

PROJECT REPORT

ON

CRICKET SHOT PREDICTOR

BIG DATA APPLICATION DEVELOPMENT

Submitted By

DHAIRYA VORA (16012121005)

JASH GANDHI (16012121008)

JAL HARSORA (16012121010)

PREET MERCHANT (16012121014)



NOVEMBER 2019

U.V. PATEL COLLEGE OF ENGINEERING
GANPAT UNIVERSITY



CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that students of B.Tech COMPUTER SCIENCE ENGINEERING (Semester VII) have completed their one full semester on project work titled “**CRICKET SHOT PREDICTOR**” satisfactorily in subject “**Big Data Application Development**” of **Computer Science & Engineering, Ganpat University** in the year 2019.

DHAIRYA VORA (16012121005)

JASH GANDHI (16012121008)

JAL HARSORE (16012121010)

PREET MERCHANT (16012121014)

Internal Guide

Prof. Aniket Patel

Date:

HOD

CSE Department

Prof. Dharmesh Darji

Date:

ACKNOWLEDGEMENT

We have been constantly putting our efforts to make this project possible. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We are highly indebted to **Prof. Aniket Patel** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

We would like to express our gratitude to members of **Institute of Computer Technology (ICT)** for their kind co-operation and encouragement which help us in completion of this project.

Our thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

ABSTRACT

Cricket shot predictor is a modern tool which can help batsmen in game of cricket. It is a tool, which can be put to use for predicting the shot played by a batsman. It can be used to gain insights of the shot played by actually playing with all the predicting equipment. We have performed the cricket shot prediction on actual dataset which was collected by playing ball-free shots to predict whether it was a straight drive, cover drive or on drive.

CONTENTS

Page No.

ABSTRACT

CERTIFICATE

ACKNOWLEDGEMENT

1. INTRODUCTION	7
1.1 Cricket Shot Predictor	7
1.2 Project Definition	7
2. PROJECT SCOPE	8
3. SOFTWARE REQUIREMENTS & HARDWARE REQUIREMENTS	9
3.1 Software Requirements	9
3.2 Hardware Requirements	9
4. PROJECT PLAN	10
5. IMPLEMENTATION DETAILS	11
5.1. Algorithms	11
5.2. Code Snapshot	12
6. TESTING	16
7. USER MANUAL	17
8. CONCLUSION AND FUTURE WORK	21

REFERENCES

List of figures:

Figure 1: Fetching Data using MATLAB	12
Figure 2: Importing python libraries	12
Figure 3: Naïve Bayes	13

Figure 4: Loading csv file, converting string column to float and converting string column to integer.	13
Figure 5: Summarizing data	14
Figure 6: Calculating probabilities	15
Figure 7: Text-to-speech	15
Figure 8: Straight drive test data	16
Figure 9: Test case output for straight drive	16
Figure 10: Cover drive test data	16
Figure 11: Test case output for straight drive	16
Figure 12: HyperIMU sensor selection	17
Figure 13: HyperIMU sensor axes	17
Figure 14: HyperIMU IP configuration	17
Figure 15: Server IP Address added in MATLAB	18
Figure 16: Streaming sensor readings	18
Figure 17: Readings received in MATLAB	19
Figure 18: Test data stored as csv file	19
Figure 19: Final result	20

List of tables:

Table 1: Project Plan	10
-----------------------	----

INTRODUCTION

1.1 Cricket Shot Predictor

Cricket batting is a dynamic interceptive task which involves the batter perceiving relative motion of the bowled ball and formulating a response for a desired goal. High degree of accuracy in spatial and temporal motion of the bat and batter before and at the instant of ball contact is critical for achieving the goal. The motion of the bat created by the batter is termed as bat swing.

We checked the possibility of estimating the angular velocity and position of the bat and bat twist during cricket bat swing at any instant using tri-axial accelerometer and tri-axial gyroscope. We worked on the cover drive, straight drive and on drive ball-free (without ball). Large spike in different axes acceleration profiles from accelerometer data were evident in swing duration. Various key features of a drive such as shot power, shot direction and angle of elevation of the bat during swing could be extracted using inertial sensors.

We have checked the possibility of using the mobile phone sensors like the accelerometer and gyroscope sensors which come in-built in order to fetch data. The sensors data is streamed live using an application directly into our computer and can be examined live. The accelerometer and gyroscope can respond to every 50 millisecond change in its motion and they can be attached directly to the back of the bat using sticky tape or a rubber band.

In this study, a mobile phone was attached to the bat and data were collected during the different shots. The current work was conducted with a set of ball-free shots by a batsman, at different speeds. Later prediction procedure was performed and output was decided.

1.2 Project Definition

In our project titled cricket shot predictor, we have collected data using mobiles phone sensors such as accelerometer and gyroscope. Firstly, we created a training dataset for every shot to be predicted and classified it. The data of a shot played was stored in an excel file and mean values of each axes of accelerometer and gyroscope were calculated. Later, the mean values of all shots were classified as a particular shot.

Next, we collect our test data which will be used in a confirmatory way, typically to verify that a given set of input to a given function produces some expected result. This test data is applied to a Naïve Bayes program. This program produces result based on probabilities.

Hence, cricket shot predictor is helpful for predicting shot played when a batsman is learning to master his skills in cricket. This can aid batsmen to understand what mistakes they are committing and correct them.

PROJECT SCOPE

Cricket shot predictor can be deployed on field when batsmen are training. In cricket the skills of a batsman are judged by how good they are at executing different types of shots. And mastering various types of shots is important. And cricket shot predictor is helpful for predicting shot played when a batsman is learning to master his skills in cricket. This can aid batsmen to understand what mistakes they are committing while playing a shot, and confirming whether the shot they play was actually correctly executed by them.

This is something that humans do automatically but computational methodologies have also been developed. In order to analyze and predict the result, machine learning algorithms such as Naïve Bayes can be used. Deep learning, which is under the unsupervised family of machine learning, can also be employed in cricket shot prediction.

SOFTWARE AND HARDWARE REQUIREMENTS

3.1 SOFTWARE REQUIREMENTS

- HyperIMU: Mobile Application for streaming mobile sensor data directly to computer.
- MATLAB: For fetching sensor data through HyperIMU.
- Python 3.6: For applying machine learning algorithm and predicting result.
Python Libraries: pandas, reader, sqrt, exp, pi, pytsx3
- Microsoft Excel: For storing datasets

3.2 HARDWARE REQUIREMENTS

- A standalone computer having 4 GB RAM
- Cricket Bat
- Mobile Phone
- Rubber band to hold mobile phone with bat

PROJECT PLAN

Time	Work Done
July	Research on implementation of project
August	Finalizing project execution plan and tools
September	Implementation of project
October	Testing

Table 1: PROJECT PLAN

IMPLEMENTATION DETAILS

5.1 Algorithms

Preprocessing Part (For Naïve Bayes):

- Step 1: Separate By Class.
- Step 2: Summarize Dataset.
- Step 3: Summarize Data By Class.
- Step 4: Gaussian Probability Density Function.
- Step 5: Class Probabilities.

Naïve Bayes:

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

$$P(\text{class}|\text{data}) = (P(\text{data}|\text{class}) * P(\text{class})) / P(\text{data})$$

5.2 Code Snapshot

```
%CodeStart-----
%Resetting MATLAB environment
instrreset
clear
clc

%Creating UDP object
UDPComIn=udp('192.168.43.26','LocalPort',12345);
set(UDPComIn,'DatagramTerminateMode','off')
%Opening UDP communication
fopen(UDPComIn);
%Reading data
t0 = clock;
headers = {'A','B','C','D','E','F'};
csvwrite('TestData.csv',headers);
] while etime(clock, t0) < 3
    csvdata=fscanf(UDPComIn);
    scandata=textscan(csvdata,'%f %f %f %f %f %f','Delimiter',' ');
    data=[scandata{1},scandata{2},scandata{3},scandata{4},scandata{5},scandata{6}];
    disp(data)

    dlmwrite('TestData.csv',data,'delimiter',' ','-append');
end
%Closing UDP communication
fclose(UDPComIn);
%Deleting UDP communication
delete(UDPComIn)
%CodeEnd-----
```

Figure 1: Fetching Data using MATLAB

The code in figure 1 is written in MATLAB, it is used to fetch sensor readings from mobile phone's accelerometer and gyroscope sensors.

In the code above, the IP Address and port number of the host PC on which MATLAB is running. Then, data is stored in a comma-separated file named TestData.csv.

```
from csv import reader
from math import sqrt
from math import exp
from math import pi
import pandas as pd
import pyttsx3
```

Figure 2: Importing python libraries

The code in figure 2 shows the libraries needed to be imported for Naïve Bayes classification, storing and fetching data from csv files for calculations (pandas, csv) and python test-to-speech (pyttsx3).

```

# Make a prediction with Naive Bayes on Train Dataset
filename = 'train.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])-1):
    str_column_to_float(dataset, i)
# convert class column to integers
ans = str_column_to_int(dataset, len(dataset[0])-1)
# fit model
model = summarize_by_class(dataset)
# define a new record
df = pd.read_csv (r'D:\COLLEGE\sem-VII\CGC\Project\TestData.csv')

# block 1 - simple stats
mean1 = df['A'].mean()
mean2 = df['B'].mean()
mean3 = df['C'].mean()
mean4 = df['D'].mean()
mean5 = df['E'].mean()
mean6 = df['F'].mean()
row = [mean1,mean2,mean3,mean4,mean5,mean6]

label = predict(model, row)
print('Data=%s, Predicted: %s' % (row, label))

```

Figure 3: Naïve Bayes

```

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
        print('[%s] => %d' % (value, i))
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

```

Figure 4: Loading csv file, converting string column to float and converting string column to integer

In figure 4, we load the dataset and convert the loaded data to numbers that we can use with the mean and standard deviation calculations. For this we will use the helper function `load_csv()` to load the file, `str_column_to_float()` to convert string numbers to floats and `str_column_to_int()` to convert the class column to integer values.

```
# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries

# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
    return sqrt(variance)
```

Figure 5: Summarizing data

Above, in figure 5, we have developed the `separate_by_class()` function to separate a dataset into rows by class. And we have developed `summarize_dataset()` function to calculate summary statistics for each column. We can put all of this together and summarize the columns in the dataset organized by class values. We have a function named `summarize_by_class()` that implements this operation. The dataset is first split by class, then statistics are calculated on each subset. The results in the form of a list of tuples of statistics are then stored in a dictionary by their class value.

```

# Predict the class for a given row
def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent

```

Figure 6: Calculating probabilities

In figure 6, function named predict() was developed to manage the calculation of the probabilities of a new row belonging to each class and selecting the class with the largest probability value.

```

# initialisation
engine = pyttsx3.init()

xyz = list(ans)
print(xyz[label])
finalvalue = xyz[label]

# predicting voice
engine.say("You played "+finalvalue+" drive.")
engine.runAndWait()

```

Figure 7: Text-to-speech

Lastly, we implement text-to-speech operation, where the final shot predicted will be spoken out loud as shown in figure 7.

TESTING

Test 1: Straight Drive

Test Data: Data is in the form of means for 3 axes of both sensors. The means are calculated by the program itself from the testing dataset generated. Hence, 6 mean values.

```
row = [0.188157, 3.224899, -2.08299, 1.132986, 0.216311, -0.64134]
```

Figure 8: Straight drive test data



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\COLLEGE\sem-VII\CGC\Project\Final.py =====
[On] => 0
[Cover] => 1
[Straight] => 2
Data=[0.188157, 3.224899, -2.08299, 1.132986, 0.216311, -0.64134], Predicted: 2
Straight drive
>>> |
```

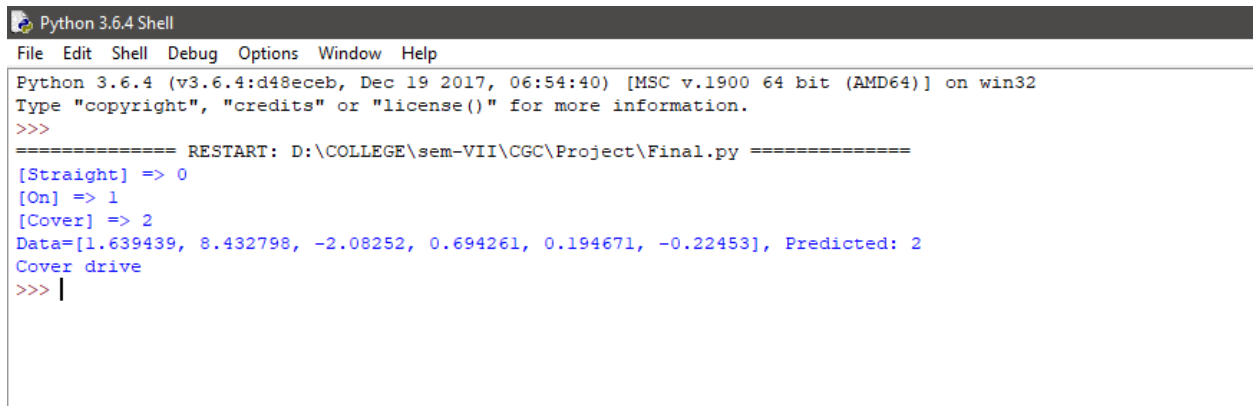
Figure 9: Test case output for straight drive

Test 2: Cover Drive

Test Data: Data is in the form of means for 3 axes of both sensors. The means are calculated by the program itself from the testing dataset generated. Hence, 6 mean values.

```
row = [1.639439, 8.432798, -2.08252, 0.694261, 0.194671, -0.22453]
```

Figure 10: Cover drive test data



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\COLLEGE\sem-VII\CGC\Project\Final.py =====
[Straight] => 0
[On] => 1
[Cover] => 2
Data=[1.639439, 8.432798, -2.08252, 0.694261, 0.194671, -0.22453], Predicted: 2
Cover drive
>>> |
```

Figure 11: Test case output for cover drive

USER MANUAL

Firstly, we open the HyperIMU mobile application on mobile phone and select the two sensors whose reading we require namely, accelerometer and gyroscope as shown in figure 12.

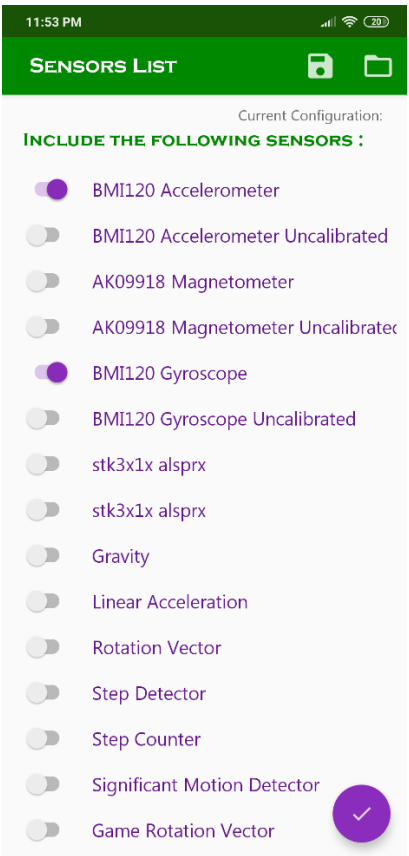


Figure 12: HyperIMU sensor selection

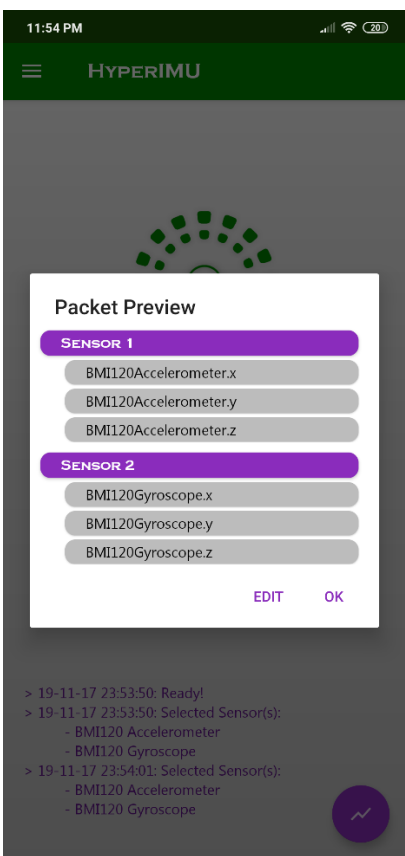


Figure 13: HyperIMU sensor axes

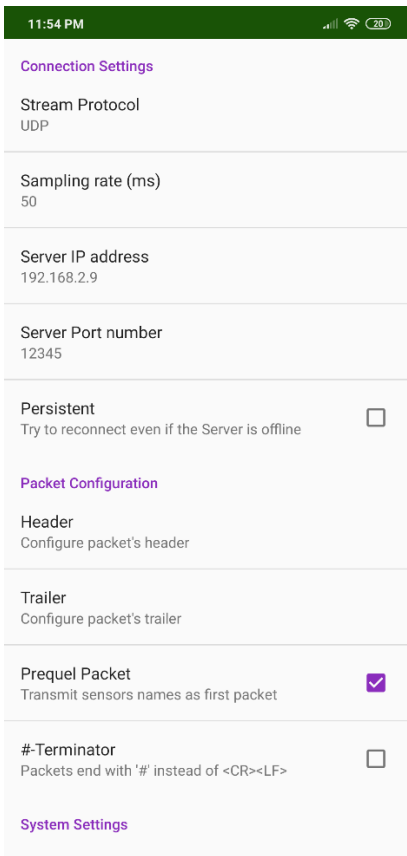


Figure 14: HyperIMU IP configuration

In Figure 13, depicts the axes in each sensor. In figure 14, we set the server IP Address i.e. the host PC IP Address and port number. Then, select stream protocol to be UDP and set sampling rate as 50ms.

```

project.m  x  +
1  %CodeStart-----
2  %Resetting MATLAB environment
3  -  instrreset
4  -  clear
5  -  clc
6  %Creating UDP object
7  -  UDPComIn=udp('192.168.2.9','LocalPort',12345);
8  -  set(UDPComIn,'DatagramTerminateMode','off')
9  %Opening UDP communication
10 -  fopen(UDPComIn);
11 %Reading data
12 -  t0 = clock;
13 -  headers = {'A','B','C','D','E','F'};
14 -  csvwrite('TestData.csv',headers);
15 -  while etime(clock, t0) < 3

```

Figure 15: Server IP Address added in MATLAB

Figure 15 shows the IP Address of server added to the code in MATLAB as previously done in figure 14 in the mobile application.

Below, figure 16 show the application when sensor is streaming the readings.

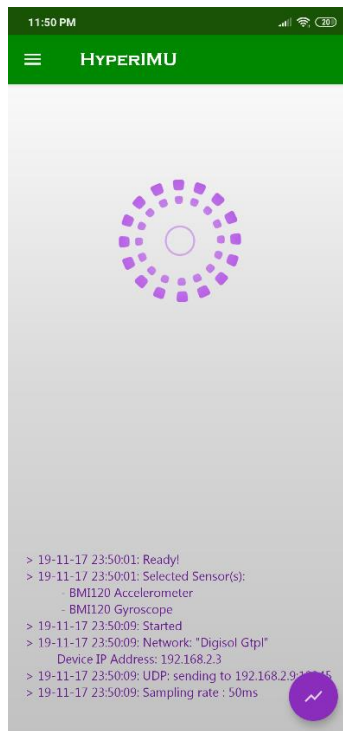


Figure 16: Streaming sensor readings

Now, when you run the MATLAB code, and play a *shot (Straight Drive)* you can see the readings in the console window of MATLAB. This is shown in figure 17.

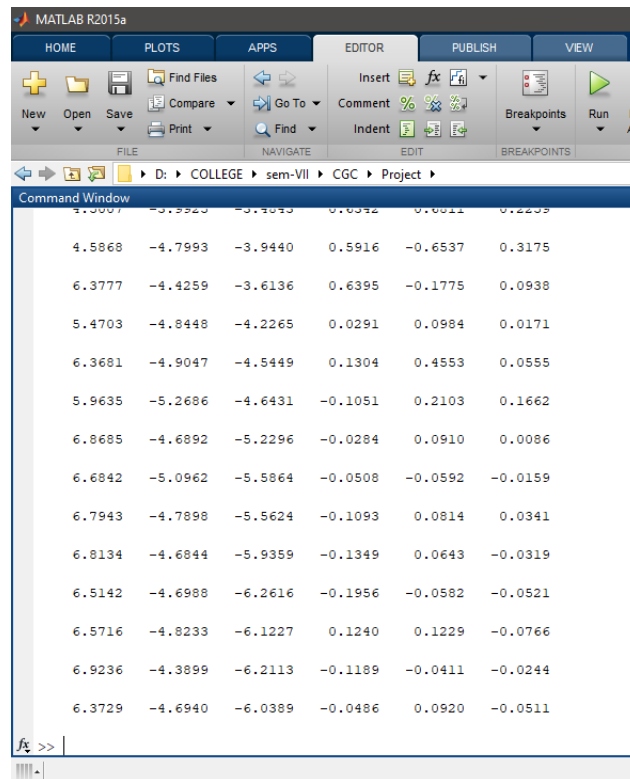


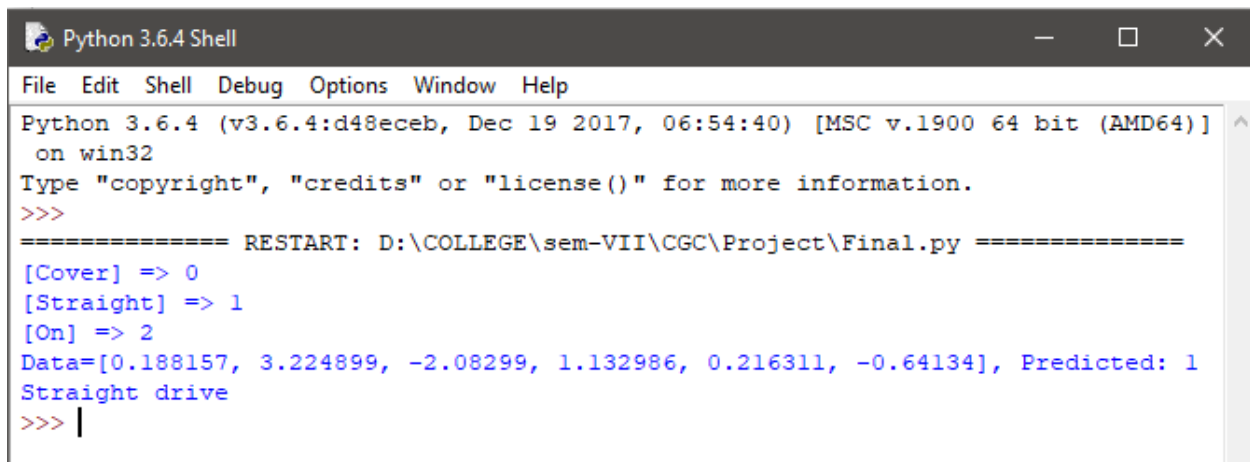
Figure 17: Readings received in MATLAB

After the shot is played, the dataset is stored as a comma-separated file by name TestData.csv as shown in figure 18.

Name	Date modified	Type	Size
Final.py	17-Nov-19 11:58 P...	Python File	5 KB
project.m	17-Nov-19 11:49 P...	M File	1 KB
TestData.csv	17-Nov-19 11:52 P...	Microsoft Excel C...	3 KB
train.csv	16-Nov-19 1:49 PM	Microsoft Excel C...	2 KB

Figure 18: Test data stored as csv file

Now, run the python code to predict the shot played.



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\COLLEGE\sem-VII\CGC\Project\Final.py =====
[Cover] => 0
[Straight] => 1
[On] => 2
Data=[0.188157, 3.224899, -2.08299, 1.132986, 0.216311, -0.64134], Predicted: 1
Straight drive
>>> |
```

Figure 19: Final result

The shot predicted by our cricket shot predictor shows “Straight drive”. This is shown in above figure 19. Along with this, a voice output is also noticed.

CONCLUSION AND FUTURE WORK

The current work was conducted with a set of ball-free shots by a batsman, at different speeds. For the data classified i.e. training data, all the tests performed with testing data which was collected by actually playing ball-free shots, has predicted correct shots.

The future scope for this project is really wide. It can be scaled well to predict many other different types of shots, and the accuracy of that shot should be displayed. Along with that, creation of an application that has the functionality of performing all the tasks, can be done.

REFERENCES

- 1) <https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/>
- 2) <https://doi.org/10.1016/j.proeng.2011.05.078>
- 3) <https://doi.org/10.1016/j.proeng.2015.07.202>
- 4) <https://doi.org/10.1016/j.proeng.2014.06.106>
- 5) <https://pypi.org/project/pyttsx3/>
- 6) <https://stackoverflow.com/questions/30612298/text-to-speech-tts-module-that-works-under-python-3>