Currency Trend Analyzer

Tim Paine
Department of Computer Engineering
Columbia University
tkp2108@columbia.edu

Mark Aligbe
Department of Computer Engineering
Columbia University
ma2799@columbia.edu

Abstract— **We present a modular, extensible, open source alternative to financial data aggregation tools like Bloomberg Terminal. Our program, written fully in Java, aggregates data feeds, visualizes time series, and leverages Apache Mahout to provide insight and trading suggestions, all in real time at milliseconds granularity. Without the expensive price tag of custom data aggregation tools, and with easy expansion to include custom trading strategies and data from markets other than currency pairs, we believe our platform is a feasible competitor to the tools of major financial information companies.**

Keywords-Visualiziation; Big Data; Analytics; Real time; Data aggregation; Financial Markets; Currency Markets; Forex Trading; Bloomberg Terminal

I. INTRODUCTION

Every day, currency markets generate millions of data points at millisecond granularity. Traders use this data to make trading decisions, executing trades at points in time where they believe they can make a profit off the difference between the price their clients are willing to pay, and the price they receive. Independent traders can make money by exploiting trends in price movement. This trading requires the brokers to have access to not only the data itself, but also technical indicators and real time analytics which can give them insight. Thus, a large amount of processing must be done on large quantities of data, all in real time.

The problem of processing and analyzing these large quantities of data in real time is well served by the tools of big data analytics. We propose and implement an open source, real time streaming, visualization, and analytics engine for processing currency data. Our system is able to aggregate multiple streams of currency data, visualize it with updating graphs, and leverage Apache Mahout to identify currency trends and attempt to predict future price movement.

II. RELATED WORKS

The most obvious competitor to an open source system is the Bloomberg terminal. Businesses on both buy and sell side pay upwards of $20,000 per user for this service. But while Bloomberg is an incredibly powerful tool for equity markets, it has significantly less usage for currency markets. The Bloomberg currency platforms provides technical indicators, as well as current and historical prices, both of which we implement to a degree.

Reuters Eikon is a competitor to the Bloomberg Terminal, at a substantially reduced cost. However, it is fundamentally the same concept, more suited towards equity analysis than other markets.

Many independent broker sites provide the technical analysis services of Bloomberg and Reuters, but for a fee. Many services will give you price data, but almost none give you granularity of less than hours.

Our open source platform is well suited to this problem. Its extensibility makes it easy to plug in new or custom technical indicators. Its machine learning engine, built on Mahout, makes it both scalable, and easy to add new machine learning based prediction algorithms.

III. SYSTEM OVERVIEW

Our system provides streaming data aggregation, visualization, and analytics. It is modular, extensible, and scalable, built in java on top of Apache Mahout. It consists of three primary components. A backend aggregates data feeds and sends price data at given intervals to our system. A visualization engine takes the feeds, and plots them on time-updating graphs. Also plotted are technical indicators, including moving averages, standard deviations, and Bollinger bands. The system is extensible, and adding additional technical indicators is simple. Finally, we have a price prediction engine that leverages Mahout's machine learning algorithms to predict price movement.

The first component is the backend data aggregator. Prices are loaded from CSV files (for simulation or backtesting of prediction strategies), or connected directly to broker APIs. For our testing, we limit ourselves to seven major currency pairs: EURUSD, USDJPY, USDCAD, USDCHF, AUDUSD, NZDUSD, and GBPUSD. For consistency, these are all converted to a USD based metric (i.e. USDJPY, USDCAD, and USDCHF are converted to JPYUSD, CADUSD, and CHFUSD, respectively). Over the course of a given time interval (configurable from milliseconds to hours), price information is collected from each of the data feeds. At the end of the time interval, the aggregated data is sent to a separate module, which holds

the most up to date data for each currency pair. This data can then be fetched from the main application for analysis and visualization.

The visualization engine is currently built on JFreeGraph. Each of the price time series is plotted, as well as the major technical indicators. Currently supported are Simple moving averages, standard deviations, and Bollinger Bands. We hope to implement additional technical indicators, but since our platform is very extensible and this is not the primary value point, we opted to only implement a subset of the most commonly used indicators. Prices from feeds are fed into Time Series objects, as well as separate arraylist, depending on the needs of the technical indicator. For example, the 10-tick Simple Moving Average graph maintains a list of the last 10 price ticks, which are used to calculate the simple moving average. We are building a more modern V2 GUI, which is still in an experimental stage. It is meant to ease the transition to a web based application.

We arrange the GUI to be windowed, in order to allow the user to select which windows they want to have open. This would be familiar to anyone with experience using the Bloomberg terminal. The GUI itself is written in Java Swing, giving it a uniform, professional, albeit somewhat dry look. An updated and web-safe form of the GUI is being developed in JavaFX. This will allow us to embed the user interface into a web application, making our application easy to tie into existing ones.

Finally, we use Mahout to make predictions about price movement. From recent price data (the previous minute/hour/day), we build a set of calculated values, such as the average of the past prices, the average of the past ten differences between a price and its previous price, etc. These are used as a training set, as we assume correlation between past price movement and future price movement. I.e. if a price had large negative price movement in the last ten ticks, and we believe the price to be moving according to a mean reversion strategy, this will indicate a strong probability of future price growth, and indicate a buy. The details of this algorithm are given in the following section.

IV. ALGORITHM

Wise men often say look to the past to see the future. Our Buy/Sell prediction takes that to heart in its analysis. With the wealth of knowledge of previous trends, we can make extremely accurate predictions about whether or not to sell based on previous trends.

The algorithm takes in two lists of trades in a market; the recent history of the market (e.g. the test set), that you want it to decide on, and the entire history of the market, sans the test set (e.g. the train set). The more training history you give it, the more judicious it can be in what it considers similar, and the more accurate its final decision will be.

The first task it performs is it computes scores of ranges of trades. The score is the sum of the difference between the ith item and the first item (i does not include the first item), multiplied by the r2 of the least linear regression. of the range. For example, the range:

{1.0, 2.0, 3.0, 4.0, 5.0}

Has a score of 10, which indicates an increasing market. A range like this, though:

{1.0, 2.0, 3.0, 4.0, 5.0, 1.0}

Has a score of 1.071. The range is heavily penalized for having a poor regression, and thus being unreliable, even though its difference sum is exactly the same (10).

The second task is to normalize all the scores, and feed them to Mahout for processing. In order for Mahout to utilize the values, we have to correlate the scores in some way. The way this is approached is by the following:

For every two scores:

- We compute the percent difference between the scores. A larger percent difference therefore means that two ranges are more dissimilar.
- We then subtract this score from -1 (1) for negative (positive) results, which gives us the exact opposite semantic: more similar scores are closer to 1 (anti-similar scores are closer to -1, i.e. the ranges trend in opposite directions).

This gives us n^2 scores, as each range has been correlated with every other range. We give this to a Mahout ItemSimilarity, which then considers the similarity matrix of all ranges.

We then ask Mahout to score the similarity of all ranges to the test range. We sort by most similar, and only filter results that are trending in the same direction (due to the small variance, some ranges may be erratic and thus be similar because of poor correlation, even though they are trending in opposite directions. Recall that the similarity of two ranges is a function of their similarity, and the other ranges they are similar to).

Once we have the most similar ranges, we score a small range of the market ahead of that range, and see which direction the market went in.

- If, for a majority of those ranges, the market went up, we say SELL (you're going to lose money otherwise)!

- If, for a majority of those ranges, the market went down, we say BUY (you're going to be rich soon)!

- Otherwise, just hold (the market is too unpredictable in similar scenarios).

We can also approximate how many ticks of the market this decision will be valid for. Going back to Cherry Picking, we peek into the future to see whether to buy or sell. By continuing to look forward into the future, we can see how long this trend will last.

To decide how long to maintain a decision, we continue evaluating the future market, until we run out of history (we reach the present), or the trend is broken. We maintain how long each similar range lasted for, average their durations, and return that as our decision.

V. Software Package Description

The software package is relatively easy to deploy. There are 4 packages of concern. The config folder holds all of the configuration files for the data streaming. This includes directory information for the CSV files we use to stream from.

The data folder holds the actual CSV files. Because of the large amount of data, we leave this folder empty to make the software package a manageable size. A single day's worth of data, 20+ MB, is available as a separate download.

The data_streamer folder holds all of the logic for aggregating and processing feeds. The data stream are implemented as Feed objects, and the time window logic is controlled by the Timer class. This in turn is run by the Market object, which holds the most up to date prices.

Finally the graph folder contains the main application, ForexTrendAnalyzer. This contains all of the visualization and analytics logic for the application.

Our application relies on a number of external JARs as well. We require hamcrest, jcommon, jfreechart, jfxrt, mahout, opencsv, and orsoncharts.

For the experimental V2 GUI, the main folder is located in data_streamer/fx. The main file is called ForexTerminal. This version will be easily ported to the web, which we will demonstrate on a live website.

VI. Experiment Results

Our system is fluid and stable, even in under the heavy strain of processing a significant amount of input data. The V1 UI is clunky, but the information is conveyed effectively. The multi window mode is convenient, and familiar to any Bloomberg users. The V2 UI, written with JavaFX, is much sleeker, but just as responsive as the V1 UI. We believe the user will find both versions comfortable and efficient for their trading stations.

Our suggestion algorithm performs fairly well for short term trades, often making suggestions on the 1-5 minute range. It excels with volatile, mean-reverting markets, and performs less well on stable or trend-following markets. We believe with further refinement, our simple Mahout strategy can be very profitable. We also hope the open source community will develop its own algorithms to plug into our platform, further improving the quality of the application.

Out prediction algorithm does a decent job at predicting price movement, depending on market trends. Following the price graph, we see that it is often able to predict price movements that occur shortly after the prediction is made. We hope to improve our algorithm, and add additional algorithms in future versions of the framework, as well as make it easier for users to add their own.

VII. Conclusion

We present a modular, extensible framework for aggregating, analyzing, and visualizing currency data streams, similar to Bloomberg, but fully open source. Our platform synchronizes multiple data streams to a timestep, collects metrics on the time series including commonly used technical indicators like simple moving averages and Bollinger bands, analyzes and predicts price movement using Apache Mahout, all in real time at milliseconds granularity. Because tools like Bloomberg Terminals can cost upwards of $20,000, we believe our platform will appeal to technologically oriented finance companies and start ups.

Furthermore, our platform is not limited to processing currency markets. Although we limit the scope of our project to currency data, the platform of aggregation, visualization, and analytics is highly extensible, and any time series can be plugged in. We hope to expand our model to other markets which do not rely as much on fundamentals, such commodity markets.

Finally, although our platform analyzes trends and provides price prediction, our algorithm is more of a proof of concept than tradeable implementation. We hope to further backtest and improve our strategies, as well as expand support for plugging in custom strategies. Because our platform is fully open source, we hope the community will expand and improve on the strategies we have written. Furthermore, we hope traders will plug in their own strategy to sit inside our platform.

Acknowledgment

The authors would like to thank Professor Lin, Ruichi Yu, and the rest of the teaching assistants

FOR EECS 6893 FOR THEIR HELP AND HARD WORK THIS SEMESTER.

REFERENCES

[1] S. Owen, R. Anil, T. Dunning, and E. Friedman, "Mahout in Action," Manning Publications Co., Shelter Island, NY, 2012.

[2] D. Loshin, "Big Data Analytics: From Strategic Planning to Enterprice Integreation with Tools, Techniques, NoSQL, and Graph", Morgan Kaufmann, Waltham, MA, 2013.

[3] K. Lien, " The Little Book of Currency Trading: How to Make Big Profits in the World of Forex," Wiley, December 28, 2010.

[4] D.J. Norman "Professional Electronic Trading," Wiley Finance, July 2002.

[5] Data freely available from HistData.com, a website run by a group of independent traders and strategy developers .