Class Project 2: Market states prediction

Joshua CAYETANO-EMOND: joshua.cayetano-emond@epfl.ch
Lorenzo GERMINI: lorenzo.germini@epfl.ch
Benoit PAHUD: benoit.pahud@epfl.ch
School of Computer and Communication Sciences, EPFL, Switzerland
December 17, 2020

***Abstract*—Trends in daily stock price movement were used to classify individual days into similar clusters using the Louvain algorithm. The result was a time series representing the daily market state. Three different supervised time series multivariate classification models, namely LSTM, random forest and TCN, were then used to attempt to predict the next day's market state using the previous days as features. Performance of the models were low, though this was likely due to a non-deterministic classification process. Nonetheless, preliminary results show that the models consistently performed better than a Markovian process of first order, indicating some longer-term memory of the markets.**

***Keywords*-Market states, Machine Learning, Long-Short Term Memory (LSTM), Random Forest, Neural Networks, Temporal Convolution Networks (TCN)**

NOTATION

A	Accuracy
F_1	F1 score
$\mathcal{L}_{\text{nll}}$	Negative log-likelihood loss
LSTM	Long Short-Term Memory
$n_{samples}$	Number of samples in the test
P	Precision
R	Recall
RF	Random Forest Classifier
TCN	Temporal Convolutional Networks
y	True market state
ŷ	Predicted market state

I. INTRODUCTION

Financial markets have long been studied and generate a lot of interest for the scientific community, as it generated interest for the authors of this paper. As such, a suitable project in the field was chosen and performed with collaboration of Professor Damien Challet and the *Laboratoire de Mathématiques et Informatique pour la Complexité et les Systèmes* from École Centrale Paris.

Predicting the evolution of individual stocks might still be beyond the realm of possibility, especially with limited computational resources; however, considering the market as a whole and predicting its state remains an interesting possibility. By looking at the general allure of the market, states can be found from which trends can be isolated, such as the well-known knowledge that gold and silver go up when other commodities go down [1].

The objective of this project is therefore to delve deeper into this market state analysis. Market trends are already known to behave similar to a Markovian process of order one, although it has already been proposed that there is a longer memory to markets, on the order of 50 days [1]. It is therefore hypothesized and hoped that by considering a longer window, machine learning could outperform such a simple Markovian system.

II. METHODOLOGY

Python 3.7.7 was used as the programming environment. The following packages were also used:

- Keras 3.1.1
- Keras-tcn 3.1.1
- Keras-tuner 1.0.2
- Matplotlib 3.2.2
- Numpy 1.17.0
- Pandas 1.0.5
- Pyarrow 1.0.1
- Python-louvain 0.14 [2]
- Scikit-learn 0.23.1
- Seaborn 0.10.1
- Sklearn 0.21.1
- Tensorflow 2.3.1

A. Data preparation

As a first step, the data had to be prepared into a usable format that could then be used to perform machine learning with. This was separated in two steps: data generation and preprocessing.

1) Data generation: The first step of the project is to generate the data, since the raw data only gives the (relative) daily change in value for around 1000 stock tickers. However, prediction of the daily change for individual stocks is not the interest of this project. As stated in the introduction, the analysis of the overall market states for given days is instead what will be used as data.

To generate these states, the Louvain algorithm was chosen and code provided by Pr. Challet was directly used. This algorithm looks at the daily changes in value of a certain quantity of stocks for a given number of days. It then compares all of the days in the range and groups days that had similar daily changes in value for the stocks into clusters.

To perform this, only the stocks without any missing values for their daily changes were taken. This resulted in close to 500 stocks to consider over approximately 4700 days. As a particularity, the Louvain algorithm only works when there are less days than stocks, and works best when the number of days taken into account is ~ 0.3 the number of stocks. Consequently, the Louvain algorithm was run

multiple times, taking windows of 150 days at a time and determining the state of the days in that window.

2) Data preprocessing: There is one major caveat with the Louvain algorithm. While it allows for rapid classification of the days in the considered window, it is non-deterministic and the labels assigned to the clusters are not necessarily given in any particular order. As such, a certain day may (and most certainly will) be assigned different labels if the algorithm is ran more than once. This also means that if adjacent windows of 150 days are taken, there is no certainty that the given labels would correspond from one window to another.

To counter this problem, the Louvain algorithm was run incrementally for every day, using that day and the 150 days preceding it, giving a state for all of the days in that window 151 day window. This was then repeated for the next day, where that day and all 150 days preceding it were considered. This lead to approximately 4500 windows of 151 days each, with strong overlap between all of the considered windows.

Once all of the states were obtained, it was possible to compare a window's label attribution to a nearby window. That is, it could be possible to compare the window containing days 150 to 300 and the window containing days 151 to 301, identical except for a single day. For each of these windows, the members belonging to each state for both windows could be obtained and compared. For example, if in the window 150 to 300, state 3 was given to days $\{152, 155, 159, 173, ...\}$ and in the window 151 to 301, state 0 was given to days $\{152, 155, 159, 172, ...\}$ then they likely correspond to the same label. As can be seen, due to the stochastic nature of the model, this correspondence is not perfect. To counter this, additional nearby windows are compared against one another and the best match is taken. It was found that comparing 10 nearby windows provided a good tradeoff between speed and quality of the data.

Following this, the possibility of new states emerging had to be kept, as there may be new market states which appear at a later stage that were not initially available. This was done by looking at the label of the new considered day in the current window (in the above example, day 301 in the second window) and seeing if it was labelled uniquely in its window. If it was, it was classified as a new state.

Finally, the data was arranged in a format that could be used for machine learning. A dataframe was created where each row represented a certain day. The state of 150 previous days' labels and of that day were given as features for that row. Then, the state of the next day was obtained from the next row and used as the output to predict for that row. With this done, the problem basically boils down to a supervised time series multivariate classification, with up to 151 features to consider in order to give the label (class) of the next day. Further preprocessing could then be done on the data to prepare for the machine learning models.

B. Selected Machine learning Models

To perform the machine learning, three different models that are appropriate for time series prediction were considered. These were also compared with a baseline scenario using a first order Markovian process

1) Markovian Classifier: The Markovian classifier was the baseline scenario against which all other models were compared. The classifier follows the first order Markov principle which consists to predict the next day's state using today's state and known transitions probabilities. To do so, the weights for each label that comes following another label were computed [3]. Once the weights were known, the following day's label were randomly predicted using a normal distribution based on the frequency weights of each label accordingly to the current day label.

2) Random Forest: The random forest classifier, proposed by Brieman [4], has been successfully applied to time series forecasting [5]. RF is an ensemble learning method that combines trees and random vectors which govern the growth of each tree in the ensemble [6].

3) LSTM: Long-Short Term Memory (LSTM) networks are a type of recurrent neural network (RNN) that can be used to make time series prediction. RNNs suffer from the issue that they have a short-term memory due to the vanishing gradient problem and have a hard time carrying information from earlier time steps to later ones [7]. LSTM solves this by adding internal gates that can regulate the flow of information, allowing for longer term memory. This model is interesting for data series with a potentially long memory, such as markets have been proposed to have.

4) TCN: Convolutional neural networks (CCNs) have been shown to be valuable tools for sequence modelling and time series forecasting in the context of deep learning. Particularly, they have been proven to exhibit longer memory than recurrent architectures with the same capacity and perform consistently better while avoiding typical downsides than more traditional recurrent neural network architectures such as LSTM [8]. A Temporal Convolutional Network (TCN) architecture was utilized, similar in the one presented in [8], which consists of a stack of dilated causal convolutional layers with the same dilation factor and an additional dense layer at the end. To assure full history coverage, a full receptive field equal or greater than the memory was always used.

C. Python code

The facilitate testing, the python code was implemented in the following way. A main file `run.py` was created from which all of the tests were be performed. This script prompts for a model to run, as well as the number of days to consider in memory. This memory value dictates the shape of the input vectors for the machine learning models. These parameters are then be passed on to the respective model file, which perform the machine learning training and prediction. The predictions of the model as well as the true labels are then be sent back to the main file which prints out the

performance indicators and save them in a file. This structure can be seen in Figure 1.

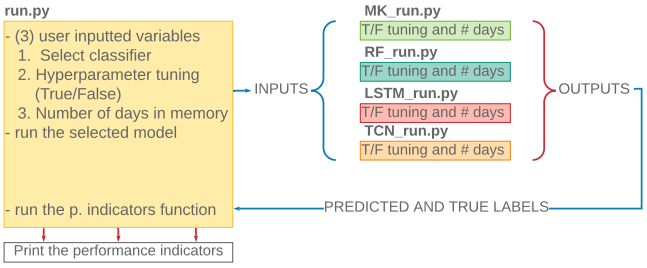

Figure 1. Python code flow

III. Models Training/Evaluation

A. Hyper-parameter tuning

Before starting the training process of each models, the best practice is to select the hyper-parameters that will make the models more efficient on the provided data [9]. Each model has its own hyper-parameters that were tuned. Table I covers all the hyper-parameters that were tuned including their search spaces.

As the hyperparameters were varied, the models were trained with the first 80% of the data as training data and then remaining 20% of the data was used for validation. Due to the data being a time series, true cross-validation was not possible, as the order of the data had to be preserved.

Table I
Hyperparameters search spaces

Model	Hyperparameter	Distribution	Range
RF	Bootstrap	Specific	True, False
	Max_depth	Discrete	10 : 110
	Max_features	Specific	Auto, sqrt
	N_estimators	Discrete	200 : 2000
	Min_sample_split	Discrete	2 : 12
	Min_sample_leaf	Discrete	1 : 4
LSTM	Number of layers	Discrete	1 : 3
	Unit number	Discrete	64 : 256
	Learning Rate	Discrete	1e-2, 1e-3, 1e-4
	Activation function	Specific	Sigmoid, tanh
TCN	Number of filters	Discrete	8 : 64
	Kernel size	Discrete	2 : 7
	Dropout rate	Continuous	0 : 0.7

1) Random Forest: The hyper-parameters tuning was made in two separate steps. The first step included a wide range grid search to define a potential zone for a more exhaustive search. Once the zones were defined, a random search was implemented on the selected range. The random search was chosen as for hyper-parameters optimization it has been proved to be more efficient and less time consuming than the classic grid search [10].

2) LSTM: The hyperparameter tuning for the LSTM method was done using the keras-tuner library, which provides an automated method for hyperparameter testing. More specifically, the Hyperband algorithm was used, as this has been shown to be faster and less computationally intensive than other methods [11]. The main parameters of interest for the LSTM method were the number of layers and unit number.

3) TCN: The hyperparameter tuning for the TCN model was the same as the one described above [11]. The main parameters of interest after various trials were the number of filters, kernel size and dropout rate. A coarse hyperparameter optimization was first performed followed by a finer one among the most promising values.

B. Prediction

Once the optimal hyperparameters were selected and the models trained, the predictions were made. This was done by using the running the model on the validation dataset. This method generated rapid predictions; however, it is not the optimal or best way to perform prediction. In reality, new data from the market would arrive on a daily basis and provide more information to better improve the model. The model would therefore be retrained every day with the new data that is available. To simulate this, a similar scenario would have to be emulated. This is commonly referred to as backtesting [12]. Unfortunately, this method can be rather expensive and computationally expensive, as a new model is retrained for every data point in the validation set. Nonetheless, the method outlined in this report was still sufficient to compare the methods to a fair degree.

C. Performance indicators

The best practice to evaluate a multi-class classifier is to use a basket of performance indicators to capture the whole outcomes and the nature of the data. Normally, Cohen's Kappa Statistic is one of the best metrics for evaluating multi-class classifiers on imbalanced data sets. Nonetheless, since the states of the market do not have an inherent distance related between them, it is not relevant to measure the proximity of the predicted classes to the actual classes as the Kappa indicator does.

With the production of the confusion matrix,the accuracy (A), prediction (P), F1 score (F_1) and recall (R) can be computed for each model. The equations are based on the scikit-learn library as following [13]:

$$\mathrm{A}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1\left(\hat{y}_i = y_i\right) \tag{1}$$

$$P(A, B) := \frac{|\ A \cap B\ |}{|A|} \text{ for some sets A and B} \tag{2}$$

$$R(A, B) := \frac{|A \cap B|}{|B|} \tag{3}$$

$$F_1(A, B) := 2 \cdot \frac{P(A, B) \times R(A, B)}{P(A, B) + R(A, B)} \tag{4}$$

F1 score is a weighted harmonic mean of precision and recall normalized between 0 and 1. An F1 score of 1 indicates a perfect balance as precision and the recall are inversely related.

IV. Results

The performance indicators were computed for each of the model and memory configurations. The results are summarized in Table II, with the best performances for each model being bolded.. Additionally, the accuracy and F1-scores are represented graphically in Figures 2 and 3, where they are compared with the baseline Markovian process.

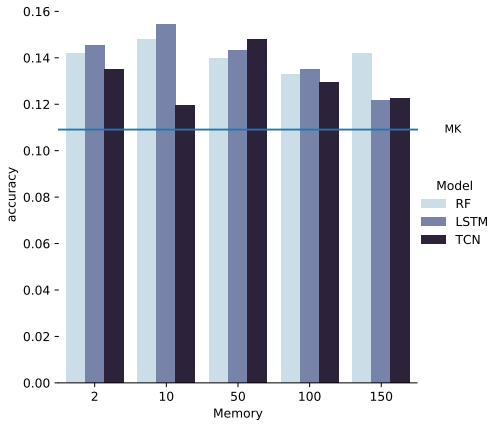

Figure 2. Accuracy for the three models

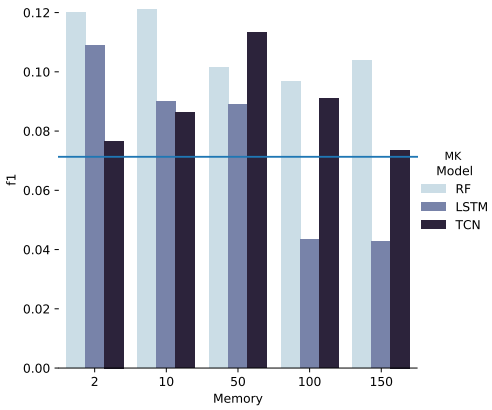

Figure 3. F1-score for the three models

The results in Table II and Figures 2 and 3 can seem rather poor at a first glance. Indeed, the highest accuracy achieved is a mere 15%, and precision suffers from a very poor score of around 7%, on average. While this can seem disappointing, proper care must be exercised. Indeed, the preprocessing performed in this project has been stated to be non-deterministic and hard to correlate when done multiple times. This has likely caused the data used to be less predictable than reality. Additionally, some of the market states identified may be correlated between one another. If the correlation between these states is established, then a better indicator such as the Kappa indicator described above could be used. As of now, the models are heavily penalized as soon as an incorrect prediction is made, even if the predicted state might be similar to the true state.

Table II
Models performance comparison.

Model	Performance Indicators	Number of days in memory				
		2	**10**	**50**	**100**	**150**
Markovian	Accuracy			**0.11**		
	Precision			**0.06**		
	F1-score			**0.07**		
	Recall			**0.11**		
RF	Accuracy	0.14	**0.15**	0.14	0.13	0.14
	Precision	0.11	0.12	0.11	0.10	**0.13**
	F1-score	0.12	**0.12**	0.10	0.10	0.10
	Recall	0.14	**0.15**	0.14	0.13	0.14
TCN	Accuracy	0.14	0.12	**0.15**	0.13	0.12
	Precision	0.06	0.08	**0.11**	0.08	0.07
	F1-score	0.08	0.09	**0.11**	0.09	0.07
	Recall	0.14	0.12	**0.15**	0.13	0.12
LSTM	Accuracy	0.15	**0.15**	0.14	0.14	0.12
	Precision	**0.13**	0.12	0.10	0.03	0.03
	F1-score	**0.11**	0.09	0.09	0.04	0.04
	Recall	0.15	**0.15**	0.14	0.14	0.12

Still, important trends can be derived from the data. It can be seen that generally on the accuracy the three models perform similarly with RF and LSTM slightly outperforming TCN. Concerning the F1-score the variances between the models' performance widens with the RF showing best results on average followed respectively by TCN and LSTM. Overall RF, because of the edge in the F1-score indicator, seems to achieve better results than the other two models but not by a very significant amount. In this case study, contrary to what is suggested by research [8], TCN does not have a statistically better performance than the LSTM model.

It is also interesting to note that models seem to improve as the number of days in memory is kept. There does however seem to be a peak in performance at around 50 days kept in memory for all the models. Some of the models, like the LSTM, even drop sharply in performance afterwards. This supports the idea that the markets have a memory of the order of around 50 days [1], as opposed to a first-order Markovian memory that is sometimes proposed. This is further supported by the models consistently performing better than the Markovian process simulated.

V. Summary

The aim of the project was to predict the next day's market state using three different machine learning models. The main challenge faced during the project from the beginning until the end was the process of the Louvain generation states into a supervised machine learning data. The low performance of the classifiers demonstrated that the high stochastic and randomness response of the market from day to day is yet hard to predict. However, with a proper way to process the data to keep the state labels time-wise, the performance will no doubt be greater. Nonetheless, the models were shown to consistenly perform better than a first-order Markovian model, indicating a long-term memory of the markets.

REFERENCES

[1] M. Marsili *et al.*, "Dissecting financial markets: sectors and states," *Quantitative Finance*, vol. 2, no. 4, pp. 297–302, 2002.

[2] T. Aynaud, "python-louvain x.y: Louvain algorithm for community detection," `https://github.com/taynaud/python-louvain`, 2020.

[3] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla, *Markov Logic*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 92–117. [Online]. Available: https://doi.org/10.1007/978-3-540-78652-8_4

[4] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[5] X. Qiu, L. Zhang, P. Nagaratnam Suganthan, and G. A. Amaratunga, "Oblique random forest ensemble via least square estimation for time series forecasting," *Information Sciences*, vol. 420, pp. 249 – 262, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025517309076

[6] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[7] M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation," , 2018.

[8] S. Bai, J. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 03 2018.

[9] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[10] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[11] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[12] J. CHEN, "Backtesting," , 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.