Quickly Analyzing Stock Charts

Table of Contents

Link to Repository

- Click here to go to our repository
- For our Demo that we did in class, click here

List of Project Participants:

- John Mikos - Lead + Developer
  - Contributions:
    - Organized and led project
    - Created initial and secondary dataset
    - Understood labels and defined next steps
    - Further research on day trading patterns for the second set of labels
    - Created a few of the base algorithms and all pricing pattern algorithms
    - In charge of presentation
- Erik Pautsch - Developer
  - Contributions:
    - Initial research into Pillow documentation and implementation
    - Implemented ArrayFromPic class to convert images into arrays
    - Implemented testing in testing.py
    - Implemented matching()
    - Debugged and refactored twoD_to_oneD()
    - Debugged and refactored imageConverter.py
- Jiaqi Fang - Developer
  - Contributions:
    - Converted 2D array into 1D array
    - Used quicksort and mergesort to sort the 1D array to make the data readable for comparing if it is increasing, decreasing or neutral. The sorted data in a 1D array. Outcome was to detect which algorithm performed better
    - Drove the conversion of the screenshot image to the black and white image
- Juan Arroyo - Researcher/Developer
  - Contributions:
    - Manually captured stock images for processing
    - Created a class to convert stock images from screenshots into the format needed for our application
    - Conducted research into image conversion using the pillow library
    - Researched stock market general information and image automation

Abstract of Project

There are many different players in the stock market. While these players may be individuals or institutions, regardless, they still need to operate based on decisions from financial data. Retrieving financial data in real time through APIs can be expensive. However, graphs representing a given financial asset's performance are often freely available to the public in near real-time.

We created a tool to capture quick, financial analysis from screen shots of these graphs. This tool takes images of graphs, converts the image into a two-dimensional array, and performs analysis on the resulting array. In this report, we go over a project narrative, some of the design considerations and specifications, the iterative design process, and constraints and challenges we ran into along the way.

In order to accomplish this task of information extraction from the images we imagined the general algorithm following these steps:

1. Overview
    a. Choose specific time frames
    b. Choose specific stocks to watch - most day traders who follow similar rules/ideas
2. Getting Screenshots
    a. Implement web-scraper on a site to take screenshot of certain dimension
3. Processing Screenshots
    a. Use similar process of PILLOW processing
    b. Remove all numbers + lines that are not the actual price movement
4. Interpreting
    a. Screenshots will have similar dimensions so algorithm results should be consistent
    b. Run on algorithms we found most efficient + accurate
5. Decision
    a. Connect to a real-time commission free API to put in values
6. Manually close out or have algorithm see if position open + direction reverse, sell

Project Narrative

Financial analysis can be an intricate and involved task. There are countless tools, methods, and visualizations made for helping analysts gain a better understanding of a particular financial environment. One such tool that is often seen is the graph, also known as charts. Whether a day trader is trying to find that perfect dip to buy, or a future retiree is mulling over whether to switch from stocks to bonds, charts provide insight into the history and performance of a financial asset. However, as useful as charts are, they provide a lot of information in a relatively densely packed space, and extracting meaningful analysis from them can be a time consuming task.

The idea for this project was to have a tool to examine the various ways we can begin to automate financial chart analysis. Designing and implementing a fully functioning tool that can provide meaningful analysis on screenshots of complex financial charts takes a good deal of time and effort. We wanted to initially focus our project's attention on simpler proof of concept graphs that allowed us to explore the various implementations of analysis algorithms and the reliability of the results they would give users. This initial interface of various functions that provide analysis data would be able to serve as a skeleton for a more robust graph-analysis tool in the future. With this idea, we might be able to take these algorithms that can do ideally the same tasks as machine learning algorithms in an even shorter timeframe.

Design Considerations

When first coming up with the design of this project, there were various aspects we had to account for. We needed initial test images, and the images had to be simple enough for us to be able to gain an understanding of where a bug might be when building the code. If we began with standard screenshots of real financial charting software, there would be far too many details and color switches in the image for us to be able to pull out meaningful data in the timeframe we are working in. We also had to determine what sorts of algorithms we might want to run on this type of data. The information we were trying to extract had to be relevant enough to financial analysis, but. At the same time, if our goals were overly complex, they would be too difficult to implement and debug, reducing the amount of algorithms we could implement and reducing our ability to analyze their effectiveness.

Design and Specification

The high-level idea of the project is to be able to identify and analyze stock charts in real time. Real time pricing information is hard to get and expensive; it is also usually delayed by at least 15 minutes. More up to date information provides better predictions for stocks. Graphs are usually easily available in real time. Some day traders will recognize and label graphs to understand where the price is most likely to go. Computers are able to recognize and react to information faster than humans, and they would have an edge against a normal human.

We generated simple graphs that were prelabeled. For simplicity with early testing, we kept the labels simple: generally increasing (labeled as positive, ‘+’), generally decreasing (labeled as negative, ‘-’), or neutral (labeled as ‘n’). We then took real screenshots of graph data, and used those new screenshots with the algorithms we had previously developed.

Designing the algorithms to work on the pictures of charts relied on our ability to capture value, or pricing, information at each corresponding x-value in a given graph. To do this, we decided we would represent the image as a two-dimensional array consisting of only binary values. Each 1 indicates part of the charted line and 0 indicates the background. At this point, while the corresponding array does represent the image, it is still relatively rudimentary and does not offer us much information in its current state. For most of our calculations, we will convert this two-dimensional array into a one-dimensional array, with each index representing an x-axis value and the value at each index representing the y-axis value. We also understood that the main goal of this project was for us to have a better understanding of which algorithms worked well and which did not. When designing each test image, we determined ahead of time whether that image should be labeled as positive (+), negative (-), or neutral (n). Knowing whether or not a given algorithm produced the correct labeling, combined. For converting the abstract data from image to be useful to analyze. We imagine that each pixel of white color has (x, y) value. Based on that, we do the following steps. First, we convert the sample images (we use the actual stock images in the final) to the useful 2D array data structure. We consider each white pixel in the image to number 1. So that all numbers of 1 stands for the line in the mapped 2D array (if you link all the 1, it should look exactly like the line of the source image). Then, we consider each index of the number 1 as a unique number then convert the 2D array into a 1D array (contains the index number of numbers. Second, by using the 1D array, we are able to use multiple algorithms to analyze the data, and we figure out which algorithm is good for analyzing the data in our project by comparing the running speed, accuracy, and extendability of each algorithm. Finally, we use the best algorithms that we tested before to analyze various real stock images instead of the simple samples.

Testing and Iterative Design

The initial design of our project revolved around a testing mindset from the beginning. We decided to start developing a solution with an almost test-driven approach. Because we were the ones that designed the test graphs, we were able to quickly determine whether or not a given algorithm produced the correct output. Testability also played a role in selecting the initial size of the image files. We needed to have an understanding of how each algorithm would work on a relatively low amount of pixels before we decided to expand the task to images containing thousands of pixels. This led us to select 28 x 28 as the dimensions, as 28 is a low enough number to manage while debugging, but also high enough to allow for interesting graph shapes.

To do the testing we manually had to assign labels to each graph that was drawn. We started at a simple approach to see if our initial graphs simply increased, descreased, or stayed the same throughout the 28x28 image. After reviewing our results and finding that they worked appropriately, we assigned new labels to the same 28x28 images that were based on pricing patterns. With this we were able to pull quick movements in the market. We however created algorithms that were a bit too fine tuned, and therefore we had to test different values to have for our cut-off.

The next step was to run our application using actual “stock” images, such images were manually captured, this proved to be a bit more challenging since our original dataset of images all had uniform properties, for example, size and colors. We had to convert the images into a format that our application could process in order to extract the data needed for our algorithms. This testing proved valuable as it allowed for us to notice where we need to change our cut-off numbers so that we can be successful in the future.

Restrictions, Limitations, and Constraints

With a project that is focused heavily on data that is very sporadic and pricing patterns are very specific, getting the correct output proved to be a bit more difficult. Beyond this, the process to success needs a lot of different parts to work together. Image manipulation was one of the challenges we first had to overcome, different sources include different data embedded into the stock charts, finding and converting the correct charts was essential to our application.Another limitation we encountered was the image capturing automation, currently our application works with manually captured stock images. Due to time constraints image automation was determined out of scope for our project. Nonetheless, some initial research was carried out about utilizing “python + selenium” to automate getting stock charts from the web.

Due to these limitations and difficulties, we failed to have completed automation for capturing images, which was our stretch goal for this project. It was out of scope due to complexity and time constraints.

There were not many missing features besides fine tuning our patterns to work even better. Beyond that, the project is missing the functionality to capture the stock image automatically from the stock market. Currently, the stock images are manually captured. Capturing images could be implemented in the future using python and selenium webdriver to automatically grab the images with a standard size.

Conclusion

This project allowed us to examine many different algorithms and how they can best be used to analyze financial charts. In addition, it allowed us to obtain a deeper understanding of algorithms and how they can be utilized to solve challenges in different aspects of engineering.

This project also provided us with a working model to actually implement this in a real world situation. We just need to add a bit more work, but we feel it will be successful.

Bibliography

- *The most detailed selenium WebDriver tutorial with python*. LambdaTest. (2022, April 4). Retrieved May 3, 2022, from https://www.lambdatest.com/blog/selenium-webdriver-with-python/
- Hayes, A. (2022, April 1). *Introduction to technical analysis price patterns*. Investopedia. Retrieved May 3, 2022, from https://www.investopedia.com/articles/technical/112601.asp
- StocksToTrade, T. B. F., Sheppe, W., Weiss, F. J., Riley, & Sebastiani, M. (2022, February 7). *Essential Stock chart patterns for traders in 2022*. StocksToTrade. Retrieved May 3, 2022, from https://stockstotrade.com/chart-patterns/
- *Top 10 chart patterns every trader should know*. Elearnmarkets. (2022, March 12). Retrieved May 3, 2022, from https://www.elearnmarkets.com/blog/chart-patterns-to-know-trading-stock/
- *Home*. Day Trading. (n.d.). Retrieved May 3, 2022, from https://www.daytrading.com/patterns
- *Advanced photo editor*. Photopea. (n.d.). Retrieved May 3, 2022, from https://www.photopea.com/