3.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scik
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-p
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->sci
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->sci
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-pl
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->sciki
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scik
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotli
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7
```

```
import scikitplot as skplt
import matplotlib.pyplot as plt
#ROC Curve for SVM
y_pred_proba=clf.predict_proba(X_testr)
skplt.metrics.plot_roc_curve(y_testr,y_pred_proba)
plt.show()
```