TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Problem Definition

Parkinsonism has vocal disorders problems that affect their speech volume level and face complexity in the pronunciation of syllables and so forth. Thus, to use vocal measurements as an effective diagnostic tool for PD recognition Parkinson disease is the critical disorder sickness second to Alzheimer's disease and the complete PD treatment has not discovered till now. The existing technique of therapies is good for tackle PD symptoms. However, researchers have made attempts to find out the effective treatment strategy that ensures recovery and treatment. In the PD diagnosis is being typically based on conducted few invasive techniques and empirical tests and examinations.

1.2 Methodologies

The methodology of the proposed system is structured into following steps:

Exploratory data analysis : We use scatter matrix plot, Pearson correlation heatmap and box plots to analyse the dataset properties.

Pre-processing of the dataset : For pre-processing of dataset we make use of MinMax scaler method.

Features Selection: For feature selection we used L1-Norm SVM algorithm.

Machine learning classifier : We explore three different classifiers in our report, they are Decision Tree Classifier, Extra Tree Classifier and SVM Classifier.

Cross - Validation : For Cross Validation we have made use of K-Fold Cross Validation method.

All of the above methods and modules will we explained in detail in further chapters of this report.

1.3 Outline of the results

The output of the project is represented as various accuracy scores as a result of the use of L1 norm SVM, Decision Tree and Extra Tree classifiers. A detailed comparison is done between the results and it is represented in the form of various colourful bar graphs.

1.4 Scope of the project

Parkinson's Disease (PD) is a neuro-degenerative disorder of central nervous system that causes partial or full loss in motor reflexes, speech, behavior, mental processing, and other vital functions. It is generally observed in elderly people and causes disorders in speech and motor abilities (writing, balance, etc.) of 90% of the patients. At present, there is no cure for PD, although medication and surgical intervention may alleviate some of the symptoms and improve quality of life for most. However, early diagnosis and frequent disease tracking are critical to maximizing the effect of treatment. This is when the technology comes into picture. Machine learning is doing wonders in all the fields and has been used to remove the constraints in the medical field too. Using a machine learning based system, we can predict the PD much earlier which would result in the early detection of the disease. Early detection would in turn result in minimizing the post effects of the disease.

1.5 Technologies/Tools Used

1. **Python** : Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

- Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read − Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain − Python's source code is fairly easy-to maintain.
- A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases − Python provides interfaces to all major commercial databases.
- GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable − Python provides a better structure and support for large programs than shell scripting.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2. **Jupyter Notebook:** The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

1.6 Organization of the report

The project has been divided into the following components:

- **Chapter 1**: Introduction. Describes the main idea i.e., objective and methodologies of this project.
- **Chapter 2**: Literature Survey. This section provides a clear idea about the existing system and the motivation that leads us to the present project.
- **Chapter 3**: Design of Proposed System. Gives basic idea of the process and methods to be followed in order to implement this project.
- **Chapter 4**: Implementation of Proposed System. Gives a detailed description of individual processes within the project.
- **Chapter 5**: Results and Discussion. This section will show the outputs obtained as a result of implementing the procedures explained in previous sections.
- **Chapter 6**: Conclusion and Future Work. We have given our observations on the results obtained and what future work we intend to do to continue this project.
- **Chapter 7**: References.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction to the Problem domain

Parkinson's disease is the critical disorder sickness second to Alzheimer's disease and the complete PD treatment has not discovered till now. Presently doctors use methods like lab tests which include blood tests, imaging tests such as MRI, CT scan, Ultrasound of brain and CET scan. These methods are effective in detecting Parkinson's disease, but they only detect it after the patient has been well affected by the disease. Parkinsonism has vocal disorders problems that affect their speech volume level and face complexity in the pronunciation of syllables and so forth. Thus, to use vocal measurements as an effective diagnostic tool for PD recognition. The existing technique of therapies is good for tackle PD symptoms. However, researchers have made attempts to find out the effective treatment strategy that ensures recovery and treatment. In the PD diagnosis is being typically based on conducted few invasive techniques and empirical tests and examinations. The invasive based techniques in order to diagnose the PD are very expensive, less efficient, as well as very complex equipment’s needed to conducts and the accuracy is also not satisfactory.

2.2 Existing System

The existing systems for the classification of the patients suffering from Parkinson’s disease involves the medical related tests as well as a few machine learning analysis techniques. Taking the medical test procedures into account, we can say that they require the physical presence of the patient no matter what the condition of the patient is. These medical results take more than 2 to 3 days for giving the final report of the patient. Other than the mentioned reasons, the main rationale why the medical examination is a drawback is because it is highly expensive and not all can afford it.

Talking about the existing analysis methods which was developed in machine learning use various methods like LASSO, Random Forest and Local learning-based feature selection. These machine learning methods surely overcome the disadvantages of the medical tests and provide reliable reports. Various features were used for classification of the patients suffering from Parkinson's disease which resulted in different analysis with varying different scores

2.3 Proposed System

The main contribution of this detection of Parkinson's disease is to propose a machine-learning based system to successfully diagnose people with PD and improve the patient's life. Machine learning predictive model SVM is used for PD and healthy people classification with better accuracy scores. The L1-Norm SVM is used for appropriate features selection that improves the classification performance of the classifier. We adopted the L1-Norm SVM for appropriate feature selection because the classification performance of L1-Norm SVM FS based method resulted in the most reliable reports when compared to other methods of classification for PD and healthy people.

Following are the key contributions of the proposed study :

1. The performance of classifier checked on selected features subsets which are selected by L1-Norm SVM algorithm along with K–folds cross-validation technique.
2. The performance also checked on full features set and compared with performance on selected features sets.
3. The system has been tested on PD dataset and achieved high classification performance.

2.4 Related works

In Reference [1]

Non-Invasion Speech tests

The non-invasion speech test was implemented by Tsanas A, Little MA, McSharry PE, Ramig LO. Tracking Parkinson's disease (PD) symptom progression often uses the unified Parkinson's disease rating scale (UPDRS) that requires the patient's presence in clinic, and time-consuming physical examinations by trained medical staff. Thus, symptom monitoring was costly and logistically inconvenient for patient and clinical staff alike, also hindering recruitment for future large-scale clinical trials. Here, for the first time, remote replication of UPDRS assessment with clinically useful accuracy (about 7.5 UPDRS points difference from the clinicians' estimates), using only simple, self-administered, and non-invasive speech tests. They were able to characterize speech with signal processing algorithms, extracting clinically useful features of average PD progression. Subsequently, they select the most parsimonious model with a robust feature selection algorithm, and statistically map the selected subset of features to UPDRS using linear and nonlinear regression techniques that include classical least squares and nonparametric classification and regression trees. At last verifications of the findings are done on the largest database of PD speech in existence (approximately 6000 recordings from 42 PD patients, recruited to a six-month, multicentre trial). These findings support the feasibility of frequent, remote, and accurate UPDRS tracking. This technology could play a key part in telemonitoring frameworks that enable large-scale clinical trials into novel PD treatments.

In Reference [2]

Adaptive Deep Brain Stimulation in Advanced Parkinson Disease

Deep brain stimulation (DBS) is an established treatment for severe Parkinson disease (PD), dystonia, and tremor, and has an emerging role in a range of other neurological and neuropsychiatric conditions. However, its widespread adoption is at present limited by cost, side effects, and partial efficacy. They diagnosed 23 PD and 8 healthy people and their dataset recorded vowels and used a Support Vector Machine (SVM) for classification and achieved classification accuracy 91.4 %. In 132 extracted features from speech signals applied dysphonia methods. The database only contained vowels and some features extraction algorithms such as Least Absolute Shrinkage Selection Operator (LASSO), Minimal Redundancy Maximal Relevance (MRMR), Relief and Local Learning Based Feature Selection (LLBFS), were used and 10 features selected from 132 were selected by FS algorithms. These 10 features were used as input parameters for classification with two machine learning algorithms (Random Forests and SVM).

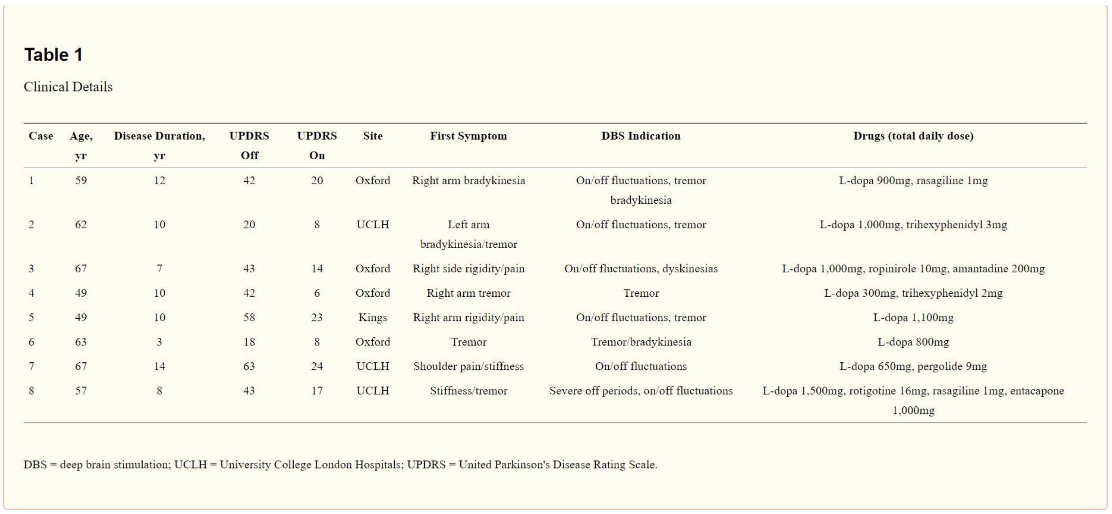


Table 1

Clinical Details

Case	Age, yr	Disease Duration, yr	UPDRS Off	UPDRS On	Site	First Symptom	DBS Indication	Drugs (total daily dose)
1	59	12	42	20	Oxford	Right arm bradykinesia	On/off fluctuations, tremor bradykinesia	L-dopa 900mg, rasagiline 1mg
2	62	10	20	8	UCLH	Left arm bradykinesia/tremor	On/off fluctuations, tremor	L-dopa 1,000mg, trihexyphenidyl 3mg
3	67	7	43	14	Oxford	Right side rigidity/pain	On/off fluctuations, dyskinesias	L-dopa 1,000mg, ropinirole 10mg, amantadine 200mg
4	49	10	42	6	Oxford	Right arm tremor	Tremor	L-dopa 300mg, trihexyphenidyl 2mg
5	49	10	58	23	Kings	Right arm rigidity/pain	On/off fluctuations, tremor	L-dopa 1,100mg
6	63	3	18	8	Oxford	Tremor	Tremor/bradykinesia	L-dopa 800mg
7	67	14	63	24	UCLH	Shoulder pain/stiffness	On/off fluctuations	L-dopa 650mg, pergolide 9mg
8	57	8	43	17	UCLH	Stiffness/tremor	Severe off periods, on/off fluctuations	L-dopa 1,500mg, rotigotine 16mg, rasagiline 1mg, entacapone 1,000mg

DBS = deep brain stimulation; UCLH = University College London Hospitals; UPDRS = United Parkinson's Disease Rating Scale.

Fig 2.4.1 - Clinical Details table

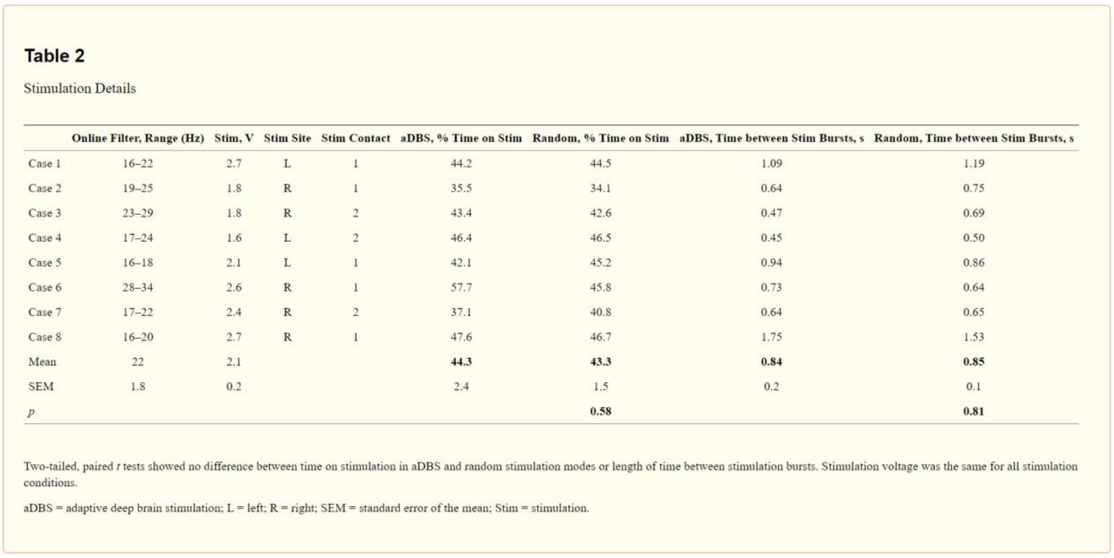


Table 2

Stimulation Details

	Online Filter, Range (Hz)	Stim, V	Stim Site	Stim Contact	aDBS, % Time on Stim	Random, % Time on Stim	aDBS, Time between Stim Bursts, s	Random, Time between Stim Bursts, s
Case 1	16–22	2.7	L	1	44.2	44.5	1.09	1.19
Case 2	19–25	1.8	R	1	35.5	34.1	0.64	0.75
Case 3	23–29	1.8	R	2	43.4	42.6	0.47	0.69
Case 4	17–24	1.6	L	2	46.4	46.5	0.45	0.50
Case 5	16–18	2.1	L	1	42.1	45.2	0.94	0.86
Case 6	28–34	2.6	R	1	57.7	45.8	0.73	0.64
Case 7	17–22	2.4	R	2	37.1	40.8	0.64	0.65
Case 8	16–20	2.7	R	1	47.6	46.7	1.75	1.53
Mean	22	2.1			**44.3**	**43.3**	**0.84**	**0.85**
SEM	1.8	0.2			2.4	1.5	0.2	0.1
p						**0.58**		**0.81**

Two-tailed, paired *t* tests showed no difference between time on stimulation in aDBS and random stimulation modes or length of time between stimulation bursts. Stimulation voltage was the same for all stimulation conditions.

aDBS = adaptive deep brain stimulation; L = left; R = right; SEM = standard error of the mean; Stim = stimulation.

Fig 2.4.2 - Stimulation Details table

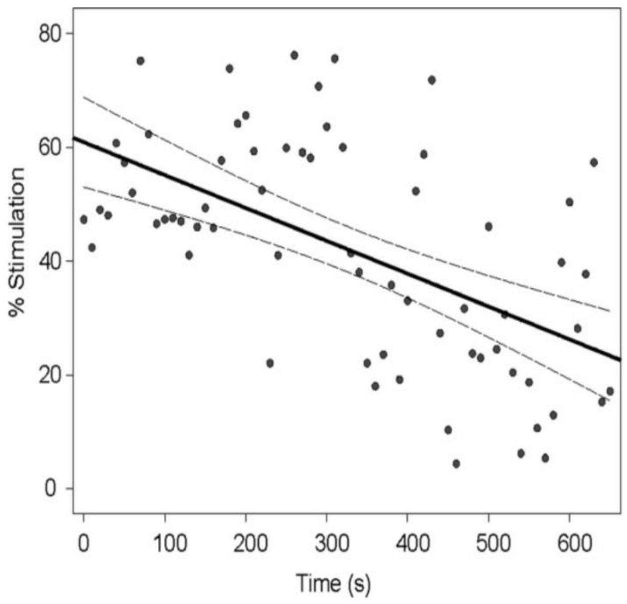


Fig 2.4.3 - Percentage Stimulation v/s Time(s) graph

LASSO

In statistics and machine learning, lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. The performance of this LASSO structure detection method was evaluated by using it to estimate the structure of a nonlinear polynomial model.

Local Learning Based Feature Selection

Feature selection for high-dimensional data is considered one of the current challenges in statistical machine learning. This feature-selection algorithm addresses several major issues with prior work, including problems with algorithm implementation, computational complexity, and solution accuracy. The key idea is to decompose an arbitrarily complex nonlinear problem into a set of locally linear ones through local learning, and then learn feature relevance globally within the large margin framework. The proposed algorithm is based on well-established machine learning and numerical analysis techniques, without making any assumptions about the underlying data distribution. It is capable of processing many thousands of features within minutes on a personal computer while maintaining a very high accuracy that is nearly insensitive to a growing number of irrelevant features.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. It is a supervised learning algorithm which is used for both classification as well as regression. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees means more robust forest. Similarly, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

In Reference [3]

Classification of speech intelligibility in Parkinson's disease

A problem in the clinical assessment of running speech in Parkinson's disease (PD) analysed by author Taha Khan was to track underlying deficits in a number of speech components including respiration, phonation, articulation and prosody, each of which disturbs the speech intelligibility. A set of 13 features, including the cepstral separation difference and Mel-frequency cepstral coefficients were computed to represent deficits in each individual speech component. These features were then used in training a support vector machine (SVM) using N-fold cross validation. The dataset used for method development and evaluation consisted of 240 running speech samples recorded from 60 PD patients and 20 healthy controls. These speech samples were clinically rated using the Unified Parkinson's Disease Rating Scale Motor Examination of Speech (UPDRS-S). The classification accuracy of SVM was 85% in 3 levels of UPDRS-S scale and 92% in 2 levels with the average area under the ROC (receiver operating characteristic) curves of around 91%. The strong classification ability of selected features and the SVM model supports suitability of this scheme to monitor speech symptoms in PD**.**

In Reference [4]

Collection and Analysis of a Parkinson Speech Dataset With Multiple Types of Sound Recordings

One of the authors Betul Erdogdu Sakar is involved in the study, using speech data from subjects which is expected to help the development of a noninvasive diagnostic. The data collected in the context of this study (Fig. 1) belongs to 20 PWP (6 female, 14 male) and 20 healthy individuals (10 female, 10 male) who appealed at the Department of Neurology in Cerrahpas¸a Faculty of Medicine, Istanbul University. . Test group consists of patients who are suffering from PD for 0 to Waveform of a voice sample belonging to a PWP (bottom) and a healthy individual (top). Amplitude of the signal (y-axis) is plotted against time duration (x-axis). 6 years. Individual ages vary between 43 and 77 (mean: 64.86, standard deviation: 8.97) along with 45 and 83 (mean: 62.55, standard deviation: 10.79) for test and control groups, respectively. From all subjects, 26 voice samples including sustained vowels, numbers, words, and short sentences are recorded. Various classification methods were used in this study.

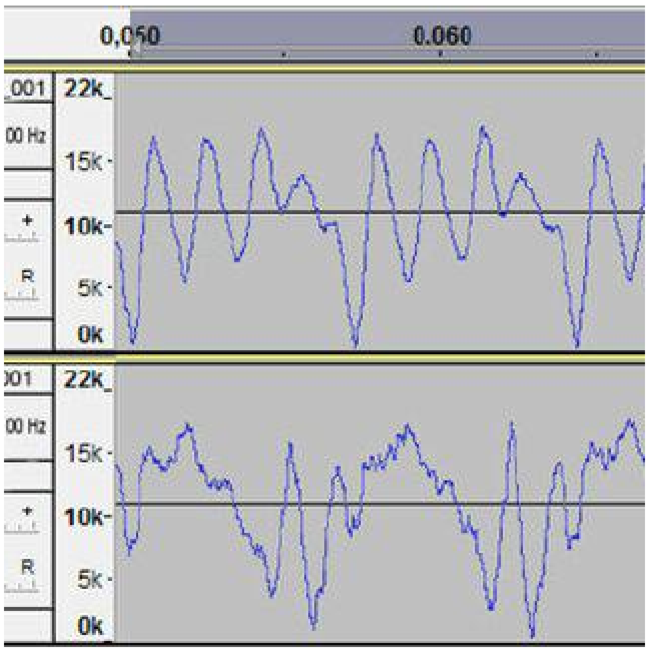


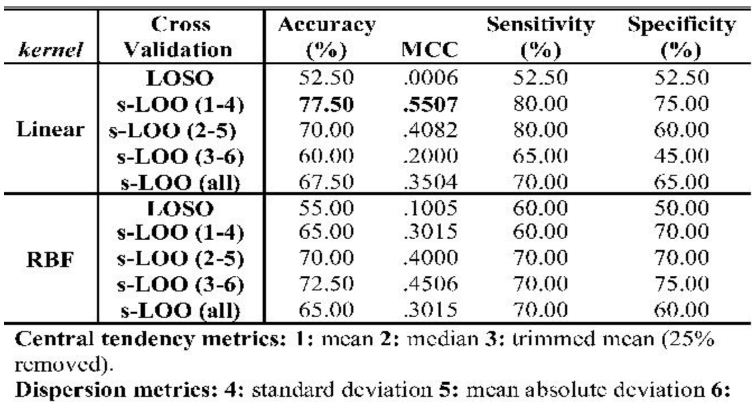
Fig 2.4.4 - Waveform of a voice sample belonging to a PWP (bottom) and a healthy individual (top)

Classification with Leave-One-Subject-Out:

Using conventional bootstrapping or leave-one-out validation methods, results in biased predictive models by sparing some samples of an individual in the training and some for the testing and creating an artificial overlap between the training and test sets. Instead, TIME-FREQUENCY-BASED FEATURES EXTRACTED FROM VOICE SAMPLES the proposed classification models in existing studies typically use leave-one-subject-out (LOSO) validation scheme in which all the voice samples of one individual is left out to be used for validation as if it is an unseen individual, and the rest of the samples is used for training. According to the LOSO validation scheme, if the majority of the voice samples of a test individual are classified as PWP, then the individual is classified as positive and otherwise negative.

Classification with Summarized Leave-One-Out:

The success of conventional LOSO validation scheme is compared with an unbiased LOO method which we called s-LOO (summarized-leave-one-out). According to the s-LOO method, the feature values of 26 voice samples of each subject are summarized using central tendency and dispersion metrics such as mean, median, trimmed mean (10% and 25% removed), standard deviation, interquartile range, mean absolute deviation, and a new form of dataset consisting of N samples is formed where N is the number of subjects. The groups of six metrics include all the metrics mentioned previously with slight differences in regulations, whereas the groups of two metrics are binary combinations of central tendency and dispersion metrics. By this method, the data are shortened in sample dimension whereas extended in feature dimension. The aim of summarizing the voice samples of subjects is to decrease the effect of variations between different voice samples of a subject.



kernel	Cross Validation	Accuracy (%)	MCC	Sensitivity (%)	Specificity (%)
Linear	**LOSO**	52.50	.0006	52.50	52.50
	s-LOO (1-4)	**77.50**	**.5507**	80.00	75.00
	s-LOO (2-5)	70.00	.4082	80.00	60.00
	s-LOO (3-6)	60.00	.2000	65.00	45.00
	s-LOO (all)	67.50	.3504	70.00	65.00
RBF	**LOSO**	55.00	.1005	60.00	50.00
	s-LOO (1-4)	65.00	.3015	60.00	70.00
	s-LOO (2-5)	70.00	.4000	70.00	70.00
	s-LOO (3-6)	72.50	.4506	70.00	75.00
	s-LOO (all)	65.00	.3015	70.00	60.00

Central tendency metrics: 1: mean **2:** median **3:** trimmed mean (25% removed).
Dispersion metrics: 4: standard deviation **5:** mean absolute deviation **6:**

Fig 2.4.5 - Performance metrics table using Linear LOSO and RBF LOSO

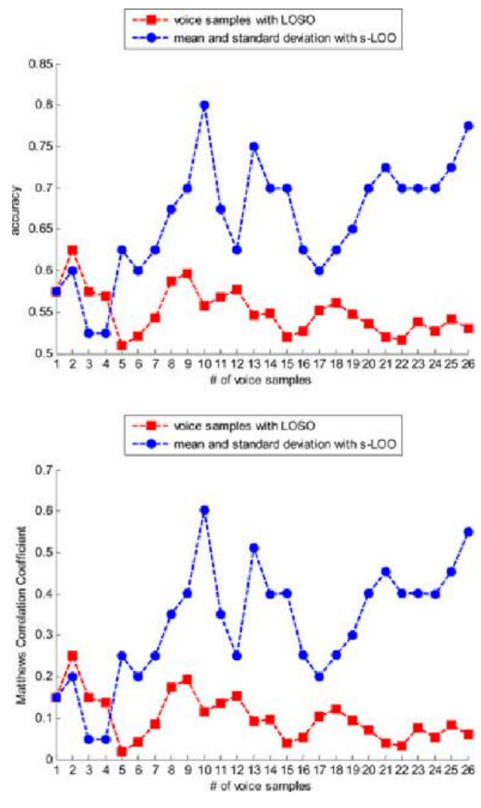


Fig 2.4.6 - (a) Accuracy v/s Voice samples with LOSO graph

(b) Matthews Correlation Coefficient v/s Voice samples with LOSO graph

CHAPTER 3
DESIGN OF THE PROPOSED SYSTEM

3.1 Block Diagram

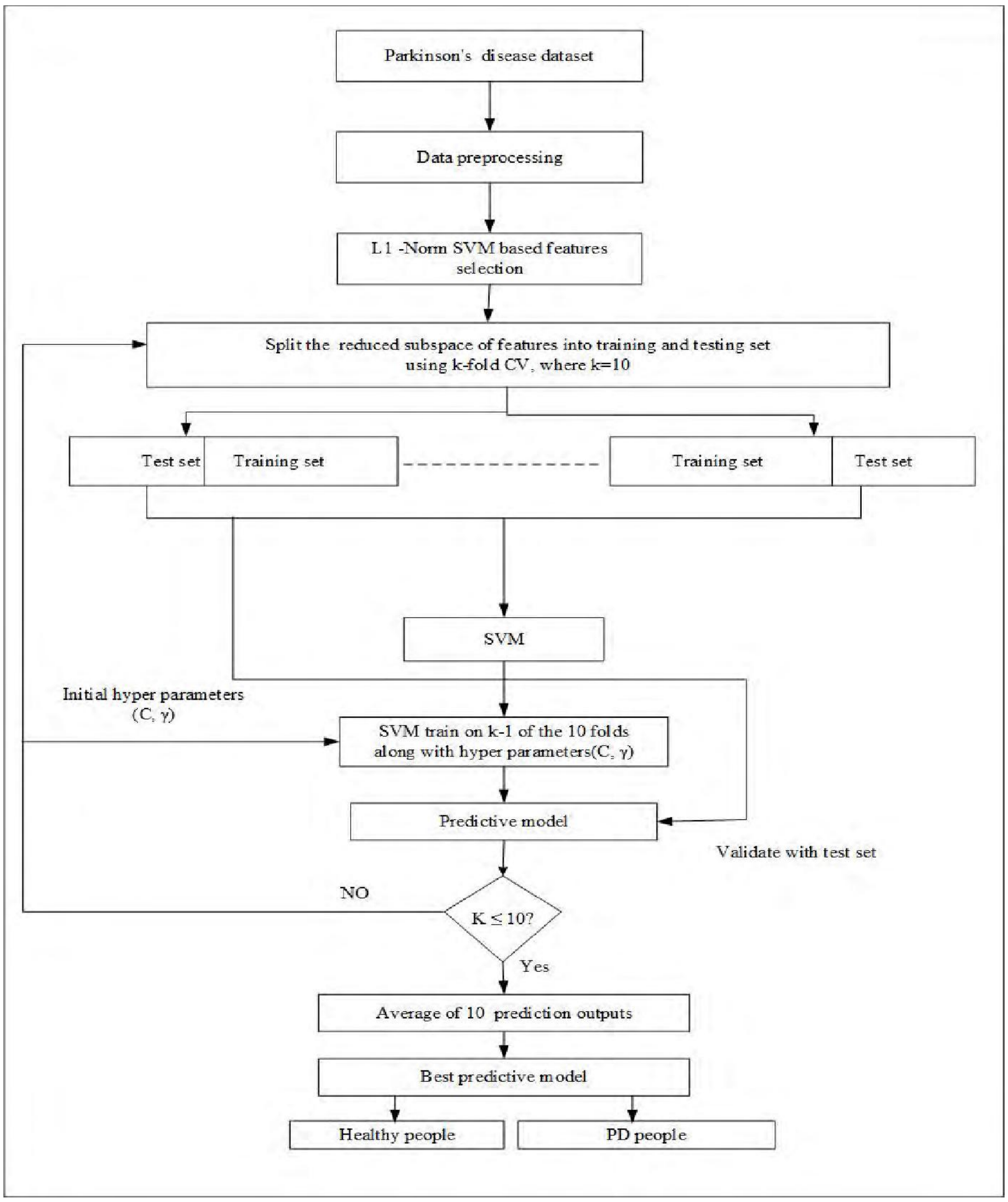

Parkinson's disease dataset
Data preprocessing
L1 -Norm SVM based features selection
Split the reduced subspace of features into training and testing set using k-fold CV, where k=10
Test set
Training set
Training set
Test set
SVM
Initial hyper parameters (C, γ)
SVM train on k-1 of the 10 folds along with hyper parameters(C, γ)
Predictive model
Validate with test set
NO
K ≤ 10?
Yes
Average of 10 prediction outputs
Best predictive model
Healthy people
PD people

Fig 3.1 - Block Diagram of proposed system

3.2 Module Description

The modules of the proposed system are as follows:

Data Acquisition

The Data set used for the project was adopted from the repository of the University of Oxford (UO) with collaboration with National centre for voice. The dataset contains the voice recordings of 31 people, including 23 people with Parkinson's disease contained 16 males and 7 females) and 8 healthy controls (males = and females = 5) were deployed in the study. In the dataset table, each column for voice and each row are related to one of 195 voice recording from an individual subject. Additionally, the people of age from 46 to 85 years with a mean value of age is 65.8 and standard division 9.8. Each of the recording is based on different measurements like vocal perturbation and nonlinear measurements and thus 23 features were extracted. Thus, the extracted dataset size is 195∗23 matrices. The table below shows the different features extracted from the voice recordings.

Label	Feature Name	Description	Min-Max	Mean , ± Std.
X1	MDVP:Fo(Hz)	The average vocal voice fundamental frequency	88.333000-260.105000	154.228641, ±41.390065
X2	MDVP:Fhi(Hz)	Maximum vocal fundamental frequency	102.145000-592.030000	197.104918, ±91.491548
X3	MDVP:Flo(Hz)	Minimum vocal fundamental frequency	65.476000-239.170000	116.324631, ±43.521413
X4	MDVP: Jitter (%)	Several measures of variation in fundamental frequency	0.001680-0.033160	0.006220, ±0.004848
X5	MDVP: Jitter (Abs)	-	0.000007-0.000260	0.000044, ±0.000035
X6	MDVP:RAP	-	0.000680-0.021440	0.003306, ±0.002968
X7	MDVP:PPQ	-	0.000920-0.019580	0.003446, ±0.002759
X8	Jitter : DDP	-	0.002040-0.064330	0.009920, ±0.008903
X9	MDVP:Shimmer	Several measures of variation in amplitude	0.009540-0.119080	0.029709, ±0.018857
X10	MDVP: Shimmer(dB)	-	0.085000-1.302000	0.282251, ±0.194877
X11	Shimmer:APQ3	-	0.004550-0.056470	0.015664, ±0.010153
X12	Shimmer:APQ5	-	0.005700-0.079400	0.017878, ±0.012024
X13	MDVP:APQ	-	0.007190-0.137780	0.024081, ±0.016947
X14	Shimmer: DDA	-	0.023370-0.104700	0.060043, ±0.029933
X15	NHR	Two measures of ratio of noise to tonal components in the voice	0.000650-0.314820	0.024847, ±0.040418
X16	HNR	-	8.441000-33.04700	21.885974, ±4.425764
X17	RPDE	Two nonlinear dynamical complexity measures	0.256570-0.685151	0.498536, ±0.103942
X18	D2	-	1.423287-3.671155	2.381826, ±0.382799
X19	DFA	Signal fractal scaling exponent	0.574282-0.825288	0.718099, ±0.055336
X20	spread1	Three nonlinear measures of fundamental frequency variation	-7.964984- -2.434031	5.684397, ±1.090208
X21	spread2	-	0.006274-0.450493	0.226510, ±0.083406
X22	PPE	-	0.044539-0.527367	0.206552, ±0.090119
y	Status	Health status of the subject Parkinson's=1 healthy=0	0.000000-1.000000	0.753846, ±0.431878

Fig 3.2- Data set Description

Pre-processing of Data

For a good representation of data, pre-processing is a very important step and machine-learning classifier should be trained and tested effectively. Techniques of pre-processing include removing of missing values, standard scalar, Min- Max Scalar have been applied to the dataset. The datatypes of the features which do not match with the other collection of data are also changed during the pre-processing step. We have used the StandardScaler and Min-Max Scalar. The StandardScalar ensures that every feature has the mean 0 and variance 1. Similarly, in Min-Max Scalar we arrange the data such that all features are between 0 and 1.

Exploratory Data Analysis

In statistics, exploratory data analysis is an approach to analysing data sets to summarize their main characteristics, often with visual methods. To get a better grasp of the data, we developed a scatter matrix that allows us to see the relationship between the features presented from the data set. There are multiple correlations seen in the matrix, but we specifically analysed the distribution of a feature when compared to the status of the patient. We noticed that across all features, those diagnosed with Parkinson's had a larger spread of data than those who did not. While healthy patients had their data clustered near 0, the Parkinson patients had the features run across the entire axis. Apart from the scatter matrix we have even used to pearson correlation heatmap for predicting the correlation. A heatmap is a graphical representation of data in which data values are represented as colours. We have even plotted box plots for comparing the influence of features on determining the status of the patient, i.e. healthy or Parkinson's disease. Box Plot is the visual representation of the depicting groups of numerical data through their quartiles. It is also used for detect the outlier in data set. It captures the summary of the data efficiently with a simple box and whiskers and allows us to compare easily across groups. Boxplot summarizes a sample data using 25th, 50th and 75th percentiles. These percentiles are also known as the lower quartile, median and upper quartile.

Feature Selection

Features selection algorithms are necessary to remove irrelevant features from feature space. The reduced features will improve the accuracy of classification and deduced execution time of the classifier. It is basically a technique where we choose those features in our data that contribute most to the target variable. Here the target variable is the accuracy score. We select features which result in the increase of the accuracy scores. We have used the L1 Norm SVM algorithm for the extraction of the best features.

Training and Testing

Train/Test is a method to measure the accuracy of your model. It is called Train/Test because we split the data set into two sets: a training set and a testing set. Training and test data are common for supervised learning algorithms. Given a dataset, its split into training set and test set.

Machine Learning Classification

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories. We make use of the SVM Classifier, Decision Tree Classifier and the Extra Tree Classifier for the classification process. The reason why we are using various classification methods is to provide a detailed comparison between the accuracy results and select the best method. Classifier in SVM depends only on a subset of points. Since we need to maximize distance between closest points of two classes (aka margin) we need to care about only a subset of points unlike logistic regression. Whereas the general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data (training data). We use Extra Tree Classifier because this class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

The classification methods are performed using all the features as well as using the extracted features by feature selection in-order to find out the difference between the accuracy values and its performance metrics.

Validation Testing

To check the proposed system performance, K-folds Cross validation (CV) method and three evaluation metrics are being used. We used K-fold cross-validation and according to k-fold the data set was split into k identical components. The k-1 groups were applied for training and leftover was used for testing purposes in each step. The k times the process is iterated. Then the average of k results is computed to get the performance of the classifier. The cross validation different Value of k was selected and used the value of k is 10 in our experiment. In 10 fold CV process, 50% of the data is used for training and 50% data will be used for testing. The 10-time repeated the validation process. In the process of each fold, all samples are randomly distributed in the training and test groups over the entire dataset prior to selection of new training and test sets for the new cycle. Finally, at the end of 10 folds process, an average of all performance metrics are computed.

3.3 Theoretical Foundations

L1-Norm Support Vector Machine(SVM)

In this project, we propose a stable feature selection method based on the L1-Norm Support Vector Machine(SVM). The basic concept of L1-Norm SVM is similar to that of Lasso Regression, but it is tailored for classification tasks, which is the model for any biomarker discoveries. L1-Norm SVM efficiently reduces the number of irrelevant or redundant features to fewer than number of samples, thus, it is appropriate for biomedical data.

Support Vector Machine Classifier

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N -the number of features) that distinctly classifies the data points. Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outlier detection. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support

Vector Machine. In our project we have used the Radial Based function and Linear kernel of the SVM classifier to find out which one results in better accuracy reports.

The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as follows and where $||x-x'||^2$ is the squared Euclidean distance between two data points xx and x′x′.

$$K(\mathbf{x}, \mathbf{x'}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{x'}||^2}{2\sigma^2}\right)$$

Decision Tree Classifier

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Extra Tree Classifier

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a "forest" to output a classification result. Conceptually, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest. Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.

K-fold Cross Validation

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once. In repeated cross-validation, the cross-validation procedure is repeated n times, yielding n random partitions of the original sample. The n results are again averaged (or otherwise combined) to produce a single estimation.

CHAPTER 4
IMPLEMENTATION OF PROPOSED SYSTEM

4.1 UML Diagrams

A UML diagram is a diagram based on the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system. UML is an acronym that stands for Unified Modelling Language. Simply, UML is a modern approach to modelling and documenting software. In fact, it’s one of the most popular business process modelling techniques. It is based on diagrammatic representations of software components. As the old proverb says: “a picture is worth a thousand words”. By using visual representations, we are able to better understand possible flaws or errors in software or business processes. We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. UML defines various kinds of diagrams to cover most of the aspects of a system. There are two broad categories of diagrams and they are again divided into subcategories-

- Structural Diagrams
- Behavioral Diagrams

1.Structural Diagrams

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable. The four structural diagrams are as follows.

i. Class diagram
ii. Object diagram
iii. Component diagram
iv. Deployment diagram

2. Behavioral Diagrams

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system. UML has the following five types of behavioral diagrams.

i. Use case diagram
ii. Sequence diagram
iii. Collaboration diagram
iv. State chart diagram
v. Activity diagram

4.1.1 Use Case Diagram

Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analysed, the functionalities are captured in use cases. From the below figure we can see that we have multiple use cases. The use cases are: import module, import dataset, dataset pre-processing, feature extraction, splitting data to train and test sets, model building, train model and prediction of output. We have a single primary actor, the "User", from whom the input is taken and output is predicted.

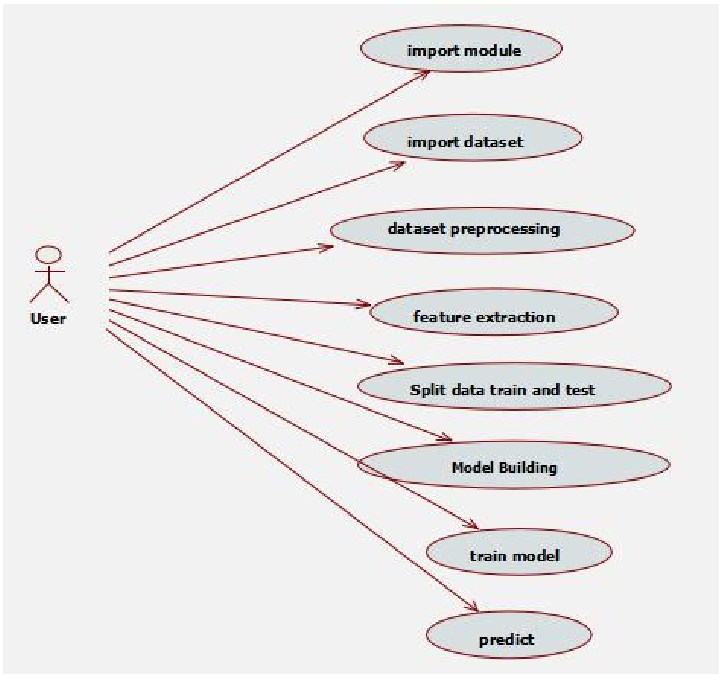


Fig 4.1 - Use Case diagram of the proposed classification system

4.1.2 Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application. Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages. Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram. Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose. The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. The purpose can be summarized by the below points

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

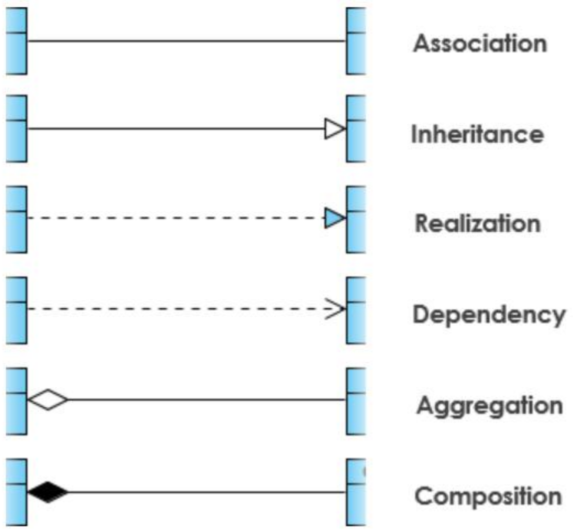
The essential elements for drawing a class diagram are:

- Class Name: The name of the class is only needed in the graphical representation of the class. It appears in the topmost compartment. A class is the blueprint of an object which can share the same relationships, attributes, operations, & semantics. The class is rendered as a rectangle, including its name, attributes, and operations in separate compartments.
- Attributes: An attribute is named property of a class which describes the object being modeled. In the class diagram, this component is placed just below the name-compartment.
- Operations

Class Visibility: The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations

Relations: UML is not just about pretty pictures. If used correctly, UML precisely conveys how code should be implemented from diagrams. If precisely interpreted, the implemented code will correctly reflect the intent of the designer.



The class diagram below represents the various classes in our project, i.e., modules, dataset, pre-processing, feature engineering, data partitioning, model building, prediction and display of result.

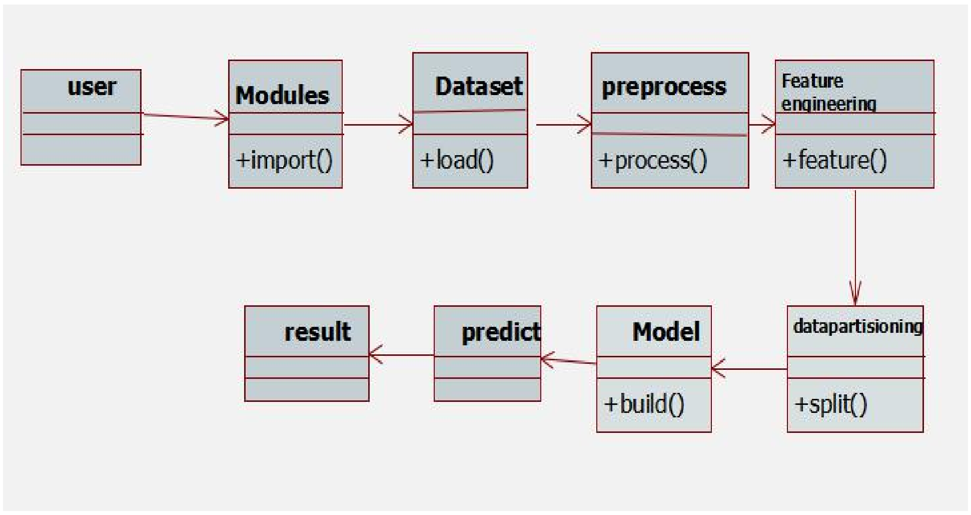


Fig 4.2 - Class diagram of the proposed classification system

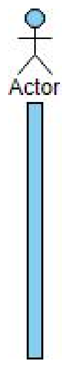
4.1.3 Sequence Diagram

UML Sequence diagrams are interaction diagrams that detail how operations are carried out. As sequence diagrams can be used to capture the interaction between objects in the context of a collaboration, one of the prime uses of sequence diagrams is in the transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams. Sequence diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent when.

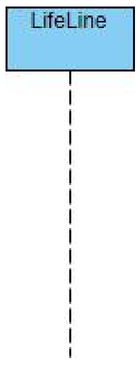
The sequence diagram notations are:

1. **Actors**: An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using

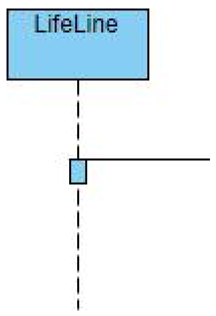
the UML diagram. An Actor plays the role of an entity that interacts with the subject (e.g., by exchanging signals and data). An actor can also be an external to the subject.



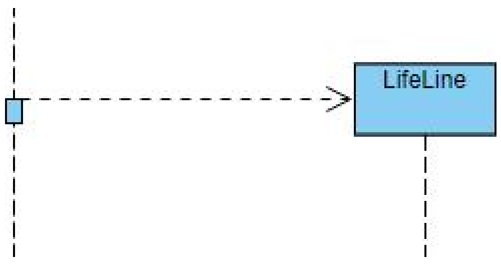
2. **Lifelines:** A lifeline is a named element which depicts an individual participant in a sequence diagram. So, each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.



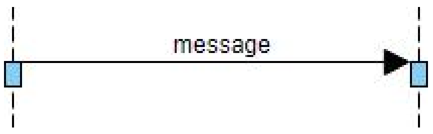
3. **Activation**: An activation is represented by a thin rectangle on a lifeline) represents the period during which an element is performing an operation. The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively.



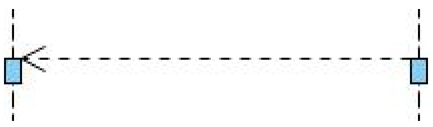
4. **Create Message**: We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.



5. **Call Message:** A call message defines a particular communication between lifelines of an interaction, which represents an invocation of operation of target lifeline.

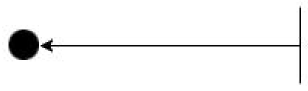


6. **Return Message:** A return message defines a particular communication between lifelines of an interaction, which represents the pass of information back to the caller of a corresponded former message.

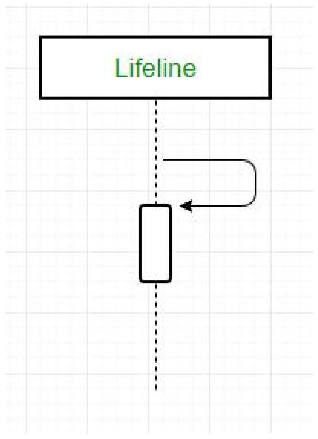


7. **Lost Message** – A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow

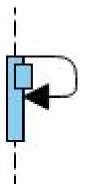
directed towards an end point from a lifeline. For example: Consider a scenario where a warning is generated.



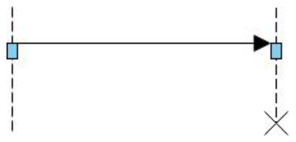
8. **Self-Message:** A self-message defines a particular communication between lifelines of an interaction, which represents the invocation of message of the same lifeline.



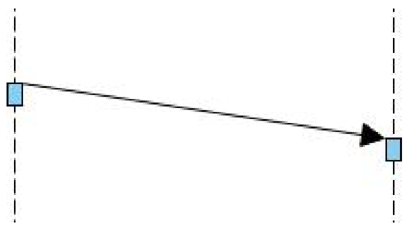
9. **Recursive Message:** A recursive message defines a particular communication between lifelines of an interaction, which represents the invocation of message of the same lifeline. It's target points to an activation on top of the activation where the message was invoked from.



10. **Destroy Message:** A destroy message defines a particular communication between lifelines of an interaction, which represents the request of destroying the lifecycle of target lifeline.



11. **Duration Message:** A duration message defines a particular communication between lifelines of an interaction, which shows the distance between two time instants for a message invocation.



The sequence diagram describes the flow of the process. It shows the exact sequence in which the project is carried out. According to the above diagram, we first import the modules and dataset, pre-process and perform feature selection on the dataset, build the model and then the result is predicted.

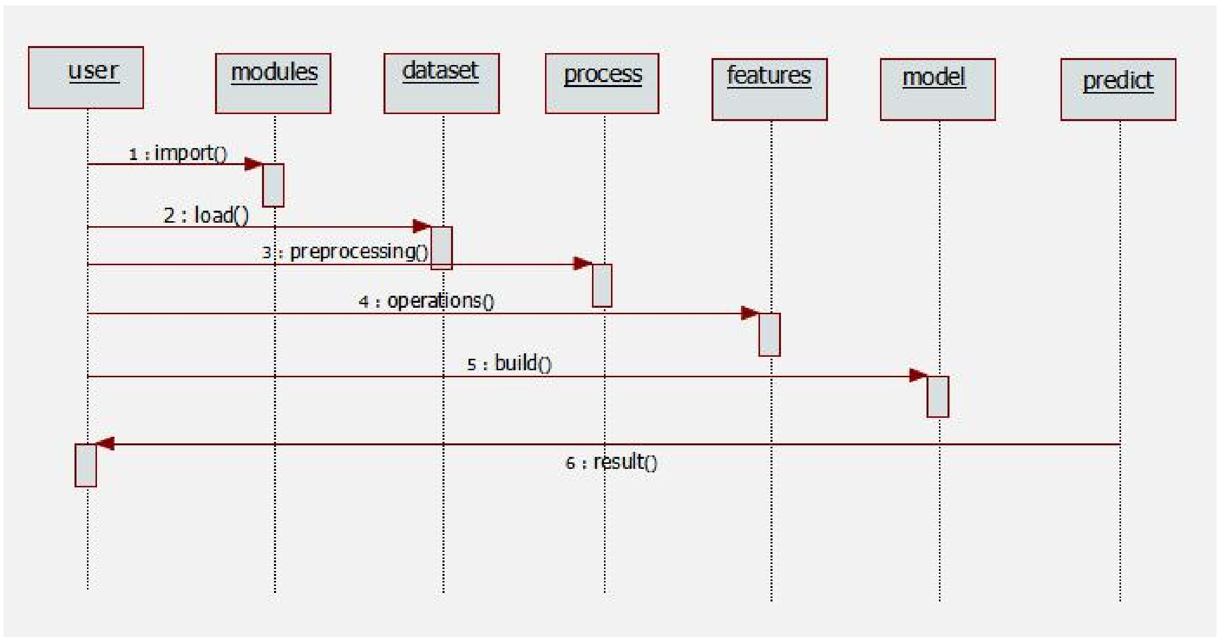
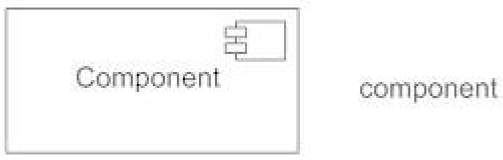


Fig 4.3 - Sequence diagram of the proposed classification system

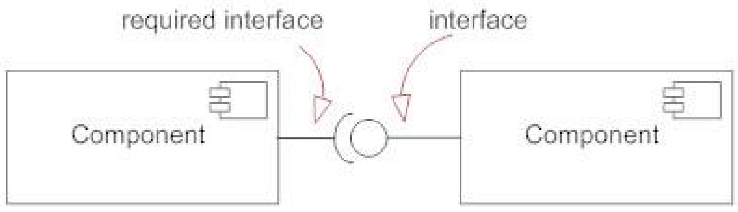
4.1.4 Component Diagram

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required function is covered by planned development. The basic symbols and notations involved in the component diagram are as follows:

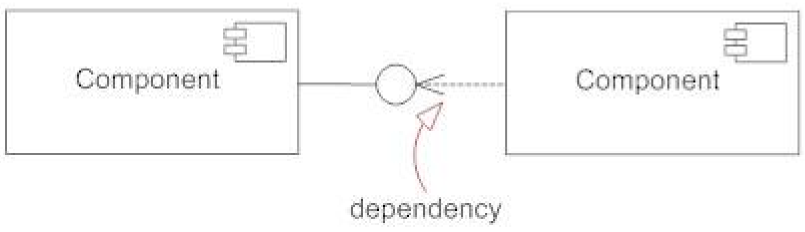
1. **Component**: A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word written above the name of the component to help distinguish it from a class.



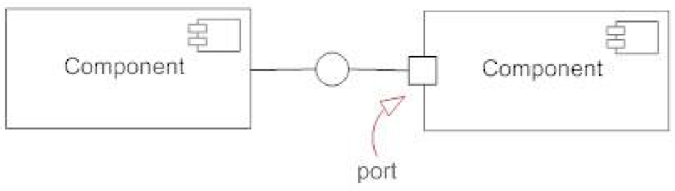
2. **Interface:** An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



3. **Dependencies:** Draw dependencies among components using dashed arrows.



4. **Port**: Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.



The component diagram below displays the major components of our project. It represents the server from which the data is collected, the collected dataset, modules required and the algorithms to be applied on the data to predict the output.

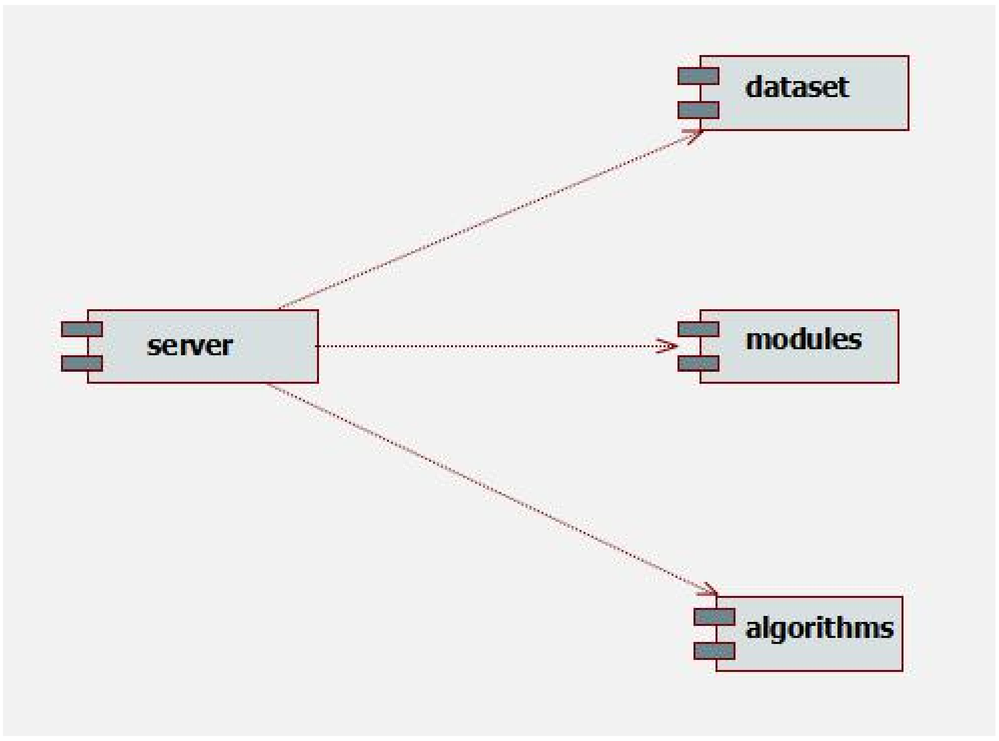


Fig 4.4 - Component Diagram of the proposed classification system

4.2 Design and Test steps

As discussed earlier in Chapter 3, the basic design of the proposed system involves seven major stages, i.e. Data Acquisition, data pre-processing, exploratory data analysis, feature selection, splitting the dataset to train and test datasets, implementing classifier(for all features and selected features) and validation testing.

In the data acquisition stage we import required libraries and convert the dataset into dataframe. Later we pre-process the data. This is done by using MinMax scaler, converting string data to numerical data. After this we perform exploratory data analysis in order to learn more about the nature of the dataset features. To gain insights on the nature of the dataset we make use of Scatter matrix, Pearson correlation heatmap and box plots. After the data is ready, we find out most important features by making use of L1- Norm SVM feature selection. Each feature is ranked by the L1- Norm SVM feature selection algorithm.

We now split the dataset with all features and selected features into two different train and test sets. First we classify train set with all features using three classifiers, Decision tree classifier, Extra tree classifier and SVM classifier. We use the same classifiers to classify train set with selected features. Later we test the validity of the model with best accuracy classifier with K- Fold cross validation for both full feature test set and selected features test set.

4.3 Algorithms and Pseudo Code

4.3.1 Modules Used

- **Numpy:** NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster that traditional Python lists. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

- **Pandas:** Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is built on the Numpy package and its key data structure is called the Data Frame. Data Frames allow you to store and manipulate tabular data in rows of observations and columns of variables.

- **Scipy:** SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

- **Seaborn:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

- **matplotlib.pyplot:** matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes.

- **sklearn:** Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours. scikit-learn is an open source project, meaning that it is free to use and distribute, and anyone can easily obtain the source code to see what is going on behind the scenes. It contains a number of state-of-the-art machine learning algorithms, as well as comprehensive documentation about each algorithm. scikit-learn is a very popular tool, and the most prominent Python library for machine learning.

    - sklearn.svm
    - sklearn.datasets
    - sklearn.feature_selection
    - sklearn.preprocessing
    - sklearn.model_selection
    - sklearn.metrics

4.3.2 Proposed System Algorithm

Begin

Step 1: Data pre-processing using StandardScaler, and MinMax Scaler on PD dataset;

i.e. $V^{-} = \frac{v-min}{max-min}(new_{max} - new_{min}) + new_{min}$ in Eq(1)

Step 2: Selected features by L1 –Norm SVM;

Step 3: For j = 1: k, performance estimation applied k-fold cross- validation, where k = 10

Training set = k-1 sub-group of 195 instances;

Testing set k-9 sub-set of 195 instances;

Step 4: Train classifiers with k-1 sub-groups with initial hyper- parameters values (C, γ);

Step 5: Validate classifier on a test set of 10- folds and achieved the best combination of hyper-parameters;

Repeat step 3 and 4;

Step 6: Compute average classification results of 10 fold processing

i.e. $E = \frac{1}{10} \sum_{i=1}^{10} Ei$; Eq(13)

Step 7: Performance of the best predictive model on j testing set;

Step 8: Finish;

4.3.3 Feature Selection with L1 - Norm SVM algorithm

Begin

Step 1: Create n instances,

$$x_i = \{x_{i1}, x_{I2} \ldots x_{in}\}$$

of the dataset; equation (3)

Step 2: Applied CV test on each instance for adjusting regularized hyperparameter C and γ ;

Step 3: Using L1-Norm SVM on each instance and calculated weight vector α for each feature;

Step 4: Remove the features who coefficient $\alpha = 0$ in each instance;

Step 5: Compute the average CV score for each instance;

Step 6: Repeat step 2 to 5 up to that no features of $\alpha = 0$ for all each instance of dataset;

Step 7: Choose the subset of features for each instance of the dataset who cross-validation score values are high;

Step 8: Combined all remaining features into a new reduced set of features;

Step 9: Produced x to reduced features set x^- that includes features in k;

Step 10: Finish.

4.3.4 Decision Tree Classifier Algorithm

Begin

Step 1: Place the best attribute of the dataset at the root of the tree.

Step 2: Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.

Step 3: Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

- The basic algorithm used in decision trees is known as the ID3 algorithm. The ID3 algorithm builds decision trees using a top-down, greedy approach. Briefly, the steps to the algorithm are: - Select the best attribute $\rightarrow$ A - Assign A as the decision attribute (test case) for the NODE. - For each value of A, create a new descendant of the NODE. - Sort the training examples to the appropriate descendant node leaf. - If examples are perfectly classified, then STOP else iterate over the new leaf nodes. For ID3, we think of the best attribute in terms of which attribute has the most information gain, a measure that expresses how well an attribute splits that data into groups based on classification.
- To define information gain precisely, we need to define a measure commonly used in information theory called entropy that measures the level of impurity in a group of examples. Mathematically, it is defined as follows :

$$Entropy : \sum_{i=1} -p * log_2(p_i)$$
$$p_i = Probabilityofclassi$$

$$Entropy(S) = -p_+ log_2 p_+ - p_- log_2 p_-$$

The information Gain is calculated as follows.

$$InformationGain = Entropy(parentnode) - [AverageEntropy(children)]$$

4.3.5 Extra Tree Classifier Algorithm

```
Begin

Step 1: Importing the required libraries.

Step 2: Loading and Cleaning the Data.

Step 3: Building the Extra Trees Forest and computing the individual feature
importances.

Step 4: Visualizing and Comparing the results.
```

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

4.4 Dataset Description

Dataset used in the research was adopted from the repository of the University of Oxford (UO) with collaboration with national center for voice developed by little and is available at the UC Irvine repository of data mining. The voice recordings of 31 people, including 23 people with Parkinson’s disease (contained 16 males and 7 females) and 8 healthy controls (males = 3 and females = 5) were deployed in the study. In the dataset table, each column for voice and each row are related to one of 195 voice recording from an individual subject. Additionally, the people of age from 46 to 85 years with a mean value of age is 65.8 and standard division 9.8. The main objective of this dataset was to classify people with Parkinson’s disease from healthy people by finding differences in vowel vocalization. The ‘‘status’’ attribute is set to 0 for healthy and 1 for PD people. For each subject, an average of 6 phonation of a vowel was recorded for 36 second and total of 195 samples were recorded. The phonations were recorded in industrial acoustic company sound-treated booth by the microphone which at distance 8 cm from mouth. The voice speech signals were stored in the computer using a computerized speech laboratory.The size of the extracted dataset is 195*23 matrixes. The following table shows the 23 features of voice signals of PD dataset.

The following are the matrix column entries (attributes):

NAME	**-**	**ASCII subject name and recording number**
MDVP:Fo(Hz)	-	Average vocal fundamental frequency
MDVP:Fhi(Hz)	-	Maximum vocal fundamental frequency
MDVP:Flo(Hz)	-	Minimum vocal fundamental frequency
MDVP:Jitter(%),MDVP:Jitter(Abs),MVP:RAP,MDVP:PPQ,Jitter:DDP	-	Several measures of variation in fundamental frequency
MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA	-	Several measures of variation in amplitude
NHR,HNR	-	Two measures of ratio of noise to tonal components in the voice status - Health status of the subject (one) - Parkinson's, (zero) - healthy
RPDE,D2	-	Two nonlinear dynamical complexity measures
DFA	-	Signal fractal scaling exponent
Spread,spread2,PPE	-	Three nonlinear measures of fundamental frequency variation

4.5 Testing Process

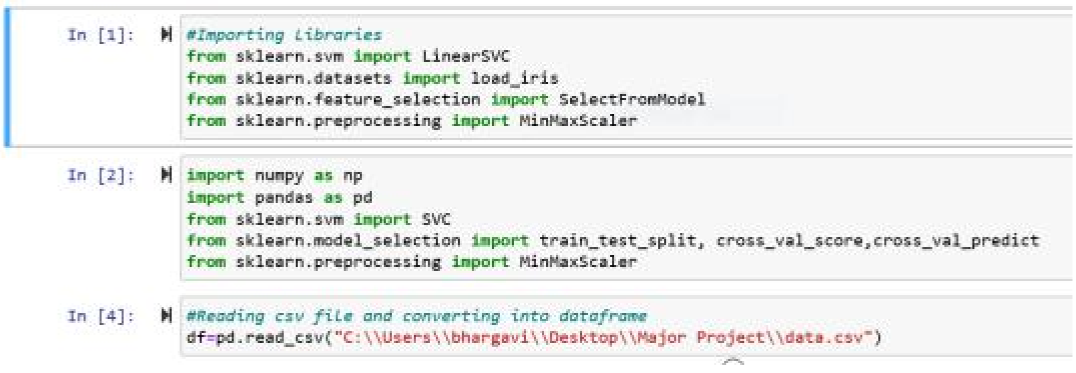
The testing process is carried out at the end. We make use of the test set to determine the accuracy of the model. In order to prove that selective features set is more accurate than full feature set, we test both the full feature test set and selected features test set. We test the model using K- Fold cross validation. We also calculate other performance metrics, i.e. Accuracy, F1- score, sensitivity, specificity and precision. We calculate the performance metrics using the confusion matrix.

```
from sklearn.metrics import classification_report,confusion_matrix

CM1 = confusion_matrix(y_test,y_pred)    # CM1 = Confusion matrix

TN1 = CM1[0][0]

FN1 = CM1[1][0]

TP1 = CM1[1][1]

FP1 = CM1[0][1]

Sensitivity = TP1/(TP1+FN1)

Specificity = TN1/(TN1+FP1)

PPV1 = TP1/(TP1+FP1)

NPV1 = TN1/(TN1+FN1)

FPR1 = FP1/(FP1+TN1)

FNR1 = FN1/(TP1+FN1)

FDR1 = FP1/(TP1+FP1)

Accuracy = (TP1+TN1)/(TP1+FP1+FN1+TN1)
```

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Importing required Python libraries

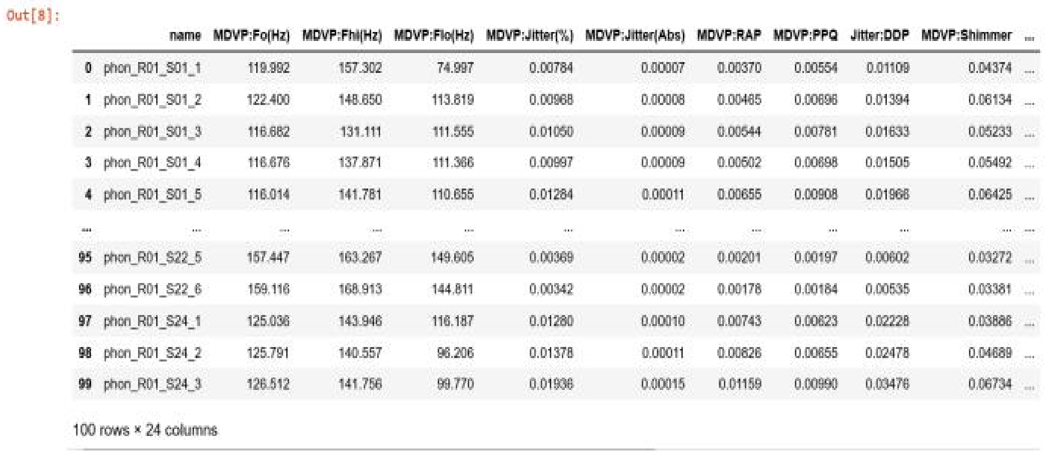


```
In [1]:   #Importing Libraries
          from sklearn.svm import LinearSVC
          from sklearn.datasets import load_iris
          from sklearn.feature_selection import SelectFromModel
          from sklearn.preprocessing import MinMaxScaler

In [2]:   import numpy as np
          import pandas as pd
          from sklearn.svm import SVC
          from sklearn.model_selection import train_test_split, cross_val_score,cross_val_predict
          from sklearn.preprocessing import MinMaxScaler

In [4]:   #Reading csv file and converting into dataframe
          df=pd.read_csv("C:\\Users\\bhargavi\\Desktop\\Major Project\\data.csv")
```

Fig 5.1 - Importing important libraries

As seen above we have first imported the required libraries for our project. We have imported sklearn.svm library to perform Linear SVM and Support Vector Machine Classification. Sklearn.feature_selection library has been imported to enable selection of features. The sklearn.preprocessing library will be used to perform scaling using MinMaxScaler and StandardScaler. Sklearn.model_selection library has been imported to perform cross validation.

5.2 Displaying the dataset after conversion to dataframe



Out[8]:

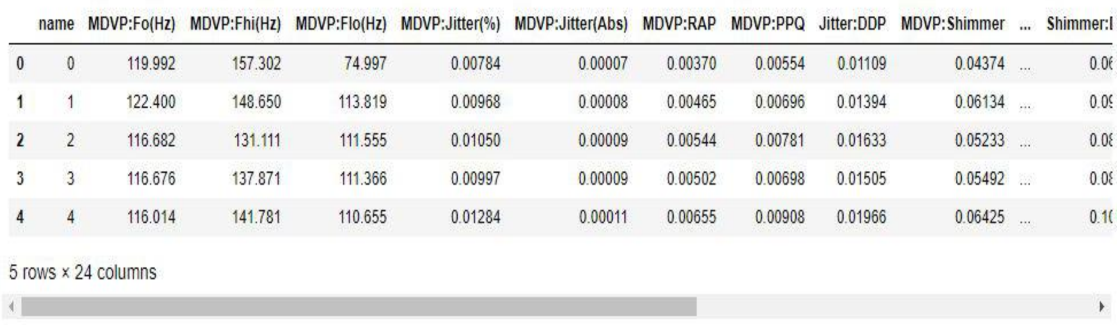
	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...
...	...	...	...	...	...	...	...	...	...	...	...
95	phon_R01_S22_5	157.447	163.267	149.605	0.00369	0.00002	0.00201	0.00197	0.00602	0.03272	...
96	phon_R01_S22_6	159.116	168.913	144.811	0.00342	0.00002	0.00178	0.00184	0.00535	0.03381	...
97	phon_R01_S24_1	125.036	143.946	116.187	0.01280	0.00010	0.00743	0.00623	0.02228	0.03886	...
98	phon_R01_S24_2	125.791	140.557	96.206	0.01378	0.00011	0.00826	0.00655	0.02478	0.04689	...
99	phon_R01_S24_3	126.512	141.756	99.770	0.01936	0.00015	0.01159	0.00990	0.03476	0.06734	...

100 rows × 24 columns

Fig 5.2 - Display of dataset

The above screenshot shows the display of the dataset after converting into a dataframe.

5.3 Renaming the name column of dataframe to numerical values



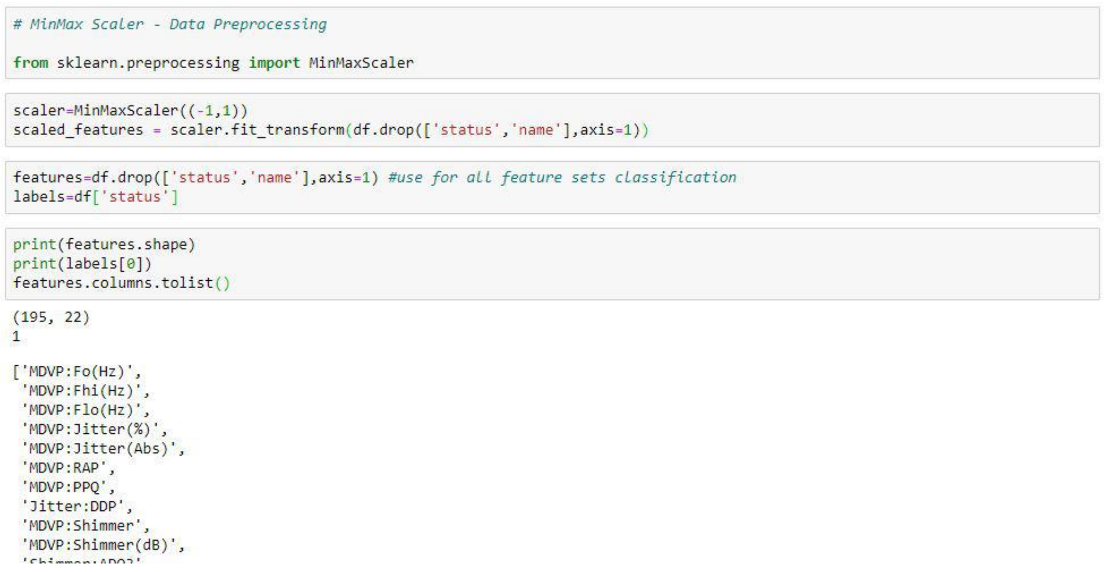
	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)	MDVP:RAP	MDVP:PPQ	Jitter:DDP	MDVP:Shimmer	...	Shimmer:I
0	0	119.992	157.302	74.997	0.00784	0.00007	0.00370	0.00554	0.01109	0.04374	...	0.06
1	1	122.400	148.650	113.819	0.00968	0.00008	0.00465	0.00696	0.01394	0.06134	...	0.09
2	2	116.682	131.111	111.555	0.01050	0.00009	0.00544	0.00781	0.01633	0.05233	...	0.08
3	3	116.676	137.871	111.366	0.00997	0.00009	0.00502	0.00698	0.01505	0.05492	...	0.08
4	4	116.014	141.781	110.655	0.01284	0.00011	0.00655	0.00908	0.01966	0.06425	...	0.1(

5 rows × 24 columns

Fig 5.3 - Renaming dataframe name column

The name column has been changed to numeric values in order to enable easier classification of data.

5.4 Performing pre-processing on dataset



```
# MinMax Scaler - Data Preprocessing

from sklearn.preprocessing import MinMaxScaler
```

```
scaler=MinMaxScaler((-1,1))
scaled_features = scaler.fit_transform(df.drop(['status','name'],axis=1))
```

```
features=df.drop(['status','name'],axis=1) #use for all feature sets classification
labels=df['status']
```

```
print(features.shape)
print(labels[0])
features.columns.tolist()
```

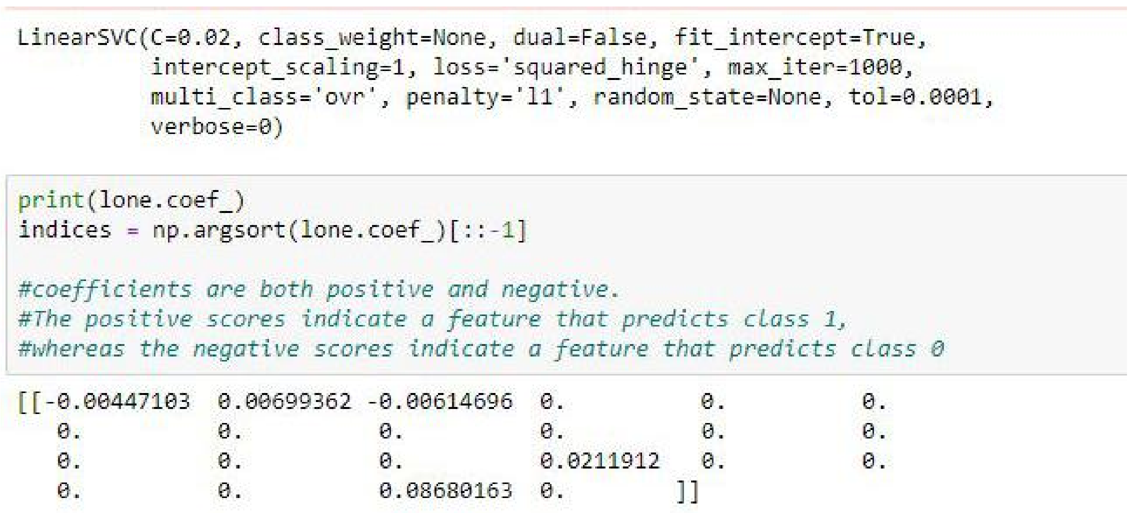
```
(195, 22)
1

['MDVP:Fo(Hz)',
 'MDVP:Fhi(Hz)',
 'MDVP:Flo(Hz)',
 'MDVP:Jitter(%)',
 'MDVP:Jitter(Abs)',
 'MDVP:RAP',
 'MDVP:PPQ',
 'Jitter:DDP',
 'MDVP:Shimmer',
 'MDVP:Shimmer(dB)',
 'Shimmer:APQ3'
```

Fig 5.4 – Pre-processing of dataset

In the above screenshot we can see that the size of the dataset as 195*22. We apply MinMax Scaler to perform scaling on the given data. We even display feature names of the columns.

5.5 Feature Selection

After pre-processing of the data set we then perform feature selection. Feature Selection is performed using the L1- Norm SVM method. The feature ranks of each of the features are displayed as output. The parameter C controls the sparsity, the smaller the C, the fewer features selected. Having penalty parameter value L1 gives sparse solutions, i.e. many of their estimated coefficients are zero.



```
LinearSVC(C=0.02, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l1', random_state=None, tol=0.0001,
          verbose=0)
```

```
print(lone.coef_)
indices = np.argsort(lone.coef_)[::-1]

#coefficients are both positive and negative.
#The positive scores indicate a feature that predicts class 1,
#whereas the negative scores indicate a feature that predicts class 0
```

```
[[-0.00447103  0.00699362 -0.00614696  0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.          0.          0.0211912   0.          0.
   0.          0.          0.08680163  0.        ]]
```

Fig 5.5.1 - Feature selection on dataset

Among the feature scores, a positive score of feature indicates that the feature is capable of accurately predicting class 1, i.e. person with Parkinson’s Disease. A negative score of feature indicates that the feature is capable of accurately predicting class 0, i.e. person who is healthy. Using these scores a graph has been plotted as follows.

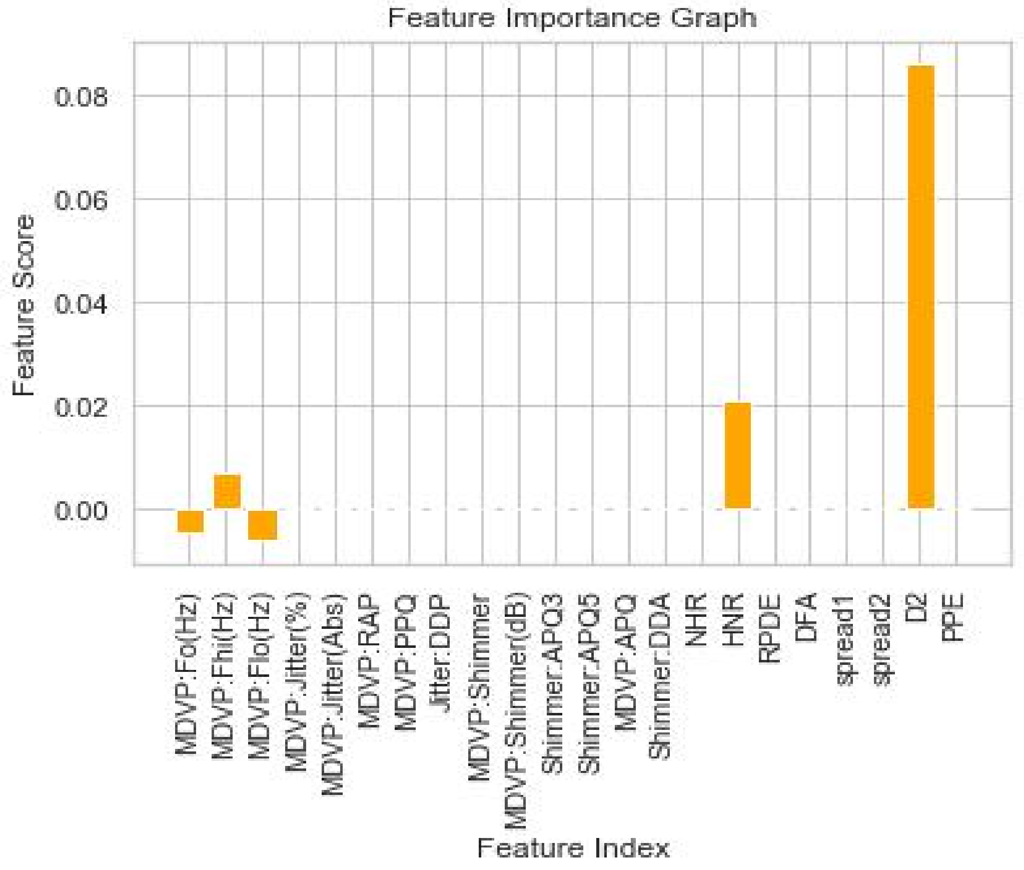


Fig 5.5.2- Feature importance Graph

From the above feature importance graph it is clear that MDVP:Fo(Hz), MDVP:FHi(Hz), MDVP:FLo(Hz), HNR and D2 have secured highest ranks among rest of the features.

5.6 Exploratory Data Analysis

To get a better grasp of the data, we perform exploratory data anaysis. We have made use of Sctter matrix, Pearson Correlation graph and Box plots. The Scatter matrix and Pearson Correlation graph help us recognize the relation between each feature. The Box-plots are used to determine the dependency of each feature in determining the status of the patient, i.e, healthy or affected by Parkinson's Disease.

5.6.1 Scatter Matrix

A scatterplot matrix is a matrix associated to n numerical arrays (data variables), X1,X2,…,Xn , of the same length. The cell (i,j) of such a matrix displays the scatter plot of the variable Xi versus Xj. There are multiple correlations seen in the matrix, but we specifically analysed the distribution of a feature when compared to the status of the patient. We noticed that across all features, those diagnosed with Parkinson's had a larger spread of data than those who did not. While healthy patients had their data clustered near 0, the Parkinson patients had the features run across the entire axis. This results in a separation between the status of 1 and 0 in the spread1, spread 2, D2, and PPE columns.

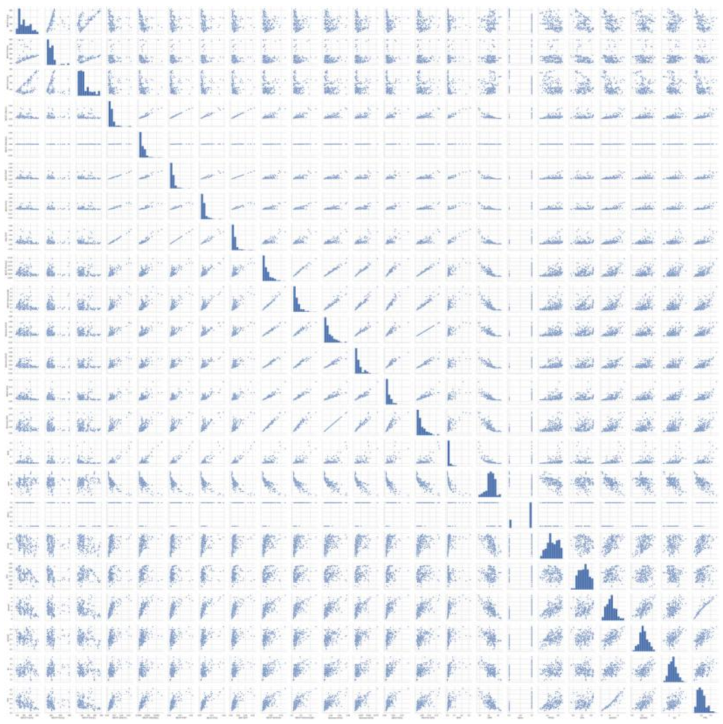


Fig. 5.6.1- Scatter Matrix

5.6.2 Pearson Correlation Heatmap

The Pearson correlation graph is used to represent the relationship between two variables that are measured on the same interval or ratio scale. It is a measure of the strength of the association between two continuous variables. Each square shows the correlation between the variables on each axis. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. The close to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is.

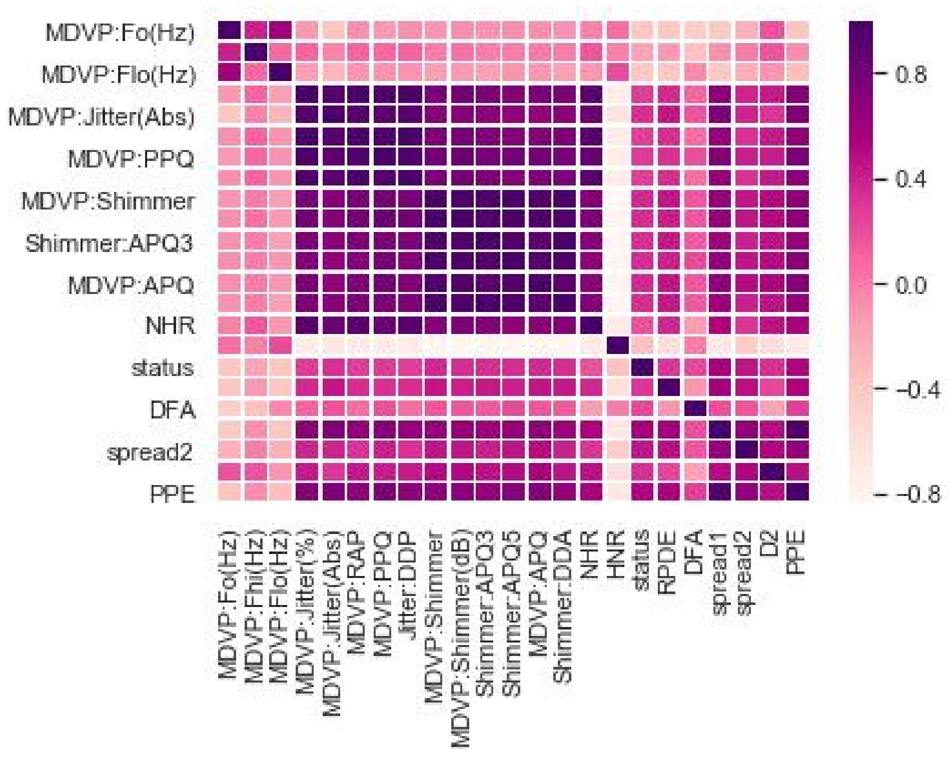


Fig 5.6.2- Pearson Correlation Heatmap

5.6.3 Box Plot

If box plots have low values of Lower quartile, median and Higher quartile for status ‘1’, then those features are responsible for Parkinson’s condition. The features with low Lower quartile, median and Higher quartile values are : MDVP:Fo(Hz), MDVP:FHi(Hz), MDVP:FLo(Hz) and HNR. The box plots are given below.

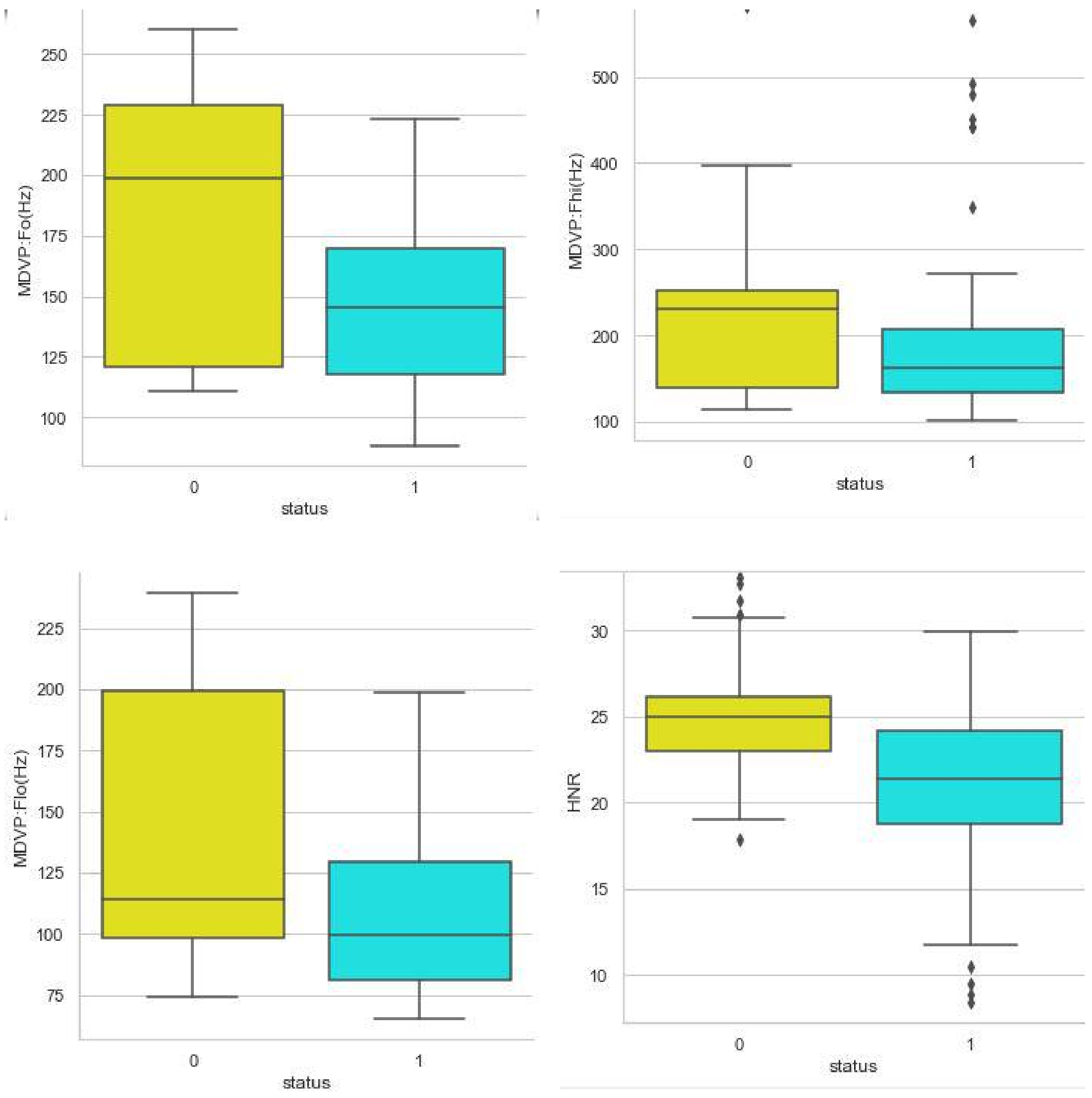
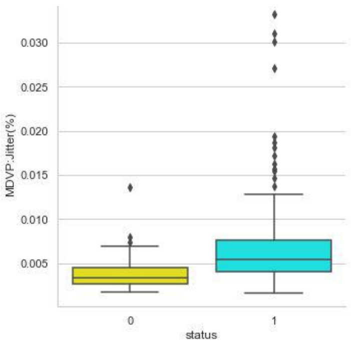
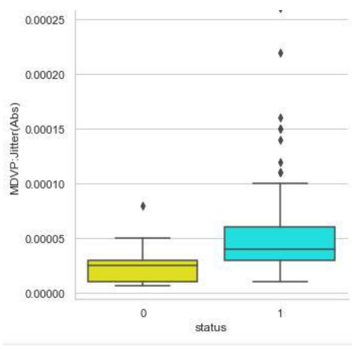
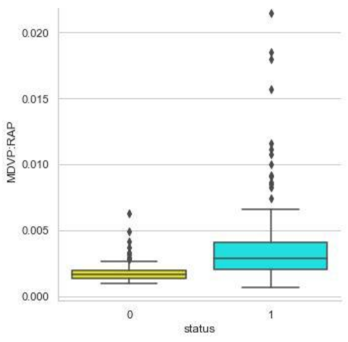
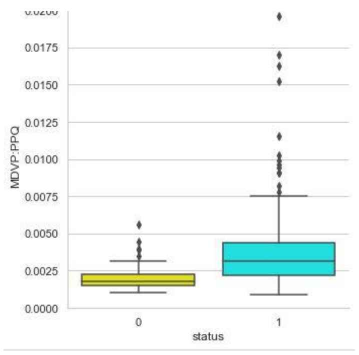


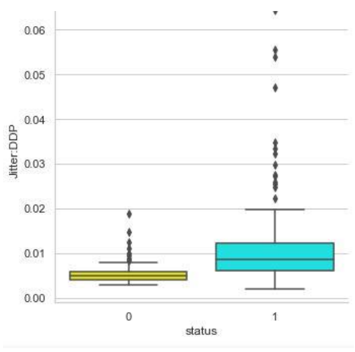
Fig 5.6.3 (a) - Features affecting detection of Parkinson’s disease

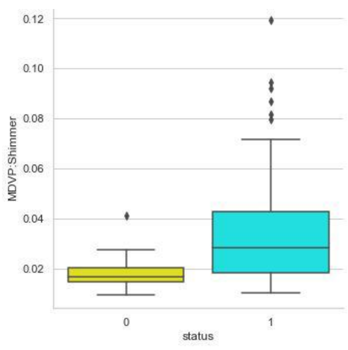

MDVP:Jitter(%)
0.030
0.025
0.020
0.015
0.010
0.005
0
1
status

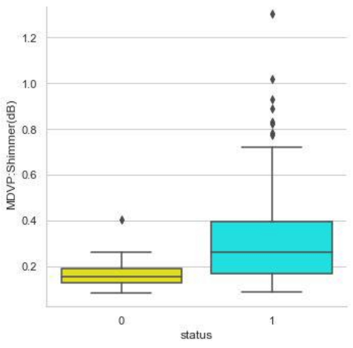

MDVP:Jitter(Abs)
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
1
status

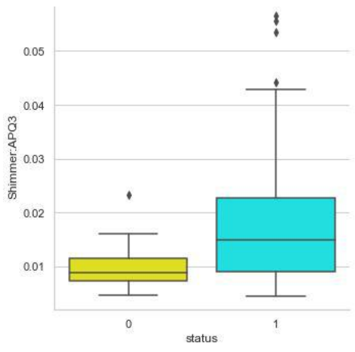

MDVP:RAP
0.020
0.015
0.010
0.005
0.000
0
1
status

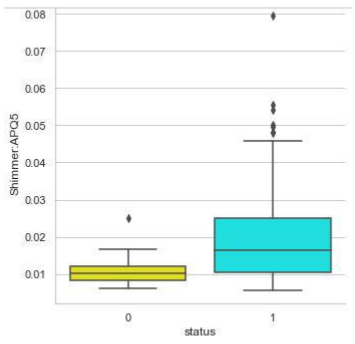

MDVP:PPQ
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
1
status


Jitter:DDP
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
1
status


MDVP:Shimmer
0.12
0.10
0.08
0.06
0.04
0.02
0
1
status


MDVP:Shimmer(dB)
1.2
1.0
0.8
0.6
0.4
0.2
0
1
status


Shimmer:APQ3
0.05
0.04
0.03
0.02
0.01
0
1
status


Shimmer:APQ5
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
1
status

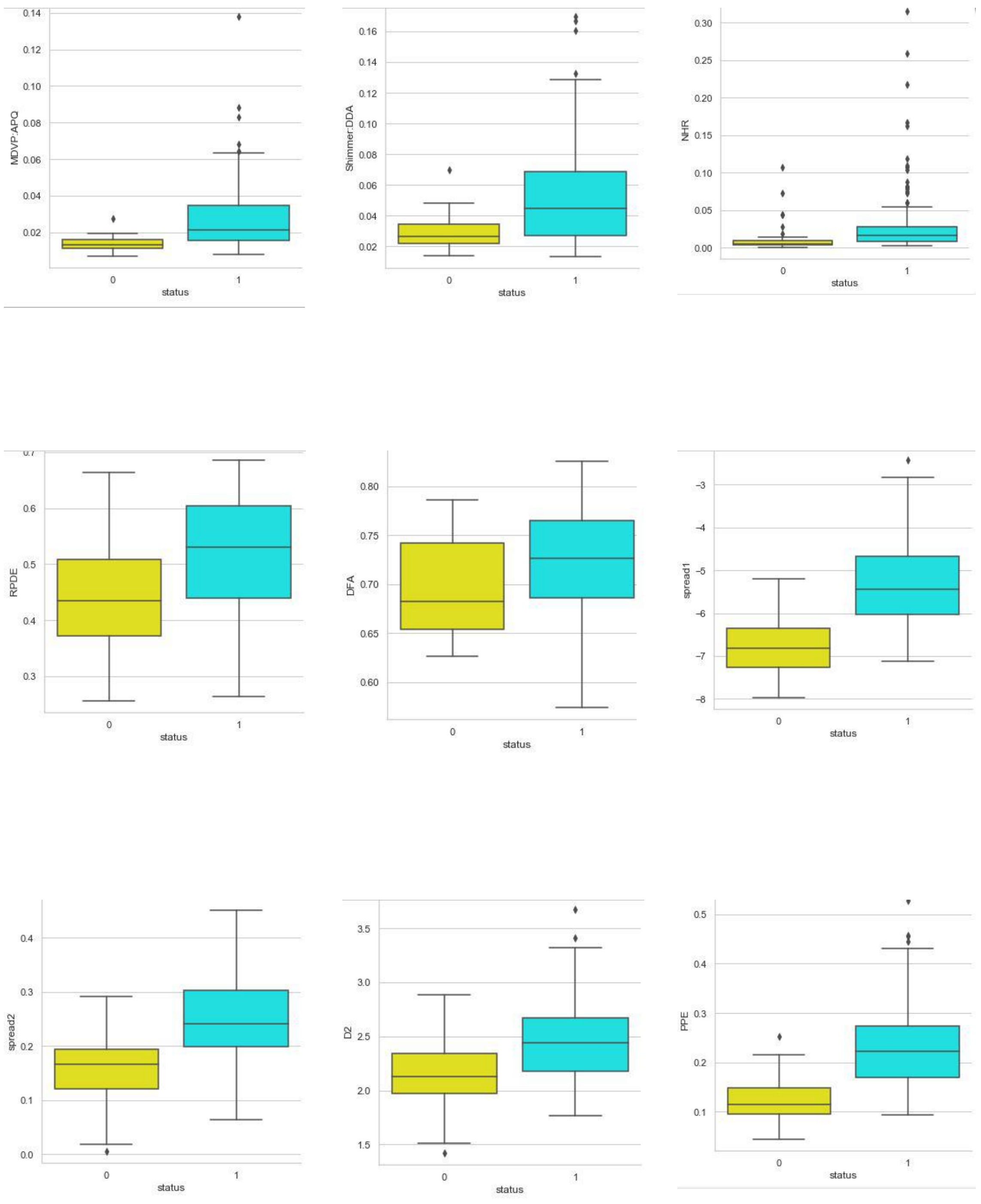
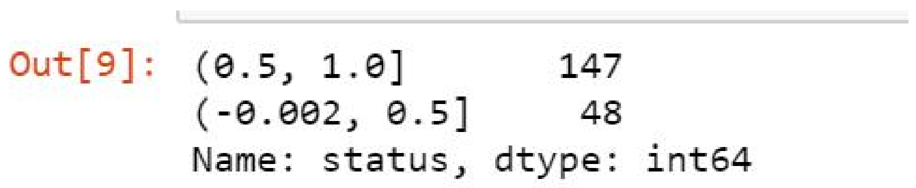


Fig 5.6.3 (b) - Features not affecting detection of Parkinson's disease

5.7 Data Distribution of Healthy and PD

The below output shows the distribution of the data sets which denote the number or count of healthy people and people with Parkinson’s disease. ‘0’ depicting the healthy people and ‘1’ depicting the people with PD. The number of healthy subjects are 48 and Parkinson’s disease affected subjects are 148. A graph has been plotted to represent the same.



```
Out[9]: (0.5, 1.0]       147
        (-0.002, 0.5]     48
        Name: status, dtype: int64
```

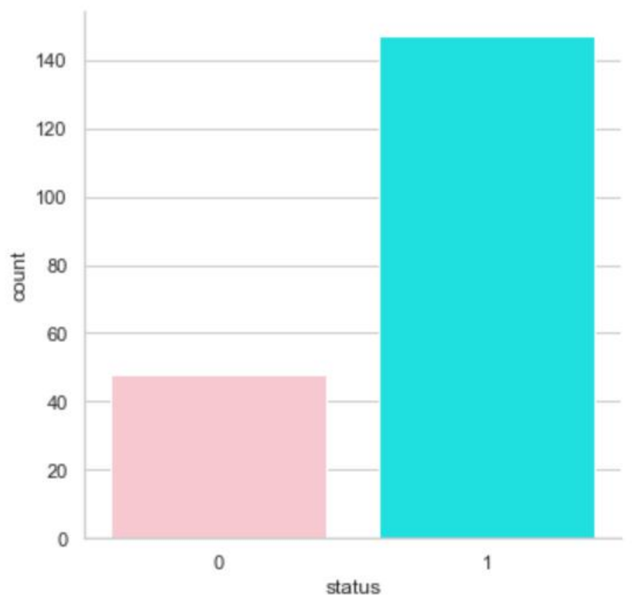
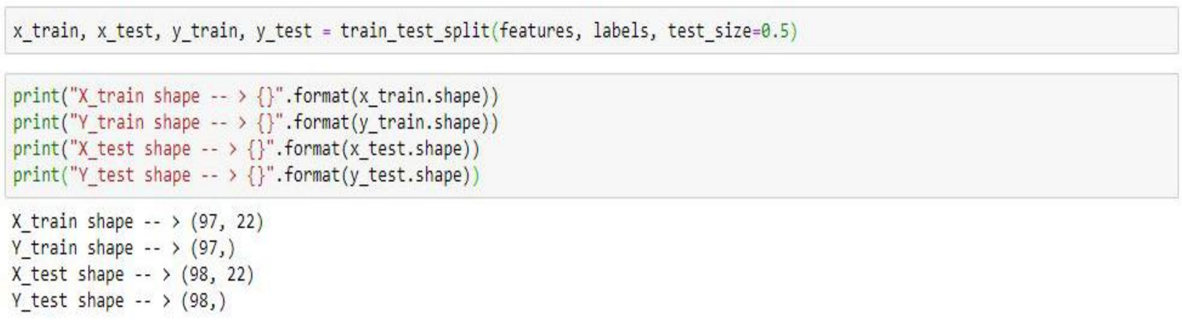


Fig 5.7 - Count of status of each patient

5.8 Distribution in Train and Test data

The following output shows the splitting of dataset with all the features and selected features.

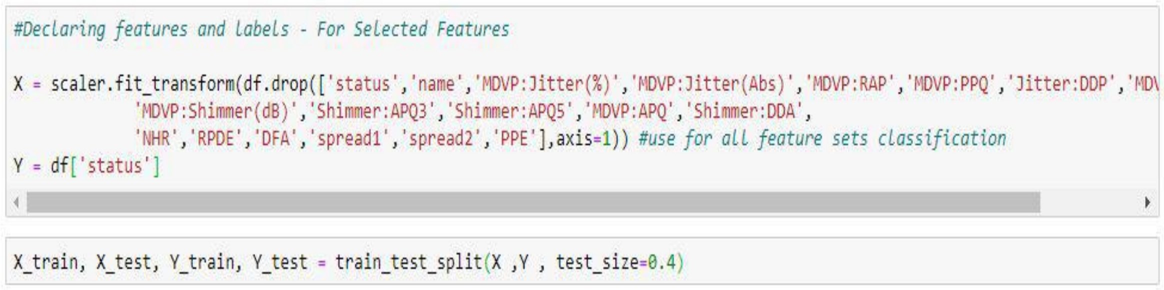


```
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.5)
```

```
print("X_train shape -- > {}".format(x_train.shape))
print("Y_train shape -- > {}".format(y_train.shape))
print("X_test shape -- > {}".format(x_test.shape))
print("Y_test shape -- > {}".format(y_test.shape))
```

```
X_train shape -- > (97, 22)
Y_train shape -- > (97,)
X_test shape -- > (98, 22)
Y_test shape -- > (98,)
```

Fig 5.8(a) - Train and test set splitting for all features



```
#Declaring features and labels - For Selected Features

X = scaler.fit_transform(df.drop(['status','name','MDVP:Jitter(%)','MDVP:Jitter(Abs)','MDVP:RAP','MDVP:PPQ','Jitter:DDP','MD
             'MDVP:Shimmer(dB)','Shimmer:APQ3','Shimmer:APQ5','MDVP:APQ','Shimmer:DDA',
             'NHR','RPDE','DFA','spread1','spread2','PPE'],axis=1)) #use for all feature sets classification
Y = df['status']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X ,Y , test_size=0.4)
```

Fig 5.8(b) - Train and test set splitting for selected features

5.8.1 Distribution in Train data

The following graph represents the distribution of healthy and Parkinson's disease patients recordings in the train set. There are 22 healthy cases and 97 Parkinson's disease cases.

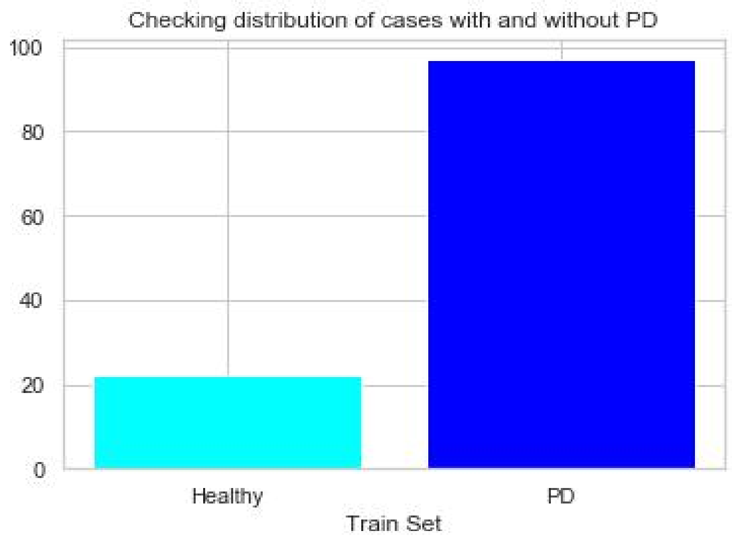


Fig 5.8.1- Distribution in Train set

5.8.2 Distribution in Test data

The following graph represents the distribution of healthy and Parkinson's disease patients recordings in the train set. There are 22 healthy cases and 98 Parkinson's disease cases.

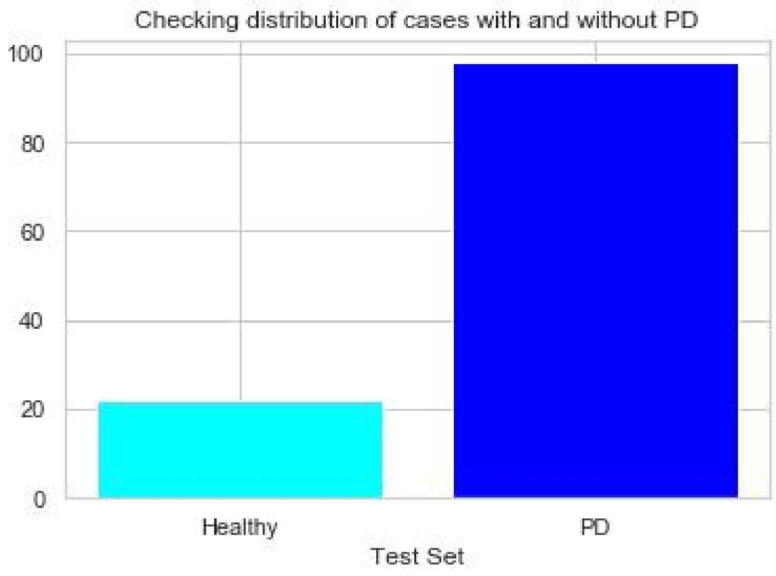
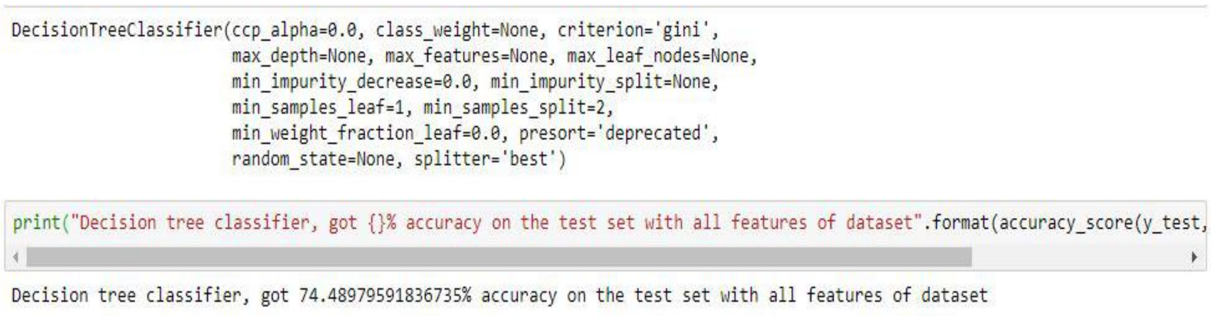


Fig 5.8.2- Distribution in Test set

5.9 Classification Using Decision Tree Classifier

5.9.1 Accuracy on considering all features

The accuracy of the Decision tree classifier on the full feature set is calculated and found to be 74.48%.



```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

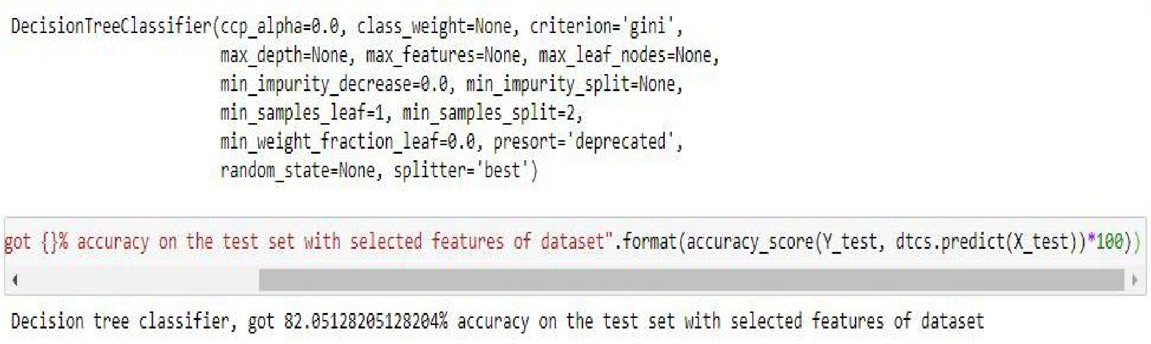
```
print("Decision tree classifier, got {}% accuracy on the test set with all features of dataset".format(accuracy_score(y_test,
```

```
Decision tree classifier, got 74.48979591836735% accuracy on the test set with all features of dataset
```

Fig 5.9.1 - Accuracy using Decision Tree Classifier (All Features)

5.9.2 Accuracy on considering selected features

The accuracy of the Decision tree classifier on the selected feature set is calculated and found to be 82.05%. It is clear from the output that the accuracy of the classifier is greater for selected feature set than for full feature set.



```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
got {}% accuracy on the test set with selected features of dataset".format(accuracy_score(Y_test, dtcs.predict(X_test))*100))
```

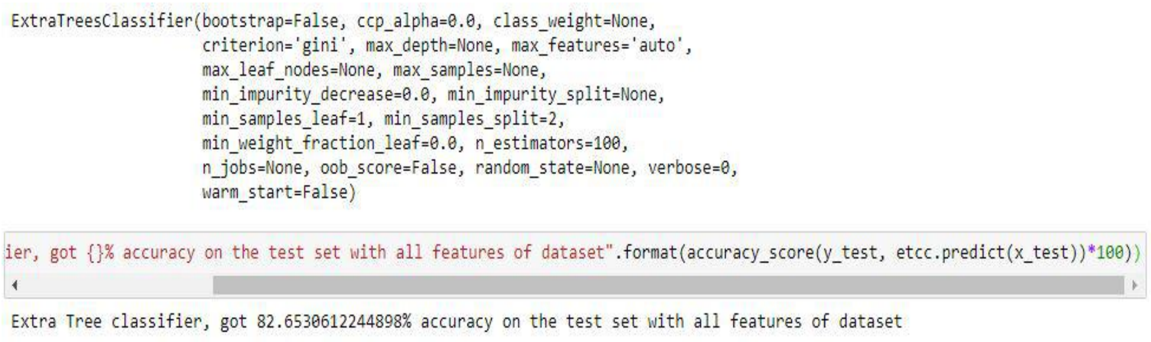
```
Decision tree classifier, got 82.05128205128204% accuracy on the test set with selected features of dataset
```

Fig 5.9.2- Accuracy using Decision Tree Classifier (Selected Features)

5.10 Classification Using Extra Tree Classifier

5.10.1 Accuracy on considering all features

The accuracy of the Extra tree classifier on the full feature set is calculated and found to be 82.65%.



```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

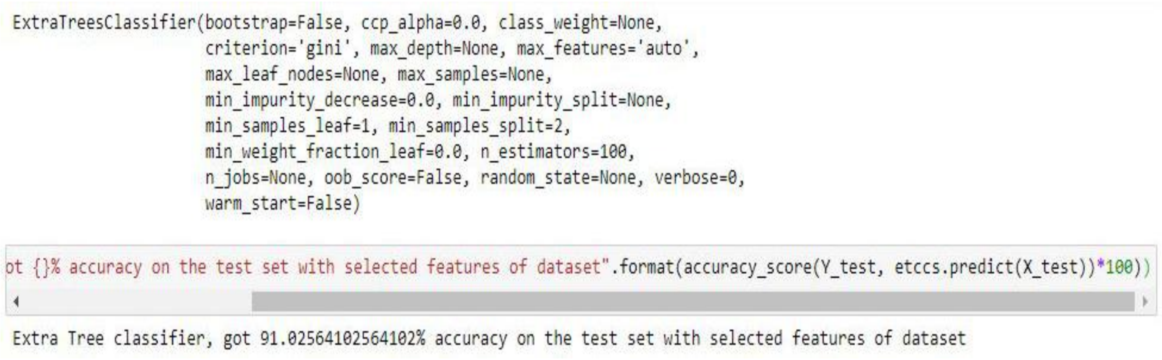
```
ier, got {}% accuracy on the test set with all features of dataset".format(accuracy_score(y_test, etcc.predict(x_test))*100))
```

```
Extra Tree classifier, got 82.6530612244898% accuracy on the test set with all features of dataset
```

Fig 5.10.1- Accuracy using Extra Tree Classifier (All Features)

5.10.2 Accuracy on considering selected features

The accuracy of the Extra tree classifier on the selected feature set is calculated and found to be 91.02%. It is clear from the output that the accuracy of the classifier is greater for selected feature set than for full feature set.



```
ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                     criterion='gini', max_depth=None, max_features='auto',
                     max_leaf_nodes=None, max_samples=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100,
                     n_jobs=None, oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

```
ot {}% accuracy on the test set with selected features of dataset".format(accuracy_score(Y_test, etccs.predict(X_test))*100))
```

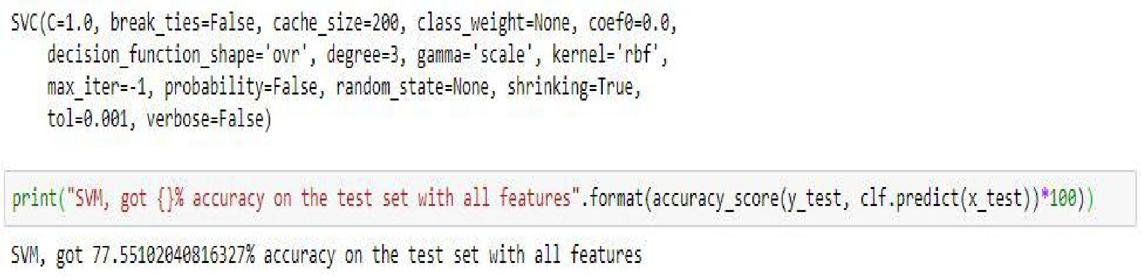
```
Extra Tree classifier, got 91.02564102564102% accuracy on the test set with selected features of dataset
```

Fig 5.10.2- Accuracy using Extra Tree Classifier (Selected Features)

5.11 Classification Using SVM Classifier

5.11.1 Accuracy on considering all Features

The accuracy of the SVM classifier on the full feature set is calculated and found to be 77.55%.



```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

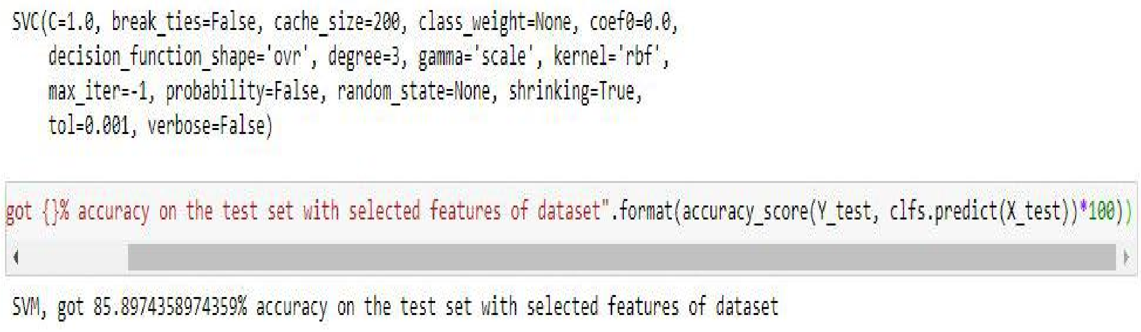
```
print("SVM, got {}% accuracy on the test set with all features".format(accuracy_score(y_test, clf.predict(x_test))*100))
```

```
SVM, got 77.55102040816327% accuracy on the test set with all features
```

Fig 5.11.1- Accuracy using SVM Classifier (All Features)

5.11.2 Accuracy on considering selected Features

The accuracy of the SVM classifier on the selected feature set is calculated and found to be 85.89%. It is clear from the output that the accuracy of the classifier is greater for selected feature set than for full feature set.



```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
got {}% accuracy on the test set with selected features of dataset".format(accuracy_score(Y_test, clfs.predict(X_test))*100))
```

```
SVM, got 85.8974358974359% accuracy on the test set with selected features of dataset
```

Fig 5.11.2- Accuracy using SVM Classifier (Selected Features)

5.12 Comparison of Classifiers

5.12.1 Accuracy comparison with all features

The following is the graph which compares the accuracy of each classifier with all the features. The respective accuracies of classifiers i.e. Decision tree classifier, Extra tree classifier and SVM classifier are 74.48%, 82.65% and 77.55%. It is clear that Extra tree classifier has highest accuracy.

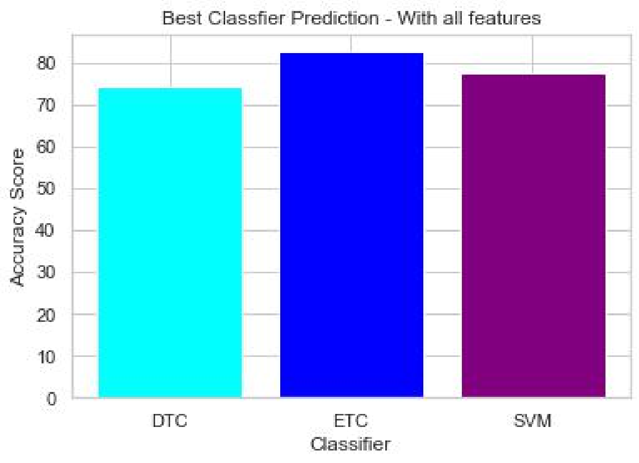


Fig 5.12.1- Best Classifier with all features

5.12.2 Accuracy comparison with selected features

The following is the graph which compares the accuracy of each classifier with selected features. The respective accuracies of classifiers i.e. Decision tree classifier, Extra tree classifier and SVM classifier are 82.05%, 91.02% and 85.89%. It is clear that Extra tree classifier has highest accuracy.

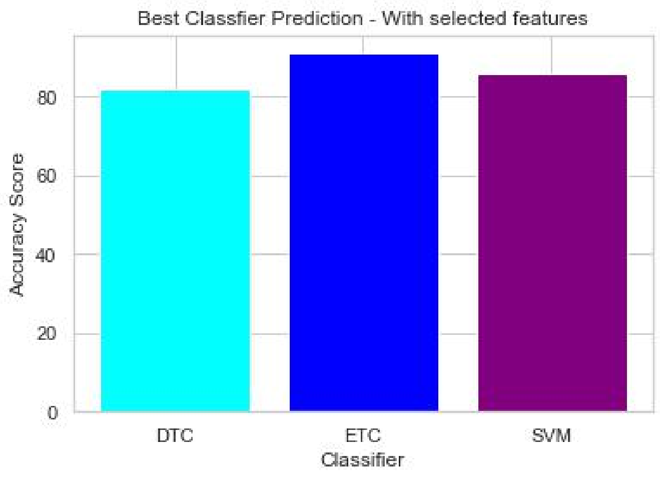


Fig 5.12.2- Best Classifier with selected features

5.13 Cross Validation Iterations

After finding out the best classifier, we have performed the k-fold cross validation on all the features and on selected features and represented the score of each iteration in the form of a bar graphs. The accuracy score obtained as average of all iterations of cross validation for all features is 84.12% and for selected features is 86.46%.

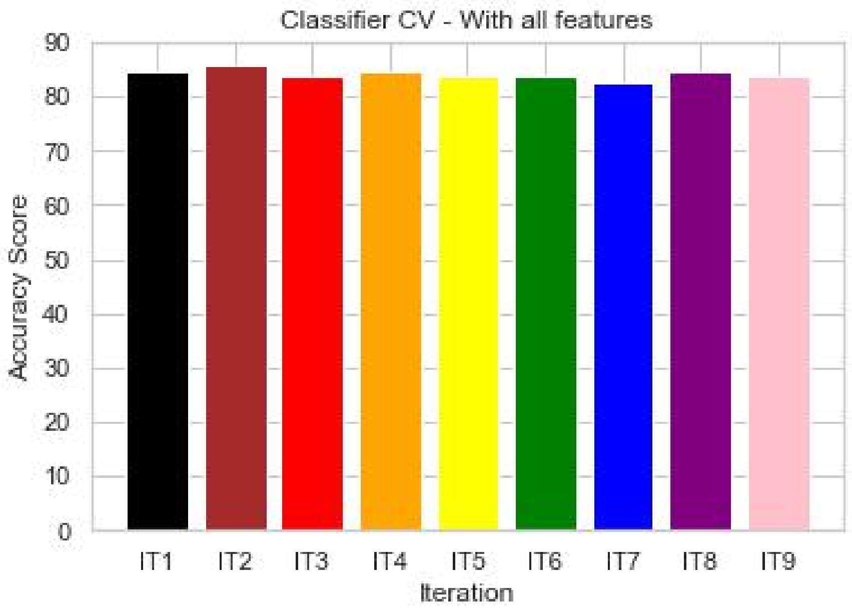


Fig 5.13.1 - Accuracy over CV - with all features

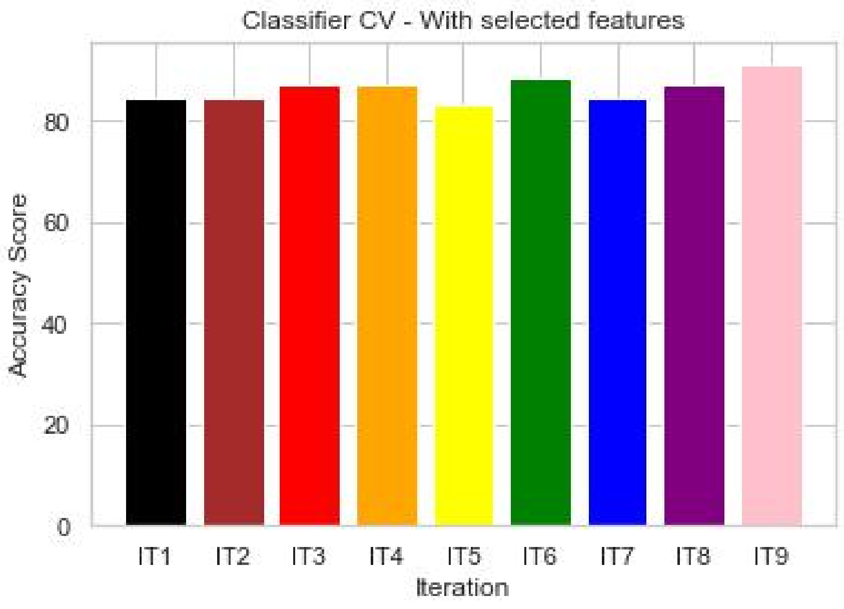
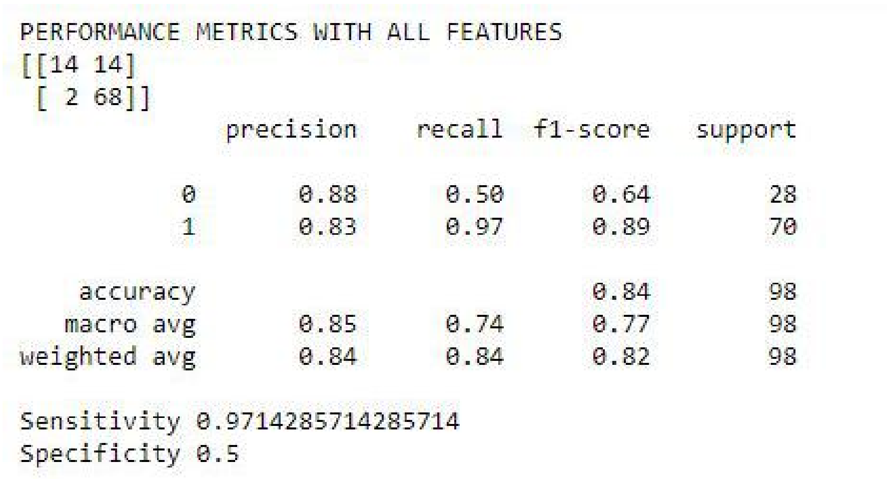


Fig 5.13.2 - Accuracy over CV - with selected features

5.14 Calculating Performance Metrics

5.14.1 Performance metrics for all features

Given below are the calculated performance metrics of the classifier for all the features. From the confusion matrix we can see that fourteen values have been mis-classified. From the precision scores it is clear that healthy subjects can be classified with an accuracy of 88% and Parkinson's disease subjects can be classified with 83% accuracy. The overall accuracy of the classifier is 84%. The F1-score for healthy subjects is 64% and for Parkinson's disease subjects is 89%. The sensitivity is 97% and specificity is 50%.

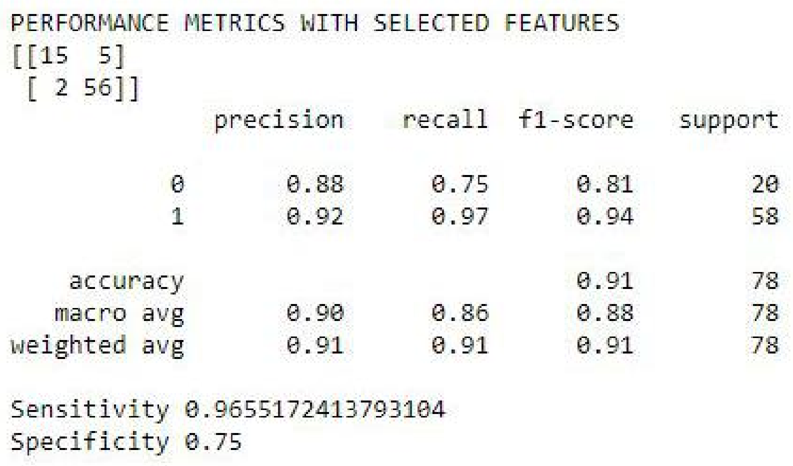


```
PERFORMANCE METRICS WITH ALL FEATURES
[[14 14]
 [ 2 68]]
              precision    recall  f1-score   support

           0       0.88      0.50      0.64        28
           1       0.83      0.97      0.89        70

    accuracy                           0.84        98
   macro avg       0.85      0.74      0.77        98
weighted avg       0.84      0.84      0.82        98

Sensitivity 0.9714285714285714
Specificity 0.5
```

Fig 5.14.1- Performance Metrics for all features

5.14.2 Performance metrics for selected features

Given below are the calculated performance metrics of the classifier for selected features. From the confusion matrix we can see that five values have been mis-classified. From the precision scores it is clear that healthy subjects can be classified with an accuracy of 88% and Parkinson's disease subjects can be classified with 92% accuracy. The overall accuracy of the classifier is 91%. The F1-score for healthy subjects is 81% and for Parkinson's disease subjects is 94%. The sensitivity is 96% and specificity is 75%.



```
PERFORMANCE METRICS WITH SELECTED FEATURES
[[15  5]
 [ 2 56]]
              precision    recall  f1-score   support

           0       0.88      0.75      0.81        20
           1       0.92      0.97      0.94        58

    accuracy                           0.91        78
   macro avg       0.90      0.86      0.88        78
weighted avg       0.91      0.91      0.91        78

Sensitivity 0.9655172413793104
Specificity 0.75
```

Fig 5.14.2- Performance Metrics for selected features

5.14.3 Comparison of Performance metrics

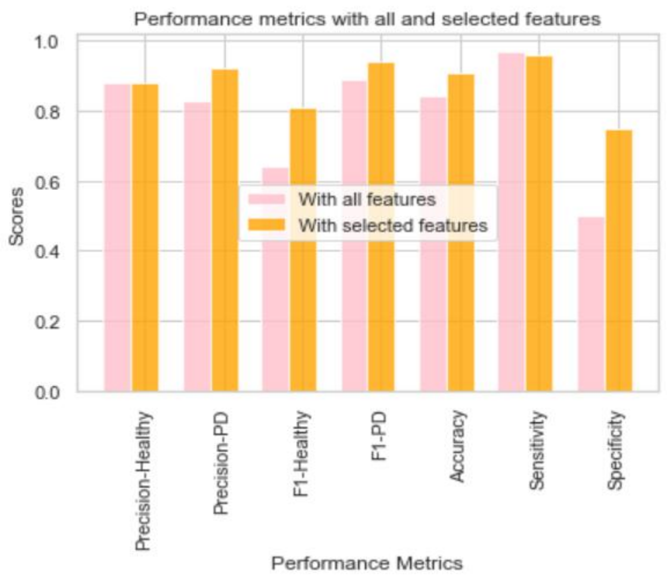


Fig 5.14.3 - Comparison of Performance Metrics

5.15 F1 Score Comparison

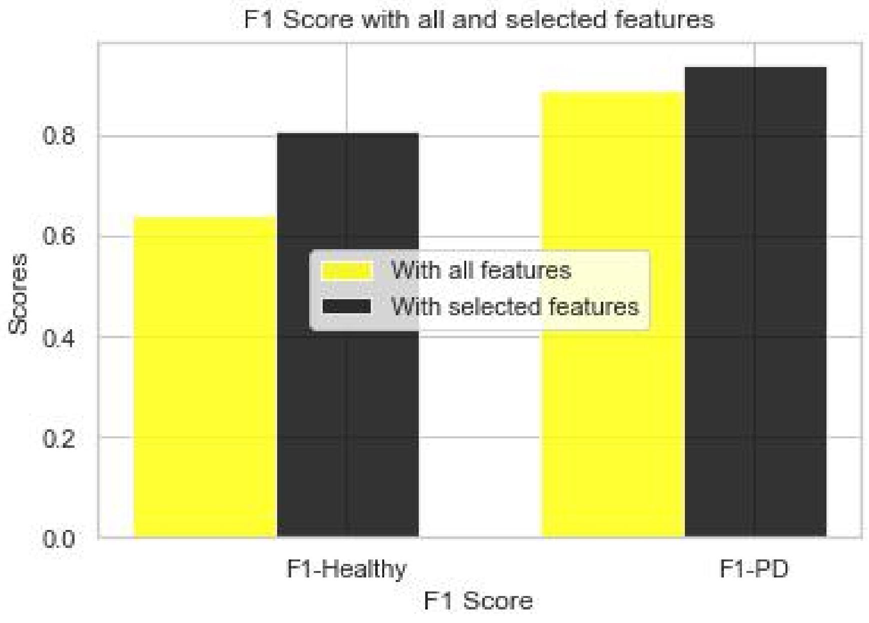


Fig 5.15 - F1 Score comparison

5.16 Accuracy Comparison

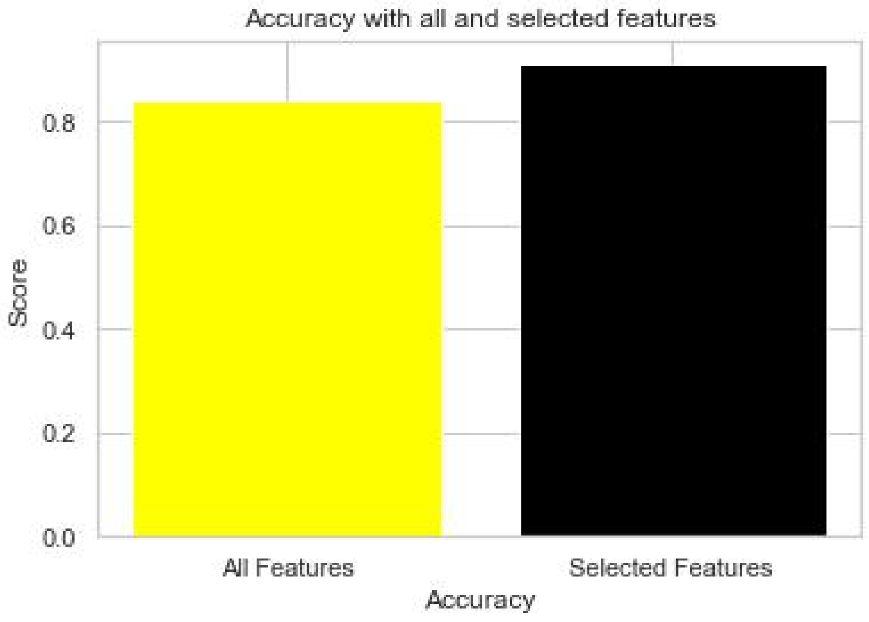


Fig 5.16 - Accuracy score comparison

CHAPTER 6

CONCLUSION

6.1 Conclusion

The detection and analysis of Parkinson’s disease has allowed us to explore classification methods other than Support Vector Machine. By analysis of the dataset and their correlation with each other, we could even extract the hyper parameters and features which had a prominent influence in the accuracy scores. During the process of exploration, we briefly used classifying methods using all the features and selected features. Hence, another innovative part of proposed study to use features selection algorithm to select a relevant subset of features that improve the classification performance of the diagnosis system. Both of them underwent through SVM Classifier, Decision Tree Classifier and Extra Tree classifier methods.

According to the proposed system, after the classification using L1 norm SVM using 10 fold cross validation, we compared the accuracies for the Radial Based Function and Linear kernel. After comparison, we could conclude that the linear kernel provided better accuracies and further used it for doing classification using the selected features. Later we used the other classifying methods to choose the best one for the analysis. After evaluation we realized that the Extra Tree Classifier provided a better accuracy of 91% with selected features in determining the patients suffering with PD. We performed the K fold cross validations on the selected features as 9 iterations and presented their accuracy scores in the form a bar graph. At last the performance metrics which includes the F1 score, accuracy, precision, recall, support, sensitivity and specificity was calculated. It is found that all the performance metrics score higher when selected features are taken instead of all the features. Also unlike the basepaper we found that Extra tree classifier is comparatively better than SVM classifier.

6.1.1 Limitations

1. the SVM algorithm has several key parameters that need to be set correctly to achieve the best classification results.
2. As the support vector classifier works by putting data points, above and below the classifying hyper plane there is no probabilistic explanation for the classification.
3. Computation for k-fold cross validation is longer
4. Another limitation in the project is that the Extra tree classifier is less efficient when there are more noisy features.

6.2 Future Work

Our project can further be extended by the following ways:

1. The prediction of the Parkinson's disease can be made more attractive by developing an attractive and fast user interface. Using our code as the back end, an easy to manage front end can be developed.
2. Other than the existing dataset, the model can be used to train data sets with other features. Other than the already existing data, new recordings can be taken and converted into the required format of the training dataset.
3. In the future other feature selection algorithms, optimization and deep neural network classification methods can be utilized to further increase the performance of the diagnosis.

REFERENCES

[1] IEEE Journal of Biomedical and Health Informatics, VOL. 17, NO. 4, JULY 2013, “Collection and Analysis of a Parkinson Speech Dataset with Multiple Types of Sound Recordings”, Betul Erdogdu Sakar, M. Erdem Isenkul, C. Okan Sakar, Ahmet Sertbas, Fikret Gurgen, Sakir Delil, Hulya Apaydin, and Olcay Kursun.

[2] Journal of Royal Society Interface (2011), “Nonlinear speech analysis algorithms mapped to a standard metric achieve clinically useful quantification of average Parkinson’s disease symptom severity”, Athanasios Tsanas, Max A. Little, Patrick E. McSharry and Lorraine O. Ramig

[3] B. E. Sakar et al., ‘‘Collection and analysis of a Parkinson speech dataset with multiple types of sound recordings,’’ IEEE J. Biomed. Health Inform.vol. 17, no. 4, pp. 828–834, Jul. 2013.

[4] Parkinson’s Disease: National Clinical Guideline for Diagnosis and Management in Primary and Secondary Care, Nat. Collaborating Centre Chronic Conditions, London, U.K., 2006.

[5] B. Harel, M. Cannizzaro, and P. J. Snyder, ‘‘Variability in fundamental frequency during speech in prodromal and incipient Parkinson’s disease: A longitudinal case study,’’ Brain Cogn., vol. 56, no. 1, pp. 24–29. Jun. 2004.

[6] A. Tsanas, M. A. Little, P. E. McSharry, J. Spielman, and L. O. Ramig, ‘‘Novel speech signal processing algorithms for high-accuracy classification of Parkinson’s disease,’’ IEEE Trans. Biomed. Eng., vol. 59, no. 5, pp. 1264–1271, May 2012.

[7] M. A. Little, P. E. McSharry, E. J. Hunter, J. Spielman, and L. O. Ramig, ‘‘Suitability of dysphonia measurements for telemonitoring of Parkinson’s disease,’’ IEEE Trans. Biomed. Eng., vol. 56, no. 4, pp. 1015–1022, Apr. 2009.

[8] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig, ‘‘Nonlinear speech analysis algorithms mapped to a standard metric achieve clinically useful quantification of average Parkinson’s disease symptom severity,’’ J. Roy. Soc. Interfaces, vol. 8, pp. 842–855, Jun. 2011.

[9] Y. Liu, J.-W. Bi, and Z.-P. Fan, ‘‘Multi-class sentiment classification: The experimental comparisons of feature selection and machine learning algorithms,’’ Expert Syst. Appl., vol. 80, pp. 323–339, Sep. 2017. doi: 10.1016/j.eswa.2017.03.042.

[10] Y. Liu, B. Jian-Wu, and Z.-P. Fan, ‘‘A method for multi-class sentiment classification based on an improved one-vs-one (OVO) strategy and the support vector machine (SVM) algorithm,’’ Inf. Sci., vol. 394, pp. 38–52, Jul. 2017. doi: 10.1016/j.ins.2017.02.016.

[11] İ. Cantürk and F. Karabiber, ‘‘A machine learning system for the diagnosis of Parkinson’s disease from speech signals and its application to multiple speech signal types,’’ Arabian J. Sci. Eng., vol. 41, pp. 5049–5059, Dec. 2016.

[12] S.-S. Hong, W. Lee, and M.-M. Han, ‘‘The feature selection method based on genetic algorithm for efficient of text clustering and text classification,’’ Int. J. Adv. Soft Comput. Appl., vol. 7, pp. 2074–8523, Mar. 2015.

APPENDIX

Importing necessary libraries :

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

from sklearn.svm import LinearSVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import ExtraTreesClassifier

from sklearn import svm

from sklearn.model_selection import KFold

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report,confusion_matrix
```

Reading dataset as dataframe :

```
df = pd.read_csv(r"C:\Users\Hp\Desktop\Major Project\NEW Parkinsons disease L1 norm SVM\data.csv")

df.head()

df.shape
```

Plotting Scatter matrix plot :

```
sns.set(style="whitegrid")

sns.pairplot(df)
```

Plotting Pearson Correlation Heatmap :

```
pearsoncorr = df.drop(['name'],axis=1).corr(method='pearson')

heat_map=sb.heatmap(pearsoncorr,cmap='RdPu',linewidth=0.5)

plt.show()
```

Changing name column values to numerical values :

```
df['name'] = np.arange(len(df))
df.head()
```

Counting status value of each patient :

```
custom_palette=["pink","cyan"]
sns.set_palette(custom_palette)
sns.catplot(x='status',kind='count',data=df)
df['status'].value_counts(bins=2)
```

Plotting Box Plot :

```
custom_palette=["yellow","cyan"]
sns.set_palette(custom_palette)
for i in df:
    if i != 'status' and i != 'name':
        sns.catplot(x='status',y=i,kind='box',data=df)
```

Implementing MinMax Scaler :

```
scaler=MinMaxScaler((-1,1))
scaled_features = scaler.fit_transform(df.drop(['status','name'],axis=1))
```

Declaring Features and Labels :

```
features=df.drop(['status','name'],axis=1)
labels=df['status']
print(features.shape)
print(labels[0])
features.columns.tolist()
```

Splitting into train and test sets for all features :

```
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.5)
print("X_train shape -- > {}".format(x_train.shape))
print("Y_train shape -- > {}".format(y_train.shape))
print("X_test shape -- > {}".format(x_test.shape))
print("Y_test shape -- > {}".format(y_test.shape))
```

Distribution graph for train set :

```
x1 = ['Healthy','PD']
y1 =[22,98]
plt.bar(x1,y1,color=['cyan','blue'])
plt.xlabel('Train Set')
plt.title('Checking distribution of cases with and without PD')
plt.show()
```

Distribution graph for test set :

```
x2 = ['Healthy','PD']
y2 =[22,98]
plt.bar(x2,y2,color=['cyan','blue'])
plt.xlabel('Test Set')
plt.title('Checking distribution of cases with and without PD')
plt.show()
```

Implementing L1-Norm SVM for feature selection :

```
lone = LinearSVC(C=0.02, penalty="l1", dual=False)
lone.fit(x_train,y_train)
print(lone.coef_)
indices = np.argsort(lone.coef_)[::-1]
```

Plotting feature importance graph :

```
x5=['MDVP:Fo(Hz)','MDVP:Fhi(Hz)','MDVP:Flo(Hz)','MDVP:Jitter(%)','MDVP:Jitter(Abs)',
'MDVP:RAP','MDVP:PPQ','Jitter:DDP','MDVP:Shimmer','MDVP:Shimmer(dB)','Shimmer:
APQ3','Shimmer:APQ5','MDVP:APQ','Shimmer:DDA','NHR','HNR','RPDE','DFA','spread1','
spread2','D2','PPE']

y5 = [-0.0044,0.007,-0.006,0,0,0,0,0,0,0,0,0,0,0,0,0.021,0,0,0,0,0.086,0]

plt.bar(x5,y5,color=['orange'])

plt.xlabel('Feature Index')

plt.ylabel('Feature Score')

plt.title('Feature Importance Graph')

plt.xticks(rotation=90)

plt.show()
```

Splitting dataset into train and test set for selected features :

```
X=scaler.fit_transform(df.drop(['status','name','MDVP:Jitter(%)','MDVP:Jitter(Abs)','MDVP:
RAP','MDVP:PPQ','Jitter:DDP','MDVP:Shimmer','MDVP:Shimmer(dB)','Shimmer:APQ3','S
himmer:APQ5','MDVP:APQ','Shimmer:DDA','NHR','RPDE','DFA','spread1','spread2','PPE'],
axis=1)) #use for all feature sets classification

Y = df['status']

X_train, X_test, Y_train, Y_test = train_test_split(X ,Y , test_size=0.5)
```

Classification using Decision tree classifier for all features :

```
dtc = DecisionTreeClassifier()

dtc.fit(x_train, y_train)

print("Decision tree classifier, got {}% accuracy on the test set with all features of
dataset".format(accuracy_score(y_test, dtc.predict(x_test))*100))
```

Classification using Extra tree classifier for all features :

```
etcc =ExtraTreesClassifier()

etcc.fit(x_train,y_train)

print("Extra Tree classifier, got {}% accuracy on the test set with all features of
dataset".format(accuracy_score(y_test, etcc.predict(x_test))*100))
```

Classification using SVM classifier for all features :

```
clf = svm.SVC()

clf.fit(x_train,y_train)

print("SVM, got {}% accuracy on the test set with all features".format(accuracy_score(y_test,
clf.predict(x_test))*100))
```

Predicting best classifier for all features set:

```
x3 = ['DTC','ETC','SVM']

y3 =[74.48,82.65,77.55]

plt.bar(x3,y3,color=['cyan','blue','purple'])

plt.xlabel('Classifier')

plt.ylabel('Accuracy Score')

plt.title('Best Classfier Prediction - With all features')

plt.show()
```

K-Fold CV for all features set :

```
kf = KFold(n_splits=9,shuffle=False)

kf.split(features)

accuracy_model = []

for train_index, test_index in kf.split(features):

     model = etcc.fit(x_train, y_train)

     accuracy_model.append(accuracy_score(y_test,model.predict(x_test),
     normalize=True)*100)

print(accuracy_model)

y_pred = etcc.predict(x_test)
```

Plotting K-Fold CV graph for all features set :

```
x4 = ['IT1','IT2','IT3','IT4','IT5','IT6','IT7','IT8','IT9']

y4=[84.6938775510204,85.71428571428571,83.6734693877551,84.6938775510204,
83.6734693877551,83.6734693877551,82.6530612244898,84.6938775510204,
83.6734693877551]

plt.bar(x4,y4,color=['black','brown','red','orange','yellow','green','blue','purple','pink'])

plt.xlabel('Iteration')

plt.ylabel('Accuracy Score')

plt.title('Classifier CV - With all features')

plt.show()
```

Calculating performance metrics for all features set :

```
CM1 = confusion_matrix(y_test,y_pred)

TN1 = CM1[0][0]

FN1 = CM1[1][0]

TP1 = CM1[1][1]

FP1 = CM1[0][1]

TPR1 = TP1/(TP1+FN1)

TNR1 = TN1/(TN1+FP1)

PPV1 = TP1/(TP1+FP1)

NPV1 = TN1/(TN1+FN1)

FPR1 = FP1/(FP1+TN1)

FNR1 = FN1/(TP1+FN1)

FDR1 = FP1/(TP1+FP1)

ACC1 = (TP1+TN1)/(TP1+FP1+FN1+TN1)

print("PERFORMANCE METRICS WITH ALL FEATURES")

print(CM1)

print(classification_report(y_test,y_pred))

print("Sensitivity",TPR1)

print("Specificity",TNR1)
```

Classification using Decision tree classifier for selected features :

```
dtcs = DecisionTreeClassifier()

dtcs.fit(X_train, Y_train)

print("Decision tree classifier, got {}% accuracy on the test set with selected features of
dataset".format(accuracy_score(Y_test, dtcs.predict(X_test))*100))
```

Classification using Extra tree classifier for selected features :

```
etccs =ExtraTreesClassifier()

etccs.fit(X_train,Y_train)

print("Extra Tree classifier, got {}% accuracy on the test set with selected features of
dataset".format(accuracy_score(Y_test, etccs.predict(X_test))*100))
```

Classification using SVM classifier for selected features :

```
clfs = svm.SVC()

clfs.fit(X_train,Y_train)

print("SVM, got {}% accuracy on the test set with selected features of
dataset".format(accuracy_score(Y_test, clfs.predict(X_test))*100))
```

Predicting best classifier for selected features set :

```
x6 = ['DTC','ETC','SVM']

y6 =[82.05,91.02,85.89]

plt.bar(x6,y6,color=['cyan','blue','purple'])

plt.xlabel('Classifier')

plt.ylabel('Accuracy Score')

plt.title('Best Classfier Prediction - With selected features')

plt.show()
```

K-Fold CV for selected features set :

```
kf1 = KFold(n_splits=9,shuffle=False)

kf1.split(X)

accuracy_model1 = []

for train_index, test_index in kf1.split(X):

        model1 = etccs.fit(X_train, Y_train)

        accuracy_model1.append(accuracy_score(Y_test,model1.predict(X_test),
        normalize=True)*100)

print(accuracy_model1)

y_preds = etccs.predict(X_test)
```

Plotting K-Fold CV graph for selected features set :

```
x7 = ['IT1','IT2','IT3','IT4','IT5','IT6','IT7','IT8','IT9']

y7=[84.61538461538461,84.61538461538461,87.17948717948718,87.17948717948718,83.3
3333333333334,88.46153846153845,84.61538461538461,87.17948717948718,
91.02564102564102]

plt.bar(x7,y7,color=['black','brown','red','orange','yellow','green','blue','purple','pink'])

plt.xlabel('Iteration')

plt.ylabel('Accuracy Score')

plt.title('Classifier CV - With selected features')

plt.show()
```

Calculating performance metrics for selected features set :

```
CM2 = confusion_matrix(Y_test,y_preds)

TN2 = CM2[0][0]

FN2 = CM2[1][0]

TP2 = CM2[1][1]

FP2 = CM2[0][1]

TPR2 = TP2/(TP2+FN2)

TNR2 = TN2/(TN2+FP2)

PPV2 = TP2/(TP2+FP2)

NPV2 = TN2/(TN2+FN2)
```

```
FPR2 = FP2/(FP2+TN2)
FNR2 = FN2/(TP2+FN2)
FDR2 = FP2/(TP2+FP2)
ACC2 = (TP2+TN2)/(TP2+FP2+FN2+TN2)
print("PERFORMANCE METRICS WITH SELECTED FEATURES")
print(CM2)
print(classification_report(Y_test,y_preds))
print("Sensitivity",TPR2)
print("Specificity",TNR2)
```

Graph for comparing all the performance metrics :

```
n_groups = 7
v_all = (0.88,0.83,0.64,0.89,0.84,0.97,0.5)
v_select = (0.88,0.92,0.81,0.94,0.91,0.96,0.75)
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8
rects1 = plt.bar(index, v_all, bar_width,alpha=opacity,color='pink',label='With all features')
rects2=plt.bar(index+bar_width,v_select,bar_width,alpha=opacity,color='orange',
label='With selected features')
plt.xlabel('Performance Metrics')
plt.ylabel('Scores')
plt.title('Performance metrics with all and selected features')
plt.xticks(index + bar_width, ('Precision-Healthy', 'Precision-PD', 'F1-Healthy', 'F1-PD', 'Accuracy','Sensitivity','Specificity'),rotation=90)
plt.legend(loc='center')
plt.show()
```

Graph for comparing F1-Score for full feature set and selected feature set :

```
n_groups = 2

f1_all = (0.64,0.89)

f1_select = (0.81,0.94)

fig, ax = plt.subplots()

index = np.arange(n_groups)

bar_width = 0.35

opacity = 0.8

rects1 = plt.bar(index, f1_all, bar_width,alpha=opacity,color='yellow',label='With all features')

rects2=plt.bar(index+bar_width,f1_select,bar_width,alpha=opacity,color='black',

label='With selected features')

plt.xlabel('F1 Score')

plt.ylabel('Scores')

plt.title('F1 Score with all and selected features')

plt.xticks(index + bar_width, ( 'F1-Healthy', 'F1-PD'))

plt.legend(loc='center')

plt.show()
```

Graph for comparing Accuracy for full feature set and selected feature set :

```
x8 = ['All Features', 'Selected Features']

y8 =[0.84,0.91]

plt.bar(x8,y8,color=['yellow','black'])

plt.xlabel('Accuracy')

plt.ylabel('Score')

plt.title('Accuracy with all and selected features')

plt.show()
```