Classification of Three Ballet Jumps with Machine Learning

Names	Ana Boburg Nystedt and Allyson Bieryla
Team Name	The Eastenders
Class	DGMD-14
Date	August 4, 2020

HARVARD
Extension School

Contents

Introduction

The purpose of this project is to try to classify three small ballet jumps using a wearable device and machine learning. There are many wearable devices available to evaluate and measure different sports. However, there are no commercially available devices for ballet training. Our project is aiming to use wearable technology to classify and hopefully identify good technique in ballet steps.

For the purpose of the project the three ballet steps were recorded in isolation without transition movements. We used one SensorTile [1] for the wearable device, and Python libraries to create Support Vector Machines (SVM)[2] machine learning models with three different kernels (linear, polynomial and radial basis function). To select our features, we used both manual extraction, and automatic feature extraction using the ExtraTreesClassifier.

We obtained the best results of 93.3% accuracy with the features extracted automatically with the ExtraTreesClassifier, and using the SVM model with the polynomial kernel.

With the results from this limited project, we conclude that it is possible to design a wearable to classify specific ballet movements. This can have many applications such as for beginner dancers trying to apply the right technique to get their specific exercises recognized. The project can be extended to recognize more movements inside a choreography.

1. System Setup

The SensorTile module is a tiny, square of approximately 1.5 cm x 1.5 cm used in the Internet of Things (IoT) to gather data through its sensors. It has an 80 MHz STM32L476JGY microcontroller and Bluetooth low energy connectivity based on BlueNRG-MS network processor. It includes a digital microphone and many motion and environmental MEMS (Micro electro-mechanical systems) sensors. The SensorTile comes with 1Mbyte of flash memory.

The SensorTile Module can be by itself the core processing unit of a wearable after being "flashed" with a program. To upload (or flash) new firmware onto the SensorTile, an external SWD debugger is needed. We will use the recommended ST-LINK/V2-1 found on the STM32 Nucleo-64 development board.

The SensorTile contains an LSM6DSM. This is a system-in-package featuring a 3D digital accelerometer and a 3D digital gyroscope performing at 0.65 mA in high-performance mode and enabling always-on low-power. The LSM6DSM has a full-scale acceleration range of $\pm 2/\pm 4/\pm 8/\pm 16$ g and an angular rate range of $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000$ dps (degrees per second).

For our project we used a big cradle, given soldering constraints. However, a smaller cradle (and small battery) comes included with the SensorTile kit.

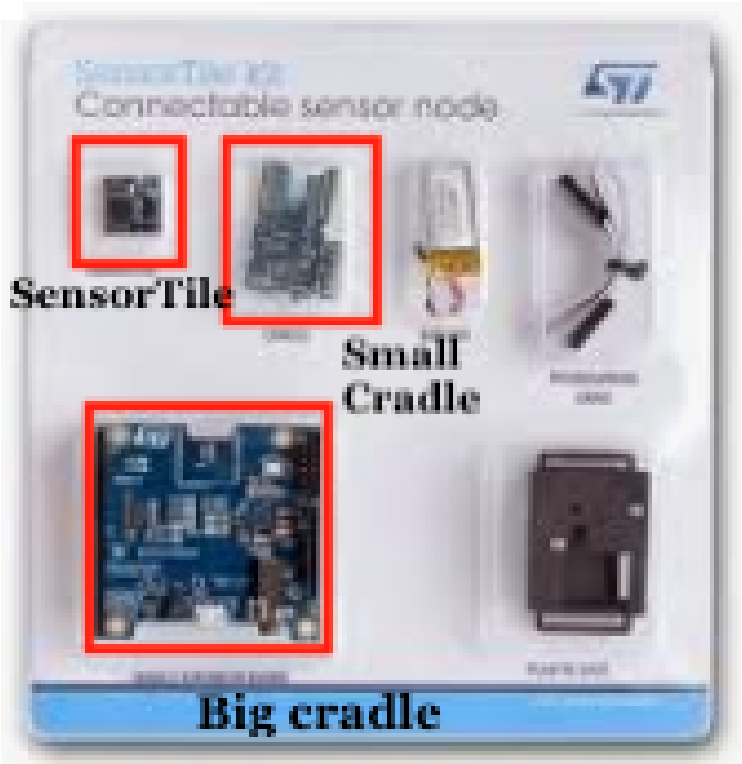


Figure 1.1: SensorTile Kit

Below is the general system setup. The SensorTile is connected to the big cradle, which is used to create the wearable device together with a battery. The wearable device then communicates with the iPhone ST BLE app via Bluetooth. The app collects the information in .csv files, one file per sensor (accelerometer, gyroscope and magnetometer) and creates an email with the three data files attached. The .csv files can be then sent for processing.

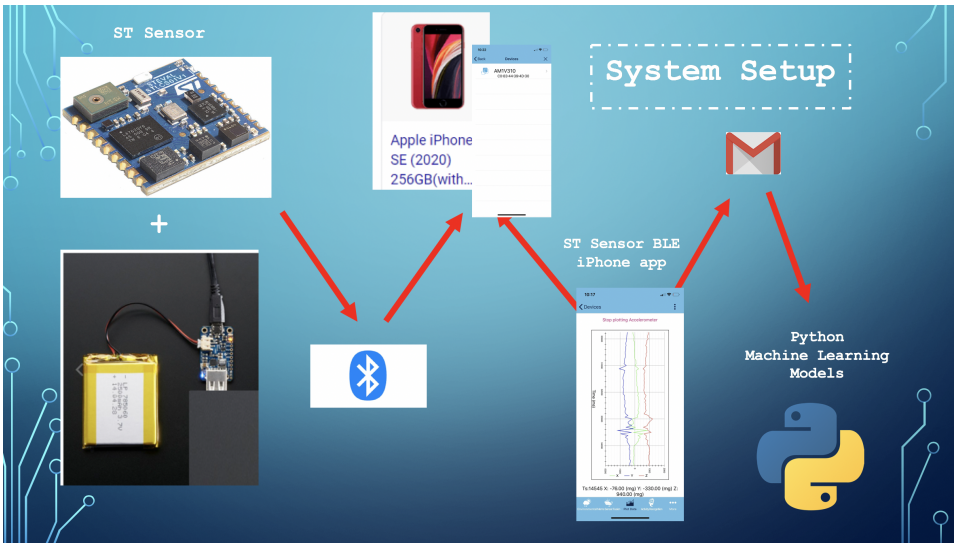


Figure 1.2: System Setup Diagram

2. Prototype Design

2.1 Initial Prototype

The initial prototype used a 3 AAA battery pack, but it was too big and heavy, shown in Fig 3.1 and Fig 5.1.

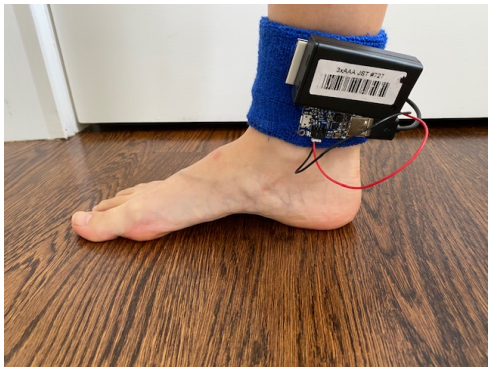


Figure 2.1: SensorTile inner ankle.

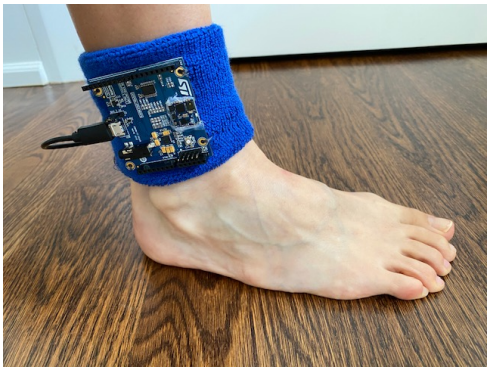


Figure 2.2: SensorTile outer ankle.

2.2 Final Prototype

The final prototype uses a Lithium Ion battery (3.7V 500 mAh with a power boost)

Figure 2.3: SensorTile inner ankle.

Figure 2.4: SensorTile outer ankle.

2.3 Future Prototype

Given time restrictions, we were not able to use the small cradle of the SensorTile included in the Kit. To be able to use, it required a special soldering technique, given the small size of the SensorTile's pins. A future prototype can take advantage of the small SensorTile cradle and use the smaller Lithium Ion battery (3.7V 100 mAh) provided in the SensorTile Kit. They are both shown at the bottom of Figure 2.5.

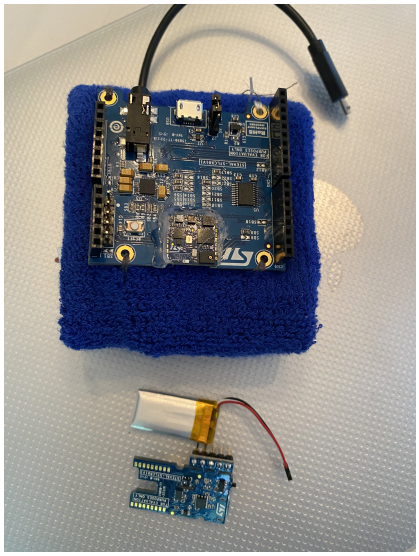


Figure 2.5: Future Prototype using small SensorTile cradle and battery (bottom of image)

3. Data Collection

3.1 Data Collection Program

To create the data collection program, we used the "System Workbench for STM32 - Bare Metal Edition", which is an Eclipse integrated IDE. It provides a software development platform for the STM32 board. This IDE helps to quickly create a C embedded project for the target board, the SensorTile.

The program [3] enabled Bluetooth LE communication with the SensorTile, and transmitted data from the accelerometer, gyroscope and magnetometer according to the parameters specified.

For this application we collected data at a sampling reate of 10 ms (100 HZ) and setup the accelerometer Full Scale to be +/- 2G (after collecting all the data and inspecting it, we observed that it would have been better to increase the Full Scale to 4G). This is mentioned in Chapter 7 of this report where future improvements are discussed.

STM32 ST-LINK Utility to flash (save) the data collection program into the SensorTile, and also the bootloader. This allows the SensorTile to work independently from the computer when it is connected to a battery.

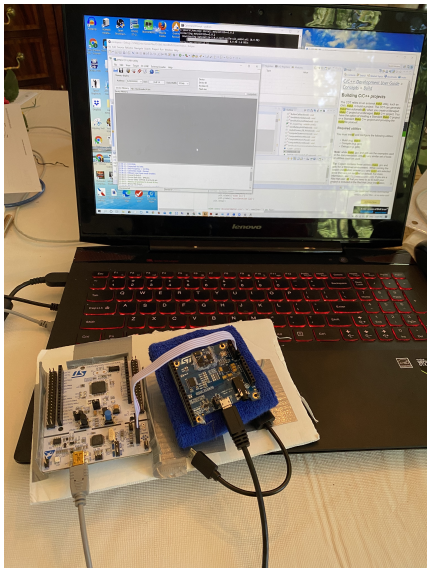


Figure 3.1: SensorTile flashing the data collection program.

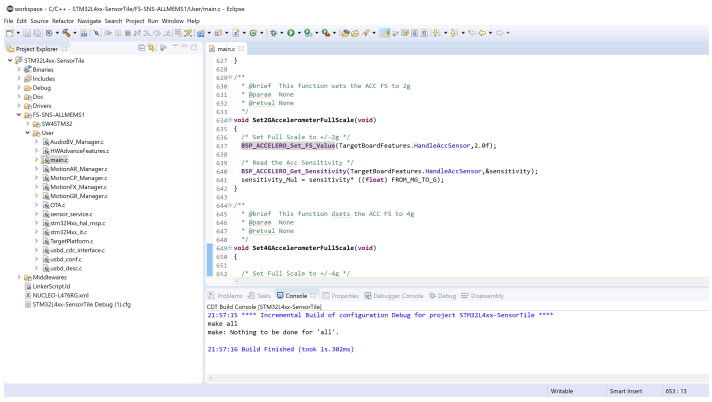


Figure 3.2: Firmware sourcecode for data collection.

3.2 ST BLE Sensor iPhone App

The ST BLE Sensor iPhone App was used to connect to the SensorTile via Bluetooth LE, and collect the data from the different sensors. We recorded each jump individually, starting and stopping at the beginning and end of each jump. The App collects information from each sensor and creates 3 CSV files, one for the accelerometer, one for the gyroscope and one for the magnetoscope. The files contain timestamp information from the iPhone, from the sensor, and the measurements from the X,Y and Z axis of the three sensors. These files were attached in an email so we could process the data on our computers.

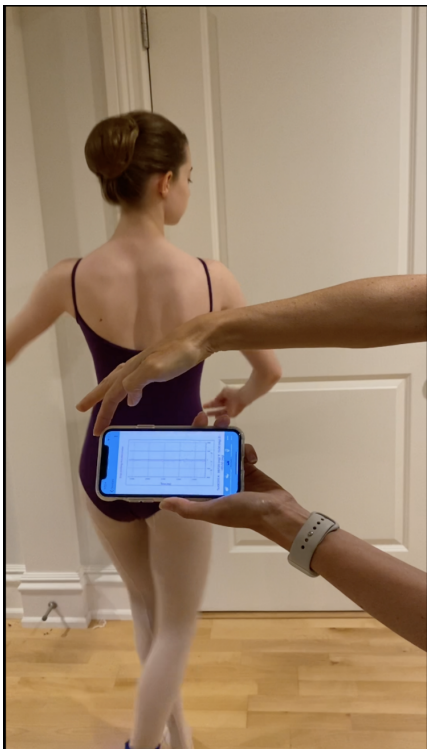


Figure 3.3: Capturing data with the ST BLE Sensor App.

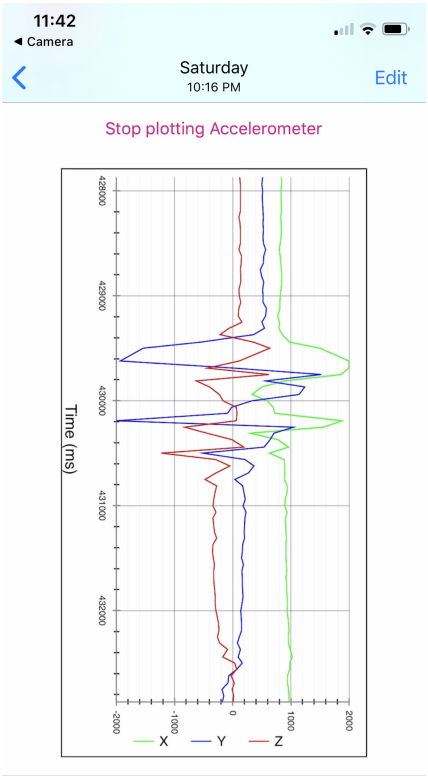


Figure 3.4: ST BLE Sensor App.

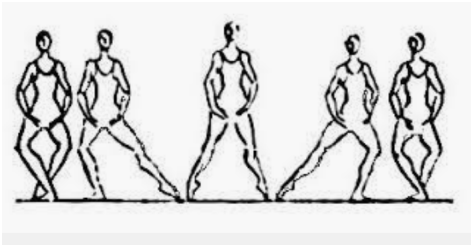
3.3 Small Ballet Jumps

The data was collected with the help of a ballet student (intermediate level), who did 50 individual repetitions of each of the small ballet jumps: (i) Glissade, (ii) Pas-de-chat, and (iii) Jeté. The dancer placed the sensor on her right ankle.

We also collected 10 repetitions of each jump from a dancer who had a beginner level for testing purposes. We think this kind of wearable could also be helpful to students trying to gauge the quality of an exercise. For example, if the particular jump is not recognized by the model, it may mean the ballet student is not using the full technique of the jump.

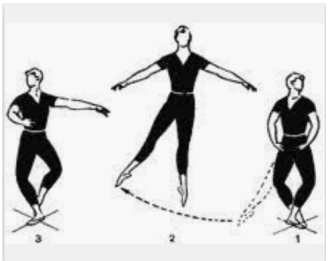
More information could be captured if we had at least two sensors, one on each foot. Also if our wearable would be smaller to place it on the foot as opposed to place it in the ankle.

Figure 3.5: Dancer with SensorTile on right foot.



(a) Glissade

(b) Pas-de-chat



(c) Jeté

Figure 3.6: Ballet Technique for each jump

4. Data Processing and ML Development Tools

All the code for the data processing, machine learning model creation, and testing can be found in the following GitHub repository. During all of these tasks, we used "pair programming" through Zoom, and many different tools, which are listed below.

Figure 4.1: Python Tools

5. Data Processing

5.1 Organizing Data

The data that were collected via the iPhone App were sent via email and then organized in directories on our local computer. The data cleaning and processing were done in Python Notebooks. We used built-in scripts from the Python Pandas library[7] to read the columns of data from the raw data in the CSV files into dataframes.

20200724_161815_Accelerometer

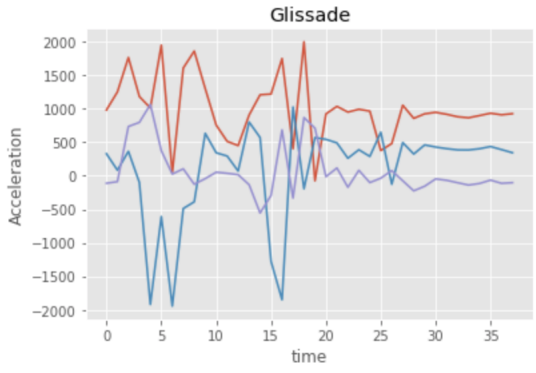
1	**Logged started on**	**24/07/2020 16:18:15**						
2	**Feature**	Accelerometer						
3	**Nodes**	AM1V310 @394D30						
4								
5	**Date**	HostTimestamp	NodeName	NodeTimestamp	RawData	X (mg)	Y (mg)	Z (mg)
6	**24/07/2020 16:27:33.115**	557888	AM1V310 @394D30	134599	7E06A407F104	1662	1956	1265
7	**24/07/2020 16:27:33.175**	557940	AM1V310 @394D30	134605	2805FF032E00	1320	1023	46
8	**24/07/2020 16:27:33.204**	557969	AM1V310 @394D30	134612	8A04CD07F5FC	1162	1997	-779
9	**24/07/2020 16:27:33.264**	558030	AM1V310 @394D30	134618	6702CD078CFE	615	1997	-372
10	**24/07/2020 16:27:33.325**	558090	AM1V310 @394D30	134624	6A015B023601	362	603	310
11	**24/07/2020 16:27:33.354**	558120	AM1V310 @394D30	134630	AEFC9BF81204	-850	-1893	1042
12	**24/07/2020 16:27:33.415**	558180	AM1V310 @394D30	134637	DBFB32F86A03	-1061	-1998	874
13	**24/07/2020 16:27:33.474**	558239	AM1V310 @394D30	134643	1BFADEFD89FE	-1509	-546	-375
14	**24/07/2020 16:27:33.516**	558281	AM1V310 @394D30	134649	C1F8D700EDFD	-1855	215	-531
15	**24/07/2020 16:27:33.564**	558330	AM1V310 @394D30	134655	92FFFDFFF400	-110	-3	244

Figure 5.1: Example portion of raw data from the Acceleration of the Pas-de-chat jump

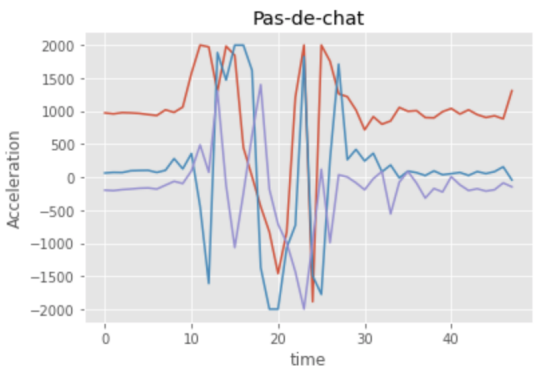
5.2 Visualizing and Cleaning Data

We plotted the X, Y, and Z components of the acceleration (as shown in Figure 5.2) and the gyroscope components (as shown in Figure 5.3) to visualize each of the jumps.

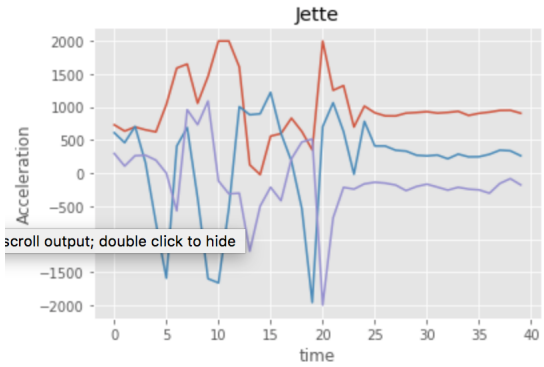
Data were visually inspected to ensure data quality. We removed any jumps that were not complete, or that just showed noise due to error in data collection. Most datasets were 3-5 seconds long. We clipped the observation if there was a longer period of data recorded because the remaining data was noise and unnecessary to the analysis. The cleaning was minimal and only 3 observations were removed from the analysis due to the above reasons.



(a) Glissade

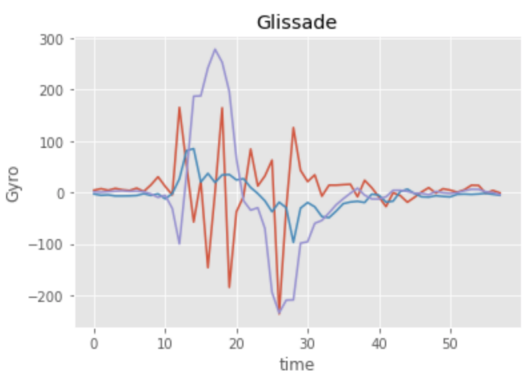


(b) Pas-de-chat



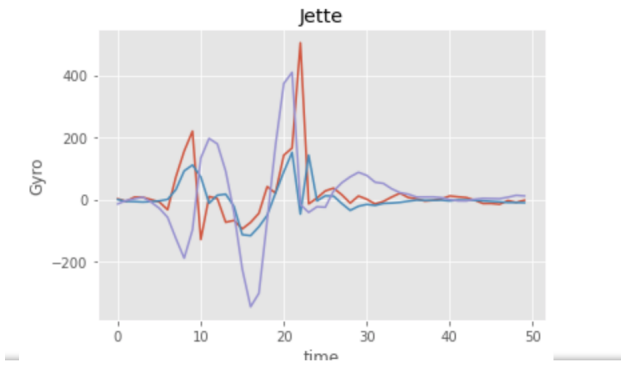
(c) Jete

Figure 5.2: Example Acceleration data for each jump



(a) Glissade

(b) Pas-de-chat



(c) Jete

Figure 5.3: Example Gyroscope data for each jump

5.3 Summarizing Data statistics

We used the describe() method, again within Python Pandas, to summarize our data statistics for each jump. The summary outputs (shown in Figure 5.4) were used to help us determine which features might be good to train the machine learning model with.

	Time	Accel_X	Accel_Y	Accel_Z
count	42.000000	42.000000	42.000000	42.000000
mean	63763.761905	1004.976190	68.142857	-28.761905
std	76.669328	419.901411	665.141488	364.025449
min	63636.000000	172.000000	-1984.000000	-1453.000000
25%	63699.500000	801.250000	-37.750000	-100.500000
50%	63764.000000	932.000000	277.500000	33.000000
75%	63827.500000	1168.500000	394.000000	142.000000
max	63892.000000	1997.000000	1047.000000	543.000000

(a) Glissade

	Time	Accel_X	Accel_Y	Accel_Z
count	1568.000000	1568.000000	1568.000000	1568.000000
mean	160991.605867	812.868622	173.739158	-52.538903
std	20238.739162	829.829175	919.825244	603.366583
min	123990.000000	-1998.000000	-1998.000000	-1997.000000
25%	143527.000000	799.500000	-46.250000	-216.250000
50%	158758.500000	969.000000	118.000000	-95.000000
75%	178866.750000	1039.250000	502.000000	126.250000
max	196247.000000	1998.000000	1997.000000	1998.000000

(b) Pas-de-chat

	Time	Accel_X	Accel_Y	Accel_Z
count	1662.000000	1661.000000	1660.000000	1660.000000
mean	74464.863418	962.194461	216.457831	-15.378916
std	21151.469656	536.490980	718.700378	426.658431
min	-59.000000	-1998.000000	-1998.000000	-1997.000000
25%	62268.500000	798.000000	180.000000	-181.000000
50%	77085.000000	922.000000	346.500000	-47.500000
75%	92463.500000	1034.000000	481.250000	151.250000
max	104486.000000	1998.000000	1997.000000	1992.000000

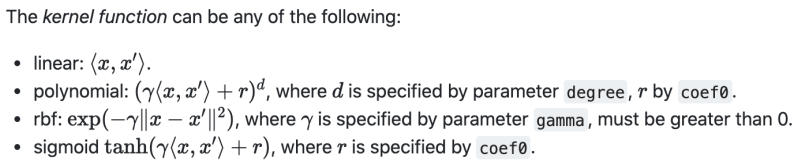
(c) Jete

Figure 5.4: Acceleration summary statistics for each jump

6. Machine Learning Model

SVM (Support Vector Machine) is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform the given data and then based on these transformations it finds an optimal boundary between the possible outputs.

We created three SVM models using Python tools, each with a different kernel: (i) Linear, (ii) Polynomial (POLY), and (iii) Radial Basis Function (RBF), to evaluate which one classified the three ballet jumps with more accuracy.


The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

Figure 6.1: Kernel Functions

The POLY kernel can have varying degrees. A POLY with degree 1 kernel is similar to that of a linear kernel. Our models used a degree 2 kernel which allowed a more flexible decision boundary. The RBF kernel, otherwise known as a Gaussian Kernel, is another example of a non-linear kernel that we used to train our model.

6.1 Feature Selection

A feature is an individual measurable property or characteristic of a phenomenon being observed. It is important that the features selected are informative and independent. In our case, we had all the measurements from the three sensors, plus any other that can be calculated from them (e.g. mean, std, magnitudes, etc).

Feature selection is the process of selecting a subset of attributes or variables of the dataset for use in model fitting. Features are selected to reduce the dimensionality of the dataset to allow for shorter training times and reduce the chance of overfitting.

Feature selection was an iterative process. We plotted many combinations of features using both the acceleration and gyroscope features. Ultimately, we decided to focus on the acceleration features because many of the gyroscope features were hard to visually distinguish from one another.

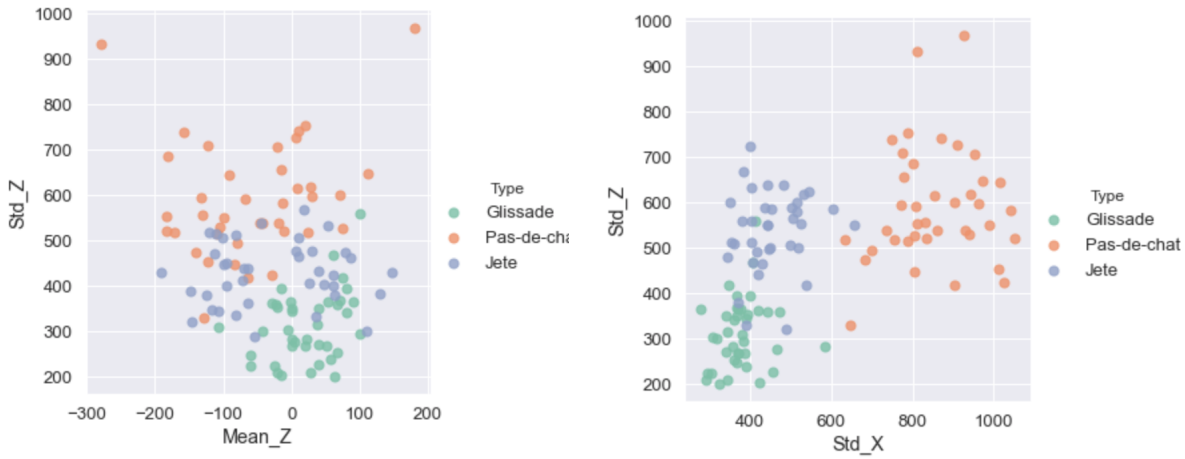
Focused on the acceleration, we first tried manual feature extraction. We chose the Mean_Z and the Std_Z values based on our visual inspection of the jumps and the summary statistics.

We then tried automatic feature extraction using the ExtraTreesClassifier from the scikit-learn package[2] in Python. This package fits a number of randomized decision trees to subsets of the data to predict the most accurate features for prediction. The output of feature importance is shown in Table 6.1.

Table 6.1: Extra Trees Classifier

Mean_X	Mean_Y	Mean_Z	Std_X	Std_Y	Std_Z
0.12277463	0.05561651	0.0722577	0.30031766	0.16928672	0.27974671

The ExtraTreesClassifier denoted Std_X and Std_Z as the most significant features. The Acceleration feature plots, based on our visual and statistical determination (Mean_Z and Std_Z) and from the ExtraTreesClassifier (Std_X and Std_Z), are shown in Figure 6.2.



(a) Acceleration Features Mean_Z and Std_Z

(b) Acceleration Features Std_X and Std_Z

Figure 6.2: Acceleration feature plots

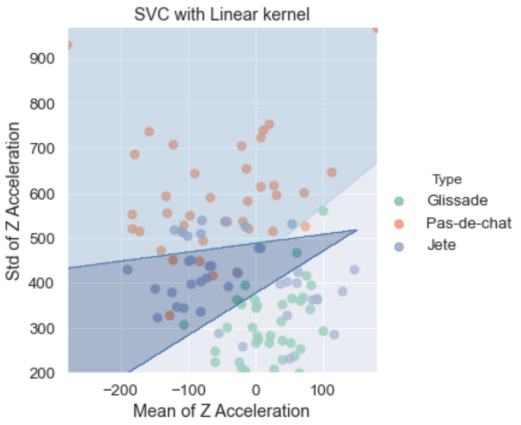
6.2 Model Training

Our training set was comprised of 39 data sets (of individual jumps) for each of the three jump types (Glissade, Pas-de-chat, Jeté), for a total of 117 data sets. We trained two sets of models (each set with different features, and using the SVM model with 3 different kernels. This gives a total of 6 models) to compare the accuracy of predictions between the model sets and which kernel performed best. Each set of models were trained on the same data sets.

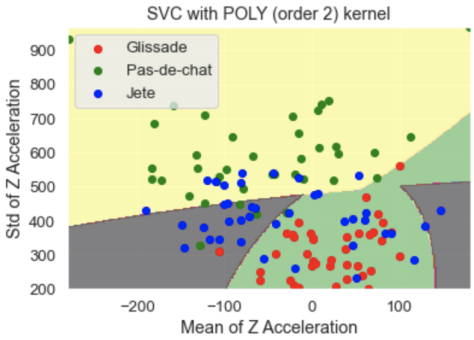
We modified the model parameters (C and gamma) to see which parameters worked best for our models. C is the error penalty parameter. A higher C value creates a higher penalty on the error and therefore a more strict classification prediction. A lower C value allows a wider range of error on the prediction and is aimed at classifying more points. Gamma is the kernel coefficient and it can be applied to the RBF and the POLY kernels. Both sets of models used the same parameters and are listed in Table 6.2. The plots from each of the models are shown in Figure 6.3 and Figure 6.4.

Table 6.2: Model Parameters

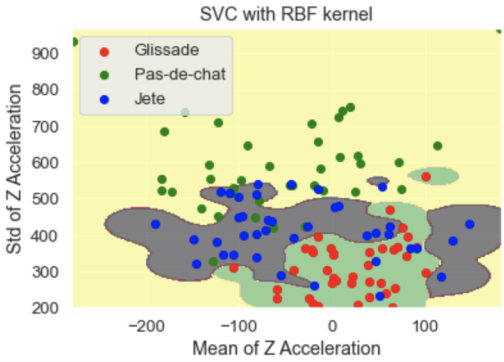
Features	Kernel	C value	Gamma value	Degree
Mean_Z, Std_Z	Linear	1	-	-
Mean_Z, Std_Z	POLY	1	auto	2
Mean_Z, Std_Z	RBF	1	0.001	-
Std_X, Std_Z	Linear	1	-	-
Std_X, Std_Z	POLY	1	auto	2
Std_X, Std_Z	RBF	1	0.001	-



(a) LINEAR Model

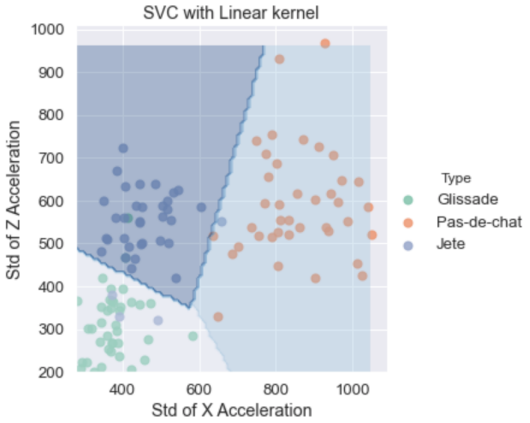


(b) POLY (degree 2) Model

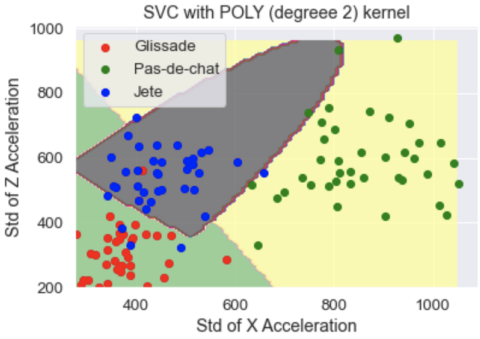


(c) RBF Model

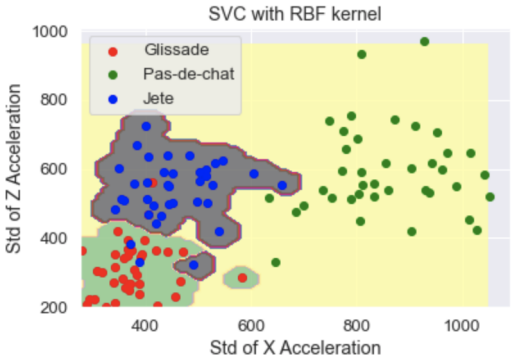
Figure 6.3: Model 1 plots using Mean_Z and Std_Z features



(a) LINEAR Model



(b) POLY (degree 2) Model



(c) RBF Model

Figure 6.4: Model 2 plots using Std_X and Std_Z features

6.3 Model Testing

We collected two test data sets. The first test set was collected from the same intermediate skill-level dancer that provided the training data set. 10 individual jumps for each jump type were collected for a total of 30 test sets. We then collected data from a beginner skill-level dancer. This dancer provided 10 jump for Glissade and Pas-de-chat, and 11 jumps for Jete, for a total of 31 jumps. Each test data set was processed following the same procedure as outlined for the training data set, including the feature extraction. Test sets were run through each model to test the model accuracy. The results are summarized in Table 6.3 and Table 6.4. The overall accuracy of the models is listed in Table 6.5.

Table 6.3: Model 2 (Std_X,Std_Z) Accuracy for jump type

Model 2 (Std_X,Std_Z)	Intermediate Dancer Data			
	Jump	Correct Prediction	Total Tests	Accuracy
LINEAR	Glissade	9	10	90.0%
LINEAR	Pas-de-chat	10	10	100.0%
LINEAR	Jete	9	10	90.0%
POLY (degree 2)	Glissade	9	10	90.0%
POLY (degree 2)	Pas-de-chat	10	10	100.0%
POLY (degree 2)	Jete	9	10	90.0%
RBF	Glissade	9	10	90.0%
RBF	Pas-de-chat	10	10	100.0%
RBF	Jete	6	10	60.0%
Model 2 (Std_X,Std_Z)	Beginner Dancer Data			
	Jump	Correct Prediction	Total Tests	Accuracy
LINEAR	Glissade	9	10	90.0%
LINEAR	Pas-de-chat	10	10	100.0%
LINEAR	Jete	8	11	72.7%
POLY (degree 2)	Glissade	9	10	90.0%
POLY (degree 2)	Pas-de-chat	10	10	100.0%
POLY (degree 2)	Jete	7	11	63.6%
RBF	Glissade	7	10	70.0%
RBF	Pas-de-chat	10	10	100.0%
RBF	Jete	6	11	54.5%

Table 6.4: Model 1 ($Mean_Z$,Std_Z) Accuracy for jump type

Model 1 ($Mean_Z$,Std_Z)	Intermediate Dancer Data			
	Jump	Correct Prediction	Total Tests	Accuracy
LINEAR	Glissade	10	10	100.0%
LINEAR	Pas-de-chat	6	10	60.0%
LINEAR	Jete	5	10	50.0%
POLY (degree 2)	Glissade	10	10	100.0%
POLY (degree 2)	Pas-de-chat	7	10	70.0%
POLY (degree 2)	Jete	7	10	70.0%
RBF	Glissade	9	10	90.0%
RBF	Pas-de-chat	9	10	90.0%
RBF	Jete	5	10	50.0%
Model 1 ($Mean_Z$,Std_Z)	Beginner Dancer Data			
	Jump	Correct Prediction	Total Tests	Accuracy
LINEAR	Glissade	8	10	80.0%
LINEAR	Pas-de-chat	2	10	20.0%
LINEAR	Jete	9	11	81.8%
POLY (degree 2)	Glissade	8	10	80.0%
POLY (degree 2)	Pas-de-chat	4	10	40.0%
POLY (degree 2)	Jete	8	11	72.7%
RBF	Glissade	8	10	80.0%
RBF	Pas-de-chat	7	10	70.0%
RBF	Jete	4	11	36.4%

Table 6.5: Overall Accuracy Table comparing results from all models. Results using all three jumps from the intermediate dancer.

Model 2 (Std_X,Std_Z)	Intermediate Dancer Data		
	Correct Prediction	Total Tests	Accuracy
LINEAR	28	30	93.3%
POLY (degree 2)	28	30	93.3%
RBF	25	30	83.3%
Model 1 ($Mean_Z$,Std_Z)	Intermediate Dancer Data		
	Correct Prediction	Total Tests	Accuracy
LINEAR	21	30	70.0%
POLY (degree 2)	24	30	80.0%
RBF	23	30	76.7%

7. Improvements

There are many future improvements that could be made to the model. To get better precision on the data, we would suggest increasing the FS of the accelerometer to 4G. We didn't realize, until after the data was collected, that the acceleration components of our current data set often saturated at the 2G level. Having the full resolution of the data would allow us to extract the features more accurately.

If we had more time, we would also collect more data for each jump. The current model is trained on ~ 40 individual repetitions of type of jump. It is almost always true that models usually benefit from more data. We would also try to gather training data from multiple dancers. This would help to avoid bias in the training data.

There are subtleties in the jumps that might not be detected when the sensor is attached to the dancer's ankle. We think that moving the sensor to the dancer's foot could allow some of the jump subtleties to be detected. We would also collect data from two sensors, one on each foot of the dancer, to detect the motion of both feet during the jumps. With the additional data at higher resolution, and with data from multiple sensors, we would then try to extract more features to train the model on.

In order to move the sensor to the foot, it would be necessary to improve the current sensor prototype, by using the small cradle and smaller battery available for the SensorTile. However, it requires a special soldering technique.

With more time, we would have also liked to deploy the selected model into a mobile App. This would allow the dancer to obtain live/immediate feedback if the ballet move was recognized or not.

Another step for the future would be to identify and classify different steps within a choreography. That is, to identified the classified movements done together with other transition steps.

Finally, we would like to explore other types of models. For our project, we explored 3 different kernels within the SVM model but it would be interesting to see how different models compare.

8. Applications

- To practice specific ballet moves.
- To automate choreography documentation, i.e. steps used in a choreography.
- Accurate and detailed measurement of a dancer's training volume. To analyze health implications. (see paper in references)
- Ballet competitions (gather statistics of specific moves or jumps).
- Figure skating competitions (gather statistics of specific jumps.)
- Choreograph lights to ballet steps.

9. Conclusions

Current Machine Learning techniques and sensor technologies can be used to classify ballet movements. This can have many different applications in ballet training, choreography documentation, choreography statistics, etc.

To generalize these results to many ballet movements, it would be necessary to use multiple sensors in different parts of the body. In addition, data needs to be collected from many dancers (even dancers from different levels - intermediate, advanced and professionals).

To increase data fidelity, it may be better to place small sensors on the feet of the dancer as opposed to the ankles.

Our test results show that the SVM kernel that provided the best accuracy was the POLY kernel. This was true for both sets of models that we tested, although the model using Std_X and Std_Z as features had the same overall accuracy when using the linear kernel. The two features that we initially selected manually (Mean_Z and Std_Z), gave an accuracy of 80.0%. The utilization of ExtraTreesClassifier, an automatic feature extraction procedure, that shows the feature importance, significantly improved our model accuracy. Using the output from the ExtraTreesClassifier, the accuracy increased to 93.3%

Bibliography

[1] SensorTile,
https://www.st.com/en/evaluation-tools/steval-stlkt01v1.html

[2] scikit-learn,
https://scikit-learn.org/stable/

[3] Firmware for data capture,
https://www.st.com/content/st_com/en/premium-content/sensortile-curriculum-fp-sns-allmems1_firmware_zip.html

[4] GitHub Repository,
https://github.com/alb4876/ballet-jumps

[5] "An Exploration of Machine-Learning Estimation of Ground Reaction Force from Wearable Sensor Data", Jan. 2020,
https://www.mdpi.com/1424-8220/20/3/740/htm

[6] "Development of a Human Activity Recognition System for Ballet Task", Feb. 2020,
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC700745

[7] Pandas,
https://pandas.pydata.org