PROGRAMMABLE EMBEDDED SYSTEMS

ANDROID ACCELEROMETER CLASSIFICATION ASSIGNMENT

Sai Joshitha Annareddy

21EE62R10

Android Code:

Main Activity Java:

```
package com.example.accelerometerassignment;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.view.View;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.widget.EditText;
import android.widget.Button;
import java.text.DecimalFormat;

import org.apache.commons.math3.exception.DimensionMismatchException;
import org.apache.commons.math3.exception.NullArgumentException;
import org.apache.commons.math3.linear.Array2DRowRealMatrix;
import org.apache.commons.math3.linear.ArrayRealVector;
import org.apache.commons.math3.linear.CholeskyDecomposition;
import org.apache.commons.math3.linear.MatrixDimensionMismatchException;
import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.linear.NonSquareMatrixException;
import org.apache.commons.math3.linear.RealMatrix;
import org.apache.commons.math3.linear.RealVector;
import org.apache.commons.math3.linear.SingularMatrixException;
import org.apache.commons.math3.util.MathUtils;

import org.apache.commons.math3.filter.MeasurementModel;

import org.apache.commons.math3.filter.ProcessModel;

import java.io.File;
import java.io.FileNotFoundException;
import java.lang.NullPointerException;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;



import com.example.accelerometerassignment.R;

public class MainActivity extends AppCompatActivity implements SensorEventListener {
    private static final String TAG = "MainActivity";
    private boolean isSensorAvailable = false;
    private TextView xText, yText, zText, xaText, yaText, zaText;
    private EditText edittext;
```

```
    private Button Button1;
    private Sensor mySensor;
    private SensorManager SM;
    private double MagnitudePrevious;
    DecimalFormat precision = new DecimalFormat("0.000");
    private double kalman_out[] = new double[3];
    StringBuilder data;
    //Current states
    private double X;
    private double Y;
    private double Z;
    private double Vx;
    private double Vy;
    private double Vz;
    private double Ax;
    private double Ay;
    private double Az;

    double time = 0;
    //previous states;

    private double timestamp = 0;
    double dT;
    private final float nanosecond = 1.0f / 1000000000.0f;

    double[][] MeasurementMatrix = {
            {0, 0, 0, 0, 0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 1}};
    RealMatrix H = MatrixUtils.createRealMatrix(MeasurementMatrix);//H
    RealMatrix R_noise =
MatrixUtils.createRealIdentityMatrix(3).scalarMultiply(0.01);//R
    RealMatrix processNoise = MatrixUtils.createRealIdentityMatrix(9);
    RealMatrix I = processNoise;
    RealMatrix Q = processNoise.scalarMultiply(0.35);//Q
    RealMatrix P = processNoise.scalarMultiply(1);//P

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final Button stopButton = (Button) findViewById(R.id.stop);
        edittext = (EditText) findViewById(R.id.editText);
        xText = (TextView) findViewById(R.id.xText);
        yText = (TextView) findViewById(R.id.yText);
        zText = (TextView) findViewById(R.id.zText);


        xText.setTextColor(Color.WHITE);
        yText.setTextColor(Color.WHITE);
        zText.setTextColor(Color.WHITE);


        final Button startButton = (Button) findViewById(R.id.start);
        X = 0D;
        Y = 0D;
        Z = 0D;
        Vx = 0D;
        Vy = 0D;
        Vz = 0D;
        Ax = 0D;
        Ay = 0D;
        Az = 0D;

        //create sensor manager
        SM = (SensorManager) getSystemService(SENSOR_SERVICE);
```

```
        //Assign Sensor Manger of type Accelerometer to our sensor
        mySensor = SM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);


    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        Log.d(TAG, "onSensorChanged: " + event.values[0]);

        double linear_acceleration[] = new double[3];



        linear_acceleration[0] = event.values[0];
        linear_acceleration[1] = event.values[1];
        linear_acceleration[2] = event.values[2];
        if (timestamp != 0) {
            dT = (event.timestamp - timestamp) * nanosecond;
        }
        timestamp = event.timestamp;
        double h = dT;

        time = time + h;

        double h2 = Math.pow(h, 2) / 2;
        double[][] state = {
                {1, 0, 0, h, 0, 0, h2, 0, 0},
                {0, 1, 0, 0, h, 0, 0, h2, 0},
                {0, 0, 1, 0, 0, h, 0, 0, h2},
                {0, 0, 0, 1, 0, 0, h, 0, 0},
                {0, 0, 0, 0, 1, 0, 0, h, 0},
                {0, 0, 0, 0, 0, 1, 0, 0, h},
                {0, 0, 0, 0, 0, 0, 1, 0, 0},
                {0, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 0, 0, 0, 0, 0, 0, 0, 1}
        };

        RealMatrix phi = MatrixUtils.createRealMatrix(state);
        double[] measuredState = new double[]{linear_acceleration[0],
linear_acceleration[1], linear_acceleration[2]};
        RealVector z = MatrixUtils.createRealVector(measuredState);//Z
        double[] initialStateEstimate = {X, Y, Z, Vx, Vy, Vz, Ax, Ay, Az};
        RealVector x = MatrixUtils.createRealVector(initialStateEstimate);
        RealMatrix s = H.multiply(P).multiply(H.transpose()).add(R_noise);
        RealVector innovation = z.subtract(H.operate(x));
        RealMatrix KalmanGain = new CholeskyDecomposition(s).getSolver()
                .solve(H.multiply(P.transpose())).transpose();
        x = x.add(KalmanGain.operate(innovation));
        P = (I.subtract(KalmanGain.multiply(H))).multiply(P);
        x = phi.operate(x);
        P = phi.multiply(P).multiply(phi.transpose()).add(Q);

        double[] estimated_state = x.toArray();

        kalman_out[0] = estimated_state[6];
        kalman_out[1] = estimated_state[7];
        kalman_out[2] = estimated_state[8];

        xText.setText("X: " + precision.format(kalman_out[0]));
        yText.setText("Y: " + precision.format(kalman_out[1]));
        zText.setText("Z: " + precision.format(kalman_out[2]));


        data.append("\n" + time + "," + kalman_out[0] + "," + kalman_out[1] + "," +
kalman_out[2] + "," + linear_acceleration[0] + "," + linear_acceleration[1] + "," +
linear_acceleration[2]);
```

```
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //Not in use
    }

    @Override
    protected void onResume() {
        super.onResume();
        if (isSensorAvailable) {
            mySensor = SM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
            SM.registerListener(this, mySensor, SensorManager.SENSOR_DELAY_NORMAL);
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (isSensorAvailable) {
            SM.unregisterListener(this);
        }
    }

    public void toggle(View view) {
        data = new StringBuilder();
        ;
        data.append("T,Xkal,Ykal,Zkal,Xa,Ya,Za");
        mySensor = SM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        SM.registerListener(this, mySensor, SensorManager.SENSOR_DELAY_NORMAL);
    }

    public void trigger(View view) {

        SM.unregisterListener(this);

        FileOutputStream fos = null;
        String value1 = edittext.getText().toString();
        String filename = value1 + ".csv";
        String path = "Accelerometerdata";
        File filelocation = new File(getExternalFilesDir(path), filename);
        try {
            fos = new FileOutputStream(filelocation);
            fos.write(data.toString().getBytes());
            fos.close();
            Log.d(TAG, "trigger:" + filename + " " + path);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

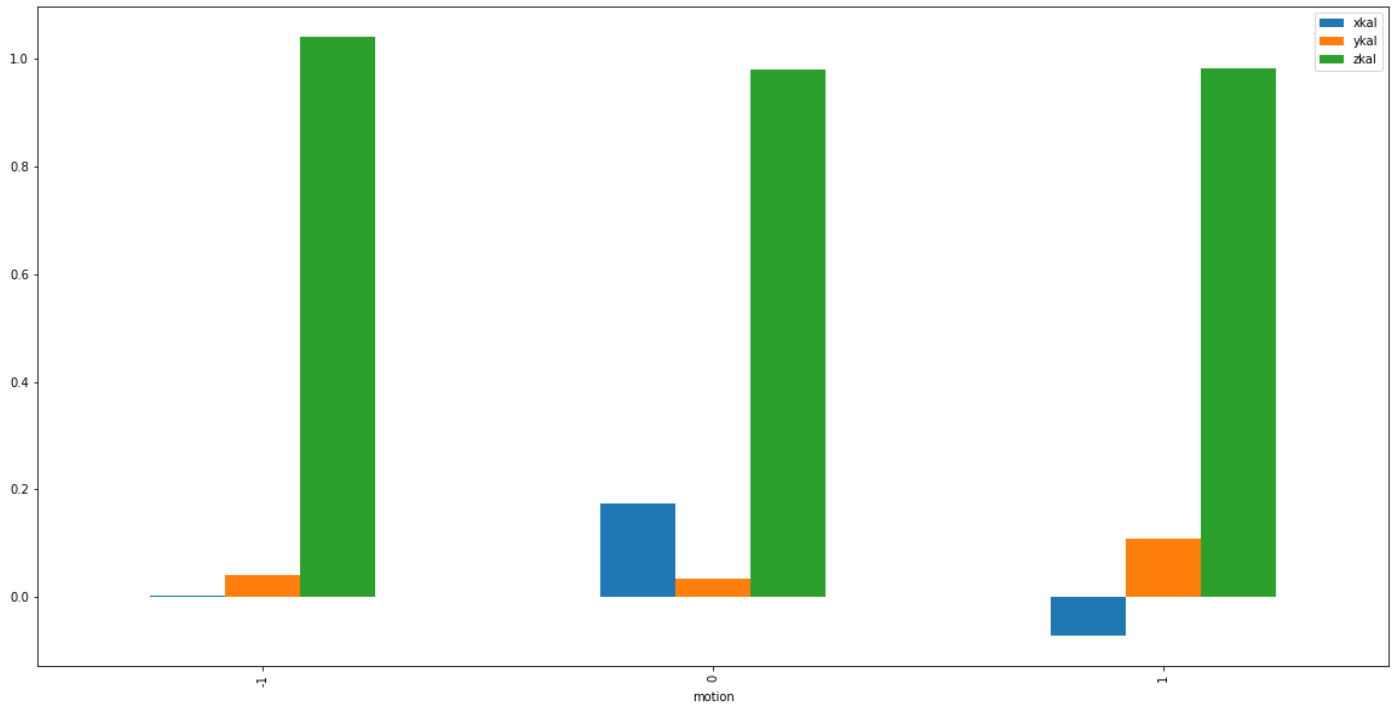
Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="?attr/colorButtonNormal"
    android:gravity="top|center"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/xText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:layout_weight="1"
        android:text="X"
        android:textSize="15dp"
        tools:layout_editor_absoluteX="152dp"
        tools:layout_editor_absoluteY="40dp" />

    <TextView
        android:id="@+id/yText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Y"
        android:textSize="15dp"
        tools:layout_editor_absoluteX="152dp"
        tools:layout_editor_absoluteY="60dp" />

    <TextView
        android:id="@+id/zText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Z"
        android:textSize="15dp"
        tools:layout_editor_absoluteX="152dp"
        tools:layout_editor_absoluteY="80dp" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:ems="10"
        android:inputType="text"
        android:textColor="@color/design_default_color_secondary"
        tools:layout_editor_absoluteX="150dp"
        tools:layout_editor_absoluteY="120dp" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="40dp"
        android:onClick="toggle"
        android:text="Start"
```

```
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="140dp" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText2"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="100dp"
        android:onClick="trigger"
        android:text="Stop"
        tools:layout_editor_absoluteX="148dp"
        tools:layout_editor_absoluteY="160dp" />

</LinearLayout>
```

Python code to classify linear motion, circular motion and sleep:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#Loading dataset

clasify = pd.read_csv('motions.csv')

clasify.head()

class_pivot =clasify.pivot_table(index='motion')

class_pivot.plot.bar(figsize=(20,10))

plt.show()

##-1 for walking/linear motion

## 1 for sleeping

## 0 for circular motion
```

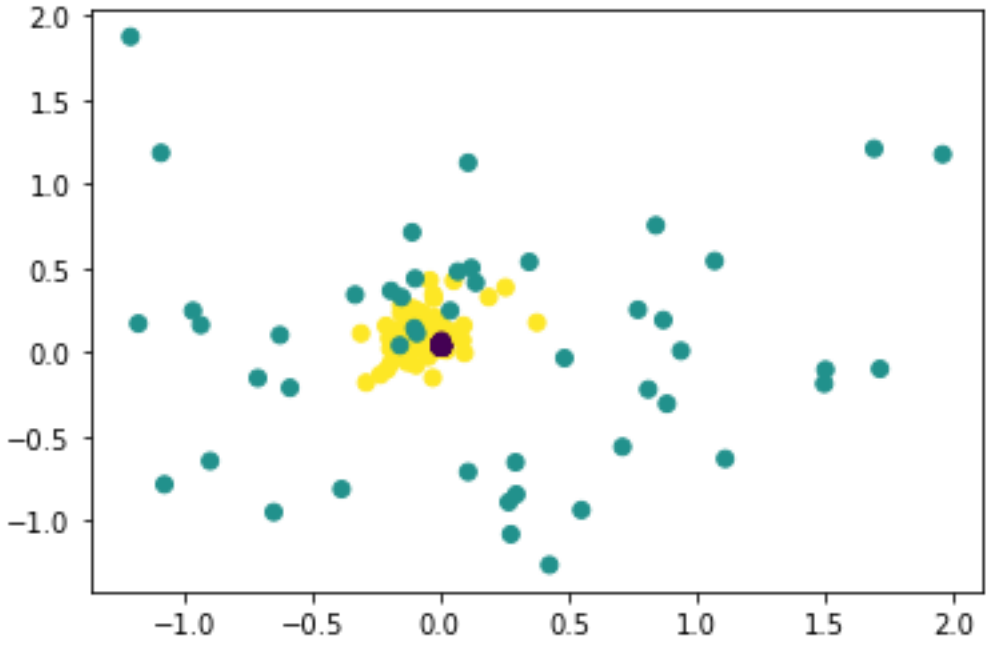



```
cols=['xkal','ykal']

X=clasify[cols]

y=clasify['motion']

from sklearn.model_selection import train_test_split


all_X=clasify[cols]

all_y=clasify['motion']


train_X,test_X,train_y,test_y=train_test_split(all_X,all_y,test_size=0.25,random
_state=0)


#train_X,test_X,train_y,test_y=train_test_split(all_X,all_y,test_size=0.4,random
_state=0)


#train_X,test_X,train_y,test_y=train_test_split(all_X,all_y,test_size=0.1,random
_state=0)

train_X.shape
```

```
(154, 2)
```

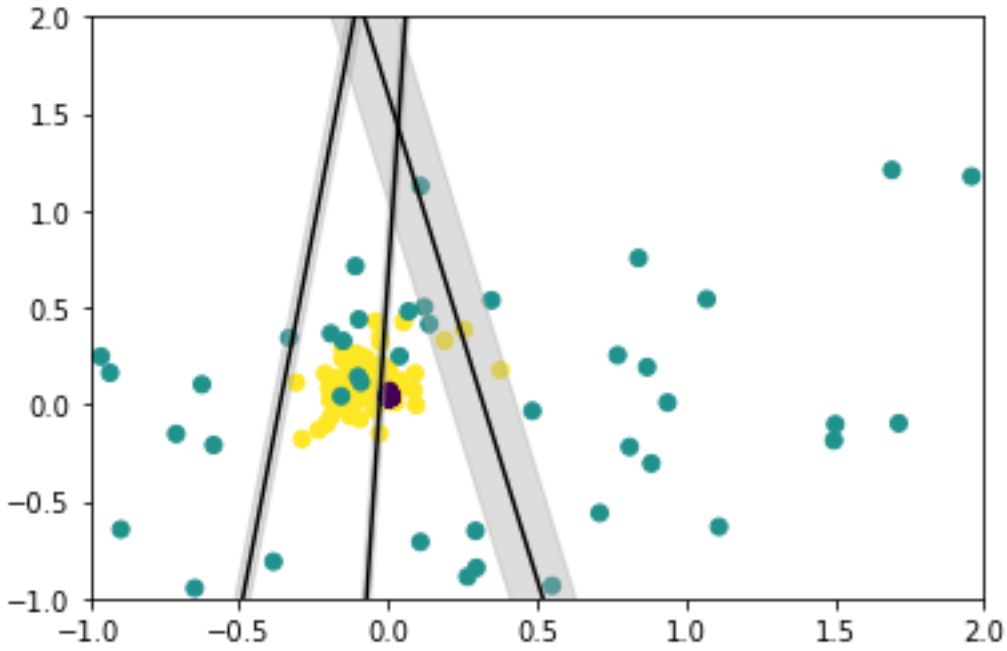
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svcmodel = SVC(kernel='rbf')
svcmodel.fit(train_X,train_y)
prediction = svcmodel.predict(test_X)
accuracy_svcm= accuracy_score(test_y,prediction) # can use for test or train
print(accuracy_svcm)
```

```
plt.scatter(clasify["xkal"], clasify["ykal"], c = clasify['motion'])
```




```
xfit = np.linspace(-1, 3.5)
plt.scatter(clasify["xkal"], clasify["ykal"], c = clasify['motion'])

for m, b, d in [(23, 0.65, 0.33),  (8, 2.9, 0.2),(-5, 1.6, 0.55)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)
```

```
plt.xlim(-1, 2);
plt.ylim(-1,2);
```




```
from sklearn.model_selection import cross_val_score

scores=cross_val_score(svcmodel,all_X,all_y,cv=10)
scores.sort()
accuracy_svc=scores.mean()

print(scores)
print(accuracy_svc)
```

```
[0.57142857 0.6        0.7        0.85       0.85714286 0.85714286
 0.85714286 0.9047619  0.95       1.        ]
0.8147619047619049
```

```
# creating a confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_y, prediction)
cm
```

```
array([[21,  0,  0],
       [ 0, 11,  1],
       [ 5,  1, 13]])
```

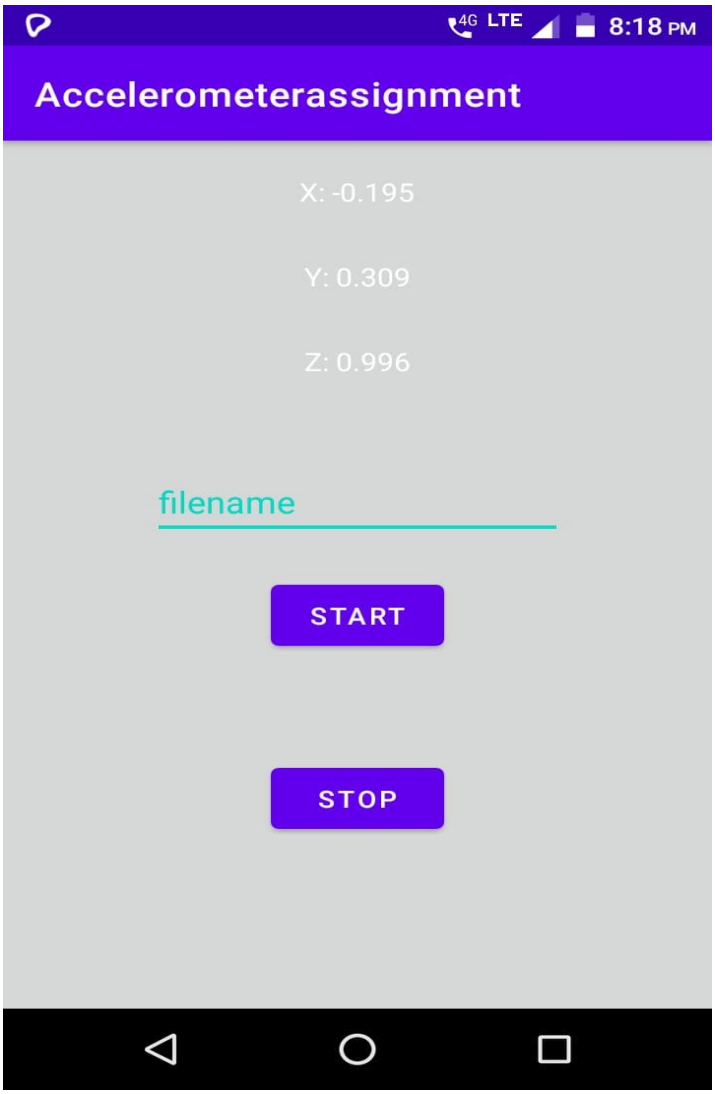
Android code : classification.java

```
package com.example.accelerometerassignment;

import android.os.Build;

import androidx.annotation.RequiresApi;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;

import libsvm.svm;
import libsvm.svm_model;
import libsvm.svm_node;
import libsvm.svm_parameter;
import libsvm.svm_problem;

public class classification {

    static int record_count = 230;
    static int feature_count = 4;


    // Labels[num_records]
    static int[] labels = new int[record_count];

    // Feature[feature_index][feature_value]
    static double[][] features = new double[record_count][feature_count];

    //Create model that will train on datapoints
    static svm_model svm_model_data;

    @RequiresApi(api = Build.VERSION_CODES.O)
    public static void main(String[]args) {

        readFile();

        svm_model_data = svmTrain(features, labels, 160);

        double[][] test_data = new double[60][4];

        for(int i = 0; i < 60; i++){
            test_data[i] = features[i+100];
        }

        double[] ypred = svmPredict(test_data, svm_model_data);

        for (int i = 0; i < test_data.length; i++){
            System.out.println("(Actual:" + labels[i+100] + " Prediction:" + ypred[i] +
")");
        }


    }



    public static svm_model svmTrain(double[][] xtrain, int[] ytrain, int train_size) {

        svm_problem prob = new svm_problem();
        prob.y = new double[train_size];
        prob.l = train_size;
```

```
        prob.x = new svm_node[train_size][feature_count];

        // For each record
        for (int instance = 0; instance < train_size; instance++) {
            double[] features = xtrain[instance];

            prob.x[instance] = new svm_node[feature_count];
            for (int feature = 0; feature < feature_count; feature++) {
                svm_node node = new svm_node();
                node.index = feature;
                node.value = features[feature];
                prob.x[instance][feature] = node;
            }

            // Set the label for this record.
            prob.y[instance] = ytrain[instance];
        }

        svm_parameter param = new svm_parameter();
        param.probability = 1;
        param.gamma = 0.5;
        param.nu = 0.5;
        param.C = 10;
        param.svm_type = svm_parameter.C_SVC;
        param.kernel_type = svm_parameter.LINEAR;
        param.cache_size = 20000;
        param.eps = 0.001;

        svm_model model = svm.svm_train(prob, param);

        return model;
    }


    public static double[] svmPredict(double[][] xtest, svm_model model) {

        double[] yPred = new double[xtest.length];

        for (int k = 0; k < xtest.length; k++) {
            double[] fVector = xtest[k];

            svm_node[] nodes = new svm_node[fVector.length];

            for (int i = 0; i < fVector.length; i++) {
                svm_node node = new svm_node();
                node.index = i;
                node.value = fVector[i];
                nodes[i] = node;
            }

            int totalClasses = 3;
            int[] labels = new int[totalClasses];
            svm.svm_get_labels(model, labels);

            double[] prob_estimates = new double[totalClasses];
            yPred[k] = svm.svm_predict_probability(model, nodes, prob_estimates);
        }

        return yPred;

    }

    /**
     *
     */
    @RequiresApi(api = Build.VERSION_CODES.O)
    public static void readFile(){
```

```
        String path = "getExternalFilesDir(Internal Storage) motionclassify.csv";

        try {
            List<String> content;
            content = Files.readAllLines(Paths.get(path));

            String[] line_contents;
            String[] line_features;

            int current_record = 0;

            //For each line in file
            for(String line : content){
                line_contents = line.split(",");
                labels[current_record] = Integer.parseInt(line_contents[0]);

                line_features = line_contents[1].split(",");
                features[current_record][0] = Double.parseDouble(line_features[0]);
                features[current_record][1] = Double.parseDouble(line_features[1]);
                features[current_record][2] = Double.parseDouble(line_features[2]);

//                System.out.println("Label: " + labels[current_record]);
//                System.out.println("F1: " + features[current_record][0]);
//                System.out.println("F2: " + features[current_record][1]);
//                System.out.println("F3: " + features[current_record][2]);

                current_record++;
            }

        } catch (IOException e) {
            e.printStackTrace();
        }


        //Parse
        // For each feature in line

    }


}
```

Results:



In android app the data from accelerometer sensor is collected and filtered using Kalman filter in Android studio. The data is converted into csv file and used in python code to classify into different motions. In python by using linear svm classifier I got 81% accuracy.