AIT 614 Optional Project Report

Proposing a data pipeline capable of ingesting streaming *Yahoo Finance* data and generate machine learning based insights on USO stock prices

Submitted by -

Abhishek Misra (G# 0101197)

Arun Reddy Bollam (G# 01040932)

Contents

Introduction

Every economic sector is dependent on the demand and supply of crude oil, hence any changes in the price of the crude oil has a significant impact on the world economy. Malliaris and Malliaris [1] in one their study examined how oil impacts gold and they determined that the impact of oil on gold is more than vice versa. In another study by Dogrul and Soytas [2], they found out an interesting relationship between crude oil prices and unemployment. So, we can say understanding how crude oil price fluctuates and which factors affect the pricing is an important but complex task. It has been reported that though economic and market factors are the main causes for the fluctuations in the global oil prices [3], thus predicting the behavior of the oil price markets can provide some insights and helps in forecasting the future oil price. Haidar et. al [4], in their study used S&P 500 data to understand the fluctuations and proposed a forecasting tool to show that the oil spot price has significant explanatory power for predicting the crude oil spot price. In this project, we wished to build a machine learning model to predict the future crude oil prices using the USO data and understand which set of features are better in prediction using Hadoop framework.

1. USO:-

The United State Oil Fund, an exchange-traded fund (ETF) security, introduced in April, 2006 was designed to track daily price fluctuations of West Texas Intermediate light, sweet crude oil that is being delivered to Cushing, Oklahoma [5]. The objective of the USO is to monitor the daily changes in the percentage terms of its shares, Net Asset per Value (NAV) to reflect changes in percentage terms of the spot price of light sweet crude oil, which in turn is measured by the determining the changes in the price of the near month futures contract for the oil traded on the NYMEX (less USO's expenses). Since USO tracks the near-month contracts listed on NYMEX, the data can be used to understand short-term fluctuations in the crude oil market. The daily closing prices of USO are available on the Yahoo Finance website and can be retrieved to generate various meaningful insights [6].

2. APACHE Hadoop framework:-

Apache Hadoop is an open-source big-data framework providing a platform for handing large data sets through distributed storage and processing. The framework is based on the assumption that hardware failures are common and hence is designed such that it automatically takes care of all the possible system failures.

The ecosystem has Hadoop compatible File System(HDFS) and MapReduce at its core hence its very often called as Hadoop MapReduce framework [7]. This framework is a perfect example to showcase power of distributed and parallel computing. Applications to process vast amount of data can be easily written on Hadoop MapReduce framework. The data is processed on multiple machines (distributed architecture) parallelly (parallel computing) on Hadoop ecosystem. In a

nutshell, the objective is to distribute the tasks across multiple clusters, which have a compatible file system.

The core of Hadoop Ecosystem is composed of Hadoop Distributed File System(HDFS) and MapReduce:

- **Hadoop Distributed File System (HDFS)** - It is a file system architecture that enables storage and management of very large-scale data. Hadoop Distributed file system (HDFS) is a storage layer of Hadoop that stores the data which is then distributed and processing is done on this partitioned data by applications like MapReduce and others. Data is stored across machine clusters, one of which acts as a node [7]. HDFS follows write-once, read-many-times philosophy. HDFS can process data in petabytes. A very useful feature Hadoop processing provides is its ability to run flawlessly on commodity hardware (commonly available hardware from local vendors). Processing data over Hadoop can enable data stream processing because of its low latency throughput. Hbase currently offers variety of low latency transactions. HDFS consists of two types of nodes, a) a *namenode* (the master) and multiple *datanodes* (slaves). The name node stores the name of the files (Fig 1). Data nodes store and fetch concerned information when accessed by the client as shown below.

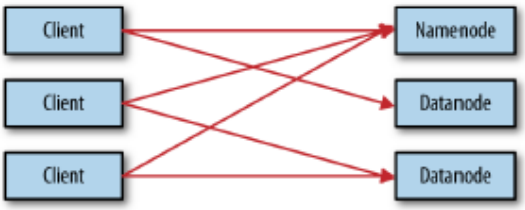


Fig1: Data Nodes and Name Node [7]

It interacts with the Hadoop through its command line interface and provides a streaming access to the file system data along based on permissions and authentication to access those files.

- **MapReduce:**
  MapReduce is a programming modality of Hadoop ecosystem specialized to process data on distributed clusters parallelly. Processing via MapReduce broadly includes a) Mapping and b) Reducing.

  Maps are functions which transform input records into intermediate records. *Map* class is a generic class and has four parameters. Mapper converts input key value pairs into intermediate key value pairs. To decide how many mappings are needed is an intelligent

question and the answer is dependent on the number and size of input files and the intended objective to be achieved.

Reducer is a function which merges the sorted outputs from in memory and on disk portions of the JVM , maintain their sorting order. A reducer partitions the data by the key of the key value pair. What combination of mapper and reducer will suit a particular problem is a sensitive problem which one should answer before data processing begins.

Shuffle is the name given to the process of selecting specific portion of output mappers, via HTTP to be handed over to the reducer.

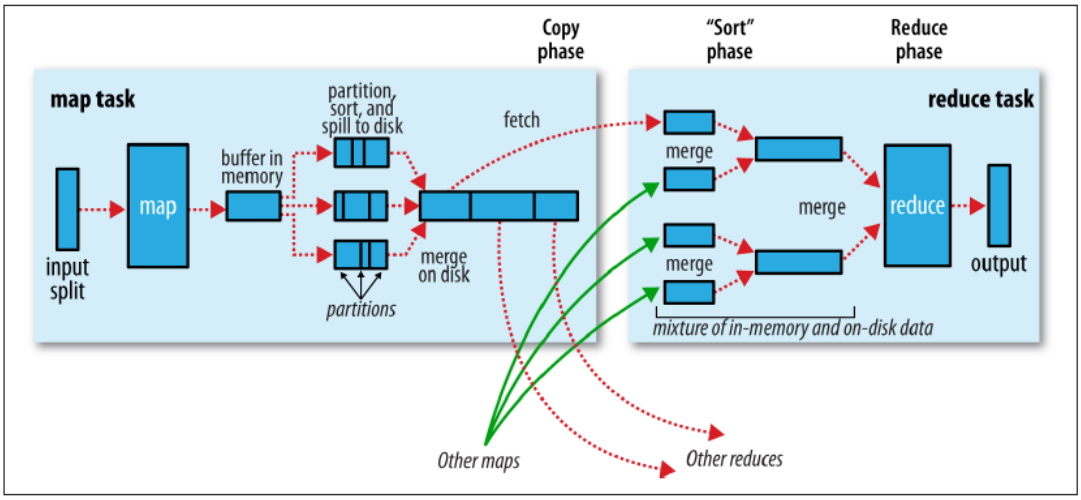


Fig 2. MapReduce Framework [7]

Other components of the Hadoop Ecosystem

Hadoop alone is not capable enough to handle all the tasks associated with the processing and management of the big data [8,9]. Besides the core components, many projects have been undertaken which support and enhance the core functionalities of the Hadoop making it possible for the organizations using Hadoop to personalize the framework according to their requirements.

These additional components of the Hadoop Ecosystem and their main purpose is shown in the figure can broadly be categorized as follows [10,11,12]–

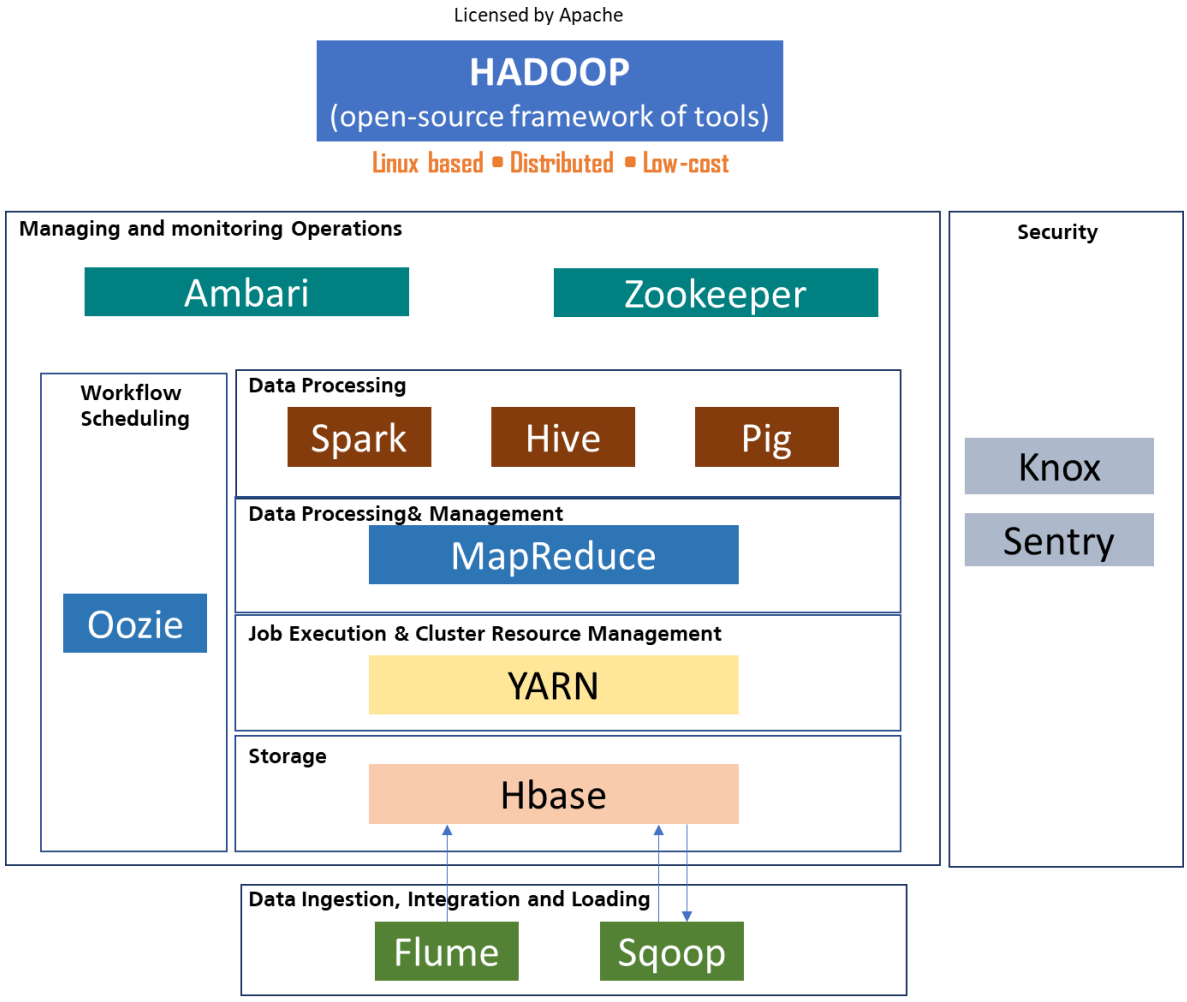


Fig 3. Concept Map of the Hadoop Ecosystem

Big Data and Hadoop Ecosystem

These components work together to process the big data. There are four phases of the big data processing including ingestion, processing, analysis and access. The first stage of the big data processing is ingesting, during which the data is ingested or transferred to Hadoop (HDFS) from relational databases (using Sqoop), services (event data using Flume) or desktop services. In the second stage, the data is stored in the distributed file system (HDFS) and the NoSQL data (HBase). Spark/MapReduce performs the data processing and then in the third stage data is queried and analysed using Pig or Hive. The fourth stage is access which is performed by tools such as Hue (web-interface) and Cloudera (search providing text based interface for exploring data) through with the analysed data can be accessed by the users.

3. Machine Learning in Spark

Spark is a leading tool in the Hadoop Ecosystem. MapReduce with Hadoop can only be used for batch processing and cannot work on real-time data. Spark can work stand-alone or over the Hadoop framework to leverage big data and perform real-time data analytics in a distributed computing environment. It was originally developed at University of California, Berkeley. It can support all sort of complex analysis including Machine Learning, Business Intelligence, Streaming and Batch processing by writing APIs in Java, Scala, Python, R. Spark is 100 times faster than Hadoop MapReduce framework for large scale data processing as it performs in-memory computations thus providing increased speed over MapReduce.

The big-data era has not only forced us to think of fast capable data-storage and processing frameworks but also platforms for implementing machine learning (ML) algorithms that has applications in many domains. With lot of ML tools available, deciding the tool that can perform analysis and implement ML algorithms efficiently has been a daunting task. Fortunately, Spark provides a flexible platform for implementing a number of Machine Learning tasks, including classification, regression, optimization, clustering, dimensionality reduction etc [13].

Spark Components/Modules:

At a top-level, Spark is divided into 2 modules : Spark Core and Spark Libraries/Extensions. [14]. In the core, spark consists of various packages and classes for performing trivial I/O operations, task scheduling, distribution, tracking, memory management etc. It is in the core that RDD are defined. Basically, the core of the Spark consists of Context and Config. Context is the entry point through which various resources of the Spark framework can be accessed and Config defines the Spark framework environment Spark Core is the underlying general execution engine for the Spark platform that all other functionality is built on top of it.

Spark Libraries/Extensions are the modules which are built over the core of the Spark to provide advances features which provides Spark the capabilities because of which it has gained its popularity. Spark MLlib is one of the popular ML module built on the top of the Spark framework that allows the businesses to perform statistical regression analysis, run classification, clustering algorithm on the data, enabling its users to do predictive analytics and generate insights.

Problem Statement

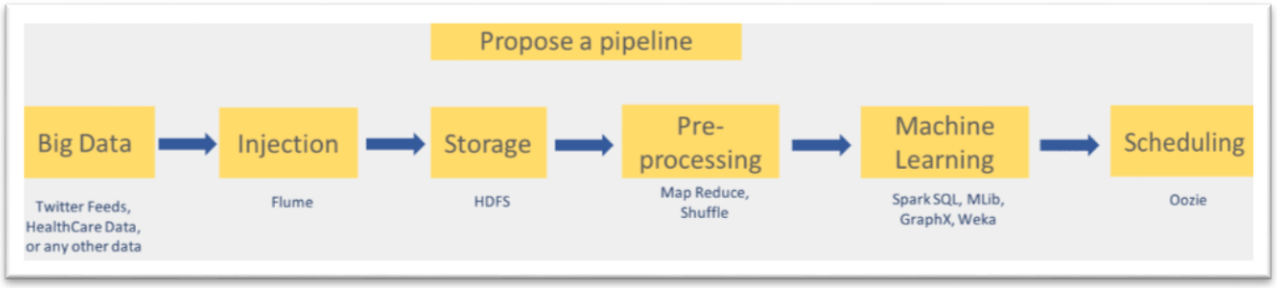
We wish to build a system which analyses US oil stocks to predict daily gains in US oil stocks based on real time data from *Yahoo Finance*. Our problem is about picking up all 13 stocks in US oil fund and divide their daily gain data into training and test data set to predict the stocks with high daily gains using Machine Learning module of Spark. Based on our analysis we propose a robust Cloudera-Hadoop based data pipeline to perform this analyses for any type and scale of data.

Significance of the problem under study

We wish to study a live stream data of US oil stock prices. Such study can help us better understand how does US Oil index affects the stock price of other stocks in US oil funds exchange. Such studies can help us predict the profitable stocks for stock traders and provide profits to US Oil stocks trader community.

Solution Approach

During project proposal, we positioned our approach to integrate multiple opensource modalities of Apache Hadoop ecosystem to build a cloud architecture which takes in real time data and process it to give valuable information to support decision making. Through this project we propose a cloud based Data Pipeline capable of ingesting massive real-time data from Yahoo finance server and generating Machine Learning based insights into global oil stock market. We divide the data into training and test sets and make our linear regression based learning model learn from the training data and then predict the correlation between stock prices based of coefficients in the regression model. We also calculate the R squared value and Mean Average Error to support our studies.

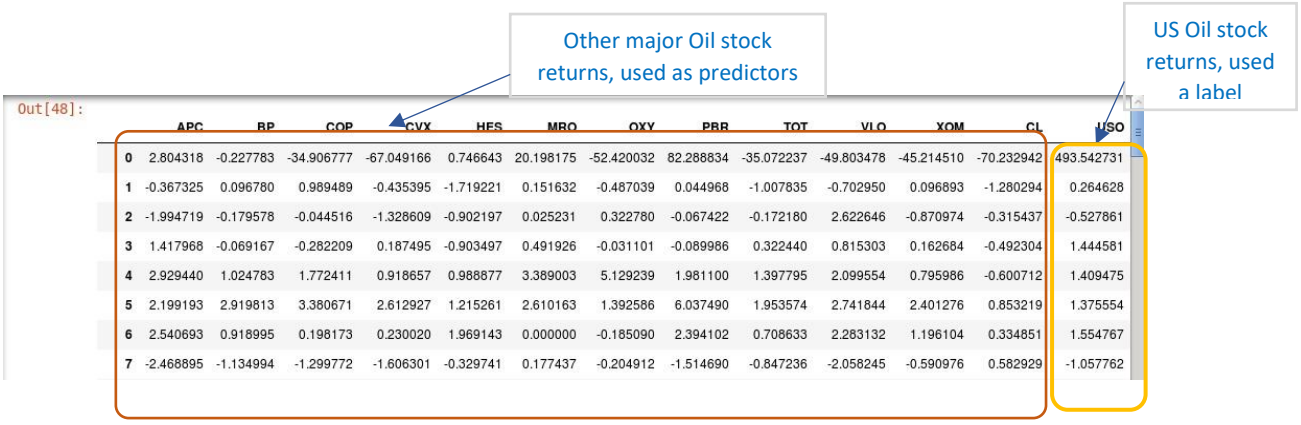


Implementation

Data Acquisition & Characterization
Data Injection
Storage
Pre-processing
Machine Learning

Data Characterization

Our data set included 13 oil stocks from S and P 500 stocks available on Yahoo Finance. We have studied prices of these stocks from April 4th, 2006 till date. Starting date so selected was based on the USO (US Oil Fund) being made available for trading oil stocks in the US. We have 13 different numerical variables of float data type. The data we processed contained 2944 rows and 13 columns. There was no null value present in the dataset. The data set is also available in CSV format for local analytics. The returns observed were actually obtained after multiplying with 100.



	APC	BP	COP	CVX	HES	MRO	OXY	PBR	TOT	VLO	XOM	CL	USO
0	2.804318	-0.227783	-34.906777	-67.049166	0.746643	20.198175	-52.420032	82.288834	-35.072237	-49.803478	-45.214510	-70.232942	493.542731
1	-0.367325	0.096780	0.989489	-0.435395	-1.719221	0.151632	-0.487039	0.044968	-1.007835	-0.702950	0.096893	-1.280294	0.264628
2	-1.994719	-0.179578	-0.044516	-1.328609	-0.902197	0.025231	0.322780	-0.067422	-0.172180	2.622646	-0.870974	-0.315437	-0.527861
3	1.417968	-0.069167	-0.282209	0.187495	-0.903497	0.491926	-0.031101	-0.089986	0.322440	0.815303	0.162684	-0.492304	1.444581
4	2.929440	1.024783	1.772411	0.918657	0.988877	3.389003	5.129239	1.981100	1.397795	2.099554	0.795986	-0.600712	1.409475
5	2.199193	2.919813	3.380671	2.612927	1.215261	2.610163	1.392586	6.037490	1.953574	2.741844	2.401276	0.853219	1.375554
6	2.540693	0.918995	0.198173	0.230020	1.969143	0.000000	-0.185090	2.394102	0.708633	2.283132	1.196104	0.334851	1.554767
7	-2.468895	-1.134994	-1.299772	-1.606301	-0.329741	0.177437	-0.204912	-1.514690	-0.847236	-2.058245	-0.590976	0.582929	-1.057762

USO Stock data (Yahoo Finance)

Data Injection

The Data was injected to HDFS using Flume. Three components worked in concert to push the data into flume, these three were Source, Channel and Sink. The source is accessed by the execution of tail command on shell, local memory acted like the channel and DFS was sink was the sink. The data injection needed configuring local database to HDFS.

The following config files were used.

Source fetches data from application server and provides to the channel

It facilitates the flow of data from source to sink. It temporarily stores the data before data gets pushed into sink

```
1  a1.sources = tail
2  a1.channels = MemoryChannel
3  a1.sinks = HDFS
4
5  a1.sources.tail.type = exec
6  a1.sources.tail.command = tail -F total_returns_output
7  a1.sources.tail.channels = MemoryChannel
8
9  a1.sinks.HDFS.channel = MemoryChannel
10 a1.sinks.HDFS.type = hdfs
11 a1.sinks.HDFS.hdfs.path = /user/cloudera/flume/events/
12 a1.sinks.HDFS.hdfs.rollSize = 0
13 a1.sinks.HDFS.hdfs.rollCount = 2000
14 a1.sinks.HDFS.hdfs.file.Type = DataStream
15 a1.sinks.HDFS.hdfs.writeFormat = Text
16 a1.sinks.HDFS.hdfs.serializer = Text
17
18
19 a1.channels.MemoryChannel.type = memory
20 a1.channels.MemoryChannel.capacity = 100000
21 a1.channels.MemoryChannel.transactionCapacity = 10000
22
23 a1.sources.tail.channel = MemoryChannel
24 a1.sinks.HDFS.channel = MemoryChannel
25
```

Sink is the final storage location.

The streaming data is read by executing the tail command

HDFS acts as a sink

Local machine memory acts as the channel

Configuration File

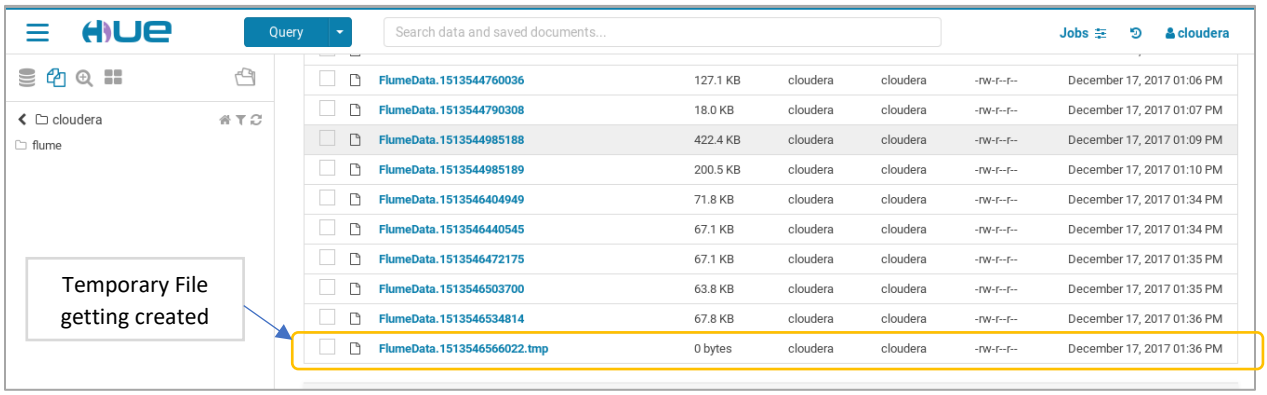
Stepwise transferring file from Flume to HDFS

```
17/12/17 13:34:00 INFO hdfs.BucketWriter: Creating /user/cloudera/flume/events//
FlumeData.1513546440545.tmp
17/12/17 13:34:30 INFO hdfs.BucketWriter: Closing /user/cloudera/flume/events//F
lumeData.1513546440545.tmp
17/12/17 13:34:30 INFO hdfs.BucketWriter: Renaming /user/cloudera/flume/events/F
lumeData.1513546440545.tmp to /user/cloudera/flume/events/FlumeData.151354644054
5
17/12/17 13:34:30 INFO hdfs.HDFSEventSink: Writer callback called.
17/12/17 13:34:32 INFO hdfs.HDFSSequenceFile: writeFormat = Text, UseRawLocalFil
eSystem = false
17/12/17 13:34:32 INFO hdfs.BucketWriter: Creating /user/cloudera/flume/events//
FlumeData.1513546472175.tmp
```

Execution of Flume Command

Storage

For storing the data HDFS on Cloudera was used. The directory address HDFS /Cloudera/flume/events/. The dataset wasn't replicated as we worked in single node. Also the copy of same data could be availed from Yahoo Finance if needed. Data always gets stored in multiple **sequential file** each entry represented only once(no duplication)

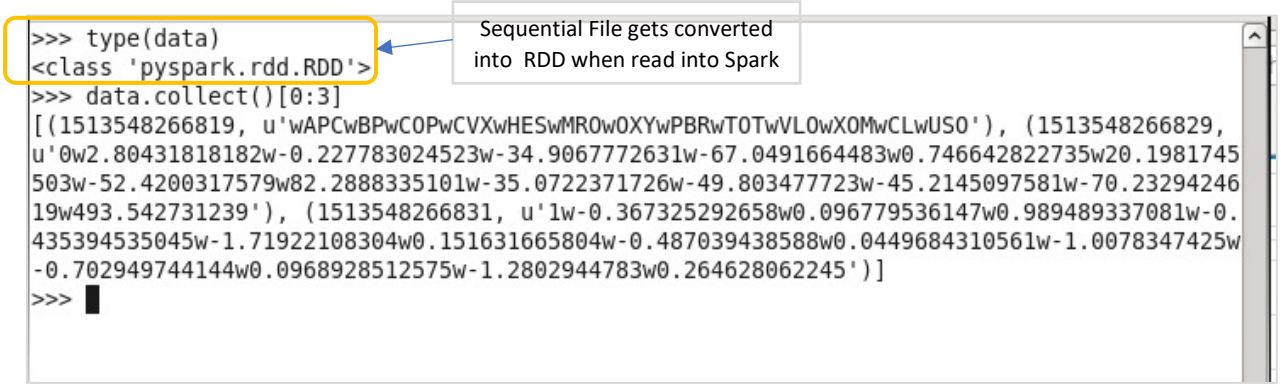


HDFS File Browser

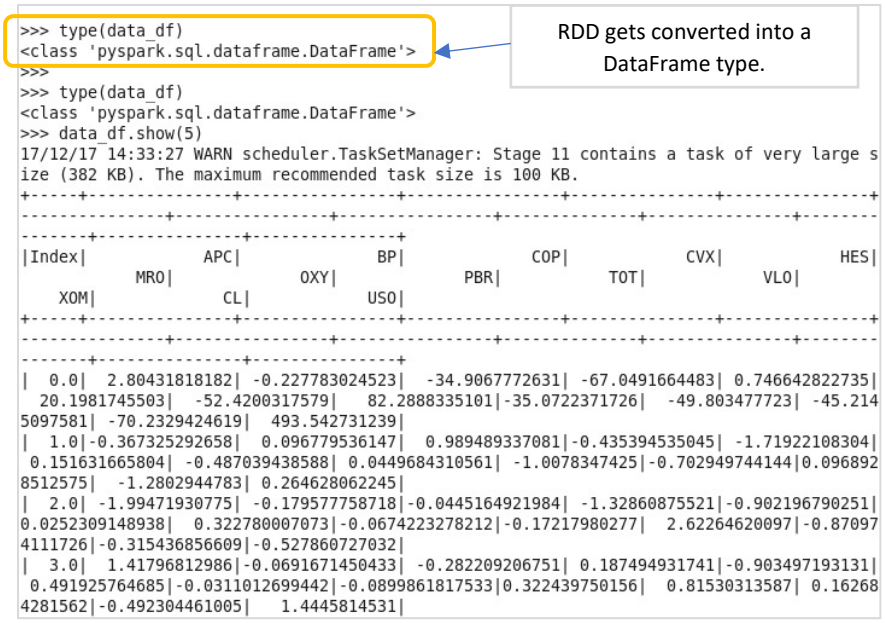
Pre-processing

PySpark which is the Python API for Spark was used to pre-process data. The sequence file is read into the spark using spark context. The sequential file is converted into a RDD (Resilient Distributed Datasets) . Converted the **rdd** into a data frame providing it with the suitable schema and specifying the data type of each input.

The input features are converted in to a single feature called features using the Dense Vector function. The dataframe which is ready to be fed into the machine learning has two columns label and the features.



Pre-processing in spark (A)

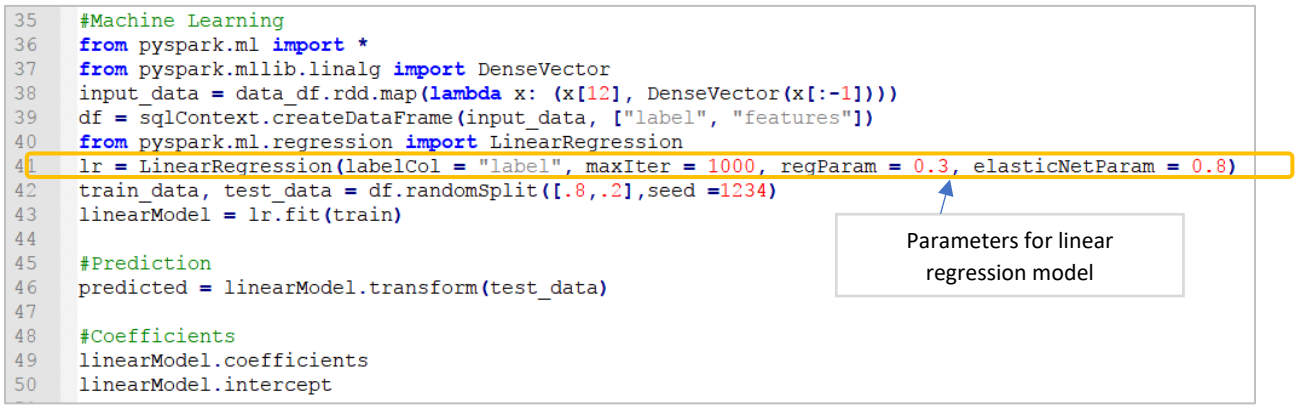


RDD gets converted into a DataFrame type.

Pre-processing in Spark (B)

Machine Learning

Machine was performed using Mlib function in SpARK. The data was split into training and testing data sets. Linear regression function is fit to the training dataset. The returns of the USO are predicted for the training dataset. Computed the mean squared error.



Parameters for linear regression model

Machine Learning in Spark

Results and Discussion

Model
Coefficients in the problem
R^2 value
MAE

I. Coefficients Regression Model

```
>>> linearModel.coefficients
DenseVector([0.0817, 0.0, 0.086, 0.0, 0.1282, 0.0675, 0.11, 0.0438, 0.0046, 0.0, 0.0, 0.0
])
>>> linearModel.intercept
-0.097938396250894261
>>>
```

Coefficients of the linear regression model

II. Predictions from the test data

```
>>> predicted.show()
17/12/17 14:45:47 WARN scheduler.TaskSetManager: Stage 64 contains a task of very large s
ize (382 KB). The maximum recommended task size is 100 KB.
+-------------+--------------------+-------------------+
|        label|            features|         prediction|
+-------------+--------------------+-------------------+
|-10.6845637584|[-4.9491097123,-3...|-2.6049027160660203|
|-10.5788227651|[-14.9250601872,-...|  -7.05498232474393|
| -9.5009208688|[-10.8727159385,-...| -3.899249074949655|
|-6.85363670134|[-3.64364154888,-...|-2.8975727636321627|
|-6.16939863769|[-0.142124449896,...|-2.4136768156571526|
|-6.09017958228|[-0.917425618212,...|-2.8860387888978787|
|-5.87589268951|[-1.21392019448,-...|-0.8646688220688936|
|-5.61224489796|[-6.92800650143,-...| -2.511037421566081|
|-5.32407407407|[-7.01621817375,-...|-1.7950806326990618|
|-5.28301886792|[-2.399253324,-1....|-1.7028769097237988|
|-5.22689712521|[-8.40440721057,-...| -7.102904756079641|
|-4.73677118918|[-12.9641017082,-...| -5.247946936700436|
|-4.37581481625|[-0.718096102365,...|-0.9344098177877528|
|-4.26355830479|[-2.96419896126,-...|-2.2629922880359676|
|-4.22997946612|[-6.29743073524,-...|-2.5846843747520647|
|-4.20672693136|[-2.70303261253,-...|-2.8176268982051402|
|-4.08275724138|[-2.96297125506,-...| -2.092221803264573|
|-4.07969639469|[-3.58258047202,-...|-2.5118325049966845|
|-4.01875740058|[-6.10991203702,-...| -2.862602079575361|
|-3.74254016014|[-4.57499240557,-...|-2.6235364472490303|
+-------------+--------------------+-------------------+
only showing top 20 rows
```

Predicted returns on USO stocks

III. Evaluation Metrics: R-squared value and MAE

```
>>> predicted = linearModel.transform(test_data)
>>> evaluator.evaluate(predicted,{evaluator.metricName:"r2"})
17/12/17 15:42:44 WARN scheduler.TaskSetManager: Stage 541 contains a task of very large
size (382 KB). The maximum recommended task size is 100 KB.
0.037933003920849973
>>> evaluator.evaluate(predicted,{evaluator.metricName:"mae"})
17/12/17 15:43:21 WARN scheduler.TaskSetManager: Stage 542 contains a task of very large
size (382 KB). The maximum recommended task size is 100 KB.
1.9936926809855011
```

R squared value

100 * Mean Average Error

From the model we see not all coefficients are positive which means that US oil stock prices are not positively correlated with the other US oil stocks used as predictors in the study. The Model was built using regularization parameter =0.3. The R squared value was calculated using evaluation function from regression evaluator package from Ml module of Spark. It explains 4% of USO stock price variation. The Mean Average Error was found to be 1.9% which suggests that linear model of regression is not suitable for predicting stock return margins from data with high dimensionality.

Conclusion & Future Direction

Through our work we were able to propose a system to help identify stocks with positive daily return margin, and which can be recommended to be potential stocks for enhanced trading. Such system will act as a Hadoop based Data Pipeline to learn from past data and decide based on streaming updates which US oil stocks are profitable to trade in. We also find scope of improvements to our study which we discuss in future directions section

We intend to continue our study further by automating the analysis using scheduling module Oozie, and obtain periodic recommendations for trading US oil stocks. We also plan to test Neural Network model based learning rather than linear regression to accurately predict the US oil stock prices.

Bibliography

1. Malliaris, A., & Malliaris, M. (2009). Time series and neural networks comparison on gold, oil and the euro. *2009 International Joint Conference on Neural Networks*. doi:10.1109/ijcnn.2009.5178780
2. Doğrul, H. G., & Soytas, U. (2010). Relationship between oil prices, interest rate, and unemployment: Evidence from an emerging market. *Energy Economics*, *32*(6), 1523-1528. doi:10.1016/j.eneco.2010.09.005
3. Gabralla, L. A., Jammazi, R., & Abraham, A. (2013). Oil price prediction using ensemble machine learning. *2013 INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE)*. doi:10.1109/icceee.2013.6634021
4. Haidar, I., Kulkarni, S., & Pan, H. (2008). Forecasting model for crude oil prices based on artificial neural networks. *2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. doi:10.1109/issnip.2008.4761970
5. USO - USCF Investments. (n.d.). Retrieved from http://www.uscfinvestments.com/uso
6. USO Historical Prices | United States Oil Fund Stock - Yahoo Finance. (n.d.). Retrieved from https://finance.yahoo.com/quote/USO/history?p=USO
7. White, T. (2011). Hadoop: the definitive guide. Sebastopol, CA: OReilly.
8. Allouche, G. (n.d.). Hadoop 101: An Explanation of the Hadoop Ecosystem - DZone Big Data. Retrieved on Oct 18, 2017 from https://dzone.com/articles/hadoop-101-explanation-hadoop
9. BMC. (n.d.). Hadoop Ecosystem and Components - BMC Software. Retrieved on Oct 18, 2017 from http://www.bmc.com/guides/hadoop-ecosystem.html
10. Hortonworks Inc. (n.d.). Apache Hadoop Ecosystem and Open Source Big Data Projects | Hortonworks. Retrieved on Oct 19, 2017 from https://hortonworks.com/ecosystems/
11. Roman, J. (n.d.). The Hadoop Ecosystem Table. Retrieved on Oct 19, 2017 from https://hadoopecosystemtable.github.io/
12. Simplilearn. (n.d.). Ecosystem And Its Components | Big Data And Hadoop Tutorial For Beginners [Video file]. Retrieved on Oct 18, 2017 from https://www.youtube.com/watch?v=R2_76yzYnNw
13. Apache Spark™ - Lightning-Fast Cluster Computing. (n.d.). Retrieved from https://spark.apache.org/