TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

Konstantin Bardõš, 153011 IAPM

ANALYSIS OF INTERPRETABLE ANOMALIES AND KINEMATIC PARAMETERS IN LURIA'S ALTERNATING SERIES TESTS FOR PARKINSON'S DISEASE MODELING

Master's thesis

Supervisors: Sven Nõmm, PhD
Aaro Toomela, PhD
Kadri Medijainen, MSc

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Konstantin Bardõš, 153011 IAPM

LURIA VAHELDUTAVATE SEERIATE TÕLGENDATAVATE ANOMAALIATE JA KINEMAATILISTE PARAMEETRITE ANALÜÜS PARKINSONI TÕBI MODELLEERIMISEL

Magistritöö

Juhendajad: Sven Nõmm, PhD

Aaro Toomela, PhD

Kadri Medijainen, MSc

Tallinn 2018

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Konstantin Bardõš

May 7, 2018

Analysis of Interpretable Anomalies and Kinematic Parameters in Luria's Alternating Series Tests for Parkinson's Disease Modeling

Abstract

Primary goal of present thesis is to conduct analysis of patterns drawn during Luria's alternating series tests, extract interpretable feature set and develop machine learning model, capable of correct differentiation of Parkinson's disease patients from healthy control subjects.

Luria's alternating series fine motor tests are being used in psychology and neurology to assess level of disorder in motion planning and execution during handwriting, which is approved biomarker for Parkinson's disease.

A novel method to analyze Luria's alternating series patterns drawn during fine motor test constitute main result of the present thesis. Majority of solutions available in the literature are based either on the analysis of entire drawing or individual strokes. Distinctive feature of the proposed approach is that it allows to analyze patterns, considering their logical structure with any required level of detail. To achieve this, unique supervised and unsupervised machine learning techniques are applied. Computer vision technique is used to split pattern into logical segments. Based on this information, feature sequences describing different kinematic properties of the drawing are constructed. During next stages, neural-network based models are used to generate feature sequences of the "expected" normal drawing, which allows to expose "unexpected" regions with anomalies.

Main outcome of current research is classifier model, capable of differentiating Parkinson's disease patients from healthy controls, providing prediction performance around 91%. Experimental part of the thesis offers technique for explaining individual predictions of obtained classifier by applying recently proposed machine learning meta-algorithm.

Present thesis is written in English and is 85 pages long, including 11 chapters, 16 tables and 19 figures.

Luria vaheldutavate seeriate tõlgendatavate anomaaliate ja kinemaatiliste parameetrite analüüs Parkinsoni tõbi modelleerimisel

Annotatsioon

Käesoleva töö põhieesmärk oli analüüsida Luria vahelduvate seeriate testide käigus joonistatud mustrid. Töö käigus arvutatakse tunnused mille alusel trennitakse masinõpe mudelid Parkinsoni tõbi diagnoosimiseks.

Psühholoogias ja neuroloogias Luria vahelduvate seeria testid kaustatakse patsiendi motoorika seisu hindamiseks. Testide tulemuste alusel uuritakse haiguse mõju liigutuse planeerimis- ja teostamis-funktsioonidele . Luria vahelduvate seeriate testid kuuluvad biomarkerite hulka, mida kasutatakse Parkinosni tõbi diagnoosimisel.

Töö põhipanuseks on uus meetod mustrite analüüsimiseks, joonistatud Luuria peenmotoorsete testide käigus. Pakutud meetodi iseloomustab joonistatud mustrite kirjeldamise viis. Nimelt, meetod võimaldab analüüsida joonistatud mustri erinevate detailiseerimise tasemetel. Selle saavutamiseks loodi unikaalne masinõpe algoritmite rakendamisjada. Esimeseks sammuks rakendatakse arvuti nägemise algoritm joonistatud mustri põhilelementide (loogiliste segmentide) tuvastamiseks. Selle alusel moodustatakse tunnuste jadad. Jadad kirjeldavad testide tulemused kinemaatiliste parameetrite keeles. Järgmiseks sammuks on tehis närvivõrgu treenimine mille abiga konstrueeritakse oodatud kinemaatiline portree mille alusel leitakse joonistamisanomaaliad.

Põhitulemuseks on klassifikaator mis võimaldab eristada patsiente Parkinsoni tõbiga kontroll rühmast täpsusega 91%. Samuti pakutud meetod annab võimaluse jälgida otsuse tegemist.

Töö on kirjutatud Inglise keeles ning sisaldab teksti 85 lehekuüljel, 11 peatükki, 16 tabelit ja 19 diagrammi.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Significant part of human population suffers from Parkinson's disease. Recent research confirms, that up to 800 people per 100 000 are affected, which brings Parkinson's disease to the list of most widely spread neurodegenerative disorders. Currently, there is no cure and we don't know particular causes of PD. While disease progresses over time, it severely reduces quality of life for the patient, therefore early diagnosis has obvious high importance.

Parkinson's disease is complex neurodegenerative disorder which mostly affects human motions. Patients demonstrate variety of symptoms such as tremor, rigidity and slowness in movements (bradykinesia) [1, 2, 3, 4]. Handwriting and drawing processes are complex fine motor activities, which require precise coordination of many muscles, hence these processes are mostly disrupted among Parkinson's disease patients. Recent studies [5, 6, 7] endorse drawing and handwriting — as a proven biomarker for Parkinson's disease.

While tablet and touch-screen technology is constantly evolving, numerous research studies present their digitized versions of miscellaneous handwriting tests, some of them investigate drawings of circle, star, spiral, clock. Others analyze sentences and character sequences.

Related literature [8, 9] confirms, that "Luria's alternating series fine motor tests" — is promising technique of disorder level evaluation in complex motion planning and execution processes during handwriting. Luria's tests are already being used in medical community among neurologists and psychologists for several years. From feature type perspective — kinematic features are most significant in distinguishing

between groups of Parkinson's disease patients and healthy controls [10, 4, 7].

Primary goal of present thesis is to conduct analysis of patterns drawn during Luria's alternating series tests, extract set of interpretable features, containing various kinematic and pressure parameters and develop machine learning model, capable of correct distinguishing between groups of healthy controls and Parkinson's disease patients.

In present thesis previous knowledge will be applied and extended, by providing a novel methodology for analyzing drawing patterns. Majority of solutions available in the literature are based either on the analysis of entire drawing or individual strokes. *First distinctive feature* of the proposed approach is that it allows to analyze patterns with respect to their logical structure and with any required level of detail.

Computer vision technique is used to split pattern into logical segments and organize unprocessed drawing data into tree-like graph structures. Based on this solution, feature sequences describing different kinematic properties of the drawing are constructed. During next stages, neural-network based models are used to generate feature sequences of the "expected" normal drawing, which allows to expose "unexpected" regions with anomalies. Anomaly detection is *second distinctive feature* of proposed methodology.

Main outcome of current research is machine learning model of classifier, capable of differentiating Parkinson's disease patients from healthy controls, providing prediction performance around 91%. Experimental part of the thesis describes *third distinctive feature* — technique for explaining individual predictions of obtained classifier by applying recently proposed "Local Interpretable Model-Agnostic Explanations" [11] meta-algorithm.

Thesis is organized as follows. *Chapter 1* consists of problem statement and analysis of related research. *Chapter 2* provides high overview of infrastructure and implementation. *Chapters 2 and 3* describe data pre-processing and clustering. *Chapter 4* defines feature extraction methodologies. *Chapter 5* explains anomaly detection process. *Chapter 7* evaluates feature statistical significance. *Chapter 8* describes classifier model creation methodologies. *Chapter 9* proposes experimental solution for explaining individual predictions of obtained classifier. *Chapters 10 and 11* offer discussion about acquired results and research strategies in future.

1.1 Luria's Tests Background

Human motions, during handwriting and drawing — are complex multilevel procedures according to Luria's research studies [12]. Each complex motion require several phases:

- *Motion Planning Phase* — consists of following sub-phases:
  - *General Planning* — during initial phase, approximate idea, reason, general representation of the movement are generated in human cortex, along with reason behind the movement and the general representation of how the movement should be executed.
  - *Motion Pattern Creation* — second phase involves detailed motion pattern generation according to general plan from previous phase. Motion pattern can also be described as series of actions ordered in time.
- *Motion Execution Phase*
  - *Motion Pattern Implementation* — series of actions are being implemented during third phase, when pattern is transformed into series of signals to spinal cord.

Luria's alternating series tests — is a proven technique to expose disorders on every phase of motion execution and planning during handwriting process. Test series is represented by combination of certain task types and multiple periodic patterns.

1.1.1 Task types

Luria alternating series tests are being executed on tablet with stylus. General idea of every test — to collect handwriting data from tested subject using certain kinds of tasks, listed as follows:

- *Trace Task* — implies tested individual to trace the drawing pattern with stylus exactly above displayed pattern reference image. Simplest task of the series and requires only motion execution phase without complex planning.
- *Copy Task* — implies tested individual to reproduce displayed reference pattern with a stylus. Usually reference pattern is displayed on the upper part of

screen area. Tested individual is being asked to draw equivalent pattern on a free area of a tablet display. Task requires both motion execution and planning. May cause difficulties even for healthy subjects, since reference drawing is shown aside, therefore possible borders of the pattern are not obvious.

- *Continue Task* — implies tested individual to continue and fully complete whole drawing pattern from few visible segments. Most difficult task, both motion execution and complex planning processes are required. May cause difficulties for healthy subjects, since reference drawing is not fully shown, possible borders of required pattern are not obvious.

1.1.2 Pattern Types

Possible subset of repetitive drawing *patterns*: *sinus* pattern, *'P'* pattern and *'PL'* pattern. Reasoning behind such naming, is because pattern silhouettes remind Greek letters *Pi* (Π) and *Lambda* (Λ). All patterns are presented on following Figure 1.1.

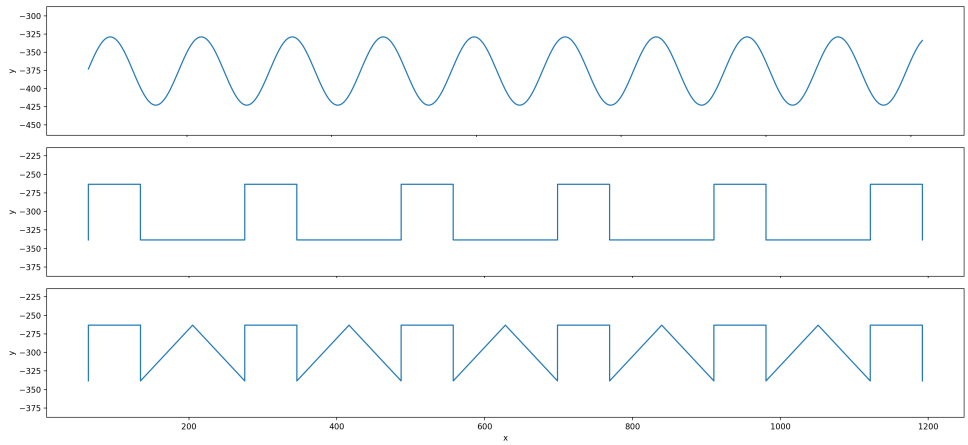

Figure 1.1: Luria pattern types — *sinus* pattern, *'P'* pattern, *'PL'* pattern

1.2 Problem statement

Statistical evidence suggests that 7 to 10 million people worldwide are living with Parkinson's disease, which makes it one of most widely spread neurodegenerative disorders. Current medicine doesn't offer complete cure, however early diagnosis may significantly improve quality of life for the patients. Several studies suggest [4, 7, 13], that handwriting might serve as biomarker for Parkinson's disease. One of the most promising handwriting evaluation technique — is "Luria's alternating series tests". Technique is already adopted by medical community and is being used among psychologists and neurologists for several years. Other various kinds of digitized drawing and handwriting tests are described and analyzed in the literature. However detailed and complete analysis of Luria's alternating series tests is still missing.

Present thesis is a part of bigger research series in Tallinn University of Technology, Tallinn University and University of Tartu, which investigates human handwriting and already consists of two master theses. First work offers quantitative analysis of kinematic features for Luria's tests and is clearly a foundation for present research [14]. Thesis introduces initial prototype of application for recording handwriting data. Similarly, second work [15] proposes next prototype of application and analyzes "Clock Drawing Test". Both share same methodology and conduct quantitative analysis of kinematic features, extracted from collected drawings.

Both theses miss essential part and don't offer any machine learning classifiers, which capable of differentiation Parkinson's disease patients from healthy controls. Therefore proposed solutions cannot be adopted by clinicians.

Other studies, however, do offer classification models [4, 16, 17, 18] with various accuracy rates. Majority of solutions available in the literature are based either on the analysis of entire drawing or individual strokes, namely — logical structure of the drawings is not taken into account, which is another possible gap. Additional important aspect should also be considered, which is trust. Among medical society black-box type machine learning systems are not considered trustworthy, therefore not widely used. Reasoning and decision tracing for each individual prediction is yet another area for research and development.

To wrap up above aspects, we can formulate following list of problems in previous and ongoing research:

- Detailed and complete analysis of "Luria's alternating series tests" technique is not present in current literature.
- Prevailing studies do not analyze separate logical parts of the drawing shapes.
- None of present studies offer anomaly detection and analysis within drawn shapes, which can be derived from previous point.
- Most of the proposed solutions lack of machine learning classification part, classification models are not being analyzed.
- Solutions with machine learning classifiers do not offer high accuracy rate.
- Clinicians do not trust machine learning systems, due to "black box" nature of underlying algorithms.
- None of the present studies offer prediction reasoning or decision tracing for Parkinson's disease classification.

Present thesis extends previous and ongoing research and offers solutions to aforestated problems, by providing *novel*, more detailed technique for drawing patterns analysis. Thesis objectives are formulated as follows:

- *Detailed analysis* of patterns drawn during *Luria's alternating series tests*
  - *Pre-processing* of drawing data, outlier and noise removal.
  - *Advanced clustering solution* for extraction of logical elements of the drawing patterns with arbitrary level of detail.
  - *Interpretable feature* extraction.
  - *Anomaly detection.*
  - *Statistical analysis* of obtained feature set.
  - Development of *machine learning classifier*, capable of precise differentiation between groups of healthy controls and Parkinson's disease patients.
- *Solution* for *prediction explanation* and decision tracing technique of obtained classifier model.

1.3 Related work

According to vast majority of research papers, most common Parkinson's disease specific handwriting and drawing impairment is "micrographia" [3, 19]. Which can be described as abnormal reduction in writing amplitude. Some authors [6, 10] even propose two types of micrographia. "Consistent" is uniform reduction in letter size, compared to writing before PD was diagnosed. Another type is "progressive" or in other words — inability to sustain normal size letters for some number of consecutive characters. Micrographia is easy to detect, since researchers deal with visible product of handwriting and size of letters and drawn objects can be assessed effortlessly, but does size is most relevant feature of PD handwriting?

With recent development of touch screens and tablets, researchers were able to accurately measure pen coordinates along with time, which gave possibility to distinguish new kinematic features of handwriting such as speed, duration, velocity, jerk, fluency. Same studies propose totally new term - "dysgraphya" [6] (from prefix "dys" in medical terminology - "impaired" or "disordered"), which easily describes motor aspects of the disease: tremor, akinesia (absence of power in movements), rigidity and slowness and their combinations with kinematic features.

Overall, from data acquisition methodology perspective, we can divide groups of researches into following main clusters. Graphic tablets (mostly preferred Intuos 4M Wacom), which can offer high precision tracking, 100Hz refresh rate also ability to record pressure. Recent studies [20, 15] successfully adopt *iPad* touch screen with *Apple Pencil* stylus and prove it more than capable of capturing drawing data precisely. Also it's worth mentioning, that *iPad* technology brings two novel measures, such as altitude and azimuth angles [15] of the *Apple Pencil*, which in theory, may give more useful features describing PD drawings. As for Wacom tablets - they offer exclusive possibility of tracking in-air movements of the pen, allowing to record data between strokes [21]. Aghanavesi et al. [18] even utilizes common smartphone screen for tapping and spiral drawing tests and successfully extract kinematic features, which contain relevant symptom information for detecting and assessing PD dexterity.

Also, we shouldn't forget forget common "pen and pencil" tests. Most of the

studies in the past were utilizing such method really well. In recent studies Raudmann et al. [22] uses pen and paper tests for writing simple and more complex sentences to investigate how PD handwriting alter from healthy controls (HC). Even multiple paper types, such as plain, horizontal or grid lined were analyzed. And research confirms, that writing of PD patients certainly differs from HC. Mainly micrographia and speed reduction were observed. As for different paper types - micrographia was less obvious, when patients performed test on lined paper, however writing speed didn't significantly improve.

As from test methodology perspective we can observe some variety of drawing tasks. Smits, Tolonen, Cluitmans, van Gils, Conway, Zietsma, Leenders, and Maurits [2] proposed set of standardized tasks, which include drawing of circle, star, spiral and writing of sentence and character sequence "elelel".

Drotár, Mekyska, Rektorová, Masarová, Smékal, and Faundez-Zanuy [7] conducted various researches [21, 4, 23] using set of handwriting tasks, which consists of single letters, bi-grams, tri-grams, single words, sentence and Archimedean spiral.

Letanneux et al. [6] also suggests, while choosing methodology, some certain aspects should be taken into consideration. The aim of the drawing tests are to focus on very low level of motor functions, writing some complicated sentences and unfamiliar words, also drawing complex shapes should be avoided, since it implies cognitive process.

Nõmm et al. [24] proposes digitalized version of "Poppelreuter's overlapping figures test", which was used in psychology and neurology for several decades to assess visual perceptual cortex function [25].

Nackaerts et al. [5] introduced "Systematic Screening of Handwriting Difficulties" test (SOS-test), where patients were tested two times within month period and were asked to copy as much as possible of a text within 5 minutes with the instruction to write as neatly and quickly as in daily life.

Korner et al. [26], Souillard-Mandar et al. [27], Brodaty and Moore [28] investigate CDT or "Clock Drawing Test", where participants are asked to draw the face of clock, mark in the hours and then draw the hands to indicate a specified time. CDT had proven its clinical validity of as a screening instrument for cognitive disorders, such as Alzheimer and Parkinson's disease or dementia and even recommended in

Denmark [26] as screening method for individuals applying for an extension of their driving licence after their 70th birthday.

Geometric and abstract shapes [24], including spiral [16] and Luria patterns [9] are superb source of kinematic features, which help to asses bradykinesia and tremor and overall dysgraphya. With sentence and character sequence writing micrographia-related features can easily be produced [2, 5].

Some research overviews performed by Pinto and Velay [10], Letanneux et al. [6] confirm, that majority of present studies distinguish most important features for PD handwriting as size, duration, speed and writing fluency (fluctuations in velocity, acceleration and jerk). Features, not related with micrographia are often call kinematic features.

Nonetheless we can meet some other feature types in various researches. Drotár et al. [4] proposes to utilize pressure measurements of the Wacom tablet, which were analyzed along with kinematic features, such as speed, duration and acceleration and showed significant discrimination power. Drotár et al. [21] also successfully adopted in-air measurements of the Wacom tablet pen, since it allows recording up to 10mm height, analyzing patients movements between individual strokes. Also, some very recent work [15] takes advantage of novel azimuth and latitude angles of iPad pencil for CDT test.

Zham, Kumar, Dabnichki, Poosapadi Arjunan, and Raghav [29] introduce "Composite Index of Speed and Pen-Pressure" or CISP and analyze its correlation with UPDRS (The Unified Parkinson's Disease rating scale) and shows, that CISP of spiral drawing is indeed strongly correlates with UPDRS.

It is really important to stress, that our thesis is a part of bigger research series in Tallinn University of Technology, which investigates human handwriting and drawing. Previous works include: "Quantitative analysis of the kinematic features for the Luria's alternating series test"[9, 14] and "Digital Clock Drawing Test Implementation and Analysis"[15]

First work conducted by Kozhenkina [14] is clearly a foundation for next studies. Thesis introduces first prototype of application for handwriting and drawing recording. Drawing data was acquired, and kinematic features were generated, quantitative analysis was performed to distinguish significant features, which separate healthy

individuals from PD patients. Similarly, Mašarov [15] introduces second prototype of application for recording another "Clock Drawing Test". Same approach was used and thesis outcome was also quantitative analysis and subset of statistically significant features extracted from drawing data. The main issue with aforementioned studies is that they miss essential part and don't offer any machine learning classifiers, which capable of distinguishing between healthy controls and Parkinson's disease patients. Therefore proposed solutions cannot be adopted by clinicians.

We can observe, that vast majority of researches produce and analyze features extracted from drawing data and most of them perform statistical quantitative analysis of each single feature. The problem with such approach - single features have low correlation coefficient with UPDRS or don't provide high discrimination power. To utilize multiple features at once and possibly find hidden relations between them, machine learning models come to the rescue.

Only some of recent researches [18, 17, 24] experiment with machine learning models and classify groups of individuals into PD or HC categories. For example Drotár et al. [4] achieved 81,3% classification accuracy using support vector machine SVM model, combining pressure and kinematic features with high discrimination power. The problem with current approach is that medical doctors cannot really utilize the above studies mainly because of the black-box nature of common machine learning models, such as neural networks NN, random forests RF or support vector machines SVM, even if they offer high accuracy rate.

To modify overall tendency of the medical community not to trust machine learning systems, we should somehow provide traceability of each individual classification result or prediction, even if our models offer high classification accuracy. In recent studies, Palczewska, Palczewski, Robinson, and Neagu [30] propose method for calculation of feature contributions for random forest RF models. It also allows for the determination of the influence of each feature on the model prediction for an individual prediction.

Ribeiro, Singh, and Guestrin [11] offer novel methodology, called "Local Interpretable Model-Agnostic Explanations" or LIME, which is innovative explanation technique that explains the predictions of any classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction.

Chapter 2

Implementation

2.1 Implementation overview

Proposed implementation consists of several stages, high overview and main aspects are specified in the following list:

1. Data pre-processing
    (a) Input data description
    (b) Removal of outliers
    (c) Creation of the Drawing Entity
2. Clustering
    (a) Shi-Tomasi algorithm
    (b) Edge entity
    (c) Improving Drawing Entity, building tree-like graph
3. Feature generation
    (a) Edge entity feature generation
    (b) Drawing entity feature generation
    (c) Creating sequences of features
4. Anomaly detection
    (a) Transformation of the sequences into training data for neural network
    (b) ”Sliding window” pre-processor component

   (c) LSTM neural network training

   (d) Anomaly detection process

   (e) Anomaly feature generation

5. Feature analysis

   (a) Statistical analysis of generated features

6. Classifier Training

   (a) Methodology overview

   (b) Classifier training and validation

   (c) Analysis of trained classifiers

7. Individual prediction explanations for obtained classifier with LIME algorithm

   (a) LIME background

   (b) LIME integration

During initial stage of data pre-processing — raw input data of iPad drawings will be transformed from transport JSON objects into Python dataframe format and wrapped into Drawing Entities. This approach mostly contributes to flexible and more effective storage and data organization. Additionally, in the intermediate stage of transformation, outliers will be detected and eliminated.

After pre-processing, Drawing will be clustered using combination of computer vision algorithm [31] and proposed heuristics for periodic pattern drawing data. New Edge Entity will be introduced to describe meaningful, self-explaining and consecutive clusters of the drawing patterns. Edge-type objects will be recursively generated and will form a tree-like graph of the whole Drawing object.

After completion of clustering process, Feature Generation stage will be initiated. Certain set of feature extraction methods will be embedded into Edge Entity and Drawing Entity to effectively compute required features in run-time. Since Drawing-object itself is ordered tree-like graph of Edge-objects, sequences of already computed features will be propagated and utilized during next stage.

We will introduce Anomaly detection algorithm, which should highlight abnormal elements in the sequence of a certain Edge-type feature. To archive this goal, we will transform 1-dimensional time-series sequence of data using Sliding Window

method into N-dimensional training dataset to predict next value based on N previous values. Certain type of neural network - LSTM will be trained to reproduce normal sequence of chosen feature, highlight abnormal elements in the sequence and yield collection of Anomaly entities for each Drawing object.

Feature Analysis will be performed to assess significance of generated feature set. We will split our dataset into control and PD populations, compute p-value, Spearman's correlation coefficient and Fisher score for Drawing-, Edge- and Anomaly-type features and will extract subset of features, which are potentially significant in differentiating PD subjects from control group and *vice versa*.

After determination of significant features, we are ready to build machine learning classifier model. Using K-fold algorithm, dataset will be divided into K consecutive folds consisting of training and validation data. In each fold several classifier models will be trained and analyzed. Accuracy, sensitivity and specificity will be measured for each model along with corresponding mean values for set of splits.

Finally, we will apply "Local Interpretable Model-Agnostic Explanations" LIME algorithm [11] and describe individual predictions within validation dataset in chosen fold, and will reveal hidden relations between features and individual prediction for chosen classifier model.

2.2 Infrastructure and Tools

To provide high overview of the infrastructure, software and hardware tools, I would split the the topic into three main clusters: client hardware and software tools, back-end server software, research and development related software.

2.2.1 Client Infrastructure and Tools

Hardware

Apple *iPad Pro* and *Apple Pencil* are crucial hardware components responsible for drawing data acquisition. Tablet has 26.77cm (10.5 inch) diagonal, LED-backlit Multi-Touch display with 2224 × 1668 pixels resolution with density of 264 ppi (pixels per inch). The *iPad Pro* scans the *Apple Pencil's* signal with resolution of 240 points per second, providing twice the data comparing to finger input.

Software

From software perspective — data was collected using custom iOS application, which itself was developed with Swift programming language and Xcode IDE (integrated development environment).

Initial implementation of aforementioned client software was initiated in scope of the course "Startup Project" ITX8549 in Tallinn University of Technology during winter semester in 2016. Author of current thesis was responsible for drawing algorithm and most of UI components of application.

Application saves acquired drawing data locally in JSON format, and also sends to remote back-end service, if internet connection is available.

2.2.2 Back-end Server Infrastructure

Software

Back-end service was developed using the popular lightweight Python web framework — Flask. Python library was used to provide REST API for saving drawing data and also to integrate web services with Amazon Simple Storage Service (S3),

which itself is file storage with high durability and web interface for data managing. All drawing data in JSON format is being synced to Amazon S3 bucket.

2.2.3 Research and development

Research, analysis and algorithm implementation was performed using Python programming language and PyCharm IDE. Following open-source Python libraries were extensively used on different stages of the algorithm:

- NumPy and Pandas — for data effective storage and manipulation
- Matplotlib — for bitmap conversion and figure plotting
- OpenCV — for computer vision algorithms
- SciPy — for signal processing, statistical analysis and feature generation
- Keras with TensorFlow back-end — for LSTM neural network training
- Scikit-learn — for training and validation of numerous classifier models
- Lime library — for explaining individual predictions of obtained classifier

2.3 Data acquisition

All tested individuals from PD and control groups were asked to complete a set of handwriting tests on iPad tablet, while seating in front of the table in comfortable position. All required meta information — time of the test, session id, test type — is recorded automatically. Anonymous subject identification number should be entered manually on initial stage of the test.

During drawing task, iPad registers all on-surface movements of Apple Pencil with resolution of 240 record samples per second. Each record is a vector-type object, with floating point number fields [x, y, t, p, a, l] and represents immediate state of the digital pencil, where:

- x, y — coordinates
- t — time stamp
- p — vertical pressure
- a, l — altitude and latitude angles

Each consecutive on-surface movement can be treated as separate stroke, therefore additional field, which represent stroke number is also saved. Drawing itself is substantially an array of pencil record samples, which on test-completion is transformed into JSON transport object and sent to remote REST service, if internet connection is available. Or stored locally otherwise.

Chapter 3

Data pre-processing

3.1 Data description

On initial stage of algorithm, we process list of JSON transport objects, which represents separate drawings. Each JSON file has specific name and collection of fields, filled with meta-information about corresponding drawing, such as *[patientId, session, time, type]* and drawing data, which is list of strokes, stroke is a list of points and point is aforementioned vector of *[x, y, t, p, a, l]*. Sample JSON file is shown in the following listing.

```
{ "session": "2CEE8303-8AD9-4D42-AFCA-0966DAA3E7FE",
  "type": "plcopy",
  "time": "2018-03-05 06:48:43 +0000",
  "patientId": "PD-15"
  "data": [
    [ { "x": 41.5938, "l": 0.96508, "a": 1.03885,
        "y": 213.914, "p": 0.33333, "t": 541925323.7092 },
      { "x": 52.4063, "l": 0.91419, "a": 2.18995,
        "y": 215.097, "p": 0.00001, "t": 541925324.55801 } ...] ...
  ]}
```

Table 3.1: Listing — Sample JSON drawing file

3.2 Drawing Entity

To effectively store and manipulate the data, we introduced Drawing Entity, which essentially is a wrapper object around existing JSON file with populated meta-information and drawing data transformed into single Pandas dataframe.

Field	Description	Type
name	Original name of the JSON file	String
testType	Type of the drawing test	String
patientType	Class of tested individual	String
dateTime	Timestamp when drawing was recorded	DateTime

Table 3.2: Drawing Entity — Metadata

Pandas dataframe is two-dimensional matrix with labeled columns and rows. Columns can potentially store different types of data. We can think of dataframe as SQL table or spreadsheet. On current stage of algorithm whole list of JSON files transformed into Drawing objects. Small transformation was necessary to perform: drawing JSON data was represented as list stroke, and stroke is a of list of points. Multiple array analysis seemed not much flexible and we flattened whole structure into single dataframe of points and populated additional column with i variable, representing stroke index of current point. Final dataframe structure and Drawing entity meta-information are described on Tables 3.3 and 3.2

Feature	Description	Unit	Type
x	x - coordinate	mm	Float
y	y - coordinate	mm	Float
t	time stamp	sec	Float
p	vertical pressure	abstract unit, range [0..6.0]	Float
a	altitude angle	degrees	Float
l	latitude angle	degrees	Float
i	stroke index	number	Integer

Table 3.3: Drawing Entity — Dataframe Structure

3.2.1 Outlier removal

An *outlier* is a separate observation, which is located relatively far from other observations. During intermediate process of *JSON* transport object conversion into *Drawing* entity, we apply *outlier* removal function to the *dataframe*.

To determine outliers, we use simple heuristics, that all Luria patterns tend to have horizontal direction and always limited in vertical range, therefore we should not have significant deviation by y-coordinate. We treat arbitrary data point p_i in vector of points $[p_1, p_2..., p_{n-1}, p_n]$ as an outlier, if scalar value y_i of vector of all y-coordinates $[y_1, y_2, ...y_{n-1}, y_n]$ is more than three standard deviations away from the vector $[y_1, y_2, ...y_{n-1}, y_n]$ average.

3.2.2 JSON to Drawing entity Conversion

Final transformation process of JSON transport object to *Drawing* entity is exposed on Figure 3.1

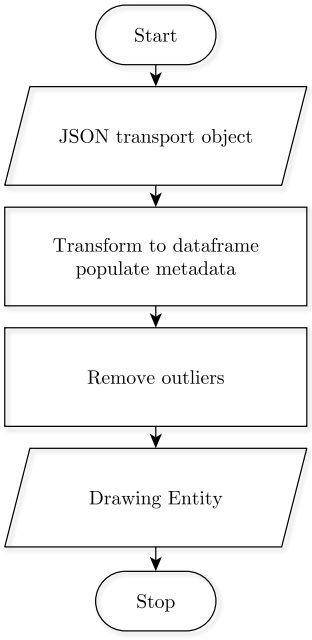

Figure 3.1: JSON to Drawing Entity Conversion — Flow Diagram

Chapter 4

Clustering

4.1 Requirements

During clustering phase, drawing data should be split into meaningful parts for subsequent feature generation, analysis and classifier creation. First obvious requirement for clustering algorithm comes from specific nature of existing dataset. We process hand-written Luria alternating tests, which essentially are continuous lines in two-dimensional space with repetitive elements, therefore could be treated as patterns. By definition, pattern is a finite sequence of elements with each element repeated in predictable manner.

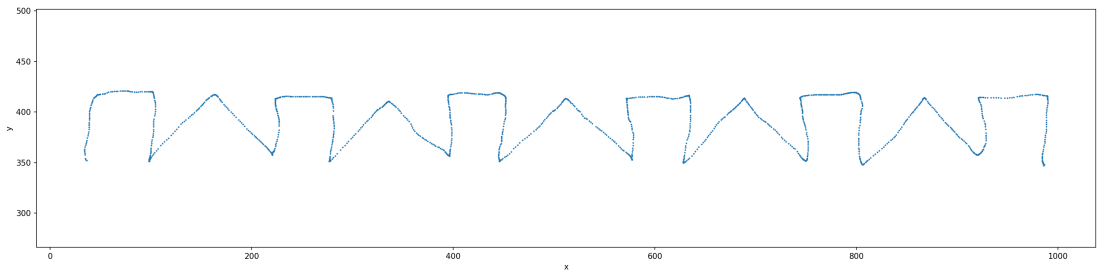

Figure 4.1: Luria Pattern — Single Sample

As shown on Figure 4.1, particular single sample of Luria 'PL' pattern consists of connected repetitive horizontal and vertical lines. Figure 4.2 however, shows multiple Luria 'PL' patterns combined into single two-dimensional figure. It is perfectly noticeable, what elements of the pattern positioned and scaled relatively to same elements from other drawing samples. Also same elements withing single sample may have slight variation in size and positioning.

Obviously, clustering algorithm should handle all these cases and process any two-dimensional continuous repetitive pattern. Cluster should represent smallest logical segment of the line. And in our particular case, smallest logical element would be straight line drawn at the same angle. For example in Luria 'PL' pattern on Figure 4.1 we could distinguish lines drawn at *[90, 0, -90, 45, -45]* degrees.

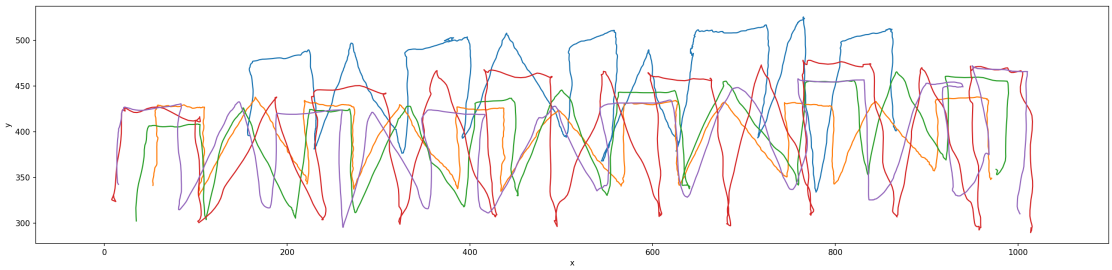

Figure 4.2: Luria Pattern — Multiple Samples Combined

Ideally, each cluster should be meaningful and human-interpretable, therefore should posses meta-information, describing ordering index or relative position within pattern sequence. To summarize all important aspects, we distinguished list of ideas and requirements:

- Algorithm should handle repetitive two-dimensional line-patterns
- Elements of the pattern may have:
  - Relative position
  - Relative scaling
- Clusters should be:
  - Consecutive
  - Meaningful
  - Traceable and interpretable
  - Self-describing

4.2 Standard Clustering Algorithms

It is worth mentioning, that set of most popular clustering algorithms was reviewed and investigated. However none of standard algorithms fulfilled aforementioned requirements and eventually was rejected for final solution.

- *K-Means* — requires pre-defined number of clusters K as input and also starts with a random choice of centroids (cluster centers) and therefore it can produce different results on different runs of the algorithm. Thus, the extracted clusters may not be repeatable and lack ordering and consistency.
- *Mean-Shift* [32] — does not require pre-defined number of clusters, it is sliding-window based algorithm, which tries to find dense areas of data points.
  Algorithm is not applicable for our task, since same Luria patterns may be drawn with different speed and acceleration, therefore density of the points will vary, which will affect clustering results.
- *DBSCAN [33] (Density-based spatial clustering of applications with noise)* — similarly to Mean-Shift, dense areas within drawing data do not represent separate elements of the pattern, algorithm will yield incorrect clusters.
- *Hierarchical Agglomerative Clustering* — algorithm tries to construct hierarchy of clusters. Each data point starts in its own cluster, and pairs of clusters are merged during algorithm execution. In order to decide which clusters should be merged a dissimilarity metric between sets of observations is essential. Most commonly used metrics are: *Euclidean, Manhattan, Maximum, Mahalanobis* distances.
  Choosing appropriate metrics for drawing patterns is not a trivial task. None of standard metrics offer good logic for splitting pattern into meaningful elements, since all lines are connected, as shown on Figure 4.1.

4.3 Corner Node Detection

As shown on Figure 4.1, smallest logical element of the pattern is straight line drawn at specific angle in the set of *[90, 0, -90, 45, -45]* degrees. Human eye can clearly see the border or connection points between the neighbour lines or so-called *corners*. Therefore it is essential to detect *corner nodes* to correctly split existing drawing patterns. Each drawing pattern apparently is a set of points in two-dimensional Descartes space, in other words — an image. Eventually, solution of aforementioned clustering task was found among classical computer vision algorithms.

4.3.1 Shi-Tomasi Algorithm

Shi-Tomasi method [31] is a classical computer vision algorithm invented in 1994. Algorithm is based on *Harris Corner Detector* method [34] with slight modification of scoring function.

OpenCV library for *Python* offers standard function *goodFeaturesToTrack()*, which is based on Shi-Tomasi method. Function input arguments are following:

- n - possible number of strongest corners
- q - quality level threshold, floating point number in range [0, 1]
- d - minimum Euclidean distance between corners

Function takes gray-scale image as input, and tries to finds n most significant corners with specified quality level q in the range of [0, 1]. Level q indicates minimum possible quality of the corner node. All corner-node-candidates below specified level q are rejected. Remaining corners are sorted in descending order based on quality score q. Neighbour corners with lower score and located closer, than minimum Euclidean distance d, are filtered out. Finally, function returns n top remaining corners of the image.

4.3.2 Shi-Tomasi Method Integration

Providing all required attributes for described corner detector function is feasible task. All drawing patterns have type and reference image, maximum number of corners is known number or can be determined experimentally. For example 'PL'

Luria pattern consists of 28 connected straight lines, therefore number of expected corners $n = 28 + 1 = 29$. Minimum Euclidean distance d between corners, as shown on Figure 4.1 is approximately $d \approx 50mm$. Same approach holds for finding quality threshold level q. Integration of *Shi-Tomasi* method into clustering workflow was be performed effortlessly with few additional pre-processing and post-processing steps. Full corner node detection workflow consists following phases:

- *Dataframe of drawing entity is converted into grayscale bitmap image* — required step for Shi-Tomasi algorithm. Conversion was performed with standard *plot()* function of *Matplotlib* library. All data points within one sample were connected with straight lines and saved into image in **.png* format with known resolution r (Figure 4.3).

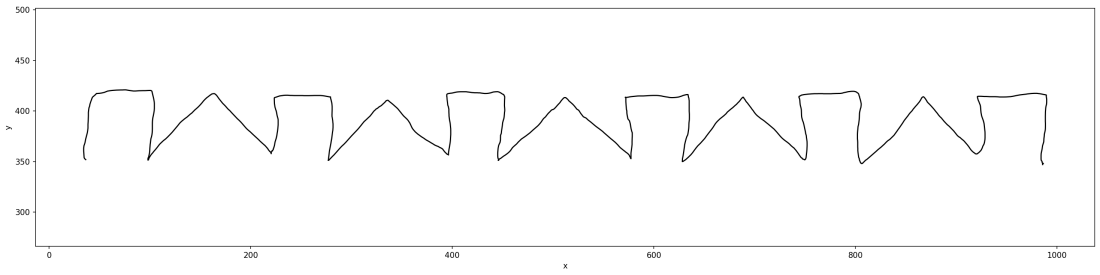

Figure 4.3: Clustering — Set of Points to Grayscale Image Conversion

- *Shi-Tomasi algorithm execution* — generated bitmap image and *[n, q, d]* input arguments are passed to *goodFeaturesToTrack()* function within *OpenCV* library. Output is similarly sized bitmap with corners as separate bits.
- *Bitmap to original coordinates conversion* — since output of Shi-Tomasi algorithm is bitmap of corners, it is required to perform conversion back to set of data points. It is non-trivial process, since during rasterisation, precision was lost and *[x, y]* coordinates were scaled according to bitmap size. However with known image resolution r it is possible to get *[x, y]* coordinates of the corners really close to existing data points.
- *Exact corner points lookup* — Since relatively precise coordinates of corner candidates are known, it is possible to choose closest points from the dataset. For that purpose special *KDtree* data-structure of *Scipy* library was generated from existing Drawing dataframe. It provides an index into a set of

any k-dimensional data points and can be used to rapidly lookup the nearest neighbors of any point.

- *Corner nodes are determined and saved as separate list in existing Drawing entity* — corner nodes are saved for subsequent clustering phases. Sample result is shown on Figure 4.4.

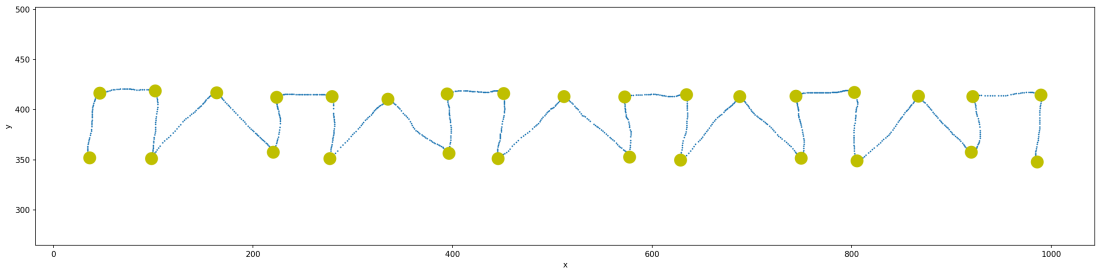

Figure 4.4: Clustering — Detected Corner Nodes

4.4 Edge Entity

After completion of previous phase, *Drawing* entity is complemented with ordered list of corners points (Figure 4.4). List of corner points is sorted by point timestamp t in ascending order. Each pair of consecutive corner points along with intermediate points represent a meaningful cluster of drawing pattern. To encapsulate data, meta-information and logic of a cluster, *Edge* entity was designed.

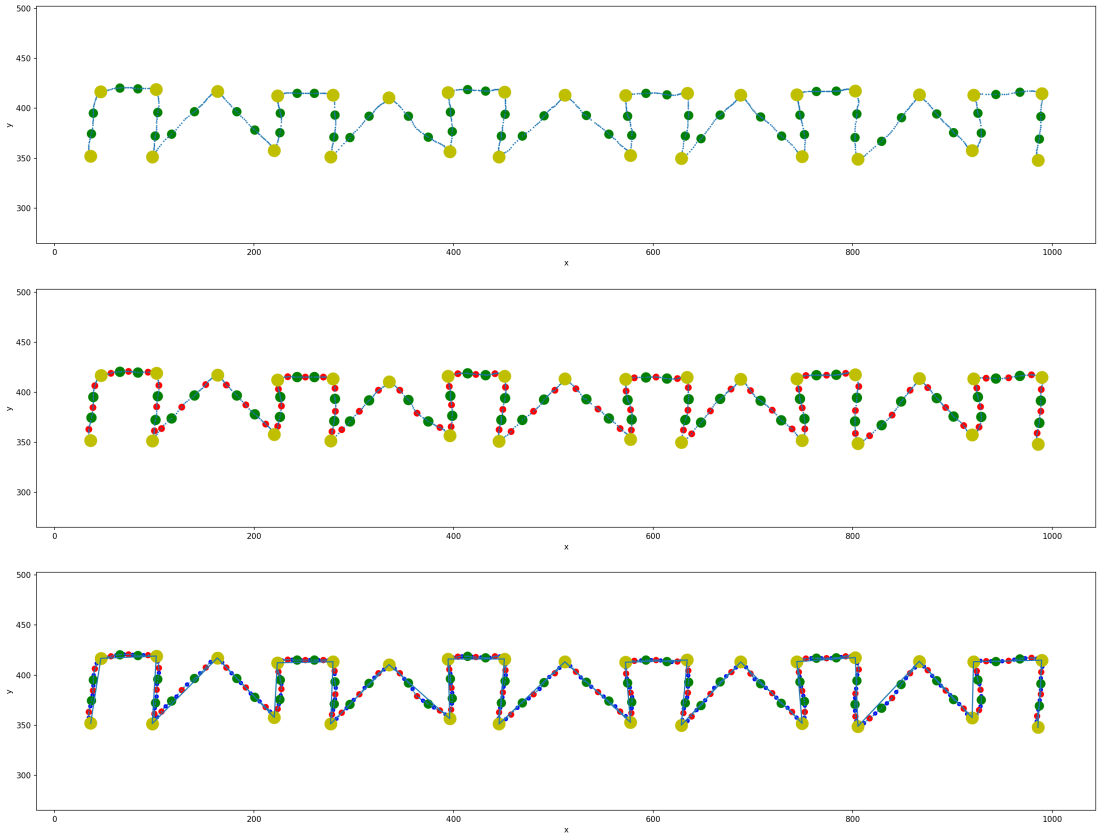

Figure 4.5: Clustering — Recursively Generated Edges with depth level $d = 1$, $d = 2$, $d = 3$

4.4.1 Edge Generation — from Corner Nodes

During first step of the algorithm, Edge objects created from consecutive pairs of neighbour corner points of the drawing. During creation of each Edge entity, we pass following arguments to constructor and store them as object fields:

- pair of corner points $[p_i, p_j]$

- index i — relative index of the corner pair
- vector of points $[p_i, p_{i+1}..., p_{j-1}, p_j]$ — limited by corresponding corner points, represents certain segment of the Drawing object

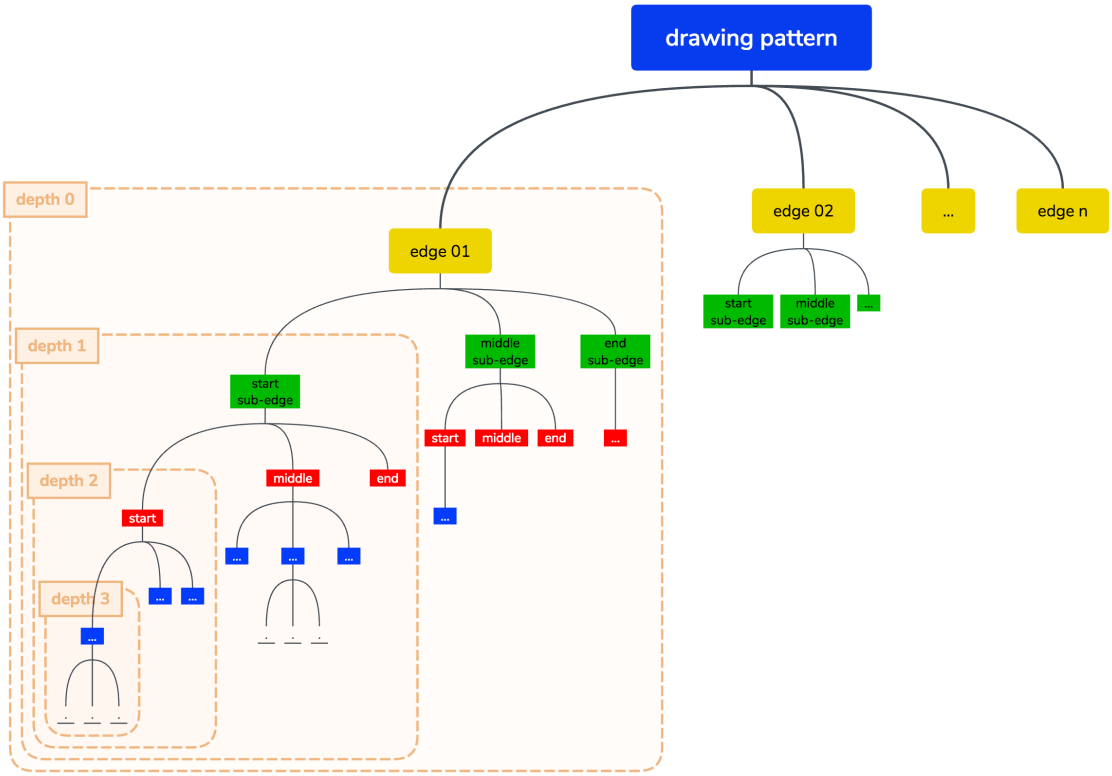

Figure 4.6: Clustering — *Drawing* object transformed into *tree-like graph* of *Edge* objects with depth level $d = 3$

4.4.2 Edge Generation — by Splitting Existing Edges

In theory, n Edge objects can be generated from $n + 1$ corner points. To improve precision, it is feasible to split each Edge object recursively into m Edge sub-objects.

Experimentally was determined best metrics for splitting strategy — *Euclidean distance* between points $[p1, p2]$. Another metrics, such as — *time interval* and *number of points* of a segment didn't produce acceptable results.

By intuition, edge represents a drawing stroke of the pattern and each stroke may have *[starting, ending]* or *[starting, middle, ending]* parts, therefore we can split Edge into $m = 2$ or $m = 3$ logical parts.

Process can be repeated recursively with required depth level (Figure 4.5) without losing logical meaning of generated sub-clusters. Given information about cur-

rent depth, local index and reference to parent Edge object and list of m references to child Edge objects, we can easily construct tree-like graph, which describes Drawing with any level of required precision (Figure 4.6).

4.4.3 Edge Entity — Fields

Edge object encapsulates cluster-related data and meta-information and possesses all required indices, references to build tree-like graph from related Drawing object. All stored fields of the Edge described in following Table 4.1:

Field	Description
index	index, relative to current depth level d
local_index	index, relative to other siblings
current_depth	depth level d of current Edge
starting_point	starting point p_i of the edge
ending_point	ending point p_j of the edge
dataframe	vector of points $[p_i, p_{i+1}..., p_{j-1}, p_j]$
parent_edge	single reference to parent *Edge* object
child_edges	list of references to child *Edge* objects
sibling_edges	list of references to sibling *Edge* objects

Table 4.1: Edge Entity — Encapsulated Data and Meta-information

4.5 Clustering Summary

Proposed clustering algorithm transforms raw data of the *Drawing* object into *tree-like graph* of *Edge* objects with required level of *detail*. All previously mentioned requirements for clustering solution were successfully fulfilled:

- Algorithm can handle two-dimensional line-patterns.
- Corner node detector is important element of algorithm and allows to extract relatively positioned and sized elements of the pattern for subsequent cluster generation.
- Cluster is represented by *Edge* object, which possesses following characteristics:
  - Consecutive:
    - *Edge* object store global and local indices.
  - Traceable and interpretable:
    - *Edge* object describes certain element of the pattern.
    - Can be easily transformed into named feature.
  - Self-explaining:
    - *Edge* is an element of tree-like *Drawing* graph.
    - *Edge* is aware about *parent*, *child* and *sibling Edges*.

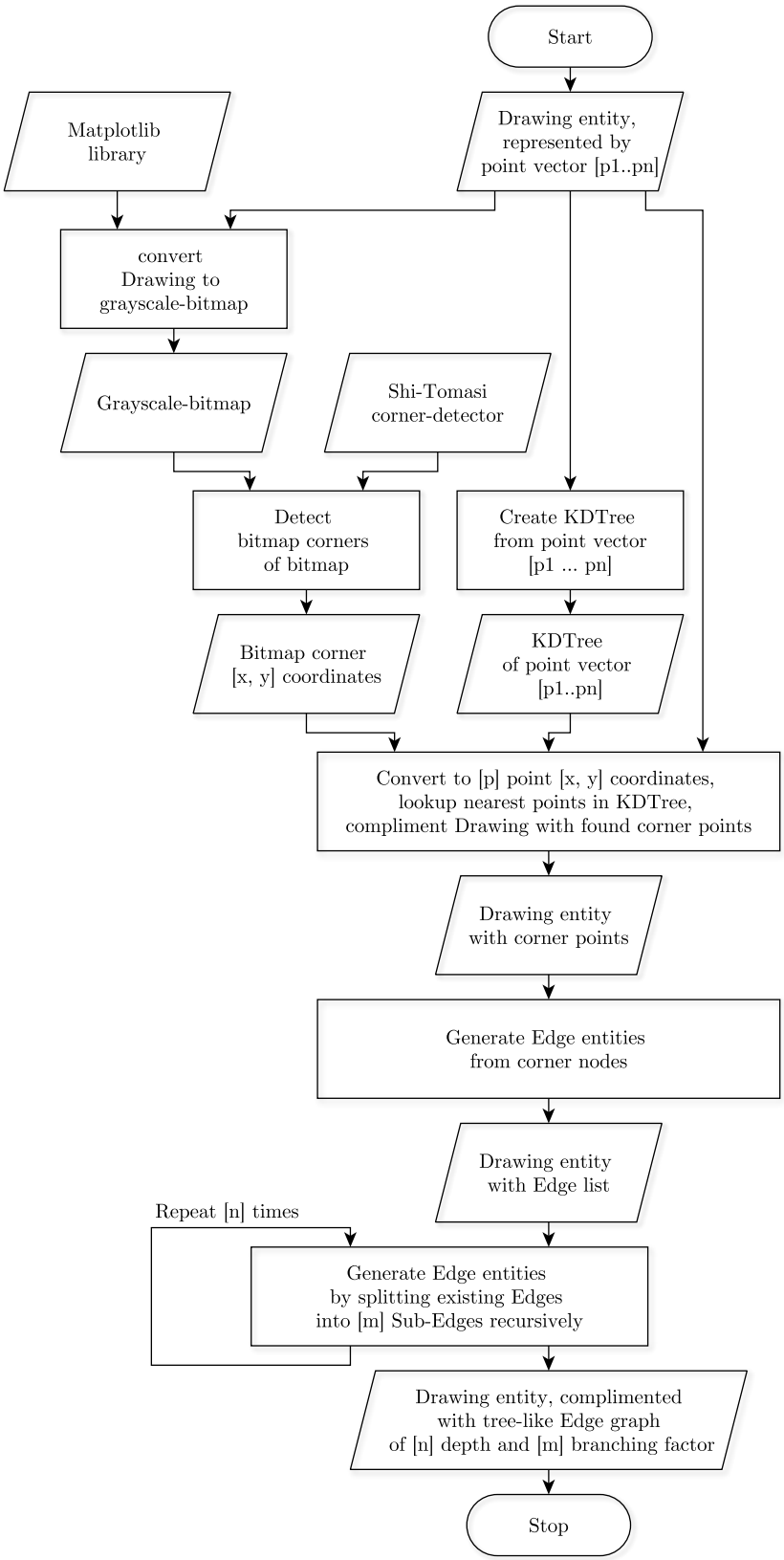

Figure 4.7: Clustering of the *Drawing* entity — Flow Diagram

Chapter 5

Features

5.1 Feature Generation

Main purpose of current implementation phase is to define and generate features, or individual measurable characteristics of the *Drawing* object or researched Luria pattern for subsequent statistical analysis and creation of classifier model.

Since discrimination power of particular feature is unknown, it is crucial to construct as many features as possible to produce accurate classifier. Features should also be informative, since it was planned to build individual prediction interpreter model using *"Local Interpretable Model-Agnostic Explanations" (LIME)* algorithm.

Following sections will describe aspects of technical implementation, methodology, feature indexing and naming, along with feature groups and individual features.

5.1.1 Methodology

It was decided from the scratch not to use straightforward naive approach for feature generation. Features are not generated and evaluated during separate iterative process, but described as computations over internal state of the entity (in our case — *Drawing* and *Edge* entities) in declarative manner using Python *@property* decorator and evaluated *lazily* in runtime on demand. *Lazy* evaluation is the computational strategy which postpones the evaluation of an expression until its value is actually required (non-strict evaluation).

Also *memoization* technique was applied. *Memoization* is an optimization strategy to speed up computations by storing the results of expensive function calls and

returning the cached result, when same inputs occur again. In our case — if feature evaluation happened, actual value of the feature is cached with possibility to re-trigger evaluation any time. All these aspects provide great computational flexibility and data integrity while working with massive amount of objects. Also it may give the ability to define potentially *infinite* tree-like graphs. Features can be added, changed or even deleted any time on entity level, which immediately affects all nodes of the whole graph.

5.1.2 Edge Naming and Indexing

Usually we are splitting each *Edge* into three *Edge* sub-objects *[1, 2, 3]*, which describe *[starting, middle, ending]* parts of the certain element of the pattern and also stored as local index of the edge. Given information of graph *depth*, local *index*, *parent* node from the *Edge* object, we can easily construct meaningful index and also deconstruct any index to human-understandable *Edge* description.

Edge Name	**Interpretation**
edge_2	Second stroke of the pattern
edge_2_1	Start of the second stroke of the pattern
edge_2_1_3	Ending of the start of the second stroke of the pattern
edge-2_1_3_2	Middle of the ending of the start of second stroke of the pattern

Table 5.1: Edge Entity — Naming and Indexing

5.1.3 Feature Engineering

Feature engineering is a process of application of function set to a set of existing features resulting in creation of new higher-order feature. It is possible to discriminate functions into three categories:

- Unary functions — functions, which take one argument, produce single feature from single feature. Sample unary functions would be — *trigonometric, square root, exponentiation.*

- Binary functions — functions, which take two arguments, produce single feature from two features. Sample binary functions would be all arithmetic functions — *sum, minus, multiplication.*
- Array functions — multiple-argument functions, produce single feature from array of features. For example — statistical functions *max, min, median*

In our case, *Drawing* object consists of *Edges*, *Edge* is an array of data points, data point itself is an array of *[x, y, t, p, l, a]* scalar values. So any array-like feature of the *Edge* or *Drawing* object can be transformed into higher-order numeric feature by applying statistical functions such as *[mean, median, min, max, standard deviation,* 1_{st} *percentile,* 99_{th} *percentile, ...]*. From *[x, y]* coordinates of data points and time *[t]* we can produce arrays of geometric and kinematic features, such as: *length, height, width, angle, duration, speed, acceleration, jerk* and similarly apply statistical functions to them and once more obtain unique higher-order features.

5.1.4 Feature Naming

Feature is always evaluated in the scope of certain *entity [Drawing, Edge]*, by applying certain *function*, therefore global *identification name* of feature is generated by concatenating three main components *[Entity name, Feature name, Function name]*:

- Entity name — *Edge* entity is named according to relative position in the graph. However *Drawing* is essentially a singleton in the current context.
- Feature name — concrete feature name.
- Function name — optional name, used in case of higher-order feature.

5.2 Feature Classes

5.2.1 Edge Features

Edge features are defined in the scope of corresponding *Edge* entity. In the context of Luria pattern, *Edge* is line-segment, represented by vector of points $[p_i, p_{i+1}, ...p_{j-1}, p_j]$ where $[p_i, p_j]$ are starting and ending points of the segment. Next Table 5.2 demonstrates subset of *Edge-related* features, final names are defined by adding variable prefixes to following feature names. Prefix is produced from corresponding name of the *Edge* instance.

Edge Feature	Description
angle	Angle of the line between points $[p_i, p_j]$
distance	Euclidean distance between points $[p_i, p_j]$
duration	Time interval between points $[p_i, p_j]$
speed	Linear speed between points $[p_i, p_j]$
pressure_median	Median pressure in vector of points $[p_i, p_{i+1}..., p_{j-1}, p_j]$
pressure_max	Max pressure in vector of points
pressure_min	Min pressure in vector of points
longitude_median	Median longitude of a pencil in vector of points
longitude_max	Max longitude of a pencil in vector of points
longitude_min	Min longitude of a pencil in vector of points
latitude_median	Median latitude of a pencil in vector of points
latitude_max	Max latitude of a pencil in vector of points
latitude_min	Min latitude of a pencil in vector of points

Table 5.2: Edge Features — Sample Subset

5.2.2 Drawing Features

Drawing-related features are defined and evaluated in the scope of corresponding *Drawing* entity. In the current context, *Drawing* is whole Luria pattern, which consists of line-segments, each represented by vector of points $[p_i, p_{i+1}, ...p_{j-1}, p_j]$

where $[p_i, p_j]$ are starting and ending points of the segment. So the whole *Drawing* is represented by vector of points $[p_1, p_2, ...p_{n-1}, p_n]$, where n is total number of points in the Drawing. Subsequent Table 5.3 demonstrates subset of *Drawing-related* features, final names are defined by adding prefix constant 'drawing' to following feature names.

Drawing Feature	Description
width	Horizontal distance of the *Drawing* $x_{max} - x_{min}$
height	Vertical distance of the *Drawing* $y_{max} - y_{min}$
area	*Drawing height* multiplied by *width*
number_of_strokes	Number of strokes. Corresponds to maximum stroke *index* in vector of points $[p_1, p_2, ...p_{n-1}, p_n]$
number_of_edges	Number of *Edge* elements in the Drawing, calculated from number of *Edges* in graph on depth level $d = 0$
completeness	Ratio between *actual* and *expected* number of *Edge* elements in the *Drawing* pattern. Expected number of *Edges* is Luria pattern-type related *constant*
angle_upper_regression_line	Angle of *upper* regression line
angle_lower_regression_line	Angle of *lower* regression line
angle_middle_regression_line	Angle of *middle* regression line
angle_upper_lower	Angle between *upper* and *lower* regression lines of the *Drawing*

Table 5.3: Drawing Features — Sample Subset

To define asymmetry measure of particular Luria pattern, it was decided to construct regression lines from *[upper, middle, lower]* corner nodes of the Drawing and evaluate angle values of corresponding lines. Classification task of extraction *[upper, middle, lower]* nodes, was solved by creating linear regression model from all existing corner nodes, which represents *middle regression line*. During next step, each corner node y-coordinate was compared to y-coordinate of middle regression

line, so all corner nodes above *middle regression line* were classified as upper-type. Or lower-type in the opposite case. Similarly, u*pper and lower linear regression lines* were constructed and angle features evaluated. Result is illustrated on Figure 5.1.

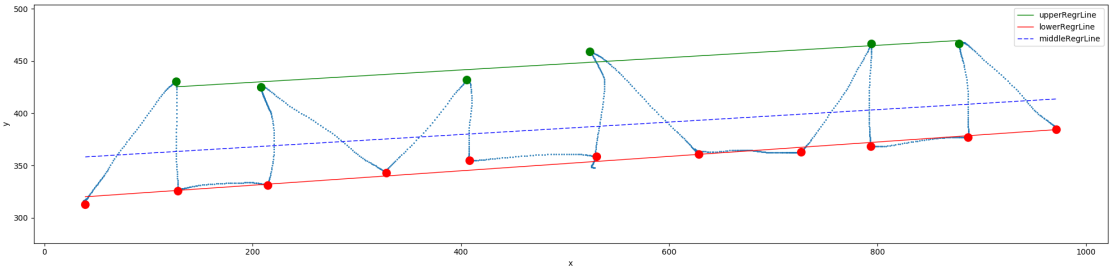

Figure 5.1: Drawing Features — *Upper, middle, lower regression lines*

5.2.3 Kinematic and Pressure Features

Kinematic and *pressure* characteristics of handwriting are described in various recent research papers of Zham et al. [29], San Luciano et al. [16], Nõmm et al. [9] and have proven high level of discrimination power between Parkinson's disease patients and healthy subjects. Kinematic and *pressure* feature set describes arbitrary vector of points $[p_1, p_2, ...p_{n-1}, p_n]$, therefore can be defined and evaluated in the scope of both *Drawing* and *Edge* entities. *Velocity, acceleration, jerk* and *pressure* higher-order features are described and used in subsequent analysis and classifier creation.

Along with standard *[median, mean, mass]* features, another higher-order feature was proposed — *number of changes*. To evaluate *number of changes*, standard function *argrelextrema()* of *Scipy* library was applied, which is sliding-window based function and yields local *extrema* of the arbitrary array. In our case — number of *extrema* points within feature vector corresponds to *number of changes* of particular *kinematic* or *pressure* feature.

Following Table 5.4 denotes subset of *kinematic* and *pressure* features. Final names are generated by adding variable prefixes to feature names. Prefix is produced from corresponding name of the *Edge* or *Drawing* instance.

Feature	Description
duration	Time period between first and last point of vector $[p_1, p_2, ...p_{n-1}, p_n]$
trajectory_length	Sum of all Euclidean distances between all neighbour points in vector
velocity_mass	Velocity mass of the point vector $[p_1, p_2, ...p_{n-1}, p_n]$
velocity_mean	Average velocity of the point vector
velocity_nc	Number of velocity changes in point vector
acceleration_mass	Acceleration mass of the point vector
acceleration_mean	Average acceleration of the point vector
acceleration_nc	Number of acceleration changes in point vector
jerk_mass	Jerk mass of the point vector
jerk_mean	Average jerk of the point vector
jerk_nc	Number of changes in jerk of point vector
pressure_diff_mean	Average difference in pressure between neighbour points
pressure_mass	Pressure mass of the point vector
pressure_nc	Number of changes in pressure

Table 5.4: Kinematic and Pressure Features — Sample Subset

Chapter 6

Anomaly Detection

6.1 Overview

Motivation behind proposed anomaly detection solution based on following assumptions. Current research subject is essentially a Luria pattern drawing. Pattern itself is sequence of repetitive groups of elements. It is expected, that tested individuals, while drawing a pattern, may produce mistakes. Healthy subjects, by intuition, should produce less mistakes, than subjects with *PD*, therefore mistake detection may have potential in Parkinson's disease prediction.

Mistakes can be interpreted as anomalies. An anomaly, by definition, is an element in the sequence with significant deviation from its expected value [35]. Expected value can be obtained from average value of similarly positioned elements of the sequence. Another option — is to obtain expected value from sequential model prediction.

Upper chart on Figure 6.1 illustrates *normal* behaviour of the *angle* sequence from the subset of *normal Drawings*. Lower chart is subset of *Drawings* with *anomalies*. Similarity between different sequences within *normal* subset is obvious, therefore it is possible to construct sequential neural network NN model, capable of reproducing comparable signal.

Proposed *anomaly detection* solution includes following important steps:

- *Sequence Extraction* — logic for *sequence* extraction from Drawing entity. *Sequence* essentially is vector of separate numeric feature x of the *Edge* entity of the Drawing object. Sequence is represented by vector $[x_1, x_2...x_n]$

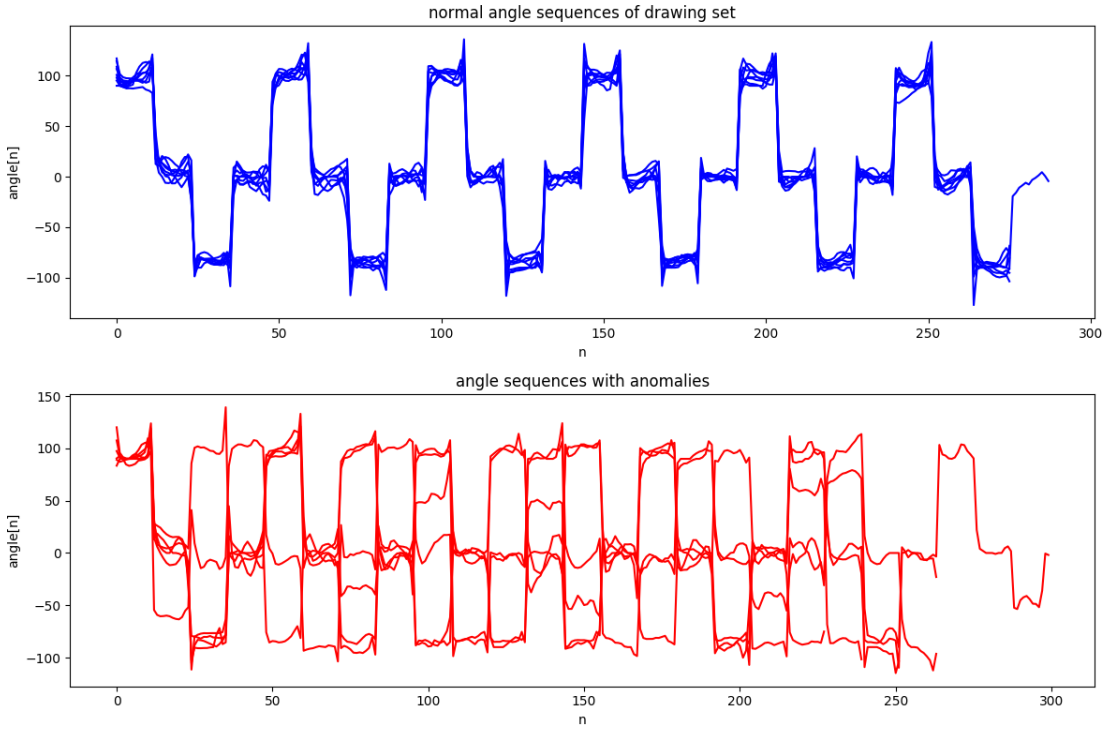

Figure 6.1: Combined Sequences of 'Angle' Feature — *Normal* Subset (Upper Chart) and Subset with *Anomalies* (Lower Chart)

- *Neural Network Training* — neural networks NN have proven high efficiency in sequential data processing. NN model should be able to reproduce normal sequence of feature x with low error rate.
- *Anomaly Detection* — process consists from two sub-processes:
  - *Sequence Prediction* — predict sequence $[y_1, y_2...y_n]$ from existing sequence $[x_1, x_2...x_n]$ of arbitrary Drawing feature x.
  - *Error Evaluation* — Compare original sequence $[x_1, x_2...x_n]$ with predicted $[y_1, y_2...y_n]$, evaluate elements with significant difference, generate *anomaly* objects.
- *Anomaly Feature Generation* — generate numeric features from generated *anomalies* for subsequent analysis.

6.2 Important Components

6.2.1 Sequence Extractor

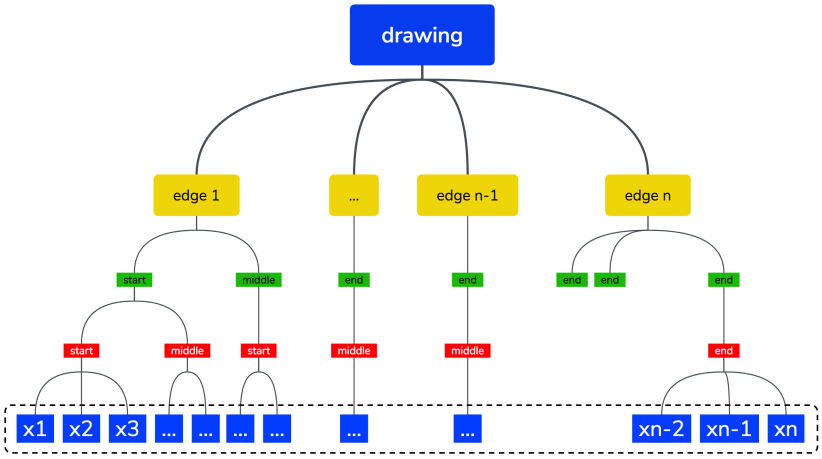

Figure 6.2: Sequence of arbitrary feature x extracted from *Edge* graph at depth $d = 4$ represented by vector $[x_1, x_2...x_n]$

After previous phases of algorithm, we've already acquired set of Drawing entities with well structured tree-like graphs of *Edge* objects. Each *Edge* object already carries meta-information about position inside graph, represented by depth d and local index i along with evaluated set of numeric features. Current architecture, showed on Figure 6.2, allows us to easily transform Drawing entity into set of feature sequences by extracting n *Edge* objects at certain *depth* d and join arbitrary *Edge* feature x values into *sequence*, represented by vector of $[x_1, x_2...x_n]$

Current component is being applied during sequential model training and sequence prediction with slight variations:

- *For Sequence Prediction* — extracts singular sequence of arbitrary feature x from singular Drawing and outputs vector of $[x_1, x_2...x_n]$, where n – is number of *Edges* at deepest available depth $d = 4$ of Edge graph.
- *For Model Training* — during fist step, extracts vector of sequences $[X_1, X_2, ...X_m]$ from subset of Drawings selected for model training. Current structure (6.1) is essentially a matrix of numeric representation of x arbitrary feature, where m – is number of Drawings in subset, n – is number of *Edges* at depth $d = 4$

of Edge graph of the Drawings. Throughout next step, we *flat map* structure (6.1) into vector of $[x_{11}, ...x_{mn}]$ producing long connected feature-sequence of all Drawings of the subset.

$$[X_1, X_2, ...X_m] = [[x_{11}, ...x_{1n}], [x_{21}, ...x_{2n}], ..[x_{m1}, ...x_{mn}]] \tag{6.1}$$

6.2.2 LSTM Neural Network

It was decided to utilize "Long Short-Term Memory" [36] or LSTM neural network for sequential data modelling and prediction. LSTMs are a very promising solution to sequence and time series related problems [37]. LSTM network was firstly proposed in 1997 by the German researcher Sepp Hochreiter [36] as a answer to gradient decay problem in traditional recurrent neural networks. LSTM neural networks in essence — are subset of regular *recurrent neural networks* or *RNNs*.

LSTM neural network propose novel *gating unit mechanism* that manages network memory cell access. *Gate unit* can block other network parts from modifying contents of the memory cells for multiple time steps, therefore LSTM networks preserve signals and propagate errors for significantly longer than traditional *RNNs*. In other words, LSTM is capable of selectively remembering patterns for extensive periods of time.

Current solution utilizes *Keras* library, which is high-level neural networks API, written in *Python* and capable of running on top of *TensorFlow* back-end.

The LSTM network processes sequences of input signals in the form of vector of tuples $[(a_1, b_1), ...(a_n, b_n)]$. For every tuple (a_i, b_i) the LSTM network takes the new input a_i and produces an estimate for the target b_i given all the previous inputs $[a_1, ...a_{i-1}]$.

Following architecture was applied during training: standard *Sequential* model was used, as a linear stack of layers with three *LSTM hidden layers* with number of neurons: *'hidden1': 64, 'hidden2': 256, 'hidden3': 100*, along with intermediate *Dropout* layers with fraction of the input units drop set to 0.2. Dropout helps to prevent overfilling of the model, by randomly setting a fraction of input units to 0 at each update during training phase. Also *activation* function was set to *linear* and loss function to *mean squared error* or *MSE*, which is adequate combination

for current regression task. However all mentioned hyper-parameters were defined experimentally during algorithm implementation.

6.2.3 Sliding-Window Pre-Processor

Current component *Sliding-Window Pre-Processor* is required for LSTM input generation, hence is extensively used during model training and sequence predictions.

LSTM network model takes input signals in the form of vector of tuples $[(a_1, b_1), ... (a_n, b_n)]$. For every tuple (a_i, b_i), LSTM network takes the input a_i and produces an estimate for the target b_i.

By applying *Sliding-Window* method to existing sequence $[x_1, ...x_m]$, we will get required structure of tuples $[(a_1, b_1), ...(a_m, b_m)]$. Where a_i itself is a vector with n values $a_i = [x_{i-n}, ...x_{i-1}, x_i]$, and b_i is single value $b_i = x_{i+1}$. Parameter n – size of the window, m – number elements of the existing sequence. In other words – we are trying to predict x_{i+1} next value in the sequence by supplying LSTM model with $[x_{i-n}, ...x_{i-1}, x_i]$ previous n values.

6.2.4 Auxiliary Entities

Following auxiliary entities will be mentioned in subsequent processes.

- *Drawing Container* — after previous phases of the algorithm, set of Drawings was already transformed from JSON transport objects. Clustering and feature extraction processes were executed. To effectively manage collection of existing Drawing objects, *Drawing Container* entity is introduced. It allows to apply logic, required for subsequent model building and analysis: sequence generation, training data pre-processing, training and test data splitting, feature analysis and classifier model generation.
- *LSTM Container* — Single *LSTM model* is used to reproduce sequence of separate feature x of the Drawing. Current solution examines subset of n different features of the Drawing, hence total number of n *LSTM models* were trained and stored in *LSTM Container* entity, which substantially is a wrapper-object over a collection of the neural network models with preserved meta-information and required logic.

6.3 LSTM Model Training

Current chapter describes process of training singular LSTM neural network for arbitrary feature x sequence modelling. Process will be repeated for every feature in the subset of pre-defined features. Trained model will be enclosed in the *LSTM Container entity* for subsequent *anomaly detection*. Whole process is illustrated on flow diagram on Figure 6.3.

- *Sequence Extraction* — certain subset of experimentally defined 'normal' *Drawings* from healthy individuals moved into isolated list of $[d_1, d_2, ...d_n]$ and wrapped into *LSTM Container* object. This subset of Drawings will not be used in any consecutive classifier modelling. With *Sequence Extractor* component, obtain vector of sequences $[[x_{11}, ...x_{1n}], [x_{21}, ...x_{2n}], ..[x_{m1}, ...x_{mn}]]$ from current subset of Drawings, then *flat map* into vector of $[x_1, ...x_m]$ where m – number of all *Edges* at certain depth of the *Edge* graph of all *Drawings* in $[d_1, d_2, ...d_n]$ list.
- *Sequence Splitting* — divide long sequence of $[x_1, ...x_m]$ into two sub-sequences $[x_1, ...x_j]$ and $[x_{j+1}, ...x_m]$ for model training and testing. Index j determined by $j = round(0.7 * m)$, where fraction 0.7 – is training rate and m - number of elements in the sequence.
- *LSTM Input Generation* — with *Sliding-Window Pre-Processor* component transform training $[x_1, ...x_j]$ and testing $[x_{j+1}, ...x_m]$ sequences into n-dimensional data in the form of tuple vectors $[(a_1, b_1), ...(a_m, b_m)]$
- *LSTM Training* — with *Keras* library compile *Sequential LSTM model* with aforementioned *hyper-parameters*, set number of *epochs* to 10. Fit model with train-input data obtained from *Sliding-Window Pre-Processor* component. On training completion measure model accuracy with previously generated test-input. Save model into *LSTM Container*, preserving required meta-data.

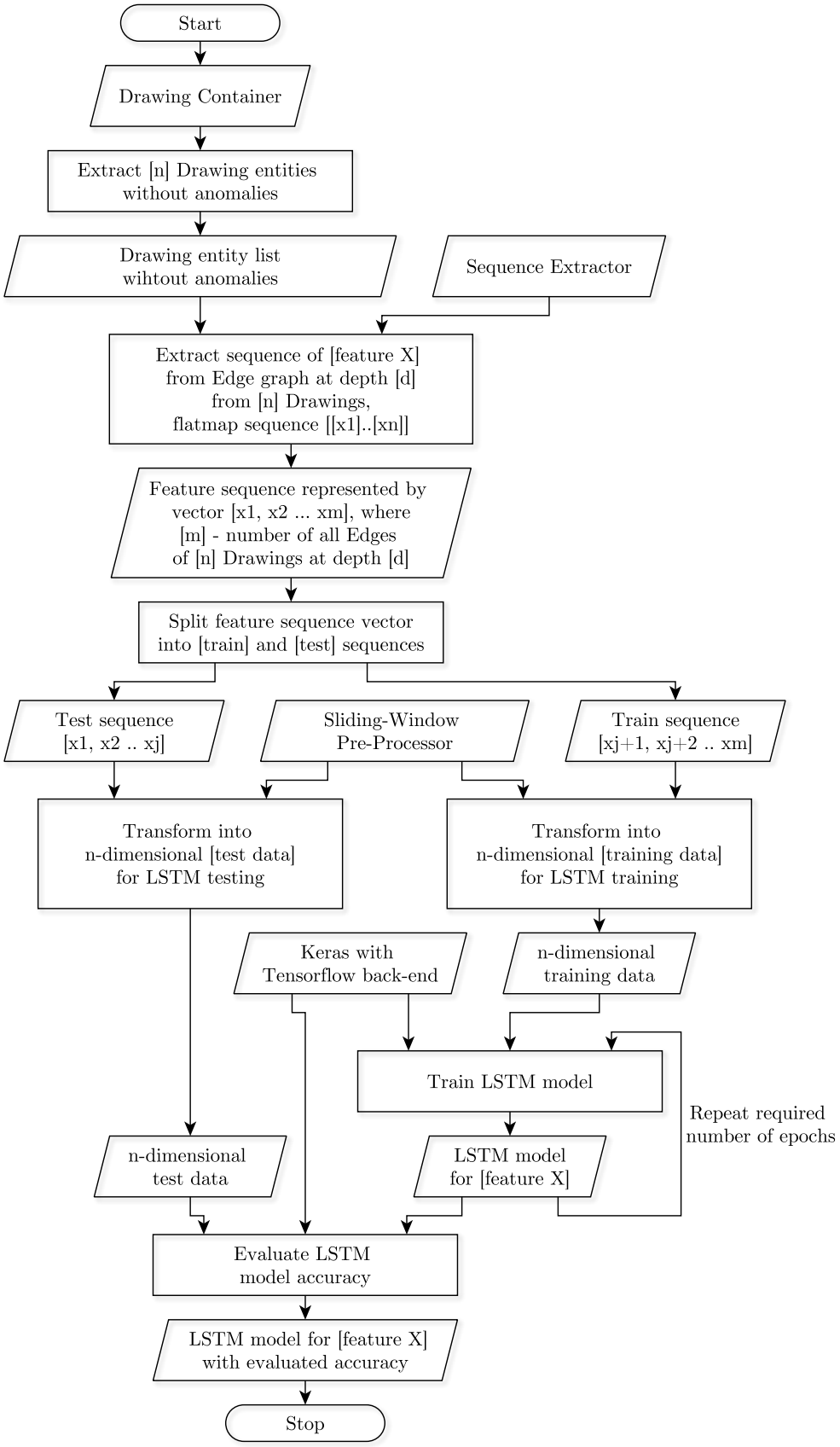

Figure 6.3: *LSTM Model* Training — Flow Diagram

6.4 Anomaly Detection

By definition, anomaly detection process usually refers to the problem of finding sequences in data, that do not conform to expected behavior.

Current algorithm utilizes most common Neural-Networks-Based *anomaly detection* method, described by Chandola, Banerjee, and Kumar [35]. Main idea is to initially predict a sequence from its previous samples and then decide whether the element of sequence is *anomaly* or *normal*, based on prediction *error*, or in other words — an *anomaly* is an element, which cannot be predicted from the past normal time-series or sequential data using trained *NN* model.

First part of the method — is already implemented: neural network, in current context — *LSTM* model is trained on the *normal* sequential dataset to learn the *normal* behaviour of the sequence, or to *accurately* predict x_{i+1} next value of the sequence from $[x_{i-n}, ...x_{i-1}, x_i]$ previous n values, if *normal*, i.e., *non-anomaly* input were supplied.

Second part of the method — is described in current chapter: each test instance, in our case — feature-sequence extracted from the *Drawing* entity, is provided as an input to the LSTM neural network model. If model predicts each next element of the sequence with error below certain *threshold*, this particular element is treated as *normal*. If measured error is above pre-defined *threshold* — element doesn't conform to expected behaviour of the sequence, hence treated as *anomaly*.

6.4.1 Anomaly Entity

In current context, *anomaly* — is an element of a feature-sequence, i.e, certain feature instance of the certain *Edge* of the Drawing object. Intuitively, *anomaly* refers to both *feature* and *Edge* instances, thus it was decided to design *Anomaly* entity in order to wrap *meta-information* about feature name, predicted and actual values, also to establish a link to related *Edge* object by storing its *reference*. Current design approach takes advantage of already existing flexible data-structure of Drawing graph by decorating its *Edge* elements with extracted *Anomaly* objects.

6.4.2 Anomaly Detection Process

Certain subset of n feature-types was already pre-defined, and n neural network *LSTM models* were trained to predict corresponding feature-sequences. Similarly, current anomaly detection process is executed for single Drawing object n times for each pre-defined feature-sequence and optionally produces Anomaly objects, which are stored in graph data-structure of Drawing entity for successive Anomaly-type feature generation and analysis. Process is illustrated on flow diagram on Figure 6.4 and consists of following steps:

- *Sequence Extraction* — with *Sequence Extractor* component, obtain sequence for particular feature x of Drawing instance. Sequence is represented by vector of $[x_1, x_2...x_n]$, where n – is number of *Edges* at deepest available depth $d = 4$ of Edge graph.
- *LSTM Input Generation* — with *Sliding-Window Pre-Processor* component transform training $[x_1, x_2...x_n]$ sequence into n-dimensional data in the form of tuple vectors $[(a_1, b_1), ..., (a_i, b_i), ...(a_m, b_m)]$, where a_i itself is a vector with n values $a_i = [x_{i-n}, ...x_{i-1}, x_i]$, and b_i is singular value $b_i = x_{i+1}$.
- *Sequence Prediction* — from *LSTM Container* extract required *LSTM model*, trained for feature x sequence prediction. Feed generated n-dimensional data from *Sliding-Window Pre-Processor* to *LSTM model*, obtain predicted sequence, represented by vector of $[y_1, y_2...y_n]$.
- *Error evaluation* — subtract predicted sequence, represented by vector $[y_1, y_2...y_n]$ from original vector $[x_1, x_2...x_n]$ and apply *square* function. Obtain vector of errors $[e_1, e_2...e_n]$, so each element is *squared standard error* $(SE)^2$ for original element x_i of the sequence. Generated $[e_1, e_2...e_n]$ may include outlier-peaks (shown on third chart of the Figure 6.5). Peaks are removed by applying *Scipy* library standard *median* filter function. Finally, locate anomalies — elements with error e_i above pre-defined threshold t, by retrieving corresponding indices of the sequence.
  - *Threshold calculation logic* — threshold was calculated separately for each *LSTM model*, during training process: feed model with *normal* test data $[a_1, a_2, ...a_n]$, obtain predicted sequence $[\hat{a_1}, \hat{a_2}, ...\hat{a_n}]$, evaluate

mean squared error MSE with (6.2). Required threshold is calculated by $t = C \times MSE$, where hyper-parameter C is defined for each model experimentally.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(a_i - \hat{a}_i)^2 \tag{6.2}$$

- *Anomaly Generation* — after previous step, indices of anomalies $[i_1, i_2, ...i_m]$ within sequence $[y_1, y_2...y_n]$ are defined and correspond to *Edge* object local indices at specified depth $d = 4$ of the *Drawing* graph. Look-up and obtain *Edge* entities by corresponding i index. Wrap meta-information about feature *type*, actual value x_i and predicted value y_i along with *reference* to corresponding *Edge* object by constructing *Anomaly* entities.

6.4.3 Anomaly Features

It was pre-defined subset of n feature-types $[t_1, t_2, ...t_n]$, extracted n feature-sequences and trained n *LSTM models* to evaluate n anomaly-types. Current work analyzes $n = 6$ feature-sequences *[angle, pressure, length, duration, longitude, latitude]* of the Edge, i.e., sequences of mean values of *[angle, pressure, length, duration, longitude, latitude]* within *Edge* objects of *Drawing* graph at depth $d = 4$.

It is feasible to recursively traverse whole graph and evaluate number of anomalies for each node of the graph at arbitrary depth level d. Feature extraction algorithm follows simple logic:

- Obtain all *Edges* objects $[e_1, e_2, ...e_m]$ of the *Drawing* graph at depth d
- Recursively *count* all anomalies for each of $[e_1, e_2, ...e_m]$ and feature-type $[t_1, t_2, ...t_n]$.
- Evaluate output for depth $d = 0$, $d = 1$
- Construct feature names by combining prefix and suffix. *Prefix* – is taken from corresponding entity name with possible values of *[Drawing, Edge_1, Edge_2, ...Edge_n]. Suffix* – is matching name of the feature type for current anomaly – *[angle, pressure, length, duration, longitude, latitude].*

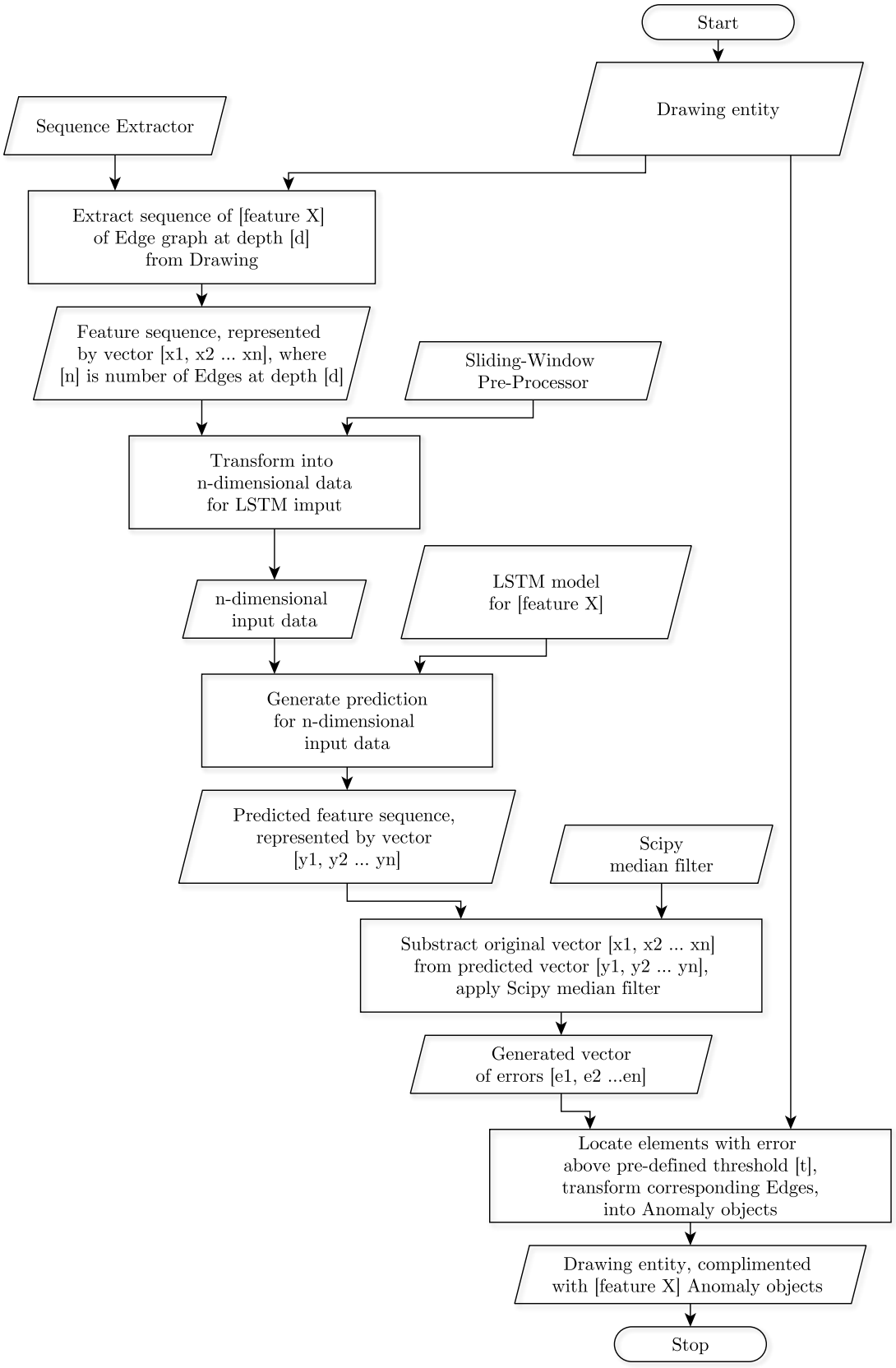

Figure 6.4: Anomaly Detection Process — Flow Diagram

6.5 Anomaly Detection Example

Following Figure 6.5 shows example output of proposed *anomaly detection* algorithm. Fourth chart represents Luria 'P' pattern in *Cartesian* coordinates, drawn with *mistakes*. Luria 'P' pattern by definition, should only contain vertical and horizontal lines. So only segments with angles with slight variation around values in subset of [90, 0, -90] are allowed. Pattern segments drawn at unexpected *angle* (edges 2 and 13) with unexpected *order* (edge 20) considered as *anomalies*.

First chart on Figure 6.5 illustrates extracted sequence of Edge angles $[a_1, ..a_n]$ of *Drawing* graph at depth $d = 4$. Second chart shows angle sequence $[\hat{a_1}, ...\hat{a_n}]$ predicted by corresponding *LSTM model* trained to reproduce normal behaviour of Luria 'P' pattern. Third chart demonstrates error vector, or squared difference between $[a_1, ..a_n]$ and $[\hat{a_1}, ...\hat{a_n}]$, represented by $[e_1, ...e_n]$ with applied *median* filter. Elements of the sequence $[e_1, ...e_n]$ above pre-defined threshold $t = C \times MSE$ marked as *anomalies* and shown on all charts as *gray circles*. Hyper-parameter for corresponding *angle-related LSTM model* is $C = 15$

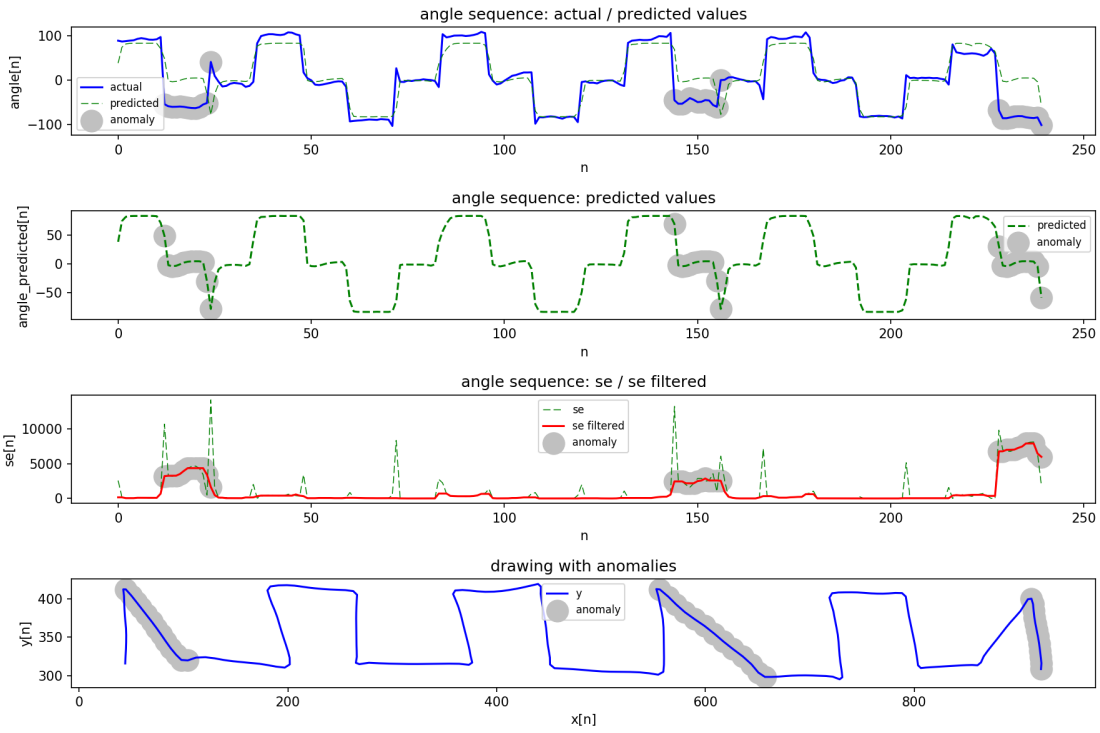

Figure 6.5: Anomaly Detection — Example Output

Chapter 7

Feature Analysis

7.1 Dataset Description

Subjects of both classes, with Parkinson's disease *PD* and healthy controls *HC* participated in research. It was totally $n_{pd} = 17$ individuals with Parkinson's disease and equal number $n_{hc} = 17$ of healthy controls, which gives total number of $n = 34$ participants.

Handwriting data was recorded for series of *Luria* and other *non-Luria* types of tests. Combination of three Luria patterns *['Sinus', 'P', 'PL']* and three task types *[continue, copy, trace]* results in series of 9 different tests. To reduce complexity and the scope of research, dataset with only *one particular test* was analyzed: copy task with 'P' pattern — or *[Luria 'P' copy]*.

From preceding phases of the algorithm, it was evaluated totally $n = 1077$ numeric features, which are distributed among three major classes:

- *Drawing-related* — total $n = 29$ aggregated higher-order features, recursively extracted from underlying *Edge* and *Anomaly* entities, describe particular instance of *Luria* pattern, or *Drawing* entity in general.
- *Edge-related* — total $n = 872$ numeric features, describe particular logical part of Luria pattern, i.e, *Edge* entity of the *Drawing* instance.
- *Anomaly-related* — total $n = 176$ numeric features, reveal number of *Anomalies* of different types inside particular *Edge* or complete *Drawing* instance.

7.2 Statistical Analysis

It was decided to apply one of the most widely used supervised feature selection methods — *Fisher scoring method* [38, 39].

Fisher score for each feature was calculated using (7.1), where μ_j and σ_j – are the *mean* and *standard deviation* of the data points in the class $j \in [pd, hc]$ for current feature, p_j – is the fraction of data points in the class j and μ – is the global *mean* of the feature. The larger the Fisher score, the greater the discriminatory potential of particular feature. To evaluate the results, Python function *fisher_score()* from *Skfeature* library was used.

$$F = \frac{\sum_{j=1}^{k} p_j(\mu_j - \mu)^2}{\sum_{j=1}^{k} p_j \sigma_j^2} \tag{7.1}$$

Along with proposed *Fisher* scoring method, additional metrics were added for feature statistical analysis and selection decision logic:

- Two-sample t-test — to reject *null hypothesis* and determine if datasets are differ significantly from each other. In current context, *null hypothesis* — is that the samples representing HC and PD subjects belong to the independent populations with *equal* means. *Alternative hypothesis* — samples representing HC and PD subjects originate from the populations with *unequal* means, hence there is significant difference between datasets.

  Standard *Scipy* function *ttest_ind()* was utilized to perform T-test. Function produces tuple of p-value and t-statistics as output. Features with high p-value $p > 0.05$ were automatically filtered out and were not used in subsequent classifier training.

- Spearman's Correlation Coefficient — ρ was adopted to additionally validate *Fisher Scoring method*, by evaluating absolute correlation $|\rho|$ to dataset class numeric values *[PD = 1.0, HC = 0.0]*. Python function *spearmanr()* from *Scipy* library was used to evaluate the results.

7.3 Feature Analysis Results

7.3.1 Drawing Features

Higher-order *Drawing* features provide information about particular instance of Luria pattern in general. Totally 97% of $n = 29$ features from current cluster passed t-test with significance level set to $\alpha = 0.05$.

Top performing drawing features are determined by Fisher score and listed in Table 7.1. According to the list, most significant of them — *drawing_acceleration_mean, drawing_jerk_mean, drawing_velocity_mean*, i.e, average acceleration, jerk and velocity of whole drawing. Highest Fisher score $F = 0.50$ was observed for *drawing_acceleration_mean*. These results are predictable, since kinematic features usually provide great discrimination power between groups of HC and PD [22, 6, 10], also our dataset confirms that subjects from PD population in average, perform drawing tasks 2-3 times slower, than healthy controls of the same age group.

Interestingly, that *drawing_acceleration_nc, drawing_velocity_nc, drawing_pressure_nc* also received high Fisher score. These features stand for total *number of changes* in acceleration, velocity and pressure during drawing test, and can, in theory, represent *tremor* — one of the primary symptoms of Parkinson's disease.

Feature Name	*Fisher* score	$\lvert\rho\rvert$	p-value	t-stat
drawing_acceleration_mean	0.5025	0.65	$2.56 \cdot 10^{-11}$	9.82
drawing_jerk_mean	0.4415	0.63	$3.42 \cdot 10^{-10}$	8.82
drawing_velocity_mean	0.4071	0.64	$1.67 \cdot 10^{-11}$	9.99
drawing_slope_mass	0.3555	0.57	$1.37 \cdot 10^{-13}$	12.01
drawing_acceleration_nc	0.3372	0.64	$6.29 \cdot 10^{-6}$	5.36
drawing_velocity_nc	0.3333	0.64	$1.46 \cdot 10^{-5}$	5.08
drawing_jerk_nc	0.3313	0.64	$3.80 \cdot 10^{-6}$	5.54
drawing_pressure_nc	0.3261	0.63	$3.96 \cdot 10^{-9}$	7.92
drawing_duration	0.3149	0.59	$2.26 \cdot 10^{-8}$	7.29
anomalies_drawing_length	0.2021	0.38	$3.68 \cdot 10^{-8}$	7.12

Table 7.1: *Drawing* features — Statistical Analysis

7.3.2 Edge Features

Edge features describe particular logical part of Luria pattern, i.e, *Edge* entity of the *Drawing* instance. Totally 78% of $n = 872$ features from current cluster passed t-test with significance level set to $\alpha = 0.05$. Highest observed Fisher score $F = 0.90$ received feature *edge_03_speed*, i.e, linear speed of third edge of the drawing, which means, that speed within particular segment of the pattern provides much higher discrimination potential, than any kinematic feature of full drawing ($F = 0.50$ was observed for *drawing_acceleration_mean*).

Highest Fisher scores, received *kinematic* feature set of different edges, identified by index i. Top non-kinematic feature with Fisher score $F = 0.46$ is *edge_14_pressure_nc*, i.e, number of changes in pressure within edge with index $i = 14$. Corresponding higher-order feature *drawing_pressure_nc* obtained much lower score of $F = 0.33$

Aforementioned numbers demonstrate, what proposed clustering technique with concepts of *Edge* entity and tree-like *graph* structure contribute to overall higher significance of generated features. Top $n = 10$ *Edge* features are presented in following Table 7.2

Feature Name	*Fisher* score	$\lvert\rho\rvert$	p-value	t-stat
edge_03_speed	0.9000	0.71	$6.11 \cdot 10^{-10}$	8.60
edge_15_velocity_mean	0.7802	0.73	$1.12 \cdot 10^{-9}$	8.38
edge_13_speed	0.7649	0.74	$1.51 \cdot 10^{-11}$	10.03
edge_03_velocity_mean	0.7591	0.69	$6.70 \cdot 10^{-10}$	8.57
edge_15_acceleration_mean	0.7464	0.73	$4.87 \cdot 10^{-9}$	7.84
edge_15_jerk_mean	0.7442	0.70	$3.40 \cdot 10^{-8}$	7.15
edge_03_acceleration_mean	0.6709	0.67	$2.06 \cdot 10^{-9}$	8.15
edge_11_velocity_mean	0.6662	0.68	$2.00 \cdot 10^{-10}$	9.02
edge_14_speed	0.6050	0.64	$4.35 \cdot 10^{-10}$	8.73
edge_11_acceleration_mean	0.5998	0.67	$5.14 \cdot 10^{-10}$	8.67

Table 7.2: *Edge* features — Statistical Analysis

7.3.3 Anomaly Features

Anomaly features represent number of observed *Anomalies* of different types inside particular *Edge* or complete *Drawing* instance. Totally 42% of $n = 176$ features from current cluster passed t-test with significance level set to $\alpha = 0.05$. Highest Fisher score $F = 0.20$ was observed for *anomalies_drawing_length*, i.e, number of length anomalies in the whole drawing. Anomaly features in Table 7.3, in general, demonstrate lower Fisher score in comparison with other feature clusters. Possible aspects and future improvement ideas may include:

- *Sequence Types* — set of analyzed feature-sequence types was pre-defined intuitively and consisted of only *[angle, pressure, length, duration, longitude, latitude]* types. Results of prior analysis *(Tables 7.1 and 7.2)* confirm, that most-performing are *kinematic* feature-types, hence changing subset of types to *kinematic* may, in theory, improve significance of *Anomaly* features.
- *Error Threshold* — every hyper-parameter C for definition of *Anomaly* threshold $t = C \times MSE$ was determined experimentally. Further tuning of C may enhance feature performance.
- *Feature Engineering* — we may improve the results by engineering advanced feature types — Anomaly *strength*, i.e, average error value of anomalies within segment. It is also feasible to sum all errors and obtain Anomaly *mass*.

Feature Name	*Fisher* score	$\lvert\rho\rvert$	p-value	t-stat
anomalies_drawing_length	0.2021	0.38	$3.68 \cdot 10^{-8}$	7.12
anomalies_edge_25_pressure	0.0891	0.31	$2.24 \cdot 10^{-2}$	−2.34
anomalies_edge_11_length	0.0695	0.14	$2.38 \cdot 10^{-3}$	3.28
anomalies_edge_07_latitude	0.0667	0.25	$3.58 \cdot 10^{-2}$	−2.15
anomalies_edge_04_pressure	0.0592	0.25	$3.46 \cdot 10^{-4}$	−3.79
anomalies_edge_03_pressure	0.0519	0.25	$3.64 \cdot 10^{-3}$	−3.02
anomalies_drawing_angle	0.0514	0.21	$4.06 \cdot 10^{-5}$	4.73

Table 7.3: *Anomaly* features — Statistical Analysis

Chapter 8

Classifier Training

8.1 Methodology

After previous analysis phase, statistically significant numeric feature subset was determined for labelled collection of drawings, received from Parkinson's disease patients PD and healthy control HC. Major goal of current research is to construct classifier model for HC and PD differentiation. Considering number of possible labels in dataset (PD and HC), we a dealing with standard *binary classification* problem, which is solvable by machine learning algorithms.

8.1.1 Classifier Algorithms

For the purpose of best performing classifier acquisition for PD discrimination task, it was decided to train, analyze and compare classical and modern machine learning algorithms from the following subset:

- k-NN — k-nearest neighbors, classical supervised learning method, yields label of particular instance. Instance is classified by a majority vote of its neighbors — from most common class among its k nearest instances from training dataset. *KNeighborsClassifier* model of *Scikit-learn* library was used for current implementation with k parameter set to $k = 3$.
- Decision Tree — classical supervised method for classification [38], where learning process is modelled with set of hierarchical decisions, based on features and represented by tree-like graph structure. Every tree node – is a decision, i.e,

some condition on one or multiple features of the dataset. Every leaf is a class label. Implementation utilizes *DecisionTreeClassifier* model of *Scikit-learn* library with parameter of $max\_depth = 5$

- Random Forest — ensemble learning method for classification tasks proposed by Ho [40] in 1995. The essence of the technique is to construct collection of simple decision-tree predictors in randomly selected feature sub-space. Final prediction of the class label determined by most common class from the output of individual decision-trees. Standard *RandomForestClassifier* model of *Scikit-learn* was applied in current solution.
- SVM — stands for *Support Vector Machine*, supervised learning algorithm proposed by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. Main idea of the method is to attempt to pass a linearly separable maximal margin hyperplane through a dataset in order to separate two classes. Hyperplane — is a linear separator for any n-dimensional space, represented by *line* in 2-dimensional, by *plane* in 3-dimensional and by *hyperplane* in higher-dimensional space.

  If dataset is non-linearly separable, the observation points are mapped to higher-dimensional space by means of a kernel function in order to find separating hyperplane. Common types of kernels are linear, polynomial and radial basis kernels. In current solution two implementations of SVM models from *Scikit-learn* library were trained and analyzed — with radial basis kernel SVM_{rbf} and linear kernel SVM_{linear}.
- AdaBoost — or *Adaptive Boosting*, machine learning meta-algorithm proposed by Schapire and Freund [41]. AdaBoost uses ensemble concept also known as *boosting*. Fundamental idea of this technique — to combine weaker classifiers in order to build a stronger learner. Algorithm constructs a strong classifier as a set of weak classifiers, each performing at least above 50% of accuracy. We used implementation of *AdaBoostClassifier* from *Scikit-learn* library. Number of estimators was set to $n = 50$ and estimator type assigned to default *DecisionTreeClassifier*.

8.1.2 Classifier Validation

To confirm correctness of accuracy measures, we applied K-Fold cross-validation technique, which consists of following steps:

1. Divide original dataset of drawings into mutually exclusive and possibly equal-sized k subsets. Each subset represent a *fold*. So result is vector of folds $[f_1, f_2, .., f_k]$.
2. For every fold f_i in $i = 1..k$ repeat:
   (a) Keep fold f_i as *validation* set and use remaining $k - 1$ folds as cross-validation *training* set
   (b) Train model using corss-validation *training* set from previous step, evaluate accuracy P^i_{acc} of the model inside f_i fold by comparing actual predictions with *validation* set.
3. Obtain model accuracy P_{acc} by calculating mean accuracy from all k folds $[f_1, f_2, .., f_k]$

Using K-Fold cross validation technique, we've achieved state, when all the data points in the original dataset were used for both *training* and *validation*. Also, each data point was used for validation only *once*. Normally, the value of $k = 10$, however considering our dataset size of $n = 34$ data-points, it was decided to use $k = 3$ folds. In our context, every fold would represent 11-12 drawings of 5-6 both *PD* and *HC* class instances. We utilized *StratifiedKFold* from *Scikit-learn* library — improved implementation of common K-Fold algorithm, where folds are constructed with possibly *equal percentage* of samples for *each class*.

8.1.3 Classifier Training Process

To analyze different combinations of models and hyper-parameters and achieve high classification accuracy P_{acc}, collection of models with *k-NN, Decision Tree, Random Forest, SVM and AdaBoost* was trained with different number of top-performing features ($n = 3$, $n = 30$, $n = 90$) from separate *Drawing, Edge, Anomaly* and *Mixed* features-clusters for each of $k = 3$ folds. All randomly generated folds with trained models and accuracy measures were saved in auxiliary *Classifier-container entity* for subsequent analysis.

8.2 Classifier Analysis

Output of previous phase is demonstrated in the form of slices: *feature space*, classifier *model type*, *feature class*. Following Tables 8.1, 8.2 and 8.3 describe model performance for different feature-spaces with top $n = 3$, $n = 30$ and $n = 90$ performing features. Each table combines slices of model types and feature classes. Analyzed model types — *k-NN, Decision Tree, Random Forest, SVM and AdaBoost* and feature classes — *Drawing, Edge, Anomaly and Mixed*.

From classifier type perspective — we can observe, that best performing model is *Random Forest* with highest accuracy $P_{acc} = 0.91$, high performance also demonstrates *AdaBoost* and classical *Decision Tree* with $P_{acc} = 0.86$ and $P_{acc} = 0.88$. For some reason, Support Vector Machine classifies our dataset with slightly lower accuracy rate of $P_{acc} = 0.67$

From feature space perspective — we can achieve $P_{acc} = 0.88$ with only $n = 3$ features. Which conforms to high statistical significance of extracted drawing parameters. Best accuracy of $P_{acc} = 0.91$ was obtained with $n = 30$ feature space. Much higher dimensionality of $n = 90$ features does not boost model accuracy, but even makes it slightly lower with $P_{acc} = 0.88$.

From feature class perspective — *Edge*-related features perform best with highest accuracy $P_{acc} = 0.91$. Fusion of all features yields classifier with $P_{acc} = 0.86$. *Drawing*-related features describe Luria pattern in general and can produce model with $P_{acc} = 0.84$. Class of separated *Anomaly* features demonstrates model with highest accuracy of $P_{acc} = 0.67$, which also conforms to statistical analysis and overall low *Anomaly* presence in the current dataset of drawings.

Classifier	P_{acc} all	P_{acc} drawing	P_{acc} edge	P_{acc} anomaly
AdaBoost	0,79	0,79	0,76	0,55
DecisionTree	0,82	0,76	0,79	0,44
k-NN	0,71	0,84	0,71	0,61
RandomForest	0,86	0,84	0,88	0,50
SVM_{linear}	0,56	0,50	0,67	0,62
SVM_{rbf}	0,56	0,62	0,56	0,57

Table 8.1: Classifier accuracy P_{acc} — trained with top $n = 3$ features

Classifier	P_{acc} all	P_{acc} drawing	P_{acc} edge	P_{acc} anomaly
AdaBoost	0,77	0,70	0,80	0,55
DecisionTree	0,70	0,80	0,80	0,44
k-NN	0,74	0,69	0,70	0,64
RandomForest	0,86	0,77	0,91	0,59
SVM_{linear}	0,53	0,50	0,50	0,62
SVM_{rbf}	0,56	0,56	0,62	0,56

Table 8.2: Classifier accuracy P_{acc} — trained with top $n = 30$ features

Classifier	P_{acc} all	P_{acc} drawing	P_{acc} edge	P_{acc} anomaly
AdaBoost	0,82	0,77	0,86	0,68
DecisionTree	0,79	0,69	0,88	0,66
k-NN	0,65	0,71	0,71	0,56
RandomForest	0,85	0,80	0,89	0,60
SVM_{linear}	0,50	0,50	0,49	0,57
SVM_{rbf}	0,56	0,62	0,56	0,64

Table 8.3: Classifier accuracy P_{acc} — trained with top $n = 90$ features

8.3 Data Visualization

Possible reasons, why classifiers achieve high prediction performance with accuracy of $P_{acc} = 0.91$ is subtle feature engineering and selection technique, which again is confirmed by high *Fisher score* after statistical analysis. For the sake of demonstration, subset of classifiers was trained with top-performing *pairs* of features in order to visualize *decision boundaries* within two-dimensional feature space.

2D scatter-plots on Figures 8.1, 8.2 and 8.3 illustrate *decision boundaries* for different classifiers trained with only $n = 2$ features, taken from separate feature clusters. Particular subplot demonstrates classifier model — k-NN (top left), Decision Tree (top right), Random Forest (bottom left), AdaBoost (bottom right). Each X and Y axis represents individual feature from different feature classes — *Drawing, Edge, Anomaly*. Healthy controls *HC* instances are marked with *yellow* dots. Scatter-plot on Figure 8.4 illustrates all instances within 3D feature space.

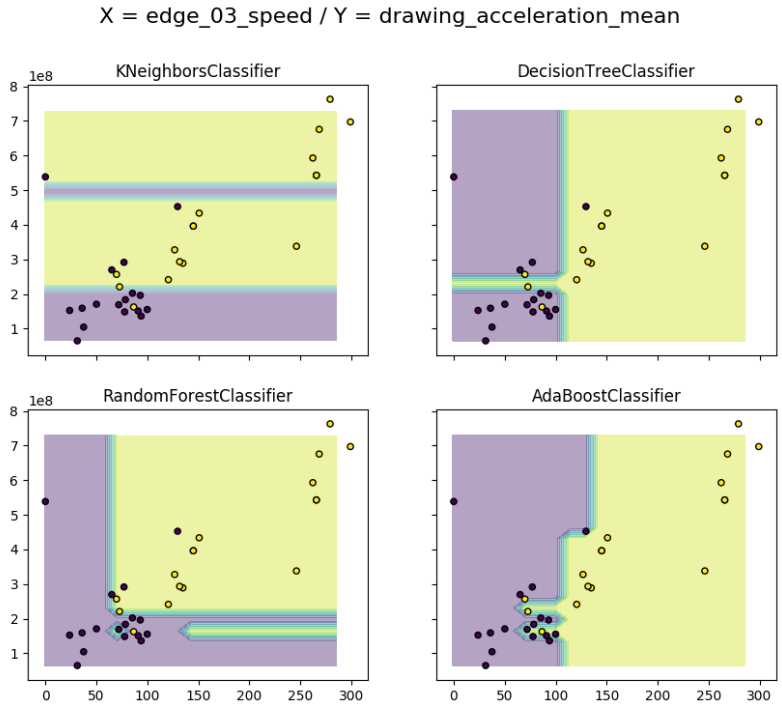

Figure 8.1: Decision Boundaries for PD Classifiers: x-axis — linear speed of third edge of the drawing, y-axis — average acceleration of the drawing. Healthy controls *HC* instances are marked with *yellow* dots.

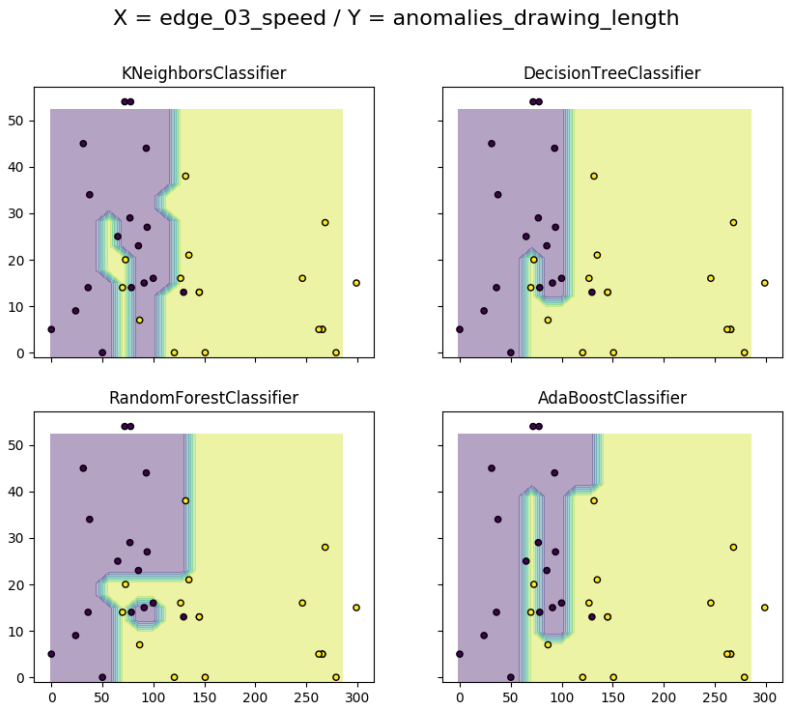

Figure 8.2: Decision Boundaries for PD Classifiers: x-axis — linear speed of third edge of the drawing, y-axis — number of length anomalies in the drawing. Healthy controls *HC* instances are marked with *yellow* dots.

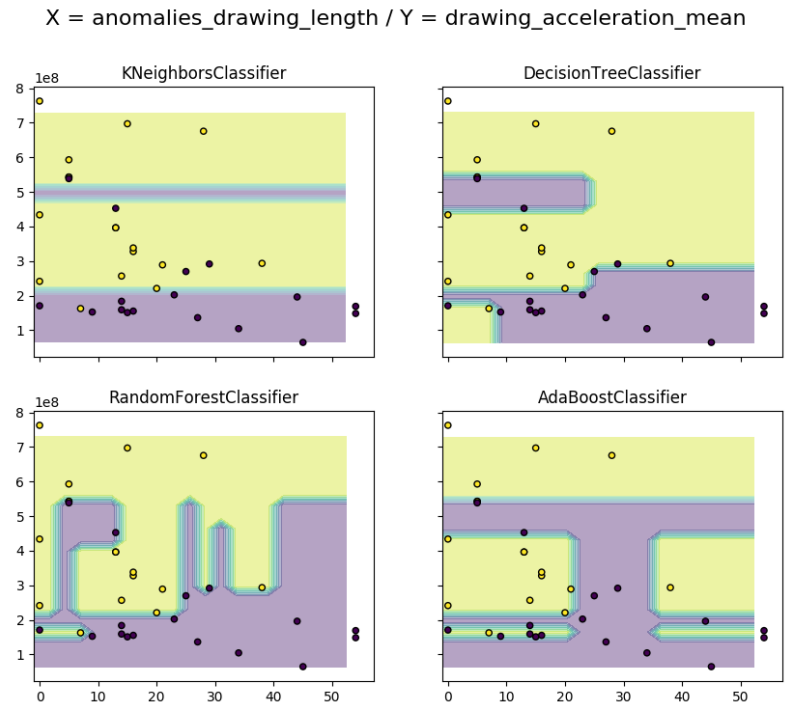

Figure 8.3: Decision Boundaries for PD Classifiers: x-axis — number of length anomalies in the drawing, y-axis — average acceleration of the drawing. Healthy controls *HC* instances are marked with *yellow* dots.

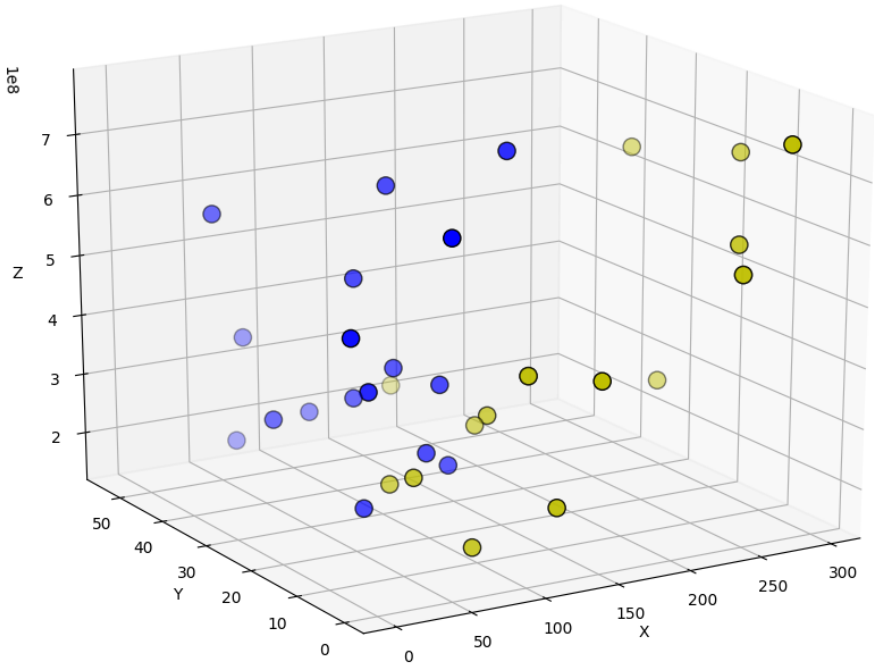

Figure 8.4: Dataset of subjects (*HC* — *yellow*, PD — *blue*) in 3D feature-space: X-axis — linear speed of third edge of the drawing, Y-axis — number of length anomalies in the drawing, Z-axis — average acceleration of the drawing.

Chapter 9

Explaining Predictions

9.1 Motivation

Outcome of the preceding phases of algorithm — numerous binary classifier models, capable of differentiating Parkinson's disease patients PD from healthy control subjects HC with up to 91% of accuracy. In theory, obtained classifier models may be integrated into decision support framework and used by clinicians for Parkinson's disease diagnosis.

There exists a major problem with practical application of present method in hospitals. Main reason would be overall *"black box"* nature of machine learning algorithms, such as neural networks, random forests or support vector machines SVM. Even if particular model offers high accuracy rate, it would not be considered as trustworthy by clinicians, primarily because of prediction making process is *not traceable*.

It is worth mentioning, that limited amount of simple machine learning models offer some level of interpretability. For example, particular models of Decision Tree, Linear Classifier can be described by *feature weights*. Recently became possible to interpret binary Random Forest classifier models with *feature contribution method*, proposed by Palczewska, Palczewski, Robinson, and Neagu [30] in 2014.

However interpretations of individual predictions were not feasible before.

9.2 Methodology

9.2.1 LIME

In 2016 Ribeiro, Singh, and Guestrin [11] came up with novel machine learning meta-algorithm, called "Local Interpretable Model-Agnostic Explanations" or *LIME*, which is innovative explanation technique, that explains the predictions of *any* classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. This method was also extended to *SP-LIME* algorithm, which yields representative and non-redundant set of predictions to provide higher overview of any machine learning model.

LIME algorithm was built around assumption, that any complex model is linear on local scale, namely — pair of close datapoints should have predictable behaviour, even the complex model is non-linear, therefore it is feasible to fit linear model around particular datapoint to create local linear representation of the global non-linear model. Linear model then can be utilized to explain predictions of the complex model locally. In general, *LIME* algorithm follows these main steps to explain global model M individual prediction for datapoint d:

1. From original datapoint d to explain, create randomly permuted points $[d_1, d_2, ..d_n]$, obtain predictions for each of $[d_1, ..d_n]$ from global model M and assign labels.
2. Evaluate distances between $[d_1, ..d_n]$ and original d
3. Choose k features and fit simple linear model M_{linear} using labelled points $[d_1, ..d_n]$ and their distances to original datapoint d.
4. Obtain feature list $[f_1, ..f_m]$ with *feature weights* from simple model M_{linear}, use them as explanation for datapoint d prediction.

Distance measuring, feature selection and model fitting from steps (2) and (3) are complex processes and may have multiple implementations and different strategies. Obtained *explanation* is not faithful globally, but it is faithful locally around d. In context of LIME, *explanation* — is textual artifact, that provides understanding of the relationship between the datapoint d features $[f_1, ..f_m]$ and particular model's prediction.

9.2.2 LIME Integration

It was decided to apply *LIME* to generate explanations for individual predictions of *HC* and *PD* datapoints from validation set and observe, how algorithm performs. *LIME* has *Python* implementation in form of *lime* package, which was integrated effortlessly into existing project.

In order to obtain explanation for individual instance d, we create *LimeTabular-Explainer* object and execute *explain_instance*() method, while providing following input parameters:

- Particular classifier model M object, which predicts class of the instance d
- Related labelled drawings $[d1, ..d_n]$ from training set
- Feature names $[f_1, ..f_m]$
- Class labels $[PD, HC]$
- Individual instance d from validation set

Output is complex *Explanation* object, which in general, contains prediction probability for each class $[PD, HC]$ and list of tuples $[(x_1, y_1), ..((x_m, y_m)]$, where $[x_1, ..x_m]$ — feature names ordered by their numeric significance values $[y_1, ..y_m]$, which were calculated from particular feature f_i weight inside linear model M_{linear} — local linear approximation of global model M. Algorithm is also capable of showing discretized ranges for every explained feature f_i of the d instance.

9.3 Experimental Result

Following Table 9.1 and Table 9.2 demonstrate sample classification explanations for random instances of HC and PD, taken from validation dataset. Classifier model M is *Random Forest* with measured accuracy $P_{acc} = 0.88$, trained with $n = 90$ features from *mixed* feature class. To confirm validity of the explanation, actual feature value of the corresponding instance was added to *LIME* output as a reference.

9.3.1 Explanation for PD instance

First explanation *(Table 9.1)* confirms, that instance is PD with 71% probability. Main reasons for such classification result, according to *Random Forest* model:

- *Velocity* is lower than expected for edge with index 3
- Increased number of *pressure changes* for edges with indexes 6 and 14
- *Linear speed* for edges 5, 12, 22 is lower, than expected
- *Duration* for edge 14 is longer, than expected

9.3.2 Explanation for HC instance

Second explanation *(Table 9.2)* confirms, that instance is actually healthy control HC with 90% probability. Primary reasons for such classification result, according to *Random Forest* model:

- High *linear speed* for edges 5, 6, 10, 22
- High *average velocity* for edges 3, 10, 11, 15
- Number of *pressure changes* in edges 6, 14 are low

Classifier Model — RandomForest, $n = 90$ features		
Instance PD-07 — *Prediction* PD (71%) / Control (29%)		
Prediction Explanations: Parkinson's Disease PD (71%)		
Feature Explanation	Feature Actual	Significance
edge_03_velocity_mean ≤ 9.07e+03	4.25e+03	0.061013
edge_14_pressure_nc > 1.95e+01	2.80e+01	0.054542
edge_03_acceleration_mean ≤ 1.92e+08	7.28e+07	0.042965
drawing_slope_mass > 1.83e+05	3.21e+05	0.037471
1.65e+01 < edge_06_pressure_nc ≤ 1.90e+01	1.90e+01	0.034442
edge_03_jerk_mean ≤ 1.21e+13	3.43e+12	0.029171
edge_22_speed ≤ 5.39e+01	3.90e+01	0.027016
1.62e+13 < edge_00_jerk_mean ≤ 2.76e+13	2.14e+13	0.018656
1.25e+00 < edge_00_pressure_mass ≤ 1.51e+00	1.45e+00	0.012146
5.32e+01 < edge_05_speed ≤ 7.01e+01	5.85e+01	0.011786
6.27e+01 < edge_12_speed ≤ 1.02e+02	1.00e+02	0.011575
edge_14_duration > 9.40e-01	1.62e+00	0.010385
Prediction Explanations: Healthy Control HC (29%)		
Feature Explanation	Feature Actual	Significance
edge_14_velocity_mean ≤ 7.31e+03	5.81e+03	-0.012261
edge_14_acceleration_mean ≤ 1.43e+08	1.36e+08	-0.012769
2.26e+08 < edge_07_acceleration_mean ≤ 3.20e+08	2.29e+08	-0.013645
5.81e+03 < edge_10_velocity_mean ≤ 9.51e+03	7.19e+03	-0.015295
6.81e+12 < edge_20_jerk_mean ≤ 1.32e+13	7.37e+12	-0.023397
6.23e+01 < edge_10_speed ≤ 1.16e+02	9.00e+01	-0.025807
5.96e+03 < edge_15_velocity_mean ≤ 1.20e+04	9.29e+03	-0.032646
7.28e+01 < edge_06_speed ≤ 1.05e+02	7.65e+01	-0.077155

Table 9.1: PD Instance Explanation — Example

Classifier Model — RandomForest, $n = 90$ features		
Instance HC-01 — *Prediction* HC (90%) / PD (10%)		
Prediction Explanations: Healthy Control HC (90%)		
Feature Explanation	Feature Actual	Significance
edge_06_speed > 1.05e+02	2.67e+02	0.111788
edge_11_velocity_mean > 1.45e+04	2.63e+04	0.074923
edge_15_velocity_mean > 1.82e+04	2.23e+04	0.073401
drawing_slope_mass ≤ 1.14e+05	8.27e+04	0.059881
edge_03_velocity_mean > 1.56e+04	3.79e+04	0.057738
edge_06_pressure_nc ≤ 1.05e+01	5.00e+00	0.057620
edge_14_pressure_nc ≤ 1.02e+01	7.00e+00	0.037617
edge_03_jerk_mean > 2.73e+13	7.69e+13	0.032639
edge_10_speed > 1.16e+02	2.60e+02	0.031768
edge_22_speed > 1.21e+02	2.67e+02	0.030603
edge_21_acceleration_nc ≤ 6.75e+00	4.00e+00	0.026311
edge_05_speed > 9.92e+01	2.46e+02	0.024375
edge_10_velocity_mean > 1.50e+04	3.28e+04	0.023728
edge_03_acceleration_mean > 3.54e+08	8.96e+08	0.021507
edge_14_duration ≤ 4.20e-01	3.30e-01	0.018153
Prediction Explanations: Parkinson's Disease PD (10%)		
Feature Explanation	Feature Actual	Significance
edge_00_jerk_mean > 2.76e+13	3.54e+13	-0.021943
edge_01_jerk_mean > 1.70e+13	4.17e+13	-0.026075
edge_14_velocity_mean > 1.29e+04	2.30e+04	-0.027546
edge_14_acceleration_mean > 2.92e+08	6.07e+08	-0.037125
edge_20_jerk_mean > 2.73e+13	7.93e+13	-0.066437

Table 9.2: Healthy Control HC Instance Explanation — Example

Chapter 10

Discussion

Overall, primary objectives of present thesis were met. We conducted detailed analysis of patterns drawn during Luria's alternating series tests. Analysis was based on data collected from 17 real patients with approved Parkinson's disease diagnosis and 17 healthy control subjects within same age group. Main outcome of the analysis is machine learning classifier, capable of differentiation Parkinson's disease patients from healthy controls, providing prediction performance around 91%. We also integrated technique for explaining individual predictions of obtained classifier based on "Local Interpretable Model-Agnostic Explanations" novel meta-algorithm.

Proposed solution is complex multilevel process with several dependent phases. During each phase we experienced certain challenges. Some design decision exceeded our expectations, few on the other hand, revealed various limitations and possible flaws. All these aspects should be discussed.

Applied "Shi-Tomasi" computer vision algorithm [31] performed really well during clustering phase, however there is no automatic validation technique for measuring quality of corner detection results. The only option — is to perform manual visual check of the image. Another possible limitation of the algorithm — is pre-defined constants for maximum corner number and minimum distance between corners, which potentially narrows usage of the method for other pattern types. Probable solution would require including heuristics before algorithm execution, which evaluates possible values for algorithm constants.

Most successful design decision during implementation — is certainly clustering solution architecture. We combined computer vision, object-oriented programming

techniques and recursion. Point, Edge and Drawing entities were introduced to describe pattern on different levels of abstraction. Computer vision assisted in splitting Drawing into Edge objects — meaningful logical parts of the pattern. Every Edge object was recursively divided into connected sub-objects of the same type while preserving references to parent and child entities. Outcome of whole clustering process — tree-like graph structure of connected pattern segments, represented by Edge objects. Clustering algorithm is capable of processing periodic drawing patterns with relatively positioned and scaled elements, preserves logical structure and describes pattern with arbitrary level of detail. Tree-like graph structure is very flexible for feature generation and interpretation, which helped to produce features with much higher statistical significance, comparing to features obtained from unclustered drawings. Subsequent anomaly detection concept was built on top of existing clustering solution. In general clustering component exceeded initial expectations.

Anomaly detection — is interesting and promising concept, derived from clustering solution, capable of identifying "unseen" or "unexpected" segments of any feature sequence. Sequences were generated with LSTM neural network, utilizing tree-like graph data structure of the Drawing object. As was already mentioned, Anomaly features showed slightly lower significance in comparison with common features obtained from pattern segments. Nevertheless, we can enhance Anomaly feature performance. For example, set of pre-defined feature sequence types was initially limited and non-kinematic. Results of subsequent analysis showed, that most-performing are common features with kinematic type, therefore changing subset of analyzed feature sequences, may improve significance. Another idea — every hyper-parameter C for definition of Anomaly threshold $t = C \times MSE$ was determined experimentally. Further tuning of C may enhance feature performance. We may also improve the results by engineering advanced feature types, such as Anomaly strength — by averaging error value of anomalies within edge, or Anomaly mass — by summing all obtained Anomaly strength values in the same region. Anomaly detection solution is neural-network based, possible enhancement may be found by applying new network types. For example, recently proposed Independently Recurrent Neural Network *IndRNN* [42] in theory, can process longer sequences and can

be stacked into deep neural network models.

Statistical analysis exposed decent performance of features, derived from certain logical pattern segments with maximum Fisher score around 0.90. Features, derived from whole pattern showed much lower discrimination power with maximum Fisher score around 0.50. This fact again, confirms our assumption, that proposed clustering technique contributes to overall higher significance of generated features. From feature type perspective, most promising — are kinematic features, which was already stated in multiple research papers. Non-kinematic — geometry features, pressure, altitude, latitude of stylus revealed their low significance in Parkinson's disease differentiation.

During classifier training, we discovered most performing algorithms — Random forest, AdaBoost and Decision Tree. Interestingly, support vector machine SVM models performed with slightly lower accuracy. This fact is worth investigating in future and possibly conduct experiments with SVM model parameters. Models were validated with K-Fold technique. Normally value of $k = 10$, however considering existing dataset size of $n = 34$ data-points, it was decided to use only $k = 3$ folds. We achieved highest average classification accuracy around 91% for Random Forest model, trained with $n = 30$ top features.

During experimental phase, "Local Interpretable Model-Agnostic Explanations" algorithm was integrated into existing project for explaining individual predictions of obtained classifier. LIME certainly is innovative explanation technique, that explains the predictions of any classifier in an interpretable and faithful manner, still authors warn about undeniable limitations. Generated explanations are not faithful globally, but are faithful only locally around certain predicted datapoint instance. Additionally, there is no objective method for measuring correctness of explanations. Nevertheless, this technique, without doubt can reveal hidden relations between instance features and particular classification result with some degree of confidence.

Chapter 11

Conclusion

Primary goal of present thesis was to conduct analysis of patterns drawn during Luria's alternating series tests, extract interpretable feature set and develop machine learning model, capable of correct differentiation of Parkinson's disease patients from healthy controls. Additional goal was to offer solution for explaining individual predictions of obtained classifier.

Research is based on handwriting data collected from 17 patients with diagnosed Parkinson's disease and 17 healthy control subjects within same age group.

Novel clustering technique was applied during analysis. Proposed clustering algorithm is computer vision based and capable of processing periodic drawing patterns with relatively positioned and scaled elements, preserving logical structure with arbitrary level of detail. Obtained concept could also be applied in other research fields.

Anomaly detection concept was also introduced and applied. Anomaly detection technique evolved from clustering concept. Method is neural network based and capable of identifying "unseen" or "unexpected" sequence segments within drawing patterns.

Detailed statistical analysis of extracted features was successfully conducted and revealed high number of significant kinematic parameters, derived from certain logical segments of the pattern. Main outcome of the analysis was machine learning classifier, capable of differentiation Parkinson's disease patients from healthy controls, providing average prediction performance around 91%.

Technique for explaining individual predictions of obtained classifier based on

"Local Interpretable Model-Agnostic Explanations" algorithm was successfully integrated into final solution and performed adequately.

Proposed technique could be included into decision support framework for Parkinson's disease screening and, in theory, adopted by clinicians in medical facilities. Obtained methodology can certainly help to reveal hidden relations between pattern instance parameters and particular classification result with some degree of confidence. With such classification reasoning, clinicians are capable of making informed decisions about whether to trust the model's predictions.

Obtained results of present thesis clearly indicate, that main goals were successfully achieved. Present research can evolve in different ways.

Most obvious research direction — extension of present machine learning binary classifier to regression model, capable of predicting severity of Parkinson's disease, expressed in unified Parkinson's disease rating scale *UPDRS*. Without doubt, it will require a reasonably large group of patients with various levels of disease severity.

Additionally, interesting and promising anomaly detection concept could be investigated much further. Novel feature engineering, parameter tuning, experiments with neural network architecture — are some of possible improvement areas.

Acknowledgments

I would like to thank my supervisor Sven Nõmm for his encouragement, inspiration and motivation, professor Aaro Toomela for knowledge sharing and collaboration, Kadri Medijainen for collecting patients' data. Last but not the least, I would like to express my deepest gratitude to my friends, family and especially my mother, for unconditional and unfailing support.

Bibliography

[1] Ahmed A Moustafa, Srinivasa Chakravarthy, Joseph R Phillips, Ankur Gupta, Szabolcs Keri, Bertalan Polner, Michael J Frank, and Marjan Jahanshahi. Motor symptoms in parkinson's disease: A unified framework. *Neuroscience & Biobehavioral Reviews*, 68:727–740, 2016.

[2] Esther J Smits, Antti J Tolonen, Luc Cluitmans, Mark van Gils, Bernard A Conway, Rutger C Zietsma, Klaus L Leenders, and Natasha M Maurits. Standardized handwriting to assess bradykinesia, micrographia and tremor in parkinson's disease. *PloS one*, 9(5):e97614, 2014.

[3] Aparna Wagle Shukla, Songthip Ounpraseuth, Michael S Okun, Vickie Gray, John Schwankhaus, and Walter Steven Metzer. Micrographia and related deficits in parkinson9s disease: a cross-sectional study. *BMJ open*, 2(3):e000628, 2012.

[4] Peter Drotár, Jiří Mekyska, Irena Rektorová, Lucia Masarová, Zdeněk Smékal, and Marcos Faundez-Zanuy. Evaluation of handwriting kinematics and pressure for differential diagnosis of parkinson's disease. *Artificial intelligence in Medicine*, 67:39–46, 2016.

[5] Evelien Nackaerts, Elke Heremans, Bouwien CM Smits-Engelsman, Sanne Broeder, Wim Vandenberghe, Bruno Bergmans, and Alice Nieuwboer. Validity and reliability of a new tool to evaluate handwriting difficulties in parkinson's disease. *PloS one*, 12(3):e0173157, 2017.

[6] Alban Letanneux, Jeremy Danna, Jean-Luc Velay, François Viallet, and Serge Pinto. From micrographia to parkinson's disease dysgraphia. *Movement Disorders*, 29(12):1467–1475, 2014.

[7] Peter Drotár, Jiří Mekyska, Irena Rektorová, Lucia Masarová, Zdeněk Smékal, and Marcos Faundez-Zanuy. Decision support framework for parkinson's disease based on novel handwriting markers. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(3):508–516, 2015.

[8] Charles J Golden, Thomas A Hammeke, and Arnold D Purisch. Diagnostic validity of a standardized neuropsychological battery derived from luria's neuropsychological tests. *Journal of consulting and clinical psychology*, 46(6):1258, 1978.

[9] Sven Nõmm, Aaro Toomela, Julia Kozhenkina, and Toomas Toomsoo. Quantitative analysis in the digital luria's alternating series tests. In *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*, pages 1–6. IEEE, 2016.

[10] Serge Pinto and Jean-Luc Velay. Handwriting as a marker for pd progression: a shift in paradigm, 2015.

[11] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[12] A. R. Luriaa. *Higher Cortical Functions in Man*. Springer, 1995.

[13] Sara Rosenblum, Margalit Samuel, Sharon Zlotnik, Ilana Erikh, and Ilana Schlesinger. Handwriting as an objective tool for parkinson's disease diagnosis. *Journal of neurology*, 260(9):2357–2361, 2013.

[14] Julia Kozhenkina. Quantitative analysis of the kinematic features for the luria's alternating series test. Master thesis, Tallinn University of Technology, 2016.

[15] Ilja Mašarov. Digital clock drawing test implementation and analysis. Master thesis, Tallinn University of Technology, 2017.

[16] Marta San Luciano, Cuiling Wang, Roberto A Ortega, Qiping Yu, Sarah Boschung, Jeannie Soto-Valencia, Susan B Bressman, Richard B Lipton, Seth

Pullman, and Rachel Saunders-Pullman. Digitized spiral drawing: A possible biomarker for early parkinson's disease. *PloS one*, 11(10):e0162799, 2016.

[17] Clayton R Pereira, Silke AT Weber, Christian Hook, Gustavo H Rosa, and João P Papa. Deep learning-aided parkinson's disease diagnosis from handwritten dynamics. In *Graphics, Patterns and Images (SIBGRAPI), 2016 29th SIBGRAPI Conference on*, pages 340–346. IEEE, 2016.

[18] Somayeh Aghanavesi, Dag Nyholm, Marina Senek, Filip Bergquist, and Mevludin Memedi. A smartphone-based system to quantify dexterity in parkinson's disease patients. *Informatics in Medicine Unlocked*, 9:11–17, 2017.

[19] Arend WA Van Gemmert, Hans-Leo Teulings, and George E Stelmach. Parkinsonian patients reduce their stroke size with increased processing demands. *Brain and cognition*, 47(3):504–512, 2001.

[20] Jonathan A Sisti, Brandon Christophe, Audrey Rakovich Seville, Andrew LA Garton, Vivek P Gupta, Alexander J Bandin, Qiping Yu, and Seth L Pullman. Computerized spiral analysis using the ipad. *Journal of neuroscience methods*, 275:50–54, 2017.

[21] Peter Drotár, Jiri Mekyska, Irena Rektorová, Lucia Masarová, Zdenek Smékal, and Marcos Faundez-Zanuy. A new modality for quantitative evaluation of parkinson's disease: In-air movement. In *Bioinformatics and Bioengineering (BIBE), 2013 IEEE 13th International Conference on*, pages 1–4. IEEE, 2013.

[22] Mari Raudmann, Pille Taba, and Kadri Medijainen. Handwriting speed and size in individuals with parkinson's disease compared to healthy controls: the possible effect of cueing. *Acta Kinesiologiae Universitatis Tartuensis*, 20:40–47, 2014.

[23] Peter Drotár, Jiří Mekyska, Zdeněk Smékal, Irena Rektorová, Lucia Masarová, and Marcos Faundez-Zanuy. Contribution of different handwriting modalities to differential diagnosis of parkinson's disease. In *Medical Measurements and Applications (MeMeA), 2015 IEEE International Symposium on*, pages 344–348. IEEE, 2015.

[24] Sven Nõmm, Konstantin Bardõš, Ilja Mašarov, Julia Kozhenkina, Aaro Toomela, and Toomas Toomsoo. Recognition and analysis of the contours drawn during the poppelreuter's test. In *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pages 170–175. IEEE, 2016.

[25] Sergio Della Sala, Marcella Laiacona, Cristina Trivelli, and Hans Spinnler. Poppelreuter-ghent's overlapping figures test: its sensitivity to age, and its clinical use. *Archives of Clinical Neuropsychology*, 10(6):511–534, 1995.

[26] Ejnar Alex Korner, Lise Lauritzen, Flemming Mørkeberg Nilsson, Annette Lolk, and Peder Christensen. Simple scoring of the clock-drawing test for dementia screening. *Dan Med J*, 59(1):A4365, 2012.

[27] William Souillard-Mandar, Randall Davis, Cynthia Rudin, Rhoda Au, David J Libon, Rodney Swenson, Catherine C Price, Melissa Lamar, and Dana L Penney. Learning classification models of cognitive conditions from subtle behaviors in the digital clock drawing test. *Machine learning*, 102(3):393–441, 2016.

[28] Henry Brodaty and Cressida M Moore. The clock drawing test for dementia of the alzheimer's type: A comparison of three scoring methods in a memory disorders clinic. *International journal of geriatric psychiatry*, 12(6):619–627, 1997.

[29] Poonam Zham, Dinesh K Kumar, Peter Dabnichki, Sridhar Poosapadi Arjunan, and Sanjay Raghav. Distinguishing different stages of parkinson's disease using composite index of speed and pen-pressure of sketching a spiral. *Frontiers in Neurology*, 8:435, 2017.

[30] Anna Palczewska, Jan Palczewski, Richard Marchese Robinson, and Daniel Neagu. Interpreting random forest classification models using a feature contribution method. In *Integration of reusable systems*, pages 193–218. Springer, 2014.

[31] Jianbo Shi et al. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[32] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995.

[33] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. Density-based spatial clustering of applications with noise. In *Int. Conf. Knowledge Discovery and Data Mining*, volume 240, 1996.

[34] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[35] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[37] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[38] Charu C Aggarwal. *Data mining: the textbook*. Springer, 2015.

[39] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[40] Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE, 1995.

[41] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT press, 2012.

[42] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. *arXiv preprint arXiv:1803.04831*, 2018.