Euklid Project

Data science in action project

Github Repository: Link (https://github.com/AndreaBuscemi/EUKLID-ITALIANO-DS-Project) if it's not working

Team:

Andrea Buscemi

Alessandro Ponzianelli

Luca Schisano

- Presentation of the topic
- Reasons of the choice
- Methods
- Conclusion
- Peer evaluation

Presentation of the topic:

For the Data science in action course our team chose the project presented by Euklid. The project proposed by the company is based on the study of the historical series of some financial assets provided by the company (such as gold, IBM, etc.) with the aim of developing forecasts on the future trends of these assets and thus creating investment strategies capable of being profitable for investors.

Challenges

Design an algotrading strategy based on historical series.

Solution

AI based approach to predict prices of commodities, indices and stocks

Results

Aggregation of different and independent methods from scientific research.

Reasons of the choice:

The project, presented by the company on February 15, was immediately chosen by our team as it was considered highly stimulating and interesting by all members, above all by the challenge posed by the company.

The choice of the project proposed by Euklid was also moved by the convergence in this theme of interests and knowledge common to all three members, therefore moved by the desire to get involved and test their previous knowledge on that argument.

In this report, the team therefore sets itself the goal of making a detailed analysis of all phases of the work, reporting all the processes that we had to apply to carry out our analysis, as well as analyzing the results obtained.

Methods:

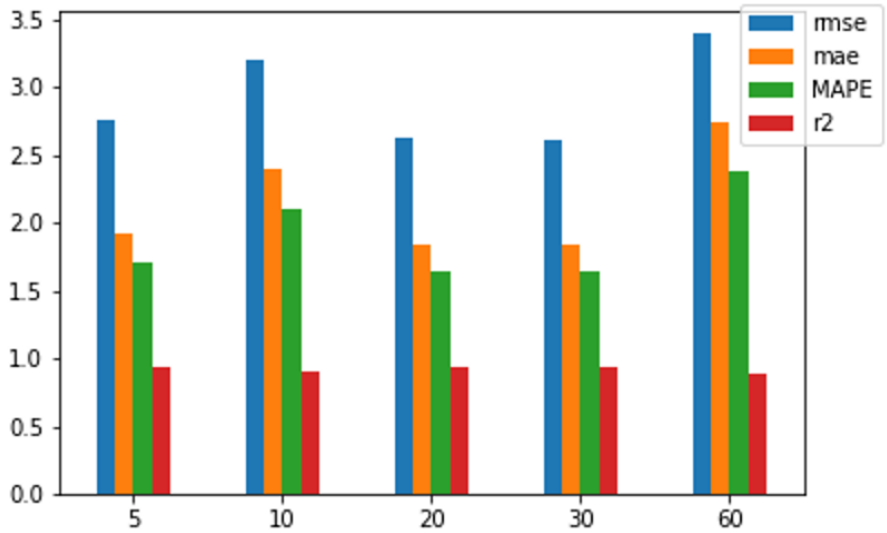
For the project the team chose to work on three different algorithms: **Support Vector Machine**, **Long Short Term Memory** and **Random Forest**; creating afterwards an ensemble method capable of putting together all the predictions made by the three algorithms to deliver a better performing strategy.

LSTM MODEL

LSTM (Long Short-Term Memory) is a Recurrent Neural Network (RNN) based architecture that is widely used in natural time series forecasting.

After a prediction is made, it is fed back into the model to predict the next value in the sequence with each prediction, some error is introduced into the model.

Data Preparation before a univariate series can be modeled, it must be prepared.
The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations must be transformed into multiple examples from which the LSTM can learn.
We need to divide our data into train and test data. The train data is 70% of the data from 31 December 1999 to 11 February 2022, about 3896 observations and the test data had 1700 observations. The first step is: standardize the data, a very important passage especially for artificial neuron network, so we standardize them into a range 1/0. Then created the sequence of observation needed to be fed by the LSTM. In order to decide how many days our model should look back, so the number of items in our sequence to pass in order to predict the output, we created different models for 5, 10, 20, 30 and 60 days, with the same architecture of the LSTM analyzing different metrics: 'RMSE', 'MAE', 'MAPE' and 'R^2'. At the end we select the number of days that our model should look back thanks to the model that show the best metrics.



For example, in the analysis with just the price close we can see how choosing 20 days back could be a good strategy because our model has the lowest 'rmse', 'mae' and mape comparing to others and a relative high r^2 so in this case the choose to look back

just twenty days do not affect in a negative way our future analysis moreover if we look back few days our computational time will be shorter.
Then we need to select how many epochs our model should be trained. To accomplish this task, we consider both the analysis of the mean absolute error as a metric and the computational time that a computer needs to process the epochs.
For example, continuing with the same example as before since the metrics shows that with 30 epochs our model increases the accuracy and also seeing that computing 20 or 30 epochs the time needed do not increase so much, we can afford to set up a model with 30 epochs.
We have done the same passages, finding different results in terms of number of sequence and number of epochs, for different dataset.
The first one is a dataset with just the price close to extract some information to predict the price close. In the second dataset we used the 'adj close' 'rolling mean 10 days' and 'rolling mean 50 days'. In the third dataset we have used all the features given by yahoo finance merged them with some indicators and thanks to a correlation matrix and a heatmap exclude the features that were correlated the most. In the last dataset we have just computed a technical study with only indicators to predict the price.
Then to compare all the dataset finding the best way to predict, thanks to a LSTM algorithm, the new price stocks we have just confronted the accuracy of every model over the four datasets and choose the best one with the accuracy higher.

'Dataset 3' Code:
I import a csv file with the indicators calculated before thanks to the talib library and merged them with the analysis of the stock's price given by yahoo finance. The thanks to the corr() function we can analyze which features are the most correlated and decide to exclude them. At the end we decide to keep only some features. Then we divide the dataset into a train and test data, the train test is scaled thanks to MinMaxScalare into number between 0 and 1 subsequentially we need to create a sequence from whom our model is trained. So we create a function with global 'y' and 'x' that takes for example

the firsts 5 days features and add the in the ‘x’ variable and the sixth in the ‘y’ variable then it takes the days from 2 to 6 and add them in the ‘x’ variable and the seventh in the ’y’ variable and so on; at the end we will have a X variable that contains all the sequence and as many columns as the features are and a ‘y’ that contain just a column with only the price close values for every day. Then we analyzed how many days look back using the same structure of the LSTM just 4 layers the first one with 100 neurons the second one with 20 neurons the third one with 25 and the last one with the output, so just a neuron. Then we moved to the test data, and we need to create also a sequence here, so we take the last values minus the number of looking back that we are considering in this analysis and create a sequence for the ‘x’ test. At the end we predict the results on this ‘x’ test and analyze them. Regarding how many days our model should look back we just use the inner function of keras to study the flow of the error through the epoch and the datatime library to find how much time several epochs need to be processed.

SVR MODEL

As an additional algorithm for the study the team decided to use a SVM.
the choice was first dictated by the members' knowledge of the algorithm itself and second by the characteristics that this algorithm possesses.
In fact the underlying motivation for using SVMs is the ability of this methodology to accurately forecast time series data when the underlying system processes are typically nonlinear, non-stationary and not defined a-priori. SVMs have also been proven to outperform other non-linear techniques including neural-network based non-linear prediction techniques such as multi-layer perceptrons.

To apply the SVM model, the team followed the following path.

- In the **first step** we recalled all the libraries necessary to read our dataset, which was provided in excel format, and all libraries necessary for the model to be applied (numpy, pandas, sklearn, seaborn and matplotlib).
- The **second step** concerns the reading of our dataset through the read.excel function and data pre-processing. In the same step, the columns of the dataset provided to be used were also selected; the first two columns of the dataset concerning the daily data of the various assets, from 31/12/1999 to 01/01/2021 ['IBM DAILY DATA', 'Price close'].

```
df = pd.read_excel("/Users/ale/Desktop/ML/Historical Seri
df.head()
df.columns

df = df.iloc[:,:2]
df.columns=['IBM DAILY DATA','Price close']
df = df.drop(0, axis=0)
df['Price close']=df['Price close'].astype('float')
```

- The **third step** of the process concerns the split of the data into train [4000:] and test ([4000:]) and the creation of a function capable of scrolling along the chosen daily data in order to make price predictions through the analysis of a 30-day "window" of data.
- The **fourth step** was designing the SVM model and applying it to train and test to make the prediction.

  The optimization of the model and therefore the tuning of the hyperparameters was made through empirical tests.

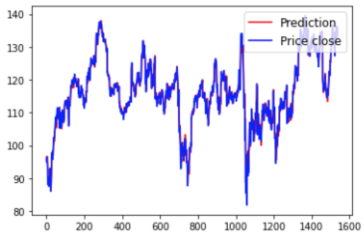
```
# design svm model
clf = SVR(kernel="rbf", C=1000.0, degree=3, epsilon=0.1, coef0=1, verbose=2)

# fit model
clf.fit(X_train,y_train)
```

Finally we study the accuracy of the svm model through the study of different indices and then we plotted our results.

```
1 # calculate RMSE
2 rmse = sqrt(mean_squared_error(y_test, predict_y))
3 print('Test RMSE: %.3f' % rmse)
```

```
# calculate MAPE
mape = np.mean(np.fabs((y_test - predict_y ) / y_test)) * 100
print('Test mape: %d' % mape)
```

```
# plot
pyplot.plot([x for x in range(1, predict_y.shape[0]+1)], predict_y, linestyle='-', color='red', label='Prediction')
pyplot.plot([x for x in range(1, y_test.shape[0]+1)], y_test, linestyle='-', color='blue', label='Price close')
pyplot.legend(loc=1, prop={'size': 12})
pyplot.show()
```



Random Forest Model

Another model developed by the team was a Random Forest Model (RF).

The RF was chosen as it was very often cited in the literature as one of the best models for stock market prediction. Especially when the goal is framed as a classification algorithm capable of predicting, based on previous day data, if the price is going to go up or down (so traditional 1/0 classification).

The model was developed exploiting many different libraries: *pandas, numpy, seaborn, matplotlib and sklearn*.

We fetched the data from Yahoo Finance through API calls already splitting every ticker data (we used 4: IBM, Oil (CL=F), Gold (GC=F) and the Nasdaq (^IXIC)) into two different datasets, one for train (from 31/12/1999 to 01/01/2021) and one for test (from 02/01/2021 up to today).

The *apply_strategy* function

While developing this model we also thought it was necessary to build our own evaluation function, capable of assessing the different models through good-for-all measures immediately giving us the main results of every model once applied into stock market trading strategies (the function called “apply_strategy”).

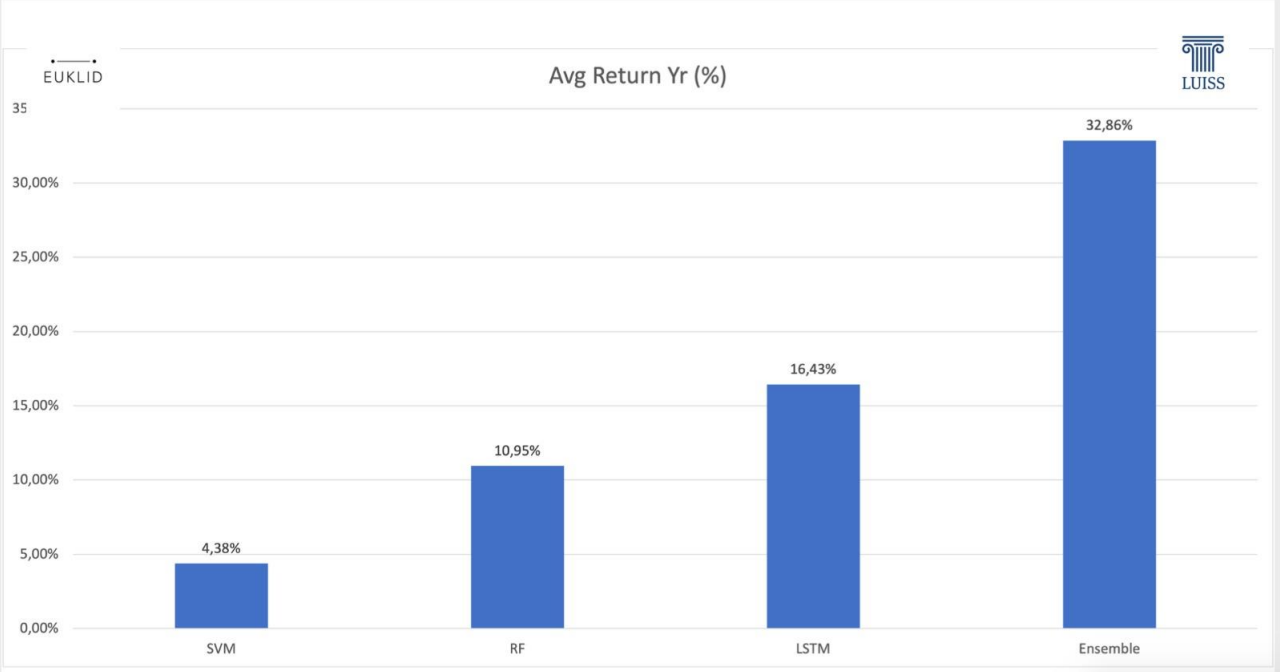
It is a simple function but since it was needed to be capable of being applied to every model without needing to fine-tune every time the data we were feeding into it, it could seem a bit cumbersome.

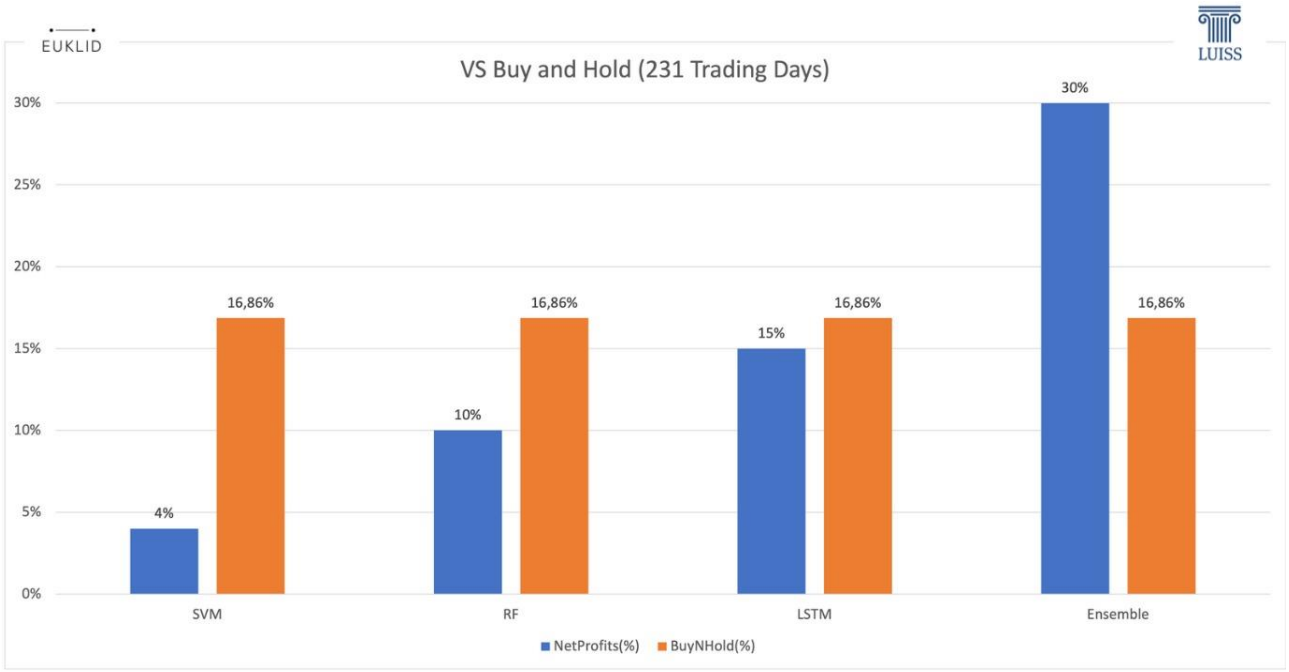
After some preliminary data pre-processing (removing useless columns and those too much correlated through the use of a heatmap), we used the GridSearchCV function to identify the best possible parameters for the RF.
It required a very long time to give an output (more than 2 hours) as we gave them a broad range to try every possible combination from.
The resulting best parameters were then selected and used to train the model on the IBM train dataset.
The resulting model was then used to predict all of the test dataset and the aforementioned apply_strategy function was used to assess the performance (you will find also some experiments we tried of moving with a step of 7 days, tragically unsuccessful).
We then finally plotted the resulting equity graphs.

The Ensemble (IBM)

The method we chose to ensemble the three algorithms is a simple one, we took each model prediction (so for each day either 1 or 0) in a common timeframe (to) in order to create a very simple but effective voting mechanism: for each day, if the sum of the three predictions is >= 2, then the prediction for the day would be 1 (hence the ensemble predicts the price is going to go up the following day), if the sum is > 2, then the model predicts a decrease in price.

The results were a very pleasant surprise! By joining all of the algorithms together, not only did we achieve a better performance than any of the other three taken individually, but the performance was actually even better than the sum of :the three algos performance! This is probably due by some intrinsic holistic propriety in the ensemble mechanism.





Peer Evaluation:

Andrea: Random Forest, apply_strategy function and ensemble

Alessandro: Support Vector Machine

Luca: Long Short Term Memory