**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**

i

**THANJAVUR, TAMIL NADU, INDIA- 613401**

**Bonafide Certificate**

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**

ii

**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.

iii

**Table of Contents**

**Title** **Page No.**

Bonafide Certificate ii

Acknowledgements iii

List of Figures iv

Abbreviations v

Abstract vi

1. Summary of the base paper 1

2 Merits and Demerits of the base paper 4

3 Source Code 5

4 Snapshots 47

5 Conclusion and Future Plans 56

6 References 57

7 Appendix -Base Paper 58

**LIST OF FIGURES**

4.1 work flow

4.2 nhr vs hnr

4.3 box plot of nhr and hnr

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

4.9 no of components vs eigen values

4.10 No of samples vs have Parkinson or not

4.11 Algorithm 1 (basic approach)

4.12 Algorithm 2 ( pca approach)

4.13 Algorithm 3 (after balancing)

4.14 Custom result using SMOTE similar to alg3

iv

**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

v

**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*

vi

**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. 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. 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

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. 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.

2

* 1. 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. 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. 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.

3

**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.

4

**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

5

*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':

6

            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)

7

        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>

8

<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>

9

3.4 styles.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;

10

*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;

}

11

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;

}

12

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*)):

13

        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)

14

    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.

15

# 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:

16

# 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

17

    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)

18

    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'])

19

# 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()

20

# 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)

21

# 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]:

22

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"))

23

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]

24

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

25

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')

26

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

27

# 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)

28

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)

29

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

30

# \*\*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)

31

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()

32

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

33

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)

34

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)

35

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]

36

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

37

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

38

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

39

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

40

# 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

41

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]:

42

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)

43

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()

44

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)

45

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]

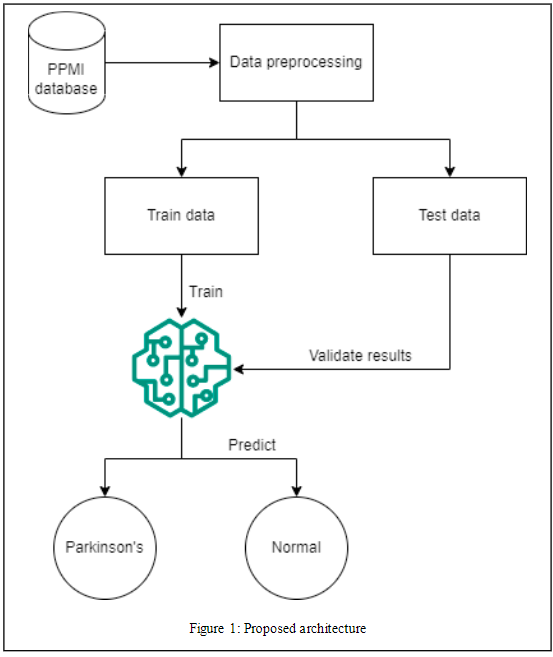
alg3['KNN']=[knn\_acc,knn\_precision,knn\_recall,knn\_roc]

alg3

46

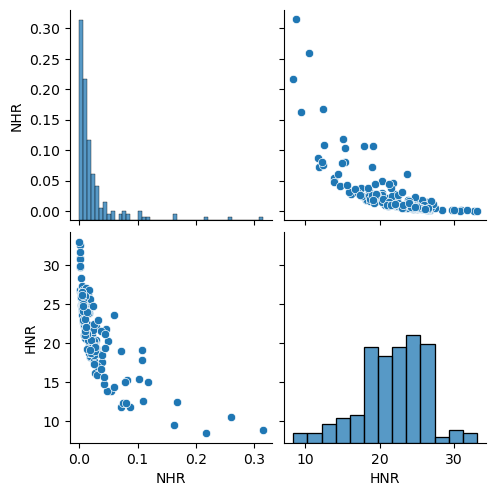
**4.Snapshots**

4.1 work flow



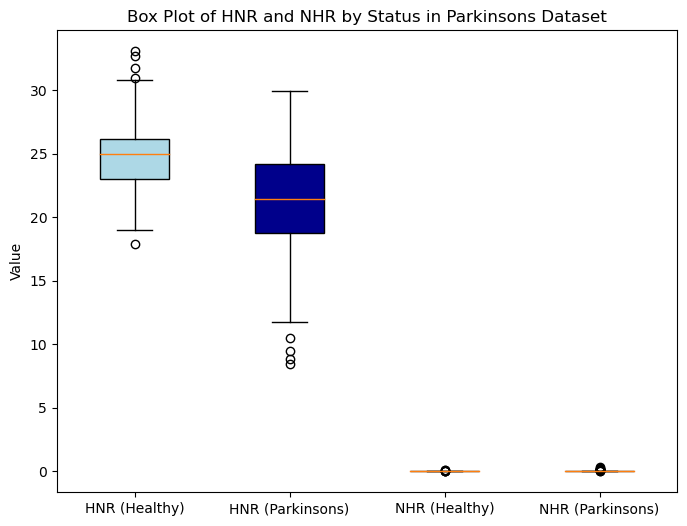
47

4.2 nhr vs hnr



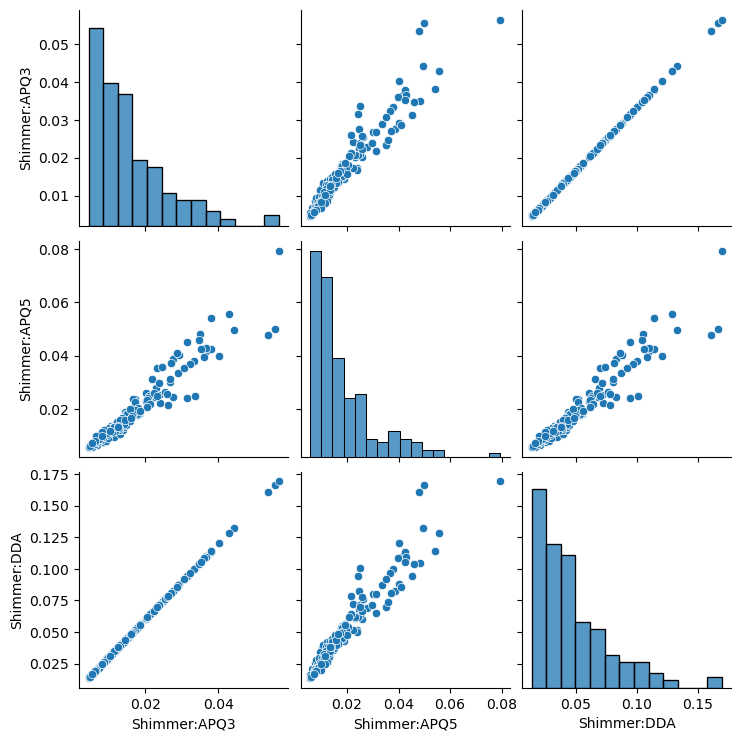
48

4.3 boxplot of hnr and nhr



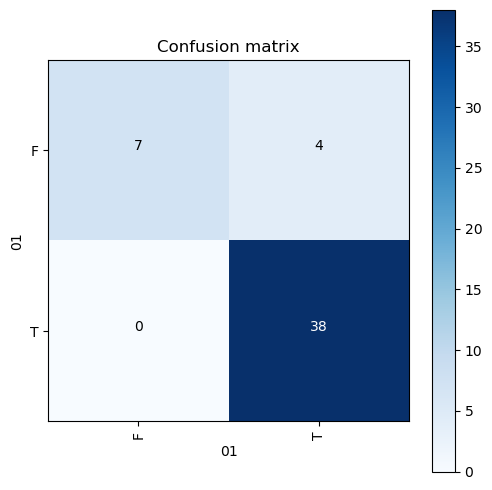
49

4.4 Shimmer data



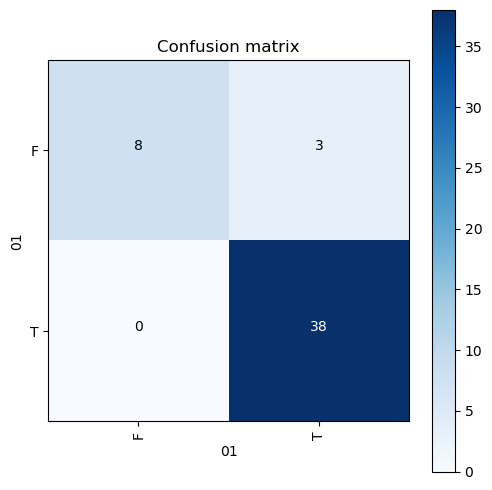
50

4.5 random forest basic algo confusion matrix



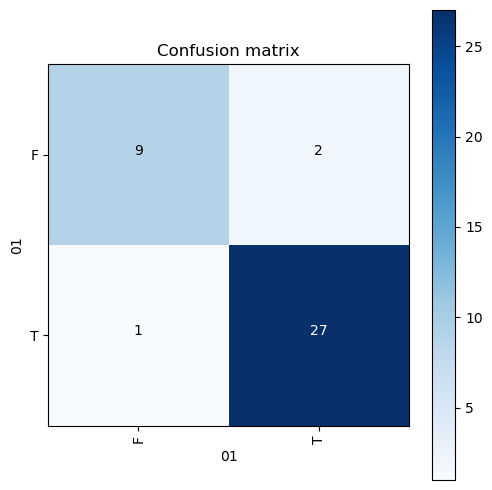
51

4.6 KNN(pca) approach confusion matrix

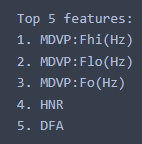


52

4.7 KNN(after balancing using smote) confusion matrix

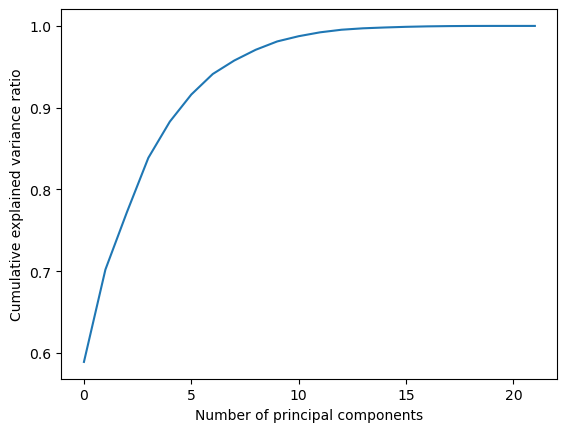


4.8 After pca top 5 features

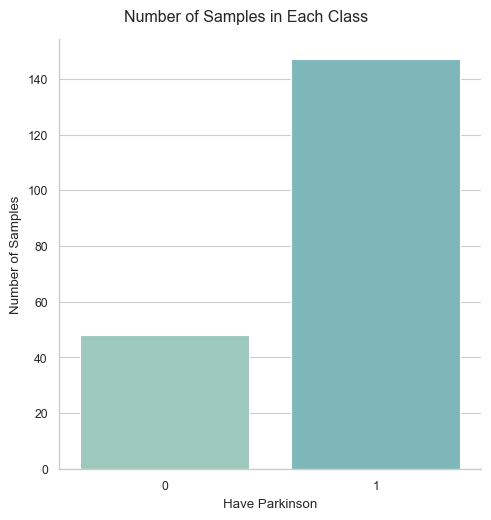


53

4.9 no of components vs eigen values

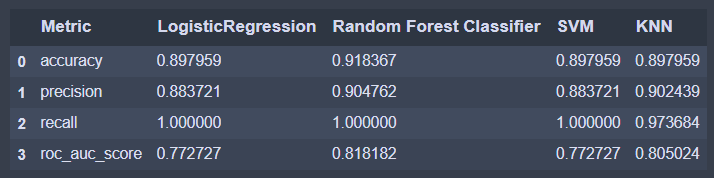


4.10 No of samples vs have Parkinson or not



54

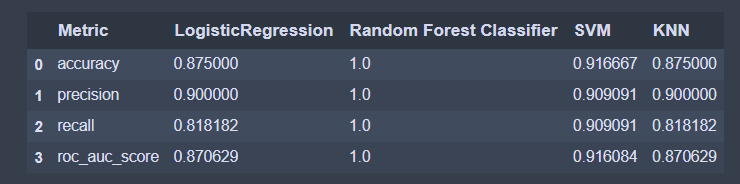
4.11 Algorithm 1 (basic approach)



4.12 Algorithm 2 ( pca approach)



4.13 Algorithm 3 (after balancing)



4.14 Custom result using SMOTE similar to algorithm 3



55

**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**.**

56

**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.

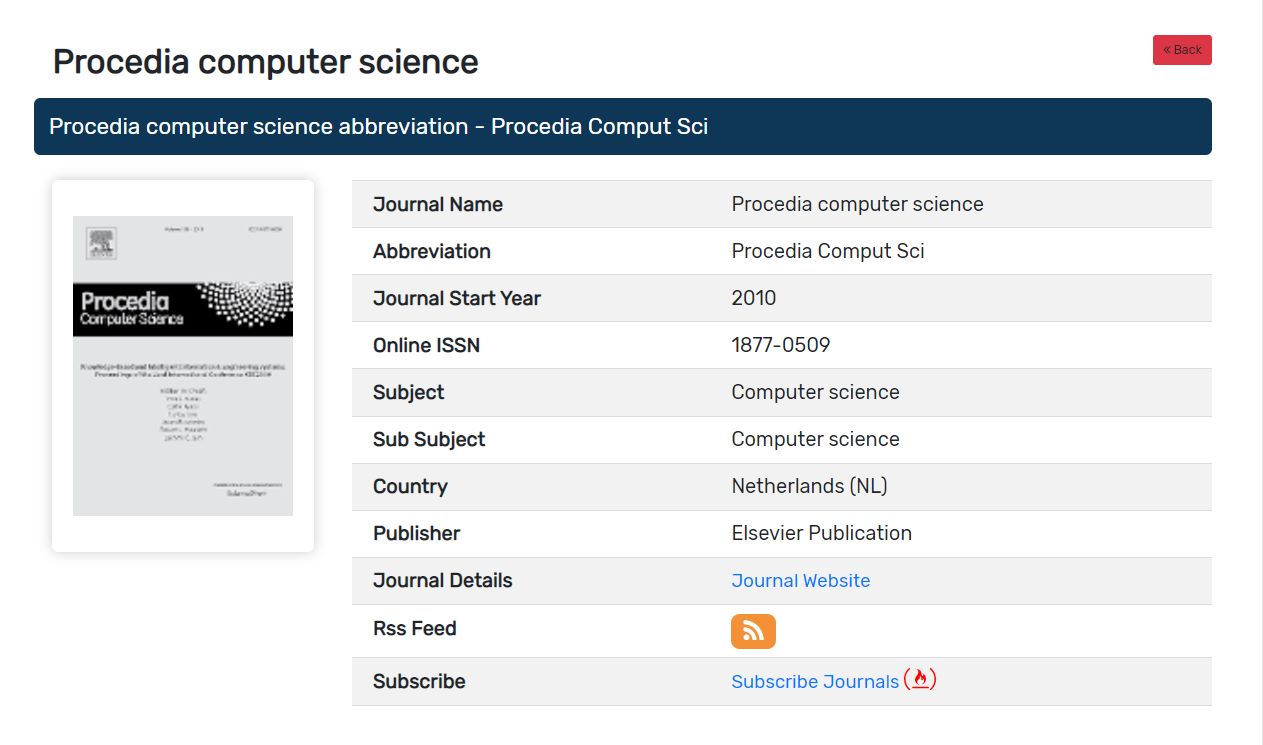
[5] Patnaik, Debasis, Mavis Henriques, and Ashin Laurel. "Prediction of Parkinson's Disorder: A Machine Learning Approach." 2022 Interdisciplinary Research in Technology and Management (IRTM). IEEE, 2022

57

**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>



58