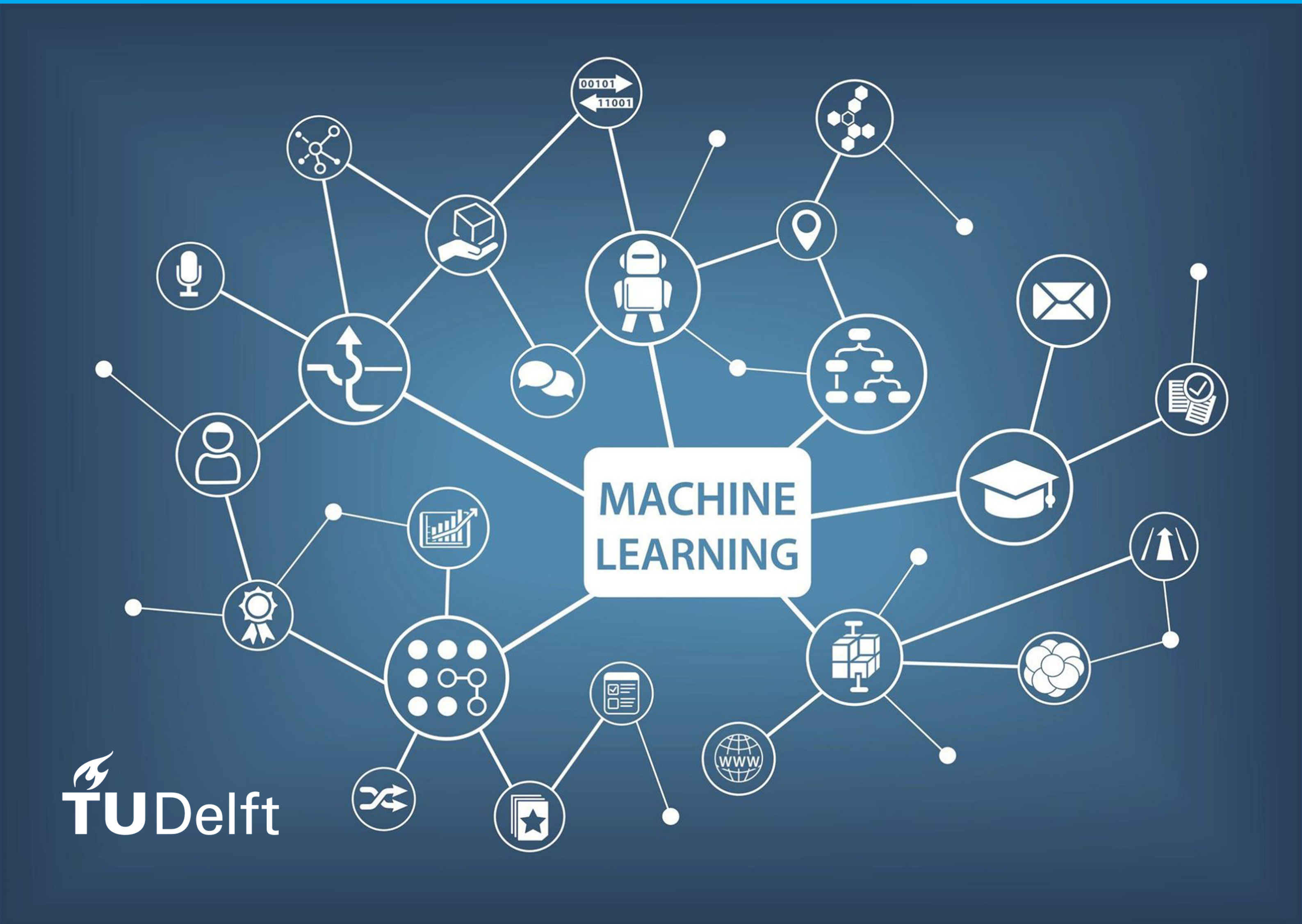
MSc thesis in Biomedical Engineering (3me)

Predicting Injury Risk with Machine Learning Methods using a Longitudinal Data Set

Lian Wu
2020



TU Delft

Predicting Injury Risk with Machine Learning Methods using a Longitudinal Data Set

by

Lian Wu

in partial fulfilment of the requirements for the degree of

Master of Science
in Biomedical Engineering

at the Delft University of Technology,
to be defended publicly on Wednesday June 24, 2020 at 1:00 PM.

Student number:	4230558
Thesis Committee:	**Prof. dr. Frans C.T. van der Helm**, Supervisor
	Delft University of Technology
	Larisa Gomaz, Daily Supervisor
	Delft University of Technology
	Dr.David Tax, External Committee Member
	Delft University of Technology
	Prof. dr.DirkJan Veeger, External Committee Member
	Delft University of Technology

Preface

I want to thank all the people that supported me during this project. I would like to first thank my supervisor, Frans van der Helm for providing me with the opportunity to be a part of this project and learn so much about Machine learning. Another thank you to my daily supervisor Larisa Gomaz who went above and beyond to help me with identifying aspects of my thesis, providing critical feedback and showing me to see problems with a different point of view. A word of gratitude to Evert Verhagen from Amsterdam UMC for providing me with the OSTRC data of this project. I also want to thank my sister who not only introduced me into the field of data analysis and Machine Learning, but also supported me throughout my entire Bachelor and Master studies. And finally, I would like to thank my parents who always supported me and pushed me to better myself. It has been a fun ride here at TU Delft and now I am ready to move on to the next step of my life.

Lian Wu
Delft, the Netherlands
June 24th, 2020

Abstract

Sport injury has long been cause of concern to athlete's performance, financial aspect and psychological impact. This is even more prominent in recent years as sports become more widely available to the mass population. Reducing the chances of athletes experiencing injuries not allows them to maintain optimal performance during training and competition, but is also psychological beneficial for the athletes. Amsterdam UMC have gathered weekly OSTRC data from 19 Waterpolo athletes over 109 weeks. By using this data, it is possible to build a model that can provide indication to possible injury risks thus helping athletes in controlling possible injuries.

Traditionally, statistical models are used for this kind of analysis. However, this often requires large amount of time and a-priori knowledge. Hence an injury risk classifier method based on longitudinal data (OSTRC) and machine learning algorithm was proposed in this study. The chosen ML algorithms for this study are ANN, LSTM and Random Forrest. To investigate the importance of time dependency between data entries, sliding window and forward chaining are used during data processing. All models trained with minimally processed data, sliding window data and forward chaining data to investigate the impact of time dependency in model accuracy.

The OSTRC data set is pre-processed, and SW and FC methods are applied. Data is then split into training and testing data set. Metric used to assess model performance are accuracy and confusion matrix. Results of this study show that both RF-SW and LSTM-SW produced accuracy of 92.17% and 91.67% respectively. Accuracy of ANN on training data indicate adequate performance, but confusion matrix indicate poor performance. Confusion of RF-SW and LSTM-SW show small variation due to difference in the test data set, which can be mitigated as more data are available. In conclusion, the high level of accuracy from both RF-SW and LSTM-SW proves that it can be used to provide insight to athletes, helping them to reduce the chances of injury.

Contents

1 Introduction

1.1. Problem Statement

The expansion in sports participation goes all the way to second half of the 20th century. The number of participants in sport activities increased due to increase of time and money reserved for sport. An overall attitude changed and sport became acknowledged as the right of everyone. By 1960s, sport was endorsed by the concept of "Sport For All", which offered sporting opportunities to the mass population. With more financial support, from the government and other sources, over the years number of professional and recreational athletes significantly increased. Previous studies conducted by the International Olympic Committee showed that between 1986 and 2012, the number of Olympic athletes increased from a few hundred to over 10000 with a great variety in sport disciplines over the years [18].

However, participation in sports activities has, among wide range of benefits, also a negative side in form of sport-related injuries. With an increase in number of participants, comes an increase in sport injuries as well. If we just take a look at the number of injuries that has occurred during hours of exposure in professional football, it gives a total injury incidence ranging from 2.48 to 9.4 per 1000 hours of athletic exposure. The injury rate during competition (game) is much higher on average, ranging between 8.7 and 65.9 injuries per 1000 athletic exposure [54] whilst it is only 1.37 to 5.8 per 100 hours of exposure during training. Ekstrand et al. [24] performed a 13 year longitudinal analysis of the UEFA Elite club and found out that there is an annual increase of 4.1% in hamstring injuries of professional football athletes since 2001.

An increased incidence and variety of sport-related injuries inevitably affect athlete's carrier quality of life and financial situation. Knowles [44] in 2007 showed that the medical costs alone for high school athletes in North Carolina were 709 dollars, with 2223 dollars per injury in human capital loss and 10432 dollars in comprehensive loss. Times [68] states that contact injuries in sports like flag football caused 1.3 billion dollars for college athletes and 19.2 billion dollars for high school athletes. The reason for such difference in medical costs is due to the much higher number of high school athletes compared to professional athletes. This is excluding the fact that professional athletes that play in major leagues would have a much higher comprehensive cost at the end. By monitoring athletes and their health status as well as their on-game and on-training performance, we can get a better insight in their progress,improve their performance and reduce injuries.

According to the National Electronic Injury Surveillance System (NEISS) in United States only in 2017, there were more than 3.5 million sports related injuries that were reported to be treated in hospital emergency departments [39]. Among all reported injuries, the most common injuries were related to strength training with equipment, this is followed by basketball, cycling and football. Some injuries resulted in a long recovery and some had more severe consequences such as an end of their careers. Douglas A. Kleiber [22] showed that forced end of a sports carrier can have a negative psychological impact on the athletes, such as lower degree of life satisfaction and self-esteem.

In order to enable athletes to participate in sports on desired level, there's a need for improving their performance, their well-being and reducing sport-related injuries. Improvement in athlete's performance should not be at the cost of increase in sport-specific injuries, but to reduce them. Having less frequent and less serious injuries, which results in more time available for the athletes to practice. Therefore, in this thesis among existing, new methods for reducing sport related injuries will be introduced. Among existing methods for preventing injuries, most common methods are warming up in combination with stretching at the beginning of a sport activity and cooling down at the end. Herman et al. [37] has shown that practical warm up strategies can reduce lower extremity injury incident in various athletes. Another method to reduce injury risk is by the use of sports equipment's. Graham [33] shows that sports equipment can result in reduction in several equipment associated injuries. Sport equipment is constantly being developed to reduce the risk of injury. Different materials were developed and improved over the years to increase its safety value. The purpose of all previously mentioned methods were implemented to reduce the occurrence of injuries and by that to allow the athlete maintain maximum performance. However, after implementing most of the existing methods for injury prevention, injury incidence in specific sport disciplines still remains very high.

The number of studies focusing on injury prevention increased recently. Besides the traditional methods for injury prevention such as warm up and stretching, mathematical models have been used for predicting the injury occurrence. New approaches, such as musculoskeletal and statistical modelling, offer data driven solutions for performance improvement of professional and recreational athletes. Both Gabbett [29] and Gabbett and Domrow [30] used statistical logistic regression models to estimate injury related elements. However, most of these models require large number of resources dedicated to develop the model with the capability of automation of the development and optimisation of the models using computers. With the recent leap in growth of computation power, development of models has become more and more eminent. This lead to a large increase in development in the field of Machine Learning (ML) and Artificial Intelligence (AI). It has great potential for modelling complex, non-linear depended data and is beneficial for problems that require large number of variables and data size. By applying ML approach to a problem such as injury prevention, we assume we would be able to predict an injury risk based on training routines ,physical characteristics and health status.

Although ML has proven to work very well is different applications, a common assumption in ML is the in dependency between input variables. In other words, all input variables in the system are independent from each other. That is rarely the case in the real world. Sometimes, a weak dependency between variables can be ignored and the model can still produce good results. However, when there is a strong dependency between variables, simply ignoring the relation could have catastrophic results.

Most of the studies that incorporate ML and cross-sectional data have shown great success. Using cross-sectional data indeed is possible to determine the hidden relationship between all relevant variables, but it ignores the possible time dependency that exists between data entries. On the other hand, longitudinal data has the characteristic of time-dependency, which normal ML methods do not take into account. Over the years of ML application, there are ML algorithms developed that can cope with dependency between variables, specially time dependency. Recurrent Neural Network [50] and Long Short Term Memory [38] are two ML algorithms that take time dependency into consideration, but research on implementation of ML algorithms for the analysis of longitudinal data is still lacking. Other benefits of ML algorithms include scalability and increased model performance with more data. A ML algorithm can be continuously used as more data becomes available to increase model performance.

1.2. Project Goal & Research Questions

Although there are more ML applications in different fields, research and application ML regarding injury risk prediction is still limited. A popular statistical modelling method to do so is called survival analysis. This method often requires extensive understanding of the data set and appropriate data manipulating techniques to be able to produce good models. The benefit of using ML is its simplicity and fast computation time compared to traditional methods. ML has proven its success in other fields and by investigating the possibility of using ML on longitudinal, it is possible to find out a model that produces good results whilst maintaining reasonable computation time.

The aim of this study is to investigate how accurate ML algorithms can predict occurrence of sport related health complaint or the injury risk of an individual athlete. Athlete are categorised into different classes that represent level of risk injury and ML algorithm will used OSTRC data provided to predict said classes. These classes are computed from the four basic OSTRC questions. This is further explained in the rest of the report. This thesis will investigate if machine learning algorithm can be used for the analysis of longitudinal data and provide the injury risk as an outcome of the analysis. An injury risk prediction model would enable coaches and athletes to adapt their training programs, perform better and reduce number of injuries.

So, in order to get a better understanding, the following research questions were proposed:

- What is the performance of different injury risk prediction models?
- Which type of ML model is best suited for injury risk prediction using longitudinal panel data?
- Does inclusion of time dependency improve model performance?
- Can ML algorithms provide adequate predictions for practical application?

1.3. Report Outline

This report consists of seven chapters. The second chapter discusses the characteristics of the OSTRC data used in this study. Information about model variables, pre-processing techniques and software used can be found here. chapter 3 discusses the methodology used to tackle the research questions. It mainly consists of injury risk definition and prediction strategies. These strategies are namely Artificial Neural Network (ANN), Long-short Term Memory (LSTM) and Random Forest (RF). Forward chaining (FC) and Sliding Window (SW), the different methods used to implement time dependency into the data set are also discussed in this chapter. The general working principle of all models and how they are trained are explained in detail. The results of all models and all time dependency implementation methods are compared in chapter 4 and discussed in chapter chapter 5. The conclusion of the thesis are included in chapter 6 and finally, some comments for future research can be found in chapter 7.

2 Data Description

This chapter describes the OSTRC longitudinal data that was in the analysis described in this study. In section 2.1 details input/output data and inclusion condition for the analysis. The section 2.2 describes the event definitions and a selection of possible predictors for the outcome of the analysis. Section 2.3 briefly introduces software used during this study.

2.1. Input Variable - OSTRC Overuse Longitudinal Data

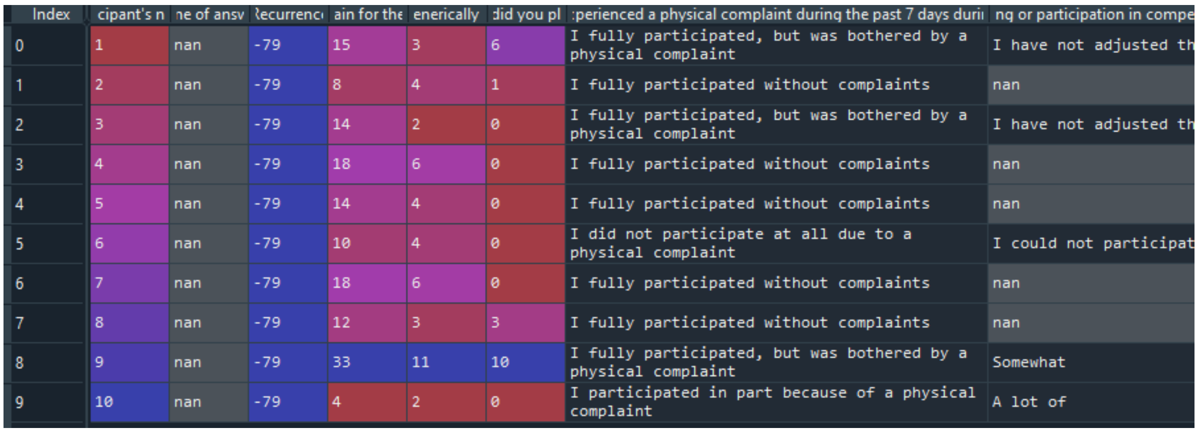
The data used during this study is provided by Amsterdam UMC. This data set includes weekly questionnaire related to professional Waterpolo athlete's physical conditions, training hours and health complaints that participants experienced during the week. This questionnaire is based on the basic Oslo Sports Trauma Research (OSTRC) structure [7]. The 4 basic questions that are used in a OSTRC questionnaire, are shown below:

1. **Participation:** Have you had any difficulties participating in normal training and competition during the past 7 days?
    - Full Participation without problems.
    - Full participation, but with problems.
    - Reduced participation due to problems.
    - Cannot participate due to problems.
2. **Modified training / Competition:**To what extent have you reduced your training volume due to problems during the past 7 days?
    - No reduction.
    - To a minor extent.
    - To a moderate extent.
    - To a major extent.
    - Cannot participate at all.
3. **Performance:** To what extent have problems affected your performance during the past 7 days?

- No effect.
- To a minor extent.
- To a moderate extent.
- To a major extent.
- Cannot participate at all.

4. **Symptoms:** To what extent have you experienced symptoms/ health complaints during the past 7 days?
    - No pain.
    - Mild pain.
    - Moderate pain.
    - Severe pain.

This is the example of four basic questions in the OSTRC questionnaire. This questionnaire was extended and adjusted for the water polo study and collect data from this study will be used in this thesis.



Index	cipant's n	ne of ansv	Recurrenc	ain for the	enerically	did you pl	:perienced a physical complaint during the past 7 days durii	ng or participation in compe
0	1	nan	-79	15	3	6	I fully participated, but was bothered by a physical complaint	I have not adjusted th
1	2	nan	-79	8	4	1	I fully participated without complaints	nan
2	3	nan	-79	14	2	0	I fully participated, but was bothered by a physical complaint	I have not adjusted th
3	4	nan	-79	18	6	0	I fully participated without complaints	nan
4	5	nan	-79	14	4	0	I fully participated without complaints	nan
5	6	nan	-79	10	4	0	I did not participate at all due to a physical complaint	I could not participat
6	7	nan	-79	18	6	0	I fully participated without complaints	nan
7	8	nan	-79	12	3	3	I fully participated without complaints	nan
8	9	nan	-79	33	11	10	I fully participated, but was bothered by a physical complaint	Somewhat
9	10	nan	-79	4	2	0	I participated in part because of a physical complaint	A lot of

Figure 2.1: The first 10 entries of raw OSTRC data-set provided for this study. Due to the large size of the data, not all variables are included in this figure.

As shown in Figure 2.1, there are variables that have sentences, or string as data entries. However, not all data entries have values in the cell. The difference between the data entries is briefly explained in Figure 2.2.

In a OSTRC questionnaire, it is common to have a series of follow up questions. This means that if there were "no complaint related" filled as answer, then the follow up questions will not have a data entry as well, showing as *nan* in the data set. Thus it is important to determine the total number of missing data points in this data set. This is presented in Table 2.1.

A total of 19 athletes participated in the study and 109 weeks of weekly OSTRC data were collected, resulting in a maximum of 2042 data entries. Some athletes missed certain week/weeks during the study, hence the total actual number of data entries is slight lower than theoretical number of data entries. Each data entry consists of multiple variables, or data points, illustrated in Figure 2.1. A total of 57260 data points were recorded. Within the total number of data points, 31439 were missing (*nan*) values. The total number of *nan* values in the data set is 54.9%, more than half of the total data entries are empty. This greatly reduced the total number of data available for this study.

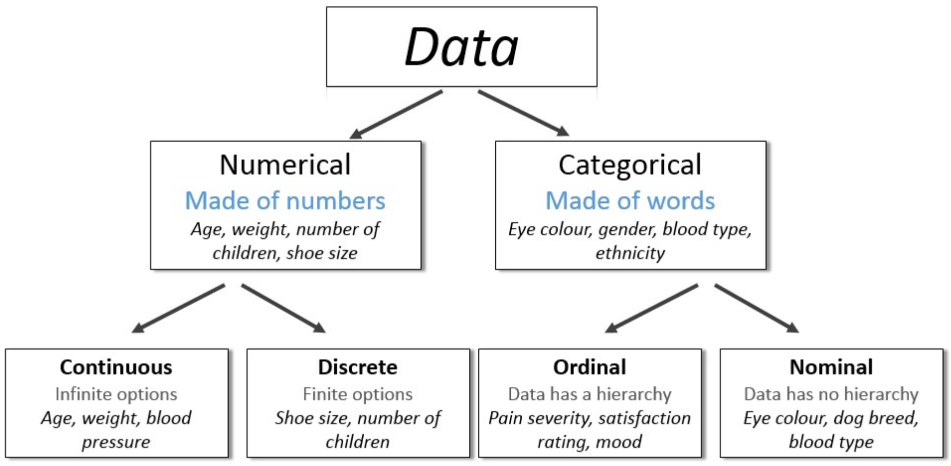

Data
Numerical
Made of numbers
Age, weight, number of children, shoe size
Categorical
Made of words
Eye colour, gender, blood type, ethnicity
Continuous
Infinite options
Age, weight, blood pressure
Discrete
Finite options
Shoe size, number of children
Ordinal
Data has a hierarchy
Pain severity, satisfaction rating, mood
Nominal
Data has no hierarchy
Eye colour, dog breed, blood type

Figure 2.2: Break down of Types of Data - Data are divided into 2 types, numerical and categorical. These 2 types of data are then further divided depending into their specific characteristic. [3]

Table 2.1: Basic statistics of the provided data set: Total and missing data points of raw OSTRC data set

	OSTRC Data
Period (Weeks)	109
Total Athletes	19
Total data entries	2042
Total data points	57260
Total Missing data points	31439 (54.9%)

On the positive side, the reason for such large amount of *nan* value is due to the fact that the athletes did not experience anything that impacted their physical or mental performance. Although this is *nan* value, it still has the potential of providing usable information and by learning from said *nan* values, it is possible for the model underlying relationship between training hours and possible discomfort.

Table 2.2: General statistics of all numerical data points in the raw OSTRC data set, including mean, min, 50% and max values of the numerical data in the raw data set.

	1	2	3	4	5	6	7
Mean	9.76	3.39	1.80	63.01	59.64	60.56	63.03
Min	0	0	0	0	0	0	0
50%	10	3	1.5	65	60	61	64
Max	36	16	14	72	100	100	100

Table 2.2 show the general statistics of numerical data entries in the data set. Due to the large name of the variables, it was replaced with numerical values, and Table 2.3 shows the conversion between variables and numerical values.

From Table 2.2, it is clear that most of the athletes spend more time in training for specific sport compared to either strength & fitness and play in matches. Most of the athletes feel moderate with respect to their motivation and energetic about next week's practice.

Table 2.3: Conversion of variable to numerical value. All "string variables" in Table 2.2 are replaced with a numerical value due to constraint in the width of the table. Left column represents the numerical value that is used to replace the variables in the right column.

	Variable
1	How many hours did you spend in sport-specific training?
2	How many hours did you spend in fitness and strength training?
3	How many hours did you spend in game/match?
4	How well have you slept in the past 7 days?
5	How rested do you currently feel?
6	How energetic do you feel at the moment?
7	How motivated do you feel for training and/or competitions in the coming week?

2.1.1. Input Variables

The study that provided the data followed 19 Athletes in a period of 109 weeks and each week, the OSTRC questionnaire was filled in. The list of all variables are included in the list below:

1. Participant's name
2. Recurrence
3. How many hours did you spend in sport-specific training?
4. How many hours did you spend in fitness and strength training?
5. How many hours did you spend in game/match?
6. Have you experienced any physical complaint during exercise in the past 7 days during exercise?
7. To what extent have you adjusted your training or participation in competitions in the past 7 days due to this physical complaint?
8. To what extent have you noticed in the past 7 days that this physical complaint has affected your performance?
9. To what extent did you experience the symptoms of this physical complaint in the past 7 days?
10. How many days in the last week have you been unable to participate fully or completely in a training or competition due to this physical complaint?
11. Does the physical complaint,pain or discomfort that you have experienced in the past 7 days concern an injury or illness?
12. ACUUT-Is this the first time you have reported this physical complaint?
13. ACUUT-During which activity did the injury occur?
14. ACUUT-Where is the injury?
15. OVERUSE-Is this the first time you have reported this physical complaint?
16. OVERUSE-During which activity did the injury occur?

17. OVERUSE-Where is the injury?
18. ILLNESS-Is this the first time you have reported this physical complaint?
19. How rested do you currently feel?
20. How energetic do you feel at the moment?
21. How motivated do you feel for training and/or competitions in the coming week?

There are in total of 21 different variables included in the provided data. Not all data entries have the same data type, a table summarising variables and their respective data types are show in Table 2.4.

Table 2.4: Summary of total number of different types of data entry

Variables with no value	1
Variables with numerical values	2,3,4,5,10,19,20,21
Variables with categorical values	6,7,8,9,11,12,13,14,15,16,17,18

It is clear from Table 2.4 that there is one variable that has no data entries at all. This is due to anonymisation of the data set done by the institution that gathered the data. Different data types will require different processing methods that are discussed in section 2.2.

2.1.2. Output Variables - Injury Risk

The aim of this study is to determine the performance of different ML algorithms on OSTRC data set. Input variables to the ML algorithms were identified in previous section. In total, there are 28 variables included in the data set for each data entry, but there is not a variable that subjectively and globally defines the injury risk status of the athlete. There is a variable that is related to injury status of the athlete, however, since all of the data entries are objective and not derived from a medical professional, it cannot be used as the output variable for this study. The four basic OSTRC questions reflect the possible consequences of a potential injury. The more severe an injury is, the longer time it requires to recover from it, as shown in Kraemer et al. [46]. Hence it is important to come up with an indicator that represents a global injury risk that not only is consistent with all athletes, but can also include factors through a time period. Since the questionnaire used to gather the data is based on the four basic questions in OSTRC, the injury status can be derived from these four questions.

In order to prepare the data for ML application, all of the data entries were transformed through encoding, explained further in section 3.3. Luckily, the answer to the four basic questions in the questionnaire has data hierarchy and ordinal encoding can be applied. Simple relation can be applied to determine the final numerical indication to the athletes injury status.

The injury risk status can be derived from the 4 basic questions since previous injury could have impact on current chances to experiencing injury and its severity. Fulton et al. [28] studied relation between injury and previous injury and concluded that consecutive injuries can alter the injury risk. For this reason, there are 2 scores that can be derived from the 4 basic questions, namely, First score (F) and Final Score (FS) to assess the injury risk. First score is calculated from the 4 basic questions relating to Participation, Modified training/ competition, Performance and Symptoms, shown in Equation 2.1. From ordinal encoding, each of the answer is assigned a value ranging from 0-5. This score is able to reflect on initial injury risk status of that week. The basic principle of this injury risk status is derived from OSTRC severity score. OSTRC severity score is the sum of the 4 basic OSTRC questions, ranging from 0 to 100. The higher the score indicate higher severity [17]. For this study, instead of a range between 0 to 100, it is categorised into 0 different categories, each category representing different level of injury risks.

$$F_n = 25\% \cdot Q_1 + 25\% \cdot Q_2 + 25\% \cdot Q_3 + 25\% \cdot Q_4 \tag{2.1}$$

where:

- Q_1 = First OSTRC Question Data point of n^{th} entry
- Q_2 = Second OSTRC Question Data point of n^{th} entry
- Q_3 = Third OSTRC Question Data point of n^{th} entry
- Q_4 = Fourth OSTRC Question Data point of n^{th} entry

Depending on the severity of the injury, the consequence is that said injury could impact the performance and injury status of the coming week/weeks. A severe injury experience by an athlete could result in consecutive low and poor performance. In order to take this phenomenon into account, data from previous week/weeks are taken into account as well to calculate the First score and Final score. According to Sharon A. Plowman [63], injury/discomfort of athlete carries on for no more than 4 weeks. This conclusion is also supported by Ekstrand et al. [25]. Among the 4 weeks, the further away from current week the data entry it is, the less impact-full its influence will have on current week. This means a different week will have different weight factors to calculate FS scores, since the athlete would have had time and help from health professional to recover from an injury. The Equation 2.2 is an example which all 4 week data entries were taken into account when computing FS score.

$$FS_n = 50\% \cdot F_n + 30\% \cdot F_{n-1} + 15\% \cdot F_{n-2} + 5\% \cdot F_{n-3} \tag{2.2}$$

where:

- n = n^{th} data entry
- F_n = First Score (F) of n^{th} entry
- F_{n-1} = First Score (F) of $n^{th} - 1$ entry
- F_{n-2} = First Score (F) of $n^{th} - 1$ entry
- F_{n-1} = First Score (F) of $n^{th} - 1$ entry

However, this relation cannot be applied to the first 3 data entries of each athlete since the requirement is to have at least 4 data entries.

In Figure 2.3, a visualisation showing the relation between F score and FS score is presented. Each F score contributes to multiple FS score of neighbour week. Each FS score is composed of F score of current and previous 3 weeks. This logic does not fully apply for the first 3 entries of the data since there are not four weeks of data to perform the calculation. In this case, the number of contributing weeks is reduced depending on which entry it is.

Table 2.5: Example of Weight factors of different conditions applied to F scores when there are missing week data entries. F_n represents the F score of n^{th} week. Percentages in the horizontal direction always add up to 100%.

Condition	F_4	F_3	F_2	F_1
All Weeks Present	50%	30%	15%	5%
Missing Week 1	60%	30%	10%	N/A
Missing Week 1-2	70%	30%	N/A	N/A
Missing Week 1-3	100%	N/A	N/A	N/A

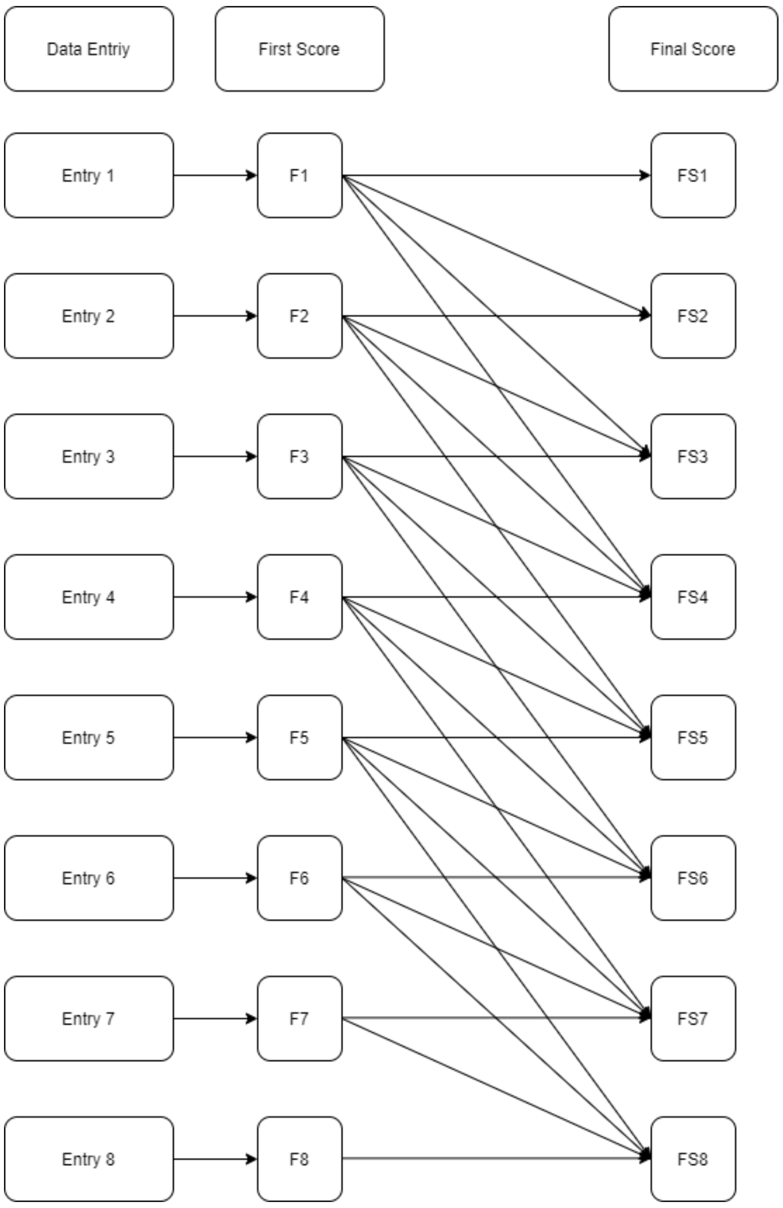


Figure 2.3: Logic flow of how F score is used to calculate FS score. Each FS score is composed of F score from current and previous 3 weeks. Missing weeks used to calculate F score will have adjusted weights, illustrated in Table 2.5.

This FS is used to categorise different injury risk status. The FS is rounded to its nearest integer, ranging from class 0 to class 5. Each class would represent different injury status of a particular athlete. Each class was define as following:

- Class 0 means that it has close to 0 injury risk of experiencing an injury.
- Class 1 means that is a low risk of experiencing mild injury and no risk in experiencing moderate or severe injury risk.
- Class 2 means that there is a medium risk of experiencing mild injury and low risk of experiencing moderate injury. The chances of experiencing severe injury is close to none.
- Class 3 means that there is a high risk of experiencing mild injury and medium risk of experiencing moderate injury. There is also a low risk of experiencing severe injury.
- Class 4 means that there is a high risk of experiencing moderate injury and medium risk of experiencing severe injury.
- Class 5 means that there is a high risk of experiencing severe injury.

A visual representation of the classes and their corresponding injury severed risk can be found in Figure 2.4. Depending on the week number, how data entry contributes to the F score and FS score is different.

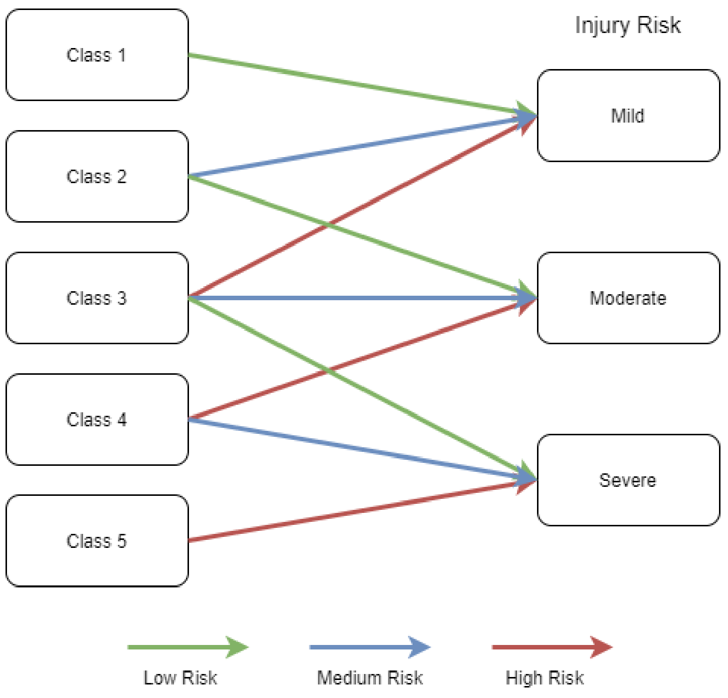


Figure 2.4: Level of Injury Risk and its corresponding class and injury severity. Class 0 means that it will have no injury risk. Whilst any other classes represent different possible injury severity.

The injury risk defined will be used as a output to the model. The input data are used to train the data whilst output data will be the target that the model is trying to predict.

2.2. Data Pre-processing

In a ML problem, all of the spotlight are generally focused on different ML algorithms. However, one of the more important procedures that allows the success of ML algorithms is data pre-processing.

Data pre-processing techniques are all methods used to process the acquired data in preparing for application of different ML algorithms.

But why is data pre-processing so important? In order for the results of ML application to be reliable, data needs to satisfy requirements of accuracy, completeness, consistency, timeliness, believability, and interpretability [36]. Having an incorrect data means that any ML algorithm will produce inaccurate results. Having an incomplete data set could lead to serious problems since it is possible that some of the ML techniques is not capable of processing missing data, this either breaks the algorithms or will just result in very unreliable results. If this is not caught in the earlier phase, could drastically increase the time required to fix the problem. Thus it is extremely important to have insight into the techniques used in data pre-processing.

According to Géron [35], Han and Pei [36], most of the data pre-processing techniques applicable to this study are categorised into the following sections:

- Data Cleaning
- Data Reduction
- Data Transformation

The pre-processing techniques are generally applied in the same order as it is presented in the above. Each of the pre-processing techniques used during this study is discussed in the next few sections.

2.2.1. Data Cleaning

It is possible for the collected data to be incomplete, such as having missing data, noisy data which contains errors or outlier data and inconsistent data meaning that the data could be stored in different formats. If none of the pre-processing techniques were applied directly to resolve the issue and ML algorithms were applied, it is highly unlikely that the results are trustworthy. Thus it is useful to apply the necessary data processing techniques before ML application. [32, 36]

Missing Data

Sometimes it is possible for some values with one or multiple attributes missing in the data set. In order to prepare the data for ML application, it is optimal to fill in the value. In the following list, some of the techniques are used to deal with missing data in a data set.

- **Ignore**
  Depending on the type and the size of the data set, it is possible to ignore (not include) this data. It is however a less feasible option when the size of the data set is relatively small. A small change in the data could have a relatively large impact on the outcome of the results. However, no matter the size of the data set, when there are multiple attributes with missing data, then the whole combination of the data can be excluded.

- **Use mean**
  Missing data can also be treated by filling in mean value of that specific category. This is more common to do when all the data in the same category is numerical data and not categorical data.

- **Use median**
  Another method to fill in missing values is by using the median value of the data. Median is the value that separates the higher half from the lower half. This is not as common as other methods.

- **Use most probable value**
  Finally, one of the more popular method is to use the most probable value to fill the missing data. One of the advantages of using this method is that it can be applied to both numerical data and categorical data.

Another situation that requires data cleaning is when there is noise present in the data. Nevertheless, since the focus will mostly be on questionnaire data, it is highly unlikely for questionnaire data to have noise, the exact methods used to deal with noisy data will not be discussed.

As mentioned previously, in a questionnaire, there are often follow up questions. If some variables have a value of zero due to an answer from previous question, then it also possible to fill in as zero instead of using the methods mentioned above. Of course this is case dependent and it can only be decided after analysis of the data set.

2.2.2. Data Integration

Depending on when, where and what kind of data is acquired, data integration might be necessary. Data might have come from different data centre or composed by different researchers meaning different data structure. Having a proper data integration can help to reduce and avoid redundancies and inconsistencies in the data set. There are 4 main problems in the data that should be solved during the integration phase, namely identification problem, redundancy & correlation and finally, value conflict.

Data from different sources could have the attributes named differently, thus how can data from different sources that have different attribute names be correlated and combined into one data set becomes the issue. Carefully identifying the different attributes and properly merging the data set attributes ensures a smoother process down the line.

Sometimes in the same data set, redundant data could be present. Redundant data is data that can be derived from other attributes. Having said data in the data set increases the computational time while providing no more new or useful information to the system. Thus it is wise to exclude such data from the data set during pre-processing phase to reduce computational time later on. Redundant data can be detected using correlation analysis. Correlation analysis can detect the correlation between attributes. If there are 2 attributes that are redundant to each other, correlation analysis would indicate a correlation of 100 %. On the other hand, if the attributes have no correlation at all, correlation analysis would show a 0 % correlation.

Finally, different data sets from different sources might differ in scale and attribute name. The best example is probably international online shopping. Buying the same product from the same website but in a different country means that the end price will be in different currency. If all the data is collected in the same data base, for the same item, some of the purchasing records would show a difference in price. If such difference is continued through the ML process, problems will arise since the algorithm does not understand the difference between the currencies. Thus it is wise before performing any ML analysis, to check and make sure that all of the prices share the same currency. Another problem could arise in the same purchasing record. For example, there is a record of purchasing showing that a 100 euro item was sold to a particular customer, the record will show that 1 item of 100 euro is sold. However, it is clear that 1 and 100 are not on the same scale. It is a common problem with the scaling of the data. If the difference of scale between data is too big, it might cause problems for certain number of ML algorithms.

2.2.3. Data Transformation

In this pre-processing step, there are methods applied to the data set that allows for easier understanding of the pattern hidden in the data set. This is called data transformation. Different methods of data transformation can be found in the following table.

- **Normalisation**
  Normalisation allows data to be scaled within a specific range. This range is usually (-1.0 to 1.0 or 0.0 to 1.0). An illustration of normalisation can be seen in Figure 2.5.

- **Attribute Selection**
  New attributes can be constructed from the given set of attributes to help the mining process.

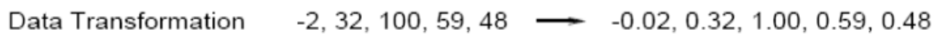


Figure 2.5: Visual illustration of Data Transformation [26]

2.3. Software

There are a wide range of different libraries in python that can be used for ML. Among these, there are some that are crucial for applying ML in Python. Scikit-learn[10] is one of the oldest and most popular ML libraries to date. It includes a wide range of ML algorithms that are commonly used. Another ML library for algorithms is Tensorflow. Tensorflow [11] is a relatively new library developed by Google. Other additional libraries for managing data include Numpy[6], Matplotlib[4] and Pandas[8]. Numpy allows for easy manipulation of large multidimensional arrays and matrices. It also includes a wide range of mathematical functions for array operation. Matplotlib mainly includes function to create plots and figures for visualisation purposes. Finally, Pandas allows manipulation of data structures of numerical tables.

3 Methodology

This chapter consists of 8 different sections. The first section discusses prediction strategies used. This is followed by methods used to properly prepare the data set in section 2. The third section explains different encoding method used. Basic principles of Artificial Neural Network (ANN), Long Short Term Memory (LSTM) and Random Forest (RF) are discussed in section 4 ,5 and 6 respectively. Performance metrics is then discussed in section 7. Finally, data structure reconfiguration is explained in section 8.

3.1. Prediction Strategies

Machine Learning (ML) has become one of the most popular methods in prediction modelling and has produced exceptional results in different fields. We are seeing a trend of companies and organisations using power of data to understand aspects of their target population or subject. In medical field, hospitals are using data mining to optimise hospital operations and performing disease detection [40]. In sports, researchers use data to have a better understanding of their athletes performance thus maximising athletes potential. Statistically speaking, data mining techniques have shown to be more successful in comparison to the failure rate. This proves that ML techniques, being part of data mining, are showing promising results.

Data are often categorised into two types, cross-sectional and longitudinal data, as shown in Figure 2.2. The most common type of data used in ML applications is cross-sectional data. Cross-sectional data is a type of data collected by observing many subjects, such as an individual athlete at one point or period of time [1]. Using cross-sectional data indeed is possible to determine the hidden relationship between all relevant variables, but it ignores the possible time dependency that resides between different variables. Longitudinal data is a collection of repeated observations of the same subjects, taken from a larger population, over longer time. The variables collected in longitudinal data are time dependent. This is a characteristic that normal ML methods do not take into account.

Time dependency is the biggest difference between longitudinal data and cross-sectional data set. Normally, data used for ML application ignores time dependency properties and treat each individual variable as independent. Hence, in order to take time dependency into the ML application, extra methods are required. There are 2 main ways to incorporate time dependency and effect into data set, namely Sliding window (SW) and Forward chaining (FC) that will further be explained.

3.1.1. Sliding window (SW)

Sliding window (SW) is a data processing method that enables the reconfiguration of data set including characteristics of of time dependency into the original data set. SW works by setting a fixed time window which all the data in the window will be considered. There is also a fixed time window for validation purposes. This window then moves along the time line, removing old data and including newer data [61]. In Figure 3.1, a visualisation of sliding window can be seen.

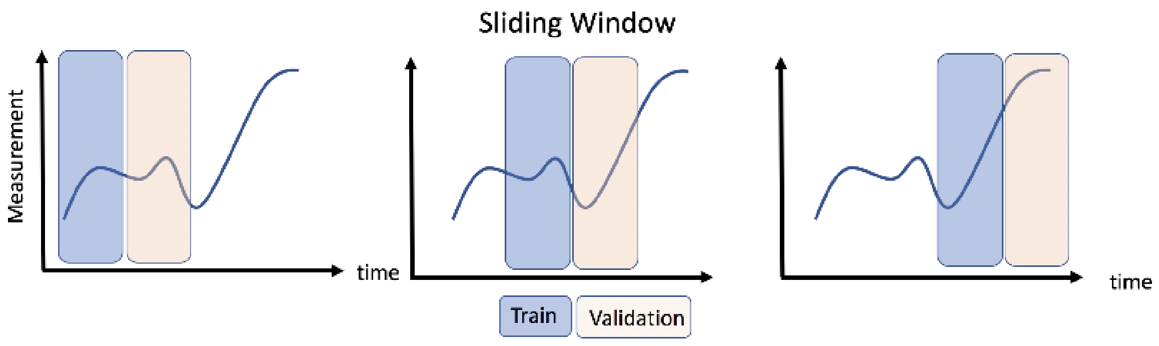


Figure 3.1: A visualisation of the working principle of SW. As time progresses, the fixed train (blue) and validation (yellow) window will move along time to discard old data and include new data for model update. [57].

By applying SW to the data, it allows the data to be constructed in such a way that information from previous data entries are also incorporated. The size of the window (blue section in Figure 3.1) is predetermined. According to Sharon A. Plowman [63], on average, typical injury is recovered within 4 week, thus the size of the window is determined to be 4 weeks.

3.1.2. Forward Chaining (FC)

Forward chaining (FC) is another method to incorporate time dependency into the data set. FC works by constantly increasing the amount of data as time progresses. This is illustrated in Figure 3.2. For FC, we train on the last data points (blue section in Figure 3.2) and validate the prediction on the next data points (yellow section in Figure 3.2), sliding training/validation window through time. In this way, we can estimate the parameters of our model. To test the validity of the model, a block of data at the end of our time series is commonly reserved for testing the model with the learned parameters [57].

FC allows the models to be trained in batches that include more and more data. By doing so, it exposes the model to new information that is related to time dependency of the data. If there is a strong relationship between data variables, time dependency and injury risk status of an athlete, FC should be able to capture this relationship better compared to original data set.

3.2. Machine Learning Concepts

In this section, a general introduction to supervised ML model is discussed in the first part of the section. This is followed by discussion about the concept of underfitting and overfitting in ML applications.

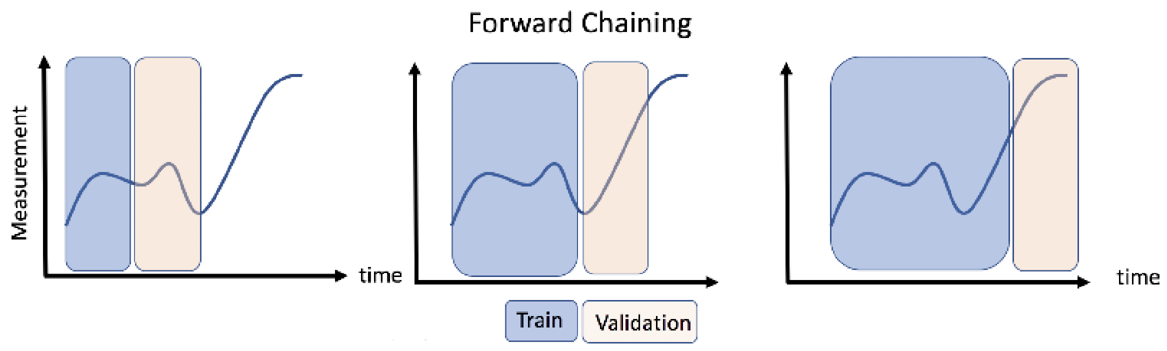


Figure 3.2: A visualisation of the working principle of Forward Chaining. As time progresses, the constant growing training data set will move along time to include new data for model update purpose [57].

3.2.1. Supervised ML Model

Depending on how much human intervention is required to interact with the algorithm, the algorithms can be categorised into supervised, unsupervised and reinforcement learning [35]. Since data was collected through specially designed questionnaire, supervised learning is the most appropriate method to use for this study.

Supervised learning builds a mathematical model that contains both input and output data. By providing input and output, the algorithm can learn and train itself the correlation between input and output variables. Algorithm then should be able to use the correlation information and predict an output given certain input. As more data is fed into the system, the more accurate future predictions can be. [35]

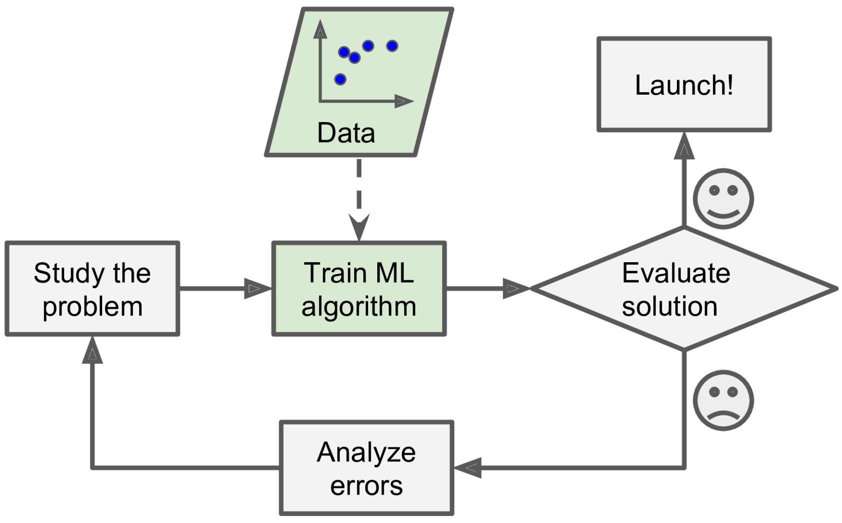


Figure 3.3: Typical Workflow of Machine Learning Approach [35].

The typical workflow of a supervised ML problem is fairly simple, shown in Figure 3.3. Data needs to be properly cleaned and processed before feeding into the algorithm for training purpose. This model can then be used to validate its performance. Errors are then analysed and reviewed to determine what can be done to improve. The model is then adjusted and retrained, and once again, validate the performance. This cycle is performed until the performance results in a convergence or has reached to an optimal balance between performance and efficiency.

3.2.2. Underfitting & Overfitting

When training a ML algorithm, there is always the chance of over-fitting or under-fitting, best shown in Figure 3.4. There are many different parameters that determine if a model will overfit or underfit. These include but not limited to number of variables, set of hyper-parameters of the model, diversity in data set. Under-fitting (left graph of Figure 3.4) means that the model did not learn sufficient information from the data and thus was unable to provide good results. On the other hand, over-fitting (right graph of Figure 3.4) means is trying to reproduce what it has learned rather than predicting based on specific input.

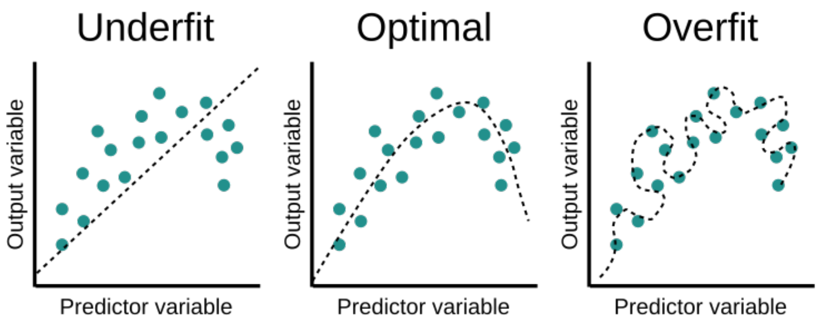


Figure 3.4: A comparison between Underfitting (Left), Overfitting (Right) and Optimal fitting (Middle) of a ML Model [2]. Over-fit tries to fit through all provided data whilst under-fit shows poor performance.

When the model is capable of reproducing the data perfectly, there is a high chance that overfitting has occurred, shown in Figure 3.4. However, when the model is over-fitted, if a new data entry is introduced that is unknown to the model, the model will perform poorly [2]. This is very dangerous since at first sight, the performance of the model might look very good, but it is not reliable. Although optimal fit will produce a less accurate model, it can be easily improved with more data or by tuning the model, this is illustrated with the middle graph of Figure 3.4.

3.3. Data Preparation

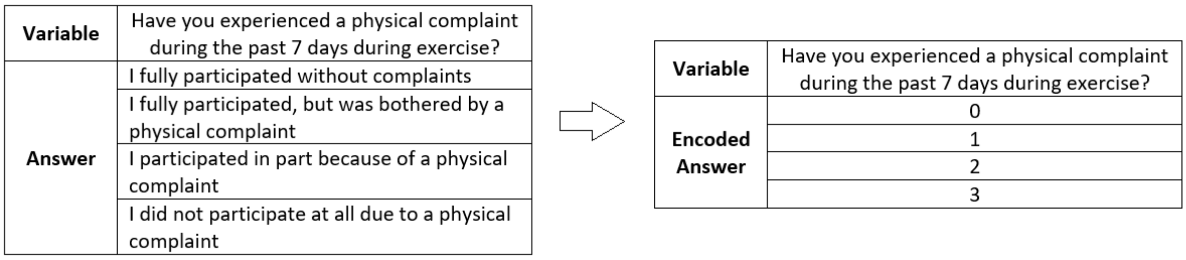
Before applying ML algorithms to the data, the data needs to be properly prepared. This mostly concerns properly encoding the data set such as it can be fed into the algorithms. Since there are only two types of data entries, mentioned previously in chapter 2, the two corresponding encoding methods are ordinal and onehot encoding, discussing in this section.

3.3.1. Ordinal Encoding

Ordinal encoding denotes the conversion of categorical (discrete) features to numerical integers, as discussed in Potdar et al. [56]. This categorical feature can be both string or integers, but require hierarchy between data of the same feature.

For example, feature "Have you experienced a physical complaint during the past 7 days during exercise?" has four discrete, string data entries. After further inspection, it is clear that there is a hierarchy between the data points. Hence they can be ordinary encoded using Figure 3.5.

The same method can be applied to other variables to properly encode all of the features that fit the requirement of ordinal encoding.



Variable	Have you experienced a physical complaint during the past 7 days during exercise?
Answer	I fully participated without complaints
	I fully participated, but was bothered by a physical complaint
	I participated in part because of a physical complaint
	I did not participate at all due to a physical complaint

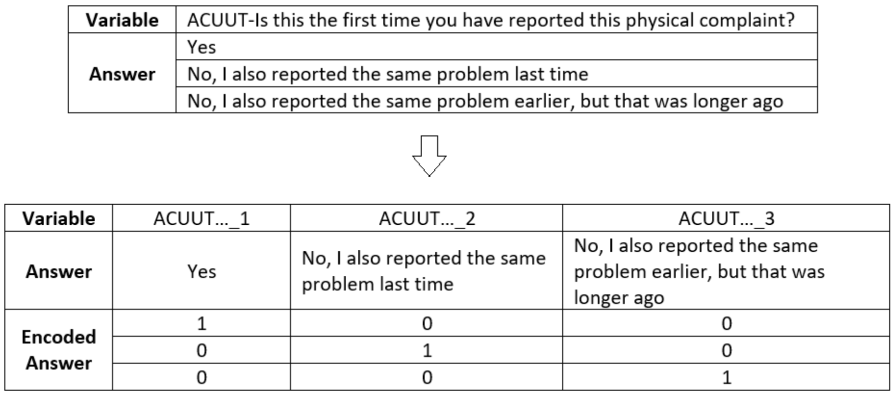
Variable	Have you experienced a physical complaint during the past 7 days during exercise?
Encoded Answer	0
	1
	2
	3

Figure 3.5: Example of application of Ordinal encoding on a set of categorical data set[19]. Table (Left) is raw data from OSTRC data whilst Table (Right) illustrates on how it is encoded. This method works for this type of data since it has data hierarchy.

3.3.2. One Hot Encoding

There are certain variables in the data set that has unique categorical value as entries, usually string values. In this case, it is not possible to apply ordinal encoding method mentioned before. The best way to deal with categorical value is by using one hot encoding[62].

One hot encoding works by converting the categorical data entries with columns of 1 and 0 that represent the same information. An example can be seen in Figure 3.6. Data entries in "Answer" from variable "ACUUT - Is this the first time..." are replaced by columns that represent actual strings in the "Answer". This is illustrated in Figure 3.6.



Variable	ACUUT-Is this the first time you have reported this physical complaint?
Answer	Yes
	No, I also reported the same problem last time
	No, I also reported the same problem earlier, but that was longer ago

Variable	ACUUT..._1	ACUUT..._2	ACUUT..._3
Answer	Yes	No, I also reported the same problem last time	No, I also reported the same problem earlier, but that was longer ago
Encoded Answer	1	0	0
	0	1	0
	0	0	1

Figure 3.6: Example of application of Onehot encoding on a set of categorical data set. Other variables that have unique entries, onehot encoding is used instead of ordinal encoding. Table (Top) represents the raw data from OSTRC whilst Table (bottom) is the same data after encoding. Each column represents each of the unique data entries.

The advantage of using one hot encoding is that it allows conversion of data to something that the computer can understand and operate, in this case, ML application ready. As shown in Figure 3.6, the data entries of variable included in the top table are converted into the values in the bottom table. As for disadvantage, as seen in Figure 3.6, the number of columns depends on the number of unique categorical data. The more different unique data there is, the higher the number of columns will be present after onehot encoding [34]. If there are multiple variables with multiple different distinct answers, the number of columns after One Hot encoding will drastically increase. This can be a problem for very large numbers since it will result in data set having far more variables compared to the number of data entries.

3.4. Artificial Neural Network (ANN)

In this section, models that belong to Deep learning is discussed, namely Artificial Neural Network. It explains how model works, what are the hyper-parameters of models, how to optimise and train.

Artificial Neural Networks are models that are inspired by the structure and/or function of biological brain. Neurons in the brain interconnect with other neurons and form a large network. Given an input from the brain, the neurons process the information and give an output. In that way, given a picture of a dog as an input, brain will process it and classify it as a dog. However, the exact dynamics of the interactions between neurons are still unknown.

Human typically experiences an external excitement, this signal is processed by the neurons in the brain and then a reaction is produced. This excitement and reaction is represented by input and output layer in ANN algorithms and the hidden layer represents the interaction neurons in the brain[26], as shown in Figure 3.7. The interaction between each neuron is randomly assigned as weight. In the hidden layer, a random bias is assigned as well. Given an input, ANN will evaluate through the hidden layer and provide an output. This is the principle of ANN. However, this kind of ANN models have defined static weights for all layers and no matter how much data is used to train, the model will not improve automatically unless the weights are updated manually or through different optimisers.

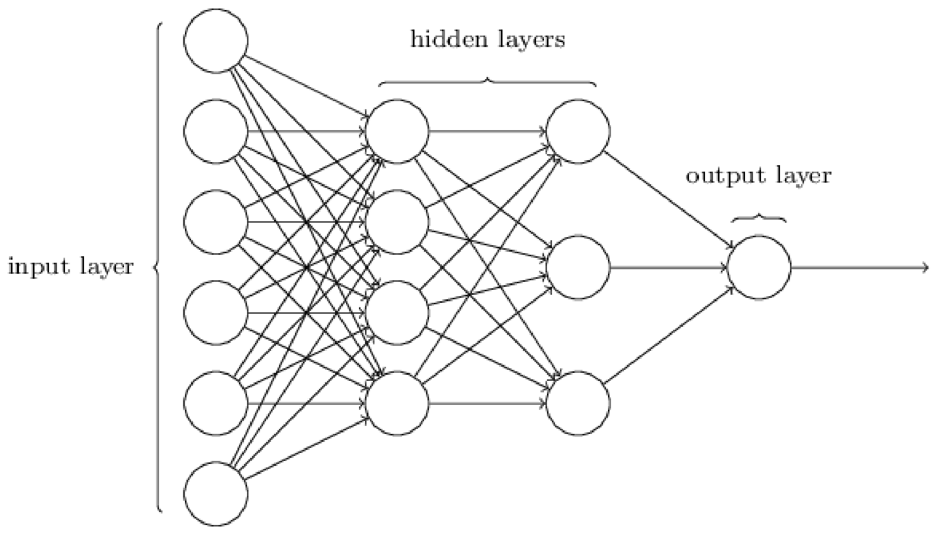


Figure 3.7: Structural illustration of a Feed forward Artificial Neural Network [52]. It includes 3 different layers, input layer, hidden layer and output layer. Input is processed through the hidden layers that provide a output through output layer.

Each node of the hidden layer can be enlarged, like it is shown in Figure 3.8. The main difference that is shown in Figure 3.8 compared to Figure 3.7 is the activation function, this is explained further in subsection 3.4.1.

3.4.1. Activation function

Before going into more detail about ANN, it is important to first understand the activation functions commonly used in neural networks. Even though ANN are modelled after human brain, computer processes still function fundamentally different from a biological brain. In ANN, different weights

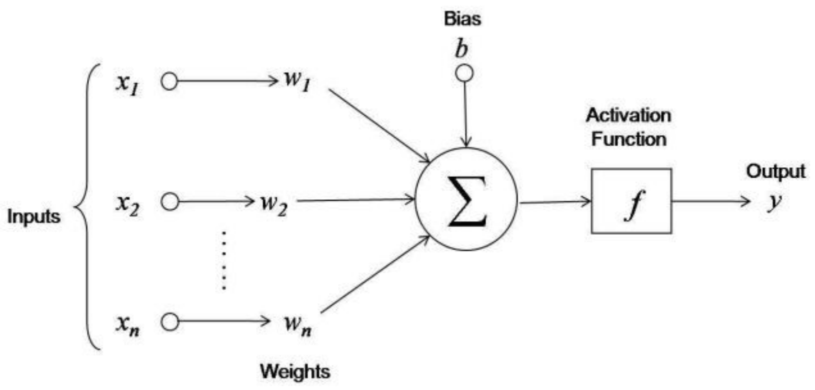


Figure 3.8: Work flow performed by a single neuron in Neural Network[13]. This includes 4 main components, input, weight, activation function and output.

are assigned with different values, illustrated in Figure 3.8.This weight represents the importance of this variable to the model. By slowly changing the weight over time such that the final output is the desired output. This is when an activation function comes into play. By having a activation function, it can check a resultant value after transformation and decide whether or not it would be important. The most common ones are called sigmoid and tanh. More detail of different activation functions are discussed next. [42]

Sigmoid Function

A sigmoid function, shown in Figure 3.9, is a function that can transform the any numerical input to an output that ranges between 0 and 1. This allows the values to be controlled in the neural network. This operation is often used to estimate if a value has contribution to the model or not. If an input value after transformation, outputs a value that is close to 0, that means this value does not contribute to the model and thus its information will not be taken into consideration.

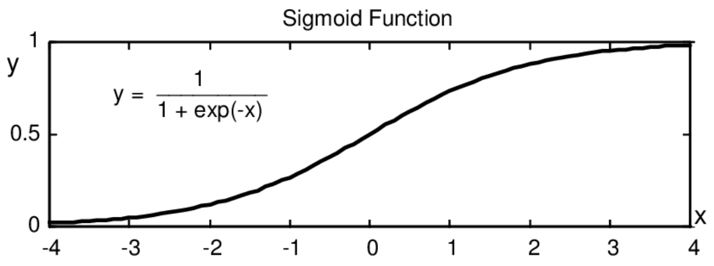


Figure 3.9: Graph of a Sigmoid Function [42]. Sigmoid function always convert input x into a value that lies between 0 and 1.

There are two main disadvantages of using sigmoid function. Firstly, a common problem that occurs in neural network, the vanishing gradient. Vanishing gradient means values that result in little change after passing through the function will have very limited contribution to the model learning process. It can result in the model not being able to learn anything else or requiring long time to converge to a stable and accurate prediction. Sigmoid function is relatively computationally expensive compared to other functions.

Hyperbolic Tangent Function

Hyperbolic Tangent function (tanh), shown in Figure 3.10, converts input data to values between -1 and 1. By limiting the values between -1 and 1, it can allow the model to limit the group of values, thus managing and controlling data flow.

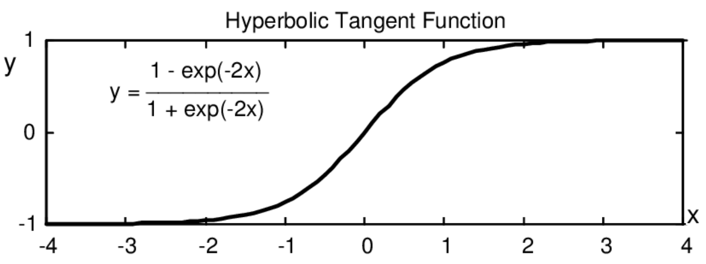


Figure 3.10: Graph of a Hyperbolic Tangent Function (tanh) [42]. Tanh function always convert input x into a value that lies between -1 and 1.

Hyperbolic Tangent function shares similar disadvantages compared to sigmoid function since it works on similar principle. However, since requirements on learning time and computational power is relatively loose, the disadvantages are not as concerning.

Softmax Function

The third most popular activation function is called softmax function [48]. The main advantage of softmax function is its capability of handling multiple classes as other function can only handle one class. The mathematical representation of softmax function can be seen as following:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} for i = 1....k \; and \; z = (z_1, ..., z_k) \in \mathbf{R}^k \tag{3.1}$$

Standard exponential function is applied to each element in z_i of the input vector z and normalise said values by dividing by the sum of all the exponentials. This normalisation ensures that the sum of the components of the output vector ($\sigma(z)$) is 1. The final output through softmax function is the probability of input being in a specific class.

Softmax functions allows for determining the probability of an input belonging to a specific class by producing a value between 0 and 1. One key difference is that it is preferable to apply softmax to the output layer, compared to other activation functions which are applied to input layer.

3.4.2. Optimiser

A ML algorithm assigns different weight factors to the input variable as it continues to learn from the data. How well the model is fitting the data is determined using error (loss) functions discussed later in the chapter. In ML application, a optimiser is often used to determine be best set of parameters that yield the lowest error (loss) function value, hence proving the best performing model [41]. This is different from the hyper-parameter tuning which is discussed later in the chapter. In this section, different optimisers typically used in a ML application are discussed.

SDG

Stochastic Gradient Descent (SGD) is a variation of the basic gradient descent [21]. Gradient descent (GD) is a first-order optimizer that is only dependent on the first order derivative of the loss function. It is then able to compute the direction to the weights needs to shift in order to reach minima. The following equation is used in GD:

$$\theta = \theta - \alpha \cdot \Delta J(\theta) \tag{3.2}$$

Whilst SGD tries to update the model's parameters more frequently compared to normal GD. The algorithm is shown in Equation 3.3, which is a small alteration from the typical GD algorithm.

$$\theta = \theta - \alpha \cdot \Delta J(\theta; x(i); y(i)) \tag{3.3}$$

where $x(i), y(i)$ are the training data set.

Adam

Unlike GD or SGD, Adam works under the principle of momentum. Momentum was introduced to reduce high variance in the SGD optimizers. It can accelerate convergence of the model in the right direction and suppress unwanted fluctuation in other directions. Adam works with first and second order to the momentum. The initial idea is to reduce optimisation speed such as it does not skip minimum.[21]

$M(t)$ represent the mean, which is the first order moment. Whilst $V(t)$ is the second order momentum, stands for uncentered variables of the gradient.

$$\hat{m_t} = \frac{m_t}{1 - {\beta_1}^t} \tag{3.4}$$

$$\hat{v_t} = \frac{v_t}{1 - {\beta_2}^t} \tag{3.5}$$

The update of the parameters are done using the following equation:

$$\theta_{t+1} = \theta_t - \frac{\nu}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{3.6}$$

3.4.3. Building a ANN model

The following describes the basic process of building and applying a ANN with integrated tuning process.

1. Check if all inputs are of correct dimensions and sizes.
2. Initiate empty model constructor.
3. Add input layer with the number of neurons, input size, activation function, kernel initialiser.
4. Add dropout layer to reduce overfitting of the model.
5. Add output layer and specifying the output dimension, and activation function.

6. Compile model with all of the layers initiated before, along with defining loss function, optimiser and performance metrics.
7. Specialise the range each hyper-parameters and create a grip to search for the best performing hyper-parameter set.
8. Specify the model, the hyper-parameter grid to tune and number of cross-validation processes.

After compiling the model, a basic ANN model is ready to be tuned with a grid of hyper-parameters and cross-validation processes.

3.5. Long Short-Term Memory Networks (LSTM)

An alternative ANN algorithm that allows the model to learn from it's error and improve itself is called Back-propagation. This is error is used to evaluate and change the weight and bias. With the changes in the hidden layer, ANN will be able to provide a better prediction next time given the same input.[35]

Back-propagation allows the model to improve itself. In order for this to work with longitudinal data, temporal effects need to be taken into account. A variation of Back-propagation algorithms called back-propagation through time (BPTT) is capable of taking temporal effect into consideration. This kind of algorithm is also called Recurrent Neural Network (RNN)[49].

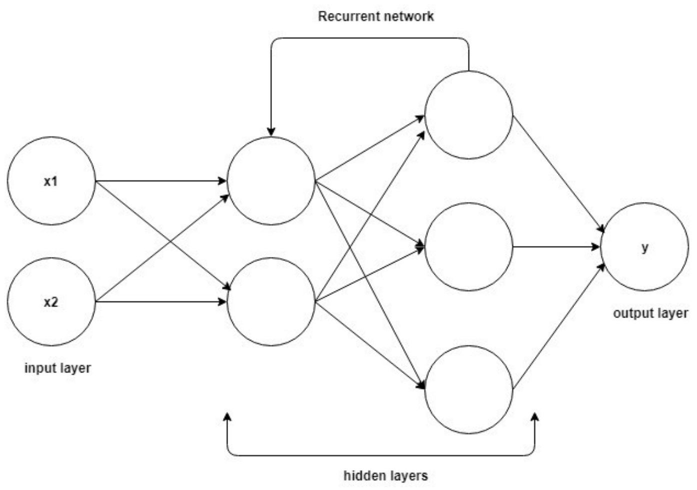


Figure 3.11: Basic structure of a Recurrent Neural Network, including input layer, hidden layers and output layer. Different from ANN model structure, RNN cell has the recurrent network that allows for feedback of information.[49]

RNN allows for a feedback system like back-propagation, but includes certain level of time dependency in the model. RNN retains information from previous time steps and influence the output of current time step. By adding this memory of previous time steps, it assumes that the sequence of data has information as well and can be preserved in the model. As more data are feed into the model, correlation between data at different time steps called "long-term dependencies" can be found. There are other small variations of RNN algorithms based on the same core principle, namely Long-short Term Memory (LSTM) and Convolution Neural Network (CNN).

In this study, it will mainly focus on LSTM version of the model. There are 4 main components to the LSTM cell, namely forget gate, cell state, input gate and output gate, as shown in Figure 3.12[35, 55]. They are called gates because they regulate the flow of information. All of the components in Figure 3.12 can be found in the list below.

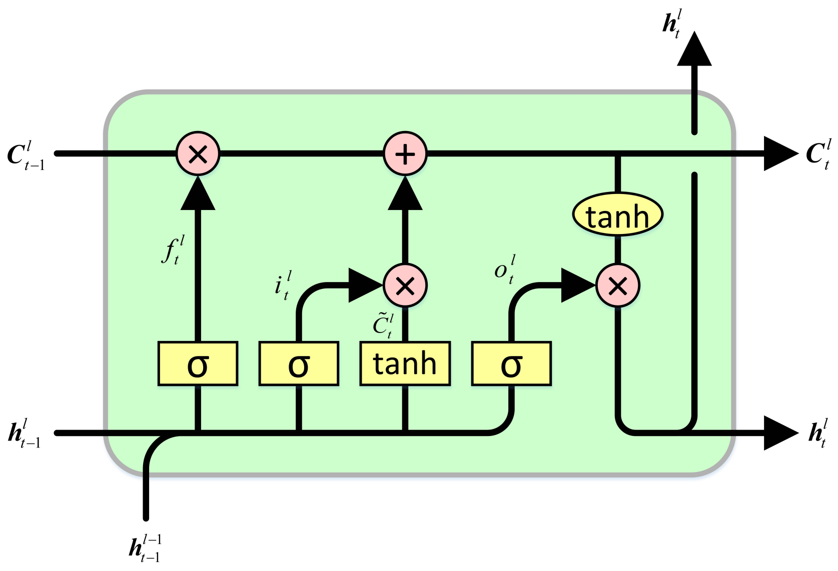


Figure 3.12: Basic Architecture of LSTM Cell, include all the gates, input, output and activation functions [65].

- ${C^l}_{t-1}$ = previous cell state
- ${C^l}_t$ = new cell state
- $\tilde{C}_t^l$ = candidate
- ${f^l}_t$ = forget gate output
- ${o^l}_t$ = output gate output
- ${i^l}_t$ = input gate output
- ${h^l}_{t-1}$ = previous hidden state
- ${h^l}_t$ = new hidden state
- ${h^{l-1}}_{t-1}$ = current input

When input variable ${h^{l-1}}_{t-1}$ is injected into the model, it is often combined with ${h^l}_{t-1}$ and becomes a new input for other gates. This combined input is then first fed through the forget gate. By pass through a sigmoid function, all of the information is converted into values between 0 and 1, labelled as ${f^l}_t$. The same combined input is again fed through both sigmoid and tanh function to produce input gate output (${i^l}_t$) and candidate ($\tilde{C}_t^l$). Feeding through sigmoid function determines what information is important to remember and feeding through the tanh function transforms the value to number between 1 and -1 to regulate the amount of information in the neural network.

To compute the new cell state (${C^l}_t$), the first step is to multiply previous cell state (${C^l}_{t-1}$) with forget gate output (${f^l}_t$). This product is then combined with the product of input gate output (${i^l}_t$) and candidate ($\tilde{C}_t^l$) to determine the new cell state (${C^l}_t$).

The combined input determined previously is again fed through a sigmoid, resulting in output gate output (${o^l}_t$). This output gate output, multiplied by the new cell state that has passed through a tanh function, creates the new hidden state (${h^l}_t$). The new cell state and new hidden state are then used as input for the next cell and the whole cycle begin again.

The cell state (C) is the key component in LSTM. It allows the system to retain some of the previous information and carrying it over to the next cell. As the model is trained with more data, the cell state can be updated to represented real life situation more accurately. [65]

3.5.1. Data Scaling

Since ANN learn the relation between input and output, the scale and distribution of the input data may vary for each feature which will impact the accuracy and performance of the model. Different features could have different unit and/or scale that varies widely from feature to feature.

There are 2 methods of data scaling, namely normalisation and standardisation. Normalisation is a re scaling that converts original scaling to the range of 0 to 1. Normalisation can be calculated using the following equation:

$$y = \frac{x - min}{max - min} \tag{3.7}$$

Standardising involves re-scaling the distribution of the data set so that the mean of observed values is 0 and the standard deviation is 1. This can be computed using the following equation:

$$y = \frac{x - \mu}{\sigma} \tag{3.8}$$

where μ is the mean and σ is the standard deviation of the data set, calculated as following:

$$\sigma = \sqrt{\frac{\sum (x - \mu)^2}{count(x)}} \tag{3.9}$$

3.5.2. Feature Selection

The ability to identify related features in the data set and removing less relevant feature can help to achieve better accuracy for a model. There are two common automated methods used in ML that help to identify the features, correlation analysis and feature importance.

Correlation Analysis

Correlation analysis is a statistical method used to evaluate the strength of relationship between quantitative variables. The value of correlation ranges between 0 and 1. A high correlation means that two or more variables have a strong relationship with each other, whilst low correlation means that the variables have a weak relationship.

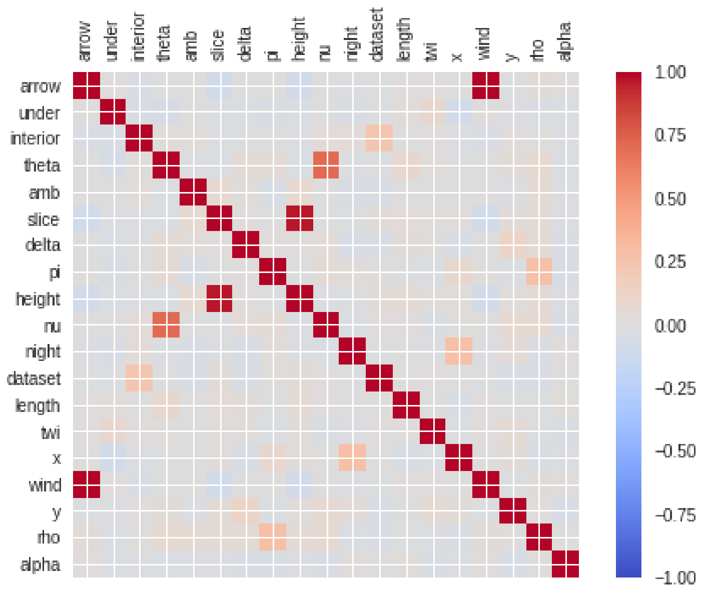


Figure 3.13: Correlation between variables of the data set. X-axis represent variables and y-axis represent the level of important. Scoring higher value in y-axis means it has a larger importance in the model and vice versa[53].

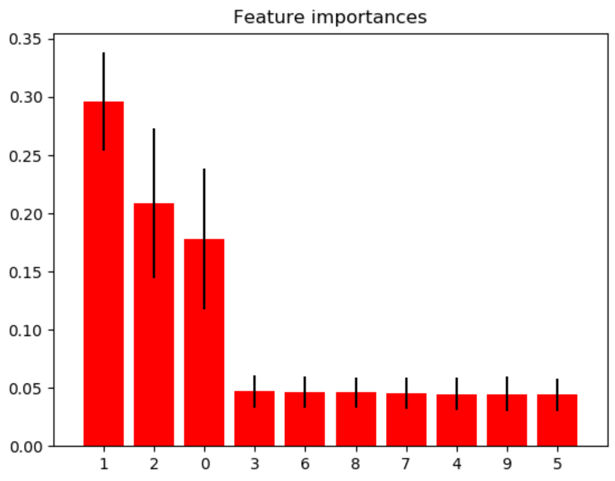


Figure 3.14: Example of Feature Importance when using Random Forest [9]. Horizontal axis represent the variable and vertical axis presents the level of importance. All of variable feature importance add up to 1. The higher a variable's feature importance is, the more this variable contributes to the predictive power of the model.

Feature Importance

Feature importance is a concept introduced in Random Forest (RF), concept discussed in section 3.6. By applying RF to the data set, it will automatically select features that the model classified as the most important [47][35]. By using RF as a method to select the features, it is possible to help increase the performance of the model.

Feature importance provided by Random Forest (RF) can provide a list variables that the model deemed the most important. This can be visualised in Figure 3.14. The model concluded that the first three variables, namely 1,2 and 0 had the highest importance. Whilst all the other variables has very low importance. This means the model even when excluding the lower importance variables from the data set, the model is still able to derive useful information regarding this use case. The discarded variables contribute little to none to the model.

3.5.3. Hyper-parameters & Tuning

Each model has its own set of hyper-parameter that defines the model as a whole. Different combination of hyper-parameters will result in varying performing models. Tuning is the process of selecting the best set of hyper-parameter that yields the best performing results. Tuning process often involves optimising the model parameters based on certain error function. These error functions are discussed in the following section.

Error (Loss) Functions

In order for the model to learn, the model needs to learn the difference between the correct answer and predicted answer, this is where error come into play.

$$J(\omega) = p - \hat{p} \tag{3.10}$$

where $J(\omega)$ is the error function, p is the actual value and $\hat{p}$ is model prediction.

Functions that are used to calculate this error is called loss function. There are different types of loss function and each of them produces different error for the same prediction. The most common ones are Binary Cross Entropy (BCE),Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Binary Cross Entropy (BCE)

Entropy is a theory borrowed from information theory. It measures of the amount of randomness in the information or data. Higher the entropy is, the more random the information is, hence the harder it is to draw any useful information from it.[66]

In ML context, the aim is always trying to match the predicted output distribution to be as close to real output distribution. This means that by minimising the entropy between two data, there are less randomness between the data sets, thus the model is more capable to representing the data. This is called cross entropy loss function. Entropy provides a slightly different way of calculating error compared to other error functions included in this section.

Mean Squared Error (MSE)

MSE is one of the more common ways of calculating error between two variables. It calculates the difference between the predicted output and true output, and squares the error. Squaring the value is done to remove any negative signs, and also provides more weight to larger differences. [43]

$$MSE = \frac{1}{n}[\sum(\hat{y}_i - \tilde{y}_i)^2] \tag{3.11}$$

where:

- n = number of errors
- $\hat{y}_i - \tilde{y}_i$ = error between two variables

There is a variation of the MSE called Root Mean Squared Error (RMSE). As the name suggests, it is simple the root value of MSE, illustrated as following:

$$RMSE = \sqrt{\frac{1}{n}[\sum(\hat{y}_i - \tilde{y}_i)^2]} \tag{3.12}$$

Mean Absolute Error (MAE)

MAE measures the difference between true value and predicted value. Since the desired error, the values are converted to positive with | | signs. Mean of course is the average of all the absolute errors. Relative to other forms of loss functions, MAE is the simplest one of them all.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - y| \tag{3.13}$$

where:

- n = number of errors
- $|y_i - y|$ = the absolute error

Hyper-parameters

Hyper-parameters are parameters that determine the general structure and characteristics of a model. The hyper-parameters of ANN are very similar to LSTM models with minor differences. For LSTM models, in general, there are 6 general hyper-parameters [23], listed below:

- k = number of nodes
- N_i = number of input neurons
- N_0 = number of output neurons
- N_h = number of hidden nodes
- N_s = number of samples in the training data
- α = scaling factor (typically 2 10)

The number of N_h is computed based on other parameters, and is often determined using the following relation:

$$N_h = \frac{N_s}{(\alpha \cdot (N_i + N_0))} \tag{3.14}$$

Dropout layer

There is another hyper-parameter that often comes along with LSTM, namely dropout layer. Dropout layer prevents the model from overfitting by ignoring randomly selected neurons during training. In this way, the model is capable of reducing the sensitivity to the specific weights of individual neurons.

Figure 3.15 illustrates how dropout feature generally works. Dropout will randomly ignore some inputs and some hidden layers, this allows some randomness to be introduced into the model, reducing the chances of the model overfitting. [16]

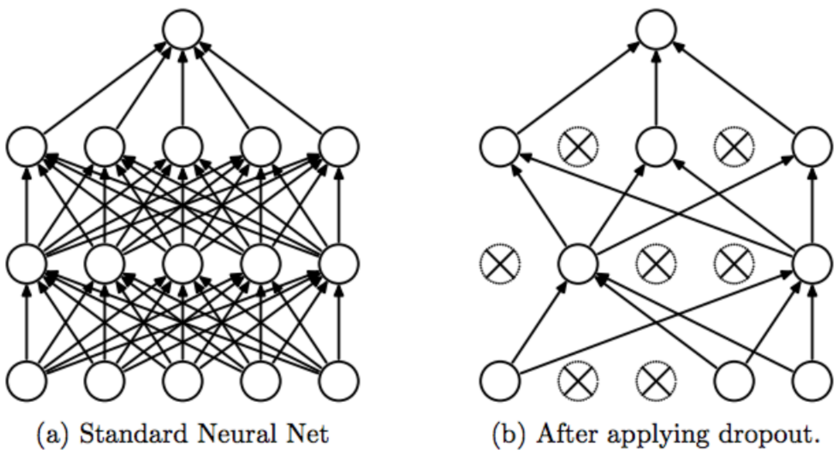


Figure 3.15: Example of dropout feature in Deep Learning algorithms [16]. Dropout allows the model to not include some of the data to allow for more diversity in the data set, reducing the chances of model over-fitting. Figure (a) represents a normal ANN whilst figure (b) is when dropout removes some of the data.

Tuning

To find out the best set of hyper-parameters that results the best results, tuning is required. There are multiple methods that can be used for tuning purposes. The most common ones are random search, grid search and genetic algorithm. In this study, tuning using grid search is used but there are benefits to random search and genetic algorithm that can improve tuning speed.

Random Search

Different set of hyper-parameters can have great impact on the initial performance of the model, since the hyper-parameters determine the characteristics of the model. The simplest way to find out the best set of hyper-parameters is by trying all possible combinations and find out the best performing set.

However, manual searching is not the most efficient way to find suitable hyper-parameters. One common method to perform this task is called random search. Random search creates a grid that contains all the hyper-parameters and allows the model to train on random combination of said parameters.

Grid Search

On the other hand, whilst random search creates a grid and tests on random combination of hyper-parameters, grid search tests on all possible combination of hyper-parameters. Since it goes through all possible combinations, grid search required more time and computational resources to find the best set of hyper-parameters.

Genetic Algorithm

Genetic Algorithms tries to apply natural selection mechanisms to Machine Learning contexts. They are inspired by the Darwinian process of Natural Selection and they are therefore also usually called as Evolutionary Algorithms.[41]

K-Fold Cross Validation

K-fold cross validation is a method commonly used in ML application to reduce the chance of a model overfitting the data, explained in subsection 3.2.2. In K-Fold, we divide our training set into N partitions and then iteratively train our model using N-1 partitions and test it with the left-over partition (at each iteration we change the left-over partition). The model then takes the training results of N times and an average overall training performance is obtained. A small visualisation of this validation method can be seen in Figure 3.16.

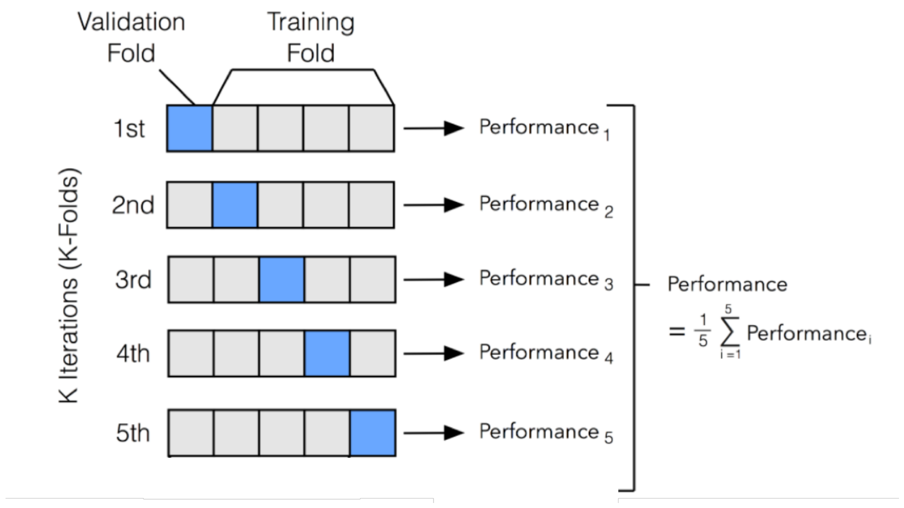


Figure 3.16: Typical illustration of K-Fold Cross Validation [5]. Training and validation is constant rotated to allow for variation in terms of data diversity. This can help with reducing over-fitting.

3.5.4. Building a LSTM model

The following describes the basic process of building and applying a LSTM with integrated tuning process.

1. Check if all inputs are of correct dimensions and sizes
2. Initiate empty model constructor
3. Add input layer and specify neurons and input shape
4. Add output layer and specify dimensions of output prediction
5. Compile model with all of the layers initiated before, specifying loss and optimiser of the model
6. Specialise the range each hyper-parameters and create a grip to search for the best performing hyper-parameter set.
7. Specify the model, the hyper-parameter grid to tune and number of cross-validation processes.

After compiling the model, a basic LSTM model is ready to be tuned with a grid of hyper-parameters and cross-validation processes.

3.6. Random Forest (RF)

Random Forest is a classification ML algorithm consisting of many decisions trees (DT). Unlike DT, RF uses bagging and feature randomness to build individual trees. This way, the model will try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. Bagging is a ensemble algorithm method that allows the model to perform random sampling and introducing replacement data. This allows the model to produce more diverse data set, reducing the chances of the model overfitting.

Decision tree is structured like a flowchart tree structure with different nodes at certain locations, a model of decisions made based on actual values of attributes in the data. There are two types of nodes in the structure, the internal node and the leaf node. The internal node denotes the attribute that is being analysed and branches will extend from the internal node representing different outcomes, and at the end of each branch, a lead node is present to classify the outcome. The first node is called the root node.

Decisions fork in tree structures until a prediction decision is made for a given class. DT are then trained on data for classification or prediction and validated on testing data set. A simple example of decision tree for determining if a person is fit or not can be seen in Figure 3.17.

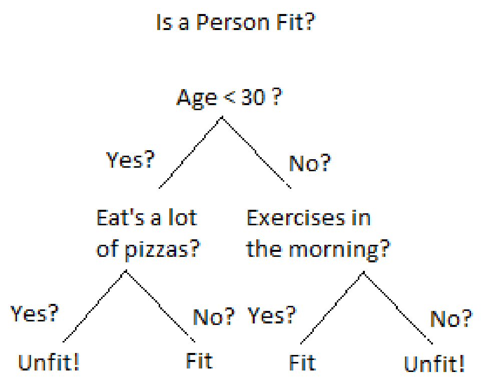


Figure 3.17: Simplified example of a Decision Tree [15]. This decision tree is used to determine is a person is fit or not.

Two types of randomnesses are built into the trees to help create unbiased trees. First, a random sample are taken from the original data to build each tree. Then, at a tree node, a different set of features are selected randomly to determine the best split. This can be seen in Figure 3.18. Given a data set, RF algorithm is split the data set into different trees that have replacement. This means that it is possible that 1 data point were sampled several times. After, each of the tree will act as an individual tree, and tree splitting will occur. At each split of individual trees, data will be subsampled again and only these will be considered for the split. This will further increase the variance of the trees. Finally, end result of each of the classifier will be averaged to produce the final classifier. Although initially, each of the trees have replacements in the data, when averaging at the end, it will eliminate the biases introduced. [15]

RF and Deep learning are based on complete different principles that achieve ML purposes. There are very little a-priori knowledge required to build a RF model. As long as it is known what is input and output to the model, building a RF model is very straight forward. RF models are very popular is different applications due to its simplicity, efficiency and good out of the box performance.

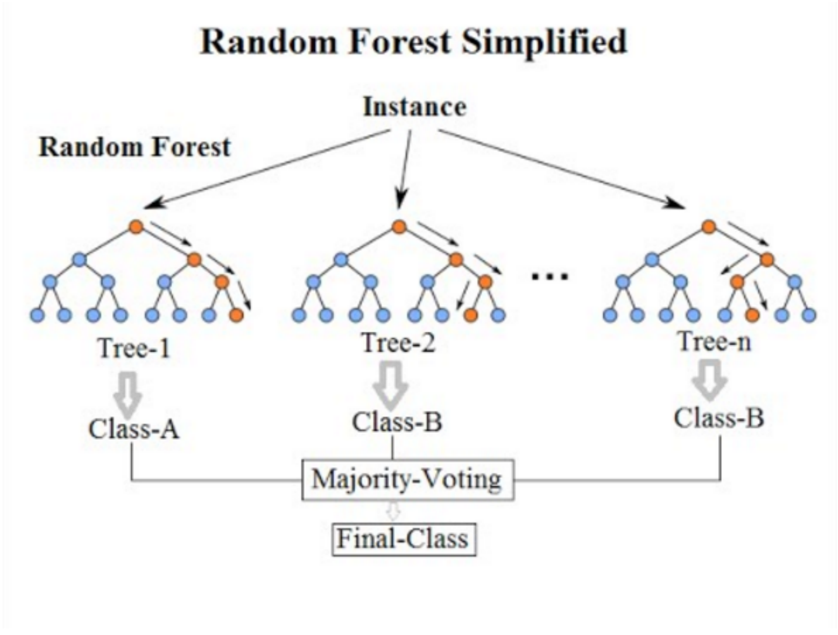


Figure 3.18: General working process of Random Forest [45]. RF consists of multiple different DT using bagging and feature randomness. All of the DT are then passed through a majority voting process to determine the final result.

However, there are some advantages of using RF compared to deep learning. These advantages are:

- The number of hyper-parameters in RF is lower than aforementioned ANN and LSTM. This will result in easier tuning process.
- DT and RF do not require a lot of pre-processing. Data set that includes variables of different scale, magnitude do not require pre-processing to apply RF. This is very advantageous when the data set includes values data entries from different sources with completely different units.
- High out of the box performance compared to other algorithms.

3.6.1. Hyper-parameters & Tuning

In this section, the hyper-parameters and tuning of RF are discussed.

Hyper-parameter

Hyper-parameters are parameters that are predefined before training the model, that determine the details of the model. Every ML model has a different set of hyper-parameters that can be tune. For RF, the list of hyper-parameters can be found in the list below:

- $n\_estimators$:This parameter specifies the total number of trees of the model.
- max_depth: The max_depth specifies the maximum depth of the tree of the model.
- $min\_samples\_split$: The min_samples_split defines the minimum number of samples required to split a leaf node.

- $min\_samples\_leaf$: The min_samples_leaf defines the minimum number of samples required to be at a leaf node.

Since the aim of this model is to be able to predict the injury status identified earlier in the chapter, the main performance that is of interest is accuracy. Of course it is also important to consider the confusion matrix since it can provide insight into the

3.6.2. Building a RF model

The following describes the basic process of building and applying a RF with integrated tuning process.

1. Check if all inputs are of correct dimensions and sizes
2. Initiate empty model constructor
3. Define the set of hyper-parameters used by the RF model.
4. Compile model with the hyper-parameters.
5. Specialise the range each hyper-parameters and create a grip to search for the best performing hyper-parameter set.
6. Specify the model, the hyper-parameter grid to tune and number of cross-validation processes.

After compiling the model, a basic RF model is ready to be tuned with a grid of hyper-parameters and cross-validation processes.

3.7. Performance Metrics

In order to compare the performance of different ML algorithms, performance metric is required. For this study, the most important performance metric to compute from the model is accuracy. High accuracy is desired for a model for this study. On the other hand, derivative product of accuracy can be used to construct confusion matrix. Confusion matrix can often provide insight that accuracy cannot.

Accuracy

Accuracy is the percentage of the items classified correctly against all of the classified item, shown in the following Equation 3.15:

$$Accuracy = \frac{items\ classified\ correctly}{all\ items\ classified} \tag{3.15}$$

3.7.1. Confusion Matrix

Confusion matrix is a matrix that is composed of 4 different components, shown in in Figure 3.19 and in the following list:

- TP = True Positive
- FP = False Positive
- TN = True Negative
- FN = False Negative

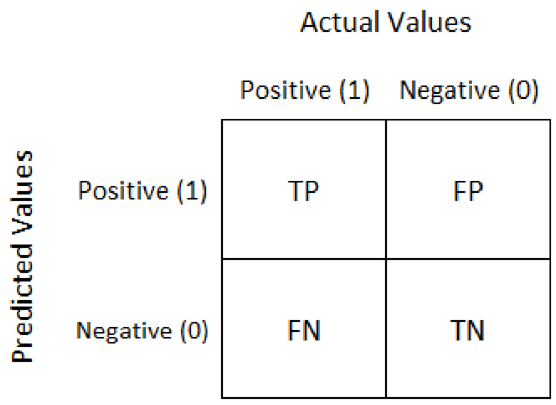


Figure 3.19: Confusion Matrix values including TP, FP, FN and TN.[51]

With the 4 different components, two important metrics can be computed, namely recall and precision. Recall measures the proportion of true positives that are correctly identified by the classifier[58]. Equation of recall is shown below:

$$Recall = \frac{TP}{TP+FN} \tag{3.16}$$

Whilst precision represents how many were classified out of all the ones that were classified as positive. The equation is shown below as:

$$Precision = \frac{TP}{TP+FP} \tag{3.17}$$

3.8. Input Data Structure Configuration

In this section, the reconfiguration of the input data structure is discussed. The structure of the input data set is changed to better represent real life scenario. As stated before, the aim of this study is to investigate the possibilities of using ML techniques to predict injury risk of athlete's using longitudinal data. A typical data entry in the data set is shown in Figure 3.20, with every row representing a sequential time step data entry. The new data structure consists of data from two different time steps, current and previous one.

As Figure 3.20 shows, after structure reconfiguration, the number of input variables increases. The first three variables of current time step are attached to the end of the previous data entry. These three variables correspond to the amount of hour an athlete has spend in strength & fitness, competition and training. There is one important assumption that is required in order for this reconfiguration to make structural sense.

Assumption: The first three variables are treated as planned training schedule by the trainer at the beginning of every week and all athletes stick strictly to this plan.

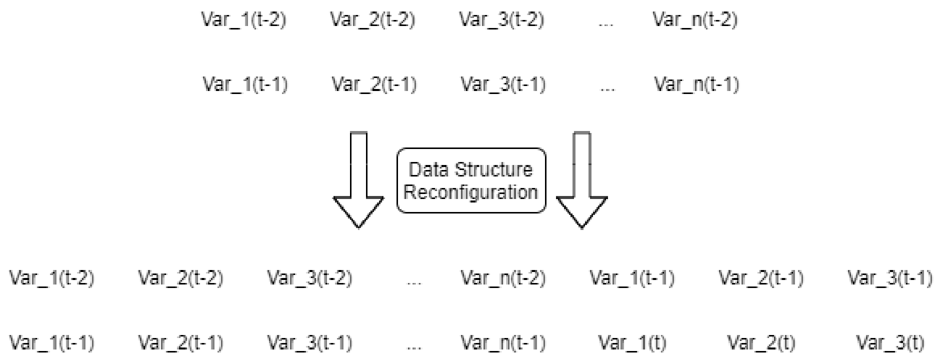


Figure 3.20: Input Data set Structure Reconfiguration of the original data set. Original data set (top) is converted to the new format (bottom) to better present all variables available at the beginning of each week. At the beginning of every week, not only OSTRC from previous week(s) are available, but the training program determined by a trainer is also known, hence it needs to be included into the same data entry.

This data structure allows for an easier integration when only the first 3 variables are available to be included as input to classify injury risk of the coming week. This is due training schedule is always available at the beginning of the week, along with OSTRC data from previous week(s). By including the training schedule of the coming week, the impact of different training schedule at different time step can be included during the training of the models.

4
Result

In this chapter, the results of ANN, LSTM and RF on OSTRC data set is presented. Each algorithm will be trained on data with 3 differently processed data set, no time dependency integration, SW and FC. All models will then be validated using testing data set.

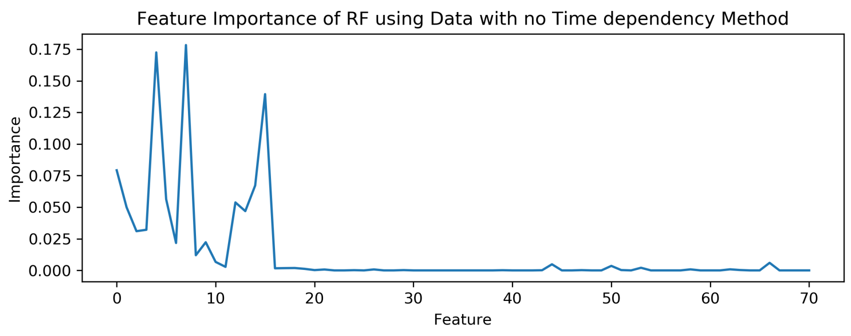
In order to verify the results, confusion matrices were used to provide more in depth insight to the performance of the model. Precision is the number of true positives predictions that actually belong to the positive class. Recall quantifies the number of positive class predictions made out of all positive entries in the data set. Accuracy is the final measurement that can reflect the performance of the model.

As stated in chapter 2, raw data has attributes that assign each data entry to specific athlete. By eliminating such attribute and treat the whole data set as anonymous and making the assumption of all data belong to the same athlete, a generalised model can be constructed. This generalised model can present the characteristics of the whole data set. In this section, the results of generalised model are presented.

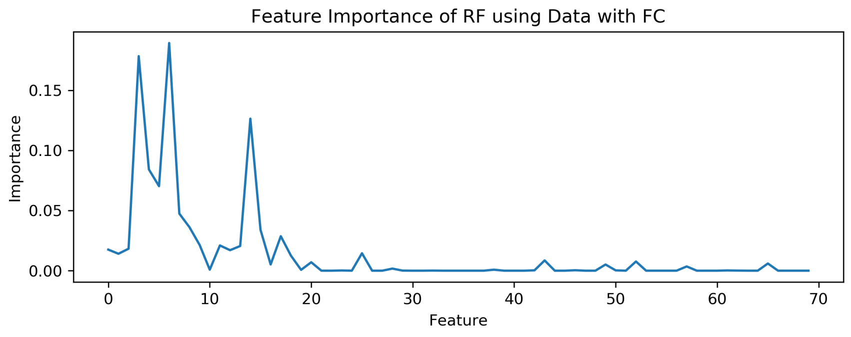
4.1. Feature Importance

This sections shows the results of feature selection on the training data. Figure 4.1 displays feature importance distributions of RF using different data sets. This includes the level of importance of all variables. The number of features (variables) in RF with SW is much higher than other 2 methods. This is due to the SW method that expands the data set by encoding values from different time steps into 1 data entry.

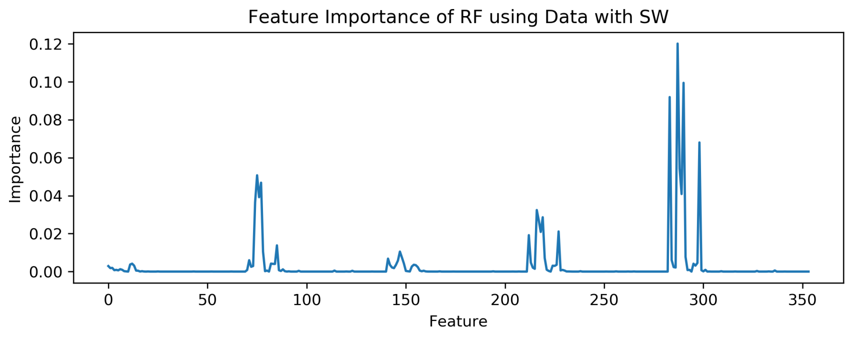
With these three feature importance distributions included in Figure 4.1 , the distinction of important feature used in different models can be discovered. Since normal RF fits the training data that has yet to be treated with time dependency implementation methods, this can serve as a basis for the other two methods to compare to. To begin with, the feature importance between normal RF and RF model using Data with FC is compared. As Figure 4.1 shows, both models identified most high importance features to be before feature 20. Although there are minor differences between important feature in both models, the general trend is similar. RF identified high importance features, like features 5, 8, 13, 15 and 16. Whilst RF-FC identified high importance features to be 4 to 7 and 15.



(a) RF with no time dependency method



(b) RF with FC



(c) RF with SW

Figure 4.1: Feature importance of all 70 features for RF models. The top figure includes Rf model with no time dependency data, middle figure includes feature importance of RF with FC method and the last figure represents feature importance of RF with SW method.

- *Feature 4*: Have you experienced physical complaint during the past 7 days during exercise?
- *Feature 5*: To what extent have you adjusted your training or participation in competition in the past 7 days due to this physical complaint?
- *Feature 6*: To what extent have you noticed in the past 7 days that this physical complaint has affected your performance?
- *Feature 7*: To what extent did you experience the symptoms of this physical complaint in the past 7 days?
- *Feature 8*: How many days in the last week have you been unable to participate fully or completely in training or competitions due to this physical complaint?
- *Feature 13*: How energetic do you fell at the moment?

- *Feature 15*: Does the physical complaint that you have experienced pain or discomfort in the past 7 days concern an injury or illness_0?
- *Feature 16*: Does the physical complaint that you have experienced pain or discomfort in the past 7 days concern an injury or illness_1?

Unlike RF and RF-FC, since RF-SW uses a different method to treat the data set with time dependency, the resulting number of features in the data after data pre-processing is different from other methods. As shown in Figure 4.1, there are a total of 355 resulting features in RF-SW model, example five times the original number of features. This means that SW method implemented previous 4 weeks of data and current week into one big data set for the model to process. In this case, the model concluded that features 74 to 77, 216 to 219. 283, 287 to 290 and 298 to be of importance. The correspondence table that describes the relation between these features and original features can be found below:

- *Feature 74*: Have you experienced physical complaint during the past 7 days during exercise? (t-3)
- *Feature 75*: To what extent have you adjusted your training or participation in competition in the past 7 days due to this physical complaint? (t-3)
- *Feature 76*:To what extent have you noticed in the past 7 days that this physical complaint has affected your performance? (t-3)
- *Feature 77*: To what extent did you experience the symptoms of this physical complaint in the past 7 days? (t-3)
- *Feature 216*: Have you experienced physical complaint during the past 7 days during exercise? (t-1)
- *Feature 217*: To what extent have you adjusted your training or participation in competition in the past 7 days due to this physical complaint? (t-1)
- *Feature 218*: To what extent have you noticed in the past 7 days that this physical complaint has affected your performance? (t-1)
- *Feature 219*: To what extent did you experience the symptoms of this physical complaint in the past 7 days? (t-1)
- *Feature 283*: Final Score (t-1)
- *Feature 287*: Have you experienced physical complaint during the past 7 days during exercise? (t)
- *Feature 288*: To what extent have you adjusted your training or participation in competition in the past 7 days due to this physical complaint? (t)
- *Feature 289*: To what extent have you noticed in the past 7 days that this physical complaint has affected your performance? (t)
- *Feature 290*: To what extent did you experience the symptoms of this physical complaint in the past 7 days? (t)
- *Feature 298*: Does the physical complaint that you have experienced pain or discomfort in the past 7 days concern an injury or illness_0?

In the case of models that use SW, a new data entry consists of multiple time step data entries. This is represented in the data-set as *t, (t-1), (t-2), (t-3)*, meaning data entries at different time steps. Whilst *t* represents current time step and *(t-3)* represents a lag of 3 time steps.

Looking at the feature importance from all models, it is clear that there are very few variables that contribute to the final model. These models are mostly connected to the 4 OSTRC questions, and previous injury risk status (F score, FS score). Whilst other variables contribute very little to the model. By removing some of the variables that contribute little to the model, it can increase model tuning, training and testing time whilst maintaining good model performance.

4.2. Artificial Neural Network (ANN)

Since ANN is the most basic version of a deep learning algorithm, ANN can serve as baseline in terms of accuracy and performance for other models to compare to.

4.2.1. Hyperparameters

In python and scikit-learn environment, an ANN models is defined with 10 different hyperparameters. These hyperparameters are shown in Table 4.1.

Table 4.1: Hyperparameter Values for ANN Model optimisation

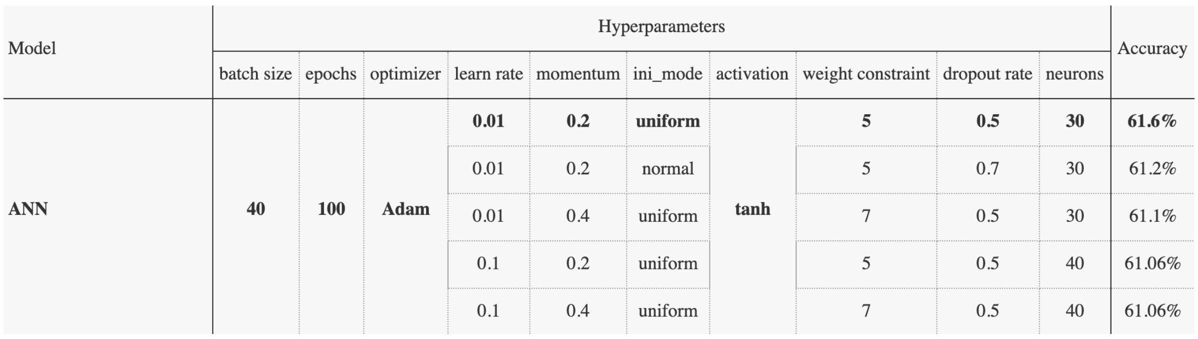
Hyperparameter	**Values**
Batch Size	40 , 80, 100
Epochs	50 ,100
optimiser	SGD, Adam, Adamax
Learning Rate	0.01, 0.1
Momentum	0.0, 0.2, 0.4
Init_Mode	uniform, normal, zero.
Activation	softmax,tanh, sigmoid
Weight Constraint	5, 7
Dropout rate	0.5, 0.7
Neurons	25, 30, 40

With all of the hyperparameters, a total of 11664 different combinations are possible. The optimisation process will go through all of the hyperparameters combination and find out which one has the highest performance metric, accuracy. It took Grid search a total of 407.3 min, 530.8 min and 3024.2 min for ANN, ANN-SW and ANN-FC respectively to optimise the hyperparameters.

4.2.2. ANN - No time dependency integration

As mentioned in chapter 2, there are 2 main ways to prepare data such that it could possibly retain time dependency in the data, SW and FC. By allowing the model to train on data set that has not been transformed using SW and FC method, it can serve as baseline to understand the performance, advantages and disadvantages of applying the other two methods.

All of the hyper parameter sets that resulted in the top 5 accuracy in Figure 4.2 were considered. These hyperparameters are then used to rebuild models and tested on a small set of data isolated from the train data set. Accuracy of each of these models are again compared to selected the hyper-parameters that results in the best accuracy. This set of hyper parameter is then used for classification in the next section.



Model	Hyperparameters										Accuracy
	batch size	epochs	optimizer	learn rate	momentum	ini_mode	activation	weight constraint	dropout rate	neurons	
				0.01	**0.2**	**uniform**		**5**	**0.5**	**30**	**61.6%**
				0.01	0.2	normal		5	0.7	30	61.2%
ANN	**40**	**100**	**Adam**	0.01	0.4	uniform	**tanh**	7	0.5	30	61.1%
				0.1	0.2	uniform		5	0.5	40	61.06%
				0.1	0.4	uniform		7	0.5	40	61.06%

Figure 4.2: Top 5 performing ANN models for each condition, their respective set of hyperparameters and accuracy.

As shown in Figure 4.2, the tuning process concluded that batch size of 40, epochs of 100, "Adam" optimiser and activation function of "tanh" were shared commonality between all models. The end accuracy of all models were very close, with the highest of 61.6 % and lowest at 61.06%. The model with the highest accuracy had 0.01 learning rate, 0.2 of momentum, "uniform" ini mode, weight constraint of 5, dropout rate of 0.5 and 30 neurons.

After further inspection, it is concluded that all models outputted prediction of class 0 & 1 and was not able to provide any other class predictions. Since the majority of the actual values are 0 & 1, keep predicting value 0 & 1 would yield a high accuracy. This means that ANN did not learn any important information from the data set. However, this might not be the case, and the same set of parameters will still be used in the classification for further inspection on the model.

Table 4.2: Confusion table of the best performing ANN model with no time dependency method

Training Time: 8.07 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	276	101	15	4	1	0	0.695
	1	0	69	47	20	9	3	0.466
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		1	0.406	0	0	0	0	

Accuracy = 63.3%

The results of ANN using best hyper parameter set is included in Table 4.2. This classifier has an end accuracy of 63.3%. Confusion matrix of this shows that the model was only capable of making predictions of class 0 and 1. This is indicated by the zeros values in precision and recall of class 2,3,4 and class 5. Having 0 recall for classifying class 2 means that none of the class 2 were actually predicted by the model as class 2. Whilst having 0 in precision shows that of all the class predicted by the model, none were actually class 2. This characteristic shows up again for classes 3,4 and 5. Further proving that the model was not able to derive useful information from the data set and yielded to predicting the most frequent value. The model is simply guessing the most frequency value. Figure 4.3 shows all of the predicted classes versus the actual classes in the test data set. It shows that the model predicted all classes as class 0 and 1.

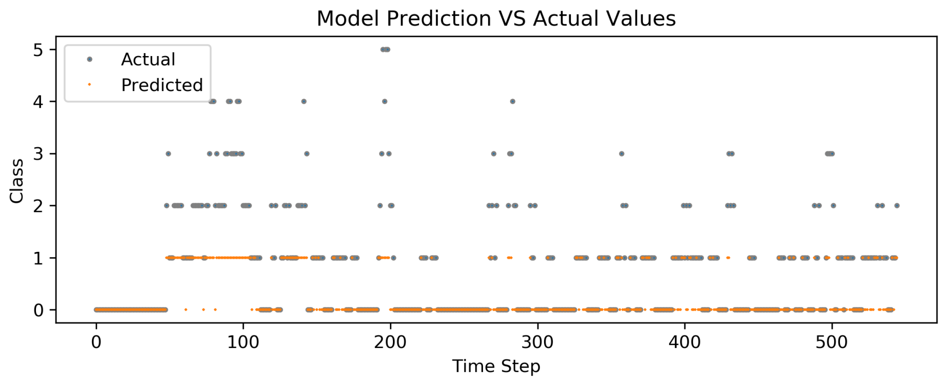
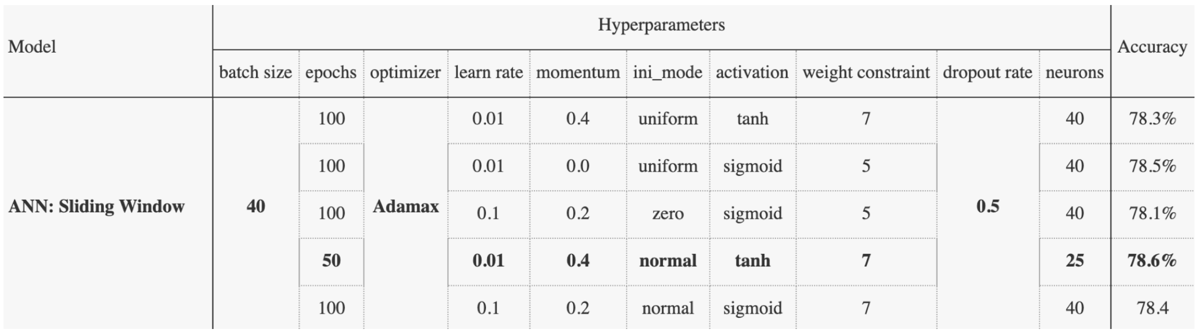


Figure 4.3: ANN Model Predictions without Time dependency method VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.2.3. ANN - SW

The results of ANN model without time dependency method on the data set were not ideal. The model was not capable of learning from the data set and perform prediction on the test data. However, by applying SW to the data, some of the time dependency can be incorporated into the data set that could change the result.



Model	Hyperparameters										Accuracy
	batch size	epochs	optimizer	learn rate	momentum	ini_mode	activation	weight constraint	dropout rate	neurons	
ANN: Sliding Window	**40**	100	**Adamax**	0.01	0.4	uniform	tanh	7	**0.5**	40	78.3%
		100		0.01	0.0	uniform	sigmoid	5		40	78.5%
		100		0.1	0.2	zero	sigmoid	5		40	78.1%
		50		**0.01**	**0.4**	**normal**	**tanh**	**7**		**25**	**78.6%**
		100		0.1	0.2	normal	sigmoid	7		40	78.4

Figure 4.4: Top 5 performing ANN-SW models for each condition, their respective set of hyperparameters and accuracy.

The same optimisation method was applied to models using SW processed data set and top 5 models are shown in Figure 4.4. Unlike ANN models with no time dependency method, models using SW to retain time dependency showed different sets of hyper-parameters yielded varying level of accuracy. The best set of hyper parameter achieved accuracy of 78.6%. By applying this set of hyper parameter on ANN model and validating on test data, the following results were obtained and included in Table 4.3.

Although ANN and SW showed promising results during training phase and accuracy of 80.59% using test data set, results on the testing data set in Table 4.3 shows otherwise. ANN and SW method produced similar results to ANN model without any time dependency method. Both models predicted all values as 1 hence having 1 in both precision and recall for classifying 1. However, all other recall and precision are zero. This can be seen in the graph of predicted classes versus actual data

Table 4.3: Confusion table of the best performing ANN model using SW method

Training Time: 10.54 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	206	0	0	0	0	0	1
	1	6	170	62	24	10	3	0.618
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		0.972	1	0	0	0	0	

Accuracy = 80.59%

set in Figure 4.5. The conclusion of the model not being able to properly learn useful information from the data can be obtained.

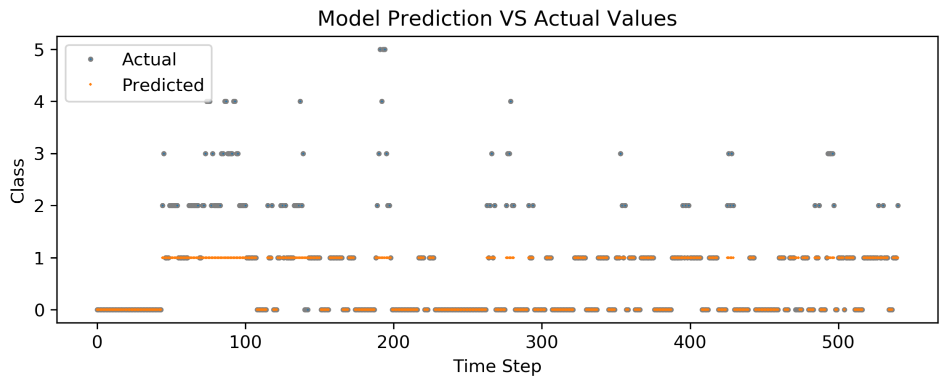
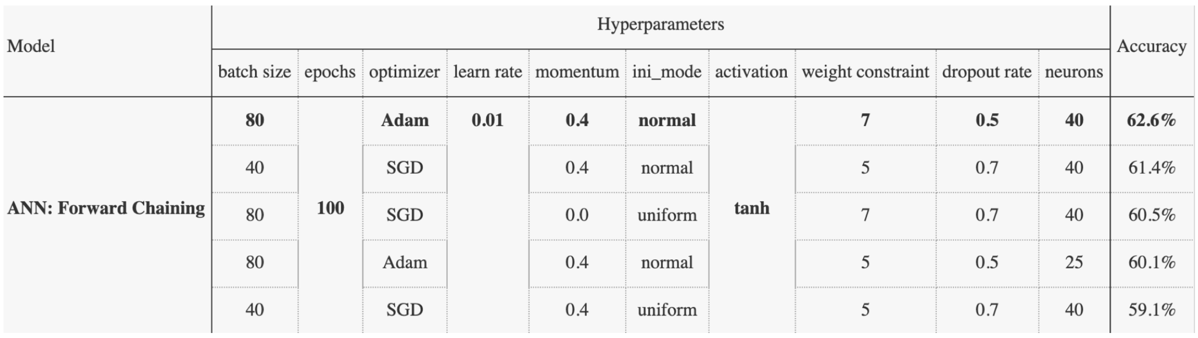


Figure 4.5: ANN-SW Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.2.4. ANN - FC

The second method to integrate time dependency into data set is FC method. By applying FC to the data set, feeding it through ANN and performing hyper parameter optimisation, the top 5 sets of hyper parameters were included in Figure 4.6. The best model achieved accuracy of 62.6% and the lowest is 59.1%.

On average, ANN-FC achieved similar results compared to normal ANN model. With the lowest model performing slightly worse than ANN's lowest model. However, results of ANN-FC model on test data set tells a different story. The model with the best set of hyperparameters did not show similar performance on testing data set. The results of the model on testing data can be seen in Table 4.4. The final accuracy of ANN-FC is 31.2%. In this case, model predicted all values to be as class 1. The model was not able to classify any predictions as any other classes other than class 1. Once again, the model was not able to derive useful information from the data set to perform proper prediction, as shown in Figure 4.7.



Model	Hyperparameters										Accuracy
	batch size	epochs	optimizer	learn rate	momentum	ini_mode	activation	weight constraint	dropout rate	neurons	
	80		**Adam**	**0.01**	**0.4**	**normal**		**7**	**0.5**	**40**	**62.6%**
	40		SGD		0.4	normal		5	0.7	40	61.4%
ANN: Forward Chaining	80	**100**	SGD		0.0	uniform	**tanh**	7	0.7	40	60.5%
	80		Adam		0.4	normal		5	0.5	25	60.1%
	40		SGD		0.4	uniform		5	0.7	40	59.1%

Figure 4.6: Top 5 performing ANN-FC models for each condition, their respective set of hyperparameters and accuracy.

Table 4.4: Confusion table of the best performing ANN model using FC method

Training Time: 6.03 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	0	0	0	0	0	0	0
	1	259	156	61	20	4	0	0.312
	2	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		0	1	0	0	0	0	

Accuracy = 31.2%

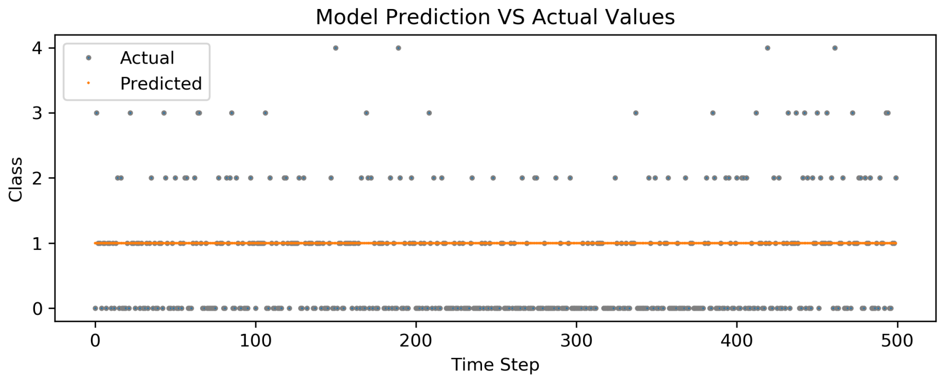


Figure 4.7: ANN-FC Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.3. Long-short Term Memory (LSTM)

As stated in chapter 3, LSTM is a variation of ANN that incorporates the mechanism to preserve memory, thus being able to use previous experience to improve classifier performance. Performing LSTM on data set that is minimally processed will serve as baseline for the other two methods to compare to.

4.3.1. Hyperparameters

For LSTM, a total of 7 different hyper-parameters are available for tuning. These hyper-parameters are included in Table 4.5.

Table 4.5: Hyper-parameter Values for LSTM Model Optimisation

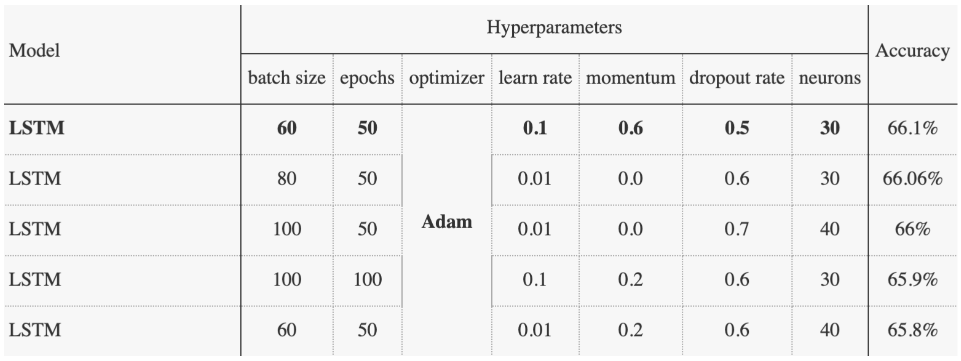
Hyperparameter	**Values**
Batch Size	40, 60, 80, 100
Epochs	50, 100
Optimiser	SGD, Adam, Adamax, Adagard
Learning Rate	0.01, 0.1, 0.2
Momentum	0.0, 0.2, 0.4, 0.6
Dropout rate	0.5, 0.6, 0.7
Neurons	25, 30, 40

Total number of possible combination of hyper-parameters is 3456. Each model require a new tuning process to determine the best set of hyper-parameters. For LSTM with no time dependency method required a total of 323.8 minutes to tune the model. LSTM with SW required 551.7 minutes and LSTM with FC required 2970.1 minutes to optimise.

4.3.2. LSTM - No time dependency integration

Processing LSTM using data that has no time dependency method applied will serve as baseline for other LSTM models to compare to. The top 5 models with the best set of hyper-parameters are included in Figure 4.8. In this case, only the necessary pre-processing will be applied to the data before LSTM training. Hyper-parameters of LSTM without time dependency method showed great diversity and range. However, the tuning process concluded that "Adam" is the best hyper-parameter compared to other optimisers. Overall, there is very small changes in model accuracy, with the highest of 66.1% and lowest of 65.8%. The set of hyper parameters with the highest accuracy was selected moving forward to the next phase.

Confusion matrix of LSTM model with testing data showed similar results compared to training phase, included in Table 4.6. The model achieved model accuracy of 64.34%, slightly lower than average LSTM model accuracy included in Figure 4.8. The model managed to achieve high recall of 0.996 for classifying class 0. Only 1 class 1 was misclassified as class 0. Whilst of 396 that were predicted as class 0, 274 were correctly classified, resulting in a precision of 0.692. There are 15 of class 2, 4 of class 3 and 1 class 4 were wrongly predicted as class 0. For class 1, recall dropped to a low value of 0.218, with a slightly higher precision of 0.822. Out 170 actual class 1's, only 37 were correctly predicted. Most of the class 1's were classified as class 0 instead and 37 were classified as class 2. However, of the 45 classification of class 2, 37 were actually class 2, with 6 predictions



Model	Hyperparameters							Accuracy
	batch size	epochs	optimizer	learn rate	momentum	dropout rate	neurons	
LSTM	**60**	**50**	**Adam**	**0.1**	**0.6**	**0.5**	**30**	66.1%
LSTM	80	50		0.01	0.0	0.6	30	66.06%
LSTM	100	50		0.01	0.0	0.7	40	66%
LSTM	100	100		0.1	0.2	0.6	30	65.9%
LSTM	60	50		0.01	0.2	0.6	40	65.8%

Figure 4.8: Top 5 performing LSTM models for each condition, their respective set of hyperparameters and accuracy.

belonging to class 2. Recall for class 2 increased to 0.565, whilst precision dropped to 0.385. There is a slightly larger distribution in terms of predicted classes compared to previous classes. In total, 62 actual class 2's , 35 were correctly predicted as class 2, 15 were predicted as class 0, both class 1 and 3 had 3 predictions. Precision of class 3 decreased slightly to 0.333 and recall dropped to 0.167. Most of actual class 3's were misclassified as class 2s'. The poor performance is also reflected on lower precision, concluding that most of the predicted class 3s', half were misclassified as class 2s'. Both class 4 and class 5 received precision and recall of zero. The model were not able to predict any of the class 4 and 5 correctly. All of the predictions made by the classifier versus actual test data can be seen in Figure 4.9.

Table 4.6: Confusion table of the best performing LSTM model with no time dependency method

Training Time: 9.78 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	274	102	15	4	1	0	0.692
	1	1	37	6	1	0	0	0.822
	2	0	31	35	15	9	1	0.385
	3	0	0	6	4	0	2	0.333
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		0.996	0.218	0.565	0.167	0	0	

Accuracy = 64.34%

Figure 4.9 provides a visualisation of the model prediction versus actual values. Most of the class 0 were correctly predicted, which is reflected on the confusion matrix as well. The lower recall of class 1 is also represented by lack of correct predictions (orange). As included in Table 4.6, both class 4 and 5 had zero precision and recall, which is illustrated in the figure as well.

4.3.3. LSTM - SW

Although LSTM already has built in functionality that allows for memory retention, applying SW to the data set allows for certain time dependency to be preserved within the data set. Figure 4.10 shows a diverse set of hyper parameters that all achieved high level of accuracy. The best model achieved a accuracy of 87.2% whilst lowest achieved accuracy of 85.7%. The set of hyper parameter with the highest accuracy level will be used to rebuild a new model and validate the model using test data set.

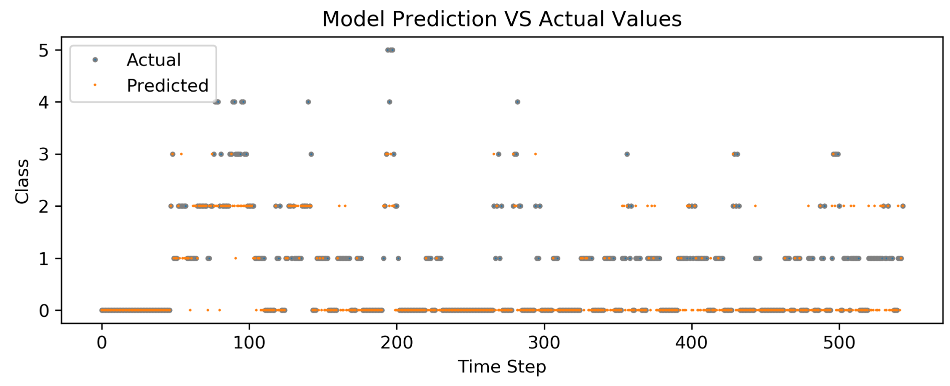
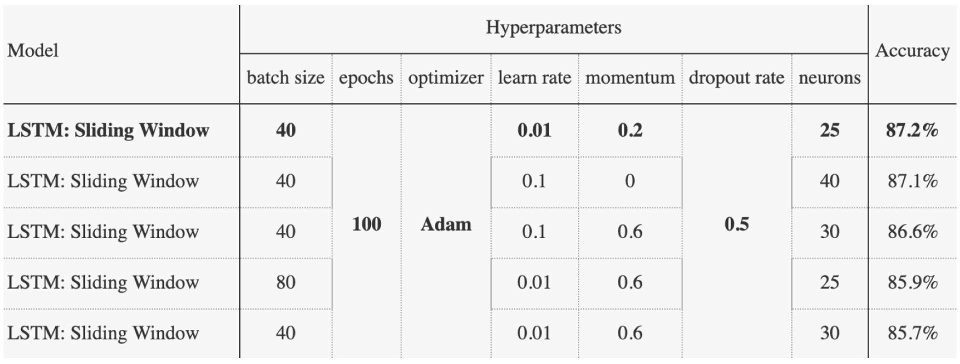


Figure 4.9: LSTM Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.



Model	Hyperparameters							Accuracy
	batch size	epochs	optimizer	learn rate	momentum	dropout rate	neurons	
LSTM: Sliding Window	**40**	**100**	**Adam**	**0.01**	**0.2**	**0.5**	**25**	**87.2%**
LSTM: Sliding Window	40			0.1	0		40	87.1%
LSTM: Sliding Window	40			0.1	0.6		30	86.6%
LSTM: Sliding Window	80			0.01	0.6		25	85.9%
LSTM: Sliding Window	40			0.01	0.6		30	85.7%

Figure 4.10: Top 5 performing LSTM-SW models for each condition, their respective set of hyperparameters and accuracy.

Confusion matrix of LSTM-SW in Table 4.7 shows promising results. For both class 0 and 1, recall and precision achieved very high values, (0.989, 1) for class 0 and (0.982,0.893) for class 1 respectively. Only 5 of actual class 0 were incorrectly classified, whilst 5 belong to class 1 and 1 belonging to class 2. All predictions of class 0 were correct, resulting in precision of 1. There were 170 class 1 in the test data set, 167 were predicted correctly, 3 were predicted as class 2. The model attempted to predicted 187 class 1's, with 167 of them being correct, whilst 15 were incorrectly classified as class 2. There is a slightly larger drop in recall and precision when classifying class 4 & 5. For class 4, the model achieved 0.3 and 0.429 for recall and precision. In total, there were 10 class 4s' and the model was only capable of predicting 3 correct. Whilst out the 7 that were predicted as class 4, 3 actually belonged to class 3 and another 3 belonged to class 5. The model achieved 0 recall and precision for classifying class 5. This means that the model was not able to predict class 5s' despite having actual class 5 data points in the test data set.

There is a clear consecutive decrease in recall and precision from class 0 to class 5. The main explanation for this kind of behaviour is due to the lack of high class data. The majority of data belong to the first three classes, specially class 0 & 1. There are very little data available that belong to class 3 to class 5 for the model to train and to validate.

Figure 4.11 shows all predicted classes and actual classes. It is clear that the model was able to perform adequately since there is a major overlap between prediction and actual classes.

Table 4.7: Confusion table of the best performing LSTM model using SW method

Training Time: 22.76 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	265	0	0	0	0	0	1
	1	5	167	15	0	0	0	0.893
	2	1	3	45	8	0	0	0.789
	3	0	0	2	15	7	0	0.625
	4	0	0	0	1	3	3	0.429
	5	0	0	0	0	0	0	0
Recall		0.978	0.982	0.723	0.625	0.3	0	

Accuracy = 91.67%

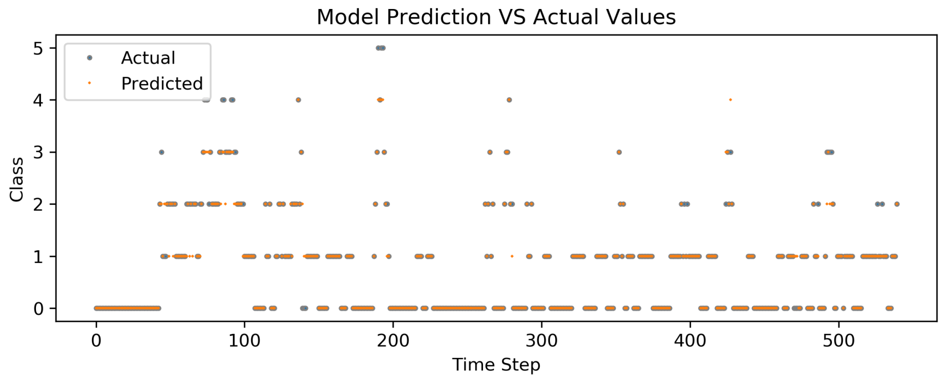
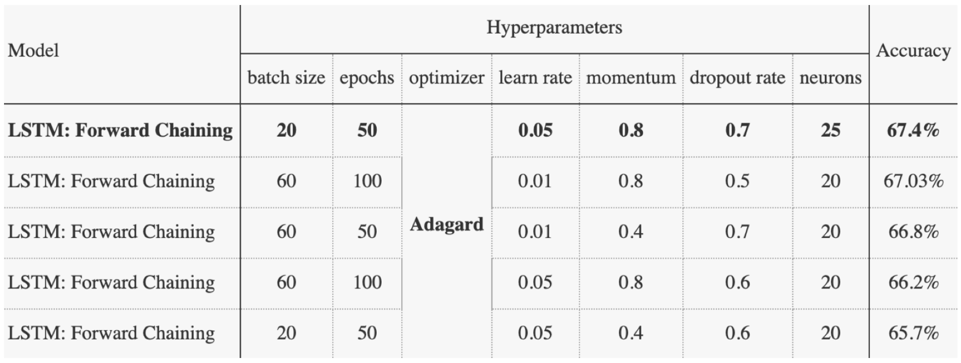


Figure 4.11: LSTM-SW Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.3.4. LSTM - FC

By implementing FC to LSTM, it can provide insight to how different time dependency methods can affect the performance of the classifiers. The best performing versions of LSTM models are included in Figure 4.12. According to Figure 4.12, the combination of LSTM and FC achieved highest accuracy of 67.4% and lowest at 65.7% on training data set. The model with the hyper-parameter set that achieved the highest level of accuracy will be used to build a model and test data set will be used to validate this model.



Model	Hyperparameters							Accuracy
	batch size	epochs	optimizer	learn rate	momentum	dropout rate	neurons	
LSTM: Forward Chaining	**20**	**50**	**Adagard**	**0.05**	**0.8**	**0.7**	**25**	**67.4%**
LSTM: Forward Chaining	60	100		0.01	0.8	0.5	20	67.03%
LSTM: Forward Chaining	60	50		0.01	0.4	0.7	20	66.8%
LSTM: Forward Chaining	60	100		0.05	0.8	0.6	20	66.2%
LSTM: Forward Chaining	20	50		0.05	0.4	0.6	20	65.7%

Figure 4.12: Top 5 performing LSTM-FC models for each condition, their respective set of hyperparameters and accuracy.

With the hyper-parameters determined previously and test data set validated on the model, confusion matrix Table 4.8 includes more detailed performance metrics of the model.

Table 4.8: Confusion table of the best performing LSTM model using FC method

Training Time: 6.84 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	259	95	11	1	0	0	0.708
	1	1	53	28	8	0	0	0.589
	2	0	7	17	10	0	0	0.5
	3	0	0	5	1	7	0	0.0769
	4	0	0	0	0	3	0	1
	5	0	0	0	0	0	0	0
Recall		0.996	0.342	0.279	0.05	0.3	0	

Accuracy = 66.0%

The overall accuracy of using LSTM with FC is 66.0%, which is similar to the level of accuracy achieved during the training phase included in Figure 4.12. A graphical representation of the model performance can be seen in Figure 4.13. The model achieve high values of recall and reasonable value of precision of 0.996 and 0.708 respectively. Out of the 260 actual class 0's, only 1 was incorrectly predicted as class 1. On the other hand, more errors were made when looking at the precision value. In total, the model predicted 366 class 0s' and 259 were classified correctly as class 0, 95 were classified as class 1. Compared to class 0, recall and precision for class 1 shows degraded performance, resulting in 0.343 and 0.589 respectively. Most of the actual class 1's were misclassified as class 0, whilst some were classified as class 2. Recall for class 2 dropped even further to 0.279 and precision decreased slightly to 0.5. Of the 61 actual class 2's, only 17 were classified correctly. The recall and precision values dropped even further to only 0.05 an 0.0769 respectively. Recall of class 4 increase slightly to 0.2 whilst having a perfect precision score. Finally, recall and precision of class 5 are both zero.

A illustration of the performance of the model is included in Figure 4.13. From the figure, it can be seen that most of the predictions provided by the model is concentrated in class 0 since it is the highest frequent value in the test data set. Predictions on classes 2,3,4 and 5 are lacking due to the lack of values in the training and testing data set.

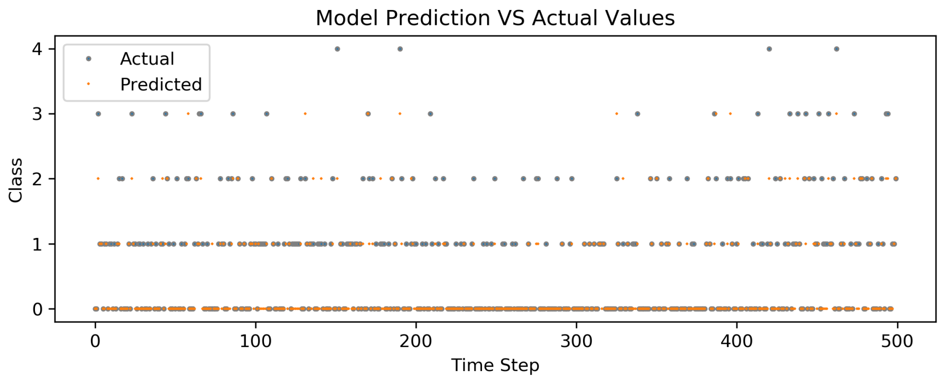


Figure 4.13: LSTM-FC Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.4. Random Forest (RF)

Compared to ANN and LSTM, RF is based on a completely different concept. RF is often used due to its impressive out of the box performance and fast learning compared to many other ML algorithms. Probst et al. [59] showed that RF had the least variability in model accuracy when tuning its hyper-parameters compared to other mainstream ML algorithms. The set of hyper-parameters of RF are vastly different from the ones in ANN and LSTM. The set and range of the hyper parameters used for RF optimisation are discussed in the following section.

4.4.1. Hyperparameters

Since Random Forest is based of different principle, the set of hyperparameters are fundamentally different from both ANN and LSTM. There are 5 hyperparameters available for RF. The hyperparameters for RF are included in Table 4.9.

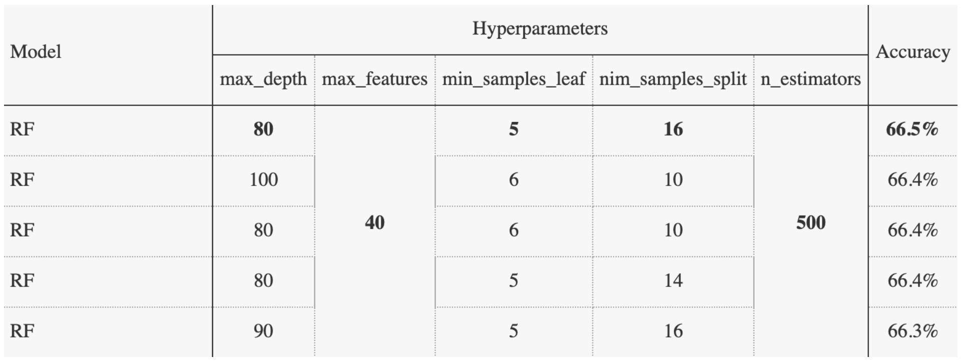
Table 4.9: Hyperparameter Values for RF Model Optimisation

Hyperparameter	Values
max_depth	80, 90, 100, 110, 120
max_features	8, 9, 10, 15, 20, 40, 60, 75
min_samples_leaf	5, 6, 7, 8, 9, 10
min_samples_split	8, 10, 12, 14, 16
n_estimators	100, 200, 300, 400, 500

The set of hyperparameters listed in Table 4.9 resulted in a total of 6000 combinations and a final optimisation time of 93.15 min for RF with no time dependency method. RF-SW and RF-FC each required tuning time of 105.3 min and 82.35 min respectively.

4.4.2. RF - No time dependency integration

According to Figure 4.14, all top 5 models achieved similar results, averaging around 66.4% accuracy. Grid search optimisation process determine that max features of 40 and 500 estimators were the best hyper-parameter value for the model. Other hyper-parameters showed slight variety in values. The set of hyper-parameter with the highest level of accuracy will be used to build a new RF model.



Model	Hyperparameters					Accuracy
	max_depth	max_features	min_samples_leaf	nim_samples_split	n_estimators	
RF	**80**		**5**	**16**		**66.5%**
RF	100		6	10		66.4%
RF	80	**40**	6	10	**500**	66.4%
RF	80		5	14		66.4%
RF	90		5	16		66.3%

Figure 4.14: Top 5 performing RF models for each condition, their respective set of hyperparameters and accuracy.

With the hyper-parameters decided previously, the RF model managed to achieved accuracy of 68.55%. The level of accuracy achieved by this model on testing data is similar to the accuracy achieved by all RF models on training data set.

Table 4.10: Confusion table of the best performing RF model without time dependency method

Training Time: 0.58 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	243	72	10	4	0	0	0.739
	1	12	65	22	1	2	0	0.637
	2	0	7	34	11	7	2	0.557
	3	0	0	6	9	4	1	0.45
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		0.953	0451	0.472	0.36	0	0	

Accuracy = 68.55%

The highest recall of 0.953 and precision of 0.739 achieved by the model is for classifying class 0. A total of 255 actual class 0 were present in the data set and only 12 were misclassified as class 1. On the other hand, the model misclassified 72 class 1's ,10 class 2's and 4 class 3's as class 0. There was a drop in precision and recall for class 1 classification. Of the 144 actual class 1's, only 65 was correctly predicted whilst 72 were misclassified as class 0. A large variety of different classes that were misclassified as class 1. The model predicted 12 of class 0, 22 of class 2, 1 of class 3 and 2 of class 4 as class 1. The model predicted a total of 61 as class 2, with 7 belonging to class 1, 11 belonging to class 3, 4 belonging to class 4 and 1 belonging to class 5. There is further reduction in recall and precision when classifying class 3. Finally, the model achieved zero in both recall and precision for both class 4 and class 5.

Figure 4.15 illustrates all prediction and actual value from the test data set. It is clear that most of the predictions performed by the model is in class 0 and some in class 1. Higher level classes has very limited number of predictions, reflected on the confusion matrix as well.

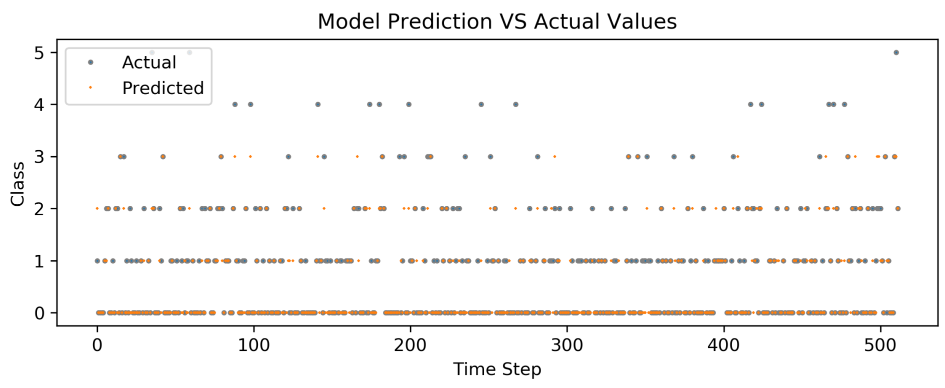
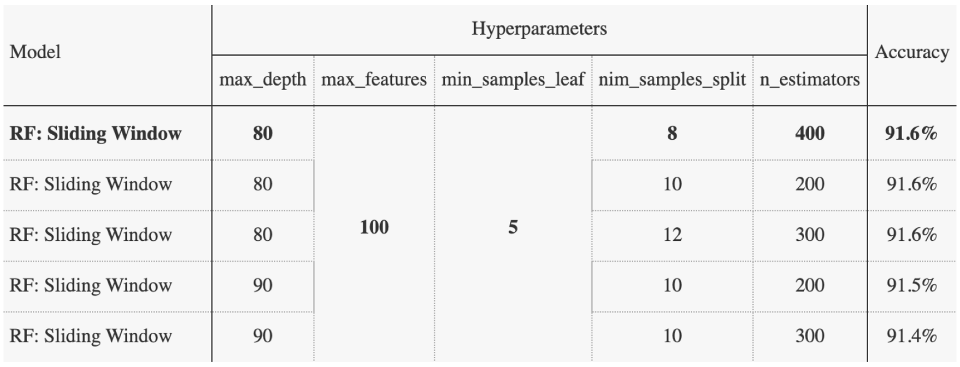


Figure 4.15: RF Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.4.3. RF - SW

The top 5 RF models with SW, shown in Figure 4.16 used varied values of max_depth, min_samples_split and n_estimators. The tuning concluded that max_features of 100 and min_samples_leaf of 5 were the best choices. All of the models resulted in high level of accuracy, with the highest being 91.6% and the lowest at 91.4%. By using the set of hyper parameters that has achieved the highest accuracy, a new RF has achieved results in Table 4.11.



Model	Hyperparameters					Accuracy
	max_depth	max_features	min_samples_leaf	nim_samples_split	n_estimators	
RF: Sliding Window	**80**			**8**	**400**	**91.6%**
RF: Sliding Window	80			10	200	91.6%
RF: Sliding Window	80	**100**	**5**	12	300	91.6%
RF: Sliding Window	90			10	200	91.5%
RF: Sliding Window	90			10	300	91.4%

Figure 4.16: Top 5 performing RF-SW models for each condition, their respective set of hyperparameters and accuracy.

The RF-SW model using aforementioned set of hyper-parameters on test data set achieved high accuracy of 92.17%. Both class 0 and class 2 achieved very high recall and precision levels. Very small numbers were misclassified in both class 0 and class 1. Recall of class 0 & 1 & 2 were 0.992, 0.966 and 0.8 respectively. Precision of class 0 & 1 & 2 were 0.996, 0.909 and 0.81. There is more clear drop in recall and precision in class 3 and on-wards. For class 3, the model achieved recall and precision of 0.435 and 0.556 respectively. In total, 23 class 3 were available in the data set and the model predicted 10 correctly. Recall and precision further decreases for class 4. Finally, for class 5, both recall and precision are both zero.

Table 4.11: Confusion table of the best performing RF model using SW method

Training Time: 2.23 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	252	1	0	0	0	0	0.996
	1	2	140	12	0	0	0	0.909
	2	0	4	68	12	0	0	0.81
	3	0	0	5	10	2	1	0.556
	4	0	0	0	1	1	0	0.5
	5	0	0	0	0	0	0	0
Recall		0.992	0.966	0.8	0.435	0.333	0	

Accuracy = 92.17%

Figure 4.17 illustrates the performance of the RF-SW model. The figure clearly shows a large overlap between predictions by the model and actual data. Some of the high level classes were not predicted correctly.

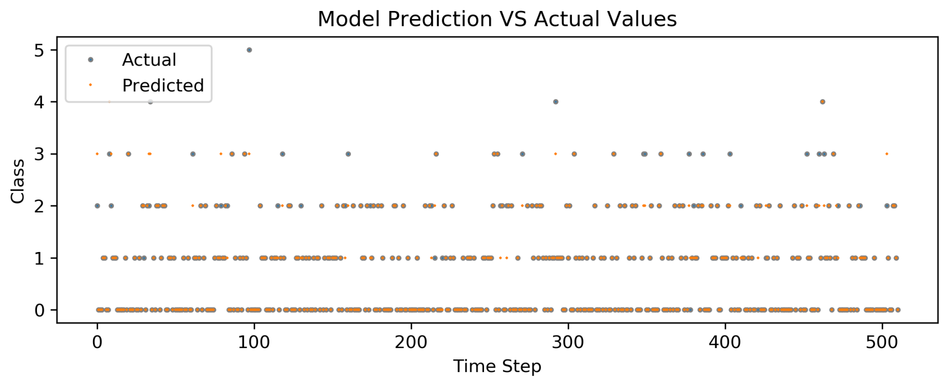
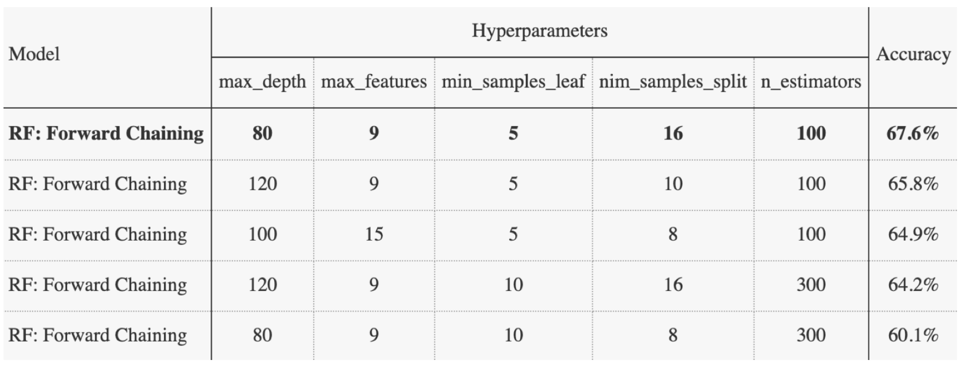


Figure 4.17: RF-SW Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.4.4. RF - FC

The last model used in this study is RF with FC. Figure 4.18 showed that RF models with FC achieved varying levels of accuracy using different set of hyper parameters. The highest accuracy achieved by this combination of model is 67.6% and the lowest was 60.1%. There is relatively high variation in the model accuracy. Thus the set of hyper parameters that resulted in the highest accuracy will be chosen.



Model	Hyperparameters					Accuracy
	max_depth	max_features	min_samples_leaf	nim_samples_split	n_estimators	
RF: Forward Chaining	**80**	**9**	**5**	**16**	**100**	**67.6%**
RF: Forward Chaining	120	9	5	10	100	65.8%
RF: Forward Chaining	100	15	5	8	100	64.9%
RF: Forward Chaining	120	9	10	16	300	64.2%
RF: Forward Chaining	80	9	10	8	300	60.1%

Figure 4.18: Top 5 performing RF-FC models for each condition, their respective set of hyperparameters and accuracy.

RF-FC model using aforementioned set of hyper-parameters achieved accuracy of 67.4%. The model achieved high recall of 0.958 and precision of 0.721 for class 0. The model predicted 344 class 0, 248 were correct classified as class 0 and 86 were classified as class 1. The confusion table indicate a consistent decrease in precision as the level of class increases until class 4. Similar trend can be found in the recall values as well. The model predicted 54 out of 154 of class 1 correctly. There is a small increase in recall for class 2 to 0.492 but with a small decrease in precision 0.492 as well. For class 3, the recall further decreases to 0.25 and precision to 0.333. Finally, for class 4 & 5, recall and precision reached 0.

It is clear from Figure 4.19 that most of predictions provided by the model is at class 0. Whilst there are a few attempts at higher classes, the number and accuracy is far lower than the lower classes.

Table 4.12: Confusion table of the best performing RF model using FC method

Training Time: 2.57 min		Actual						Precision
		0	1	2	3	4	5	
Predicted	0	248	86	10	0	0	0	0.721
	1	11	54	14	1	0	0	0.675
	2	0	16	30	14	1	0	0.492
	3	0	0	7	5	3	0	0.333
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
Recall		0.958	0.346	0.492	0.25	0	0	

Accuracy = 67.4%

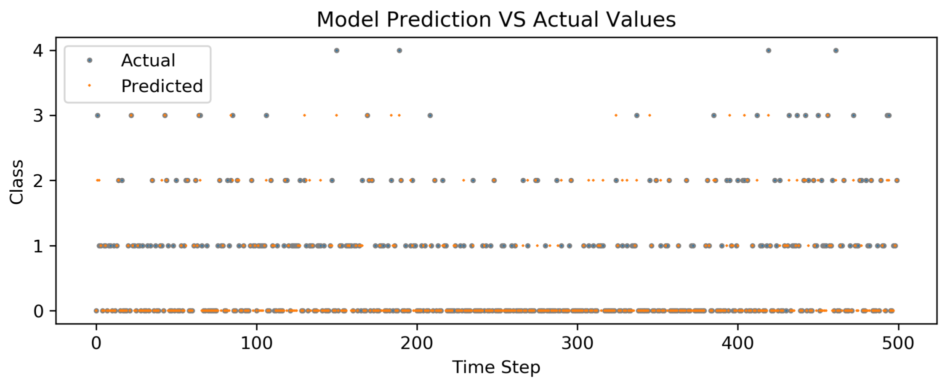


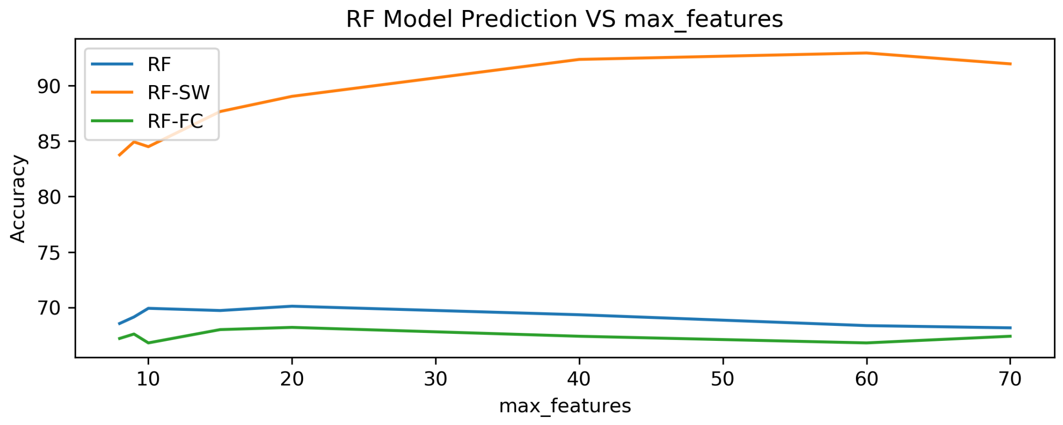
Figure 4.19: RF-FC Model Predictions VS Actual Data. Horizontal axis represent sequential order to the test data. Vertical axis represent different class label.

4.5. Hyper-parameter Sensitivity Analysis

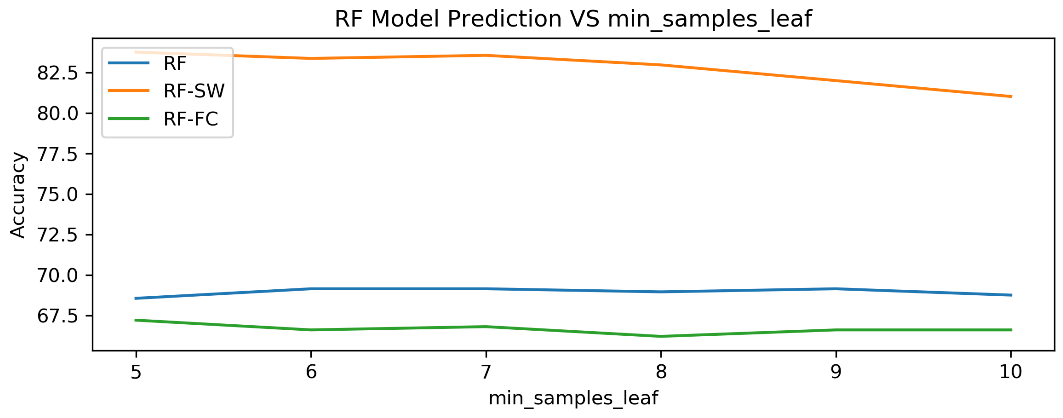
In this section, sensitivity analysis of different models and their respective hyper-parameters are discussed. The changes in accuracy of the model due to changes in a specific hyper-parameter are displayed in a graph. As stated before in the analysis of ANN models, all ANN models performed identically poor on both training and test data. Initial sensitivity analysis on ANN models show that changes in hyper-parameter have no impact in performance of the model. Thus any further sensitivity analysis on ANN model will not yield no useful conclusion. Hence sensitivity analysis of ANN models are not included in this section.

4.5.1. RF Model

All five different hyper-parameters in RF models were used in sensitivity analysis. However, only two are included, as shown in Figure 4.20a & Figure 4.20b. Other hyper-parameters such as *min_samples_split*, *max_depth* and *n_estimators* showed no change in performance, thus having very low sensitivity to all 3 parameters. Graphs of said three hyper-parameters can be seen in Appendix A.



(a) Hyper-parameter sensitivity analysis of max_features in RF Model



(b) Hyper-parameter sensitivity analysis of min_samples_leaf in RF Model

Figure 4.20a presents the hyper-parameter sensitivity of *max_features* on all RF models. It can be observed RF-SW showed the largest sensitivity to *max_features*. The largest change is between 10 and 20 *max_features*. The accuracy achieved its peak after 50 *max_features*. On the other hand, RF showed small increase in accuracy from 5 to 20 *max_features* and constant drop up until 70 features.

RF-FC showed very minimal changes in accuracy when changing *max_features*. Sensitivity analysis of *max_features* show that both RF-FC and RF are fairly insensitive to this hyper-parameter. This is to be expected since in section 4.1, most of the features in the data-set do not contribute much to the model and only a handful of features display reasonable level of importance.

In Figure 4.20b shows the hyper-parameter sensitivity of *min_samples_leaf* of all RF models. Compared to other hyper-parameters, changes in *min_samples_leaf* only show small changes in model performance for RF and RF-FC models. RF-SW showed slightly larger reduction in accuracy starting from 7 until 10 *min_samples_leaf*. However, the drop in accuracy is only of less than 3% with a large increase in *min_samples_leaf*. Hence, although RF-SW showed the largest change in terms of accuracy in reaction to *min_samples_leaf*, the model itself is still no very sensitive to the hyper-parameter.

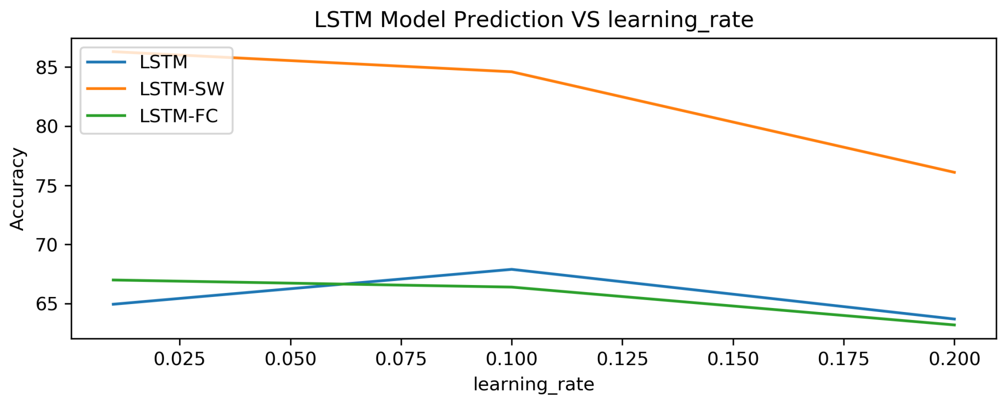
4.5.2. LSTM Model

There are a total of 7 difference hyper-parameters for LSTM models. Four of the seven hyper-parameters showed changes in model performance, whilst the other 3 did not show significant changed, included in Appendix A.

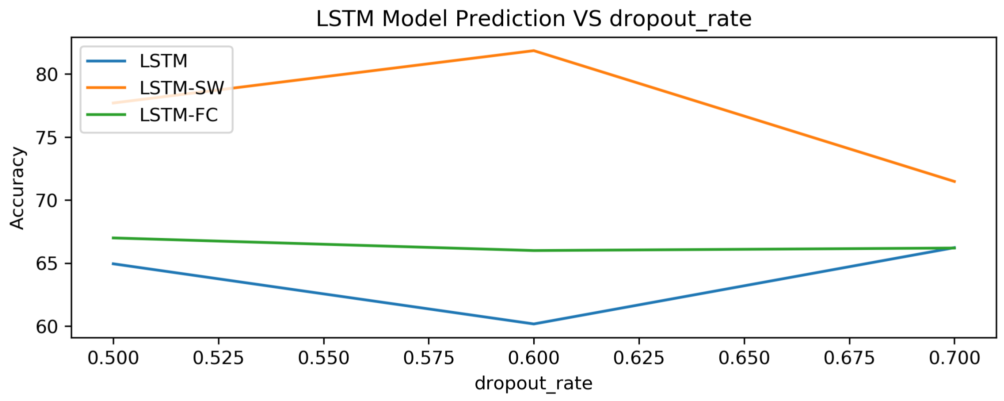
The Figure 4.21a presents the LSTM model performance with different *learn_rate*. It shows that *learn_rate* of 0.1 is a turning point for all models. Both LSTM-FC and LSTM-SW displayed constant decrease in accuracy. At *learn_rate* of 0.1, the decrease rate increased further more. LSTM-SW showed larger changes in terms of accuracy compared to LSTM-FC, showing that LSTM-SW is much more sensitive to *learning_rate*. On the other hand, LSTM showed increase in accuracy from 0.01 until 0.1 *learn_rate* and at *learn_rate* of 0.1, the accuracy started to decrease. However, the level of elevation in terms of accuracy is much smaller compared to other 2 models. From Figure 4.21a, it can observed that both LSTM and LSTM-FC is not very sensitive to changes in *learning_rate* whilst LSTM-SW is much more sensitive to this hyper-parameter.

Additionally, Figure 4.21b presents the LSTM model performance with different *dropout_rate* values. LSTM-SW showed accuracy increasing *dropout_rate* from 0.5 to 0.6, whilst increase from 0.6 until 0.7 showed a large reduction in accuracy. This proves that LSTM-SW is quite sensitive to *dropout_rate*. Opposite trend is observed in LSTM model. LSTM model showed reduction in accuracy when increasing *dropout_rate* from 0.5 to 0.6 and a large increase from 0.6 to 0.7. The elevation is slightly smaller compared to LSTM-SW, indicating a smaller sensitivity. LSTM-FC showed little to no change in accuracy from 0.5 to 0.7 *dropout_rate*. The drop in model performance can also be contributed by the model structure. Dropout layer is designed to reduce the chances of model over-fitting the data. Percentage of dropout is directly related to the number of random inputs excluded from each update cycle. If the dropout value is too high, too many inputs were excluded and the model will not be able to represents the generalised data set, which is illustrated in the change in accuracy of LSTM-SW model. On the other hand, behaviour of LSTM seems abnormal since it is not likely to seen increase in accuracy as *dropout_rate* increases.

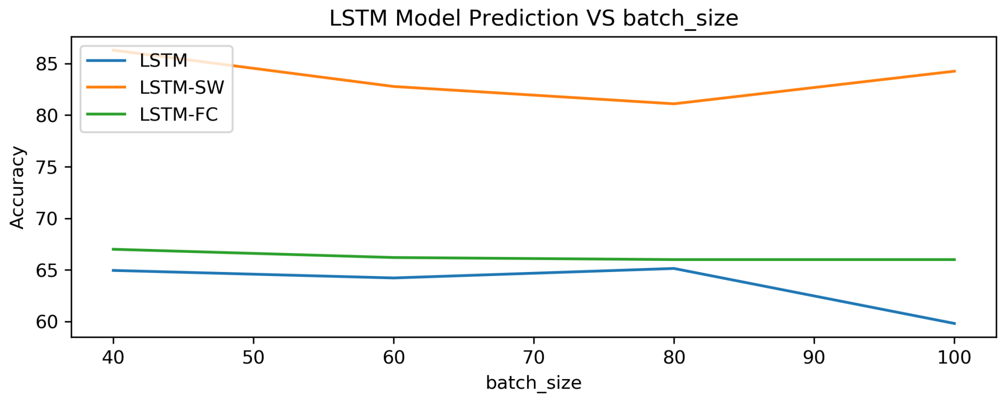
In Figure 4.21c, the LSTM displayed a drop in accuracy of 5% with *batch_size* of 80. On the other hand, both normal LSTM-SW showed slight increase in performance when *batch_size = 80*. LSTM-FC showed little to no change in accuracy with respect to change in *batch_size*, indicating low sensitivity. The level of depreciation in accuracy of LSTM-SW is slightly smaller compared to LSTM. This characteristic is more prominent after *batch_size* of 80. This indicates that LSTM-SW is more sensitive to lower *batch_size* number whilst LSTM is more sensitive to large *batch_size* values.



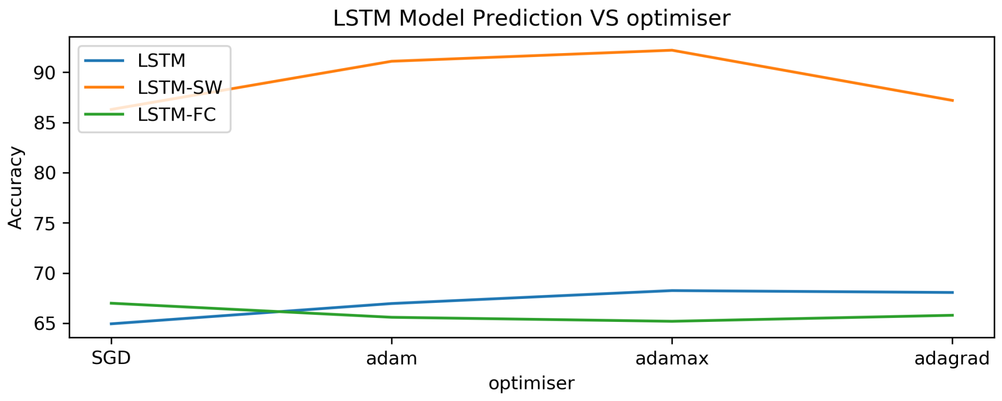
(a) Hyper-parameter sensitivity analysis of learn rate in LSTM Model



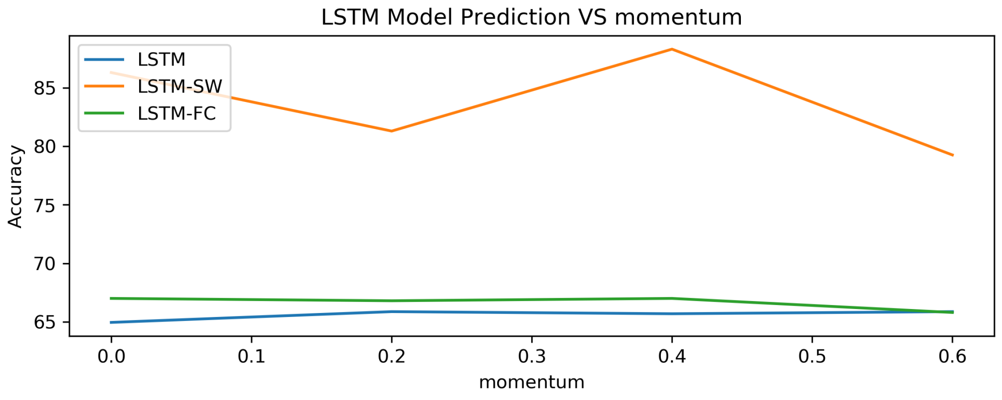
(b) Hyper-parameter sensitivity analysis of dropout in LSTM Model



(c) Hyper-parameter sensitivity analysis of batch size in LSTM Model



(a) Hyper-parameter sensitivity analysis of optimiser in LSTM Model



(b) Hyper-parameter sensitivity analysis of momentum in LSTM Model

Figure 4.22a shows how different optimiser impacts different model performance. In this case, both LSTM & LSTM-FC showed very small different in accuracy when using different optimisers, indicating low sensitivity. On the other hand, LSTM-SW showed the much larger sensitivity. However, the largest change in accuracy is 5%. This level of change is still rather small.

Finally, LSTM and LSTM-FC showed similar trend in sensitivity analysis of *optimisers* and *momentum*. Both LSTM and LSTM-FC showed small changes in accuracy as momentum increases from 0 to 0.6. LSTM-SW on the other hand, showed a oscillating behaviour. Increasing *momentum* from 0 to 0.2 resulted in more than 5% decrease in accuracy. When *momentum* increased to 0.4, the accuracy increased over 7 %. As *momentum* increased to 0.6, the accuracy again dropped to below 80%. This kind of behaviour is very uncommon. Although the reason behind this kind of behaviour is still unknown, accuracy peaked at *momentum* of 0.4. This proves that LSTM-SW is quite sensitive to *momentum* hyper-parameter.

5
Discussion

This chapter presents in-depth discussion of the findings by analysing the results presented in chapter 4. By comparing results between different ML algorithms and between similar literature, it is possible to conduct the advantages and disadvantages of this study in the area of Sport Injury Risk Prediction.

The aim of this study is to investigate the performance of different ML algorithms on longitudinal OSTRC data set when performing injury risk prediction. Data related to game time, training time, physical complaint, changes to training programme and other related questions of the previous week were recorded through weekly questionnaire. Data processing methods were implemented to refine the data to better suit for ML application. The data can then better represent the characteristics of the original data and help to better recognise different injury status. These classifiers are then trained through training data set validated using testing data set. The confusion matrices were used to visualise the results of each individual best performing models with the best performing set of hyper parameters.

Since the amount of researches on ML application for injury (and/or injury risk) is limited, results from this study will be compared to other studies that use ML techniques for classification in the medical application. As more statistical models are developed in the CAS project, this can also be used to compare to traditional statistical models. Most of the studies focusing on ML classification applies to image classification. However, performance of ANN, LSTM and RF on other studies would serve as a baseline to compare to. The models using the same algorithm will be compared and finally, the best of all algorithms are compared between each other and different studies.

5.1. ANN Comparison

The first ML algorithm used in this study was ANN. This, along with 2 different methods to integrate time dependency into the data set, resulted in 3 different versions of ANN classifier. These versions are ANN, ANN-Forward Chaining (ANN-FC) and ANN-Sliding Window(ANN-SW).

When using training data to train the models, normal ANN classifier produced highest accuracy of 61.6%, whilst ANN-FC and ANN-SW achieved 62.6% and 78.6% respectively. Both ANN and ANN-FC produced very similar result on the training data set. ANN model showed slightly more stable in terms of accuracy compared to ANN-FC. Similar accuracy results could indicate similar performance on the test data set. However, this is not the case. On test data set, ANN achieved accuracy of 63.3% whilst ANN-FC only managed to produce accuracy of 31.2%, only half of the accuracy of normal ANN. Whilst ANN managed to provide adequate prediction on different classes, confusion matrix of ANN-FC concludes that the model predicted all values as class 1. This resulted in ANN-FC having perfect recall for class 1 whilst only having precision of 0.312. The only difference between ANN and ANN-FC is different method used for data preparation. Based on the training data, FC seems to have minimal impact on the performance, providing initial conclusion that FC does improve model performance. In this case, implementation of FC to the data set reduced the performance of the ANN-FC compared to data with no time dependency method incorporated. The reason for ANN-FC model to perform this way is still unknown due to the black-box characteristic of ANN models. Hence the initial conclusion from training data set still holds.

Furthermore, ANN-SW showed another significant improvement compared to ANN-FC & ANN. The highest ANN-SW classifier had an accuracy of 78.6%, compared to ANN-FC of 61.6%, that is a 17.3% percent increase on training data. SW method shows promising results in helping the model to learn and improve the classifier. ANN-SW achieved accuracy of 80.59%, marginally better than ANN method and more than twice the accuracy of ANN-FC. The results of ANN-SW on test data set and training data set, it can be concluded that SW is a better data preparation method. SW helps to retain more time dependent information in the data set, allowing the model to learn from it.

However, despite the high accuracy, analysis on the confusion matrix of ANN-SW on test data proves that the model still lack in predicting power on high level classes. A possible explanation for this result could be due to the relative lack of high class data points in the training and test data set. The data set did not provide the model with enough information to train on, thus not being able to classify during validation phase. Another explanation for the low recall ad precision on high classes could be due to the models inability to predict. Since majority of the data set consists of class 0 and class 1 values, the model simply decided to predict the most frequent values to achieve high accuracy instead of relying on information learned from the data set during training phase. Due to the lack of information on the interaction in a ANN model (black-box), further analysis on the cause is not possible.

According to Azar and El-Said [14], ANN managed to achieve 95.11% accuracy. Singh and Urooj [64] and Abubacker et al. [12] produced an accuracy of 87.27% and 97.66% respectively. All three studies achieved higher model performances than ANN in this study. Although this study focuses on longitudinal data whilst other studies did not, the high accuracy of other studies could be used as a reference to know the achievable level of accuracy of a ANN model.

In summary, both results from the training data set and test data set indicate that ANN and ANN-FC are not suitable for this specific application. Although initial accuracy of ANN-SW on training data showed promising results. The accuracy and confusion matrix indicated otherwise. The best performing version of the ANN classifier did not manage to classify anything other than class 0 & 1. However, due to the black-box nature of ML algorithms, it is not possible to investigate the possible underlying characteristics of ANN algorithm that contributes of these results.

5.2. LSTM Comparison

The second ML algorithm used in this study was Long short term memory (LSTM). By implementing SW and FC, once again there are 3 different versions of the LSTM classifiers. These versions are LSTM, LSTM-Forward Chaining (LSTM-FC) and LSTM-sliding window(LSTM-SW).

The highest achieved accuracy for all 3 versions of LSTM are 66.1%, 67.4% and 87.2% respectively. LSTM-FC only showed slight improvement over the normal LSTM model. This small difference could be contributed to the small differences in training and testing data set and not the benefit of implementing FC to the data. On training data set, LSTM and LSTM-FC achieved 64.34% and 66.0% respectively. This level of accuracy is similar to the performance on training data set. Since there is no noticeable improvement compare LSTM over LSTM-FC indicates that FC does not contribute to better performing model. This conclusion is also supported the results of LSTM models using training data set. Confusion matrix of these matrices also provide similar values in terms of precision and recall values.

On the other hand, LSTM-SW model showed significant improvement in accuracy compared to other 2 models. On training data, LSTM-SW achieved highest accuracy of 87.2%, which is much higher compared to other 2 methods. Whilst LSTM-FC achieved even higher accuracy of 91.67%. Since there methods such as dropout and cross validation in place, the chances of overfitting the model is not likely. Prediction performance of LSTM-SW is specially high for classes 0 to 3. However, performance on classes 4 & 5 is considerably lower. A possible explanation to this could be due to lack of data of these classes. Not having enough data entries of class 4 and 5 would result in not enough information trained by the model, thus not being able to predict these classes. The even lower number of class 5 data entries in the data set contributed to the 0 values for both recall and precision. Nevertheless, the level of accuracy achieved by LSTM-SW is leaps ahead compared to LSTM and LSTM-FC. This concludes that SW is more capable of retaining time dependency information and the model is capable of learning it more easily. This conclusion is specially true when compared to FC method.

There are other studies that use longitudinal data to perform predictions. This includes Reddy and Delen [60] which uses LSTM and longitudinal data to prediction readmission for lupus patients. Unfortunately, the study did not compare LSTM to other more drastically different algorithms such as RF or Support Vector Machine (SVM). However, Reddy and Delen [60] showed that LSTM was able to achieve accuracy of 70.54%, proving that LSTM was able to take advantage of longitudinal data. The LSTM model used during this study was capable of producing even higher accuracy at 85+% on training data and 90+% on test data, further proving the benefits of LSTM whilst using longitudinal data. Since it is unknown what kind data was used for Reddy and Delen [60], it cannot conclude that the LSTM method used in this study is superior, but both methods produced high level of model accuracy.

One major characteristic that differentiates LSTM from ANN is the mechanism in place the retains information from previous time steps. As mentioned in chapter 3, LSTM has different gates that contain parameter weight from previous time steps. This information is updated every cycle and passed through to the next time step. Even when the same data set is passed through ANN and LSTM, LSTM slightly outperformed ANN, specially when using SW method. Initial conclusion is that the memory retaining mechanism in LSTM managed to learn the time dependency in the data set, thus producing better results. However, due to lack of more severe injury data, the small variation between LSTM and ANN when using FC could be contributed to difference in training and test data set.

In summary, LSTM-SW showed much better results compared to other 2 LSTM models. There is very little improvement for implementing FC compared to vanilla LSTM models. Whilst SW significantly improved the accuracy of LSTM classifiers.

5.3. RF Comparison

Recurrently, there are 3 versions of the RF models, namely vanilla RF, RF-Forward Chaining (RF-FC) and RF-sliding window(RF-SW).

Normal RF models produced very constant accuracy, averaging 66.4%, with the highest accuracy of 66.5% on training data set. The results of RF-FC is similar to RF models at average of 64.52%. Comparable results are also achieved when validating RF and RF-FC models, 68.55% and 67.4%. Variation on the test and training data set could lead to the small differences in accuracy level. Since RF-FC did not show significant improvement, it can be concluded that FC method did not contribute to improvement of the RF models.

RF-SW showed large improvement in terms of accuracy and recall & precision in confusion matrix. On training data set, RF-SW achieved accuracy of 91.5%, significantly outperforming both RF and RF-FC. On testing data set, RF-SW reached even higher accuracy of 92.17%. Confusion matrix of RF-SW presents high recall and precision for class 0,1 & 2. For classes 3 and 4, recall and precision reduced slightly. This is be due to the lower number of data available for the model to learn. The same reason contributed to lower recall and precision of class 5 classification. However, RF-SW show enough improvement in model performance to conclude that SW is far more capable to retaining time dependency compared to FC method.

Dong et al. [20] managed to achieve an accuracy of 97.7% when classifying mammograms. This is slightly higher than the accuracy provided by the RF model used in this study. Gallego-Ortiz and Martel [31] and Fernández-Carrobles et al. [27] showed RF accuracy around 92% and 90% respectively, which is comparable to the accuracy by RF-SW. Hence it can be concluded that the accuracy produced by RF-SW in this study is within expected range. Talukder et al. [67] performed injury risk predictive modelling on NBA players. The study used a form of longitudinal data set to train and test a RF model. Since the type of data set, data processing and performance metrics are different, the results cannot be directly compared. However, the study concludes that RF managed to achieve high level of accuracy proves that application of RF on longitudinal data is a viable option. This conclusion is also reflected in the results of RF in this study.

In summary, all RF variations showed similar result in both training and testing data set. RF-SW had the best performance in terms of accuracy and confusion matrix values. SW for RF showed significant improvement over FC and normal RF.

5.4. RF & LSTM Comparison

In this section, a comparison between the best of each ML algorithms will be is included.

Among all 3 different types of ML algorithms with 3 different types of data preparation, the best performing were the ones that used SW during data preparation. This conclusion is only applicable for RF and LSTM. Classifier ANN was not able to learn from the data and provide adequate classifications results. Thus ANN will not be included in this comparison going forward.

According the accuracy and confusion matrix of classifier RF and LSTM on both test and training data, the best classifiers are RF-SW and LSTM-SW. This proves that by applying SW to the data set, it allowed the classifier to learn more about the time dependency within the data set and produce better results.

In general, the accuracy of LSTM-SW is slightly higher than RF-SW, by 0.5%. However, the values in the confusion matrix in each classifiers indicate there are some advantages to LSTM-SW compared to RF-SW. In sports, the more severe an injury is, the more impactful it could be to the athlete and related stakeholders. This is specially true in professional environment. Thus it is more important to for the model to be able to classify high class injury risks. For RF-SW, according to Figure 4.17, recall for class 3,4& 5 are 0.435, 0.333 and 0 respectively. Precision for class 3,4&5 are 0.556,0.5 and 0 respectively. For LSTM-SW, according to Figure 4.11 recall for class 3,4& 5 are 0.625, 0.3 and 0 respectively. Precision for class 3,4& 5 are 0.625, 0.429 and 0 respectively.

For class 3, recall of LSTM-SW is much higher than RF-SW. This indicates that LSTM-SW model is more capable of predicting class 3 injury risks. Recall for class 4 showed similar performance. Precision of RF-SW for class 3 is slightly lower than LSTM-SW. This proves that LSTM-SW is less likely to misclassify other injury risk classes as class 3. On the other hand, precision of RF-SW for class 4 is slightly higher than LSTM-SW model. However, testing data for LSTM-SW includes more class 4 data entries compared to RF-SW test data set. Hence although precision of RF-SW is high, no firm conclusion can be derived. Both models achieved 0 in recall and precision for class 5 injury risk. It is believed that the main contributor to this problem is lack of class 5 injury risk data entries in the data set. LSTM-SW showed significant higher accuracy compared to Reddy and Delen [60]. This indicates that SW was capable of capturing the temporal features in the OSTRC longitudinal.

In general, based on the confusion matrix and accuracy, LSTM+SW outperformed RF+SW by a small amount. However, RF is able to tune, train and test much more efficiently compared to LSTM. During this study, computation time is not as important as accuracy, hence long computation time was not an issue. However, from a practical point of view, if this model would be applied a study that requires fast prediction, RF+SW might be a better option.

5.5. Computation Time Comparison

In this section, the tuning time of all the models are compared. The training & validation time of the models are compared as well.

5.5.1. Tuning Time

Table 5.1 displays all tuning time that all models used during this study. All time recorded are in minutes.

Table 5.1: Tuning time of all models used in this study

Model	**ANN**	**LSTM**	**RF**	**Unit**
No time dependency	407.3	323.8	93.15	min
SW	530.8	551.7	105.3	
FC	3024.2	2970.1	82.35	

In general, models using FC required the longest amount of time to tune. Both ANN and LSTM required around 50 hours to find the best set of hyperparameters. Whilst no vanilla models and SW methods required much less time to tune. All variables of RF required much less time compared to other algorithms. This coincides with the fact that RF is much more efficient and require less time to tune whilst maintaining a high level of accuracy.

5.5.2. Training & Validation Time

Table 5.2 displays all training & validation time that all models used during this study. All time recorded are in minutes.

Table 5.2: Training & Validation time of all models used in this study

Model	**ANN**	**LSTM**	**RF**	**Unit**
No time dependency	8.07	9.78	0.58	min
SW	10.54	22.76	2.23	
FC	6.03	6.84	2.57	

Different from tuning time, results of training & validation time show that SW requires longer time to train on train data set and validate using test data set. On average, models that utilise FC requires the least time to train and validate. The results of RF in Table 5.2 also verifies the fast computation capabilities of RF model.

6
Conclusion

In this study, pre-processing of OSTRC data, data encoding, model training, tuning and validation are performed. Model performance metrics along with computation time are compared to validate the performance of the model as well as to determine which is the best performing model. In order to come to the conclusion of this study, the research questions posed in the introduction are answered.

Research Question 1: *What is the performance of different injury risk status prediction models?*

The performance of the models are indicated using accuracy as well as confusion matrix of each model. There are 3 different algorithms using 3 differently prepared data, totalling 9 different classifiers. Generally speaking, among these classifiers, the best overall performing classifier on training data set is RF-SW. Although LSTM-SW out performs RF-SW in terms of accuracy, since the optimisation, training and validation process of LSTM is significantly longer than RF, the small difference between accuracy can be compensated for its fast computation time. The baseline of all the models are any of the ANN models. Both LSTM and RF showed significant improvements over ANN, the highest being above 90% accuracy with reasonable levels of recall and precision.

Research Question 2: *Which type of ML model is best suited for injury risk status prediction using longitudinal panel data (OSTRC)?*

Both performance metrics (accuracy, confusion matrix) are used to identify the best performing model of all. There is a condition that need to be considered which is whether or not the level of detail in the data set and data structure are similar to the one used in this study. If that is the case, the answer to the question is RF-SW since it has high accuracy and fast computation time comparing to LSTM-SW. On the other hand, if the data set is larger and more detailed, then LSTM-SW is a more viable option. This is because although the computational time of LSTM is much longer, the advantage of being able to learn more about the time dependency allow more personalisation of the model for specific athlete or sport.

Research Question 3: *Does inclusion of time dependency improve model performance?*

Typical ML algorithms ignores possible pre-existing time dependency in the data set. However, for problems like injury risk that changes with over time, the inclusion of time dependency could significantly change the model's performance. As stated in the chapter 4, only ANN did not show any improvement in terms of accuracy when using time dependency methods. For RF and LSTM, not all time dependency methods were successful. Application of FC did not show significant improvement in terms of accuracy, but SW did improve the classifiers accuracy in a significant level. It is safe to say that having time dependency, specifically SW greatly improves model performance.

Research Question 4: *Can ML algorithms provide adequate predictions for practical application?*
Looking at the accuracy of RF-SW and LSTM-SW, both classifiers were able to provide accuracy above 90%. Although there is not prior studies conducted for this kind of application, having such a high accuracy classifier is still proves that ML can provide good prediction of injury status. LSTM has shown to be able to learn from time dependency in the data and provide good predictions. By providing the classifier with more and more detailed data, it is possible to improve the system even more. Current computation time for both RF and LSTM is not suitable for real-time application. By using a more powerful computer, it is also possible to significantly reduce required computational time for optimisation, training and validation of the models.

Main research question: *Is is possible to predict injury risk of athletes using OSTRC data set and ML algorithms, and if so, to what extent?*

The results of this study indicate the possibility of applying ML algorithms to longitudinal data set. Although in theory, both FC and SW had the ability to integrate time dependency in the data set, the results says otherwise. In all versions of the models, implementation of FC did not show any significant improvement in terms of model accuracy compared to normal data. On the other hand, implementation of SW showed significant improvement across all models. A horizontal comparison between 3 different algorithms indicate that both LSTM and RF achieved high accuracy. This is specially true when using SW processed data set. Both LSTM-SW and RF-SW achieved accuracy above 90%. Analysis on the confusion matrix indicate high recall and precision on the lower level class injury risk. Recall and precision on higher injury risk class is still lacking. This is specially true for class 5. This is due to the lack of data entries for these high level classes in the data set. There is a large different between tuning and computation time of RF-SW and LSTM-SW models. Tuning process of LSTM models is significantly longer compared to RF-SW whilst the absolute difference in computation time is not as significant. Since the OSTRC data is always collected on a weekly basis, a slightly longer computation time does not affect its applicability.

We conclude that SW is the best performing data preparation method for longitudinal OSTRC data set. Whilst both LSTM and RF provided high level of accuracy. Based on the confusion matrix of these two models, LSTM slightly outperforms RF in terms of high injury risk level prediction. As more data become available, the potential for LSTM to improve its accuracy even more is very high. At current state, it can be concluded that ML algorithms can provide adequate predictions as recommendations for athletes to take into consideration.

The tools, scripts used in this report is available at the project's github page:
https://https://github.com/lianwu371/OSTRC-Longitudinal-Project.

7
Future Work

Despite certain limitations, this study has shown that ML algorithms do have a spot to play in predicting injury risk status of athletes. LSTM and RF along with SW achieved high level of accuracy. In order to further validate the applicability of these models in real life, more data will require. Nevertheless, there are certain directions and improvements for further research that can be taken into account.

- More data could benefit the accuracy of the models. Now there are only 2000 entries for all athletes, averaging around 100 weeks per athlete. However, due to the infrequent cases of injuries that occur and were recorded in the questionnaire, it is more beneficial to have more injury related data entries. This allows the model to train on more high class injury status and validate more accurately on test data set.
- More detailed data could help to personalise models by only training the models using data from particular athlete. By having more personal data such as weight, age, height, sex, and other, the model could be trailer fitted to said specific athlete. Or it could help the model to recognise the importance of said features when it comes to injury status.
- More frequent data could help the model to capture smaller injury related features that could have being forgotten when filling a weekly questionnaire. A good solution would be to perform daily monitor through an easier method such as online. By providing athletes with easier data collecting method and more frequent data gathering, it can significantly increase the resolution of the data set, providing ML algorithms with more data to train and test with.
- Optimising the optimisation process to reduce computation time. There are multiple ways to tackle this problem. Firstly, the easiest way to solve this problem is to use a better computer. Having more powerful hardware could significantly improve computation time. Secondly, try to find out a way that is faster at optimising the variables. Since grid search is used in this study, it will cycle through all possible hyper parameter combinations. By using methods such as random search, or gradient search, it could significantly reduce optimisation. But the down side of these methods is its possibility of reaching a local minima.
- More sophisticated injury risk status indicator. Currently a linear relation is used to determine the injury risk. However, by including a more complex and sophisticated method to compute injury status, it has the possibility of increasing classification accuracy. Another method would be involve medical professional into the project and provide professional diagnosis of athletes injury status.

- Apply other ML algorithms to compare the performance of the models. The ML algorithms used in this study are the basic version of their respective algorithm type. For example, there are other variations of ANN and RF that were not included in this study. Only after comparison, the best classifier can be decided.

Bibliography

[1] Cross-sectional data. URL https://www.sciencedirect.com/topics/mathematics/cross-sectional-data.

[2] Overfitting and underfitting. URL https://www.educative.io/edpresso/overfitting-and-underfitting.

[3] URL https://gauraw.tech/tutorials/machinelearning/machine-learning.md.html.

[4] matplotlib. URL https://matplotlib.org/.

[5] Model selection. URL http://ethen8181.github.io/machine-learning/model_selection/model_selection.html.

[6] Numpy. URL https://numpy.org/.

[7] Ostrcsite. URL https://www.ostrc.no/.

[8] pandas. URL https://pandas.pydata.org/.

[9] Feature importances with forests of trees¶, . URL https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html.

[10] scikit-learn, . URL https://scikit-learn.org/.

[11] Tensorflow. URL https://www.tensorflow.org/.

[12] Nirase Fathima Abubacker, Azreen Azman, Shyamala Doraisamy, and Masrah Azrifah Azmi Murad. An integrated method of associative classification and neuro-fuzzy approach for effective mammographic classification. *Neural Computing and Applications*, 28(12):3967–3980, 2017.

[13] Arthur Arnx. First neural network for beginners explained (with code), Aug 2019. URL https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf.

[14] Ahmad Taher Azar and Shaimaa Ahmed El-Said. Probabilistic neural network for breast cancer classification. *Neural Computing and Applications*, 23(6):1737–1751, 2013.

[15] Rajesh S. Brid. Decision Trees a simple way to visualize a decision. URL https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb. (2019, August 16).

[16] Amar Budhiraja. Learning less to learn better-dropout in (deep) machine learning, Mar 2018. URL https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learn

[17] Benjamin Clarsen, Grethe Myklebust, and Roald Bahr. Development and validation of a new method for the registration of overuse injuries in sports injury epidemiology: the oslo sports trauma research centre (ostrc) overuse injury questionnaire. *Br J Sports Med*, 47(8):495–502, 2013.

[18] International Olympic Committee. Factsheet the games of the olympiad. 08 2013.

[19] Michael DelSole. What is one hot encoding and how to do it, Apr 2018. URL `https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179`.

[20] Min Dong, Xiangyu Lu, Yide Ma, Yanan Guo, Yurun Ma, and Keju Wang. An efficient approach for automated mass segmentation and classification in mammograms. *Journal of digital imaging*, 28(5):613–625, 2015.

[21] Sanket Doshi. Various optimization algorithms for training neural network, Mar 2019. URL `https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6`.

[22] Stephen C. Brock Douglas A. Kleiber. The effect of career-ending injuries on the subsequent well-being of elite college athletes. 09 1992.

[23] Karsten Eckhardt. Choosing the right hyperparameters for a simple lstm using keras, Nov 2018. URL `https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046`.

[24] Jan Ekstrand, Markus Waldén, and Martin Hägglund. Hamstring injuries have increased by 4% annually in men's professional football, since 2001: a 13-year longitudinal analysis of the uefa elite club injury study. *Br J Sports Med*, 50(12):731–737, 2016.

[25] Jan Ekstrand, Werner Krutsch, Armin Spreco, Wart van Zoest, Craig Roberts, Tim Meyer, and Håkan Bengtsson. Time before return to play for the most common injuries in professional football: a 16-year follow-up of the uefa elite club injury study. *British journal of sports medicine*, 54(7):421–426, 2019.

[26] Peter Eston. *Data Mining Concepts and Techniques*, volume 0 of *0*, chapter 3, pages 84–124. Morgan Kaufmann, 225Wyman Street,Waltham, MA 02451, USA, 3 edition, 7 2011.

[27] M Milagro Fernández-Carrobles, Gloria Bueno, Oscar Déniz, Jesús Salido, Marcial García-Rojo, and Lucía González-López. A cad system for the acquisition and classification of breast tma in pathology. *Stud Health Technol Inform*, 210:756–60, 2015.

[28] Jessica Fulton, Kathryn Wright, Margaret Kelly, Britanee Zebrosky, Matthew Zanis, Corey Drvol, and Robert Butler. Injury risk is altered by previous injury: a systematic review of the literature and presentation of causative neuromuscular factors. *International journal of sports physical therapy*, 9(5):583, 2014.

[29] Tim J Gabbett. The development and application of an injury prediction model for noncontact, soft-tissue injuries in elite collision sport athletes. *The Journal of Strength & Conditioning Research*, 24(10):2593–2603, 2010.

[30] Tim J Gabbett and Nathan Domrow. Relationships between training load, injury, and fitness in sub-elite collision sport athletes. *Journal of sports sciences*, 25(13):1507–1519, 2007.

[31] Cristina Gallego-Ortiz and Anne L Martel. Improving the accuracy of computer-aided diagnosis for breast mr imaging by differentiating between mass and nonmass lesions. *Radiology*, 278 (3):679–688, 2016.

[32] Aurelien Geron. *Chapter 2. End-End Machine Learning Project*, page 66–74. O'Reilly Media.

[33] Robert Graham. Protection and prevention strategies, Feb 2014. URL `https://www.ncbi.nlm.nih.gov/books/NBK185338/`.

[34] Prince Grover. Getting deeper into categorical encodings for machine learning, Jul 2019. URL `https://towardsdatascience.com/getting-deeper-into-categorical-encodings-for-machine-learning-2312acd347c8`.

[35] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, volume 0 of *0*. O'Reilly Media, 1005 Gravenstein Highway North Sebastopol, CA 95472, 2 edition, 6 2019. ISBN 9781492032632.

[36] Jiawei Han and Jian Pei. *Chapter 3: Data Preprocessing*, page 83–111. Morgan Kaufmann, 3 edition.

[37] Katherine Herman, Peter Malliaras, and Dylan Morrissey. The effectiveness of neuromuscular warm-up strategies, that require no additional equipment, for preventing lower limb injuries during sports participation: a systematic review. *BMC medicine*, 10:75, 07 2012. doi: 10.1186/1741-7015-10-75.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[39] Insurance Information Institute. Facts + Statistics: Sports injuries. URL `https://www.iii.org/fact-statistic/facts-statistics-sports-injuries`. (2019, August 15).

[40] Intellipaat. 7 Big Data Examples: Applications of Big Data in Real Life. URL `https://intellipaat.com/blog/7-big-data-examples-application-of-big-data-in-real-life/`. (2019, June 15).

[41] Pier Paolo Ippolito. Hyperparameters optimization, Sep 2019. URL `https://towardsdatascience.com/hyperparameters-optimization-526348bb8e2d`.

[42] Pawan Jain. Complete guide of activation functions, Jun 2019. URL `https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044`.

[43] Chayan Kathuria. Regression-why mean square error?, Dec 2019. URL `https://towardsdatascience.com/https-medium-com-chayankathuria-regression-why-mean-square-error-a8`

[44] S B Knowles. Cost of injuries from a prospective cohort study of north carolina high school athletes. 12 2007.

[45] Will Koehrsen. Random Forest Simple Explanation. URL `https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d`. (2019, August 16).

[46] William Kraemer, Craig Denegar, and Shawn Flanagan. Recovery from injury in sport: considerations in the transition from medical care to performance care. *Sports health*, 1(5):392–395, 2009.

[47] Eryk Lewinson. Explaining feature importance by example of a random forest, Feb 2019. URL `https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e`.

[48] Hamza Mahmood. Softmax function, simplified, Nov 2018. URL https://towardsdatascience.com/softmax-function-simplified-714068bf8156.

[49] Cory Maklin. LSTM Recurrent Neural Network Keras Example. URL https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term (2019, August 27).

[50] Danilo P. Mandic, Jonathon A. Chambers, and undefined undefined undefined. *Recurrent Neural Networks Architectures*, page 69–72. John Wiley, 2001.

[51] Sarang Narkhede. Understanding confusion matrix, Aug 2019. URL https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[52] Michael Nielsen. Neural networks and deep learning, Dec 2019. URL http://neuralnetworksanddeeplearning.com/chap1.html.

[53] Sebastian Norena. Finding correlation between many variables (multidimensional dataset) with python, Nov 2018. URL https://medium.com/@sebastiannorena/finding-correlation-between-many-variables-multidimensional-dataset-with-python-5deb3

[54] Daniel Pfirrmann. Analysis of injury incidences in male professional adult and elite youth soccer players: A systematic review. 05 2016.

[55] Mikael Phi. Illustrated guide to lstm's and gru's: A step by step explanation, Jul 2019. URL https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[56] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4):7–9, 2017.

[57] Pourya. Sw-fc-figures, Apr 2019. URL https://towardsdatascience.com/time-series-machine-learning-regression-framework-9ea33929009a.

[58] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[59] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*, 2018.

[60] Bhargava K Reddy and Dursun Delen. Predicting hospital readmission for lupus patients: An rnn-lstm-based deep-learning methodology. *Computers in biology and medicine*, 101:199–209, 2018.

[61] Pablo Ruiz. Mlapproachesfortimeseries, May 2019. URL https://towardsdatascience.com/ml-approaches-for-time-series-4d44722e48fe.

[62] Raheel Shaikh. Choosing the right encoding method-label vs one-hot encoder, Nov 2018. URL https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b.

[63] Denise L.smith Sharon A. Plowman. *Exercise Physiology for Health Fitness and Performance*. Philadelphia : Wolters Kluwer/Lippincott Williams & Wilkins Health, 2014. ISBN 9781451176117 1451176112. URL https://books.google.nl/books?id=GkpPLE5uQuwC&printsec=frontcover#v=onepage&q&f=false.

[64] Satya P Singh and Shabana Urooj. An improved cad system for breast cancer diagnosis based on generalized pseudo-zernike moment and ada-dewnn classifier. *Journal of medical systems*, 40(4):105, 2016.

[65] Manik Soni. Understanding architecture of lstm cell from scratch with code., Jun 2018. URL `https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4`.

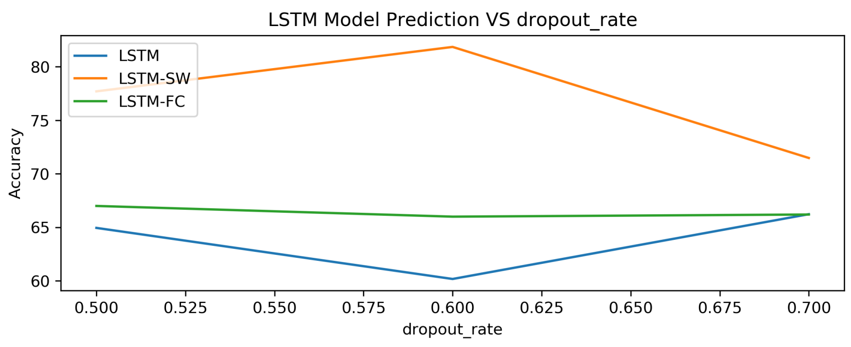
[66] Sam T. Entropy: How decision trees make decisions, Jul 2019. URL `https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8`.

[67] Hisham Talukder, Thomas Vincent, Geoff Foster, Camden Hu, Juan Huerta, Aparna Kumar, Mark Malazarte, Diego Saldana, and Shawn Simpson. Preventing in-game injuries for nba players. In *Read at the MIT Sloan Sports Analytics Conference*, pages 11–12, 2016.

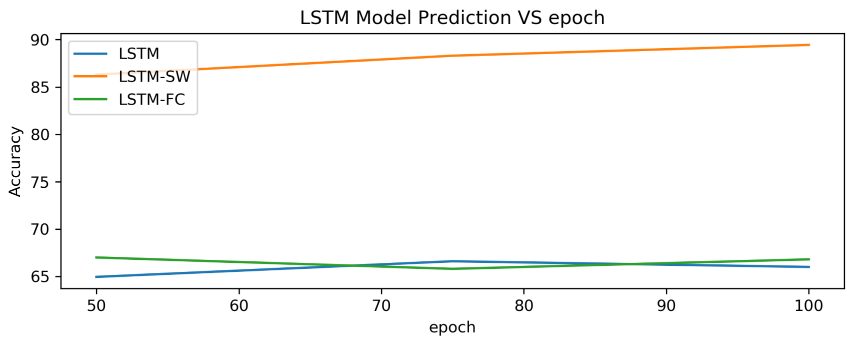
[68] The New York Times. Cost of Contact in Sports Is Estimated at Over 600,000 Injuries a Year. URL `https://www.nytimes.com/2017/09/29/health/sports-injuries-football-yale.html`. (2019, August 2).

A

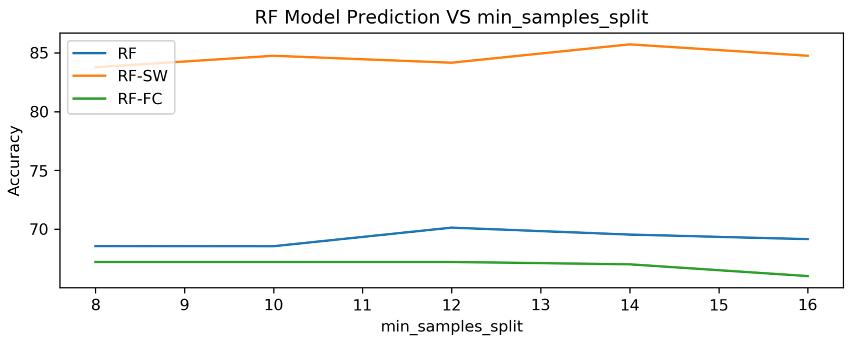
Appendix A: Hyper-parameter Sensitivity Analysis



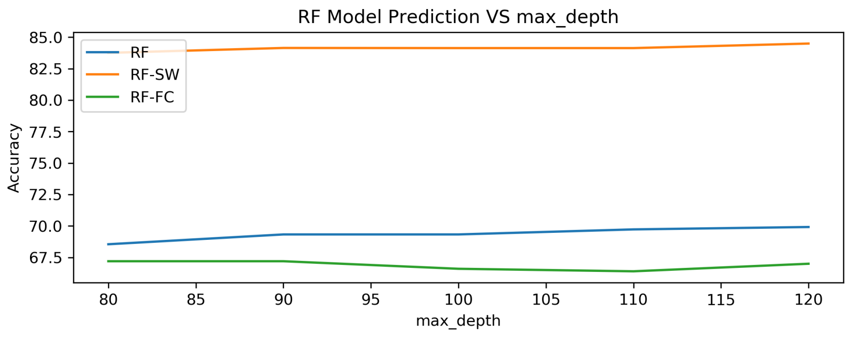
(a) Hyper-parameter sensitivity analysis of dropout in RF Model



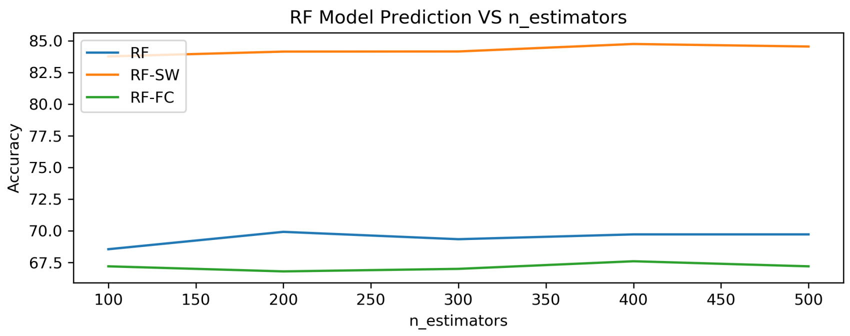
(b) Hyper-parameter sensitivity analysis of epoch in RF Model



(a) Hyper-parameter sensitivity analysis of min_samples_split in RF Model



(b) Hyper-parameter sensitivity analysis of max_depth in RF Model



(c) Hyper-parameter sensitivity analysis of n_estimators in RF Model

B

Appendix B: Python Code

This chapter includes the encoding, Injury risk computation, Sliding window and Forward chaining codes. Unfortunately, parts of the code did not fit the page. In case a more detail viewing on the codes is desired, it can be found in GitHub, along with the rest of the code:
https://https://github.com/lianwu371/OSTRC-Longitudinal-Project

B.1. Ordinal Encoding

```
#—————————————————Ordinal Encoding———————————————————————————————

def ordinalencoding(df):

    title_lst = list(df.columns)
    string_title_lst=title_lst[5:21]

    for i in range(len(string_title_lst)):
        #valuecounts = df[string_title_lst[i]].value_counts()

        #Haveyouexperiencedaphysicalcomplaintduringthepast7daysduringexercise
        if i == 0:
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣fully␣participated␣wi
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣fully␣participated ,␣b
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣participated␣in␣part␣
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣did␣not␣participate␣a

        #Towhatextenthaveyouadjustedyourtrainingorparticipationincompetitionsinthepast7daysdu
        elif i == 1:
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣have␣not␣adjusted␣thi
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("Somewhat", 1)
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("A␣lot␣of", 3)
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("Mediocre", 2)
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣could␣not␣participate

        #Towhatextenthaveyounoticedinthepast7daysthatthisphysicalcomplainthasaffectedyourperfo
        elif i == 2:
            df[string_title_lst[i]]= df[string_title_lst[i]].replace("My␣performance␣was␣not␣
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Somewhat", 1)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("A␣lot␣of", 3)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Mediocre", 2)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣could␣not␣partic

    #Towhatextentdidyouexperiencethesymptomsofthisphysicalcomplaintinthepast7days
    elif i == 3:
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣wasn't␣bothered"
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Somewhat", 1)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("A␣lot␣of", 3)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Mediocre", 2)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("I␣could␣not␣partic

    #Howmanydaysinthelastweekhaveyoubeenunabletoparticipatefullyorcompletelyinatrai
    elif i == 4:
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Fully␣participated
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("All", 5) #Alle

    #ACUUT-Isthisthefirsttimeyouhavereportedthisphysicalcomplaint
    elif i == 6:
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Yes", 0)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣also␣reporte
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣reported␣the

    #OVERUSE-Isthisthefirsttimeyouhavereportedthisphysicalcomplaint
    elif i == 9:
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Yes", 0)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣also␣reporte
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣reported␣the
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Damn␣it", 1)

    #ILLNESS-Isthisthefirsttimeyouhavereportedthisphysicalcomplaint
    elif i == 12:
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("Yes", 0)
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣also␣reporte
        df[string_title_lst[i]]= df[string_title_lst[i]].replace("No,␣I␣reported␣the

    #Scale the 0-100 values to 0-1:
    #Howresteddoyoucurrentlyfeel
    elif i == 13:
        df[string_title_lst[i]]= df[string_title_lst[i]]/100

    #Howenergeticdoyoufeelatthemoment
    elif i == 14:
        df[string_title_lst[i]]= df[string_title_lst[i]]/100

    #Howresteddoyoucurrentlyfeel
    elif i == 15:
        df[string_title_lst[i]]= df[string_title_lst[i]]/100
    return df

df = ordinalencoding(df)
```

B.2. Onehot Encoding

```
#———————————————————————————————Onehot encoding——————————————————————————————————————————

def onehotencoding(df):

    title_lst = list(df.columns)
    string_title_lst=title_lst[5:21]

    obj_columns = []
    for i in range(0,5):
        number =[5,7,8,10,11]
        obj_columns.append(string_title_lst[number[i]])

    newdf = df[['Doesthephysicalcomplaintthatyouhaveexperiencedpainordiscomfortinthepast7day

    df_encoded = pd.get_dummies(data=newdf, columns=obj_columns)

    #drop the columns that were just onehot encoded

    for i in range(0,len(obj_columns)):
        df = df.drop([obj_columns[i]],axis=1)

    # merge with main df bridge_df on key values
    df = df.join(df_encoded)

    #Delete non–relevant variables
    #del i,number,obj_columns, df_encoded, newdf

    return df, df_encoded

df,df_encoded = onehotencoding(df)
```

B.3. Injury Risk Computation

```
#————————————————————————Determine Injury Status by 4 Question Indicators——————————————————
#Logic of injury status definition

#Columns for injury status indication
#Participations – Haveyouexperiencedaphysicalcomplaintduringthepast7daysduringexercise
#Modified training – Towhatextenthaveyouadjustyourtrainingorparticipationincompetitionsinthep
#Performance – Towhatextenthaveyounoticedinthepast7daysthatthisphysicalcomplainthasaffectedyo
#Symptoms – Towhatextentdidyouexperiencethesymptomsofthisphysicalcomplaintinthepast7days

#n = number of weeks to be taken into account to determine injury status
```

```
def injurystatusdef(df):

    #Define injury weight
    injury_weight = [0.25 , 0.25 , 0.25 , 0.25]
    final_weight_0 = [0.5 , 0.3 , 0.15 , 0.5]
    final_weight_1 = [0.6 , 0.3 , 0.1]
    final_weight_2 = [0.7 , 0.3]
    final_weight_3 = [1]

    #Data management

    df = df.sort_values(by=['Participantsname', 'Recurrence'])

    df_injury = df[["Participantsname","Recurrence","Haveyouexperiencedaphysicalcomplai
    df_injury["Haveyouexperiencedaphysicalcomplaintduringthepast7daysduringexercise"] =
    df_injury["Towhatextenthaveyouadjustedyourtrainingorparticipationincompetitionsinthe
    df_injury["Towhatextenthaveyounoticedinthepast7daysthatthisphysicalcomplainthasaffect
    df_injury["Towhatextentdidyouexperiencethesymptomsofthisphysicalcomplaintinthepast7d
    df["Firstscore"] = df_injury["Haveyouexperiencedaphysicalcomplaintduringthepast7day
    df_injury["Firstscore"] = df[["Firstscore"]]
    participantsname = df_injury[["Participantsname"]].to_numpy()
    firstscore = df_injury[["Firstscore"]].to_numpy()

    injury_list = np.zeros(len(df_injury))
    final_injury = np.zeros(len(df_injury))

    for i in range(3,len(df_injury)):
        j = i - 1
        k = i - 2
        l = i - 3

        if participantsname[i] == participantsname[j] == participantsname[k] == participa
            if firstscore[j] == firstscore[k] == firstscore[l] == 0:
                final_injury[i] = 1 * firstscore[i]
            if (firstscore[k] == firstscore[l] == 0) and firstscore[j] != 0:
                final_injury[i] = 0.6 * firstscore[i] + 0.4 * firstscore[j]
            if firstscore[l] == 0 and firstscore[j] != 0 and firstscore[k] !=0:
                final_injury[i] = 0.5 * firstscore[i] + 0.25 * firstscore[j] + 0.15 * fi
            else:
                final_injury[i] = final_weight_0[0] * firstscore[i] + final_weight_0[1]

        if participantsname[i] == participantsname[j] == participantsname[k] != participa
            final_injury[i] = final_weight_1[0] * firstscore[i] + final_weight_1[1] * fi
            final_injury[j] = final_weight_2[0] * firstscore[j] + final_weight_2[1] * fi
            final_injury[k] = final_weight_3[0] * firstscore[k]

    df["Finalinjury"] = final_injury
    df["Finalinjury"] = np.ceil(df["Finalinjury"]).astype(int)
    df = df.drop("Firstscore", axis=1)
    return df

df = injurystatusdef(df)
```

B.4. Sliding Window

```
#------------------------------Sliding Window------------------------------
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        """
        Frame a time series as a supervised learning dataset.
        Arguments:
                data: Sequence of observations as a list or NumPy array.
        (this means that dataframe needs to be modified into a list or array)
                n_in: Number of lag observations as input (X).
                n_out: Number of observations as output (y).
                dropnan: Boolean whether or not to drop rows with NaN values.
        Returns:
                Pandas DataFrame of series framed for supervised learning.
        """
        n_vars = 1 if type(data) is list else data.shape[1]
        df1 = DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df1.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df1.shift(-i))
                if i == 0:
                        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
        # put it all together
        agg = concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg

test_df = df
test_df = test_df.drop("Participantsname",axis=1) #athlete name
test_df = test_df.drop("Recurrence",axis=1) #recurrence

values = test_df
data_sliding = series_to_supervised(values, 4)

del values, test_df, df_encoded, stats
```

B.5. Forward Chaining

```
#—————————————————————————————————Forward Validation——————————————————————————————————————

def walking_forward(data, n, personalised , athelete_name):
    #n = the number of entries wanted to be included in the data, do each the length of
    #personalised = 1 if input required is for personalised model

    if personalised == 1:
        dict_of_athletes = dict(tuple(data.groupby("Participantsname")))
        data_walking_X = dict()
        data_walking_Y = dict()

        athlete_data = dict_of_athletes[athlete_name]

        athlete_data = athlete_data.drop(["Recurrence","Participantsname"], axis=1)

        athlete_data_X = athlete_data.drop("Finalinjury", axis=1)
        athlete_data_Y = athlete_data("Finalinjury")

        for i in range(1,1600):
            data_per_X = athlete_data_X.head(i)
            data_per_Y = athlete_data_Y.head(i)
            data_walking_X[i] = data_per_X
            data_walking_Y[i] = data_per_Y

        test_X = athlete_data_X.tail(500)
        test_Y = athlete_data_Y.tail(500)

    if personalised == 0:
        #the data in recurrence and then athlete
        data_walking_X = dict()
        data_walking_Y = dict()

        athlete_data = data.sort_values(by=['Recurrence','Participantsname'])
        athlete_data = athlete_data.drop(["Recurrence","Participantsname"],axis=1)

        athlete_data_X = athlete_data.drop("Finalinjury", axis=1)
        athlete_data_Y = athlete_data["Finalinjury"]

        #slice the data base on the number of entries
        j = 0
        for i in range(1,1600,300):
            data_per_X = athlete_data_X.head(i)
            data_per_Y = athlete_data_Y.head(i)
            data_walking_X[j] = data_per_X
            data_walking_Y[j] = data_per_Y
            j = j + 1
        test_X = athlete_data_X.tail(500)
        test_Y = athlete_data_Y.tail(500)

    return data_walking_X, data_walking_Y, test_X, test_Y
```