

Algorithmic Trading using the Deutsche Börse Public Dataset

Marjan Ahmed

*Dept of Computer Science
University of Illinois
at Urbana-Champaign
Vancouver, BC, Canada
marjana2@illinois.edu*

Fan Yang

*Dept of Computer Science
University of Illinois
at Urbana-Champaign
Palo Alto, CA, United States
fanyang3@illinois.edu*

Dilruba Hawk

*Dept of Computer Science
University of Illinois
at Urbana-Champaign
New York, NY, United States
dilruba2@illinois.edu*

Wang Chun Wei

*Dept of Computer Science
University of Illinois
at Urbana-Champaign
Brisbane, QLD, Australia
wcwei2@illinois.edu*

Abstract—This report details the findings, implementation and research method of the teams algorithmic trading model developed using machine learning frameworks implemented leveraging AWS cloud computing infrastructure. Github project website: <https://github.com/weiwangchun/cs498cca>

I. INTRODUCTION

We intend to build a cloud hosted backtesting framework for algorithmic trading. The objective is to analyze a given data set of equities containing price-volume data and develop medium to high frequency strategies for trading equities quickly via cloud computing. A simple web interface will be constructed allowing users to control and tweak trading parameter settings, and see in-sample and out-of-sample trading performance (i.e., hypothetical profit and loss, tradings costs, turnover, portfolio risk and Sharpe ratio).

Given the limitations of our dataset, we focus on using a feature set that includes (i) stock returns, (ii) implied stock volatility, (iii) order imbalance, (iv) trading volumes and (v) number of trades to predict future stock prices. These information are all publically available, and hence if the German market was at least weakly efficient, trading profitably using this subset of information would be difficult. For a given stock at a given time, we compute its most recent feature set, its long run average feature set and also its feature set percentile compared to its historical feature sets (i.e., how does it compare to history) and lastly compute its feature set percentile compared to its peers' feature sets (i.e., how does it compare to its peers). We attempt to use both a simple logistic regression model and a neural network to predict future stock price.

Our results are mildly successful, but after accounting for transaction costs, we conclude that we are unable to construct a commercially viable trading strategy.

II. DATASET

The chosen dataset is the Deutsche Börse public dataset available on Amazon Web Services (AWS). It contains trading data in 2 minute intervals for every tradeable security listed on the Eurex and Xetra trading platforms located in Frankfurt, Germany. The dataset is circa 3 GB in size

with over 18,000 files and is located at: <https://s3.eu-central-1.amazonaws.com/deutsche-boerse-xetra-pds/> In aggregate, we collected data from the 2nd of January 2018 to the 1st of April 2019 (314 unique trading days). In total, this had 18,854,026 rows. Each row represented a 2 minute trading interval for a particular stock. There were 2,837 unique stocks in the dataset. A snapshot of the dataset is shown below.

ISIN	Mnemonic	SecurityDesc	SecurityType	Currency	\
AT0000A0E9W5	SANT	S+T AG (Z.REG.MK.Z.)O.N.	Common stock	EUR	
DE0000A0D554	NDX1	NORDEX SE O.N.	Common stock	EUR	
DE0000A0WMP16	AIXA	AIXTRON SE NA O.N.	Common stock	EUR	
DE000A1EWNN0	ADS	ADIDAS AG NA O.N.	Common stock	EUR	
DE000A1ML7J1	VNA	VONOVA SE NA O.N.	Common stock	EUR	

SecurityID	Date	Time	StartPrice	MaxPrice	MinPrice	EndPrice	\
2504159	2018-01-02	08:00	18.15	18.150	18.14	18.140	
2504290	2018-01-02	08:00	8.95	8.950	8.95	8.950	
2504428	2018-01-02	08:00	11.61	11.625	11.61	11.625	
2504471	2018-01-02	08:00	167.30	167.700	167.30	167.400	
2504501	2018-01-02	08:00	41.58	41.720	41.50	41.500	

TradedVolume	NumberOfTrades
201	2
1831	3
74	2
1326	12
2775	15

III. TECHNOLOGIES AND TOOLS

To implement the objective of this project, we have chosen to leverage the vast computing and storage resources offered by AWS. The technologies used in this project are as follows:

- Simple Storage Service (S3) for storing the dataset and outputs.
- Elastic Cloud Compute (EC2) for initial exploratory analysis and model development.
- Elastic MapReduce (EMR) - for Deployment in Production.
- Route53 for front end web interface.

A step-by-step methodical approach was implemented starting from creating an AWS account, creating users (team members), creating roles and permissions for team members and applications using Identity Access Management (IAM), downloading the data into the created S3 bucket using python helper functions and then use PySpark, built on AWS EMR clusters for exploratory analysis of the data. EMR cluster creation process involved creating a 3-node cluster. Selection of PySpark for model development led to inclusion of the relevant applications, in this case, PySpark and JupyterHub.

JupyterHub helps in line block execution of codes and display results subsequently helping us explore the data and keep track of every outcome. This process will help us test different predictive models and algorithms and fine tune the process in developing the final model before production deployment.

After the model is finalized, We intend to develop a front end Web Interface using Amazon Route 53. A domain named datainsight.guru is chosen where users can choose tickers and, control and tweak trading parameters to see the outcome of the predictive model.

IV. METHOD DESIGN

In terms of the overall methodology, we choose S3 to store the stock price data and choose to use PySpark built on AWS EMR clusters¹ to analyze the data. We use boto3 package in Python to upload stock price data from the Deutsche public dataset to S3². Lastly, we use EMR Notebook to run PySpark jobs. Announced in Nov 2018, EMR Notebook is essentially a Python Jupyter notebook pre-configured to connect to EMR and S3.

The general process of the project is as follows:

- 1) Stock price-volume data is uploaded on S3
- 2) User (via web interface) selects the time horizon and trading parameters (i.e., trading costs, trading frequencies, lookback period, size of trading portfolio, ...)
- 3) We iterate through the time period provided by the user, at each point t :
 - A training set is constructed based on a lookback period (k), i.e. from $t - k$ to $t - 1$
 - A ‘trading model’ (described in the next section) will be fitted / applied to the training set
 - Stocks will be bought and sold in the test set (time t)
 - Profitability will be recorded.
- 4) Results will be displayed back to the user via the web interface

A. Trading Model

Initially, we opted to examine momentum strategies (Jegadeesh and Titman, 1993) on the German market. However, given the Deutsche database has a relatively short history (circa 2 years), we decided to instead focus on higher frequency trading strategies. In particular, we will focus on implementing pairs trading (Vidyamurthy, 2004; Zhang and Zang, 2008) on relatively high frequency equities data. Over user selected trading frequencies (highest frequency = 2 minutes, lowest frequency = 1 day), we systematically search for cointegrated pairs of stocks on the Xetra and Eurex stock exchanges using both the Engle-Granger and Johansen methods. Our trading strategy involves simultaneously buying and selling top cointegrated pairs whose spread has diverged, i.e., betting on convergence.

¹We created a 3-nodes cluster following the AWS Cluster creation process.

²See our codes in Github

Our trading model incorporated both time-series and cross-sectional data. We decided to focus on a daily frequency trading model. Since our data was in 2-minute frequency, we convert the raw data into the follow key features:

- 1) **Total Volume:** an aggregation of 2-minute frequency volume for each stock on a daily basis
- 2) **Total Trades:** the aggregate number of trades made on a given stock per day
- 3) **VWAP:** daily volume weighted average price (granularity at the 2-minute frequency)
- 4) **Daily Return:** the total return made on the stock per day
- 5) **Realized Volatility:** the absolute difference between the maximum price and the minimum price on a stock on a given day
- 6) **Order Imbalance:** a daily measure of trade imbalance denoted as,

$$OI = \frac{|B - S|}{B + S}$$

where B is the total number of buy initiated trades and S is the total number of sell initiated trades. We approximate buy/sell initiation as follows: if in a given 2 minute interval, the end price is greater than the start price, the trades within that interval are classified as buy (and vice versa).

The features above alone are not enough to build a predictive model. We extend by looking at 4 dimensions of these features. To predict today’s returns of a given stock, we analyze the following:

- 1) Yesterday’s feature set of given stock
- 2) Historical long run average feature set of given stock
- 3) Yesterday’s feature set as a percentile rank compared to the given stocks’ historical feature set (gauge for anything abnormal)
- 4) Yesterday’s feature set as a percentile rank compared to yesterday’s peer feature set (compare this stock’s features against all other stocks)

We excluded VWAP as a predictive feature, leaving us with 5×4 (dimensions) = 20 features for classification modelling.

We classify stock returns into three categories for classification:

- 1) 0 Negative: returns < -0.001
- 2) 1 Neutral: $-0.001 \leq \text{returns} \leq +0.001$
- 3) 2 Positive: returns $> +0.001$

Then we trained a two-layer neural network model, where each layer contains a linear classifier and between the two layers there is a rectified linear unit $f(x) = \max(0, x)$.

The first layer consumes all the 20 features and output 10 intermediately features, and the second layer consume that 10 features and is supposed to output the 3 labels above. Our loss function is the Cross Entropy function

V. EVALUATION / RESULTS

Since the main objective of CS 498 Cloud Computing Application is to understand cloud computing applications and

their successful deployment, we emphasized the preliminary evaluation results on demonstrating our ability to successfully create and deploy the cloud computing technologies listed in the Method Design section of this report.

The various stages of deployment that we have completed so far:

- 1) Using IAM to create users and apply roles and permissions for accessing the applications

User name	Groups	Access key age	Password age	Last activity	MFA
admin	CCA	3 days	3 days	None	Not enabled
fan	CCA	3 days	3 days	None	Not enabled
marzan	CCA	3 days	3 days	3 days	Not enabled
wang	CCA	3 days	3 days	Today	Not enabled

- 2) Creating Roles for enabling application and user access to applications
- 3) Creating S3 bucket for data storage and access
- 4) IAM Role for S3 Bucket access by other applications used in this project
- 5) Create EMR cluster (see Figure 1)
- 6) Configuration for Python Jupyter Notebook to connect to EMR and S3

Create notebook

Name and configure your notebook

Name your notebook, choose a cluster or create one, and customize configuration options if desired. Learn more [\[?\]](#)

Notebook name* emr_notebooks

Names may only contain letters (a-z), numbers (0-9), hyphens (-), or underscores (_).

Description

256 characters max.

Cluster* Choose an existing cluster emr_cluster_j-1589WWWWHA1N1 Create a cluster

Security groups Use default security groups Choose security groups (vpc-030b0d5d9f756146)

AWS service role* EMR_Notebooks_DefaultRole

Notebook location* Choose an S3 location where files for this notebook are saved. Use the default S3 location s3://vpc-emr-resources-322925635065-us-east-1/notebooks/ Choose an existing S3 location in us-east-1

Tags

* Required

[Cancel](#) [Create notebook](#)

- 7) Run PySpark job on Jupyter Notebook (see Figure 2)

A. Random Batch Backtests

We randomly sample 100 stocks from our database, to run our backtest across the full history.

We found our logistic regression model to be lacklustre in performance. In our training sample, it yielded an accuracy of 52.5%. The confusion matrix as follows:

TABLE I
IN-SAMPLE CONFUSION MATRIX (TRAINING) - RANDOM BATCH

Predicted	Actual		
	Negative	Neutral	Positive
Negative	603	105	551
Neutral	137	9622	164
Positive	4884	7641	4693

This degraded to 43.2% in the test set.

TABLE II
OUT-OF-SAMPLE CONFUSION MATRIX (TEST) - RANDOM BATCH

Predicted	Actual		
	Negative	Neutral	Positive
Negative	35	7	34
Neutral	14	420	10
Positive	401	670	409

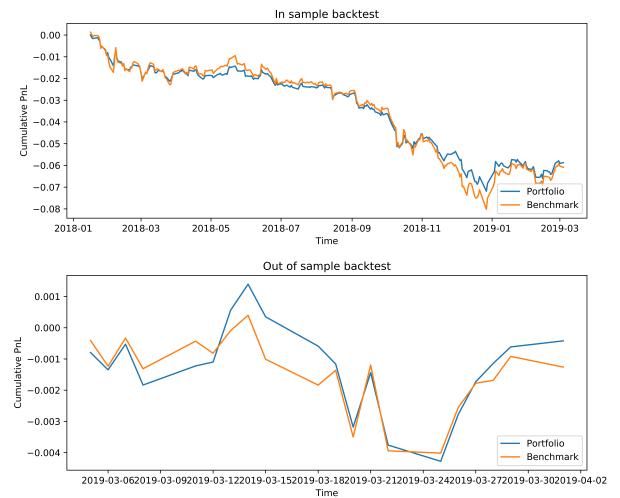


Fig. 3. Training and Test - Profit and Loss

B. Full Sample Backtests

When we ran our prediction model across all the stocks over all the dates in the public dataset. Our logistic regression model performed marginally better, In our training sample, it yielded an accuracy of 67.5%, and in our test sample, it yielded an accuracy of 60.0%. We find that whilst it is better at predicting neutral, reasonable at predicting negative states, it has a poor record at predicting positive states.

TABLE III
IN-SAMPLE CONFUSION MATRIX (TRAINING) - FULL BACKTEST

Predicted	Actual		
	Negative	Neutral	Positive
Negative	26315	4557	23716
Neutral	115279	516430	116790
Positive	1220	254	1147

TABLE IV
OUT-OF-SAMPLE CONFUSION MATRIX (TEST) - FULL BACKTEST

Predicted	Actual		
	Negative	Neutral	Positive
Negative	1680	322	1545
Neutral	9783	32312	10951
Positive	78	12	57

VI. DISCUSSION

Much of the team discussions focused on selecting the appropriate technologies that can help us achieve our objective of

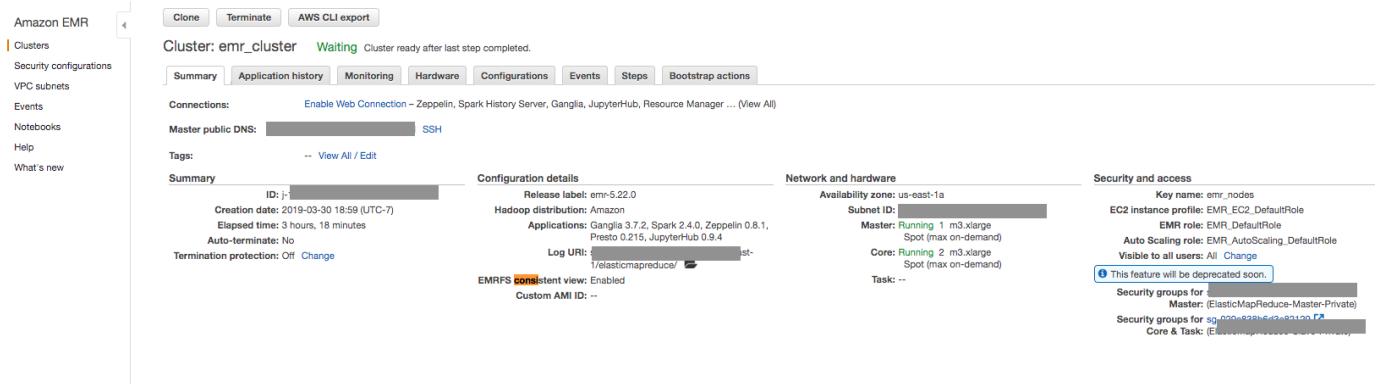


Fig. 1. EMR Cluster

```
In [4]: sc
A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

<SparkContext master=yarn appName=livy-session-0>

In [5]: rdd = sc.textFile('s3n://emr.bucket.test.capacity/2018-07-03_BINS_XETR15.csv')

In [6]: for x in rdd.collect(): print (x)
Progress for collect at <stdin>:1          Job Progress: 2/2 Tasks Complete
Stage [ID]: name at [source]:[line]      Status      Task Progress      Elapsed Time (seconds)      Failed Task Logs
Stage [1]: collect at <stdin>:1      COMPLETE      2/2      12.12

ISIN,Mnemonic,SecurityDesc,SecurityType,Currency,SecurityID,Date,Time,StartPrice,MaxPrice,MinPrice,EndPrice,TradedVolume,NumberOfTrades
"DE0005200000","BEI","BEIERSDORF AG O.N.", "Common stock", "EUR", 2504906, 2018-07-03, 15:00, 97.46, 97.46, 97.46, 97.46, 75,2
"DE0007251803","SAZ","STADA ARZNEIMITT. NA O.N.", "Common stock", "EUR", 2505089, 2018-07-03, 15:00, 80.08, 80.08, 80.08, 80.0
8,28,1
"DE000SHL1006","SHL","SIEMENS HEALTH.AG NA O.N.", "Common stock", "EUR", 3058562, 2018-07-03, 15:00, 35.465, 35.465, 35.465, 3
5.465,200,1
"DE0005557508","DTE","DT.TELEKOM AG NA", "Common stock", "EUR", 2504954, 2018-07-03, 15:00, 13.495, 13.495, 13.495, 13.495, 127
76,4
"LU1704650164","BFSA","BEFESA S.A.ORD.REG. EO 1", "Common stock", "EUR", 2759536, 2018-07-03, 15:00, 45.45, 45.45, 45.45, 45.45, 45.4
```

Fig. 2. Running PySpark on Jupyter Notebooks

developing a successful application that is ready for real world usage. Cost and the computing power of the technologies was also an item of discussion. AWS can be quite expensive if we don't select the proper technologies that can efficiently execute the tasks needed. For example, PySpark framework built on EMR was chosen to achieve efficiency and speed that will not only help us reduce cost but deliver fast results to the users.

VII. FUTURE WORK

The project's initial setup and testing of the applications' seamless integration with each other has been completed. Now the framework or the infrastructure is in place, next steps include creating or selecting the appropriate predictive algorithms to help us build a successful model. After the model is suc-

sfully created, the project will move into production phase, interconnected applications checked for seamless integration and development of the front end user interface by using Amazon's Route 53 application.

VIII. DIVISION OF WORK

Fan Yang and Wang Chun Wei, contributed to developing the Python helper codes for connecting to and downloading the data into S3. Along with their prior Machine Learning knowledge, and in collaboration with Marjan Ahmed and Dilruba Hawk, the dataset was chosen and initial model development strategy such as selecting the appropriate Machine Learning model (for example, regression) were chosen. Marjan Ahmed and Dilruba, along with inputs from Fan Yang and Wang Chun

Wei, created the Amazon account, created users, roles and permissions using IAM and created EC2 and S3. Fan Yang also tested the jupyter notebook and ran a computing job using the EMR cluster. Dilruba will help with developing the front end web application using Route 53 for this project.

REFERENCES

- [1] Jegadeesh, N., and S. Titman (1993) Returns to buying winners and selling losers: Implications for stock market efficiency, *Journal of Finance*, 48, 65 - 91
- [2] Vidyamurthy, G. (2004) *Pairs Trading: Quantitative Methods and Analysis*, New Jersey: John Wiley & Sons.
- [3] Zhang, H. and Zang, Q. (2008) Trading a meanreverting asset: Buy low and sell high. *Automatica* Vol. 44, 1511-1518