POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

Automatic Classification of Parkinson's disease patients vs Healthy controls using a vision-based finger-tapping test

Supervisors

Prof. Gabriella OLMO

Doc. Gianluca AMPRIMO

Candidate

Seyedeh Neda HARIRI

December 2023

Summary

In recent years, there have been remarkable advancements in machine learning. The rise in computational power and the abundance of data have made these systems indispensable in various fields, such as disease follow-up. Parkinson's disease (PD) is the second most prevalent neurodegenerative disorder worldwide. The most notable symptoms of PD are bradykinesia, rest tremor, postural instability, and rigidity. In clinical practice, a variety of diagnostic techniques, including the finger-tapping test, the walk test, the gait analysis, and the evaluation of speech impairment, are used to examine these symptoms.
According to the Movement Disorder Society's published guidelines, the finger-tapping test (FTT) is one of the most frequently administered evaluations of bradykinesia, but manual visual evaluations can result in score discrepancy due to clinicians' subjectivity. Moreover, the application of wearable sensors necessitates physical contact and may inhibit the natural movement patterns of PD patients. The information related to these patterns was provided in the vision-based 3D Parkinson's disease (PD) hand dataset, consisting of 133 finger-tapping video samples, including recordings of 35 PD patients and 60 healthy controls. This endeavor has considered the movement of the index and thumb digits toward and away from one another. Then, Python libraries were used to acquire the distance and velocity signals of these movements. Utilizing the tsfresh library for feature extraction was the next step. Identifying and deriving meaningful features from time series is a time-consuming process because scientists and engineers must take into account the numerous algorithms of signal processing and time series analysis. Then to remove redundant features and improve the classification algorithm's accuracy, feature selection based on extracted features has been implemented. Methods of Boruta and Principal component analysis (PCA) were applied to select meaningful features. Furthermore, several supervised machine learning classification algorithms have been evaluated to determine whether or not a person has PD. These algorithms include k-nearest neighbors (KNN), random forest (RF), eXtreme Gradient Boosting (XGB), and support vector machines (SVM). These algorithms were selected based on their ability to correspond to medical evaluation criteria, their visualization capabilities, and the data size and computation constraints of

real-world applications. A 5-fold cross-validation method was exploited to verify the final classification accuracy in order to quantitatively compare the performance of distinct classifiers. In fact, two feature selection methods and four machine learning classifiers were combined to work out accuracy, precision, recall, and f1-score. These results would lead to the evaluation of the optimal combination method for diagnosing PD.
According to the results obtained, the combination of Boruta as the feature selector and Support Vector Machine as the classifier demonstrated the highest performance across all four metrics. This point should be noted that obtaining 100% in some metrics is probably due to data scarcity.

Acknowledgements

I wish to express my genuine and profound appreciation and gratitude to my supervisors, Prof. Gabriella Olmo and Doc. Gianluca Amprimo, for granting me the opportunity to do research and work under their guidance. I express my gratitude and fondness towards my parents for their unwavering support and affection that they have provided me with throughout the entirety of my existence. Lastly, I would like to extend my gratitude to my colleagues at the Polytechnico di Torino University for their invaluable assistance in accomplishing my academic pursuits and thesis. I am extending my gratitude and admiration to those individuals who have profoundly influenced my life and academic trajectory.

Table of Contents

List of Tables

List of Figures

Acronyms

AI

Artificial Intelligence

ML

Machine Learning

PD

Parkinson’s Disease

HC

Healthy Controls

UPDRS

Unified Parkinson’s Disease Rating Scale

MDS

Movement Disorder Society

QOL

Quality of Life

FTT

Finger Tapping Test

FT

Finger Tapping

PCA

Principal Component Analysis

KNN	K Nearest Neighbors
RF	Random Forest
XGB	Extreme Gradient Boosting
SVM	Support Vector Machine
CV	Cross Validation
OOB	Out of the Bag
SRM	Structural Risk Minimization
SD	Standard Deviation

Chapter 1

Introduction

Parkinson's Disease (PD) is a neurodegenerative disorder with a prevalence of 1% in the over-65 population, which is projected to increase significantly as the global population ages [1]. Standardized scales aim to quantify the progression and severity of disease, symptoms, and quality of life. The Unified Parkinson's Disease Rating Scale (UPDRS) is the most common and globally recognized scale for assessing disease severity. In 2008, the Movement Disorder Society (MDS) created and published the revised UPDRS scale, also known as the MDS-UPDRS [2].

The primary symptoms of Parkinson's disease are bradykinesia, rest tremor, rigidity, and postural instability. In practice, physicians use a variety of tests, including the finger-tapping test, gait analysis, and speech impairment examination to evaluate these symptoms. Among all PD symptoms, bradykinesia is the most essential according to the guidelines published by the Movement Disorder Society, and the finger tapping test (FTT) is one of the most extensively used tests to assess bradykinesia [3].

Clinicians frequently utilize the FTT task. It involves tapping the tips of the thumb and index finger as swiftly and with as much amplitude as possible for a predetermined amount of repetitions or seconds. Numerous research intended to objectively evaluate FTT. Three major categories can be identified: (i) peripheral device-based solutions, such as including inertial measurement devices and instrumented gloves (ii) smartphone-based evaluation wherein the interaction with the device's display is translated into an FTT-equivalent; (iii) vision-based systems employing either RGB, RGB-Depth, or Depth video cameras [1]. The first form of treatment is typically more invasive, may be impractical in unsupervised settings, and is therefore difficult for patients to administer on their own. Smartphone-based solutions discover only indirect correlations between the metrics collected by the touchscreen and the severity scores, making them difficult to interpret in terms of the standardized task. Lastly, vision-based approaches utilize video cameras and markerless tracking systems based on shallow or deep learning models, which first estimate a 2D or 3D hand skeleton and then evaluate a series of FTT-descriptive features. These characteristics may be used to classify subjects or to estimate a severity score [1].

The initial stage involves capturing the hand movements of healthy and PD patients. 133 finger-tapping video samples, including recordings of 35 PD patients and 60 healthy controls, were provided for this study. The details of the experimental session, participants, and materials used to capture the FFT were taken from a different study [1]. The data extracted from the recordings were then converted to JSON format for use in subsequent operations. Then, python libraries have been used to acquire signals such as distance and velocity. In this undertaking, the movement of the index and thumb fingers toward and away from one another has been considered. Distance is the quantitative or occasionally qualitative measurement of the separation between two objects or points. Velocity is the measurement of the pace and direction of an object's motion. A foundational concept in kinematics, the branch of classical mechanics that studies the motion of bodies, is velocity. Average velocity equals total distance traveled divided by total duration Joints 4 and 8 must be analyzed to compute these signals for the thumb and index finger.

The tsfresh library for feature extraction has been used. Tsfresh is a tool utilized for the methodical creation of features from time series and other sequential data [4]. These data have the characteristic of being arranged according to an independent variable. The predominant independent variable is typically time, sometimes referred to as a time series.

The next step was feature selection. Feature selection involves the identification and selection of the most impactful features from a set of features to decrease the dimensionality of the feature space [5]. It is a crucial undertaking in the field of statistical pattern recognition. The majority of feature selection algorithms have been developed using objective functions that are typically intuitive but may deviate from the fundamental aims of feature selection [6]. The Boruta and Principal Component Analysis (PCA) methods were utilized to identify significant features and remove redundant ones.

The classification was the following action. To determine whether a person has Parkinson's disease or not, various supervised classification methods in the field of machine learning have been implemented. Machine Learning categorization labels input data [7]. Labels divide the data into two groups. The techniques encompassed in this set are k-nearest neighbors (KNN), random forests (RF), eXtreme Gradient Boosting (XGBoost), and support vector machine with (SVM). Machine Learning categorization labels input data. Labels divided the data into two groups, here. These algorithms were selected based on their capacity to align with medical assessment standards and their potential for visualization, while also taking into account data volume and computational constraints in practical scenarios [3].

Subsequently, K-fold cross-validation was used to mitigate the problem of overfitting when there was insufficient data available. During cross-validation, a subset of training data sets was generated and used to train the machine learning model. Cross-validation was employed to address the issue of overfitting. The model was trained using k-1 subsets and subsequently evaluated on the kth subset of the data. The subset with the kth index was excluded from the training process [7], [8].

Afterward, accuracy, recall, precision, and f1-score metrics assessed the performance of the classification models and measured the efficacy of a retrieval mechanism [8].

Chapter 2

Literature Survey

2.1 Overview on Parkinson's Disease

Parkinson's disease (PD), which affects millions of people worldwide and is becoming more prevalent as the world's population ages, is now the second most prevalent neurological disease after Alzheimer's disease. Both motor and non-motor symptoms are experienced by people with Parkinson's disease (PD). The former include postural instability, rigidity, bradykinesia, gait impairments, resting tremors, and relevant alterations in their speech, including articulation, phonation, fluency, and prosody landmarks. It should be mentioned that these physical limitations present a significant problem for the patient and their family because these illnesses affect patients' moods and attitudes in addition to their motor symptoms and communication abilities. The latter include anxiety, sadness, and sleep disturbance. Poor quality of life (QOL), higher health care costs, and a greater load on carers are always consequences of these symptoms [9], [10].

The literature shows that despite these negative symptoms, many PD patients have limited access to traditional face-to-face healthcare because of obstacles including a distance barrier, financial burden, mobility issues, or a lack of time. A potential health risk resulted from the fact that more than 40% of PD patients did not obtain specialized care from neurologists soon after diagnosis. Sustainable healthcare is therefore crucial from a therapeutic perspective for these individuals since it has the potential to lessen the severity of their PD and improve QOL [9].

On the other hand, patients with neurodegenerative disorders must deal with comorbidity as another issue because these individuals frequently have many illnesses, such as cardiovascular disease, incontinence, etc. Therefore, it is essential to establish multidisciplinary teams to provide patients with growing neurological impairment with the necessary care. Neurologists should make up these teams or professional groups for the diagnostic and early stages, general practitioners for the monitoring phase, and rehabilitation services for the preservation of the patient's quality of life [10].

Moreover, the typical wait time between appointments is one of the most important aspects of a patient's treatment. Monitoring long-term patients necessitates therapeutic activities including pharmacological treatment and rehabilitation programs that aid patients in maintaining their neuromotor competence, which adds cost and complexity. In the course of the medical consultation, the professional only spends about 15 minutes observing the patient. The patient is then given a specific dose by the neurologist to control their symptoms. The primary problem with this regimen is that the prescribed dosage won't be reviewed for a while. It is important to note that the patient's medication status (on/off state) has not been taken into account in the current monitoring of patients. Dyskinesia, bradykinesia, and other symptoms might not be present at this time for the consultation. This may lead to incorrect pharmacological treatment or ineffective monitoring. Finally, if a lesser dose is administered, patients may have less self-autonomy. On the other hand, if the dose is too high, the effectiveness of the medication may be diminished quickly [10].

2.2 Disease Follow-up

In light of the current state of Parkinson's disease management, monitoring technologies for PD patients have the potential to improve the quality of life (QoL) through continuous, objective self-monitoring. These technologies also offer crucial data that might be shared with medical professionals to help them better manage patient care [11].

For this purpose, there are various mobile applications in marketplaces (Google Play, Apple App Store, and Windows Store) related to PD [11].

Apps are divided into the following categories based on their intended use:

- 1. PD-related apps: applications that aren't expressly made for PD but could be helpful in controlling the condition.
- 2. PD-specific applications, containing three subcategories:
  - Informational applications: apps that enlighten users about the illness and are aimed at medical professionals, patients, families, or caregivers.
  - Applications for assessment: These applications include numerous tests for evaluating people with Parkinson's disease, analyzing things like speech, balance, tremors, gait, and upper-limb coordination, among other things.
  - Apps for treatment: These apps offer patients and medical practitioners a set of instructions for pharmacologically treating Parkinson's disease

(PD) or for neurorehabilitation, such as physiotherapy, cognitive therapy, and speech therapy [11].

Additionally, the presence of PD and its severity have been determined according to Data Science. According to its definition, data science is the examination of enormous amounts of massive and raw data that have the ability to produce insights that aid businesses in developing through strategic decision-making. Technology-wise, data science is transforming how many sectors operate, how they use their data, and how they tackle their issues. Data science is assisting firms in comprehending their surroundings, analyzing their current problems, and surfacing previously undiscovered opportunities [12].

The amount of data that is being generated at this rapid rate comes from a variety of sources, including log files, emails, social media, sales data, patient information data, sports performance data, sensor data, security alarms, and many more. Data can be found in both organized and unstructured forms, and both types of data can reveal a variety of patterns that can be used to generate solutions. These patterns can also be shown effectively to help comprehend the nature of the results and suggest subsequent steps [12].

Machine learning can be seen as a subset of artificial intelligence that analyzes data and makes wise choices using computer algorithms based on its discoveries without having to be explicitly programmed. Machine learning is what gives computers the ability to solve issues on their own and generate precise predictions utilizing the available data [12].

2.3 Finger Tapping Test

The finger tapping (FT) test is a simple and efficient exercise that is commonly employed in clinical practice to assess motor function, particularly for screening purposes. The FT test is crucial for diagnosing Parkinson's disease (PD). Neurologists in clinical practice assess hypokinesia, fatigue, and particularly slowness by using a five-point scale ranging from 0 to 4, as outlined in the Movement Disorder Society-sponsored revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS). A score of 0 indicates normal functioning, while a score of 4 indicates a severe symptom. Additional factors, such as age and medication status, may have a considerable impact on the mobility of FT. For instance, individuals who are in good health may experience motor dysfunction as a result of the natural aging process. In such cases, Levodopa can be employed as an effective treatment for patients with Parkinson's disease. Consequently, individuals in good health may receive a rating of 1 or higher due to the effects of aging in the FT test, whereas patients with Parkinson's disease may have a rating of 0 in the ON state. The diagnostic accuracy of Parkinson's disease in the population is modest (75 -

85%) due to the similarities between healthy individuals experiencing aging and patients with Parkinson’s disease. Specifically, the FT score of those with moderate Parkinson’s disease may be comparable to that of individuals without the disease. Therefore, it is difficult to identify people with moderate Parkinson’s disease using the FT task. Furthermore, it is crucial to identify patients with mild Parkinson’s disease (PD) as early identification and therapy can effectively mitigate the risk of dyskinesia and significantly enhance their quality of life[13].

Chapter 3

Data Set and Feature Extraction

3.1 Resources and Techniques

The finger-tapping test (FTT) is one of the most often conducted assessments of bradykinesia, as stated in the published guidelines provided by the Movement Disorder Society. The information associated with movement patterns was provided in the vision-based 3D Parkinson's disease (PD) hand dataset. It entails tapping the tips of the thumb and index finger as quickly and forcefully as you can for a predetermined amount of seconds or repetitions. The details of the experimental session, participants, and materials used to capture the FFT were taken from a different study [1].

3.1.1 System for the acquisition and processing of data

The GMH-D algorithm serves as the foundation for the finger-tapping acquisition block, which tracks hand joints during tasks involving the hands in clinical assessments. The combination of Google Mediapipe Hands' marker-less tracking and the depth estimation from the Azure Kinect camera enables accurate and impartial tracking of the trajectories of hand joints even in extremely dynamic tasks like finger-tapping. This characteristic guarantees excellent accuracy and stability when estimating features associated with finger motion. Furthermore, through simulated trials conducted by a group of healthy subjects, the authors demonstrated that the algorithm can be used to extract features to characterize FT at different speeds and with altered amplitude, producing good results for automatic classification [1].

The 10-second frontal recordings of FT tasks are obtained using a customized GMH-D implementation created in Unity® (Unity Technologies, San Francisco, CA, USA). The mini PC is running Windows 10 and has a 9th generation Intel® CoreTM processor (2.4 GHz quad-core), 16 GB RAM, NVIDIA GeForce RTX 2060 6GB GDDR6, HDMI, and USB3 ports. The acquisition software processes video recordings in real-time at a rate of 30 frames per second, resulting in an output JSON file that includes the trajectories of 21 virtual hand joints. The second block involves offline data processing in order to: (i) segment the distance between the Thumb-Tip (TT) and Index-Finger-Tip (IFT) joints in order to identify individual FT movements; (ii) extract a series of features describing the characteristics of the mean FT movements and their regularity throughout the task using spectral properties and the coefficient of variation; (iii) choose the best feature set (Fopt) to differentiate the PD subjects from the HC group. In the final block of the system, well-known shallow learning models are used to classify data based on ideal features. To lessen the potential impact of model overfitting during the testing stage, the Leave-One-Subject-Out (LOSO) procedure is used for the comparison and evaluation of these models [1].

The thumb and index finger coordinates have been examined at various time stamps and in the x, y, and z axes. The entirety of this study has been conducted using Python and its associated libraries. Figure 3.1 shows, that joints number 4 and 8 were analyzed that point to thumb and index fingers, respectively.

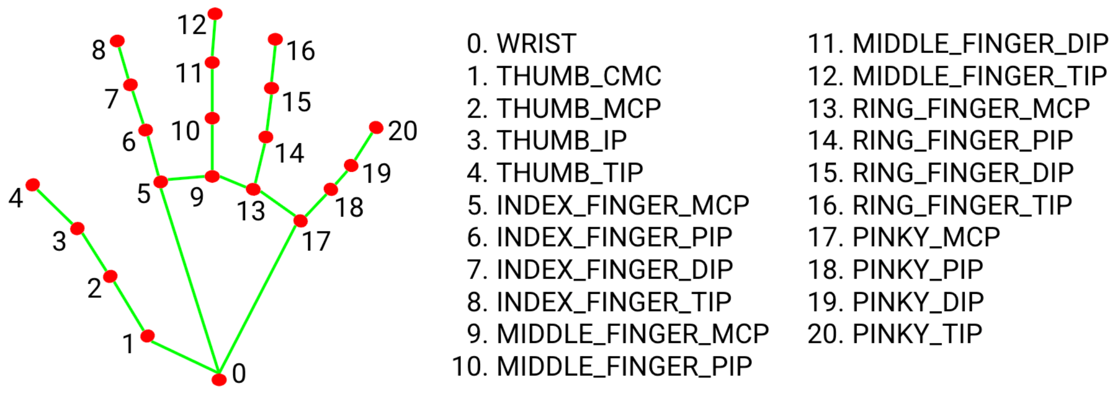

Figure 3.1: Hand Joints Position

3.1.2 Participants and Experimental Session

A controlled experiment was conducted to evaluate the effectiveness of the proposed system in objectively assessing FT and its ability to assist in the diagnosis and monitoring of PD. The experimental phase encompassed a collective of 95 participants, comprising individuals with Parkinson's disease (PD) as well as healthy controls (HC). 35 individuals diagnosed with Parkinson's disease, consisting of 15 females with an average age of 65.1 ± 9.2 years, were recruited from the "Associazione Amici Parkinsoniani Piemonte ONLUS" in Turin, Italy, to participate in the data collection. Both the recruitment and the experimental sessions took place at the Association's Offices. Therefore, it was not feasible to obtain clinical information regarding the stage of the disease or the progression of symptoms. The HC group consisted of 60 individuals (27 females) who were caregivers for PD subjects and association personnel. The average age of the group was 53.8 ± 7.9 years. None of the individuals had a history of neurological or cognitive disorders. The exclusion criteria for both groups encompassed individuals with dementia or any psychiatric disorders that would hinder their ability to complete the task effectively [1].

The procedure was carried out following the guidelines of the Declaration of Helsinki and received approval from the Ethics Committee of A.O.U. Citta della Salute e della Scienza di Torino (Approval No. 00384/2020). Each participant was given comprehensive information regarding the purpose and implementation of the study, and they all provided written informed consent for the observational study [1].

The experimental session was conducted as part of a comprehensive observational study, involving a large number of items and tasks to evaluate symptoms from various perspectives. Consequently, each participant in the study carried out only one acquisition. The control group exclusively completed the task using their dominant hand. In contrast, the PD subjects, who may experience varying levels of impairment in their left and right upper limbs due to Parkinson's disease, performed the task twice. Specifically, they completed the task once with their right hand and once with their left hand. Furthermore, as the clinical scoring evaluates each limb separately, the left and right FT executions were treated as individual data points in the subsequent analysis [1].

3.2 Python Libraries

Python is a programming language that is designed for universal use and operates at a high level of abstraction. The design concept of this approach prioritizes code readability by utilizing considerable indentation [14].

Python is a programming language that supports multiple programming paradigms. The programming languages completely support both object-oriented programming and structured programming, with many of their capabilities also accommodating functional programming and aspect-oriented programming. It provides a collection of pre-existing mathematical functions, including a comprehensive math module, which enables you to carry out many mathematical operations on numbers. Libraries such as NumPy, Pandas, and Matplotlib allow the effective use of Python in scientific computing. Python is frequently employed in artificial intelligence and machine learning endeavors, leveraging packages such as scikit-learn.

3.3 Distance and Velocity Signals

In order to conduct a quantitative analysis of FTT movement, the distance between the tips of the thumb and index finger, as well as the related velocity, are calculated as the fundamental signals, similar to previous studies [3].

The term "distance" refers to the quantitative or, on occasion, qualitative evaluation of the degree to which two things or points are separated from one another.

The velocity of an object can be used to determine both the speed at which it is moving and the direction in which it is moving. The idea of velocity is essential to the study of kinematics, which is the subfield of classical mechanics that focuses on the movement of bodies. The formula for calculating average velocity is total displacement divided by total time.

As shown in figure 3.1, 4 and 8 joint positions have been considered, in this study. For computing distance, the interval between two points 4 and 8 was measured, and to calculate velocity, the change in distance was divided by the change in time. Figure 3.2 shows 3D Distance and Velocity between tips of thumb and index fingers for Healthy and PD cases.

3.4 Feature Extraction by tsfresh Library

Furthermore, to conduct a more comprehensive analysis of this approach, kinematic characteristics such as amplitude, velocity, and rhythm have been retrieved. Tsfresh employs 3D distance and velocity data to compute various time series characteristics, transforming these kinematic parameters into quantitative representations. Tsfresh is a renowned and efficient machine learning tool designed for extracting time series features [3]. The input data of tsfresh library consisted of time series with 133 dimensions. Tsfresh provides 63-time series characterization techniques and is capable of computing 794-time series characteristics[4]. By applying this library, 748 features were produced.

Systematic feature engineering from time series and other sequential data is accomplished with tsfresh [4]. These data have the characteristic of being arranged according to an independent variable. The predominant independent variable is typically time, sometimes referred to as a time series. Additional instances of sequential data include reflectance and absorption spectra, which are organized based on their wavelength dimension. To maintain simplicity, all various forms of sequential data have been categorized as time series. Tsfresh streamlines the process of extracting features by automatically calculating and returning all of

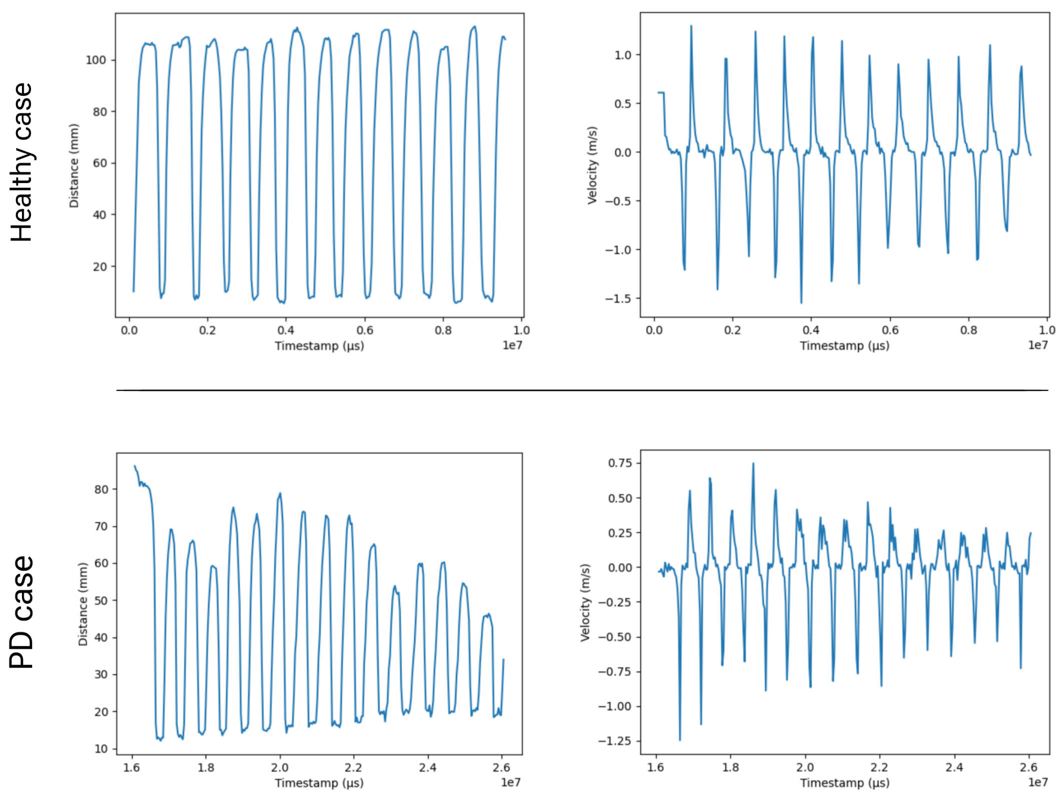

Figure 3.2: 3D Distance and Velocity between tips of thumb and index fingers for Healthy and PD cases

those characteristics. Furthermore, tsfresh is fully compatible with the Python packages pandas and scikit-learn. The retrieved characteristics can be utilized to characterize the time series, hence providing novel insights into the time series and their dynamics. Additionally, time series clustering and training of machine learning models for time series can be accomplished using them.

To use the tsfresh package, the following module has been imported into the code. This module contains the feature calculators that take time series as input and calculate the values of the feature.

```
tsfresh.feature_extraction.feature_calculators
```

The table 3.1 encompasses all the feature computations that are compatible with the present iteration of tsfresh [15]:

abs_energy(x)	Returns the absolute energy of the time series which is the sum over the squared values
absolute_maximum(x)	Calculates the highest absolute value of the time series x.
absolute_sum_of_changes(x)	Returns the sum over the absolute value of consecutive changes in the series x
agg_autocorrelation(x, param)	Descriptive statistics on the autocorrelation of the time series.
agg_linear_trend(x, param)	Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.
approximate_entropy(x, m, r)	Implements a vectorized Approximate entropy algorithm.
ar_coefficient(x, param)	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR(k) process.
augmented_dickey_fuller(x, param)	Does the time series have a unit root?
autocorrelation(x, lag)	Calculates the autocorrelation of the specified lag, according to the formula
benford_correlation(x)	Useful for anomaly detection applications. Returns the correlation from first digit distribution when
binned_entropy(x, max_bins)	First bins the values of x into max_bins equidistant bins.
c3(x, lag)	Uses c3 statistics to measure non linearity in the time series
change_quantiles(x, ql, qh, isabs, f_agg)	First fixes a corridor given by the quantiles ql and qh of the distribution of x.
cid_ce(x, normalize)	This function calculator is an estimate for a time series complexity (A more complex time series has more peaks, valleys etc.).
count_above(x, t)	Returns the percentage of values in x that are higher than t
count_above_mean(x)	Returns the number of values in x that are higher than the mean of x

count_below(x, t)	Returns the percentage of values in x that are lower than t
count_below_mean(x)	Returns the number of values in x that are lower than the mean of x
cwt_coefficients(x, param)	Calculates a Continuous wavelet transform for the Ricker wavelet, also known as the "Mexican hat wavelet" which is defined by
energy_ratio_by_chunks(x, param)	Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series.
fft_aggregated(x, param)	Returns the spectral centroid (mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.
fft_coefficient(x, param)	Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm
first_location_of_maximum(x)	Returns the first location of the maximum value of x.
first_location_of_minimum(x)	Returns the first location of the minimal value of x.
fourier_entropy(x, bins)	Calculate the binned entropy of the power spectral density of the time series (using the welch method).
friedrich_coefficients(x, param)	Coefficients of polynomial h(x), which has been fitted to the deterministic dynamics of Langevin model
has_duplicate(x)	Checks if any value in x occurs more than once
has_duplicate_max(x)	Checks if the maximum value of x is observed more than once
has_duplicate_min(x)	Checks if the minimal value of x is observed more than once
index_mass_quantile(x, param)	Calculates the relative index i of time series x where q% of the mass of x lies left of i.

kurtosis(x)	Returns the kurtosis of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).
large_standard_deviation(x, r)	Does time series have large standard deviation?
last_location_of_maximum(x)	Returns the relative last location of the maximum value of x.
last_location_of_minimum(x)	Returns the last location of the minimal value of x.
lempel_ziv_complexity(x, bins)	Calculate a complexity estimate based on the Lempel-Ziv compression algorithm.
length(x)	Returns the length of x
linear_trend(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.
linear_trend_timewise(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.
longest_strike_above_mean(x)	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x
longest_strike_below_mean(x)	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x
matrix_profile(x, param)	Calculates the 1-D Matrix Profile[1] and returns Tukey's Five Number Set plus the mean of that Matrix Profile.
max_langevin_fixed_point(x, r, m)	Largest fixed point of dynamics :math:argmax_x h(x)=0‘ estimated from polynomial h(x), which has been fitted to the deterministic dynamics of Langevin model
maximum(x)	Calculates the highest value of the time series x.
mean(x)	Returns the mean of x
mean_abs_change(x)	mean_abs_change(x)
mean_change(x)	Average over time series differences.

mean_n_absolute_max(x, number_of_maxima)	Calculates the arithmetic mean of the n absolute maximum values of the time series.
mean_second_derivative_ central(x)	Returns the mean value of a central approximation of the second derivative
median(x)	Returns the median of x
minimum(x)	Calculates the lowest value of the time series x.
number_crossing_m(x, m)	Calculates the number of crossings of x on m.
number_cwt_peaks(x, n)	Number of different peaks in x.
number_peaks(x, n)	Calculates the number of peaks of at least support n in the time series x.
partial_autocorrelation(x, param)	Calculates the value of the partial autocorrelation function at the given lag.
percentage_of_reoccurring_ datapoints_ to_all_datapoints(x)	Returns the percentage of non-unique data points.
percentage_of_reoccurring_ values_to_ all_values(x)	Returns the percentage of values that are present in the time series more than once.
permutation_entropy(x, tau, dimension)	Calculate the permutation entropy.
quantile(x, q)	Calculates the q quantile of x.
query_similarity_count(x, param)	This feature calculator accepts an input query subsequence parameter, compares the query (under z-normalized Euclidean distance) to all subsequences within the time series, and returns a count of the number of times the query was found in the time series (within some predefined maximum distance threshold).
range_count(x, min, max)	Count observed values within the interval [min, max).
ratio_beyond_r_sigma(x, r)	Ratio of values that are more than r * std(x) (so r times sigma) away from the mean of x.
ratio_value_number_to_ time_series_length(x)	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case.

root_mean_square(x)	Returns the root mean square (rms) of the time series.
sample_entropy(x)	Calculate and return sample entropy of x.
set_property(key, value)	This method returns a decorator that sets the property key of the function to value
skewness(x)	Returns the sample skewness of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).
spkt_welch_density(x, param)	This feature calculator estimates the cross power spectral density of the time series x at different frequencies.
standard_deviation(x)	Returns the standard deviation of x
sum_of_reoccurring_data_ points(x)	Returns the sum of all data points, that are present in the time series more than once.
sum_of_reoccurring_values(x)	Returns the sum of all values, that are present in the time series more than once.
sum_values(x)	Calculates the sum over the time series values
symmetry_looking(x, param)	Boolean variable denoting if the distribution of x looks symmetric.
time_reversal_asymmetry_ statistic(x, lag)	Returns the time reversal asymmetry statistic.
value_count(x, value)	Count occurrences of value in time series x.
variance(x)	Returns the variance of x
variance_larger_than_standard _deviation(x)	Is variance higher than the standard deviation?
variation_coefficient(x)	Returns the variation coefficient (standard error / mean, give the relative value of variation around the mean) of x.

Table 3.1: List of all the feature calculations supported by tsfresh

Chapter 4

Feature Selection

4.1 Overview on Feature Selection

Feature selection is the process of choosing the most impactful features from a set of features to decrease the dimensionality of the feature space [5].

The input variables provided to the machine learning models are referred to as features. Every column in the dataset represented a distinct feature. In order to train an ideal model, it is critical to exclusively utilize the fundamental features. Excessive inclusion of features in the model can result in the collection of insignificant patterns and learning from irrelevant data. The process of selecting the significant parameters of the data is referred to as Feature Selection.

Machine learning models adhere to a fundamental principle: the output is determined by the input. When inputs of the model have low-quality or irrelevant data, poor quality or lack of usefulness of output results can be anticipated. Here, "garbage" denotes extraneous or irrelevant information within the dataset. In order to train a model, vast amounts of data have been gathered to enhance the machine's learning capabilities. Typically, a substantial amount of the data acquired is irrelevant information, and certain columns in our dataset may not have a major impact on the performance of our model. Moreover, an abundance of data can impede the training process and result in decreased model speed. The model may potentially acquire knowledge from this extraneous data and hence yield erroneous results. There are two types of feature selection models:

- Supervised Models: Supervised feature selection is an approach that uses the output label class to pick features. The target variables are utilized to find the variables that have the potential to enhance the effectiveness of the model.
- Unsupervised Models: Unsupervised feature selection is a method that does not require the output label class to choose features. Unlabelled data is utilized for this purpose.

Feature selection is a crucial challenge in various statistical pattern recognition applications, including image processing, speech recognition, text mining, and bio-informatics. It typically involves the application of complex combinatorial mathematics, particularly discrete optimization, to identify the most relevant features. Feature selection has garnered significant attention in the field of pattern recognition and machine learning due to its significance in applications and the obstacles it poses for optimization [6]. In this study, Boruta and PCA feature selectors have been employed.

4.2 Boruta

The Boruta algorithm is a wrapper method that utilizes the random forest classification algorithm as its foundation. The Boruta algorithm employs randomness to introduce variability into the system and gather data from a random sample set. The random forest classification algorithm is efficient, typically requiring minimal parameter adjustment, and provides a quantitative measure of feature value. It is a technique that combines numerous unbiased weak classifiers, specifically decision trees, to achieve classification through a voting process. These trees are autonomously created using various bagging samples from the training set. The significance measure of an attribute is determined by the decrease in classification performance resulting from randomly shuffling attribute values among objects. The computation is performed individually for each tree in the forest that uses a specific property for categorization. Next, the mean and standard deviation of the decrease in accuracy are calculated. Alternatively, the significance measure can be determined by calculating the Z score, which is obtained by dividing the average loss by its standard deviation. Regrettably, the Z score does not have a clear correlation with the statistical significance of the feature importance obtained from the random forest algorithm, as its distribution does not follow a normal distribution with a mean of 0 and a standard deviation of 1. However, in Boruta, the Z score has been used as the measure of relevance because it considers the variations in the average accuracy loss among the trees in the forest.

As the direct use of the Z score to assess importance is not possible, an external reference is used to determine if the importance of a specific characteristic is noteworthy. It is advantageous to ascertain if the importance is distinguishable from random fluctuations. In order to achieve this objective, the information system has been expanded by including deliberately randomized attributes. For every attribute, a corresponding "shadow" attribute is generated by rearranging the values of the original attribute among different objects. Subsequently, a classification utilizing all features of this expanded system is conducted and the significance of each attribute is calculated. The significance of a shadow property can only be non-zero as a result of random fluctuations. Therefore, the collection of significance levels of shadow qualities serves as a benchmark for determining the attributes that are genuinely significant. The significance measure fluctuates as a result of the inherent randomness in the random forest classifier. Furthermore, it is responsive to the existence of non-essential features in the information system, including those that are considered secondary or peripheral. Furthermore, it relies on the specific manifestation of shadow characteristics. Hence, it is necessary to iterate the re-shuffling process in order to produce statistically reliable outcomes.

Essentially, Boruta operates on the same principle as the random forest classifier. It involves introducing randomness into the system and gathering outcomes from a collection of randomized samples. This approach is useful to mitigate the distorting effects of random variations and correlations. Here, this additional randomization results in a more distinct understanding of which traits are truly significant.

This approach leads to reducing the distorting effects of random fluctuations and correlations. Through iterative processes, the algorithm identifies and eliminates unimportant features by combining the interactions among different features. Ultimately, it determines the genuinely significant features. The precise sequence of actions in the Boruta algorithm is as follows:

- The source feature matrix T is created by selecting N vegetation indices. The shadow feature matrix S is formed by shuffling T. A new feature matrix, denoted as M, is created by merging matrices T and S.

- The input variable, the new feature matrix M, is used to calculate the Z-value for each variable. The Z-value is obtained by dividing the average loss by the standard deviation. This process allows the feature relevance to be determined.

- The highest Z value in the shadow feature denoted as Zmax, is determined and subsequently compared to the Z value of each variable. If the Z value of a variable exceeds Zmax, it is designated as significant, while if it is considerably lower than Zmax, it is designated as insignificant.

- Irreversibly eliminates insignificant attributes from the collection of features.

- Continue iterating through steps (1)-(4) until all variables have been tagged [16].

In practical scenarios, the temporal complexity of the aforementioned technique is around $O(P \cdot N)$, where P represents the number of characteristics and N represents the number of objects. Although it may need a significant amount of time for large data sets, this endeavor is crucial in order to generate a statistically meaningful assortment of pertinent characteristics [17].

In this study, after data preparation, the Random Forest classifier was instantiated with specific configuration parameters including n_jobs, class_weight, max_depth, and max_features. The n_jobs was set to -1 to enable the utilization of all available processors for parallel processing, resulting in a faster training process. The parameter class_weight='balanced' was used to account for any imbalances in the classes within the dataset. It ensured that each class had an equal influence during the training process. The parameter max_depth was set to 7 to restrict the maximum depth of the decision trees in the Random Forest algorithm. This was done to manage the complexity of the model and prevent overfitting and based on the Boruta library explanation, it has been recommended to set it from 3 to 7. The max_features=None, allowed the algorithm to use all available features. The Boruta feature selector, named boruta_selector, was initialized using a preconfigured Random Forest classifier. BorutaPy is a Python implementation of the Boruta algorithm, which is based on tree-based classifications, specifically random forests. The Boruta feature selector was employed on the dataset, using the feature set (X) and target variable (y). This step evaluated the significance of each feature according to the criteria of the Boruta algorithm. The boruta_selector.support_ attribute was utilized to determine features that are considered essential by Boruta. The notable characteristics were subsequently aggregated into a collection of string elements known as selected_features. In order to finalize the set, an additional attribute, referred to as the target column, is added to the list of chosen attributes. The function returned a tuple, which was essentially a list that included the selected attributes and the total count of those attributes. The parameter max_iter was established to 100 as its default mode to specify the upper limit on the number of iterations that the Boruta algorithm executes. If convergence was reached, it terminated prematurely. The parameter n_estimators='auto' automatically determined the number of trees in the Random Forest based on the size of the dataset. In addition, verbose=0 regulated the level of verbosity during the Boruta feature selection procedure, where a value of 0 indicates that no output is displayed. In the following, the selected features by Boruta are shown in figures 4.1, 4.2, and 4.3 for distance, velocity, and combination of distance and velocity, respectively:

Boruta for Distance
Distance___has_duplicate
Distance___length
Distance___percentage_of_reoccurring_datapoints_to_all_datapoints
Distance___fft_coefficient___attr_"real"___coeff_45
Distance___fft_coefficient___attr_"real"___coeff_49
Distance___fft_coefficient___attr_"abs"___coeff_50
Distance___fft_coefficient___attr_"angle"___coeff_87
Distance___range_count___max_1000000000000.0___min_0
is_pd

Table 4.1: Selected Features by Boruta for Distance

Boruta for Velocity
Velocity___length
Velocity___percentage_of_reoccurring_values_to_all_values
Velocity___percentage_of_reoccurring_datapoints_to_all_datapoints
Velocity___fft_coefficient___attr_"real"___coeff_89
Velocity___fft_coefficient___attr_"imag"___coeff_17
Velocity___fft_coefficient___attr_"abs"___coeff_17
Velocity___fft_coefficient___attr_"angle"___coeff_87
Velocity___range_count___max_1___min_-1
Velocity___agg_linear_trend___attr_"rvalue"___chunk_len_50___f_agg_"max"
is_pd

Table 4.2: Selected features by Boruta for Velocity

Boruta for Distance and Velocity Combination
Distance__length
Distance__sum_of_reoccurring_values
Distance__fft_coefficient__attr_"abs"__coeff_17
Distance__range_count__max_1000000000000.0__min_0
Velocity__length
Velocity__percentage_of_reoccurring_values_to_all_values
Velocity__percentage_of_reoccurring_datapoints_to_all_datapoints
Velocity__fft_coefficient__attr_"imag"__coeff_17
Velocity__fft_coefficient__attr_"angle"__coeff_87
Velocity__range_count__max_1__min_-1
is_pd

Table 4.3: Selected Features by Boruta for Distance and Velocity Combination

4.3 Principle Component Analysis (PCA)

Dimension reduction is a widely used operation in the mainstream of image processing. The Principle Component Analysis (PCA) model, which is a highly popular method for reducing dimensions, has found extensive application in image processing. This includes tasks such as image reconstruction, image denoising, image recognition, image fusion, and subspace learning [18].

Principal Component Analysis (PCA) is a widely used unsupervised learning method that enhances interpretability while simultaneously reducing information loss. It facilitates the identification of the most prominent characteristics in a dataset and simplifies the process of visualizing the data in both two and three dimensions. Principal Component Analysis (PCA) facilitates the identification of a series of linear combinations of variables.

This method employs dimensional reduction on the input dataset by reducing the principal components that have less impact on its variance while preserving the aspects of the dataset that have the greatest influence. Dimensionality refers to the number of features or variables employed in the investigation. While these traits may hold significant importance, their relevance may vary based on the specific application. PCA is a technique used to detect patterns in data and represent the data in a manner that emphasizes their similarities and differences. Due to the complexity of high-dimensional data and the absence of graphical representation, it is challenging to identify patterns in the data. PCA is a potent technique used to analyze data. It achieves dimensionality reduction by projecting the original data onto a lower-dimensional space defined by the K dominant eigenvectors of the data's covariance matrix [19].

Correlation is a statistical term that quantifies the direction and intensity of the linear relationship between two variables. In the context of Principal Component Analysis (PCA), the covariance matrix is computed to represent the pairwise correlations between all variables in the dataset. This matrix is square. The primary elements of the data are computed utilizing the eigenvectors. The eigenvectors of the covariance matrix represent the primary modes of variation in the data.

Indeed, high-dimensional data, such as datasets with many variables, can make it hard to see and analyze variable relationships. PCA and other dimensionality reduction approaches keep the most important data while lowering the number of variables. PCA transforms the original variables into a set of new variables known as principal components. These components are formed through linear combinations of the original variables. Each principle component is a linear combination of the original variables. The first principal component represents the highest amount of variation in the data, followed by the second principal component which captures the second highest amount of variation, and so on. The dimensionality of the dataset is determined by the number of main components utilized in the investigation. The goal of PCA is to identify a reduced set of main components that capture the most significant variation in the data.

Principal Component Analysis (PCA) is a method of linear transformation that detects the directions in which the data exhibits the most amount of variation. It then projects the data onto these directions, known as principal components. Every principal component is a linear combination of the original variables, with the first principal component capturing the highest amount of variance in the data, followed by the second principal component capturing the second highest amount, and so forth. The graph 4.1 displays two distinct main components, PC1 and PC2, which are not dependent on each other. It should be noted that PC1 corresponds to the eigenvector that accounts for the majority of the variance in the information. PC2 represents a smaller amount of information (variance).

Explained variance is a statistical metric that quantifies the amount of variation in a dataset that can be ascribed to each of the main components (eigenvectors) produced by the principal component analysis (PCA) technique. The proportion of the overall variance that is "accounted for" by each component has been informed by PCA. It is crucial because it enables users to prioritize the components based on their significance and concentrate on the most significant ones when evaluating the analysis outcomes.

The concept of explained variance is valuable for evaluating the significance of each component. Typically, the more the variance accounted for by a major component, the more significant that component becomes. The measure of explained variance can be utilized to determine the optimal number of dimensions to retain in a reduced dataset. Additionally, it can be employed to evaluate the efficacy of

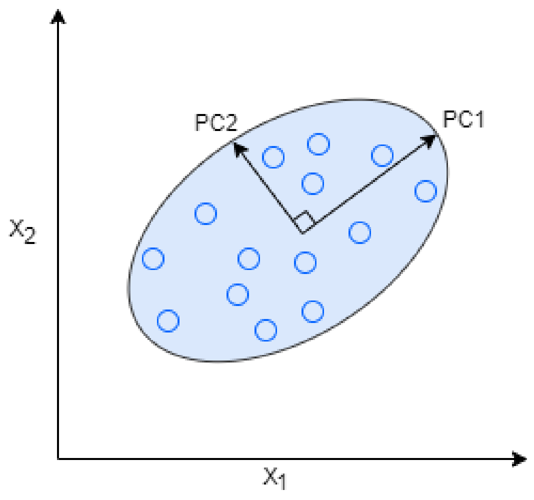

Figure 4.1: Two distinct main components of PCA

the machine learning model. Typically, a model that has principal components with a high degree of explained variation will have strong predictive capability, whereas a model with principal components that have a low degree of explained variance may not be as precise.

The concept of explained variance can be expressed mathematically by calculating the ratio of a specific eigenvalue to the sum of all eigenvalues associated with the eigenvectors. If there are N eigenvectors, then the explained variance for each eigenvector (principal component) can be calculated as the ratio of the eigenvalue λ_i of the corresponding eigenvector and the sum of all eigenvalues ($\lambda_1 + \lambda_2 + ... + \lambda_n$), stated as follows:

$$\frac{\lambda_i}{\lambda_1 + \lambda_2 + ... + \lambda_n} \tag{4.1}$$

It is important to remember that while performing eigen decomposition on a transformation matrix, which is the covariance matrix in the case of principal component analysis (PCA), a collection of eigenvectors and their corresponding eigenvalues are obtained. The eigenvectors correspond to the principal components that capture the majority of the information (variance) conveyed by the features (independent variables). The explained variance ratio quantifies the amount of variance that can be accounted for by a specific eigenvector.

SciKit-learn offers a useful tool called Pipeline that allows for the sequential connection of transformers and a final classifier, which has been used in this thesis. The pipeline consists of two primary components: Principal Component Analysis (PCA) for reducing dimensionality and a Decision Tree Classifier for performing classification. A decision tree is a machine learning algorithm that utilizes feature values to divide the data into branches and make decisions. Here, the decision tree was employed for the purpose of classification. The Pipeline was created by iterating through a list of tuples. Every tuple consisted of a name (a sequence of characters) and an estimator (an object representing a machine-learning model or transformer). In this instance, it was explicitly stated that the initial step was Principal Component Analysis (PCA) referred to as 'pca', and the subsequent step was a Decision Tree Classifier denoted as 'tree'.

In addition, a graphical method was used to obtain the PCA outcomes, aiding in the determination of the ideal number of principal components to retain by considering the explained variance and a specified threshold. After data processing, the PCA was initialized and fitted on the standardized data. Moreover, the cumulative explained variance ratio from the PCA results and the first derivative of the explained variance ratio were computed. Subsequently, the number of selected features based on the first derivative and a specified threshold (20% of the inter-quartile range) was determined as shown as 3 bar plots in figures 4.2, 4.3, and 4.4.

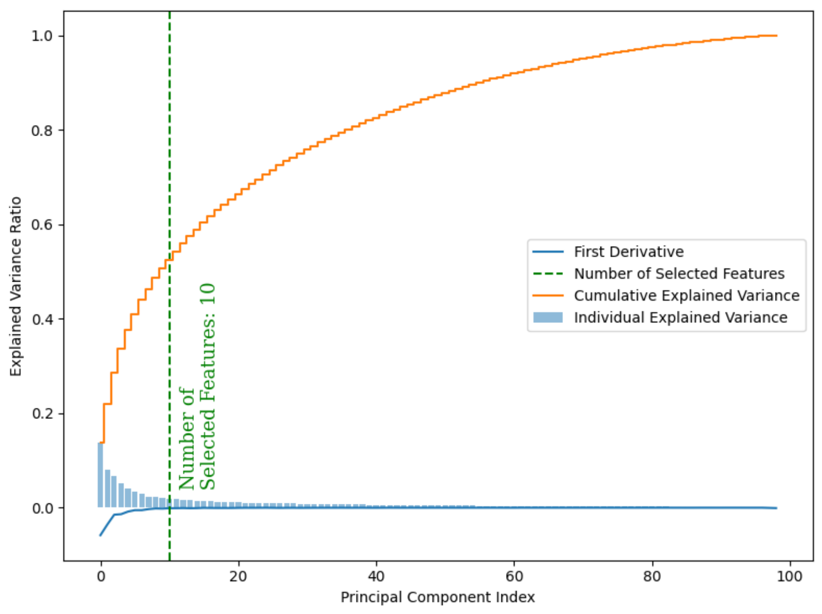

Figure 4.2: Comulative Explained Variance Plot of PCA for Distance

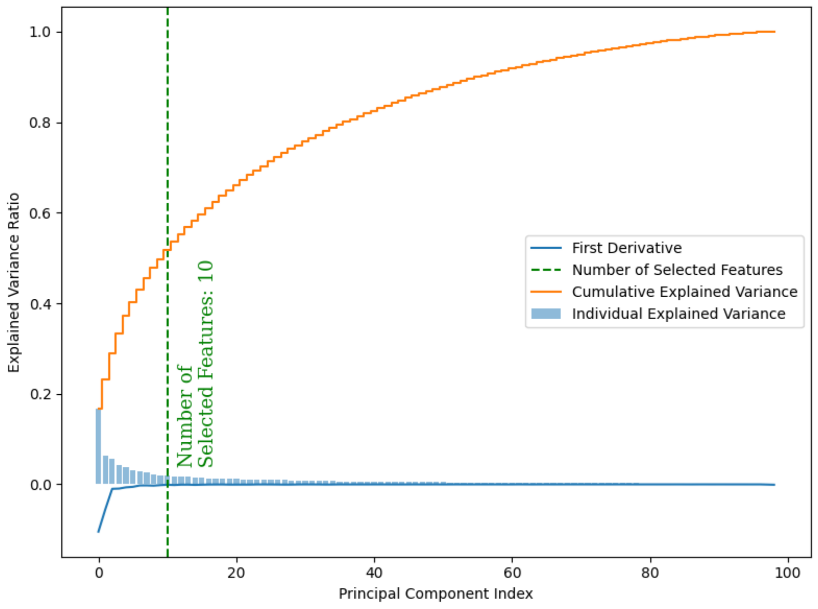

Figure 4.3: Comulative Explained Variance Plot of PCA for Velocity

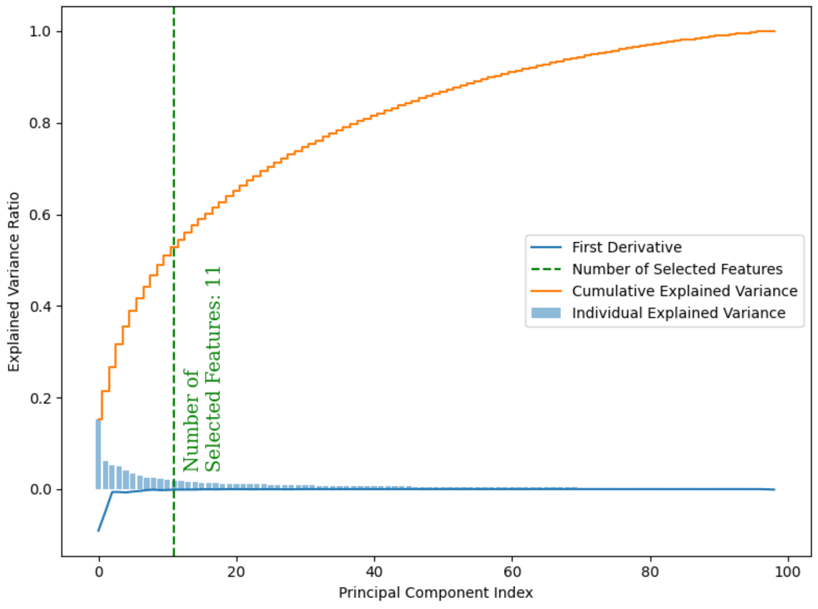

Figure 4.4: Comulative Explained Variance Plot of PCA for Distance and Velocity Combination

4.4 Scatter Plot

The progress in information technology has produced a surplus of data that typically exists in tabular formats, such as database tables and spreadsheets. Multiple visualization techniques exist for representing multivariate or multidimensional data. Scatter plots are a diverse, polymorphic, and generally beneficial approach. A scatter plot is a graphical representation that can display two or three variables on a two-dimensional or three-dimensional coordinate system. Visual characteristics are commonly employed to depict supplementary values. Scatter plot visualizations are highly valuable during the initial phases of analysis. They can effectively demonstrate correlations and patterns in data with few dimensions, while also offering a concise overview of a substantial amount of data. Landscape visualizations are less effective than other methods for visual search and visual memory, particularly when examining the relationship between two variables. Scatter plots have been thoroughly examined to reduce both non-dimensional and dimensional data. However, the efficacy of these components declines as the number of dimensions and data points grows. Furthermore, scatter plots are not effective in representing data sets with a large number of dimensions due to the restricted mapping dimensions. The excessive charting and overlapping of data points can impede the accuracy of the derived information [20].

Scatter Plots of the first three principal components of PCA and the first three features of Boruta were drawn to show outliers and categorize PD and HC. Outliers, conversely, are data points that exhibit substantial deviation from the remaining data. One can ascertain their identity based on their positioning in relation to the majority of data points in a scatter plot.

Figures 4.5, 4.6, and 4.7 display scatter plots of distance, velocity, distance, and velocity combination using PCA and figures 4.8, 4.9, and 4.10 display scatter plots of distance, velocity, distance, and velocity combination using boruta.

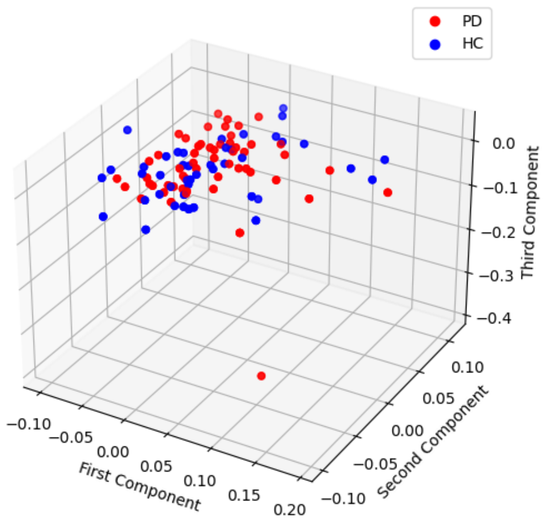

Figure 4.5: Scatter Plot of PCA for Distance

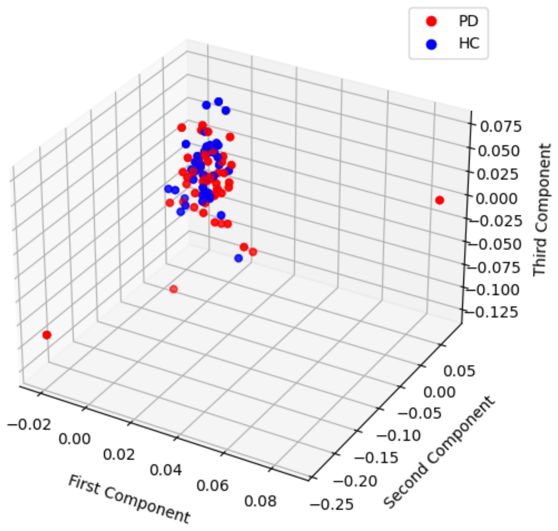

Figure 4.6: Scatter Plot of PCA for Velocity

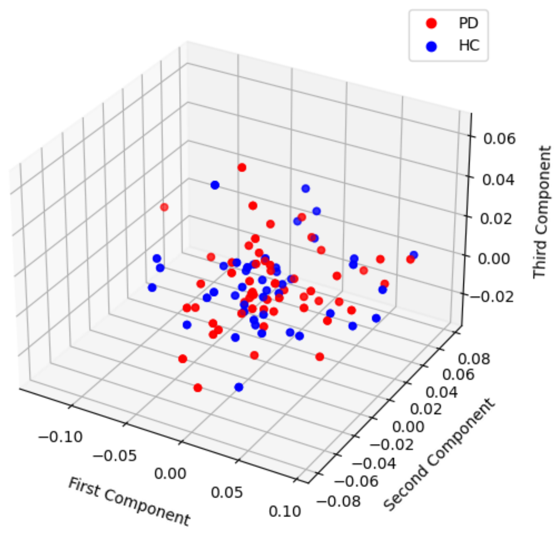

Figure 4.7: Scatter Plot of PCA for Distance and Velocity Combination

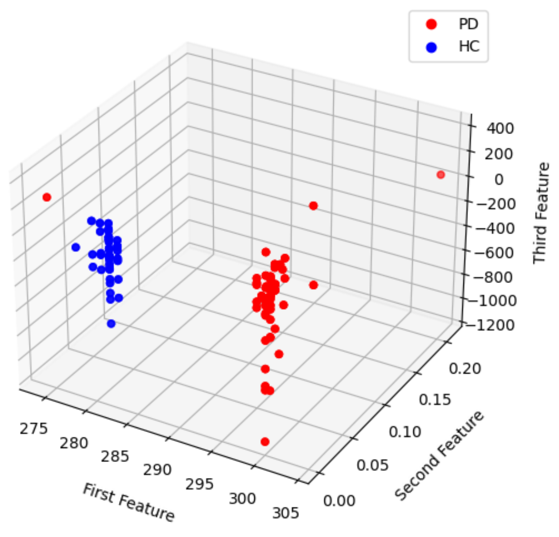

Figure 4.8: Scatter Plot of Boruta for Distance

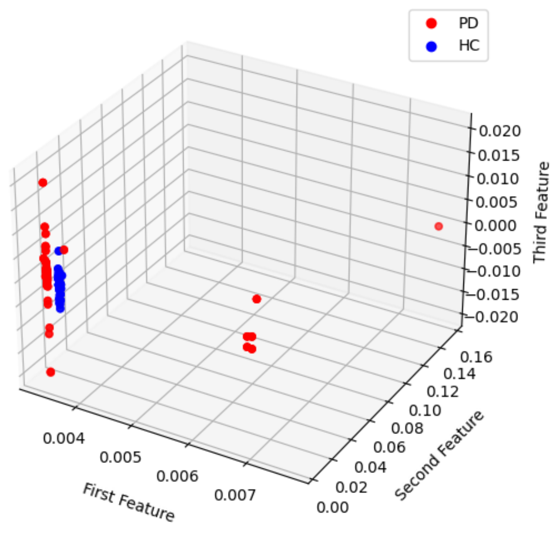

Figure 4.9: Scatter Plot of Boruta for Velocity

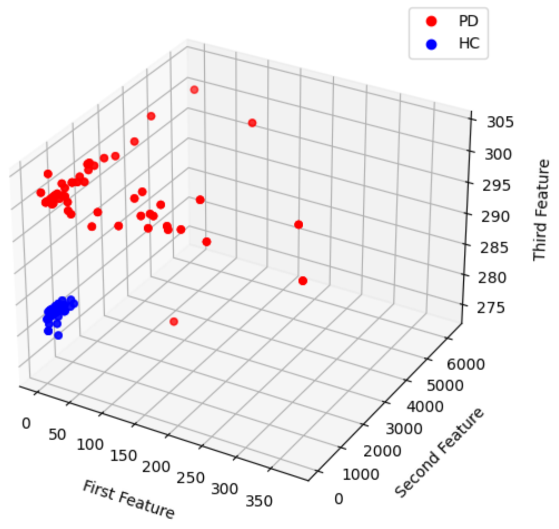

Figure 4.10: Scatter Plot of Boruta for Distance and Velocity Combination

Chapter 5

Classification

5.1 Overview on Machine Learning Classification

Machine learning is the process of training computers to acquire knowledge and mimic human behavior through the input of data. It emphasizes the utilization of information and the replication of human learning processes, systematically striving for precision. Supervised learning involves training machines by utilizing labeled data, which is often referred to as training data, in order to make predictions. "Labeled Data" refers to information that has been assigned one or more names and is already recognized by the computer. Supervised learning finds use in several real-world scenarios such as image and object recognition, predictive analytics, consumer sentiment analysis, spam detection, and numerous more. Supervised learning models are taught by utilizing labeled data, which is often referred to as training data, in order to make predictions. The primary purpose of the model is to identify and classify fresh input data when assessed using a separate input data set that was not utilized during the following training process. The computer is equipped with the ability to identify and distinguish various patterns, shapes, and differences. Moreover, it classifies recently found information by analyzing similarities, patterns, structure, and differences, and predicts the accurate outcome.

Supervised learning addresses a range of computational challenges that arise in real-world scenarios, such as spam detection, object and picture recognition, and numerous others. It leverages historical data to enhance performance and forecast outcomes based on prior experiences. The training data can be reused unless there is any alteration in the features.

Supervised learning can be further classified into two problems which are: Classification and Regression Classification is the systematic procedure of identifying and segregating newly observed data in order to assign them to specific categories.

Machine Learning classification is the process of assigning a label to a given input data set. The supplied data is categorized into several groups based on their label [7]. In this study, the input data was assigned to two labels. "is_pd" has been allocated labels of "1" or "0" here. If an individual had Parkinson's disease, the model predicted that the input data "is_pd" would result in a prediction of "1", else it would predict "0". Moreover, 75% of the data frame has been designated for training, while the remaining 25% has been allocated for testing. Furthermore, the 'random_state' was set to 80 to determine the seed for the random number generator.

Supervised learning models can be utilized to build several commercial applications, some of which are enumerated below:

- Image and object recognition: Supervised learning techniques can be employed to detect, separate, and categorize items from movies or photographs. They become highly valuable when employed in various computer vision methodologies and image analysis.

- Predictive analysis: One extensive application of supervised learning models is in developing predictive analytics systems that provide deep insights into various business data of interest. This enables enterprises to anticipate precise results based on a given outcome characteristic, aiding business leaders in justifying decisions or adapting to serve the organization.

- Customer Sentiment Analysis: By utilizing supervised learning algorithms and associations, it is possible to extract and organize crucial facts from vast amounts of information, including context, emotion, and intent, with minimal human involvement. This can be quite beneficial in gaining a superior comprehension of client interactions and can be employed to enhance brand engagement efforts.

- Spam Detection: Companies can utilize classification techniques to construct data sets that can discern patterns or irregularities in new data, effectively distinguishing between spam and non-spam communications.

Supervised learning is concerned with creating a machine learning model that can establish a relationship between the data and the outcome, allowing it to predict the output for fresh data sources. It is the most precise subset of machine learning. It is the most commonly employed form of machine learning and has shown to be an exceptional tool in numerous industries.

5.2 K Nearest Neighbor (KNN)

KNN, short for K Nearest Neighbor, is a commonly used algorithm in the field of artificial intelligence. It is employed for tasks such as classification and regression.

An exceptional feature of KNN is that it does not explicitly construct a model from the training data, but rather memorizes the training patterns to make predictions on new data. KNN operates by identifying the k nearest data points to a given query point, based on a distance metric such as Euclidean or Manhattan distance. The calculation assigns the value or importance of most of those k nearest neighbors to the query location. KNN is a non-parametric algorithm that does not make any assumptions about the distribution of the data and can do well on small datasets. However, the effectiveness of KNN can be limited by the problem of dimensionality, in which the distance between data points becomes less meaningful as the number of features rises. The KNN algorithm can be employed to impute missing values for both categorical and continuous data. The KNN algorithm can be employed to identify the closest neighbors for continuous variables by utilizing the Euclidean distance or another distance metric. Once the closest neighbors have been identified, the missing value can be filled in by utilizing those neighbors' average or middle values. This approach assumes that the missing value is comparable to the values of its closest neighbors. Therefore, the mean or median of the nearest neighbors can be used to replace the missing value. The KNN algorithm can be employed to identify the k-nearest neighbors of categorical variables. Using a similarity metric, such as Jaccard similarity or cosine similarity, as a basis [21].

The k-nearest neighbors (k-NN) algorithm is a non-parametric, supervised learning classifier that uses closeness to classify or predict the grouping of a given data point. Although it has the capability to handle both regression and classification issues, this technique is predominantly employed for classification tasks. It operates on the underlying concept that points with similar characteristics tend to be located in close proximity to each other. In classification problems, a class label is assigned based on the majority vote, meaning that the label that appears most frequently around a given data point is utilized. Although legally classified as "plurality voting", the term "majority vote" is more frequently employed in the literature. The differentiation lies in the fact that "majority voting" necessitates a majority over 50%, which is most effective when there are solely two categories. When dealing with numerous classes, such as four categories, it is not always necessary to obtain 50% of the votes in order to determine the class. A class label can be assigned if the vote exceeds 25%.

In summary, the objective of the k-nearest neighbor algorithm is to determine the closest neighbors of a specified query location in order to provide a class label to that point. To do this task, KNN has several prerequisites:

- To identify the data points that are nearest to a specific query point, it is necessary to compute the distance between the query point and the other data points. Distance measurements aid in the creation of decision borders, which divide query points into distinct regions.

- The k value in the K-NN method specifies the number of neighbors that will be examined to determine the categorization of a given query point. For instance, when k is equal to 1, the instance will be allocated to the class that its closest neighbor belongs to. Determining the value of k requires careful consideration as selecting different values can result in either overfitting or underfitting. less values of k can exhibit a greater amount of variation, but have a less amount of systematic error, while bigger values of k can result in a greater amount of systematic error and a smaller amount of variation. The selection of k will primarily rely on the characteristics of the input data, as datasets containing a higher number of outliers or noise are expected to yield superior results when using larger values of k.

This study has implemented a method to determine the optimal value of k for the KNN classifier. The approach involved iterating over a range of possible values using a for loop and selecting the prediction with the highest score. Initially, the variable prediction was assigned the value of negative infinity ('float('-info)'). This was done to guarantee that the initial prediction would consistently be regarded as an enhancement. The code subsequently entered a loop wherein it iterated through values of k ranging from 1 to 19, inclusively. The KNN model that has been trained, was subsequently utilized to generate predictions on the test data by employing the 'predict_with_knn' method. If the current prediction score exceeded the previous highest 'prediction' score, the variable 'prediction' would be updated with the new score. Upon the completion of the loop, the method returned the most accurate 'prediction' obtained from the range of values for k. The optimal prediction was the one that possessed the highest prediction score.

The K-NN technique has been widely used in various applications, mostly in the field of classification. Several examples of these applications include:

- Data preprocessing: It involves handling missing values in datasets. One approach to estimate these missing values is by the use of the KNN algorithm, which performs a process called missing data imputation.

- Recommendation Engines: The KNN algorithm utilizes clickstream data from websites to automatically suggest further material to consumers.

- Finance: It has also been applied in various finance and economic scenarios.

- Healthcare: KNN has been applied in the healthcare field to forecast the likelihood of heart attacks and prostate cancer. The method operates by computing the gene expressions with the highest probability.

- Pattern recognition: KNN has also been important in detecting patterns, particularly in the classification of text and digits. This has been particularly beneficial in discerning handwritten numerals that one may encounter on documents or postal envelopes.

Similar to other machine learning algorithms, K-NN possesses both advantages and disadvantages. The suitability of the option depends on the specific project and its intended use.

- Advantages:
  - Simple to execute: Due to the algorithm's straightforwardness and precision, it is often one of the initial classifiers that a novice data scientist will acquire.
  - Demonstrates high adaptability: The algorithm adapts to incorporate any new data by adjusting its parameters when additional training samples are introduced, as all training data is stored in memory.
  - Limited number of hyperparameters: KNN has specifically the k value and the distance measure. This is in contrast to other machine learning algorithms, making it relatively simpler.
- Disadvantages:
  - Not scalable: Due to its sluggish nature, the KNN algorithm requires more memory and data storage than other classifiers. This can incur significant expenses in terms of both time and finances. Increased memory and storage capacity will result in higher business expenditures, while larger amounts of data may lead to longer computational times. Various data structures, such as Ball-Tree, have been developed to tackle computational inefficiencies. However, the choice of an appropriate classifier depends on the specific business situation at hand.
  - The dimensionality curse: The KNN algorithm is susceptible to the curse of dimensionality, resulting in poor performance when dealing with high-dimensional data inputs. This is also known as the peaking phenomenon, where the algorithm reaches its optimal number of features and any more features lead to an increase in classification mistakes, particularly when the sample size is small.
  - Susceptible to overfitting: Owing to the "curse of dimensionality", KNN is also more susceptible to overfitting. Although feature selection and dimensionality reduction techniques are used to avoid this situation, the choice of k can also influence the behavior of the model.

5.3 Random Forest (RF)

The Random Forest Algorithm is very popular due to its user-friendly nature and versatility, which allows it to efficiently address both classification and regression problems. The algorithm's efficacy resides in its capacity to manage intricate datasets and alleviate overfitting, rendering it a viable instrument for diverse prediction tasks in the field of machine learning.

Random Forrest (RF) is a general principle of solving the classifier combination problem by utilizing base classifiers constructed like trees. A Random Forrest Decision Tree is a tree that is generated randomly from a set of potential trees, with random features being considered at each node. The term "at random" indicates that each tree in the set has an equal probability of being selected, meaning that the trees have a "uniform" distribution. The generation of random trees can be accomplished with high efficiency, allowing for the combination of extensive sets of trees. Randomly selecting trees typically results in precise models. Random Forrest is a highly efficient approach for data mining that encompasses the tasks of categorization and regression. It can categorize an object or instance into a predetermined group of categories by considering its attributes, such as age or gender. A Decision Tree begins at the root node and progresses downwards. The starting point of the tree is called a root node whereas where the chain finishes is known as the “leaf” node. Each internal node might have multiple branches extending from it. A node symbolizes a specific attribute, whereas the branches symbolize a spectrum of values [22].

The algorithm is explained briefly for each of the N iterations (N represents the number of trees to construct) as follows:

- Sample Data Selection: The bootstrap method should be used to choose the sample data set that will be used for model training. A bootstrap sample with the same size as the training data is made for every tree.

- Growing the tree: Using splitting rules, the tree is fully grown on this bootstrap. The tree is not trimmed.

- Attribute selection: For every node, only a random subset of the features of a predetermined size is taken into account.

- The tree is preserved in its current state without any pruning. You can use this tree to classify other types of data.

- Output: Each tree in the forest receives the variable vector as input, and each tree provides a classification result (also known as a tree's "votes") for a class. Out of all the trees in the forest, the classification with the most votes is chosen by the forest. The majority vote (classification) from each individually trained tree is used to determine the overall prediction [22].

An ensemble of decision trees makes up a random forest. The ensemble technique refers to the act of mixing multiple models. A set of models is employed to generate predictions instead of relying on a single model.
Ensemble employs two distinct methodologies:

- Bagging: It refers to a machine learning ensemble method where multiple models are trained independently on different subsets of the training data and their predictions are combined to make a final output. It generates a distinct subset of training data by randomly selecting samples with replacements, and the final result is determined by a majority vote process. Bagging, also known as Bootstrap Aggregation, serves as the ensemble technique in the Random Forest algorithm.

- Boosting: It enhances the performance of weak learners by constructing sequential models in order to get the best accuracy in the final model. For instance, ADA BOOST and XG BOOST. Boosting is an ensemble learning technique. Multiple boosting techniques exist, with AdaBoost being the pioneering and highly effective algorithm specifically designed for binary classification. AdaBoost, short for Adaptive Boosting, is a widely used boosting algorithm that merges numerous "weak classifiers" into a single "strong classifier" There exist alternative boosting approaches.

The process of the Random Forest algorithm is described in the following:

- Sample data for model training will be selected using the bootstrap method. A bootstrap sample of the same size as the training data is generated for each tree.

- Tree growth: The bootstrap method is used to fully build the tree by applying splitting rules. The tree remains unpruned.

- Attribute selection: Each node only considers a randomly selected subset of the available characteristics, with a fixed size.

- No pruning is executed, and the tree is preserved in its current state. This tree can be utilized to categorize more datasets.

- Each tree in the forest receives the variable vector as input and produces a classification result, which is referred to as the tree's 'vote' for a class. The forest selects the categorization with the highest number of votes, considering all the trees in the forest. The overall forecast is determined by taking the majority vote from all the separately trained trees, resulting in a classification [22].

Key characteristics of Random Forest:

- Diversity: Individual trees are constructed without considering all traits, resulting in each tree being unique.
- Resistant to the curse of dimensionality: As each tree does not take into account all the features, the number of features is decreased.
- Parallelization: Each tree is generated autonomously using distinct data and properties. This implies that we can utilize the entire processing power of the CPU to construct random forests.
- Train-Test split: In a random forest, there is no need to separate the data into training and testing sets because there will always be a 30% portion of the data that is not used by the decision tree.
- Stability: It is achieved through the utilization of majority voting or averaging to determine the outcome.

The Comparison of Decision Tree vs Random Forest is shown in table 5.1: Random forest is an ensemble of decision trees; yet, there are notable variations in their behavior.

Decision trees	Random Forest
Decision trees often encounter the issue of overfitting when they are allowed to develop without any constraints.	Random forests are constructed using data subsets, and the final result is determined by calculating the average or majority ranking. This approach effectively addresses the issue of overfitting.
A solitary decision tree exhibits greater computational efficiency.	It has a relatively slower pace.
When a decision tree receives a data set containing features as input, it will generate rules to make predictions.	The random forest algorithm collects observations in a random manner, constructs a decision tree, and then calculates the average outcome. It does not employ any predetermined set of formulas.

Table 5.1: Comparison of Decision Tree vs Random Forest

Random forests are significantly more effective than decision trees when the trees exhibit diversity and meet the required criteria.

Hyperparameters in random forests are utilized to optimize model performance and predictive accuracy or to improve computational efficiency.
Hyperparameters to increase the predictive power:

- n_estimators: Number of trees the algorithm builds before averaging the predictions.

- max_features: Maximum number of features random forest considers splitting a node.
- mini_sample_leaf: Determines the minimum number of leaves required to split an internal node.
- criterion: How to split the node in each tree? (Entropy/Gini impurity/Log Loss)
- max_leaf_nodes: Maximum leaf nodes in each tree

Optimizing hyperparameters to enhance the speed:

- The parameter "n_jobs" specifies the number of processors that the engine is permitted to utilize. When the value is 1, it is restricted to utilizing only one processor. However, if the value is -1, there are no limitations imposed.
- The parameter "random_state" is used to regulate the level of unpredictability in the sample. The model will consistently yield the same outcomes when it possesses a fixed random state value and is provided with identical hyperparameters and training data.
- The term "OOB" stands for "out of the bag". The method employed is a cross-validation technique using a random forest algorithm. In this scenario, one-third of the sample is allocated for evaluation purposes rather than being utilized for training the data. These samples are referred to as out-of-bag samples.

Within the scope of this research, the 'random_forest_classifier_init' procedure has encapsulated the procedure of initializing, training, and utilizing a Random Forest classifier for tasks involving classification using 'train_random_forest_model' and 'predict_with_random_forest'. Additionally, the scoring_type parameter denoted that the method may assess the model's performance using a particular scoring metric.

Benefits of Random Forest:

- It is applicable in both classification and regression tasks.
- It addresses the issue of overfitting by determining the output through majority vote or averaging.
- It has strong performance even in the presence of null or missing values in the data.
- Each decision tree is autonomous and does not rely on the others, demonstrating the characteristic of parallelization.

- It exhibits excellent stability by aggregating responses from a substantial number of trees.
- While constructing each decision tree, diversity is maintained by not considering all traits, however, this may not hold true in every situation.
- It is impervious to the negative effects caused by high-dimensional data. As each tree does not take into account all the qualities, the feature space is diminished.
- There is no need to partition the data into training and testing sets because the decision tree created by Bootstrap will always have 30% of the data that it has not seen.

Drawbacks of Random Forest:

- Random forest is far more intricate than decision trees, as it involves making decisions by following the tree's route.
- The training duration is longer compared to other models because of its inherent complexity. Every decision tree must produce output for the provided input data whenever it needs to make a prediction.

In conclusion, the random forest algorithm is an excellent option for those seeking to construct a model quickly and effectively. One of its notable advantages is its ability to effectively manage missing values. This technology is highly effective and frequently utilized in numerous sectors due to its exceptional performance and efficiency. The system is capable of processing binary, continuous, and categorical data. In general, the random forest model is characterized by its speed, simplicity, flexibility, and robustness, however, it does have certain limits.

5.4 Extreme Gradient Boosting (XGB)

Extreme Gradient Boosting (XGB) is a publically accessible library that productively and effectively implements the gradient boosting algorithm to enhance performance [23]. This algorithm utilizes parallel computing to expedite the generation of decision trees, resulting in increased efficiency. Parallelization is facilitated by the use of base learners, which can alternate between external and internal loops. Typically, the outer loop computes inner loop characteristics while generating the decision tree's leaves. The XGB algorithm primarily prioritizes depth, resulting in improved computational performance. Another essential feature of XGB is that optimizing the space available on the disk increases its usage capacity while handling big datasets that do not fit in memory [24]. XGBoost is a technique used for ensemble learning. Occasionally, it may be inadequate to solely depend on the outcomes of a single machine-learning model. Ensemble learning provides a methodical approach to merge the prediction capabilities of several learners. Bagging and boosting are popular ensemble learning techniques, as mentioned before. While these two techniques have the potential to be applied to many statistical models, their most prevalent application has been in the context of decision trees.

- Bagging: Whereas decision trees are known for their high interpretability, they also demonstrate significant variability in their behavior. If a single training dataset is randomly split into two parts, each part can be used to train a decision tree and obtain two models. When both of these models are implemented, they will produce distinct outcomes. Decision trees are commonly linked to high variance as a result of this characteristic.
  Bagging or boosting aggregation techniques are effective in mitigating the variance in any learning algorithm. The base learners of the bagging technique consist of multiple decision trees that are generated simultaneously. The learners are trained using data that has been sampled with replacement. The ultimate forecast is the combined result obtained by averaging the outputs of all the learners.
- Boosting: During the boosting process, the trees are constructed in a sequential manner, with each succeeding tree specifically designed to minimize the errors made by the previous tree. Every tree assimilates knowledge from its ancestors and adjusts the leftover errors. Therefore, the subsequent tree in the sequence will acquire knowledge from an enhanced version of the residuals.
  The base learners utilized in boosting are weak learners characterized by a strong bias and a predictive power slightly superior to random guessing. Each of these weak learners provides crucial information for prediction, allowing the boosting strategy to create a strong learner by skillfully merging these weak learners. The ultimate robust learner reduces both the bias and the variation.

Unlike bagging methods such as Random Forest, which allows trees to grow to their full potential, boosting employs trees with a reduced number of splits. These small, superficial trees are highly susceptible to interpretation. Optimal selection of parameters like as the number of trees or iterations, the learning rate of gradient boosting, and the depth of the tree can be achieved by validation approaches like k-fold cross validation. The presence of a high quantity of trees can potentially result in overfitting. Therefore, it is imperative to meticulously select the stopping criterion for boosting.
The gradient-boosting ensemble technique comprises three straightforward steps:

- A preliminary model, denoted as F_0, is established to forecast the target variable y. This model will be linked to the residual (y – F_0).

- A new model, denoted as h_1, is fitted to the residuals obtained from the previous phase.
- Now, the combination of F_0 and h1 results in F_1, which is an amplified form of F_0. The mean squared error resulting from F_1 will be lower than the mean squared error resulting from F_0.

$$F_1(x) < -F_0(x) + h_1(x)$$

In order to enhance the performance of F_1, we can replicate the residuals of F_1 and construct a novel model, F_2:

$$F_2(x) < -F_1(x) + h_2(x)$$

This process can be repeated for 'm' iterations until the residuals have been decreased to the greatest extent possible.

$$F_m(x) < -F_{m-1}(x) + h_m(x)$$

Distinctive Characteristics of XGBoost are explained in the following.

- Regularization: XGBoost offers the ability to apply penalties to complex models using both L1 and L2 regularization techniques. Regularization mitigates the issue of overfitting.
- Dealing with data that is thinly distributed: Data sparsity can occur due to missing values or data processing techniques such as one-hot encoding. XGBoost utilizes a split finding method that is cognizant of sparsity, enabling it to effectively handle various sorts of sparsity patterns present in the data.
- Weighted quantile sketch: It is a statistical technique used to estimate the quantiles of a dataset, taking into account the weights assigned to each data point. The majority of current tree-based algorithms are capable of identifying the split points when the data points have equal weights, utilizing the quantile sketch approach. Nevertheless, they lack the necessary capabilities to process data with assigned weights. XGBoost utilizes a distributed weighted quantile sketch algorithm to efficiently manage data with weights.
- Block structure for parallel learning: XGBoost has the capability to utilize several cores on the CPU, which results in faster computing. This is achievable due to the presence of a block structure in its system design. Information is organized and stored in memory units known as blocks. Unlike previous methods, this allows for the reutilization of the data layout in later iterations, rather than recomputing it. This capability is especially beneficial for tasks such as identifying splits and subsampling columns.

- Cache awareness: XGBoost necessitates non-sequential memory access to retrieve gradient statistics based on row index. Therefore, XGBoost has been specifically engineered to maximize the utilization of hardware resources. The process involves assigning internal buffers to each thread, which serve as storage for the gradient statistics.

- Out-of-core computing: This feature enhances the utilization of the available disk space and maximizes its efficiency when managing large datasets that exceed the memory capacity.

The advantages of XGBoost are listed subsequently.

- High accuracy: XGBoost is renowned for its exceptional accuracy and has consistently demonstrated its superiority over other machine learning algorithms in numerous predictive modeling applications.

- Scalability: XGBoost exhibits excellent scalability, enabling it to efficiently process vast datasets containing millions of rows and columns.

- Efficiency: XGBoost is specifically engineered to possess high computational efficiency, enabling rapid model training even on extensive datasets.

- Flexibility: XGBoost is capable of handling diverse data types and objectives, encompassing regression, classification, and ranking tasks.

- Regularization: XGBoost employs regularization methods to prevent overfitting and enhance generalization capabilities.

- Interpretability: XGBoost offers feature importance scores to aid consumers in comprehending the significance of different aspects in producing predictions.

- Open-source: XGBoost is a freely available library that is extensively utilized and endorsed by the data science community.

The xg_boost_classifier_init function was used in this study inside a class. The objective of this procedure was to initialize and train an XGBoost classifier by utilizing training data, and subsequently employ the trained model to generate predictions on test data. The train_xg_boost_model was implemented to train an XGBoost model using the given training data (train_data) and labels (train_labels). The function produced the trained XGBoost model, referred to as xg_boost_model. The scoring_type parameter signified that the method may assess the model's performance using a particular scoring metric.

5.5 Support Vector Machines (SVM)

Support Vector Machines (SVM) is an innovative statistical learning method that has gained significant attention in the field of pattern recognition and machine learning. According to the statistical learning theory, it focuses on the concept of structural risk minimization (SRM) which enhances the ability to make accurate predictions in a wide range of situations. SVM is applicable for both classification and regression applications [25].

The goal of the support vector machine technique is to identify a hyperplane in N-dimensional space (N being the number of features) that effectively separates the data points into discrete classes. Support vector machines (SVMs) are a collection of supervised learning techniques utilized for the purposes of classification, regression, and the identification of outliers [26].

The benefits of support vector machines include:

- Efficient in spaces with a large number of dimensions.
- Remains effective when the number of dimensions exceeds the number of samples.
- Utilizes a subset of training points known as support vectors in the decision function, resulting in efficient memory usage.
- Flexible: many kernel functions can be designated for the decision function. Standard kernels are available, however, users can also define their own custom kernels.

The drawbacks of support vector machines encompass:

- When the number of features greatly exceeds the number of samples, it is vital to carefully select Kernel functions and regularization terms in order to prevent over-fitting.
- SVMs do not inherently offer probability estimates. Instead, these estimates are obtained through a computationally intensive process known as five-fold cross-validation [26].

This research utilized SVC class implementations to perform binary and multi-class classification on a given dataset. Similar to previous classifiers, SVC, NuSVC, and LinearSVC require two arrays as input: an array X with dimensions (n_samples, n_features) containing the training samples, and an array y with class labels (strings or integers) with dimensions (n_samples). Support Vector Machines exhibit significant computational and storage demands that escalate proportionally with the number of training vectors [26].

In this research, svm_model_classifier_init method was initialized within a class. The objective of this procedure was to initialize and train a Support Vector Machine (SVM) classifier by utilizing training data, and subsequently employ the trained model to generate predictions on test data. The train_svm_model method was used to train an SVM model using the given training data (train_data) and labels (train_labels). The method yields two outputs: the trained SVM model (svm_model) and a scaler (scaler) employed for feature scaling. Then the predict_with_svm function was utilized to generate predictions on the test data (test_data) using the trained SVM model and the scaler acquired during training. Also, the scoring_type parameter indicated that the method may assess the model's performance using a particular scoring metric.

Chapter 6

Evaluation

6.1 Overview on Evaluation

The domain of assessment for information retrieval and natural language processing systems is intricate. There exist two distinct forms of evaluation: qualitative evaluation and quantitative evaluation. Qualitative evaluation involves requesting feedback from users or user groups to determine if the output of an information retrieval system is satisfactory or not. Qualitative evaluation mostly centers on the subjective experiences of one or more users regarding a system. Quantitative evaluation involves using a systematic method to measure and express the outcomes of an information retrieval system in numerical form [8].

Quantitative evaluation serves various goals. There may also be distinct restrictions on the quantity of data utilized for training and evaluation. There are situations where prioritizing high recall is more important than high precision, while on other occasions, the converse is true [8].

A development set is utilized during the process of system development. A development set refers to a collection of data that is used either to create rules for an artifact or to train a machine learning system. In the field of machine learning, the development set is commonly referred to as the training set. Its purpose is to train the machine learning system. A portion of the training set can be allocated for error analysis of the method, and the machine learning algorithm can be fine-tuned based on the identified mistakes, a process known as parameter tuning. This portion of the training dataset is referred to as the development test set [8].

A separate test set is reserved just for evaluating the performance of the artifact. This test set is not utilized during the development or training phases and is commonly referred to as held-out data [8].

6.2 Cross-validation

In cases where data is limited, a technique known as k-fold cross-validation is employed. This involves dividing the entire dataset into k folds, with k-1 folds used for training and the remaining 1 fold used for evaluation. This process is repeated, with the folds being switched each time until all folds have been trained and tested on the remaining k-1 folds. Finally, an average is calculated based on the results. In cross-validation, a subset of training data sets is generated and used to train the machine learning model. Cross-validation is employed to address the issue of overfitting [7], [8].

In this study, the number of folds (k) and random_state was set to 5 and 42, respectively. The 'random_state' parameter is frequently utilized in functions that involve randomness to regulate the initial value for the random number generator.

6.3 Metrics

Precision and recall are two measures utilized to quantify the effectiveness of a retrieval system. Precision is a metric that calculates the ratio of correct instances retrieved to all retrieved instances, as shown in formula 6.1. The recall metric calculates the ratio of correctly retrieved instances to all instances that should have been retrieved, as shown in formula 6.2. Instances refer to entities within a text or an entire document within a collection of documents (corpus) that have been retrieved. A confusion matrix, as shown in the table 6.1, is commonly utilized to elucidate the distinct entities.

The F_1-score is a metric that represents the harmonic mean of precision and recall. It is often referred to as the F score, which typically refers to the F_1-score specifically. The formula 6.3 provides the calculation for the F_1-score. Accuracy is a metric that quantifies the ratio of correctly identified occurrences, including both positive and negative, out of all the instances that were identified. Accuracy is calculated as the weighted average of precision and the inverse of precision, as shown in formula 6.4 [8].

	Positive predicted annotation	Negative predicted annotation
Positive gold annotation	True positive (tp)	False negative (fn)
Negative gold annotation	False positive (fp)	True negative (tn)

Table 6.1: Confusion matrix: predicted annotation is what the algorithm retrieves or annotates and gold annotation is what was marked up or annotated by a human

$$Precision : P = \frac{tp}{tp + fp} \tag{6.1}$$

$$Recall : R = \frac{tp}{tp + fn} \tag{6.2}$$

$$F - score : F_1 = F = \frac{tp}{tp + fn} \tag{6.3}$$

$$Accuracy : A = \frac{tp + tn}{tp + tn + fp + fn} \tag{6.4}$$

Chapter 7

Results

This chapter presents the assessment of the feature selectors and classifiers. The evaluation utilizes four metrics, namely Accuracy, F_1 score, Precision, and Recall, which have been previously specified in the previous chapter.

The results for both feature selectors, PCA and Boruta, have been categorized into three groups:

- Distance signal
- Velocity signal
- Combination of distance and velocity signals

The data for each group has been partitioned into 5 folds using the cross-validation technique. Furthermore, with the evaluation metrics for each fold, the Mean and Standard Deviation (SD) values of the scores across all folds have been documented for each classifier evaluation.

The number of selected features for each evaluation group is listed below.

- Number of selected features in PCA for distance: 10
- Number of selected features in PCA for velocity: 10
- Number of selected features in PCA for distance and velocity combination: 11
- Number of selected features in Boruta for distance: 9
- Number of selected features in Boruta for velocity: 10
- Number of selected features in Boruta for distance and velocity combination: 11

7.1 PCA for Distance

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8571	0.8571	1.0	0.7142	0.8333	0.8523	0.0908
RF	1.0	1.0	1.0	1.0	0.8333	0.9666	0.0666
XGB	1.0	0.8571	0.7142	0.8571	0.8333	0.8523	0.0908
SVM	1.0	1.0	1.0	0.8571	0.8333	0.9380	0.0761

Table 7.1: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8571	0.8888	1.0	0.75	0.8	0.8592	0.0850
RF	1.0	1.0	0.8888	1.	0.8	0.9377	0.0812
XGB	0.85714286	0.8571	0.75	0.8888	0.8	0.8306	0.0494
SVM	1.0	1.0	1.0	0.8571	0.8	0.9314	0.0859

Table 7.2: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	0.8	1.0	0.75	1.0	0.9099	0.1113
RF	1.0	1.0	0.5714	1.0	0.6666	0.8476	0.1890
XGB	1.0	1.0	0.75	0.8	1.0	0.9099	0.1113
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.3: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.75	1.0	1.0	0.75	0.6666	0.8333	0.1394
RF	1.0	1.0	1.0	1.0	0.6666	0.9333	0.1333
XGB	0.75	0.75	0.75	1.0	0.6666	0.7833	0.1130
SVM	1.0	1.0	1.0	0.75	0.6666	0.8833	0.1452

Table 7.4: Recall of 5 folds of classifiers

7.2 PCA for Velocity

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.7142	0.7142	0.7142	0.7142	0.8333	0.7380	0.0476
RF	0.8571	1.0	0.7142	0.7142	0.6666	0.7904	0.1227
XGB	1.0	1.0	0.8571	0.8571	0.8333	0.9095	0.0743
SVM	1.0	0.8571	0.8571	0.7142	0.8333	0.8523	0.0908

Table 7.5: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8	0.8	0.8888	0.8	0.8571	0.8292	0.0371
RF	0.8888	1.0	0.75	0.75	0.75	0.8277	0.1015
XGB	1.0	1.0	0.75	0.8888	0.6666	0.8611	0.1337
SVM	1.0	0.8888	0.8888	0.75	0.8571	0.8769	0.0799

Table 7.6: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.6666	0.6666	0.75	0.6666	0.75	0.7	0.0408
RF	1.0	1.0	0.75	0.75	0.6	0.82	0.1568
XGB	1.0	1.0	0.75	0.8	1.0	0.9099	0.1113
SVM	1.0	0.8	0.8	0.75	0.75	0.82	0.0927

Table 7.7: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	1.0	1.0	1.0	0.0
RF	1.0	1.0	0.75	0.75	1.0	0.9	0.1224
XGB	1.0	1.0	0.75	1.	0.6666	0.8833	0.1452
SVM	1.0	1.0	1.0	0.75	1.0	0.95	0.0999

Table 7.8: Recall of 5 folds of classifiers

7.3 PCA for Distance and Velocity Combination

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	0.8571	0.7142	0.5714	0.8333	0.7952	0.1441
RF	1.0	1.0	0.7142	0.7142	0.8333	0.8523	0.1281
XGB	1.0	1.0	0.5714	0.7142	0.8333	0.8238	0.1660
SVM	1.0	1.0	0.8571	0.7142	0.8333	0.8809	0.1085

Table 7.9: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	0.8888	0.8	0.6666	0.8571	0.8425	0.1094
RF	1.0	1.0	0.8888	0.8571	0.8571	0.9206	0.0658
XGB	1.0	1.0	0.75	0.75	0.8571	0.8714	0.1120
SVM	1.0	1.0	0.8888	0.75	0.8571	0.8992	0.0942

Table 7.10: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	0.8	0.6666	0.6	0.75	0.7633	0.1367
RF	1.0	1.	0.6666	1.0	0.75	0.8833	0.1452
XGB	1.0	1.0	0.6	0.75	0.75	0.82	0.1568
SVM	1.0	1.0	0.8	0.75	0.75	0.86	0.1157

Table 7.11: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	0.75	1.0	0.95	0.0999
RF	1.0	1.0	1.0	0.75	1.0	0.95	0.0999
XGB	1.0	1.0	0.75	0.75	1.0	0.9	0.1224
SVM	1.0	1.0	1.0	0.75	1.0	0.95	0.0999

Table 7.12: Recall of 5 folds of classifiers

7.4 Boruta for Distance

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	1.0	1.0	1.0	0.0
RF	1.0	1.0	1.0	0.9705	1.0	0.9941	0.0117
XGB	0.9705	0.9705	0.9705	0.9705	1.0	0.9764	0.0117
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.13: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	1.0	1.0	1.0	0.0
RF	1.0	1.0	1.0	0.9703	1.0	0.9940	0.0118
XGB	0.9703	0.9703	0.9703	0.9703	1.0	0.9762	0.0118
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.14: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	1.0	1.0	1.0	0.0
RF	1.0	1.0	1.0	0.9687	1.0	0.9937	0.0124
XGB	0.9687	0.9687	0.9687	0.9687	1.0	0.975	0.0125
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.15: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	1.0	1.0	1.0	1.0	1.0	1.0	0.0
RF	1.0	1.0	1.0	1.0	1.0	1.0	0.0
XGB	0.9736	0.9736	0.9736	0.9736	1.0	0.9789	0.0105
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.16: Recall of 5 folds of classifiers

7.5 Boruta for Velocity

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.7941	0.6764	0.7352	0.7941	0.7647	0.7529	0.0440
RF	1.0	1.0	1.0	0.9705	1.0	0.9941	0.0117
XGB	0.9705	0.9705	0.9705	0.9705	1.0	0.9764	0.0117
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.17: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.7939	0.6761	0.7350	0.7939	0.7647	0.7527	0.0440
RF	1.0	1.0	1.0	1.0	1.0	1.0	0.0
XGB	0.9703	0.9703	0.9703	0.9703	1.0	0.9762	0.01185
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.18: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.7986	0.7586	0.7588	0.7986	0.7765	0.7782	0.0178
RF	1.0	1.0	1.0	0.9687	1.0	0.9937	0.0124
XGB	0.9687	0.9687	0.9687	0.9687	1.0	0.975	0.0125
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.19: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8017	0.7035	0.7491	0.8017	0.7754	0.7663	0.0369
RF	1.0	1.0	1.0	0.9736	1.0	0.9947	0.0105
XGB	0.9736	0.9736	0.9736	0.9736	1.0	0.9789	0.0105
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.20: Recall of 5 folds of classifiers

7.6 Boruta for Distance and Velocity Combination

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8823	0.8823	0.9117	0.8823	0.9411	0.9	0.0235
RF	1.0	1.0	1.0	0.9705	1.0	0.9941	0.0117
XGB	0.9705	0.9705	0.9705	0.9705	1.0	0.9764	0.0117
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.21: Accuracy of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8819	0.8807	0.9116	0.8819	0.9403	0.8993	0.0236
RF	1.0	1.0	1.0	0.9703	1.0	0.9940	0.0118
XGB	0.9703	0.9703	0.9703	0.9703	1.0	0.9762	0.0118
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.22: F_1 score of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8823	0.8807	0.9166	0.8823	0.9403	0.9004	0.0240
RF	1.0	1.0	1.0	0.9687	1.0	0.99375	0.01249
XGB	0.9687	0.9687	0.9687	0.9687	1.0	0.975	0.0125
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.23: Precision of 5 folds of classifiers

Classifiers	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std Dev
KNN	0.8877	0.8807	0.9210	0.8877	0.9403	0.9035	0.0231
RF	1.0	1.0	1.0	0.9736	1.0	0.9947	0.0105
XGB	0.9736	0.9736	0.9736	0.9736	1.0	0.9789	0.0105
SVM	1.0	1.0	1.0	1.0	1.0	1.0	0.0

Table 7.24: Recall of 5 folds of classifiers

7.7 Results Summary

In this section, a summary of the reported results is listed to compare the Mean value for each feature selector and the classifier used in that

PCA for Distance (*Mean* ± *Std.Dev.*)				
Classifiers	Accuracy	F_1 score	Precision	Recall
KNN	0.8523 ± 0.0908	0.8592 ± 0.0850	0.9099 ± 0.1113	0.8333 ± 0.1394
RF	0.9666 ± 0.0666	0.9377 ± 0.812	0.8476 ± 0.1890	0.9333 ± 0.1333
XGB	0.8523 ± 0.0908	0.8306 ± 0.0494	0.9099 ± 0.1113	0.7833 ± 0.1130
SVM	0.9380 ± 0.0761	0.9314 ± 0.0859	1.0 ± 0.0	0.8833 ± 0.1452
PCA for Velocity (*Mean* ± *Std.Dev.*)				
KNN	0.7380 ± 0.0476	0.8292 ± 0.0371	0.7 ± 0.0408	1.0 ± 0.0
RF	0.7904 ± 0.1227	0.8277 ± 0.1015	0.82 ± 0.1568	0.9 ± 0.1224
XGB	0.9095 ± 0.0743	0.8611 ± 0.1337	0.9099 ± 0.1113	0.8833 ± 0.1452
SVM	0.8523 ± 0.0908	0.8769 ± 0.0799	0.82 ± 0.0927	0.95 ± 0.0999
PCA for Distance and Velocity combination (*Mean* ± *Std.Dev.*)				
KNN	0.7952 ± 0.1441	0.8425 ± 0.1094	0.7633 ± 0.1367	0.95 ± 0.0999
RF	0.8523 ± 0.1281	0.9206 ± 0.0658	0.8833 ± 0.1452	0.95 ± 0.0999
XGB	0.8238 ± 0.1660	0.8714 ± 0.1120	0.82 ± 0.1568	0.9 ± 0.1224
SVM	0.8809 ± 0.1085	0.8992 ± 0.0942	0.86 ± 0.1157	0.95 ± 0.0999

Table 7.25: Results summary of PCA

Boruta for Distance (*Mean* ± *Std.Dev.*)				
Classifiers	Accuracy	F_1 score	Precision	Recall
KNN	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
RF	0.9941 ± 0.0117	0.9940 ± 0.0118	0.9937 ± 0.0124	1.0 ± 0.0
XGB	0.9764 ± 0.0117	0.9762 ± 0.0118	0.975 ± 0.0125	0.9789 ± 0.0105
SVM	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
Boruta for Velocity (*Mean* ± *Std.Dev.*)				
KNN	0.7529 ± 0.0440	0.7527 ± 0.0440	0.7782 ± 0.0178	0.7663 ± 0.0369
RF	0.9941 ± 0.0117	1.0 ± 0.0	0.9937 ± 0.0124	0.9947 ± 0.0105
XGB	0.9764 ± 0.0117	0.9762 ± 0.0118	0.975 ± 0.0125	0.9789 ± 0.0105
SVM	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
Boruta for Distance and Velocity combination (*Mean* ± *Std.Dev.*)				
KNN	0.9 ± 0.0235	0.8993 ± 0.0236	0.9004 ± 0.0240	0.9035 ± 0.0231
RF	0.9941 ± 0.0117	0.9940 ± 0.0118	0.9937 ± 0.0124	0.9947 ± 0.0105
XGB	0.9764 ± 0.0117	0.9762 ± 0.0118	0.975 ± 0.0125	0.9789 ± 0.0105
SVM	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 7.26: Results summary of Boruta

Chapter 8

Conclusion and Future Development

8.1 Conclusion

Parkinson's disease is a progressive disorder that impacts the neurological system and the bodily functions regulated by the nerves. This has a significant impression on a vast number of individuals globally. The onset of symptoms is gradual. So monitoring and conducting follow-ups on PD are both effective and crucial. As per the criteria issued by the Movement Disorder Society, the finger-tapping test (FTT) is commonly used to assess bradykinesia. The data regarding these patterns was presented in the vision-based 3D Parkinson's disease (PD) hand dataset. This dataset comprises 133 video clips of finger-tapping, including recordings of 35 PD patients and 60 healthy controls. This attempt has examined the motion of the index and thumb digits as they approach and separate from one another. Subsequently, Python libraries were employed to obtain the distance and velocity data of these movements. The subsequent activity was employing the tsfresh library for the purpose of extracting features. The Boruta and Principal Component Analysis (PCA) methods were applied to identify significant features to remove redundant features. Moreover, numerous supervised machine learning classification methods have been assessed to ascertain the presence or absence of Parkinson's disease in an individual. The algorithms encompassed in this set are K-Nearest Neighbors, Random Forests, Extreme Gradient Boosting, and Support Vector Machine. In addition, Cross-validation with 5 folds was used to mitigate the problem of overfitting. Ultimately, the machine-learning models were evaluated using the criteria of accuracy, precision, recall, and f1-score.
According to the results, the highest percentage of evaluating metrics of the classifier were as follows:

- PCA:
  - For the "Distance" dataset: "RF"
  - For the "Velocity" dataset: "XGB"
  - For the "Distance and Velocity Combination" dataset: "RF"
- Boruta:
  - For the "Distance" classifier: "SVM" and "KNN"
  - For the "Velocity" classifier: "SVM"
  - For the "Distance and Velocity Combination" dataset: "SVM"

According to the results obtained, the combination of Boruta as the feature selector and Support Vector Machine as the classifier demonstrated the best performance across all four metrics. This point should be noted that obtaining 100% in some metrics is probably due to data scarcity.
In conclusion, this research proposes an approach for conducting a finger-tapping test for Parkinson's disease (PD) patients using 3D vision technology. The findings outlined in this study indicate that the suggested system has the potential to be a valuable tool for assisting clinicians in Parkinson's disease follow-up.

8.2 Future Development

This section explores future development prospects aimed at improving patient quality of life during the Parkinson's disease period. These advancements have the potential to significantly enhance patient care and well-being, opening the door to a more sympathetic and practical method of treating Parkinson's disease.

- **Enhanced Diagnostic Precision**

  This study serves as a basis on which a vision-based analytic approach for three-dimensional hand movement can be used to diagnose PD. In the future, efforts may be aimed at improving the accuracy of the diagnostic model. There is obvious scope for improvement by widening the data set and encompassing different phases of disease and minor variances in movement patterns within Parkinson's disease. The developmental model could also fine-tune how it differentiates between distinct levels of illness development in PD.

- **Real-Time Monitoring and Early Intervention**

  Given the advances in technology, developing this model into a real-time monitoring system would be an interesting area for further research. A real-time monitoring system may make it possible for early detection of such changes that are associated with the progression of PD. This is critical considering that the success of neurodegenerative disorders management depends on when one intervenes during this process.

- **Incorporation of Multimodal Data**

  Future developments would involve the inclusion of multi-modal data with a view to improve the model's accuracy and resilience. The existing study is dedicated to the analysis of hand movements; however, integration of these data with voicing aspects, gait analysis as well as clinical examinations can potentially yield more precise results regarding diagnosing PD patients. Integration of these two parameters would provide an improved diagnostic mechanism that incorporates a wider range of symptoms.

- **Personalized Treatment Plans**

  While the above-mentioned model lays emphasis on diagnosis, the next stage could focus on individualized therapies. In this regard, the model would help identify movement patterns that are related to individual answers to specific therapeutic approaches in order to design tailored therapy strategies applicable to PD patients. Such an individualized approach can help to enhance treatment results and enhance general well-being in patients suffering from Parkinson's illness.

- **Collaboration with Healthcare Professionals**

  The future is here as machine learning models and healthcare professionals work together to address AI advances in healthcare. Developers should consider a smooth flow of the developed mode into the day-to-day clinic. It also includes close collaboration with health care providers to make sure that the model matches the present diagnostic procedures and complementary the decisions made during a clinical situation.

- **Assessing Disease Progression**

  Developing new models that can determine the progression of Parkinson's disease is an area worth pursuing in future studies. Using different kinds of rating scales like the MDS-UPDRS could improve the evaluation of how the disease progresses over time. The utilization of artificial intelligence algorithms on movement recordings will give a numerical indication of the degree of insult and allow tracking of PD progression.

In conclusion, this thesis provided an outline for further insightful investigations into Parkinson's disease follow-up. By acknowledging these potentialities, one can work towards developing more accurate, focused, and efficacious strategies to address this prevalent neurodegenerative disorder.

Bibliography

[1] Gianluca Amprimo, Irene Rechichi, Claudia Ferraris, and Gabriella Olmo. «Objective Assessment of the Finger Tapping Task in Parkinson's Disease and Control Subjects using Azure Kinect and Machine Learning». In: *2023 IEEE 36th International Symposium on Computer-Based Medical Systems (CBMS)*. 2023, pp. 640–645. DOI: 10.1109/CBMS58004.2023.00293 (cit. on pp. 1, 2, 8–10).

[2] Renee M. Hendrick and Mohammad T. Khasawneh. «An Investigation into the Use and Meaning of Parkinson's Disease Clinical Scale Scoresn». In: (May 2021). DOI: 10.1155/2021/1765220 (cit. on p. 1).

[3] Zhilin Guo, Weiqi Zeng, Taidong Yu, Yan Xu, Yang Xiao, Xuebing Cao, and Zhiguo Cao. «Vision-Based Finger Tapping Test in Patients With Parkinson's Disease via Spatial-Temporal 3D Hand Pose Estimation». In: *IEEE Journal of Biomedical and Health Informatics* 26.8 (2022), pp. 3848–3859. DOI: 10.1109/JBHI.2022.3162386 (cit. on pp. 1, 3, 11).

[4] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas Kempa-Liehr. «Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)». In: *Neurocomputing* 307 (May 2018). DOI: 10.1016/j.neucom.2018.03.067 (cit. on pp. 2, 11).

[5] Hongwei Wang. «A method of feature selection for continuous features base on similarity degrees of interval numbers». In: *2013 International Conference on Information and Network Security (ICINS 2013)*. 2013, pp. 1–5. DOI: 10.1049/cp.2013.2464 (cit. on pp. 2, 18).

[6] Dingcheng Feng, Feng Chen, and Wenli Xu. «Efficient leave-one-out strategy for supervised feature selection». In: *Tsinghua Science and Technology* 18.6 (2013), pp. 629–635. DOI: 10.1109/TST.2013.6678908 (cit. on pp. 2, 19).

[7] Vinod Jain and Mayank Agrawal. «Heart Failure Prediction Using XGB Classifier, Logistic Regression and Support Vector Classifier». In: *2023 International Conference on Advancement in Computation Computer Technologies (InCACCT)*. 2023, pp. 1–5. DOI: 10.1109/InCACCT57535.2023.10141752 (cit. on pp. 2, 3, 33, 48).

[8] Hercules Dalianis. «Evaluation Metrics and Evaluation». In: *Clinical Text Mining: Secondary Use of Electronic Patient Records.* Cham: Springer International Publishing, 2018, pp. 45–53. ISBN: 978-3-319-78503-5. DOI: `10.1007/978-3-319-78503-5_6`. URL: `https://doi.org/10.1007/978-3-319-78503-5_6` (cit. on pp. 3, 47, 48).

[9] Yan-Ya Chen, Bing-Sheng Guan, Ze-Kai Li, Qiao-Hong Yang, Tian-Jiao Xu, Han-Bing Li, and Qin-Yang Wu. «Application of telehealth intervention in Parkinson's disease: A systematic review and meta-analysis». In: *Journal of Telemedicine and Telecare* 26.1-2 (2020). PMID: 30153767, pp. 3–13. DOI: `10.1177/1357633X18792805`. eprint: `https://doi.org/10.1177/1357633X18792805`. URL: `https://doi.org/10.1177/1357633X18792805` (cit. on p. 4).

[10] Daniel Palacios-Alonso, Guillermo Meléndez-Morales, Agustín López-Arribas, Carlos Lázaro-Carrascosa, Andrés Gómez-Rodellar, and Pedro Gómez-Vilda. «MonParLoc: A Speech-Based System for Parkinson's Disease Analysis and Monitoring». In: *IEEE Access* 8 (2020), pp. 188243–188255. DOI: `10.1109/ACCESS.2020.3031646` (cit. on pp. 4, 5).

[11] Jonathan A. Stamford, Peter N. Schmidt, and Karl E. Friedl. «What Engineering Technology Could Do for Quality of Life in Parkinson's Disease: A Review of Current Needs and Opportunities». In: *IEEE Journal of Biomedical and Health Informatics* 19.6 (2015), pp. 1862–1872. DOI: `10.1109/JBHI.2015.2464354` (cit. on pp. 5, 6).

[12] Surekha Tadse, Muskan Jain, and Pankaj Chandankhede. «Parkinson's Detection Using Machine Learning». In: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS).* 2021, pp. 1081–1085. DOI: `10.1109/ICICCS51141.2021.9432340` (cit. on p. 6).

[13] Junjie Li, Huaiyu Zhu, Haotian Wang, Bo Wang, Zhidong Cen, Dehao Yang, Peng Liu, Wei Luo, and Yun Pan. «A Three-Dimensional Finger-Tapping Framework for Recognition of Patients With Mild Parkinson's Disease». In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 31 (2023), pp. 3331–3340. DOI: `10.1109/TNSRE.2023.3296883` (cit. on p. 7).

[14] Saman Siadati. *Fundamentals of Python programming.* Apr. 2018. DOI: `10.13140/RG.2.2.13071.20642` (cit. on p. 10).

[15] Maximilian Christ. *Overview on extracted features.* Accessed on 16/07/2023. 2023. URL: `https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html#overview-on-extracted-features` (cit. on p. 12).

[16] Renjun Wang, Nigela Tuerxun, and Jianghua Zheng. «Scaling effects of chlorophyll content in walnut leaves estimations with coupling Boruta algorithm and machine learning model». In: *2023 11th International Conference on Agro-Geoinformatics (Agro-Geoinformatics)*. 2023, pp. 1–5. DOI: `10.1109/Agro-Geoinformatics59224.2023.10233602` (cit. on p. 20).

[17] Miron B. Kursa and Witold R. Rudnicki. «Feature Selection with the Boruta Package». In: *Journal of Statistical Software* 36.11 (2010), pp. 1–13. DOI: `10.18637/jss.v036.i11`. URL: `https://www.jstatsoft.org/index.php/jss/article/view/v036i11` (cit. on p. 20).

[18] Danyang Wu, Han Zhang, Feiping Nie, Rong Wang, Chao Yang, Xiaoxue Jia, and Xuelong Li. «Double-Attentive Principle Component Analysis». In: *IEEE Signal Processing Letters* 27 (2020), pp. 1814–1818. DOI: `10.1109/LSP.2020.3027462` (cit. on p. 23).

[19] Wayo Puyati and Aranya Walairacht. «Efficiency Improvement for Unconstrained Face Recognition by Weightening Probability Values of Modular PCA and Wavelet PCA». In: *2008 10th International Conference on Advanced Communication Technology*. Vol. 2. 2008, pp. 1449–1453. DOI: `10.1109/ICACT.2008.4494037` (cit. on p. 23).

[20] Quang Vinh Nguyen, Mao Lin Huang, and Simeon Simoff. «Enhancing Scatterplots with Start-plots for Visualising Multi-dimensional Data». In: *2020 24th International Conference Information Visualisation (IV)*. 2020, pp. 80–85. DOI: `10.1109/IV51561.2020.00023` (cit. on p. 28).

[21] Rishabh Bathija, Vanshika Bajaj, Chandni Megnani, Jasmine Sawara, and Sanjay Mirchandani. «Revolutionizing Recruitment: A Comparative Study Of KNN, Weighted KNN, and SVM - KNN for Resume Screening». In: *2023 8th International Conference on Communication and Electronics Systems (ICCES)* (2023), pp. 834–840. URL: `https://api.semanticscholar.org/CorpusID:260387560` (cit. on p. 34).

[22] Ch. Ravi Sekhar, Minal, and E. Madhu. «Mode Choice Analysis Using Random Forrest Decision Trees». eng. In: *Transportation Research Procedia* 17 (2016), pp. 644–652. ISSN: 2352-1465 (cit. on pp. 37, 38).

[23] Asad Ullah, Huma Qayyum, Muhammad Khateeb Khan, and Fawad Ahmad. «Sepsis Detection Using Extreme Gradient Boost (XGB): A Supervised Learning Approach». In: *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*. 2021, pp. 1–6. DOI: `10.1109/MAJICC53071.2021.9526260` (cit. on p. 41).

[24] P. Nagaraj, M. Venkat Dass, Erukala Mahender, and Kallepalli Rohit Kumar. «Breast Cancer Risk Detection using XGB Classification Machine Learning Technique». In: *2022 IEEE International Conference on Current Development in Engineering and Technology (CCET)*. 2022, pp. 1–5. DOI: `10.1109/CCET56606.2022.10080076` (cit. on p. 41).

[25] Alireza Kazemi, Reza Boostani, Mahmoud Odeh, and Mohammad Rasmi AL-Mousa. «Two-Layer SVM, Towards Deep Statistical Learning». In: *2022 International Engineering Conference on Electrical, Energy, and Artificial Intelligence (EICEEAI)*. 2022, pp. 1–6. DOI: `10.1109/EICEEAI56378.2022.10050469` (cit. on p. 45).

[26] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 45).