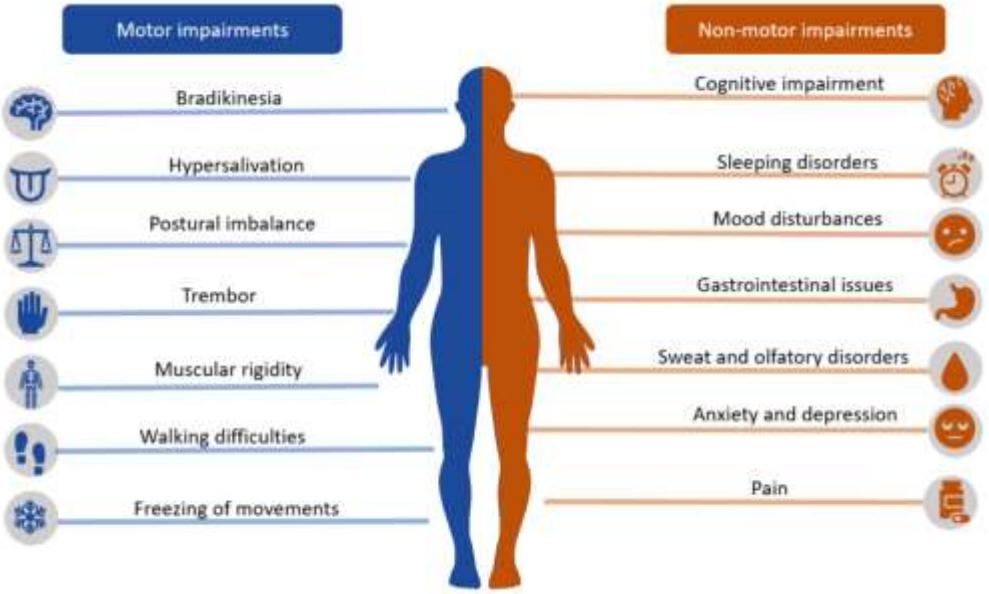
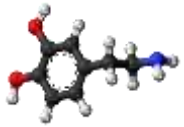
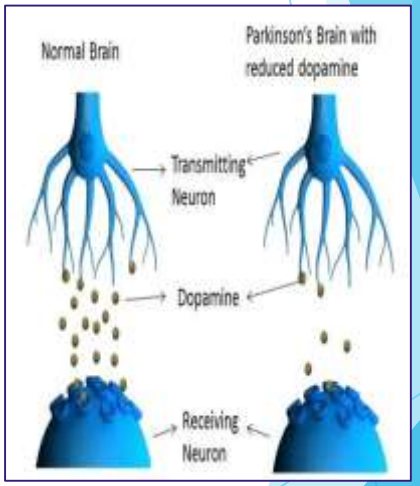
DETECTION OF PARKINSON DISEASE USING MACHINE LEARNING



WHAT IS PARKINSON DISEASE?

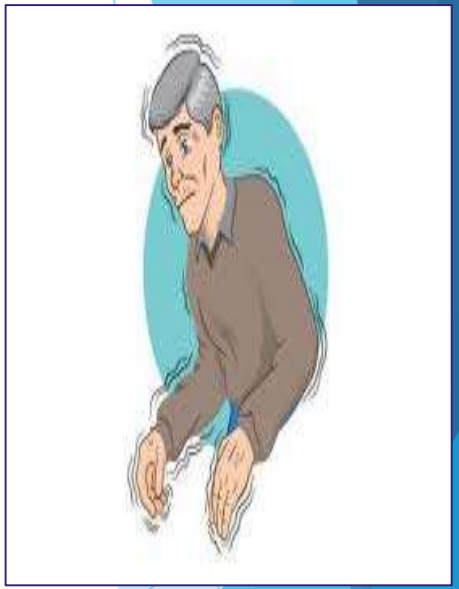


- It is a progressive neurological disorder
- Its main cause is lack of production of Dopamine hormone in our brain.
- Parkinson is non curable disease but early detection could help in controlling the disease.



SYMPTOMS OF PARKINSON'S DISEASE:

- Tremor (trembling) in hands, arms, legs, jaw, or head.
- Stiffness of the limbs and trunk.
- Slowness of movement.
- Impaired balance and coordination, sometimes leading to falls
- Shimmering of voice..



OUR APPROACH

PARKINSON DETECTION

VOICE ANALYSIS

1.Feature extraction

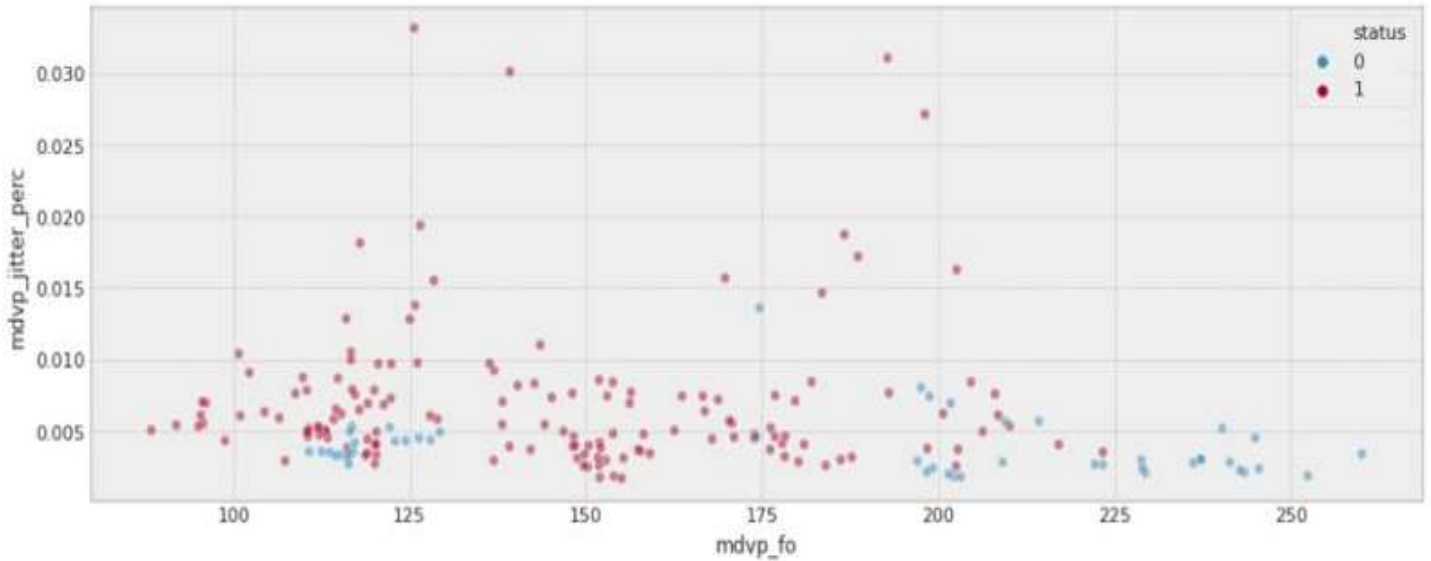
2.Model training.

3.Model prediction

EXTRACTED FEATURES FROM SPEECH RECORDINGS:

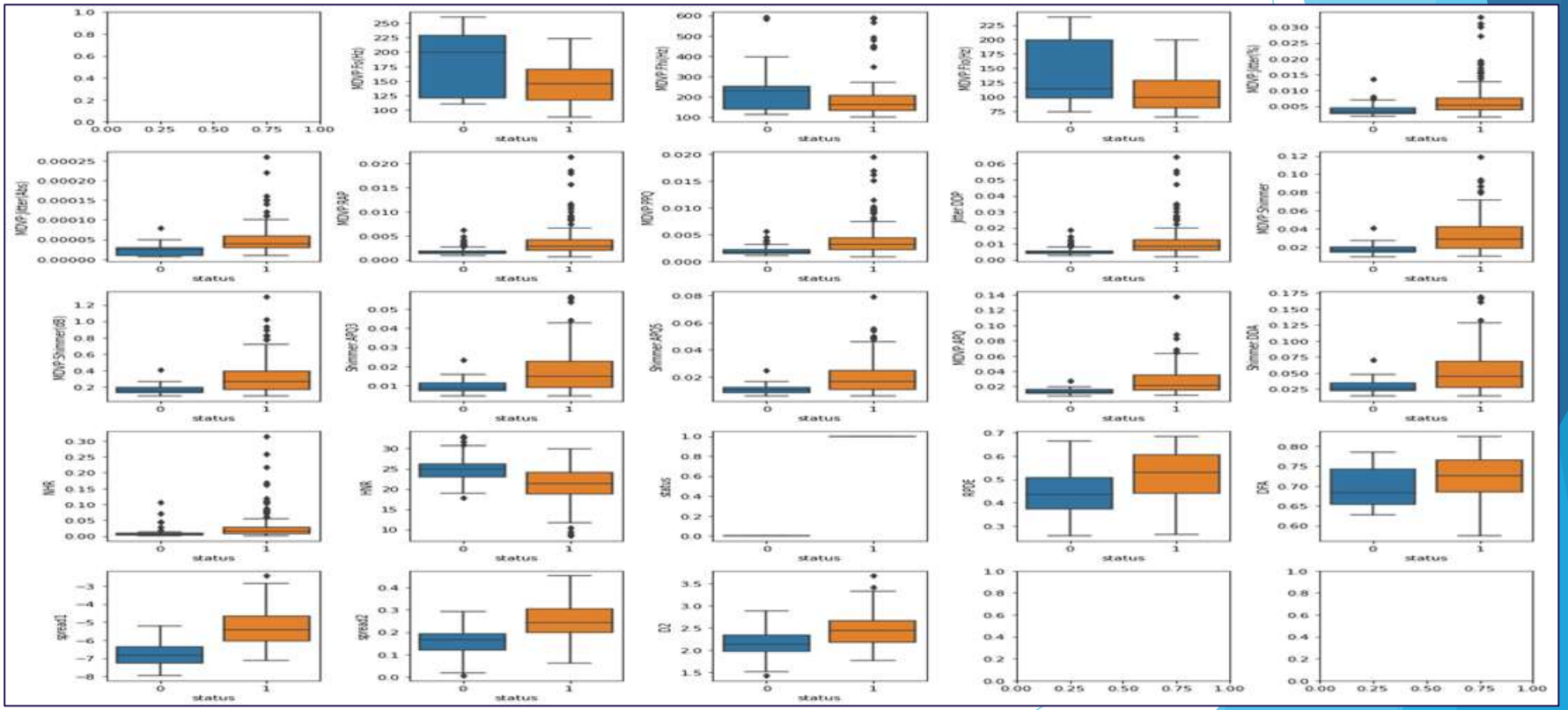
ABBREVIATIONS	FEATURE DESCRIPTION
MDVP:FO	Average vocal fundamental frequency
MDVP:Fhi	Maximum vocal fundamental frequency
MDVP:Flo	Minimum vocal fundamental frequency
MDVP:Jitter(%)	MDVP jitter in percentage
MDVP:Jitter(Abs)	MDVP absolute jitter in ms
MDVP:RAP	MDVP relative amplitude perturbation
MDVP:PPQ	MDVP five-point period perturbation quotient
Jitter:DDP	Average absolute difference of differences between jitter cycles

ABBREVIATIONS	FEATURE DESCRIPTION
MDVP:Shimmer	MDVP local shimmer
MDVP:Shimmer(dB)	MDVP local shimmer in dB
Shimmer:APQ3	Three-point amplitude perturbation quotient
Shimmer:APQ5	Five-point amplitude perturbation quotient
MDVP : APQ11	MDVP 11-point amplitude perturbation quotient
Shimmer:DDA	Average absolute differences between the amplitude
NHR	Noise to harmonics ratio
HNR	Harmonics to noise ratio

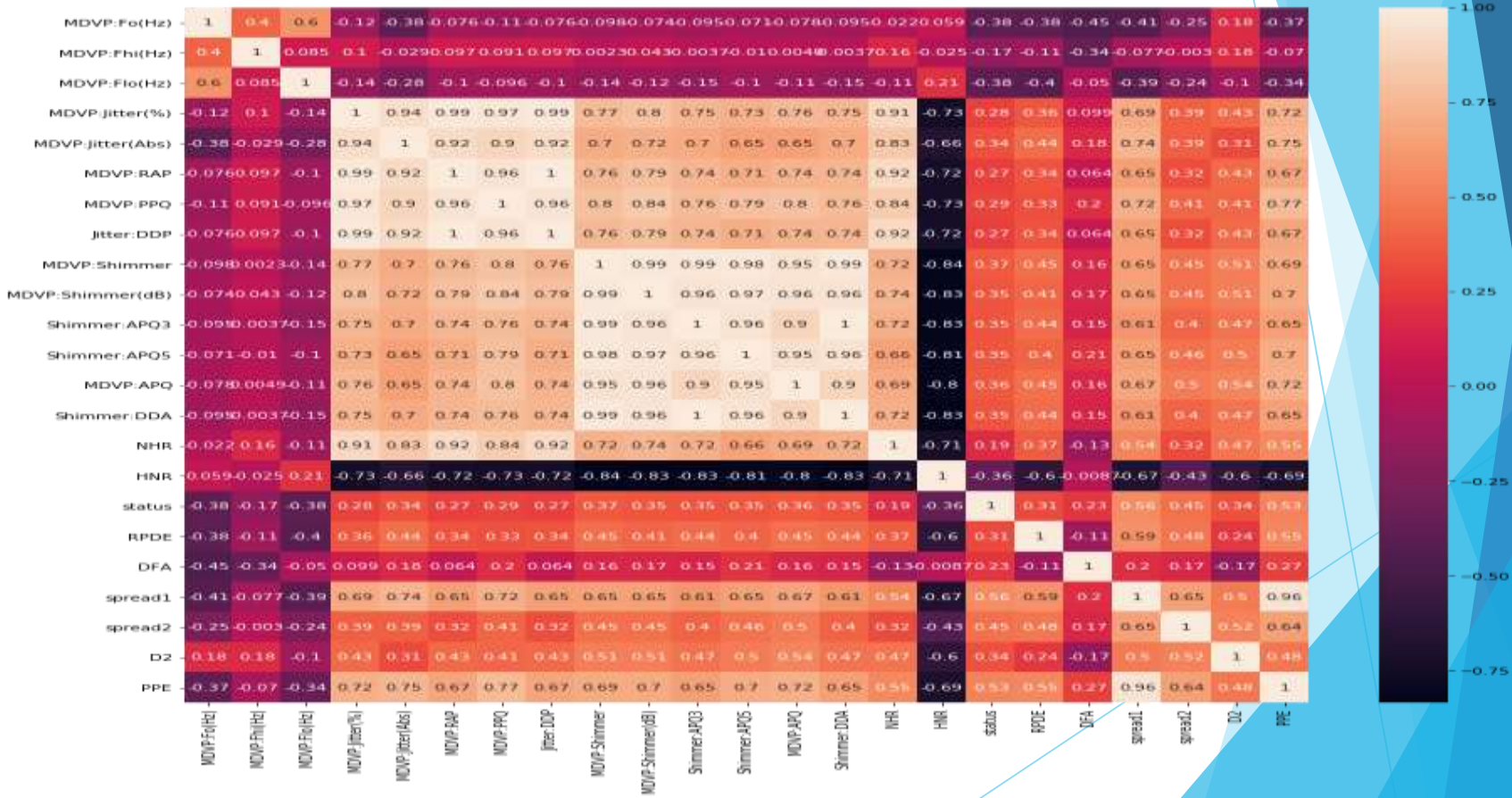


- From the above plot ,we can see that the people without Parkinson's have their fundamental frequencies that is either high or very low and the percentage of jitter is usually lower than 0.005%.

GRAPHICAL REPRESENTATION



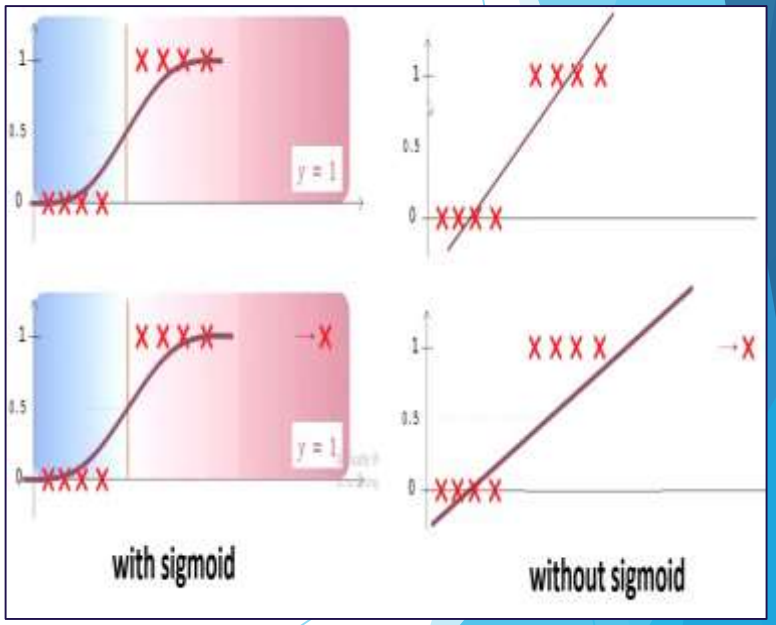
CORRELATION MATRIX



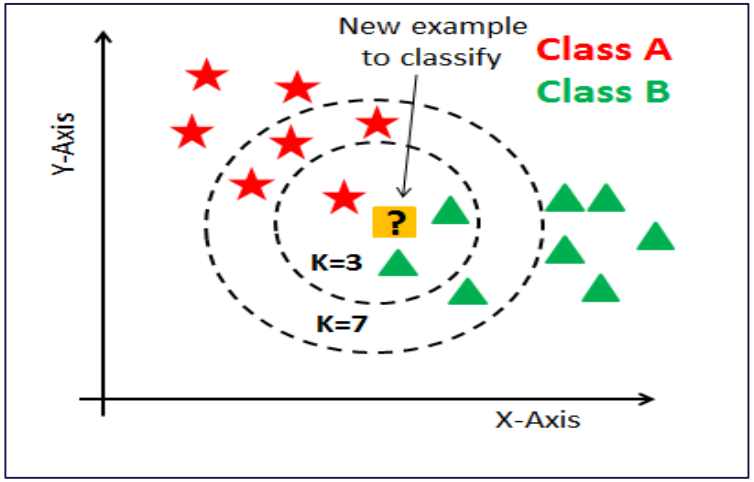
LOGISTIC REGRESSION

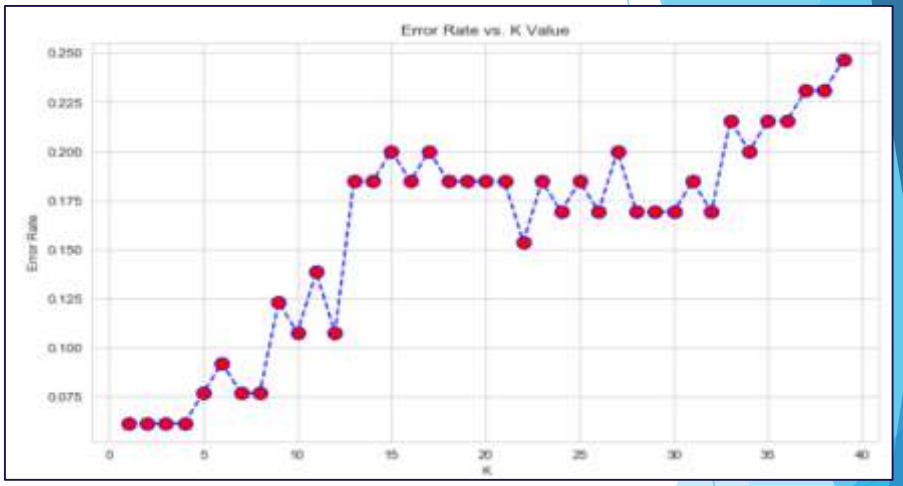
WHY NOT LOGISTIC REGRESSION ?

- Logistic regression classifies by drawing a linear boundary hence it is seldom used for real world scenario where nonlinear classification boundary is required.
- Large number of features.
- ACCURACY - 89.23%



KNN

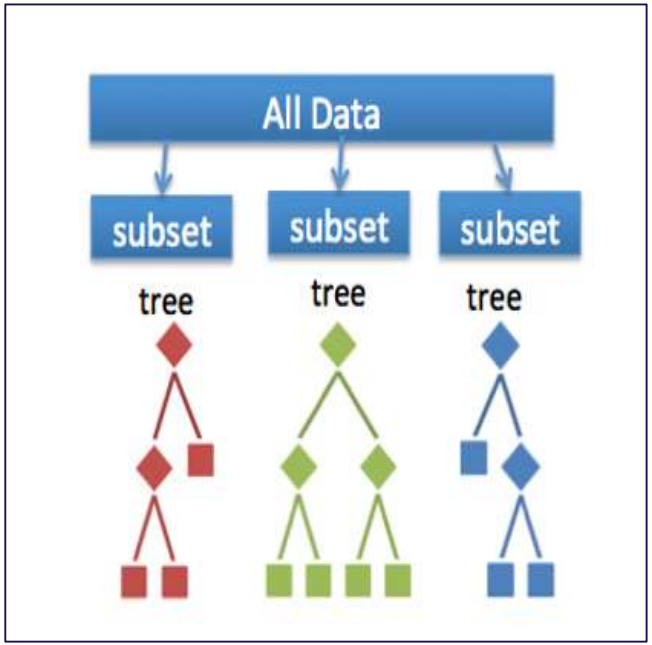




WHY NOT K NEAREST NEIGHBOUR ?

- If dataset increased then it would give high computational overhead.
- It assumes and gives equal importance to all features which is not suitable for our result.
- ACCURACY – 90.7%

RANDOM FOREST



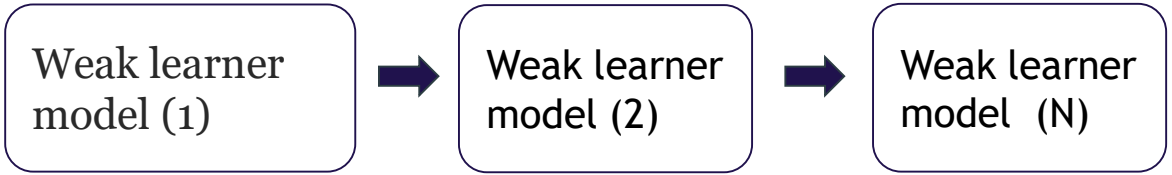
WHY NOT RANDOM FOREST ?

- As in this model all are independent decision trees , so there is no chance of self learning for more appropriate detection.

- ACCURACY -86.15%

XGBoost

Sequentially combines multiple models:



- F(i)=F(i-1)+f(i)
- F(i) is current model, F(i-1) is previous model and *small* f(i) represents a weak model
- For any incorrect prediction, larger weights are assigned to misclassified samples and lower ones to samples that are correctly classified.
- ACCURACY - 92.307%

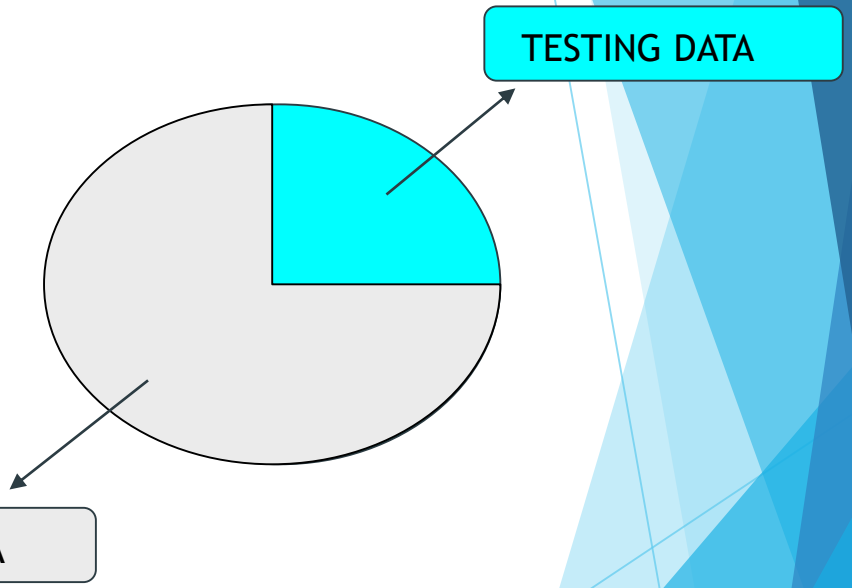
ADVANTAGES OF XGBOOST

- Accuracy of the model is increased by combining multiple weak learners.
- (Accuracy = 92.307%).
- Less computation steps are required for training of the model.
- Tree pruning: Nodes which does not contribute much for the classification of leaf nodes are removed.
- Parallelization: It divides the data into blocks which are parallelly assigned to multiple threads for optimized usage of CPU.

HOW WE DECIDED OUR MODEL ?

CROSS VALIDATION SCORES:

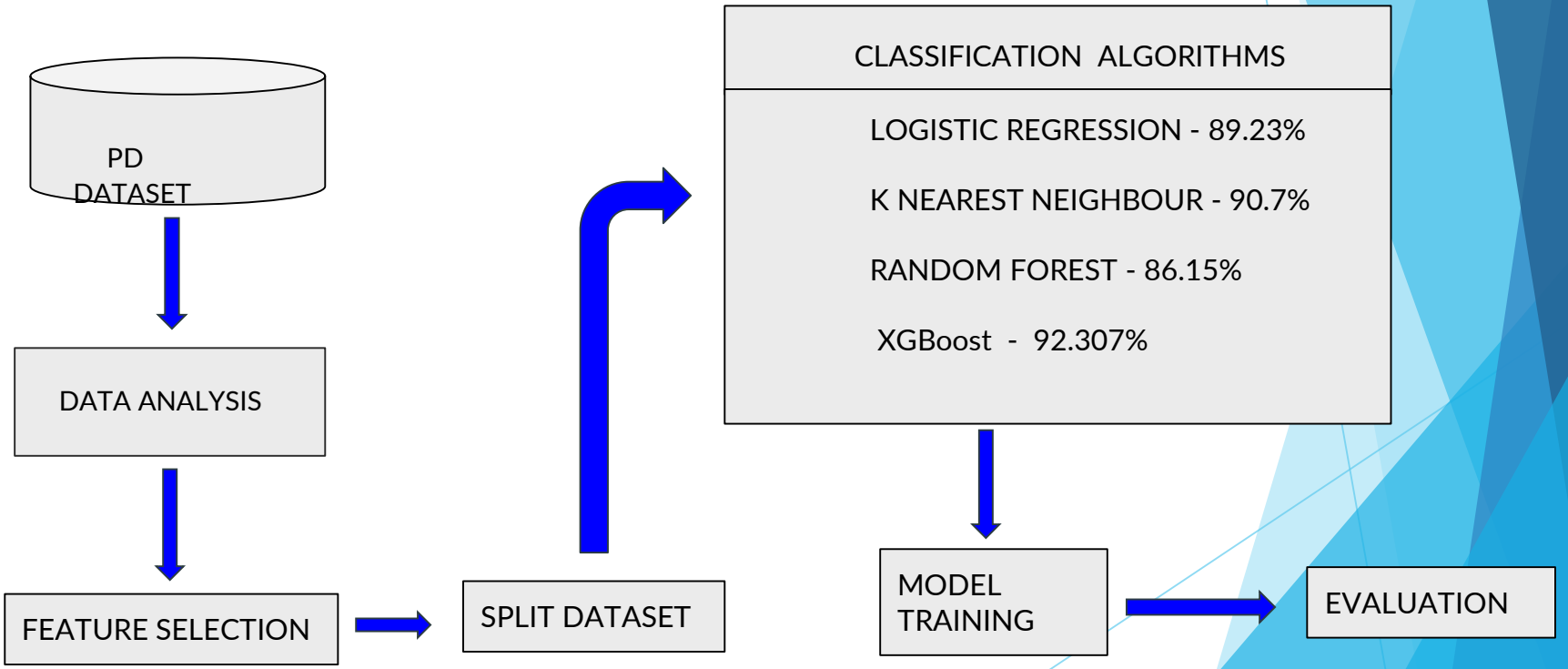
- LOGISTIC REGRESSION - 86.5
- RANDOM FOREST - 88.74
- XG BOOST - 91.02



Hyper parameter:

[objective ='binary:hinge', colsample_bytree = 0.3, learning_rate=0.3, max_depth = 5, alpha = 10, n_estimators = 10]

MODEL FLOW DESIGN FOR DETECTION USING VOICE DATASET



SCALING & SPLITTING

```
]:  1 import numpy as np
    2 from sklearn.preprocessing import MinMaxScaler
    3 from sklearn.model_selection import train_test_split,cross_val_score
    4 from sklearn.metrics import accuracy_score,mean_squared_error
```

```
]:  1 features=dfo.drop(['status','name'],axis=1)
    2 labels=dfo['status']
```

```
]:  1 scaler=MinMaxScaler((-1,1))
    2 x=scaler.fit_transform(features)
    3 y=labels
```

```
]:  1 x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.2, random_state=5)
```

```
print('log reg',lr,lr.mean())#
print('knn',knc,knc.mean())#
print('rfc',rfc,rfc.mean())#
print('xgb',xgb,xgb.mean())#
```

```
log reg [0.90625    0.90322581 0.87096774 0.83870968 0.80645161] 0.8651209677419356
knn [0.8125     0.90322581 0.90322581 0.90322581 0.93548387] 0.8915322580645162
rfc [0.90625    0.93548387 0.87096774 0.93548387 0.87096774] 0.9038306451612904
xgb [0.90625    0.93548387 0.83870968 0.96774194 0.90322581] 0.9102822580645162
```

```
lr=cross_val_score(LogisticRegression(),x_train,y_train)
xgbc=cross_val_score(XGBRFClassifier(),x_train,y_train)
xgb=cross_val_score(XGBClassifier(),x_train,y_train)
svm=cross_val_score(SVC(),x_train,y_train)
#nb=cross_val_score(MultinomialNB(),x_train,y_train)
dtc=cross_val_score(DecisionTreeClassifier(),x_train,y_train)
adb=cross_val_score(AdaBoostClassifier(),x_train,y_train)
bbc=cross_val_score(BaggingClassifier(),x_train,y_train)
etc=cross_val_score(ExtraTreesClassifier(),x_train,y_train)
gbc=cross_val_score(GradientBoostingClassifier(),x_train,y_train)
rfc=cross_val_score(RandomForestClassifier(),x_train,y_train)
#vc=cross_val_score(VotingClassifier(estimators),x_train,y_train)
knc=cross_val_score(KNeighborsClassifier(),x_train,y_train)
```

```
import xgboost as xgb
import pandas as pd
import numpy as np
X,y = dfo.iloc[:,1:-1],dfo.iloc[:,-1]
```

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=43)
```

```
xg_reg = xgb.XGBRegressor(objective ='binary:hinge', colsample_bytree = 0.3, learning_rate = 0.3,
                max_depth = 5, alpha = 10, n_estimators = 10) #learning_rate = 0.1
```

```
xg_reg.fit(X_train,y_train)

preds = xg_reg.predict(X_test)
```

```
print(preds)
```

```
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.]
```

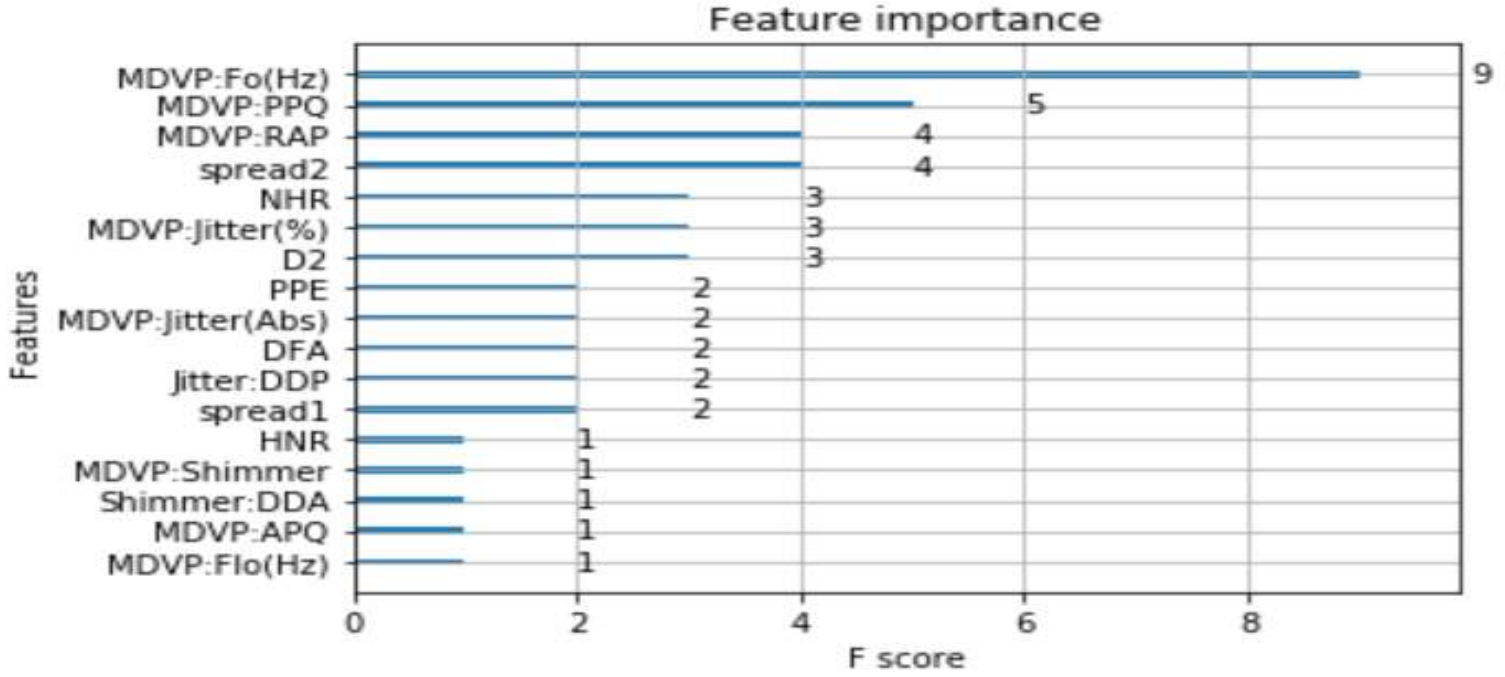
```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))

from sklearn.metrics import accuracy_score
print('XGBoost model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, preds)*100))
```

```
RMSE: 0.277350
XGBoost model accuracy score: 92.3077
```

```
import matplotlib.pyplot as plt
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [14, 5]
plt.show()
```

RUNTIME ANALYSIS

```
1  import matplotlib.pyplot as plt
2  xgb.plot_importance(xg_reg)
3  plt.rcParams['figure.figsize'] = [14, 5]
4  plt.show()
```



```
# This is the function to measure voice pitch
def measurePitch(voiceID, f0min, f0max, unit):
    sound = parselmouth.Sound(voiceID) # read the sound
    pitch = call(sound, "To Pitch", 0.0, f0min, f0max) #create a praat pitch object
    meanF0 = call(pitch, "Get mean", 0, 0, unit) # get mean pitch
    stdevF0 = call(pitch, "Get standard deviation", 0 ,0, unit) # get standard deviation
    harmonicity = call(sound, "To Harmonicity (cc)", 0.01, 75, 0.1, 1.0)
    hnr = call(harmonicity, "Get mean", 0, 0)
    pointProcess = call(sound, "To PointProcess (periodic, cc)", f0min, f0max)
    localJitter = call(pointProcess, "Get jitter (local)", 0, 0, 0.0001, 0.02, 1.3)
    localabsoluteJitter = call(pointProcess, "Get jitter (local, absolute)", 0, 0, 0.0001, 0.02, 1.3)
    rapJitter = call(pointProcess, "Get jitter (rap)", 0, 0, 0.0001, 0.02, 1.3)
    ppq5Jitter = call(pointProcess, "Get jitter (ppq5)", 0, 0, 0.0001, 0.02, 1.3)
    ddpJitter = call(pointProcess, "Get jitter (ddp)", 0, 0, 0.0001, 0.02, 1.3)
    localShimmer =  call([sound, pointProcess], "Get shimmer (local)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    localdbShimmer = call([sound, pointProcess], "Get shimmer (local_dB)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    apq3Shimmer = call([sound, pointProcess], "Get shimmer (apq3)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    aqpq5Shimmer = call([sound, pointProcess], "Get shimmer (apq5)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    apq11Shimmer =  call([sound, pointProcess], "Get shimmer (apq11)", 0, 0, 0.0001, 0.02, 1.3, 1.6)
    ddaShimmer = call([sound, pointProcess], "Get shimmer (dda)", 0, 0, 0.0001, 0.02, 1.3, 1.6)


    return meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq5Jitter, ddpJitter, localShimmer, localdbSh
```

```
def runPCA(df):
    #Z-score the Jitter and Shimmer measurements
    features = ['localJitter', 'localabsoluteJitter', 'rapJitter', 'ppq5Jitter', 'ddpJitter',
                'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shimmer', 'apq11Shimmer', 'ddaShimmer']
    # Separating out the features
    x = df.loc[:, features].values
    # Separating out the target
    #y = df.loc[:,['target']].values
    # Standardizing the features
    x = StandardScaler().fit_transform(x)
    #PCA
    pca = PCA(n_components=2)
    principalComponents = pca.fit_transform(x)
    principalDf = pd.DataFrame(data = principalComponents, columns = ['JitterPCA', 'ShimmerPCA'])
    principalDf
    return principalDf
```

```
# Go through all the wave files in the folder and measure pitch
for wave_file in glob.glob("audio/*.wav"):
    sound = parselmouth.Sound(wave_file)
    (meanF0, stdevF0, hnr, localJitter, localabsoluteJitter, rapJitter, ppq5Jitter, ddpJitter, localShimmer, localdbShimmer,
    file_list.append(wave_file) # make an ID list
    mean_F0_list.append(meanF0) # make a mean F0 list
    sd_F0_list.append(stdevF0) # make a sd F0 list
    hnr_list.append(hnr)
    localJitter_list.append(localJitter)
    localabsoluteJitter_list.append(localabsoluteJitter)
    rapJitter_list.append(rapJitter)
    ppq5Jitter_list.append(ppq5Jitter)
    ddpJitter_list.append(ddpJitter)
    localShimmer_list.append(localShimmer)
    localdbShimmer_list.append(localdbShimmer)
    apq3Shimmer_list.append(apq3Shimmer)
    aqpq5Shimmer_list.append(aqpq5Shimmer)
    apq11Shimmer_list.append(apq11Shimmer)
    ddaShimmer_list.append(ddaShimmer)
df = pd.DataFrame(np.column_stack([file_list, mean_F0_list, sd_F0_list, hnr_list, localJitter_list, localabsoluteJitter_list
                                   columns=['voiceID', 'meanF0Hz', 'stdevF0Hz', 'HNR', 'localJitter', 'localabsoluteJitter', 'ra
                                            'ppq5Jitter', 'ddpJitter', 'localShimmer', 'localdbShimmer', 'apq3Shimmer', 'apq5Shi
```

```
data_p={'MDVP:Fo(Hz)':list(df['meanF0Hz']),'MDVP:Fhi(Hz)':[dfo['MDVP:Fhi(Hz)'].mean()]*len(df),'MDVP:Flo(Hz)':[dfo['MDVP:Flo
'MDVP:Jitter(Abs)':[dfo['MDVP:Jitter(Abs)'].mean()]*len(df),'MDVP:RAP':list(df['rapJitter']),'MDVP:PPQ':list(df['ppq5Jitter'
'MDVP:Shimmer':list(df['localShimmer']),'MDVP:Shimmer(dB)':list(df['localdbShimmer']),'Shimmer:APQ3':list(df['apq3Shimmer'])
'MDVP:APQ':list(df['apq11Shimmer']),'Shimmer:DDA':list(df['ddaShimmer']),'NHR':[dfo['NHR'].mean()]*len(df),
'HNR':list(df['HNR']),'RPDE':[dfo['RPDE'].mean()]*len(df),'DFA':[dfo['DFA'].mean()]*len(df),'spread1':[dfo['spread1'].mean()
'spread2':[dfo['spread2'].mean()]*len(df),'D2':[dfo['D2'].mean()]*len(df),'PPE':[dfo['PPE'].mean()]*len(df)}
dfsample = pd.DataFrame.from_dict(data_p)
dfsample.head()
```

	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitt
0	113.77680569854286	197.104918	116.324631	0.0014213010729582554	0.000044	0.0007095201587491845	0.0007645157141077805	0.0021285604762
1	201.9450715201333	197.104918	116.324631	0.004846807759666541	0.000044	0.0025087786720272213	0.002842666400876423	0.007526336016
2	111.15389977177607	197.104918	116.324631	0.0019680369258500167	0.000044	0.0010017621778934044	0.0010133799253859255	0.003005286533
3	212.09857905298142	197.104918	116.324631	0.0031189223175170765	0.000044	0.0016619598167540847	0.0018064149400136825	0.004985879450
4	232.57400286508445	197.104918	116.324631	0.0020022482298386506	0.000044	0.001116431883016024	0.0009879185883070157	0.0033492956490

5 rows × 22 columns

RUNTIME OUTPUT

```
1  pred1 = LR.predict(dfsample)
```

```
1  pred1
```

```
array([0, 0, 0, 0, 0, 0], dtype=int64)
```

```
1  for t in pred1:#our output at runtime
2      if t==0:
3          print(' not parkinson')
4      else:
5          print('parkinson')
```

```
 not parkinson
 not parkinson
 not parkinson
 not parkinson
 not parkinson
 not parkinson
```

CONCLUSION

- The proposed model could be the first indication in detection of Parkinson.
- Hassle free detection
- Using Machine Learning we can apply a proactive approach towards detection.

Moreover…

- Real Time Voice Analysis for Parkinson Detection.
- This could be achieved for feature extraction using python library parselmouth and voice analysis software praat .

FURTHER DEVELOPMENT

- Interfacing project with : -

  - Website
  - Application

- Applicability of our project can be increased by taking into account other symptoms.
- We would be starting campaigns using social media, to increase awareness about Parkinson.

Hardware used:

- Personal Computer
- Laptop

Software Used: (language/library)

- For Data Analysis:
1. Pandas
2. NumPy
3. Matplotlib.pyplot

For Machine Learning:
Scikit-Learn

PYTHON LIBRARIES

●**NumPy** : It's a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

●**SkLearn** : A free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, k-means etc. and is designed to interoperate with NumPy and SciPy.

●**Keras** : Its a open source neural network library written in Python. Capable of running on top of TensorFlow. Designed to enable fast experimentation with deep neural networks, its pros being user-friendly, modular, and extensible.

●**Matplotlib** : A plotting library for the Python programming language and its numerical mathematics extension.

PYTHON LIBRARIES

- **Os:** provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality.
- **OpenCV:** OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc.
- **Pandas:** Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks.
- **Parselmouth:** Audio processing library

REFERENCES

- https://www.nia.nih.gov/health/parkinsons-disease
- https://www.parkinson.org/understanding-parkinsons/what-is-parkinsons
- https://archive.ics.uci.edu/ml/datasets/parkinsons
- XGBoost: A Deep Dive into Boosting | by Rohan Harode | SFU Professional Master's Program in Computer Science | Medium
- Saunders-Pullman R, Derby C, Stanley K, Floyd A, Bressman S, Lipton RB, et al. Validity of spiral analysis in early Parkinson's disease. *Mov Disord* (2008) 23(4):531–7. doi:10.1002/mds.21874
- San Luciano M, Wang C, Ortega RA, Yu Q, Boschung S, Soto-Valencia J, et al. Digitized spiral drawing: a possible biomarker for early Parkinson's disease. *PLoS One* (2016) 11(10):e0162799. doi:10.1371/journal.pone.0162799
- Sisti JA, Christophe B, Seville AR, Garton AL, Gupta VP, Bandin AJ, et al. Computerized spiral analysis using the iPad. *J Neurosci Methods* (2017) 275:50–4.

THANK YOU