NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

COMPUTER NETWORKING

MSc THESIS

Collision Avoidance System implementation based on EEG frequency Analysis using ML Techniques

Georgios N. Kiomourtzis
Leonidas V. Kouros

Supervisors: **Athanasia Alonistioti,** Associate Professor
Vasileia Magoula, PhD Candidate,
Nikolaos Koursioumpas, PhD Candidate

ATHENS
March 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΚΤΥΩΣΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη συστήματος αποφυγής σύγκρουσης στο χώρο μέσω ανάλυσης Δεδομένων Ηλεκτροεγκεφαλογράφου με χρήση Μηχανισμών Μηχανικής Μάθησης

Γεώργιος Ν. Κιομουρτζής
Λεωνίδας Β. Κούρος

Επιβλέποντες: **Αθανασία Αλωνιστιώτη,** Αναπληρώτρια Καθηγήτρια
Βασιλεία Μαγουλά, Υποψήφιος Διδάκτωρ
Νικόλαος Κουρσιουμπάς, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ
ΜΑΡΤΙΟΣ 2020

MSc THESIS

Collision Avoidance System implementation based on EEG frequency Analysis using ML Techniques

Georgios N. Kiomourtzis
S.N: M1501
Leonidas V. Kouros
S.N: M1581

Supervisors **Athanasia Alonistioti,** Associate Professor
Vasileia Magoula, PhD Candidate
Nikolaos Koursioumpas, PhD Candidate

March 2020

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη συστήματος αποφυγής σύγκρουσης στο χώρο μέσω ανάλυσης Δεδομένων Ηλεκτροεγκεφαλογράφου με χρήση Μηχανισμών Μηχανικής Μάθησης

Γεώργιος Ν. Κιομουρτζής
A.M: M1501
Λεωνίδας Β. Κούρος
A.M: M1581

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αθανασία Αλωνιστιώτη,** Αναπληρώτρια Καθηγήτρια
Βασιλεία Μαγουλά, Υποψήφιος Διδάκτωρ
Νικόλαος Κουρσιουμπάς, Υποψήφιος Διδάκτωρ

Μάρτιος 2020

ABSTRACT

In this thesis we describe the implementation of a collision avoidance system using electroencephalography (EEG) signals. EEG signals are gathered from the driver of a vehicle and transmitted through a mobile application service running in the background of the driver's mobile device.

Assuming the EEG signals of the driver's brain activity can be accessed by a mobile device, we use that data to classify the driver's eye state (open or closed). Our aim is to alert the targeted driver as well as neighbouring drivers of an eye closed state and a vehicle control loss probability.

The drivers' mobile devices have a bi-directional communication with an edge server, responsible for collecting the EEG data, and sending the appropriate alert signal to the devices after classifying the eye state of the drivers. In order to achieve that, the server uses different Machine Learning models, to which it is already trained with an appropriate dataset provided for the purposes of analogous projects.

Collected data and eye state classifications are also send to a Backhaul Server, responsible for storing all data in database entries and providing the Edge Server with the best Machine Learning model for data classification.

We present the design and the implementation of the architecture of the system and its composites, including the client-side mobile device installed application, the signal analysis and processing, as well as the generation of the best classification model by training some of the most well-known ML algorithms.

We managed to achieve a successful real-time communication between the mobile device and the server measuring a metric-defined score of 0.795 accuracy prediction with the use of Support Vector Machines classifier algorithm.

This undoubtedly allows a margin for improvement, shows however the potential of developing a reliable Collision Avoidance System through the use of advanced Machine Learning Classification models combined with the provision of enlarged sizes of EEG data for the ML models to be trained on.

SUBJECT AREA: Signal Processing

KEYWORDS: Internet of Things, signal processing, principal component analysis, machine learning

ΠΕΡΙΛΗΨΗ

Στο παρόν έγγραφο περιγράφουμε την υλοποίηση ενός συστήματος αποφυγής σύγκρουσης χρησιμοποιώντας σήματα ηλεκτροεγκεφαλογραφημάτων (EEG). Τα σήματα συλλέγονται από τον οδηγό ενός οχήματος και αποστέλλονται μέσω υπηρεσίας εφαρμογής κινητού τηλεφώνου, εκτελούμενης στο παρασκήνιο της συσκευής του οδηγού.

Υποθέτοντας ότι στα EEG σήματα μπορεί να υπάρχει πρόσβαση από μία κινητή συσκευή, χρησιμοποιούμε τα δεδομένα αυτά για να κατηγοριοποιήσουμε την κατάσταση των οφθαλμών του οδηγού (ανοιχτοί ή κλειστοί). Ο στόχος μας είναι να ειδοποιήσουμε τον υπό εξέταση οδηγό αλλά και τους γειτονικούς οδηγούς για πιθανή κατάσταση κλειστών οφθαλμών και απώλειας ελέγχου του οχήματος.

Οι κινητές συσκευές των οδηγών έχουν αμφίδρομη επικοινωνία με έναν edge server, υπεύθυνο για την συλλογή των EEG δεδομένων και την αποστολή του απαραίτητου σήματος ειδοποίησης στις συσκευές, αφού ταξινομήσουν την κατάσταση των οφθαλμών. Για τον σκοπό αυτό ο server χρησιμοποιεί διάφορα μοντέλα Μηχανικής Μάθησης, με τα οποία έχουν ήδη εκπαιδευτεί με αντίστοιχο σύνολο δεδομένων που παραχωρήθηκε για τους σκοπούς ανάλογων εργασιών.

Τα συλλεγόμενα δεδομένα και η ταξινόμηση της κατάστασης των οφθαλμών αποστέλλονται επίσης και σε έναν Backhaul Server, υπεύθυνο για την αποθήκευση όλων των δεδομένων σε εγγραφές βάσης δεδομένων και για την παροχή του καλύτερου μοντέλου Μηχανικής Μάθησης για ταξινόμηση δεδομένων.

Στην παρούσα διπλωματική παρουσιάζεται ακόμα και ο σχεδιασμός και την υλοποίηση της αρχιτεκτονικής του συστήματος και των συστατικών του, περιλαμβάνοντας την εγκατεστημένη στην συσκευή εφαρμογή, την ανάλυση και επεξεργασία των σημάτων καθώς και την παραγωγή του καλύτερου μοντέλου ταξινόμησης εκπαιδεύοντας μερικούς από τους πιο γνωστούς αλγόριθμους Μηχανικής Μάθησης.

Καταφέραμε να επιτύχουμε επικοινωνία σε πραγματικό χρόνο μεταξύ των κινητών συσκευών και του server, μετρώντας μέσω ορισμένων μετρικών αποτέλεσμα ακρίβειας 0,795 με τη χρήση του Support Vector Machines αλγόριθμου ταξινόμησης.

Αυτό αναμφίβολα επιτρέπει όριο για βελτίωση, δείχνει ωστόσο την δυναμική ανάπτυξης ενός αξιόπιστου Συστήματος Αποφυγής Συγκρούσεων μέσω της χρήσης εξελιγμένων μοντέλων ταξινόμησης Μηχανικής Μάθησης, σε συνδυασμό με την παροχή αυξημένου όγκου EEG δεδομένων για την εκπαίδευση των μοντέλων μηχανικής μάθησης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Σήματος

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ίντερνετ των πραγμάτων, επεξεργασία σήματος, ανάλυση κύριων χαρακτηριστικών, μηχανική μάθηση

This page intentionally left blank

CONTENTS

LIST OF FIGURES

LIST OF IMAGES

LIST OF TABLES

PREFACE

This thesis took place on behalf of the MSc course of the department of informatics and Telecommunications of the National And Kapodistrian University of Athens during the 2019-2020 semester.

1. INTRODUCTION

1.1 Scope of the current thesis

According to the European Statistics Service (Eurostat) there were more than 25000 road accident fatalities in Europe just in the year 2017 [1]. Although the number has decreased over the past years it is still above the European Union's target to reduce this figure to less than 16000 by the year 2020 [2].

On the other side of the Atlantic Ocean the National Center for Biotechnology Information of the United States of America (NCBI) reports that a considerable proportion of vehicle accidents occurs when drivers tend to fall asleep when driving under monotonous conditions [3].

A study by the NCBI involving drivers in Norway indicates that sleep or drowsiness was a contributing factor in 4% of all accidents, as reported by the drivers who were at fault for the accident, rising to a factor of almost 20% in night-time accidents [4].

In an attempt to contribute to minimizing sleep-related accidents the authors implemented a collision avoidance system alerting drivers tending to fall asleep as well as other drivers around them, by analysing brain waves measured with Electroencephalography signals.

Electroencephalography (EEG) is a field in medical science that involves the study of the brain's electrical function [5]. For many applications requiring human input, measuring brain activity can be of substantial benefit. For instance, brain stimuli have been used as an input mode for computer games, to track emotions, for handicapped persons to control devices and more

An investigation on the differences between the eye states (open-closed), based in EEG input has already been addressed by several papers and researchers, the studies being nowadays more promising by exploiting the use of more and advanced classification algorithms as well as measuring performance on a collected corpus [6].

The following parts describe the overview of the architecture used in the system, analysing its components, the methods used to process and extract features from the EEG data, covering the comparison of 4 classification algorithms and the steps taken towards maximising the metrics for a satisfying accurate prediction of a driver's eye-closed state.

In our research we have avoided to deepen into the mathematical background of the algorithms used for the classification of an eye state and focused more on the impact their implementation has on generating a model with an optimal score on correctly alerting before a possible dangerous situation.

1.2 Technologies Used

The application installed on the drivers' mobile device was developed in Android Studio software development kit. Using the Message Queue Transport Telemetry (MQTT) protocol [7] and the Eclipse Paho project implementation [8], the device communicates with the Edge Server, set up with Java Oracle SE 8, which in turns communicates with the Backhaul Server through Web Socket Sessions.

The Backhaul Server is set up using Python 3.7, exploiting the SciPy library ecosystem [9] for data feature extraction and scikit-learn for data preprocessing and Machine Learning model training and classification.

All data are stored by the Backhaul Server in the database using the MySQL connector library.

1.3 EEG Signals

For the purposes of the project we have been provided with an EEG Eye State Data Set [10] by the Center for Machine Learning and Intelligent Systems [11] Repository developed by University of California, Irvine.

The data set was collected from one continuous EEG measurement with the Emotive EEG Neuroheadset [12], each feature set entry consisting of 14 EEG values and a value indicating the eye state.

The process of data acquisition involves electrodes being placed on marking points of the human head, labeled according to adjacent brain areas F(frontal), P(Parietal), O(Occipital), T(Temporal) and C(Central) [13].

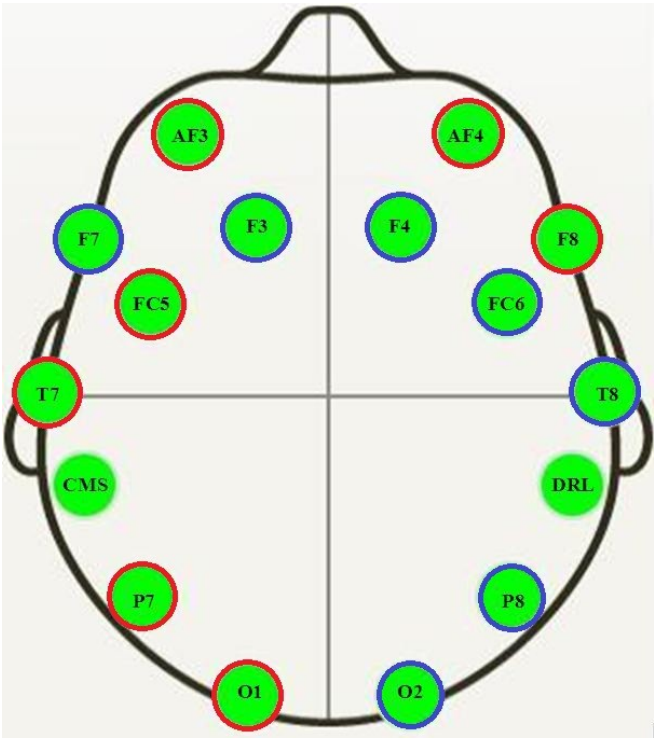

Image 1: Sensor positions in the human head

The device has two electrode arms, each containing 9 locations (7 sensors and 2 references). Two sensor locations (M1 / M2) already have rubber sensors fitted because they are the alternative positions for the default references (P3 / P4).

It provides good coverage of the frontal and prefrontal lobes and also provides coverage of the temporal, parietal and occipital lobes.

The specifications of the hardware used are the following;

Table 1: Emotiv Technical Specifications

Headset Version	EPOC v1.0
Number of Channels	14 (plus CMS/DRL references, P3/P4 locations)
Channel names (International 10-20 locations)	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4
Sampling Method	Sequential sampling. Single ADC
Sampling Rate	128 SPS (2048 Hz internal)
EEG Resolution	14 bits 1 LSB = 0.51μV (16 bit ADC, 2 bits instrumental noise floor is discarded)
Bandwidth	0.2 - 45Hz, digital notch filters at 50Hz and 60Hz
Filtering	Built-in digital 5th order Sinc filter
Dynamic Range (input referred)	8400 uV(pp)
Coupling Mode	AC coupled
Connectivity	Proprietary 2.4GHz wireless
Battery Capacity	LiPo battery 680mAh
Battery Life (typical)	12 hours
Impedance Measurement	Real-time contact quality using patented system
IMU Part	IDG500
Gyroscope	2-axis +/-8g
Motion Sampling	128 Hz
Motion Resolution	10 bit

2. ARCHITECTURE

2.1 Overview

The implemented system is based on three components to achieve a bidirectional transaction of data. The process involves the transmission of the EEG signals from the driver, their analysis and classification, and an alert to the subscribed devices if needed. The process information is then stored for historic view or other purposes.

2.2 Android Client

The Android Client application is responsible for collecting the EEG signals from the driver and forwarding them to the next component in real time. The client's role also includes informing of the vehicle's location position and coordinates, as well as reporting back to the drivers the response from the server. Any response indicating potential vehicle control loss caused by an eye-closed state situation, is presented with a visual and audio alert, to all subscribed devices in a close distance.

2.3 Edge Server

The Edge Server component is the one to receive all data sent by mobile devices in its area.Able to classify the eye state of each driver using the more accurate Machine Learning algorithm, it sends the classification outcome back to the terminals in its jurisdiction. All data are also forwarded to the next element of the architecture, the Backhaul Server.

2.4 Backhaul Server

If the Edge Server is the heart or the architecture, configuring data traffic and information exchange from and to the applications, the Backhaul Server is the brain of the system, determining how the classification of the signals will be made and having two main functionalities. By gathering data from several Edge Servers, the Backhaul Server can successfully train different Machine Learning models aiming to provide the Edge Servers with the most accurate algorithm for them to use in the classification process. At the same time, it operates as the brain's memory, storing all data for further and future usage.

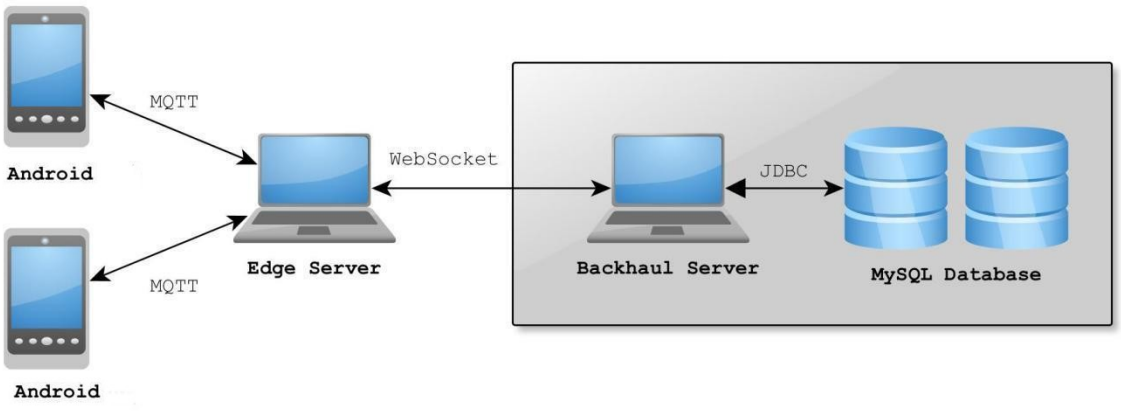

Image 2: Component Architecture

3. ANDROID CLIENT

3.1 Overview

The android application installed on the drivers' device is designed with a minimal interface, as its main purpose is to start the service [14] which is responsible for sending data to the Edge Server and alerting the drivers both visually and acoustically if a response from the Edge Server indicates so.

The main screen that the user can see when starting the *AndroidClient* is the following;

Image 3: Main screen of the android client

A service is an application component that can perform long-running operations in the background, running continuously even if the user switches to another application.

When starting the application, the user is prompted to give internet access, as well as access to accelerometer, location and GPS data. Starting the service from inside the application initiates the bidirectional communication between the Android Client service and the Edge Server through the MQTT protocol.

The client sends the device International Mobile Equipment Identity (IMEI) to the Edge Server and subscribes to the equivalent MQTT Topic created by the server, upon its confirmation of creation. As soon as the connection is established the client begins publishing its messages including the EEG signal data and the data provided by the accelerometer and the location sensors of the device. By using asynchronous communication methods, the service expects to receive a response from the Edge Server for a chunk of messages it publishes, indicating the eye state of the driver, therefore the drivers' state at the current time. In case of a potential eye-closed situation foreground services on the device are activated starting the device's alarm sound and opening a notification alert on the devices' display.

Understanding the difficulty to implement data acquisition in real life situation, the Android Client is provided with a csv file containing the EEG Signals data to be used as input in the application, thus effectively simulating a real time application, by sending 128 measurements per second, as the sampling freequency in the original dataset is 128Hz.

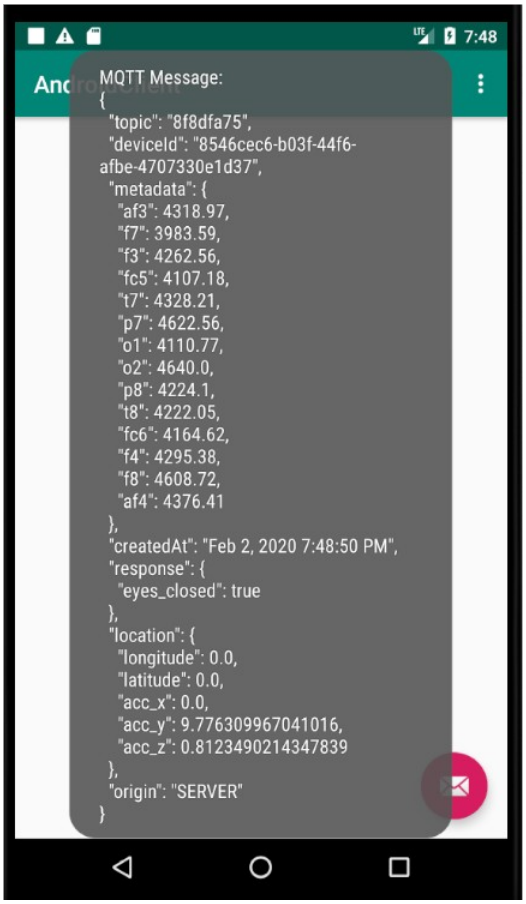

Image 4: message indicating an alert status of the application

3.2 The MQTT Protocol

MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers.

It is an official Organization for the Advancement of Structured Information Standards (OASIS) [15] standard having amongst others, ability for:

- **Better Error Reporting** – in particular, a reason code has been added to responses for publications (PUBACK/PUBREC). MQTT originated with use cases like sensors along an oil pipeline – if their publications fail to be transmitted then the sensor will take no action. However, the use cases for MQTT are now much broader and an app on a phone may well want to warn the user if data is not being transmitted successfully.

Return codes are now present on all acknowledgments (along with optional reason strings that contain human readable error diagnostics).

- **Shared Subscriptions** – If the message rate on a subscription is high, shared subscriptions can be used to load balance the messages across a number of receiving clients.
- **Message Properties** – Metadata in the header of a message. These are used to implement the other features in this list but also allow user defined properties e.g. to assist in message encryption by telling the receiver which key to use to decrypt the message contents
- **Message Expiry** – An option to discard messages if they cannot be delivered within a user-defined period of time.
- **Session Expiry** – If a client does not connect within a user defined period of time, state (e.g. subscriptions and buffered messages) can be discarded without needing to be cleaned up.
- **Topic Alias** – Allows topic strings in messages to be replaced with a single number, reducing the number of bytes that need to be transmitted if a publisher repeatedly uses the same topics.
- **Will Delay** – Allows a message to be published if a client is disconnected for more than a user defined period of time. Allowing notifications about outages of important client applications without being swamped by false positives.
- **Allowed Function Discovery** – At the start of a connection, limits like the maximum packet size and number of (QoS>0) messages inflight can be transmitted to inform the client what it is allowed to do.

4. EDGE SERVER

4.1 Overview

The Edge Server is the component that has a direct real-time bi-directional communication with the devices in its area through the MQTT protocol. Responsible to handle and organize the devices' data, it creates a different subscription topic for every device to publish, upon receiving the device's International Mobile Equipment Identity (IMEI).

After confirming the creation of the subscription topic with the mobile device the Edge Server waits for the EEG signals, the location and the accelerometer data responding asynchronously to each device and forwarding data to the Backhaul server every second.

Being the intermediate component of the system and communicating with the Backhaul Server through Web Sockets, the Edge Server has been provided with the most accurate Machine Learning model, which is used to classify each device's driver's eye state. Gathering 128 messages from each device every second and after classifying the predicted result, responds to the devices with the outcome and forwards both the data and its assertion to the Backhaul.

4.2 Design

The edge server component is implemented in Java, latest Oracle 8 stable version as a spring boot application, with an embedded jetty server and mongoDB as an intermediate persistence provider.

4.3 Implementation

The application, when deployed, starts listening on a default topic for messages. Clients carrying the EEG mobile application are by default on initialization sending a broadcast message in this topic with their unique identification code. The edge server, upon receiving this message, looks in the data source for such an ID and if not found, generates a unique hash for that identifier, which then broadcasts back to that topic. The edge server immediately generates a new topic named after that hash. This topic is intended to represent the unique channel of communication between the specified client and the edge server.

An example response from the EEG server with the topic information follows;

Table 2: Server response to a client broadcast

Response to a client broadcast	{
	"uuid": "b8cda31f-51af-4ccd-8122-9f0065154dd7",
	"private_topic": "999c6471",
	"created_at": "2020-02-02T17:23:40.000Z",
	"subscribed_at": "2020-02-02T17:23:41.492Z",
	"origin": "SERVER",
	"_class": "gr.uoa.di.eeg.edge.domain.EegTopic"
	}

We implemented this logic in order to keep every single client's communication clean and free from noise from other clients. The client receives this message and starts broadcasting on this topic the required metadata, such as all the data from the 14 EEG sensors, accelerometer data, GPS data, etc. with a frequency of 128 messages per second.

An example payload in JSON format is presented below;

Table 3: Client message transmitted to EEG server

Client request with a single row of metadata	{
	"topic": "999c6471",
	"device_id": "b8cda31f-51af-4ccd-8122-9f0065154dd7",
	"eyes_state": {
	"af3": 4321.02978515625,
	"f7": 4004.6201171875,
	"f3": 4284.10009765625,
	"fc5": 4153.330078125,
	"t7": 4345.64013671875,
	"p7": 4587.18017578125,
	"o1": 4093.330078125,
	"o2": 4616.919921875,
	"p8": 4202.56005859375,
	"t8": 4232.81982421875,
	"fc6": 4209.740234375,
	"f4": 4281.02978515625,
	"f8": 4628.2099609375,
	"af4": 4389.740234375
	},
	"created_at": "2020-02-02T17:23:41.562Z",
	"location": {
	"lon": 0,
	"lat": 0,
	"ax": 0,
	"ay": 9.776309967041016,
	"az": 0.8123490214347839
	},
	"origin": "CLIENT",
	"_class": "gr.uoa.di.eeg.edge.domain.EegMessage"
	}

The server receives and buffers these data to the limit of 128, and when this upper limit is reached, wraps those 128 messages as a single message and forwards them to the classifier, waiting for a response. Each message from the 128 is of course persisted in the intermediate datastore for historical reasons. Upon receiving the response from the classifier, prepares a message for this chunk of messages which then broadcasts back to the specific client, at the given topic.

This message carries the same information as before, with an additional entry named 'response', containing the classification result, as per the example below:

Table 4: Server response to client with classification metadata

Server response with classification result	{
	"topic": "999c6471",
	"device_id": "b8cda31f-51af-4ccd-8122-9f0065154dd7",
	"eyes_state": {
	"af3": 4287.18017578125,
	"f7": 3992.31005859375,
	...
	"f4": 4272.31005859375,
	"f8": 4605.1298828125,
	"af4": 4353.330078125
	},
	"created_at": "2020-02-02T17:23:42.402Z",
	"location": {
	"lon": 0,
	"lat": 0,
	"ax": 0,
	"ay": 9.776309967041016,
	"az": 0.8123490214347839
	},
	"origin": "SERVER",
	"_class": "gr.uoa.di.eeg.edge.domain.EegMessage",
	"response": {
	"eyesClosed": false
	}
	}

If the classified data respond to 'true', I.e. indicating an eyes closed state, the message is also asynchronously broadcasted to the backhaul server for being persisted there as well. Such a message is described below;

Table 5: Message forwarding with criticality information

Forwarding message to backhaul	{
	"device_id": "b8cda31f-51af-4ccd-8122-9f0065154dd7",
	"lon": 0,
	"lat": 0,
	"criticality_lvl": 1
	}

All operations are non-blocking, by utilizing the Completable Future [16] API of the standard Java edition, therefore there is no competence on resources for communication with the broker, or with the data source. That way, the application can easily handle several concurrent clients without blocking, therefore scale gracefully.

5. BACKHAUL SERVER

5.1 Overview

The Backhaul Server is the basic and most important component of the system, determining it functionality and effectiveness mainly by providing an accurate classification and prediction model to the Edge Server. In order to achieve such effectiveness, the Backhaul needs to be trained and tested on a large amount of labeled data, with the use of different Machine Learning algorithms.

The preparation of the Backhaul component consists of different stages, including data preprocessing, processing and analysis, self-training and testing with different techniques before reaching the best possible model on which the actual real-time classification will be used for.

Able to communicate with a number of Edge Servers, also has practical functionality having the responsibility to save all data records, making it the starting and the ending point of the system, the foundation on which the implementation relies on.

5.2 Design

The backhaul server is designed bearing in mind that it will supposedly serve many different edge servers for different cases. Such a case is the one scrutinized in this document. The server therefore is a standalone python project that is able to perform different machine learning experiments on such datasets.

5.3 Implementation

The backhaul server has the web socket client implemented, waiting for a single – at the moment – connection with the edge server in place. When such a connection is established, it accepts the log messages – the ones classified as ‘true’ and persists those in the data source, for historical and experimental reasons.

Apart from that, a module named ‘feature extraction’ is implemented containing the methods for the signal transformation techniques and the generation of the feature vectors that are fed to the machine learning algorithms for experiments.

Moreover, the ‘machine learning’ package is implemented containing several classes that implement the different ML algorithms chosen for the experiments, a grid_pipeline.py file that performs the experiments and outputs the results on a file, as well as an ensemble file that has some experiments using a voting classifier.

5.4 Signal Processing

The signal processing module contains the relevant files responsible for the Power spectral density (PSD) and the Short Time Fourier Transforms (STFT). Each module accepts the dataset, which contains 128 lines of data per second, and performs the required time-frequency transformations generating the delta (0, 4), theta (4, 8), alpha (8, 12), beta (12, 40) and gamma (40, 100) (Hz) bands of the respective data, for each one of the 14 sensors of the device. These bands are then flushed to a file containing 117 lines, a full spectrum of the full experiment which took place in about two minutes.

The band spectrum is described hereby:

Table 6: Brain Activity and Frequency

Band	Frequency (Hz)	Mental Activities
Delta	0 - 4	Deep sleep, lucid dreaming, increased immune functions, hypnosis
Theta	4 - 8	Deep relaxation, meditation, increased memory, focus, creativity, lucid dreaming, hypnagogic state
Alpha	8 - 12	Light relaxation, “super learning”, positive thinking
Beta	12 - 40	Fully awake, normal state of alertness, stress and anxiety
Gamma	40 - 100	Associated with information rich task processing and high level information processing

5.5 Power Spectral Density Transform

Power estimation and changes in EEG signals related to muscle fatigue over the motor cortex area for Adductor Pollicis muscle measured in relaxed and contraction states are classified using change in the power spectrum [17].

Power Spectral Density (PSD) describes how power of a signal or time series is distributed over frequency. Power can be the actual physical power or simply identified with the squared value of the signal. For example, statisticians study the variance of a function over time x(t) (or over another independent variable), and using an analogy with electrical signals (among other physical processes), it is customary to refer to it as the power spectrum even when there is no physical power involved. If one were to create a physical voltage source which followed x(t) and applied it to the terminals of a 1-ohm resistor, then indeed the instantaneous power dissipated in that resistor would be given by x(t)2 watts [18].

The analysis we have used from the PSD is the Average Band Power, which consists in computing a single number that summarizes the contribution of the given frequency band to the overall power of the signal [19]. In order to compute the average band power, we first need to compute an estimate of the power spectral density. The most widely-used method to do that is the Welch's periodogram, which consists in averaging consecutive Fourier transform of small windows of the signal, with or without overlapping. The method is based on the concept of using periodogram spectrum estimates, which are the result of converting a signal from the time domain to the frequency domain [20].

From the periodogram, the average band power is computed by estimating the area between the frequency range of each band, using the composite Simpson’s rule [21].

Below are presented some plots in regards to the Welch’s periodogram;

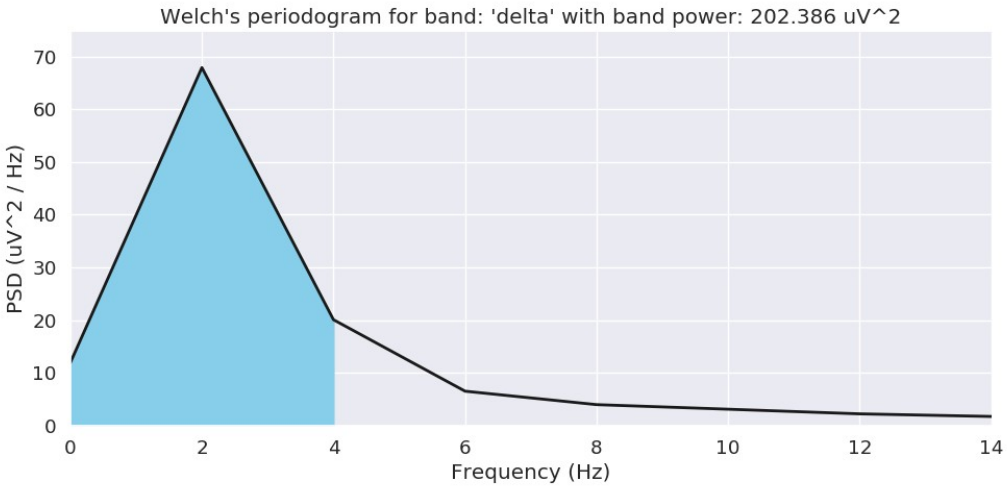

Figure 1: Welch's periodogram for band: delta

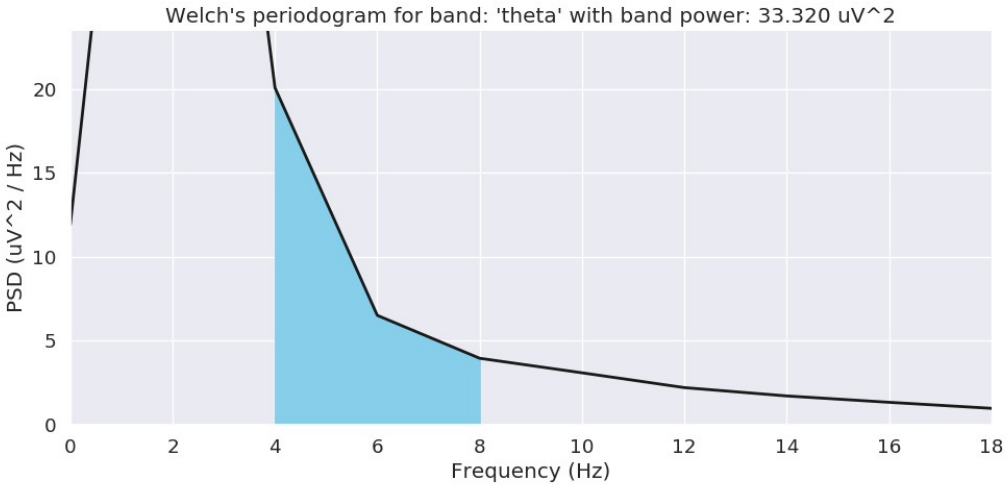

Figure 2: Welch's periodogram for band: theta

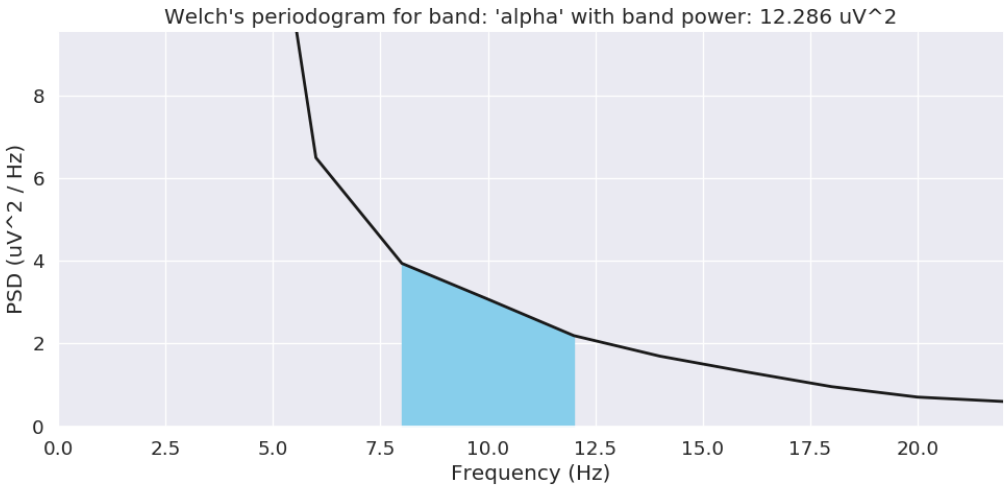

Figure 3: Welch's periodogram for band: alpha

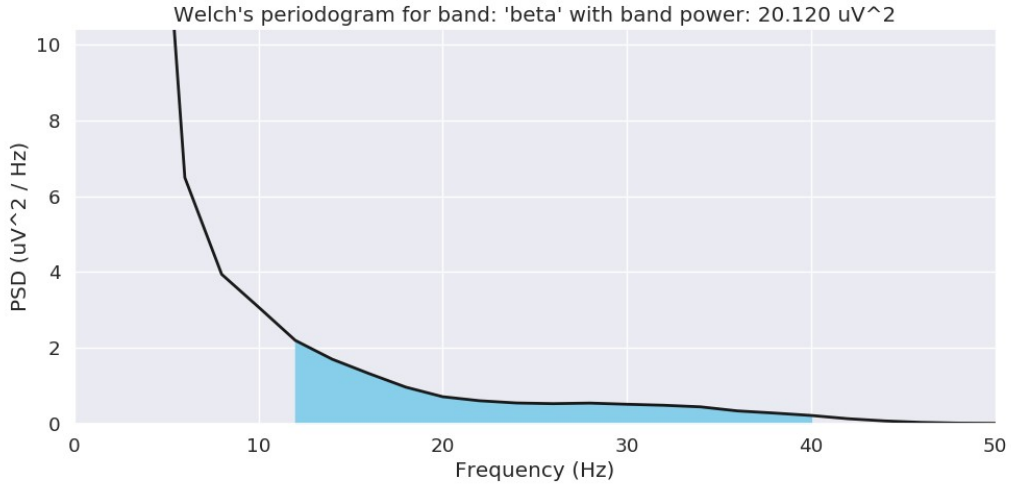

Figure 4: Welch's periodogram for band: beta

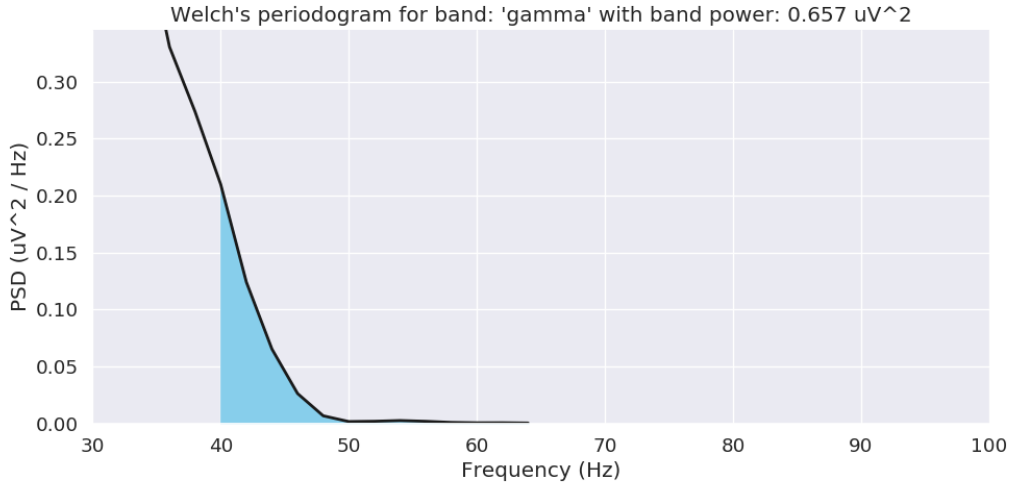

Figure 5: Welch's periodogram for band: gamma

5.6 Short Time Fourier Transform

The second method used for decomposing the EEG signals into frequency components is the Short Time Fourier Transformation (STFT) [22]. STFT divides an input signal into N sections according to a specified sliding window, and performs a Fast Fourier Transformation (FFT) on each section, revealing the Fourier spectrum of each segment, thus giving the ability to produce a spectrogram, a function of frequency and time. The window used for the STFT transformation is the Hamming window [23], considered to be a moderate window, balancing between resolution and dynamic range.

For each band's frequency range and for every second, we take the absolute of the corresponding STFT output, and the standard deviation of the STFT is used to produce the feature vector file to be used for the model generation.

After applying the two methods, we come up with 117 lines of data, each line referring to a second, consisting of 14 arrays corresponding to the EEG sensors, each array including data for all of the 5 bands.

Finally, to extract the feature vector, we flatten the nested band arrays to produce lines with 70 elements, concluding to a final array of 117x70 for the feature vector file.

6. MODEL GENERATION

6.1 Overview

For the purposes of model generation, we have tried four of the most common classifiers, namely Naïve Bayes, K Nearest Neighbors (KNN), Random Forest (RF) and Support Vector Machine (SVM).

The classifiers are fed to a Scikit-learn model selector GridSearchCV, together with a range of values for each classifier's parameters, in order to thoroughly test which classifier produces the highest prediction metrics and the parameters used for it.

The process of feature extraction and model generation is repeated over different windows of the signal processing, data-scaling methods and tolerance values for dimensionality reduction.

6.2 Approach used

We selected a large range of windows for each one of the two transformation methods applied, and for each window we run a grid search for all four classifiers with several parameter ranges. When one iteration was done, the classifier performing with the best accuracy was selected and saved to a file for further reference.

For each iteration and classifier, the module constructs a new pipeline which performs the following steps

- Load initial dataset
- Remove outliers
- Perform the transformation according to the method (STFT/PSD) and the window size passed
- Feed the transformed dataset to the grid search;

The grid search is configured each time to perform 10-fold validation on the test set, which is the 80% of the corpus. This approach is used across all different executions.

On section *'Experiments'* we further analyze the steps after this initial approach, which were to also scale the feature set, perform principal component analysis and re execute the tests with a wider parameter range for the most performing classifiers.

6.3 Binary Classification techniques

6.3.1 Data Cleaning

A quick and dirty implementation of outlier detection and removal involves calculating the mean and the standard deviation values of the columns of the dataset, that is the mean and standard deviation values for each of the 14 EEG sensors. We have defined an upper and lower bound of 4 times the standard deviation and removed all lines corresponding to these data from the dataset. The remaining dataset with 14304 lines of data was then passed to the algorithms for feature extraction and classification.

6.3.2 Data Preprocessing

Data preprocessing tasks include scaling the dataset to the {-1, 1} range in order to feed the classifiers, along with Principal Component Analysis (PCA) [24]

- **Feature Scaling**

We conducted experiments with two scalers, StandardScaler and RobustScaler. Scaling of the dataset is important for most of the classifiers used, because, for example, SVM's kernel methods are based on distance, so without scaling all features to comparable ranges, the features with the largest range would completely dominate during computation of the kernel matrix.

- **PCA**

Due to the nature of our experiments, the dataset, although small in size, has a long set of features, namely 70. For that reason, we decided to try incorporating dimensionality reduction using Singular Value Decomposition. As it will be shown in the experiments section, we managed to describe 99.9% of the dataset using only 59 of the 70 dimensions, thus gaining a lot in effort estimation times. We applied the scikit-learn implementation of the principal component analysis algorithm [25]

6.4 Binary classification using selected classifiers

Hereby we present each one of the four machine learning algorithms we used for our experiments.

- **Naïve Bayes**

The simplest of the classifiers used, Naïve Bayes is a probabilistic classifier, based on applying Bayes' theorem, assuming independence between the features of the vector describing the classified element [26].

Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event [27]. Bayes' theorem is stated mathematically using the following equation

$$P(A|B) = P(B|A) * \frac{P(A)}{P(B)}$$

where $P(A|B)$ is the likelihood of event A occurring given that B is true, $P(B|A)$ is the likelihood of event B occurring given that A is true, $P(A)$ and $P(B)$ are the probabilities of observing events A and B.

The different implementations of the Naïve Bayes classifiers use a series of models regarding the distribution of $P(B|A)$, such as the Bernoulli distribution, Categorical, Complement, Multinomial, etc. In our implementation we have used the GaussianNB implementation of the scikit-learn library [28].

- **K-Nearest Neighbors**

Another simple ML algorithm, K-Nearest Neighbors (KNN) classifies a new data point by calculating its distance from the training points and using the K nearest ones to vote for its label classification [29]. The simple version of KNN assumes that all the features of the vector describing a data point are of equal importance, whereas the weighted version can assign weights to dimensions reflecting those which are more impacting on the classification result.

The distance of a given point to one in the training set can be calculated using different metrics, such as Euclidian, Minkowski Manhattan and Chebysev. In our attempts to identify the most accurate model for KNN classifier we have tried all of these metrics, with a range of 3 to 15 neighbors, using both uniform weights or weights defined by the inverse of their distance.

- **Random Forest**

A little more complex, Random Forest (RF) classifier [30] is based on Decision Trees [31] which in general classify a point depending on a single condition. From the original dataset, a number of subsets with the same size as the original is randomly drawn, with the possibility of every point being drawn more than once. Each one is trained independently, the final classifier being the average of those.

From each subset created, a random subset of the dimensions is selected for the split of the decision tree, repeating with another variable at each step until no more splits are possible.

When a new point is entered for classification, it is classified by each of the decision trees created, and the average of the results becomes the final outcome of the algorithm. The acts of bootstrapping the dataset and aggregating to make a decision are referred as 'Bagging'.

Allowing duplicate entries in the selected subsets, results in having a number of entries not used by the algorithm, giving them the ability to act as new data for testing and making the evaluation of the classification more accurate.

A typical size of the variables selected for the splitting of decision trees is the square root of the original variables, and a log of the original size was tried also. Other parameters used in the quest for an optimal model include the number of trees to consider, within a range of 5 to 1000, and the minimum number of samples required to continue slitting internal nodes, from 3 to 10.

We incorporated the scikit-learn implementation of the RF classifier [32] which gives the ability to parameterize several dimensions, some of which we considered in our experiments, but not including, are the following:

• n_estimators: The number of trees in the forest. This dimension is irrelevant to the dataset dimension, and can have any value possible, provided the hardware can complete the calculations in a reasonable time.

• criterion: The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

• max_depth: The maximum depth of each decision tree. If not specified, then the nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples

• min_samples_split: The minimum number of samples required to split an internal node

• max_features: The number of features to consider when looking for the best split

• class_weight: Weights associated with classes.

- **Support Vector Machines**

The Support Vector Machines (SVM) classifier [33] focuses on finding a decision boundary that maximizes the distance between the closest dataset points that have different classification. In the case that a decision boundary is not possible to find, SVM

takes advantage of kernel functions to project the dataset points to a higher dimension, making the separation of the classes clearer and transforming a new point to the new higher dimension before comparing it to the decision boundary and classify it. The points in the training dataset that are on the gutter of the decision boundary area are called the 'support vectors' of the set.

The SVM classifier was tried with a polynomial kernel up to a degree of 5, as well as the radial basis function (RBF) kernel, based on the exponential of the squared difference of two samples, applying different parameters to both kernel functions and appeared to result to the optimal model of all classifiers tested.

The parameters that can be tuned for this classification method are many, and we decided to tackle with the following

• C: The penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly.

• degree: Used when kernel is set to 'poly'. It's the degree of the polynomial used to find the hyper-plane to split the data.

• coef0: Independent term in the kernel function

• gamma: Useful only for non linear hyper-planes. The higher the gamma value it tries to exactly fit the training data set

• shrinking: shrinking heuristics for speeding up the optimization

6.5 Metrics

To evaluate the performance of the investigated ML algorithms on the provided dataset and acquire the best possible model to be applied to the system, we have used the following metrics from sklearn library:

- **Accuracy:** The ratio of the predicted labels that match the true corresponding labels of the dataset.

- **Precision:** The ratio $\frac{TP}{TP+FP}$, where TP is the number of true positives and FP the number of false positives. This is intuitively the ability of the classifier not to label as positive a sample that is negative

- **Recall:** The ratio $\frac{TP}{TP+FN}$ where TP is the number of true positives and FN the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

- **F-Measure:** Also known as F_1 score or balanced F-score, the F-Measure can be interpreted as a weighted average of the precision and recall, where F-Measure reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F-Measure are equal. The formula for F-Measure is

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

6.6 Experiments

We run the experiments using a predefined range of windows, performing various iterations while trying to improve on the classification outcome.

We compared the classification results based on four metrics that tend to describe the characteristics of a model; Accuracy, prediction, recall, and F-measure. We were mainly monitoring results with higher accuracy and then F_1 score.

For all experiments, we perform the following:

- Remove outliers from the original dataset
- Transform the result using a frequency range of 128 (128 events per second) with the method passed (STFT/PSD) and the window length specified
- Run the Grid Search with a specified set of classification parameters, applying cross-validation techniques with a percentage of 80-20 to the dataset.

6.6.1 Run with standard/robust scaling

Initial run was with all the 'default' steps and with standard scaling of the dataset, for the window range {24, 108} with step 2, and with the following parameter grid per classifier selected;

Table 7: Parameter Set of Experiment with standard/robust scaling

Classifier	Parameter Set
Naive Bayes	{}
K-Nearest Neighbors	'n_estimators': [5, 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features': [6, 'auto', 8]
Random Forest	'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski'], 'weights': ['uniform', 'distance'], 'n_neighbors': [3, 5, 7, 9, 11, 13, 15], 'p': [2, 3, 4, 5, 6]
Support Vector Machines	'kernel': ('linear', 'rbf', 'poly'), 'C': [0.1, 0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'degree': [1, 2, 3]

The experiment provided the following results

Table 8: Top 5 results with standard scaling

method	win_len	classifier	accuracy	precision	recall	f1	params
psd	32	SVM	0,735	0,713	0,811	0,716	{'C': 5, 'degree': 1, 'kernel': 'poly'}
psd	36	SVM	0,735	0,774	0,683	0,697	{'C': 2, 'degree': 1, 'kernel': 'rbf'}
psd	24	SVM	0,726	0,786	0,72	0,692	{'C': 0.5, 'degree': 1, 'kernel': 'linear'}
psd	40	SVM	0,718	0,753	0,662	0,679	{'C': 7, 'degree': 1, 'kernel': 'rbf'}
psd	48	SVM	0,709	0,807	0,625	0,645	{'C': 2, 'degree': 1, 'kernel': 'poly'}

T

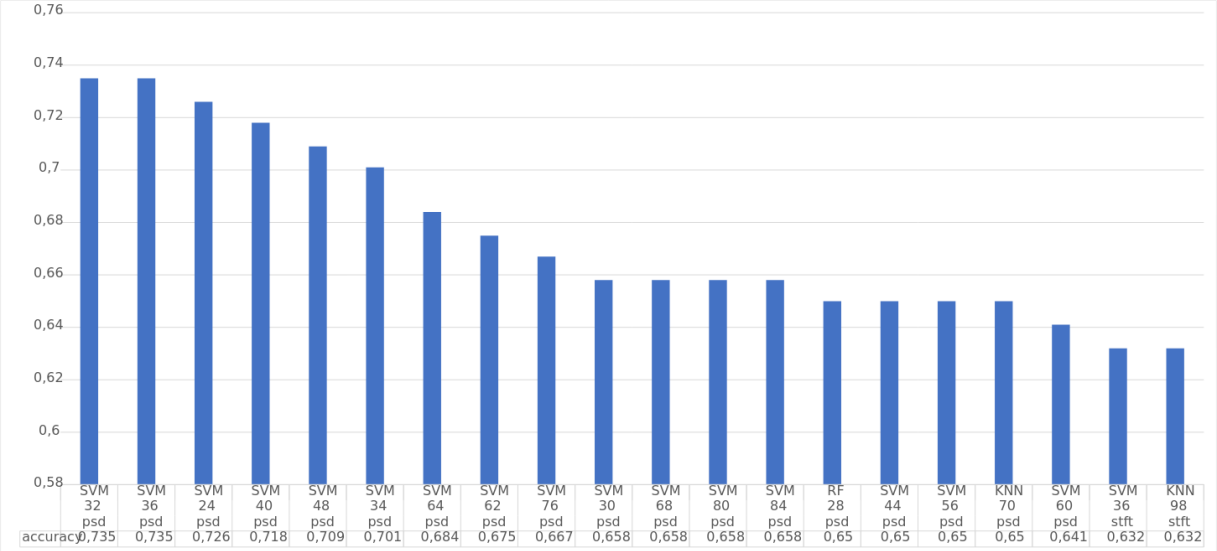

Figure 6: Top 20 results with standard scaling

The same run with the only difference of using the robust scaler, produced the following results;

Table 9: Top 5 results with robust scaling

method	win_len	classifier	accuracy	precision	recall	f1	params
PSD	24	SVM	0,726	0,754	0,72	0,698	{'C': 1, 'degree': 1, 'kernel': 'linear'}
PSD	32	SVM	0,718	0,7	0,72	0,698	{'C': 3, 'degree': 1, 'kernel': 'poly'}
PSD	36	SVM	0,718	0,747	0,7	0,679	{'C': 3, 'degree': 1, 'kernel': 'rbf'}
PSD	34	SVM	0,709	0,709	0,68	0,68	{'C': 2, 'degree': 1, 'kernel': 'rbf'}
PSD	40	SVM	0,709	0,758	0,66	0,665	{'C': 2, 'degree': 1, 'kernel': 'rbf'}

We can already observe three patterns;

- PSD transformation outperforms STFT
- Better results arrive from the smaller window sizes
- SVM seems to perform better than the others.

A graph indicating a wider range of results is the following;

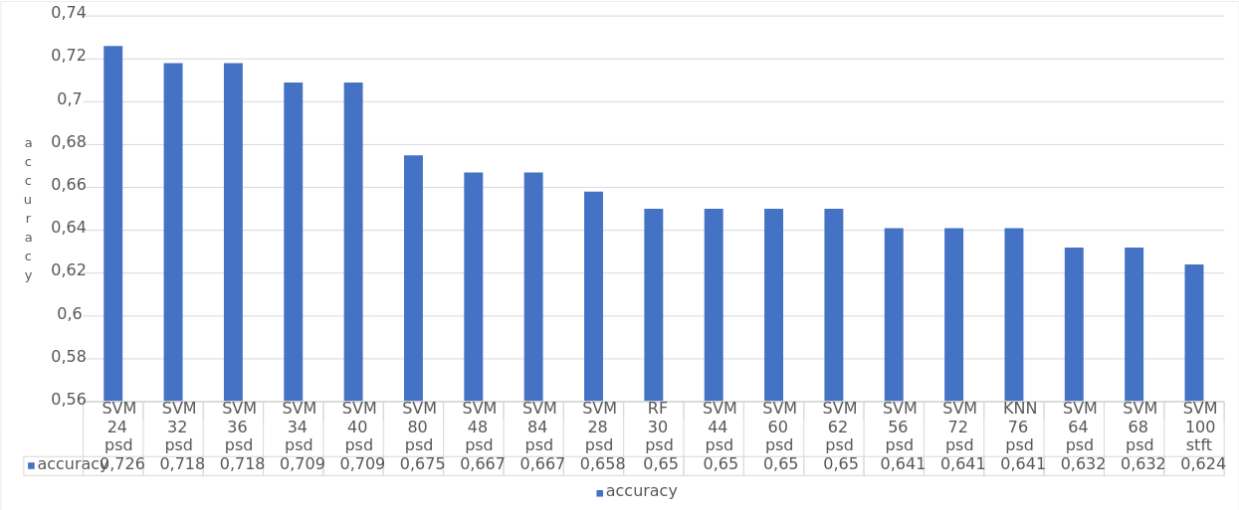

Figure 7: Top 20 results with robust scaling

6.6.2 Run with standard/robust scaling and Principal Component Analysis

After the first run, we concluded that the feature set was probably too large to be understood by a classification model in adequate time. Therefore, we tried to reduce the dimensions of the feature set, by applying principal component analysis (PCA). By performing some tests on the transformed dataset, we kept 3 PCA vectors, one that describes the supplied dataset at 99.9% (thus losing only 0.1% of information) and keeping only 59 of the 70 dimensions, one that describes it at the 99% level, keeping only 45 dimensions and one that describes it on the 90% (discarding 10% of information) and keeping 30 dimensions.

The parameter set of the current experiment is the following

Table 10: Parameter Set of Experiment with standard/robust scaling and PCA

Classifier	Parameter Set
Naive Bayes	{}
K-Nearest Neighbors	'n_estimators': [5, 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features': [6, 'auto', 8]
Random Forest	'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski'], 'weights': ['uniform', 'distance'], 'n_neighbors': [3, 5, 7, 9, 11, 13, 15], 'p': [2, 3, 4, 5, 6]
Support Vector Machines	'kernel': ('linear', 'rbf', 'poly'), 'C': [0.1, 0.25, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'degree': [1, 2, 3]

We present the top 10 results on the following table.

Table 11: Top 10 results with PCA

method/ win_lenu	scaler	tolerance	classifier	accuracy	precision	recall	f1	best_params
PSD 32	standard	0.999	SVM	0.744	0.726	0.811	0.721	{'C': 4, 'degree': 1, 'kernel': 'poly'}
PSD 40	robust	0.99	RF	0.735	0.806	0.664	0.669	{'max_features': 'auto', 'n_estimators': 700}
PSD 36	standard	0.999	RF	0.726	0.79	0.606	0.664	{'max_features': 6, 'n_estimators': 300}
PSD 36	standard	0.99	RF	0.726	0.756	0.643	0.683	{'max_features': 8, 'n_estimators': 500}
PSD 48	standard	0.99	SVM	0.726	0.805	0.625	0.645	{'C': 1, 'degree': 1, 'kernel': 'poly'}
PSD 32	robust	0.99	RF	0.726	0.757	0.606	0.659	{'max_features': 6, 'n_estimators': 900}
PSD 32	standard	0.99	SVM	0.718	0.71	0.792	0.694	{'C': 3, 'degree': 1, 'kernel': 'poly'}
PSD 32	robust	0.999	RF	0.718	0.79	0.585	0.634	{'max_features': 6, 'n_estimators': 200}
PSD 36	robust	0.999	SVM	0.718	0.766	0.831	0.688	{'C': 2, 'degree': 1, 'kernel': 'rbf'}
PSD 36	robust	0.99	SVM	0.718	0.786	0.831	0.693	{'C': 2, 'degree': 1, 'kernel': 'rbf'}

We can already observe that by emitting an insignificant percentage of the withheld information, we can already observe similar, albeit even better results, also with minimized execution time.

6.6.3 Exhaustive Run for selected classifiers

From the above experiments we made the following observations:

- Smaller windows perform better
- PSD outperforms STFT
- SVM and RF outperform the other two classifiers
- PCA with tolerance at 99.9% increases the execution time without altering the behavior of the classifiers

Therefore, we prepared and executed an exhaustive run with only 2 of the 4 classifiers in hand, with a smaller range of selected windows for the frequency transformation, namely {32, 48} with a step of 2, with a much wider range of parameter selection for the tuning of the models, and with only the PSD transformation method.

The parameter set for the exhaustive experiment is presented;

Table 12: Exhaustive run with RF/SVM

Classifier	Parameter Set
Random Forest	'n_estimators': [5, 10, 50, 100, 300, 500, 600, 700, 800, 900, 1000], 'criterion':["gini" ,"entropy"], 'max_depth': [3, 4, 5, 8, 10], 'max_features': ['sqrt', 'auto', 'log2'], 'min_samples_split': [2, 3, 5, 10], 'class_weight': [None, 'balanced', 'balanced_subsample']
Support Vector Machines	'kernel': ('rbf', 'poly'), 'C': [0.1, 0.25, 0.5, 1, 5, 10, 20, 30, 4, 50, 60], 'degree': [2, 3, 4, 5], 'gamma': [100, 10, 1, 0.1, 0.005, 0.0025, 0.002, 0.001, 0.0001], 'coef0': (0.1, 0.2, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 3, 4, 5, 6, 7, 8, 9, 10), 'shrinking': (True, False)

Results are presented in the below table;

Table 13: Top 8 results with PSD and parameter tuning

win_len scaler	classifier	accuracy	precision	recall	f_1	best_params
32 robust	**SVM**	**0.795**	**0.755**	**0.868**	**0.786**	**{'C': 5, 'coef0': 5, 'degree': 2, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}**
36 standard	SVM	0.778	0.777	0.78	0.752	{'C':60 , 'coef0': 0.75, 'degree': 2, 'gamma': 0.002, 'kernel': 'rbf', 'shrinking': True}
40 robust	SVM	0.769	0.792	0.885	0.751	{'C': 2, 'coef0': 1.25, 'degree': 4, 'gamma': 0.005, 'kernel': 'poly', 'shrinking': True}
36 standard	RF	0.769	0.865	0.72	0.724	{'class_weight': None, 'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto', 'min_samples_split': 3, 'n_estimators': 100}
36 standard	SVM	0.769	0.756	0.89	0.749	{'C': 30, 'coef0': 1.25, 'degree': 4, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36 standard	SVM	0.769	0.756	0.89	0.749	{'C': 30, 'coef0': 1.25, 'degree': 4, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36 standard	SVM	0.769	0.754	0.76	0.74	{'C': 50, 'coef0': 1.5, 'degree': 3, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36 robust	SVM	0.769	0.766	0.78	0.752	{'C': 40, 'coef0': 0.75, 'degree': 2, 'gamma': 0.002, 'kernel': 'rbf', 'shrinking': True}
36 standard	SVM	0.761	0.772	0.812	0.747	{'C': 0.25, 'coef0': 5, 'degree': 3, 'gamma': 0.005, 'kernel': 'poly', 'shrinking': True}

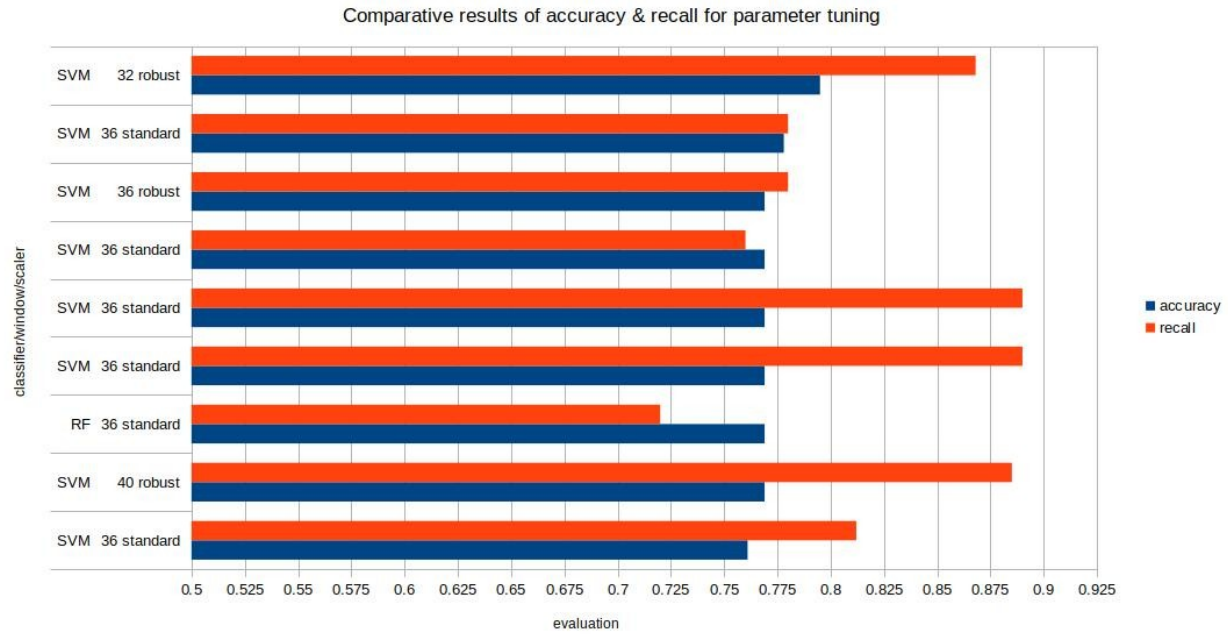

Figure 8: Top results with PSD and parameter tuning

6.6.4 Run with Voting Classifier

After that, we tried an execution of the top 5 results with a voting classifier, with both hard and soft voting methods. We observed no better results.

Table 14: Top 5 results with Voting Classifier

win_len	scaler	Tol.	classifier	accuracy	precision	recall	f_1	best_params
36	standard	0.999	SVM	0.778	0.777	0.776	0.752	{'C': 60, 'coef0': 0.75, 'degree': 2, 'gamma': 0.002, 'kernel': 'rbf', 'shrinking': True}
36	standard	0.999	SVM	0.769	0.756	0.887	0.749	{'C': 30, 'coef0': 1.25, 'degree': 4, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36	standard	0.999	SVM	0.769	0.754	0.758	0.74	{'C': 50, 'coef0': 1.5, 'degree': 3, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36	standard	0.999	SVM	0.744	0.756	0.739	0.713	{'C': 30, 'coef0': 1.5, 'degree': 3, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
36	standard	0.999	RF	0.735	0.795	0.624	0.672	{'max_features': 'auto', 'n_estimators': 600}

6.6.5 Results

From the above experiments, we concluded that the best accuracy/F_1-score we could reach with this dataset was with the following settings;

Table 15: Best performing classifier

Transformation method	PSD
Window Length	32
Scaling method	Robust
PCA tolerance	99.9%
Classifier	Support Vector Machines
Parameter tuning	{'C': 5, 'coef0': 5, 'degree': 2, 'gamma': 0.001, 'kernel': 'poly', 'shrinking': True}
Accuracy achieved	0.795

6.6.6 Accompanying CD/DVD / repository link

Android Client: https://bitbucket.org/gkiomourtzis/mqtt-android-client/src/master/

Edge Server: https://bitbucket.org/gkiomourtzis/edge-server/src/master/

Backhaul Server: https://bitbucket.org/gkiomourtzis/backhaul-server/src/master/

7. CONCLUSIONS

In the current document we tried to implement a PoC simulation of a real-time collision avoidance system using machine learning for predicting the eye state of a given substitute based on time-based EEG recordings. After iterating over experiments, we reached in a state of approximately 80% accuracy in the prediction, using the Support Vector Machines classifier. Thus, we can conclude that the methods presented here are promising and a real time system like the proof of concept we tried to present here may have genuine value.

Having a more objective approach in our metrics evaluation we have mainly focused our attention to accuracy and f-measure scores. Taking into account, however, that the goal is to prevent and alert for a potential dangerous situation, considering the recall along with the others would be a more convenient approach. As mentioned in the metrics content, recall score favours the correct prediction of positive samples, which in our preventive and alertive objective has a major importance. In simple words we prefer to give a false alert for a non-dangerous eye-state condition rather than missing a true eye-state closed condition and a recall evaluation score seems to be more appropriate in that case. However, the model we have concluded on has already an 87% score in this metric and is of the highest amongst all, although not the highest.

In addition, accuracy in model generation for eye state classification could be enhanced by a dataset greater in variety and quantity of data, taking into consideration that all EEG recordings were extracted from only one individual and for a short amount of time. Acquiring sample data for testing for a range of a person's characteristics such as age or fatigue, as well as observing the eye-state of the subject in a natural condition rather than instructing the person to have his eyes open or closed on demand, would also contribute to making the labelling of the dataset more realistic and as close to real conditions as possible.

Future research in regards of the current project could be implemented with the inclusion of deep learning (DL) techniques such as convolutional neural networks (CNNs), which take advantage of perceptron [34], another ML algorithm, by expanding it to a multilayer level, and is considered to be a more sophisticated model simulating the biological human brain activity. Although our initial intention was to include such techniques in this paper, we believe that a specific research in this area would exploit the most of DL's capabilities and offer more specialized and detailed results.

ABBREVIATIONS - ACRONYMS

EEG	Electroencephalography
ML	Machine Learning
NCBI	National Center for Biotechnology Information of the United States of America
MQTT	Message Queue Transport Telemetry
IMEI	International Mobile Equipment Identity
NB	Naive Bayes
KNN	K Nearest Neighbors
RF	Random Forest
SVM	Support Vector Machines
DL	Deep Learning
CNN	Convolutional Neural Network
PSD	Power Spectral Density
STFT	Short Time Fourier Transform
PCA	Principal Component Analysis
M2M	Machine to Machine
POC	Proof of Concept
GPS	Global Positioning System
JSON	Javascript Simple Object Notation
API	Application Programming Interface

REFERENCES

[1] https://ec.europa.eu/eurostat/statistics-explained/index.php/Road_accident_fatalities_-_statistics_by_type_of_vehicle

[2] https://ec.europa.eu/commission/presscorner/detail/en/MEMO_19_1990

[3] HORNE, Jim; REYNER, Louise. Vehicle accidents related to sleep: a review. Occupational and environmental medicine, 1999, 56.5: 289-294.

[4] SAGBERG, Fridulv. Road accidents caused by drivers falling asleep. Accident Analysis & Prevention, 1999, 31.6: 639-649.

[5] TATUM IV, William O. (ed.). Handbook of EEG interpretation. Demos Medical Publishing, 2014.

[6] RÖSLER, Oliver; SUENDERMANN, David. A first step towards eye state prediction using eeg. Proc. of the AIHLS, 2013.

[7] http://mqtt.org/

[8] https://www.eclipse.org/paho/

[9] https://www.scipy.org/

[10] https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State#

[11] https://cml.ics.uci.edu/

[12] https://www.emotiv.com/

[13] EEG Interpretation through Short Time Fourier Transform for Sensory Response Among Children S.S. Hussin and R. Sudirman, Australian Journal of Basic and Applied Sciences 2014

[14] https://developer.android.com/guide/components/services

[15] https://www.oasis-open.org/

[16] https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html

[17] A. A. Abdul-latif, I. Cosic, D. K. Kumar, B. Polus and C. Da Costa, "Power changes of EEG signals associated with muscle fatigue: the root mean square analysis of EEG bands," Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004., Melbourne, Vic., Australia, 2004, pp. 531-534.

doi:10.1109/ISSNIP.2004.1417517

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1417517&isnumber=30674

[18] YOUNGWORTH, Richard N.; GALLAGHER, Benjamin B.; STAMPER, Brian L. An overview of power spectral density (PSD) calculations. In: Optical Manufacturing and Testing VI. International Society for Optics and Photonics, 2005. p. 58690U.

[19] https://raphaelvallat.com/bandpower.html

[20] https://en.wikipedia.org/wiki/Welch%27s_method

[21] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simps.html

[22] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.stft.html

[23] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.hamming.html#scipy.signal.windows.hann

[24] WOLD, Svante; ESBENSEN, Kim; GELADI, Paul. Principal component analysis. Chemometrics and intelligent laboratory systems, 1987, 2.1-3: 37-52.

[25] https://scikit-learn.org/stable/modules/decomposition.html#pca

[26] RISH, Irina, et al. An empirical study of the naive Bayes classifier. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. 2001. p. 41-46.

[27] Joyce, James (2003), Zalta, Edward N. (ed.), "Bayes' Theorem", The Stanford Encyclopedia of Philosophy (Spring 2019 ed.), Metaphysics Research Lab, Stanford University, retrieved 2020-01-17

[28] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

[29] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

[30] LIAW, Andy, et al. Classification and regression by randomForest. R news, 2002, 2.3: 18-22.

[31] PRIYAM, Anuja, et al. Comparative analysis of decision tree classification algorithms. International Journal of current engineering and technology, 2013, 3.2: 334-337.

[32] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[33] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[34] http://web.csulb.edu/~cwallis/artificialn/History.htm