ASTrA - Another Stock Trading App

Near real-time price prediction and portfolio optimization using a Deep Learning ML model and Sharpe ratio optimization

Northwestern School of Professional Studies

MSDS 498 Data Engineering Capstone | June 2022

Enis Becirbegovic
EnisBecirbegovic2022@u.northwestern.edu

Oliver Zarate
OliverZarate2021@u.northwestern.edu

Mark Stockwell
MarkStockwell2021@u.northwestern.edu

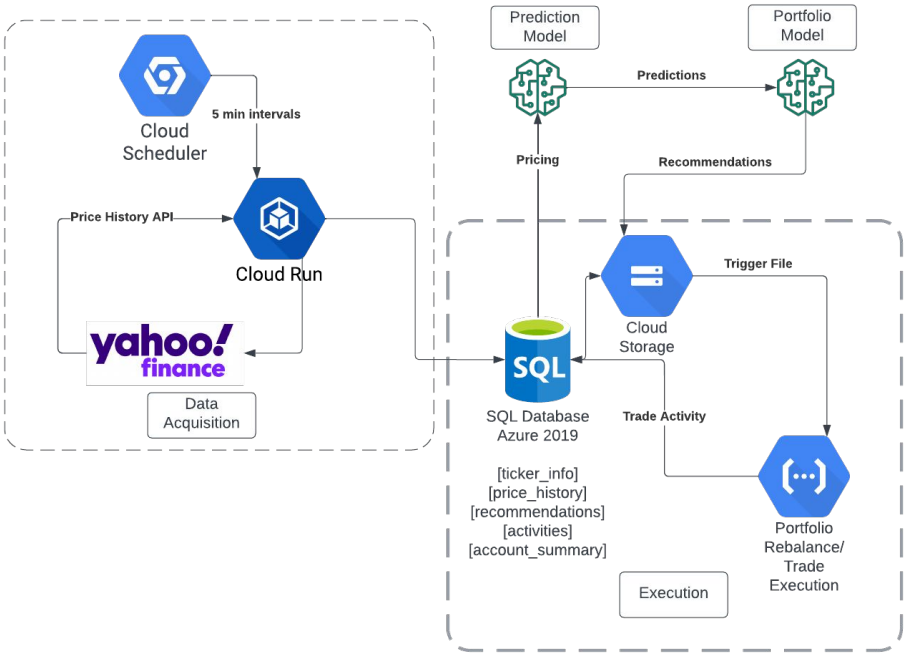
Siying Zhang
SiyingZhang2022@u.northwestern.edu

Overview

https://github.com/enisbe/stock-trading-app

ASTrA is a near real-time (~5 min latency) stock trading application that uses an NBeats (Neural basis expansion analysis for interpretable time series forecasting) deep learning machine learning algorithm to predict stock prices based on previous n periods. Stock predictions are fed into a portfolio optimization algorithm that uses Sharpe Ratios to minimize risk and maximize return. The process is as follows:

1. Google Cloud Scheduler is used to call a Cloud Run http endpoint every 5 minutes. The Cloud Run application takes a list of S&P 500 ticker symbols and retrieves the last 30 days of price history at 5 min intervals. This data is upserted to a SQL database (Azure).
2. Once the latest prices are available, the prediction model is called to calculate the expected price of each security over the next time period.
3. The predictions are passed to a second model which calculates the top 20 stocks to acquire and percent allocation for next time period.
4. A portfolio allocation file is uploaded to Google Cloud Storage, which triggers the trading execution Cloud Function. This function buys/sells securities as needed to achieve the desired allocation. Activity including details transaction history is written to the database for analysis. Upon completion of rebalance process an account summary with portfolio balance is also written.

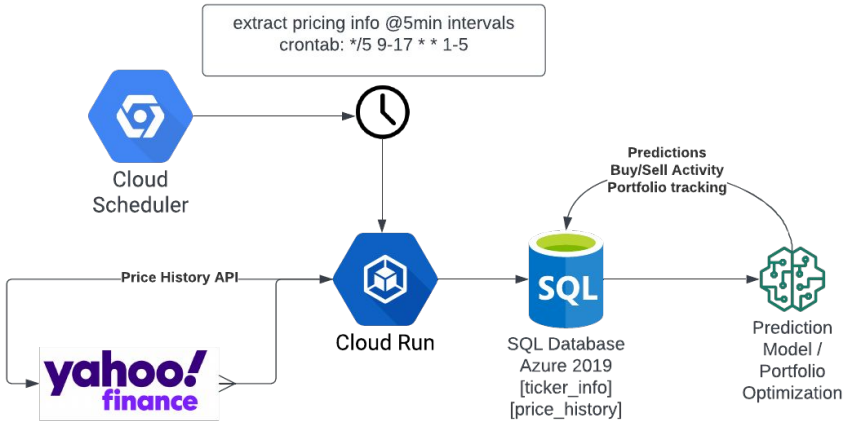


Data Acquisition

https://github.com/mgstockwell/yahoo_sp500_etl

Data Acquisition ETL Workflow

1. A Cloud Run application written in Python/Flask uses the yfinance python package to extract price history for all 500 S&P stocks.
2. Price history is written to an Azure database using Microsoft ODBC drivers and pyodbc package. The process loads a temp table with a large amount of data, then loads the main table with any rows
3. The Cloud Run application has a /dump_table function that simplifies data distribution to the prediction model, essentially an http data source.
4. A Cloud Scheduler job is called with a list of tickers to collect data. 10 jobs are configured to run simultaneously with 50 tickers each, completing in about 1 minute.



Prediction Model

Objective

- Build a deep learning model that predicts S&P prices – h periods ahead.
- Deploy model into production that makes predictions based on streaming stock data. Predicts next market hour in 5 minute increments.
- Pass prediction to target allocation logic

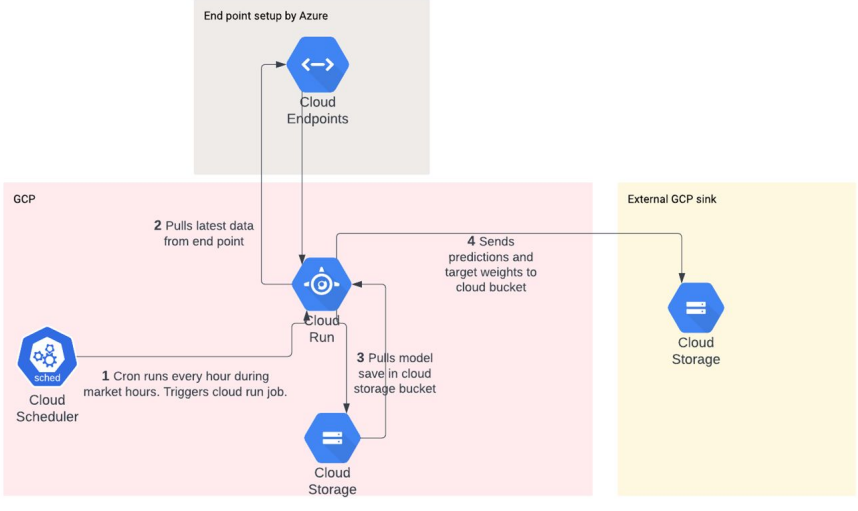
Implementation

- N-BEATS - Deep Learning architecture designed for time series forecasting.
- Single model trained on all S&P companies. Uses historical 5 minute price data
- Single cloud run services hosts flask api
- Orchestrated by cloud scheduler

ML Pipeline

https://github.com/enisbe/stock-trading-app/tree/main/cloud_run_predict

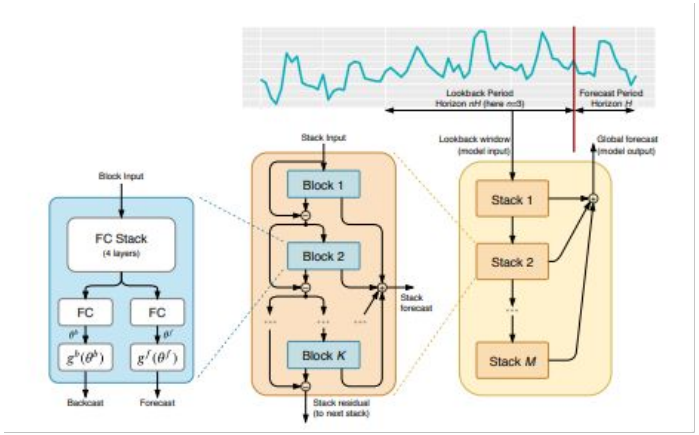
- Development:
  - Code base: Code spaces
  - Model: Google Colab
- Continuous Integration:
  - Cloud Build, triggered on github repo push
  - Image built on dockerfile by cloud build
- Model Location:
  - Cloud Storage Bucket (more economical than Vertex AI endpoint)
- Orchestration:
  - Cloud Scheduler triggers endpoint throughout the day during market day / hours

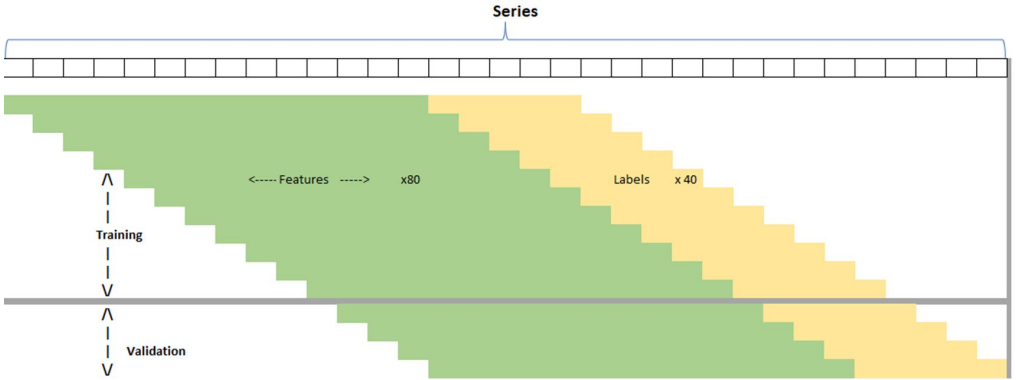


Model Overview

https://arxiv.org/abs/1905.10437

- N-BEATS- training input is 60, 5 minute price vector. Label - 12 minute 5- minute price vector
- Trained on 70k points, included sliding window across 500 stocks
- Training uses a windowing scheme
- Prediction uses a single sequence (past 60 observations or 5 hours) to predict 1 hour forward

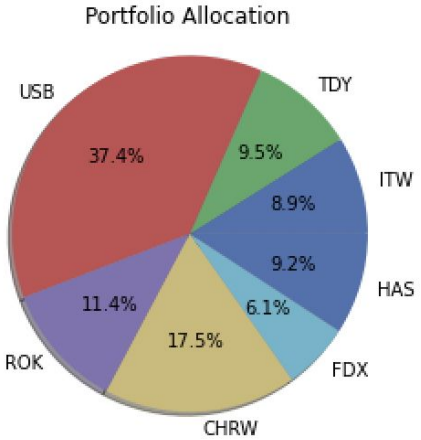


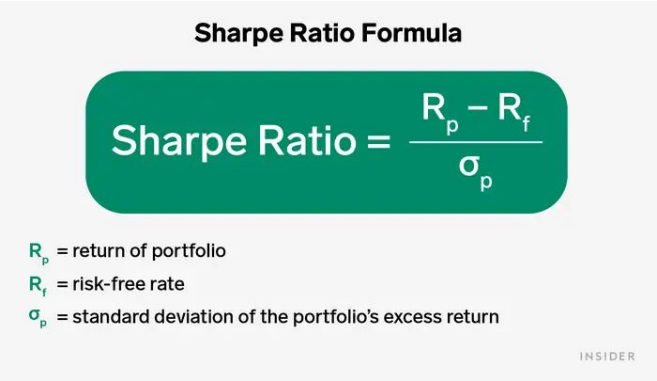


Portfolio Strategy Model

Portfolio Strategy Model Workflow

1. Receive the predicted prices from the forecast model and select the top 20 stocks with the highest expected return.
2. Conduct portfolio optimization among the selected stocks and allocate the weight to each stock within the portfolio.
3. The objective of the optimization model is to maximize the sharpe ratio of the portfolio and the sum of the weights of stocks are set to 1 as the constraint.
4. After the weights allocation is complete, the symbol of selected stocks along with the assigned weight is passed to the trading execution model for execution.

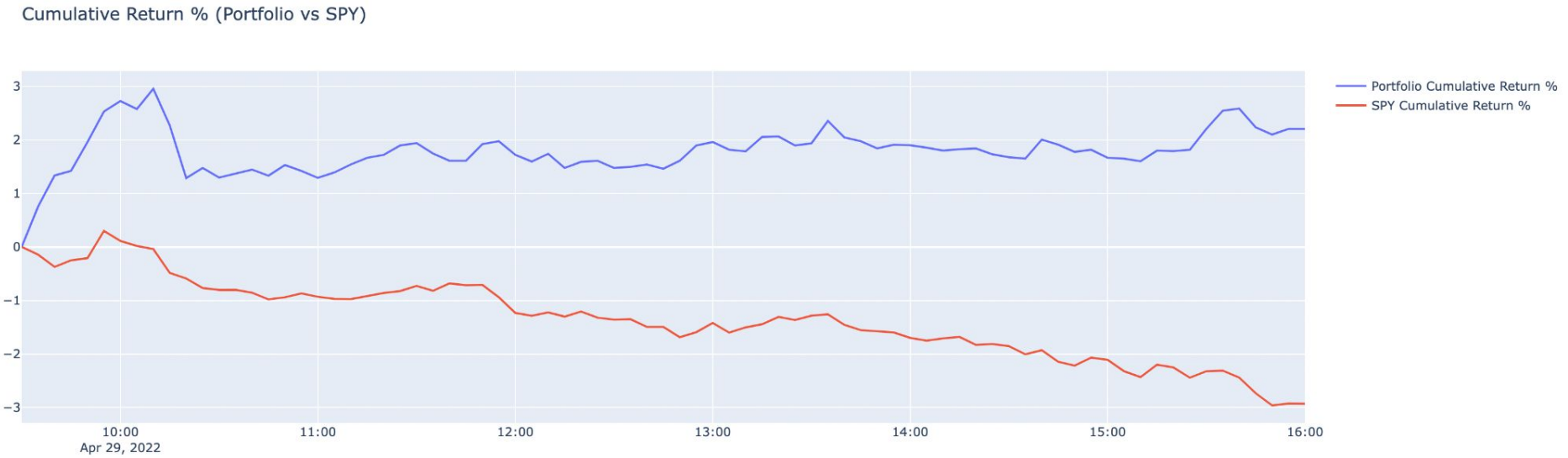




https://www.businessinsider.com/personal-finance/sharpe-ratio

Backtesting Model

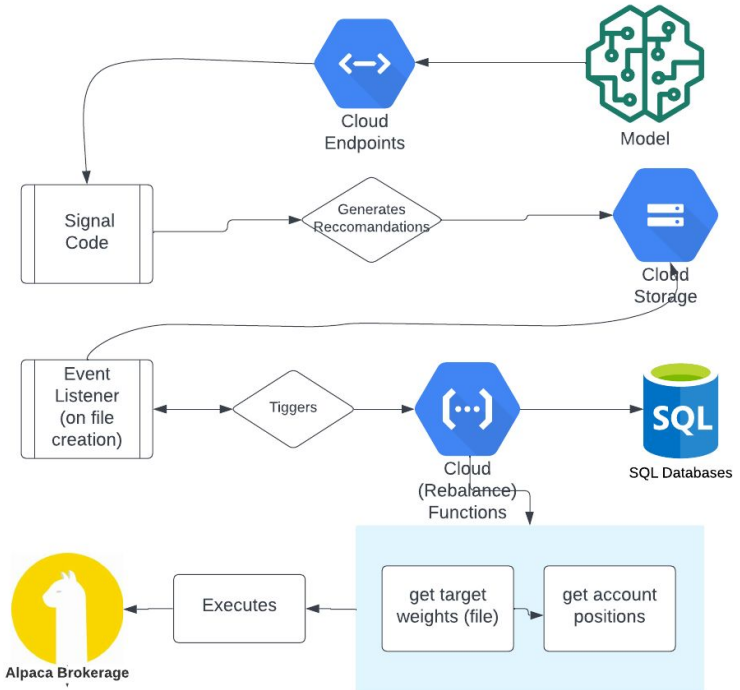
After the portfolio optimization is achieved, the backtesting model will backtest the performance of optimized portfolio using historical data to assess the profitability of the strategy. Furthermore, the backtesting model also provides a comparison between the performance of our optimized portfolio and the performance of benchmark index -- S&P 500 to show if our strategy outperforms the benchmark.



Rebalance and Execute

The rebalance process - event driven

1. Signal generation
2. Events generated to rebalance are triggered by a new model predictions dropped into a Google Cloud Storage **execution** bucket
3. Rebalance and execute procedure (google cloud function):
    a. Create rebalancing actions
    b. Execute the trades
    c. Record to database

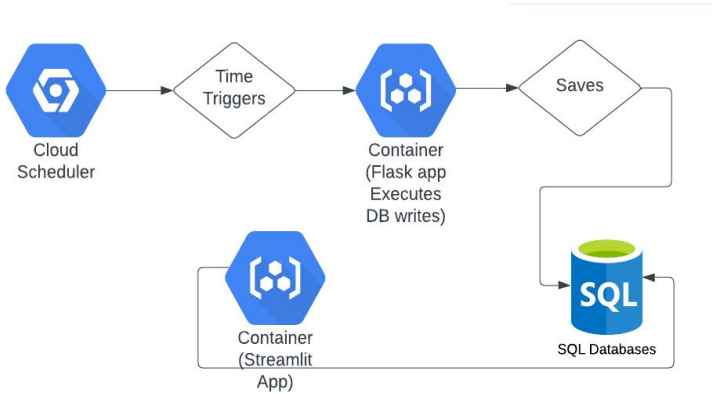


Monitoring and Front-End (https://frontend-4453yfddza-uc.a.run.app/)

Monitor and Front-end are container that support the information collection and review of trading activities

1. Flask application container server executes GET and POST requests to record the activities, recommandations, and account history to SQL server database
2. Streamlit container (front-end) connects to SQL database and summarizes the account activities

Design and Architecture



Front-end