2019

LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data

Alexander Verner
Nova Southeastern University, verner.alexander@gmail.com

NSUWorks Citation

Alexander Verner. 2019. *LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data*. Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (1074) https://nsuworks.nova.edu/gscis_etd/1074.

LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data

by

Alexander Verner

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Engineering and Computing
Nova Southeastern University

2019

We hereby certify that this dissertation, submitted by Alexander Verner, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

[illegible]
Sumitra Mukherjee, Ph.D.
Chairperson of Dissertation Committee

April 25, 2019
Date

[illegible]
Michael J. Laszlo, Ph.D.
Dissertation Committee Member

April 25, 2019
Date

[illegible]
Francisco J. Mitropoulos, Ph.D.
Dissertation Committee Member

April 25, 2019
Date

Approved:

Meline Kevorkian
Meline Kevorkian, Ed.D.
Interim Dean, College of Engineering and Computing

April 25, 2019
Date

College of Engineering and Computing
Nova Southeastern University

2019

An Abstract of a Dissertation Submitted to Nova Southeastern University

in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

LSTM Networks for Detection and Classification of Anomalies in Raw Sensor Data

by
Alexander Verner
March 2019

In order to ensure the validity of sensor data, it must be thoroughly analyzed for various types of anomalies. Traditional machine learning methods of anomaly detections in sensor data are based on domain-specific feature engineering. A typical approach is to use domain knowledge to analyze sensor data and manually create statistics-based features, which are then used to train the machine learning models to detect and classify the anomalies. Although this methodology is used in practice, it has a significant drawback due to the fact that feature extraction is usually labor intensive and requires considerable effort from domain experts.

An alternative approach is to use deep learning algorithms. Research has shown that modern deep neural networks are very effective in automated extraction of abstract features from raw data in classification tasks. Long short-term memory networks, or LSTMs in short, are a special kind of recurrent neural networks that are capable of learning long-term dependencies. These networks have proved to be especially effective in the classification of raw time-series data in various domains. This dissertation systematically investigates the effectiveness of the LSTM model for anomaly detection and classification in raw time-series sensor data.

As a proof of concept, this work used time-series data of sensors that measure blood glucose levels. A large number of time-series sequences was created based on a genuine medical diabetes dataset. Anomalous series were constructed by six methods that interspersed patterns of common anomaly types in the data. An LSTM network model was trained with k-fold cross-validation on both anomalous and valid series to classify raw time-series sequences into one of seven classes: non-anomalous, and classes corresponding to each of the six anomaly types.

As a control, the accuracy of detection and classification of the LSTM was compared to that of four traditional machine learning classifiers: support vector machines, Random Forests, naive Bayes, and shallow neural networks. The performance of all the classifiers was evaluated based on nine metrics: precision, recall, and the F1-score, each measured in micro, macro and weighted perspective.

While the traditional models were trained on vectors of features, derived from the raw data, that were based on knowledge of common sources of anomaly, the LSTM was trained on raw time-series data. Experimental results indicate that the performance of the LSTM was comparable to the best traditional classifiers by achieving 99% accuracy in all 9 metrics. The model requires no labor-intensive feature engineering, and the fine-tuning of its architecture and hyper-parameters can be made in a fully automated way. This study, therefore, finds LSTM networks an effective solution to anomaly detection and classification in sensor data.

Acknowledgments

Several people have guided, supported and encouraged me throughout my journey to pursue a doctoral dissertation from this esteemed institution.

First and foremost, I would like to express my appreciation to Dr. Sumitra Mukherjee, my academic advisor, for his guidance, strategic advice and invaluable comments provided during this challenging endeavor.

I would also like to thank my committee members: Dr. Francisco J. Mitropoulos and Dr. Michael J. Laszlo for encouraging my study and its positive evaluation.

I am most sincerely grateful to my parents, Igor and Marina, and to my brother Uri, for the time they invested in making my work better.

I thank my grandparents, Matvey and Eleonora, for believing in me and supporting my aspirations.

I am grateful to my wife Nadezhda for taking care of all my needs, and my kids Artiom and Polina, for being my source of inspiration along the way.

Table of Contents

List of Tables

Tables

List of Figures

Chapter 1 - Introduction

Sensor validation relies on understanding of several basic concepts, such as the dependence of a sensor-based system on the validity of the data, the relation between various sensor faults and anomalies in the data, the importance of timely detection of the data, etc. This role of this chapter is to familiarize the reader with the field of detection and classification of anomaly in sensor data and explain the essence of the proposed dissertation. The introduction is divided into the following sections:

- Background
- Problem Statement
- Dissertation Goal
- Relevance and Significance
- Summary

Background

Sensors are widely used in industry, daily life and medicine. Wearable or implanted medical devices, such as insulin sensors, have been used for several decades to non-invasively collect electrical, thermal and optical signals generated by the human body (Mosenia, Sur-Kolay, Raghunathan, & Jha, 2017). Vibration, temperature and noise sensors are used in nuclear power plants, gas turbines and continuous stirred-tank reactors, (Luo, Misra, & Himmelblau, 1999; Gribok, Hines, & Uhrig, 2000; Yan & Goebel, 2003). Numerous other examples of complex sensor-based systems that operate on data received from the sensors have been well documented. These observations suggest that, ultimately, the correct functioning of these systems strongly depends on the reliability of sensor data. For instance, Figure 1 (Geng, Tang, Ding, Li, & Wang, 2017) shows a multisensor-based noninvasive continuous glucometer that uses several sensors to measure blood glucose levels:

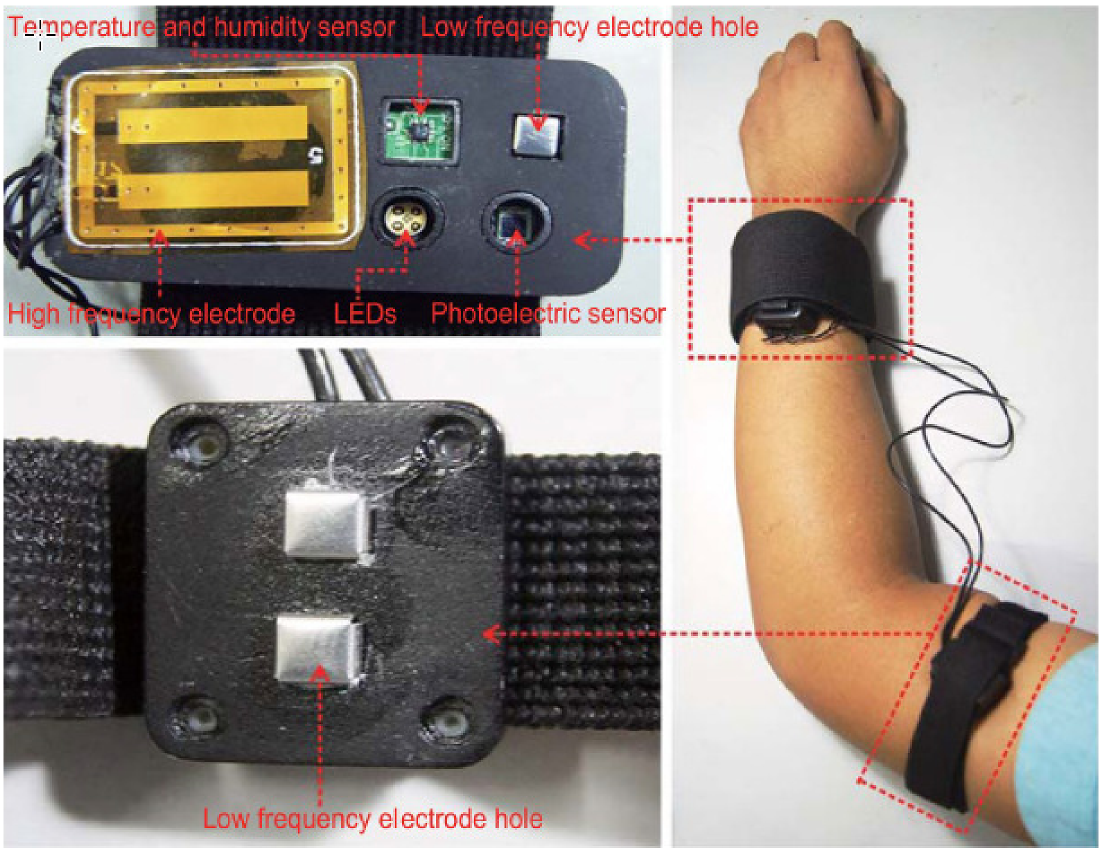

Temperature and humidity sensor
Low frequency electrode hole
High frequency electrode
LEDs
Photoelectric sensor
Low frequency electrode hole

Figure 1. A multisensor-based noninvasive continuous glucometer.

Unfortunately, sensors often fail. These failures can be caused by a variety of reasons, such as physical damage (both intentional and unintentional), manufacturer defects, software errors, unmonitored environmental conditions, incorrect calibration or configuration, improper human-computer interaction (HCI), misuse, and even hacking and abuse for malicious purposes (Van Der Meulen, 2004; Sikder, Petracca, Aksu, Jaeger, & Uluagac, 2018). When these failures occur, they lead to anomalies appearing in the data, making it permanently unreliable, or in the least, for a certain amount of time. When data with anomalies is fed upstream to high-level processing stages, this often leads to system's incorrect behavior of the system with unpredictable and even dangerous consequences, such as insulin overdose or nuclear plant shutdown. With sensors becoming so ubiquitous and implications of their failure becoming so significant, timely detection of anomaly has also become extremely important. Classification of anomalies is also of high priority, since it helps to identify the root cause of the failure and take relevant preventive actions.

Problem Statement

Detection and classification of anomalies in raw time-series sensors data in a way that does not require domain knowledge is an open research challenge. Until now, the vast majority of existing anomaly detection and classification methods have been based on hand-crafted features (Goldstein, & Uchida, 2016; Parmar, & Patel, 2017; Yi, Huang, & Li, 2017). These methods require considerable human effort from domain experts and are strongly tailored to specific system design and type of data.

Much scientific effort has been devoted to the subject of sensor data validation and various algorithms of anomaly detection and classification have been proposed for sensor-based systems over the past decades Yao, Cheng, & Wang, 2012; Pires, Garcia, Pombo, Florez-Revuelta, & Rodriguez, 2016). A detailed review of these methods appears in Chapter 2 of this dissertation.

The very first approaches were primitive and mainly involved sensors redundancy in hardware with further majority voting. Due to their high cost, they were soon replaced by a series of approaches that were based on mathematical (mostly statistical) models that spotted errors by quantifying the degree of relationship between the measured distribution and the predicted one.

However, over time, sensor-based systems became more complex and included a variety of sensors differing by characteristics, such as type, manufacturer, degree of quality, accuracy, stability, deterioration over time, resistance to noise and external factors (Worden, 2003; Jager et al., 2014). As a result, statistical models became impractical for these multipart systems (Pires et al., 2016), and were replaced by approaches based on classical ML models, such as shallow neural networks (SNN) (Hopfield, 1982), principal components analysis (PCA) (Pearson, 1901), Random Forest (Ho, 1995), etc. Unlike statistical approaches that focus on understanding the process that generated the data, ML techniques focus on constructing a mechanism that improves its accuracy based on previous results (Patcha, Park, 2007).

For more than two decades, various classical ML models have been successfully used to detect anomaly in sensor data (Wise, & Gallagher, 1996; Luo, Misra, & Himmelblau, 1999; Misra, Yue, Qin, & Ling, 2002; Tamura, & Tsujita, 2007; Auret, & Aldrich, 2010; Singh & Murthy, 2012; Jager et al., 2014; Cerrada, et al., 2016; Zhang, Qian, Mao, Huang, Huang, & Si, 2018). However, despite of their success, ML methods that involve manual feature extraction have two serious shortcomings: firstly, these methods cannot operate on raw data. Instead, they operate on features extracted from the data, which is labor intensive and requires considerable effort from domain experts. To construct them, a domain expert needs to take into consideration the peculiarities of the analyzed sensors to manually control and tune their input (Bengio, Courville, & Vincent, 2013). Secondly, such methods are domain specific. Being tailored to concrete sensor-based systems and concrete type of data, these approaches are highly sensitive to the slightest changes in the system or the data (Ibarguengoytia, 1997).

Unfortunately, modern sensor-based systems are very dynamic. Their complexity constantly increases, which requires periodical design changes. The nature of data that these systems rely on is also volatile and often changes. When these radical changes occur, previous feature-based solutions cease to function properly and require fundamental redesign or refactoring, which again results in considerable human effort.

These limitations make most of the current methods inefficient and point to the need for an approach that would provide significantly longer-lasting solutions to detect and classify anomalies, do not require domain knowledge, and can be relatively easily adjusted to new system design or new types of data. This is the challenge that makes data anomaly detection in sensor data an unsolved problem that requires further consideration.

Dissertation Goal

This dissertation systematically investigated the effectiveness of long short-term memory networks (LSTMs) (Hochreiter, & Schmidhuber, 1997) for anomaly detection based on raw sensor data. As a proof of concept, it planned to use time-series data on glucose level measurements. Samples of anomalous time-series sequences, generated by using six methods that represent common types of anomaly, were used along with non-anomalous time-series sequences. An LSTM network model was trained on a medical dataset with real data to classify raw time-series sequences into one of seven classes: non-anomalous, and classes corresponding to the six types of anomalies.

The model's generalization success was estimated by means of *k*-fold cross-validation (Kohavi, 1995). To evaluate its performance (classification accuracy) a confusion matrix was used to compute precision, recall, and F1 score (F-measure)

(Sammut, & Webb, 2017). Each of the three metrics examined in micro, macro and weighted perspective, resulting in a total of 9 metrics. As a control, the performance of the LSTM model, trained on raw time-series data, was compared to performance of traditional classifiers, such as SNNs, support vector machines (SVMs), random forests (RF) (Ho, 1995), and naive Bayes classifier (NBC) that were trained on hand-crafted features based on knowledge about the common sources of anomaly.

Relevance and Significance

As discussed in the Introduction section of this chapter, sensor-based systems are highly dependent on validity of data received from the sensors. Anomalies in sensor data caused by various failures lead to incorrect system behavior with unpredictable and even dangerous consequences. Therefore, precise and timely detection of anomalies and identification of their types is of extreme importance.

As previously mentioned, former methods, as well as vast majority of recent ones, operate on hand-crafted features, which results in significant limitations, such as considerable human effort and lack of resilience to changes in system or underlying data. It is clear that an alternative approach to anomaly detection and classification that would be able to process raw data and automatically generate the required features will be particularly valuable. One method that meets these requirements is LSTM. Research has shown that this contemporary ML model is very effective in automated extraction of abstract features from various raw time-series (Salehinejad, 2017). Moreover, LSTM is capable of learning long-term dependencies from sequential and time-series data (Dam et al., 2017; Karim, Majumdar, Darabi, & Chen, 2018). As a result, solutions based on LSTM have several substantial benefits:

- They remove the necessity for a domain expert.
- They significantly reduce human effort.
- They decrease design complexity.
- They automatically adjust to new types of data.
- They will have longer lifespans.
- They are automated to a large extent and require significantly less calibration to address concrete problems.

For reasons that remain unclear, LSTM is little used in anomaly detection and classification. This model has existed for over twenty years, but solutions based on LSTM have only started to appear in recent years (Pires et al., 2016; Parmar, & Patel, 2017). A thorough search through recent survey papers revealed that the number of proposed LSTM-based systems for anomaly detection in raw sensor data is negligible in comparison with systems based on feature engineering. These facts indicate that there is no confidence in the research community that LSTM is a promising approach for anomaly detection in raw time-series. However, despite the absence of proper recognition and attention from the research community, the aforementioned benefits of LSTM make it an extremely well-suited method for the above task.

Previous experimental supports this claim. For instance, Fehst, Nghiem, Mayer, Englert and Fiebig (2018) compared techniques based on manual feature engineering and feature subset selection for dimensionality reduction with automatic feature learning through LSTM. The results of their study show that LSTM has significantly higher accuracy. Other works (Chauhan, & Vig, 2015; Malhotra, Vig, Shroff, & Agarwal, 2015; Taylor, Leblanc, & Japkowicz, 2016; Yoshikawa, Belkhir, & Suzuki, 2017; Zhang, & Zou, 2018) have also reported on the successful use of LSTM for solving practical anomaly detection problems in domains such as medicine, automotive, power, aviation, etc. In his master's thesis, Singh (2017) claims that LSTMs are suitable for general purpose time-series modeling and anomaly detection and proves this by successfully applying the model on three real-world datasets from different domains. There are other works as well; the complete list appears in the *Existing LSTM-based Solutions* section of Chapter 2.

The proposed research considers the effectiveness of LSTM in modeling time-series of raw data, and its abilities of the capturing long range dependencies. It takes into account the full compatibility of the above qualities to the field of application, i.e. anomaly detection and classification in sensor data. Finally, it relies on the optimistic experience of and impressive results of existing works that recommended LSTM-based solutions. These three factors combined constitute a well-founded motive to consider further investigation of this pertinent and important approach.

While the proposed dissertation deals with a practical problem of detecting sensor faults, it also aims to contribute to the knowledge base of computer science. The study deals with the application of ML methods for analyzing sensor data and may provide new directions and ideas for the effective application of these methods. Particularly, the study is expected to contribute to an LSTM-based approach in the analysis of medical-sensor data. Finally, evaluation of the extent in which LSTM can replace approaches based on feature engineering will contribute to the study of the effectiveness and practical value of this model.

Summary

This chapter begins by providing some background information in anomaly detection. It introduces multi-sensor systems and their dependency on valid sensor data. It discusses the problem of sensors failures that cause anomaly in the data, and the importance of its timely detection and classification. The chapter then continues with a brief overview of existing methods, explains their dependence on manual extraction of features from the data, and the problem imposed by these limitations in light of complex and constantly changing modern systems. Next, the chapter defines the goal of the proposed dissertation: systematic investigation of the effectiveness of LSTM for anomaly detection based on raw sensor data and clarifies how it can be achieved by the planned study experiment. Finally, this chapter positions the importance of precise and timely detection of anomalies and identification of their types. It emphasizes the benefits of the LSTM model, its effectiveness and accuracy, and the its resulting eminent appropriateness as a method for detection and classification of anomaly in sensor data. These claims are justified by mentioning several relevant works of research. Finally, the chapter elaborates on the contribution of the proposed dissertation to the knowledge base of computer science.

The remainder of this dissertation proposal is organized as follows: Chapter 2 provides literature review on types of anomaly, detection and classification methods, feature engineering and others. It serves as the baseline and starting point of the research. Chapter 3 presents the methodology of the investigational portion of the study. It presents a detailed experiment design, choice and dataset preprocessing, and each of the five stages of the planned experiment. Chapter 4 shows the results of the experimental part of the study. Chapter 5, the last one in this dissertation, concludes results of the investigation and draws inferences from them.

Chapter 2 - Literature Review

The primary focus of this dissertation's research is detection and classification of anomaly in sensor data. There is a considerable research-community interest in this area. The following sections review the relevant literature:

- Anomalies as Outliers
- Anomalies in Sensor Data
- Taxonomy of Anomaly Detection and Classification Methods
- Techniques based on Feature Engineering
- Techniques based on Automated Feature Extraction
- Existing LSTM-based Solutions
- Summary

Anomalies as Outliers

Invalid data differs from valid data by the fact that it contains a certain amount of anomalous values that are discordant observations and do not conform to expected behavior (Hayes, & Capretz, 2014). Statistically, anomalies can be referred to as outliers in baseline distribution of the data or of its features (Stoecklin, 2006). Therefore, in the context of data validation, failure localization and recognition are equivalent to detection and classification of outliers. Anomalies can be classified into the following four categories (Parmar, & Patel, 2017):

1. *Point anomaly*. These are individual data points that are considered anomalous with respect to the remaining data. This type of anomalies refers to data points that

are given without contextual aspect, such as time. Figure 2 (Chandola, Banerjee, & Kumar, 2009) shows an example point anomaly in a 2D dataset:

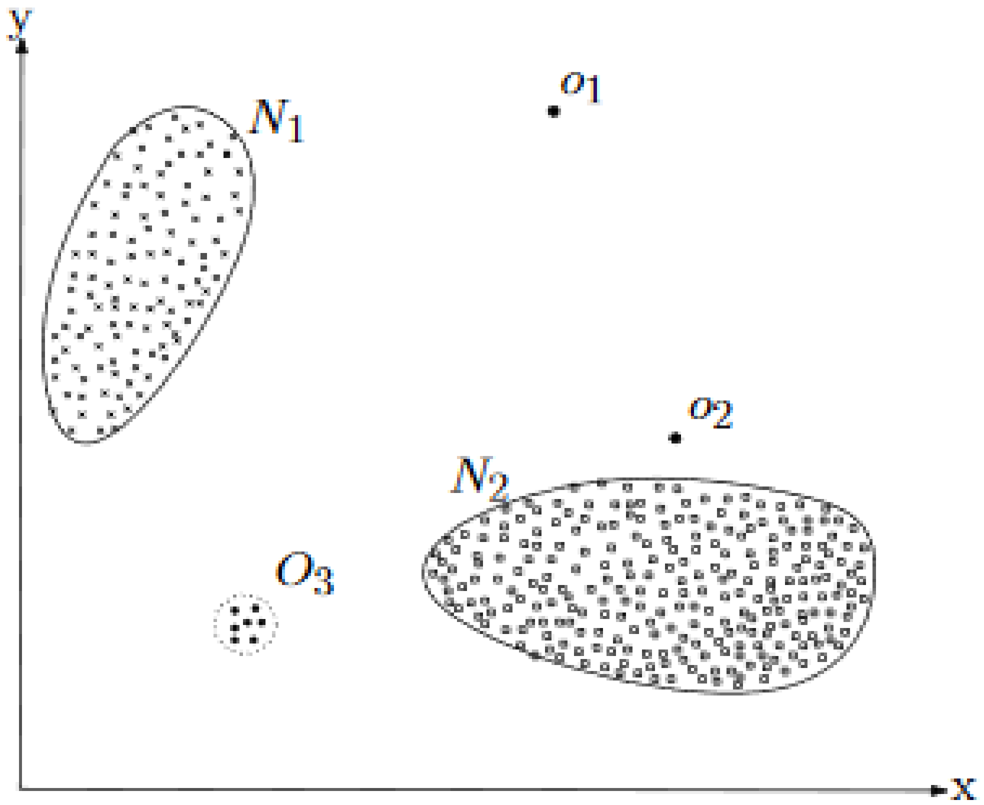


Figure 2. An example of point anomaly: points o_1 and o_2, and all the points within the region O_3 are outliers.

As seen in the Figure, most observational data points reside within the two regions N_1 and N_2. This data in these regions can be considered normal. On the contrary, points o_1 and o_2, and region O_3 are anomalous. They are sufficiently distant from the normal regions, which makes them outliers.

2. *Contextual anomaly* (also called conditional anomaly). These are individual data points that are anomalous in a specific context, but not otherwise. The context is defined by the structure of the dataset and is problem specific. Each data point is defined by attributes that define the neighborhood (context) of the point, and behavioral attributes that express its non-contextual characteristics. When dealing with time-series data, such as the one examined by this work, an outlier is defined

as a data point that must have both contextual anomaly and behavioral anomaly. In other words, it must have anomalous values that appear within an anomalous context. Figure 3 (Chandola, Banerjee, & Kumar, 2009) shows an example of contextual anomaly in a 1D time-series data of temperature over the year:

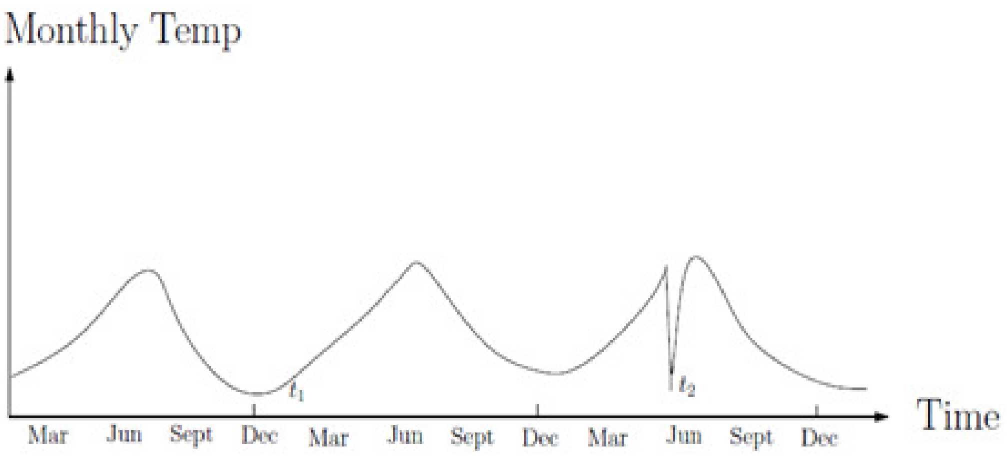


Figure 3. Example of contextual anomaly, t_2.

Points t_1 and t_2 have about the same low values. On the one hand, t_1 is considered normal since it stays within the climatic norm for the period of December to March. On the other hand, t_2 is an outlier, since such low temperatures is not expected in June. It is plain to see that the time context (months in this example) affect the normality of the data.

3. *Collective anomaly*. This type of anomaly refers to a collection of related data points, which are anomalous with respect to the entire dataset. Two notes are worth mentioning: firstly, individual data points in such an anomalous collection may not be anomalies by themselves, while only their common existence shows up as anomaly. Secondly is that this type of anomaly can appear in both context-aware data and contextless data. Figure 4 (Chandola, Banerjee, & Kumar, 2009) shows an example of collective anomaly in data from an ECG sensor:

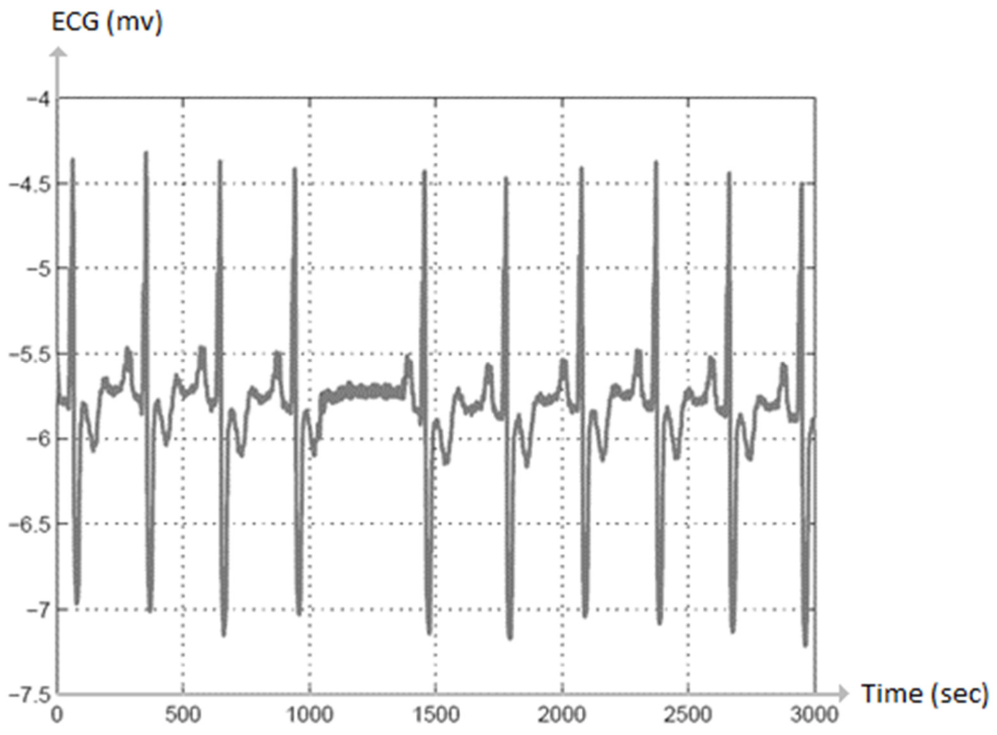


Figure 4. Example of collective anomaly, 1000-1500 secs.

Each individual data point in the close-to-zero section between 1000 and 1500 seconds is not an anomaly by itself - as seen on the graph that the curve of the periodic oscillations crosses zero value several times. However, if the number of subsequent low-value points is large enough, then they appear to be an anomaly as a group (in this case a premature atrial contraction, PAC).

Anomalies in Sensor Data

As previously mentioned, timely validation of raw sensor readings is critical to prevent invalid data from causing damage to the system, both detection and classification of invalid data are equally important. Whereas the first protects the system from being damaged by invalid sensor data readings, the second allows the nature of the fault to be understood and allow relevant corrections and improvements to be made in the sensor's system to prevent future reappearances (Scherbakov, Brebels, Shcherbakova, Kamaev, Gerget, & Devyatykh, 2017). A detailed review of data correction methods, given by Pires et. al. (2016), is beyond the scope of this work.

The task of outlier detection and classification is especially multifaceted and challenging when sensor data is concerned, and stems from the following factors (Chandola, Banerjee, & Kumar, 2009):

1. The boundary between normal and anomalous behavior is often not precise. Anomalous observations may lie very close to normal, and vice-versa, which makes it very difficult to define a region with every possible normal behavior.
2. When anomalies originate from malicious actions, they appear as normal sensor data, making it very difficult to define normal behavior.
3. A defined notion of normal behavior that was sufficiently representative at some point in time may stop being credible later. In many domains the normal behavior dynamically evolves as the system collects new data. Therefore, applying a technique of one specific domain to another is not often impractical.
4. When machine learning (ML) based anomaly methods are used, labeled data for training and validation of models is often a major issue.
5. The data is often interspersed with noise which tends to be quite similar to the actual anomalies. It is very difficult to distinguish and remove such noise.

Taxonomy of Anomaly Detection and Classification Methods

Through the course of history many scientists and researchers have devoted their time to the problem of detection and classification of anomaly in sensor data. One of the simplest sensor failure detection methods is to use multiple sensors of the same type, then compare the readings and decide by majority-voting which, if any, of them are damaged. Not to mention the fact that this redundancy method cannot classify the anomaly, it sometimes fails to detect the anomaly, e.g. if an environmental factor, such as noise, affects the values of all sensors.

Obviously, more advanced and generic methods were needed to serve complex sensor-based systems and started to appear as early as in the 19th century (Edgeworth, 1887). Researchers have formulated the problem of outlier detection and classification by adopting concepts from various disciplines, such as statistics, information theory, spectral theory, ML and data mining (Chandola, Banerjee, & Kumar, 2009). These formulations were based on model of normal data patterns and generation of outlier score for each new data sample. The following types of models have been suggested (Aggarwal, 2015):

1. *Extreme values-based models*. These models use boundary values to define the range of normal data. Outliers are data points that exceed these boundaries. Data points can be univariate as well as multivariate.
2. *Clustering-based models*. These models create clusters of data points that occur together. Outliers are data points that appear far away from the clusters.
3. *Distance-based models*. These models use the distribution of total distance from each data point to its *k*-nearest neighbors. For an outlier, this distance is significantly larger than for other data points.
4. *Density-based models*. These models use the local density of the data points. Outliers have low density.
5. *Probability-based models*. Similar to clustering-based models, these models create data points clusters. However, instead of using distance to determine outlier score, it is determined by the probabilistic fit of a data point.

6. *Information-theoretic models*. This type of model is significantly different from the previous ones. When constructing a normal data model, its space requirements are analyzed. Then, for each new point, the difference on space requirements between construction of a normal model with or without this data point is examined. If the difference is large, the point is reported as an outlier.

These models produced many techniques to anomaly detection and classification that can be grouped into three fundamental approach types (Hodge, & Austin, 2004):

1. *Type 1*. Techniques that detect the outliers with no prior knowledge of the data. In ML, these are unsupervised clustering methods. The models are constructed by processing all data, while the most distant points are marked as potential outliers.
2. *Type 2*. Techniques that are based on labeled data. This approach models both normal and abnormal data and uses supervised learning-based classification.
3. *Type 3*. Techniques of this type label only normal data. In this approach, also known as *novelty detection* or *novelty recognition*, semi-supervised ML algorithms, that are trained on the normal data, learn to recognize abnormality patterns by comparing them to those previously seen.

Techniques Based on Feature Engineering

In feature engineering, domain knowledge is used to analyze the raw data and process it to create informative hand-crafted features, which are then used to train the machine learning models. Below is a list of various feature-based methods, based on the works of Patcha and Park (2007), Yao et al. (2012), and Pires et al. (2016):

1. *Profile-based behavioral analysis*. The idea behind this method is to use hand-crafted features to create a profile of data that is considered normal. This profile

then serves as a gauge of valid data patterns. If new data significantly deviates from the profile, it is considered anomalous. A set of rules determines the conditions that trigger the detection of an outlier. This could be a significant deviation of a certain critical feature, a combination of important features, or a common deviation score. The work of Branisavljevic, Kapelan and Prodanovic (2011) used this method to create a real-time data anomaly detection scheme.

2. *Bayesian network techniques* (Friedman, Geiger, & Goldszmidt, 1997). This approach is based on a model that analyzes and encodes relationships of features in a dataset in a form or directed acyclic graph (DAG). The encoded relationships can be either casual or probabilistic, whereas the features are often chosen to be statistical. For a new sample, the model estimates its probabilistic behavioral likelihood, and decides whether it is normal or an outlier. Kruegel, Mutz, Robertson and Valeur (2003) applied this technique in a multi-sensor system for classification and suppression of false alarms.
3. *PCA and clustering techniques*. PCA (Pearson, 1901) significantly compresses dataset representation by converting data from the original feature space into a reduced one. The method preserves most of the information by leaving just a few principal components - linear combinations of original features that maximize the variance. After applying PCA, the clustering phase starts. Some metrics, e.g. Canberra, is applied to the normal part of the data to compute a centroid in the new feature space. The distance of new samples from the normal data centroid is large for anomalous samples and small for normal ones. This approach was used by Shyu and Chen and Sarinnapakorn and Chang (2003) in an unsupervised learning scheme for intrusion detection.

4. *Inductive rule generation techniques* (Quinlan, 1987). This approach derives a set of association rules and frequent patterns from the train set. Each rule implies the category (either normal or anomalous) of the target variable by a range of feature values. For example, Decision Trees (DTs) (Quinlan, 1986) implement generate impurity measures-based rules generation. Bombara, Vasile, Penedo, Yasuoka and Belta (2016) used custom rule generation scheme in DT to detect and classify anomalies in data of maritime environment and automotive powertrain system.
5. *Fuzzy logic techniques* (Novak, Perfilieva, & Mockor, 2012). This approach analyzes the features and builds a set of fuzzy rules that describe behavioral patterns of normal data, which are then used to form intervals of normal and anomaly data. For new samples, the rules determine whether they fall inside a normal interval or in one of the anomaly intervals. Linda, Manic, Vollmer, and Wright (2011) have proposed a fuzzy logic-based algorithm anomaly detection and classification in network cyber security sensors.
6. *Genetic algorithms (GAs)* (Whitley, 1994). Originally these evolutionary algorithms used mutation, crossover and selection operators to find solutions for optimization and search problems. However, due to their flexibility and robustness, they have been adjusted for other uses. One adaptation is to use a GA to derive highly effective classification rules. Another is to use GA to select an effective subset of features for other ML algorithms. Both customization methods have been put into practice in detection and classification of anomaly in network traffic (Hassan, 2013; Chouhan, & Richhariya, 2015).
7. *Traditional ML models*. This approach uses traditional ML models, such as SNNs, SVMs (Cortes, & Vapnik, 1995), RFs (Ho, 1995), and NBCs (Rish, 2001) data. When traditional ML models are used, anomaly detection and classification is

carried out as supervised learning task. For instance, Gribok, Hines and Uhrig (2000) have used an SVM to validate labeled nuclear reactor sensor data.

Techniques Based on Automated Feature Extraction

ML algorithms based on feature engineering show excellent results. However, as previously mentioned, manual creation of the features is both domain specific and labor intensive. The accuracy of classification that relies on feature engineering largely depends on how well the hand-crafted features are constructed (Bengio, Courville, & Vincent, 2013). To cope with these shortcomings, recent research is directed to find models that are capable of automated feature extraction (Salahat, & Qasaimeh, 2017). An exhaustive literature search (Hodge, & Austin, 2004; Chandola, Banerjee, & Kumar, 2009; Chandola, Cheboli, Kumar 2009; Parmar, & Patel, 2017; Heaton, 2017; Fehst et al., 2018) reveals several types of approaches that use automated feature extraction, also known as automated feature engineering and automated feature learning:

1. *Fuzzy logic-based approach.* Recently, methods that use fuzzy logic have been extended to automate the production of association rules. Modern fuzzy-logic-based methods use sliding window method to divide the time-series into portions (subsequences). Further, an algorithm of the fuzzy logic inheritance system analyzes these subsequences, looking for general patterns in their shape and amplitude. Since the analysis is carried out in an automated way, these patterns serve as the automatically created features for subsequence. The algorithm also automatically produces a set of association rules that define the order of appearance, mutual dependency relations and other characteristics of internal

associations. Izakian and Pedrycz (2013) have used fuzzy-logic to detect anomalies in precipitation measurements and arrhythmia ECG signals.

2. *Genetic programming-based approach*. In traditional ML, GP methods have been used to select an effective subset of features to reduce the cost of computation during the classification process and improve its efficiency. These methods have been recently significantly enhanced and awarded new capabilities. The first is the ability to derive (synthesize) new features from existing ones; another is to derive new features directly from raw data. Both abilities assume no prior knowledge on the probabilistic distribution of the data, which allows GP methods to be used for automated feature extraction. Guo, Jack and Nandi (2005) have used GP-based technique to detect and classify faults in raw data of vibration rotating machine.
3. *Deep feedforward neural networks-based approach*. When the dataset is large enough, special types of neural networks can be used for automated feature extraction. Two types of networks have been used for detection and classification of anomaly in raw data:
   A. *Self-organizing map* (SOM) (Kohonen, 1990). These are competitive learning algorithms for classification problems that do topological mapping from the input space to clusters. To be more specific, SOMs find statistical relationships between data points in a high dimensional space and convert them to geometrical relationships in a two-dimensional map formed by the output neurons. Recent enhancements allowed SOMs to automatically extract important features from raw input data and store them in a structure that preserves the topology. The weight vectors of the output neurons serve as prototypes of the data points and as centroids of clusters of alike data points.

Barreto and Aguayo (2009) have used several variations of SOM to detect anomalies in data of a solenoid sensor in NASA's hydraulic valve.

B. *Radial basis function (RBF) neural networks* (Lowe, & Broomhead, 1988). A classical RBF network is a three-layer feedforward neural network in which hidden layer nodes use RBF as a nonlinear activation function. The hidden layer of an RBF network performs a nonlinear transformation of the input, whereas its output layer maps the nonlinearity into a new space. RBF optimization is linear and is carried out by adjusting the weights to minimize the mean square error (MSE). Improvements made by Lowe and Tipping (1997), Karayiannis and Mi (1997), and Rojas et al. (2002) allowed to extract features from an RBF network and resize the network as required, making it possible to use the RBF networks with time-series, e.g. to detect novelty elements (Oliveira, Neto, & Meira, 2004).

4. *Recurrent neural networks* (RNNs) (Bengio, Simard, & Frasconi, 1994). An RNN is comprised of multiple copies of the same modules, each being a neural network that passes a message to its successor. RNNs have two inputs: the present and the recent past. Each recurrent module serves as a memory cell. This makes RNNs perfectly suited for ML problems that involve analysis of sequential data with temporal dynamics, such as time-series data of a sensor. Weights are applied to both current and previous input and can be adjusted by the RNN gradient descent and backpropagation through time (BPTT) (Werbos, 1988). RNNs can operate on raw data and do not require hand crafted features. Like deep feedforward neural networks, they learn appropriate feature representations in their hidden layers (Lipton, Berkowitz, & Elkan, 2015). An ordinary RNN has short-term memory, which means that it can only catch time dependency in rather short sequences of

data. To overcome this limitation, two special types of RNNs have been invented: long short-term memory (LSTM) and gated recurrent unit (GRU) (Cho et al., 2014). Both have more complex (gates-based) architecture of each recurrent module, which lets them more accurately maintain memory of important correlations and analyze much longer sequences. In LSTM, the architecture of each recurrent module is more complicated and uses more gates, whereas a GRU module exposes the full hidden content without taking control over the flow of data. As a result, LSTM is more powerful, while GRU is computationally more efficient (Chung, Gulcehre, Cho, & Bengio, 2014). Shipmon, Gurevitch, Piselli and Edwards (2017) have used ordinary RNNs, LSTM and GRU to detect anomaly in time-series data of network traffic.

Existing LSTM-Based Solutions

Since this dissertation focuses on LSTM, it is worth mentioning existing work on the subject of detection and classification of anomaly in time-series data. During the last few years, various LSTM-based solutions have been proposed by the research community to detect anomaly in time-series data from various domains, such as medicine (Chauhan, & Vig, 2015), space (Hundman, Constantinou, Laporte, Colwell, & Soderstrom, 2018), automotive (Taylor, Leblanc, & Japkowicz, 2016), power (Malhotra et al., 2015), astronomy (Zhang, & Zou, 2018), web traffic (Kima, & Cho, 2018), machinery (Singh, 2017), economy (Ergen, Mirza, & Kozat, 2017), etc.

An especially interesting piece of research was carried out by Fehst et al. (2018). Similar to this dissertation, the authors used LSTM to detect anomaly in time-series data of sensors that measured drinking-water quality. LSTM's automatic feature learning was compared to a traditional ML method called logistic regression, which used manually prepared features. Experiment results (F1-score) show that LSTM classification quality was superior to that of the traditional approach.

Summary

This chapter began by the describing anomalies as outliers and reviewed several types of the latter. It then observed the peculiarities of detecting anomaly in sensor data. The chapter continued by examining main anomaly detection and classification techniques based on both feature engineering and automatic feature extraction. The chapter ended by listing a number of LSTM-based solutions to anomaly detection in time-series data in multiple domains. The next chapter refers to methodology of the proposed dissertation.

Chapter 3 - Methodology

The proposed dissertation investigates the use of LSTM model for the problem of anomaly detection in sensor data. Sensor data can be seen as a time-series, while, as previously mentioned, LSTMs are well-suited for finding difficult patterns in the shape and amplitude of this kind of data. In addition, LSTMs are capable of automatically extracting features from raw time-series data. Therefore, the choice of model in this work can be considered convenient for this task. In order to control the LSTM and estimate its abilities of automated feature extraction, the same problem was solved using several traditional ML algorithms that operated on hand-crafted features. The following ML methods have been selected: SVM, Random Forest, naive Bayes classifier, and shallow neural network. Since this work focuses on sensor data, all five algorithms were trained and tested on a dataset that contains sensor measurements. To realistically demonstrate both experimental approaches, the dataset was carefully chosen.

This chapter describes the planned experiment in the following sections:

- Experiment Design
- Chosen Dataset
- Data Cleaning and Preprocessing
- Experiment 1: Classification via SVM
- Experiment 2: Classification via Random Forest
- Experiment 3: Classification via Naive Bayes
- Experiment 4: Classification via a Shallow Neural Network
- Experiment 5: Classification via LSTM

Experiment Design

Below are the design decisions for each stage of the experimental work:

1. *Choosing a dataset*. A dataset must be carefully selected to match the goal, scope and facilities of the experiment (LaLoudouana, Tarare, Center, & Selacie, 2003). For the proposed study, the following of dataset requirements have been devised:

a. The dataset must contain measurements of real sensors, and have sufficient data for both training and testing.

b. The dataset should be well-balanced, having each class represented by roughly the same number of samples.

2. *Preprocessing the data*. The data may suffer from phenomena such as clearly illegal values, missing values, incomplete samples etc. In this case it needs to be cleaned or completed. E.g., in cases where anomalous samples are missing these samples need to be generated and added to existing data. Finally, the data needs to be adjusted to the specific experiments. In case of the proposed work, the samples need to be converted to feature vectors and time-series.

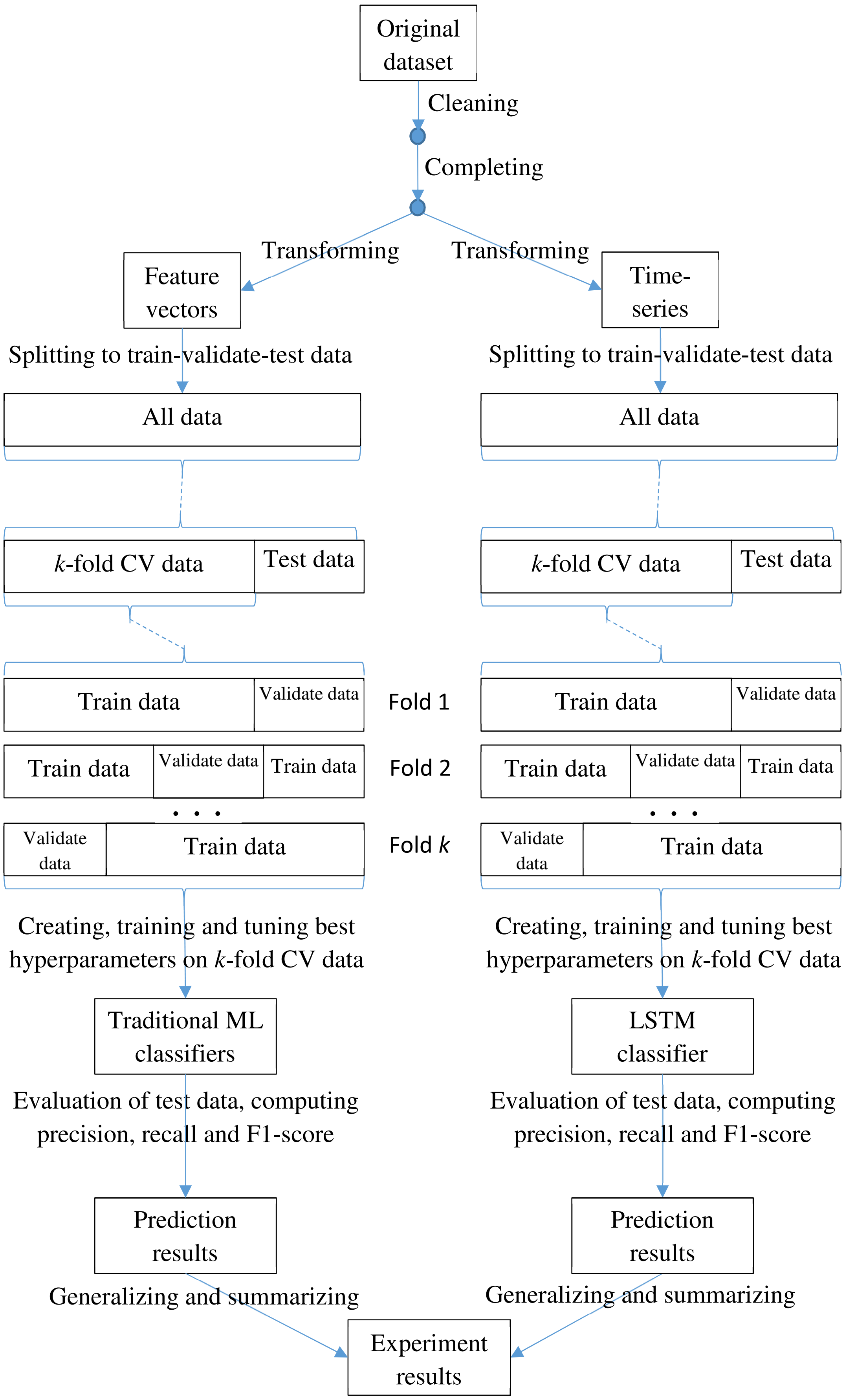
3. *Choosing the classifiers*. As previously mentioned, anomaly detection and classification should be carried out by using ML algorithms of two types. The first is based on traditional feature-based ML algorithms, namely SVM, RF, NBC and SNN. The second type, an LSTM network, should operate on raw time-series and automatically extract features. Altogether, five classification experiments are to be conducted. First four experiments estimate anomaly detection and classification abilities of the traditional ML models. These models should be trained and validated on feature vectors that were prepared during the dataset preprocessing phase. The fifth classification experiment estimates the ability of LSTM to

automatically extract features from raw time-series data and use this knowledge to detect and classify the anomaly. Due to wise construction of statistical domain knowledge-based features, the simpler algorithms can achieve very high classification accuracy. To assess the benefits of LSTM, its accuracy should be matched to that of the traditional models.

4. *Training and cross-validating the classifier*. One of the main requirements of ML is to build a computational model with both high prediction accuracy and good generalization abilities (Mitchell, 1997). Improperly trained model memorizes the training examples and overfits, resulting in poor generalization on unseen data. To train a good model the bias-variance tradeoff (Kononenko, & Kukar, 2007) should be considered, i.e. the right balance between good prediction on training data and good generalization of new data needs to be found. Cross-validation (CV) (Picard, & Cook, 1984) is considered a conventional approach to ensure good generalization and prevent overfitting. *k*-fold CV is a slightly more effective (Reitermanova, 2010) variation of CV that also automates the process of dataset partitioning. To keep the models of this study robust and resistant to overfitting, each they should be trained through 10-fold CV.
5. *Tuning the classifiers*. Tuning a model's architecture and hyperparameters can significantly improve its accuracy. This type of optimization relies less on theory and more on experimental results. Following the work of Bergstra and Bengio (2012), this study should apply two most popular strategies for hyperparameter optimization: automated grid search and manual search. For traditional ML models that require significantly less time to train, grid search through 10-fold CV will be used to find the best combination of architecture and hyperparameters. To prevent overfitting, the models should be trained through cross-validation. For the LSTM

the long training times require manual search of the optimal setup. However, the LSTM should be also trained through 10-fold CV.

6. *Testing the model.* After being trained, cross-validated, and tuned, the model's accuracy should be tested. To ensure credible testing, the JDRF dataset should be split in a way called train-cross-validate-test, following the recent trend (Kuhn, 2013). This method splits the data into two parts with ratio of 0.9. Training and cross-validation data is used for training of the model and finding the best set of hyperparameters via *k*-fold CV, while the testing data is used for testing.
7. *Evaluating the model.* In order to keep the evaluation results trustworthy and ensure that the accuracy is measured from every angle, this study should follow the recommendations of Goutte and Gaussier (2005). Once tested, a confusion matrix should be constructed from computed true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) test results. Then, the following nine metrics should be computed: micro-precision, macro-precision, weighted precision, micro-recall, macro-recall, weighted-recall, micro-F1 score, macro-F1 score, and weighted F1-score. In LSTM, there should be another two metrics: the interrelations between accuracy and loss of the model, and the number of epochs elapsed during the training process. This will allow to estimate the effectiveness of the model apart from its accuracy.
8. *Reasoning on LSTM's advantages.* Estimation of the algorithms' classification accuracy should serve as the quantitative analysis that is required to examine its effectiveness. By comparing the overall classification results of the traditional ML algorithms with the LSTM, it should possible to discern if the expected results in automated feature engineering have been achieved and if the LSTM can be used as a preferable alternative to classical models in the domain of sensor data.



Original dataset
Cleaning
Completing
Transforming
Transforming
Feature vectors
Time-series
Splitting to train-validate-test data
Splitting to train-validate-test data
All data
All data
k-fold CV data
Test data
k-fold CV data
Test data
Train data
Validate data
Fold 1
Train data
Validate data
Train data
Validate data
Train data
Fold 2
Train data
Validate data
Train data
Validate data
Train data
Fold k
Validate data
Train data
Creating, training and tuning best hyperparameters on k-fold CV data
Creating, training and tuning best hyperparameters on k-fold CV data
Traditional ML classifiers
LSTM classifier
Evaluation of test data, computing precision, recall and F1-score
Evaluation of test data, computing precision, recall and F1-score
Prediction results
Prediction results
Generalizing and summarizing
Generalizing and summarizing
Experiment results

Figure 5. Design of the planned experiment.

9. *Generalizing and summarizing the experiment results*. The experiments should provide sufficient "food for thought" from their results to facilitate the addressing of the following questions:
   - What is the meaning behind the achieved results?
   - Could these results be foreseen?
   - Would the results have been roughly the same if another dataset was used?

Figure 5 visualizes the phases of the planned experiment. It can be seen that the flow of the traditional ML algorithms and the flow of the LSTM are very much alike. The only difference is the type of data they are applied on. This aspect keeps the experiment straightforward, equitable and precise.

Chosen Dataset

For the proposed dissertation, the Juvenile Diabetes Research Foundation (JDRF) Continuous Glucose Monitoring (CGM) Clinical Trial dataset (JDRF CGM Study Group, 2008) was chosen. This dataset contains genuine measurements of blood glucose levels, which deserve to be called sensor data since blood glucose meters and similar devices are sensor-based. An analysis of the chosen dataset both quantitative and qualitative (Verner, & Butvinik, 2017) revealed that its data is heterogeneous, inconsistent, partially complete and has uneven number of measurements per day. Such data is highly realistic and is expected to provide veridical experiments.

The clinical population examined to prepare the dataset consisted of both healthy people and patients suffering from intensively-treated type 1 diabetes and glycated hemoglobin (HbA1c). Patients were divided into three age groups (over 25 yrs., 15-24 yrs., 8-14 yrs.). The measurement period for every patient was about 26 weeks. Table 1 (Verner, & Butvinik, 2017) shows blood glucose level ranges for the patient groups:

Table 1. Blood glucose level ranges for different patient groups.

Group	Age	Glucose Level Range in mg/100 ml		
		Fasting	1 r. After Meal	2 hr. After Meal
Normal w/o Diabetes	11-61	<90	<140	<110
Obese w/o Diabetes	14-62	<90	<140	<110
Normal w/ Diabetes	24-69	80-120	>170	>140
Obese w/ Diabetes	15-65	80-150	>170	>140

The total number of samples in the dataset was 772,061 samples, which is sufficient amount of diverse data to train and test both traditional ML algorithms and LSTM.

The chosen dataset had a couple of issues. First, it contained only normal data and no anomalous samples. This problem was solved by manually generating anomalous data with different types of anomaly from the original data. Another problem was that the data was in rather inconvenient form of numbered measurements with patient IDs and timestamps. The solution was to regroup the data in a way that provided both feature vectors and time-series. The next section covers the details of these procedures.

Data Cleaning and Preprocessing

The dataset was reorganized in a way that placed all subsequent daily measurements of a single patient as a time-series. Since the number of measurements per day differed by both dates and patients, this produced a large amount of variable length time-series. Several series appeared to be too short due to missing or corrupted data. These series provide an insufficient amount of information to be analyzed when detecting outliers and were, therefore, deleted from the dataset. After cleaning out the short time-series, the dataset had 3,826 normal time-series left. Based on the normal sequences, six anomalous time-series were created by embedding different anomalous elements into the normal series. The following six patterns of anomaly were used:

1. Random numbers of extremely large, or extremely small values.
2. Chaotic subsequences of arbitrary length, in which elements significantly and unexpectedly vary from one another.
3. Exorbitantly-long subsequences with constant value.
4. Unusually large number of appearances of certain value.
5. Excessively long subsequences of strictly increasing or decreasing values.
6. Excessively long subsequences of two alternating elements.

This resulted in 3,826 x 7 = 26,782 time-series, each labelled by its anomaly type.

As previously mentioned, 90% of the data was devoted to training with 10-fold cross-validation, where the remaining 10% were used for testing. Table 2 specifies the number of samples in each category:

Table 2. Detailed data split by categories.

Category	Number of Samples		
Training with cross-validation - Fold 1	2410		
Training with cross-validation - Fold 2	2410		
Training with cross-validation - Fold 3	2410		
Training with cross-validation - Fold 4	2410		
Training with cross-validation - Fold 5	2410		
Training with cross-validation - Fold 6	2410	24103	26782
Training with cross-validation - Fold 7	2410		
Training with cross-validation - Fold 8	2410		
Training with cross-validation - Fold 9	2410		
Training with cross-validation - Fold 10	2413		
Testing	2679		

Such amount of data was sufficient amount for both traditional and deep ML. The necessary data for the LSTM was prepared. However, to properly finish the transformation phase it was required to convert each time-series into a corresponding feature vector, to be used by traditional ML algorithms. To prepare a small number of precise and informative features, this study analyzed the nature of the data and used the available domain knowledge. Blood glucose levels act as a bounded curve that fluctuates over time, which is typical for sensors that measure physical quantities. One important task was to find the boundaries of the data. Figure 6 (Verner, & Butvinik, 2017) shows a histogram of the values and their number of appearances:

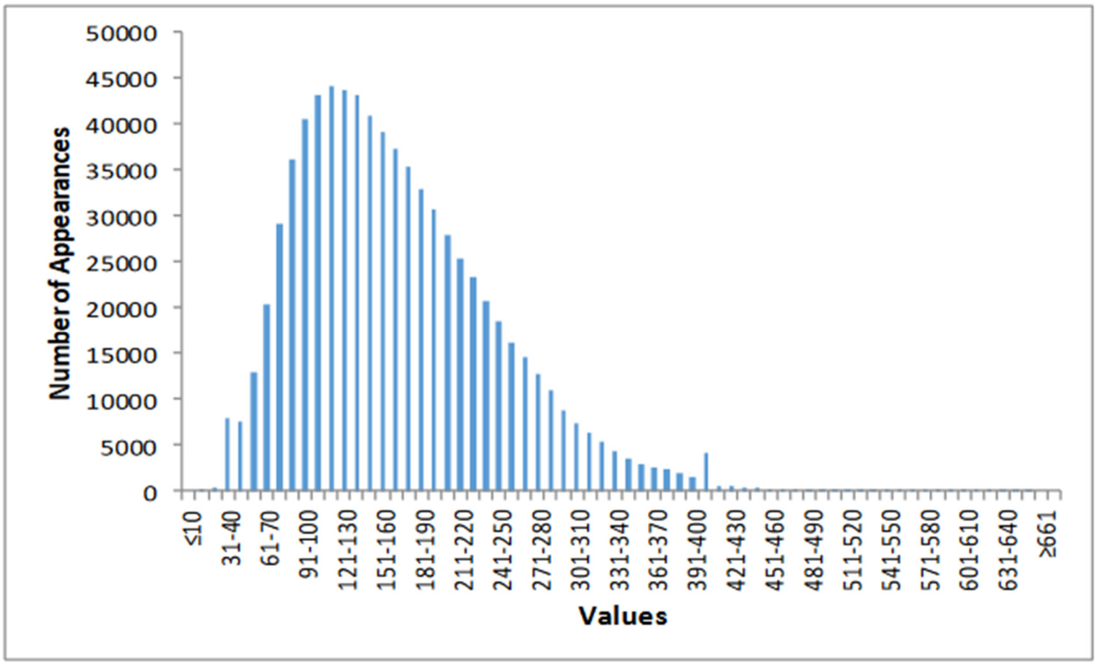


Figure 6. Histogram of JDRF values and their number of appearances.

As seen in the Figure, the JDRF dataset contains both extremely low and extremely high values, such as 19 and 471 mg/dL. Since higher values are occasionally inherent to the human body, it was decided to extend the valid range to [19, 800] mg/dL.

The distribution form has asymmetric Gaussian distribution, which is known to be well described by statistical functions (Gupta, Nguyen, & Sanqui, 2004). The latter implies that manually created features should have statistical nature. In recognition of this fact and the domain knowledge on the types of anomaly, this study proposes eight statistical features that were manually created to estimate the boundaries, averages, deviations and fluctuation patterns of sensor data to identify the six anomaly types.

Each feature operates on a single time-series of an arbitrary length (yet, long enough to be left in the dataset). Taken altogether, the features form a (labeled) feature vector of fixed length. Table 3 describes these features and the anomaly types they identify:

Table 3. Hand-crafted features for traditional ML algorithms.

Feature Verbal Description	Feature Mathematical Description	Identified Anomaly Patterns
Minimal value	$f_1 = MIN\left\{x^{(i)\langle 1\rangle}, x^{(i)\langle 2\rangle}, \ldots, x^{(i)\left\langle T_x^{(i)}\right\rangle}\right\}$	1
Maximal value	$f_2 = MAX\left\{x^{(i)\langle 1\rangle}, x^{(i)\langle 2\rangle}, \ldots, x^{(i)\left\langle T_x^{(i)}\right\rangle}\right\}$	1
Maximal absolute difference between adjacent elements	$f_4 = \left\lvert x^{(i)\langle k\rangle} - x^{(i)\langle k-1\rangle}\right\rvert \mid \forall j \neq k :$ $\left\lvert x^{(i)\langle k\rangle} - x^{(i)\langle k-1\rangle}\right\rvert \geq \left\lvert x^{(i)\langle j\rangle} - x^{(i)\langle j-1\rangle}\right\rvert$	2
Ratio of maximal number of succeeding repetitions of a constant element to the overall sequence length	$f_5 = \frac{MAX_j\left\{j \mid \exists k : x^{(i)\langle k\rangle} = x^{(i)\langle k+1\rangle} = \ldots = x^{(i)\langle k+j-1\rangle}\right\}}{\left\langle T_x^{(i)}\right\rangle}$	3
Mode frequency	$f_3 = COUNT\left(x^{(i)\langle k\rangle}\right) \mid \forall j \neq k :$ $COUNT\left(x^{(i)\langle k\rangle}\right) > COUNT\left(x^{(i)\langle j\rangle}\right)$	4
Ratio of length of the longest sequence of strictly increasing succeeding elements to the overall sequence length	$f_7 = \frac{MAX_j\left\{j \mid \exists k : x^{(i)\langle k\rangle} < x^{(i)\langle k+1\rangle} < \ldots < x^{(i)\langle k+j-1\rangle}\right\}}{\left\langle T_x^{(i)}\right\rangle}$	5
Ration of the length of the longest	$f_7 = \frac{MAX_j\left\{j \mid \exists k : x^{(i)\langle k\rangle} > x^{(i)\langle k+1\rangle} > \ldots > x^{(i)\langle k+j-1\rangle}\right\}}{\left\langle T_x^{(i)}\right\rangle}$	5

sequence of strictly decreasing elements to the overall sequence length		
Length of the longest sequence with two alternating constant elements	$f_9 = MAX_j \left\{ \begin{array}{l} \quad j \mid \exists k,l,m: \\ \quad l \neq m \qquad\qquad \wedge \\ l = x^{(i)\langle k \rangle} = x^{(i)\langle k+2 \rangle} = \ldots = x^{(i)\left\langle k+2\left\lfloor \frac{j-1}{2} \right\rfloor \right\rangle} \wedge \\ m = x^{(i)\langle k+1 \rangle} = x^{(i)\langle k+3 \rangle} = \ldots = x^{(i)\left\langle k+2\left\lfloor \frac{j}{2} \right\rfloor -1 \right\rangle} \end{array} \right\}$	6

Verner and Butvinik (2017) trained an SVM on the features extracted from the JDRF data that were much like those discussed above. Their model achieved 100% precision and 99.22% recall in binary classification of the data to anomalous and valid. These results were sufficient evidence to assume that if thoughtful hand-crafted features will be used, traditional ML models will be able to achieve high accuracy in anomaly detection and (multiclass) classification.

Experiment 1: Classification via SVM

Support vector machine (Cortes, & Vapnik, 1995) is a supervised learning algorithm that that can be employed for both classification and regression tasks. This ML model is rather simple and is based on feature engineering. However, when the features are properly chosen, SVM becomes a very powerful tool for learning complex non-linear functions. SVM considers the feature vectors as data points in multi-dimensional space and tries to find the best segregation of these points into two classes.

In the simpler case of linearly-separable data, the SVM acts as a linear classifier. In this case, its decision on the class of each point is based on the value of a linear combination of its features. In more complicated cases, when the data is not linearly inseparable, SVM can still be applied by using the a (non-linear) kernel method. The use of non-linear kernels, e.g. radial basis function (RBF), is also known as the *kernel trick* (Hofmann, Scholkopf, & Smola, 2008). SVM separates the points of different classes by drawing a boundary referred to as hyperplane, which is determined by two factors: support vectors and margins. With a high enough number of dimensions, a hyperplane that separates the classes can always be found. Support vectors are critical points (at least one of each class) that are closest to the hyperplane. Margins are the distances between the hyperplane and the nearest data points (of either class). The goal of SVM is to find the optimal hyperplane with the greatest possible margins that would increase the chances of new data to be correctly classified.

SVM can be used not just for binary classification, but also for multiclass classification. In the latter case, hyperplane construction is repeated a number of times,

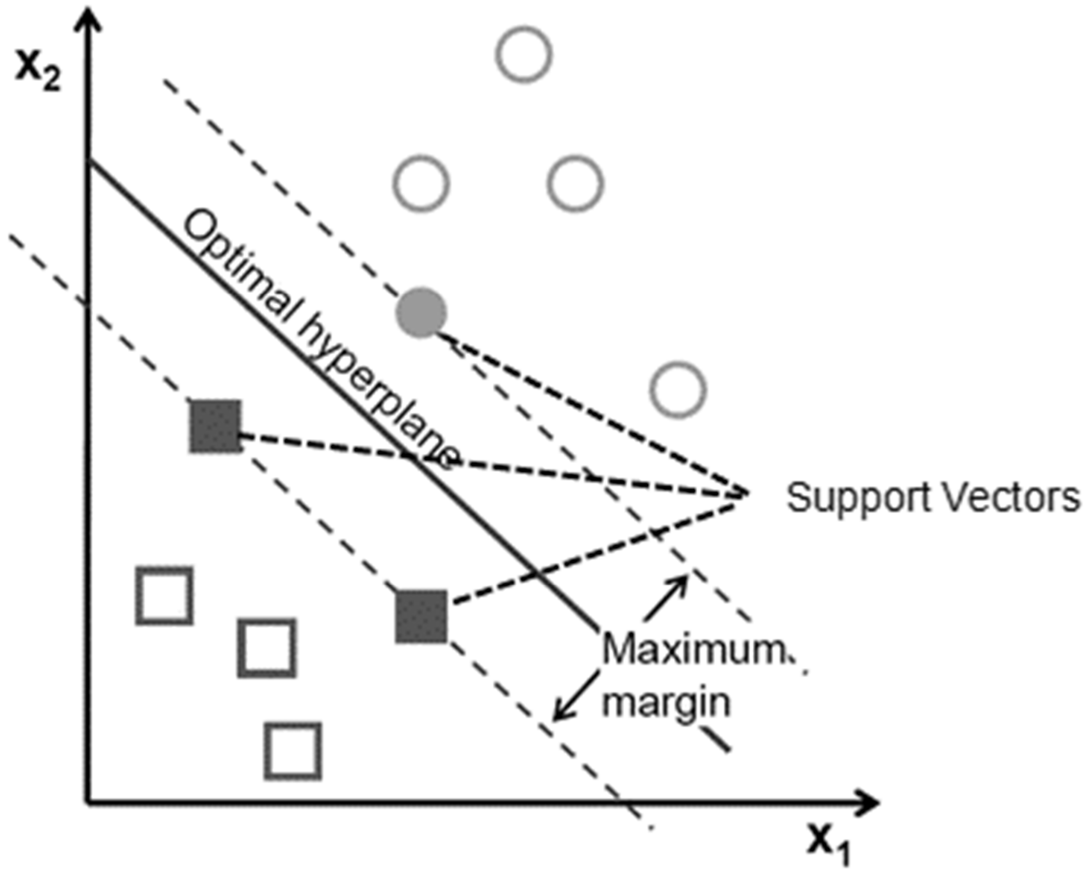


Figure 7. Using SVM to separate two-dimensional data.

until data points of all classes are delineated and separated in the multi-dimensional space. Figure 7 shows a simple case of an SVM that separates two-dimensional data:

Below is a list of SVM's advantages and disadvantages by Auria and Moro (2008):

Advantages:

- Convex optimization assures optimal classification of linearly separable data.
- Works well on small datasets.
- Can be used with regularization techniques to reduce overfitting.
- Is very accurate with a custom domain knowledge-based kernel function.

Disadvantages:

- Significantly longer training times in case of large datasets.
- Less effective on datasets with noisy data and overlapping classes.
- Can be strongly affected by overfitting if inappropriate kernel method is chosen.

To achieve the optimal accuracy on a concrete dataset, the hyperparameters of the SVM should to be tuned. Below is a list of those, based on works of Duan, Keerthi and Poo (2003), and Eitrich and Lang (2006):

- Cost parameter, noted as C.
- Kernel method.
- Free parameter of the RBF kernel, noted as γ (or, sometimes, as σ).

Experiment 2: Classification via Random Forest

Random forest or random decision forest (Ho, 1995) is an ensemble feature-based learning method for both classification and regression. RF is known to provide high accuracy even without fine tuning (Fernandez-Delgado et al., 2014), which makes it a popular ML algorithm. The basic building blocks of a random forest are decision trees

(Breiman, 1984). This simpler model is a flowchart-like structure that is constructed in a top-down manner. Every internal node in a decision tree is a condition on a single

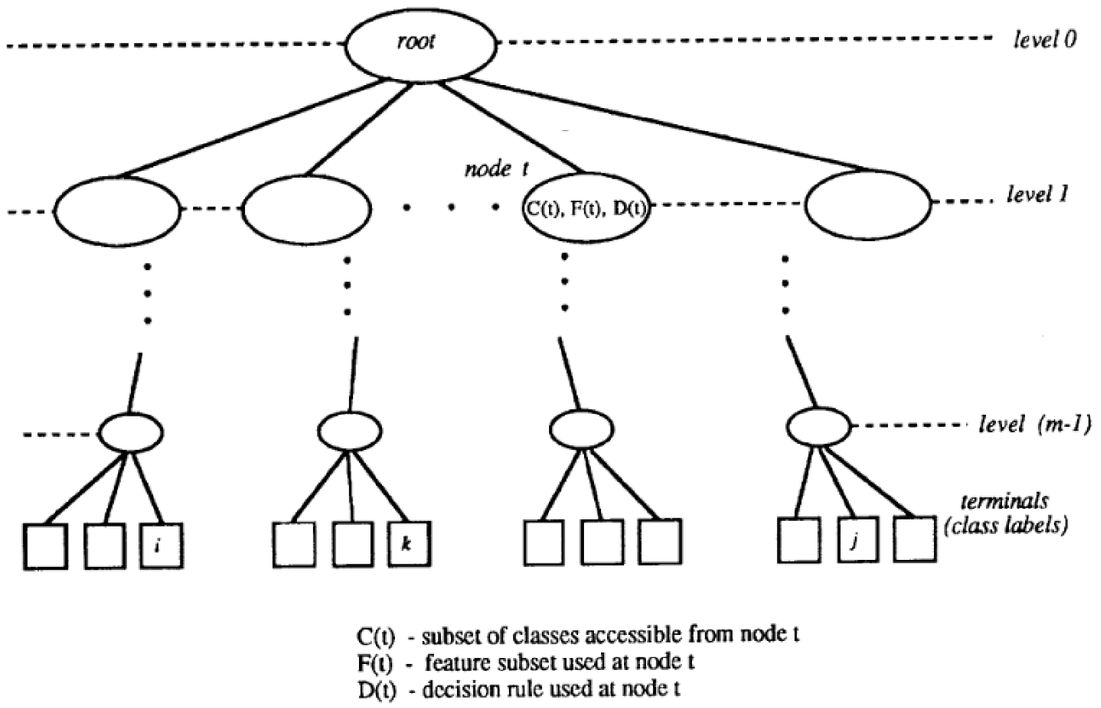

root
level 0
node t
C(t), F(t), D(t)
level 1
level (m-1)
terminals (class labels)
i
k
j
C(t) - subset of classes accessible from node t
F(t) - feature subset used at node t
D(t) - decision rule used at node t

Figure 8. Example of general balanced decision tree.

feature that is intended to split the data by similar response values. Figure 8 (Safavian, & Landgrebe, 1991) shows an example of general balanced decision tree:

Construction of decision tree with optimal binary splits is an NP-complete problem (Laurent, & Rivest, 1976). Nevertheless, different metrics are used to locally choose optimal features and conditions at each split based on informativeness criteria called impurity. For classification tasks, two of most widely used metrics are Gini Index and Information Gain, while for regression trees the popular metrics is variance (Raileanu, & Stoffel, 2004).

A serious shortcoming of decision trees is their high sensitivity to noise and susceptibility to overfitting on the training data (Ho, 1995). To deal with this issue, RF builds a random ensemble of decision trees, then combines and averages their predictions to get more accurate and stable results. For classification problems the final class is a result of majority-voting, i.e. it is the most frequent class; and for regression, the mean value is chosen. Figure 9 shows how the process of majority-voting in a typical random forest:

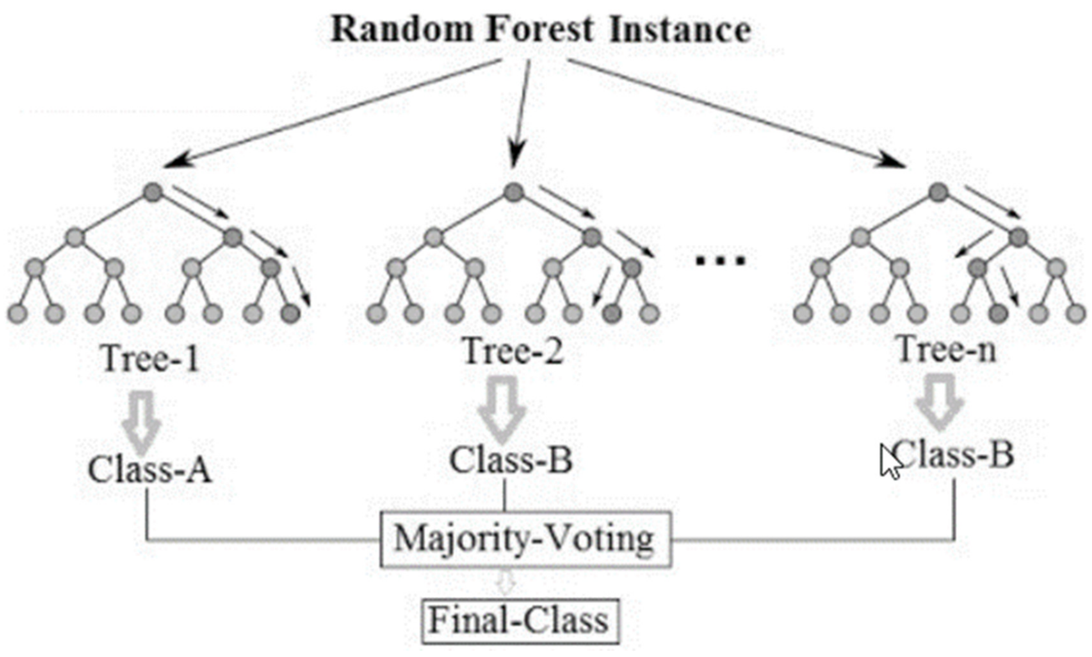


Figure 9. Example of majority-voting on the final class in Random Forest.

To prevent RF trees from being correlated, they are trained on diverse subsets of features and samples. According to Dietterich (2000), the most popular techniques for constructing RFs are bagging (Breiman, 1996) and boosting (Freund, & Schapire, 1996). Bagging uses bootstrapping (Tibshirani, & Efron, 1993), in which the DTs are trained on equally sized subsets of the original dataset, created by random sampling with replacement. In boosting, the DTs are trained one by one on the whole dataset, adjusting samples weight and gradually correcting previous step DT to reduce the total error. Two popular boosting algorithms are adaptive boosting (AdaBoost) (Freund, Schapire, & Abe, 1999) and gradient boosting (Friedman, 2001).

Below is a list of advantages and disadvantages of the RF, based on the works of Ali, Khan, Ahmad and Maqsood (2012), and Prajwala (2015):

Advantages:

- Easy to use, since it often produces good prediction results without tuning.

- Quickly trained.
- Can handle binary, categorical and numerical features without scaling.
- Bagging can be parallelized since each DT can be built independently.
- Bagging reduces overfitting by reducing the variance while retaining the bias.
- Performs implicit feature selection and can serve as an informativeness indicator.

Disadvantages:

- Often has large number of trees, which results in large memory footprint.
- Large forests can significantly slow down prediction and prevent real-time usage.
- Is not descriptive enough and its results are quite difficult to interpret.
- Favors categorical features with large number of values.
- Favors smaller groups of correlated features.

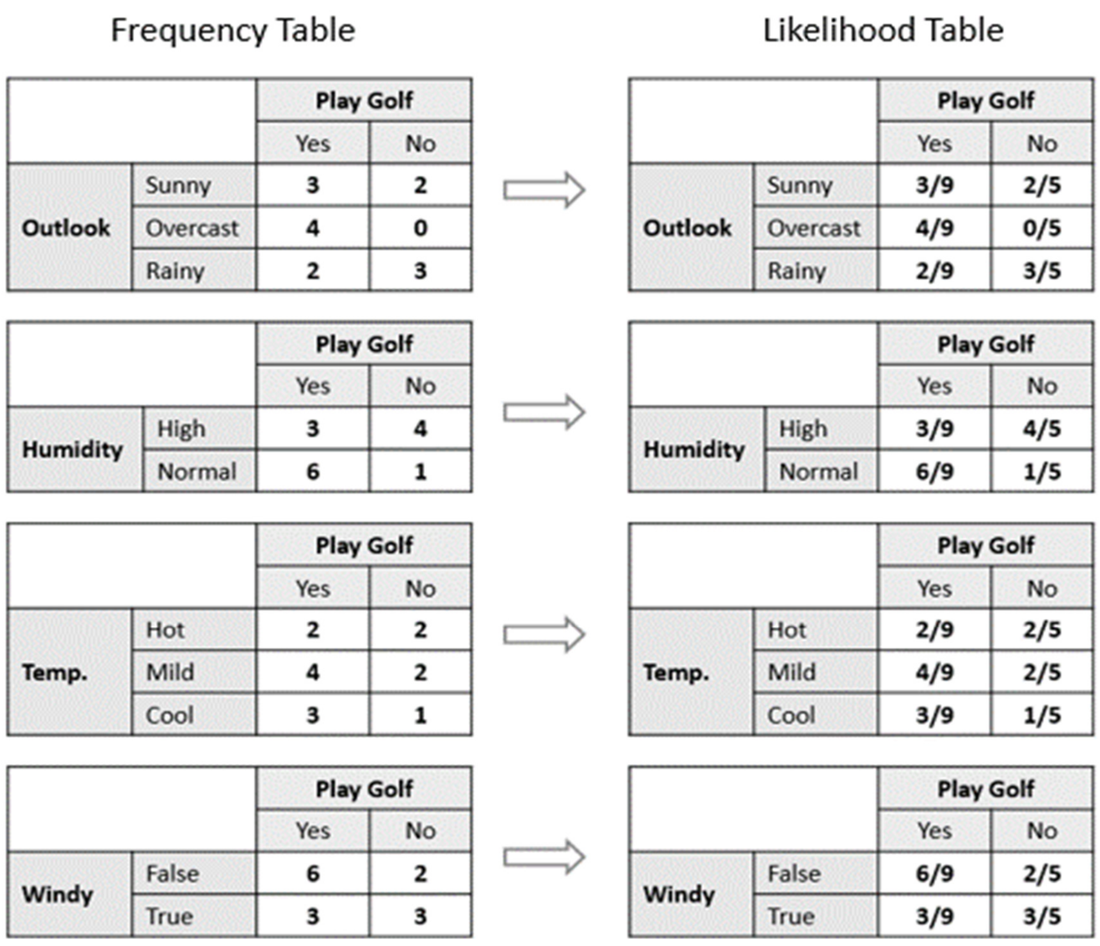
Based on the work of Probst, Wright and Boulesteix (2018), the following hyperparameters allow to achieve the optimal bias-variance trade-off for the RF:

- Size of randomly picked candidate features subset, denoted by *mtry*.
- Size of training samples subset.
- Replacement with sampling flag.
- Maximal size of internal node, denoted as *nodesize*.
- Minimal size of leaf node, denoted as *leafsize*.
- Number of trees in a single forest.
- Informativeness criteria.

Experiment 3: Classification via Naive Bayes Classifier

Naive Bayes classifier (Russell, & Norvig, 1995) is another feature-based supervised learning algorithm. It was originally intended to be used for classification tasks, but with some modifications it can be used for regression as well (Frank, Trigg, Holmes, & Witten, 2000). NBC is rather simple and is often used as a baseline for comparison to other models. However, if properly trained, it can perform well on most classification tasks, and is often significantly more accurate than more sophisticated methods (Frank et al., 2000; Ashari, Paryudi, & Tjoa, 2013). This technique is based on two theoretical aspects: Bayes' theorem (Bayes, Price, & Canton, 1763) and an assumption that all the features of the given dataset are of completely independent of one another. In probability terms, this assumption can be formulated as following: each feature has an independent contribution to the probability that an arbitrary sample belongs to a particular class, regardless of any correlations between the features.

Apparently, features are often dependent on one another, which makes the above assumption naive (hence the name of the classifier) and the probability estimations inaccurate. However, despite its simplicity, NBC often manages to achieve relatively



Frequency Table

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	3/9	2/5
	Overcast	4/9	0/5
	Rainy	2/9	3/5

		Play Golf	
		Yes	No
Humidity	High	3/9	4/5
	Normal	6/9	1/5

		Play Golf	
		Yes	No
Temp.	Hot	2/9	2/5
	Mild	4/9	2/5
	Cool	3/9	1/5

		Play Golf	
		Yes	No
Windy	False	6/9	2/5
	True	3/9	3/5

Figure 10. The process of creation of frequency and likelihood tables, demonstrated on the Golf Dataset.

high accuracy by using the loss function (Wald, 1949), such as zero-one loss (Friedman, 1997). This function defines the error as the number of wrong predictions and results in assigning the maximum probability to the correct class (Domingos, & Pazzani, 1997). The computations of NBC rely in two types of mathematical objects: frequency- and likelihood tables. For each feature, these tables convert the frequencies of the classes to posterior probabilities, which are then compared predict the outcome class. Figure 10 (Gerard, 2017) shows the above process on the Golf Dataset.

Below is a list of pros and cons of the NBC, based on the work of Al-Aidaroos, Bakar and Othman (2010), that help to estimate its applicability for a concrete dataset.

Advantages:

- Easy to understand, implement and update.
- Computationally efficient for both binary and multiclass classification.
- Can be trained even on a very small dataset if its features are independent.
- Naturally robust to missing, noisy and irrelevant features.
- When predicting, provides not only the class of a sample, but also the probability.

Disadvantages:

- Works well on numerical features, only if the latter have a concrete distribution.
- Is sensitive to redundant or correlated features.
- Is susceptible to the so called zero-frequency problem (Wu, Cai, & Zhu ,2013).
- In case of dependent features, the computed probabilities cannot be trusted.
- Does not work well with imbalanced datasets.

The basic NBC requires no hyperparameters. However, other variants use various techniques to reduce the “naivety”. For example, Laplace smoothing (Manning, Raghavan, & Schutze, 2008) modifies the probabilities in the zero-probability problem. These variants of NBC make it more robust. Below are the popular ones:

- Gaussian naive Bayes (John, & Langley, 1995).
- Multinomial naive Bayes (Manning, Raghavan, & Schutze, 2008).
- Bernoulli naive Bayes (Manning, Raghavan, & Schutze, 2008).
- Complement naive Bayes (Rennie, Shih, Teevan, & Karger, 2003).
- Scaled naive Bayes (Martinez, Webb, Chen, & Zaidi, 2016).

A more thorough list of naive Bayes variants can be found in the work of Al-Aidaroos, Bakar and Othman (2010) that shows 18 improved algorithms for various applications.

Experiment 4: Classification via Shallow Neural Network

Artificial neural network (ANN), or simply neural network, is a graph-like computational model that can do both classification and regression. Thanks to excellent results, this model has raised a considerable interest in both research and industry and is successfully used in various domains (Zhang, Patuwo, & Hu, 1998). The basic computational units of a neural network are the neurons. A neuron receives its inputs either from external sources or from other neurons and computes an output. Each input has an associated weight that represents the relative importance of current input in relation to other inputs.

To produce the output, the neuron applies a special non-linear activation function, e.g. sigmoid or hyperbolic tangent (Gomes, Ludermir, & Lima, 2011), on the weighted sum of its inputs and adds the relevant bias. The activation function performs a fixed mathematical operation that determines which neurons to consider when making the prediction. The bias provides every neuron with a trainable constant value that slightly decreases or increases the result of the activation function. Figure 11 (Fonseca, Navaresse, & Moynihan, 2003) shows the information processing in a single neuron:

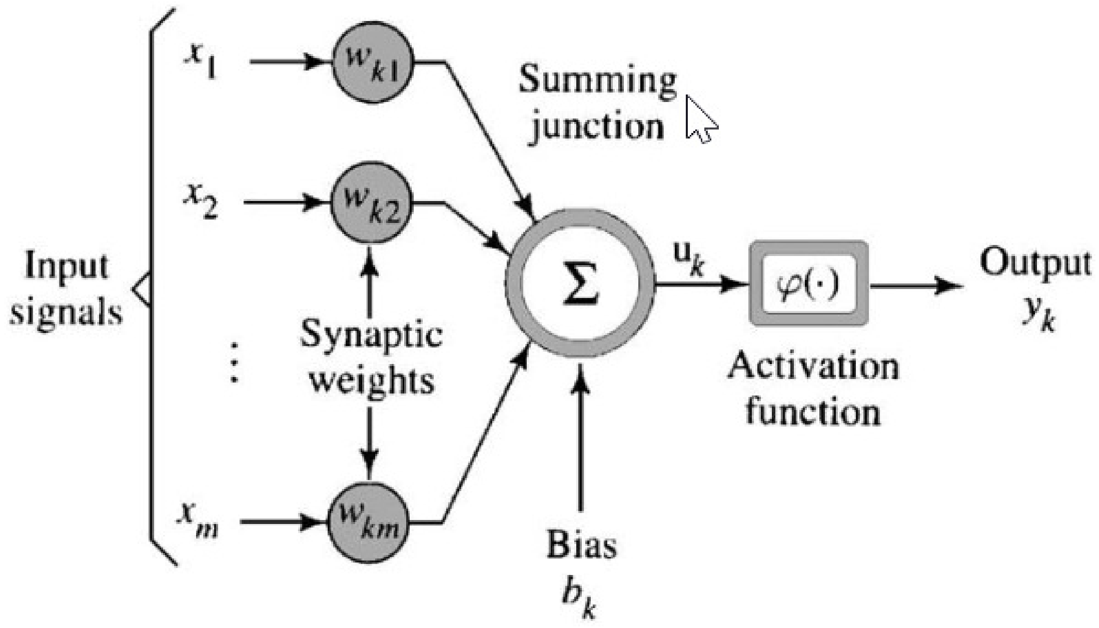


Figure 11. Information processing in a single neuron.

ANNs have three types of layers: input, hidden and outer. The (single) input layer contains neurons that get their input from the features and pass it on to the (first) hidden layer. Neurons of the (possibly multiple) hidden layer(s) perform computations and transfer the results to the output layer. The (single) output layer neurons do additional computations and produce the prediction result. SNNs have a single hidden layer.

The weights and the biases are initialized either randomly, or by a special technique, e.g. the one suggested by Yam and Chow (2000) or those proposed by Fernandez-Redondo and Hernandez-Espinosa (2001). The weights are iteratively adjusted to near-optimal values by two optimization methods: forward propagation and backpropagation (Werbos, 1982). Backpropagation's main optimization algorithms are: batch gradient descent, mini-batch gradient descent and stochastic gradient descent (SGD) (Ruder, 2016). After each full traversal of adjustments, the loss function computes a penalty score for wrong predictions on the training set, to see if it dropped below a predefined threshold. When found, the final values of weights and biases are used for prediction on new samples. Figure 12 (Tu, 1996) shows an example of SNN trained to predict the probability of patients' death based on their age and sex:

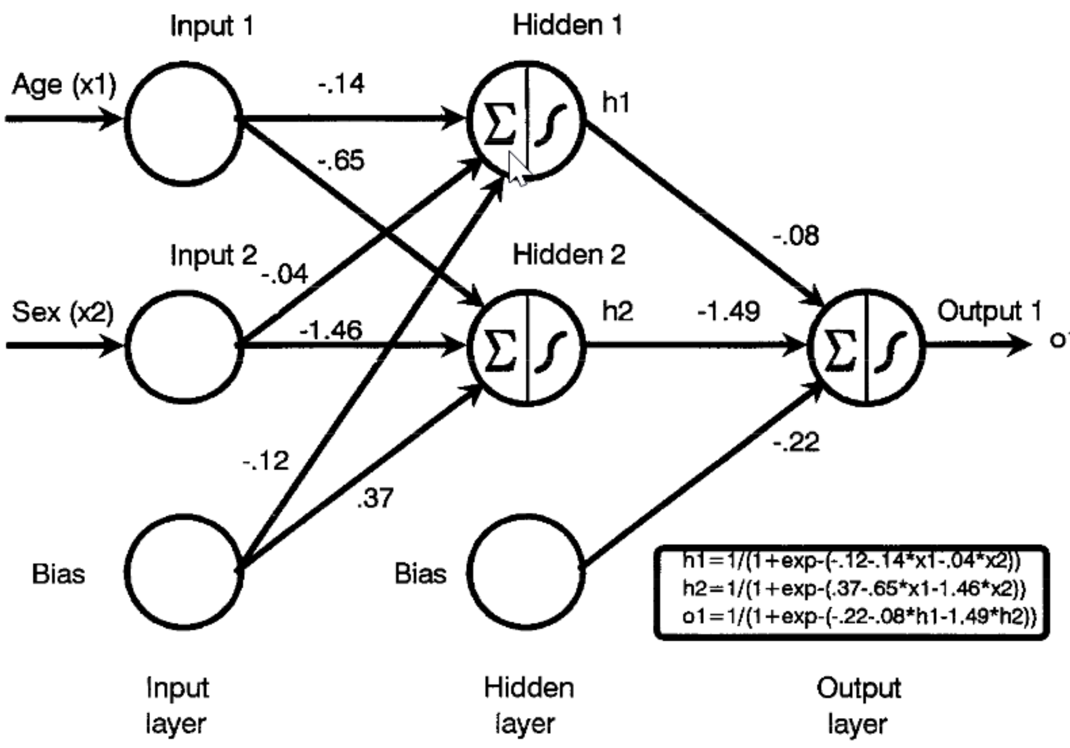


Figure 12. Architecture of a typical shallow neural network.

Below are the benefits and limitations of the SNNs, based on the work of Tu (1996):

Advantages:

1. Detect complex non-linear relationships between target variable and features.
2. Solve classification and regression problems equally good.
3. Operate well on numerous types of data relatively well without fine-tuning.
4. Handle various dependencies between the features.
5. Capable of modeling nonlinear data with numerous features, e.g. images.
6. Prediction is very fast, especially in shallow ANNs.

Disadvantages:

1. Internal relationships between the target variable and the features are indistinct.
2. Needs large train set to achieve high accuracy and is computationally expensive.

3. Prone to overfitting.

4. Can forget dependencies that they learned from old data after learning new one.

5. Are practically limited due to a single hidden layer.

SNNs have several hyperparameters that determine its structure, affect the training speed etc. Below is a list of the important ones based on the work of Bengio (2012):

1. Number of hidden neurons in the hidden layer.

2. Activation function.

3. Learning rate in backpropagation, denoted by *alpha*.

4. Optimization algorithm.

5. Maximal number of weights adjustment iterations.

6. Weights initialization method.

Experiment 5: Classification via LSTM

Long short-term memory networks (Hochreiter, & Schmidhuber, 1997) are a special kind of recurrent neural networks. All RNNs, including the LSTM, consist of units. These are ANNs with multiple hidden layers that contain cycles from subsequent neurons to preceding ones. These cycles create a special recurrent layer within the network, called hidden state, which acts like a memory and allows the RNN unit to handle time sequenced data. Since the unit is recurrent, it handles the input by timesteps. At each timestep t, the unit receives x_t, which is an input sequence element (vector) at position (time) t, and h_{t-1}, which is the hidden state vector from previous timestep and the output of the unit at time $t-1$. Figure 13 shows a single RNN unit:

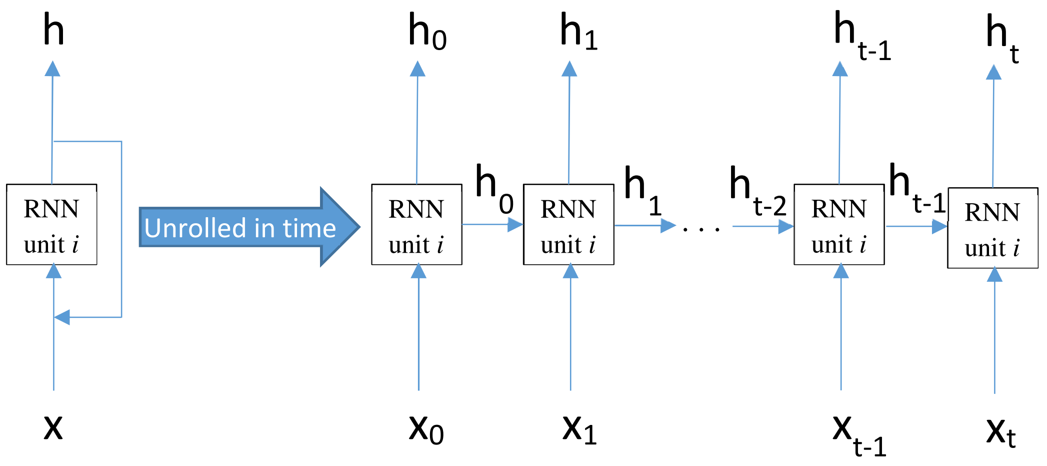


Figure 13. RNN unit - unfolding of events over time.

Units of ordinary RNNs, such as Elman (Elman, 1990) and Jordan (Jordan, 1997) networks, have simple architecture. Their update process is simple as well. When processing the timed input sequence, forward propagation iteratively updates the input and output vectors, timestep by timestep, starting with timestep $t=1$ and ending by $t=\tau$ (last element index). Figure 14 (Greff, Srivastava, Koutnik, Steunebrink, & Schmidhuber, 2017) shows a typical RNN unit:

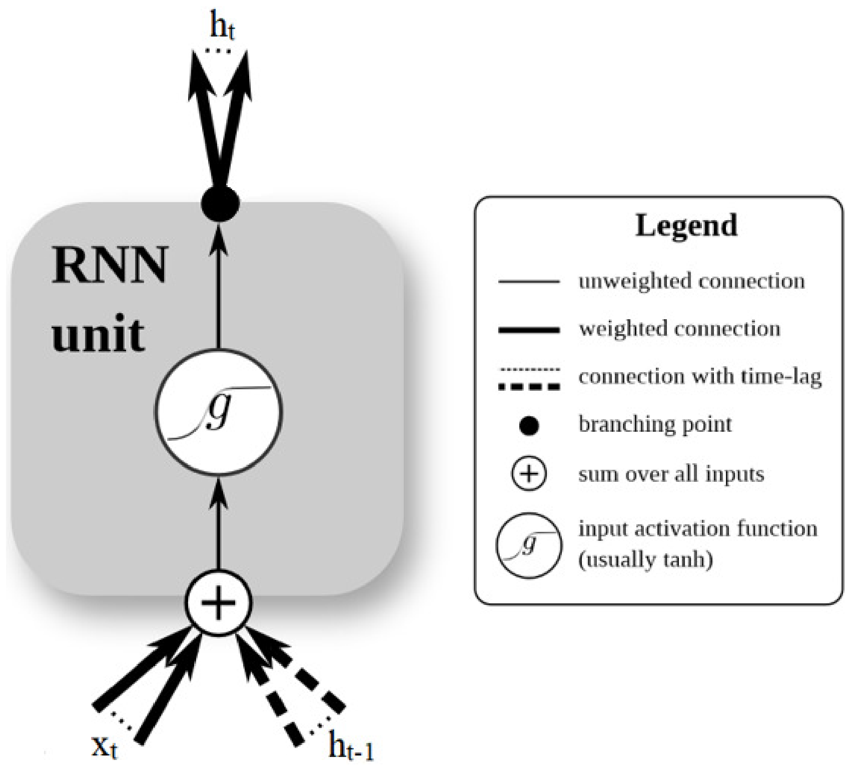


Figure 14. Architecture of a single ordinary RNN unit.

In each iteration, the RNN cell considers the current input and the previous hidden state and produces the current hidden state - the predicted output of the unit. The decision is based on weight matrices of current input and previous output, and a bias, which are optimized during the backpropagation to make the right decisions. A hyperbolic tangent activation function is applied on the summation result to get the hidden state squashed into a range of (-1,1). If the output sequence need to be transformed into a more convenient form, then the output function y_t can be computed from current hidden state by using another layer, called dense layer, with weights, bias and a final non-linear activation function, such as softmax (Bishop, 2006).

In practice, each RNN unit handles a single input value from the input features vector. Therefore, RNN networks often consist of RNN cell that contains several stacked RNN units, corresponding to the length of the vector. In addition to being deep in time, RNNs can be made deep in space by stacking by multiple recurrent hidden layers (each having multiple units) on top of each other (Graves, Mohamed, & Hinton, 2013). A stacked RNN architecture usually ends with a dense layer.

Deep RNNs can recognize significantly more complex patterns in the data than ordinary neural networks (Bianchini, & Scarselli, 2014), and even learn time dependencies in the data. However, their memory cells do not have enough level of control over the memorization process to precisely regulate which part of the information is unimportant and should be thrown away, and which is important enough to keep for handling data of subsequent timesteps (Salehinejad et al., 2017). In addition, when time-series with long time lag patterns, ordinary RNN units have practical difficulties due to particularities of the backpropagation process (Bengio, Simard, & Frasconi, 1994). RNN units rely on BPTT – a special variation of backpropagation that considers recurrent layers, which is known for being ineffective for time-series because of two problems: One, called exploding gradient, (Bengio, Simard, & Frasconi, 1994) can be easily solved by gradient clipping (Mikolov, 2012). The other one, vanishing gradient, is more serious and prevents from ordinary RNNs to become inappropriate for tasks which involve learning long-term dependencies.

LSTM solves the vanishing gradients problem by using a combination of real time recurrent learning (RTRL) (Williams, & Zipser, 1989) and a full BPTT (Robinson, & Fallside, 1987; Werbos, 1988; Williams, 1989; Williams, & Peng, 1990, Graves, & Schmidhuber, 2005). In addition to full BPTT, LSTM has a significantly more complex unit, having four recurrent hidden layers instead of one and special constructs that interact in a special way to control the memorization process to a far greater extent that RNN’s unit.

The original LSTM design by Hochreiter and Schmidhuber (1997) had several flows and was significantly improved by various researchers (Gers, Schmidhuber, & Cummins, 1999, Gers, & Schmidhuber, 2000; Gers, & Schmidhuber, 2001; Gers,

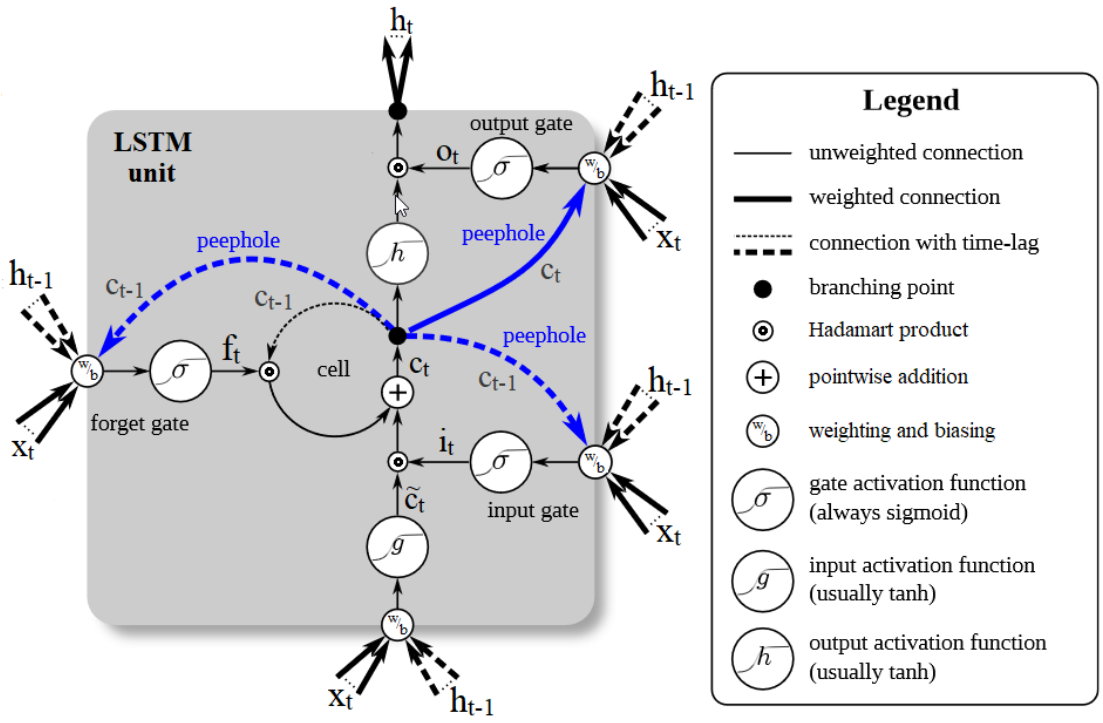


Figure 15. Architecture of a single unit of vanilla LSTM.

Schraudolph, & Schmidhuber, 2002, Graves, & Schmidhuber, 2005), resulting in a so called vanilla LSTM that is to date the most commonly used in the literature (Greff et al., 2017). This study refers to the vanilla LSTM as baseline architecture. Figure 15 (Greff et al., 2017) shows the schematics of a vanilla LSTM unit's architecture. Like RNN units, LSTM units can be stacked in a single LSTM layer, whereas LSTM layers can also be stacked.

The update process in LSTM unit much resembles the one that takes place in RNN. Vanilla LSTM unit contains several components. First is the memory cell, also called Constant Error Carousel (CEC), which is capable of memorizing information with long time lags. The content of a memory cell is called the cell state; and another three components are gate layers: input, forget, and output. Each gate layer consists of the relevant gate itself and a following Hadamard product (or Schur product) operator (Davis, 1962). The gates are special FFNNs with sigmoid activation function that control information flow by preventing part of it to proceed any further. Each gate receives three input vectors: input vector of current timestep, hidden state vector (output vector) from previous timestep, and the cell state vector from current or previous timestamps (depends on the gate). Below are the equations, according to which the update computations are carried out:

d - length of input vectors (number of features)
h - length of hidden-state (number of units in single cell)
$x_t \in R^d$ - input vector at time *t*
$h_t \in R^h$ - hidden-sate vector at time *t*
$f_t \in R^h$ - forget gate activation vector at time *t*
$i_t \in R^h$ - input gate activation vector at time *t*
$o_t \in R^h$ - output gate activation vector at time *t*
$c_t \in R^h$ - state vector at time *t*
$\tilde{c}_t \in R^h$ - potential changes in cell state vector at time *t*

$U_f, U_i, U_o, U_c \in R^{d \times h}$ - input weight matrices of forget gate, input gate, output gate and cell state, respectively

$W_f, W_i, W_o, W_c \in R^{h \times h}$ - output weight matrices of forget gate, input gate, output gate and cell state, respectively

$D_f, D_i, D_o \in R^{h \times h}$ - peephole diagonal weight matrices of forget gate, input gate and output gate, respectively

$b_f \in R^h$ - bias of forget gate

$b_i \in R^h$ - bias of input gate

$b_o \in R^h$ - bias of output gate

$b_o \in R^h$ - bias of cell state gate

σ - sigmoid activation function

tanh - hyperbolic tangent activation function

$\circ$ - Hadamart product operator

$$
\begin{aligned}
f_t &= \sigma\left(U_f x_t + W_f h_{t-1} + D_f c_{t-1} + b_f\right) \\
i_t &= \sigma\left(U_i x_t + W_i h_{t-1} + D_i c_{t-1} + b_i\right) \\
\tilde{c}_t &= \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
o_t &= \sigma\left(U_o x_t + W_o h_{t-1} + D_o c_t + b_o\right) \\
h_t &= o_t \circ \tanh\left(c_t\right)
\end{aligned}
$$

All the input vectors of a gate are weighted. For input and hidden-state vectors ordinary weight matrices are used, whereas for the hidden-state vector a diagonal weight matrix is used (Graves, 2013). The gate then adds a (single) bias and applies a sigmoid activation function. When backpropagation takes place, the weights and biases of the gates are optimized to let the gate make correct decisions on required cell state changes. The components of the vector produced by the gate are numbers between 0 and 1, and the gate vector's length corresponds to the length of the cell state vector. These numbers determine which part of the information arriving from the input vectors is let through to the cell state vector. Zero value means "let nothing through," while a value of one means "let everything through". Next, the gate and the cell state vectors both go into the Hadamard product operator that does pointwise multiplication and applies the relevant changes to the cell state vector.

Three additional components of the unit are peephole connections that pass weighted cell state vectors as input for the gates, to let them consider the contents of the memory cell when making their decisions. Finally, the last component is an input layer. This is another FFNN that operates in almost the same way as a gate. It receives weighted input vector of the current timestep and the hidden-state vector of the previous timestep, applies an activation function, and creates a vector of potential changes in the cell state vector. Altogether, the components of a unit act as a single mechanism that operates in several steps.

At the first step, the forget gate layer decides what unimportant information should be removed from the cell state. The layer's gate considers current input vector, previous hidden state vector, and the previous cell state vector (all weighted and biased) and produces a vector with information that should be removed from the cell state. These changes are then applied on previous cell state vector, to get a vector with forget updates for the cell state vector.

At the second step, the LSTM decides what new information is important enough to be stored in the cell state. This step requires a common effort of two layers: input gate layer and input layer. The input gate considers current input vector, previous hidden state vector and previous cell state vector, and decides which components of the cell state vector are updated. The input layer considers the current input vector and previous hidden-state vector, and computes a candidate state vector, which holds the new information that is added to the existing cell state vector. The result vectors of the two layers are combined into a single vector, which contains the store updates for the cell state vector. Finally, the update vector is pointwise added to the previous cell state vector, which frees it from irrelevant information and fills it with new data. This completes the update operations and produces the current cell state vector, which is ready to be used in the next recurrence iteration.

The last step is for the LSTM to decide what part of the current cell state (which at this point is fully computed) is going to become the current hidden state, i.e. the output of the unit. This step takes place in three stages. In the first stage, the output gate layer considers the current input, the previous hidden state and the current cell state, and produces a vector that decides which components of the current cell state enters the hidden state. In the second stage, the current cell state vector is replaced by a hyperbolic tangent activation function. This pushes its components to between -1 and 1 and prepares it to be combined with the output gate result vector. In the third stage the two vectors are combined. This produces the current hidden-state vector, which, as stated above, is also the vector the unit provides the outside world.

As opposed to numerous traditional ML models, LSTLs analyze the fluctuations of the time-series over time and not just the statistical multi-dimensional distribution of its values. They determine whether the new curvatures in the data significantly differ from those previously seen. Figure 16 (Kim, & Cho, 2018) illustrates this idea on a curve of 1D web traffic time-series data:

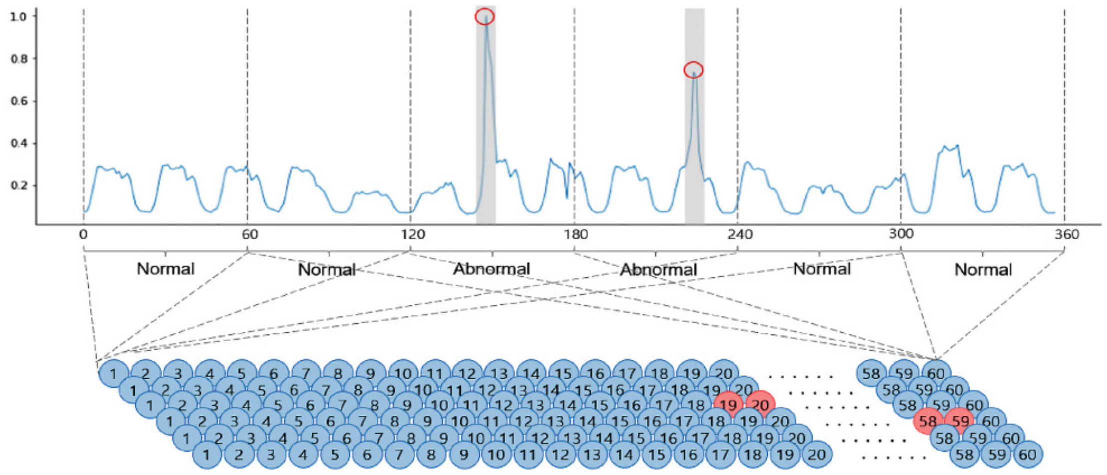


Figure 16. Anomaly detection in one-dimensional web traffic time-series data.

The two areas marked as abnormal contain peaks that are superfluously high as compared with the rest of the knolls. This quality is particularly relevant to sensor data since the it is essentially a time-series and suggests that LSTM has a very high chance of being the most appropriate model for such analysis.

A list of advantages and disadvantages of the LSTM, based on the work of (Hochreiter, & Schmidhuber, 1997; Otte, Liwicki, & Krechel, 2014) appears below:

Advantages:

1. Can store information over huge number of steps.
2. Can handle noise, distributed representations, and continuous values even in sequences with long time lags.
3. Can process time-series with varying length straightforwardly.
4. Can handle extremely long sequences.
5. Can distinguish two subsequences by two widely separated elements.
6. Can recognize a pattern even if is separated by a wide time lag from last occurrence.
7. Works well with relatively little fine tuning.
8. Has constant update complexity per weight and timestep.
9. Can be parallelized and speed up by using multiple CPUs and GPUs.

Disadvantages:

1. Is significantly less effective on small amounts of data.
2. Rather long training time due to large amount of required training data.
3. Cannot solve some theoretic problems, e.g. the strongly delayed XOR problem.
4. Have relatively large number of weights, which causes large memory footprint.
5. Unable to precisely count discrete time steps.

Based on the works of Reimers and Gurevych (2017), Merity, Keskar and Socher, (2017), and Bengio (2012), the LSTM can be tuned by changing the following:

1. The size of a hidden state, i.e. the number of LSTM units that comprise it.
2. Number of LSTM layers in stacked LSTM model.
3. Optimization algorithm.
4. Algorithm for an exploding gradient.
5. Method and rate of the variational dropout regularization technique.
6. Size of the mini-batch descent.
7. Number of training epochs.
8. LSTM architectures, such LSTM with embedding layer, BLSTM etc.

Resources

For this research, the following basic and available resources were required:

- A laptop or desktop computer with high-speed access to the internet.
- An extremely powerful computational platform on Amazon Web Services.
- Scientific computing and programming software, as following:
  - The Python programming language (Van Rossum, & Drake, 2018).
  - Integrated development environments (IDE) for Python, such as PyCharm (2018), and Jupyter Notebook (2014).
  - Frameworks for Python, such as NumPy (2018), matplotlib (Hunter, 2007), SciPy (Jones, Oliphant, & Peterson, 2014), Pandas (2018), TensorFlow (Abadi et al., 2016), Keras (Chollet, 2015), Scikit-learn (Pedregosa et al., 2011), etc.
  - Software architecture tools, such as Enterprise Architect (2018).
- Software for documentation, data management and presentations, such as Microsoft Office (2018).

- Additional software, in which the need may arise in the future.

Summary

This chapter began by the providing a detailed design of the planned experiment, according to the following major components: choosing a dataset, preprocessing the data, choosing the classifiers, tuning the classifiers' hyperparameters, estimating classification accuracy, reasoning on LSTM's advantages, and generalizing and summarizing the experiment results. Then the chapter continued by focusing on the dataset. An analysis and justification of the chosen dataset was followed by elaboration on cleaning and preprocessing procedures, such as transforming samples into feature vectors and into time-series, which needed to be carried out before the data could be analyzed by ML models. Finally, the last and largest portion of the chapter contained an in-depth review and analysis of the five ML models, namely: SVM, RF, NBC, SNN, and LSTM, that were chosen to analyze the sensor data and detect anomalies within it. The focus was on LSTM since this approach is the heart of this dissertation.

Chapter 4 - Results

The five experiments designed to be carried out in Chapter 3 were conducted. The collected data was processed and prepared for:

1. Estimating the accuracy and effectiveness of LSTM detection of anomalies in normal data, and classifying the anomaly type.
2. Estimating the degree to which traditional ML models, which are based on feature engineering, can be replaced by a deep learning (DL) LSTM model, such as an LSTM that operates on raw data.

The examined models, both traditional ones and the LSTM, have numerous variations that can be built by changing their architecture and hyperparameters. In this study, four variations have been chosen to represent each model and show the effect of its most important tunable elements on the prediction accuracy. Each variation was trained with 10-fold cross-validation on the training/validation set, then tested on the test set. For each model, the testing results were shown as four histograms reflecting its evaluation according to the nine metrics of accuracy described in Chapter 2: micro-precision, macro-precision, weighted precision, micro-recall, macro-recall, weighted-recall, micro-F1 score, macro-F1 score, and weighted F1-score. Each accuracy score varied from 0 (lowest) to 1 (highest). In this study, accuracy score above 0.95 (corresponding to 95%) is considered high.

The complete source code of the experimental part of this dissertation is available in the supplementary archive. Appendix A contains screenshots with a full specification of the examined architectures and hyperparameters, as well as detailed evaluation results.

Results of Dataset Processing

The first objective of the experimental part was to process the dataset, create normal and anomalous sequences and ensure that the sequences of each type are significantly distinctive from the other types. This was necessary so the traditional ML algorithms can learn the patterns and achieve high accuracy scores. This objective was achieved. All ML classifiers showed very high accuracy scores on all the chosen nine metrics. Following the experiment’s design, the original dataset was compiled into a single file and transformed into normal sequences, each consisting of ordered patient measurements per day. These sequences were then used to create anomalous sequences by splicing anomalous elements of six types, resulting in seven types of data.

Figure 17 shows a small subset of randomly picked sequences (one for each type):

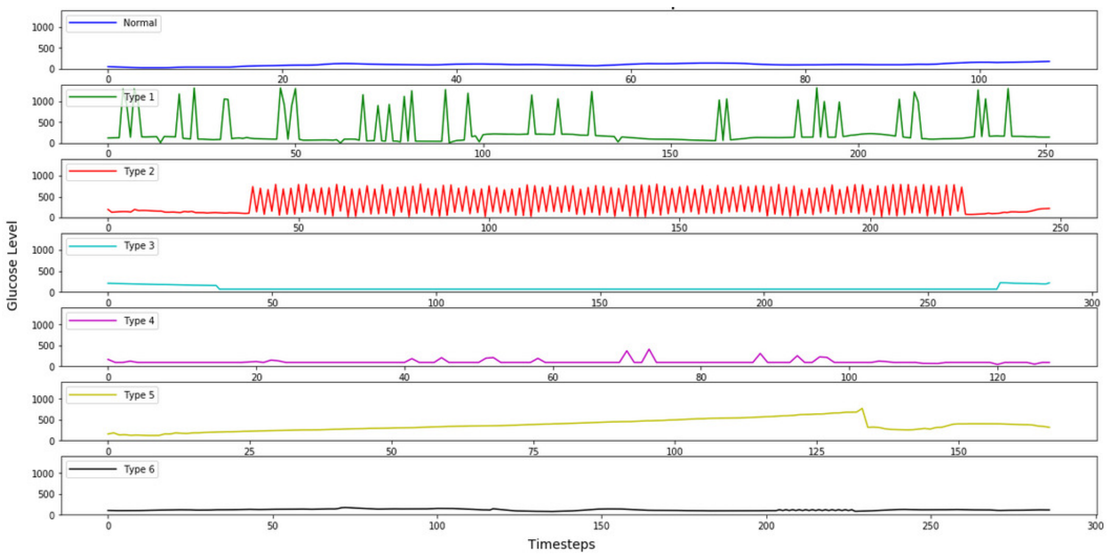
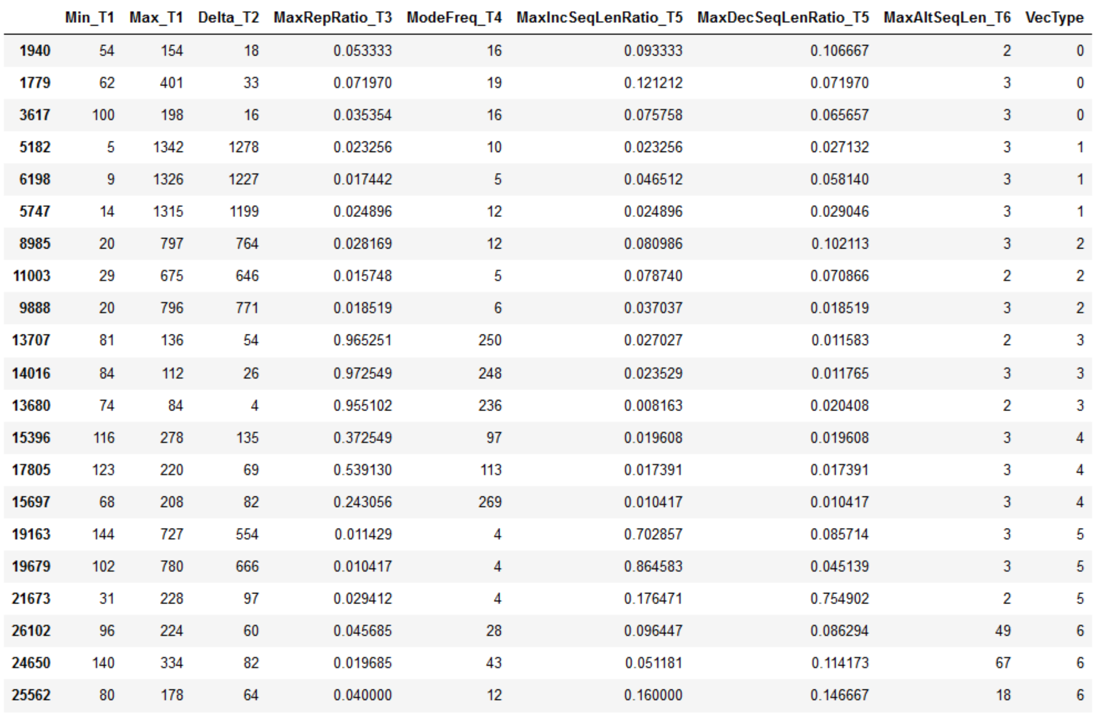


Figure 17. Discrete sequences of measured glucose levels, represented as continuous curves.

Each sequence consists of discrete consecutive values of blood glucose level measurements and is represented as a curve. It can be clearly seen from the diagram that each anomaly type distorts the normal curve in its own way. These sequences of raw data were ready to be analyzed by the LSTM.

For traditional ML models, that could not classify these sequences in their original form, feature vectors of fixed length were prepared according to Table 3, one for each sequence. By using a small amount of wisely constructed features, it was possible to clearly reveal the differences between the normal data and each type of anomaly. Table 4 shows random feature vectors with matching labels:

Table 4. Random feature vectors, constructed from normal and anomalous



	Min_T1	Max_T1	Delta_T2	MaxRepRatio_T3	ModeFreq_T4	MaxIncSeqLenRatio_T5	MaxDecSeqLenRatio_T5	MaxAltSeqLen_T6	VecType
1940	54	154	18	0.053333	16	0.093333	0.106667	2	0
1779	62	401	33	0.071970	19	0.121212	0.071970	3	0
3617	100	198	16	0.035354	16	0.075758	0.065657	3	0
5182	5	1342	1278	0.023256	10	0.023256	0.027132	3	1
6198	9	1326	1227	0.017442	5	0.046512	0.058140	3	1
5747	14	1315	1199	0.024896	12	0.024896	0.029046	3	1
8985	20	797	764	0.028169	12	0.080986	0.102113	3	2
11003	29	675	646	0.015748	5	0.078740	0.070866	2	2
9888	20	796	771	0.018519	6	0.037037	0.018519	3	2
13707	81	136	54	0.965251	250	0.027027	0.011583	2	3
14016	84	112	26	0.972549	248	0.023529	0.011765	3	3
13680	74	84	4	0.955102	236	0.008163	0.020408	2	3
15396	116	278	135	0.372549	97	0.019608	0.019608	3	4
17805	123	220	69	0.539130	113	0.017391	0.017391	3	4
15697	68	208	82	0.243056	269	0.010417	0.010417	3	4
19163	144	727	554	0.011429	4	0.702857	0.085714	3	5
19679	102	780	666	0.010417	4	0.864583	0.045139	3	5
21673	31	228	97	0.029412	4	0.176471	0.754902	2	5
26102	96	224	60	0.045685	28	0.096447	0.086294	49	6
24650	140	334	82	0.019685	43	0.051181	0.114173	67	6
25562	80	178	64	0.040000	12	0.160000	0.146667	18	6

In the header of the table, the index *N* in the postfix _T<*N*> of each feature represents the corresponding type of anomaly (denoted by VecType). It can be seen that vectors of a given type can be isolated from other vectors by comparing the relevant columns.

Results of Experiment 1

Experiment 1 used SVM to detect and classify anomalies. The fact that this model can be quickly constructed and trained, made it feasible to use grid search to traverse through the numerous combinations of its hyperparameters and find the best variant. Figure 18 shows the nine accuracy scores of each of the four chosen SVMs:

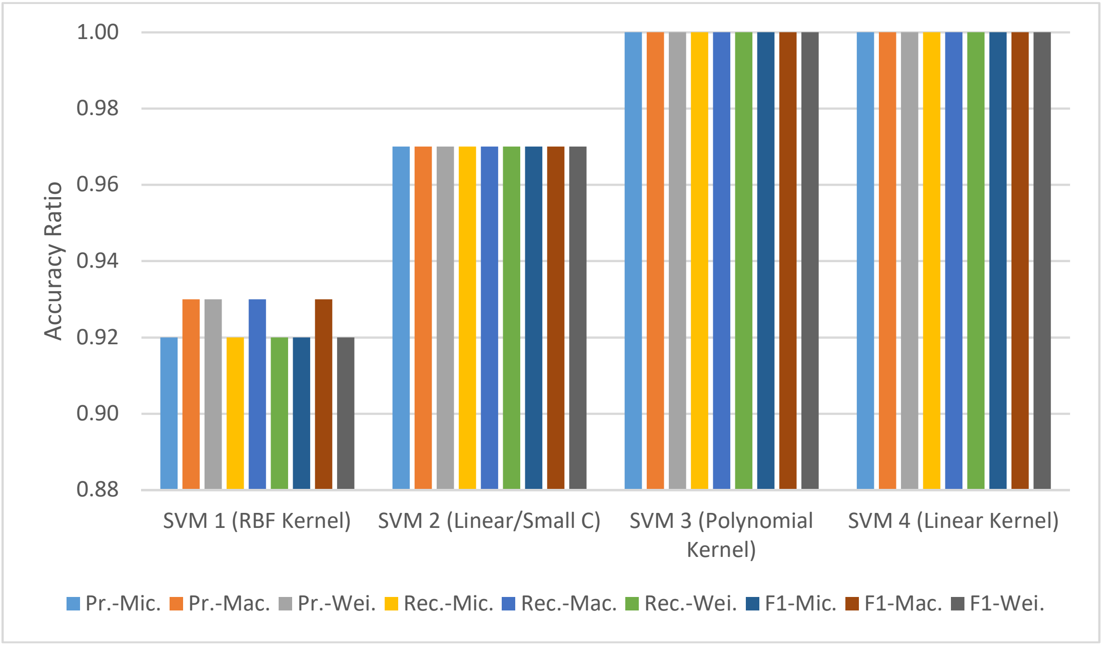
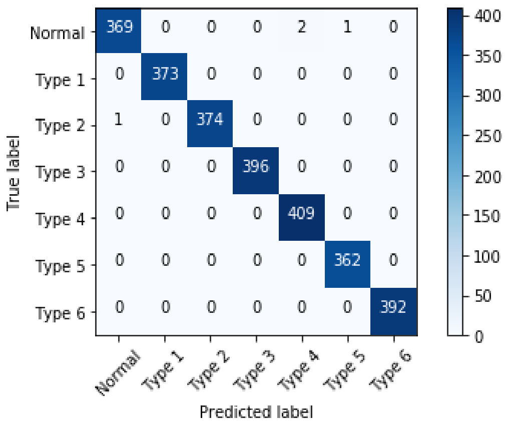


Figure 18. Histograms of four different SVM models' accuracy scores, measured by the nine metrics used in the study.

The SVM has several important hyperparameters. One is the kernel type. Figure 18 shows SVMs with RBF, polynomial, and linear kernels. Polynomial and linear kernels both provided very high results. Another central hyperparameter is *C* - the ratio of data points of the opposite class that can violate the hyperplane. Setting *C* to a very small value of 0.001 heavily penalized SVM 2 for violations and undermined accuracy.

Table 5. Confusion matrix and full classification report of the best-found SVM classifier.



True label
Predicted label
400
350
300
250
200
150
100
50
0

	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	369	0	0	0	2	1	0
Type 1	0	373	0	0	0	0	0
Type 2	1	0	374	0	0	0	0
Type 3	0	0	0	396	0	0	0
Type 4	0	0	0	0	409	0	0
Type 5	0	0	0	0	0	362	0
Type 6	0	0	0	0	0	0	392

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       1.00      1.00      1.00       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

For the chosen dataset, the optimal value (used in SVM 4) was 9. This model achieved a full 1.0 score with just 4 out of 2,679 misclassified samples. Table 5 shows its confusion matrix and a full classification report with prediction details.

The obtained results show that a rather simple linear SVM was able to find good support vectors and construct a hyperplane that correctly classified nearly all data points. Similar results were observed in the study if Verner and Butvinik (2017). However, to achieve such high accuracy, precise and tailored features had to be chosen for the model. As previously mentioned, construction of these features is a labor-intensive task and requires an in-depth domain knowledge. Therefore, this experiment inferences were as following:

- High accuracy can be achieved when the features are properly chosen.
- The LSTM model is expected to achieve accuracy of the same order, yet without the overhead of feature engineering.

Results of Experiment 2

In Experiment 2, the Random Forest model was chosen to detect and classify anomalies. The RF model has large amount of hyperparameters, which made it impractical to exhaustively search through over all the possible combinations. Therefore, several hyperparameters, for instance, the size of features subset, were manually analyzed for its effect on the prediction accuracy. Accuracy scores of the most interesting RF variants are shown in Figure 19:

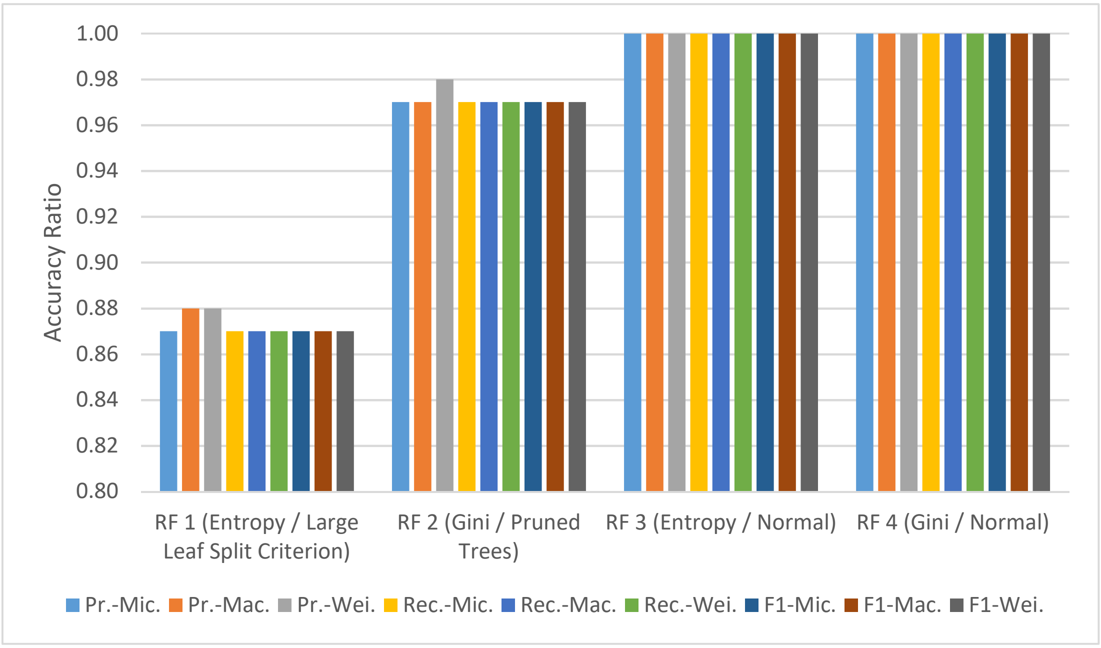


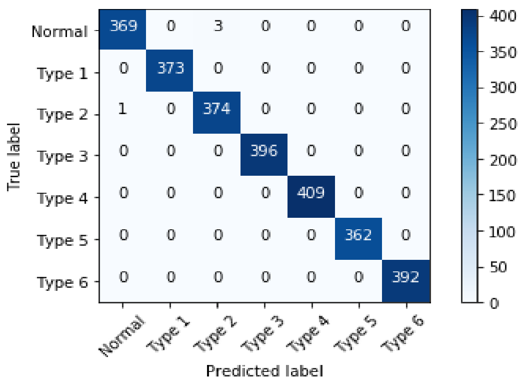
Figure 19. Histograms of four different Random Forest model's accuracy scores, measured by 9 metrics.

Several hyperparameters had an interesting effect on the accuracy. One is the maximal depth of the RF trees. According to Probst, Wright and Boulesteix (2018), pruned trees constitute a form of regularization, as they decrease the variance of the model and increase its bias. In this study, trees pruning have slightly reduced the accuracy. The negative effect could have been significantly stronger, but the small number of precise features appeared to be sufficient for the primitive trees of RF 2 to filter out the right type of the data by a few simple conditions. Another hyperparameter with similar regularization effect (Genuer, 2012) is the minimal number of samples required to be at leaf nodes. In RF 1 this value was chosen to be large, which allowed many samples to be held in a minimal leaf node and prevented the right splits to occur in the Forest's trees, even though only a small number of them was required. The result was an over simplified model with mediocre accuracy. Finally, the impurity criteria hyperparameter had practically no impact on the accuracy: both RF 3 and RF 4 had excellent results. Raileanu and Stoffel (2004) explain this by saying that that Gini and

Entropy are virtually equal by their effectiveness, differing mainly in terms computation speed.

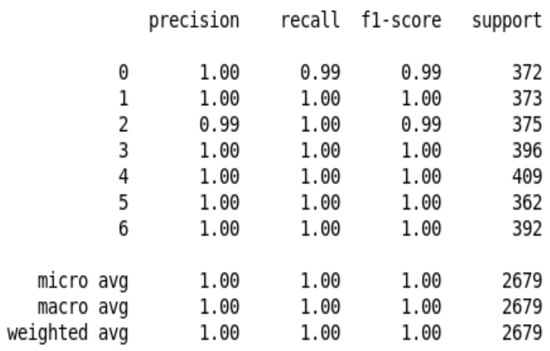
The best-found RF (see Appendix A for a full list of its hyperparameters) repeated the extremely high results of the SVM, scoring 1.0 in all metrics with 4 out of 2,679 wrong predictions. Table 6 shows the full results of this model:

Table 6. Confusion matrix and full classification report of the best-found Random Forest classifier.



True label \ Predicted label	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	369	0	3	0	0	0	0
Type 1	0	373	0	0	0	0	0
Type 2	1	0	374	0	0	0	0
Type 3	0	0	0	396	0	0	0
Type 4	0	0	0	0	409	0	0
Type 5	0	0	0	0	0	362	0
Type 6	0	0	0	0	0	0	392

400
350
300
250
200
150
100
50
0



```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       0.99      1.00      0.99       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

RF is another proof that high accuracy can be achieved with traditional ML models if the chosen features are properly chosen. However, as in the case with SVM, the model could not operate on raw data and required hand-crafted features, with all the attendant limitations. The LSTM model is awaited to achieve close accuracy, yet without being dependent on the domain knowledge factor.

Results of Experiment 3

Experiment 3 concerned the naive Bayes classifier. Variations of NBC have very few hyperparameters, for which, as the experiments have shown, the default values of Scikit-learn work the best. Therefore, instead of searching for the best combination of hyperparameters, the accuracy of several NBC models in their default configuration were compared. Figure 20 shows the accuracy scores of Bernoulli, complement, multinomial, and Gaussian NBCs:

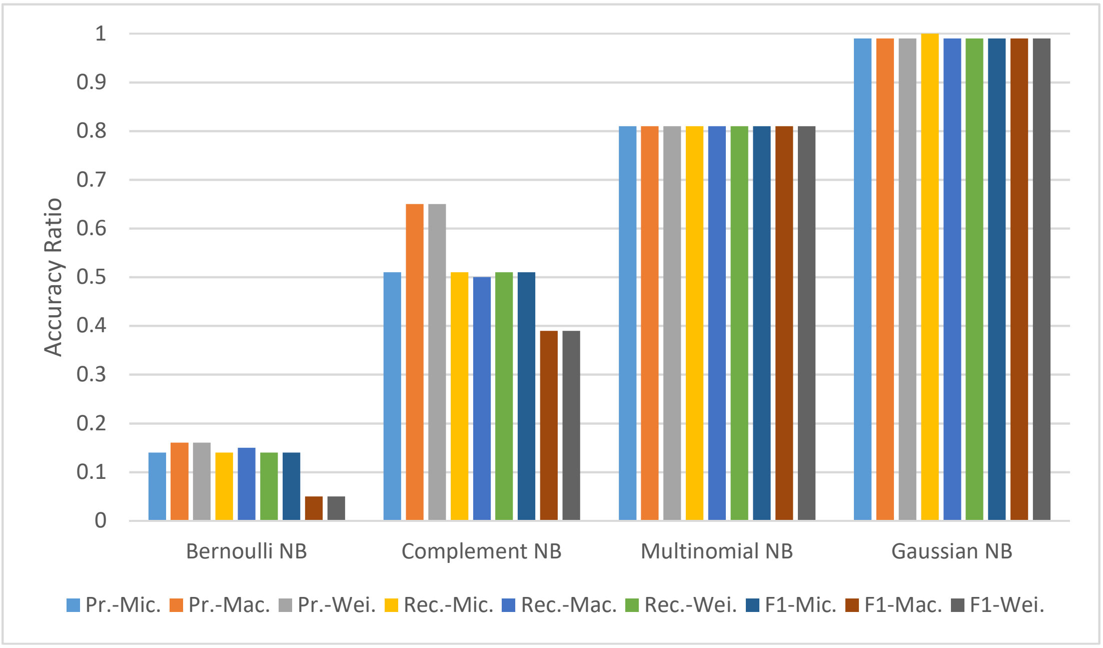


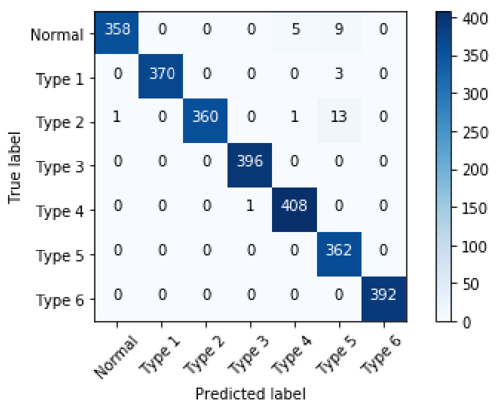
Figure 20. Histograms of four naive Bayes classifier models' accuracy scores, measured by in 9 metrics.

NBC is a classic example to demonstrate the high dependence of traditional ML models on the type of data. For instance, the Bernoulli classifier only considers the presence or absence of a feature instead of counting its number of appearances. For most types of data, including the one used in this study, this oversimplified approach results in poor accuracy. Complement NBC confers an advantage for imbalanced datasets, where the weights for classes with few training samples shrink due to under-studied bias effect (Rennie et al., 2003). However, the dataset used in this study was perfectly balanced, which deprived the Complement NBC of its benefits and resulted in below-average performance. In addition, the model appeared to be highly unstable, since its accuracy scores significantly differ from one another. For example, having an average weighted precision of 0.65, the model had weighted F1-score of only 0.39.

Multinomial NBC had another dataset issue that prevented it from achieving high results. The probabilistic computations of this model are adjusted to multinomial distribution for all the feature pairs. As a result, it works well for data that can be easily turned into counts, such as word counts in text, but suffers from lack of accuracy when the data does not fit well into multinomial distribution. In case of this study, it did not.

Table 7 shows the detailed results of the Gaussian NBC, the best-found model:

Table 7. Confusion matrix and full classification report of the best-found naive Bayes classifier.



	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	358	0	0	0	5	9	0
Type 1	0	370	0	0	0	3	0
Type 2	1	0	360	0	1	13	0
Type 3	0	0	0	396	0	0	0
Type 4	0	0	0	1	408	0	0
Type 5	0	0	0	0	0	362	0
Type 6	0	0	0	0	0	0	392

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       372
           1       1.00      0.99      1.00       373
           2       1.00      0.96      0.98       375
           3       1.00      1.00      1.00       396
           4       0.99      1.00      0.99       409
           5       0.94      1.00      0.97       362
           6       1.00      1.00      1.00       392

   micro avg       0.99      0.99      0.99      2679
   macro avg       0.99      0.99      0.99      2679
weighted avg       0.99      0.99      0.99      2679
```

This NBC variant achieved the best accuracy, scoring 0.99 in all metrics with 33 out of 2,679 misclassified samples. However, the true reason behind such high accuracy hides in the expressiveness of the constructed features. The chosen features had very low-entropy distribution, which, according to Rish (2001), yields high accuracy for the naive Bayes. In fact, if less informative features had been used, the results would have been significantly less impressive. Being able to automatically extract features, the LSTM is anticipated to overcome this limitation.

Results of Experiment 4

Experiment 4 addresses shallow neural networks. SNNs can be tuned by changing both their architecture and their hyperparameters, which makes the number of combinations very large. However, effective parallelized training of this model in

Scikit-learn made it possible to find the best accuracy by means of grid search. Four particularly interesting variations of SNN are presented in Figure 21:

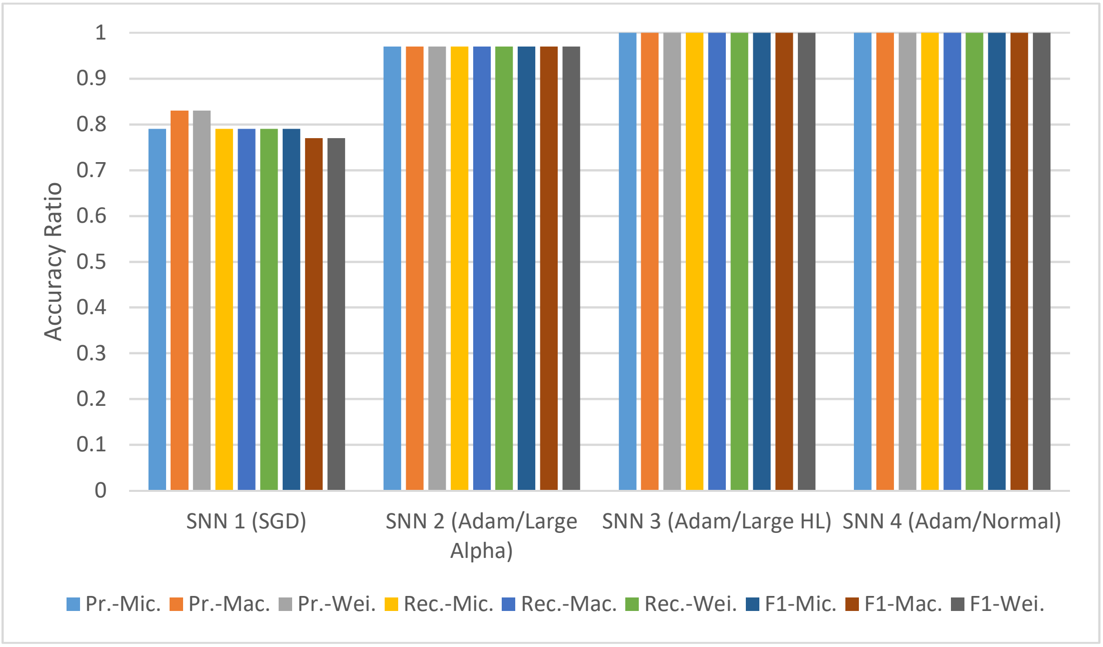


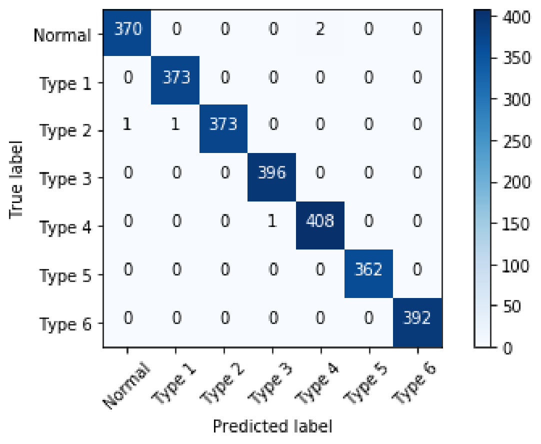
Figure 21. Histograms of four shallow neural network models' accuracy scores, measured by nine metrics.

Conducted experiments have shown that reasonable changes in the architecture and hyperparameters of the SNNs had little effect on the model's accuracy, which was quite high even for the default setup. The latter confirms the robustness of the model and matches the conclusions of Ba and Caruana (2014). Logistic (sigmoid) activation function was found to be slightly superior to others. Similarly, Adam (Kingma, & Ba, 2014) outperformed other optimization algorithms, especially SGD. According to Papamakarios (2014), this technique computes the gradient of a single feature, which is often a noisy estimate of the true gradient. In this study, the SGD was unable to complete the weights optimization within the devoted 200 epochs.

Another important hyperparameter is L_2 regularization, also known as weight decay. This method drives the weights closer to the origin. It heavily penalizes the model for steep weights, preferring balanced ones and encouraging it to make equal use of all input features rather than heavily use just a few of them. However, the penalty may be too high, in which case the network becomes oversimplified and accuracy declines (Ng, 2004). SNN 2 demonstrates this aspect.

Table 8 shows the full prediction details of the most accurate SNN variant:

Table 8. Confusion matrix and full classification report of the best-found shallow neural network classifier.



True label \ Predicted label	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	370	0	0	0	2	0	0
Type 1	0	373	0	0	0	0	0
Type 2	1	1	373	0	0	0	0
Type 3	0	0	0	396	0	0	0
Type 4	0	0	0	1	408	0	0
Type 5	0	0	0	0	0	362	0
Type 6	0	0	0	0	0	0	392

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       372
           1       1.00      1.00      1.00       373
           2       1.00      0.99      1.00       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

The model achieved 1.0 scores on all nine metrics, having only 5 out of 2,679 misclassified samples. In fact, most of the SNN variants that were tried out during the exhaustive search showed high accuracy, which characterizes this model as the most robust one of the four examined. However, SNNs still have two flaws:

- They cannot operate on raw data and require hand-crafted features.
- Being a FFNN and having just one hidden layer, the SNN is too simple to learn complicated long-term dependencies in the data.

The LSTM is expected to overcome these limitations, while preserving the robustness.

Results of Experiment 5

The last and most important experiment conducted in this dissertation was to use the LSTM model to detect and classify anomaly in the JDRF dataset. Its purpose was to explore the effectiveness of the model and see whether it is capable of the following:

1. Sense long-term temporal dependencies in time-series data.

2. Operate on time-series of variable length.

Numerous variations of an LSTM network could have been created by changing its architecture and its hyperparameters. However, unlike previously examined simpler models, LSTM training is significantly more time-consuming. Even on extremely powerful Amazon Web Services computing platform with 24 CPUs, 4 GPUs and 128GB of RAM the average training of a single LSTM variation lasted about ten hours. This constraint made the use of grid search impossible and compelled this study to take another approach. Several dozen of LSTM variations were evaluated step by step, gradually improving the accuracy by trying out different directions in changing architecture and hyperparameters. Much reliance in this process was placed on the experience of other researchers, in particular on the work of Reimers and Gurevych (2017) that evaluated the performance of over 50,000 different variations of LSTM.

Several hyperparameter optimizations were applied to all the LSTM variants. Similar to SNN, the number of epochs was limited. Lipton, Kale, Elkan, and Wetzel (2015) showed that after a certain point in the training, LSTM starts overfitting the training data. This requires to find a convergence tradeoff point, at which the model both fits the training data well and keeps its generalization ability high. In this work, an optimal threshold of 100 epochs was determined by means of early stopping and freezing of best-found model during full training. Another global optimization was variational dropout. This form of regularization is commonly used to improve the accuracy of deep neural networks (Gal, & Ghahramani, 2016). In this study the optimal value (for both input, output, and recurrent layers) was found to be 0.3. The final common method was the optimization algorithm. Andrychowicz et al. (2016) consider it a critical factor that can significantly hamper or haste the convergence of the LSTM. To verify this claim, several optimizers, such as RMSProp (Tieleman, & Hinton, 2012) and Adadelta (Zeiler, 2012), were examined. As in the case of SNNs, SGD showed the slowest convergence rate, while Adam was chosen as the best.

Next step was applying a batch gradient descent algorithm. According to Li, Zhang, Chen and Smola (2014), using small chunks of data to update the model's weights speeds-up stochastic convex optimization problems without being dropped into local minima. Batch size had to be carefully picked, since improper choice would cause a negative effect. Small batches could increase the cost of the noise, whereas large ones could significantly slow down the training. To find the best size, this work followed the theoretical guidelines of Neishi, Sakuma, Tohda, Ishiwatari, Yoshinaga, and Toyoda (2017), and the practical recommendations of You, Demmel, Keutzer, Hsieh, Ying, and Hseu (2018). Batch of 512 samples was found optimal for the JDRF dataset.

Further optimization attempts considered architectural changes. Figure 22 shows the result of four LSTM variants that delineated the most important decision in these searches, whereas Figure 23 shows a pair of graphs that reflect their accuracy and loss changes during cross-validated training versus the number of epochs:

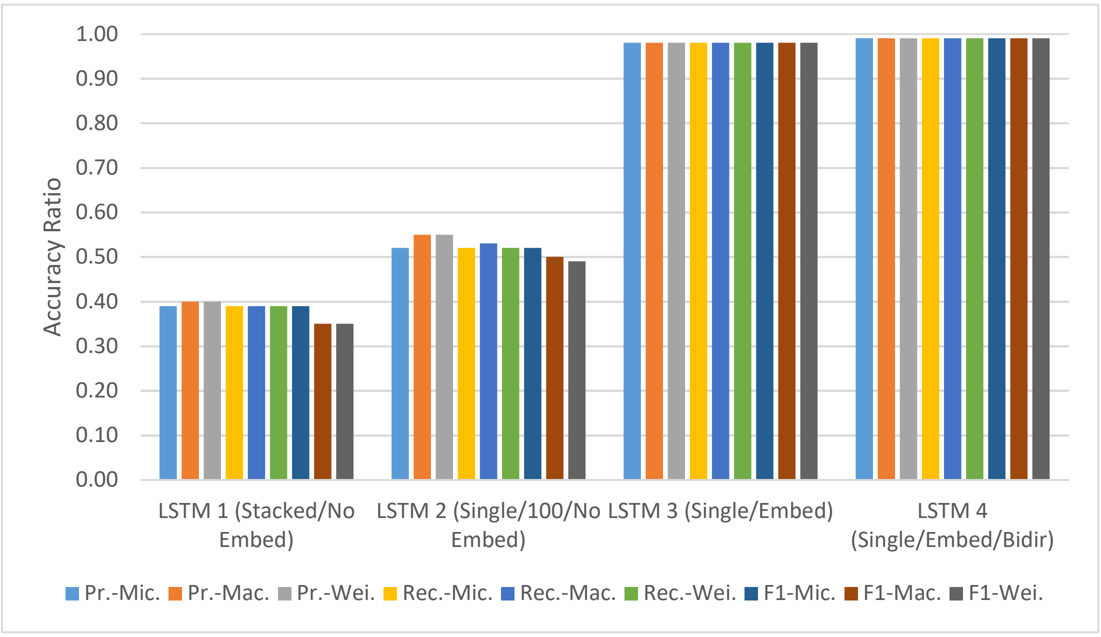
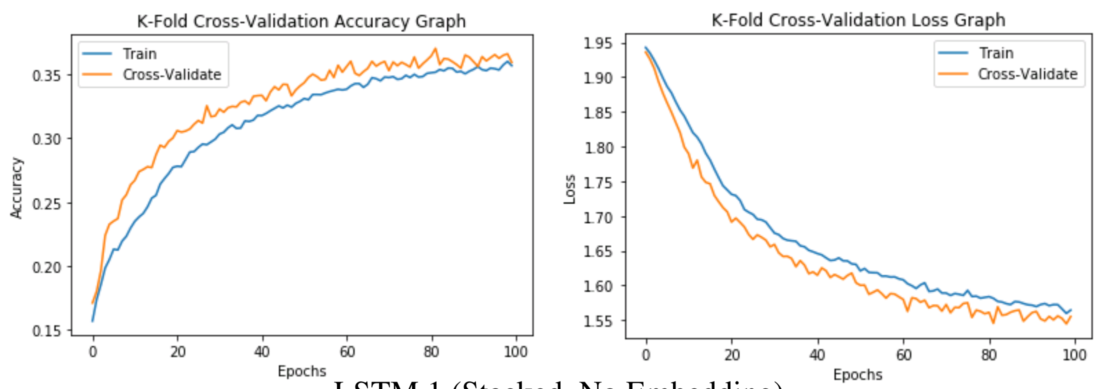
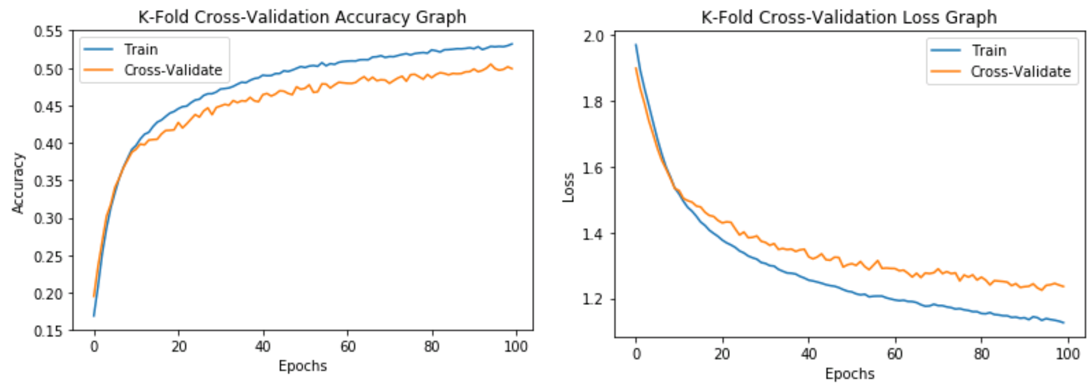


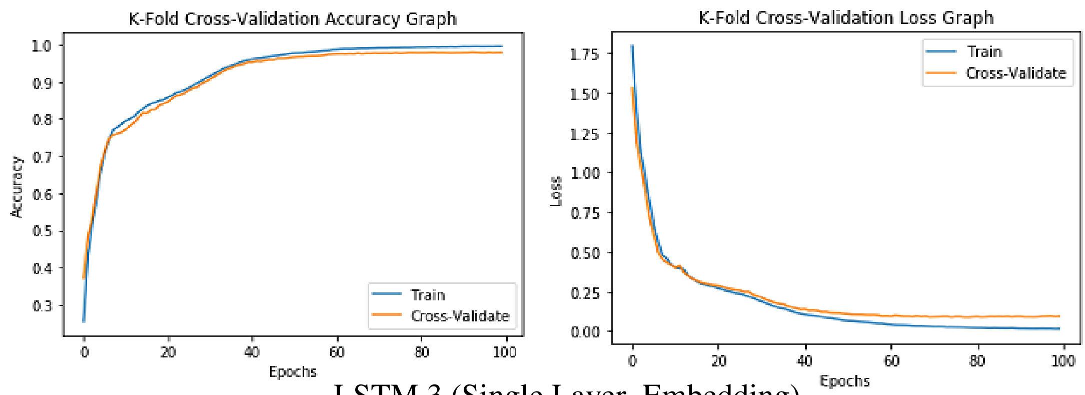
Figure 22. Histograms of four LSTM models' accuracy scores, measured by nine metrics.



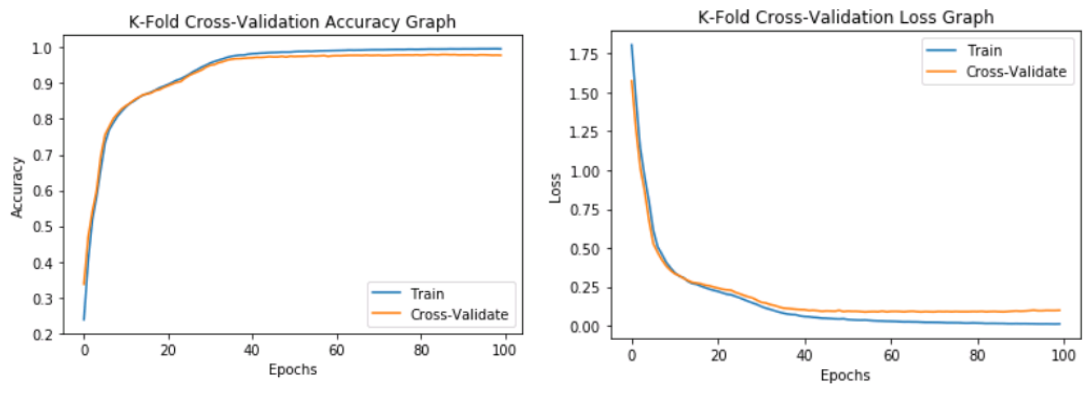
LSTM 1 (Stacked, No Embedding)



LSTM 2 (Single Layer, No Embedding)



LSTM 3 (Single Layer, Embedding)



LSTM 4 (Single Layer, Embedding, Bidirectional)

Figure 23. Graphs of four LSTM models' accuracy and loss versus epochs during *k*-fold cross-validation.

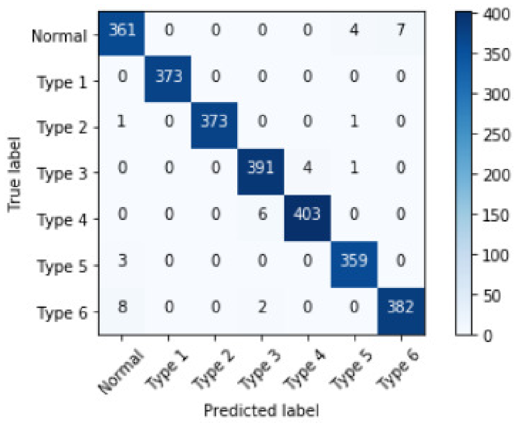
Stacked architecture was examined first, relying on the work of Graves, Mohamed and Hinton (2013) that found that the spatial depth of an LSTM network is more important, in terms of accuracy, than the number of its memory cells. LSTM 1 had two stacked layers with 10 LSTM units in the first and another 35 in the second. In this setup, the role of the first layer was to convert the input sequence to another sequence with chunks of sensor measurements over time, whereas the second layer classified these chunks. Having two levels of abstraction, the hidden state of the second level could operate at a different timescale than the first one, and was capable of learning more complicated long-term patterns. Nevertheless, the experiments have shown the opposite: the model scored slightly below 0.4. A conceivable explanation of such low results is that, although sequence to sequence modeling works well for one type of ML tasks, e.g. machine translation, it fails to obtain substantial progress in other tasks. In case of numeric time-series classification, the sequence produced by the second layer appeared to have smaller degree of informativeness than the original time sequence.

The setback of previous attempt pointed to the advisability of choosing a single-layered LSTM, while increasing the number of units to 100. According to Hermans and Schrauwen (2013), RNNs with deeper architecture are in general more accurate. This assumption proved correct. LSTM 2 was able to increase the accuracy by 10%. However, accuracy was still unacceptably low and required significant improvements.

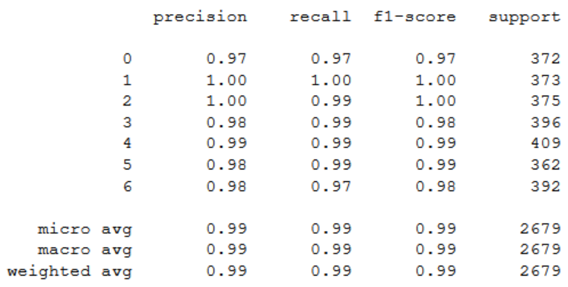
A rule of thumb, it can be said that the more information the LSTM can extract from the data, the better the prediction accuracy will be (Neishi et al., 2017). Therefore, the next architectural change focused on this aspect. So far, the first layer of the LSTM contained ordinary units. This approach considers the magnitude of the values and tries to analyze the fluctuations in the data by estimating the order in which the magnitudes were positioned. Another, slower, but more sophisticated, approach is to put an additional embedding layer at the front of the LSTM. This technique is widely used in text analysis (Palangi et al., 2016). In sequence classification, embedding maps each value into a continuous vector space, where the distance between words represents proximity. Using this technique, LSTM 3 could sense the relation between the frequencies of measurements that were closely positioned in the time-series. By extracting this valuable contextual information, the model achieved 98% accuracy.

The final architectural change was to use bidirectional LSTM architecture. BLSTM (Graves, & Schmidhuber, 2005) requires a special back-propagation technique, which further slows down the training. However, according to Graves (2012), BLSTMs are noticeably better in supervised labeling than unidirectional LSTMs. As previously said, in this architecture, two models are trained on the input sequence: one for positive time direction, and the other for negative one. Through this method, LSTM 4 was able to extract even more information from the raw data by considering the relation of each glucose level measurement to both preceding (past) and subsequent (future) measurements of the input time-series. To compensate the model for slightly longer time lag patterns in the negative direction, the number of units in each of the two LSTM parts was increased to 120. Altogether, these changes were able to push the accuracy to 99% in all metrics. Table 9 shows the results of the final and best-found LSTM.

Table 9. Confusion matrix and full classification report of the best-found LSTM network classifier.



True label \ Predicted label	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	361	0	0	0	0	4	7
Type 1	0	373	0	0	0	0	0
Type 2	1	0	373	0	0	1	0
Type 3	0	0	0	391	4	1	0
Type 4	0	0	0	6	403	0	0
Type 5	3	0	0	0	0	359	0
Type 6	8	0	0	2	0	0	382



```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       372
           1       1.00      1.00      1.00       373
           2       1.00      0.99      1.00       375
           3       0.98      0.99      0.98       396
           4       0.99      0.99      0.99       409
           5       0.98      0.99      0.99       362
           6       0.98      0.97      0.98       392

   micro avg       0.99      0.99      0.99      2679
   macro avg       0.99      0.99      0.99      2679
weighted avg       0.99      0.99      0.99      2679
```

LSTM 4 had only 36 out of 2,679 misclassified samples. In an applied research study like the current one, such accuracy can be considered almost flawless prediction, commensurate with that of feature-based traditional models. For fair, it should be said that the training time of LSTM 4 was significantly larger than that of the traditional algorithms. However, following its accuracy and loss curves in Figure 23, it can be seen that the model managed to fully converge within about 60 epochs, which is rather fast for an LSTM. Besides, once trained, the model acts many times faster when predicting new time-series. Given that the LSTM operated on raw data and, unlike the traditional models, had no clues on the anomaly types, its results are truly impressive. To conclude, it can be said that the LSTM network definitely qualifies as successful for anomaly detection and classification in sensor data.

Summary

This chapter provided the results of the experimental part of the dissertation and a detailed analysis of the examined models. The first four experiments considered classical ML models that operated on feature vectors. The last experiment related to a more contemporary deep learning approach that used the LSTM model to process raw data. Each experiment tried out different model variants, trying to find the best one.

Experiment 1 analyzed the SVM classifier. Numerous attempts have been made to change the model's kernel, violations penalty coefficient etc. Experiment 2 analyzed RF, whereas the best accuracy was searched by changing hyperparameters such as leaf split criterion, impurity, maximal trees depth etc. Experiment 3 reviewed the NBC. Several modifications of this model, e.g. Gaussian NB, Bernoulli NB, etc. were compared to find the most accurate model. The fourth experiment observed SNNs. The accuracy was tuned by changing the architecture of the network, applying regularization, changing the optimization algorithm and so on. The final experiment studied the LSTM model. A relatively large number of LSTM variations were worked through, gradually improving the accuracy by changing the hyperparameters and the architecture of the model.

Altogether 20 models were analyzed in depth. The results of experiments 1 through 4 showed that all classical ML models provided very high prediction accuracy when operating on feature vectors. Although it was important to confirm this fact, this experimental part of the dissertation was to some extent foreseen. As seen in Chapters 2 and 3, various studies have shown various anomaly detection and classification solutions based on traditional ML models. Basically, the purpose of the first experimental part was to show that older traditional ML models can provide very high prediction accuracy when provided with appropriate features, and serve as mean of control for the deep learning approach, carried out in the second part.

This last part was significantly more interesting and novel. Its goal was to show that LSTM can detect and classify anomaly in sensor data by analyzing raw data, having absolutely no domain knowledge neither on the data, nor on the anomalies covertly spliced into it. The results of the second part experiments show that the LSTM model successfully managed to detect and classify the anomaly in blood glucose level sensor data. The best-found model achieved a very high accuracy of 0.99 in all the nine metrics used in the study that thoroughly estimated the model from all the aspects of correct prediction.

Chapter 5 - Conclusions, Implications, Recommendations, and Summary

Having both the theoretical and the experimental parts of the dissertation completed, the role of this chapter is to summarize the study. The chapter briefly concludes the results achieved in the experimental part. It indicates the evidence of fully accomplishing the objectives stated in the study and delineates its strengths and weaknesses. It debates on the contribution of the dissertation to the field of anomaly detection in sensor data. It discusses the impact that the dissertation may have on the research community and elaborates on the future work that can be done in the direction that it has set. Finally, the chapter presents a short standalone report of the entire work.

The summary chapter is divided into the following sections:

- Conclusions
- Implications
- Recommendations
- Summary

Conclusions

The goal of this dissertation was to investigate the effectiveness of the LSTM for anomaly detection based on raw sensor data. The results of the conducted experiments clearly demonstrate that the model is highly efficient by based on the following facts:

- The LSTM was able to achieve 99% accuracy in the strict oversight of nine metrics that covered all aspects of prediction.
- The model had absolutely no knowledge the domain of the data.

- It required no human labor, except for fine-tuning, which can be made in a fully automated way according to Hwang and Sung (2015).
- Operating on raw sensor data, the LSTM was able to detect and identify randomly and deeply abided spliced anomalies of six different types.

Based on the above, it can be said that the objectives of this dissertation have been fully accomplished.

As any study with a practical element, this dissertation had several weaknesses and limitations. One of them is the fact that the training of the LSTM model was extremely computationally intense. A powerful deep learning workstation with vast CPU, GPU and RAM resources had to be used to find a model that provided the sufficient level of accuracy. Moreover, even on this machine, the training process of each model took many hours, especially of the BLSTM. As opposed to the LSTM, traditional ML models require far less time to train. With this in mind, it can be said that the hardware and time demands of the LSTM may in some degree prevent it from being widely used.

Another weakness of the study is that, due to limited time available, it was impossible in this study to carry out an exhaustive search over large multi-dimensional space of hyperparameters and architectures to evaluate virtually every possible LSTM variation. Manual tuning process required certain skills and knowledge, which is indeed a certain restriction. However, its impact is relatively minor, since it is possible to implement an algorithm that will analyze the results after each execution, take the right tactical decisions, and train an accurate LSTM model, starting from a basic variant and improving accuracy step-by-step.

Finally, another weakness lies in the data used in the study, which was ranged and of integer type. The conducted experiments have shown that the LSTM model can effectively detect and classify anomalies in this type of data. However, more complex types of data, e.g. measurements of sensors that return floating point values, or a vector of numbers, might appear significantly more difficult to analyze and predict. Theoretically this case is handled by LSTM's decoding layers. For instance, Rui, Jing, Rong-gui, and Shu-guang (2013) made a decoding layer for CAPTCHA images. However, an accuracy drop might take place and prevent this solution from being used.

Implications

The research conducted in this dissertation contributes several new directions to the field of anomaly detection in sensor data. As discussed in Chapter 2, there are currently only a few LSTM-based solutions for anomaly detection and classification. At the time of writing this dissertation, only a single study was found that proposed to use LSTM to detect anomaly in time-series data of sensors. The results of the experiments indicate that LSTM is an appropriate, convenient and recommended solution to open research problem stated in Chapter 1: detection and classification of anomalies in raw time-series sensors data in a way that does not require domain knowledge.

The contribution of this dissertation is that it explores the effectiveness of LSTM for detection and classification of anomaly in sensor data, and indicates that this model can successfully solve the posed problem. The positive results of the study motivate further research on using LSTM as a promising approach to sensor anomaly detection.

Recommendations

The positive results of this dissertation encourage additional research that will further investigate and improve the LSTM-based approach for the detection and classification of anomalies in sensor data. There are several possible directions for expanding the research. One is to change the input data type. In this direction, the following actions are legitimate:

- See if if the LSTM-based approach can be applied for sensor anomaly detection in other areas, such as physics, industrial, avionics, telecom etc.
- Check whether the approach can be used for other type of sensors, e.g. temperature, light, touch, flow and level etc.
- Verify that the approach works equally good for other type of sequential data, e.g. multidimensional decimal values, alphanumeric codes, and so on.

Another direction is to investigate the limitations and feasibility of the LSTM-based approach in aspects such as:

- Computational requirements.
- Challenges of implementation of the algorithm in software and/or hardware.
- Processing (prediction) speed of new data.
- Lengths of sequences sufficient for analysis.
- Integration into larger systems.

Finally, additional techniques may be required to be applied to the LSTM to allow it analyze more complex data and achieve acceptable levels of accuracy. Below is a list of possible improvements:

- One of the previously mentioned limitations of the BLSTM was very long training times. Training time is directly affected by the convergence rate. Therefore, it the latter is improved, it will take much less time to train the model. One of the methods that have not been applied in this dissertation is batch normalization. Laurent, Pereyra, Brakel, Zhang, and Bengio (2016) applied this method to input-to-hidden transitions of the network, i.e. the ones used in non-stacked LSTM. Cooijmans, Ballas, Laurent, Gulcehre, and Courville (2016) have applied it to hidden-to-hidden transitions, corresponding to those of the stacked-LSTM. Both groups of researchers managed to achieve faster convergence. Therefore, it is worthwhile exploring this hyperparameter and see if it can speed up the training.

- Another hyperparameter that can reduce training time is layer normalization (Ba, Kiros, & Hinton, 2016). This technique much resembles batch normalization, but is independent of batch size. According to the authors (ibid.), this method is very effective at stabilizing the hidden state dynamics and can substantially reduce the training time.
- One more idea for future work is to apply L_2 Regularization. According to Merity et al., (2017), this method helps to control the norm of the resulting model and reduces overfitting.
- The last suggestion concerns a very sophisticated network called nested LSTM (Moniz, & Krueger, 2018) that was designed to learn especially complicated multi-level dependencies in the data. Nested LSTMs have multiple memory levels. The value of a memory cell in this architecture is computed by an LSTM cell, which has its own inner memory cell. According to the authors, this LSTM is more accurate than both stacked and single-layer LSTMs.

Summary

Various sensor-based systems are widely used in our everyday life. These systems heavily depend on the validity of data they received from underlying sensors. Anomalous sensor data can cause unpredicted system behavior and have dangerous consequences. Therefore, providing the sensor-based systems with mechanisms that detect anomalies and identify their types in an automated manner is highly import.

For many years scientists and researchers have been conducting research in the field of data validation and have suggested many solutions. A large portion of recent solutions is based on traditional ML algorithms. These solutions made substantial progress in anomaly detection and classification of sensor data, and many of them have been implemented in large industrial systems with great benefits. However, even though current ML-based solutions have high accuracy and work well, they have a serious caveat - traditional ML models are based on hand-crafted features.

Manual feature construction requires both domain expertise and considerable human effort. In addition, these features are tailored to a concrete type of data and bounded with particular system design. Apparently, modern sensor-based systems are characterized by frequent changes in their design and in the type of the data they process. These events entail constant adaptation of anomaly detection mechanisms. This maintenance of the sensor-based systems is highly labor-intense and economically unsustainable.

These shortcomings point to the need for an approach that will provide significantly longer-lasting solutions to the problem of anomaly detection and classification in sensor data. These are solutions that do not require domain knowledge and could be relatively easily adjusted to new system design or new types of data. A recent research trend focuses on creating solutions that are based on contemporary ML models with automated feature extraction. The advantages of these new solutions are obvious.

One research trend that has quickly gained popularity is in the field of anomaly detection in DL algorithms. These ML models can operate on raw data, from which they automatically extract features. In particular, LSTM networks have proved to be especially effective in the classification of raw time-series data in various domains. Similar to other deep learning algorithms, these models do not require manually prepared features and are capable of analyzing raw time-series. However, their key advantage over other DL models is the ability to learn complicated long-term patterns in time-series data. This characteristic is especially important for sensor data.

Despite these clear advantages of LSTM and the promising results of previous studies that used LSTM to detect anomaly in time-series data, to date, the number of LSTM-based solutions in the field of anomaly detections and classification is relatively small compared to other methods. In fact, only one study suggesting to detect anomaly in time-series data of sensors by means of LSTM was found. Anticipating the potential feasibility of the LSTM, in this dissertation it was decided to investigate the effectiveness of this model for anomaly detection and classification of sensor data, and see whether it can be considered an accurately and robust solution that requires no domain knowledge and no labor-intense feature engineering.

To provide veridical experiments, the experiments were chosen to be carried out on a genuine medical dataset. Being heterogeneous, inconsistent, incomplete, etc., this data created the difficult conditions that helped to perform a trustworthy and credible verification of the LSTM's capabilities. The original normal data was substantially expanded by splicing random and deeply abided anomalies of six types. The LSTM had to analyze raw sensor data (both normal and anomalous) in its most basic form, having zero information on the type and nature of the anomaly.

As a control, the accuracy of detection and classification of the LSTM was compared to that of traditional classifiers. While the LSTM was trained on raw time-series data, traditional classifiers were trained on feature vectors that were derived from the raw data. The features were carefully picked to be precise and informative.

Each feature focused on a single type of anomaly, making it relatively easy for the traditional ML models to differentiate the anomaly types. The generalization abilities of all the models in this dissertation were estimated by means of 10-fold cross-validation. Their performance (classification accuracy) was evaluated based on the following nine metrics derived from the confusion matrix: micro-precision, macro-precision, weighted precision, micro-recall, macro-recall, weighted-recall, micro-F1 score, macro-F1 score, and weighted F1-score.

In Experiment 1 the SVM classifier was chosen. Various modifications of this model were tried by means of the grid search technique to find a combination of hyperparameters that provided the highest accuracy. The best found SVM showed nearly perfect accuracy of 1.0 in all metrics.

Experiment 2 the used the Random Forest classifier to classify the anomaly. This model had significantly more hyperparameters to work through. However, the fact that building and training of the model could be made in parallel made it possible to use exhaustive search and find a very accurate model. The chosen Random Forest also scored 1.0 in all accuracy metrics.

The third experiment addressed naive Bayes classifier. This model is significantly simpler than the others, and had no hyperparameters that required tuning. By traversing through several types of this model, the Gaussian NBC was chosen. Despite its simplicity, it was able to take advantage of the features' high informativeness and achieve very high accuracy of 99%.

Experiment 4 considers shallow neural networks. For this model it was also possible to use parallel grid search, which allowed the filtering of numerous models with inappropriate sets of architecture and hyperparameters. The best-found shallow neural network was another nearly perfect model, showing a full 1.0 score in all metrics.

Given the remarkable results of the first four experiments, it can be said that classical ML models are capable of precisely detecting and classifying anomaly in sensor data when operating on precise and informative feature vectors. However, although it was important to confirm this fact, this part of the experiments was to some extent foreseen. As seen in Chapters 2 and 3, various studies have shown various anomaly detection and classification solutions based on traditional ML models. Hence, its purpose was more to show that older traditional ML models can serve as mean of control for the deep learning approach, carried out in Experiment 5.

The final experiment of this dissertation studied the LSTM model. Although this model requires significantly more time to train, the use of an extremely powerful deep learning workstation on Amazon's cloud computing services made it possible to examine a relatively large number of LSTM variations. A wisely chosen strategy of gradually changing architecture and hyperparameters towards accuracy maximization resulted in a model that made correct predictions in 99% of the cases. The nine metrics chosen to keep the results trustworthy ensured that this high score was not obtained by chance and reflects the true capabilities of the network.

The results of the last experiment showed that the LSTM model is feasible for solving the daunting task of anomaly detection and classification in sensor data. The high accuracy clearly demonstrates the effectiveness of LSTM. Even though it was operating on raw sensor data with absolutely no domain knowledge, the LSTM was still able to successfully segregate normal data, as well as detect and identify all six types of anomaly that was randomly positioned and deeply buried. The model required no human labor, except for fine-tuning, which was manually performed in this study, but can be fully automated. Considering the arguments above, it is fair to say that the goals of this dissertation have been fully met.

Appendices

Appendix A - Detailed Experiment Results

This appendix provides detailed results of experiments 1 through 5. As previously mentioned, each experiment carried out in this study considered models of the same type, differing in their hyperparameters, architecture etc. From these models four were chosen to be closely inspected. After being trained with k-fold cross-validation, the models were tested on previously unseen data and evaluated by means confusion matrix and further computation of the following nine metrics:

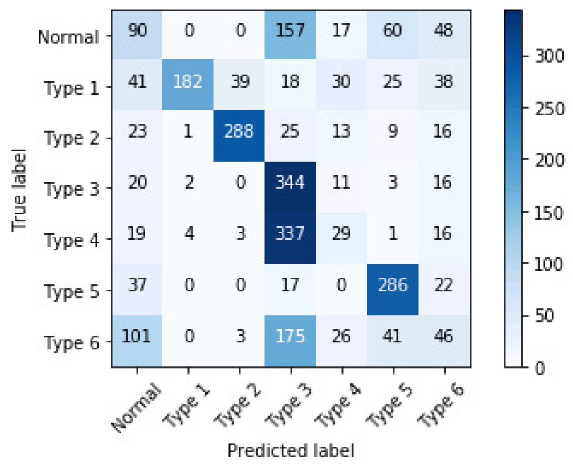
1. Micro-precision.
2. Macro-precision.
3. Weighted-precision.
4. Micro-recall.
5. Macro-recall.
6. Weighted-recall.
7. Micro-F1 score.
8. Macro-F1 score.
9. Weighted F1-score.

Below are screenshots with a summary of each model's hyperparameters, architecture (if necessary) and detailed accuracy results, corresponding to the above evaluation method. These screenshots can also be seen in the code (Jupyter notebooks).

Experiment 1

SMV 1

```
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               156000
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 707
=================================================================
Total params: 156,707
Trainable params: 156,707
Non-trainable params: 0
_________________________________________________________________
None
Model hyperparameters:
{
    "Optimizer": "SGD",
    "Dropout rate": 0.3,
    "Recurrent dropout rate": 0.3,
    "Batch size": 512
}.
Evaluating the model.
Confusion matrix:
```

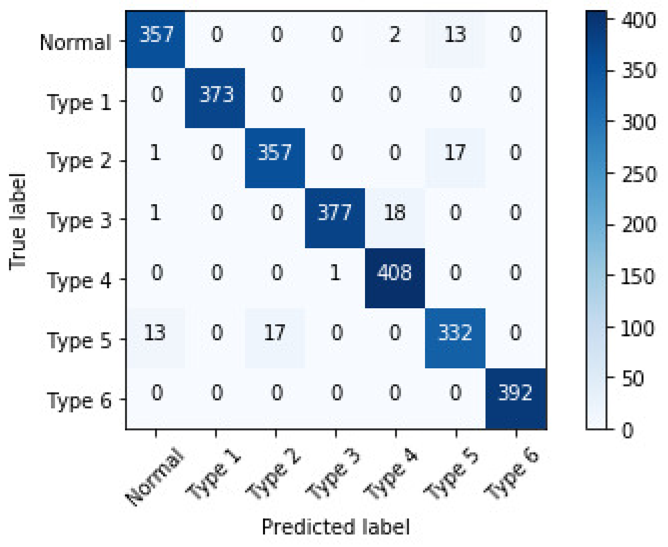


```
Classification report:
              precision    recall  f1-score   support

           0       0.27      0.24      0.26       372
           1       0.96      0.49      0.65       373
           2       0.86      0.77      0.81       375
           3       0.32      0.87      0.47       396
           4       0.23      0.07      0.11       409
           5       0.67      0.79      0.73       362
           6       0.23      0.12      0.15       392

   micro avg       0.47      0.47      0.47      2679
   macro avg       0.51      0.48      0.45      2679
weighted avg       0.50      0.47      0.45      2679
```

Experiment 1

SMV 2

```
Best hyperparameters found for the model:
{
    "C": 0.001,
    "kernel": "linear"
}
Evaluating the model.
Confusion matrix:
```

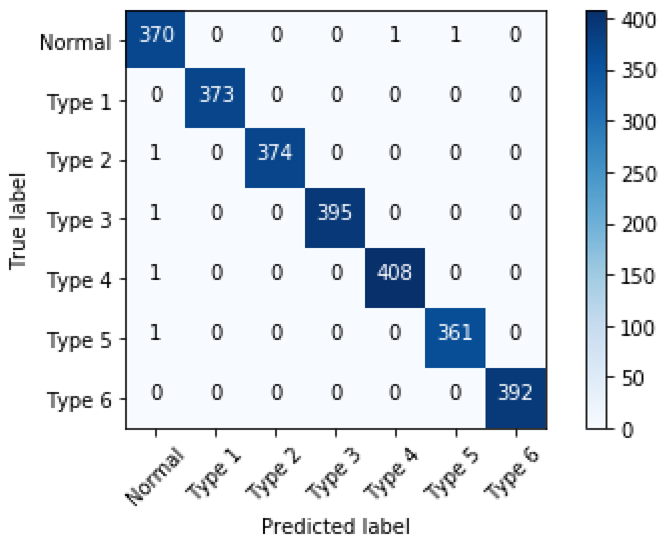


```
Classification report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       372
           1       1.00      1.00      1.00       373
           2       0.95      0.95      0.95       375
           3       1.00      0.95      0.97       396
           4       0.95      1.00      0.97       409
           5       0.92      0.92      0.92       362
           6       1.00      1.00      1.00       392

   micro avg       0.97      0.97      0.97      2679
   macro avg       0.97      0.97      0.97      2679
weighted avg       0.97      0.97      0.97      2679
```

Experiment 1

SMV 3

```
Best hyperparameters found for the model:
{
    "C": 9,
    "gamma": "scale",
    "kernel": "poly"
}
Evaluating the model.
Confusion matrix:
```



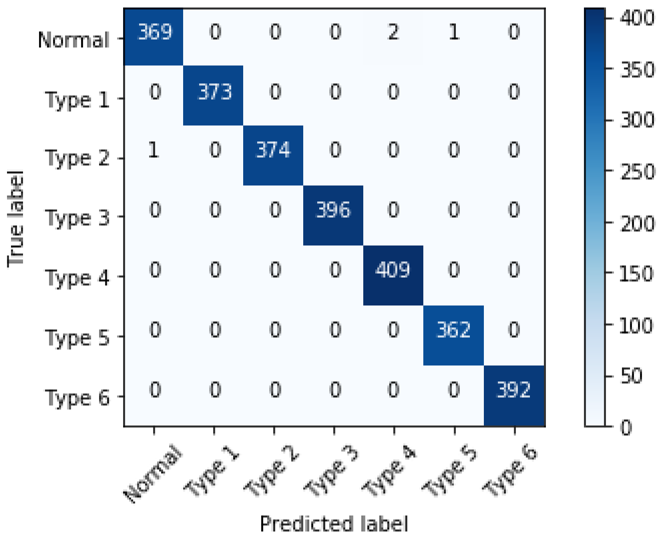
	Normal	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Normal	370	0	0	0	1	1	0
Type 1	0	373	0	0	0	0	0
Type 2	1	0	374	0	0	0	0
Type 3	1	0	0	395	0	0	0
Type 4	1	0	0	0	408	0	0
Type 5	1	0	0	0	0	361	0
Type 6	0	0	0	0	0	0	392

```
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       1.00      1.00      1.00       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

Experiment 1

SMV 4

```
Best hyperparameters found for the model:
{
    "C": 9,
    "kernel": "linear"
}
Evaluating the model.
Confusion matrix:
```

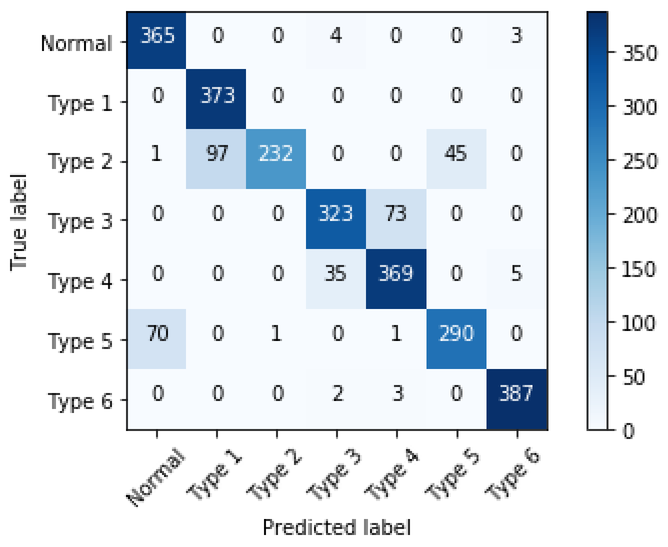


```
Classification report:
             precision    recall  f1-score   support

           0       1.00      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       1.00      1.00      1.00       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

Experiment 2

RF 1

```
Best hyperparameters found for the model:
{
    "bootstrap": true,
    "criterion": "entropy",
    "max_depth": null,
    "max_features": "auto",
    "min_samples_leaf": 0.1,
    "min_samples_split": 2,
    "n_estimators": 500
}
Evaluating the model.
Confusion matrix:
```

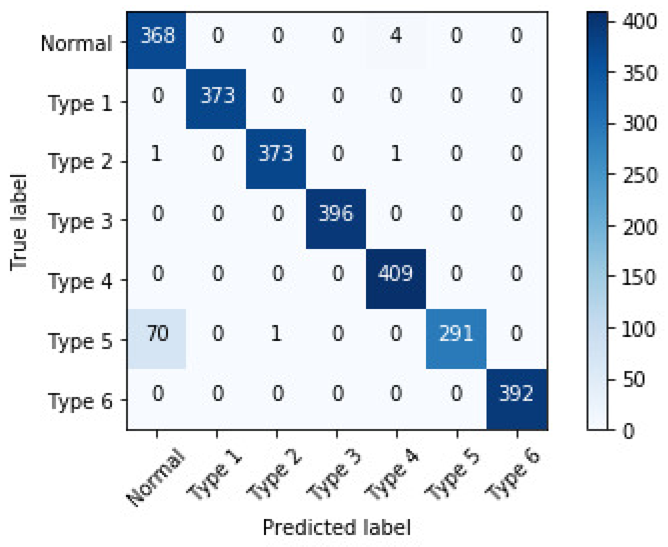


```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90       372
           1       0.79      1.00      0.88       373
           2       1.00      0.62      0.76       375
           3       0.89      0.82      0.85       396
           4       0.83      0.90      0.86       409
           5       0.87      0.80      0.83       362
           6       0.98      0.99      0.98       392

   micro avg       0.87      0.87      0.87      2679
   macro avg       0.88      0.87      0.87      2679
weighted avg       0.88      0.87      0.87      2679
```

Experiment 2

RF 2

```
Best hyperparameters found for the model:
{
    "bootstrap": true,
    "criterion": "gini",
    "max_depth": 2,
    "max_features": "auto",
    "min_samples_leaf": 1,
    "min_samples_split": 2,
    "n_estimators": 500
}
Evaluating the model.
Confusion matrix:
```

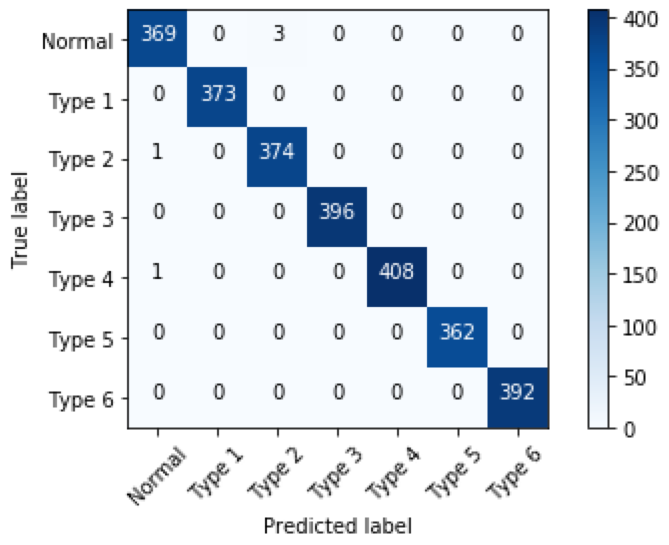


```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.99      0.91       372
           1       1.00      1.00      1.00       373
           2       1.00      0.99      1.00       375
           3       1.00      1.00      1.00       396
           4       0.99      1.00      0.99       409
           5       1.00      0.80      0.89       362
           6       1.00      1.00      1.00       392

   micro avg       0.97      0.97      0.97      2679
   macro avg       0.97      0.97      0.97      2679
weighted avg       0.98      0.97      0.97      2679
```

Experiment 2

RF 3

```
Best hyperparameters found for the model:
{
    "bootstrap": true,
    "criterion": "entropy",
    "max_depth": null,
    "max_features": "auto",
    "min_samples_leaf": 1,
    "min_samples_split": 2,
    "n_estimators": 500
}
Evaluating the model.
Confusion matrix:
```

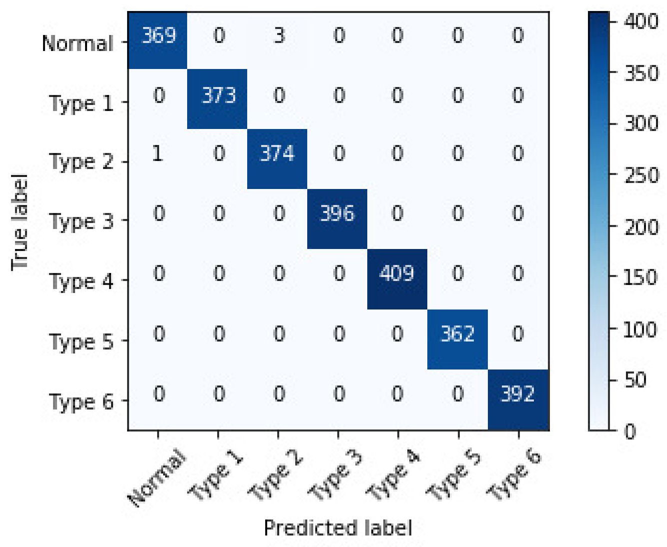


```
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       0.99      1.00      0.99       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

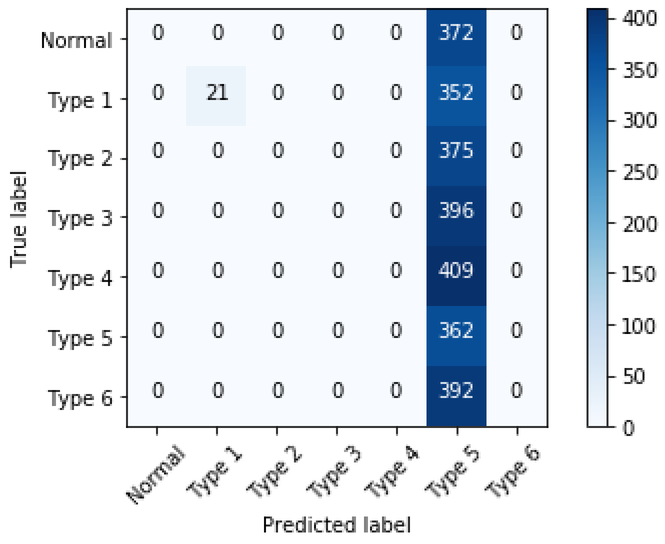
Experiment 2

RF 4

```
Best hyperparameters found for the model:
{
    "bootstrap": true,
    "criterion": "gini",
    "max_depth": null,
    "max_features": "auto",
    "min_samples_leaf": 1,
    "min_samples_split": 2,
    "n_estimators": 500
}
Evaluating the model.
Confusion matrix:
```



```
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       372
           1       1.00      1.00      1.00       373
           2       0.99      1.00      0.99       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

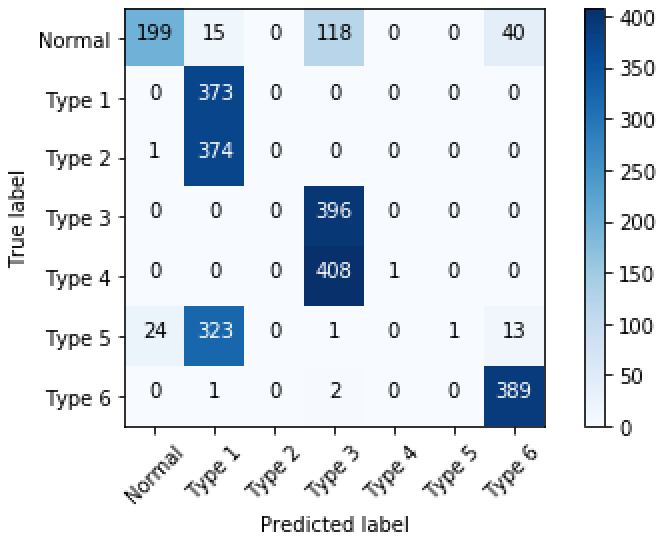
Experiment 3
 	
Bernoulli NB
	```
	Model type: BernoulliNB.
	Evaluating the model.
	Confusion matrix:
	```
	
	```
	Classification report:
	              precision    recall  f1-score   support

	           0       0.00      0.00      0.00       372
	           1       1.00      0.06      0.11       373
	           2       0.00      0.00      0.00       375
	           3       0.00      0.00      0.00       396
	           4       0.00      0.00      0.00       409
	           5       0.14      1.00      0.24       362
	           6       0.00      0.00      0.00       392

	   micro avg       0.14      0.14      0.14      2679
	   macro avg       0.16      0.15      0.05      2679
	weighted avg       0.16      0.14      0.05      2679
	```

Experiment 3

Complement

NB

```
Model type: ComplementNB.
Evaluating the model.
Confusion matrix:
```



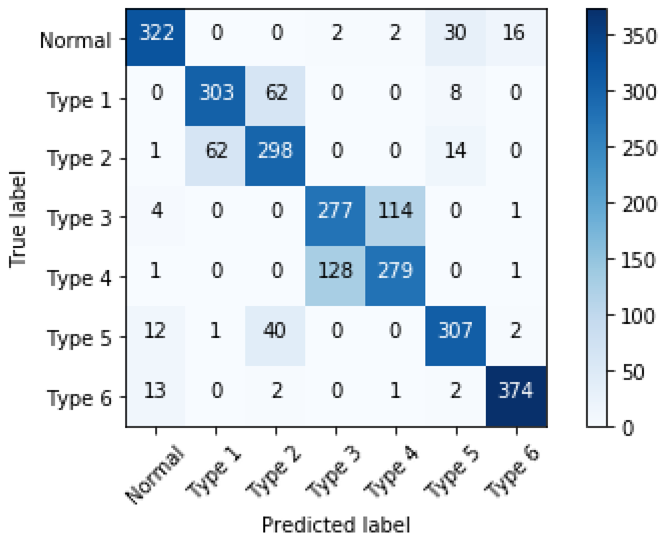
```
Classification report:
              precision    recall  f1-score   support

           0       0.89      0.53      0.67       372
           1       0.34      1.00      0.51       373
           2       0.00      0.00      0.00       375
           3       0.43      1.00      0.60       396
           4       1.00      0.00      0.00       409
           5       1.00      0.00      0.01       362
           6       0.88      0.99      0.93       392

   micro avg       0.51      0.51      0.51      2679
   macro avg       0.65      0.50      0.39      2679
weighted avg       0.65      0.51      0.39      2679
```

Experiment 3

Multinomial

NB

```
Model type: MultinomialNB.
Evaluating the model.
Confusion matrix:
```

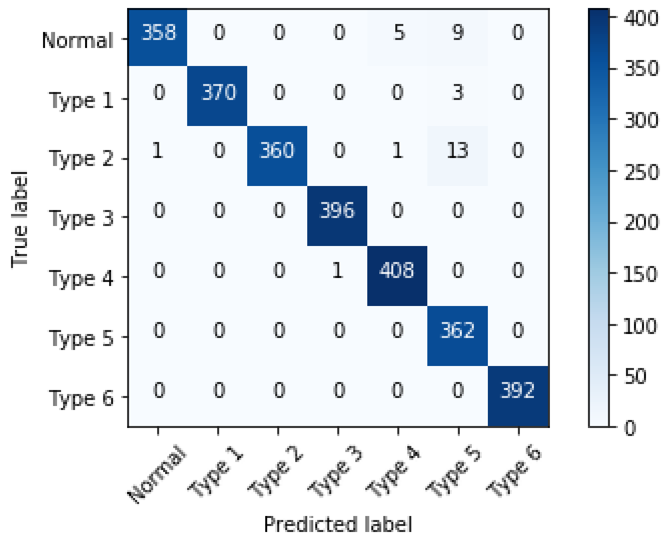


```
Classification report:
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       372
           1       0.83      0.81      0.82       373
           2       0.74      0.79      0.77       375
           3       0.68      0.70      0.69       396
           4       0.70      0.68      0.69       409
           5       0.85      0.85      0.85       362
           6       0.95      0.95      0.95       392

   micro avg       0.81      0.81      0.81      2679
   macro avg       0.81      0.81      0.81      2679
weighted avg       0.81      0.81      0.81      2679
```

Experiment 3

NB 4

```
Model type: GaussianNB.
Evaluating the model.
Confusion matrix:
```

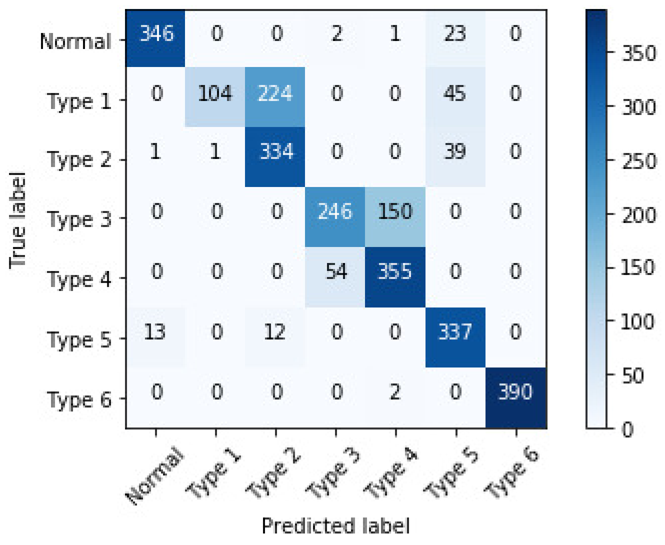


```
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       372
           1       1.00      0.99      1.00       373
           2       1.00      0.96      0.98       375
           3       1.00      1.00      1.00       396
           4       0.99      1.00      0.99       409
           5       0.94      1.00      0.97       362
           6       1.00      1.00      1.00       392

   micro avg       0.99      0.99      0.99      2679
   macro avg       0.99      0.99      0.99      2679
weighted avg       0.99      0.99      0.99      2679
```

Experiment 4

SNN 1

```
Best hyperparameters found for the model:
{
    "activation": "logistic",
    "alpha": 0.0001,
    "hidden_layer_sizes": [
        100
    ],
    "max_iter": 200,
    "solver": "sgd"
}
Evaluating the model.
Confusion matrix:
```

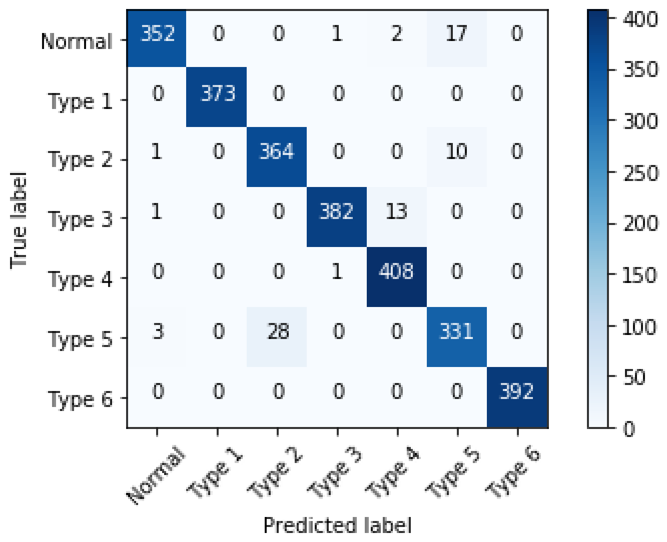


```
Classification report:
              precision    recall  f1-score   support

           0       0.96      0.93      0.95       372
           1       0.99      0.28      0.44       373
           2       0.59      0.89      0.71       375
           3       0.81      0.62      0.70       396
           4       0.70      0.87      0.77       409
           5       0.76      0.93      0.84       362
           6       1.00      0.99      1.00       392

   micro avg       0.79      0.79      0.79      2679
   macro avg       0.83      0.79      0.77      2679
weighted avg       0.83      0.79      0.77      2679
```

Experiment 4

SNN 2

```
Best hyperparameters found for the model:
{
    "activation": "logistic",
    "alpha": 1,
    "hidden_layer_sizes": [
        100
    ],
    "max_iter": 200,
    "solver": "adam"
}
Evaluating the model.
Confusion matrix:
```

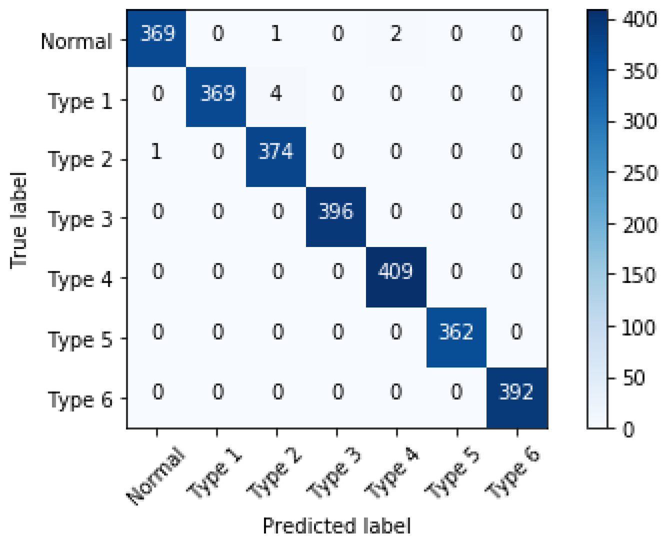


```
Classification report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97       372
           1       1.00      1.00      1.00       373
           2       0.93      0.97      0.95       375
           3       0.99      0.96      0.98       396
           4       0.96      1.00      0.98       409
           5       0.92      0.91      0.92       362
           6       1.00      1.00      1.00       392

   micro avg       0.97      0.97      0.97      2679
   macro avg       0.97      0.97      0.97      2679
weighted avg       0.97      0.97      0.97      2679
```

Experiment 4

SNN 3

```
Best hyperparameters found for the model:
{
    "activation": "logistic",
    "alpha": 0.0001,
    "hidden_layer_sizes": [
        100
    ],
    "max_iter": 200,
    "solver": "adam"
}
Evaluating the model.
Confusion matrix:
```

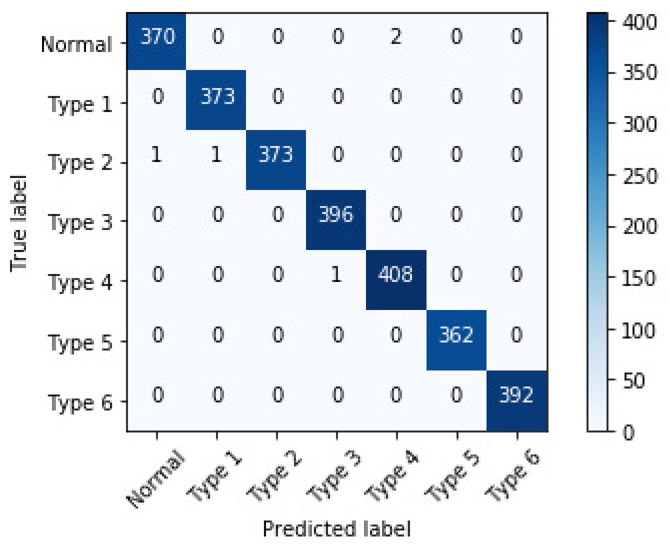


```
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       372
           1       1.00      0.99      0.99       373
           2       0.99      1.00      0.99       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

Experiment 4

SNN 4

```
Best hyperparameters found for the model:
{
    "activation": "logistic",
    "alpha": 0.0001,
    "hidden_layer_sizes": [
        15
    ],
    "max_iter": 200,
    "solver": "adam"
}
Evaluating the model.
Confusion matrix:
```

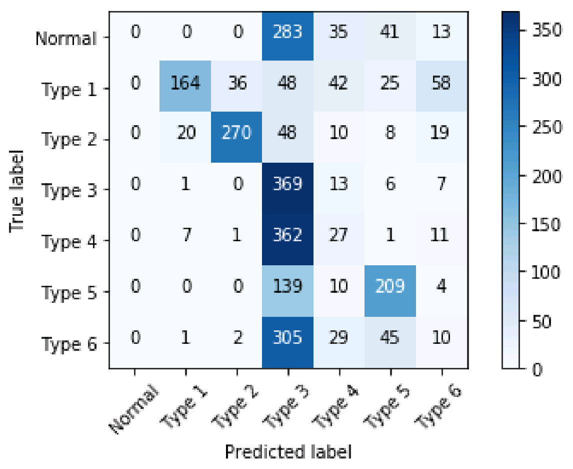


```
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       372
           1       1.00      1.00      1.00       373
           2       1.00      0.99      1.00       375
           3       1.00      1.00      1.00       396
           4       1.00      1.00      1.00       409
           5       1.00      1.00      1.00       362
           6       1.00      1.00      1.00       392

   micro avg       1.00      1.00      1.00      2679
   macro avg       1.00      1.00      1.00      2679
weighted avg       1.00      1.00      1.00      2679
```

Experiment 5

LSTM 1

```
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_13 (LSTM)               (None, 1, 10)             12000     
_________________________________________________________________
lstm_14 (LSTM)               (None, 35)                6440      
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 252       
=================================================================
Total params: 18,692
Trainable params: 18,692
Non-trainable params: 0
_________________________________________________________________
None
Model hyperparameters:
{
    "Optimizer": "Adam",
    "Dropout rate": 0.3,
    "Recurrent dropout rate": 0.3,
    "Batch size": 512
}
Evaluating the model.
Confusion matrix:
```

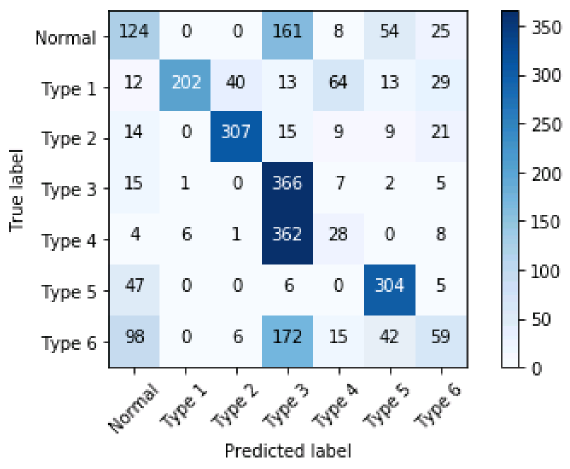


```
Classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       372
           1       0.85      0.44      0.58       373
           2       0.87      0.72      0.79       375
           3       0.24      0.93      0.38       396
           4       0.16      0.07      0.09       409
           5       0.62      0.58      0.60       362
           6       0.08      0.03      0.04       392

   micro avg       0.39      0.39      0.39      2679
   macro avg       0.40      0.39      0.35      2679
weighted avg       0.40      0.39      0.35      2679
```

Experiment 5

LSTM 2

```
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               156000
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 707
=================================================================
Total params: 156,707
Trainable params: 156,707
Non-trainable params: 0
_________________________________________________________________
None
Model hyperparameters:
{
    "Optimizer": "Adam",
    "Dropout rate": 0.3,
    "Recurrent dropout rate": 0.3,
    "Batch size": 512
}
Evaluating the model.
Confusion matrix:
```

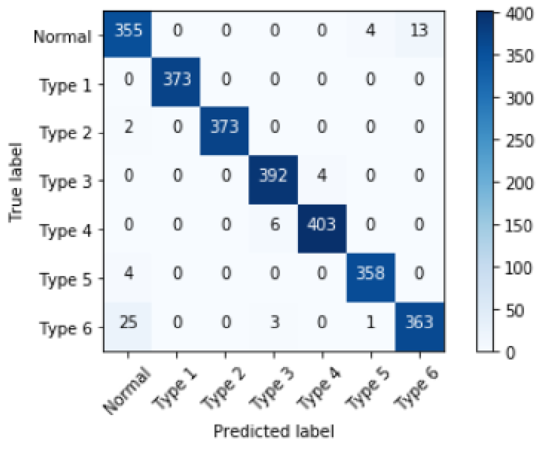


```
Classification report:
              precision    recall  f1-score   support

           0       0.39      0.33      0.36       372
           1       0.97      0.54      0.69       373
           2       0.87      0.82      0.84       375
           3       0.33      0.92      0.49       396
           4       0.21      0.07      0.10       409
           5       0.72      0.84      0.77       362
           6       0.39      0.15      0.22       392

   micro avg       0.52      0.52      0.52      2679
   macro avg       0.55      0.53      0.50      2679
weighted avg       0.55      0.52      0.49      2679
```

Experiment 5

LSTM 3

```
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 289, 1351)         1825201
_________________________________________________________________
lstm_10 (LSTM)               (None, 100)               580800
_________________________________________________________________
dense_10 (Dense)             (None, 7)                 707
=================================================================
Total params: 2,406,708
Trainable params: 2,406,708
Non-trainable params: 0
_________________________________________________________________
None
Model hyperparameters:
{
    "Optimizer": "Adam",
    "Dropout rate": 0.3,
    "Recurrent dropout rate": 0.3,
    "Batch size": 512
}
Evaluating the model.
Confusion matrix:
```

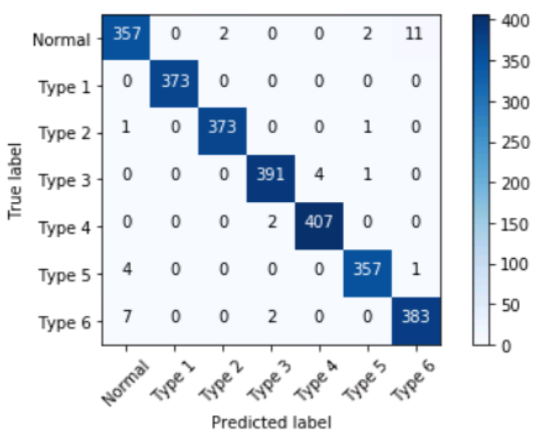


```
Classification report:
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       372
           1       1.00      1.00      1.00       373
           2       1.00      0.99      1.00       375
           3       0.98      0.99      0.98       396
           4       0.99      0.99      0.99       409
           5       0.99      0.99      0.99       362
           6       0.97      0.93      0.95       392

   micro avg       0.98      0.98      0.98      2679
   macro avg       0.98      0.98      0.98      2679
weighted avg       0.98      0.98      0.98      2679
```

Experiment 5

LSTM 4

```
Model summary:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 289, 1351)         1825201
_________________________________________________________________
bidirectional_6 (Bidirection (None, 200)               1161600
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 1407
=================================================================
Total params: 2,988,208
Trainable params: 2,988,208
Non-trainable params: 0
_________________________________________________________________
None
Model hyperparameters:
{
    "Optimizer": "Adam",
    "Dropout rate": 0.3,
    "Recurrent dropout rate": 0.3,
    "Batch size": 512
}
Evaluating the model.
Confusion matrix:
```



```
Classification report:
             precision    recall  f1-score   support

          0       0.97      0.96      0.96       372
          1       1.00      1.00      1.00       373
          2       0.99      0.99      0.99       375
          3       0.99      0.99      0.99       396
          4       0.99      1.00      0.99       409
          5       0.99      0.99      0.99       362
          6       0.97      0.98      0.97       392

  micro avg       0.99      0.99      0.99      2679
  macro avg       0.99      0.99      0.99      2679
weighted avg       0.99      0.99      0.99      2679
```

Appendix B - Third Party Libraries

This dissertation used a large number of third-party libraries for the Python 3.6 programming language. The table below lists this supplementary piece of software along with their versions and a short description as a token of appreciation to the contribution of other researchers and computer scientists that created these building blocks. All the third-party libraries and frameworks were used “as is”, without making any source code changes and/or configuration changes. In addition to these external libraries, many libraries that became part of the Python programming language, e.g. pickle, were intensively used in the study.

Library / Framework Name	Version Number	Short Description
Jupyter Notebook	5.7.4	A web-based integrated development environment for interactive computing.
Keras	2.2.4	Keras greatly simplifies deep learning tasks in Python. It runs on top of TensorFlow and has built-in and easy-to-work-with implementations of various neural networks models. In this study Keras was used to create, train, store, load and do all other necessary operations with LSTM.
Matplotlib	3.0.2	Matplotlib is a plotting library. It provides a vast number of built-in plots for various data graphical representation purposes. In this study it was used to visualize dataset sequences and LSTM train history.
nbimporter	0.3.1	nbimporter is a library for importing IPython notebooks, used by Jupyter IDE for Python, as modules. In this study it was used to simplify call supplementary functions from the main flow of the five experiments.
NumPy	1.16.1	NumPy is a fundamental Python library for scientific computing, making it simpler to work with arrays, matrices, use high-level math functions etc. This library was heavily used in this work for operations on data structures and various computations.
Pandas	0.24.1	Pandas is a library that greatly simplifies datasets processing, display and analysis. In this study it was used to extract the data from the JDRF dataset and prepare it for the ML models.
scikit-learn	0.20.2	Scikit-learn contains various traditional ML models for classification, regression, clustering and other tasks. In

		this work, implementations of all the classifiers of Experiments 1 through 4 were taken from this library.
TensorFlow	1.12.0	TensorFlow is widely used for machine learning, deep learning and numerical computations. In this study it was used as the backend for executing grid search with cross-validation and training the LSTM on multiple CPUs and GPUs.
xmltodict	0.12.0	xmltodict is a library that simplifies reading and writing data to and from XML files. This work used it to read user settings from the configuration file.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). Tensorflow: a system for large-scale machine learning. OSDI, 16, 265-283.

Aggarwal, C. C. (2015). *Data mining: the textbook.* Yorktown Heights, NY: Springer.

Al-Aidaroos, K. M., Bakar, A. A., & Othman, Z. (2010). Naive Bayes variants in classification learning. IEEE International Conference on Information Retrieval & Knowledge Management (CAMP), 276-281.

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI), 9(5)*, 272.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, 3981-3989.

Ashari, A., Paryudi, I., & Tjoa, A. M. (2013). Performance comparison between Naive Bayes, decision tree and k-nearest neighbor in searching alternative design in an energy simulation tool. *International Journal of Advanced Computer Science and Applications (IJACSA), 4(11).*

Auret, L., & Aldrich, C. (2010). Unsupervised process fault detection with random forests. *Industrial & Engineering Chemistry Research, 49(19)*, 9184-9194.

Auria, L., & Moro, R. A. (2008). Support vector machines (SVM) as a technique for solvency analysis.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450.*

Ba, J., & Caruana, R. (2014). Do deep nets really need to be deep?. *Advances in neural information processing systems*, 2654-2662.

Barreto, G. A., & Aguayo, L. (2009). Time series clustering for anomaly detection using competitive neural networks. *International Workshop on Self-Organizing Maps,* 28-36. Springer, Berlin, Heidelberg.

Bayes, T., Price, R., & Canton, J. (1763). An essay towards solving a problem in the doctrine of chances.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Neural networks: Tricks of the trade*, 437-478. Springer, Berlin, Heidelberg.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence, 35(8),* 1798-1828.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks, 5(2)*, 157-166.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13(Feb)*, 281-305.

Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems, 25(8)*, 1553-1565.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Cambridge: Springer.

Bombara, G., Vasile, C. I., Penedo, F., Yasuoka, H., & Belta, C. (2016, April). A decision tree approach to data classification using signal temporal logic. *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, 1-10. ACM.

Branisavljevic, N., Kapelan, Z., & Prodanovic, D. (2011). Improved real-time data anomaly detection using context classification. *Journal of Hydroinformatics, 13(3),* 307-323.

Breiman, L. (1984). *Classification and Regression Trees*. New York: Routledge.

Breiman, L. (1996). Bagging predictors. *Machine learning, 24(2)*, 123-140.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., ... & Layton, R. (2013). API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238.*

Cerrada, M., Zurita, G., Cabrera, D., Sanchez, R. V., Artes, M., & Li, C. (2016). Fault diagnosis in spur gears based on genetic algorithm and random forest. *Mechanical Systems and Signal Processing, 70*, 87-103.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR), 41(3)*, 15.

Chandola, V., Cheboli, D., & Kumar V., (2009). Detecting Anomalies in a Time Series Database. Technical Report. ACM.

Chauhan, S., & Vig, L. (2015). Anomaly detection in ECG time signals via deep long short-term memory networks. IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015, 36678, 1-7.

Chollet, F. (2015). Keras, GitHub. https://github.com/fchollet/keras

Chouhan, P., & Richhariya, V. (2015). Anomaly detection in network using genetic algorithm and support vector machine. *International Journal of Computer Science and Information Technologies, 6(5)*, 4429-4433.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555.*

Cooijmans, T., Ballas, N., Laurent, C., Gulcehre, C., & Courville, A. (2016). *Recurrent batch normalization*. arXiv preprint arXiv:1603.09025.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning, 20(3)*, 273-297.

Davis, C. (1962). The norm of the Schur product operation. *Numerische Mathematik, 4(1)*, 343-344.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning, 40(2)*, 139-157.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine learning, 29(2-3)*, 103-130.

Duan, K., Keerthi, S. S., & Poo, A. N. (2003). Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing, 51*, 41-59.

Edgeworth, F. Y. (1887). Xli. on discordant observations. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 23(143),* 364-375.

Eitrich, T., & Lang, B. (2006). Efficient optimization of support vector machine learning parameters for unbalanced datasets. *Journal of computational and applied mathematics, 196(2)*, 425-436.

Elman, J. L. (1990). Finding structure in time. *Cognitive science, 14(2)*, 179-211.

Ergen, T., Mirza, A. H., & Kozat, S. S. (2017). Unsupervised and Semi-supervised Anomaly Detection with LSTM Neural Networks. *arXiv preprint arXiv:1710.09207.*

Fehst, V., La, H. C., Nghiem, T. D., Mayer, B. E., Englert, P., & Fiebig, K. H. (2018). Automatic vs. manual feature engineering for anomaly detection of drinking-water quality. *Proceedings of the Genetic and Evolutionary Computation Conference Companion,* 5-6. ACM.

Fernandez-Delgado, M., Cernadas, E., Barro, S., & Amorim, D. (2014). Do we need hundreds of classifiers to solve real world classification problems?. *The Journal of Machine Learning Research, 15(1)*, 3133-3181.

Fernandez-Redondo, M., & Hernandez-Espinosa, C. (2001). Weight initialization methods for multilayer feedforward. ESANN, 119-124.

Fonseca, D. J., Navaresse, D. O., & Moynihan, G. P. (2003). Simulation metamodeling through artificial neural networks. *Engineering Applications of Artificial Intelligence, 16(3)*, 177-183.

Frank, E., Trigg, L., Holmes, G., & Witten, I. H. (2000). Naive Bayes for regression. *Machine Learning, 41(1)*, 5-25.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Icml, 96*, 148-156.

Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society for Artificial Intelligence, 14(771-780)*, 1612.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data mining and knowledge discovery, 1(1)*, 55-77.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning, 29(2-3),* 131-163.

Gal, Y., & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. Advances in neural information processing systems, 1019-1027.

Geng, Z., Tang, F., Ding, Y., Li, S., & Wang, X. (2017). Noninvasive continuous glucose monitoring using a multisensor-based glucometer and time series analysis. *Scientific reports, 7(1)*, 12650.

Genuer, R. (2012). Variance reduction in purely random forests. Journal of Nonparametric Statistics, 24(3), 543-562.

Gerard, N. (2017). Golf Dataset. Retrieved October 07, 2018 from https://gerardnico.com/data_mining/weather.

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN). Neural Computing: New Challenges and Perspectives for the New Millennium, 3*, 189-194.

Gers, F. A., & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks, 12(6)*, 1333-1340.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM. *Neural Computation, 12(10),* 2451-2471.

Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of machine learning research, 3(Aug)*, 115-143.

Goldstein, M., & Uchida, S. (2016). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one, 11(4), e0152173.

Gomes, G. S. D. S., Ludermir, T. B., & Lima, L. M. (2011). Comparison of new activation functions in neural network for forecasting financial time series. *Neural Computing and Applications, 20(3)*, 417-439.

Goutte, C., & Gaussier, E. (2005). A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. *European Conference on Information Retrieval*, 345-359. Springer, Berlin, Heidelberg.

Graves, A. (2012). Supervised sequence labelling. In Supervised sequence labelling with recurrent neural networks (pp. 5-13). Springer, Berlin, Heidelberg.

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks, 18(5-6)*, 602-610.

Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 6645-6649.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems, 28(10)*, 2222-2232.

Gribok, A. V., Hines, J. W., & Uhrig, R. E. (2000, November). Use of kernel based techniques for sensor validation in nuclear power plants. *Third American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation and Control and Human-Machine Interface Technologies, Washington DC.*

Guo, H., Jack, L. B., & Nandi, A. K. (2005). Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 35(1),* 89-99.

Gupta, A. K., Nguyen, T. T., & Sanqui, J. A. T. (2004). Characterization of the skew-normal distribution. Annals of the Institute of Statistical Mathematics, 56(2), 351-360.

Hassan, M. M. M. (2013). Network intrusion detection system using genetic algorithm and fuzzy logic. *International Journal of Innovative Research in Computer and Communication Engineering, 1(7)*, 1435-1445.

Hayes, M. A., & Capretz, M. A. (2014). Contextual anomaly detection in big sensor data. 2014 *IEEE International Congress on Big Data,* 64-71.

Heaton, J. T. (2017). *Automated Feature Engineering for Deep Neural Networks with Genetic Programming* (Doctoral dissertation, Nova Southeastern University, USA).

Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. Advances in neural information processing systems, 190-198.

Ho, T. K. (1995). Random decision forests. *Proceedings of the third international conference on document analysis and recognition, 1,* 278-282. IEEE.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9(8)*, 1735-1780.

Hodge, V., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review, 22(2),* 85-126.

Hofmann, T., Scholkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. The annals of statistics, 1171-1220.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences, 79(8)*, 2554-2558.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., & Soderstrom, T. (2018). Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. *arXiv preprint arXiv:1802.04431.*

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering, 9(3)*, 90-95.

Hwang, K., & Sung, W. (2015). Single stream parallelization of generalized LSTM-like RNNs on a GPU. *arXiv preprint arXiv:1503.02852.*

Ibarguengoytia, P. H. (1997). *Any time probabilistic sensor validation* (Doctoral dissertation, University of Salford, UK).

Izakian, H., & Pedrycz, W. (2013). Anomaly detection in time series data using a fuzzy c-means clustering. *IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, 1513-1518. IEEE.

Jager, G., Zug, S., Brade, T., Dietrich, A., Steup, C., Moewes, C., & Cretu, A. M. (2014). Assessing neural networks for sensor fault detection. *2014 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, 70-75.

John, G. H., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, 338-345. Morgan Kaufmann Publishers Inc.

Jones, E., Oliphant, T., & Peterson, P. (2014). SciPy: open source scientific tools for Python.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. Advances in psychology, 121, 471-495. North-Holland.

Jupyter Notebook [Computer software]. (2014). Retrieved from https:// jupyter.org/

Juvenile Diabetes Research Foundation Continuous Glucose Monitoring Study Group (2008). JDRF randomized clinical trial to assess the efficacy of real-time continuous glucose monitoring in the management of type 1 diabetes: research design and methods. *Diabetes Technol Ther. 10(4)*, 310-321.

Karayiannis, N. B., & Mi, G. W. (1997). Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques. *IEEE Transactions on Neural networks, 8(6)*, 1492-1506.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2018). LSTM fully convolutional networks for time series classification. *IEEE Access, 6*, 1662-1669.

Kim, T. Y., & Cho, S. B. (2018). Web traffic anomaly detection using C-LSTM neural networks. *Expert Systems with Applications, 106*, 66-76.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai, 14 (2)*, 1137-1145.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE, 78(9)*, 1464-1480.

Kononenko, I., & Kukar, M. (2007). *Machine Learning and Data Mining: Introduction to Principles and Algorithms,* Horwood Publishing.

Kruegel, C., Mutz, D., Robertson, W., & Valeur, F. (2003). Bayesian event classification for intrusion detection. Proceedings of the 19th Annual Computer Security Applications Conference, 14-23. IEEE.

Kuhn, M., & Johnson, K. (2013). *Applied predictive modeling*. New York: Springer.

LaLoudouana, D., Tarare, M. B., Center, L. T., & Selacie, G. U. A. N. A. (2003). Data set selection. *Journal of Machine Learning Gossip, 1*, 11-19.

Laurent, C., Pereyra, G., Brakel, P., Zhang, Y., & Bengio, Y. (2016). Batch normalized recurrent neural networks. *IEEE 2016 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2657-2661.

Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information processing letters, 5(1)*, 15-17.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 661-670.

Linda, O., Manic, M., Vollmer, T., & Wright, J. (2011, April). Fuzzy logic based anomaly detection for embedded network security cyber sensor. *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, 202-209. IEEE.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019.*

Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to diagnose with LSTM recurrent neural networks. arXiv preprint arXiv:1511.03677.

Lowe, D., & Broomhead, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex systems, 2(3)*, 321-355.

Lowe, D., & Tipping, M. E. (1997). Neuroscale: novel topographic feature extraction using RBF networks. Advances in neural information processing systems, 543-549.

Luo, R., Misra, M., & Himmelblau, D. M. (1999). Sensor fault detection via multiscale analysis and dynamic PCA. *Industrial & Engineering Chemistry Research, 38(4)*, 1489-1495.

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 89–94.

Manning, C.D., Raghavan, P., & Schutze, M. (2008). *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.

Martinez, A. M., Webb, G. I., Chen, S., & Zaidi, N. A. (2016). Scalable learning of Bayesian network classifiers. *The Journal of Machine Learning Research, 17(1)*, 1515-1549.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. arXiv preprint arXiv:1708.02182.

Microsoft Office [Computer software]. (2018). Retrieved from

https://products.office.com

Mikolov, T. (2012). Statistical language models based on neural networks. *Brno University of Technology*.

Misra, M., Yue, H. H., Qin, S. J., & Ling, C. (2002). Multivariate process monitoring and fault diagnosis by multi-scale PCA. *Computers & Chemical Engineering, 26(9)*, 1281-1293.

Mitchell, T. M. (1997). *Machine learning*, McGraw Hill, New York.

Moniz, J. R. A., & Krueger, D. (2018). Nested LSTMs. *arXiv preprint arXiv:1801.10308*.

Mosenia, A., Sur-Kolay, S., Raghunathan, A., & Jha, N. K. (2017). Wearable medical sensor-based system design: A survey. *IEEE Transactions on Multi-Scale Computing Systems, 3(2)*, 124-138.

Neishi, M., Sakuma, J., Tohda, S., Ishiwatari, S., Yoshinaga, N., & Toyoda, M. (2017). A bag of useful tricks for practical neural machine translation: Embedding layer initialization and large batch size. Proceedings of the 4th Workshop on Asian Translation (WAT2017), 99-109.

Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. Proceedings of the twenty-first international conference on Machine learning, 78. ACM.

Novak, V., Perfilieva, I., & Mockor, J. (2012). Mathematical principles of fuzzy logic, *Springer Science & Business Media, 517*.

NumPy [Computer software]. (2018). Retrieved from http://www.numpy.org/

Oliveira, A. L., Neto, F. B., & Meira, S. R. (2004). Combining MLP and RBF neural networks for novelty detection in short time series. Mexican International Conference on Artificial Intelligence, 844-853. Springer, Berlin, Heidelberg.

Otte, S., Liwicki, M., & Krechel, D. (2014). Investigating Long Short-Term Memory Networks for Various Pattern Recognition Problems. International Workshop on Machine Learning and Data Mining in Pattern Recognition, 484-497. Springer, Cham.

Palangi, H., Deng, L., Shen, Y., Gao, J., He, X., Chen, J., Song, X., & Ward, R. (2016). Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), 24(4), 694-707.

pandas [Computer software]. (2018). Retrieved from https://pandas.pydata.org/

Papamakarios, G. (2014). Comparison of Modern Stochastic Optimization Algorithms.

Parmar, J. D., & Patel, J. T. (2017). Anomaly Detection in Data Mining: A Review. *International Journal, 7(4).*

Patcha, A., & Park, J. M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks, 51(12),* 3448-3470. Elsevier.

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11)*, 559-572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research, 12(Oct)*, 2825-2830.

Picard, R. R., & Cook, R. D. (1984). Cross-validation of regression models. Journal of the American Statistical Association, 79(387), 575-583.

Pires, I. M., Garcia, N. M., Pombo, N., Florez-Revuelta, F., & Rodriguez, N. D. (2016). Validation Techniques for Sensor Data in Mobile Health Applications. *Journal of Sensors*, 2016.

Prajwala, T. R. (2015). A comparative study on decision tree and random forest using R tool. *International journal of advanced research in computer and communication engineering, 4(1)*, 196-1.

Probst, P., & Boulesteix, A. L. (2017). To tune or not to tune the number of trees in random forest?. *arXiv preprint arXiv:1705.05654.*

Probst, P., Wright, M., & Boulesteix, A. L. (2018). Hyperparameters and Tuning Strategies for Random Forest. *arXiv preprint arXiv:1804.03515.*

PyCharm [Computer software]. (2018). Retrieved from https://www.jetbrains.com/pycharm/

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning, 1(1)*, 81-106.

Quinlan, J. R. (1987). Generating production rules from decision trees. IJCAI *87,* 304-307.

Raileanu, L. E., & Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence, 41(1)*, 77-93.

Reimers, N., & Gurevych, I. (2017). Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799.*

Reitermanova, Z. (2010). Data splitting. WDS, 10, (31-36).

Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of naive Bayes text classifiers. *Proceedings of the 20th international conference on machine learning (ICML-03)*, 616-623.

Rish, I. (2001). An empirical study of the naive Bayes classifier. *IJCAI 2001 workshop on empirical methods in artificial intelligence, 3 (22),* 41-46. IBM.

Robinson, A. J., & Fallside, F. (1987). *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering.

Rojas, I., Pomares, H., Bernier, J. L., Ortega, J., Pino, B., Pelayo, F. J., & Prieto, A. (2002). Time series analysis using normalized PG-RBF network with regression weights. *Neurocomputing, 42(1-4)*, 267-285.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747.*

Rui, C., Jing, Y., Rong-gui, H., & Shu-guang, H. (2013). A novel LSTM-RNN decoding algorithm in CAPTCHA recognition. *Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*, 2013, 766-771. IEEE.

Russell, S., Norvig, P., & Intelligence, A. (1995). *Artificial Intelligence. A modern approach*. Prentice-Hall, New Jersey.

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics, 21(3)*, 660-674.

Salahat, E., & Qasaimeh, M. (2017). Recent advances in features extraction and description algorithms: A comprehensive survey. *2017 IEEE International Conference on Industrial Technology (ICIT)*, 1059-1063.

Salehinejad, H., Baarbe, J., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent Advances in Recurrent Neural Networks. *arXiv preprint arXiv:1801.01078.*

Sammut, C., & Webb, G. I. (2017). *Encyclopedia of machine learning and data mining*. New York, NY: Springer US.

Shcherbakov, M. V., Brebels, A., Shcherbakova, N. L., Kamaev, V. A., Gerget, O. M., & Devyatykh, D. (2017). Outlier detection and classification in sensor data streams for proactive decision support systems. *Journal of Physics: Conference Series, 803(1), 012143. IOP Publishing.*

Shipmon, D. T., Gurevitch, J. M., Piselli, P. M., & Edwards, S. T. (2017). Time Series Anomaly Detection; Detection of anomalous drops with limited features and sparse examples in noisy highly periodic data. *arXiv preprint arXiv:1708.03665.*

Shyu, M. L., Chen, S. C., Sarinnapakorn, K., & Chang, L. (2003). A novel anomaly detection scheme based on principal component classifier. *Miami University Coral Gables, Florida, Department of Electrical and Computer Engineering.*

Sikder, A. K., Petracca, G., Aksu, H., Jaeger, T., & Uluagac, A. S. (2018). A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *arXiv preprint arXiv:1802.02041.*

Singh, A. (2017). Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM).

Singh, S., & Murthy, T. R. (2012). Neural network based sensor fault detection for flight control systems. *International Journal of Computer Applications, 59(13).*

Stoecklin, M. (2006). Anomaly detection by finding feature distribution outliers. *Proceedings of the 2006 ACM CoNEXT conference*, 32-33.

Tamura, M., & Tsujita, S. (2007). A study on the number of principal components and sensitivity of fault detection using PCA. *Computers & Chemical Engineering, 31(9)*, 1035-1046.

Taylor, A., Leblanc, S., & Japkowicz, N. (2016). Anomaly detection in automobile control network data with long short-term memory networks. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 130-139.

Tibshirani, R. J., & Efron, B. (1993). An introduction to the bootstrap. *Monographs on statistics and applied probability, 57*, 1-436.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: *Neural networks for machine learning, 4(2), 26-31.*

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology, 49(11)*, 1225-1231.

Van Der Meulen, M. (2004). On the use of smart sensors, common cause failure and the need for diversity. *6th International Symposium Programmable Electronic Systems in Safety Related Applications.*

Van Rossum, G., & Drake, F. L. (2018). *Python tutorial*. Open Documents Library.

Verner, A., & Butvinik, D. (2017). A Machine Learning Approach to Detecting Sensor Data Modification Intrusions in WBANs. *IEEE 16th International Conference on Machine Learning and Applications - ICMLA 2017*, 161-169.

Wald, A. (1949). Statistical decision functions. *The Annals of Mathematical Statistics*, 165-205.

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. *Proceedings of the 10th IFIP Conference, System modeling and optimization, 762-770*. Springer, Berlin, Heidelberg.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model. *Neural networks, 1(4)*, 339-356.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing, 4(2),* 65-85.

Williams, R. J. (1989). *Complexity of exact gradient computation algorithms for recurrent neural networks*. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science.

Williams, R. J., & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation, 2(4)*, 490-501.

Williams, R. J., & Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science, 1(1)*, 87-111.

Wise, B. M., & Gallagher, N. B. (1996). The process chemometrics approach to process monitoring and fault detection. *Journal of Process Control, 6(6)*, 329-348.

Worden, K. (2003). Sensor validation and correction using auto-associative neural networks and principal component analysis. *Proceedings of the IMAC-XXI*, 973-982.

Wu, J., Cai, Z., & Zhu, X. (2013). Self-adaptive probability estimation for naive Bayes classification. The 2013 International Joint Conference on Neural Networks (IJCNN), 1-8. IEEE.

Yam, J. Y., & Chow, T. W. (2000). A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing, 30(1-4)*, 219-232.

Yan, W., & Goebel, K. F. (2003, August). Sensor validation and fusion for gas turbine vibration monitoring. *System Diagnosis and Prognosis: Security and Condition Monitoring Issues III, 5107*, 106-118. International Society for Optics and Photonics.

Yao, Z. G., Cheng, L., & Wang, Q. L. (2012). Sensor fault detection, diagnosis and validation-a survey. *Applied Mechanics and Materials, 229*, 1265-1271. Trans Tech Publications.

Yi, T. H., Huang, H. B., & Li, H. N. (2017). Development of sensor validation methodologies for structural health monitoring: A comprehensive review. *Measurement, 109*, 200-214.

Yoshikawa, N., Belkhir, N., & Suzuki, S. (2017). Recurrent Neural Network-based Fault Detector for Aileron Failures of Aircraft. *ASCC 2017-The 2017 Asian Control Conference*.

You, Y., Demmel, J., Keutzer, K., Hsieh, C., Ying, C., & Hseu, J. (2018). Large-Batch Training for LSTM and Beyond. *arXiv preprint arXiv:* 1901.08256.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, D., Qian, L., Mao, B., Huang, C., Huang, B., & Si, Y. (2018). A Data-Driven Design for Fault Detection of Wind Turbines Using Random Forests and XGboost. *IEEE Access, 6*, 21020-21031.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting, 14(1)*, 35-62.

Zhang, R., & Zou, Q. (2018). Time Series Prediction and Anomaly Detection of Light Curve Using LSTM Neural Network. Journal of Physics: Conference Series, 1061 (1), 012012. IOP Publishing.