Tampere University

Bahareh Darvishmohammadi

ROBOT JOINT TYPE RECOGNITION USING MACHINE LEARNING

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Jouni Mattila,
Assoc. Prof. Joni Kämäräinen,
Dr. Mohammad Mohammadi Aref
May 2020

ABSTRACT

Bahareh Darvishmohammadi: Robot joint type recognition using machine learning
Master of Science Thesis
Tampere University
Master's Degree Programme in Information Technology
Major: Data Engineering and Machine Learning
Examiners: Prof. Jouni Mattila, Assoc. Prof. Joni Kämäräinen, Dr. Mohammad Mohammadi Aref
May 2020

Reconfigurable robots and mountable measurement systems face attraction due to significant changes in development of wireless networks and internet of things. This research benefits cross disciplinary viewpoints to build a self-aware robotic module that utilizes data analysis and machine learning for a network of sensors mounted on an arbitrarily chosen serial manipulator.

Serial robotic arms consist of series of links and joints. Joints connect links to each other with possibility of generating rotational or translational motion. The goal of the study is to detect the mechanical joint type of robotic manipulators which can be rigid (body), revolute joints, prismatic joints, corrupted and random data via machine learning techniques. The dataset of robotics manipulators is collected via Aaria platform with diverse structures. Measurements and classifications are based on raw data collected from real and simulated sensors. These sensors are capable of providing absolute values of acceleration and angular velocity measured by Inertial Measurement Units (IMUs). Raw input data is transformed into useful features by feature extraction.

The machine learning models which are explored in this study consists of logistic regression, 5-nearest neighbors, linear discriminant analysis, random forest, XGBoost and LSTM models. The most accurate methods are XGBoost and LSTM with accuracy of approximately 69% and 70% respectively.

Keywords: machine learning, robotic, IMU sensors

PREFACE

The conducted study in this thesis was performed at Department of Automation and Hydraulic Engineering at Tampere University during the years 2018-2020.

I would like to express my gratitude to my supervisor Dr. Mohammad Mohammadi Aref for his excellent guidance during my thesis work. I also would like to thank my supervisor Prof. Jouni Mattila who provide me the opportunity to work on such an interesting topic. Additionally, I would like to thank my supervisor Associate Prof. Joni Kämäräinen who has being willing to provide support whenever required. Moreover, I want to thank my co-worker, Arttu Hautakoski at Department of Automation and Hydraulic Engineering for his contribution to this project.

Last but not least, I want to thank my parents, all my friends and specially my partner, Mika who supported me during this journey and encouraged me to pursue my goals.

Tampere, 27 May 2020

Bahareh Darvishmohammadi

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

5NN	5-Nearest neighbours
Acc	Accuracy
CNN	Convolutional neural networks
DNN	Deep neural network
FN	False negative
FNN	Feed-forward neural network
FP	False positive
fpr	FP rate
IMU	Inertial measurement units
KNN	K-nearest neighbours
LDA	Linear discriminant analysis
LR	Logistic regression
LSTM	Long short term memory
ML	Machine learning
MLE	Maximum likelihood estimation
PCA	Principal component analysis
Prec	Precision
Rec	Recall
RF	Random forest
RNN	Recurrent neural network
Spec	Specificity
TN	True negative
tnr	TN rate
TP	True positive
tpr	TP rate
XGBoost	Extreme gradient boosting

a	Linear acceleration
ω	Angular velocity

1. INTRODUCTION

Conventionally, robotic manipulators are designed and developed for performing a particular task. Theoretically, it is possible to reprogram a manipulator for a new task. Although, specific manipulator's configuration causes to apply the robot for limited number of tasks. For instance, a manipulator which is appropriate for moving objects on the table, does not have the capability of picking up heavy objects in vertical orientation [1]. It is also feasible to use manipulators with various configurations for a new task, if the task's prerequisite is identified. However, for uncertain conditions such as a construction site, nuclear equipment or a space station, manipulator requires more abilities and using manipulators with fixed configuration is not useful [2]. A solution for this problem is using a self-reconfigurable manipulator which can change their shape by readjusting the connection between their segments. For example, considering a space mission, there are limitation for sending objects with high mass and volume to the space. By sending reconfigurable robots to the space, it is possible avoid high consumption of mass and volume. Additionally, reconfigurable manipulators have three main benefits which are explained as following.

- Versatility: They can adapt themselves to new situation. The re-configurability make it possible for the robot to reassemble their segments in order to form a new shape which are more appropriate for a new task.
- Robustness: The robot is capable of substituting its faulty segments automatically which enables self-repair.
- Low Cost: instead of designing many different robots for different tasks, re-configurability allows use of the same robot for various tasks [3].

Furthermore, the aim of controlling of the robot is to perform a beneficial robotic task. In order to control a robot with high precision, knowledge about structure of the robot and kinematic connections is necessary [4]. Knowledge of robot's structure might not be always available. For example, large firms who construct the robotic manipulator and sell it to other companies for various applications, avoid providing information about robots structure because of confidentiality and commercial purposes. Therefore, without having knowledge about robot's structure, it is not possible to perform the robotic task.

In this work, the goal is specifically to recognize the mechanical joint type of the robot from the robot's movement data via machine learning techniques. Collecting a large dataset from movement of real robots is not only a time consuming task but also it is not possible to extend the solution to robots with different configuration. Therefore, the robots movement data is gathered through reconfigurable manipulator Simulator called Aaria [5] platform. Aria performs simulation for serial manipulators in Simulink SimMechanics environment. It provides artificial data from robot's movement and can learn deep features that are typical for the robotics structures. The details of simulation procedure of artificial data generation is explained in [5], [4].
Moreover, this thesis provides a solution for recognizing kinematic connection for the reconfigurable manipulators by analysing the data which comes from IMU sensors. Consequently, the major emphasis of this thesis is the recognition of manipulator's joint type on the basis of IMU sensory data. The reason why IMU sensors are appropriate option is that not only, they are small and easy to set up but also they are applied on variety of real-world problems. On the contrary, the disadvantage of IMU sensors is that they can produce very noisy output signals [6].

The rest of the thesis is organized as following. In chapter 2, theoretical background of the study is reviewed. This chapter consists of definition of machine learning (ML) and its techniques, performance evaluation of the ML models and feature extraction. In chapter 3, the methodology of the study, different phases of the experiment are described and acquired results are discussed. Last but not least, the conclusion of the work and possible future improvements are discussed in chapter 4.

2. THEORETICAL BACKGROUND

2.1 Machine Learning

A field of computer science which relates human learning ability with computer programs referred as Machine Learning. The phrase machine learning was invented by Samuel in 50's century. It includes intelligent actions which could be delivered from human to machine. The phrase machine does not mean a physical machine but an automatic system [7].

Machine Learning is defined mainly as a computer program which learns from its experience and promotes its efficiency. One of the definitions which broadly used for machine learning is as below:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [8]

Machine Learning methods are categorized as supervised learning, unsupervised learning and semi-supervised learning. Supervised learning is used broadly in machine learning which training data consists of two variable pairs (x, y). The aim is to predict y with respect to input x. Furthermore, the mapping function f(x) is the learning function which used for prediction and generate the output y with respect to input x. There exists various types of mapping functions. Some example of mapping functions comprised of decision trees, random forest, logistic regression, support vector machines, neural networks and Bayesian classifiers. Commonly, unsupervised learning incorporate pattern discovery in data which does not have corresponding target values. Various kinds of unsupervised learning tasks include clustering, dimensionality reduction, manifold learning, and density estimation, etc. For instance, clustering is the task of identifying groups in observed data which each group has similar samples [9]. Finally, semi-supervised learning is a method which the data contains both labelled and unlabelled samples. Labelled and unlabelled data are used to enhance the accuracy and models on training data and unseen samples. Semi-supervised learning is very beneficial in practice since labelling the data requires a lot of human effort and gathering unlabelled data is a lot easier. Consequently it can lower the expenses significantly [10].

Supervised learning techniques are divided into two categories of classification and regression methods. For Classification methods the output variable y is a discrete variable. While, in regression methods the output variable y is a real-valued variable.

Furthermore, the data labels are accessible for supervised learning methods within the models training and testing phases. Assume that input variables $x_1, x_2, \cdots, x_n$ are allocated to classes $C_1, C_2, \cdots, C_n$. As a result, the classification tasks can be partitioned into the following categories:

- Binary: In this category, there are two distinct classes C_1, C_2 and the input can be classified into one of these classes. Binary classification is a very common task in machine learning.
- Multi-class: In this category, there are several distinct classes $C_1, C_2, \cdots, C_n$ and the input can be classified into one of these classes.
- Multi-labelled: The input can be classified into several of the distinct classes.
- Hierarchical: The classes are partitioned into subclasses or organized into super classes and the input can be classified into one of these classes [11].

Universally, the process of learning consists of three phases including dividing the dataset into three parts which is referred as training, validation and test sets. Firstly, training sets are instances of data which is used to build the machine learning model. Moreover, validation sets include instance of data which is used to tune the hyper-parameters of the model. Last but not least, test sets consist of instances of data which is used to evaluate the performance of the model. Furthermore, the labeled data obeys the same rules which has described as above.

2.2 Classification Methods

2.2.1 Logistic Regression

Often identifying relationship between response attribute and explanatory attributes is essential part of data analysis. Regression techniques are used for recognizing this relationship. Logistics regression is a famous regression technique which the target attribute is discrete. Logistic regression is similar to linear regression in the sense that they both belong to generalized linear model category. However, there are differences between linear regression and logistic regression. The type of target variable is discrete for logistic regression but linear regression has continues output. In addition, logistic regression uses maximum likelihood estimation (MLE) method, while linear regression is estimated through ordinary Least squares approach.

Logistic regression is mostly applied to binary classification tasks but it is also possible to extend it for multiclass classification task. It is a specific case of linear regression.

However, logistic regression uses logistic function instead of hyperplane (in linear regression) to estimate the probability of binary classes. The logistic or sigmoid function is defined as formula (1) and logistic curve is represented in Figure 1. The logistic function is also called sigmoid function which can map the output of linear regression formula to a number between 0 and 1. Formula (2) explains linear regression equation where y is response variable and $x_1, x_2, \dots, x_n$ are explanatory variables. By applying the sigmoid function on linear regression formula, equation (3) can be obtained which estimates the probability of classes in logistic regression. The reason for this substitution is that classification task requires a probability of classes instead of real-valued number [12].

$$f(x) = \frac{1}{1+ e^{-(x)}} \tag{1}$$

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \tag{2}$$

$$p = \frac{1}{1+ e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n)}} \tag{3}$$

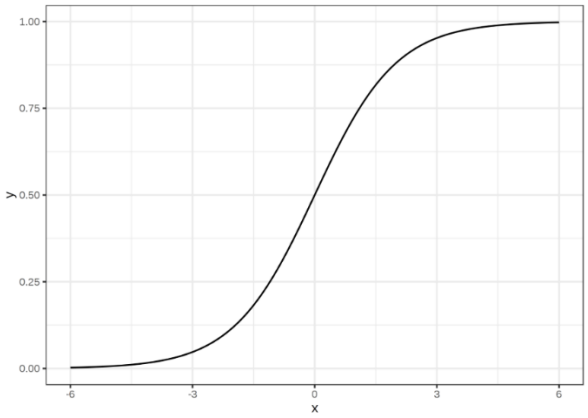


Figure 1. *Sigmoid function. Figure borrowed from [12].*

2.2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a methods for classification and is categorized as a member of linear classifiers. It is proposed by popular statistician Fisher in 1936 [13]. LDA is applied to problems which there is a possibility of separating classes with linear decision surface. The reasons why LDA is interesting is that it provides a solution which

does not take so much computational power, is designed for multi-classification problems and does not require hyper-parameters tuning [14].

Furthermore, LDA is beneficial when the intra-class frequencies are not equal. This technique maximizes the proportion of the inter-class variance to intra-class variance on any specific sort of dataset and ensures that classes and separable in as much as possible. Additionally, it assists to have a better insight to the distribution of the data. Figure 2 demonstrates that LDA separate the classes using linear decision boundary [15], [14].

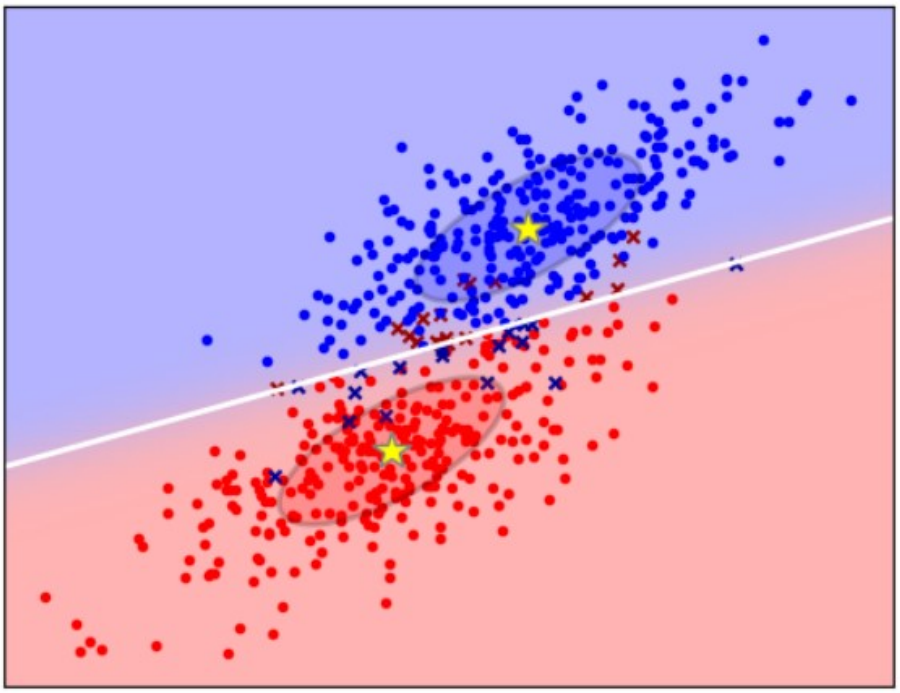


Figure 2. *LDA uses a linear decision boundary for separating the classes. Figure borrowed from [14].*

2.2.3 K-Nearest Neighbors

K-nearest neighbor (KNN) is very popular techniques because of its simplicity and high training performance. It has achieved great success in many classification and regression problems such as handwritten digits and satellite image scenes. The major idea underlying KNN is to discover K number (a pre-chosen integer number) of training samples which has closest distance to the new test sample and chooses the class label with majority of vote among them. For instance, supposing, there are two distribution of classes in a binary classification problem which is illustrated in Figure 3. The distribution of red points and blue points belongs to class A and B respectively. The new green point

could be classified as either class A or class B. As can be seen from the Figure 3, if 3-nearest neighbor is applied for classifying the green point, two blue points from class B and one red point from class A, having smallest distance two the green point. Consequently the new green point belongs to class B which has the majority of votes.

It is usually common to select the K greater than one since higher number of K makes the classifier more robust to outliers. Furthermore, KNN is a lazy learning algorithm which means that instead of constructing a function of training data, it memorizes the training data and use it in the testing stage. As a result, testing stage will be slower than training stage since testing requires all the training data in the memory [16], [17], [18].

In order to define the dissimilarity between samples, distance metrics which are based on characteristic of samples, can be calculated. There exists various distance metrics such as Euclidean distance, Chebyshev distance and Manhattan distance. Euclidean distance is the most frequent choice of distance and its mathematical formula can be obtained from formula (4). Chebyshev distance can be explained as highest value of difference of two vector on any coordinate dimension. The mathematical definition of Chebyshev and Manhattan distance is explained as formula (5) and (6) respectively.

$$E(x,y) = \sqrt{\sum_{i=1}^{k}(x_i - y_i)} \tag{4}$$

$$C(x,y) = \max(|x_i - y_i|)\ ,\ i = 1,2,\dots,n \tag{5}$$

$$M(x,y) = \sum_{i=1}^{k}|x_i - y_i| \tag{6}$$

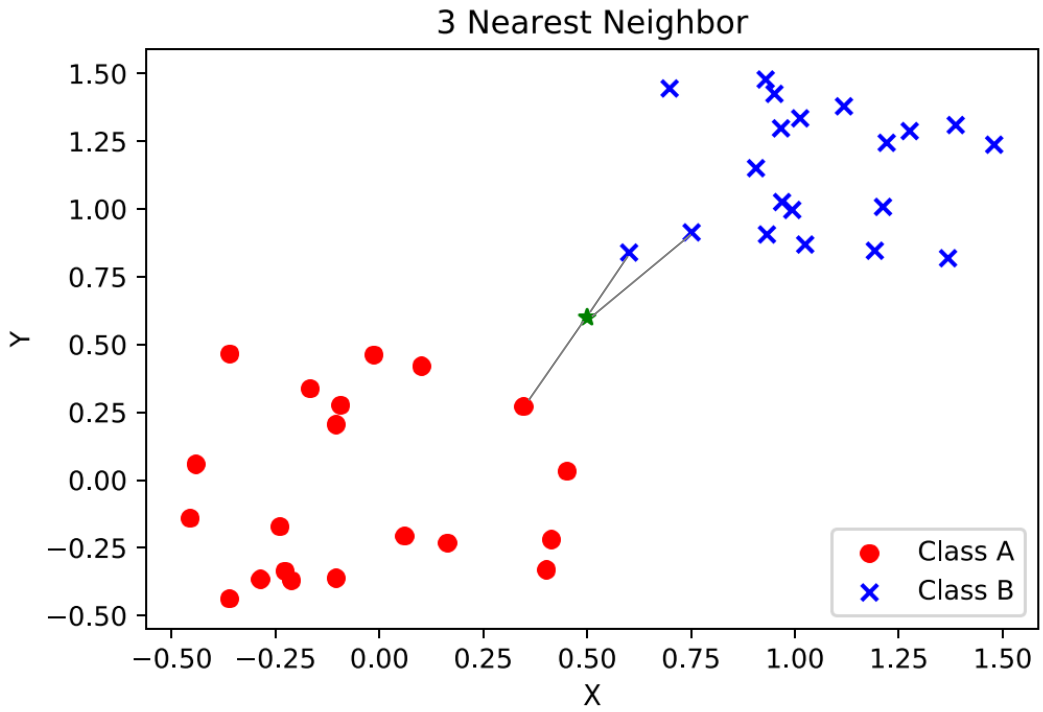


Figure 3. *3-Nearest neighbors of a new sample*

2.2.4 Random Forest

Random Forest (RF) is an ensemble method based on decision trees which is proposed by Breiman in 2001 [19]. Ensemble methods are techniques which merge the output of multiple classifiers in order to promote performance of the model [20]. The disadvantage of decision trees is that they can cause overfitting which means training data can be memorized by the model with the weak capability to generalize on unseen samples. RF overcome this problem by learning many imperfect trees. In other words, RF shuffle training samples and choose subset of them. After that, each decision tree is trained with selected training samples and attributes. Finally, it predicts the unseen samples by taking majority of votes among classes.

2.2.5 XGBoost

XGBoost is short phrase for extreme gradient boosting. It is an implementation of Gradient boosting machine which is proposed by Friedman in 2001 [21]. Moreover, XGBoost is originally written in C++ language. It is used for different tasks such as regression, classification and ranking [22].

Gradient boosting is one of the machine learning techniques which has applied to various problems and demonstrates the state-of-the-art performance on many classification benchmarks. It has applied in challenges such as Netflix prize and it has been successful many times. Moreover, it is one of the existing methods from the category of ensemble classifiers. The most significant reason why XGBoost has been successful, is its scalability in all situations. It executes more than ten times quicker than other available famous solutions on a single machine. One of the reasons of scalability of XGBoost is the algorithmic optimization. It consists of innovative tree learning algorithms which can manipulate spars data. Moreover, with the parallel and distributed computing capability of XGBoost the learning process will become faster. Consequently, the machine learning model will be generated with higher speed [23].

The principle behind boosting algorithms is that it converts many weak learners into strong learners. A weak learner is a machine learning algorithm which is slightly better than chance. Strong learner is a machine learning algorithm which can achieve good performance. Boosting is fulfilled by learning a collection of weak models on the subsets of the data. Moreover, a weight is assigned to each weak learners according to their performance. After that, all weak learner prediction is multiplied by their weights, acquiring single weighted prediction which is much better than any of the weak learner's prediction [24].

2.2.6 Artificial Neural Networks

Artificial neural networks, demonstrate widely various techniques and algorithm such as classification, regression, auto-association, signal processing, time series application and clustering. Neural networks proposed by McCulloch and Pitts in 1943 [7]. The idea of neural network motivated by biological neural model primitively. However, it has moved to a different direction and become the subject of data science and machine learning. Figure 4 represents a biological neuron. As can be seen from the figure, dendrites is input signal for an individual neuron and output signal is generated alongside axon. Finally, axon is split to different branches and synapses link axon to dendrites of other neurons [25].

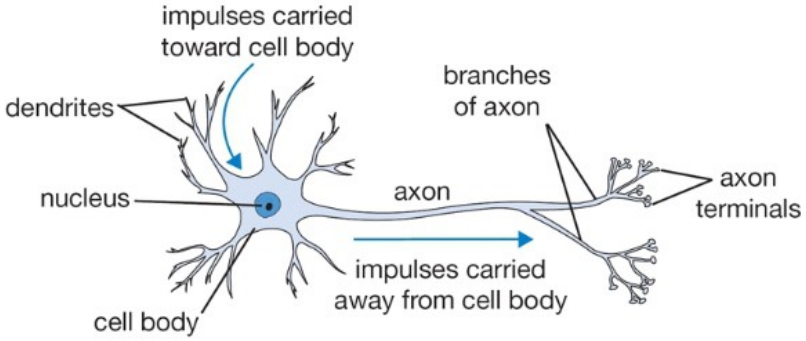


Figure 4. *A biological model of neuron. Figure borrowed from [25].*

The model of an artificial neuron is represented in Figure 5. As can be seen from the figure, each artificial neuron has several inputs which comes from other neurons through links. Each link has a weight which is similar to intensity of synapse in model of biological neuron. The model of a neuron consists of sum of dot product of inputs and weights and if it goes beyond a specific threshold, the neuron is activated and sends the signal through an activation function. After that, it is sent to adjacent neurons. Moreover, in mathematical terms it can be explained as $y = f(\sum_{i=0}^{n} w_i x_i - T)$ where y, f, w and T are output of the neuron, activation function, weights and threshold respectively [26].

The simplest architecture of neural network is called feedforward network which consists of several layers of neurons. The most prevalent kind of layers are fully connected layers in which two neighbouring layers are linked to each other in a pairwise manner. Figure 6, illustrates this feedforward network which is a 3-layer network. It contains three inputs, two hidden layer and one output layer [25].

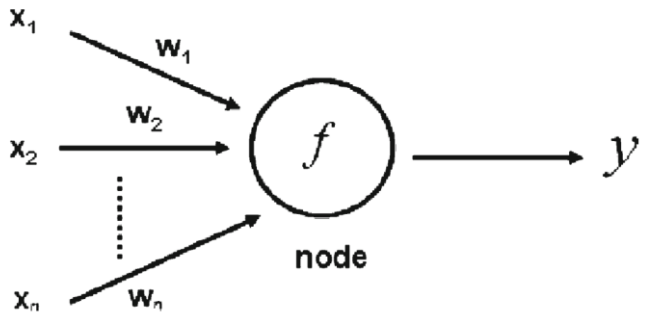


Figure 5. *Model of an artificial neuron. Figure borrowed from [26].*

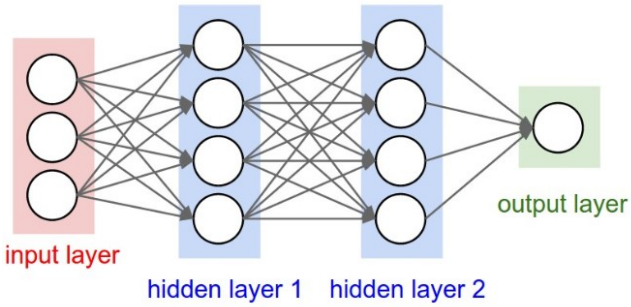


Figure 6. *Example of feedforward neural network. Figure borrowed from [25].*

Training the neural network consists of two stages: forward propagation and backward propagation. During forward pass, an input is fed into neural network and output of the network is generated. Backward propagation is called backpropagation which is an efficient neural network algorithm. In backpropagation phase, the gradient of loss function with respect to weights is calculated and weights are updated to minimize the loss function.

Another type of neural networks are recurrent neural networks (RNN) which are suitable for analysing time series data. The architecture of RNNs are similar to multi-layer feedforward neural networks. However, the difference is that there are links between hidden layers which make the network capable of maintaining information about to the past. Originally, the RNNs are simple and potential models but it is difficult to train them because of exploding and vanishing gradient [27].

Figure 7 represents the diagram of recurrent neural network (RNN). As can be seen from the figure, first input x is fed into the RNN unit. In the next step, the internal hidden state is updated. Finally, the output is produced. This process is repeated when RNN unit reads a new input. Moreover, Figure 8 illustrates RNN computational graph at each time step. As can be seen from the graph, sequence of inputs is fed into the network. The recurrence relation can be expressed in mathematical term as function f, where h_t is new hidden state, f_W is a function with parameter W, h_{t-1} is previous hidden state and x_t is input vector at a time step. It should be noted that the same function f_W and weight W is used at every time step of computation.

$$h_t = f_W(h_{t-1}, x_t) \tag{7}$$

The simplest kinds of recurrent neural network (RNN) is called vanilla RNN. In mathematical form, it can be explained as the formula (8). In this formula, weight matrix W_{hh} is multiplied by previous hidden state and another weight matrix W_{xh} is multiplied by input

at that time step, then we add two expression together and squash it thought tanh function to have non-linearity in the system. Additionally the output at each time step can be obtained from formula (9) where y_t, W_{hy} and h_t are output at a time step, weight matrix and current hidden state respectively [28].

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{8}$$

$$y_t = W_{hy}h_t \tag{9}$$

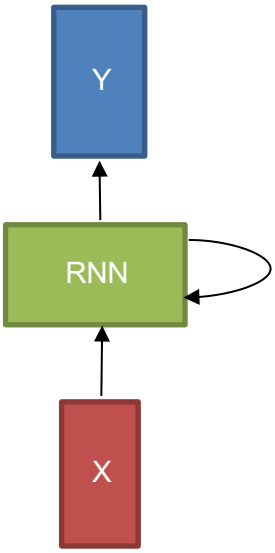


Figure 7. *Diagram of recurrent neural network. Figure borrowed from [28].*

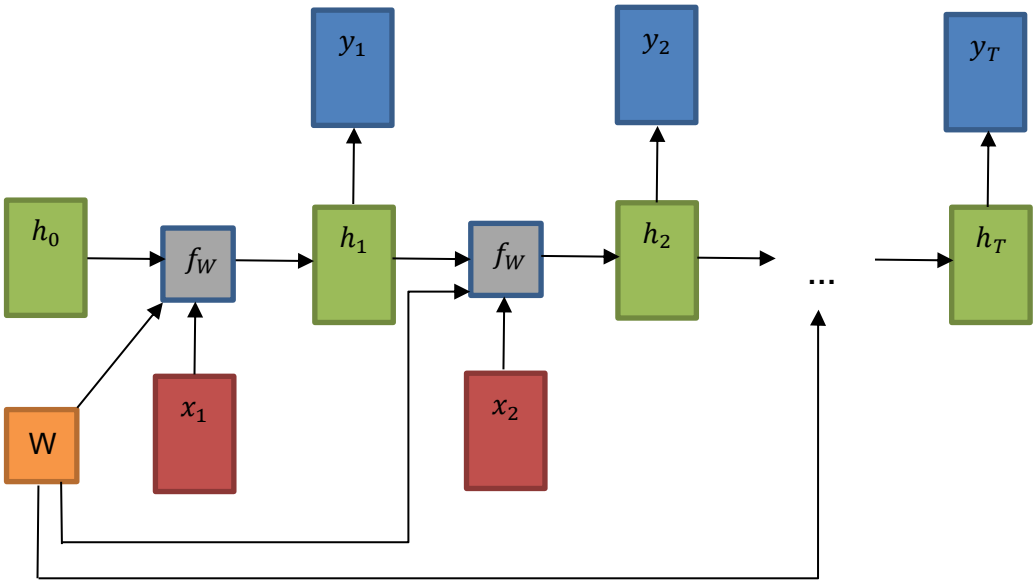


Figure 8. *RNN computational graph. Figure borrowed from [28].*

There are more complex RNN architectures which solve the problem of exploding and vanishing gradients. One example of such models is Long Short-Term Memory (LSTM) which is capable of obtaining long-term dependencies [29]. The LSTM architecture is proposed by Hochreiter and Schmidhuber in 1997 [30]. The core concepts underlying LSTM topology are memory cell and non-linear gates [29].

As has been described, the vanilla recurrent neural network (RNN) had one hidden state which could be updated at each time step. LSTM retains two hidden states at every time step of the computation. One hidden state is h_t which is similar to vanilla RNN and the other vector is cell state c_t which is an internal LSTM vector. In addition, LSTM has four gates which are called f (forget gate), i (input gate), o (output gate) and g. Each of these gates has different tasks. Input gate decides how much input comes to the cell, forget gate decides how much of cell memory will be forgotten at the previous time step, output gate decides how much output will goes out and g gate decides how much data is written to the input cell. Furthermore, different activation functions are applied to each gates to have non-linearity in the system. Input, forget and output gate use sigmoid activation and g gate uses tanh activation function which means that the output values of input, forget and output gate are either 0 or 1 and the output values of g gate is either -1 or 1 [28].

The internal diagram of LSTM can be seen in the Figure 9. The cell state and current hidden state can be obtained from formula (11) and (12) respectively. As can be seen from formula (11) in the first term, the previous cell state is multiplied element wise by the forget gate. As the forget gate is a vector of zeros and ones, this multiplication means that whether the element of previous cell state should be forgotten in case the forget gate is zero or the element of previous cell state should be remembered in case the forget gate is one. In the next term of the formula (11), input gate and g gate are multiplied elementwise. As the input gate is the vector of zeros and ones, this multiplication means that for each element of cell state whether should be written on that element in case input gate is one or the element of cell state should be kept the same in case input gate is zero. Moreover, as the g gate has the values of either 1 or -1, the second term means that whether the cell state should be incremented or decremented by one. Consequently, from the formula (11) it can be concluded that the pervious cell state should be forgotten or remembered either decremented or incremented by one at each time step. In other words, the cell state is like a scaler integer counter. Now that the cell state is calculated, it can be applied to the formula (12) for calculating the hidden state. The cell state the squashed through a tanh activation function which result in a value between zero and one. Then the result value is multiplied by output gate [28].

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \tag{10}$$

$$c_t = f \odot c_{t-1} + i \odot g \tag{11}$$

$$h_t = o \odot \tanh(c_t) \tag{12}$$

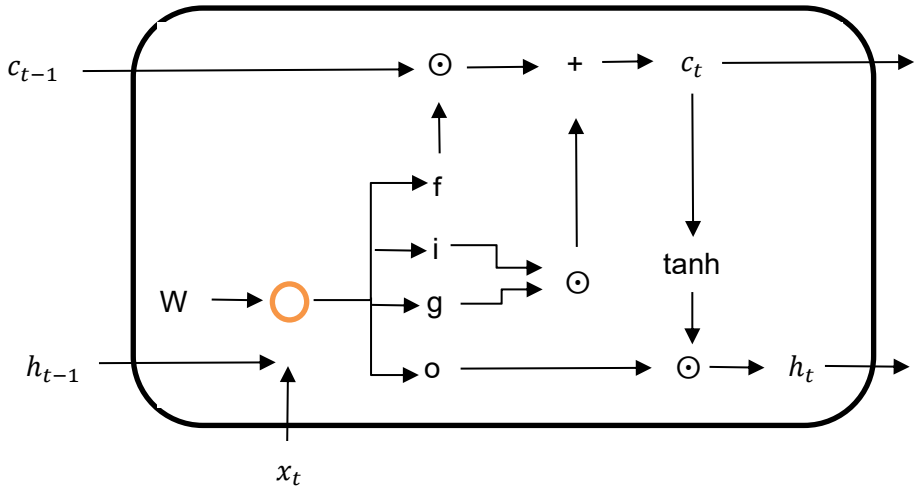


Figure 9. *LSTM Internal Diagram. Figure borrowed from [28].*

2.3 Performance Evaluation

Evaluation of the performance of the model includes approximation of how precisely the model predicts labels on the unseen data. This process includes the comparison of predicted labels and true labels [31]. Model performance is one of the significant factors in learning process.

Generally, multiple classifiers can be trained on the same dataset and comparison of their performance is one perspective of performance evaluation. In the other words, for a specific problem several classifiers might be applied and the best technique in terms of error rate and computational efficiency, can be selected. Although, it is not feasible to choose the best classifier for all type of datasets since classification performance depends on different factors such as sample size, dimensionality of the data and the competences of data analyst [32].

2.3.1 Confusion Matrix

Various performance evaluation techniques are introduced in books and literatures. The most common method is confusion matrix visualization. Confusion matrix illustrates the number of correct and incorrect prediction for each of the classes. For simplicity, confusion matrix for binary classification task is presented in this thesis and it is shown in Table 1. As can be seen from the table, confusion matrix consists of four partition called true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Each partition has defined as following.

- True positive: the number of correctly predicted instances as positive class
- True negative: the number of correctly predicted instances as negative class
- False positive: the number of incorrectly predicted instances as positive class
- False negative: the number of incorrectly predicted instances as negative class

Table 1. *Confusion matrix for binary classification problem. Table borrowed from [32].*

		Predicted classes	
		Positive	Negative
Actual classes	Positive	True positive	False negative
	Negative	False positive	True negative

In fact, other performance evaluation measures such as accuracy, recall and precision can be derived from confusion matrix. Table 2 represents lists of prevalent metric which are computed based on confusion matrix.

Table 2. *Performance measures calculated based on confusion matrix. Table borrowed from [32].*

Evaluation measure	Description
Accuracy (*Acc*)	$\frac{TP+TN}{TP+FN+FP+TN}$
FP rate (*fpr*)	$\frac{FP}{FP+TN}$
TP rate (*tpr*) or Recall (*Rec*)	$\frac{TP}{TP+FN}$
TN rate (*tnr*) or Specificity (*Spec*)	$\frac{TN}{FP+NT}$
Precision (*Prec*)	$\frac{TP}{TP+FP}$

Even though confusion matrix is a good representation of classification performance, in order to compare performance of various models conveniently, confusion matrix can be converted to a single number which is called accuracy of the model. Accuracy is defined as the ratio of correctly classified instances to all the instances and can be obtained from formula (13) [31].

$$Accuracy = \frac{The\ number\ of\ correctly\ classified\ instances}{Total\ number\ of\ instances} \tag{13}$$

False positive rate (*fpr*) is defined as proportion of negative samples which is incorrectly classified as positive samples. True positive rate (*tpr*) or recall (*Rec*) is explained as proportion of positive samples which are correctly classified as positive. Additionally, true negative rate (*tnr*) or specificity (*Spec*) is the ratio of negative samples which are correctly classified as negative. In other words, false positive rate is simply equal to “1-Spec”. Precision (*Prec*) is the proportion of true positive to the total number of samples which are classified as positive. Moreover, precision specifies the capability of the classifier not to predict a negative sample as positive [33].

2.3.2 Cross Validation

Cross validation is one of the most common data resampling techniques which is used to evaluate the performance of the model and to prevent overfitting [34]. Cross validation procedure consists of dividing the entire data into k equal partitions. After that, classifier is trained with k-1 partition, the remaining kth partition is used as test set and the testing accuracy is calculated. This procedure is repeated until every partition is used as a test set. The performance of the model is equal to average of recorded testing accuracies. This process called k-fold cross validation. Figure 10 is a schematic of 5-fold cross validation [35].

Cross validation is among the most accurate measures for evaluating the performance. However, one of the drawbacks of cross validation method is that it demands significant amount of computations [32].

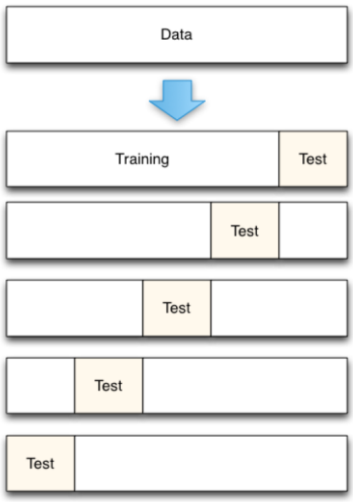


Figure 10. *Diagram of 5-fold cross validation. Figure borrowed from [35].*

2.4 Feature Extraction

Feature corresponds to an input attribute. Although there is difference between raw inputs attributes and features. Features are attributes which are built for raw input variables. Feature building techniques often consist of transmitting the raw data into beneficial information. In some methods the process of constructing the feature is incorporated in the modelling phase. While in some other techniques, constructing the features is included in the preprocessing stage. For instance, in artificial neural networks modelling, hidden layers of the network build features internally. In preprocessing phase, assuming

that $X = [x_1, x_2, \dots, x_n]$ is a vector with *n* dimension. The x_i members, represent the original features. After converting vector x to x′ which is the feature vector, it will have n′ dimensions. Methods which can be included in pre-processing stage are standardization, normalization, signal enhancement, extraction of local features, linear and non-linear space embedding methods, non-linear expansions and feature discretization.

Feature extraction is a very important stage in data analysis procedure, mainly because it can have a significant influence on achievement of statistical or machine learning models. One should avoid removing beneficial information while building features. The raw features can be included in the preprocessed data in order to evaluate the performances of different models and differentiate them. It is always better to include more information than to eliminate important data [7].

3. ANALYSIS AND RESULTS

The goal of this section is to explore a machine learning model for detecting mechanical joint type of the robotic manipulator. The mechanical joint types can be rigid (body), revolute, prismatic, corrupted or random. The proposed methodology consists of two stages of feature extraction and classification. The classification phase is performed using both traditional and modern machine learning techniques[1]. In the first phase, useful features are extracted from raw input data on the basis of mechanical equations. In the next phase, joint types are classified by using both raw input data and feature engineered data. Finally, the results of classification task on raw input data and feature engineered data are compared with each other. Additionally, in this phase features importance are calculated by using permutation importance technique.

3.1 Data Collection

The data is collected via “modular and reconfigurable simulation platform for robotic manipulators” [5] called Aaria. Aaria is a model which is simulated using Matlab Simulink SimMchanics tool with diverse structures. The simulation model can be configured with parameters for the mechanical joint types. The aim of data collection is to gather sensor data from various manipulators. It is also possible to gather sensor data via the physical manipulators. However, it is difficult to collect a large dataset because it is too time consuming. Therefore, a reconfigurable simulator is constructed which can convert its structures via changing its parameters. Two example of configured manipulators with different joint types are demonstrated in Figure 11 and Figure 12 [5]. The artificial data simulation process is described in more details in [5].

[1] Traditional machine learning techniques refers to algorithms such as LR, KNN, etc. which has been studied for decades and they are the foundation of machine learning while modern machine learning techniques include deep neural networks.

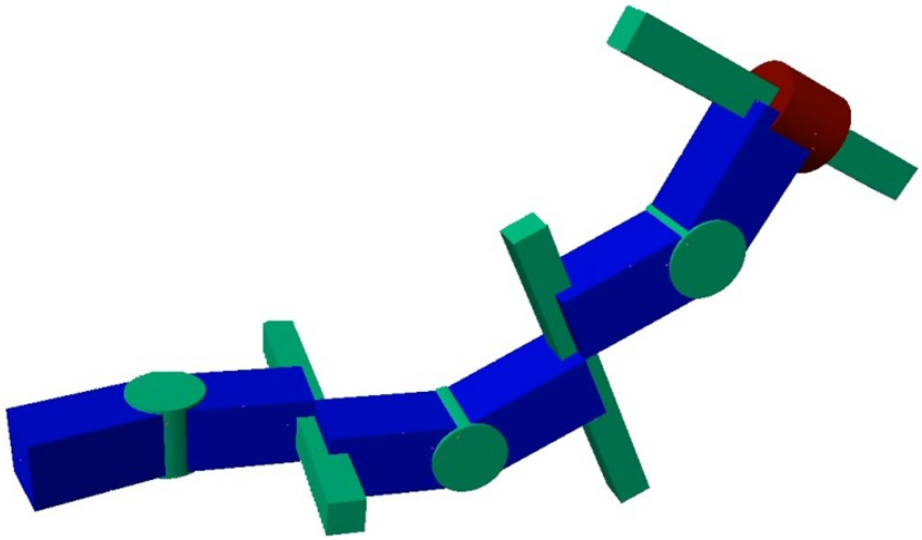


Figure 11. *An example of configured manipulator with three prismatic and three revolute joints. Figure borrowed from [4]*

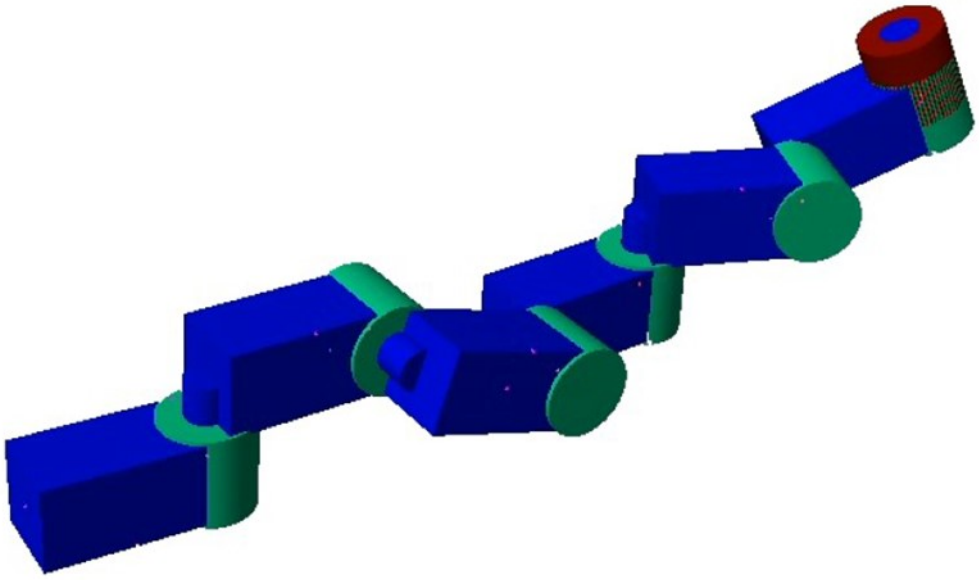


Figure 12. *An example of configured manipulator with six revolute joints. Figure borrowed from [4]*

The data is measured using inertial measurement unit (IMU) sensors. In total, four IMU sensors is used to measure the data. Two IMU sensors is placed before the joint and two sensors after the joint which is represented in Figure 13. Each sensor output angular velocity and linear acceleration in the X, Y and Z axes. The output of the each sensor

can be seen from the Figure 14. As can be seen from figure 14, each sensor output 6 variables. As there are 4 sensors, the number of outputs from all the sensors will be 24. Additionally, sensors record the data every 0.01 second. Consequently, each instance of data is a 100×24 length vector.

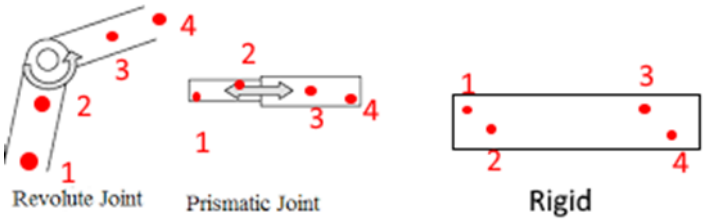


Figure 13. *Placement of IMU sensors for each joint*

Angular velocity X	Angular velocity Y	Angular velocity Z	Linear-Accelertion X	Linear-Acceleration Y	Linear-Acceleration Z

Figure 14. *Outputs of each IMU sensors*

The dataset consists of 720418 samples, 24 input variable and 1 target variable. The target variable includes the mechanical joint type of the manipulator, which can be a rigid body, revolute joint, prismatic joint, corrupted joint or random data. Rigid body is a type of joint which does not move or bend. Revolute joint are joints which turn like a knee. Prismatic joints are joints that extend like a piston. Corrupted joint type are simply artificially generated joint type by randomizing sensor data and random joint type is artificial random data. Corrupted joint types are generated artificially to emulate lack of consistent sensory data. The benefit of having corrupted and random data is that they can avoid presumed validation of sensory data. Target variable can be one of the numbers 0, 1, 2, 3, 4 which is defined as following.

- 0: Rigid joint
- 1: Revolute joint
- 2: Prismatic joint
- 3: Corrupted
- 4: Random

3.2 Preprocessing

One of the challenges in data analysis is dealing with high dimensional data. This problem called curse of dimensionality. Although, by increasing the number of dimensions, the dimension space grows, the number of instances stays the same. Therefore, density of data is reduced and this can have a negative effect on data modelling such as overfitting. In order to avoid such issues, dimensionality reduction techniques such as principle component analysis (PCA) can be applied. However, this technique was not beneficial in case of reconfigurable manipulator data since it causes losing of information and reducing the accuracy of the model after PCA. Therefore, a specific technique is proposed for this purpose which is explained in this section.

Preprocessing of the dataset consists of two step. Some parts of the data contain very high values which can be due to error in simulation phase. Therefore, in the first step of preprocessing in order to make the data consistent, high positive values are replaced with 20 and high negative values are replaced with -20. The second step is for reducing dimensionality of the data. As mentioned in the previous section, each instance of data is a 100×24 length vector. In the second step of preprocessing, for reducing the dimension of the data, each input variable which is a 100 length vector decreased to a 10 length vector by taking the mean of the each 10 values. Consequently, each 100 length vector is converted to 10 length vector. This brand new dataset is called mean dataset which has 240 inputs and will be used for data modelling.

3.3 Feature Extraction

As mentioned in the second chapter, feature extraction is an essential part of data analysis. In this thesis, features are extracted from mean dataset and original dataset based on mechanical equations. In this process several input data are combined and features are built from them. The goal of feature extraction is to increase the accuracy of proposed machine learning models. There exist four categories of features. The first category of features are created on the basis of angular velocities subtraction for each X, Y and Z axes in 3D space. Additionally, as the goal is to distinguish the joint type, the relevant angular velocities for subtraction task are the ones which are coming from IMUs placed before and after the joint of manipulator (for example IMU1 and IMU3). The first feature category can be formulated by equations (14) and (15). In total, 40 features are extracted in this category. The second category of features are extracted based on mechanical formula which is cross product of linear acceleration and angular velocity. The second category of features can be obtained from equation (16). In sum, there are 40 features

in this category. The third category of features is built based on subtraction of angular velocities coming from IMUs before and after the joint (for example IMU1 and IMU3) and features are extracted by subtracting the first element and tenth element of subtraction vectors in each X, Y and Z axes. Additionally, other sets of features are created in this category by subtracting the first element and 5th element of subtraction vectors in each X, Y and Z axes. In total, there are 12 features which belongs to this category. Finally, the forth category of features are extracted from the original dataset by addition of first 50 elements of linear accelerations in each X, Y and Z axes. In addition, other sets of features are created by addition of 100 elements of linear accelerations in each X, Y and Z axes. The forth category of features has 24 elements and can be formulated as equations (21) and (22). In total, 116 features are extracted from mean and original dataset.

$$\|\omega_1 - \omega_3\|_2 + \|\omega_1 - \omega_4\|_2 \quad (14)$$

$$\|\omega_2 - \omega_3\|_2 + \|\omega_2 - \omega_4\|_2 \quad (15)$$

$$(a \times \omega) \quad (16)$$

$$(\omega_1 - \omega_3)_1 - (\omega_1 - \omega_3)_{10} \quad (17)$$

$$(\omega_1 - \omega_3)_1 - (\omega_1 - \omega_3)_5 \quad (18)$$

$$(\omega_2 - \omega_4)_1 - (\omega_2 - \omega_4)_{10} \quad (19)$$

$$(\omega_2 - \omega_4)_1 - (\omega_2 - \omega_4)_5 \quad (20)$$

$$\sum_{i=1}^{50} a_i \quad (21)$$

$$\sum_{i=1}^{100} a_i \quad (22)$$

3.4 Data Visualization

The aim of data visualization in this section is get insight into feature extraction task. As mentioned in the previous section, one of the feature categories are extracted based on subtraction of angular velocities which are coming from IMUs before and after the joint (for example IMU1 and IMU3). Additionally, this subtraction is extended to linear accelerations. The features which are chosen for visualization are formulated as equations (23), (24), (25), (26). Figure 15, Figure 16, Figure 17, Figure 18 and Figure 19 demonstrate subtractions of angular velocity and linear acceleration for rigid joint, revolute joint, prismatic joint corrupted and random respectively. Each figure consists of six plot which three of them belongs to angular velocities and the remaining ones belong to linear accelerations. For example, in Figure 15, the three left plots show subtraction of angular velocities and the right three plots show subtraction of linear accelerations for combinations of X, Y, Z axes. Five instances of data for each joint are selected randomly for

visualization. In each plot the same colour indicates the data which belong to the same instance.

$$\omega_1 - \omega_3 \tag{23}$$

$$\omega_2 - \omega_4 \tag{24}$$

$$a_1 - a_3 \tag{25}$$

$$a_2 - a_4 \tag{26}$$

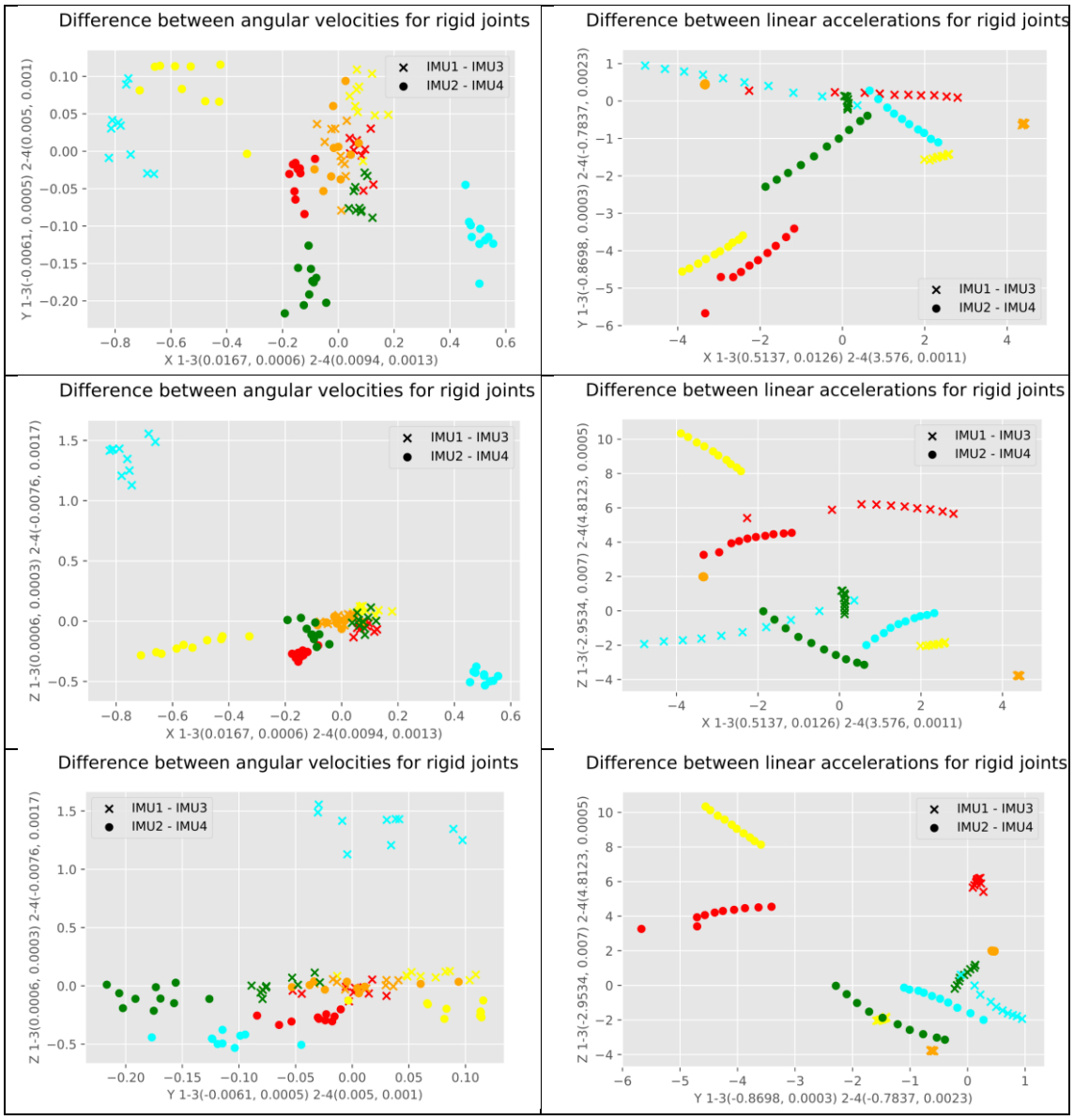


Figure 15. *Subtractions of angular velocities and linear accelerations for rigid joint*

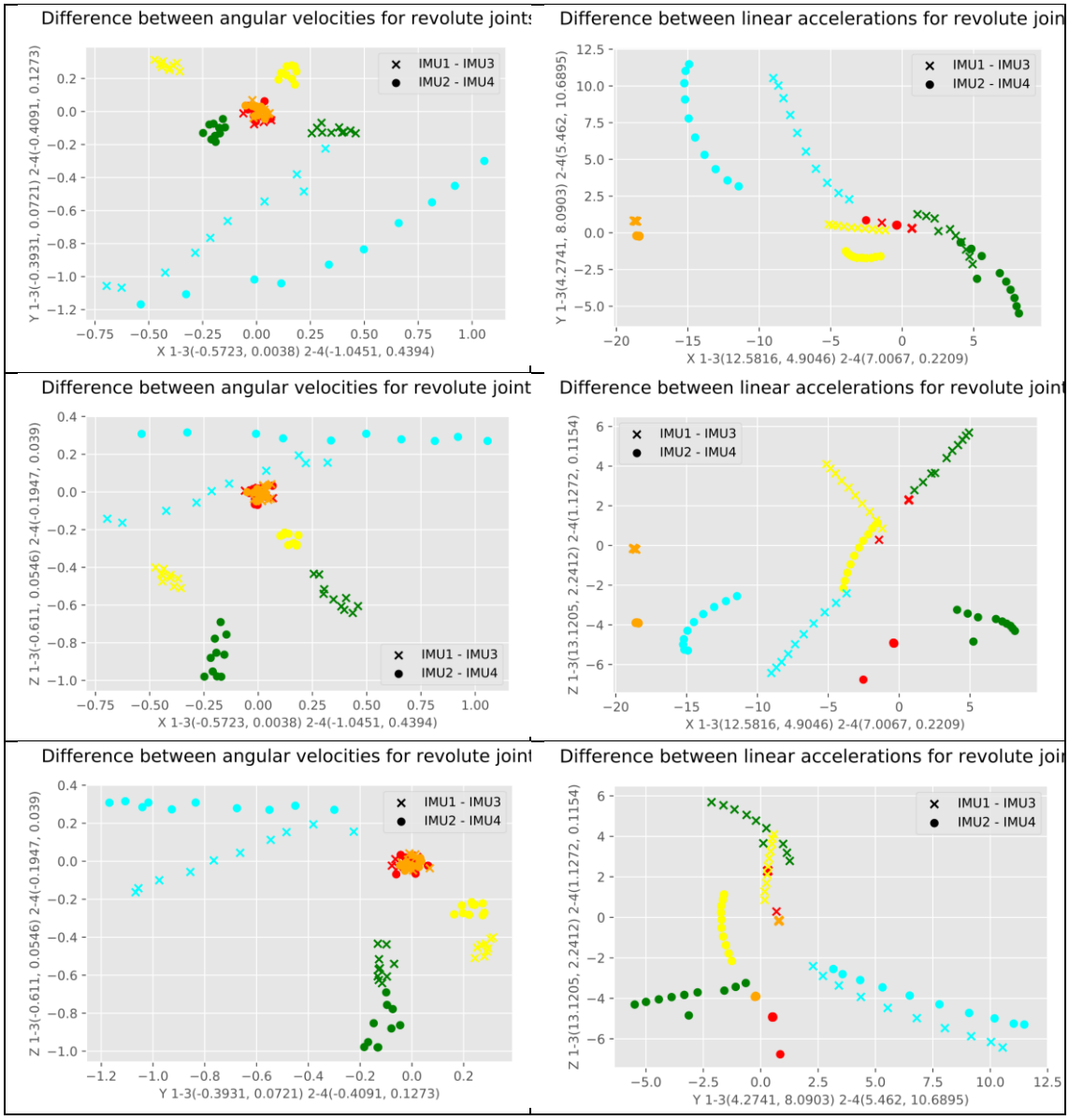


Figure 16. *Subtractions of angular velocities and linear accelerations for revolute joint*

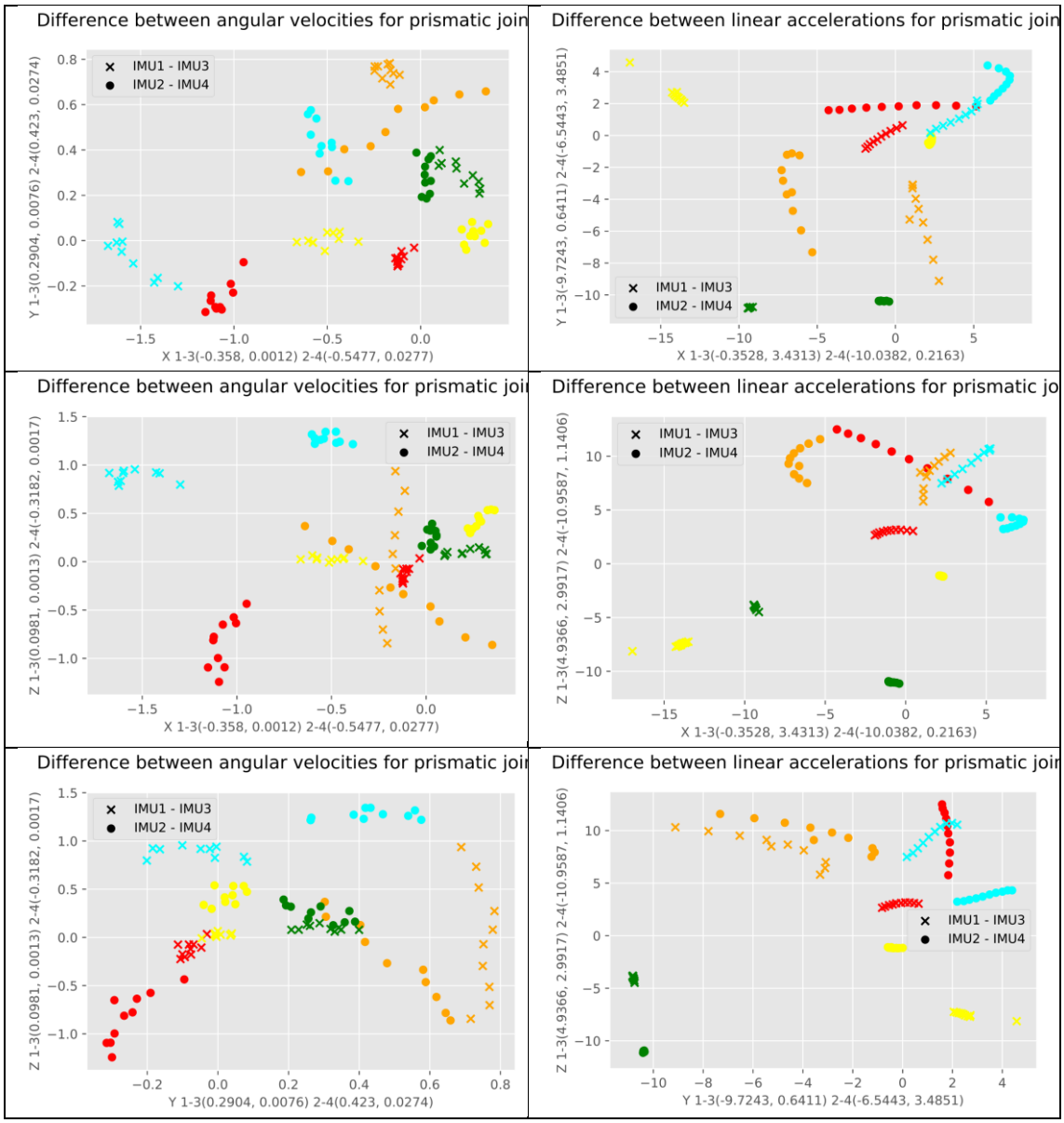


Figure 17. *Subtractions of angular velocities and linear accelerations for prismatic joint*

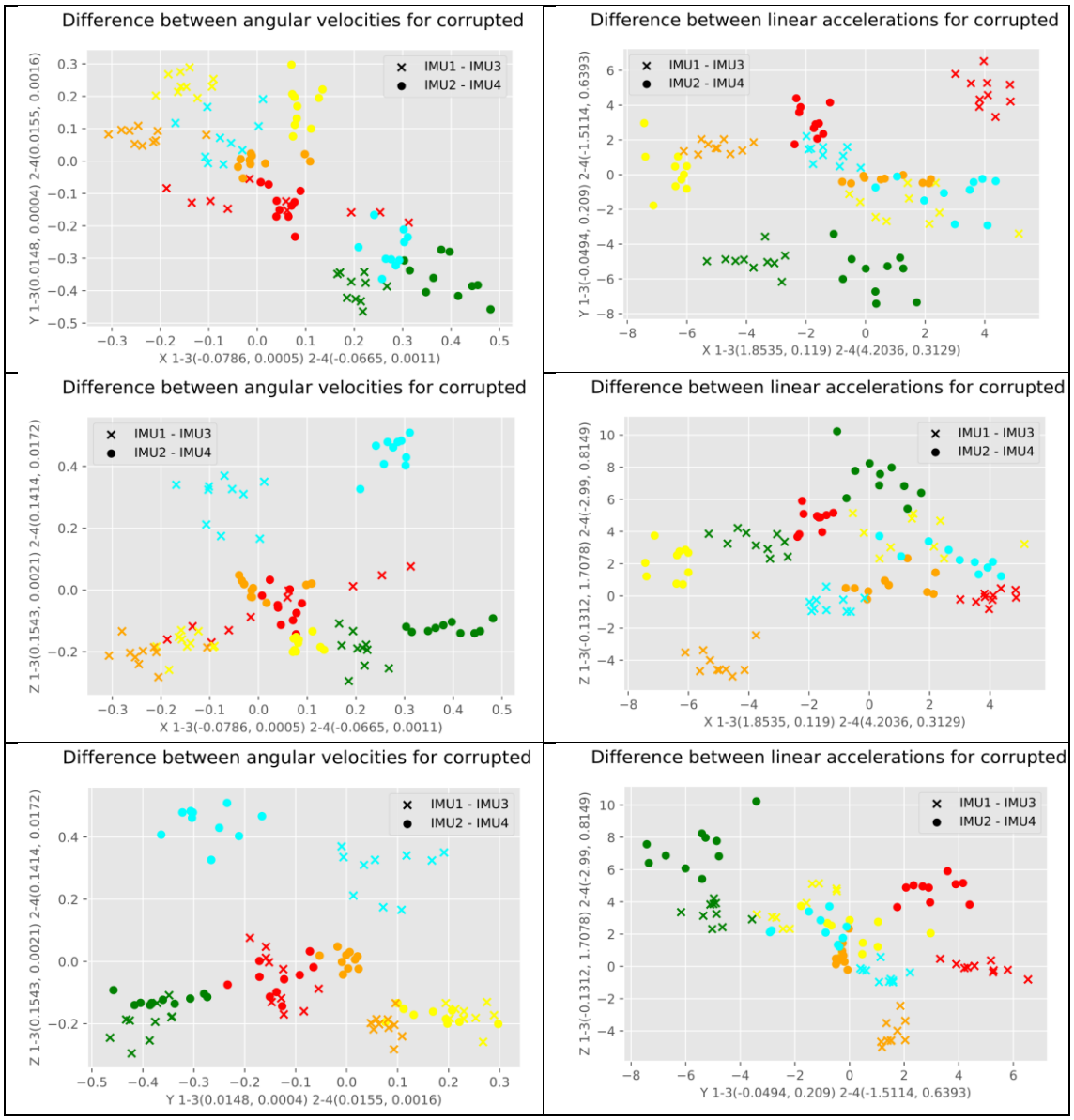


Figure 18. *Subtractions of angular velocities and linear accelerations for corrupted*

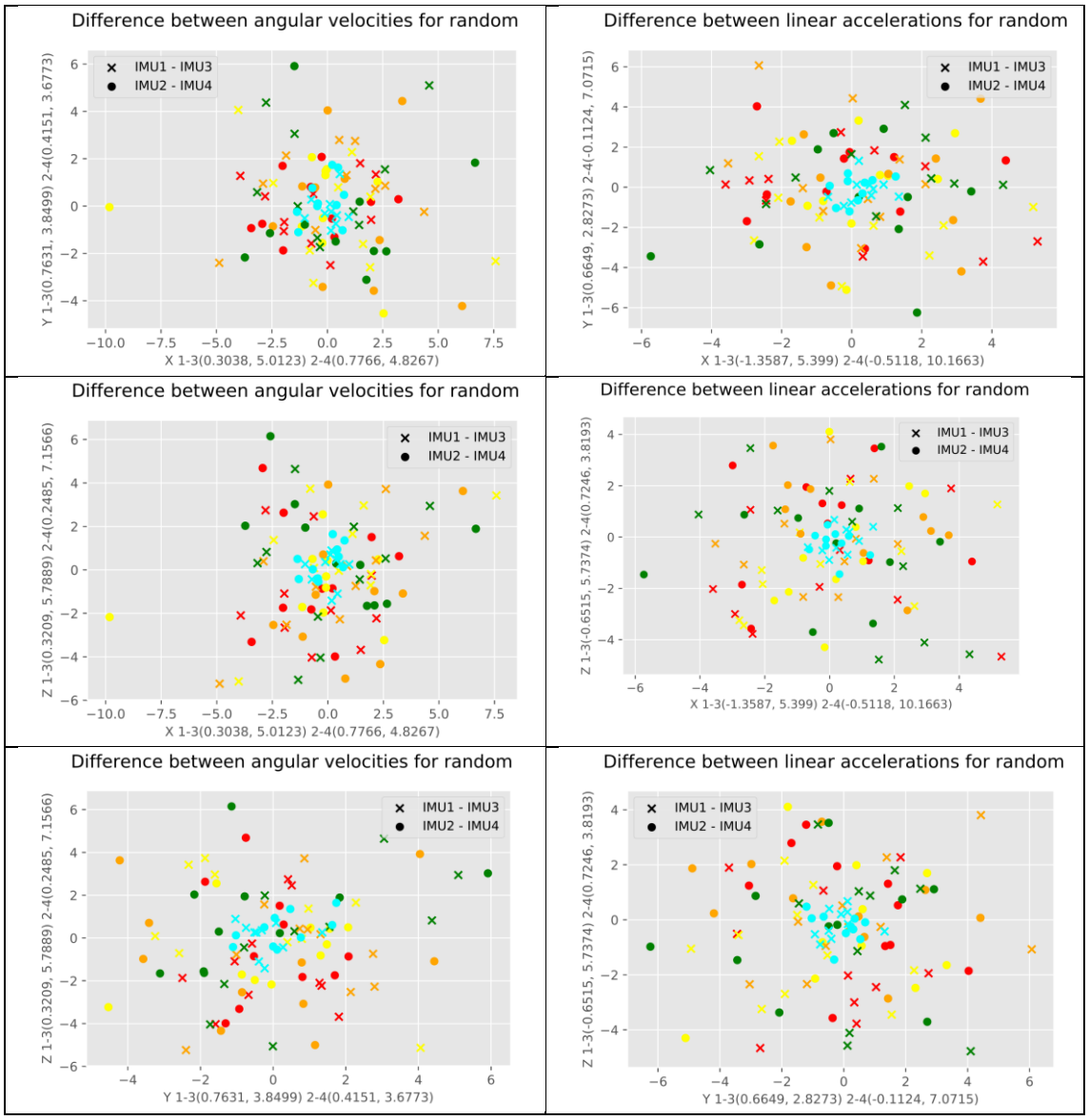


Figure 19. *Subtractions of angular velocities and linear accelerations for random*

3.5 Classification Methods

In the preprocessing phase mean dataset with 240 inputs is created. Additionally, in the feature extraction section 116 features are extracted from the mean and original dataset. As a result, two mean and feature engineered datasets are available for detecting the mechanical joint type of the manipulator. The target variable (joint type) is existed as a label for both datasets and it is categorical. Therefore, this is specified as a classification task which is in the category of supervised learning. For performing the classification task, data is divided into two group of training and test sets. 600349 instances are chosen for training set and 120069 instances for test set. In order to identify the mechanical joint type of the manipulator, various machine learning techniques are applied to mean and feature engineered datasets. Furthermore, their performance is measured on unseen samples via accuracy error metric. The algorithms which are used for classification include logistic regression (LR), linear discriminant analysis (LDA), random forest (RF) and XGBoost classifier. The tools and libraries which are used in this section are consist of Anaconda python distribution, Pandas [36], [37], Numpy [38], [39] and Scikit-learn [40], [41] libraries.

3.5.1 Logistic Regression

The LR model is fine-tuned with different combination of regularization hyper-parameters C and penalty on mean dataset. In fact, the effect of regularization parameter C and penalty is studied on the accuracy of the model. The execution time of hyper-parameter-tuning was approximately 98680 seconds and accuracy score of scikit-learn is used for performance evaluation. The result of fine tuning the hyper-parameters C and penalty on mean dataset are represented in the Table 3.1. As can be seen from table 3.1, accuracies for any combination of parameters are less than 50% which is less than random guessing. It is important to note that this algorithm could not obtain a reasonable accuracy on the training data neither (27%). Therefore, LR classifier is not able to detect the mechanical joint type and hence it is not suitable model for mean dataset.

Similarly, the LR model is also fine-tuned with the same hyper-parameters (C and penalty) on feature engineered dataset. The execution time of this process was almost about 192778 seconds. The result of hyper-parameter tuning after feature extraction are represented in Table 3. As can be seen from the Table 3, after feature extraction the accuracies of LR model for all combination of hyper-parameters are promoted significantly. The best accuracy of hyper-parameters combination belongs to C=10 and Penalty=L1. The accuracy corresponding to these hyper-parameters tuning is almost 55.77%.

Table 3. *Fine tuning of LR classifier with C and penalty hyper-parameters on mean dataset*

C	Penalty	Score (%)
1e-06	L1	24.96
1e-06	L2	23.19
1e-05	L1	25.01
1e-05	L2	24.52
0.0001	L1	26.57
0.0001	L2	26.07
0.001	L1	26.19
0.001	L2	26.6
0.01	L1	26.49
0.01	L2	26.68
0.1	L1	26.66
0.1	L2	26.69
1.0	L1	26.69
1.0	L2	26.69
10	L1	26.69
10	L2	26.69

Table 4.*Fine tuning of LR classifier with C and penalty hyper-parameters on feature engineered dataset*

C	Penalty	Score (%)
1e-06	L1	36.9181
1e-06	L2	51.7334
1e-05	L1	46.0536
1e-05	L2	54.4100
0.0001	L1	53.6612
0.0001	L2	55.2777
0.001	L1	55.3700
0.001	L2	55.6104
0.01	L1	55.7004
0.01	L2	55.7401
0.1	L1	55.7583
0.1	L2	55.7660
1.0	L1	55.7669
1.0	L2	55.7598
10	**L1**	**55.7670**
10	L2	55.7627

Furthermore, after selecting the suitable hyper-parameters for classification task, the LR model is trained combination of these hyper-parameters with highest accuracy. The confusion matrices without normalization and with normalization are shown in Figure 20. As

can be seen from the normalized confusion matrix, LR classifier was able to identify 0.72 of the rigid joints, 0.39 of the revolute joints, 0.36 of the prismatic joints, 0.52 of Corrupted and all (1.0) of random correctly.

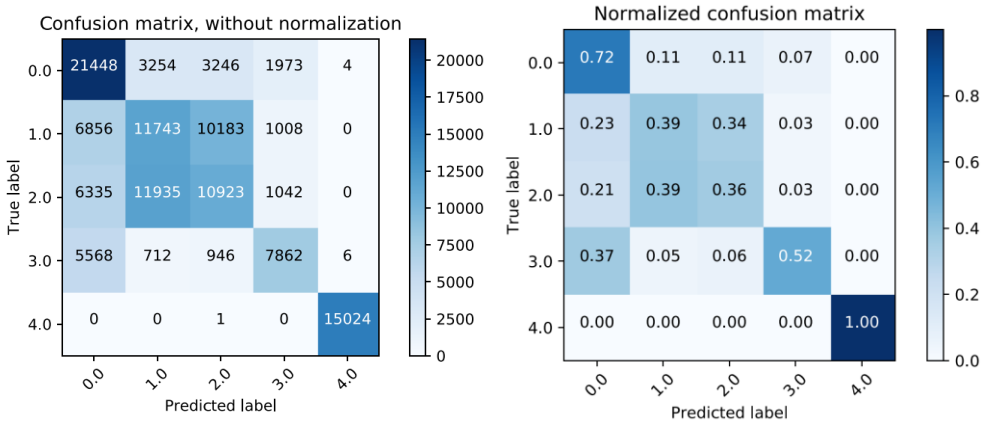


Figure 20. *Confusion matrices without and with normalization of LR model for feature engineered dataset*

3.5.2 K-Nearest Neighbors

The hyper-parameters which is used for KNN algorithm include the number of neighbors and distance metric. The algorithm is tuned with number of neighbors to 3, 5, 7, 11 and 13 in order to discover the best accuracy. However, the accuracies were almost the same for selected number of neighbors. Therefore, number of neighbors is chosen to 5 and Euclidean distance is applied for the metric hyper-parameter. The rest of the hyper-parameters remained as default for the algorithm. After that 5NN model is trained on mean dataset and the accuracy of prediction on unseen samples was approximately 61.21%. The execution time of training of 5NN algorithm on mean dataset was almost about 53487 seconds. The confusion matrices without normalization and with normalization are shown in the Figure 21. As can be seen from normalized confusion matrix, classifier was able to predict 0.75 of the rigid joints, 0.34 of the revolute joints, 0.34 of the prismatic joints, all of Corrupted and all of random correctly.

Similarly, KNN model is also fine-tuned with the same hyper-parameter (number of neighbors) on feature engineered dataset in order to specify how number of neighbors can effect the accuracy of the KNN model. Although, the accuracies were the same for all 3, 5, 7, 11 and 13 number of neighbors. Therefore, number of neighbors is chosen to 5 and Euclidean distance is applied for the metric hyper-parameter. After that, 5NN model is trained on feature engineered dataset the accuracy of prediction on unseen samples was almost 60.22 %, which is very close to the accuracy of mean dataset data. The execution time of training 5NN model on feature engineered dataset was about 6021

seconds. These results prove that it is possible train the 5NN model with reduced dimensions (feature engineered dataset) and keep the accuracy almost the same. The advantage of training with feature engineered dataset is that since the dataset contain less dimension compared to mean dataset, less time and power is consumed for training the machine learning model. The confusion matrices without normalization and with normalization are shown in the Figure 22. As can be seen from the normalized confusion matrix, 5NN classifier was able to predict 0.7 of the rigid joints, 0.35 of the revolute joints, 0.35 of the prismatic joints, all of Corrupted and all of random correctly. Confusion matrices remain almost the same before and after feature extraction.

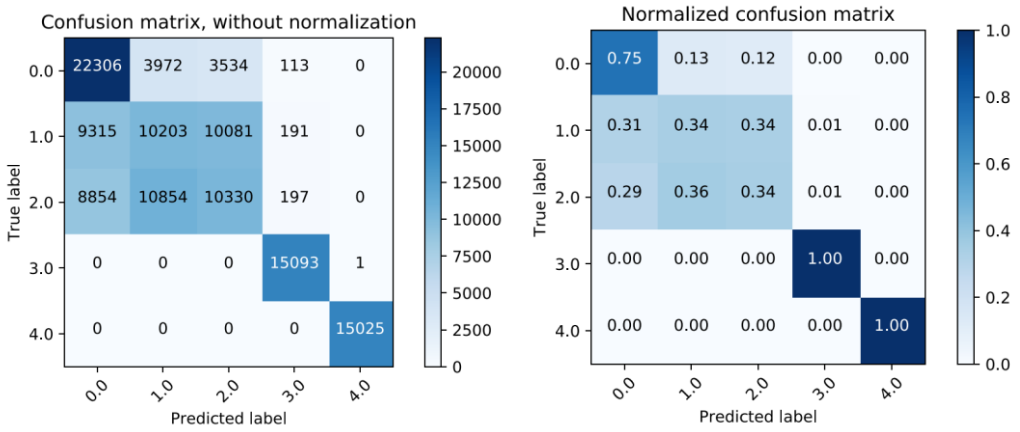


Figure 21. *Confusion matrices without and with normalization for 5NN model on mean dataset*

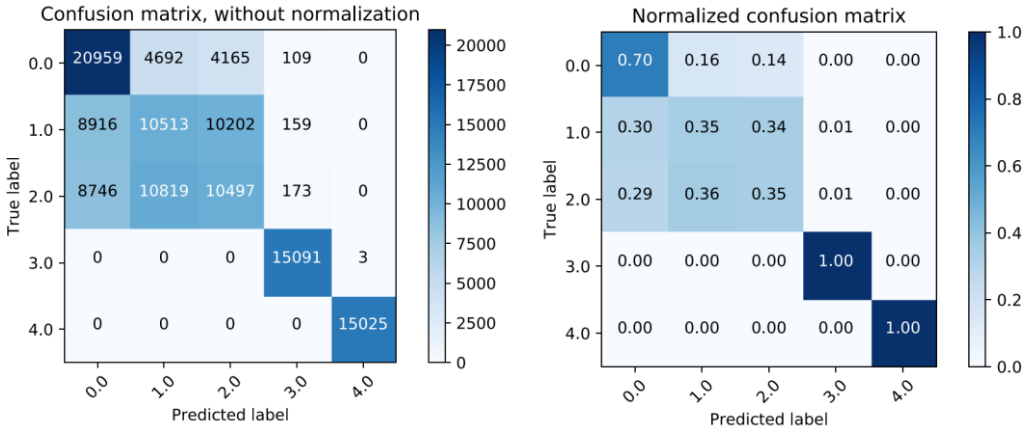


Figure 22. *Confusion matrices without and with normalization for 5NN model on feature engineered dataset*

3.5.3 Linear Discriminant Analysis

The LDA model is trained on mean dataset with default hyper-parameters. The accuracy of prediction on test set was around 26% which is less than random guessing. Therefore, LDA classifier is not able to detect the mechanical joint type and hence it is not suitable model for mean dataset.

Similarly, the LDA model is trained on feature engineered dataset and the accuracy of prediction on test set was 52%. The execution time of training the LDA model was almost 141 seconds. Furthermore, confusion matrices without normalization and with normalization are shown in the Figure 23. As can be seen from the normalized confusion matrix, LDA model was able to predict 0.5 of the rigid joints, 0.38 of the revolute joints, 0.34 of the prismatic joints, 0.93 of Corrupted and 0.8 of random correctly. The accuracy of this model is lower than tested other machine learning models so far. Consequently, the LDA is not an appropriate model for detecting the mechanical joint type of the manipulator.

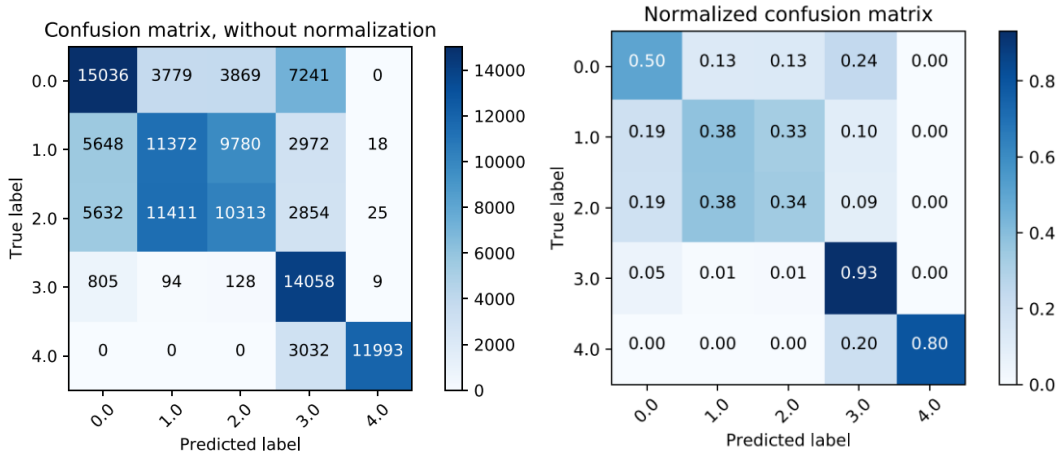


Figure 23. *Confusion matrices without and with normalization for LDA model on feature engineered dataset*

3.5.4 Random Forest

The hyper-parameter which has been used for this classifier include the number of trees in the forest. The number of trees is chosen equal to 100. The rest of the hyper-parameters remained default for the algorithm. Moreover, the RF model is trained on mean dataset and the accuracy of prediction on test set was approximately 67%. The execution time of training RF model was almost about 28354 seconds. Figure 24 represents confusion matrices without normalization and with normalization. As can be seen from the

normalized confusion matrix, RF model was able to predict 0.96 of the rigid joints, 0.37 of the revolute joints, 0.35 of the prismatic joints, all of Corrupted and all of random correctly.

Similarly, the RF model is trained on feature engineered dataset with the same hyperparameters and the accuracy of prediction on unseen samples was 68%. The execution time of training process was about 15479 seconds. By comparing the accuracies of mean dataset and feature engineered dataset, it can be concluded that feature engineered dataset is a good representation of mean dataset because the accuracies were almost the same for both datasets. Additionally, the benefit of feature extraction is that less time is consumed for training the dataset and the same accuracy can be achieved. The confusion matrices without normalization and with normalization are illustrated in Figure 25. As can be seen from the normalized confusion matrix, classifier was able to predict 0.94 of the rigid joints, 0.39 of the revolute joints, 0.4 of the prismatic joints, 0.99 of Corrupted and all of random correctly.

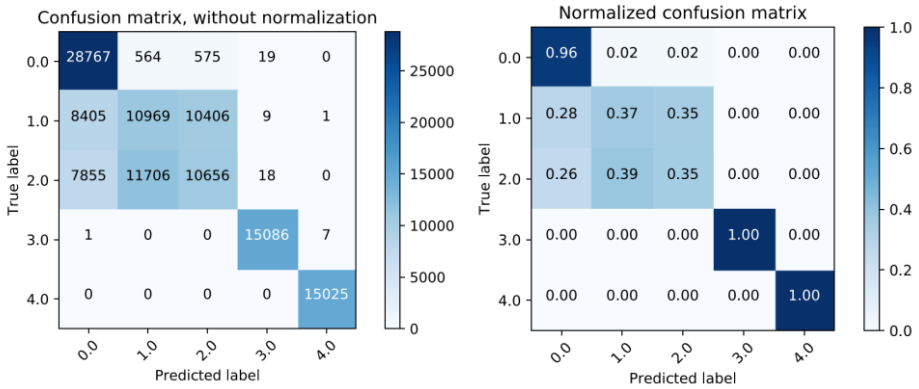


Figure 24. *Confusion matrices without and with normalization for RF model on mean dataset*

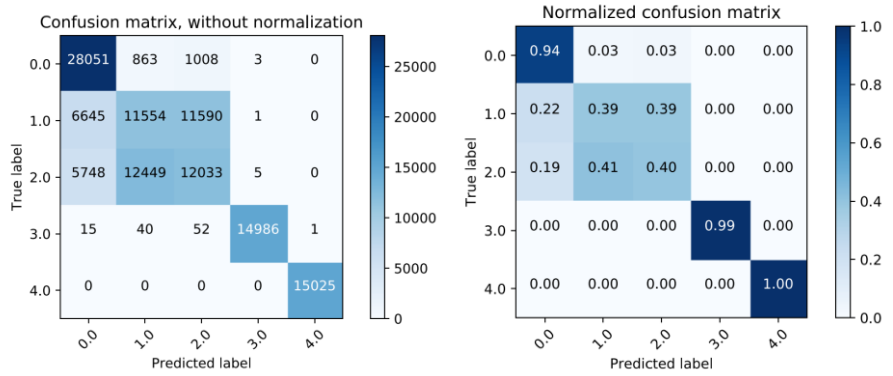


Figure 25. *Confusion matrices without and with normalization for RF model on feature engineered dataset*

3.5.5 XGBoost

The hyper-parameters which is used for XGBoost model include the number of trees, learning rate, objective, colsample_bytree, max_depth and alpha. These hyper-parameters are chosen as 100 trees, 0.1, reg:logistic, 0.3, 10 and 10 respectively. The rest of the parameters remained default for the algorithm. After that XGBoost model is trained on mean dataset. The accuracy of prediction on unseen samples was approximately 68.34%. The execution time of XGBoost model was about 6348 seconds.

The confusion matrices without normalization and with normalization are shown in Figure 26 . As can be seen from the normalized confusion matrix, XGBoost model was able to predict 0.97 of the rigid joints, 0.37 of the revolute joints, 0.38 of the prismatic joints, all of Corrupted and all of random correctly.

Similarly, the XGBoost model is also trained on feature engineered dataset with the same hyper-parameters and the accuracy of prediction on unseen samples was almost 68.63%. The execution time of training process was about 3052 seconds. The confusion matrices without normalization and with normalization are shown in the Figure 27. As can be seen from the normalized confusion matrix, XGBoost classifier was able to predict 0.95 of the rigid joints, 0.37 of the revolute joints, 0.43 of the prismatic joints, all of Corrupted and all of random correctly.

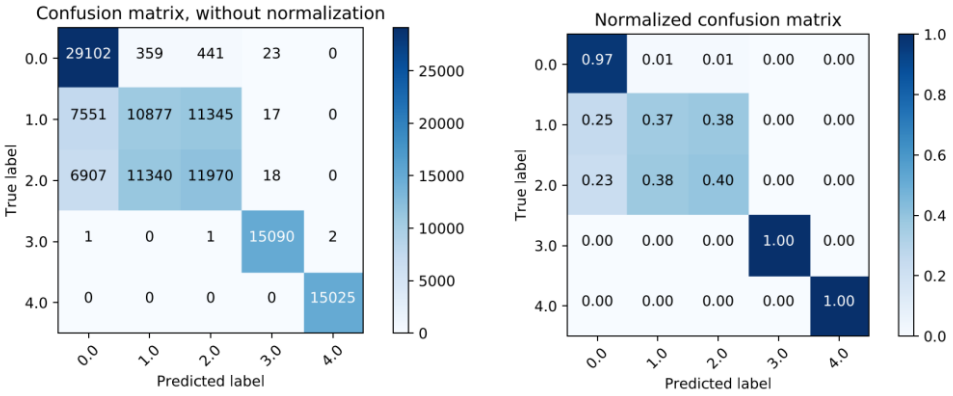


Figure 26. *Confusion matrices without and with normalization for XGBoost model on mean dataset*

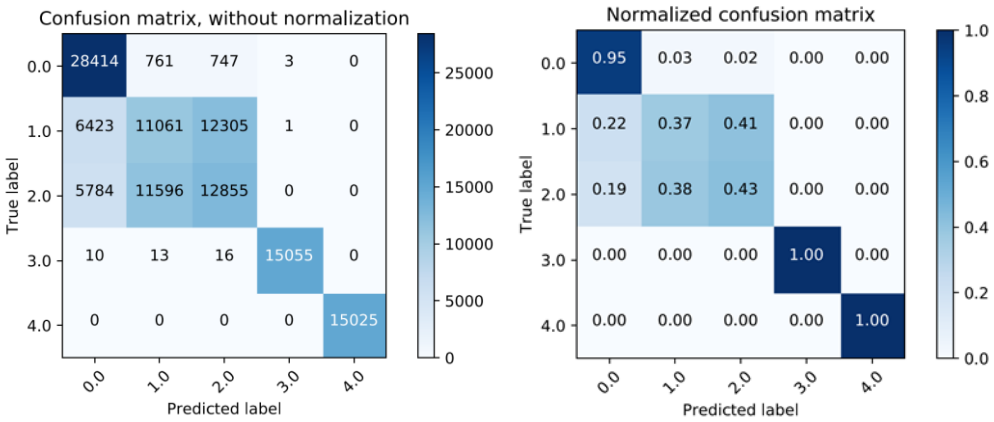


Figure 27. *Confusion matrices without and with normalization for XGBoost model on feature engineered dataset*

3.6 Permutation Importance

In this section we would like to investigate which features have the most significant effect on prediction. For this purpose, each feature is shuffled in a random manner. Consequently, the relation of the feature with the labels is destroyed. The prediction is performed by utilizing the combination of permuted features and residual features. If the accuracy of the model reduced significantly, it can be concluded that these features have effect on prediction. In other words, they are predictive feature. Additionally, the subtraction of model accuracy before and after permutation can be a rational scale for feature importance [42]. This technique called permutation importance.

The procedure of calculating the permutation importance is described in the following steps:

- First step: train the model
- Second step: shuffle values of first feature in the dataset and make predictions using the shuffled dataset.
- Third step: restore the dataset to its original order. Repeat the second step with the next features in the dataset. In this way, the importance of each feature is calculated [43].

In this thesis, calculating the permutation importance is performed using the ELI5 python package. “Eli5 is a Python package which helps to debug machine learning classifiers and explain their predictions.” [44]. Moreover, first the feature engineered data is trained using different classifiers such as random forest, XGBoost, logistic regression, linear discriminant analysis. In the next step permutation importance is computed with the help of ELI5 library. The results of the permutation importance computation for each model is represented in the Table 5, Table 6, Table 7 and Table 8.

As can be seen from the tables, first column of each table represents the amount of decrease or increase of the model performance with random shuffling of each feature. Furthermore, each random shuffling is repeated several times since with shuffling one feature each time, there might be different number for decrease or increase of the model performance. In other words, there might be some randomness in the decrease or increase of the model performance. The number after ± shows the variation of model performance in each random re-shuffling [43]. The most important features are at the top of each table and highlighted in bold.

The negative values of permutation importance for each feature show that the model performance is increased by shuffling that feature. In fact, the prediction becomes more accurate with reshuffling that feature. Moreover, the importance of the features are decreasing from top to down. In each table the most important feature is located at the top

of the table. For example, in the case of random forest the most important feature belongs to third category of features which is subtraction of first element and tenth element of angular velocity of sensor 2 and sensor 4 in Y axis (V1_10_2_4_y feature) [43].

Table 5.*Permutation importance for random forest model (a) Permutation importance with positive values. (b) Permutation importance with negative values.*

Weight	Feature
0.0019 ± 0.0010	**V1_10_2_4_y**
0.0016 ± 0.0010	acc1z_sum_IMU1
0.0014 ± 0.0009	acc1y_sum_IMU3
0.0014 ± 0.0006	Norm-IMU2-7-W^-1a
0.0013 ± 0.0004	V1_10_1_3_y
0.0013 ± 0.0008	Norm-vel-diff-1-3-1-4(7)
0.0012 ± 0.0012	acc1y_sum_IMU4
0.0012 ± 0.0006	Norm-IMU1-6-W^-1a
0.0011 ± 0.0004	Norm-acc-diff-2-3-2-4(6)
0.0011 ± 0.0010	V1_5_1_3_y

(a) Top 10 important features for RF model

Weight	Feature
-0.0000 ± 0.0006	acc2y_sum_IMU4
-0.0000 ± 0.0004	Norm-IMU4-10-W^-1a
-0.0000 ± 0.0009	Norm-acc-diff-1-3-1-4(9)
-0.0001 ± 0.0005	Norm-IMU3-9-W^-1a
-0.0001 ± 0.0004	Norm-acc-diff-2-3-2-4(7)
-0.0001 ± 0.0004	Norm-acc-diff-1-3-1-4(5)
-0.0001 ± 0.0006	Norm-vel-diff-2-3-2-4(9)
-0.0002 ± 0.0008	V1_10_2_4_x
-0.0002 ± 0.0003	V1_5_2_4_x
-0.0003 ± 0.0003	Norm-IMU4-1-W^-1a

(b) 10 features with negative permutation importance for RF model

Table 6.*Permutation importance for XGBoost model (a) Permutation importance with positive values. (b) Permutation importance with negative values.*

Weight	Feature
0.0078 ± 0.0006	**V1_10_2_4_y**
0.0051 ± 0.0007	V1_10_1_3_y
0.0048 ± 0.0007	V1_5_1_3_y
0.0036 ± 0.0003	V1_5_2_4_y
0.0021 ± 0.0004	Norm-IMU1-2-W^-1a
0.0020 ± 0.0005	Norm-IMU1-1-W^-1a
0.0020 ± 0.0006	acc1y_sum_IMU4
0.0020 ± 0.0010	acc2z_sum_IMU1
0.0019 ± 0.0007	acc2y_sum_IMU3
0.0019 ± 0.0006	Norm-IMU2-2-W^-1a

(a) Top 10 important features for XGBoost model

Weight	Feature
-0.0000 ± 0.0004	acc2z_sum_IMU4
-0.0000 ± 0.0007	Norm-vel-diff-2-3-2-4(4)
-0.0000 ± 0.0007	Norm-acc-diff-2-3-2-4(3)
-0.0000 ± 0.0003	Norm-IMU1-7-W^-1a
-0.0000 ± 0.0005	Norm-vel-diff-2-3-2-4(8)
-0.0001 ± 0.0006	Norm-vel-diff-1-3-1-4(9)
-0.0001 ± 0.0004	Norm-vel-diff-1-3-1-4(8)
-0.0001 ± 0.0009	Norm-vel-diff-1-3-1-4(3)
-0.0001 ± 0.0002	Norm-IMU3-9-W^-1a
-0.0001 ± 0.0004	Norm-IMU3-5-W^-1a

(b) 10 features with negative permutation importance for XGBoost model

Table 7.*Permutation importance for logistic regression model (a) Permutation importance with positive values. (b) Permutation importance with negative values.*

Weight	Feature
0.0449 ± 0.0012	**Norm-IMU1-10-W^-1a**
0.0391 ± 0.0018	Norm-IMU1-2-W^-1a
0.0369 ± 0.0015	Norm-IMU2-10-W^-1a
0.0344 ± 0.0009	Norm-IMU2-2-W^-1a
0.0284 ± 0.0008	Norm-vel-diff-1-3-1-4(10)
0.0225 ± 0.0013	Norm-IMU4-2-W^-1a
0.0224 ± 0.0014	Norm-acc-diff-1-3-1-4(10)
0.0223 ± 0.0008	Norm-IMU4-10-W^-1a
0.0198 ± 0.0024	Norm-vel-diff-1-3-1-4(2)
0.0193 ± 0.0009	Norm-IMU3-10-W^-1a

(a) Top 10 important features for LR model

Weight	Feature
-0.0000 ± 0.0003	V1_5_1_3_y
-0.0000 ± 0.0008	Norm-vel-diff-2-3-2-4(3)
-0.0000 ± 0.0002	V1_10_2_4_z
-0.0001 ± 0.0001	V1_10_2_4_x
-0.0001 ± 0.0003	V1_5_1_3_x
-0.0001 ± 0.0003	V1_5_2_4_z
-0.0001 ± 0.0002	V1_5_1_3_z
-0.0002 ± 0.0003	V1_10_1_3_y
-0.0002 ± 0.0001	V1_5_2_4_x
-0.0002 ± 0.0006	acc1x_sum_IMU4

(b) 10 features with negative permutation importance for LR model

Table 8.*Permutation importance for linear discriminant analysis model (a) Permutation importance with positive values. (b) Permutation importance with negative values.*

Weight	Feature
0.0343 ± 0.0011	**Norm-acc-diff-1-3-1-4(10)**
0.0211 ± 0.0019	Norm-acc-diff-1-3-1-4(1)
0.0113 ± 0.0005	Norm-vel-diff-1-3-1-4(1)
0.0103 ± 0.0008	acc1y_sum_IMU3
0.0087 ± 0.0014	Norm-IMU1-10-W^-1a
0.0069 ± 0.0010	Norm-vel-diff-1-3-1-4(10)
0.0067 ± 0.0011	Norm-IMU3-10-W^-1a
0.0060 ± 0.0009	acc1x_sum_IMU1
0.0059 ± 0.0009	Norm-IMU2-10-W^-1a
0.0055 ± 0.0007	Norm-IMU1-2-W^-1a

(a) Top 10 important features for LDA model

Weight	Feature
-0.0000 ± 0.0001	V1_5_2_4_x
-0.0000 ± 0.0002	V1_5_1_3_z
-0.0000 ± 0.0002	V1_5_2_4_y
-0.0001 ± 0.0000	V1_10_1_3_z
-0.0001 ± 0.0002	V1_5_1_3_x
-0.0001 ± 0.0001	V1_5_2_4_z
-0.0001 ± 0.0001	V1_10_1_3_y
-0.0002 ± 0.0004	acc1y_sum_IMU2
-0.0002 ± 0.0003	Norm-IMU2-4-W^-1a
-0.0002 ± 0.0002	Norm-IMU2-8-W^-1a

(b) 10 features with negative permutation importance for LDA model

3.7 Deep Neural Networks Techniques

In recent years, deep neural networks (DNNs) have made dramatic improvement in different areas such as speech recognition, time series prediction and image processing. Moreover, these improvements are because of potential capability of feature learning and representation of deep neural networks. Many types of DNNs have been created such as feed-forward neural networks (FNN), convolutional neural networks (CNN) and recurrent neural networks (RNNs) [45]. Recurrent neural networks (RNNs) are suitable for creating powerful model from sequential data. As has been described in the previous chapter, one type of RNNs is long short term memory network (LSTM) which capable of learning long-term dependencies. They have the ability to remember information for long period of time. In this thesis, a LSTM model is proposed and applied to the dataset. Additionally, LSTM network is achieved the best results among other techniques.

3.7.1 Preprocessing

In the first step, dataset is divided into train and test sets. 600349 instances out of 720418 instances are chosen as a training set and 120069 instances are chosen as a test set. In other words, 1/6 of the total instances (720418) is considered a test set and the remaining of the data for the training set. Furthermore, the LSTM model requires a three dimensional input in the form of (samples, time steps, features). In the case of reconfigurable manipulator data, there are 100 time steps and each time step has 24 features. In the second step the three dimensional input data is reshaped to the required format. The output data is specified as class labels which are integer numbers. Therefore, one-hot coding method should be applied to the class labels in order to prepare the data and make it appropriate for training the LSTM network which is multi-class classification model. The one hot coding technique which is used for this purpose is to_categorical() function in Keras. The output of the model consists a vector of size five which is the probability of given window be classified as five joint types.

3.7.2 LSTM model

The LSTM model is created via Keras API in anaconda python programming environment. The LSTM model architecture consists of two hidden layer of LSTM nodes stacked on top of each other, which each hidden layer contains 30 nodes. Hyperbolic tangent is used as an activation function and hard sigmoid applied as recurrent activation function which is the default option of LSTM in Keras API. Finally, a dense fully connected layer with softmax activation function is determined to translate the features evoked by hidden layers and prepare the output layer for prediction. Figure 28 illustrate the architecture of proposed LSTM model.

The major challenge to train the neural network models is choosing the number of epochs. Applying too many epochs causes overfitting the training dataset. In contrast, applying too few epochs leads to underfitting. One of the techniques to avoid overfitting or underfitting is called early stopping. This technique provides the possibility of determining ideal number of training epochs. Besides, training can be stopped until the model performance is not improving anymore. Additionally, the model performance can be measured on the validation set.

The LSTM model is trained via Keras API and Google's TensorFlow backend. First, the LSTM model is trained without early stopping with 300 epochs. Additionally, learning rate

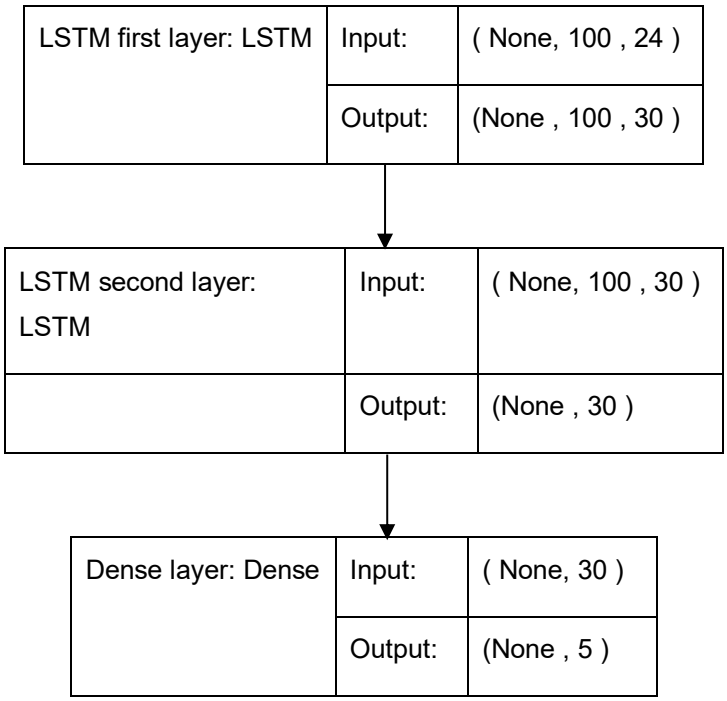


Figure 28. *Neural network model architecture based on LSTM*

of 0.001 and batch size of 64 is used for model configuration and fitting. The model performance is measured on train and test sets. Figure 29 represent plot of model loss on train and test sets during the training phase. As can be seen from the figure, until epoch 12 both train and test loss are decreasing and after that test loss is increasing. An early stopping is necessary for this kinds of situations in order to prevent overfitting. After applying early stopping technique, training is stopped after epoch 12. Figure 30 shows plot of train and test loss during training phase with early stopping technique. From the Figure 30, we can see that training is stopped at epoch 12 and model loss on both train and test set is decreasing. Table 9 shows the obtained model accuracy and loss on train and test sets. In addition, the confusion matrices without normalization and with normalization are shown in the Figure 31. As can be seen from the normalized confusion matrix, LSTM model was able to identify 0.98 of the rigid joints, 0.22 of the revolute joints, 0.62 of the prismatic joints, all of Corrupted and all of random correctly. These results shows that the proposed LSTM model is not able recognize the joint types accurately. However, it is able to recognize that there is a connection in manipulator accurately.

Table 9.*Model accuracy and loss on train and test set*

	Loss	Accuracy
Train	0.5408844012005567	70.76 %
Test	0.5514948857478028	69.82 %

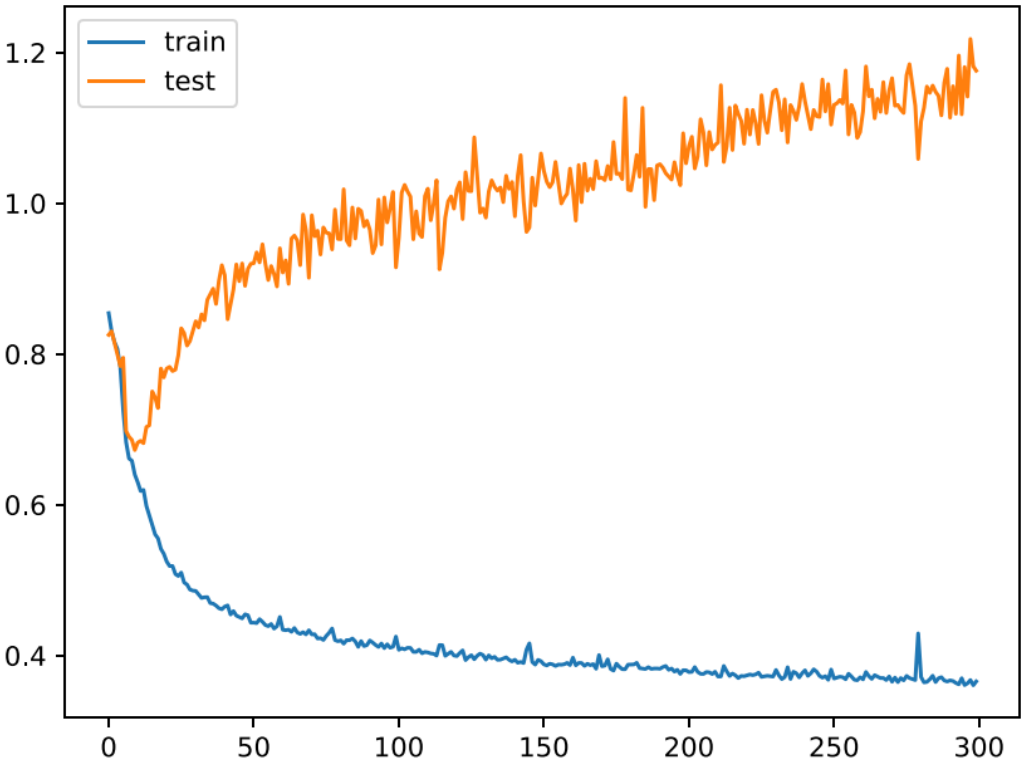


Figure 29. *Plot of model loss on train and test sets during training*

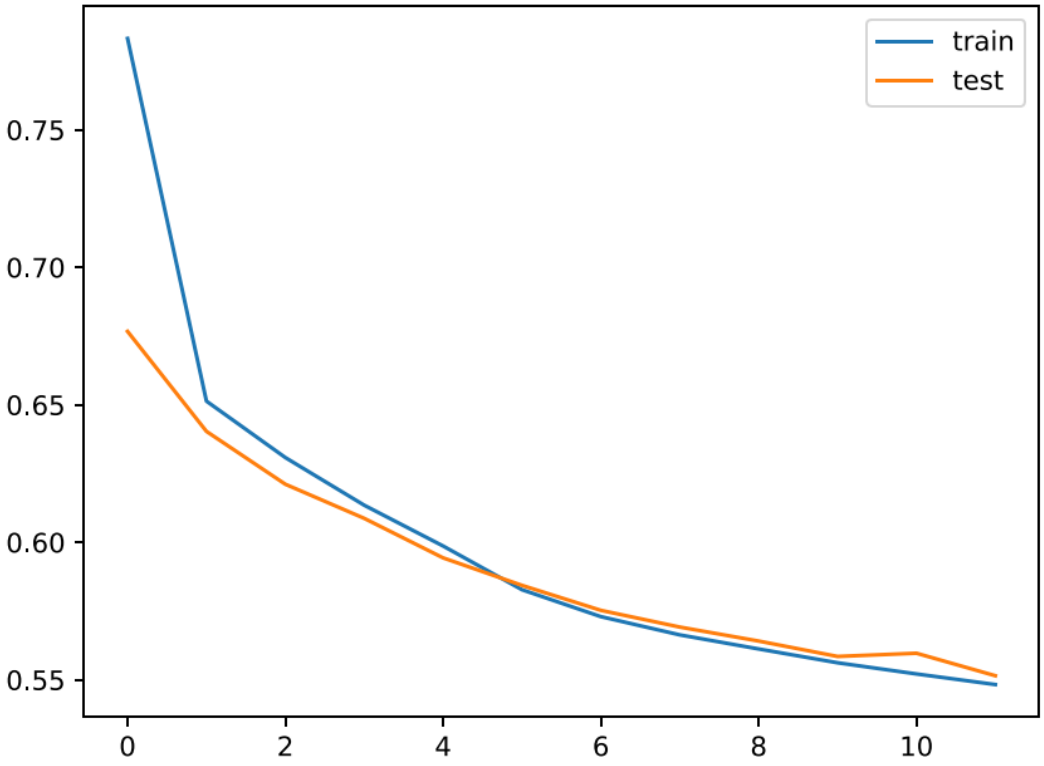


Figure 30. *Plot of model loss on train and test sets during training with early stopping*

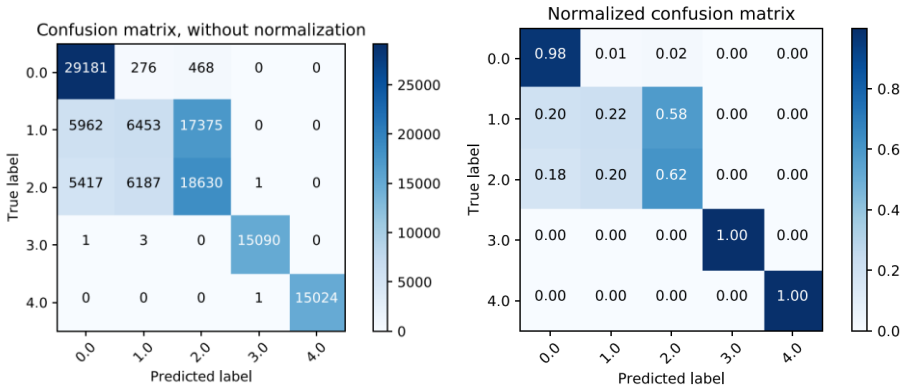


Figure 31. *Confusion matrices without and with normalization for LSTM model*

4. CONCLUSIONS

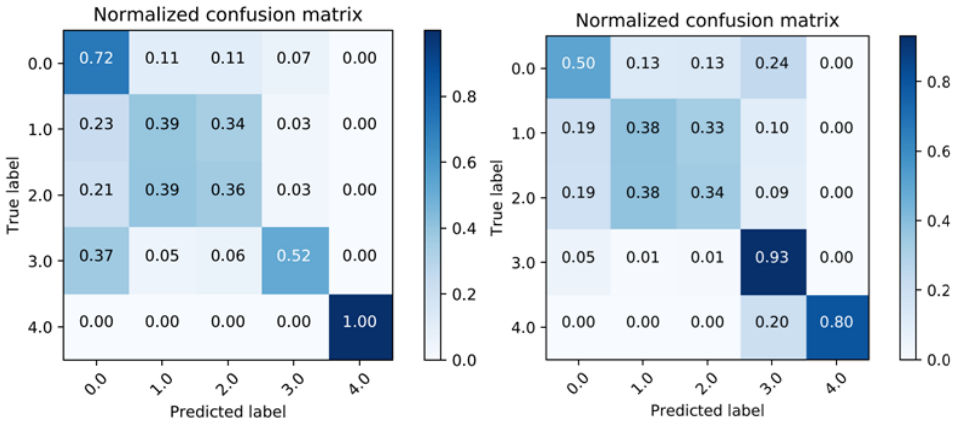
This thesis addresses a solution for recognition of mechanical joint type of the robotic manipulator from its sensory data via machine learning. The manipulator is simulated with reconfigurable structures via Aaria platform in order to gather a large dataset [5]. Recognizing mechanical joint type of the manipulator can be beneficial in automated calibration, kinematic analysis, and manipulator configuration or monitoring of articulated structures [6].

For detecting the mechanical joint type various machine learning models are explored which include LR, KNN, LDA, RF, XGBoost models and recurrent neural network technique. In this section, the performance of studied models are compared and best performed models are specified. Table 10 shows obtained accuracies for different models on feature engineered dataset. As can be seen from the Table 10, RF, XGBoost and LSTM models seems like best models for recognition of joint type of the manipulator. In summary, the LDA model might not a suitable solution since it has obtained the accuracy near to random guessing. In contrast, LSTM performed best compared to other models. One of the reason why XGBoost and RF are performing better in comparison to other traditional ML methods is that they combine several models in order to increase the performance.

Joint type recognition is a quite challenging task. Traditional ML methods might require domain expertise, human intervention and feature extraction while deep neural networks models have the capability of feature learning and extraction automatically. Additionally, combination of deep neural networks and traditional methods might provide promoted accuracy for joint type detection task. As can be seen from Table 10, both LSTM and XGBoost performs better in comparison to other algorithms. One path for improvements could be applying LSTM for identifying the features and feed the learned features to XGBoost model.

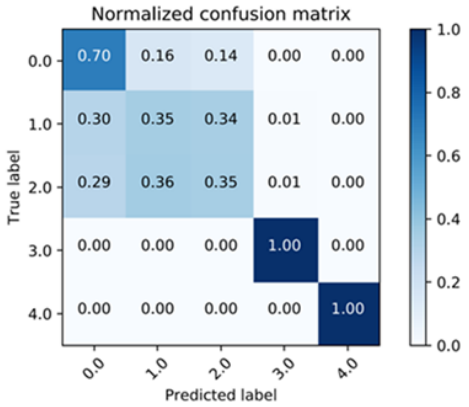
Moreover, Figure 32 illustrates the comparison of studied ML models for feature engineered dataset via confusion matrix visualization. As can be seen from the Figure 32, RF, XGBoost and LSTM are the most accurate models in recognizing rigid body with the accuracy of 94%, 95% and 98% respectively. Experimented ML models are not very accurate in recognizing the revolute and prismatic joints. In other words, they are not accurate in detecting the connection type. However, by combining the accuracies of revolute and prismatic joints, it is possible to obtain more accurate results. Therefore, the studied ML models are more accurate for recognizing if there is a connections in the

manipulator. The accuracies of connections for ML models are 75%, 72%, 70%, 79%, 80% and 84% respectively. Consequently, LSTM model is the most accurate ML model for detecting the kinematic connections among the experimented ML models.

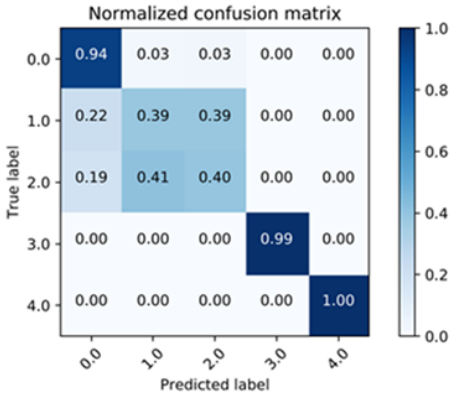


a) LR model

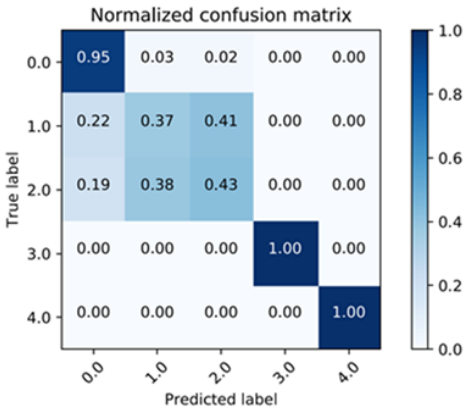
b) LDA model



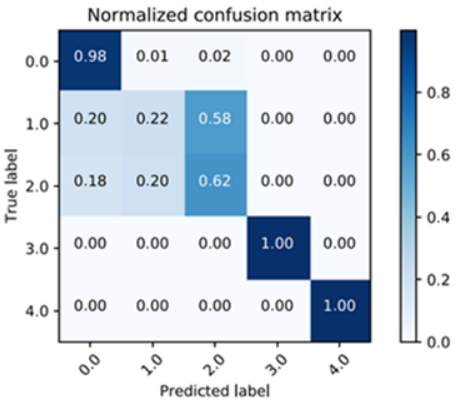
c) 5NN model



d) RF



e) XGBoost



f) LSTM

Figure 32. *Comparison of confusion matrices of experimented ML models.*

Table 10. *Accuracies of ML model on feature engineered dataset*

ML Model	Accuracy
LR	55.77 %
5NN	60.22 %
LDA	52 %
RF	68 %
XGBoost	68.63 %
LSTM	69.82 %

REFERENCES

[1] L. Kelmar and p. k. khosla, “Automatic generation of kinematics for a reconfigurable modular manipulator system,” *Proceedings. 1988 IEEE International Conference on Robotics and Automation,* vol. 2, p. 663, 1988.

[2] D. Schmitz, P. Khosla and T. Kanade, “The CMU Reconfigurable Modular Manipulator”, The Robotics Institute Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1988.

[3] M. Yim, S. Behnam and W.-m. Shen, “Modular Self-Reconfigurable Robot Systems,” *IEEE,* vol. 14, no. 1, pp. 43, 45, March 2007.

[4] A. Hautakoski, “Aaria: A Simulation Framework of Reconfigurable Manipulators for Deep Learning Scenarios,” pp. 1-47, March 2019, Available: http://urn.fi/URN:NBN:fi:tty-201903251328.

[5] A. Hautakoski, M. M. Aref and J. Mattila, “Reconfigurable Manipulator Simulation for Robotics and Multimodal Machine Learning Application: Aaria,” *Conference on Automation Science and Engineering,* pp. 1-7, 2018.

[6] M. M. Aref and J. Mattila, “Deep Learning of Robotic Manipulator Structures by Convolutional Neural Network,” *2018 Ninth International Conference on Intelligent Control and Information Processing (ICICIP),* pp. 236, 241 and 242, 2018.

[7] I. Guyon, S. Gunn, M. Nikravesh and A. Z. Lotfi, Feature Extraction, Springer-Verlag Berlin Heidelberg, 2006, pp. 1, 2, 3, 4 and 45.

[8] T. . M. Mitchell, Machine Learning, McGraw-Hill Science/Engineering/Math, 1997, p. 2.

[9] T. M. Mitchell and M. I. Jordan, “Machine learning: Trends, perspectives, and prospects,” *SCIENCE,* vol. 349, no. 6245, pp. 257-258, 2015.

[10] S. Basu, “Semi-Supervised Learning,” in *Encyclopedia of Database Systems*, Springer, Boston, MA, 2009, p. 2613.

[11] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Elsevier,* vol. 45, no. 4, p. 428, 2009.

[12] C. Molnar, Interpretable machine learning. A Guide for Making Black Box Models Explainable, 2019.

[13] R. A. Fisher, “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS,” *Annals of eugenics,* pp. 179-188, 1936.

[14] “1.2. Linear and Quadratic Discriminant Analysis — scikit-learn 0.22.2 documentation,” [Online]. Available: https://scikit-learn.org/stable/modules/lda_qda.html#lda-qda. [Accessed 12 April 2020].

[15] S. Balakrishnama and A. Ganapathiraju, “Linear discriminant analysis-a brief tutorial,” p. 1, 1998.

[16] “CS231n Convolutional Neural Networks for Visual Recognition,” Stanford University, [Online]. Available: https://cs231n.github.io/classification/#k---nearest-neighbor-classifier. [Accessed 12 April 2020].

[17] “1.6. Nearest Neighbors — scikit-learn 0.22.2 documentation,” [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html. [Accessed 12 April 2020].

[18] P. Cunningham and S. J. Delany, “k-Nearest Neighbour Classifiers,” University College Dublin, Dublin Institute of Technology, p. 1, 2007.

[19] L. BREIMAN, “Random Forests,”, Machine learning p. 5–32, 2001.

[20] A. R. Webb and K. D. Copsey, “Ensemble methods,” in *Statistical Pattern Recognition, 3rd Edition*, John Wiley & Sons, 2011, p. 361.

[21] J. H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine,” *The Annals of Statistics,* vol. 29, no. 5, pp. 1189-1232, 2001.

[22] T. Chen and T. He, “xgboost: eXtreme Gradient Boosting,” p. 1, 2019.

[23] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* p. 785, 2016.

[24] “(Tutorial) Learn to use XGBoost in Python,” DataCamp, 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/xgboost-in-python. [Accessed 12 April 2020].

[25] “CS231n Convolutional Neural Networks for Visual Recognition,” Stanford University, [Online]. Available: https://cs231n.github.io/neural-networks-1/. [Accessed 12 April 2020].

[26] J. Zou, Y. Han and S.-S. So, Overview of Artificial Neural Networks, vol. 458, Humana Press, 2008, pp. 17-18.

[27] R. Pascanu, T. Mikolov and Y. Bengio, “On the difficulty of training recurrent neural networks,” *Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013.,* vol. 28, p. 1, 2013.

[28] F.-F. Li, J. Johnson and S. Yeung , “Lecture 10: Recurrent Neural Networks,” Stanford University, 2017. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf. [Accessed 12 April 2020].

[29] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Transactions on Neural Networks and Learning Systems,* vol. 28, no. 10, p. 1, 2017.

[30] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[31] P.-N. Tan, M. Steinbach, A. Karpatne and V. Kumar, “Classification: Basic Concepts and Techniques,” in *Introduction to Data Mining, 2nd Edition*, Pearson, 2018, pp. 118-119.

[32] A. R. Webb and K. D. Copsey, “Performance assessment,” in *Statistical Pattern Recognition, 3rd Edition*, John Wiley & Sons, 2011, pp. 404, 408, 409 and 424.

[33] “3.5. Model evaluation: quantifying the quality of predictions — scikit-learn 0.15-git documentation,” [Online]. Available: https://scikit-learn.org/0.15/modules/model_evaluation.html. [Accessed 12 April 2020].

[34] D. Berrar, “Cross-Validation, Encyclopedia of Bioinformatics and Computational Biology,” *Elsevier ,* vol. 1, pp. 542-545, 2019.

[35] “Cross-validation for parameter tuning, model selection, and feature selection,” Data School, [Online]. Available: https://github.com/justmarkham/scikit-learn-videos/blob/master/07_cross_validation.ipynb. [Accessed 12 April 2020].

[36] The pandas development team, “pandas-dev/pandas: Pandas,” Zenodo, feb 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[37] W. McKinney, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, 2010.

[38] T. E. Oliphant, A guide to NumPy, Trelgol Publishing, 2006.

[39] S. v. d. Walt, S. C. Colbert and G. Var, “The NumPy Array: A Structure for Efficient Numerical Computation,” *Computing in Science & Engineering,* vol. 13, no. 2, pp. 22-30, 2011.

[40] F. Pedregosa, G. Varoquaux and et al, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[41] L. Buitinck, G. Louppe and et al, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108-122.

[42] M. Du, N. Liu and X. Hu, “Techniques for Interpretable Machine Learning,” *CoRR,* vol. abs/1808.00033, pp. 2-3, 2018.

[43] “Permutation Importance,” Kaggle, [Online]. Available: https://www.kaggle.com/dansbecker/permutation-importance. [Accessed 12 April 2020].

[44] “ELI5’s documentation,” ELI5, [Online]. Available: https://eli5.readthedocs.io/en/latest/overview.html. [Accessed 12 April 2020].

[45] J. Zhang and C. Zong, “Deep neural networks in machine translation: An overview,” *IEEE Computer Society,* pp. 2-3, 2015.