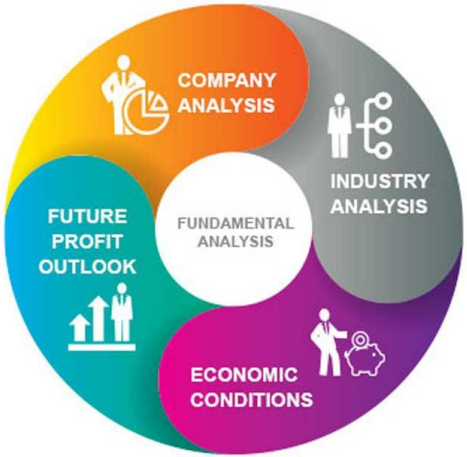
Distributed Stock Price Forecasting

Leonardo Emili (1802989)
Alessio Luciani (1797637)

Introduction

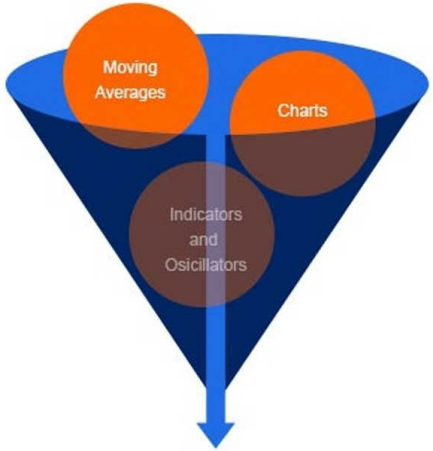
- Stock price forecasting aims at predicting the value of a stock in the future
- Stock trend affected by multiple factors (e.g. context, sudden events)
- Fundamental analysis vs technical analysis

Fundamental Analysis



VS

Technical Analysis



Source: https://fxcomfort.com/

Our approach

- Forecast adjusted close price of the following day based on past days
- Ticker-agnostic formulation
- Enrich historical prices with fundamental data

Our setting

- Apache Spark as distributed computing framework (i.e. PySpark)
- Data wrangling operations through high-level interfaces (e.g. Spark SQL)
- Design and implementation of recurrent neural networks using deep learning framework (i.e. PyTorch)
- Petastorm as a bridge between PySpark DataFrames and PyTorch Dataloaders

Dataset

- Huge dataset with stock price data from 2003 to 2013
- Daily stock market prices from S&P 500 (~10GB of size, available here)
- S&P 500 companies fundamental data from Yahoo Finance (available here)
- Dataset splitting according to temporal windows: training from 2003 to 2011, validation from 2011 to 2012, test from 2012 to 2013

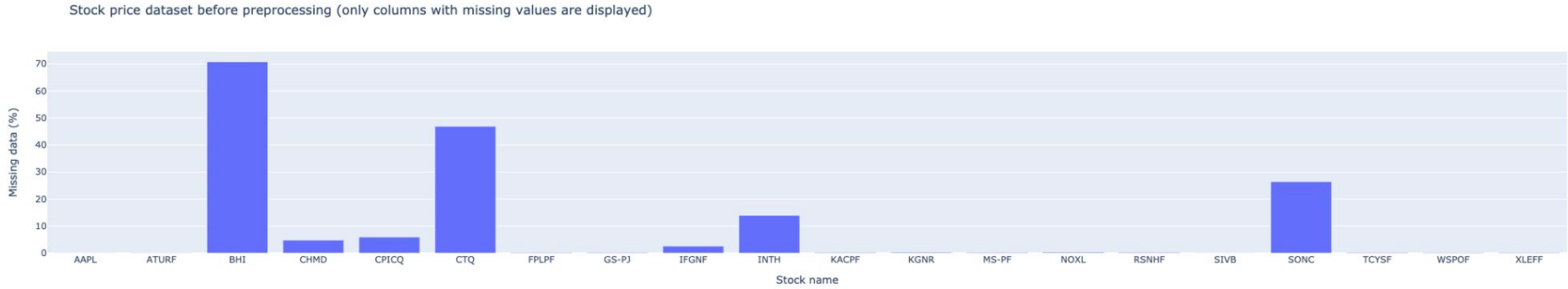
(smaller) Dataset

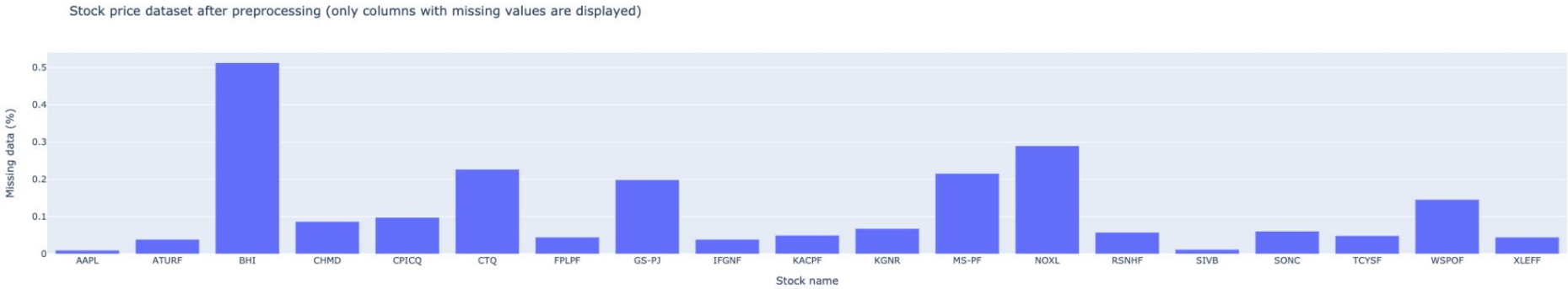
- To speed up the computation on few nodes, only consider a subset of the available data (less than 10%)
- Consider the following stocks for training: AAPL, MSFT, GE, PFE, T
- The final dataset consists of more than 13.000 rows and 35 features

Data preprocessing

- The original dataset was not perfect
- Perform dataset preprocessing (i.e. presence of NaNs, missing value imputation, missing weekdays, trailing noisy data)
- Scale numerical features for faster convergence (e.g. SGD may benefit from it)

Noisy data removal

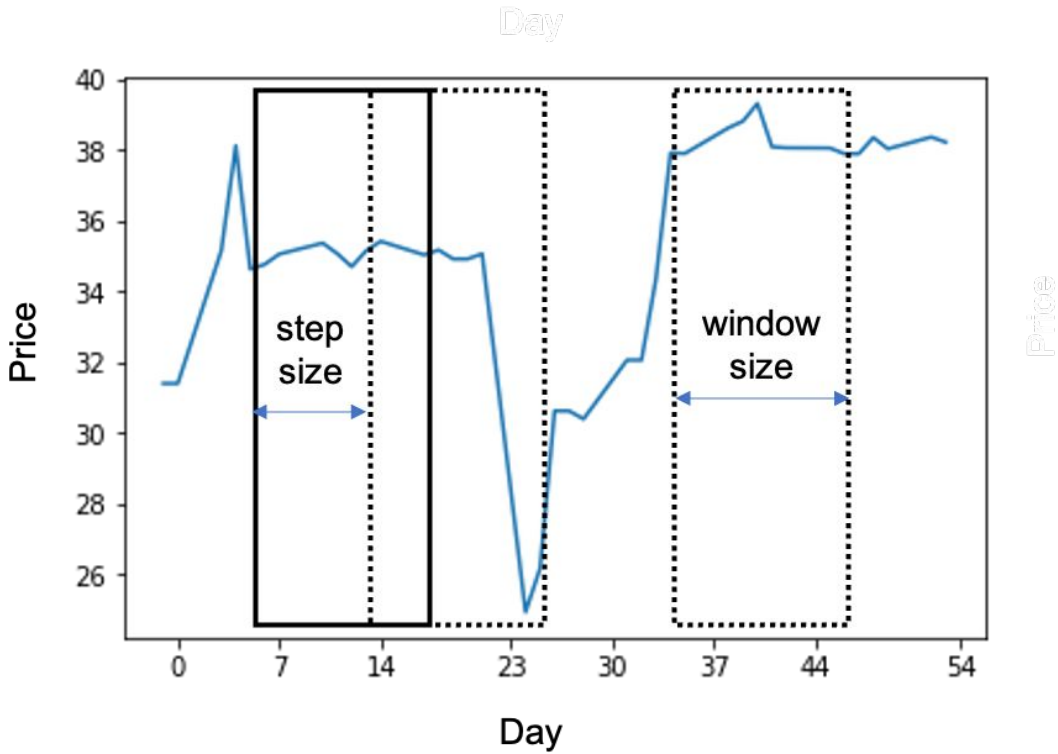




Feature engineering

- Align daily prices with the latest available financial report
- Add technical indicators to boost model performances (i.e. SMA and RSI)
- Within each dataset split, create overlapping windows of size *n* (the size of the lag)

Sliding window technique



Model training

- Use a Long Short Term Memory (LSTM) for interpreting sequences
- Train on the entire group of stocks in the dataset and minimize the MSE
- Use shuffled windows from all the stocks and predict the respective *n+1*th day

Evaluation framework

- Evaluate the models using R^2 and MSE
- Develop a trading strategy and evaluate its real performances on test data (i.e. 1-year profit and operation accuracy)
- Benchmark our model against PySpark regressors (i.e. DecisionTreeRegressor, RandomForestRegressor)

Trading strategy algorithm

```
cnt <- 0
correct <- 0
pnl <- .0
for each currentPrice, real, prediction in samples do
    if shouldBuy(real, prediction) then
        ret <- (real - currentPrice) / currentPrice
        pnl <- pnl + ret
        cnt <- cnt + 1
        if ret >= 0 then
            correct <- correct + 1

accuracy <- correct / cnt
return pnl,accuracy
```

Experiments (1/2)

- For reproducibility, all the experiments have been logged and available here
- Use the best performing model after hyperparameter tuning
- Competitive performance of the ensemble technique

Experiments (2/2)

	MSE	R2	Adjusted R2	Operation accuracy	Profit
DecisionTreeRegressor	0.078	0.852	-	55.45%	35.97%
RandomForestRegressor	0.104	0.803	-	**57.01%**	51.61%
LSTM	**0.021**	**0.939**	**0.897**	56.52%	**58.35%**

Conclusions

- Recurrent neural networks effectiveness on the task
- Good performances of the regressors on the downstream task (i.e. a binary classification task)
- Not straightforward evaluation of the best approach (i.e. multiple metrics are required)

Future work

- Train the models on all the available data using more nodes
- Use a distributed deep learning training framework (e.g. Horovod)
- More feature engineering (e.g. sentiment analysis on financial data, experiment different strategies for missing values imputation)

Thanks