

Siemens Stock Market Big Data Analysis

MECD - EGD - GROUP 2

Cátia Teixeira - Farzam Salimi

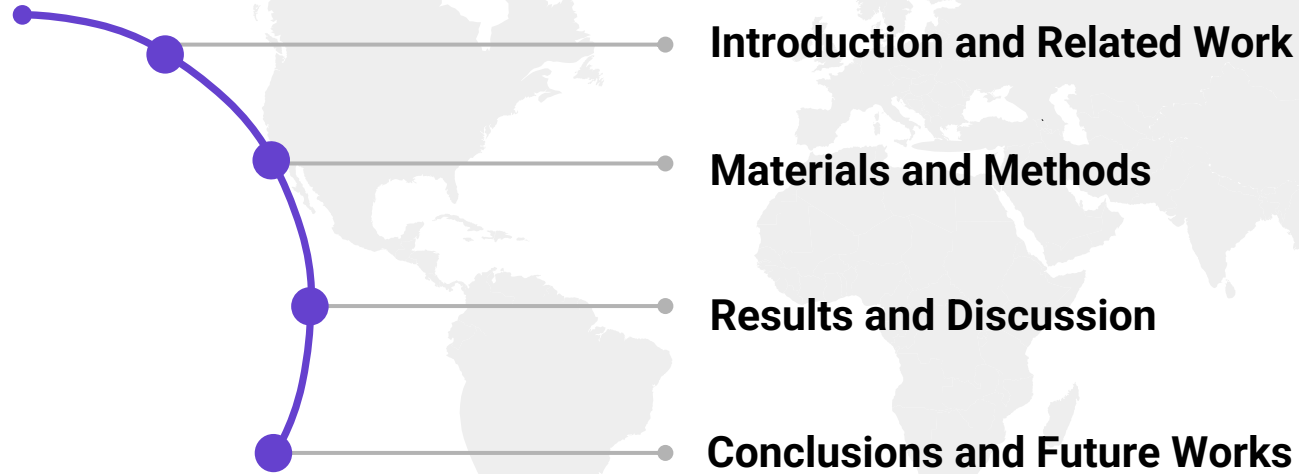
Nikolas Ivakko - Pedro Brás

Rojan Aslani - Sónia Ferreira

Vasco Bartolomeu



Content



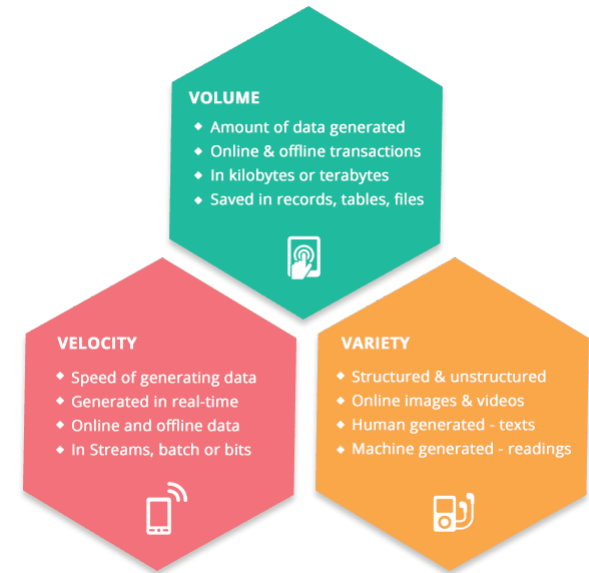
Introduction & Related Work

Introduction

What is Big Data?

- Big data **refers to data sets that are too large or complex to be dealt with by traditional data-processing software.**
- Can be mined for information and used in machine learning projects, predictive modelling and other advanced analytics applications.
- **Originally associated**, in 2001 by Doug Laney, then an analyst at consulting firm Meta Group Inc. **with three key concepts: volume, variety, and velocity.** More recently, several other V's have been added, including veracity, value and variability. ([Botelho, B 2022](#))

THE 3Vs OF BIG DATA



Introduction

Importance of Big Data?



Doesn't revolve around how much data we have.



The value lies in how it's used it.

Big data analysis challenges include:

capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy, and data source.

Introduction

Problem Description

- Frequently changing prices;
- Difficult to predict price evolution;
- Often involves using both numerical and news-based information;
- Delays in decisions may have significant losses.

Requirements:

- Fast analysis of thousands of observations;
- High accuracy;
- Strong computational power.



Target: Decide when to buy or sell a Siemens stock

Label: *'buy_or_sell'*

Introduction

Related Work

ML methods	Number of articles	Research articles
1. Linear regression	1	Rana et al. (2019)
2. Naive bayes	3	Alsubaie et al. (2019); Nabi et al. (2019); Singh and Khushi (2021)
3. Gaussian Naive bayes (GNB)	1	Ampomah et al. (2021)
4. K-nearest neighbors	2	Chen and Hao (2017); Singh and Khushi (2021)
5. Lasso estimate	1	Aloraini (2015)
6. Broad learning system (BLS)	1	Li et al. (2022)
7. SVM	11	Cai et al. (2012); Alsubaie et al. (2019); Kumar et al. (2016); Nabi et al. (2019); Yuan et al. (2020); Labiad et al. (2016), Rana et al. (2019); Chen and Hao (2017); Siddique and Panda (2019); Singh and Khushi (2021); Iacomin (2015)
Tree-based ML methods	-	-
8. Decision tree	3	Nabi et al. (2019); Singh and Khushi (2021); Qolipour et al. (2021)
9. RF	6	Nabi et al. (2019); Yuan et al. (2020); Labiad et al. (2016); Singh and Khushi (2021); Ampomah et al. (2020); Qolipour et al. (2021)
10. Gradient boosted tree	2	Labiad et al. (2016); Qolipour et al. (2021)
Neural network methods	-	-
11. ELM	1	Das et al. (2019)
12. OSELM	1	Das et al. (2019)
13. RBPNN	1	Das et al. (2019)
14. Deep generative model	1	Haq et al. (2021)
15. ANN	2	Alsubaie et al. (2019); Yuan et al. (2020)
16. LSTM	3	Botunac et al. (2020); Shen and Shafiq (2020); Rana et al. (2019)

(Htun, 2023)

Materials & Methods

Materials

Dataset

Source:

[Kaggle](#): Stock Market Data

Size:

≈ 680 MB

- Rows: ≈ 670 K
- Cols: 60

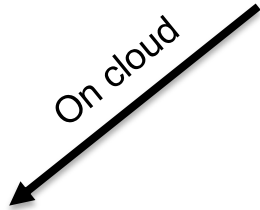
Other info:

- Time interval: Jan 2015 – Oct 2022
- Frequency: 60 seconds
- **Clean** (no duplicates or missing values)
- **Not normal distribution**

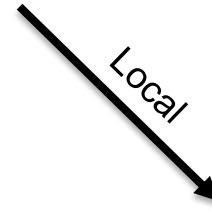
Features	
Time	Date;
Market Values	Open, high, low, close, volume;
Numericals	Sma5, sma10, etc;
Performance Metrics	ADX5, ADX10, etc;

Materials

Softwares

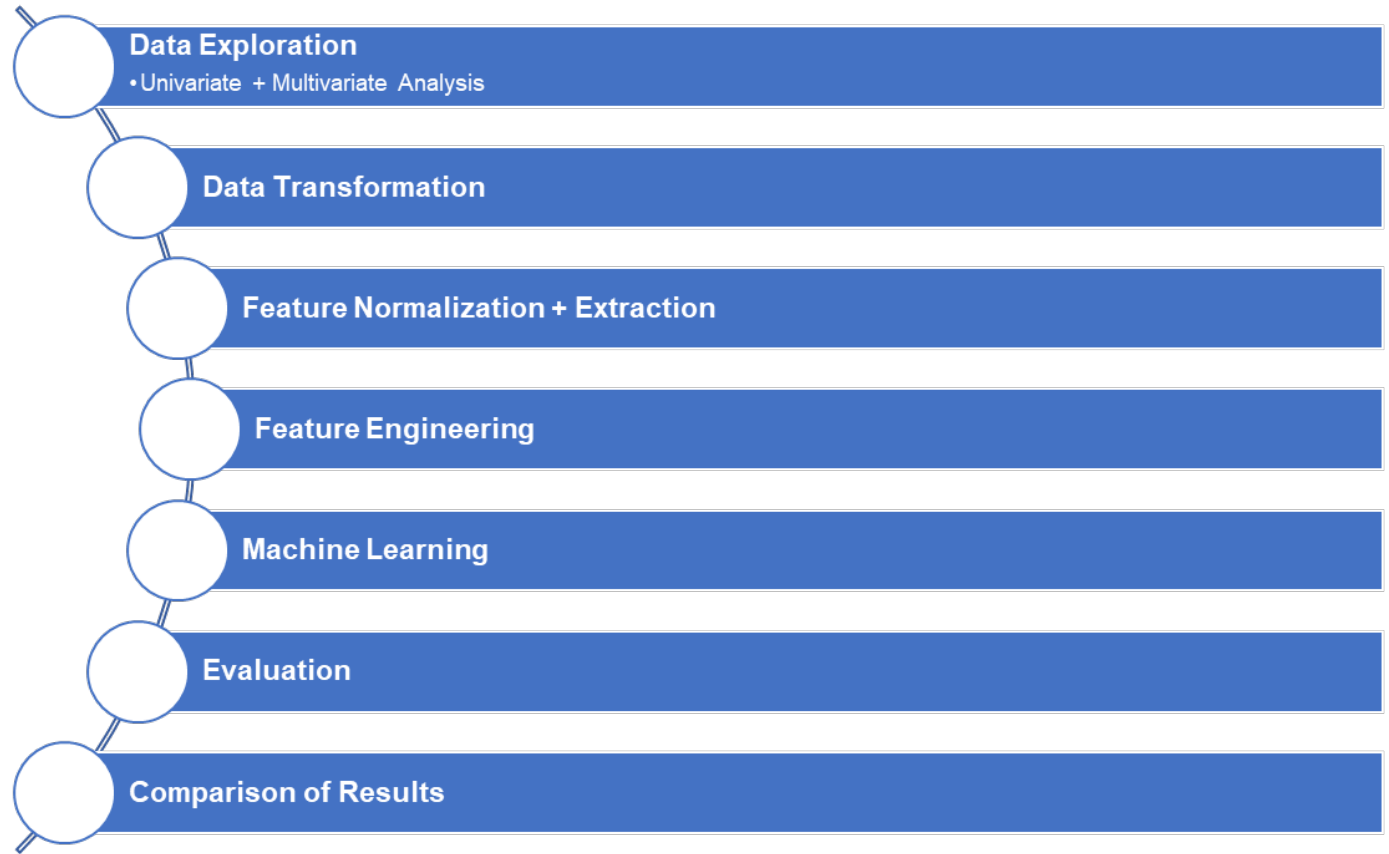


Cloud Dataproc



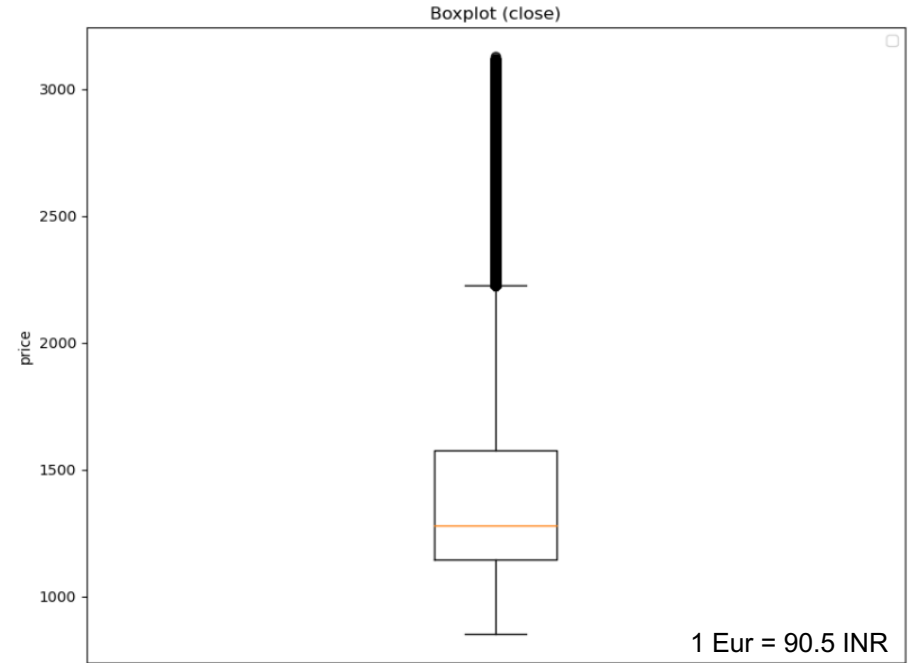
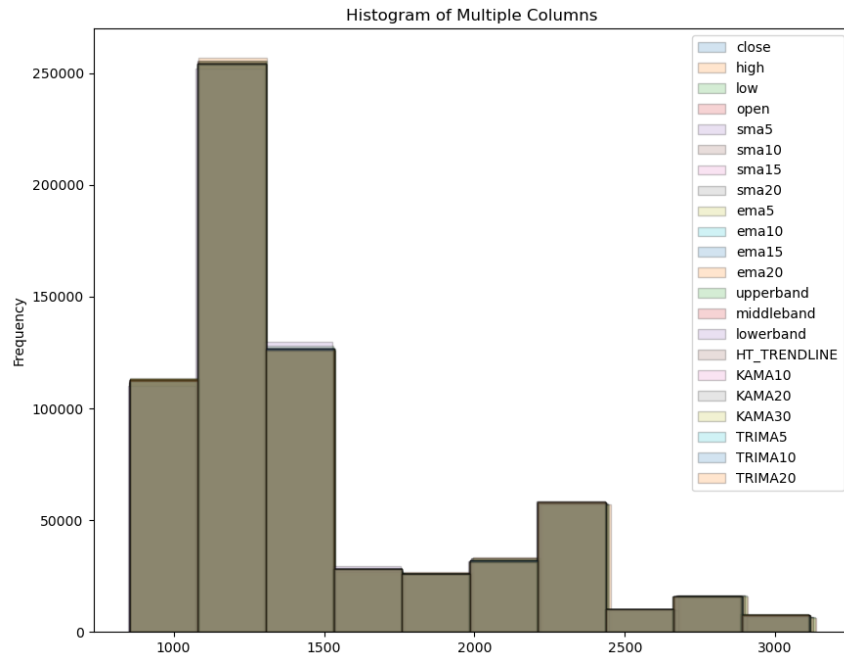
Methods

Plan



Methods

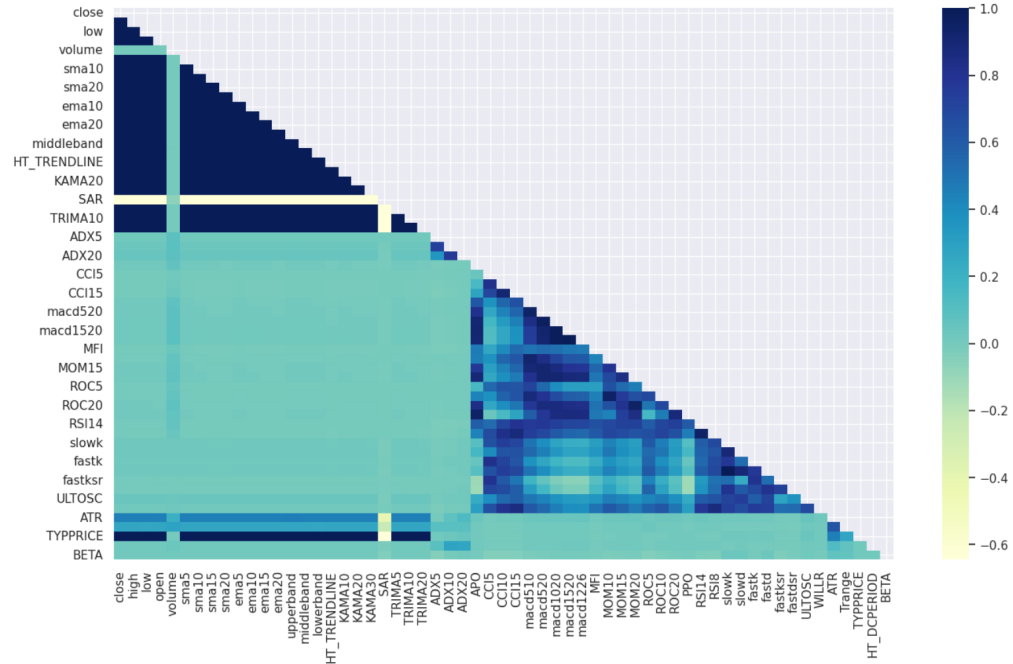
Univariate analysis



Methods

Multivariate analysis

Example of parameter (close) evolution by date



Methods

Feature Normalization + Extraction

Normalization:

- To each feature it was removed the average and divided by the variance.

Principal Component Analysis (PCA)

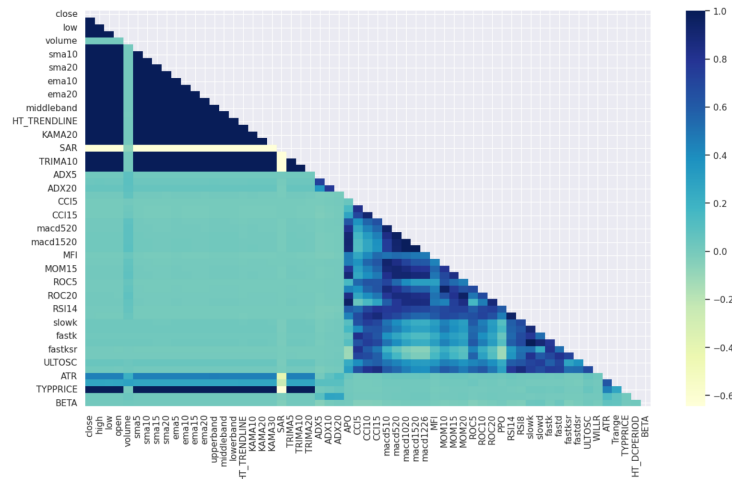
Two different datasets were applied to PCA:

- All data: 59 features were converted to **12** components explaining **95%** of the variance.

Explained Variance Ratio [0.40237949 0.25498293 0.09904388 0.04128876 0.02609325 0.02199506 0.02007322 0.01669642 0.01603183 0.01513495 0.01296261 0.01021636]

- Correlated Data: 22 features were converted to **1** component explaining **99%** of the variance.

Explained Variance Ratio [0.99998649]



Methods

Feature Engineering

Target binary variable was created by evaluating if the following high price of the stock was above the current high price

```
# Create a new DataFrame with the 'buy_or_sell' column using Spark SQL
buy_sell_df = spark.sql("SELECT date, high, LAG(high) OVER (ORDER BY date) AS next_high FROM stock_data")
buy_sell_df = buy_sell_df.withColumn('buy_or_sell', when(buy_sell_df['next_high'] > buy_sell_df['high'], 1).otherwise(0))

# Convert the DataFrame to an RDD
rdd = data.select('date_only', 'open').rdd
rdd = rdd.repartition(10)

buy_sell_df.show()
```

date	high	next_high	buy_or_sell
2015-02-02 04:48:00	1081.0	null	0
2015-02-02 04:49:00	1081.8	1081.0	0
2015-02-02 04:50:00	1082.6	1081.8	0
2015-02-02 04:51:00	1083.25	1082.6	0
2015-02-02 04:52:00	1085.0	1083.25	0
2015-02-02 04:53:00	1086.45	1085.0	0
2015-02-02 04:54:00	1085.25	1086.45	1
2015-02-02 04:55:00	1084.85	1085.25	1
2015-02-02 04:56:00	1084.85	1084.85	0
2015-02-02 04:57:00	1086.7	1084.85	0
2015-02-02 04:58:00	1089.4	1086.7	0
2015-02-02 04:59:00	1089.4	1089.4	0
2015-02-02 05:00:00	1089.9	1089.4	0
2015-02-02 05:01:00	1089.4	1089.9	1
2015-02-02 05:02:00	1088.9	1089.4	1
2015-02-02 05:03:00	1088.0	1088.9	1
2015-02-02 05:04:00	1090.5	1088.0	0
2015-02-02 05:05:00	1092.5	1090.5	0
2015-02-02 05:06:00	1092.5	1092.5	0
2015-02-02 05:07:00	1091.6	1092.5	1

only showing top 20 rows

Methods

Feature Engineering

Proceeded to use RDD's with **10 partitions** since the dataset size is around 600MB with 671024 instances.

Used **rdd.map()** function operate between rows and converted to a **dataframe** in order to **join** sub-table with the main table and removed the columns that were used to calculate the new variable

```
rdd = data.select('date', 'macd510', 'macd1226').rdd
rdd = rdd.repartition(10)

# Define a function to compute the MACD
def macd_func(row):
    macd510 = row['macd510']
    macd1226 = row['macd1226']
    macd = macd510 - macd1226
    return (row['date'], macd)

# Apply the map function to the RDD
mapped_rdd = rdd.map(macd_func)

# Convert the result RDD back to a DataFrame
macd_df = mapped_rdd.toDF(['date', 'macd'])

data = data.join(macd_df, on='date')

data = data.drop('macd510', 'macd520', 'macd1020', 'macd1520', 'macd1226', 'ema5', 'ema10', 'ema15', 'ema20')
```


Methods

Feature Engineering

Some features such as **Momentum**, that evaluate how the closing price of the stock shifts from one day to another, are not calculated within the date variable but the day so they need to be further grouped with a SQL query.

```
rdd = data.select('date_only', 'avg_close', 'avg_close_prev_day').rdd
rdd = rdd.repartition(10)

# Define a function to compute the Bollinger Bands width for a row
def momentum(row):
    avg_close = row['avg_close']
    avg_close_prev_day = row['avg_close_prev_day']
    mom = avg_close - avg_close_prev_day
    return (row['date_only'], mom)

# Apply the map function to the RDD
mapped_rdd = rdd.map(momentum)

# Convert the result RDD back to a DataFrame
momentum_df = mapped_rdd.toDF(['date_only', 'momentum'])

momentum_df.createOrReplaceTempView("momentum_df")

momentum_df = spark.sql("SELECT date_only, AVG(momentum) as momentum FROM momentum_df GROUP BY date_only ORDER BY date_only")

data = data.join(momentum_df, on='date_only')
data = data.drop('MOM20', 'MOM15', 'MOM10')
```

Methods

Feature Engineering

At the end of our **feature engineering** work, we were able to aggregate all of the different timed moving averages and create new features that **reduced** the dimensionality of our problem in **40%** (from 59 to 34 features) while **maintaining the accuracy of the models**.

```
df.columns
```

```
['date', 'close', 'high', 'low', 'open', 'volume', 'sma5', 'sma10', 'sma15', 'sma20', 'ema5', 'ema10', 'ema15', 'ema20', 'upperband', 'middleband', 'lowerband', 'HT_TRENDLINE', 'KAMA10', 'KAMA20', 'KAMA30', 'SAR', 'TRIMA5', 'TRIMA10', 'TRIMA20', 'ADX5', 'ADX10', 'ADX20', 'APO', 'CCI5', 'CCI10', 'CCI15', 'macd510', 'macd520', 'macd1020', 'macd1520', 'macd1226', 'MFI', 'MOM10', 'MOM15', 'MOM20', 'ROC5', 'ROC10', 'ROC20', 'PPO', 'RSI14', 'RSI8', 'slowk', 'slowd', 'fastk', 'fastd', 'fastksr', 'fastdsr', 'ULTOSC', 'WILLR', 'ATR', 'Trange', 'TYPPPRICE', 'HT_DCPERIOD', 'BETA']
```

```
data.columns
```

```
['date_only', 'date', 'close', 'high', 'low', 'open', 'volume', 'HT_TRENDLINE', 'SAR', 'APO', 'MFI', 'PPO', 'slowk', 'slowd', 'fastk', 'fastd', 'fastksr', 'fastdsr', 'ULTOSC', 'WILLR', 'ATR', 'Trange', 'TYPPPRICE', 'HT_DCPERIOD', 'BETA', 'avg_close', 'avg_close_prev_day', 'avg_open', 'buy_or_sell', 'macd', 'bollinger_bands_width', 'com_channel_index', 'rsi', 'momentum', 'sma', 'kama', 'adx', 'roc', 'trima']
```

Methods

Modeling

Random Forest (RF)



- Supervised machine learning algorithm.
- **Used for classification and regression problems.**
- Build decision trees on different samples.
- Takes a majority vote for classification and an average for regression.



Support Vector Machine (SVM)

- Set of supervised learning methods
- Compared to newer algorithms like neural networks, they have **two main advantages**:
 1. **higher speed**
 2. **better performance with a limited number of samples** (in the thousands).

Methods

Dataprox description and cluster configuration



Clusters	Manager Node			
	Region	Series	Machine Type	Primary Disk Size
1	Europe	N2	2vCPU-8 GB memory	500
2	Europe	N2	2vCPU-8 GB memory	100

Clusters	Worker Nodes			
	Series	Machine Type	Primary Disk Size	Number of Worker
1	N2	2vCPU-8 GB memory	500	2
2	N2	2vCPU-8 GB memory	100	3

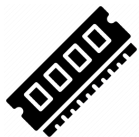
Results & Discussion

Results

Experimental set overview



	3 workers			2 workers		
	No transformation	PCA	Feature Engineering	No transformation	PCA	Feature Engineering
Random Forest	X	X	X	X	X	X
Random Forest RDD	X	X	X	X	X	X
SVM	X	X	N.A.	X	X	N.A.



	3 workers					
	Driver Memory	Executor Memory	Executer Instances	Executer cores	Parallelism	Total number of executors
Random Forest (no transformation)	1GB	1GB	2	1	default	2
	8GB	4GB	12	2	2	2



	3 workers	
	Local	Cloud
Random Forest_RDD (no transformation)	X	X

Results

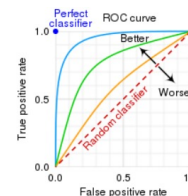
Performance of Classifiers



		3 workers			2 workers		
		No transformation	PCA	Feature Engineering	No transformation	PCA	Feature Engineering
Random Forest	Run time	2' 41"	3' 56"	10' 59"	3' 19"	5' 46"	12' 04"
	ROC	0.82	0.81	0.78	0.82	0.81	0.78
Random Forest RDD	Run time	2' 08"	4' 07"	7' 56"	2' 33"	5' 13"	15' 02"
	ROC	0.58 (*)	0.62 (*)	0.58 (*)	0.62 (*)	0.63 (*)	0.66 (*)
SVM	Run time	3' 36"	4' 17"	N/A	4' 26"	5' 39"	N/A
	ROC	0.73	0.73	N/A	0.73	0.73	N/A

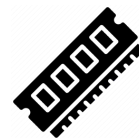
		No transformation	PCA	Feature Engineering
Local	ROC	0.77	0.72	0.73

(*) results for this model gave different ROC results locally because of different pySpark version on the cloud (local:3.4 | cloud:3.1)



Results

Different Spark configurations



	Config	Measure	No transformation	Feature Engineering
Random Forest	A	Run time	2' 41''	10' 59''
		ROC	0.82	0.78
	B	Run time	2' 08''	6' 11''
		ROC	0.82	0.78

Config	Driver Memory	Executor Memory	Executor Instances	Executor cores	Parallelism	Total number of executors	Workers
A (default)	1GB	1GB	2	1	default	2	3
B	8GB	4GB	12	2	2	2	3

Results

Local vs Cloud



		3 workers	
		Local	Cloud
Random Forest_RDD (no transformation)	Run time	2' 34''	2' 08''
	ROC	0.77	0.58 (*)

Docker memory	PC Processor	PC CPU
8 GB	Intel(R) Core(TM) i7-8750H	@ 2.20GHz 2.21 GHz

(*) results for this model gave different ROC results locally because of different pySpark version on the cloud (local:3.4 | cloud:3.1)

Results

Different number of workers – CPU Utilization

3 workers — (blue line)
2 workers — (red line)

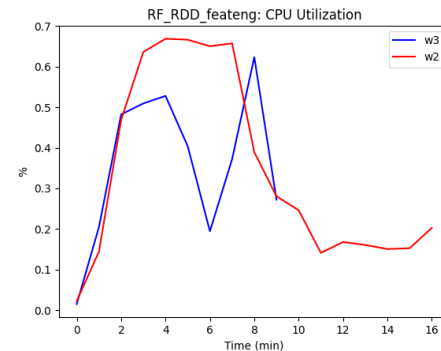
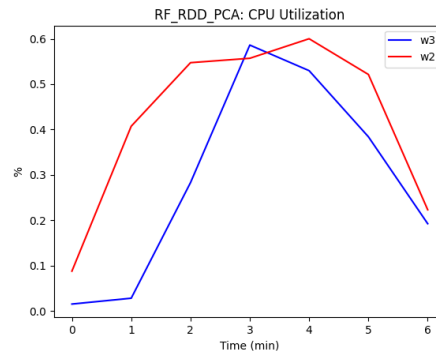
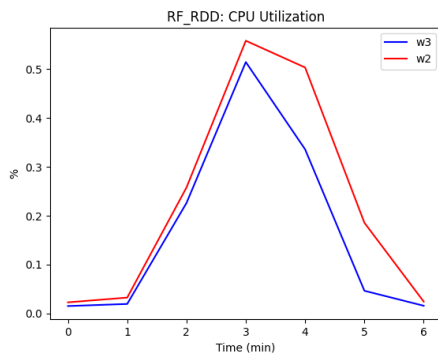
Full Dataset

PCA

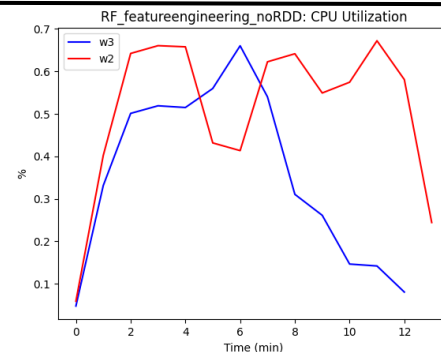
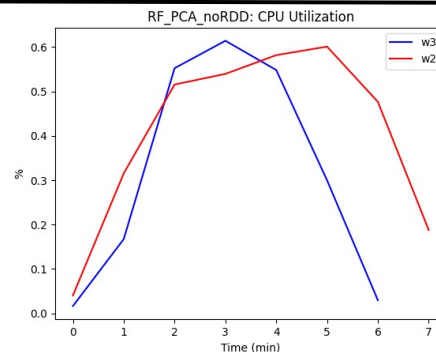
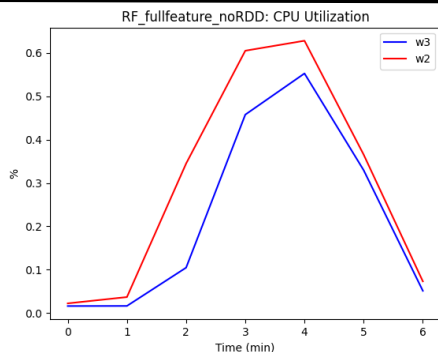
Feature Engineering

Random Forest

RDD

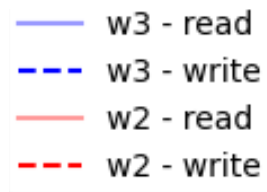


Not RDD



Results

Different number of workers – Disk Bytes

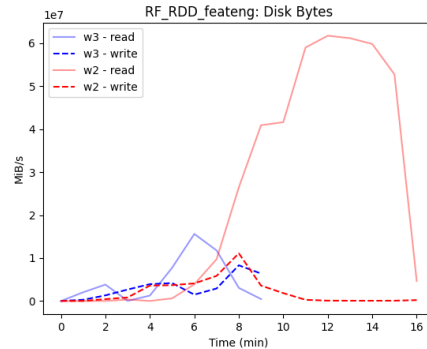
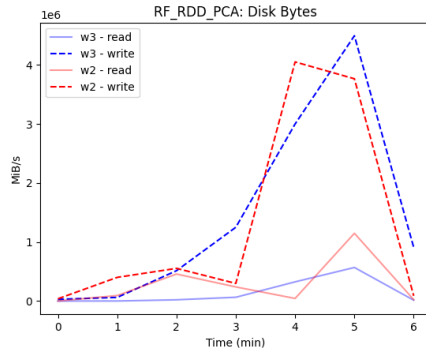
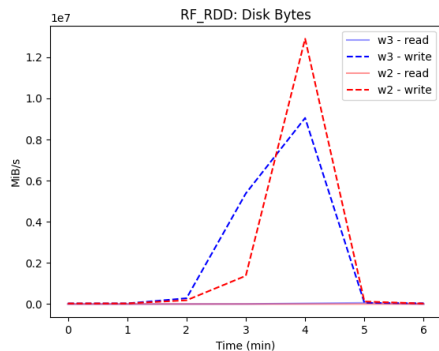


Full Dataset

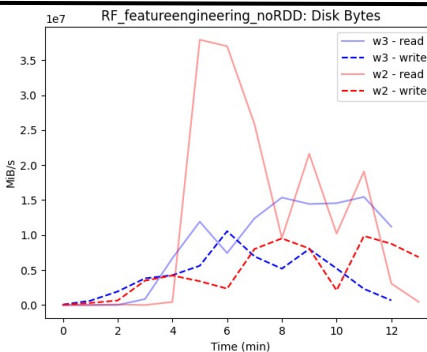
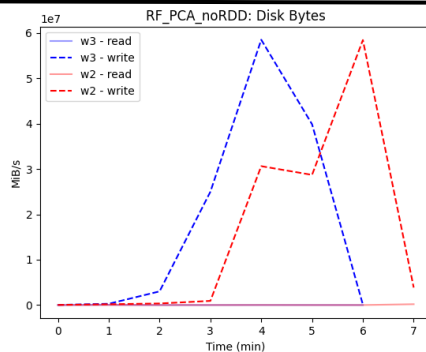
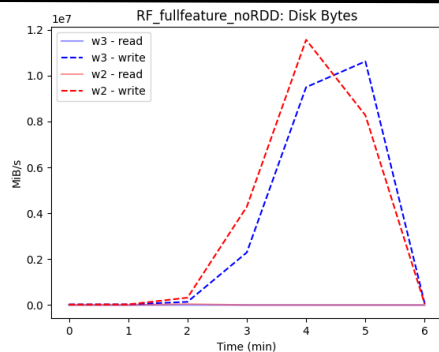
PCA

Feature Engineering

RDD



Not RDD



Random Forest

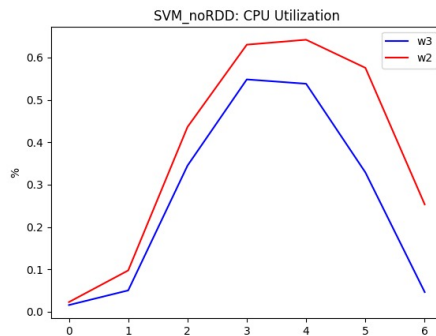
Results

Different number of workers – no RDD

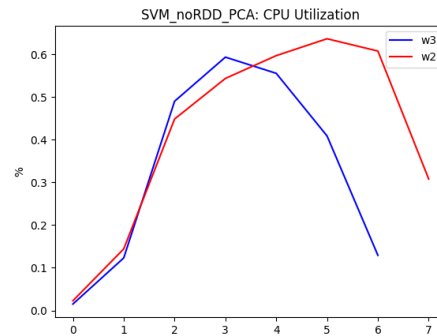
SVM

CPU
Utilization

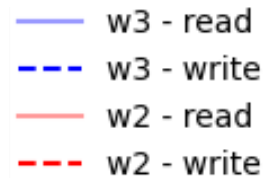
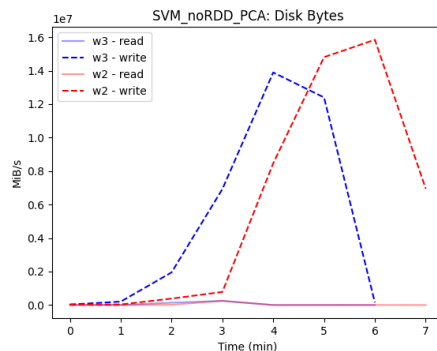
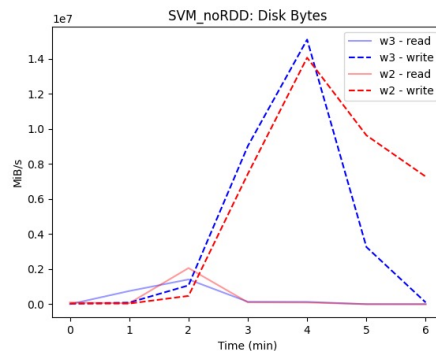
Full Dataset



PCA



Disk
Bytes

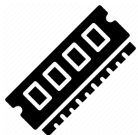


Conclusions & Future Works

Conclusions



- Random Forest (RF) produces the most accurate results, followed by the SVM and RF RDD in the cloud
- The runtime on PCA (+25%) and feature engineering (+300%) is greater due to the fact that the runtimes include the computation of the features, as well as the forecast algorithm



- As expected, 3 workers are faster than 2 workers. Workers to run time relation is not linear
- On average, RF RDD library takes less time than RF library using DataFrame
- From configuration A (Spark session default) to configuration B, the run time decreased by 44%

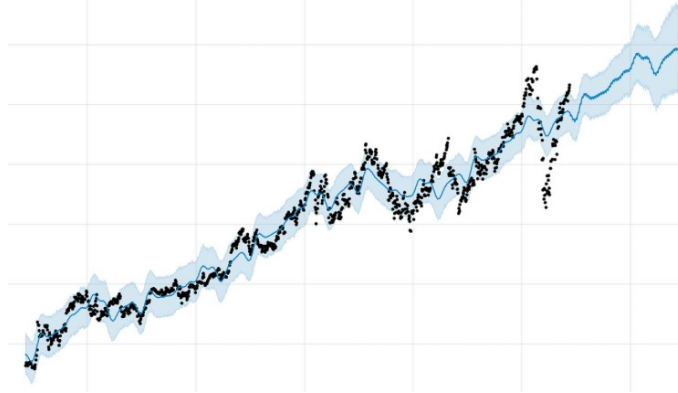


- Cloud is $\approx 20\%$ faster than running files locally

Future Works

Feature Engineering and further approaches

- Relationship between multiple stocks, e.g. SIEMENS+BMW+BOSCH
- Time Series Buffering and forecasting
- Multiple data input, e.g. News Sentiment Analysis
- Inclusion of company fundamental data, such as financial statements or macroeconomic factors
e.g. liquidity



Future Works

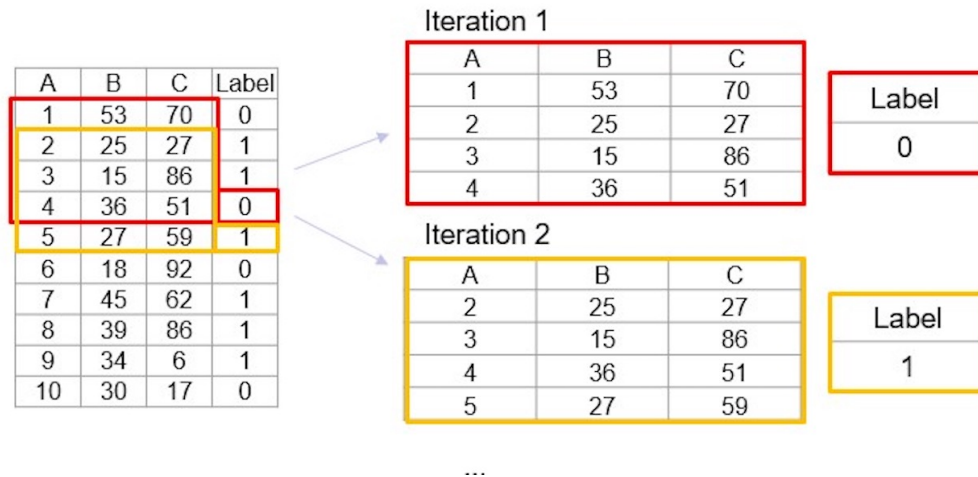
LSTM

- Multivariate time series data: A dataset of multiple time-dependent variables measured at the same time points.
- LSTM (Long Short-Term Memory) Networks: A type of neural network that can capture the complex temporal dependencies in sequential data.
- Why LSTM Networks work for multivariate time series classification:
 - can process multiple time series variables simultaneously and learn their interactions.
 - can handle variable-length time series and missing values, making it suitable for real-world datasets.
 - has a memory mechanism that can remember long-term dependencies, allowing it to capture patterns that occur over longer periods of time.
- Applications:
 - Predicting stock prices, sales forecasting, disease diagnosis...

Future Works

Multivariate Timeseries Classification

- Example Window size: 4
- The buffered data can then be fed into the LSTM as a sequence of inputs
- LSTM able to learn the dependencies and patterns within the buffered data and use this information to make predictions
- Capture temporal dependencies over a longer time horizon than just looking at individual time points



Future Works

PySpark + Tensorflow

Conventional Approach:

- CSV data to Pandas Dataframe
- Preprocess/ Scale Dataframe
- Sliding Window over Dataframe
- Create Tensorflow Dataset with generator function

PySpark Challenges

- Data can be distributed
- PySpark dataframe does not directly support direct iteration/slicing like .iloc()
- Approach should be applicable for large/ distributed data
- Adapting initial approach very inefficient

```
def gen(data):
    shape = data.shape
    for i in range(shape[0]-10):
        x = data.iloc[i:i+10]
        y = x.pop('buy_or_sell')
        y = y.iloc[-1]
        yield x, y

def makedataset(data):
    dataset = tf.data.Dataset.from_generator(lambda: gen(data), output_signature=(
        tf.TensorSpec(shape=(10, 7), dtype=tf.float32),
        tf.TensorSpec(shape=(), dtype=tf.float32)))
    return dataset
```

```
HS = 10
COLS = ['date', 'close', 'high', 'low', 'open', 'volume', 'buy_or_sell']
data = data.select(*COLS)

def gen(data):
    shape = data.count()
    for i in range(HS-1, shape):
        # Get the block of rows for this iteration
        block = data.withColumn("row_num", row_number().over(Window.orderBy("date")))\
            .filter((col("row_num") >= i-HS) & (col("row_num") < i))\
            .orderBy("date")\
            .toPandas()
        x = block.iloc[:, 1:-2]
        y = block.iloc[-1, -2]
        yield tf.convert_to_tensor(x.values, dtype=tf.float32), tf.convert_to_tensor(y, dtype=tf.float32)

def makedataset(data):
    dataset = tf.data.Dataset.from_generator(lambda: gen(data),
        output_signature=(tf.TensorSpec(shape=(HS, len(COLS)-2), dtype=tf.float32),
            tf.TensorSpec(shape=(), dtype=tf.float32)))
    return dataset
```


	date	close	high	low	open	volume	buy_or_sell
2015-02-02 04:48:00	1080.1	1081.0	1077.6	1077.6	960	0	0
2015-02-02 04:49:00	1080.5	1081.8	1079.0	1080.1	2041	0	0
2015-02-02 04:50:00	1082.0	1082.6	1080.6	1080.6	1136	0	0
2015-02-02 04:51:00	1082.35	1083.25	1082.0	1082.0	2661	0	0
2015-02-02 04:52:00	1085.0	1085.0	1082.15	1082.9	1730	0	0
2015-02-02 04:53:00	1085.25	1086.45	1085.2	1085.95	3161	0	0
2015-02-02 04:54:00	1084.85	1085.25	1083.5	1085.2	905	1	1
2015-02-02 04:55:00	1082.85	1084.85	1082.85	1084.85	1248	1	1
2015-02-02 04:56:00	1084.05	1084.85	1082.85	1082.85	941	0	0
2015-02-02 04:57:00	1086.0	1086.7	1084.3	1084.3	2327	0	0
2015-02-02 04:58:00	1088.5	1089.4	1085.05	1085.05	6122	0	0
2015-02-02 04:59:00	1088.0	1089.4	1087.15	1088.45	3202	0	0
2015-02-02 05:00:00	1088.7	1089.9	1087.5	1088.0	4352	0	0
2015-02-02 05:01:00	1088.55	1089.4	1087.75	1088.7	1812	1	1
2015-02-02 05:02:00	1087.7	1088.9	1087.0	1088.9	1856	1	1

```
[(<tf.Tensor: shape=(8, 5), dtype=float32, numpy=
array([[1080.1, 1081. , 1077.6, 1077.6, 960. ],
       [1080.5, 1081.8, 1079. , 1080.1, 2041. ],
       [1082. , 1082.6, 1080.6, 1080.6, 1136. ],
       [1082.35, 1083.25, 1082. , 1082. , 2661. ],
       [1085. , 1085. , 1082.15, 1082.9, 1730. ],
       [1085.25, 1086.45, 1085.2, 1085.95, 3161. ],
       [1084.85, 1085.25, 1083.5, 1085.2, 905. ],
       [1082.85, 1084.85, 1082.85, 1084.85, 1248. ]], dtype=float32)>,
 <tf.Tensor: shape=(), dtype=float32, numpy=1.0>),
 (<tf.Tensor: shape=(8, 5), dtype=float32, numpy=
array([[1080.5, 1081.8, 1079. , 1080.1, 2041. ],
       [1082. , 1082.6, 1080.6, 1080.6, 1136. ],
       [1082.35, 1083.25, 1082. , 1082. , 2661. ],
       [1085. , 1085. , 1082.15, 1082.9, 1730. ],
       [1085.25, 1086.45, 1085.2, 1085.95, 3161. ],
       [1084.85, 1085.25, 1083.5, 1085.2, 905. ],
       [1082.85, 1084.85, 1082.85, 1084.85, 1248. ],
       [1084.05, 1084.85, 1082.85, 1082.85, 941. ]], dtype=float32)>,
 <tf.Tensor: shape=(), dtype=float32, numpy=0.0>)]
```

- Very runtime intense approach
- Further exploration with RDD for preprocessing speed up
- Interruption after approx. 2.5 Hours (training did not start)




References



[Htun, H.H., Biehl, M. & Petkov, N. Survey of feature selection and extraction techniques for stock market prediction. *Financ Innov* 9, 26 \(2023\). <https://doi.org/10.1186/s40854-022-00441-7>](https://doi.org/10.1186/s40854-022-00441-7)



[Botelho, B, “Big data”. TechTarget, January 2022. Retrieved in 04 May 2023.](#)



[Ramirez-Gallego, S. et al., A Survey on data preprocessing for data stream mining: Current status and future directions, *Jornal Neurocomputing* \(2017\)](#)



[Ardagna, D., et al.: Predicting the performance of big data applications on the cloud. *J. Supercomput.* 77, 1321–1353 \(2021\).](#)