香港中文大學 (深圳)

The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE

CSC3180: Fundamentals of Artificial Intelligence

Course Project Report
A Senior Falling Alert System Embedded in Smart Wristband

Author:

Chen Boyi 119010010

Li Zihan 119010167

Zeng Zhuoru 119010417

May, 2022

Abstract

This project focus on the falling alert system to protect elder people who life alone. This project use sensing data collected from right wrist sensor (From Simulated Falls and Daily Living Activities Data Set , UCI Machine Learning Repository) to classify an action between falling and non-falling state. The problem this project works on, classifying falling or non-falling state, is a **Multivariate Variable-length Time Series Classification (TSC) Problem**. Three approaches are taken to train a classifiaction model. Approach one is to preprocess data and reform the problem as a non time series classification problem, and then use SVM algorithm to perform classification. Approach two is to preprocess time series data to equal length first and then apply classification algorithms for equal-length time series (RocketClassifier, HiveCoteV2 are used). Approach two is to apply classification algorithm directly on variable-length time series (KNN with DTW is used).

1. Introduction

1.1 Definition of Artificial Intelligence

Artificial Intelligence is generally speaking an intelligent machine. Intelligence indicates that the machine possesses some sort of cognitive capabilities and is capable of performing tasks (take action). Artificial Intelligence has many different definitions. According to the “strength of AI capabilities”, we can divide AI into strong AI (General AI) and weak AI (Narrow AI). Strong AI is capable of understanding and learning any intellectual task that a human being can. In another word, we want strong AI to act like humans. In contrast to strong AI, weak AI is not expected to have general cognitive abilities and is designed to solve exactly one specific problem. That is to say, we expect weak AI to act rationally.

1.2 Current Development of AI

Strong AI is a charming and attractive goal. For a system to be recognized as a strong AI, it must be able to achieve human-level performance in all the cognitive tasks like natural language processing, reasoning, and learning. However, strong AI is too challenging for human modern science to develop. Thus, strong AI is rather a hypothetical concept than a feasible theory today. People mostly focus on the sub-discipline of cognition, that is the development of weak AI. As for our ultimate pursuit, strong AI may be brought into reality when humans can unify the whole discipline with a law like what Newton did to the classical mechanics’ system.

1.3 AI application

A trend can be found in today’s AI industry. From a technology point of view, the hottest disciplines are machine learning, deep learning, natural language processing, and machine vision. From an end-user application point of view, the burst of the market lies in healthcare, automobile, games, finance, the internet, and media. The growing market and prosperity of the AI industry are promising in the years to come.

1.4 Our ideas

With the development of society, here rises one severe social problem, “aging of the population”. In some highly-developed countries, the elderly makes up a large part of the total population. However, the bad news is that under the pressure of life and economy, more and more young people choose to leave their parents and live far away from them to seek opportunities. The elderly is left alone, putting them in a dangerous situation. When the elderly encounters some diseases or danger, it’s hard for them to get help from others. Therefore, in our exploration of AI in the future, we would like to focus on AI applications in elder healthcare. We will look into the senior falling alert system and make our innovation with AI technologies.

2. The novelty of our ideas

Aimed to help with the problem raised above, we came up with the idea of producing an intelligent wristband with some danger detection functions to help the elderly. The primary function of the smart wristband is to detect if the user falls onto the floor. Although there are already some proven techniques in the market, like intelligent AI cameras, they have some limitations. The first limitation is that the process of doing human posture recognition with images is complicated. Due to the complexity of the AI model, the detection process’s speed is a

little bit slow. Another significant limitation of AI cameras is that they can't be used in some personal areas like the bathroom or bedroom due to users' privacy concerns.

Our smart wristband can ideally overcome those challenges mentioned earlier. The inputs to our AI system are simple, and they are some spatial coordinates and velocity information generated by the sensors integrated into the wristband. Compared to images, they are much easier for the AI system to handle. And the wristband can be used in any area, which is better than the cameras. When the elderly is out of the range of the camera, the camera is useless, but the ring can play an important role. The elderly can use the smart wristband in both in-door and out-door situations. Additionally, our wristband can be used in combination with the cameras to improve the accuracy of the output of the AI system.

3. Data Description

3.1 Data Set Introduction

The sensor within a wristband is used for detection. The sensor contains three axis accelerometer, gyroscope, which are important for determining the fall, to collect data and those data are regarded as the inputs of our model. We access to the UCI Machine Learning Repository to get the data set of our training and testing samples (Ã–zdemir, Barshan, 2014). The whole data set contains activities performed by 17 volunteers. Each volunteer performs 16 normal activities and 20 falls with five or six repetitions. Each activity contains a series of data captured by the sensor in the rate of 25.0Hz.

3.2 Data Selection

Two approach of selecting data is applied during our experiments. In the first approach, we select data from two highly distinguishable categories. In the second approach, we mix data from different action categories to form the two class (FALL & NON-FALL).

3.2.1 Approach 1: Data from Two Highly Distinguishable Action Categories

Among the 20 classes of Fall Actions and 16 classes of Non-Fall Actions in the dataset, one from each of these two categories (Fall & Non-Fall) were picked and used as trianing and testing data for the model:

FALL Category (With FALL lable in classification):

- 901 front-lying, from vertical falling forward to the floor

NON-FALL Category (With NON-FALL label in classification):

- 801 walking-fw, walking forward

The testing performance of our selected models are extremly good with this data set, with almost 100% accuracy. We add more data from differnet action categories to increase the diversity of our dataset and make it closer to real life situations, and here comes the second approach.

3.2.2 Approach 2: Data from Multiple Action Categories

Among the 20 classes of Fall Actions and 16 classes of Non-Fall Actions in the dataset, five from each of these two categories (Fall & Non-Fall) were picked and used as trianing and testing data for the model:

FALL Category (With FALL lable in classification):

- 801 walking-fw, walking forward
- 803 jogging, running
- 806 bending-pick-up, bending to pick up an object on the floor
- 811 sit-chair from vertical, to sitting with a certain acceleration onto a chair (hard surface)
- 815 lying-bed, from vertical lying on the bed

NON-FALL Category (With NON-FALL label in classification):

- 901 front-lying, from vertical falling forward to the floor
- 905 front-quick-recovery, from vertical falling on the floor and quick recovery
- 909 back-sitting, from vertical falling on the floor, ending sitting
- 915 left-sideway, from vertical falling on the floor, ending lying
- 917 rolling-out-bed, from lying, rolling out of bed and going on the floor

3.3 Data Analysis

3.3.1 Approach 1: Data from Two Highly Distinguishable Action Categories

Each of the 901 and 801 class in the dataset includes data from 17 volunteers performing the same action repeatedly for around 5 times. Each time series record has a length of around 400 to 700 timestamp, which is around 16 second to 28 seconds time. The total number of "FALL" records are 90, the total number of "NON-FALL" records are 92. The dataset is balanced.

Distribution of data series length is shown in Figure 3.3.1-1 below. Input data are in variable length and this prolem will be handled by us with two approaches. Approach 1, preprocess data to the same length and apply classification algorithm for fixed length time series on them. Approach 2, directly apply classification algorithms that can work on variable length time series.

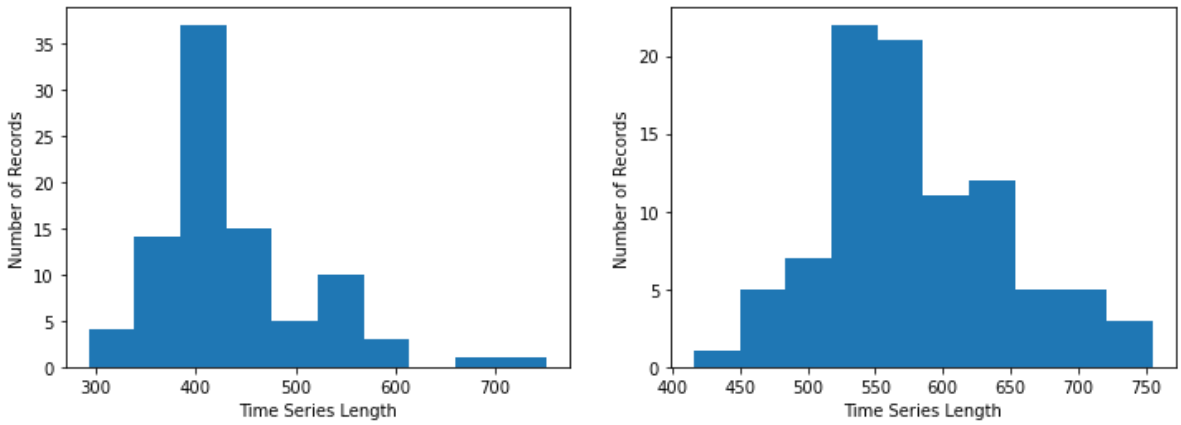


Figure 3.3.1-1

A demo of different data is shown below. Figure 3.3.1-2 Shows two falling samples with different length.

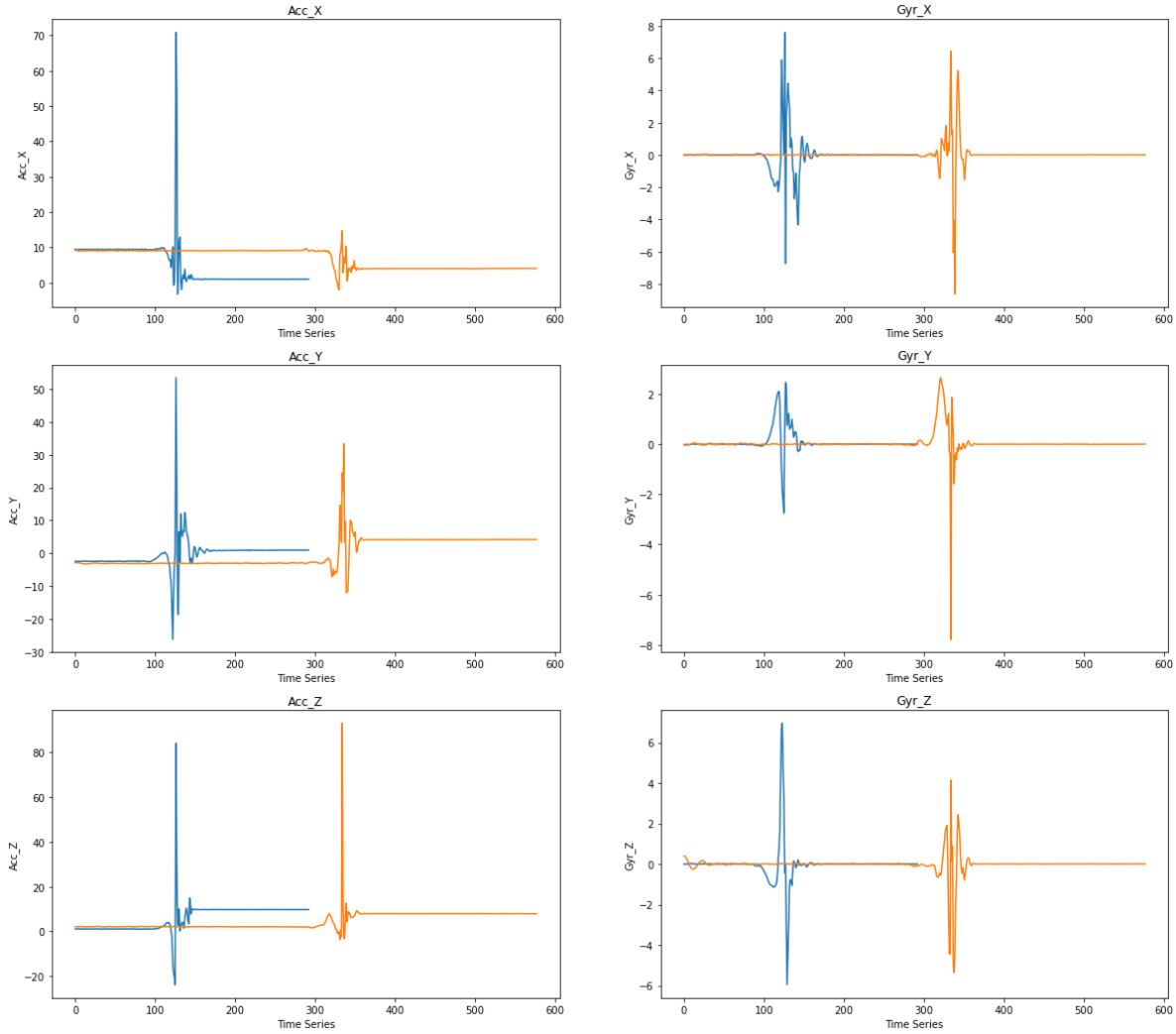


Figure 3.3.1-2 Falling Data with Different Length

Figure 3.3.1-3 Shows two non-falling samples with different length.

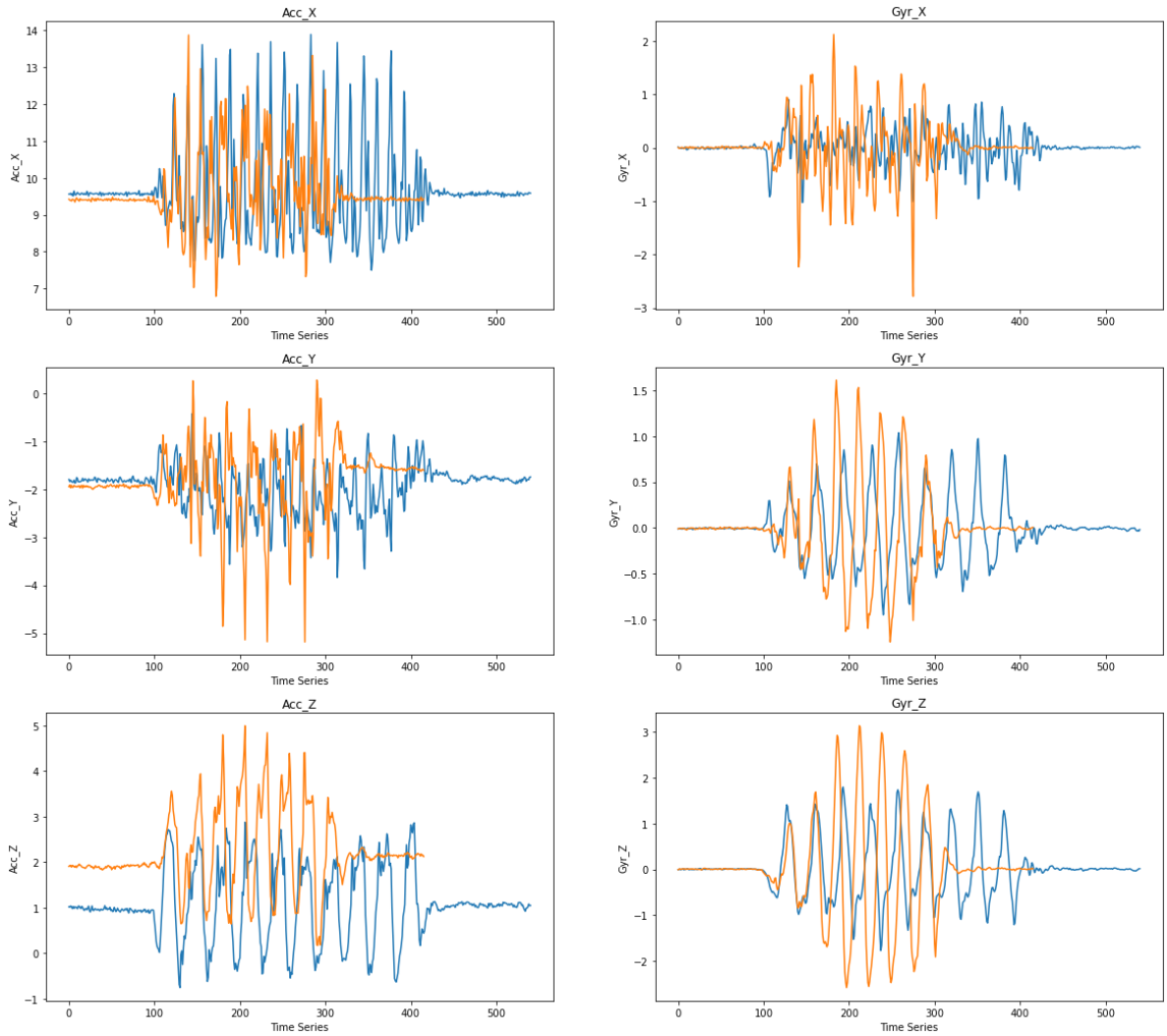


Figure 3.3.1-3 Non-Falling Data with Different Length

Figure 3.3.1-4 shows a sample of falling data and a sample of non-falling data.

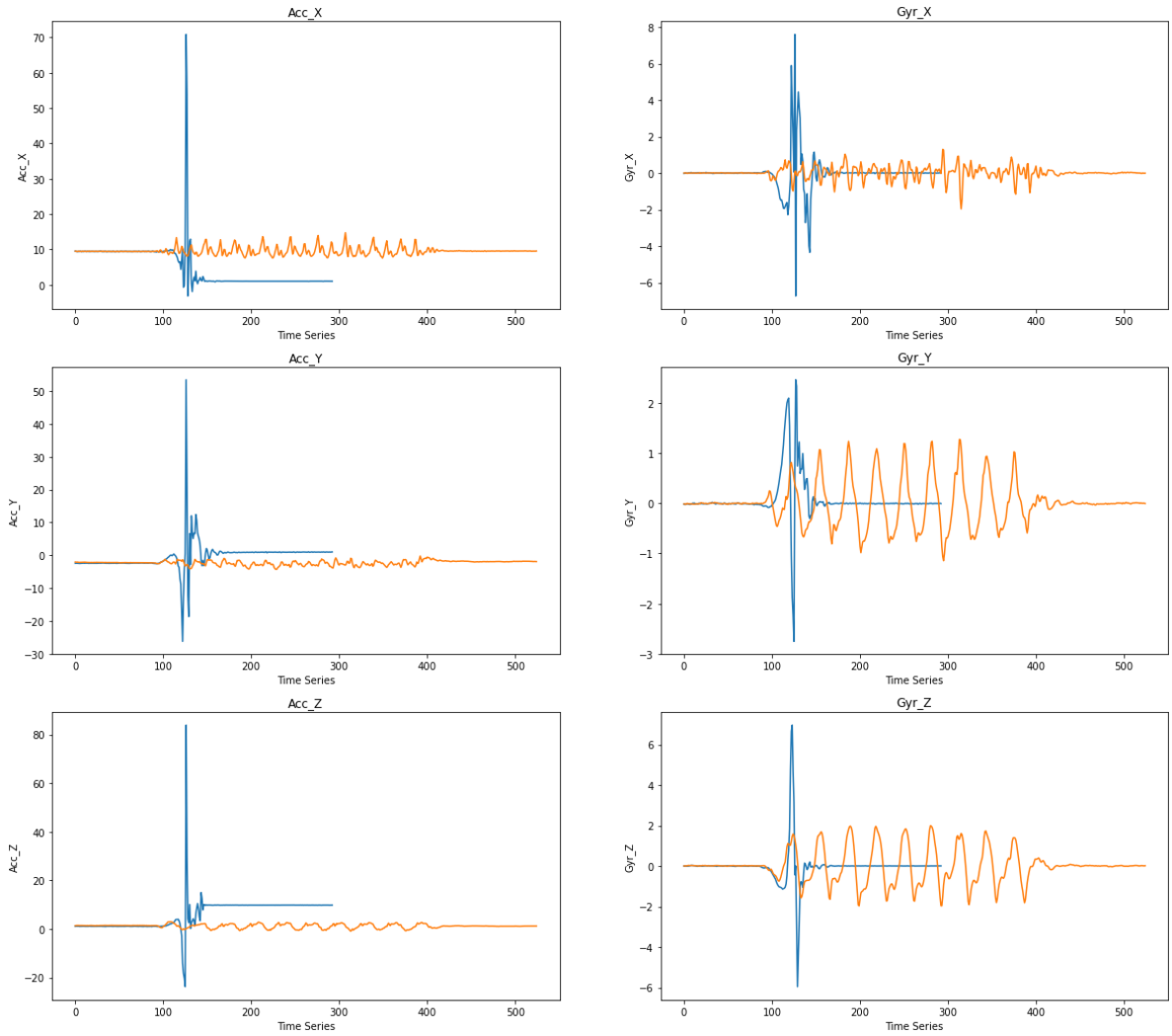


Figure 3.3.1-4 Falling VS. Non-Falling Data with Different Length

3.3.2 Approach 2: Data from Multiple Action Categories

Each of the class in the dataset includes data from 17 volunteers performing the same action, 1 sample from each volunteer is selected. Each time series record has a length of around 400 to 700 timestamp, which is around 16 second to 28 seconds time. The total number of "FALL" records are 85, the total number of "NON-FALL" records are 85. The dataset is balanced.

Distribution of data series length is shown in Figure 3.3.2-1 below.

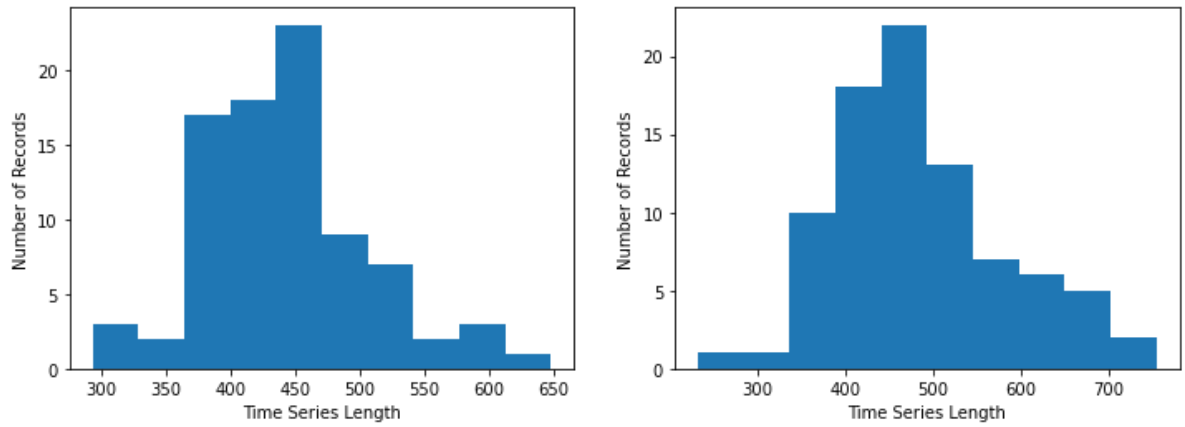


Figure 3.3.2-1 Falling VS. Non-Falling Data with Different Length

A demo of different data is shown below. Figure 3.3.2-2 Shows 5 falling samples from different classes.

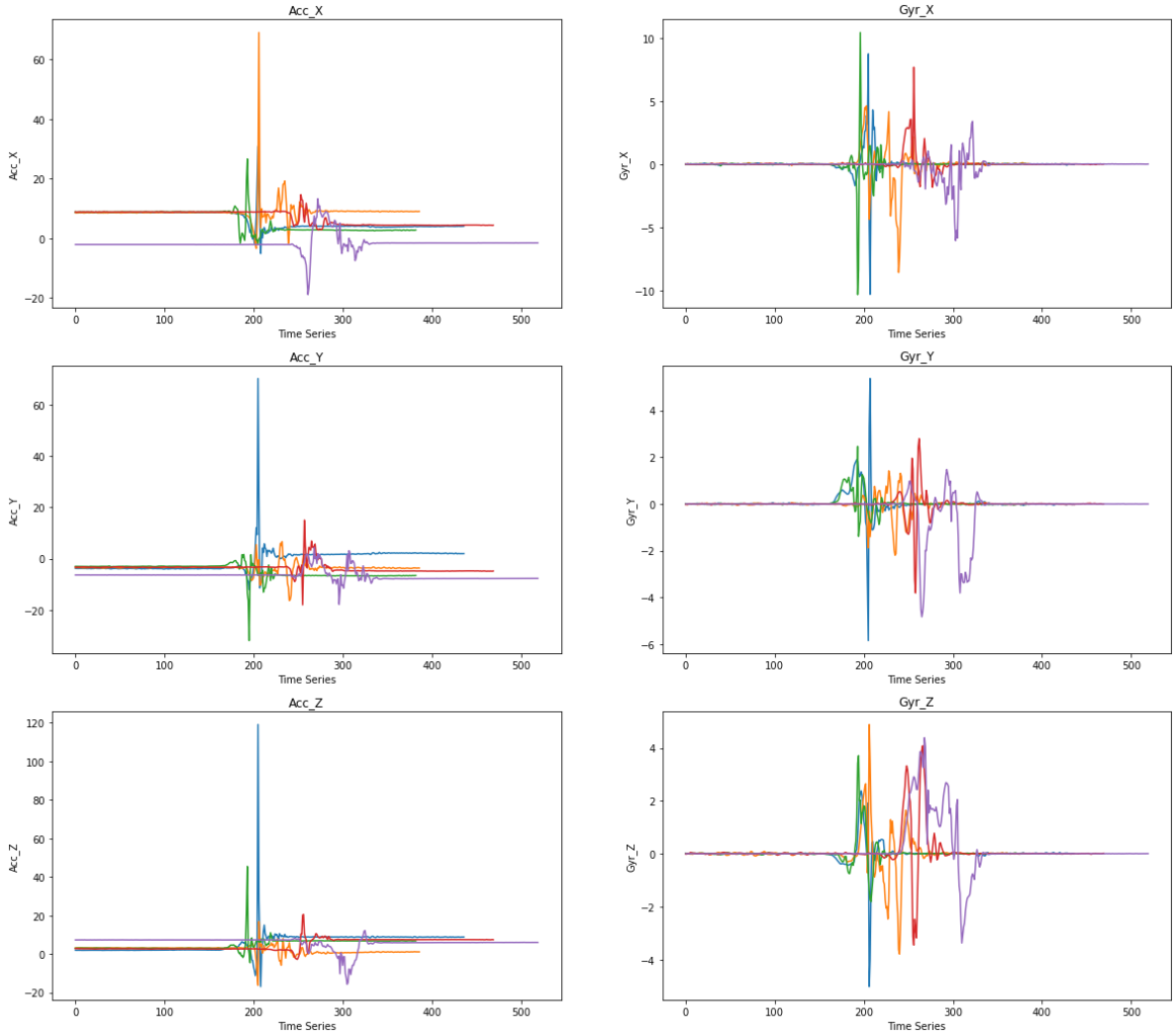


Figure 3.3.2-2 Falling Data with Different Length

Figure 3.3.2-3 Shows two non-falling samples from 5 different classes.

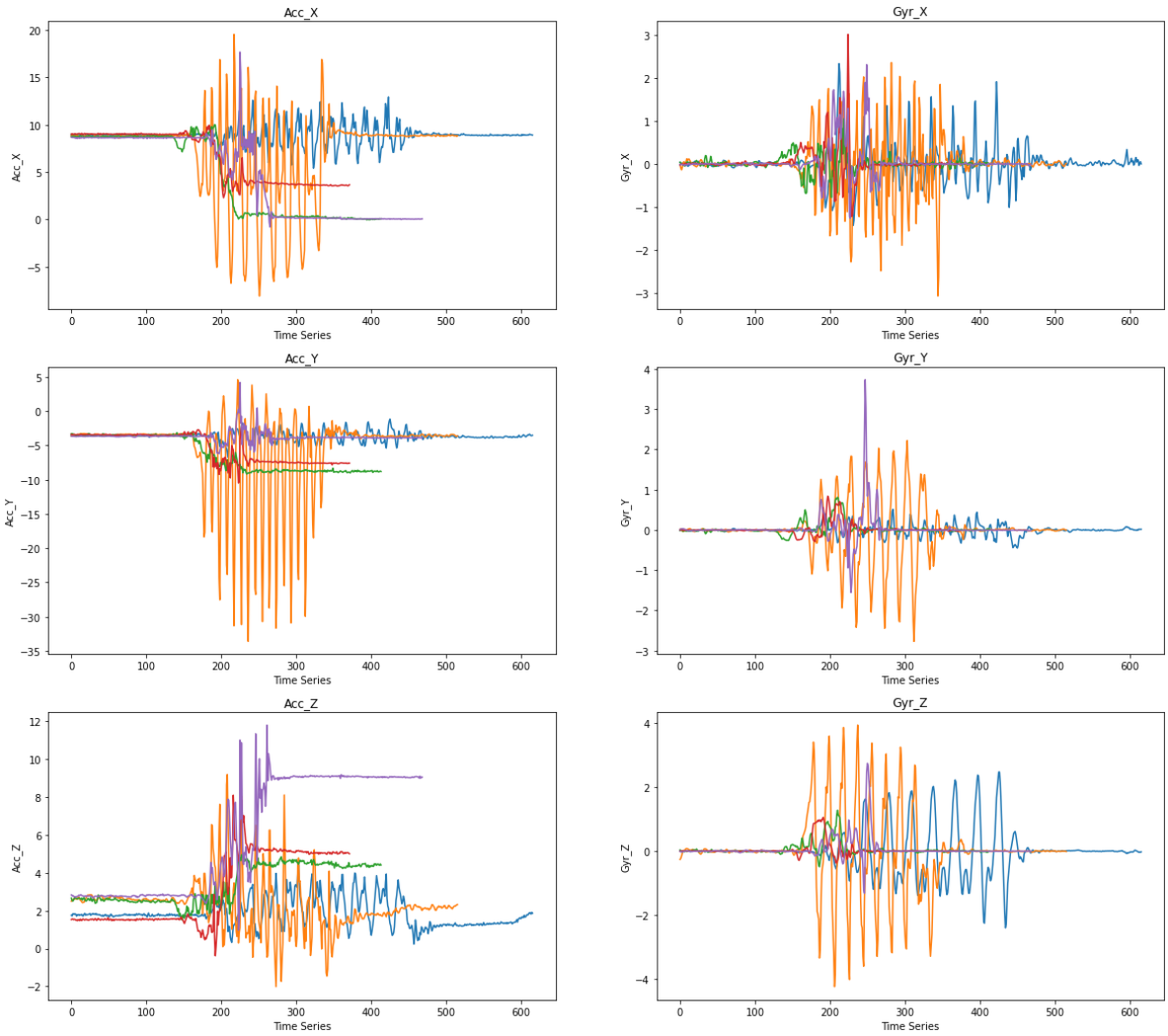


Figure 3.3.2-3 Non-Falling Data with Different Length

3.4 Training and Test Set Separation

80% of the whole data set will be treated as the training data and the rest 20% is the testing data.

4. Models Implementation

4.1 TSC Approach 1: Reform Problem to Non Time Series Classification - SVM

4.1.1Brief Introduction to SVM and Our Project

support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space so as to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In our project, we aim to use the input from wristband to detect the falling activities of daily life (ADLs). The dataset of our project mainly contains time series labeled by two kinds, falling (which is represented by 1) and non-falling (which is represented by 0). Compared to traditional threshold method for the detection of falling ADLs, a hyperplane of the support vector machine was used as the separating plane to carry out the classification functionality.

4.1.2 SVM with Raw Data Input and Problems

Our data are different to traditional time series input for SVM, where each time stamp in the time series has a label or value, our data uses the whole time series as the input and assign a label to that specific time series. For example, in our dataset, we have a txt file named "101_901_1.txt", which means the volunteer 101 for the experiment performs front-lying (from vertical falling forward to the floor) and the sensor gathers the data for a period of time and this whole time series is labeled as falling.

There comes some problems, firstly, different time series are of different length, which need to be modified before they are fed to the model for training and testing. Secondly, if we consider the whole time series as one input, the size of it is extremely large, and the type of it is an array of numpy arrays, which is not suitable for the model we choose (we use the SVC model from sklearn.svm package).

4.1.3 Data Processing

4.1.3.1 Assign Label to Each Time Stamp in One Specific Time Series

Due to the problems mentioned above, we came up with one solution to them. That is to assign label to each time stamp in one specific time series. Here we took '101_901_1.txt' for example again, there are several rows of values, which belong to six fields namely, 'Acc_X', 'Acc_Y', 'Acc_Z', 'Gyr_X', 'Gyr_Y' & 'Gyr_Z', and we know this time series corresponds to a falling ADL. So, we add '1' to the end of each row. After this process, each time stamp is assigned one value, making it suitable for the model. Below is the data frame we used for model training after data processing.

index	0	1	2	3	4	5	6
0	8.949280	-3.509521	1.754761	0.040436	-0.011587	0.009727	0
1	8.933473	-3.459883	1.722622	0.010109	-0.007057	0.014496	0
2	8.935547	-3.492737	1.828003	0.004196	-0.000143	0.015450	0
3	8.948612	-3.551388	1.830053	0.012732	-0.003529	0.005150	0
4	8.926392	-3.558350	1.800537	0.021935	-0.006437	0.005722	0
...	...	...	...	...	...	...	..
259494	-0.594378	-2.276421	9.757018	-0.000191	-0.001240	0.007391	1
259495	-0.617981	-2.253723	9.780884	0.004196	-0.004721	-0.001144	1
259496	-0.609922	-2.256703	9.747791	0.001049	-0.002527	-0.000954	1
259497	-0.592041	-2.278137	9.800720	0.005531	-0.007772	0.002098	1
259498	-0.586128	-2.268314	9.776974	0.004005	0.002289	0.004721	1

Figure 4.1.3.1-1

But here is one big problem for this kind of data processing method, that is in the time series labeled as falling, a certain portion of data is not falling, but we mark them as falling. So a large portion of data is wrongly marked, resulting in a poor performance of the SVM model. The final accuracy of the model is around 58.02%, which is relatively low. So a better data processing method is needed for a better model.

4.1.3.2 Use The Value with Max Absolute Value to Represent the Whole Time Series

After looking at the plot of the data input, we come up with another data processing method, to use the value with max absolute value to represent the whole time series. Below are two pictures for falling and non-falling ADLs:

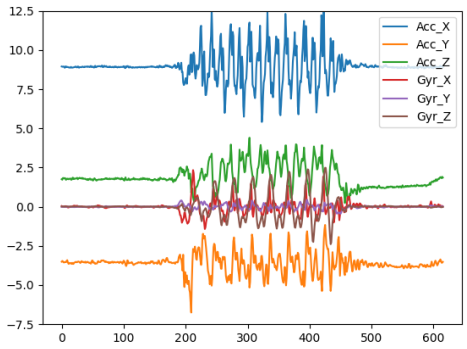


Figure 4.1.3.2-1

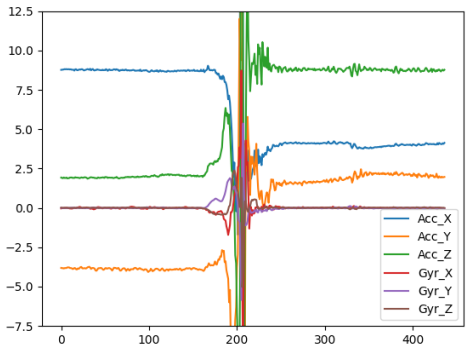


Figure 4.1.3.2-2

We can observe that the max absolute values of non-falling ADL is smaller than that of falling ADLs. So we may use the max absolute value to represent one specific time series. The reason why we take absolute value is that the elderly may falling down to the ground in multiple directions, so the sign of the data field may differ from each other. But the sign can't decide whether the elderly falls or not, but the absolute value matters. So we use the max absolute values.

4.1.4 Testing and Training Data Split

As our dataset involves several cases of falling and non-falling ADLs, a good way to split the data into testing data and training data is to use random number generator so that the model can take each case into consideration, so as to increase the accuracy. Here we use the train_test_split package function from sklearn.model_selection package and to the split and training for multiple times to higher prediction accuracy.

4.2 TSC Approach 2: Time Series Classification Algorithms on Fixed Length Time Series - RocketClassifier & HiveCoteV2

4.2.1 Data Preprocessing

Four ways of data preprocessing are implemented: Truncate data to fixed length, Sample padding with mean, Uniform scaling, Sample padding with regression prediction.

The target of data preprocessing is to extend or shrink different time series sample to the same length while not affecting the information and pattern they contain. An ideal data format is shown below in Figure 4.1.1-1, that all the training data are of the same length.

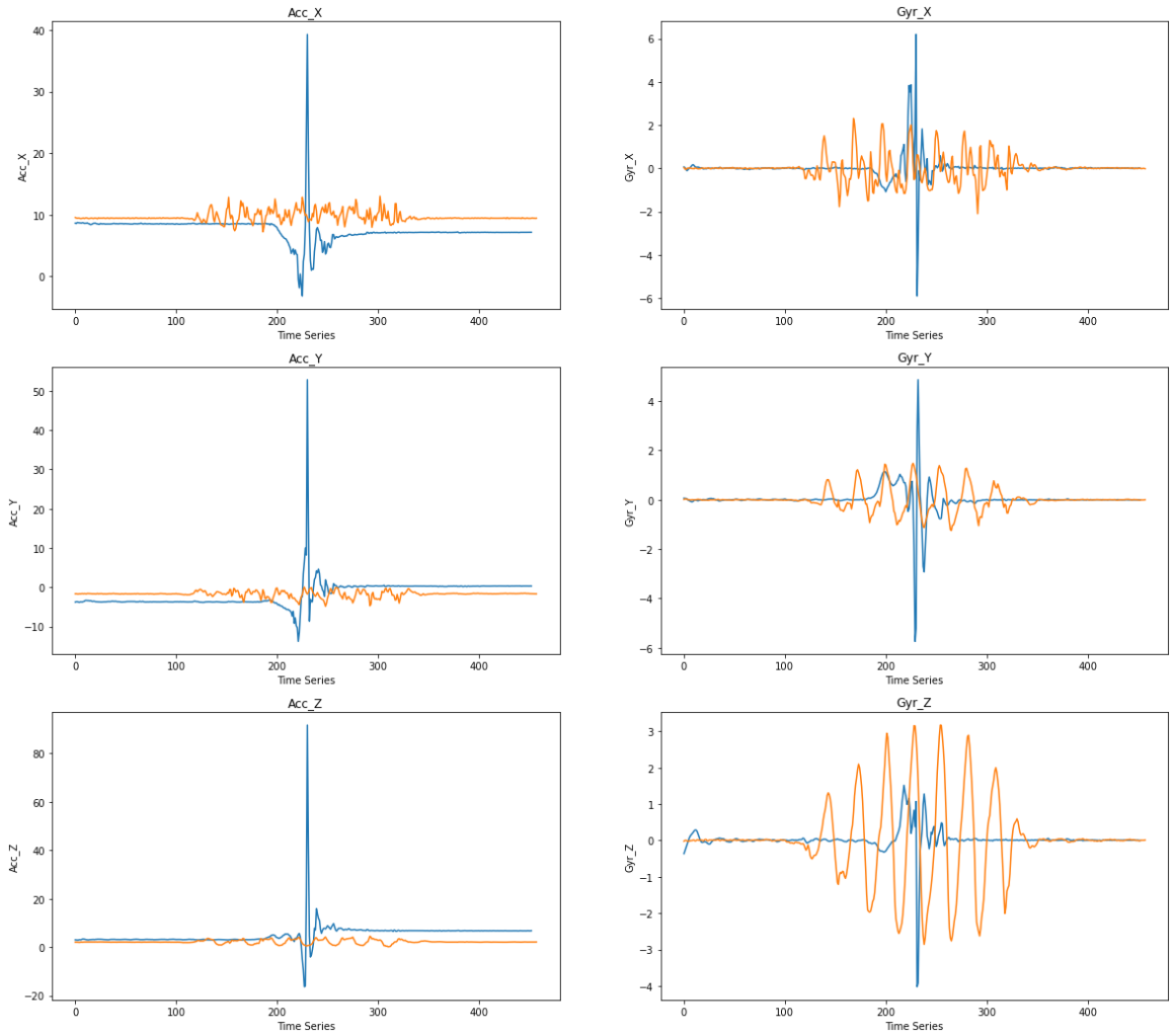


Figure 4.2.1-1 Falling VS. Non-Falling Data with Same Length

4.2.1.1 Truncate Data to Fixed Length

From data analysis in Section 3.3, we can see that most time series has the length between 400 to 600, and the key features appear around the mid point of the time peroid. So the key features of the majority of samples appear around 200 to 300 timestamp count. These observation gives us the support on data truncation. In practice, to avoid losing key features, we only use samples with length from 400 to 600 and we truncate all these samples to the unified length of 400. A demo of the truncated data is shown below in Figure 4.1.1.1-1

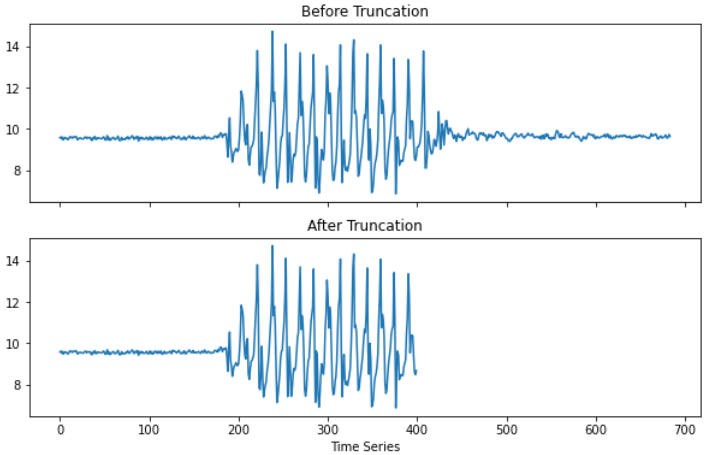


Figure 4.2.1.1-1 Data Truncation Demo

4.2.1.2 Sample Padding with Mean

4.2.1.2.1 Suffix Padding

Data truncation has a rick of lossing valuable information in data. Extending data to a similar length is a better choice. We try to extend all samples which has a length smaller than 600 with Numerical Mean of the last 20% data from the tail of time series and truncate samples longer than 600 timestamp to make the record length equals to a unified 600 rows. A demo of the extended data is shown below in Figure 4.1.1.2-1

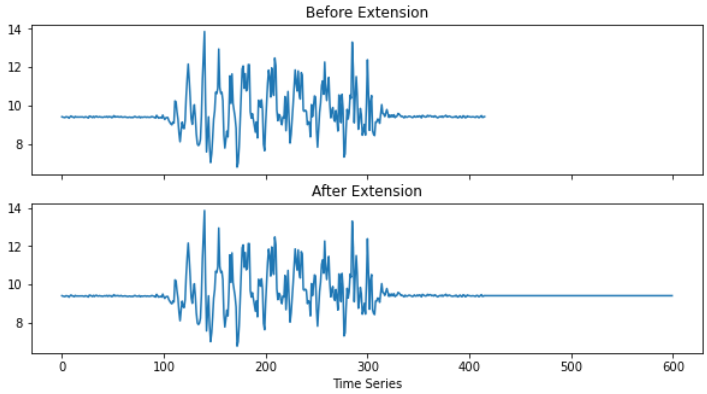


Figure 4.2.1.1-1 Data Truncation Demo

4.2.1.2.2 Suffix and Prefix Padding

Comparing to suffix padding alone, the combination of suffix and prefix padding allow us to better align key features arond the mid-point of any given fixed length time series. In this sub-approach, original data is extended towards both suffix and prefix end with the mean of the 10% closest data.

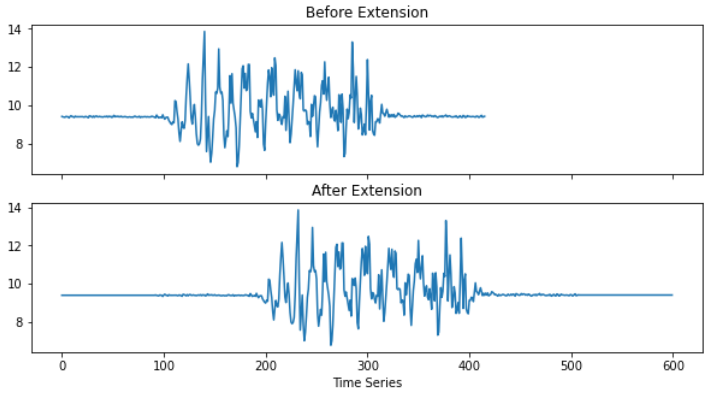


Figure 4.2.1.1-1 Data Truncation Demo

4.2.1.3 Uniform Scaling

We can use uniform scaling to scale up or scale down samples to make them in uniform length of 600 timestamps. Uniform Scaling (Keogh, E., 2003) works as follows:

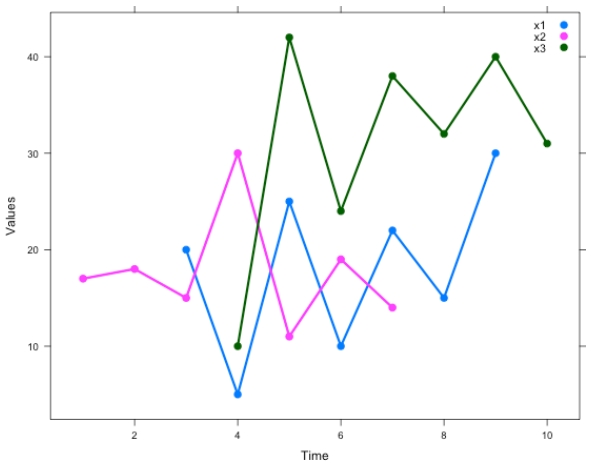


Figure 4.2.1.3-1 Before Uniform Scaling

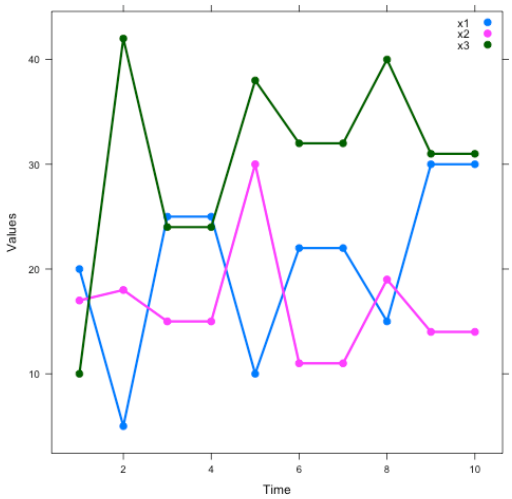


Figure 4.2.1.3-2 After Uniform Scaling

Q is a time series with length n

$$Q = q_1, q_1, \ldots, q_i, \ldots q_n$$

To scale time series Q to produce a new time series QP of length p, the formula is:

$$QP_i = Q_{\left\lceil j*\frac{n}{p} \right\rceil}, 1 \leq j \leq p$$

A demo of scaled time series is shown in Figure 4.1.1.3-3

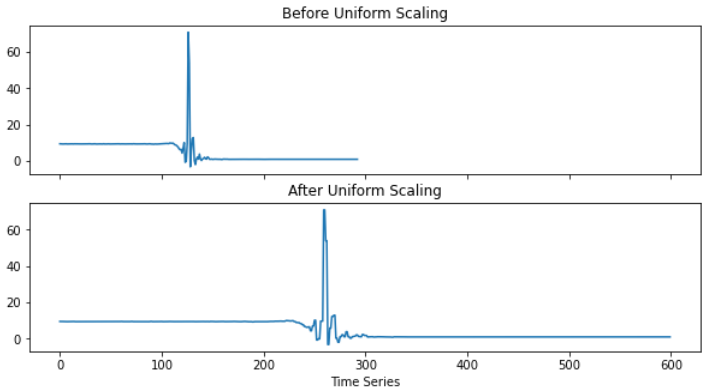


Figure 4.2.1.3-3 Uniform Scaling Demo

4.2.1.4 Sample Padding with Regression Prediction

The scikit-Learn package Iterative Imputer is used to process data. This models each feature with missing values as a function of other features, and uses that estimate for imputation. It does so in an iterated round-robin fashion: at each step, a feature column is designated as output `y` and the other feature columns are treated as inputs `X`. A regressor is fit on `(X, y)` for known `y`. Then, the regressor is used to predict the missing values of `y`. A demo of extended time series is shown in Figure 4.1.1.4.-1

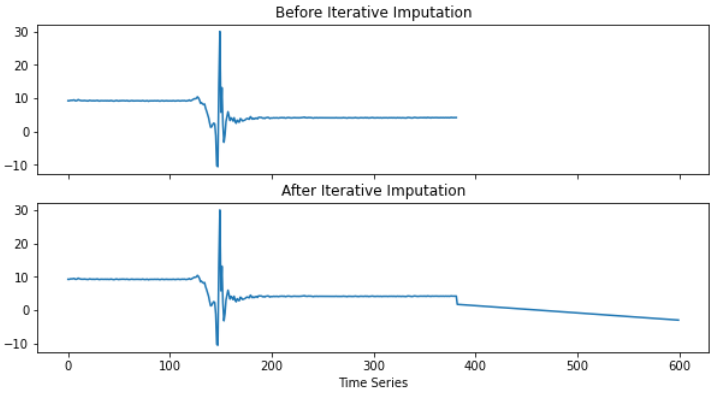


Figure 4.2.1.4-1 Iterative Imputer Demo

4.2.2 Model Training

Two algorithms wich works on fixed-length time series classification is applied on the preprocessed data, RocketClassifier and HiveCoteV2. The input feature data is a 3D-array of shape (n,6,m). `n` is the number of samples, `6` is the 6 feature we select (`Acc_X`, `Acc_Y`, `Acc_Z`, `Gyr_X`, `Gyr_Y`, `Gyr_Z`), `m` is the length of the time series and `m` should be either 400 or 600 based on how the data is preprocessed. The input label data is a 1D-arry with length `m`.

4.2.2.1 RocketClassifier

The RocketClassifier from sktime is used. RockectClassifier is a classifier wrapped for the ROCKET transformer using RidgeClassifierCV. Most methods for time series classification that attain state-of-the-art accuracy have high computational complexity, requiring significant training time even for smaller datasets, and are intractable for larger datasets. Additionally, many existing methods focus on a single type of feature such as shape or frequency. Building on the recent success of convolutional neural networks for time series classification, this simple linear classifiers using random convolutional kernels achieve state-of-the-art accuracy with a fraction of the computational expense of existing methods is implemented.

4.2.2.2 HiveCoteV2

The HIVECOTEV2 package from sktime is used. HIVE-COTE2 stands for Hierarchical Vote Collective of Transformation-based Ensembles V2. It is an ensemble of the STC, DrCIF, Arsenal and TDE classifiers from different feature representations using the CAWPE structure as described by Middlehurst e.t (2021). HIVE-COTE forms its ensemble from classifiers of multiple domains, including phase-independent shapelets, bag-of-words based dictionaries and phase-dependent intervals. Since it was first proposed in 2016, the algorithm has remained state of the art for accuracy on the UCR time series classification archive. Over time it has been incrementally updated, culminating in its current state, HIVE-COTE 1.0. Comprehensive changes were proposed to the HIVE-COTE algorithm which significantly improve its accuracy and usability, presenting this upgrade as HIVE-COTE 2.0. Two novel classifiers, the Temporal Dictionary Ensemble (TDE) and Diverse Representation Canonical Interval Forest (DrCIF) are introduced, which replace existing ensemble members. Additionally, the Arsenal, an ensemble of ROCKET classifiers are introduced as a new HIVE-COTE 2.0 constituent. HIVE-COTE 2.0 is significantly more accurate than the current state of the art on 112 univariate UCR archive datasets and 26 multivariate UEA archive datasets.

4.2.3 Model Evaluation

Training on different combination of preprocess methods and algorithms are conducted, for convinience, preprocess methods and training model will be replaced by the following notations:

Preprocess Method	Shorthand Notation
Truncate Data to Fixed Length	P1
Suffix Padding	P2
Suffix and Prefix Padding	P3
Uniform Scaling	P4
Regression Prediction Padding	P5

Machine Learning Model	Shorthand Notation
RocketClassifier	M1
HiveCoteV2	M2

The combination of Truncate Data to Fixed Length as preprocess method and RocketClassifier as machine learning model is denoted as M1-P1 in the folloing sections.

4.2.3.1 Data Selection Approach 1: Data from Two Highly Distinguishable Action Categories

4.2.3.1.1 RocketClassifier

Hyper-parameter Setting

Data in data selection approach 1 are highly distinguishable, so hyper-parameters are kept as default settings, which is good enough to produce satisfying training results.

Hyper-parameter	Value
num_kernels	10000
rocket_transform	'rocket'
max_dilations_per_kernel	32
n_features_per_kernel	4
n_jobs	1
random_state	None

Performance Comparison

Evaluation Matrix	M1-P1	M1-P2	M1-P3	M1-P4	M1-P5
Training Time (Wall time)	11.8 s	25.9 s	26.5 s	25.8 s	26.1 s
Prediction Time (Wall time)	3.07 s	6.69 s	6.76 s	6.67 s	6.73 s
Accuracy Score	1.00	1.00	1.00	1.00	1.00
Precision Score	1.00	1.00	1.00	1.00	1.00
Recall Score	1.00	1.00	1.00	1.00	1.00
F1 Score	1.00	1.00	1.00	1.00	1.00
ROC AUC Score	1.00	1.00	1.00	1.00	1.00

Notice: The trianing in above table is performed on a ROG Strix Laptop with CPU: AMD Ryzen 9 5900HX with Radeon Graphics

Confusion Matrix & ROC Curve

Confusion Matrix and ROC Curve for RocketClassifier with different proprocess methods are similar, only one of them is presented below.

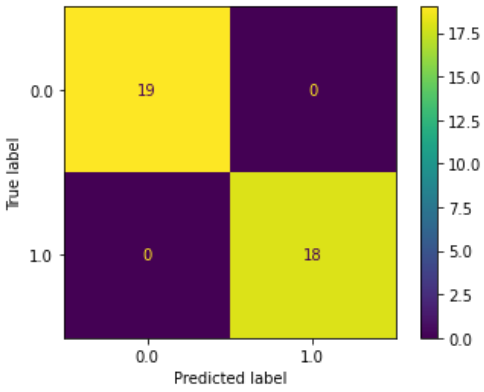


Figure 4.2.3.1.1-1 Confusion Matrix

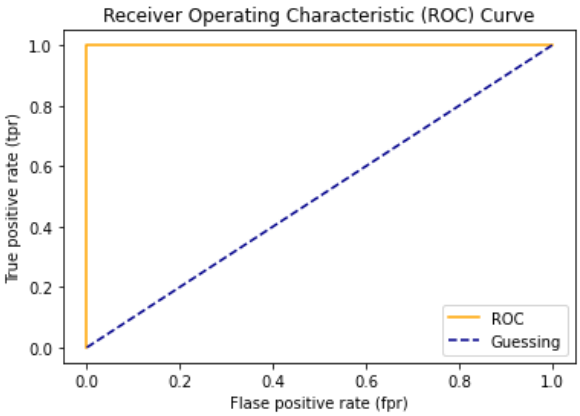


Figure 4.2.3.1.1-2 ROC Curve

4.2.3.1.2 HiveCoteV2

Hyper-parameter Setting

Hyper-parameters are kept as default settings except `time_limit_in_minutes` is set to 1.

Hyper-parameter	Value
stc_params	None
drcif_params	None
arsenal_params	None
tde_params	None
time_limit_in_minutes	0
save_component_probas	False
verbose	0
n_jobs	1
random_state	None

Performance Comparison

Evaluation Matrix	M2-P1	M2-P2	M2-P3	M2-P4	M2-P5
Training Time (Wall time)	1min	1min 23s	1min 30s	1min 16s	1min 26s
Prediction Time (Wall time)	10.4 s	14.7 s	14 s	12.9 s	13.1 s
Accuracy Score	1.00	1.00	1.00	1.00	1.00
Precision Score	1.00	1.00	1.00	1.00	1.00
Recall Score	1.00	1.00	1.00	1.00	1.00
F1 Score	1.00	1.00	1.00	1.00	1.00
ROC AUC Score	1.00	1.00	1.00	1.00	1.00

Notice: The trianing in above table is performed on a ROG Strix Laptop with CPU: AMD Ryzen 9 5900HX with Radeon Graphics

Confusion Matrix & ROC Curve

Confusion Matrix and ROC Curve for HiveCoteV2 with different proprocess methods are similar, only one of them is presented below.

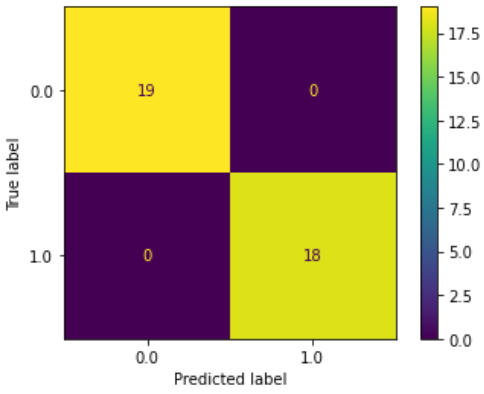


Figure 4.2.3.1.2-1 Confusion Matrix

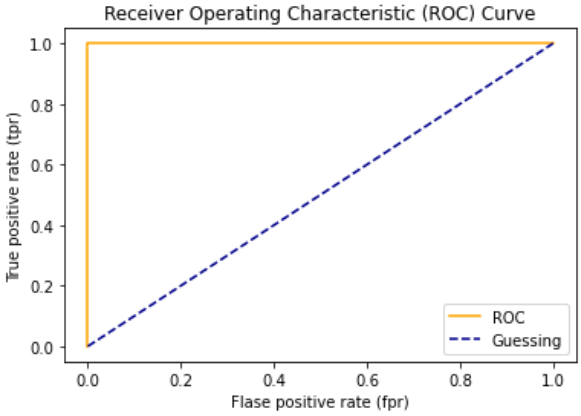


Figure 4.2.3.1.2-2 ROC Curve

4.2.3.2 Data Selection Approach 2: Data from Multiple Action Categories

4.2.3.2.1 RocketClassifier

Hyper-parameter Setting

Hyper-parameters are kept as default settings.

Hyper-parameter	Value
num_kernels	10000
rocket_transform	'rocket'
max_dilations_per_kernel	32
n_features_per_kernel	4
n_jobs	1
random_state	None

Performance Comparison

Evaluation Matrix	M1-P1	M1-P2	M1-P3	M1-P4	M1-P5
Training Time (Wall time)	11.3 s	25.8 s	24.9 s	24.5 s	24.4 s
Prediction Time (Wall time)	2.97 s	6.49 s	6.28 s	6.18 s	6.1 s
Accuracy Score	0.91	0.94	0.97	0.97	0.97
Precision Score	0.92	0.88	1.00	0.93	0.93
Recall Score	0.92	1.00	0.93	1.00	1.00
F1 Score	0.92	0.93	0.96	0.97	0.97
ROC AUC Score	0.92	0.95	0.96	0.97	0.97

Notice: The trianing in above table is performed on a ROG Strix Laptop with CPU: AMD Ryzen 9 5900HX with Radeon Graphics

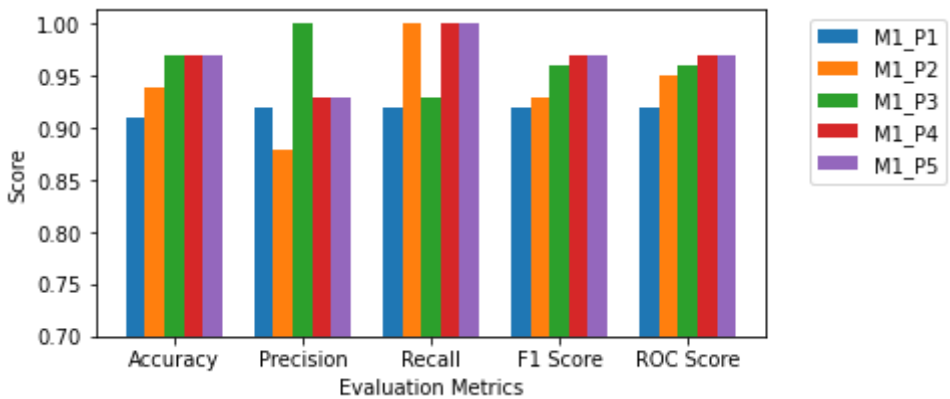


Figure 4.2.3.2.1 Performance Comparison

Confusion Matrix & ROC Curve

- M1-P1

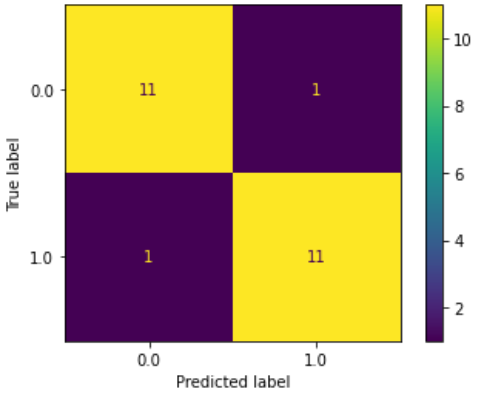


Figure 4.2.3.2.1-1 M1-P1 Confusion Matrix

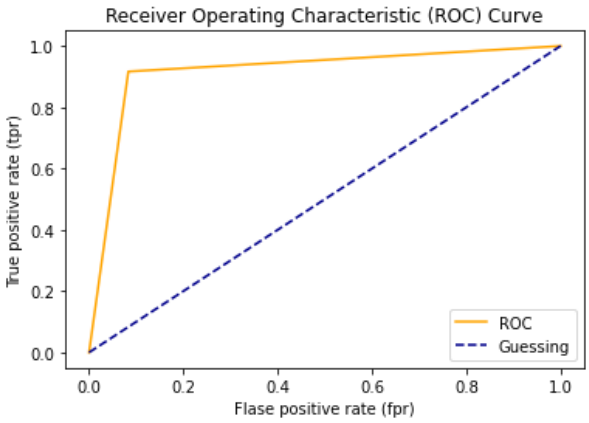


Figure 4.2.3.2.1-2 M1-P1 ROC Curve

- M1-P2

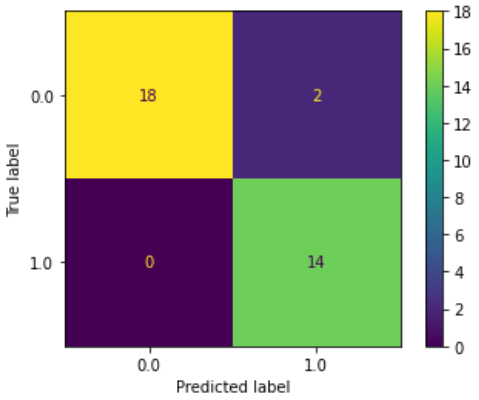


Figure 4.2.3.2.1-3 M1-P2 Confusion Matrix

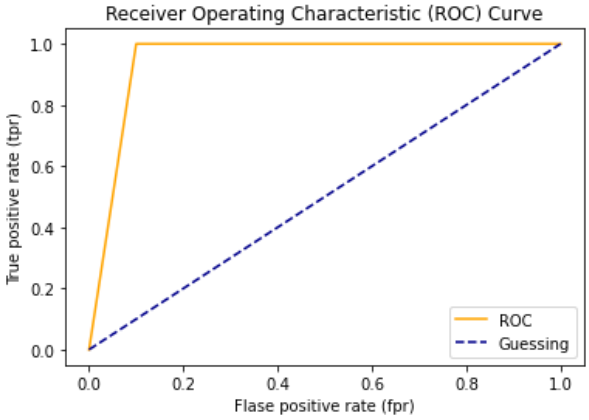


Figure 4.2.3.2.1-4 M1-P2 ROC Curve

- M1-P3

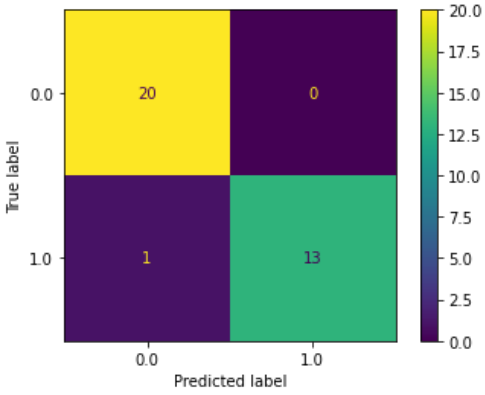


Figure 4.2.3.2.1-5 M1-P3 Confusion Matrix

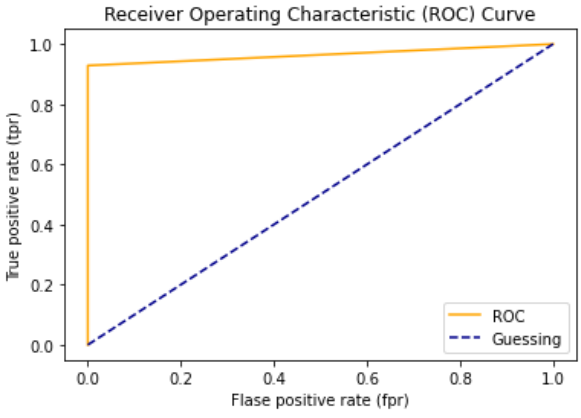


Figure 4.2.3.2.1-6 M1-P3 ROC Curve

- M1-P4

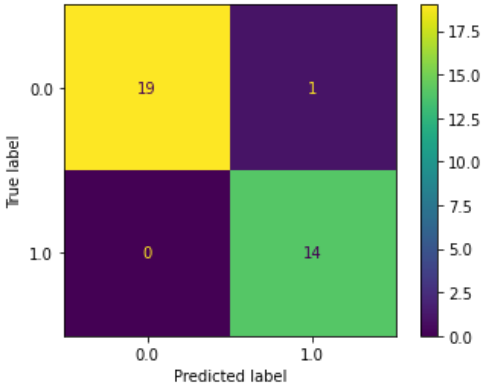


Figure 4.2.3.2.1-7 M1-P4 Confusion Matrix

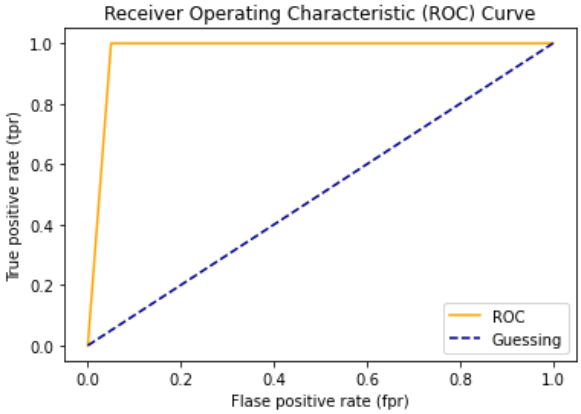


Figure 4.2.3.2.1-8 M1-P4 ROC Curve

- M1-P5

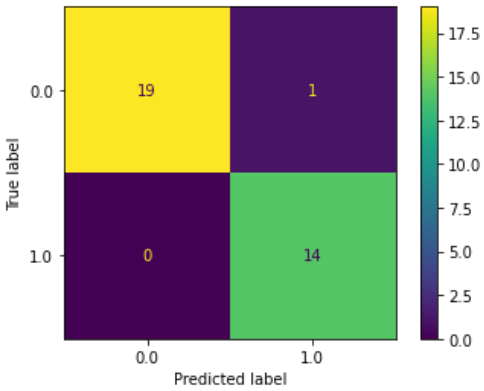


Figure 4.2.3.2.1-9 M1-P5 Confusion Matrix

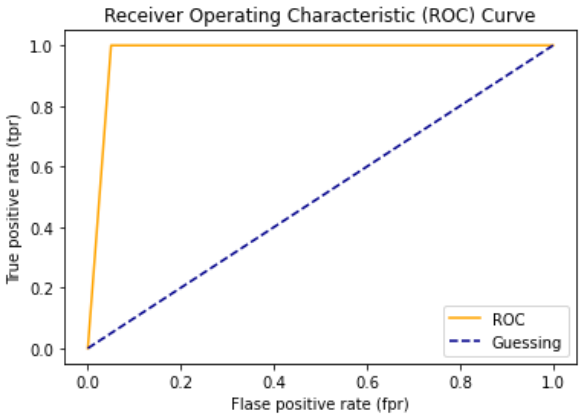


Figure 4.2.3.2.1-10 M1-P5 ROC Curve

4.2.3.2.2 HiveCoteV2

Hyper-parameter Setting

Data in data selection approach 1 are highly distinguishable, so hyper-parameters are kept as default settings except `time_limit_in_minutes` is set to 1. Since the performance are extremly good, so there is nood need for us the tune hyper-parameters.

Hyper-parameter	Value
stc_params	None
drcif_params	None
arsenal_params	None
tde_params	None
time_limit_in_minutes	0
save_component_probas	False
verbose	0
n_jobs	1
random_state	None

Performance Comparison

Evaluation Matrix	M2-P1	M2-P2	M2-P3	M2-P4	M2-P5
Training Time (Wall time)	1min 2s	1min 32s	1min 26s	1min 31s	1min 25s
Prediction Time (Wall time)	9.87 s	14 s	14.4 s	13.8 s	12.7 s
Accuracy Score	0.92	0.97	0.85	0.97	0.97
Precision Score	0.92	0.93	0.74	0.93	0.93
Recall Score	0.92	1.00	1.00	1.00	1.00
F1 Score	0.92	0.97	0.85	0.97	0.97
ROC AUC Score	0.98	1.00	1.00	1.00	1.00

Notice: The trianing in above table is performed on a ROG Strix Laptop with CPU: AMD Ryzen 9 5900HX with Radeon Graphics

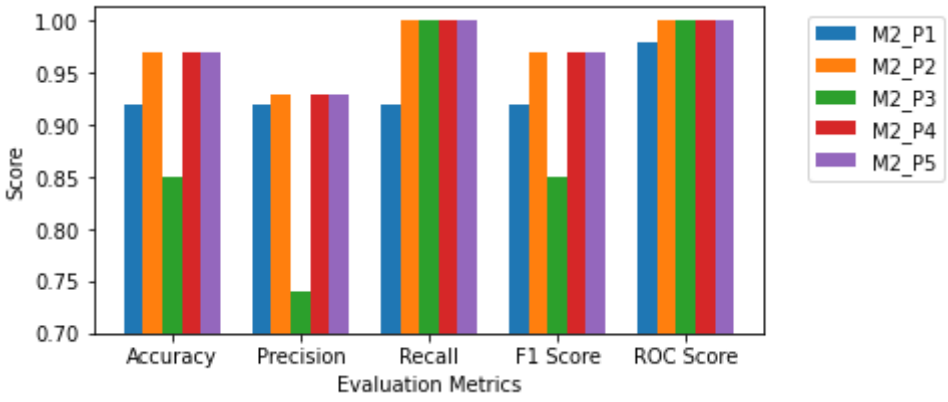


Figure 4.2.3.2.2 Performance Comparison

Confusion Matrix & ROC Curve

- M2-P1

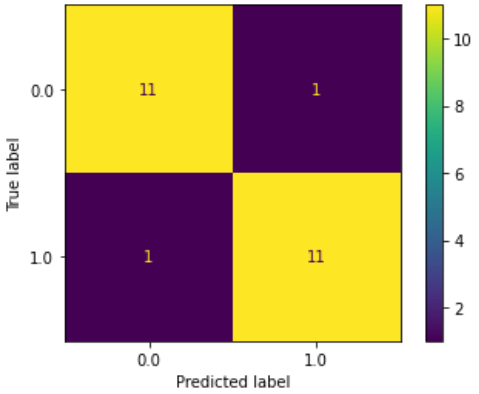


Figure 4.2.3.2.2-1 M2-P1 Confusion Matrix

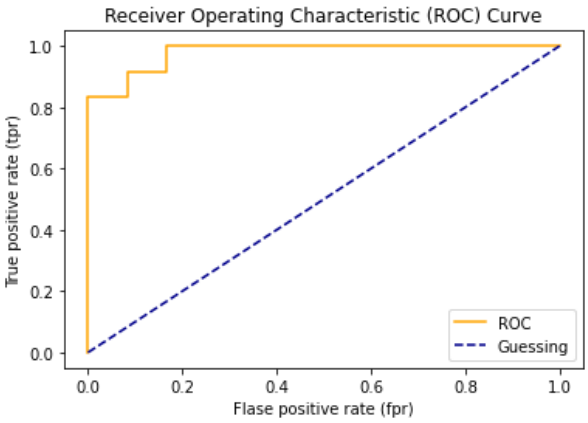


Figure 4.2.3.2.2-2 M2-P1 ROC Curve

- M2-P2

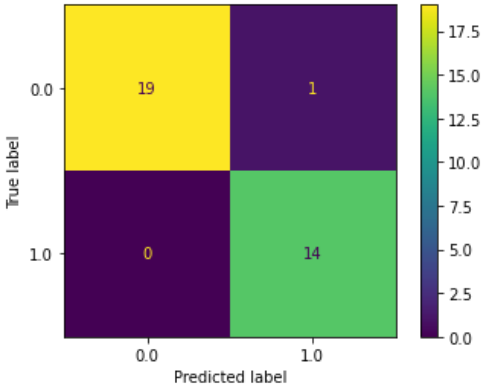


Figure 4.2.3.2.2-3 M2-P2 Confusion Matrix

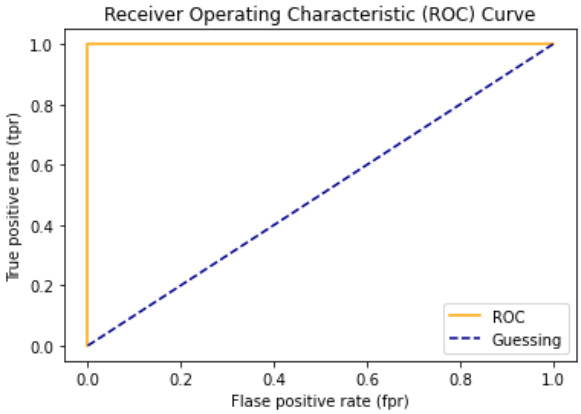


Figure 4.2.3.2.2-4 M2-P2 ROC Curve

- M2-P3

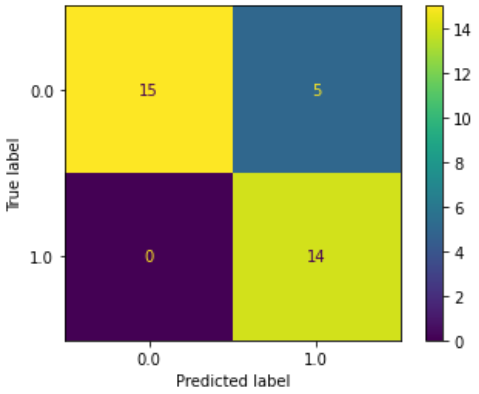


Figure 4.2.3.2.2-5 M2-P3 Confusion Matrix

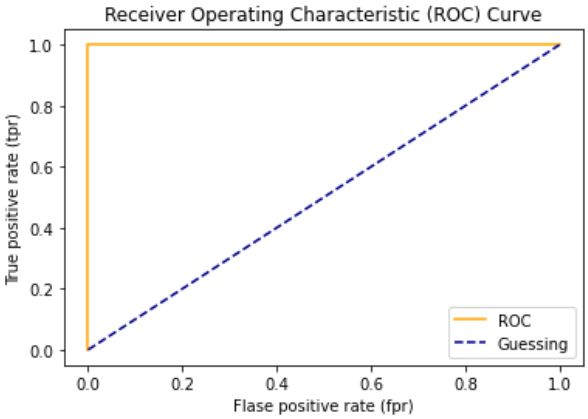


Figure 4.2.3.2.2-6 M2-P3 ROC Curve

- M2-P4

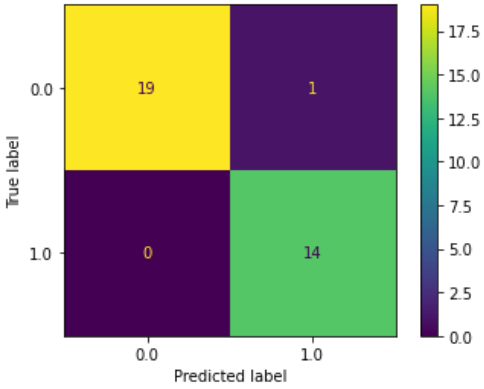


Figure 4.2.3.2.2-7 M2-P4 Confusion Matrix

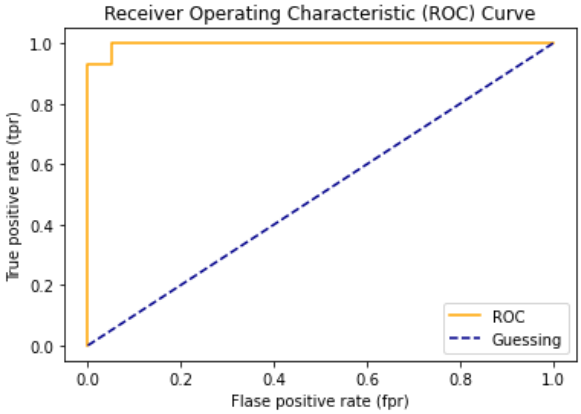


Figure 4.2.3.2.2-8 M2-P4 ROC Curve

- M2-P5

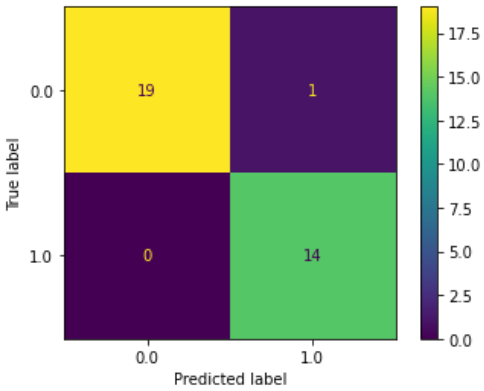


Figure 4.2.3.2.2-9 M2-P5 Confusion Matrix

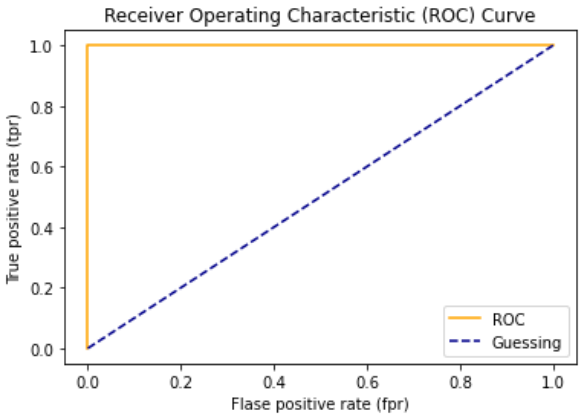


Figure 4.2.3.2.2-10 M2-P5 ROC Curve

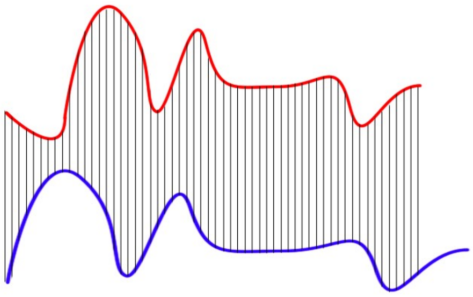
4.3 TSC Approach 3: Time Series Classification Algorithms on Varying Length Time Series

KNN & DTW

Traditionally KNN algorithm focuses on the Euclidean distance among an unlabeled vector and all vectors in the training phase and then assign the label which is most frequent among the k training samples nearest to the test point. Due to the prediction of falls in time series, the things different from using traditional KNN to predict at a given point is to predict the behavior in a sequence of time. Moreover, since two time series may have different length, only Euclidean distance method which is a kind of one-to-one match does not perform well (Figure 4.1.1). DTW overcomes this issue by developing a one-to-many match so that the troughs and peaks with the same pattern are perfectly matched and there is no left out for both curves (Figure 4.1.2) (Jeremy, 2020). Therefore, DTW is applied to combining with KNN approach.

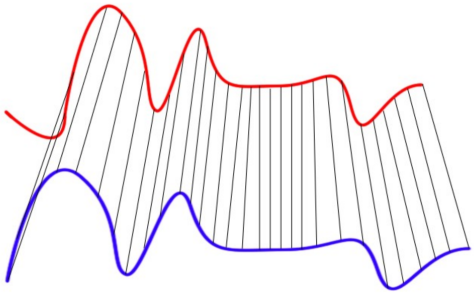
Dynamic time warping (DTW) is an algorithm designed for measuring the similarity between two temporal sequence in time series analysis. It follows certain restriction and rules:

- Every index from the first sequence must be matched with one or more indices from the other sequence, and vice versa.



Euclidean Matching

Figure 4.3.1



Dynamic Time Warping Matching

Figure 4.3.2

- The first index from the first sequence must be matched with the first index from the other sequence (but it does not have to be its only match).
- The last index from the first sequence must be matched with the last index from the other sequence (but it does not have to be its only match).
- The mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing, and vice versa, i.e. if **j** > **i** are indices from the first sequence, then there must not be two indices **l** > **k** in the other sequence, such that index **i** is matched with index **l** and index **j** is matched with index **k**, and vice versa ("Dynamic time warping", n.d.).

In order to predict the falls in time series, the matrix whose columns represent the values of features and rows represent records at given time points is built. Denote one such matrix derived from training data as X and one such matrix derived from test data as Y. Then the smallest path can be found using dynamic programming to evaluate the following recurrence, which defines the cumulative distance matrix (D) at each entry as the distance found in the current cell and the minimum of the cumulative distances of the adjacent elements (Shokoohi-Yekta et al., 2017):

$$D[i][j] = d(y_i,\ x_j) + min(D[i-1][j-1],\ D[i-1][j],\ D[i][j-1]), \quad d(y_i,\ x_j) = |y_i - x_j|$$

The above formula applies DTW to the single feature prediction, so y_i and x_j are the elements and the distance is calculated by the absolute value of the difference between these two elements. However, the features used for falls prediction are 3 axis accelerometer, gyroscope and magnetometer. In this case, y_i and x_j are the row vectors, so the distance is calculated by the Euclidean distance between these two vectors. Here is a general form of DTW:

$$D[i][j] = d(y_i,\ x_j) + min(D[i-1][j-1],\ D[i-1][j],\ D[i][j-1]), \quad d(y_i,\ x_j) = \sum (y_{in} - x_{jn})^2$$

And then np.shape function in numpy library is imported to get the dimension (m x n) of the D matrix, so the smallest path equals to $D[m-1][n-1]$. Finally, after the distance among the test point and all training samples is calculated, the label which is most frequent among the k training samples nearest to the test point is assigned.

5. Results

5.1 TSC Approach 1: Reform Problem to Non Time Series Classification - SVM

SVM with Data Processing

Data Selection Approach 2: Data from Multiple Action Categories

After data processing and testing & training data split for several times, SVM is proved to be a good model for falling ADLs detection.

Before showing the results of the model, here is one more issue to be mentioned. There are several different kernel functions SVM model, and different kernel functions may affect the prediction accuracy. For each available kernel function, 10 times training and testing are carried out and linear function turns out to be the best, which give the highest prediction accuracy 93.02% and a mean prediction accuracy around 88% over 10 times testing.

Among 43 testing cases, 40 cases are correctly detected, 3 non-falling cases are wrongly labeled as falling, no falling cases are labeled at non-falling.

```
prediction
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0]
real
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1]
```

Figure 5.1.1

5.2 TSC Approach 2: Time Series Classification Algorithms on Fixed Length Time Series

5.2.1 Data Selection Approach 1: Data from Two Highly Distinguishable Action Categories

By comparing the performance of RocketClassifier and HiveCoteV2 algorithms with combination of different preprocessing methods, we can have the folloing observation:

- Performance of both RocketClassifier and HiveCoteV2 with any of the preprocessing methods has extremly good performance, always has an accuracy of 100%

We can make the following inference based on the above observation:

- The dataset is too distinguishable that the two FALL and NON-FALL categories can be classified easily.

  The inference insipired us to change the formation of dataset and lead us to pick a second approach in data selection.

5.2.2 Data Selection Approach 2: Data from Multiple Action Categories

By comparing the performance of RocketClassifier and HiveCoteV2 algorithms with combination of different preprocessing methods, we can have the folloing observation:

- RocketClassifier requires less training time and predicting time in general comparing to HiveCoteV2.
- Model's performance is highly related to preprocessing methods.
- Truncate Data to Fixed Length preprocessing method has the poorest performance among the 5 selected methods.
- Sample Padding with Regression Prediction and Uniform Scaling preprocessing methods have the best performance among the 5 selected methods.
- Suffix Padding with Mean preprocessing method has a satisfying performance with both of the two model.

- Suffix and Prefix Padding with Mean preprocessing method has good performance with RocketClassifier but has poor performance with HiveCoteV2.

5.3 TSC Approach 3: Time Series Classification Algorithms on Varying Length Time Series

KNN & DTW

Data Selection Approach 2: Data from Multiple Action Categories

With the application of KNN and DTW algorithm, the accuracy of prediction is 79.41%. In detail, the false positive rate is 0, which means that there do not exist the situation where no falls is predicted as falls. Additionally, the false negative rate is 24.14%, which means that there are 24.14% proportion of falls is wrongly predicted as no falls. Focused on the specific kind of falls, the falls -- "909 back-sitting, from vertical falling on the floor, ending sitting" -- is hard for prediction.

In order to improve the accuracy, especially in the aspect of improving the precision of correctly predicting falls, data preprocessing such as data normalization plays an important role. This is because some large scale features will dominate in computing the Euclidean distance. Two kinds of normalization methods are used.

L1 Normalization

As for the given feature, its values consist of a column vector X. As for each value,

$$X_i = \frac{X_i}{\sum |X_i|}$$

Figure 5.1.1 shows the example data with no falls before normalization and it can be found that different features have different scale values, which may affect the performance of KNN. Figure 5.1.2 shows the data after L1 normalization and it can be seen that all features are scaled in the same range. And then those normalized data samples are used for prediction. The results show that the accuracy reaches 100%, which means all the categories of falls or no falls in a given time series are correctly predicted.

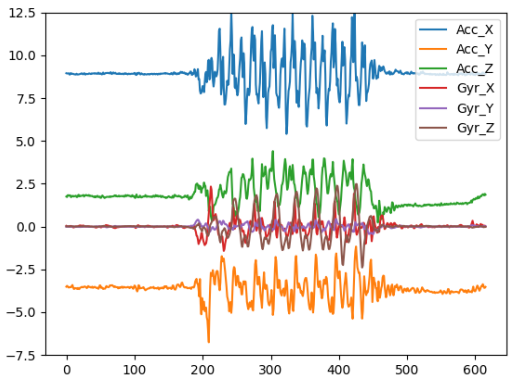


Figure 5.3.1

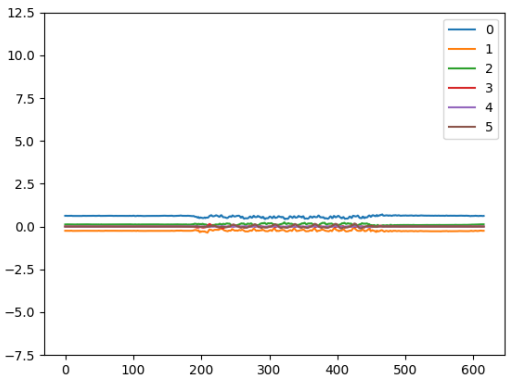


Figure 5.3.2

L2 Normalization

As for the given feature, its values consist of a column vector X. As for each value,

$$X_i = \frac{X_i}{\sqrt{\sum X_i^2}}$$

Figure 5.1.3 shows the example data with falls before normalization and it can be found that different features have different scale values, which may affect the performance of KNN. Figure 5.1.4 shows the data after L2 normalization and it can be seen that all features are scaled in the same range. And then those normalized data samples are used for prediction. The results show that the accuracy reaches 100%, which means all the categories of falls or no falls in a given time series are correctly predicted.

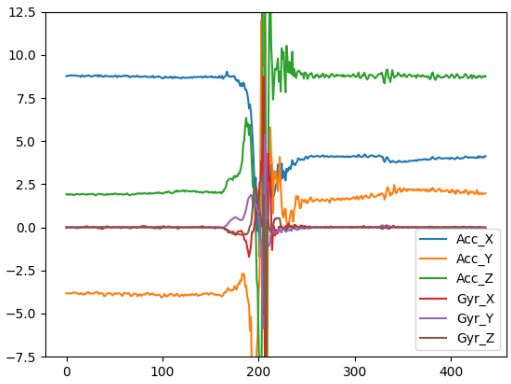


Figure 5.1.3

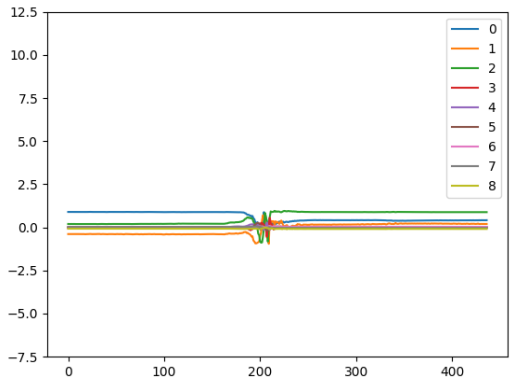


Figure 5.1.4

Therefore, data normalization is essential to KNN algorithm. And in this project, both L1 normalization and L2 normalization perform well.

6. Conclusions

The three approaches of solving the variabel-length time series classification problem, Time Series Classification Algorithms on Fixed Length Time Series and Time Series Classification Algorithms on Varying Length Time Series are explored. Both of these two approches are proved to be able to solve the problem as shown in our project. The strength and weakness of different approach, models and preprocessing methods are also compared.

Future study can make improvements in multiple directions. First, more work can be done in enlarging and adding further diversity to the dataset, for example including more classes from both FALL and NON-FALL categories. Second, a large space of performance improvement exists in hyper-parameter tunning. Third, more data preprocessing methods and time series classification algorithms can be taken into the study. Fourth, the reason why certain data preprocessing methods performs poorly with some models but performs satisfyingly with some other models can be further investigated into.

Reference

Ã–zdemir, A.T., Barshan, B. (2014). Simulated Falls and Daily Living Activities Data Set [Data set]. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Simulated+Falls+and+Daily+Living+Activities+Data+Set#

Dempster, Angus and Petitjean, Francois and Webb, Geoffrey I, ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels, 2019, arXiv:1910.13051 https://doi.org/10.48550/arXiv.1910.13051

Dynamic time warping. (2022, April 11). In Wikipedia. https://en.wikipedia.org/wiki/Dynamic_time_warping

Jeremy, Z. (2020, Feburary 1). Dynamic time warping. Medium. https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd

Keogh, E. (2003). Efficiently Finding Arbitrarily Scaled Patterns in Massive Time Series Databases. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds) Knowledge Discovery in Databases: PKDD 2003. PKDD 2003. Lecture Notes in Computer Science(), vol 2838. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-39804-2_24

Middlehurst, Matthew, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. "HIVE-COTE 2.0: a new meta ensemble for time series classification." Machine Learning (2021). https://doi.org/10.48550/arXiv.2104.07551

Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., & Keogh, E. (2017). Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, *31*(1), 1–31. https://doi.org/10.1007/s10618-016-0455-0

Tan, Chang Wei & Petitjean, François & Keogh, Eamonn & Webb, Geoffrey. (2019). Time series classification for varying length series. https://doi.org/10.48550/arXiv.1910.04341