Oscillation Classifier for a 2 dimension trajectory

Berger Julien, Mammou Rhita, Bertringer Maxence
ML4Sciences Machine Learning project, EPFL, Lausanne, Switzerland

***Abstract*—Exploring the collaboration behaviour between individuals especially the leader - follower phenomena is an interesting aspect in the computer human interaction.**

Hence, throughout a collaborative game, the goal of our ML4Science project is to see how players interact with each other without being able to talk. More specifically, we will have to detect the leader that will lead the group into winning the game via oscillations that he will make to call players into following him.
In this paper, we will explain how we pre-processed the data, what methods we used to classify players between leader or follower,expose the results, and finally, select the best model for this time series classification.

I. Introduction

The game we are dealing with is a game developed during a student semester project. It is played by 6 players that have to contribute together without communicating in order to push boxes together (boxes would not move if there aren't the proper amount of players pushing them) into a target point. Once all the players push all the boxes into the final target, they all win the game.
The data of this game is composed of time series positions for each player. The goal of our project is to analyze those trajectories in order to detect the oscillations of players that tend to represent the leader of the group, and thus classify each trajectory. In this case, the classification is binary: either we have an oscillation (1) or no oscillation (0).
For this purpose, we gathered the data of about 10 games of around 15 minutes each. The idea was to see how we can deal with time-series data, do proper data preprocessing and data augmentation, and then choose models that are relevant to trajectory binary classification[1]. Since several machine learning models are adapted to binary classification, we tested the most relevant ones, namely the logistic regression, the random forest classifier, the KNN classifier and also the LSTM neural network. For each model, we assessed the performance by computing the testing accuracy and the corresponding precision (TP/TP+FP). All the results were summarized using a confusion matrix. Finally, we selected the model that performed best for the classification of oscillations. This means that we selected the model which had both a high testing accuracy and a high precision. Precision is particularly important for this task because having low precision (i.e. having a lot of false positives), would lead us to believe that a player is a leader when he is not.

II. PRE-PROCESSING STEPS

The initial data that we have is composed of the trajectory of each player in each game, one data point (time, x-coordinate,y-coordinate) per second. Having not much games at first so not much data, especially not so much oscillation, we had to find a solution : so we finally decided to "fake" data, and play many games doing some oscillations. Then we had to label manually all the trajectories giving label 1 to a trajectory with oscillation and label 0 for one without. In order to pre-process this data, we decided to split the trajectory into sub-trajectories of 15 seconds each, so each data point has 15*8*2 (= 240) features corresponding to the x and y coordinates of the trajectory. The final feature array was thus composed of all sub-trajectories of all players of all games. In total we have 2582 data points. The 15 seconds cut was an arbitrary choice, but seems to be quite a good choice
In order to augment the data, we added the Fourier coefficients of the trajectories, but also the euclidean distances between two consecutive coordinates into the features, ending with around 470 features. Moreover, for each trajectory with an oscillation, we cut it in three part (left, middle, right) from which we create new trajectories : beginning from the middle of the left adding 15 seconds represents a another trajectory. We repeat this process with a shift of 1 second until the trajectory hits the middle of the right part. We did what we call a sliding windows between the oscillation so that we still have an oscillation but what is surrounding it is different.

III. FIRST MODEL : LOGISTIC REGRESSION

The first model we used is a basic logistic regression[2] to classify our trajectories into oscillation or not oscillation. After splitting the data into training and testing, we fit the model and compute the prediction probabilities for the trajectory to be an oscillation. We ended up with an accuracy of 68% and a precision of 56%, which is not good at all as one could expect.

In order to improve the performance of our model, we went through cross validation to find the best probability threshold (the standard one being 0.5). However we didn't find any better results. Nonetheless, we plotted the ROC curve in Figure 1 : performance measurement for the classification problems. It is plotted with True Positive Rate (the probability that the actual oscillation will be tested as such) against the False positive rate (the probability that the output is telling that there is an oscillation while there's not). In this direction, we computed the area under the curve which tells how much the model is capable of distinguishing between classes. So the Higher the AUC, the better the model is at distinguishing between trajectories with oscillations and without. We obtained a score of 0.68, which is not so bad, but is not enough and should definitely be improved.

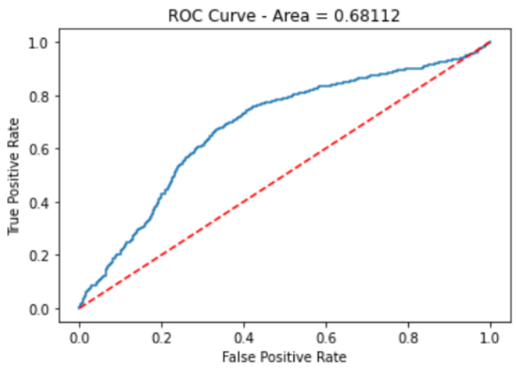

Figure 1. Receiver Operating Characteristic Curve

IV. SECOND MODEL : RANDOM FOREST

The random forest algorithm was proposed by Leo Breiman and Adèle Cutler in the 2000's and is therefore much more recent than the logistic regression.
In its most classical form, it performs parallel learning on multiple randomly constructed decision trees trained on different subsets of data. Instead of relying on a single decision tree, the random forest takes the prediction from each tree. The final output is then predicted based on the majority vote [3].
We trained and tested a random forest classifier with a selection of different features. The x and y coordinates of the trajectories, the Fourier coefficients of these trajectories, the distances between two consecutive coordinates, and all possible combinations of these features. The best results were obtained using the features corresponding to the distances between two consecutive coordinates and also by parameterizing the algorithm such that nodes are expanded until all leaves are pure.
We managed to reach a testing accuracy of 71% with this method. However, the precision was equal to 61% and was not satisfying in our opinion. Indeed, our aim is to minimize the number of False Positives and a precision of 61% means that the number of false positives is relatively high. The results can bee seen in Figure.
We thus turned to other models such as the KNN algorithm and neural networks[4].

V. THIRD MODEL : K-NEAREST NEIGHBORS

The K-nearest neighbours algorithm is a simple but powerful algorithm to make classifications. It classifies a new point on the basis of a majority vote between the K nearest neighbours. This means that a distance measure has to be defined [2].
In our case, we chose the Euclidean distance. The number of neighbours K is also a parameter that must be chosen with care. If K is too small, we obtain very complex decision boundaries meaning that the bias is small but the variance is large. On the other hand, if K is too big, we obtain simple decision boundaries meaning that the variance is small but the bias is large. To solve this bias-variance trade-off and choose the best K, we used cross-validation for different values of K.
As for the logistic regression and the random forest algorithm, we trained and tested the KNN algorithm on different features. That is, the x and y coordinates of the trajectories, the Fourier coefficients of these trajectories, the distances between two consecutive coordinates, and all possible combinations of these features. This time the best results were obtained using the x and y coordinates as features.

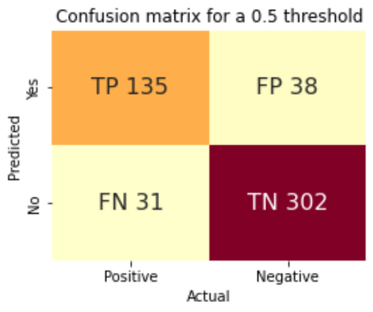

Figure 2. KNN Confusion Matrix with PCA

It should be noted that the KNN classifier does not perform well in high dimensions. Indeed, in high dimensions, the distances between neighbors are large and a large part of the feature space has to be explored to find neighbors, meaning that the locality between neighbors is lost and thus the performance of the model is reduced. This phenomena refers to the curse of dimensionality [2].
For this reason, we also tried to reduce the dimension of the feature space using the principle component analysis (PCA). Cross-validation was used to find the best dimensionality reduction and it turned out that 31 was the optimal feature dimension. In this feature space the optimum K was equal to 23. With these parameters the KNN algorithm performs particularly well with an accuracy of 86% and a precision of 79%. see Figure 2

VI. FOURTH MODEL : LONG SHORT TERM MEMORY

LSTM is a special category of artificial recurrent neural networks. LSTM had feedback connections, and can process not only one data point but an entire sequence (time series) of data. As it is presented in the Figure 3, a common LSTM unit is composed of a cell, an input gate, and a forget gate. The cell is able to remember values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

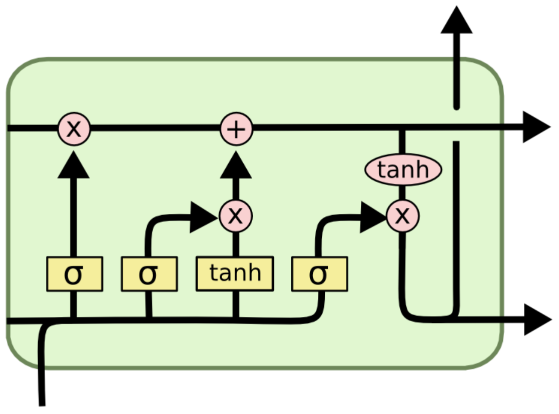

Figure 3. LSTM Architecture

Hence, we used LTSM for our time series classification, as there can be a shift of time between different oscillations and as it capture long-term temporal dependencies.
This model was implemented with two layers, then it was tested with different features dimensions : first with the x and y coordinates, and then with the additional Fourier coefficient, To select the best possible models, we tried with different hyperparameters.

We resumed the results in the two following tables (Figure 4 and Figure 5).

	Epochs	Batch size	Drop out	Recurent drop out	Number neurons	Threshold	Features used
LSTM 1	50	32	0.4	0.4	100	0.5	x, y
LSTM 2	50	32	0.4	0.4	100	0.5	x,y Fourier coefficien
LSTM 3	100	32	0.6	0.6	150	0.5	x,y+ Fourier coefficien
LSTM 4	200	32	0.2	0.2	150	0.5	x,y Fourier coefficien

Figure 4. The different LSTM models

Thus, the best results that we have is with the Simple LSTM 4 with 200 epochs, with the augmented features. We increased the number of epochs because it allows more training and less over-fitting. It performs with an accuracy of 83% and has a Positive Recall of 75%. It has also the best confusion matrix in Figure 6 :

	Final Loss	Accuracy	Positive Presision	Positive Recall	Negative Precision	Negative Recall
LSTM 1	0.394	79,66%	66,50%	78,53%	88,20%	80,23%
LSTM 2	0,358	79,76%	65,82%	87,64%	91,91%	75,53%
LSTM 3	0,3625	74,07%	65,97%	53,37%	77,26%	85,20%
LSTM 4	0.2065	83,84%	75,11%	83,90%	89,94%	83,81%

Figure 5. Results of LSTM models

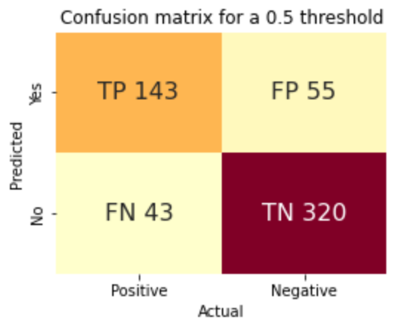

Figure 6. Best LSTM Model Confusion Matrix

VII. FIFTH MODEL : CONVOLUTIONAL NEURAL NETWORK AND LONG SHORT TERM MEMORY

The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence classification [?]. We defined a CNN LSTM model using Keras by first defining the CNN layer or layers, wrapping them in a TimeDistributed layer and then defining the LSTM and output layers.

The best model we found have an accuracy of 79% and a positive recall of 70%, which is obviously not what we aspire as a result.

We assumed that this model was tailored for more spacial structure inputs like images, or videos but not relevant to the trajectory classification task.

VIII. VALIDATION : USER DATA

In our implementation, we trained and tested the data we created. But then we had new data to do the validation part, so we used our two best models : LSTM and KNN.

With PCA and KNN, We manage to find a 100% match with the correct label of the trajectory.

PCA and KNN seems to be reliable

For KNN, we fit all the oscillations, and we have an accuracy of 84%.

IX. MODEL SELECTION

During this project, we examined many machine learning models suitable for binary classification. We moved from basic models like logistic regression to complex neural

networks like LSTM. As previously explained, we sought to find the model capable of predicting oscillations with the highest accuracy but also with the lowest possible false positive rate (high precision).
The logistic regression model had an accuracy of 68% and a precision of 56% which is not optimal at all. This shows the limitations of such basic models when dealing with relatively high-dimensional data.
The random forest algorithm performed better with an accuracy of 71% and a precision of 61%. The accuracy and the precision are are still not satisfying meaning that the random forest algorithm is not adapted to such a classification task. However, the KNN algorithm outperformed the logistic regression and the random forest algorithm in terms of accuracy and precision. Indeed, the accuracy was equal to 86% and the precision was equal to 78% which is what we hoped to achieve.
The last models we used were the LSTM and CNN LSTM neural networks. we obtained some better result with LSTM but not as good as KNN.

Based on the testing accuracy and precision, the best model is thus the PCA and KNN algorithm seems to be the best for our problem.

X. Conclusion

In conclusion, the goal of the project was to detect signal characteristics of a leader-follower behavior. More precisely, to detect oscillations that a leader player would tend to make so that other players follow him. In order to detect these oscillations, we trained and tested several models on 2D trajectories that we created and labeled manually. We selected the best model based on the testing accuracy and precision. Logistic regression and random forest did not give satisfying results as compared to KNN and LSTM. The big issue we had to deal with was by far the unbalanced distribution of true oscillation and non oscillation

XI. Python libraries

All the results of the report were obtained using python libraries. We used *numpy* to deal with arrays and *pandas* to display matrices and cope with them. For the visualization of the results, we used *matplotlib* and *seaborn*. Finally, for the neural network computations we used *keras* which is a deep learning API, running on top of *TensorFlow*.

Acknowledgements

We would like to thank Hala Khodr and Anastasia koloskova for their assistance during the project.

References

[1] A brief survey of time series classification algorithms | by alexandra amidon | towards data science. [Online]. Available: https://towardsdatascience.com/a-brief-introduction-to-time-series-classification-algorithms-7b4284d31b97

[2] Machine learning CS-433. [Online]. Available: https://www.epfl.ch/labs/mlo/machine-learning-cs-433/

[3] E. D. S. Lab. Applied data analysis (ADA). [Online]. Available: http://dlab.epfl.ch/teaching/fall2020/cs401/

[4] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," vol. 33, no. 4, pp. 917–963. [Online]. Available: http://arxiv.org/abs/1809.04356