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 commerically viable trading strategy.

II. Dataset

The chosen dataset is the Deutsche Borse 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	
DE000A0D6554		NDX1	NORDEX SE O.N.	Common stock	EUR	
DE000A0WMPJ6		AIXA	AIXTRON SE NA O.N.	Common stock	EUR	
DE000A1EWWW0		ADS	ADIDAS AG NA O.N.	Common stock	EUR	
DE000A1ML7J1		VNA	VONOVIA 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 applictions, 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 predicivte models and alogrithms 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 clusters[1] to analyze the data. We use boto3 package in Python to upload stock price data from the Deustche public dataset to S3[2]. 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 brought 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.

[1]We created a 3-nodes cluster following the AWS Cluster creation process.
[2]See 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 $\times$ 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 $\leq$ returns $\leq$ +0.001
3) 2 Positive: returns $>$ +0.001

We run a logistic regression model to train the features on the classification labels. We also tried implementing a neural network model (and a simplier 2 layer fully connected model), however, these did not outperform the simplier logistic model.

Our trading strategy is simple. At the beginning of a given trading day, we use historical features to try and predict the total returns for all the stocks at the end of the day. For stocks where we predict a negative outcome, we short (short sell) it with a k percent weight. For stocks where we predict a positive outcome, we long (buy) it with a k percent weight. For stocks where we predict a neutral outcome, we refrain

from investing. We define $k = 1/\#$of stocks. Hence, each position is equal weighted. For our benchmark strategy, we will invest (long) k percent weight to all of the stocks, i.e., a classic equal weighted portfolio. We will attempt to beat this equal weighted portfolio with our algorithmic trading strategy.

V. Evaluation / Results

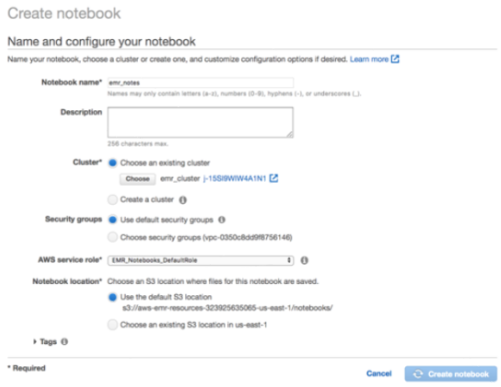
A. EMR PySpark Setup

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 demostrating 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

2) Creating Roles for enabling application and user access to applictions
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



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

B. 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 as follows:

This degraded to 43.2% in the test set.

TABLE I
In-sample Confusion Matrix (Training) - Random Batch

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

TABLE II
Out-of-sample Confusion Matrix (Test) - Random Batch

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

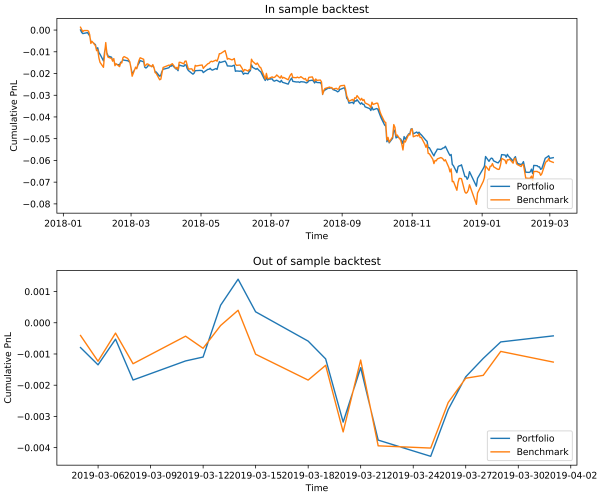


Fig. 3. Training and Test - Profit and Loss

C. Full Sample Backtests

When we ran our prediction model across all the stocks (2,837) over all the dates (2nd Jan, 2018 to 1st Apr, 2019) 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

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

We plot the performance of our model vs the benchmark, which is an equal weighted portfolio of all the stocks listed on the Deutsche Bourse. We find reasonably good performance

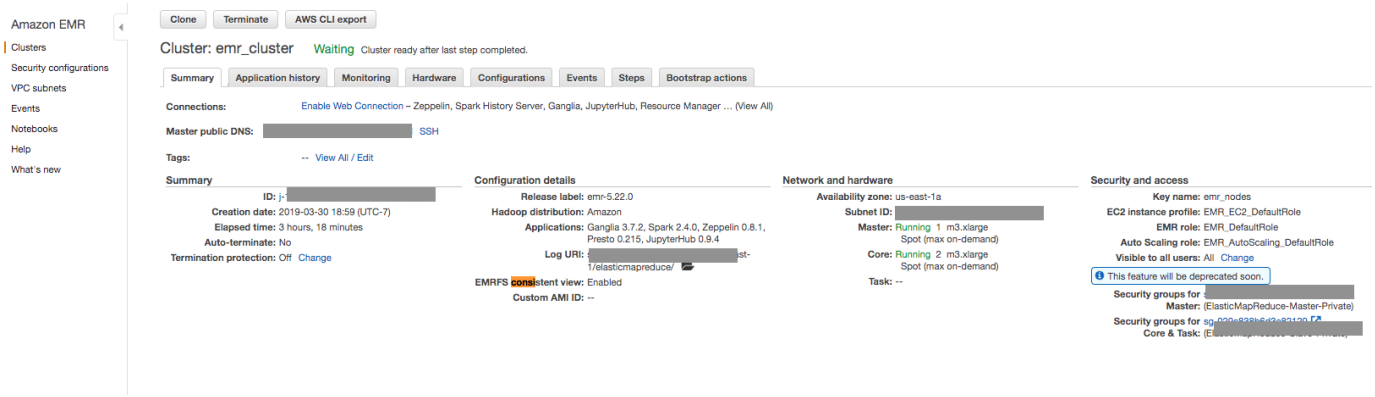


Fig. 1. EMR Cluster

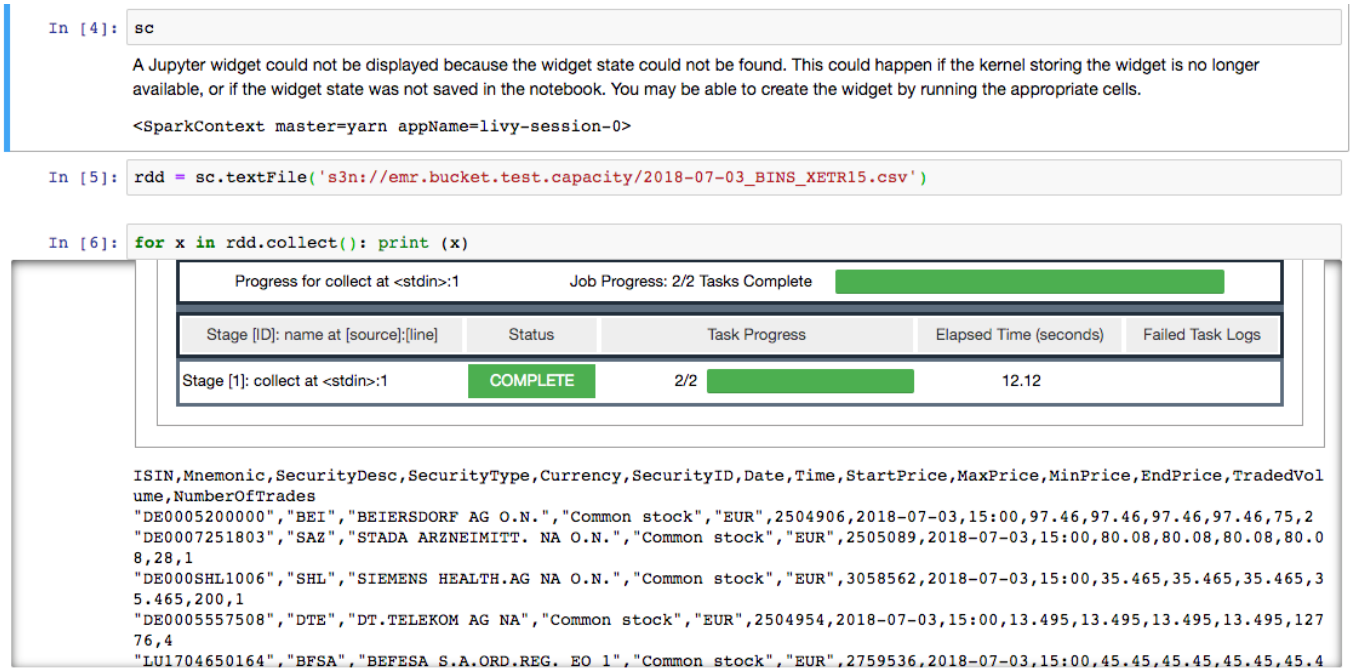


Fig. 2. Running PySpark on Jupyter Notebooks

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

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

of our performance in-sample (on the trainings set), despite not being very good at predicting positive movements. This is in part due to the fact that the German DAX had a poor run during 2018. Our trading strategy also outperformed in the test set, yielding a portfolio with lower volatility and high returns than the benchmark equal weighted portfolio. However, these results do not include trading costs. Since we are trading on a daily basis, it is most likely that these results would become negative after applying broker transaction costs.

TABLE V
PORTFOLIO PERFORMANCE

	Training Set		Test Set	
	Portfolio	Benchmark	Portfolio	Benchmark
Mean	1.79%	-2.47%	1.47%	0.08%
Standard Deviation	0.73%	2.62%	0.56%	2.01%
Sharpe Ratio	2.45	- 0.94	2.63	0.04

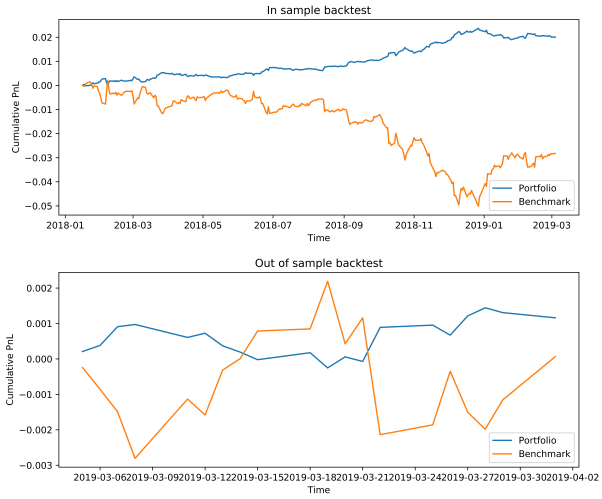


Fig. 4. Training and Test - Profit and Loss (Full Sample)

VI. Discussion

Much of the team discussions focused on selecting the appropriate technologies that can help us achieve our objective of developing a successful application thats 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 dont select the proper technologies that can efficient 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 projects initial set up and testing of the applications seamless integration with each other has been completed. Now the framework or the infratructre is in place, next steps include creating or selecting the appripriate predictive algorithms to help us build a successful model. After the model is successully created, the project will move into production phase, interconnected applications checked for seamless integration and development of the front end user interface by using Amazons 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 Macine Learning knowledge, and in collaboartion 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 suing 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