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

Eduard Vigula, 186106 IAIB

HYPERPARAMETER TUNING FOR MACHINE LEARNING BASED PARKINSON'S DISEASE DIAGNOSTICS

Bachelor's thesis

Supervisors: Elli Valla, MSc
Sven Nõmm, PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Eduard Vigula, 186106 IAIB

HÜPERPARAMEETRITE HÄÄLESTAMINE MASINÕPPEL BASEERUVA PARKINSONI TÕVE DIAGNOSTIKA JAOKS

Bakalaureusetöö

Juhendajad: Elli Valla, MSc

Sven Nõmm, PhD

Tallinn 2021

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: Eduard Vigula

25.05.2021

Hyperparameter Tuning for Machine Learning Based Parkinson's Disease Diagnostics

Abstract

Primary goal of present thesis is to improve the quality of drawing tests analysis for Parkinson's disease diagnostics. The majority of the machine-learning-based results use kinematic- and pressure- based features to describe the drawing process. While highly accurate results have been reported in the literature, very little attention is paid to the problem of tuning the hyperparameters of classification models and the inclusion of the tested subject's personal information. These two problems constitute the scope of the present research.

In the study Fisher score and principal component analysis were used for feature selection in data preparation phase. In this research six classification models, that represent different classification approaches, were trained on Parkinson's disease diagnosed patients data. Cross-validation, grid and random searches were implemented to optimize and evaluate performance of the aforementioned classifiers. In the evaluation stage classifier model behavior was analyzed with classification metrics, such as accuracy, precision, sensitivity, specificity, f1-score and auc roc curve.

The main outcome of current research is a framework that executes feature selection and provides hyperparameter tuning with performance evaluation for different classification models. It was demonstrated that tuning of the hyperparameters significantly affects model's prediction. Some of the best results show that accuracy increased from 76.7% to 92.4%, precision from 68.4% to 96% and f1 score from 65.7% to 80.8%.

Inclusion of patient's additional data had a positive impact on model's behavior. Generally, extra features combinations had much better overall performance than initial ones. After optimization process extra features had the highest metric scores almost in every fine-motor test.

The thesis is in English and contains 44 pages of text, 5 chapters, 11 figures, 8 tables.

Hüperparameetrite häälestamine masinõppel baseeruva Parkinsoni tõve diagnostika jaoks

Annotatsioon

Käesoleva lõputöö peamine eesmärk on parandada Parkinsoni tõve diagnostika joonistustestide analüüsi kvaliteeti. Suurem osa masinõppepõhistest tulemustest kasutab joonistamisprotsessi kirjeldamiseks kinemaatilisi ja rõhupõhiseid tunnuseid. Ehkki kirjanduses on esitatud väga täpseid tulemusi, pööratakse klassifitseerimismudelite hüperparameetrite häälestamise raskustele ja testitava isiku isiklike andmete kaasamisele väga vähe tähelepanu.

Uuringus kasutati andmete ettevalmistamise etapis funktsioonide valimiseks Fisheri skoori ja peakomponentide analüüsi. Selles uuringus treeniti Parkinsoni tõvega diagnoositud patsientide andmete põhjal kuut klassifikatsioonimudelit, mis esindavad erinevaid klassifitseerimisviise. Eespool nimetatud klassifikaatorite toimivuse optimeerimiseks ja hindamiseks viidi läbi ristvalideerimine, ammendavad ja juhuslikud otsingud. Hindamisetapis analüüsiti klassifikaatori mudeli käitumist klassifitseerimismõõdikutega.

Uurimustöö peamine tulemus on raamistik, mis teostab tunnuste valiku ja pakub hüperparameetrite häälestamist koos toimivuse hindamisega erinevate klassifikatsioonimudelite jaoks. Tulemustest tuleneb, et hüperparameetrite häälestamine mõjutab oluliselt mudeli üldist headust. Mõned parimad tulemused näitavad, et *accuracy* kasvas 76,7%-lt 92,4%-ni, *precision* 68,4%-lt 96%-ni ja *f1-score* 65,7%-lt 80,8%-le.

Patsiendi täiendavate andmete lisamine mõjutas positiivselt mudeli käitumist. Üldiselt oli lisatunnuste kombinatsioonide üldine jõudlus palju parem kui algsete andmete oma. Pärast optimeerimisprotsessi näitasid lisatunnused kõige kõrgemad tulemused peaaegu igas peenmotoorses testis.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 44 leheküljel, 5 peatükki, 11 joonist, 8 tabelit.

List of abbreviations and terms

AUC	Area Under The Curve
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
HC	Healthy Control
IDE	Integrated Development Environment
KNN	K-Nearest Neighbors
PC	Personal Computer
PCA	Principal Component Analysis
PD	Parkinson's Disease
ROC	Receiver Operating Characteristics
SVC	Support Vector Classification
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
TPR	True Positive Rate

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to statistical evidence Parkinson's disease is one of the most widely spread neurodegenerative disorders worldwide. At the present moment there is no known cure from the disease. Nevertheless, early diagnosis and proper therapy may relieve the patients from the symptoms and enhance the quality of everyday life.

Parkinson's disease is complex neurodegenerative disorder that mostly affects human muscle movements. Diagnosed patients experience variety of symptoms such as tremor, stiffness of the limbs, slowness in movements (bradykinesia) [1, 2], problems with coordination and balance. Therefore, recent studies endorse handwriting and drawing as a proven biomarker for Parkinson's disease, since these processes demand high synchronization of many muscles and appear difficult for Parkinson's disease patients [3, 4, 5].

Fine-motor tests have been used to detect neurological disorders for over a century. Over this period of time many different versions have been presented that investigate drawing of patterns or analyze handwriting of the sentences. While tests are generally performed with plain paper and pencil, advances in technology have also led to the use of tablet PCs [6]. With their help, it is also possible to record kinematic, geometric and pressure parameters [1, 7, 8, 9] that are invisible for the naked eye. For example, writing and drawing speed, acceleration, pen pressure, angles, etc. Machine-based analysis of the features calculated from these parameters helps to reduce the subjectivity of doctors due to different experiences in diagnostics and provides additional information for decision-making. In order for this method to be used in medical practice, it is important to evaluate and optimize the accuracy

of the algorithms used. While the analysis of fine-motor tests based on machine learning is a widespread trend, the selection and tuning of the hyperparameters of the models used has not received much attention in published research. Moreover, personal communication and some preliminary results have indicated that patient's additional information that can provide insight on patient's lifestyle can also benefit classification processes.

Machine learning algorithms have hyperparameters that allow one to manipulate the behavior of the model to specific dataset. These parameters are specified during model configuration and it can be challenging to foresee the best values of a given algorithm on a given dataset. Therefore, it is common to use grid and random search strategies for different hyperparameter values [10].

Present thesis is a part of bigger research series in Tallinn University of Technology, Tallinn University and University of Tartu. This research was conducted to study how selection and tuning of hyperparameters affect classifier's performance, since it has not been a main focus in previous studies before.

Primary goal of present thesis is to analyze whether and what effect the tuning of hyperparameters of machine learning models on Parkinson's disease diagnosed patient data has on the final results. Also, investigate whether and what effect the inclusion of patient personal data (such as gender, age, dominant hand, weight and height) for model training has.

The initial data is pre-processed and additional data is integrated with it. Feature selection is applied to reduce high dimensionality of the data using Fisher score and principal component analysis. After completing data preparation phase hyperparameters of six classification algorithms were looked into to define hyperparameters with the biggest impact on models' optimization. Selected hyperparameters are used in cross-validation, grid and random searches to evaluate the performance of each model based on six evaluation metrics.

Main outcome of current research is a framework that is capable of pre-processing given data, providing feature selection option and hyperparameter tuning with performance evaluation for various classification models. Also, make it easy to integrate patient's personal information with initial dataset and make it possible to analyze the effect of including personal data on model training. As a result, it was con-

firmed that the tuning of the hyperparameters may significantly improve classifier's behavior and inclusion of patient's additional data also has a positive impact on classification.

Thesis is organized as follows. *Chapter 1* consists of background description and problem statement. *Chapter 2* gives insight on data preparation, data description and feature selection. *Chapter 3* describes used classification algorithms and how to validate their performance. *Chapter 4* explains the optimization processes and describes obtained results. *Chapter 5* offers discussion about acquired results.

Chapter 2

Data Preparation

2.1 Data Description

At the initial stage of the research, data computed from the raw signals was given. Several drawing tests are included in the dataset. Each row of the data represents a Parkinson's disease diagnosed patient or an age-matched healthy control subject. For each performed test case there are features that stand for different kinematic, pressure and geometric properties. To effectively store, access and process data, given Excel file was converted into Pandas dataframe. Example of the features dataframe is shown on Table 2.1, where HC is a healthy control subject and PD is a diagnosed patient.

test name	label	feature 1	...	feature n
spiral	HC01	36.2351	...	5135.421
spiral	HC02	43.22	...	544.0021
spiral	HC03	12.0	...	3590.1
spiral	PD01	107.3	...	3525.74
spiral	PD02	153.044	...	2535.7

Table 2.1: Data Description — Dataframe Example

Aforementioned data needs to be *cleaned* and divided into subsets by category. Then, each drawing test's dataset needs to be rectified and implemented in classifiers' training for hyperparameter tuning processes.

2.1.1 Drawing Tests

Over the course of one hundred years various drawing and writing tests have been developed to diagnose different disorders and assess their progress [11].

The present research is based on the information conducted from fine-motor skill tests that were performed by Parkinson's disease diagnosed patients and control subjects of the same age group. Fine-motor tests were performed using iPad and digital pen. During the test different kinematic, geometric and pressure parameters were recorded and afterwards used to define features that this research uses to train classification algorithms on.

A subset of conducted fine motor tests is shown below.

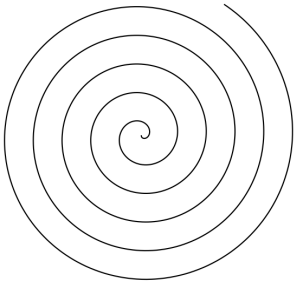

Figure 2.1: Drawing Tests — Archimedean Spiral

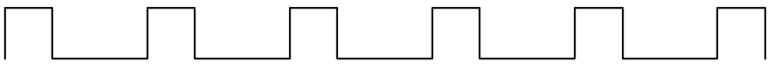

Figure 2.2: Drawing Tests — Luria's Alternating Series Test 1

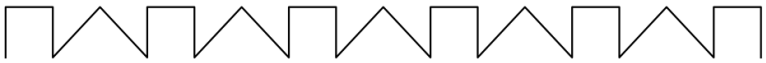

Figure 2.3: Drawing Tests — Luria's Alternating Series Test 2

2.2 Additional Data

Patient's personal information integration and analysis have crucial part in the present thesis. It is speculated that patient's personal information that helps to describe or give insight on person and his/her lifestyle may also assist with the diagnosis of Parkinson's disease. In order to study whether and how additional data affects the diagnosis, extra data needs to be integrated into the classification workflow. However, the extra data will only be implemented together with the most appropriate variables that were obtained during feature selection. It is essential to validate whether additional information is more relevant than already existing attributes.

In present study gender, age, dominant hand, weight and height attributes were integrated. It was demonstrated in [12] that gender has a significant impact on gross-motor skill analysis for Parkinson's disease diagnostics. Also, age is considered a natural factor that causes degradation of fine motor movements.

Feature	Unit	Type
gender	male=0, female=1	Integer
age	years	Integer
dominant hand	right=1, left=0	Integer
weight	kg	Float
height	cm	Float

Table 2.2: Additional Data — Extra Features Description

After successful data integration, combinations of initial and extra feature will be implemented in classification processes.

2.2.1 Data Imputing

Incomplete data might not be suitable for some classification algorithms and needs to be pre-processed to be applied [10]. There are several approaches to handle absent values in datasets and the decision must be based on the essence and type of given data.

In the current research missing values will be imputed with mean strategy that works well with small datasets. Chosen method is perfect for current problem, since initial data is relatively small with only numeric values and after integrating extra features will also contain a lot of missing values. The main aim is to maintain existing *records* and avoid information loss.

2.3 Feature Selection

Feature is a property that helps to describe and analyze observable object. In datasets features, also known as "attributes" or "variables", appear as columns and number of attributes may vary widely. In this research the initial data is high *dimensional*, there are **142** initial variables to consider with. Therefore, to increase the effectiveness of classification algorithms it is crucial to find features that are the most informative and discriminating [13].

Feature selection is one of the most important steps in data pre-processing process. In fact, real data may contain features of varying relevance. Attributes that do not contribute much to class labeling may harm the accuracy of classifiers and be the source of computational inefficiency. Feature selection helps to eliminate redundant features without dropping classifier's performance. Additionally, feature selection reduces the probability of *overfitting* - a problem that occurs when model becomes too attuned to the data it was trained on and is not appropriate for any other dataset. Also, training algorithms become faster on hardware level due to decreased number of operations needed in classification process [13, 10].

Feature selection methods for classification can be categorized into three primary types [13].

1. *Filter models*: The quality of a feature or a subset of features is evaluated using some mathematical criterion. Irrelevant features are filtered out using based on the criterion.

2. *Wrapper models*: It is assumed that a classifier is available to evaluate how well the model performs with a particular subset of features. The relevant set of features is determined by wrapping a feature search algorithm around the classification model.

3. *Embedded models*: The solution to a classification model often contains useful hints about the most relevant features. These features are isolated, and the classifier is retrained on the pruned features.

In this research Fisher score method that belongs to filter models was selected and implemented.

2.3.1 Fisher Score

Charu C. Aggarwal's definition: "The Fisher score is naturally designed for numeric attributes to measure the ratio of the average *interclass* separation to the average *intraclass* separation. The larger the Fisher score, the greater the discriminatory power of the attribute."

Let μ_j and σ_j, respectively, be the mean and standard deviation of data points belonging to class j for a particular feature, and let p_j be the fraction of data points belonging to class j. Let μ be the global mean of the data on the feature being evaluated. Then, the Fisher score F for that feature may be defined as the ratio of the interclass separation to intraclass separation:

$$F = \frac{\sum_{j=1}^{k} p_j(\mu_j - \mu)^2}{\sum_{j=1}^{k} p_j \sigma_j^2} \tag{2.1}$$

The numerator quantifies the average interclass separation, whereas the denominator quantifies the average intraclass separation. Aggarwal states that the attributes with the largest value of the Fisher score might be more suitable for classification algorithms [13].

2.4 Dimensionality Reduction

In real data sets, a significant number of correlations exist among different attributes. In some cases, hard constraints or rules between attributes may uniquely define some attributes in terms of others. Frequently, these correlations are imperfect and hard to detect. However, aforementioned correlations and constraints imply the existence of some subsets of the dimensions that can be used to predict the values of the other dimensions. Also, it is notable to mention that high dimensionalty raises the risk of encountering *curse of dimensionality*. A problem that deals with analyzing and organizing data in high-dimensional spaces [13].

There are several methods that are used for correlation analysis and dimensionality reduction. *Principal component analysis* (PCA) and singular value decomposition (SVD) are two natural procedures that help to automate the way of dimensionality reduction with axis rotation. Both methods are closely related, but not identical on definition level. SVD is a more general framework and can be used to perform PCA as a special case. However, the concept of principal component analysis is intuitively easier to understand [13].

In the present research PCA will be applied to discover possible hidden correlations among initially given and extra features. Built-in PCA functionality of *scikit-learn* library will be implemented and transformed data will be used on classification models.

2.4.1 Principal Component Analysis

Principal component analysis is a procedure that is used to reduce the dimensionality of datasets that have great amount of features. The goal of PCA is to transform initial dataset into a smaller one and preserve as much information as possible.

PCA method applies data's covariance matrix to compute principal components. Principal components are the attributes with the greatest possible variance. In order to find principal component PCA performs axis-rotation to maximize the sum of the squared distances of the attribute values projected in the *n-th* dimension. The following principal component is calculated in the same way, but has to be perpendicular to the previous one [13, 10].

The principal components can be organized by their importance by ranking *eigenvectors* in order of their *eigenvalues*. Eigenvectors and eigenvalues come in pairs and their number is equal to the number of the data dimensions. Eigenvectors of covariance matrix are the directions of the axes with the most variance. Eigenvalues are the variance coefficients attached to eigenvectors, that describe the amount of information in principal component. Later, the eigenvalue of each principal component is divided by the sum of eigenvalues to compute the percentage of variance accounted for by each component.

Chapter 3

Classification

3.1 Classifiers

Classification algorithms are trying to solve labeling problems. Classification is a type of supervised learning where data is already labeled and important attributes are separated into well defined categories. The data is fed to the algorithm that knows what features are vital and thus has a *ground truth* for classification [10].

The data is divided into training and testing sets. The testing process helps to analyze how well classifier predicts the labels after being given before unseen dataset. It is crucial to avoid testing on the data that classifier was trained on, since the outcome would be extremely biased due to model's knowledge of the patterns [10].

Scikit-learn library, that was heavily used in the present research, provides various classifiers that are easy to implement. For the particular problem that this research tries to solve following classifier were applied.

3.1.1 Decision Tree

Decision Trees are classification models that implement *hierarchical* tree-like structure for classification process. Tree nodes represent decisions. Each decision is based on one or more features in the training data and is referred as *split criterion*, that divides the training data into two or more parts. Combination of split criteria and nodes above it define subset of the data space [13].

The induction algorithm uses *internal nodes* to navigate and *leaf nodes* to label

dominant class. The root node is a special case of internal node that corresponds to the whole feature space. The decision tree starts with the full training dataset and recursively partitions the training data in each node. Partitioning is based on splitting criterion and the growth is stopped based on *stopping criterion*. For example, all data samples belong to one class [13].

The decision tree classifier has several advantages. Unlike most machine learning models, there is almost no need for data preparation. Classification's computational speed is high due to cost of deduction being logarithmic. However, the decision trees are also vulnerable to overfitting [10].

In this research during the tuning of classifier's hyperparameters main focus was on splitting criterion and leaf nodes. Scikit-learn's *DecisionTreeClassifier* was integrated into main framework and it's performance was studied on the previously pre-processed data.

Hyperparameters shown in Table 3.1 were chosen for optimization processes in the present research.

Hyperparameter	Value
criterion	{"gini", "entropy"}, default="gini"
splitter	{"best", "random"}, default="best"
`max_features`	int, float or {"auto", "sqrt", "log2"}, default=None
`max_depth`	int, default=None
min_samples_split	int or float, default=2

Table 3.1: DecisionTreeClassifier — Hyperparameters

Criterion defines the function to measure the quality of a split by the Gini impurity or the information gain with "entropy". **Splitter** sets the strategy for node splitting. **Min_samples_split** is the minimum number of samples that are needed to split an internal node. **Max_depth** controls the maximum depth of the tree and if it is not defined, then nodes are expanded until all leaves are pure or the minimum number of samples for each node is reached. **Max_features** parameter is used to consider number of features when selecting the best split [10].

3.1.2 Random Forest

Random Forest classifier consist of numerous decision trees that operate as an ensemble. Each decision tree in the random forest performs it's own prediction computations and then the class with the most votes gets to be chosen as a model's prediction [13].

The low correlation between individual decision trees is the reason why random forest outperforms each model's separate prediction. The greater the number of uncorrelated trees in the random forest model the higher is the chance to accurately predict the label. The majority vote helps to bypass errors, that are made by a few single decision trees, and keep moving in the right direction. However, features still have to possess enough information in order to correctly predict the class [10, 13].

Each decision tree's behavior is regulated by *bagging*. Bagging is a process that randomly samples data with replacement and the size of the training subset stays the same. However, using decision trees only with bagging is not enough, since it is statistically likely that split choices at the top levels of the tree remain roughly equal to bootstrapped sampling [13]. Therefore, correlation between individual models persists and the handling of the error reduction is limited.

Additionally, *random-split selection* approach is applied to increase randomness of the single decision tree. An integer parameter q, that represents the size of the attributes' subset, regulates the node's splitting execution in the tree. Using small values of q reduce the correlations in the random forest. On the other hand, bigger q values enlarge single model's accuracy, but also result in correlated trees. Therefore, the goal is to achieve the best possible trade-off. The method is inefficient if the dimensionality is small and then the combinations of the attributes at each node's split are introduced. The combinations help to operate with the small number of attributes [13].

The *RandomForestClassifier* algorithm introduced in Scikit-learn is flexible and has many hyperparameters to operate with. During the research the centre of attention was the number of trees in the model. Main hyperparameters used during optimization can be seen in the Table 3.2, where **n_estimators** defines the number of decision tress in the model and other hyperparamters characterize the behavior of decision trees [10].

Hyperparameter	Value
n_estimators	int, default=100
criterion	{"gini", "entropy"}, default="gini"
`max_features`	int, float or {"auto", "sqrt", "log2"}, default="auto"

Table 3.2: RandomForestClassifier — Hyperparameters

3.1.3 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm that assumes that classes with similar attributes exist nearby. KNN is a type of *instance-based* learning that simply stores instances of the training data. The algorithm computes distances between test instance and its k-nearest neighbors. Then the dominant label from these k-nearest neighbors is selected as an applicable one [10].

The optimal k value highly depends on the features of the data and may be determined by calculating error rate for different k values or using *leave-one-out cross-validation* method for the training subset. Usually, a larger k suppresses the effects of noise, but makes the classification boundaries less distinct. The performance is greatly affected by the distance function that searches for the neighbors [13].

Advantages of the current algorithm is that it is simple and adjustable. It can be used to solve both classification and regression problems. Nevertheless, KNN gets significantly slower when implemented on the data with high dimensionality.

Scikit-learn's implementation of KNN classifier *KNeighborsClassifier* offers high customization options. In the present research high attention was given to the choice of the algorithm and the k value. The performance speed of the KNN was not an issue since the data dimensionality was previously reduced to a few observable features.

The Table 3.3 shows used hyperparameters and their default values. **N_neighbors** represents the number of neighbors used to define a label. **Weights** is a function used in prediction that defines point's relevance. **Algorithm** parameter is an approach used to compute the nearest neighbors. *BallTree* and *KDTree* algorithms use **leaf_size** in their implementations and the size have significant impact on the

classifier's speed performance. The distance calculation is decided by **metric**. In this research "euclidean", "manhattan" and "minkowski" distance metrics were used [10].

Hyperparameter	Value
n_neighbors	int, default=5
weights	{"uniform", "distance"} or callable, default="uniform"
algorithm	{"auto", "ball_tree", "kd_tree", "brute"}, default="auto"
leaf_size	int, default=30
metric	str or callable, default="minkowski"

Table 3.3: KNeighborsClassifier — Hyperparameters

3.1.4 AdaBoost

AdaBoost uses boosting approach to make a strong learner form a sequence of *weak learners* and thus reducing the overall bias. In boosting a weight is assigned to each training instance and with each iteration the weights are modified depending on classifier performance. The future models are build from the results of the previous ones using the same weak learning algorithm. The final verdict is produced through a weighted majority vote or a sum [10].

During boosting iterations weight is added to each of the training samples. In the initial iteration weak learners train on the original data and with each following iteration the weights are modified. The algorithm forces weak learners to concentrate on the training examples that were miscalculated by increasing their weight and decreasing weight of correctly predicted ones. For better variance reduction the weights must be modified less aggressively between rounds. The approach finishes when the classification accuracy is 100% or the maximum number of iteration rounds is reached. Some implementations of the algorithm reset the weights if accuracy falls below 50% since it means that the classifier performs worse than guessing classifier [13, 10].

Boosting may inappropriately overtrain the model due to excessive noise in the data. The reason behind under-performance is that boosting mistakes noise for the

bias component of instances near the *incorrectly modeled decision boundary* [13].

Scikit-learn's *AdaBoostClassifier* implements the algorithm known as *AdaBoost-SAMME*. The implementation allows to set base estimator using **base_estimator**, if "None" then *DecisionTreeClassifier* with **max_depth=1** is used. Also, modify the number of iterations at which boosting process is terminated with **n_estimators** and learning rate based on weights with **learning_rate**. There is a choice between SAMME discrete boosting and SAMME.R real boosting algorithms that selected by **algorithm** parameter. Moreover, a possibility to reproduce output across multiple function calls by defining random state using **random_state** [10].

Hyperparameter	Value
base_estimator	object, default=None
n_estimators	int, default=50
learning_rate	float, default=1
algorithm	{"SAMME", "SAMME.R"}, default ="SAMME.R"
random_state	int, RandomState instance or None, default=None

Table 3.4: AdaBoostClassifier — Hyperparameters

3.1.5 Logistic Regression

Logistic Regression is a linear model for classification that calculates the probability of an outcome. Logistic regression categorizes the target variable into a discrete category [10]. The algorithm's name comes from the core function it is build on - *logistic function*, also known as *sigmoid function*:

$$P = \frac{1}{1 + e^{-L}} \tag{3.1}$$

where:

- P is the probability of the logistic regression model for a individual example
- L represents the *logit function*, the logarithm of the odds.

A specified threshold value is used to determine the category of a particular test instance. For example, if threshold is set to 0.7 then all instances with probability above the threshold are categorized as class "A" and all other as class "B".

The classification algorithm is commonly used for *binary classification* problems, but can also be applied to multi-class classification problems [13]. Logistic regression can be classified into three types:

- **Binomial**: exist only two possible types of the dependent variables, such as "pass" or "fail", 0 or 1, etc.
- **Multinomial**: exist three or more possible unordered types of the dependent variables, such as "dog", "cat", "mouse".
- **Ordinal**: exist three or more possible ordered types of the dependent variables, such as "bad", "good", "excellent".

The *LogisticRegression* class from scikit-learn library can fit binary, One-vs-Rest or multinominal logistic regression with optional regularization. The class implements many solver algorithms to handle optimization problem. However, introduced solvers only support specific regularization norms [10].

Hyperparameters presented in the Table 3.5 were used for tuning processes. Hyperparameter named **solver** selects the algorithm for optimization. In this research only "liblinear" was used, since Parkinson's disease patient data is small and represents binary classification problem. Moreover, some of **penalty** parameters that specify the norm used in the penalization, are incompatible with most of the solvers making it impossible to perform exhaustive search hyperparameter tuning. **Intercept_scaling** defines the constant's value used in prediction and is efficient only when using "liblinear" solver. **C** is the inverse of regularization strength, where smaller values state stronger regularization. **Max_iter** sets the maximum number for iterations and **multi_class** defines classification method. In this research only "ovr" was used, since the algorithm selects it automatically if the data is binary [10].

3.1.6 SVC

Support Vector Classification (SVC) is another popular supervised learning algorithm for classification. SVC, or SVM (Support Vector Machine), tries to find optimal *hyperplane* in n-dimensional space to separate different classes [10].

Hyperparameter	Value
penalty	{"l1", "l2", "elasticnet", "none"}, default="l2"
intercept_scaling	float, default=1
C	float, default=1.0
solver	{"newton-cg", "lbfgs", "liblinear", "sag", "saga"}, default="lbfgs"
max_iter	int, default=100
multi_class	{"auto", "ovr", "multinomial"}, default="auto"

Table 3.5: LogisticRegression — Hyperparameters

Hyperplane is a function that is used to define the *decision boundary* between different labels. *Support vector points* are data points that are closer to the hyperplane and define it's position and orientation. The *margin* is the distance from hyperplane to support vector points. The greater the distance, the better is the class separation [13].

SVM can perform linear classification, but also is suitable for non-linear classification due to so-called *kernel trick*. Kernel method converts low dimensional input space into high dimensional input space and tries to find a way to separate the data [13].

The algorithm is used to classify numeric binary data. However, it can also operate on multi-dimensional dataset by converting it to binary form using conversion approaches [13]:

- **one-against-rest**: k different binary problems are created. Each problem corresponds to a class, thus making a total of k models.
- **one-against-one**: a training dataset is created for each of the $\binom{k}{2}$ pairs of classes and producing a total of *k*(k-1)/2* models.

The classifier is super effective when there are more features than training samples. SVM performs classification very well on complicated binary data. Despite being a powerful tool the algorithm is also hard to optimize and selecting the right kernel can be challenging. The computation performance downgrades as dataset gets larger [10, 13].

Scikit-learn's *SVC* class, one of many implementations of SVM algorithms in the library, was utilized. The algorithm is the most suitable for smaller data samples, because the fit time scales tremendously and might be impractical for large datasets. The one-vs-one approach is applied. Optimization provides five kernel types and additional hyperparameters to modify kernel functions.

In the Table 3.6 there are shown main hyperparameters that were used during classifier's tuning. **C** is the requlatization parameter that must be strictly positive. **Kernel** specifies the kernel type used in the classification algorithm. In this research focus was on "poly" and "rbf" kernel types, since these two showed the best results with the given dataset. For "poly", polynominal, kernel type there is additional hyperparameter called **degree** that is ignored by all the other kernels. **Shrinking** parameter helps to shorten the training time if the number of iterations is huge. However, it might have a negative impact on optimization speed, if stopping tolerance is large. **Class_weight** uses parameter C to adjust class weights, by default all classes have weight one. Parameter **coef0** is an independent term in kernel function and is only significant in "poly" and "sigmoid" [10].

Hyperparameter	Value
C	float, default=1.0
kernel	{"linear", "poly", "rbf", "sigmoid", "precomputed"}, default="rbf"
degree	int, default=3
shrinking	bool, default=True
class_weight	dict or "balanced", default=None
coef0	float, default=0.0

Table 3.6: SVC — Hyperparameters

3.2 Performance Evaluation

Evaluation of model's performance is crucial to succeed in classification. To study model's behavior the data needs to be divided into the training and the testing samples. Testing model on the training dataset is forbidden since the classification algorithm already knows the patterns and will be extremely bias. In the testing process the model tries to label before unseen data. After the predictions are made, the predicted and actual values are matched. The result of matching is computed into a metric to measure the performance of the model.

There are many different metrics that help to evaluate the model's behaviour. The most common metric is accuracy since everybody seems to know what it does. However, there are much better metrics to take into account when dealing with imbalanced classification. For example, recall and precision provide much more information about a classification model and how it acts.

In the current research classification algorithms are evaluated with accuracy, precision, recall, specificity, F1 score and AUC-ROC curve. Scikit-learn library provides easy implementation for all aforementioned metrics with a few lines of code [10].

3.2.1 Confusion Matrix

A confusion matrix, also known as an error matrix, is a table that is often used in machine learning to describe classifier's performance. The table on Figure 3.1 represents combinations of actual and predicted values. With the help of confusion matrix several useful evaluation metrics can be measured such as *accuracy*, *precision*, *recall*, *specificity*, *F1 score* and *AUC-ROC curve* [10].

- **True positives (TP)**: prediction is positive and actual is positive.
- **False positives (FP)**: prediction is positive, but actual is negative.
- **True negatives (TN)**: prediction is negative and actual is negative.
- **False negatives (FN)**: prediction is negative, but actual is positive.

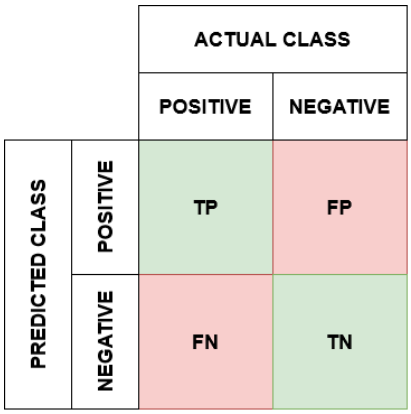

Figure 3.1: Evaluation — Confusion Matrix

3.2.2 Accuracy

Accuracy measures the fraction of correct predictions form 0 to 1. High percentage of accurate predictions indicates better performance. Accuracy is the most common metric to judge model's performance. However, in case of imbalanced datasets accuracy can be misleading and thus cause model's inaccurate evaluation.

$$Accuracy = \frac{true\ positives + true\ negatives}{true\ positives + false\ positives + true\ negatives + false\ negatives} \quad (3.2)$$

3.2.3 Precision

Precision measures the fraction of actual positives among those examples that are predicted as positive. The range is also 0 to 1 and a larger value indicates better prediction.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \quad (3.3)$$

3.2.4 Sensitivity

Sensitivity, also known as *recall* or *true positive rate*, measures the percentage of actual positives that are predicted as positive. Larger value is considered as a better predictive accuracy.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3.4}$$

3.2.5 Specificity

Specificity, or *true negative rate*, measures the proportion of negatives that are correctly recognized. Higher percentage of true negatives indicates better performance. Both sensitivity and specificity are widely used metrics in medical diagnostics.

$$Specificity = \frac{true\ negatives}{false\ positives + true\ negatives} \tag{3.5}$$

3.2.6 F1 Score

F1 score is a harmonic mean of precision and sensitivity. Both metrics are taken into account in the following equation:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.6}$$

3.2.7 AUC-ROC Curve

AUC (*Area Under The Curve*) ROC (*Receiver Operating Characteristics*) curve is one of the most vital evaluation metrics that helps to describe classification model's behaviour. Basically it tells how well model distinguishes between classes. In this research's context, the higher the AUC the better the model is at differentiating between Parkinson's disease diagnosed patients and test subjects.

The ROC curve is plotted with TPR (true positive rate) against the FPR (false positive rate) where TPR is on the y-axis and FPR is on the x-axis.

The range of AUC is from 0 to 1. The higher the value the better is separability. If model's prediction is completely wrong the value will be 0, which means that model predicts one class as another in all cases.

3.3 Model Validation

Cross-validation is a technique used in machine learning that helps to assure that model uses correct patterns from the data, to validate model's stability. Also, cross-validation has great impact on the tuning of hyperparameters. The basic idea behind the technique is to train model on the subset of the dataset and then evaluate the model on the complementary subset. The dataset is divided into the training subset, that is meant to learn patterns, and the testing subset, that is used to evaluate model's prediction capabilities [13, 10].

General steps of cross-validation:

1. Divide data into two portions
2. Use the first portion to train the model
3. Validate model on the second one

The segmentation of data greatly affects validation, especially if data is small. Testing model on small samples of data that does not accurately represent the training dataset may cause incorrect evaluations. There are several methodologies for model validation that help to reduce *underfitting* and *overfitting* [13]:

1. ***Holdout method*** - is the most basic one. The labeled data is randomly divided into two subsets. Usually the training data corresponds to two-thirds or more of the initial dataset and the remaining is used for testing. The method can be repeated numerous times to provide better estimation. The problem of this approach is that random variations might be overrepresented in the training data and underrepresented in the testing data. Therefore, repetition is suggested to minimize the error of the evaluation. Moreover, the data will not be used to the full potential since only subset of data is used in the model training.
2. ***K-Fold cross-validation*** - compensates for the lack of data samples. When there is not enough data to train a model, an underfitting problem may occur due to small training subset. By reducing the training data for validation there is a risk of losing vital patterns viable for accuracy and thus producing

label bias. K-Fold cross-validation provides plentiful of data for model training and validation testing. The data is divided into n subsets, one subset is used for testing purposes and *n-1* subsets are used for model training. Holdout method is repeated n times using each time a different subset for training. The error is averaged through all repetitions thus improving prediction accuracy. The solution remarkably reduces bias and variance, since most of the data information is used in both training and testing subsets.

3. ***Stratified K-Fold cross-validation*** - is a variation of K-Fold cross-validation technique that is designed to balance the percentage of labels in each fold. Thus making it good for classification problems.

4. ***Leave-P-Out cross-validation*** - leaves p data points out of training data and uses for validation. Process is repeated for all possible combinations and in case of large p sample can be computationally expensive. A particular case known as *leave-one-out* cross validation is more preferable since the number of combinations is equal to the number of original data points.

Scikit-learn library provides different data splitting strategies. In the current research *Stratified K-Fold* with predefined random seed was applied. Scikit-learn's algorithms, that implement cross-validation, select it as default if the target data is a binary or the estimator is a classifier. Also, it is a good choice when dealing with small dataset and helps to balance class ratio in each stratified fold [10].

Chapter 4

Hyperparameter Tuning

4.1 Methods

Initially given data consists of numerous calculated features that were obtained from various fine motor tests on iPad tablets. In the initial stage of data preparation - features will be transformed from excel file into Python dataframe object and duplicate test cases will be removed. Then, feature selection will be performed with Fisher scoring algorithm for each drawing test to determine which features have the biggest weight during classification process. Later, patient's personal information will be added to the initial dataset. Unfortunately, additional features' dataset is incomplete, since not all information was provided. Therefore, missing values will be imputed in order to save size of already small data. Lastly, principal component analysis will be performed on combinations of the best selected features and extra features to study additional information's relevance on classification.

After completing pre-processing and transformation phase, the data will be ready to be implemented on various classifiers to discover the best performing classification models. Primary goal of current research is to examine how tuning of hyperparameters affects classification. Secondary one is to investigate whether patient's additional information has effect on diagnosis of Parkinson' disease. The current research's dataset only has two labels standing for a Parkinson's disease diagnosed patient and a test subject, HC and PC respectively. Therefore, it can be considered as a standard *binary classification* problem that can be solved using machine learning algorithms.

In this research six classification algorithms were implemented and respective hyperparameters studied. The research process was performed using k-fold cross-validation and search techniques to estimate hyperparameter's impact on label prediction. Performance was evaluated based on the results obtained from cross-validation with classifier's default hyperparameter settings. Not all hyperparameters are equally important, they highly depend on given data and the problem that classifier tries to solve. Therefore, hyperparameters with the biggest effect were looked into. In parallel, patient's additional information was examined on the same models using same search techniques.

Research, analysis and tuning of hyperparameters was performed using Python programming language and PyCharm IDE. Also, Jupyter Notebook was implemented to make interactive graphs. Following open-source Python libraries were extensively used on different study stages:

- NumPy and Pandas — for data effective storage and manipulation
- Matplotlib, Plotly and Seaborn — for figure plotting
- Scikit-learn — for training and validation of numerous classifier models

4.1.1 Cross-Validation

As was mentioned in *Chapter 3.3* cross-validation is a crucial part of the model building process. The *cross_validation* function from scikit-learn library allows to specify multiple metrics for evaluation.

The algorithm supports different data splitting strategies including *StratifiedK-Fold* that was used in this research. *StratifiedKFold* is a variation of standard k-fold method, but returns stratified folds where each set contains approximately the same percentage of samples of each target class as the complete set [10].

The initial Parkinson's disease patients data was not arbitrary. Therefore, as it was suggested in scikit-learn's documentation **shuffling** was enabled to get more meaningful results. The **random_state** parameter was set to 42 to preserve same folds across all iterations.

Initial cross-validation using six classifiers with default hyperparameters for each test case was executed. The models were validated on different combinations of top performing features based on Fisher scoring and patient's additional features. In

this research top five features for each test case were selected and then last one or two features were swapped with additional ones. In total there were 16 collections of features for each test instance.

4.1.2 Grid Search

Grid search is an exhaustive hyperparameter tuning technique that searches for the best parameters defined by some metric score. Scikit-learn's *GridSearchCV* generates all the possible combinations of parameter values for given dataset, evaluates them and then the best hyperparamter values are retained [10]. Just as in *cross_validate* method *StratifiedKFold* with fixed random state was applied.

The Figure 4.1 illustrates Decision Tree's grid search precision score results. Three dimensions represent **max_depth**, **max_features** and **min_samples_split** hyperparamters and each sphere is a possible combination. The bigger the sphere the larger is the precision score. The Figure clearly indicates that by lowering **max_depth** value the score increases. Therefore, it is safe to say that some hyperparameters are more important than others and even by the slightest modification of one parameter whole outcome could be different.

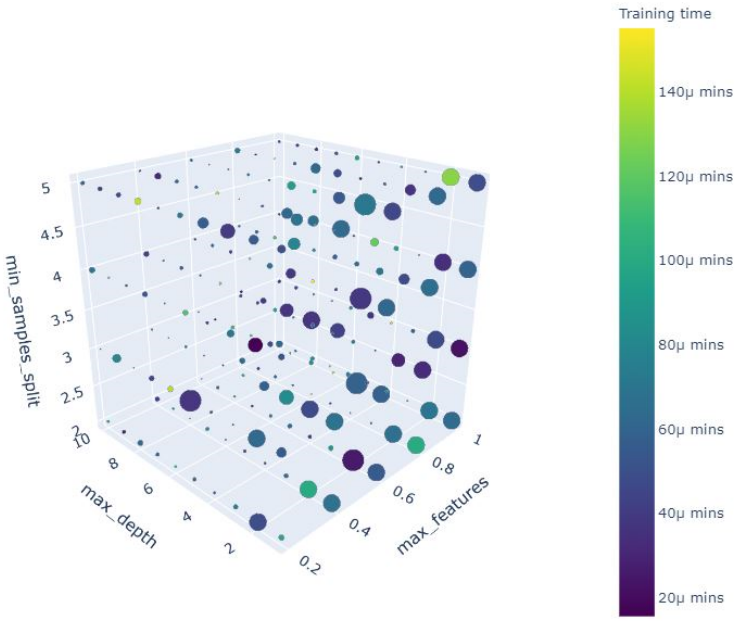

Figure 4.1: Grid Search — Decision Tree's Precision

4.1.3 Random Search

Random search is an alternative method for hyperprameter optimization. *RandomizedSearchCV* executes randomized search over distribution of possible hyperparameter values for each iteration. Random search helps to include more hyperparameters and values since the computational cost can be chosen independently [10]. Computational budget, or sampling iterations, is specified using the **n_iter** parameter. Sampling is done using a dictionary, similar to *GridSearchCV's* one. *StratifiedKFold* was applied once again.

Figures 4.1 and 4.2 display the main difference between grid and random searches. While exhaustive search provides all possible combinations, it also requires a lot of resources. On the other hand, randomized search implementation is able to manage computation costs, but might not be able to find best hyperparameter values to maximize the scores.

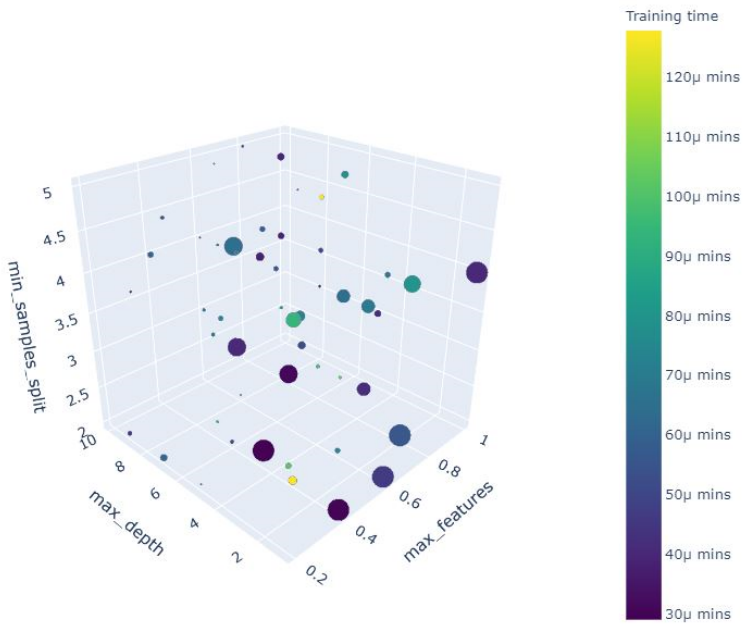

Figure 4.2: Random Search — Decision Tree's Precision

4.2 Results

The results obtained from the experiment indicate that most combinations with extra features perform much better than original top five attributes. For example, Decision Tree classifier's performance scores shown in Figure 4.3 make it clear that original features combination, that is the second from the top, has lower values than most of the others. For particular test replacing "slopes_mass" with "dom hand" feature increased the accuracy score from 0.727 to 0.804, precision from 0.684 to 0.848, f1 score 0.662 to 0.746 and other metrics grew as well.

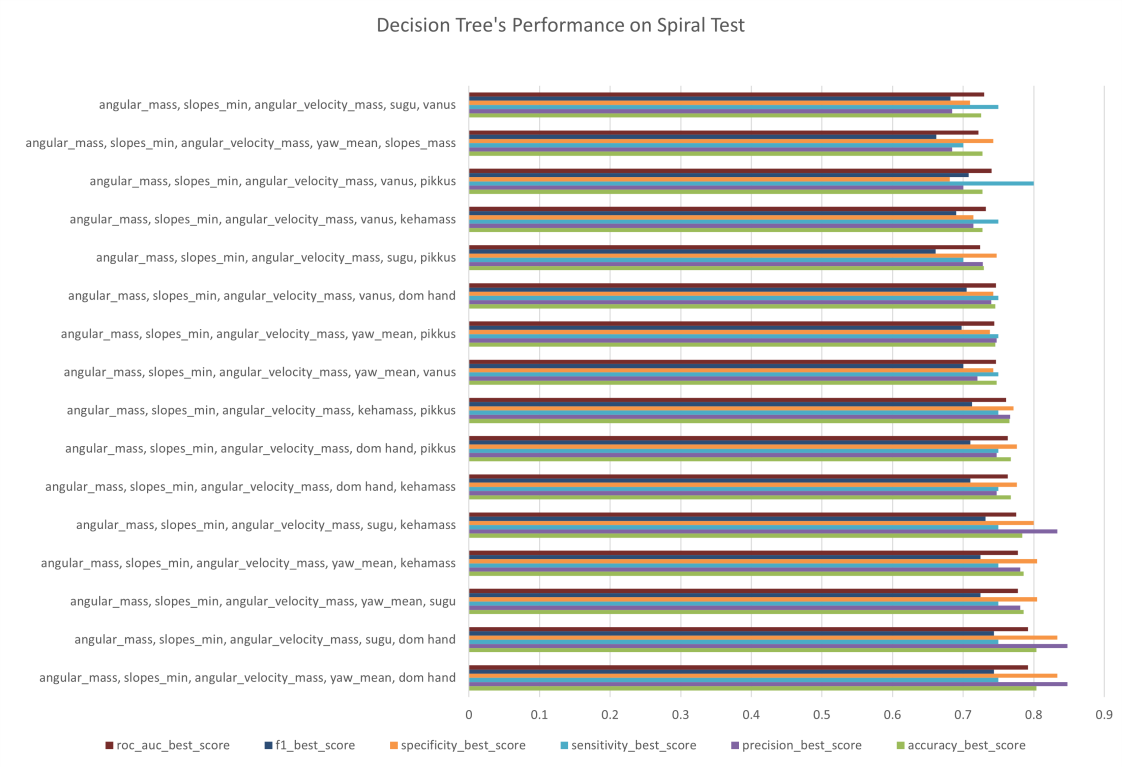

Figure 4.3: Decision Tree — Default Performances Scores

Figure 4.4 shows that variance between combinations in *RandomForestClassifier*, *DecisionTreeClassifier* and *AdaBoostClassifier* results are much notable than in *SVC*, *LogisticRegression* and *KNeighborsClassifier* algorithms' outcomes. Means, that are marked as crosses, and medians, as thick bars, of score metrics state that attribute combinations have more impact on algorithms such as AdaBoost, Random Forest and Decision Tree classifiers due to inner computations that are based on feature values.

Tuning of the hyperparameters certainly increased overall performance of all

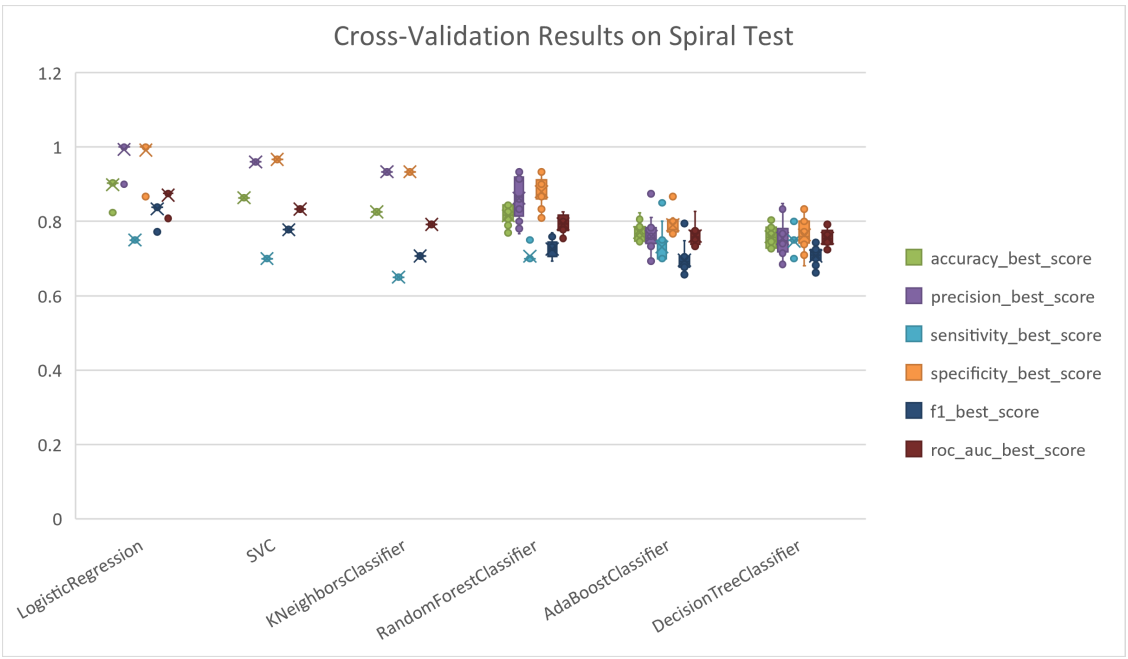

Figure 4.4: Cross-Validation — Classifiers' Performances

classification algorithms. For example, Decision Tree's accuracy on spiral test case went from 0.727 to 0.882 for initial top five features and to 0.904 for new combination of attributes. Applying PCA transformation on the training dataset increased accuracy even more, raised it up to 0.924, and other metrics have significant grow as well. Top scores were achieved by including dominant hand feature.

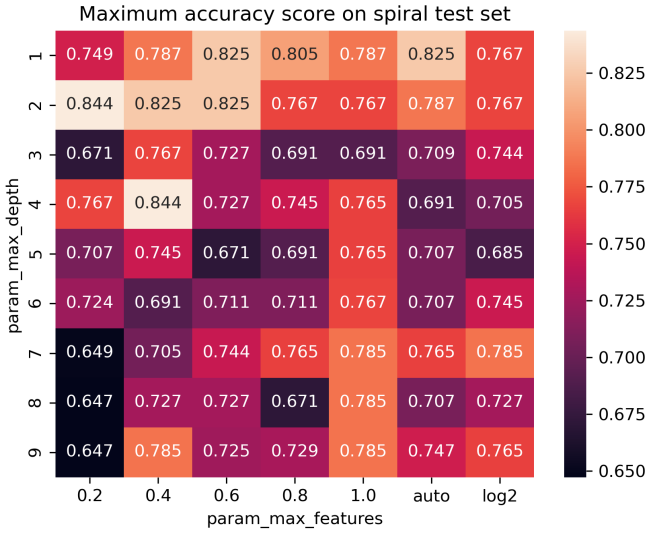

Figure 4.5: Grid Search — Decision Tree's Accuracy

As for Random Forest classification best scores were also achieved after perform-

ing PCA transformations. Cross-validation with default hyperparameters on initial features gave accuracy of 0.807 and precision of 0.814. After optimization new best scores across all combinations were 0.924 and 0.96 respectively. Combinations, that included original attributes, dominant hand and height, were the most successful ones.

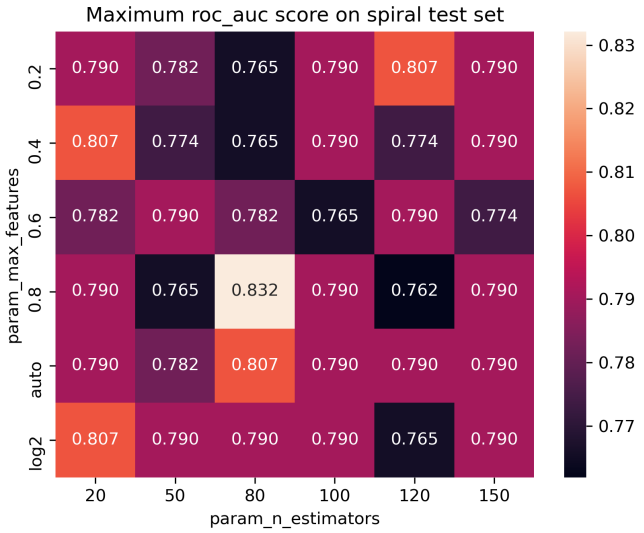

Figure 4.6: Grid Search — Random Forest's AUC-ROC Curve

KNN results on default settings were identical for all feature combinations. This means that implemented algorithm in scikit-learn library for particular classifier most likely uses only features with the most information for classification. The outcome was 0.825 for accuracy and 0.933 for precision, but with low sensitivity score of 0.65 points. After optimization all metric scores grew, recall increased to 0.8 points.

AdaBoost classifier's scores with default hyperparameters and original features were all around 0.75 points. Better attributes with PCA transformed data gave up to 0.824 accuracy and 0.874 precision points and lifted other metrics as well. Dominant hand, gender and height were the most frequent attributes in top lists.

Logistic Regression's results were also almost identical when default settings were used. However, top five features showed lower scores than other combinations. It is possible that original features caused overfitting. The lowest accuracy score was 0.824 and the highest 0.904 points. Hyperparameter tuning had no effect on improving the scores. PCA transformation had a negative impact on overall performance.

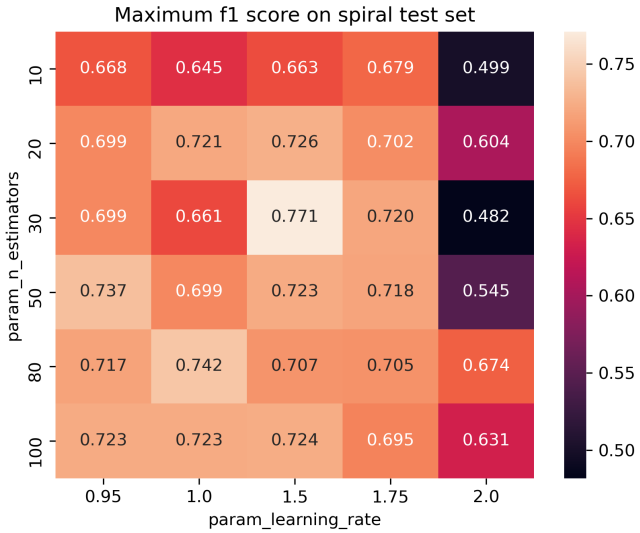

Figure 4.7: Grid Search — AdaBoost's F1 Score

SVC classifier's performance with default settings were 0.864 for accuracy and 0.96 fro precision with 0.7 for sensitivity. Optimization helped to slightly increase overall behavior.

Chapter 5

Conclusion

Primary goal of present thesis was to analyze whether and what effect the tuning of hyperparameters of machine learning models for Parkinson's disease diagnosed patients data has on the final results. Also, investigate whether and what effect the inclusion of patient's personal data (such as gender, age, dominant hand, weight and height) has on the model performance.

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

In the beginning of the data pre-processing phase initial raw data was studied and Fisher scoring algorithm was applied to filter out the most promising features. This method greatly reduced the data dimensionality and also decreased chance of overfitting.

Later patients' additional data was merged with initially given data. Missing values in the dataset were imputed using median strategy. The goal of imputing was to save as much information as possible.

To study how patient's additional data influences model training, combinations of previously selected top features and additional attributes were given to hyperparameter tuning algorithms. In detail, top five features of each test were selected and one to two least viable attributes were replaced by additional ones covering all combinations.

Additionally, principal component analysis was performed to inspect how the model's optimization is affected by it. The training data subsets of different feature combinations were transformed and then used in hyperparameter tuning processes.

The results obtained during optimization show that even slightest hyperparameter tuning may significantly change model's behavior. Overall goodness of all classifiers increased. Models, that had good accuracy, but other metrics were poor, enhanced their performance even further. Some of the best achieved scores were accuracy of 92.6% and precision of 96.7%.

As for patient's additional data, it is clear that including extra features helped to achieve better performance. In fact, performing cross-validation with default hyperparameter settings on additional features combinations showed greater metric scores than on initial attributes almost in every fine-motor test. Generally, extra attributes had also better results after hyperparamter tuning.

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

There are two possible directions to apply findings of the present study. The first one is to generalize its findings to other drawing and writing tests. The second one is to apply developed workflow to the case of gross-motor tests.

Acknowledgments

I would like to thank my supervisors Elli Valla and Sven Nõmm for their support, encouragement and patience through all the process of writing this thesis. I would also like to thank my family and friends for always being there for me.

Bibliography

[1] 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. ISSN 0933-3657. doi: https://doi.org/10.1016/j.artmed.2016.01.004.

[2] S. Nõmm, K. Bardõš, I. Mašarov, J. Kozhenkina, A. Toomela, and T. Toomsoo. Recognition and analysis of the contours drawn during the poppelreuter's test. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 170–175, Dec 2016. doi: 10.1109/ICMLA.2016.0036.

[3] S. Nõmm, A. Toomela, J. Kozhenkina, and T. Toomsoo. Quantitative analysis in the digital luria's alternating series tests. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–6, Nov 2016. doi: 10.1109/ICARCV.2016.7838746.

[4] Mathew Thomas, Abhishek Lenka, and Pramod Kumar Pal. Handwriting analysis in parkinson's disease: Current status and future directions. *Movement Disorders Clinical Practice*, 4(6):806–818, 2017. doi: 10.1002/mdc3.12552.

[5] S. Nõmm, K. Bardõš, A. Toomela, K. Medijainen, and P. Taba. Detailed analysis of the luria's alternating series tests for parkinson's disease diagnostics. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1347–1352, Dec 2018. doi: 10.1109/ICMLA.2018.00219.

[6] Sabrina Gerth, Thomas Dolk, Annegret Klassert, Michael Fliesser, Martin H. Fischer, Guido Nottbusch, and Julia Festman. Adapting to the surface: A comparison of handwriting measures when writing on a tablet computer and

on paper. *Human Movement Science*, 48:62 – 73, 2016. ISSN 0167-9457. doi: http://dx.doi.org/10.1016/j.humov.2016.04.006.

[7] C. Marquardt and N. Mai. A computational procedure for movement analysis in handwriting. *Journal of Neuroscience Methods*, 52(1):39 – 45, 1994. ISSN 0165-0270. doi: http://dx.doi.org/10.1016/0165-0270(94)90053-1.

[8] Sven Nõmm and Aaro Toomela. An alternative approach to measure quantity and smoothness of the human limb motions. *Estonian Journal of Engineering*, 19(4):298–308, 2013. ISSN 1406-0175.

[9] S. Nomm, A. Toomela, and J. Borushko. Alternative approach to model changes of human motor functions. In *Modelling Symposium (EMS), 2013 European*, pages 169–174, Nov 2013. doi: 10.1109/EMS.2013.30.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Joseph Jankovic. Parkinson's disease: clinical features and diagnosis. *Journal of Neurology, Neurosurgery & Psychiatry*, 79(4):368–376, 2008.

[12] Hedi Kähär, Pille Taba, Sven Nõmm, and Kadri Medijainen. Microsoft kinect-based differences in lower limb kinematics during modified timed up and go test phases between men with and without parkinson's disease. *Acta Kinesiologiae Universitatis Tartuensis*, 23:86–97, 2017.

[13] Charu C. Aggarwal. *Data Mining: The Textbook*. Springer Publishing Company, Incorporated, 2015. ISBN 3319141414, 9783319141411.

Appendix 1 — Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Eduard Vigula

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Hüperparameetrite häälestamine masinõppel baseeruva Parkinsoni tõve diagnostika jaoks", mille juhendaja on Elli Valla ja kaasjuhendaja on Sven Nõmm

    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

25.05.2021