**Title:** Investigating the Accuracy and Prediction Speed of K-Nearest Neighbours and Support Vector Machines on the Classification of Skateboarding Tricks Using Inertial Measurement Unit Signals

**Research question:** How does the accuracy and prediction speed of K-Nearest Neighbours compare to the use of Support Vector Machines in detecting skateboarding tricks using Inertial Measurement Unit Signals.

# Contents

### Introduction

Skateboarding is a sport pursued by people all around the world, it involves riding on a board with four wheels, and performing complex tricks which involve coordination, balance and creativity of the skater. With the limitless possibilities of tricks able to be performed, skateboarding encapsulates skill sets of all levels, and the highest level being within competition skateboarding, with skateboarding's debut in the 2020 Tokyo Olympics. Evaluation of tricks is purely subjective, making it open to bias and imprecise scoring. Under rapid movement, it becomes difficult to identify what trick is being performed. The Head Judge for Skatepark of Tampa & Board pointed to the many difficulties in providing a judgment in a skateboarding event (Pappalardo, 2014). My research question proposes the idea of using support vector machines (SVM) and K- nearest neighbors (KNN) to perform data classification upon gyroscope and accelerometer data using an Inertial Measurement Unit (IMU). The investigation aims to detect and classify what trick a skateboarder performs, with the objective of this paper hoping to facilitate skateboarding competition judging, where accuracy and execution time of detecting tricks is extremely important. I will be comparing different data classification algorithms by its accuracy and prediction speed.

## Background Information

### Machine Learning Classification Algorithms

There are two types of machine learning algorithms, supervised and unsupervised. The main difference between them is supervised algorithm is trained upon labeled data, and unsupervised is trained upon unlabeled data. For our cases of classification, we must use labeled data, eg. the

name of the trick, performed to train our machine learning model. A classification algorithm is a type of supervised machine learning technique used to identify the category of new observations based upon old training observations. A trained model can be used to identify the class of a given dataset and output it correspondingly. Classification algorithms are used in many real life situations, some of them being:

- Character recognition
- Email spam detection
- Biometric identification (such as identifying cancerous tumors cells)

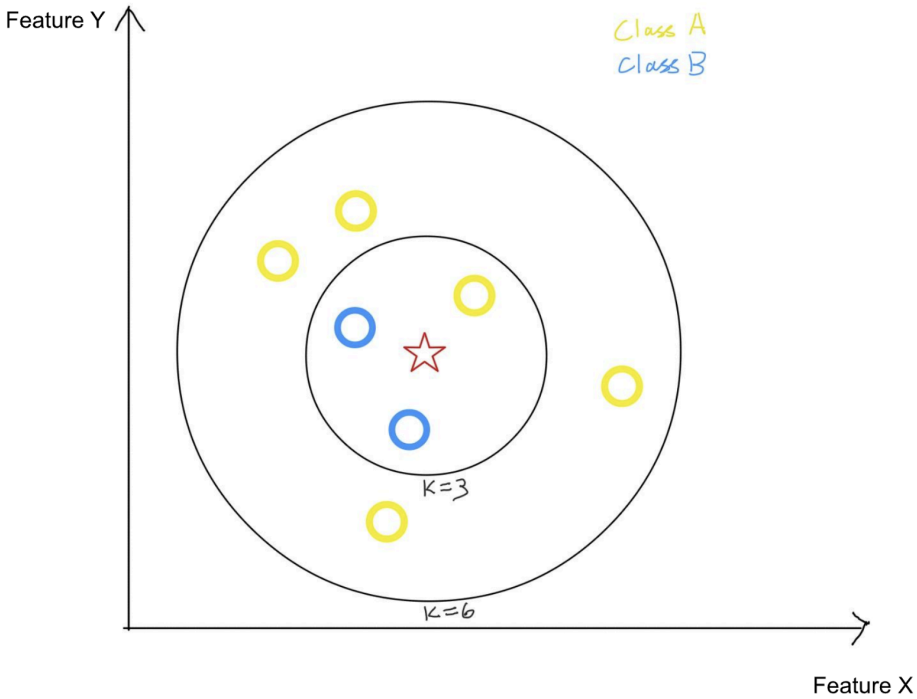
There are many classification algorithms. This paper will focus mainly on K-nearest neighbors and support vector machines

**K-Nearest Neighbors**

K-Nearest neighbor (KNN), is one of the simplest and fundamental supervised machine learning algorithms. Each skateboarding trick has its own unique data pattern. Selected for its ease of use, KNNs classify new skateboarding data by comparing its pattern with similar pre-classified patterns. In KNNs, data points are mapped onto an n-dimensional plane, with the axis being different features (an individual measurable property of the data), and new data points are classified by their proximity to neighboring data points. The algorithm does this by calculating the distance between each data point. This is typically done by using metrics such as the Euclidean distance.

KNN classifies data with a majority voting system, selecting the majority vote of K-nearest neighbors. We usually select odd numbers of K as a rule of thumb, this is to avoid ties in voting (Varghese, 2018).

Fig. 1 illustrates the process of KNNs. With class A being yellow and class B being blue, every observation is plotted onto a 2-dimensional plane in this case, with Feature X and Feature Y as its axis. With a new observation plotted (star), the distance between the point and all other points are calculated. The most frequent class of its K closest neighbors will be used to classify the new point. In this case, if K = 3, the data will be classified as class B (blue). If K = 6, the data will be classified as class A (yellow).




(Fig. 1: KNN voting system)

For a given data point x and its neighbor y, the Euclidean distance can be calculated with the formula:

$$d(x,\ y)\ =\ \sqrt{\sum_{i\ =\ 1}^{n}\left(x_{i}-y_{i}\right)^{2}}$$

Where $x,\ y$ are two points in Euclidean n-space

$x_i$, $y_i$ are Euclidean vectors

$n$ is the n-space or dimension

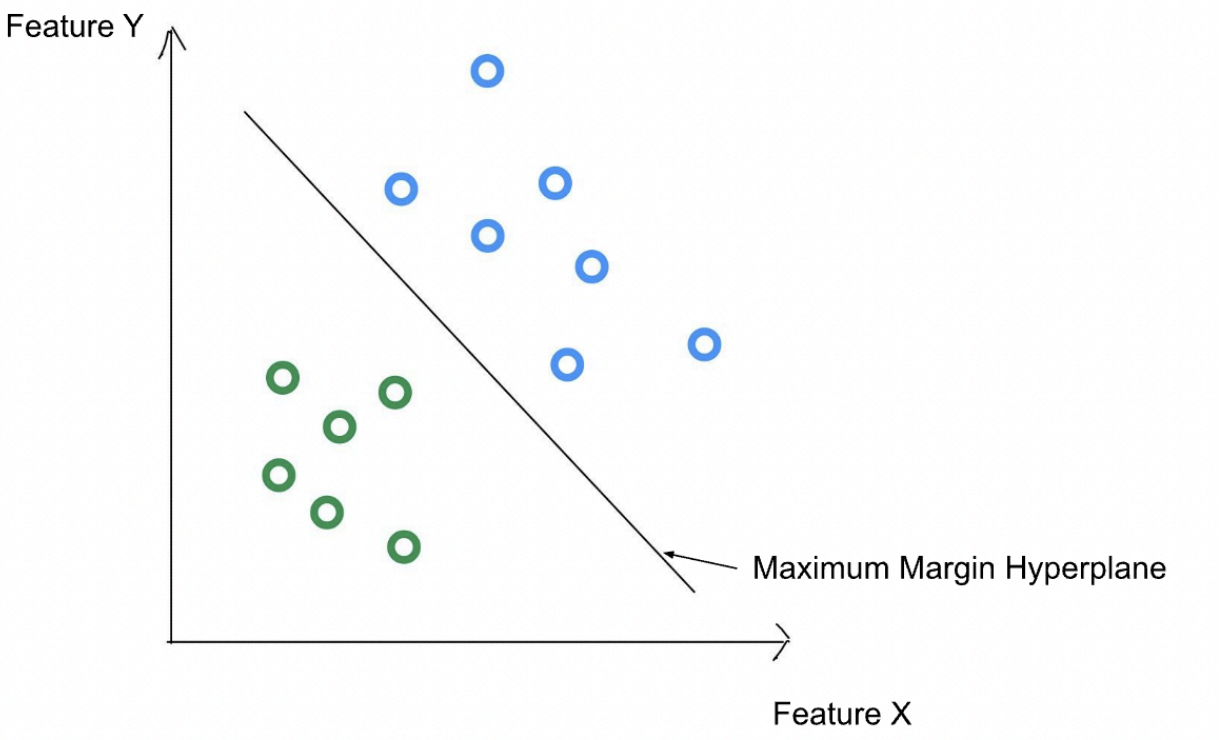
With the calculation of distance between all neighbors, the data point will be labeled with the most frequent class amongst its k-nearest neighbors. For our skateboarding case, each point will represent our IMU value measured, and each feature will be a given time frame. Each point will have its corresponding label, which is the trick performed.

In the context of skateboarding data, a graph of two IMU observations is plotted on a 2 dimensional scatterplot at two given time frames as its axis. The nearest neighbors of a point will represent the closest data points of tricks which have similar magnitude of acceleration or orientation at the two given time frames. By comparing a point with its neighboring points, it can be classified with the most commonly appearing class in its neighborhood.

**Support Vector Machines**

Support Vector Machine (SVM) is a type of supervised machine learning algorithm used for regression, outlier detection and, for our case, classification. SVM is selected for its complexity, its ability to find complex relationships in a wide range of situations. It does this by finding the optimal hyperplane, or maximum marginal hyperplane, which maximizes the margin between two classes of data points in a high dimensional space. This hyperplane separates two classes of data points, and can classify new data points by analyzing its location relative to the hyperplane.

**Linear SVM**



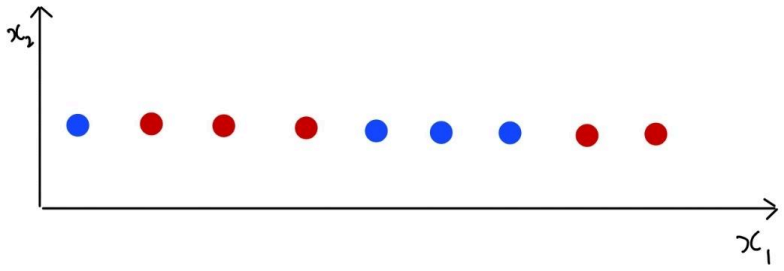

(Fig. 2: Illustration of a linear hyperplane)

As observed in Fig 2. The SVM will find the optimal hyperplane, placing it between two classes, blue and green. New observations will be classified according to its position relative to the hyperplane, with observations on the left of the hyperplane classified as green and observations on right classified as blue.

**Quadratic SVM**

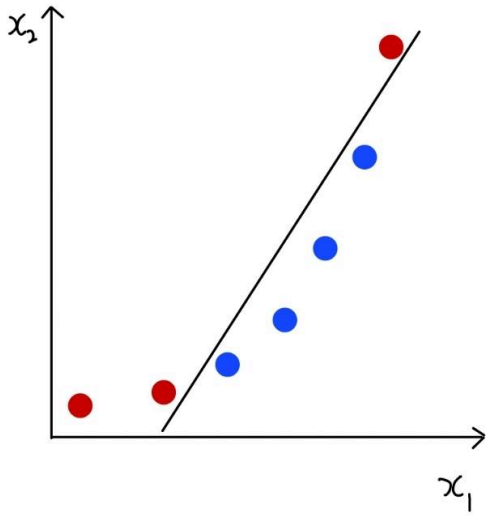
However, sometimes data may not be linearly separable. To get over this problem, SVMs will transform our data into a higher dimensional space.

By mapping the data into a higher dimensional space, linear separation can be performed. This is done by using kernel functions. As high dimension data is hard to visualize, let's start off with one dimensional data. Take the following set of data (Fig. 3), which is not linearly separable but able to be separated by a quadratic function.




(Fig. 3: Quadratically separable hyperplane)

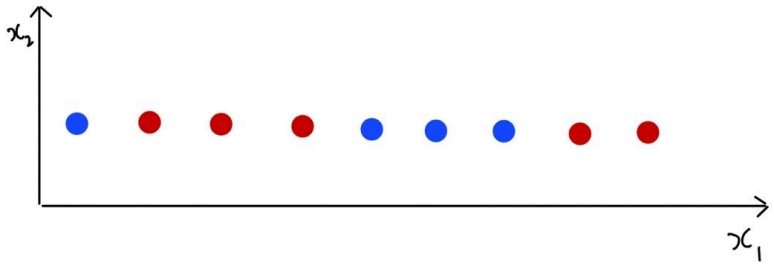
If we map $\phi(x) = x^2$ the data becomes linearly separable



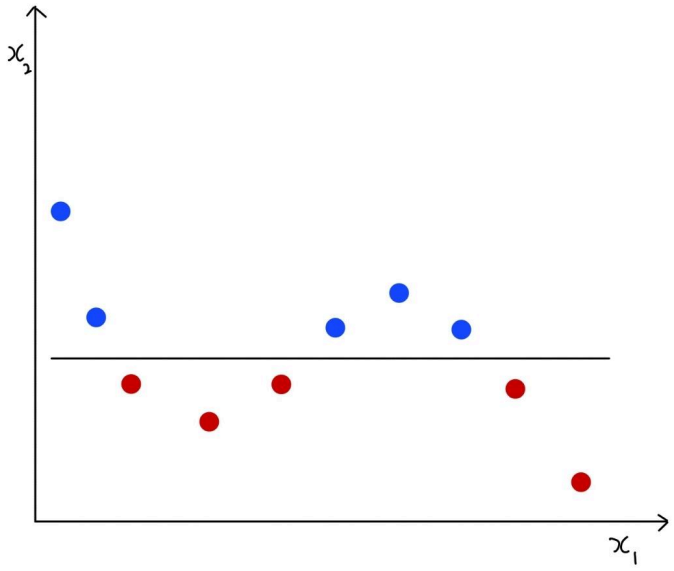

(Fig. 4: Quadratic transformation applied)

**Cubic SVM**

Similarity, data represented in fig 5. shows a non linearly separable dataset. However, when a cubic transformation is applied, the data becomes separable.




(Fig. 5: cubic separable hyperplane)




(Fig. 6: Cubic transformation applied)

SVM kernel functions are used to transform data points into a higher-dimensional space. Depending on the characteristics of the data, different kernel functions may work better, making it easier to find a hyperplane to separate different classes. Multi-dimensional data can be processed through what's called the "kernel trick". This allows for SVMs and other machine-learning algorithms to operate in high-dimensional feature space without explicitly computing the coordinates of data in that space.

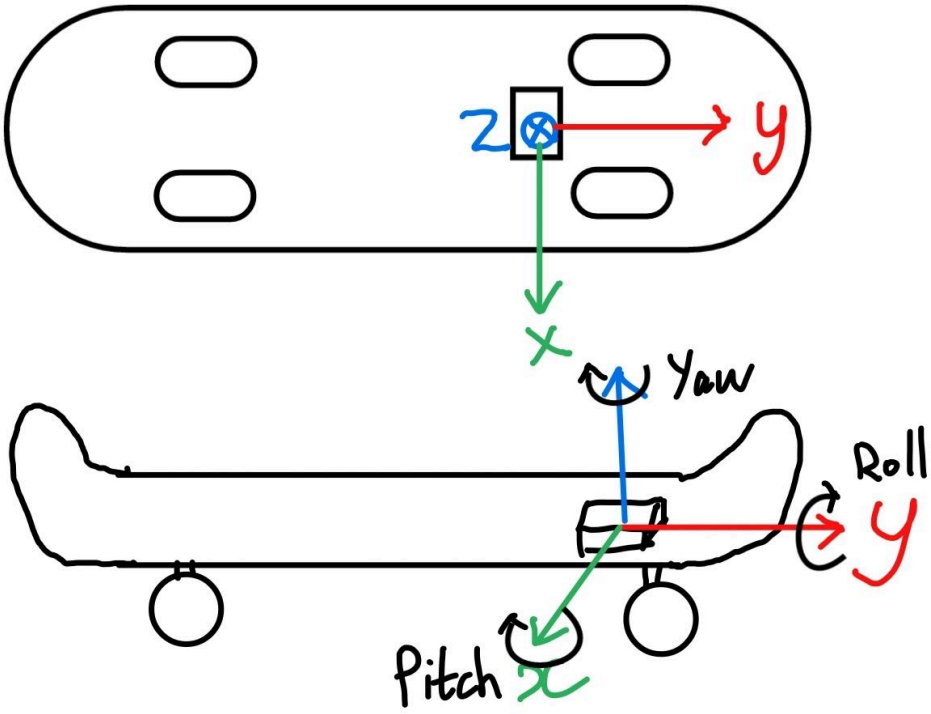
## Methodology

The investigation is conducted in several steps:

1) Data collection of gyroscopic values of 5 different skateboarding tricks, of those including:
    - Ollie
    - Kickflip
    - Pop shuv
    - Frontside 180
    - Backside 180
2) Data is converted into a compatible format and processed through an EMA filter. This is done to smoothen out results and outliers to provide a more accurate comparison.
3) The data will then be used to train the machine learning model, both SVM and KNN
4) Cross validation is used in a 4:1 ratio of training and testing data. 20% of the data will be kept aside from training to be used to test the accuracy and prediction speed of the produced model.
5) Evaluation of results and effectiveness

### Trick selection

5 basic tricks were selected to train the SVM, those including the ollie, frontside 180, backside 180, pop shuvit and kickflip. These tricks were selected on the basis of ease of performance, being fundamental skateboarding tricks (Nicholson, 2021), and their own distinctive characteristics (Table 1).



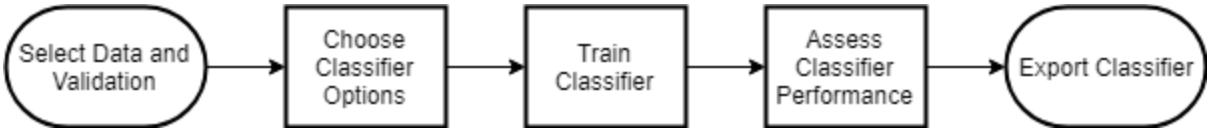

(Fig 7. Axes of data presented on skateboard)

| Name of trick | Change in data |
|---|---|
| Ollie | Pitch axis fluctuation |
| Frontside 180 | Yaw axis -180 degree turn, small pitch axis fluctuation |
| Backside 180 | Yaw axis 180 degree turn, small pitch axis fluctuation |
| Pop shuvit | Yaw axis 180 degree turn, large pitch axis fluctuation |
| Kickflip | Roll axis 360 degree turn, large pitch axis fluctuation |

(Table 1: Trick information)

**Choice of programming language**

The language used to write the data processing algorithm is MATLAB's built in language. MATLAB is a high-level programming language which allows for matrix manipulation. This makes data processing of IMU signals more simple. Apart from that advantage, the main reason for this selection is MATLAB's built in application, Classification Learner. This application allows for automated training of supervised machine learning algorithms and to access and compare different classifiers.




(Fig. 8: MatLab technical flowchart (MatLab, 2023))

**Variables**

In the context of supporting judging in skateboarding contests, two variables are extremely important for success. First, in order to reliably classify skateboarding tricks, accuracy of the model must be high. Second, prediction speed is also extremely important. Skateboarders in contests in real-time often perform tricks in quick succession, and judges also need to quickly keep up with the tricks performed.

Model performance will be compared based on two measures:

- Accuracy (Validation)
- Prediction speed

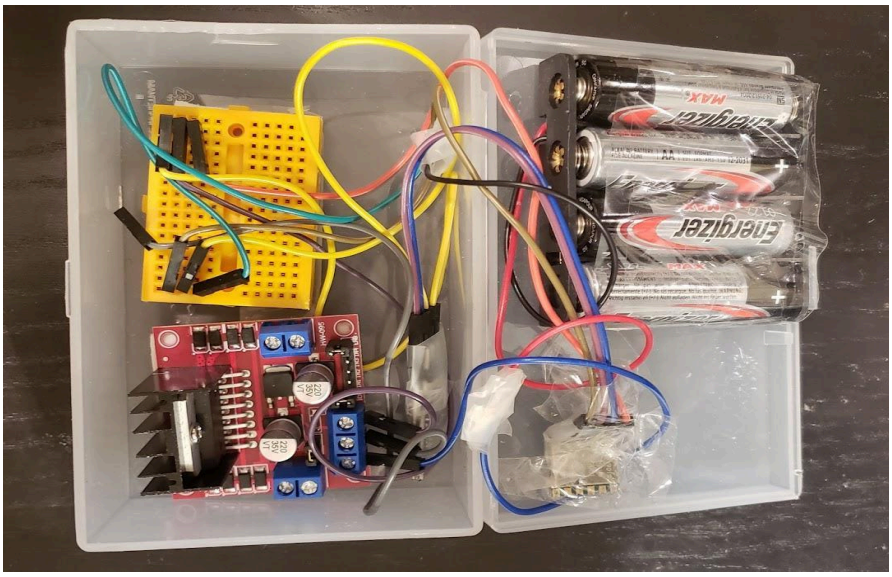
The software will provide both results directly after model validation.

Some factors are also kept constant to ensure a suitable comparison between models. (Table 2)

| Control Variables | Value | Function | Affect |
|---|---|---|---|
| Number of observations per trick | 150 observations for 20 times performed<br><br>3000 total observations | Used as input data to train models | Accuracy, prediction speed |
| Processor GPU and RAM | M2 Macbook air 2022 | Used to train and compute model algorithms | prediction speed |
| Training and testing data | 80% Training, 20% Testing | Data input for model training | Accuracy |
| MatLab version | MatLab r2021b | Application used to train and test model | Accuracy, prediction speed |
| Data collection hardware | MPU9250 gyroscope | Used for data collection | Accuracy |
| EMA weightage | 0.95 | Used to process data for better accuracy | Accuracy |

(Table 2: Control variables)

**Hardware**




(Fig. 9: Prototype)

The initial design was implemented with

- mpu9250 gyroscope
- sampling rate of 1000Hz
- wireless transmitter
- 4 AAA batteries.

General Problems

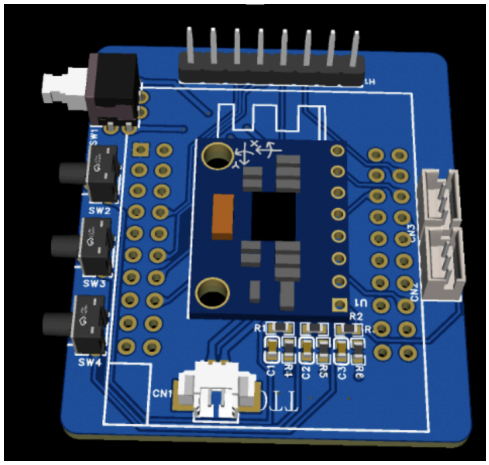
- Bulky and heavy, affecting skater's performance
- Falls off when a trick is being performed
- Gyroscope not properly attached (using tape)
- shakes alot when a trick is being performed

Final design:

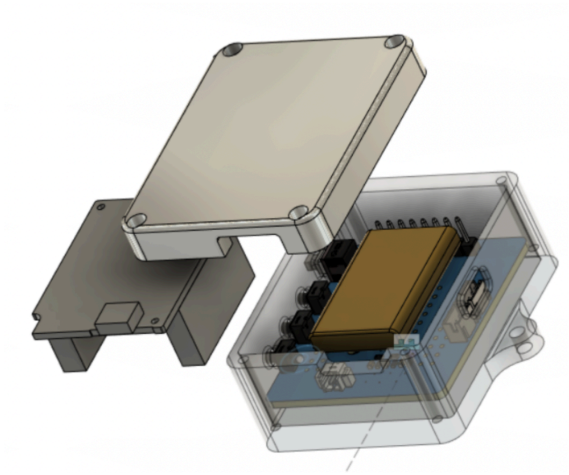
The final innovation is a small contraption built using a 3d printed box containing a mpu9250 gyroscope attached to an esp32 microcontroller with wifi transmission. It is connected to a single small rechargeable 3.7V 230 mAH battery.

It is a significantly smaller contraption as compared to the prototype. Based on the flaws observed from the prototype testing, screw holes were also fitted for the need to mount the device onto the skateboard more securely if needed.

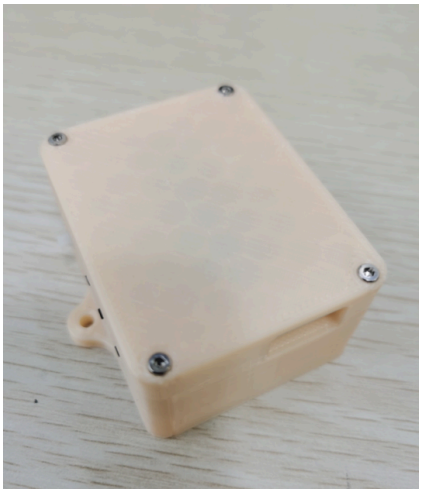
(Fig. 10: Device attached to a skateboard)




(Fig. 11: 3D model of gyroscope)



(Fig. 12: ESP32 microcontroller with mpu9250 attached)



(Fig. 13: Hardware device)

**Data collection**

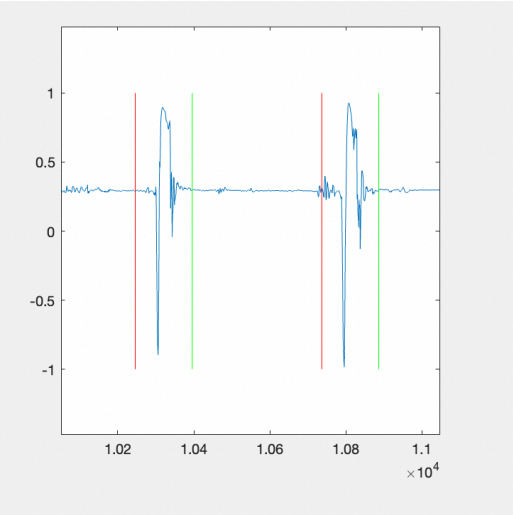
Using the onboard WiFi transmitter, collected data samples from the gyroscope of tricks performed are sent to computer software. The mpu9250 gyroscope is 6 DoF (degrees of freedom), allowing me to have access to the X, Y and Z axis acceleration and also the roll pitch and yaw angles of the gyroscope. Over 250000 observations were collected in total.

(Fig. 14: Skateboarder performing data collection)

**Data processing**

Collected data must be reformatted for MatLab's Classification Learner application, where the models will be trained. MatLab's Classification Learner requires each vector of data to be of equal size. We do this by setting a threshold value on the Z axis acceleration, then taking a boundary of readings whenever the threshold is surpassed. For our case, we will take a total of 150 observations for each trick performed to encapsulate the entirety of the trick as close as possible (Fig. 15). With 5 tricks performed 20 times each, a total of 15000 observations are used for model training. The data is then normalized, scaling the data between -1 and 1 using MatLab's minmax function. This is done to properly scale the data so that the magnitude of acceleration is considered. This is important to consider tricks performed at an incline, where the magnitude of the Z axis acceleration may be less than tricks performed on a flat horizontal

ground. By scaling the data the performance improves so a better comparison of the machine learning algorithms can be compared.



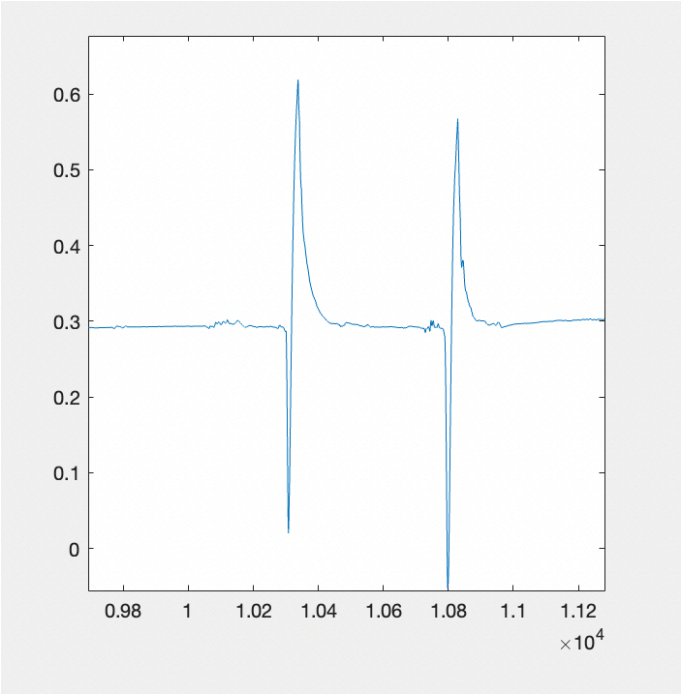

(Fig 15. Two sets of observations of Z axis acceleration on an ollie

Vertical axis - Z axis acceleration

Horizontal axis - observation number )

Analyzing the collected data signals, it is evident that a lot of noise is present (Fig 15). This essentially means that the data samples collected are distorted and corrupted. This can be observed by the small irregular fluctuations of data and the frequency of outlier data comparing the two performances. Noise can result in reduced accuracy for the machine learning model. Hence, data processing methods must be employed to filter out noise, remove outliers and smoothen out the data. For the purposes of this paper, an exponential weighted average (EMA) filter is used. EMA is an infinite impulse response filter, meaning that it filters data values based on all previous inputs. It takes a weighted contribution of previous inputs to output a new value,

with this weightage exponentially decreasing with time. This means that the data signal may never reach its true value, but will converge to it indefinitely. This will help remove the fluctuating spikes in our data illustrated in fig. 15, overall producing a much smoother output filtering out noise (Fig. 16).




(Fig. 16: Same two sets of observations of Z axis acceleration on an ollie post processing

Vertical axis - Z axis acceleration

Horizontal axis - observation number )

The equation for an EMA filter is:

$$y[i] = \alpha \cdot x[i] + (1 - \alpha) \cdot y[i - 1]$$

Where:

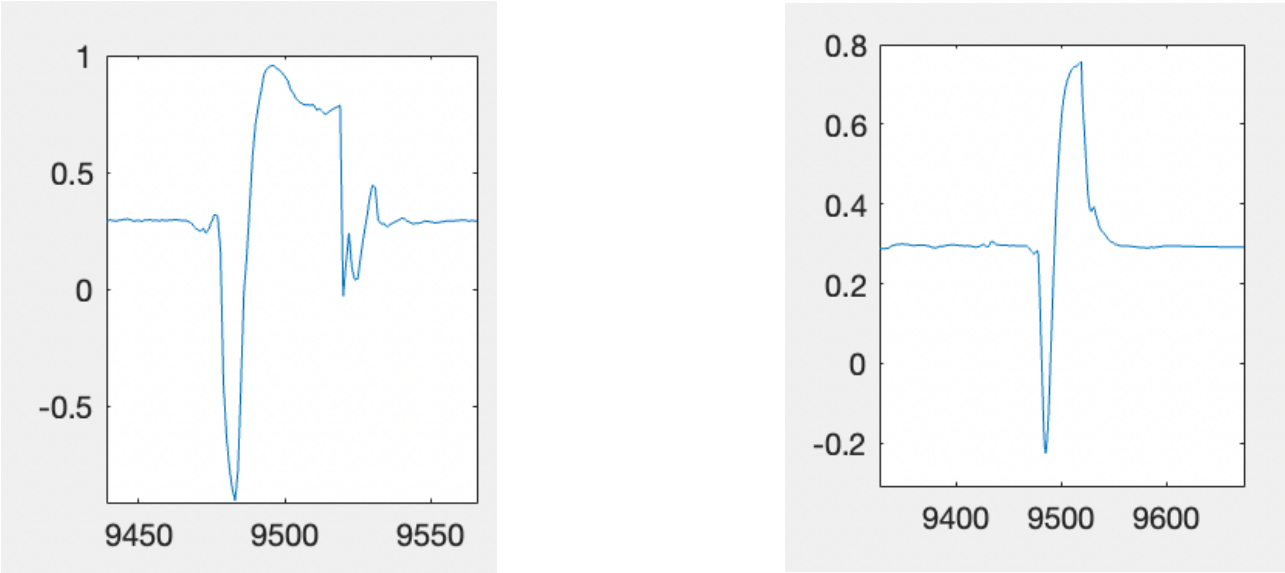
y is the output

x is the input

α is a constant between 0 and 1

[i] is the sample number

The constant α will determine the aggressiveness of the filter (illustrated in fig. 17),




(Fig. 17: α = 0.1 (left), α = 0.9 (right))

α → 0: more aggressive filtering

α = 0: output will be constant and not change at all

α → 1: less aggressive filtering

α → 1: no change between input and output value

(Hunter, 2014)

**Cross validation**

Before training the machine learning models, a portion of the total data must be set aside for validation purposes. This testing data is then used to test the accuracy and prediction speed of the trained models. However, the portion selected as training and testing data will affect the performance of the model. To decide which portion of the data is selected, cross validation is used.

The best data used to train the model is determined by cross validation. In a 4:1 ratio of training to testing data, the selection of the 80% of data used to train and 20% to test will differ the accuracy of the model. To make this selection, k-fold cross validation divides the data into k random sets of observations, and takes one set to test and the rest to train the machine learning model. This process is then repeated with every set to obtain the best performing set of data used to train the model.

**Feature selection**

The features selected for training the different models are simply the IMU signals of the trick at a given time frame. 150 signals are retrieved per trick performance, each used as a separate feature for model training. With 5 tricks each performed 20 times, each feature will contain 100 observations.

It should be noted that we will only be using Z axis acceleration for model training. This is because every trick will have at least a substantial effect on the Z axis acceleration, with the largest fluctuations in value. Hence, Z axis acceleration will be the most optimal axis to use to train the machine learning models for comparison.

**Hyperparameters**

Hyperparameter tuning is an essential part of training machine learning models. Models will be trained with different parameters in order to find the best performing model.

For SVMs, the parameter which will be changed is the kernel function. The kernel function has the highest impact on the model's performance, and other parameters are used to fine tune the model. Hence, to gauge model performance overall, we will keep the other parameters as control variables and only change the kernel function. This will include three models: linear, quadratic and cubic kernel function.

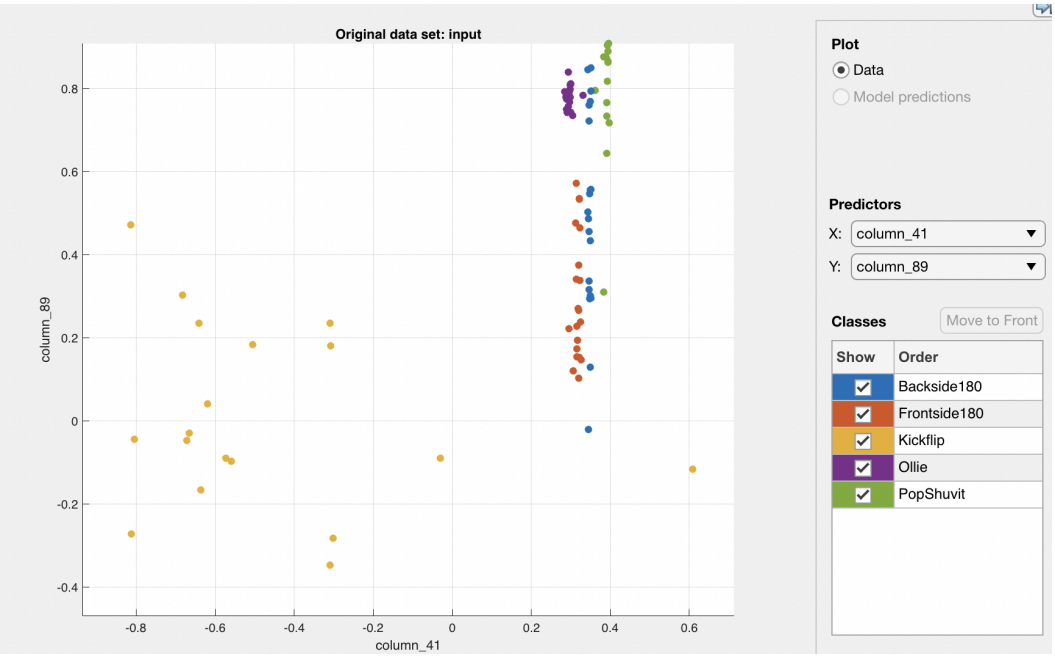
For KNNs, the parameter adjusted will be the value of K, the number of nearest neighbors. Similarly, the value of K holds the most significant impact on the model and other parameters will be kept as control variables. For this paper, K = 1, 3 and 5 is selected as the parameters to be tested.

**Hypothesis**

SVM is said to be better at taking care of outliers than KNN (Varghese, 2020). This is significant as rapid movements of skateboarding may lead to many outliers in data, and even after signal processing, could result in a significant impact in accuracy. In terms of prediction speed, SVM kernel methods require additional computation time when compared to KNN's with a smaller dataset. However, on larger datasets, KNN's will suffer from increased prediction time due to the large amounts of distance calculations it needs to perform. (Anuuz Soni, 2020). Hence, in terms of speed, I hypothesize that SVM will outperform KNNs. With 15000 observations, the dataset size is large. Hence, It could be hypothesized with our data set that KNN's will perform slower than SVM due to the larger time needed to calculate the distances between each point.

Varghese (2020) hypothesized KNN to outperform SVM when the number of training data exceeded the number of features. This hypothesis holds true in Palaniappan (2014) paper, which contained a larger set of training data than features, being 68 recordings and 13 features. Palaniappan's experiment resulted in a 100% highest accuracy for KNN but only a 96.13% highest accuracy for SVMs. Similarly, Bouteldja's (2020) comparison between SVM and KNN also contained a larger data set of training data than features. Again, KNN proved superior, performing with the best accuracy at 77.28% and SVM at a lower 75.14%.

However, I believe Varghese's hypothesis to not hold true for my case. Let's consider two random observations for reference.

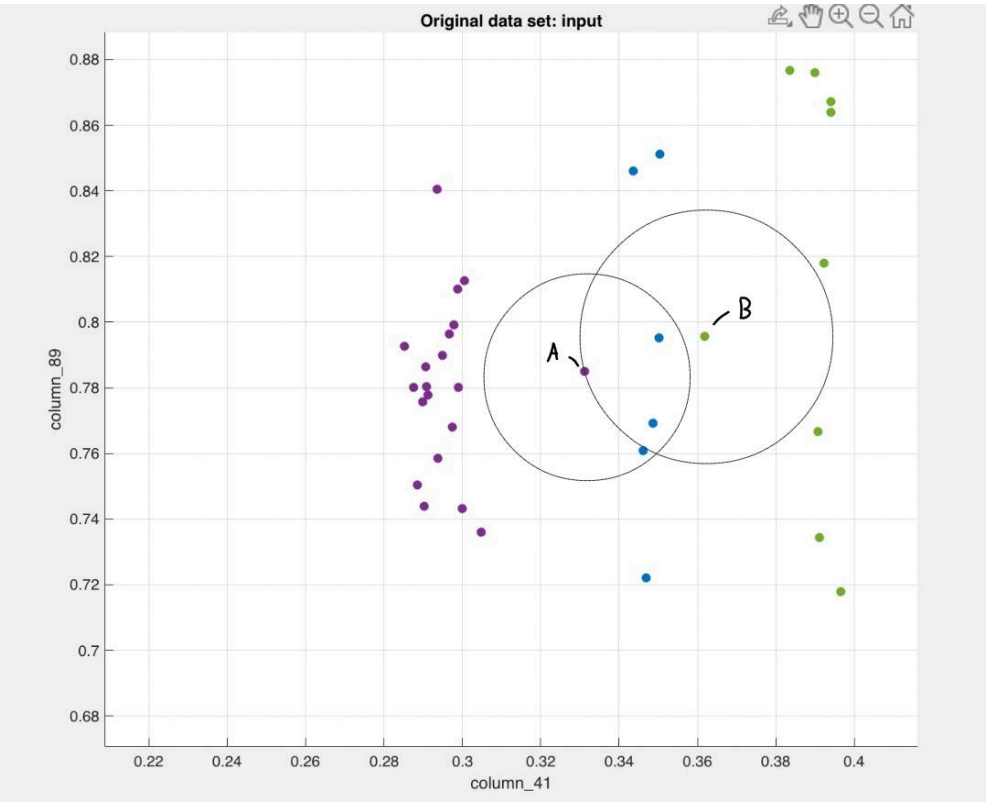



(Fig. 18: two randomly selected observations of data points

Vertical axis - 89th Z axis acceleration value

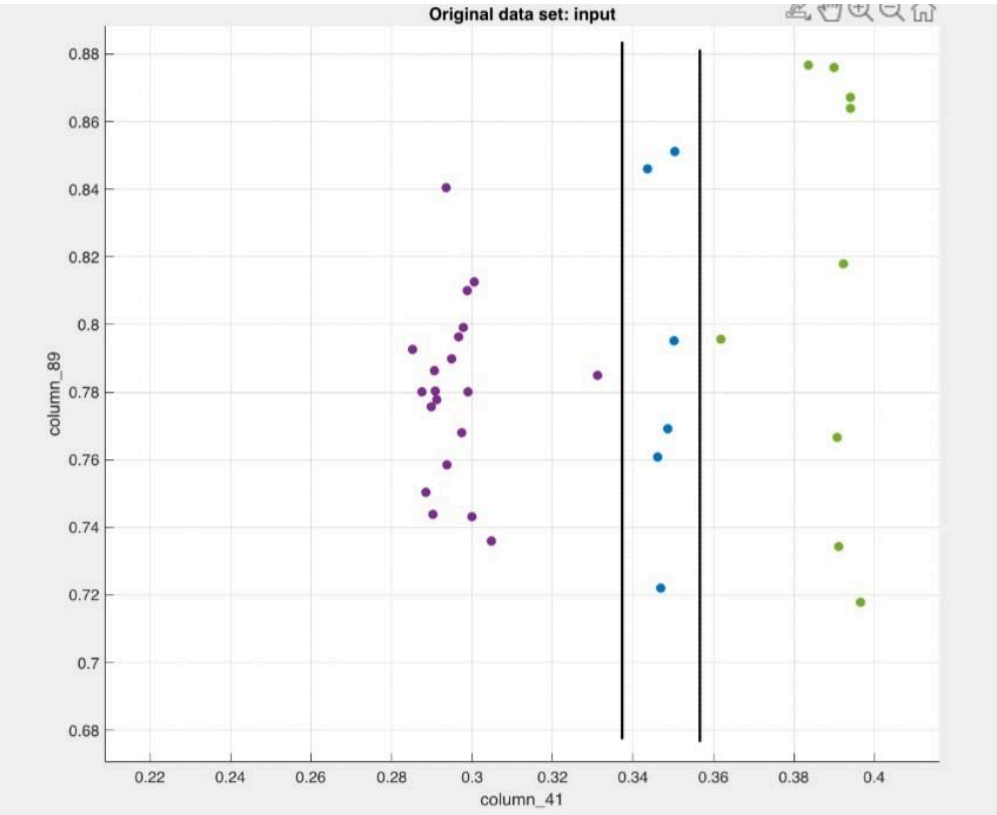
Horizontal axis - 41st Z axis acceleration value)

The above figures depict two, randomly picked, observations of Z axis acceleration with the vertical axis being observation 89 and horizontal axis being observation 41. A pattern can be observed here, with much of the values being clustered together in straight lines. For this case, points A and B (Fig. 19) are example points which result in misclassification for K = 3, with A being predicted as blue instead of purple and B predicted as blue instead of green.



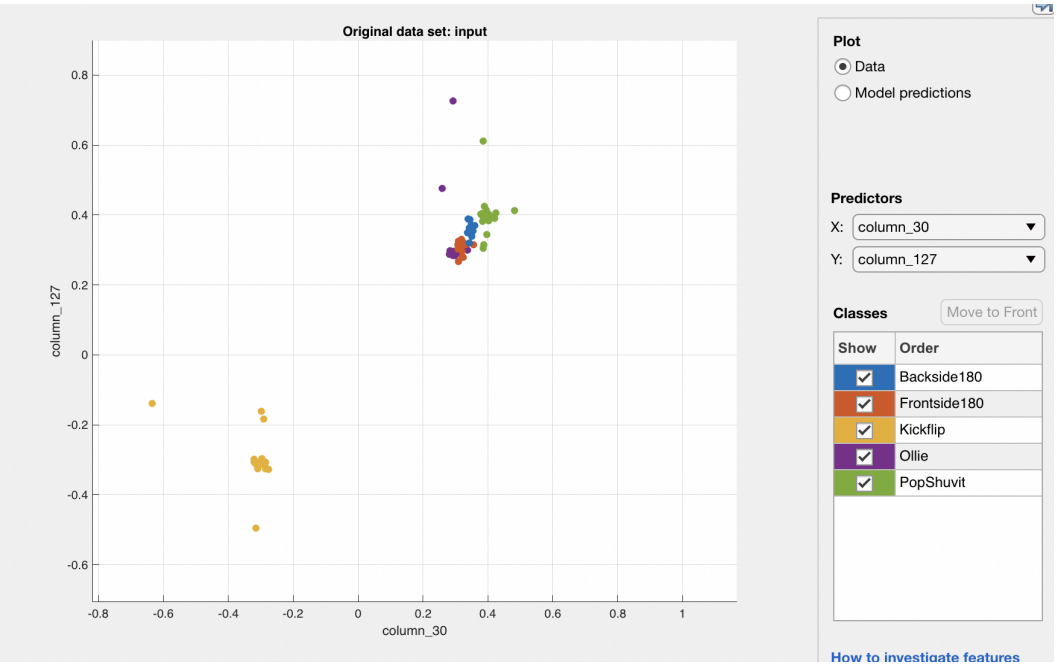

(Fig. 19: KNN data points)

An SVM will produce a decision boundary for separating values. With these features, a linear SVM will probably work the best. A potential linear decision boundary is illustrated in fig. 20. Data on the left or right of the decision boundary will be classified accordingly. The linear hyperplane illustrates zero misclassification, with every observation being correctly classified. Hence, in this case specifically, SVMs will outperform KNN in terms of accuracy.




(Fig. 20: Potential SVM linear hyperplane)

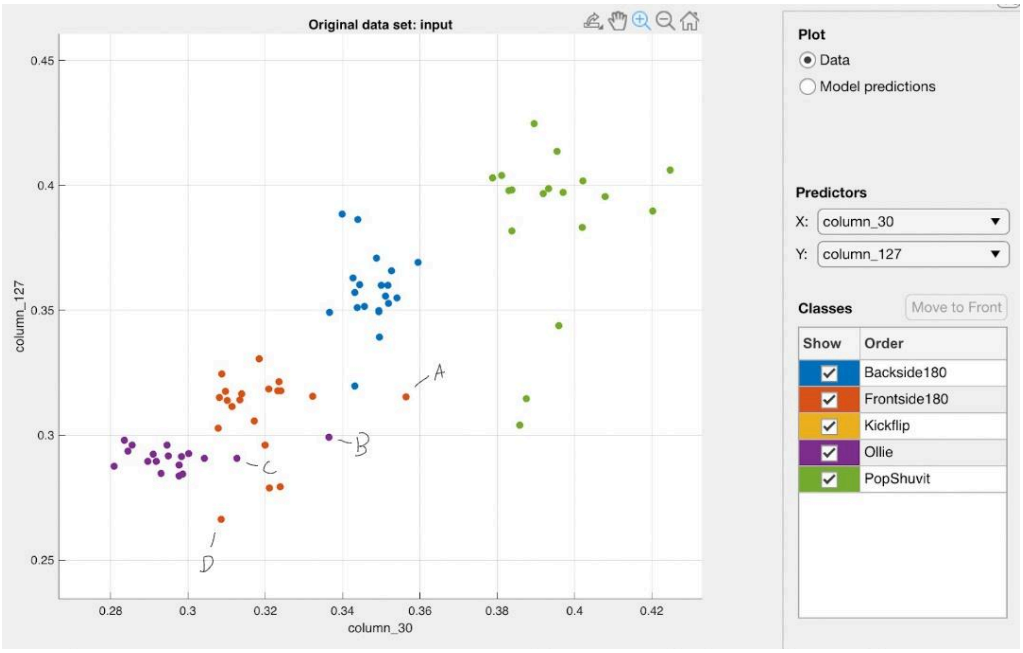
Let us select another case, observation 30 and 127, selected randomly for reference. Upon first glance, the observations are grouped together in radical clusters, and thus KNNs will perform better due to its closest neighbors being calculated radially.



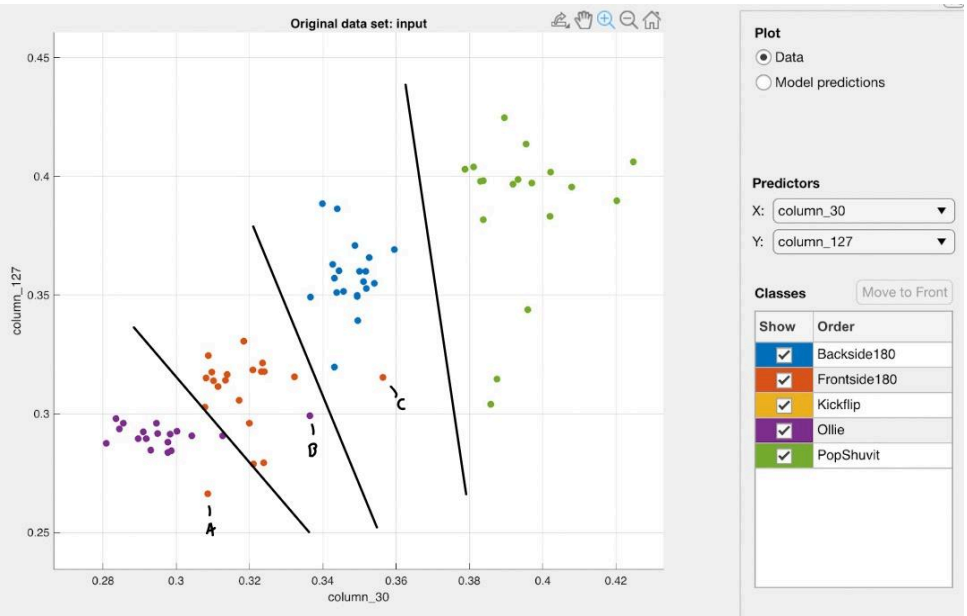

(Fig. 21: two randomly selected observations of data points

Vertical axis - 30th Z axis acceleration value

Horizontal axis - 127th Z axis acceleration value)




(Fig. 22: potential points of misclassification by KNN)




(Fig. 23: Potential linear hyperplane)

Upon further inspection, even a linear hyperplane will classify the data points with a higher accuracy. With a potential hyperplane drawn, only 3 points are potentially misclassified, A B and C (Fig. 23). However, KNN contains more than 3 points potentially misclassified, A B C and D (Fig. 22).

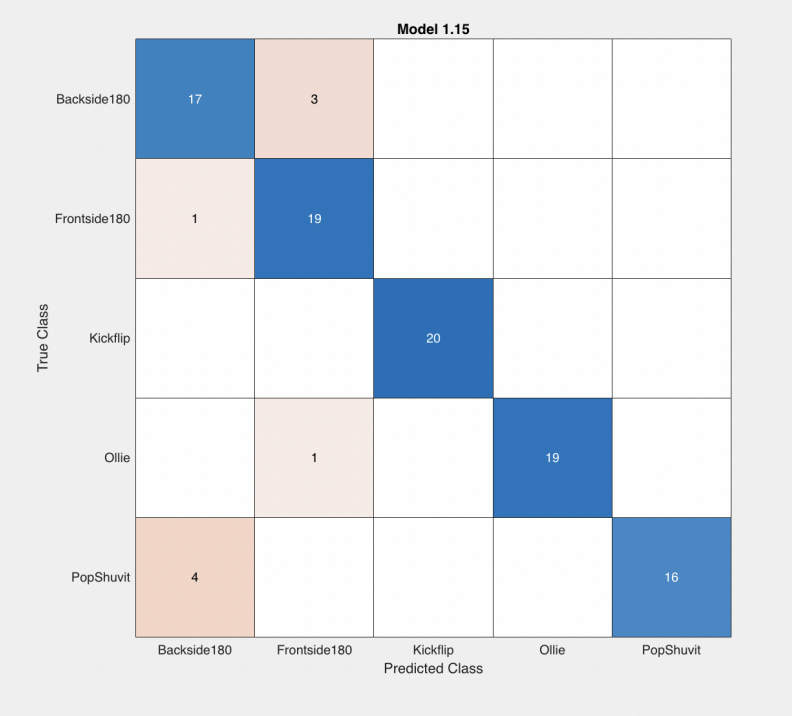
In both cases selected, SVM tends to outperform KNN in terms of accuracy. Hence, I believe Varghese's (Varghese, 2020) hypothesis to be false for my case. My analysis suggests that SVM will out perform KNN despite having more training data than labels, going against Varghese's hypothesis, and the performance evaluation of these two models may have nothing to do with the number of classes or labels at all. Instead, I believe that accuracy of models is more significantly impacted by the nature of training data, and for my case, SVM will outperform KNN.

# Results

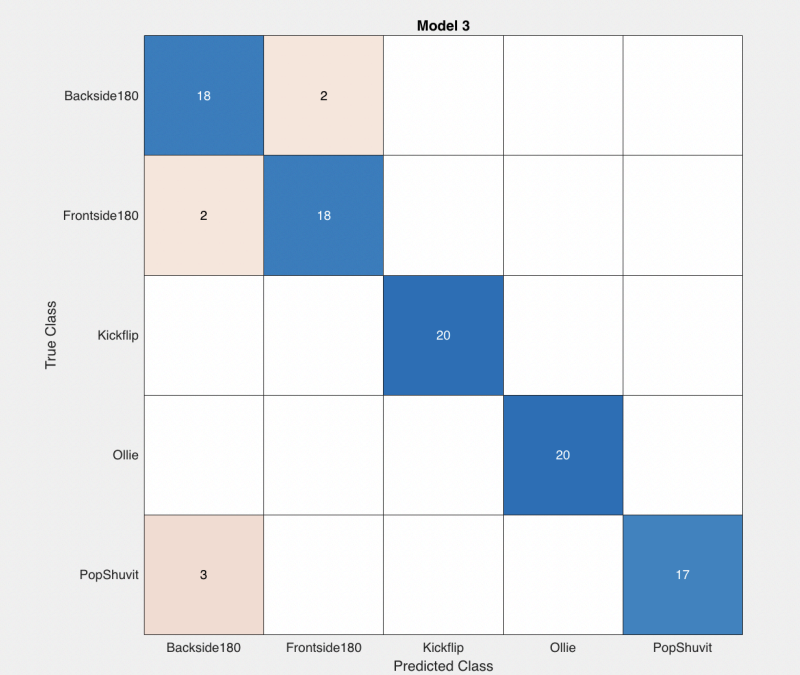
## Performance evaluation

The accuracy of outputs can be visualized on a confusion matrix. It is one of the simplest and straightforward measures of performance evaluation (Flach, 2019). The confusion matrix is read corresponding to its two axes, the true class and predicted class. True class represents the category which the data belongs to, and predicted class is the category the model classifies the data to using the testing data provided. The accuracy of the model is evaluated by taking the percentage of correct observations. Figures 24 to 29 depict the confusion matrix of Z axis acceleration data classification.

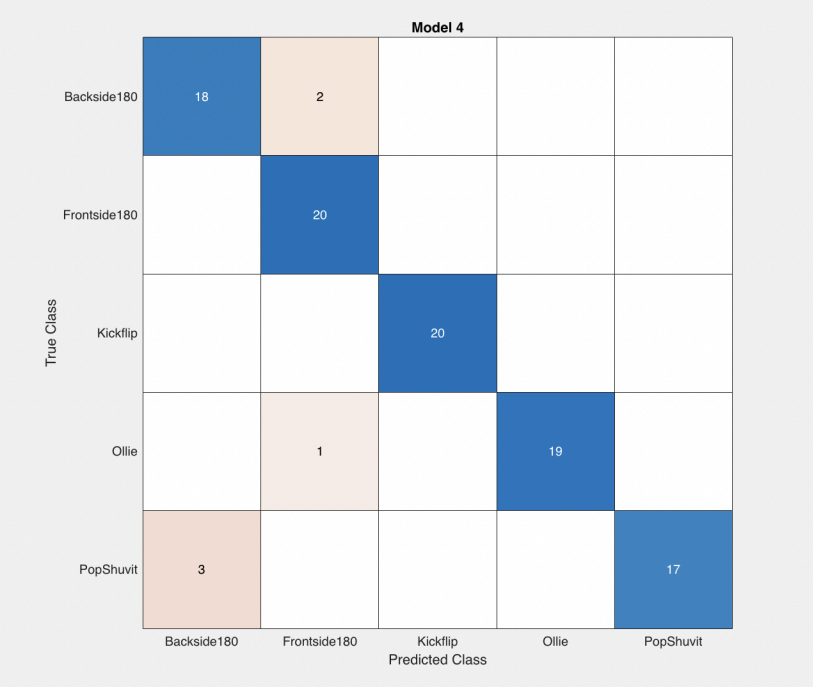
Confusion Matrix of KNN:




(Fig. 24: Confusion matrix KNN (K = 1))




(Fig. 25: Confusion matrix KNN (K = 3))




(Fig. 26: Confusion matrix KNN (K = 5))

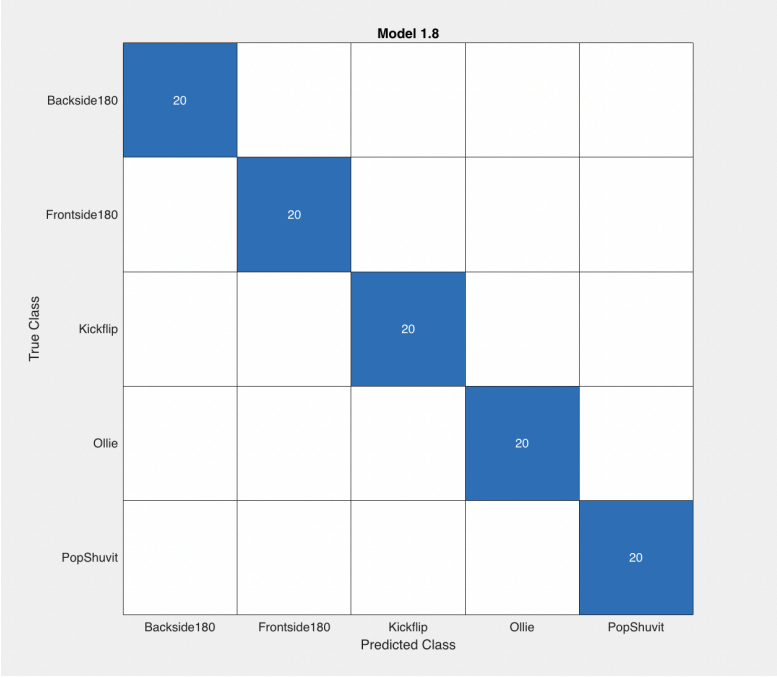
| KNN | K = 1 | K = 3 | K = 5 |
|---|---|---|---|
| Accuracy (Validation) | 96.0% | 93.0% | 94.0% |
| Prediction speed | ~1700 obs/sec | ~2300 obs/sec | ~2400 obs/sec |

(Table 3: Performance of KNN)

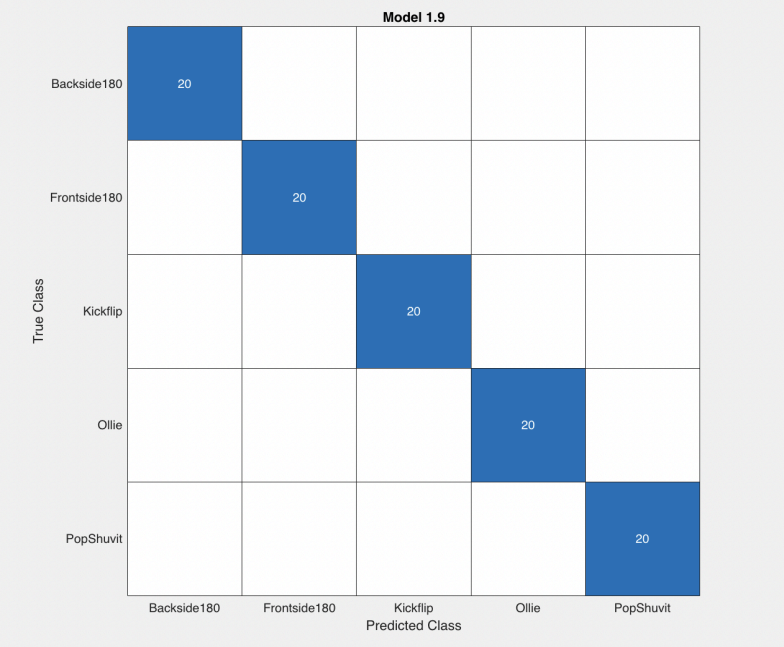
Discussion:

All three KNN models have above a 90% success rate, with the highest accuracy being parameter K = 1 with 96.0%. However, it is the slowest performing model, only calculating ~1700 observations per second. The fastest performing model is where K = 5, with a lower 94.0% accuracy but ~2400 observations per second.

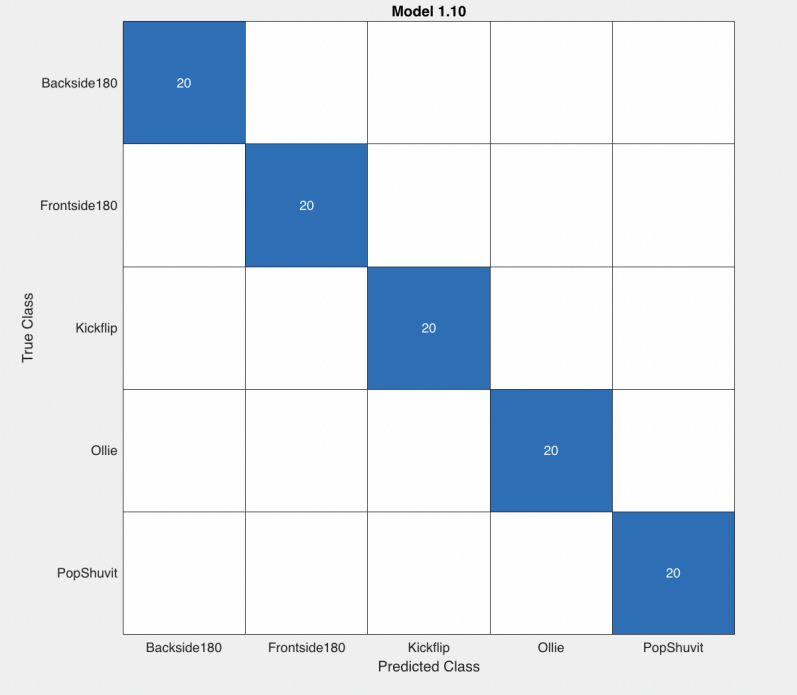
Confusion Matrix of SVM:



(Fig 27. Confusion matrix linear SVM)




(Fig 28. Confusion matrix quadratic SVM)




(Fig 29. Confusion matrix cubic SVM)

| | Linear SVM | Quadratic SVM | Cubic SVM |
|---|---|---|---|
| Accuracy (Validation) | 100.0% | 100.0% | 100.0% |
| Prediction speed | ~3200 obs/sec | ~3000 obs/sec | ~3100 obs/sec |

(Table 4: performance of SVM)

Surprisingly, all three models achieved a perfect accuracy of 100.0% on the Z axis angular acceleration data. The best performing model is a linear SVM, with the highest prediction speed of ~3200 observations per second. To get a rough sense on how fast that is, every trick contains 150 observations and the model will classify the trick in only 0.05 seconds.

**Conclusion**

The incorporation of machine learning for the classification of skateboarding tricks proves to be successful. With only using the Z axis angular acceleration, the most accurate, and also the fastest, model has a perfect accuracy in the classification of tricks.

Comparing the two algorithms, SVM is superior to KNN in terms of both accuracy and speed. The highest accuracy for SVM is 100% and a speed of ~3200 obs/sec while KNN only has a highest accuracy of 96% and highest speed of ~2400 obs/sec. This leaves SVMs the obvious choice for classification of skateboarding tricks.

I originally hypothesized that SVM will outperform KNN in terms of prediction speed due to the large amount of calculations needed in a large data set. This holds true as every SVM kernel contains a higher prediction speed compared to that of KNN's. I additionally hypothesized that SVMs will outperform KNN in terms of accuracy as well, and was also proven to be correct.

Furthermore, Varghese's hypothesis that SVM outperforms KNN when the number of features exceeds the number of training data (Varghese 2020) proves to be incorrect. Despite the number of features (150) exceeding the number of training data (15000), SVM still obtained a higher accuracy than KNN. Thus, I theorize that performance between SVM and KNN is more based upon the shape of the data itself, whether there are many outliers or whether some classes overlap each other.

**Evaluation of method and extension & limitations**

For the purposes of comparison, analysis focused solely on Z axis acceleration while the IMU consisted of 5 other unused features: Y axis acceleration, X axis acceleration, roll, pitch and yaw. This is used because every skateboarding trick has a considerable effect on the Z axis acceleration. Hence, it is most suitable to be used for model training. Getting the highest accuracy, however, can be done by combining all 6 of the IMU signals using sensor fusion algorithms (Udacity Team, 2020) and will result in a much higher accuracy for both KNN and SVM models. Nevertheless, this may result in the accuracy of classification being too high, making it hard to draw conclusions between the two machine learning models.

**Limitations:**

It is important to note that accuracy is calculated using the only testing data provided. With a total of 100 tricks performed, only 20% of it is used as testing data, and so the model is only tested 20 times. This means the accuracy achieved cannot be concluded with absolute certainty, and more testing data will be needed to provide a more accurate result.

Secondly, all data collection, training and testing data, is performed by myself. Model performance will fit my own skateboarding movements more than others. To facilitate skateboarding contests and trick detection, data collection must be performed by more than one skater to avoid overfitting.

In the future, other hyperparameters, such as regularization parameter and kernel coefficient for SVMs, can be investigated to provide a better comparison between both machine learning algorithms. Additionally, classification of skateboarding tricks can be extended into other machine learning algorithms such as neural networks and decision trees.

## References


Bouteldja, S. (2020, January). A comparative analysis of SVM, K-NN, and decision trees for high resolution satellite image scene classification. 10.1117/12.2557563

Flach, P. (2019, July 17). Performance Evaluation in Machine Learning: The Good, the Bad, the Ugly, and the Way Forward. https://doi.org/10.1609/aaai.v33i01.33019808

Hunter, G. (2022, May 25). *Exponential Moving Average (EMA) Filters*. mbedded.ninja. Retrieved October 4, 2023, from https://blog.mbedded.ninja/programming/signal-processing/digital-filters/exponential-moving-average-ema-filter/

Krishnamurthy, B. (2022, October 28). *ReLU Activation Function Explained*. Built In. Retrieved October 4, 2023, from https://builtin.com/machine-learning/relu-activation-function

MatLab. (n.d.). *Classification Learner App - MATLAB & Simulink*. MathWorks. Retrieved October 4, 2023, from https://www.mathworks.com/help/stats/classification-learner-app.html

Nicholson, J. (n.d.). *5 Beginner Skateboard Tricks That Are Super Easy To Learn*. Skate The States. Retrieved October 4, 2023, from https://skatethestates.com/skateboard-tricks-for-beginners/

Palaniappan, R. (2014, June 27). A comparative study of the SVM and K-nn machine learning algorithms for the diagnosis of respiratory pathologies using pulmonary acoustic signals. 10.1186/1471-2105-15-223

Pappalardo, A. (2014, April 7). *How Do You Judge A Skateboard Contest?* Jenkem. https://www.jenkemmag.com/home/2014/04/07/how-do-you-judge-a-skateboard-contest

Saini, A. (2021, October 12). *Guide on Support Vector Machine (SVM) Algorithm*. Analytics Vidhya. Retrieved October 4, 2023, from https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/

Soni, A. (2020, July 3). *Advantages And Disadvantages of KNN | by Anuuz Soni*. Medium. Retrieved October 4, 2023, from https://medium.com/@anuuz.soni/advantages-and-disadvantages-of-knn-ee06599b9336

Udacity Team. (2020, August 25). *Sensor Fusion Algorithms Explained*. Udacity. Retrieved October 4, 2023, from https://www.udacity.com/blog/2020/08/sensor-fusion-algorithms-explained.html

Varghese, D. (2018, December 6). *Comparative Study on Classic Machine learning Algorithms | by Danny Varghese*. Towards Data Science. Retrieved October 4, 2023, from https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

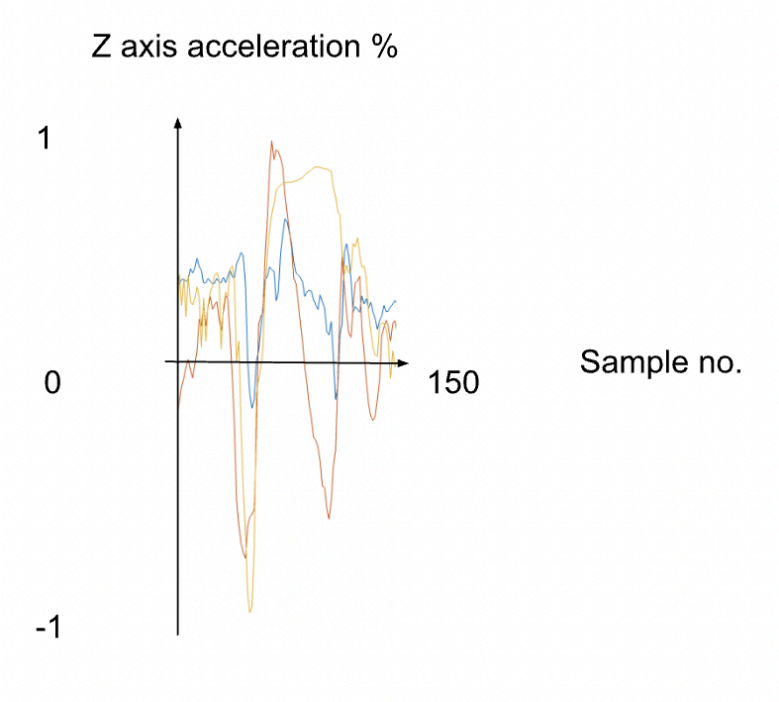
Wilimitis, D. (2018, December 12). *The Kernel Trick in Support Vector Classification | by Drew Wilimitis*. Towards Data Science. Retrieved October 4, 2023, from https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

## Appendix

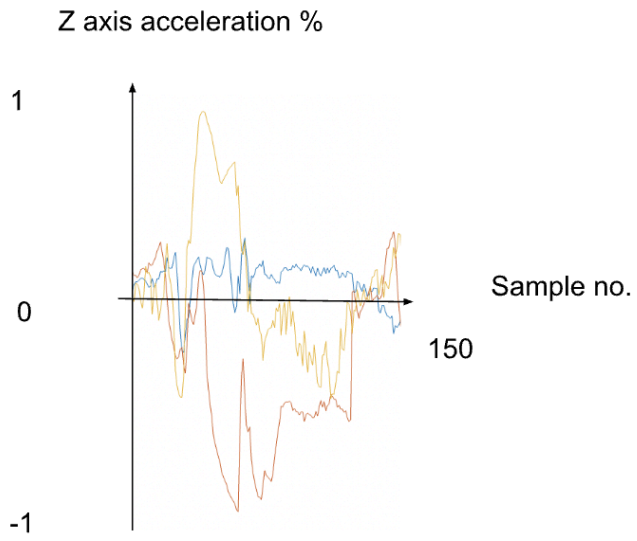
### Appendix A - Data of a instance of trick

The following contains a single instance of a trick being performed. Total data is too large to be displayed, so only one performance is shown.

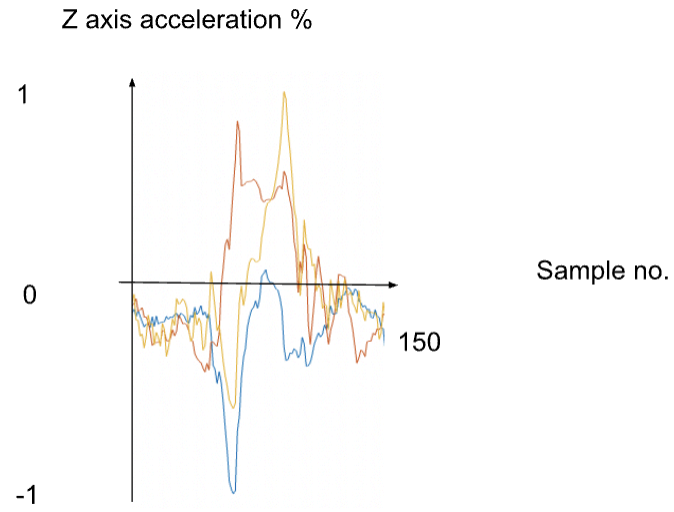
Ollie - https://docs.google.com/spreadsheets/d/15xpiXoEB-YMA8Ygsi2SYR4b8pS-e6i2LBaRCXOHON5w/edit?usp=sharing



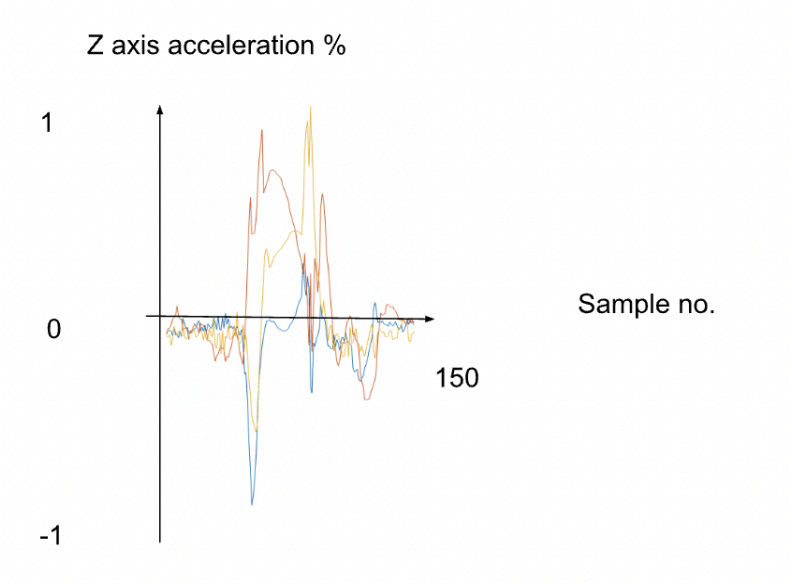
Frontside 180 - https://docs.google.com/spreadsheets/d/1mAQFfieMJW8SlHpaIx2bLNWI8ZibhKG8VFbAa9fSZeQ/edit?usp=sharing



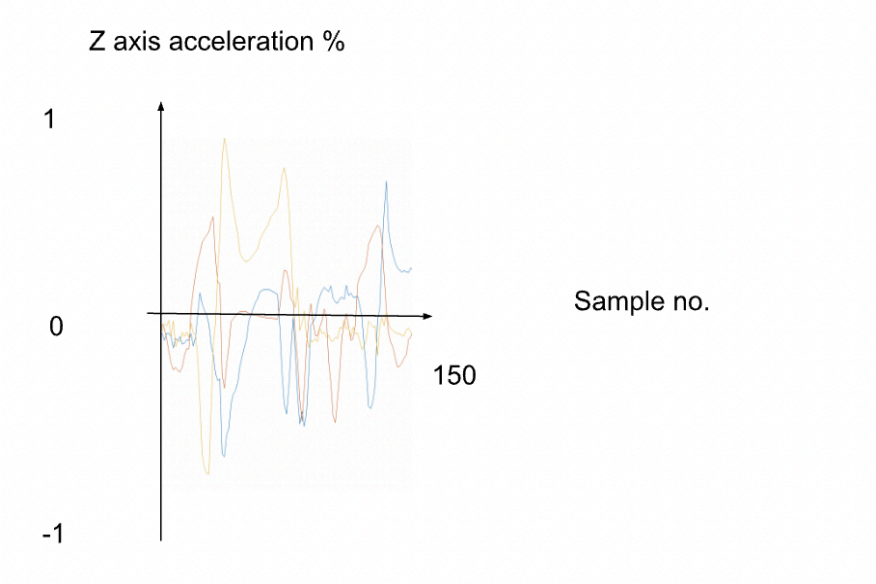

Backside 180 - https://docs.google.com/spreadsheets/d/1A85uMwd2We2f2g5wGmQiAIM-B8_V4JsfpqfSzEflmH4/edit?usp=sharing



Pop shuvit - https://docs.google.com/spreadsheets/d/1X_nxq9og-ahLzXiLbfXeCeSBDPuJHlPR1R_lc_mqiyY/edit?usp=sharing




Kickflip - https://docs.google.com/spreadsheets/d/1Im7QFJS1aNVuyOpc720lebojmgKJzqEtQ9BiNoTFC6E/edit?usp=sharing



**Appendix B - MatLab code for data processing**

```
%Read in collected data


clear

clc

Ollie=readmatrix("Ollie.csv");

Frontside180=readmatrix("frontside180.csv");

Backside180=readmatrix("backside180.csv");

PopShuvit=readmatrix("popshuvit.csv");

Kickflip=readmatrix("kickflip.csv");

% save into one matrix

save Final

%%

%load matrix

load('Final.mat')

%obtain labels

TrainData=[];

for Act=1:1:5

        switch Act

        case 1

                A=Ollie(:,10);TrainData=[TrainData;DataProcessing(A,"Ollie")];

        case 2
```

```
            A=Frontside180(:,10);TrainData=[TrainData;DataProcessing(A,"Frontside180")];
        case 3
            A=Backside180(:,10);TrainData=[TrainData;DataProcessing(A,"Backside180")];
        case 4
            A=PopShuvit(:,10);TrainData=[TrainData;DataProcessing(A,"PopShuvit")];
        case 5
            A=Kickflip(:,10);TrainData=[TrainData;DataProcessing(A,"Kickflip")];
        otherwise
            disp('other value')
        end
end
input=TrainData(:,1:end-1);
output=TrainData(:,end);
input = str2double(input)
clc
%%
%%
%Data processing and segmentation function
%Din = Segmented action data frames and labels
function Din=DataProcessing(A,act)
%normalization
        [A, ~]=mapminmax(A');
```

```
A=A';
%Start EMA filter
beta=0.95;
AA=zeros(length(A),1);
AA(1)=A(1);
for i=1:length(A)
        if i==1
                AA(i)=A(i);
        else
                AA(i)=beta*AA(i-1)+(1-beta)*A(i);
        end
end
B=(A-AA);
d=25;
E=zeros(length(A)-d,1);
for i=d+1:length(A)
        E(i-d)=sum(B(i-d:i));
end
beta2=0.96;
EE=zeros(length(E),1);
EE(1)=E(1);
for i=1:length(E)
```

```
        if i==1
                EE(i)=E(i);
        else
                EE(i)=beta2*EE(i-1)+(1-beta2)*E(i);
        End
End
Gate=1.5; %threshold value
Index=[];%Stores value before threshold
Value=[]; %Stores value after threshold
Midval=[]; % middle value
Flag=false;
for i=1:length(EE)
        if EE(i)>=Gate && Flag==false
                Flag=true;
                index1=i;
                val1=EE(i);
        elseif EE(i)<Gate && Flag==true
                Flag=false;
                index2=i;
                val2=EE(i);
                Index=[Index;index1,index2];
                Value=[Value;val1,val2];
```

```
                Midval=[Midval,floor((index1+index2)/2)];
            end
        end
    end
```