EARLY DETECTION OF PARKINSON'S DISEASE USING MACHINE LEARNING

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

SANKARANARAYANAN.S
(Reg. No.:124156079, B. Tech, CSE Artificial Intelligence and Data Science)

UPENDHAR.S
(Reg. No.:124156080, B. Tech, CSE Artificial Intelligence and Data Science)

PESHWAR RAJESH
(Reg. No.:124157042, B. Tech, CSE Cyber Security and Blockchain Technology)

May 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMILNADU, INDIA - 613 401

THANJAVUR, TAMIL NADU, INDIA- 613401

<u>Bonafide Certificate</u>

This is to certify that the report titled **"Early detection of Parkinson's disease using machine learning"** submitted as a requirement for the course, CSE300 : **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr.Sankaranarayanan.S(Reg.No.:124156079,B.Tech,CSE Artificial Intelligence and Data Science)** , **Mr.Upendhar.S(Reg.No.:124156080,B.Tech,CSE Artificial Intelligence and Data Science)** , **Mr.Peshwar Rajesh(Reg.No.:124157042,B.Tech,CSE Cyber Security and Blockchain Technology)** during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor:

Name with Affiliation:

Date:

Mini project *Viva Voce* held on ________________

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. S. Gopalakrishnan** , Associate Dean, Department of Computer Application , **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology

Our guide **Mrs.Jeyasheela Rakkini M J**, AP-II, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through project.

Table of Contents

LIST OF FIGURES

ABBREVIATIONS

(PD)	Parkinson’s disease
(PWP)	Patients with Parkinson’s
(MDVP)	Multi-Dimensional voice program
(SVM)	Support Vector Machine (SVM)
(RF)	Random Forest
(KNN)	K-Nearest Neighbors
(LR)	Logistic Regression
(PCA)	Principal Component Analysis

ABSTRACT

Parkinson's disease (PD) is a type of neurodegenerative disorder affecting 60% of people over the age of 50 years, which is quite significant. Patients with Parkinson’s (PWP) face mobility challenges and speech difficulties, making physical visits for treatment and monitoring a hurdle. PD can generally be treated through early detection, thus enabling patients to lead a normal life. The rise of an aging population over the world emphasizes the need to definitely detect PD early, remotely and accurately in a big way. This paper highlights the use of machine learning techniques in telemedicine to detect PD in its early stages. Research has been basically carried out on the multidimensional voice program (MDVP) audio data of 30 PWP and healthy people during training of 4 machine learning (ML) models. Comparison of results of classification by Support Vector Machine (SVM), particularly Random Forest, K-Nearest Neighbors (KNN) and Logistic Regression models, for all intents and purposes yield basically Random Forest classifier as the generally ideal Machine Learning (ML) technique for detection of PD. Random Forest classifier model particularly has a detection accuracy of 91.83% and sensitivity of 0.95. Through the findings of this paper, we aim to go for all intents and purposes that promote the use of ML in telemedicine, thereby providing a new lease of life to patients suffering from Parkinson's disease.

Keywords: *KNN, PCA, MDVP*

1.Summary of Base Paper

Title: Early detection of Parkinson’s disease using machine learning

Journal name: Procedia computer Science

Publisher: Elsevier Production Year: 2023

1.1 Introduction:

Parkinson's disease (PD) is a degenerative nervous system ailment that has an impact on mobility. It is characterized by non-motor symptoms like sadness and cognitive decline as well as motor symptoms like tremors, stiffness, and slowness of movement. It is brought on by the degeneration of dopamine-producing neurons in the brain. Early diagnosis and treatment of PD depend on early recognition of the condition. Parkinson’s disease is a major health concern that affects millions of people worldwide, and early detection of the disease is crucial for timely intervention. Although there is currently no cure for PD, there are medicines that can help control the symptoms and enhance the lives of individuals who have PD.

1.2 Dataset:

The Parkinson's Disease dataset from the UCI Machine Learning Repository is a collection of biomedical voice measurements from 42 patients with early-stage Parkinson's disease and 23 healthy control individuals. The data contains 23(columns) attributes of audio and 195(rows) patient data. Features include jitter, shimmer, pitch etc. Using these features, we can distinguish between individuals with Parkinson's disease and without Parkinson’s disease

Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data

1.3 Methodology:

We take the Database and preprocess the data. Refer figure 4.1.
Then using train data and test data we can train the model and validate the results.
Finally, our ML model will predict if the input sample has Parkinson’s disease or not.

1.3.1 Algorithm for approach 1(BASIC APPROACH): Models are trained on 22 attributes of data

- Collect MDVP audio data from PPPMI and UCI databases
- Perform data analysis to detect skew, imbalance and distribution of variables in data
- Scale the data to common range using Standard Scaler
- Split dataset into testing and training sets, where training data is 75% of total
- Train SVM, logistic regression, random forest and KNN models.

1.3.2 Algorithm for approach 2(PCA APPROACH): Principal Component Analysis (PCA) is applied to identify 5 key attributes

• Collect MDVP audio data from PPPMI and UCI databases
• Perform data analysis to detect skew, imbalance and distribution of variables in data
• Scale the data to a common range using Standard Scaler
• Identify variance in every column of data and apply Principal Component Analysis (PCA) to identify 5 most relevant features to model training, out of 22 attributes.
• Split dataset into testing and training sets, where training data is 75% of total
• Retrain SVM, logistic regression, random forest and KNN models.
• Compare classification results using confusion matrix, ROC-AUC curve and accuracy

1.3.3 Algorithm for approach 3(POST BALANCING): Imbalance removal in dataset

• Collect MDVP audio data from PPPMI and UCI databases
• Perform data analysis to detect skew, imbalance and distribution of variables in data
• The dataset is imbalanced, with 109 records of PWP and 40 records of normal people, as illustrated in figure 4.10. The imbalance is resolved by up sampling the minority class to reach 109 records each.
• Scale the data to common range using Standard Scaler
• Split dataset into testing and training sets, where training data is 75% of total
• Retrain SVM, logistic regression, random forest and KNN models.
• Compare classification results using confusion matrix, ROC-AUC curve and accuracy

1.4 Data Preprocessing:

To organize data and address missing attributes in the dataset, data wrangling is used. The noise to harmonic tone (NHR) ratio and the harmonic tone to noise (HNR) ratio for PWP are shown in Figure 4.2. As the disease progresses through its stages, speech noise increases, increasing NHR. The skewed statistics and low NHR score (0.3) suggest a low-quality voice.

Figure 4.3 shows a box plot of each of the dataset's 22 properties. It shows how data are distributed and skewed over a median quartile. Figure shows records in orange (PWP records) and blue (regular records). Due to the higher speech noise, NHR data points for PWP have the most outliers. Similar to PWP records, HNR records feature maximum data outliers that are below the median.

The pair plot of shimmer data is shown in Figure 4.4. It is utilized to draw attention to how the shimmer of the voice changes for PWP patients as opposed to healthy ones. It demonstrates that Shimmer: APQ3 and Shimmer: DDA have a linear relationship while Shimmer: APQ5 and Shimmer: APQ3 have an asymmetric relationship.

1.5 Model training:

In this study, three different classification methods—Logistic Regression, Random Forest, Support Vector, and K Nearest Neighbors—are investigated.

- 195 records and 22 attributes make up the entire dataset.
- Following Principal Component Analysis (PCA), the dataset had 195 records and 5 characteristics.
- A balanced dataset had 109 records and 22 attributes.

1.6 Model evaluation:

We examine the outcomes of 3 techniques and 9 trained models to find the best model. Metrics such as the ROC-AUC curve, confusion matrix, accuracy, precision, recall, and F1 score were chosen for comparison. Refer Figure 4.11,4.12.4.13,4.14 for results

1.7 Results:

Using vowel phonation data, the Random Forest classifier can classify Parkinson's disease with 91.835% accuracy and 0.95 sensitivity. Given that each of the 22 attributes in the MDVP dataset is given equal weight, the results of the Random Forest model are optimal. This study also shows the findings of the SVM model, which, after PCA is applied to the dataset, yields accuracy and sensitivity values of 91.836% and 0.94, respectively. Both SVM and Random Forest models exhibit good outlier performance and are strong models. No false positive findings are predicted by the models. For balanced datasets, the K nearest neighbor's (KNN) model also performs well since categorization into two categories without the use of data assumptions is preferred. As a result, we advise using the Random Forest model to categorize the disease's progression.

2.Merits and Demerits

2.1 Merits: -

- Relevance: Research into early Parkinson's disease identification is essential, and applying machine learning techniques for this goal is in line with recent developments in medical technology.
- Algorithm Variety: The study investigates the application of a number of machines learning algorithms, including KNN, logistic regression, SVM, and random forest. This exemplifies a thorough strategy that enables a comparison of various approaches and their efficiency in early identification.
- Practical Application: Parkinson's disease can be identified early, which allows for prompt treatment and better patient outcomes. If the article is successful in proving that machine learning algorithms work in this situation, it may have important practical ramifications for patients and healthcare practitioners.

2.2 Demerits: -

- Analysis insufficient: The research might gloss over crucial issues like feature selection, dataset properties, or model interpretability when discussing the application of machine learning techniques for Parkinson's disease early diagnosis. This might cut down on the analysis's breadth and depth.
- Better models present: There are better models which use deep learning methods such as ANN, CNN which gives better accuracies 96.45% than 93.8% in this paper

2.3 Inference from the above Merits and Demerits: -

The literature review on Parkinson's disease discusses various machine learning techniques and how they might be used for detection and diagnosis. The papers use deep learning methods including CNN, LSTM, and bespoke deep learning models along with algorithms like XGBoost, KNN with entropy, random forests, and others. The findings suggest promising accuracy ranges for Parkinson's disease prediction of 83.6% to 96.45%. Limited sample sizes, biased or unrepresentative data, overfitting, a lack of transparency, inadequate explanation for feature selection, and the necessity for external validation are among the more typical constraints. Some studies also lack comprehensive information on procedures, data properties, and feature extraction methods. In spite of these drawbacks, the literature evaluation indicates that machine learning techniques have the potential to improve early detection, analyze genetic and transcriptome data, and contribute to a better understanding of Parkinson's disease pathogenesis.

3.Source code

3.1 app.py

```
from flask import Flask, render_template, request
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import resample
import pandas as pd

def load_data(num_records):
    url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data'
    data = pd.read_csv(url)
    data.drop('name', axis=1, inplace=True)
    X = data.drop('status', axis=1)
    y = data['status']
    return X[:num_records], y[:num_records]

def knn_model(X, y):
    knn = KNeighborsClassifier()
    knn.fit(X, y)
    return knn

def lr_model(X, y):
    lr = LogisticRegression()
    lr.fit(X, y)
    return lr
```

```
def svm_model(X, y):
    svm = SVC()
    svm.fit(X, y)
    return svm


def rf_model(X, y):
    rf = RandomForestClassifier()
    rf.fit(X, y)
    return rf


app = Flask(__name__)


@app.route('/')
def home():
    return render_template('home.html')


@app.route('/predict', methods=['POST'])
def predict():
    num_records = int(request.form['num_records'])
    model_name = request.form['model_name']
    algorithm_number=int(request.form['algorithm_number'])
    X, y = load_data(num_records)
    if algorithm_number==1:
        if model_name == 'knn':
            model = knn_model(X, y)
        elif model_name == 'lr':
            model = lr_model(X, y)
        elif model_name == 'svm':
```

```
        model = svm_model(X, y)
      elif model_name == 'rf':
        model = rf_model(X, y)
      accuracy = model.score(X, y)
      return render_template('result.html', accuracy=accuracy, model_name=model_name)

    if algorithm_number==2:
      scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X)
      pca = PCA(n_components=5)
      X_reduced = pca.fit_transform(X_scaled)
      if model_name == 'knn':
        model = knn_model(X_reduced, y)
      elif model_name == 'lr':
        model = lr_model(X_reduced, y)
      elif model_name == 'svm':
        model = svm_model(X_reduced, y)
      elif model_name == 'rf':
        model = rf_model(X_reduced, y)
      accuracy = model.score(X_reduced, y)
      return render_template('result.html', accuracy=accuracy, model_name=model_name)

    if algorithm_number==3:
      X_resampled, y_resampled = resample(X[y == 1], y[y == 1], replace=True,
n_samples=X[y == 0].shape[0], random_state=42)
      X_resampled = pd.concat([X[y == 0], X_resampled])
      y_resampled = pd.concat([y[y == 0], y_resampled])
      scaler = StandardScaler()
      X_scaled = scaler.fit_transform(X_resampled)
```

```
        if model_name == 'knn':
            model = knn_model(X_scaled, y_resampled)
        elif model_name == 'lr':
            model = lr_model(X_scaled, y_resampled)
        elif model_name == 'svm':
            model = svm_model(X_scaled, y_resampled)
        elif model_name == 'rf':
            model = rf_model(X_scaled, y_resampled)
        accuracy = model.score(X_scaled,y_resampled)
        return render_template('result.html', accuracy=accuracy, model_name=model_name)

if __name__ == '__main__':
    app.run(debug=True,port=8000)
```

3.2 home.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Parkinson's Dataset</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <h1>Parkinson's Dataset</h1>
    <form action="/predict" method="POST">
        <label for="num_records">Number of records for training:</label>
        <input type="number" id="num_records" name="num_records" required><br><br>
        <label for="algorithm_number">Algorithm to use:</label>
        <select id="algorithm_number" name="algorithm_number" required>
            <option value="1">basic algorithm</option>
            <option value="2">pca approach</option>
            <option value="3">After balancing the data</option>
        </select><br><br>
        <label for="model_name">Model to use:</label>
        <select id="model_name" name="model_name" required>
            <option value="knn">K-Nearest Neighbors</option>
            <option value="lr">Logistic Regression</option>
            <option value="svm">Support Vector Machines</option>
            <option value="rf">Random Forest</option>
        </select><br><br>
        <input type="submit" value="Train and Predict">
    </form>
</body>
</html>
```

3.3 result.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Parkinson's Dataset</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='styles2.css') }}">
</head>
<body>
    <h1>Results</h1>
    <p>Accuracy using {{ model_name }}: {{ accuracy }}</p>
</body>
</html>
```

3.4 styles.css

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
}

h1 {
    color: #1a1a1a;
}

form {
    background-color: #ffffff;
    padding: 20px;
    border: 1px solid #cccccc;
    border-radius: 5px;
    margin: auto;
    max-width: 500px;
}

label {
    display: block;
    margin-bottom: 10px;
    font-weight: bold;
    color: #333333;
}

input[type="number"], select {
    padding: 5px;
    margin-bottom: 15px;
    border: 1px solid #cccccc;
    border-radius: 3px;
    width: 100%;
    box-sizing: border-box;
}

input[type="submit"] {
    background-color: #1a1a1a;
    color: #ffffff;
    padding: 10px 20px;
    border: none;
    border-radius: 3px;
    cursor: pointer;
    font-weight: bold;
}

input[type="submit"]:hover {
    background-color: #333333;
}
```

3.5 styles2.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
}

h1 {
    color: #1a1a1a;
}
```

```
p {
   font-size: 20px;
   color: #333333;
   margin: 50px auto;
   max-width: 500px;
   text-align: center;
}

label {
   font-weight: bold;
}

input[type="submit"] {
   background-color: #1a1a1a;
   color: #ffffff;
   padding: 10px 20px;
   border: none;
   border-radius: 3px;
   cursor: pointer;
   font-weight: bold;
}

input[type="submit"]:hover {
   background-color: #333333;
}
```

3.5 parkinsondisease.ipynb

```
#!/usr/bin/env python

# coding: utf-8


# ## Exploratory Analysis

# To begin this exploratory analysis, first import libraries and define functions for plotting the data using `matplotlib`.


# In[16]:


from mpl_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt # plotting

import numpy as np # linear algebra

import os # accessing directory structure

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


# In[17]:


# Distribution graphs (histogram/bar graph) of column data

def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):

    nunique = df.nunique()

    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have between 1 and 50 unique values

    nRow, nCol = df.shape

    columnNames = list(df)

    nGraphRow = (nCol + nGraphPerRow - 1) // nGraphPerRow

    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')

    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()


# In[18]:


# Correlation matrix
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns') # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1
unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns
({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w',
edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()


# In[19]:


# Scatter and density plots
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()


# Now we are ready to read in the data and use the plotting functions to visualize the data.
# In[20]:

nRowsRead = 1000 # specify 'None' if want to read whole file
df1 = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'pd_speech_features.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')

# Let's take a quick look at what the data looks like:

# In[21]:

df1.head(5)

# Distribution graphs (histogram/bar graph) of sampled columns:

# In[22]:

plotPerColumnDistribution(df1, 10, 5)

# Correlation matrix:

# In[23]:

plotCorrelationMatrix(df1, 188)

# Scatter and density plots:
```

```
# In[24]:


plotScatterMatrix(df1, 20, 10)


# In[25]:


df1.info()


# In[26]:


df1.iloc[0:10,-1]


# In[27]:


df1.columns


# In[28]:


df1['status'].head()


# In[29]:


def plot_roc_curve(y_test, y_pred):
    # calculate the fpr and tpr for all thresholds of the classification
    fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
    roc_auc = metrics.auc(fpr, tpr)
    plt.figure(figsize=(8, 6))

    # method I: plt
    plt.title('Receiver Operating Characteristic',fontsize=14)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
    plt.legend(loc = 'lower right',fontsize=11)
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.005, 1])
    plt.ylim([0, 1.005])
    plt.ylabel('True Positive Rate',fontsize=12)
    plt.xlabel('False Positive Rate',fontsize=12)
    plt.grid(color='r', linestyle='--', linewidth=0.2)
    plt.show()

# In[30]:


import itertools
from sklearn import metrics
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize = (5,5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('01')
    plt.xlabel('01')

# In[31]:

df1.isnull().sum()

# In[32]:

import seaborn as sns
sns.pairplot(df1)

# In[33]:

sns.heatmap(df1.drop(['name'],axis=1))

# In[34]:

sns.pairplot(df1,vars=['NHR','HNR'])
```

```
# In[35]:


# Extract HNR and NHR columns from the dataset
hnr_data = df1["HNR"]
nhr_data = df1["NHR"]
status_data = df1["status"]

# Create a figure and subplot for the box plot
fig, ax = plt.subplots(figsize=(8, 6))

# Group the data by status and plot the box plots for HNR and NHR with different colors
boxplot = ax.boxplot([hnr_data[status_data == 0], hnr_data[status_data == 1],
                      nhr_data[status_data == 0], nhr_data[status_data == 1]],
                     labels=["HNR (Healthy)", "HNR (Parkinsons)", "NHR (Healthy)", "NHR 
(Parkinsons)"],
                     patch_artist=True)

# Set the colors for the box plots
colors = ["lightblue", "darkblue", "lightgreen", "darkgreen"]
for patch, color in zip(boxplot["boxes"], colors):
    patch.set_facecolor(color)

# Set the labels and title
ax.set_ylabel("Value")
ax.set_title("Box Plot of HNR and NHR by Status in Parkinsons Dataset")

# Show the plot
plt.show()
```

```
# In[36]:

sns.pairplot(df1,vars=['Shimmer:APQ3','Shimmer:APQ5','Shimmer:DDA'])

# In[37]:

X=df1
X=X.drop(['status','name'],axis=1)
y=df1['status']

# In[38]:

X.head()

# **Algorithm 1 implementation**

# In[39]:

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score,classification_report,roc_auc_score,confusion_matrix
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# **SVM**

# In[40]:

# Train models
svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
svm_acc=accuracy_score(y_test,y_pred)
svm_roc=roc_auc_score(y_test, y_pred)
svm_recall=recall_score(y_test,y_pred)
svm_precision=precision_score(y_test,y_pred)
print("test accuracy",svm_acc)
print("roc_auc_score",svm_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

# **LogisticRegression**

# In[41]:
```

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
lr_acc=accuracy_score(y_test,y_pred)
lr_roc=roc_auc_score(y_test, y_pred)
lr_recall=recall_score(y_test,y_pred)
lr_precision=precision_score(y_test,y_pred)
print("test accuracy",lr_acc)
print("roc_auc_score",lr_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

# **Random Forest Classifier**

# In[42]:

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
rf_acc=accuracy_score(y_test,y_pred)
rf_roc=roc_auc_score(y_test, y_pred)
rf_precision=precision_score(y_test,y_pred)
rf_recall=recall_score(y_test,y_pred)
print("test accuracy",rf_acc)
print("roc_auc_score",)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
```

```
plot_confusion_matrix(confusion_mtx, "FT")

plot_roc_curve(y_test, y_pred)


# **KNN**


# In[43]:


knn = KNeighborsClassifier()

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

knn_acc=accuracy_score(y_test,y_pred)

knn_roc=roc_auc_score(y_test, y_pred)

knn_recall=recall_score(y_test,y_pred)

knn_precision=precision_score(y_test,y_pred)

print("test accuracy",knn_acc)

print("roc_auc_score",knn_roc)

confusion_mtx = confusion_matrix(y_test, y_pred)

print(classification_report(y_test, y_pred, target_names="FT"))

plot_confusion_matrix(confusion_mtx, "FT")

plot_roc_curve(y_test, y_pred)


# **Final result of algorithm1**


# In[44]:


alg1=pd.DataFrame(columns=['Metric','LogisticRegression','Random Forest
Classifier','SVM','KNN'])

alg1['Metric']=['accuracy','precision','recall','roc_auc_score']

alg1['LogisticRegression']=[lr_acc,lr_precision,lr_recall,lr_roc]
```

```
alg1['Random Forest Classifier']=[rf_acc,rf_precision,rf_recall,rf_roc]
alg1['SVM']=[svm_acc,svm_precision,svm_recall,svm_roc]
alg1['KNN']=[knn_acc,knn_precision,knn_recall,knn_roc]
alg1

# **Algorithm 2 implementation**

# In[45]:

#pca analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the Parkinson's Disease dataset
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/parkinsons/parkinsons.data')

# Separate the target variable from the features
target = data.status
features = data.drop(['name', 'status'], axis=1)

# Scale the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Perform PCA
```

```
pca = PCA()
pca.fit(scaled_features)

# Extract the explained variance ratios and eigenvalues
explained_var_ratios = pca.explained_variance_ratio_
eigenvalues = pca.explained_variance_

# Plot the scree plot
plt.plot(np.cumsum(explained_var_ratios))
plt.xlabel("Number of principal components")
plt.ylabel("Cumulative explained variance ratio")
plt.show()

# Identify the number of principal components that explain at least 85% of the variance
num_components = np.where(np.cumsum(explained_var_ratios) >= 0.85)[0][0] + 1

# Extract the principal components
principal_components = pca.transform(scaled_features)[:, :num_components]

# Create a new dataframe with the principal components
principal_df = pd.DataFrame(data=principal_components, columns=[f"PC{i}" for i in range(1,
num_components+1)])

# Add the target variable to the new dataframe
principal_df['status'] = target.values
print("no of components",num_components)
# Plot the first two principal components
plt.scatter(principal_df['PC1'], principal_df['PC2'], c=principal_df['status'], cmap='coolwarm')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
principal_df

# In[46]:

loadings = pca.components_.T

# For the first principal component, get the variable names with the highest absolute loadings
component_idx = 0
component_loadings = loadings[:, component_idx]
variable_names = list(data.columns)
sorted_variable_names = [variable_names[i] for i in component_loadings.argsort()[::-1]]

# Print the variable names associated with the first principal component
print(sorted_variable_names)

# In[47]:

# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# Load data

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/parkinsons/parkinsons.data')

X=data

X=X.drop(['status','name'],axis=1)

y=data['status']


# Scale the data using StandardScaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Identify variance in every column of data and apply Principal Component Analysis (PCA) to
identify 6 most relevant features to model training, out of 22 attributes

pca = PCA(n_components=5)

X_reduced = pca.fit_transform(X_scaled)


# Split dataset into testing and training sets, where training data is 75% of total

X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.25,
random_state=42)


# Retrain SVM, logistic regression, random forest and KNN models using the reduced feature set
obtained from PCA


svm = SVC()

svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)

svm_acc=accuracy_score(y_test,y_pred)

svm_roc=roc_auc_score(y_test, y_pred)

svm_recall=recall_score(y_test,y_pred)

svm_precision=precision_score(y_test,y_pred)
```

```
print("SVM test accuracy",svm_acc)
print("SVM roc_auc_score",svm_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
lr_acc=accuracy_score(y_test,y_pred)
lr_roc=roc_auc_score(y_test, y_pred)
lr_recall=recall_score(y_test,y_pred)
lr_precision=precision_score(y_test,y_pred)
print("LR test accuracy",lr_acc)
print("LR roc_auc_score",lr_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
rf_acc=accuracy_score(y_test,y_pred)
rf_roc=roc_auc_score(y_test, y_pred)
rf_precision=precision_score(y_test,y_pred)
rf_recall=recall_score(y_test,y_pred)
print("RF accuracy",rf_acc)
print("RF roc_auc_score",)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
knn_acc=accuracy_score(y_test,y_pred)
knn_roc=roc_auc_score(y_test, y_pred)
knn_recall=recall_score(y_test,y_pred)
knn_precision=precision_score(y_test,y_pred)
print("KNN accuracy",knn_acc)
print("KNN roc_auc_score",knn_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

alg2=pd.DataFrame(columns=['Metric','LogisticRegression','Random Forest
Classifier','SVM','KNN'])
alg2['Metric']=['accuracy','precision','recall','roc_auc_score']
alg2['LogisticRegression']=[lr_acc,lr_precision,lr_recall,lr_roc]
alg2['Random Forest Classifier']=[rf_acc,rf_precision,rf_recall,rf_roc]
alg2['SVM']=[svm_acc,svm_precision,svm_recall,svm_roc]
alg2['KNN']=[knn_acc,knn_precision,knn_recall,knn_roc]
alg2
```

```
# **Normalize the Input features X**

# # Correlation

# In[48]:


import seaborn as sb
plt.figure(figsize = (16,5))
corr = pd.DataFrame(X_reduced).corr()
ax= sb.heatmap(corr, cmap="BrBG",annot=True, linewidths=.5)

# In[49]:


# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA

# Load Parkinson's Disease data set from UCI Machine Learning Repository
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data'
data = pd.read_csv(url)

# Separate the features and labels
X=data
X=X.drop(['status','name'],axis=1)
y=data['status']

# Perform PCA on the data
pca = PCA(n_components=5)
```

```
X_pca = pca.fit_transform(X)

# Get the names of the top 5 features with the highest loadings in the first 5 principal components
feature_names = data.columns[1:-1]
component_names = [f'PC{i}' for i in range(1, 6)]
components_df = pd.DataFrame(pca.components_, columns=feature_names,
index=component_names)
top_feature_names = components_df.abs().idxmax(axis=1).values

# Print the names of the top 5 features
print("Top 5 features:")
for i, feature_name in enumerate(top_feature_names):
    print(f"{i+1}. {feature_name}")

# In[50]:



from sklearn.feature_selection import SelectKBest, chi2

# Load Parkinson's Disease data set from UCI Machine Learning Repository
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data'
data = pd.read_csv(url)

X=data
X=X.drop(['status','name'],axis=1)
y=data['status']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaler.fit(X)
X=scaler.transform(X)
kbest = SelectKBest(chi2, k=5)
X_chi2 = kbest.fit_transform(X, y)

# Get the names of the top 5 features with the highest chi-square scores
feature_names = data.columns[1:-1]
scores = kbest.scores_
top_feature_indices = scores.argsort()[::-1][:5]
top_feature_names = feature_names[top_feature_indices]

# Print the names of the top 5 features
print("Top 5 features:")
for i, feature_name in enumerate(top_feature_names):
    print(f"{i+1}. {feature_name}")

# **algorithm3**
#


# In[51]:


# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score,roc_auc_score,confusion_matrix,classification_report

# Load data from the PPPMI and UCI databases

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/parkinsons/parkinsons.data')

X=data

X=X.drop(['status','name'],axis=1)

y=data['status']


# Resample the minority class using up-sampling to balance the dataset

X_resampled, y_resampled = resample(X[y == 1], y[y == 1], replace=True, n_samples=X[y ==
0].shape[0], random_state=42)

X_resampled = pd.concat([X[y == 0], X_resampled])

y_resampled = pd.concat([y[y == 0], y_resampled])


# Scale the data to a common range using Standard Scaler

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_resampled)


# Split dataset into testing and training sets, where training data is 75% of total

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_resampled, test_size=0.25,
random_state=42)


# In[52]:



svm = SVC()

svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
```

```
svm_acc=accuracy_score(y_test,y_pred)
svm_roc=roc_auc_score(y_test, y_pred)
svm_recall=recall_score(y_test,y_pred)
svm_precision=precision_score(y_test,y_pred)
print("SVM test accuracy",svm_acc)
print("SVM roc_auc_score",svm_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
lr_acc=accuracy_score(y_test,y_pred)
lr_roc=roc_auc_score(y_test, y_pred)
lr_recall=recall_score(y_test,y_pred)
lr_precision=precision_score(y_test,y_pred)
print("LR test accuracy",lr_acc)
print("LR roc_auc_score",lr_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
rf_acc=accuracy_score(y_test,y_pred)
```

```
rf_roc=roc_auc_score(y_test, y_pred)
rf_precision=precision_score(y_test,y_pred)
rf_recall=recall_score(y_test,y_pred)
print("RF accuracy",rf_acc)
print("RF roc_auc_score",)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
knn_acc=accuracy_score(y_test,y_pred)
knn_roc=roc_auc_score(y_test, y_pred)
knn_recall=recall_score(y_test,y_pred)
knn_precision=precision_score(y_test,y_pred)
print("KNN accuracy",knn_acc)
print("KNN roc_auc_score",knn_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

alg3=pd.DataFrame(columns=['Metric','LogisticRegression','Random Forest
Classifier','SVM','KNN'])
alg3['Metric']=['accuracy','precision','recall','roc_auc_score']
alg3['LogisticRegression']=[lr_acc,lr_precision,lr_recall,lr_roc]
alg3['Random Forest Classifier']=[rf_acc,rf_precision,rf_recall,rf_roc]
```

```
alg3['SVM']=[svm_acc,svm_precision,svm_recall,svm_roc]
alg3['KNN']=[knn_acc,knn_precision,knn_recall,knn_roc]
alg3

# **algorithm 3 with a different approach**

# In[53]:

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import itertools

from sklearn import datasets, metrics
from sklearn.metrics import confusion_matrix

from sklearn.neighbors import KernelDensity
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.decomposition import FastICA
from sklearn.metrics import accuracy_score, log_loss
import sklearn.metrics as metrics
from sklearn.metrics import classification_report

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

# In[54]:

sns.set_style('whitegrid')
sns.set_context('paper')
sns.set_palette('GnBu_d')
a = sns.catplot(x='status', data=data, kind='count')
a.fig.suptitle('Number of Samples in Each Class', y=1.03)
a.set(ylabel='Number of Samples', xlabel='Have Parkinson')
plt.show()

# In[55]:

from sklearn.model_selection import train_test_split
X=data
X=X.drop(['status','name'],axis=1)
y=data['status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8)

# In[56]:

from sklearn import preprocessing
```

```
min_max_scaler = preprocessing.MinMaxScaler()
X_train = min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)


# In[57]:


from sklearn.dummy import DummyClassifier


# setting up testing and training set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=27)
min_max_scaler = preprocessing.MinMaxScaler()
X_train = min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)


# DummyClassifier to predict only target 0
dummy = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
dummy_pred = dummy.predict(X_test)


# checking unique labels
print('Unique predicted labels: ', (np.unique(dummy_pred)))


# checking accuracy
print('Test score: ', accuracy_score(y_test, dummy_pred))


# In[58]:


# Modeling the data as is
# Train model
from sklearn.linear_model import LogisticRegression
```

```
lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)

# Predict on training set
lr_pred = lr.predict(X_test)

# Checking accuracy
accuracy_score(y_test, lr_pred)

# In[59]:

from sklearn.utils import resample

X=data
X=X.drop(['status','name'],axis=1)
y=data['status']
# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=27)

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
parkinson = X.loc[X['status'] == 1]
not_parkinson = X.loc[X['status'] == 0]
# upsample minority
fraud_upsampled = resample(not_parkinson,
                 replace=True, # sample with replacement
                 n_samples=len(parkinson), # match number in majority class
                 random_state=27) # reproducible results
```

```
# combine majority and upsampled minority

upsampled = pd.concat([parkinson, fraud_upsampled])

y_train_up = upsampled.loc[:,'status']

X_train_up = upsampled.drop(['status'], axis=1)

min_max_scaler = preprocessing.MinMaxScaler()

X_train_up = min_max_scaler.fit_transform(X_train_up)

X_test = min_max_scaler.transform(X_test)

upsampled['status'].value_counts()


# In[60]:


smote = LogisticRegression(solver='liblinear').fit(X_train_up, y_train_up)


smote_pred = smote.predict(X_test)

print("--------------------------------------------------")

print("||==============================================||")

print("|| Oversample Minority Class Accuracy:=> {:.2f} % ||".format(accuracy_score(y_test,
smote_pred)*100))

print("||==============================================||")

print("--------------------------------------------------")


# In[61]:


from imblearn.over_sampling import SMOTE

#

X=data

X=X.drop(['status','name'],axis=1)

y=data['status']

# setting up testing and trainingsets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=27)


min_max_scaler = preprocessing.MinMaxScaler()


X_train = min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)


sm = SMOTE(sampling_strategy='mirity', random_state=27)
# X_train_smote, y_train_smote = sm.fit_resample(X_train, y_train)


# oversampled_train = pd.concat([pd.DataFrame(y_train_smote, columns=['class']),
pd.Datrame(X_train_smote)], axis=1)
# oversampled_train['class'].value_counts()
# oversampled_train oversampled_train


# In[62]:


# smote = LogisticRegression(solver='liblinear').fit(X_train_smote, y_train_smote)
#
smote_pred = smote.predict(X_test)


print("-------------------------------------------------")
print("||=============================================||")
print("|| Oversample Minority Class Accuracy:=> {:.2f} % ||".format(accuracy_score(y_test,
smote_pred)*100))
print("||=============================================||")
print("-------------------------------------------------")


# In[63]:
```

```
X_train = X_train_up
y_train = y_train_up

# In[64]:

def center(X):
    newX = X - np.mean(X, axis = 0)
    return newX

def standardize(X):
    newX = center(X)/np.std(X, axis = 0)
    return newX

# In[65]:

X=data
X=X.drop(['status','name'],axis=1)
y=data['status']

from sklearn.feature_selection import VarianceThreshold

# Create VarianceThreshold object with a variance with a threshold of 0.5
# thresholder = VarianceThreshold(threshold=.5)

# # Conduct variance thresholding
# X_high_variance = thresholder.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=8)
```

```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()
X_train = min_max_scaler.fit_transform(X_train)
X_test = min_max_scaler.transform(X_test)

X = min_max_scaler.transform(X)

plt.style.use('default')
from sklearn.metrics import accuracy_score

# In[66]:


svm = SVC()
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
svm_acc=accuracy_score(y_test,y_pred)
svm_roc=roc_auc_score(y_test, y_pred)
svm_recall=recall_score(y_test,y_pred)
svm_precision=precision_score(y_test,y_pred)
print("SVM test accuracy",svm_acc)
print("SVM roc_auc_score",svm_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

lr = LogisticRegression()
```

```
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
lr_acc=accuracy_score(y_test,y_pred)
lr_roc=roc_auc_score(y_test, y_pred)
lr_recall=recall_score(y_test,y_pred)
lr_precision=precision_score(y_test,y_pred)
print("LR test accuracy",lr_acc)
print("LR roc_auc_score",lr_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
rf_acc=accuracy_score(y_test,y_pred)
rf_roc=roc_auc_score(y_test, y_pred)
rf_precision=precision_score(y_test,y_pred)
rf_recall=recall_score(y_test,y_pred)
print("RF accuracy",rf_acc)
print("RF roc_auc_score",)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
knn_acc=accuracy_score(y_test,y_pred)
knn_roc=roc_auc_score(y_test, y_pred)
knn_recall=recall_score(y_test,y_pred)
knn_precision=precision_score(y_test,y_pred)
print("KNN accuracy",knn_acc)
print("KNN roc_auc_score",knn_roc)
confusion_mtx = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred, target_names="FT"))
plot_confusion_matrix(confusion_mtx, "FT")
plot_roc_curve(y_test, y_pred)

alg3=pd.DataFrame(columns=['Metric','LogisticRegression','Random Forest
Classifier','SVM','KNN'])
alg3['Metric']=['accuracy','precision','recall','roc_auc_score']
alg3['LogisticRegression']=[lr_acc,lr_precision,lr_recall,lr_roc]
alg3['Random Forest Classifier']=[rf_acc,rf_precision,rf_recall,rf_roc]
alg3['SVM']=[svm_acc,svm_precision,svm_recall,svm_roc]
alg3['KNN']=[knn_acc,knn_precision,knn_recall,knn_roc]
alg3
```

4.Snapshots

4.1 work flow

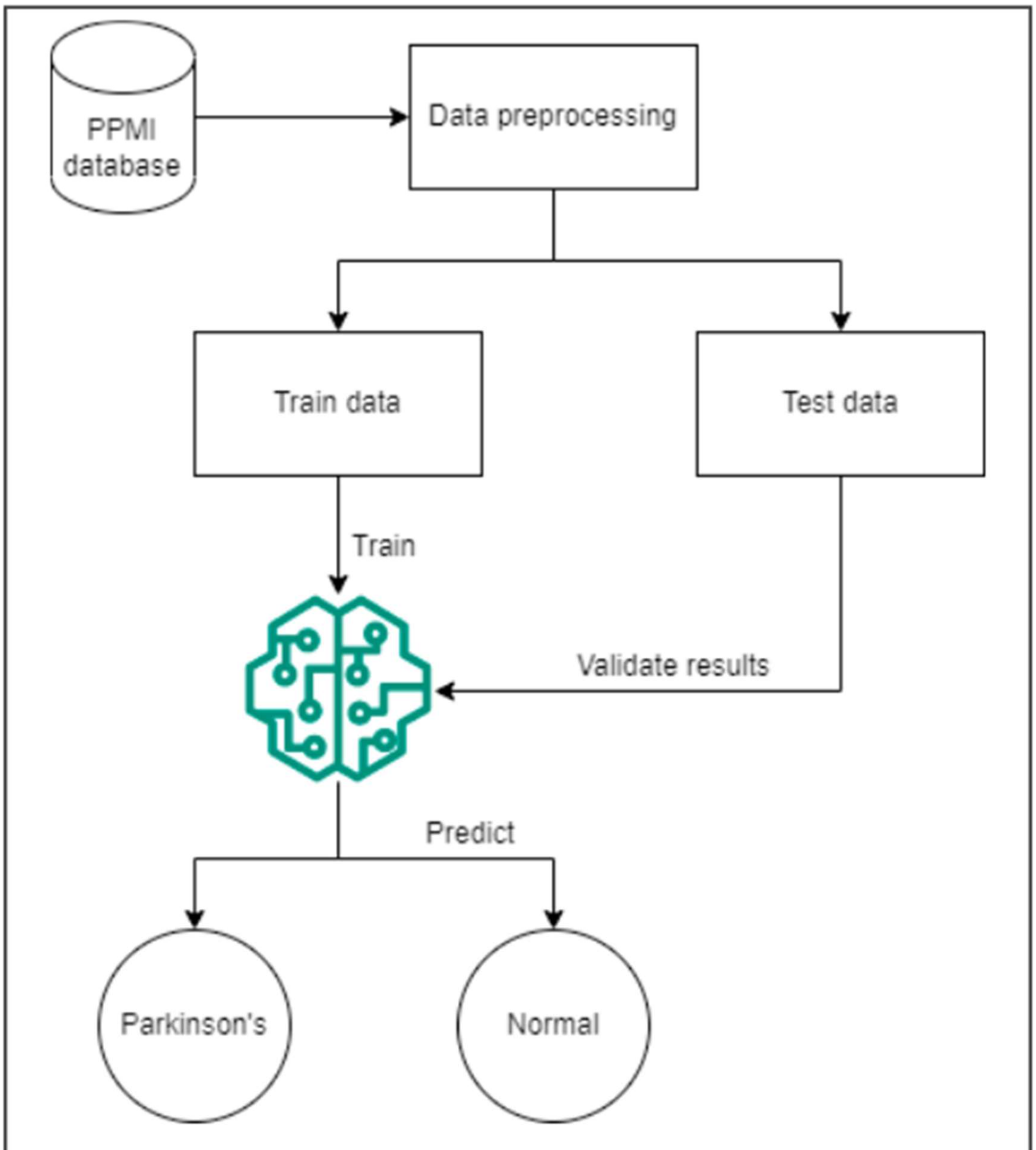
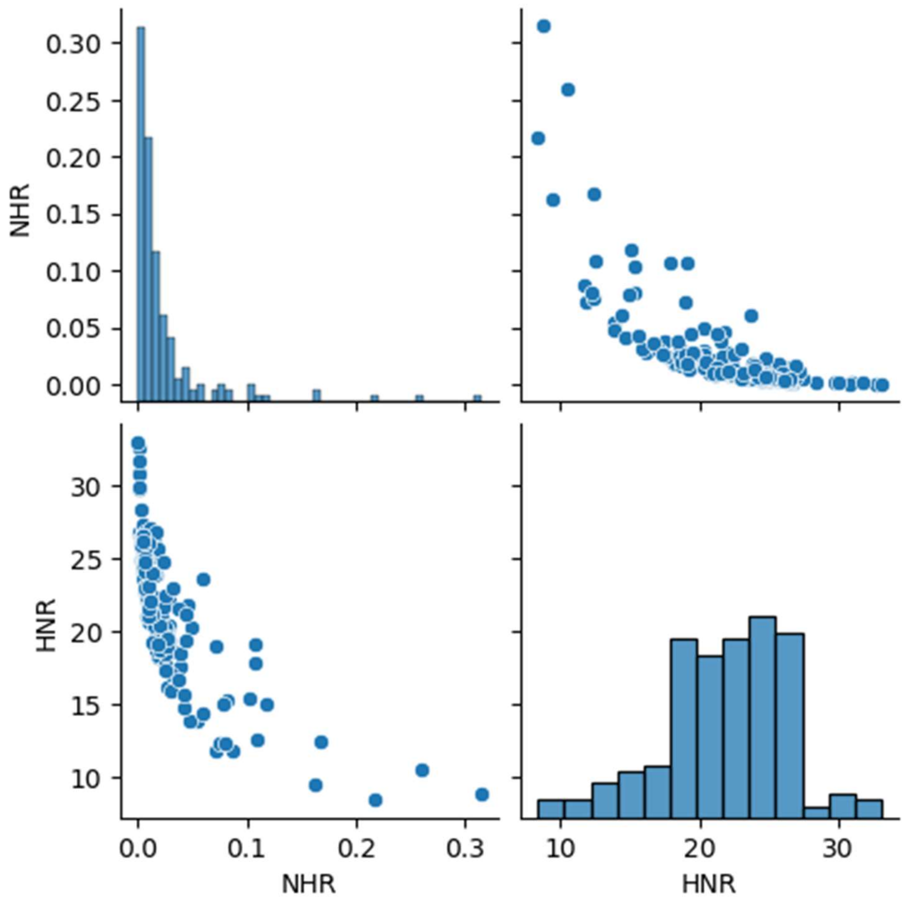


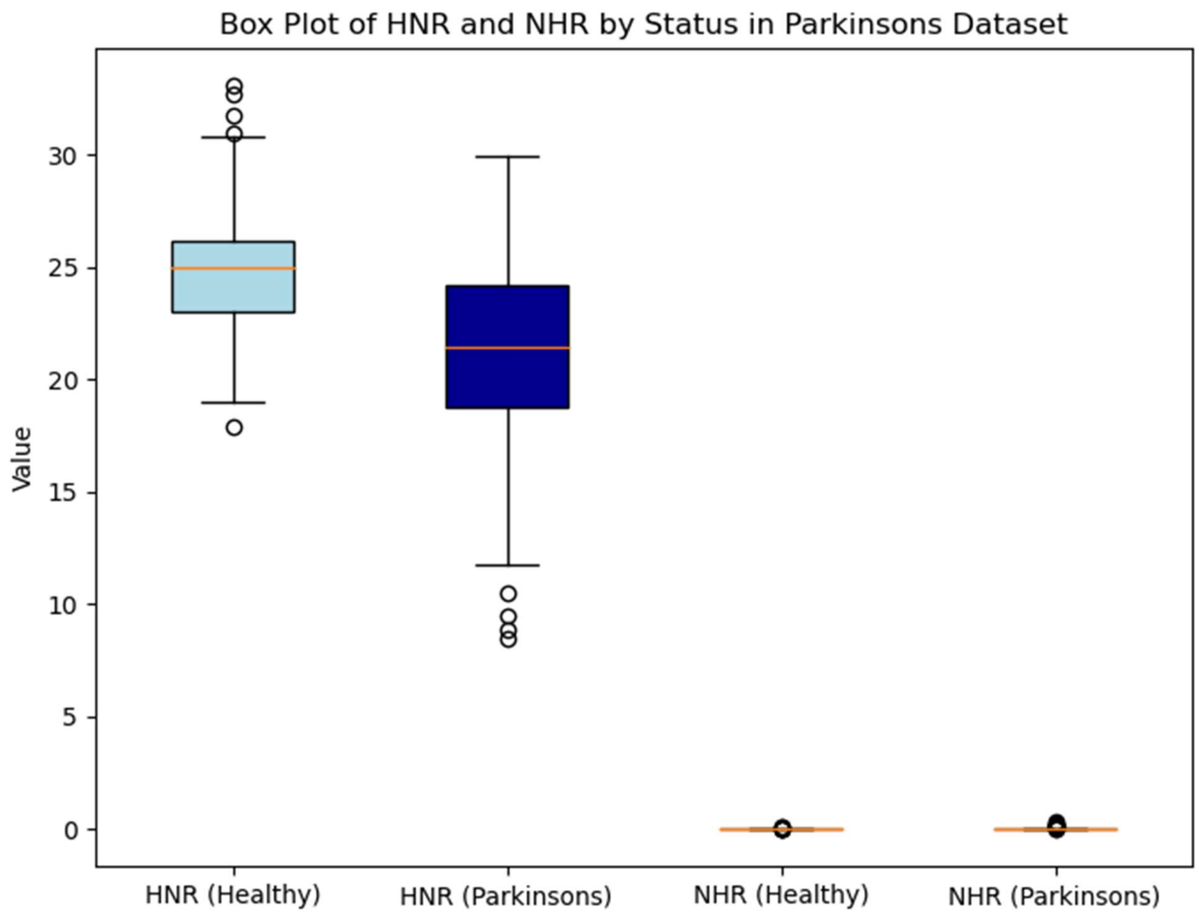
Figure 1: Proposed architecture

4.2 nhr vs hnr

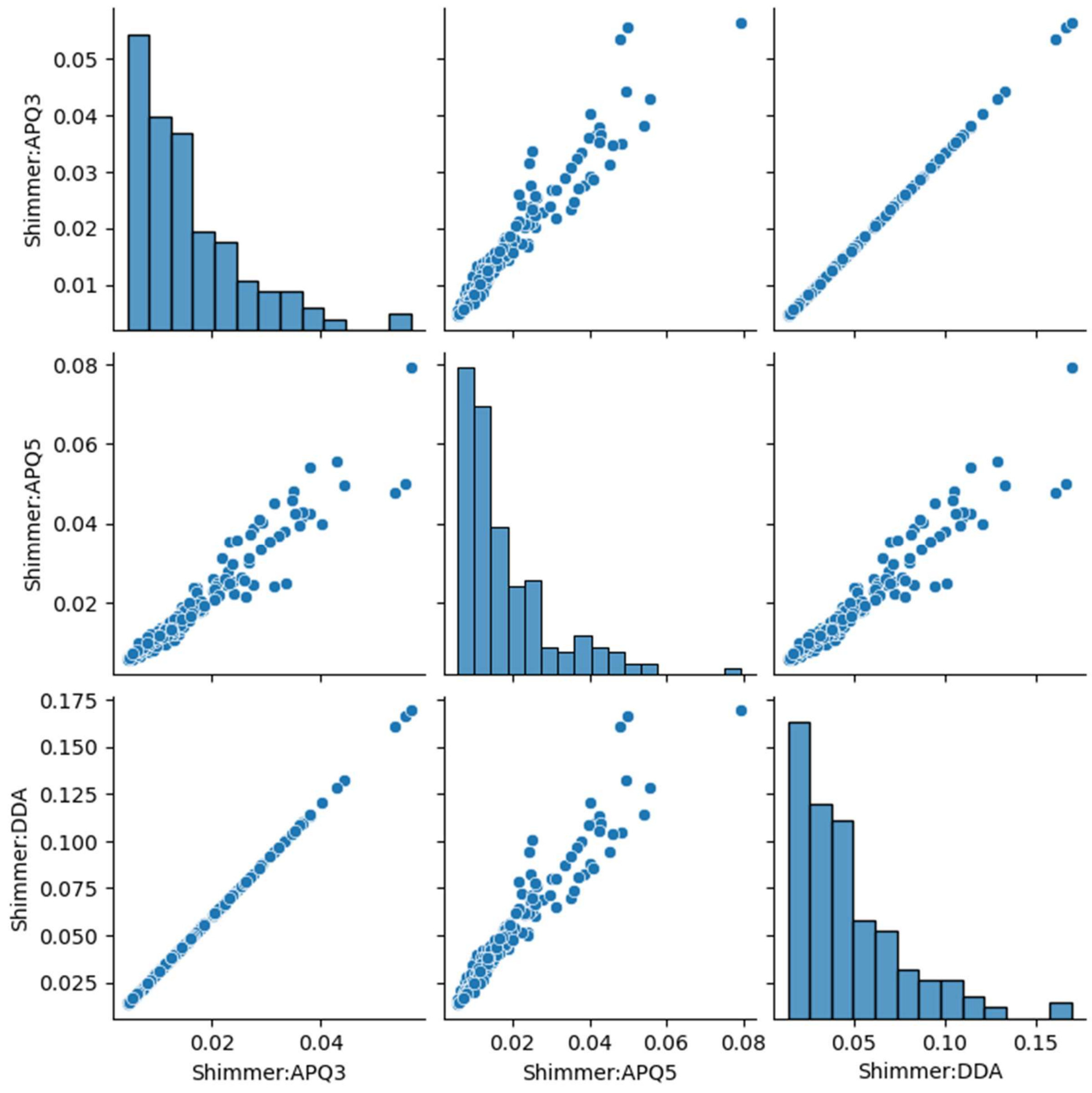


NHR
HNR
0.00
0.05
0.10
0.15
0.20
0.25
0.30
10
15
20
25
30
0.0
0.1
0.2
0.3
NHR
10
20
30
HNR

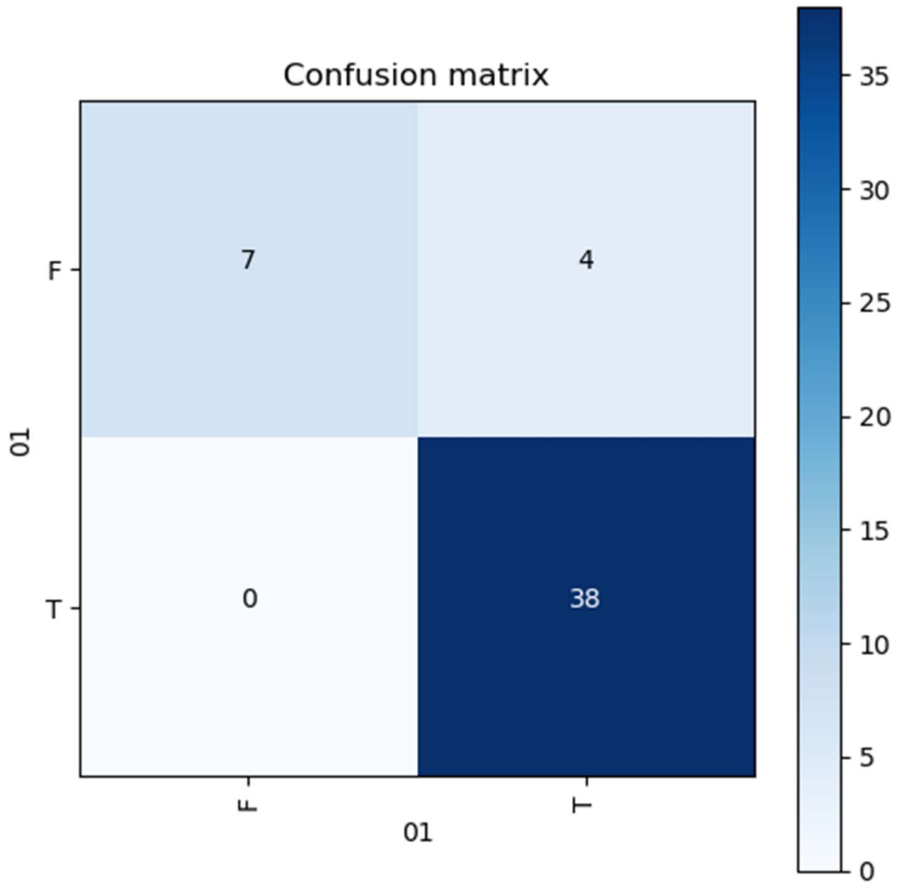
4.3 boxplot of hnr and nhr



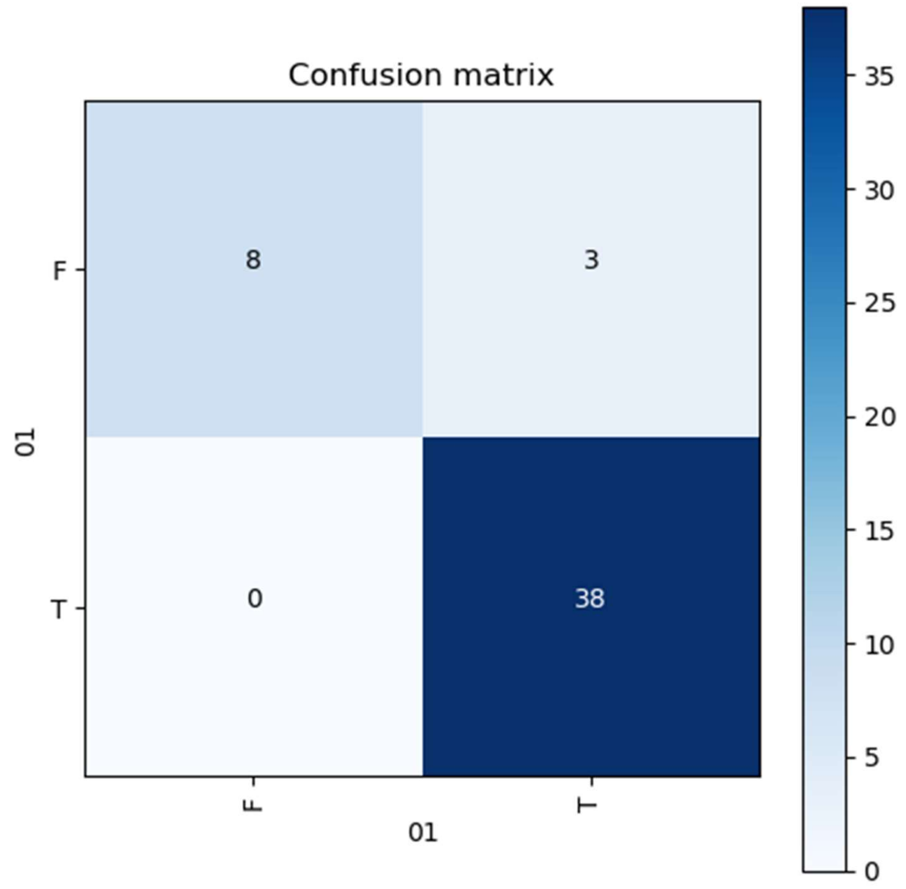
4.4 Shimmer data



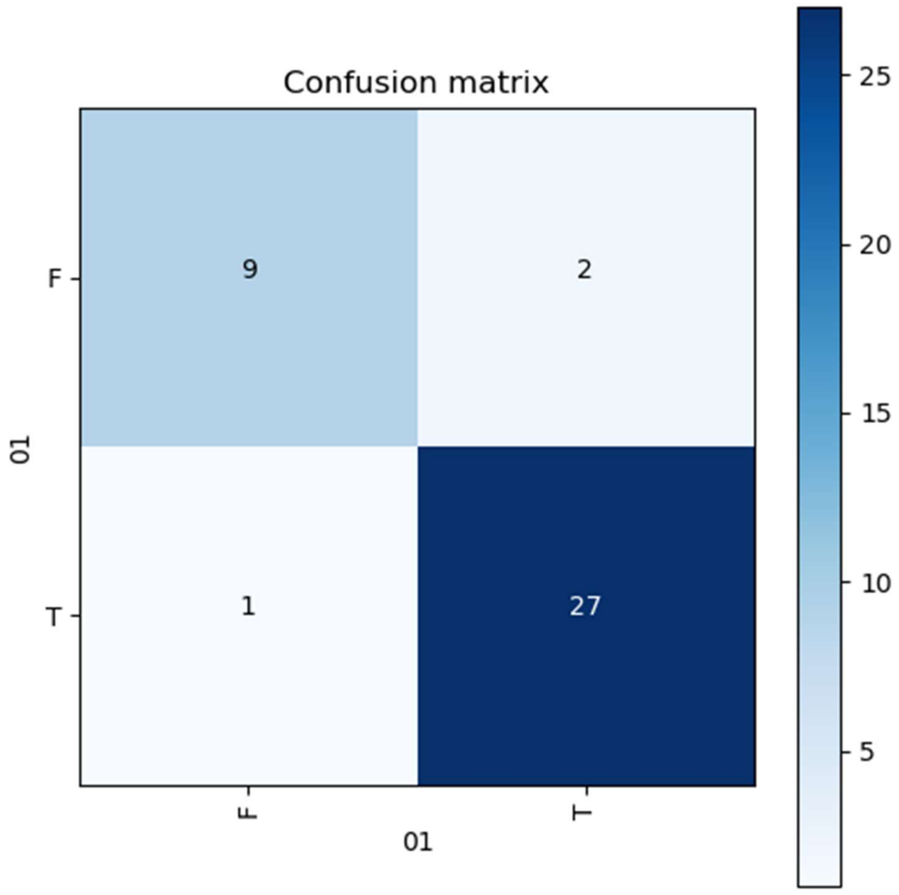
4.5 random forest basic algo confusion matrix



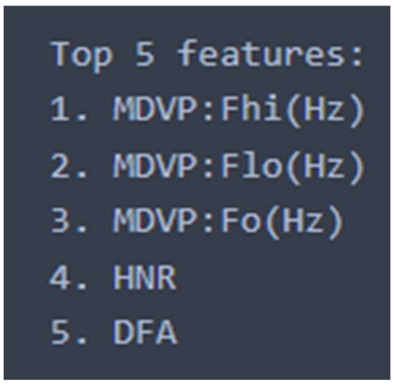
4.6 KNN(pca) approach confusion matrix



4.7 KNN(after balancing using smote) confusion matrix

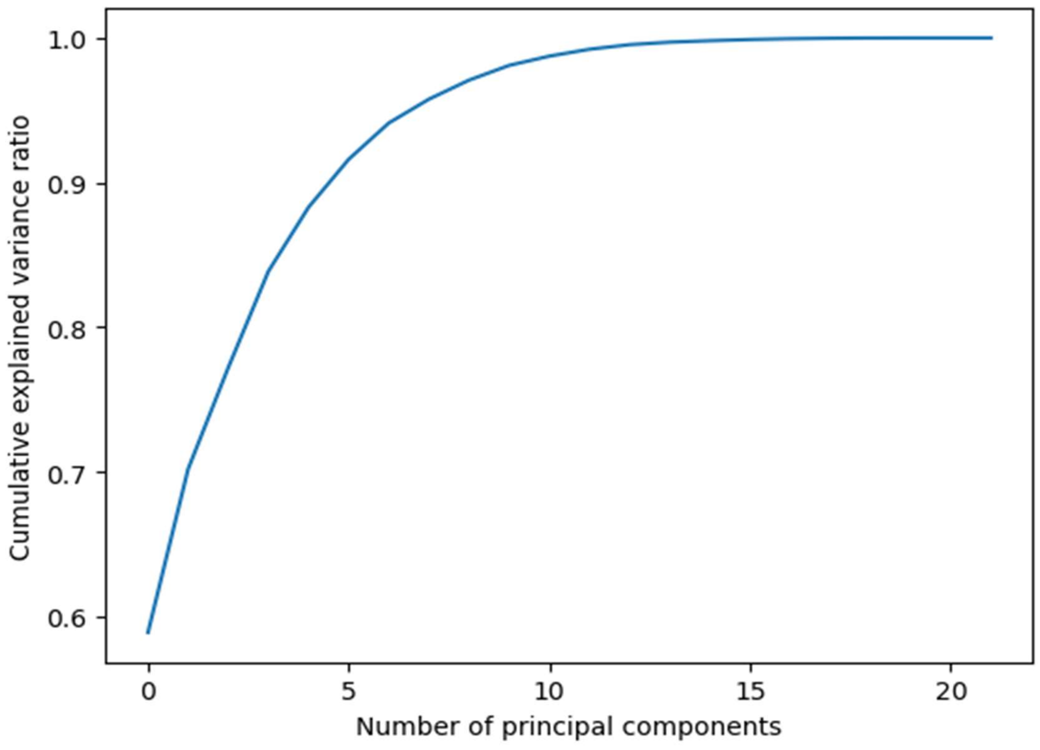


4.8 After pca top 5 features

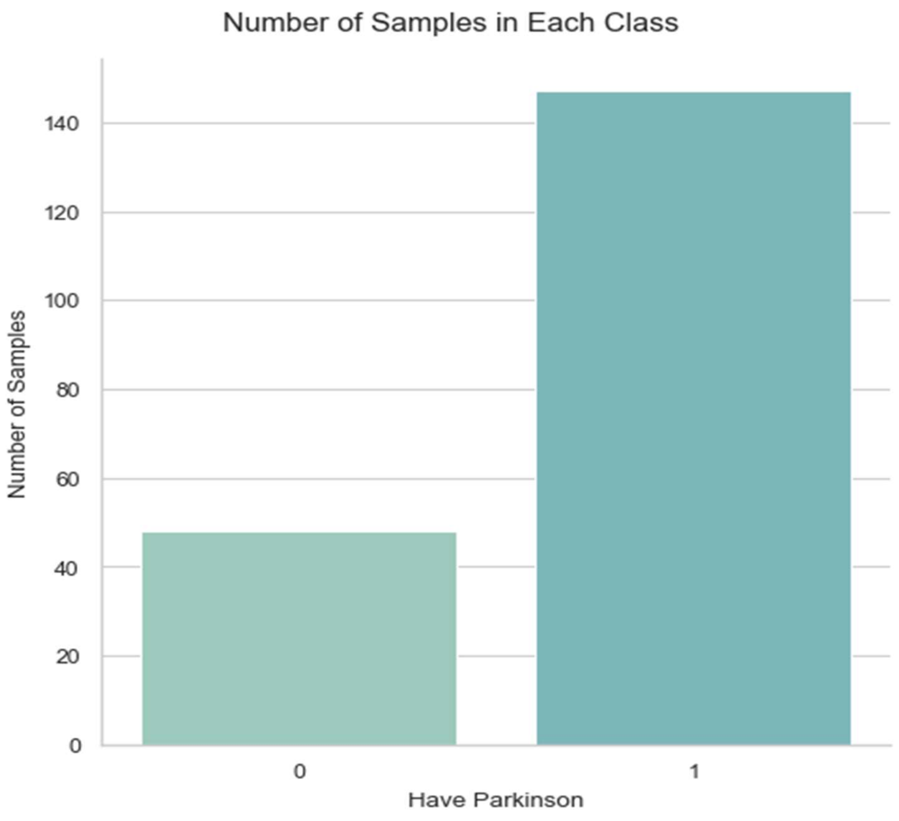


```
Top 5 features:
1. MDVP:Fhi(Hz)
2. MDVP:Flo(Hz)
3. MDVP:Fo(Hz)
4. HNR
5. DFA
```

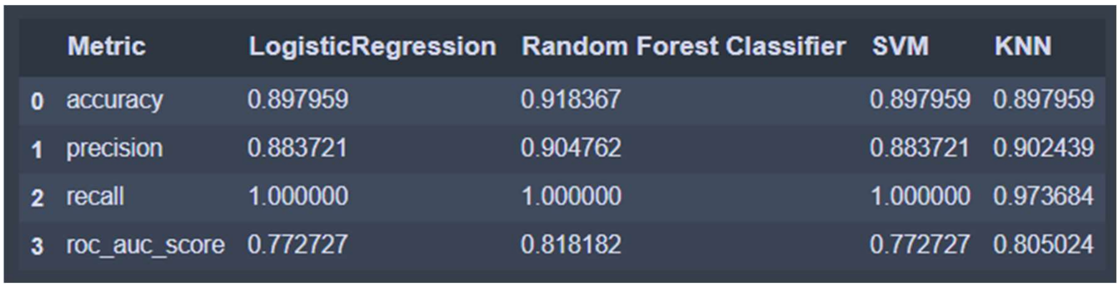
4.9 no of components vs eigen values



4.10 No of samples vs have Parkinson or not

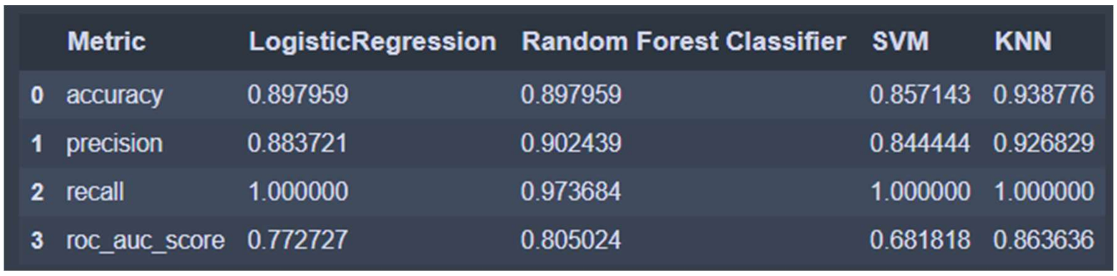


4.11 Algorithm 1 (basic approach)



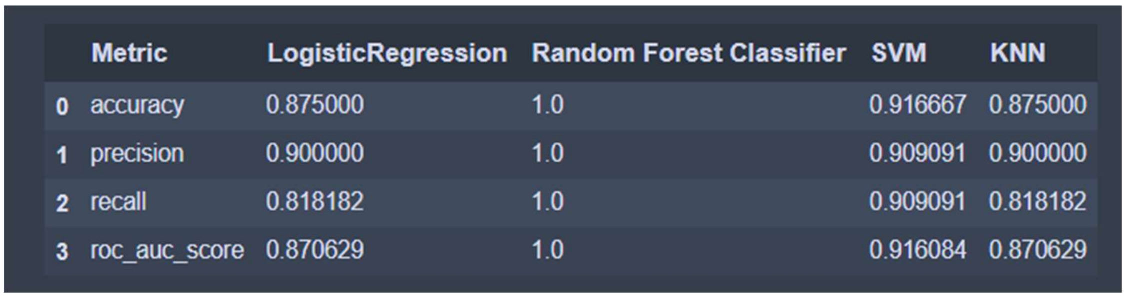
	Metric	LogisticRegression	Random Forest Classifier	SVM	KNN
0	accuracy	0.897959	0.918367	0.897959	0.897959
1	precision	0.883721	0.904762	0.883721	0.902439
2	recall	1.000000	1.000000	1.000000	0.973684
3	roc_auc_score	0.772727	0.818182	0.772727	0.805024

4.12 Algorithm 2 (pca approach)



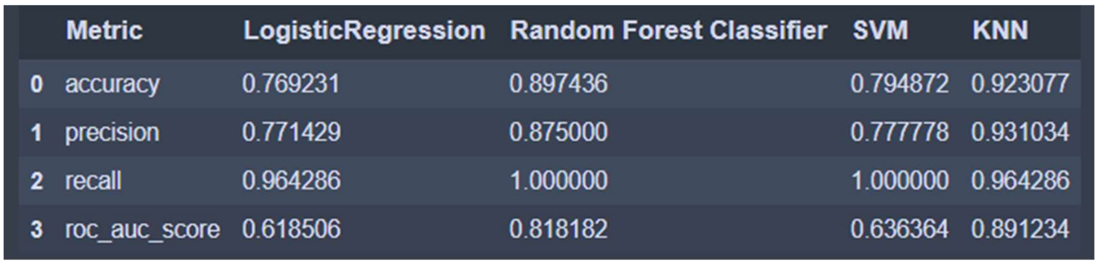
	Metric	LogisticRegression	Random Forest Classifier	SVM	KNN
0	accuracy	0.897959	0.897959	0.857143	0.938776
1	precision	0.883721	0.902439	0.844444	0.926829
2	recall	1.000000	0.973684	1.000000	1.000000
3	roc_auc_score	0.772727	0.805024	0.681818	0.863636

4.13 Algorithm 3 (after balancing)



	Metric	LogisticRegression	Random Forest Classifier	SVM	KNN
0	accuracy	0.875000	1.0	0.916667	0.875000
1	precision	0.900000	1.0	0.909091	0.900000
2	recall	0.818182	1.0	0.909091	0.818182
3	roc_auc_score	0.870629	1.0	0.916084	0.870629

4.14 Custom result using SMOTE similar to algorithm 3



	Metric	LogisticRegression	Random Forest Classifier	SVM	KNN
0	accuracy	0.769231	0.897436	0.794872	0.923077
1	precision	0.771429	0.875000	0.777778	0.931034
2	recall	0.964286	1.000000	1.000000	0.964286
3	roc_auc_score	0.618506	0.818182	0.636364	0.891234

5.Conclusion and Future plans

Top five features after Principal Component analysis observed from our findings are MDVP: Flo (Hz), DFA, D2, MDVP: Fo (Hz), MDVP: Shimmer. Our findings indicate that random forest approach on basic implementation gives 91.83% accuracy and 1.00 recall, whereas KNN on PCA approach gives us 93.8% accuracy with 0.92 precision and 1.00 recall, Also KNN on Balanced approach with SMOTE gives us 92.3% accuracy with 0.93 precision and 0.96 recall.

The results can be improved in the future by combining audio and REM sleep data, as audio data alone is not a sufficient biomarker for Parkinson's disease categorization. These results, we think, will inspire telemedicine to classify PD using mobile recorded audio**.**

6.REFERENCES

[1] TC, Ezhil Selvan, and Vishnu Durai RS. "Prediction of Parkinson's disease using XGBoost." 2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS). Vol. 1. IEEE, 2022.

[2] Fang, Zhaozhao. "Improved KNN algorithm with information entropy for the diagnosis of Parkinson's disease." 2022 International Conference on Machine Learning and Knowledge Engineering (MLKE). IEEE, 2022.

[3] Polat, Kemal. "A hybrid approach to Parkinson disease classification using speech signal: the combination of smote and random forests." 2019 scientific meeting on electrical-electronics & biomedical engineering and computer science (EBBT). Ieee, 2019.

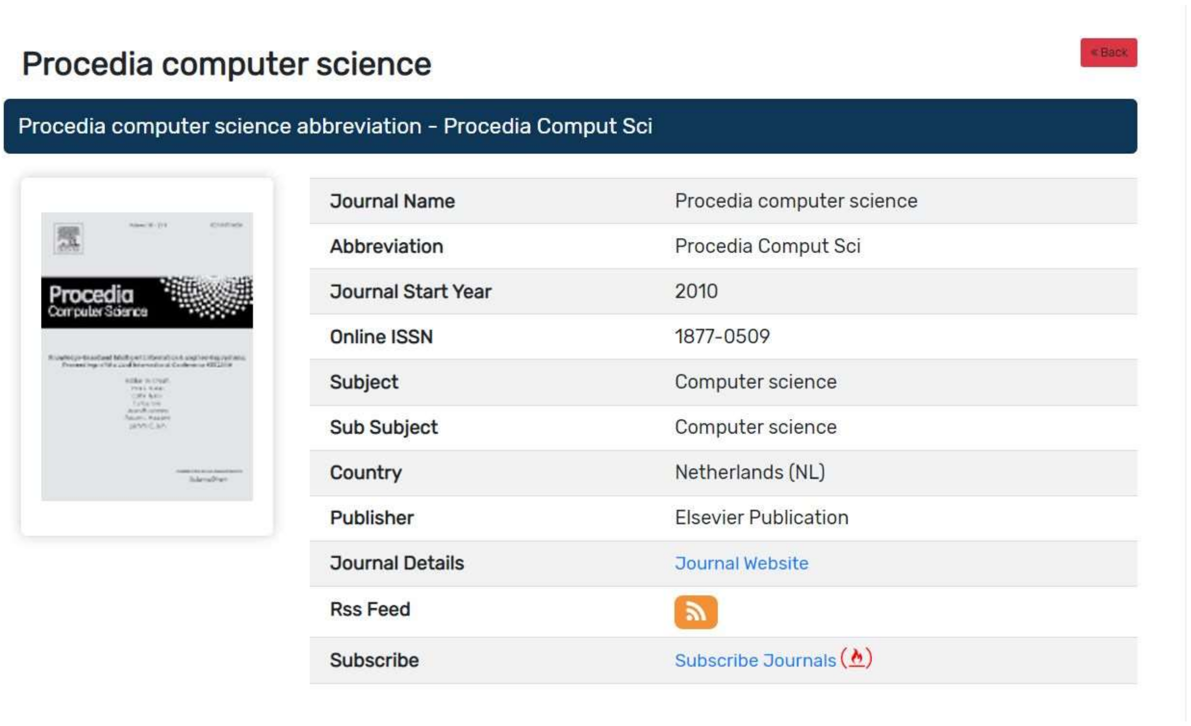
[4] Exley, Trevor, et al. "Predicting UPDRS Motor Symptoms in Individuals With Parkinson's Disease From Force Plates Using Machine Learning." IEEE Journal of Biomedical and Health Informatics 26.7 (2022): 3486-3494.

[5] Patnaik, Debasis, Mavis Henriques, and Ashin Laurel. "Prediction of Parkinson's Disorder: A Machine Learning Approach." 2022 Interdisciplinary Research in Technology and Management (IRTM). IEEE, 2022

7.Appendix-Base paper

Base paper link : https://www.sciencedirect.com/science/article/pii/S1877050923000078

Data set link : https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/parkinsons.data



Procedia computer science

Procedia computer science abbreviation - Procedia Comput Sci

Journal Name	Procedia computer science
Abbreviation	Procedia Comput Sci
Journal Start Year	2010
Online ISSN	1877-0509
Subject	Computer science
Sub Subject	Computer science
Country	Netherlands (NL)
Publisher	Elsevier Publication
Journal Details	Journal Website
Rss Feed	
Subscribe	Subscribe Journals