Monty Matlab - Group 1 Project Documentation

Wisam Waad Noori Fatbardh Smajli Wantao Li Menayl Shah

I. Introduction

This documentation contains a summary of the work done by Group 1 within the scope of the Monty MATLAB project in SS20/21. The task of this project is to differentiate, or as we say in machine learning terms, classify between normal walks and silly walks. A silly walk is a very weird way of walking which was introduced in an aired sketch in the Monty Python television show. To the human eye the difference between a normal walk and a silly walk is very evident, and you would gather crazy looks doing the silly walk outside (speaking from experience). However, the challenge is to train an intelligent algorithm, in our case a machine learning model, to do the classification task for us.

II. Data Collection

The first and at the same time a very crucial part of this project was to gather data. This is the part where we had to actually leave the house with our phones in the pockets and go for long walks. The challenge here was to find an abandoned place to gather silly walk data. To record the data we use the data logging feature which the MATLAB Mobile application provides. We record the acceleration data, which consists of three vectors that describe the acceleration in x, y and z direction. When taking a closer look at the data, it can be seen that there is an apparent difference between the data containing the normal walks and the silly walk data. The normal walk data not only looks more periodic, but it also has much more spikes, i.e. highs and lows than the silly walk data.

A. Variations of data

To account for the fact that our model might need to predict previously unseen data, we decided to cover different kinds of variants for both classes of walks. In the case of the normal walk that means recording data with different walking speeds. For the silly walk we decide to record data using different variants of silly walk. This, in our opinion will greatly influence the prediction performance of our machine learning model and there is a higher chance that outliers can be predicted, as they might be in our recorded dataset. Figure 1 shows some of the different silly walks that we used to gather our silly walk data. This enables our model to detect different kinds of silly walk data, rather than being hard coded to just one silly walk. Below the three images in Fig. 1 that show our walks, we can see the resulting acceleration data from the respective walk above. Here, the walks are plotted for a time span of 20 seconds, such that we can compare the data visually. Apparently, the different walks provide us with very different acceleration data.

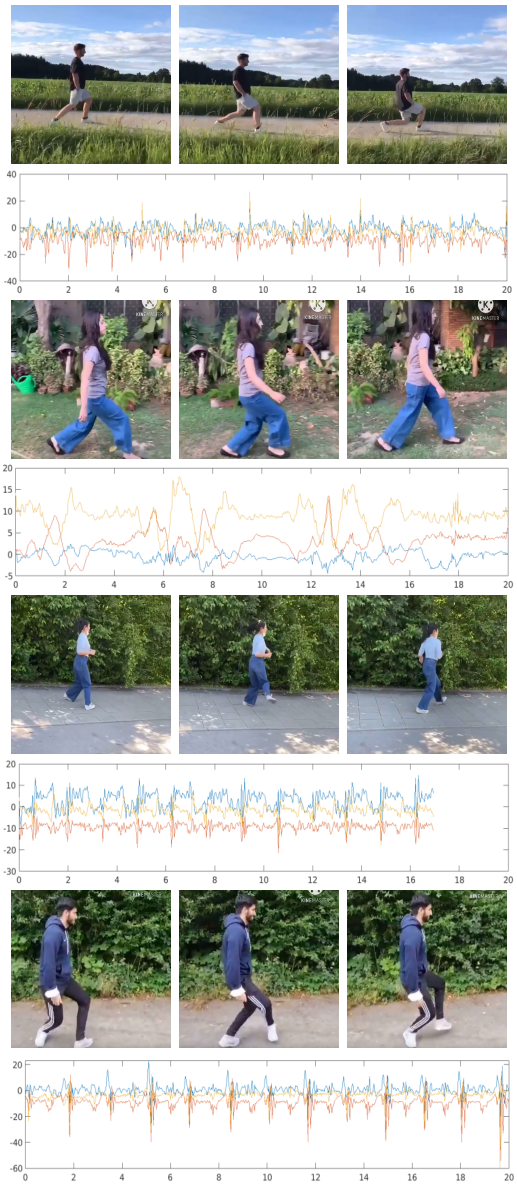

Fig. 1. Variations of silly walks and resulting acceleration data

B. Data distribution and evaluation metrics

Since it is much easier to collect normal walk data than silly walk data, it is not surprising that in the end our dataset had more normal walk data. To be accurate our dataset consists

of 57 minutes normal walks and 48 minutes silly walks. To take this imbalance of data into consideration, we evaluate our model's performance on different metrics. While accuracy is the most obvious metric to measure a model's prediction performance, it does not account for data imbalance. Therefore, we additionally consider precision and recall as evaluation metrics. Precision describes the ratio of true positives to true positives and false positives. Recall gives us the ratio between true positives to true positives and false negatives. It is evident that there is a trade-off between these two ratios.

III. Implementation

The second part of the project is concerned with the implementation, as we want our algorithm to do the classification part for us. This implementation consists of preparing the data, feeding the data into the model for training, and using the trained model for inference.

A. Extracting the data

An essential part of the implementation is getting the data into the right shape or form for training, i.e. having the data X in a cell array and the corresponding labels Y in a categorical array. This enables us to directly feed the processed data into the machine learning model.

The recorded logs consist of a class *Acceleration* which contains the recorded acceleration in x, y and z directions, as well as the timestamps of the recording. We concatenate the three vectors into a matrix called *data*. Since the timestamps are not necessary in our case, we discard those and instead create a time vector called *time* starting from 0 with a step size equal to the sampling rate at which the data was recorded. The *data* and *time* arrays are saved.

The *extractData.m* function takes the inputs *matFileContent*, *matFileName*, *targetSamplingRateHZ* and *windowLengthSeconds* and returns the above mentioned cell array X and categorical array Y as outputs.

B. Model architecture

We choose a long-short term memory (LSTM) model for classification for the reason that the LSTM is very powerful, it might be even too powerful for the task at hand.

Since we have three channels, x, y and z, the input size of the model is three and as this is a binary classification task ('Normal walk' or 'Silly walk'), the output size of the model is two. A normalization layer, a dropout layer with 50% dropout probability and a ReLU layer are added between the fully connected layers and the LSTM layer. Our model contains a hidden layer with 50 neurons, an LSTM layer with 25 neurons and a final fully connected layer with 2 neurons. This number was chosen after trying different values for the number of hidden neurons and finding that the above mentioned neuron numbers yields the best results in our case. We train the model for 25 epochs with a mini batch-size of 32. Both parameters can be changed arbitrarily, but we found that this does not make a huge impact, as long as the number of epochs is not lower than 25 and the mini batch-size is lower than 16.

Using the above mentioned model architecture and a reasonably large amount of data, we can train the model fairly quickly. However, we cannot directly observe problems that occur during training such as overfitting. To account for this and to have an evaluation metric during training, we have decided to split the training data into a training and validation set. This helps to directly see if there are irregularities like overfitting. We have tested different splits and came to the conclusion that a $65 : 35$ split into training and test set works best for our application.

To tackle the overfitting problem we have made several adjustments. Reducing the model size (number of hidden neurons), $L2$-Regularization, collecting more data, using a learning rate scheduler. We found that collecting more data made the difference for us.

Upon classifying the test data we can evaluate the model on data it has not seen yet. This shows how good the model's classification performance really is and returns the accuracy of correct classifications.

C. GUI

The GUI allows the user to train the model mentioned above with their own data and the freedom of changing the mini batch-size and the number of epochs. The training is visualized through plots of the accuracy and the loss over the epochs or during the training once it is finished or live ('live training progress'-button). To use this model for classification select the option 'trained'. Our pre-trained model is selected by the option 'provided'. The evaluation metric accuracy and balanced accuracy are given on the right panel as well as the confusion chart. The trained model can be exported.

IV. Performance and Evaluation of Model

To evaluate our model's performance, we first train the model for 25 epochs using the training data. The training accuracy and training loss converge to 93% and 0.2%, respectively. The hyperparameters and options used for the training are as follows: we use a batch size of 32 and an initial learning rate of 0.0001, which is decreased by the factor 0.9 every 10 epochs. We use a gradient threshold of 1 and $L2$-Regularization with a regularization factor of 0.015. The resulting evaluation metrics can be seen in the following table.

Accuracy	93.1417%
Balanced Accuracy	93.14%
Precision	91.82%
Recall	94.76%