

Using AutoGluon with AWS to predict stock prices

Capstone Project

Edgar Villegas

February 22nd, 2022

I. Definition

Project Overview

Stock market prediction is the attempt of determining the future value of a company stock. The successful prediction of a stock price can yield significant profit¹.

Individuals, hedge funds and all kinds of investors have been using different types of financial models to make profitable investments on company stocks². With the scientific and technological advances in the last years, plus the plenty availability of data, Machine Learning has also become an option to achieve these objectives.

This project aims to build an AI powered web application to predict stock prices, by leveraging AWS technologies and consuming data from [Yahoo Finance](#).

The machine learning algorithm will be delegated to [AutoGluon](#), an open Auto ML technology developed by Amazon.

Live project running on <https://d28uo0kei40s00j.cloudfront.net/>. Github repo [here](#).

Problem Statement

A crucial part of making profitable investments is to be able to estimate future close prices of a stock, based on its current trading information (Open/Close/Highest prices, volume, etc.) as well as the historical data, which means, that data in the past.

For example, let's assume we have daily trading information for Apple (AAPL) for February 2021.

Sample Input, (February Lookback)

Date	Open	High	Low	Close	Volume	Adjusted Close
2021-02-16	135	136	133	133	80576300	132
2021-02-17	131	132	129	131	97918500	130
...

¹ (Wikipedia)

² (Udacity)

2021-02-25	125	126	121	121	148199500	120
2021-02-26	123	125	121	121	164560400	121

And we want to predict the Adjusted Close price for March (in USD). Adjusted Close is a “normalized” version of the Close price, that takes into account the stock splits and dividends. For practical purposes, we can consider it as the actual close price, and it is our target for prediction.

The following would be the solution outcome.

Sample Output (March Forecast)

Date	Predicted Adj Close
2021-03-01	118
2021-03-03	124
...	...
2021-03-30	135
2021-03-31	140

Needless to say, given the prediction consists of real numbers (prices are in USD), this is a regression problem.

Metrics

Given the regression nature of this problem, the model evaluation is going to use the RMSE (root mean squared error) as evaluation metric.

The RMSE is defined as³:

$$x_{\text{RMS}} = \sqrt{\frac{1}{n} (x_1^2 + x_2^2 + \dots + x_n^2)}.$$

Where x is each datapoint’s error (difference between predicted and actual value)

RMSE is Autogluon’s default metric for regression problems.

³ (Wikipedia)

II. Analysis

Datasets and Inputs

The dataset is actual trading data from Yahoo Finance. It will be acquired **on demand** by performing a GET request to a parameterized url (dependent on the ticker and a date range).

`https://query1.finance.yahoo.com/v7/finance/download/{ticker}?period1={period1}&period2={period2}&interval=1d&events=history&includeAdjustedClose=true`

For example:

`https://query1.finance.yahoo.com/v7/finance/download/AAPL?period1=1539907200&period2=1618790400&interval=1d&events=history&includeAdjustedClose=true`

which will return a CSV like the following:

```
Date,Open,High,Low,Close,Adj Close,Volume
2021-02-16,135.490,136.009,132.789,133.190,132.403,80576300
2021-02-17,131.250,132.220,129.470,130.839,130.066,97918500
2021-02-18,129.199,130.000,127.410,129.710,128.943,96856700
2021-02-19,130.240,130.710,128.800,129.869,129.102,87668800
2021-02-22,128.009,129.720,125.599,126.000,125.255,103916400
```

This data comes clean, so it does not need transformations or processing.

The dataset includes the following fields

Field Name	Type	Description
Date	Datetime	Date of the stock data
Open	Float	Open price at the beginning of the day
High	Float	Highest price during the day
Low	Float	Lowest price during the day
Close	Float	Close price
Adj Close	Float	Adjusted close price, takes into account dividends and splits. This will be the target value.
Volume	Int	Transaction volume for the day

Data Exploration

Note: you can find the code for this in the data_exploration.ipynb notebook.

We'll use Apple's (AAPL) last 3 years of data from Yahoo Finance for exploration. The sample csv url is:

<https://query1.finance.yahoo.com/v7/finance/download/AAPL?period1=1550793600&period2=1645488000&interval=1d&events=history&includeAdjustedClose=true>

A glance to the data:

	Date	Open	High	Low	Close	Adj_Close	Volume
0	2019-02-22	42.895000	43.250000	42.845001	43.242500	42.105236	75652800
1	2019-02-25	43.540001	43.967499	43.487499	43.557499	42.411949	87493600
2	2019-02-26	43.427502	43.825001	43.292500	43.582500	42.436302	68280800
3	2019-02-27	43.302502	43.750000	43.182499	43.717499	42.567745	111341600
4	2019-02-28	43.580002	43.727501	43.230000	43.287498	42.149048	112861600
...
751	2022-02-14	167.369995	169.580002	166.559998	168.880005	168.880005	86185500
752	2022-02-15	170.970001	172.949997	170.250000	172.789993	172.789993	64286300
753	2022-02-16	171.850006	173.339996	170.050003	172.550003	172.550003	61177400
754	2022-02-17	171.029999	171.910004	168.470001	168.880005	168.880005	69589300
755	2022-02-18	169.820007	170.539993	166.190002	167.300003	167.300003	82614200

Let's check some useful dataset statistics

	Open	High	Low	Close	Adj_Close	Volume
count	628.000000	628.000000	628.000000	628.000000	628.000000	6.280000e+02
mean	78.181644	79.155991	77.198416	78.226035	77.156960	1.349292e+08
std	31.799664	32.194446	31.233977	31.725079	31.829701	6.111411e+07
min	35.994999	36.430000	35.500000	35.547501	34.464806	4.544800e+07
25%	50.683750	51.097500	50.239375	50.686250	49.475485	9.249275e+07
50%	68.684998	69.791252	67.537498	68.772499	67.759963	1.185214e+08
75%	113.964998	115.460001	112.227501	113.579998	112.508511	1.615198e+08
max	143.600006	145.089996	141.369995	143.160004	142.101807	4.265100e+08

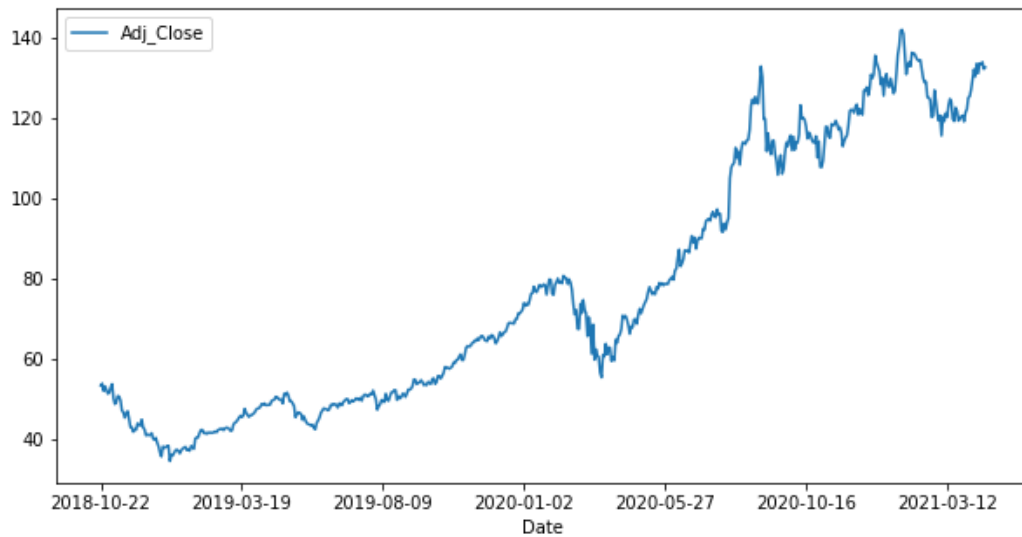
We can see that the only feature that varies considerably is the Volume (by checking its range and std). The rest is not as spread. We'll visualize it in the next section.

Exploratory Visualization

Let's plot the evolution of our variable of interest, the Adjusted Close price through time:

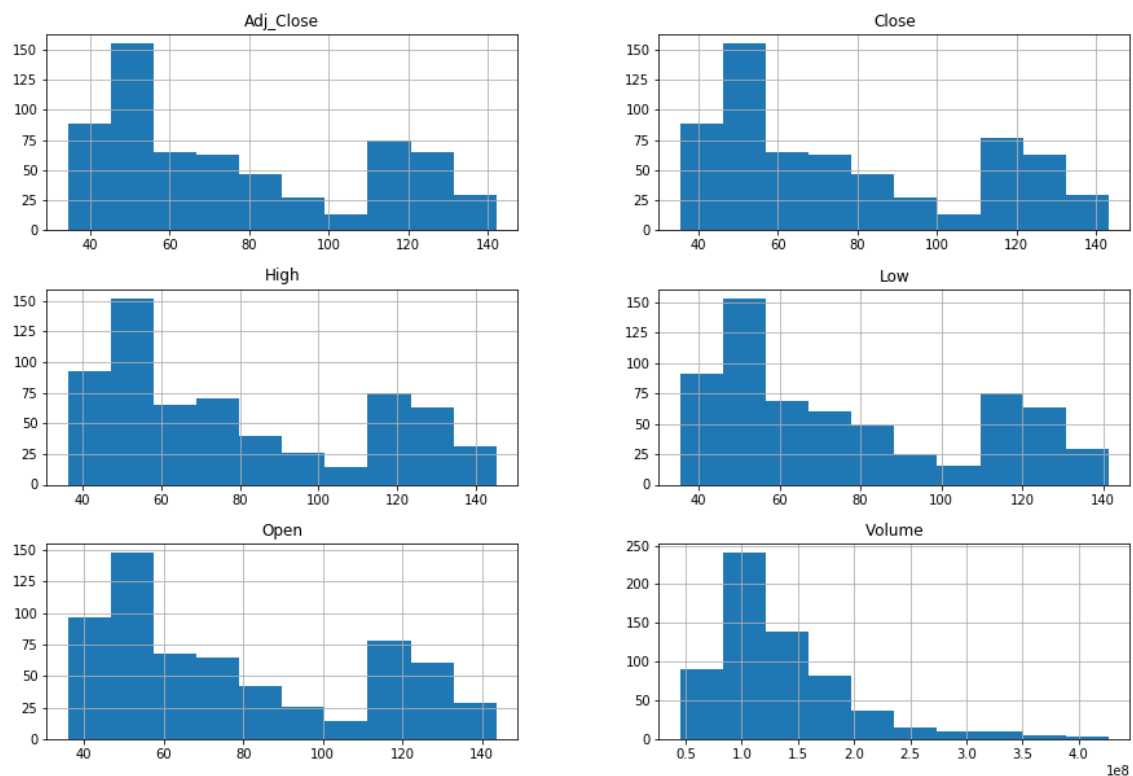
```
[72]: df.plot(x='Date', y='Adj_Close', figsize=(10,5))
```

```
[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7f38619207d0>
```



For this stock, we can observe an ascending behavior in general. Prices vary between 40 and 140 aprox.

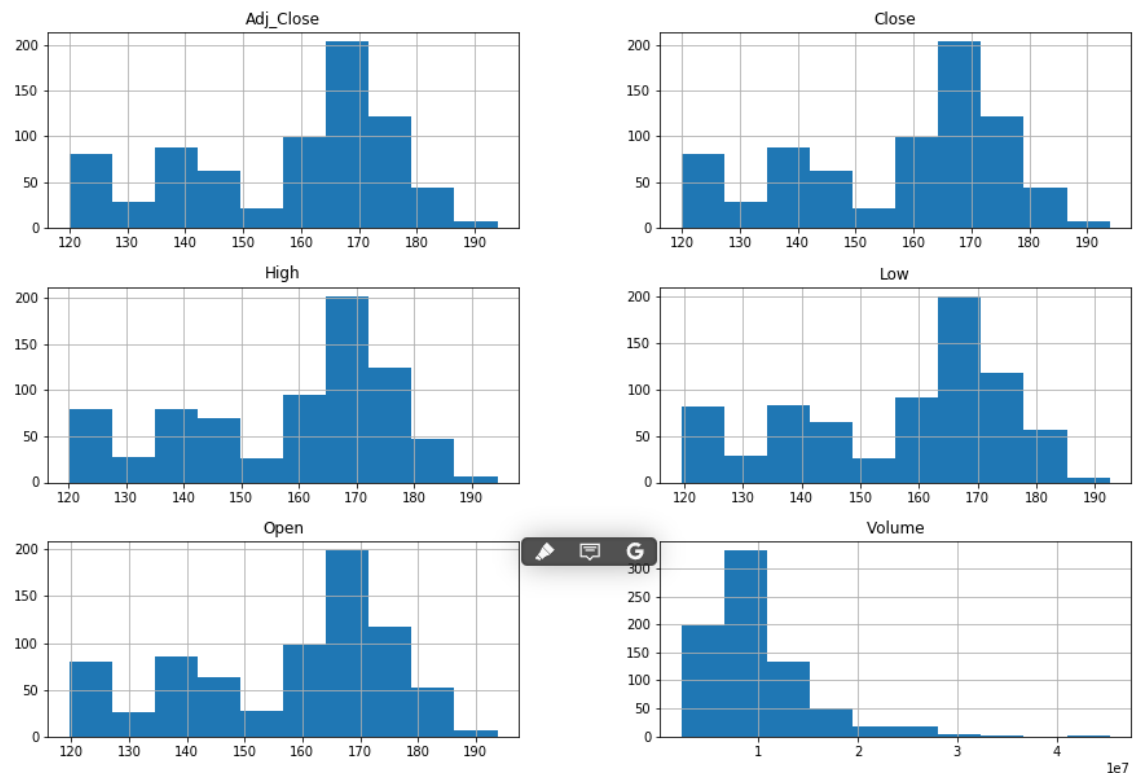
One of the best ways of having a glance to the data distribution is through a histogram.



We can observe that all numeric fields have a similar distribution, bimodal in this example. This could be very dependent on the stock

Except for the volume, which seems to follow a log normal distribution.

Now let's check the distribution of another stock to check the differences. Let's plot a histogram for the same period for Gold (GLD)



We can observe that the prices distribution is very different. However, the volume is also log normal.

We don't notice the presence of outliers or atypical data.

Algorithms and techniques

The project uses AutoGluon (chosen for for simplicity reasons) within AWS ecosystem to make these predictions for a specific ticker (a ticker is a stock's identifier). The input will be the stock data for a date range in the past, and the output will be the predicted close price for a later date range.

We are going to use the best model suggested by Autogluon.

The final user is intended to provide:

- Ticker (for example, AAPL)
- Lookback date range (for example, Jan-Dec 2021)
- Forecast date range (for example, Jan 2022)

After acquiring the lookback data, an AutoGluon model will be trained. Finally, the model will be queried providing the forecast date range to get the predicted stock prices (adjusted close).

About AutoGluon

AutoGluon-Tabular is an open-source AutoML framework that requires only a single line of Python to train highly accurate machine learning models on an unprocessed tabular dataset such as a CSV file. Unlike existing AutoML frameworks that primarily focus on model/hyperparameter selection, AutoGluon-Tabular succeeds by ensembling multiple models and stacking them in multiple layers. Experiments reveal that our multi-layer combination of many models offers better use of allocated training time than seeking out the best. A second contribution is an extensive evaluation of public and commercial AutoML platforms including TPOT, H2O, AutoWEKA, auto-sklearn, AutoGluon, and Google AutoML Tables. Tests on a suite of 50 classification and regression tasks from Kaggle and the OpenML AutoML Benchmark reveal that AutoGluon is faster, more robust, and much more accurate.⁴

We chose AutoGluon for simplicity and to experiment with this technology. AutoGluon-Tabular fits this problem very well given the tabular nature of the dataset.

Benchmark

It would be ideal to use a domain well known model as a benchmark. However, running another model would add complexity to the project that we cannot afford. Therefore, for simplicity we're going to use the simple moving average (average of the previous n-points) as benchmark model.

In statistics, a moving average (rolling average or running average) is a calculation to analyze data points by creating a series of averages of different subsets of the full data set. It is also called a moving mean (MM) or rolling mean and is a type of finite impulse response filter.

In financial applications a simple moving average (SMA) is the unweighted mean of the previous k data-points.⁵

In conclusion, we're going to compare the moving average for the Adjusted Close, for the last week with our Autogluon's model prediction.

⁴ (Cornell University)

⁵ (Wikipedia)

III. Methodology

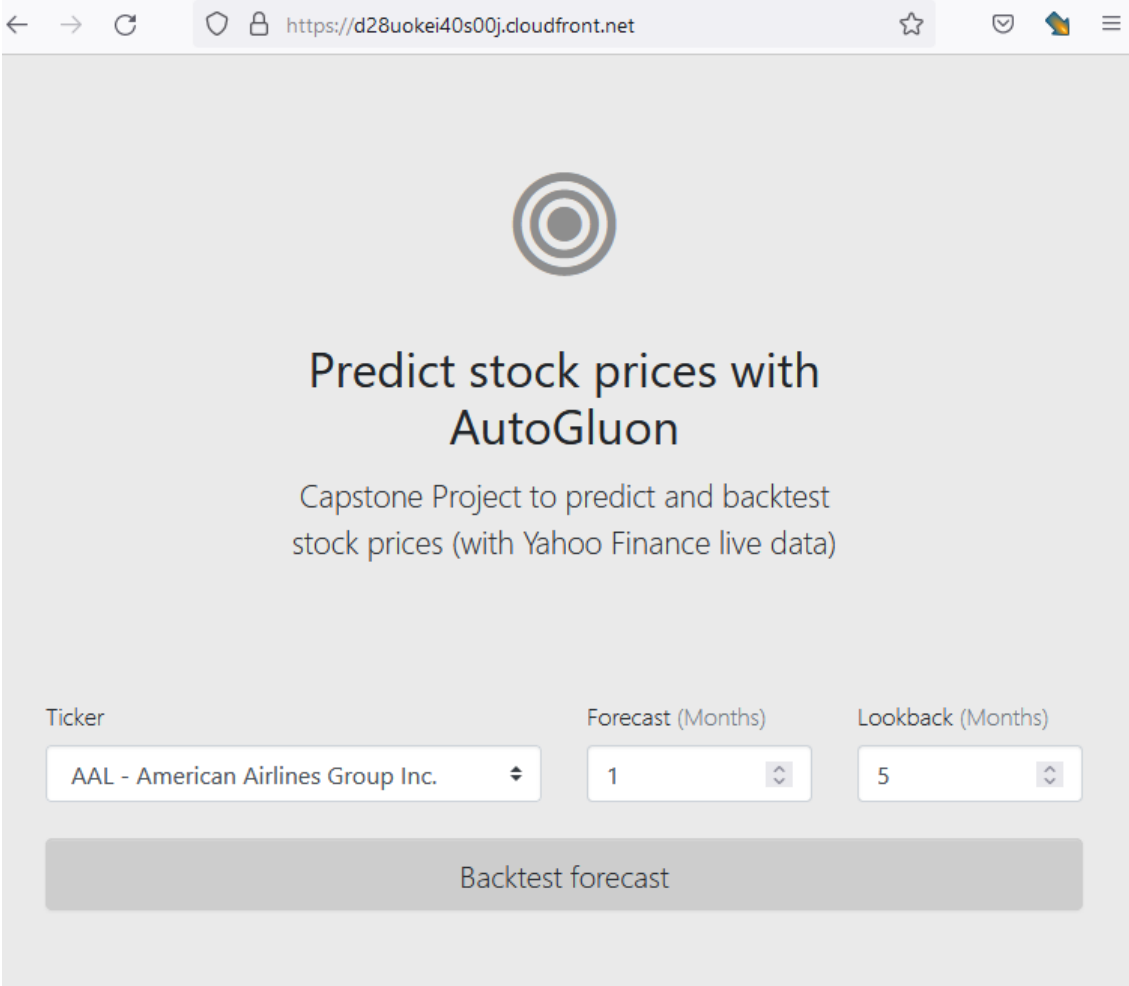
Data Preprocessing

The data comes clean from source (Yahoo Finance) so it does not need cleaning. Also, given it provides an Adjusted Close field, we don't need to do further calculations.

Additionally, given the data is already in tabular format, it can go as-is to autogluon.

Implementation

Let's start by the Web UI. It consists in a form where the user specifies the ticker, lookback and forecast periods in months.

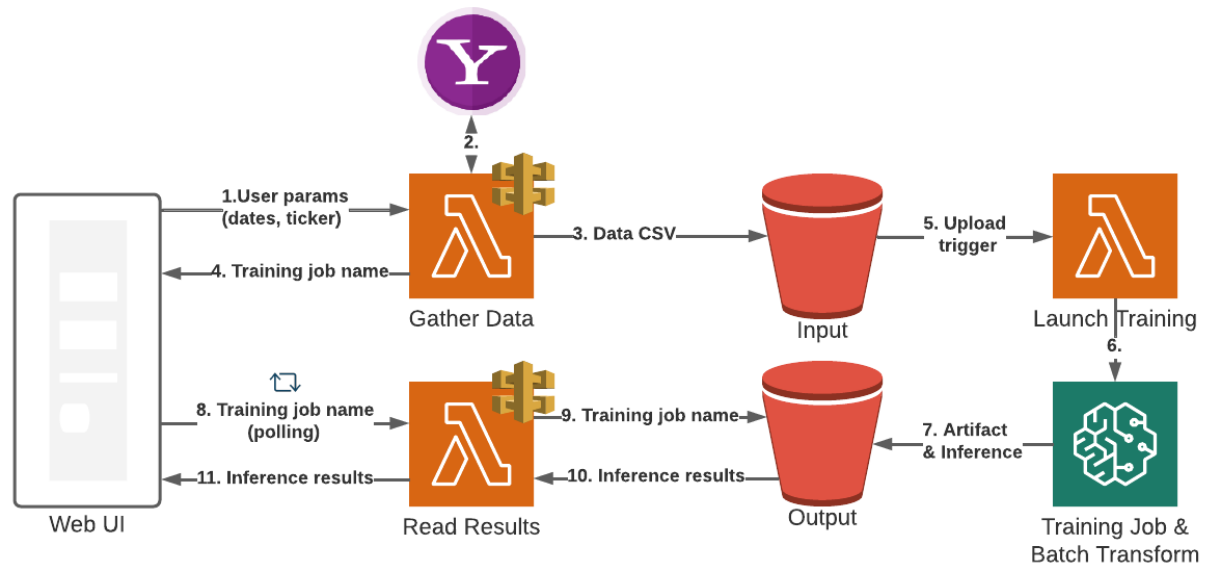
A screenshot of a web browser displaying a web application. The browser's address bar shows the URL <https://d28uokeyi40s00j.cloudfront.net>. The web page has a light gray background. At the top center is a logo consisting of three concentric circles. Below the logo, the text "Predict stock prices with AutoGluon" is displayed in a large, dark font. Underneath this, a subtitle reads "Capstone Project to predict and backtest stock prices (with Yahoo Finance live data)". Below the subtitle, there are three input fields: "Ticker" with a dropdown menu showing "AAL - American Airlines Group Inc.", "Forecast (Months)" with a numeric input field showing "1", and "Lookback (Months)" with a numeric input field showing "5". At the bottom of the form is a large, gray button labeled "Backtest forecast".

This webapp is working live on <https://d28uokeyi40s00j.cloudfront.net/>

It has been developed in ReactJS, hosted on S3 with Cloudfront. Code in the **react-web-ui/** folder.

When clicking the "Backtest Forecast" button, the whole ML flow will be executed on AWS.

Complete flow



1. When the user clicks the “Backtest forecast” button, it calls the Gather Data lambda (through API Gateway), sending it the user params filled in the form (ticker, lookback period, forecast period). Code in **lambda_gather_data.py**.
2. This lambda uses the user params form a url to request Yahoo a CSV with the input data. It creates train.csv for the lookback period and test.csv for the lookback one.
3. The data CSVs is uploaded to S3. A training job name is formed by concatenating the ticker, params and a timestamp. For example, AAPL-f5-b30-2022-02-22-06-21-15-991155.
4. The training job name is returned to the web UI. Then it starts polling for inference results with that job name (step 8)
5. The S3 data storage triggers the training lambda. (This has been based on this [AWS blog post\[4\]](#)). Code in **lambda_launch_train.py**.
6. The training lambda launches a training job,
The entry point has the training algorithm, AutoGluon with default parameters. Code in **train-infer.py**.
The main parameter sent to this AutoGluon entrypoint is the s3 input data location (for train.csv and test.csv) and the s3 output location (check step 7).
The training output is the model.tar.gz artifact.

After successful training, this script immediately runs a batch transform to get the forecast prediction (inference for test.csv).

7. The model artifact and the inference results are be saved to S3
8. The UI, which knows the training job name, polls the Read Results lambda to know if the inference results are ready. (Actually, polling started right after step 4).
9. The Read Results lambda checks in the output bucket if the results are ready
10. Once ready, the lambda reads the results and converts them to JSON
11. The final JSON results are sent to the UI to be displayed

Finally, the React UI will show the backtest vs the ground truth in a chart, plus the benchmark for visual comparison. Check the Results section for details.

Refinement




There has been no model refinement in this project due to timing constraints. The model is using Autogluon's defaults. However, it's planned to do the following in a future:

- Add custom features like Bollinger bands and different moving averages (as features, not as benchmark)
- Play with different Autogluon's api parameters, and hyperparameters.

IV. Results

Model Evaluation and Validation

After AutoGluon's has chosen the best model and performed inference with it, the following output files are produced:

<input type="checkbox"/>	 results_leaderboard.csv
<input type="checkbox"/>	 results_model_performance.txt
<input type="checkbox"/>	 results_test_predictions.csv

results_model_performance.txt has the metrics evaluated for the best model. For example:

```
{
  "root_mean_squared_error": 1.2620023498621271,
  "mean_absolute_error": 0.9268035727539001,
  "explained_variance_score": 0.9692686004751757,
  "r2_score": 0.9689596714926992,
  "pearson_correlation": 0.9861297733250114,
  "mean_squared_error": 1.5926499310575306,
  "median_absolute_error": 0.7515256494140772
}
```

results_leaderboard.csv is a summary of the different models that AutoGluon tried internally, and the score each one took.

results_test_predictions.csv is the most important one, as it contains the actual predictions

```
True,Predicted
132.323532,132.39465
132.71121200000002,132.40796
131.160461,131.17374
133.526382,133.22577
133.924026,133.84166
133.595978,133.72504
132.79075600000002,132.83812
132.69134499999998,132.97389
130.683289,130.69337
131.756897,131.65422
127.09461200000001,127.41887
```

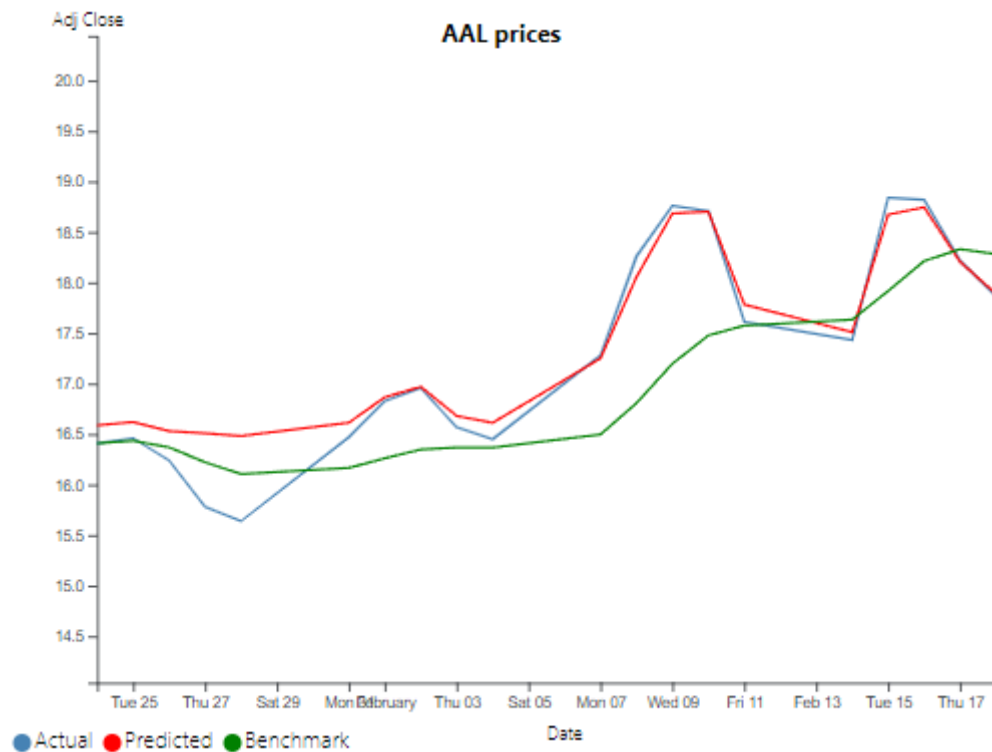
(It contains the true va not contain the other fields, such as date, so it will have to be joined with the original test csv)

These results are shown in the Web UI, including the benchmark



Case 1: Short term forecast

Let's examine the model performance for AAPL, 1 month forecast and 5 months lookback,



As you can see by comparing the true value (blue) with the predicted one (red), **the predictions are very accurate.**

The inference obtained a RMSE of 1.04:

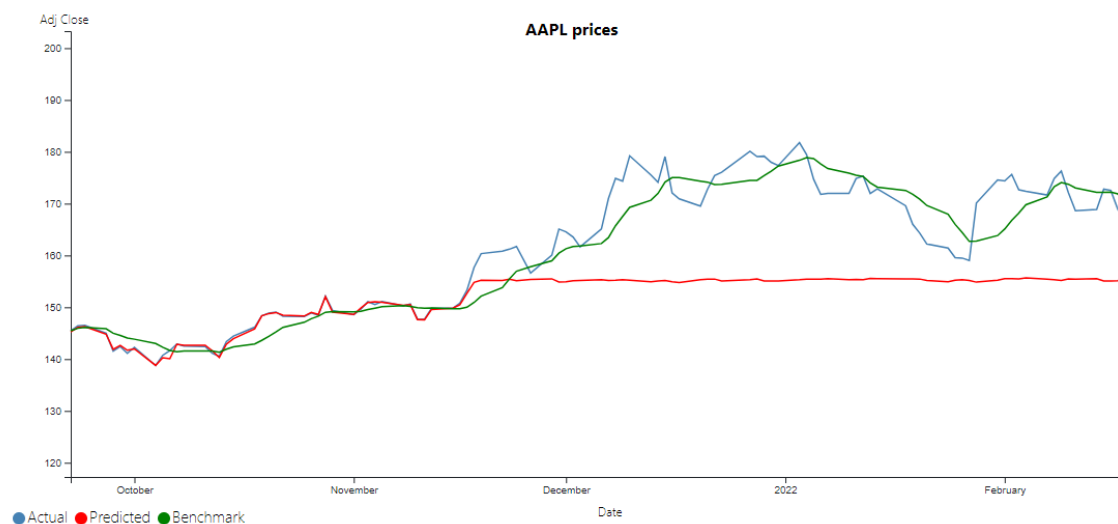
"root_mean_squared_error": 1.0481035333211786,

which is a low enough value.

Case 2: Long term forecast

However, when the forecast period is long, the model starts to perform poorly.

Consider the following example run for a lookback of 30 months and a forecast of 5 months:



As you can see, there are good predictions until mid December, then it practically predicts a constant value, which is far from ideal.

The RMSE in this case is 12.78, much higher than the previous case, which confirms the poor performance.

This can be replicated with other tickers, the outcome is the same.

Justification

We can observe that our benchmark model (simple moving average, green line) is not good enough to be compared to the trained model. It can be useful as a reference, but not for benchmarking



By evaluating the long term Forecast case in the previous section, we can conclude that the proposed solution definitely **does not** solve the problem.

For the short term forecast, although the evaluated cases and metrics look promising, it needs more trials with more tickers to confirm it works.

Also, it's not clear how long can this term be. This needs further research.

V. Conclusion

Reflection

In this project, we developed an end-to-end ML solution to predict stock prices. The data is acquired on demand from Yahoo Finance, which needed no preprocessing. Then, the chosen algorithm for this regression problem was AutoGluon, due to its simplicity (and the author's motivation to try it out) and promising results. For our case, the model is also trained on demand for the time periods that the user selects. Finally, the inference results are shown in the UI as a chart which allows to do visual comparisons (between the actual, predicted and benchmark values) very easily.

Although AutoGluon is tremendously practical, it's not a silver bullet. We've seen that our model outperforms for long term forecasts. This is not AutoGluon's culprit, it's just that it does not solve this problem with default values. Further research is needed, to fine tune Autogluon or to use another algorithm more specific for time series and forecasting.

All this project was done by leveraging AWS technologies such as Api Gateway, Cloudfront, S3, Lambda, Sagemaker. Also a React web application was developed to have a final user deliverable.

Improvement

There are several improvements for this project that became out of scope due to time constraints, but could be done in a future. Some of them are:

- Use data of other tickers simultaneously to make prediction more accurate
- Add more features like Bollinger bands, moving average, etc.
- Use Sagemaker's Processing and Batch Transform Jobs
- Add serverless inference as an alternative to Batch Transform
- Add monitoring and profiling
- Use AWS SNS and websockets to notify the client when batch transform is done
- Compare with a proven domain specific benchmark model
- Get the forecast term threshold for good enough accuracy. It should be measurable
- Use an algorithm specifically tailored for time series, like TS-Gluon or DeepAR.

VI. References

Cornell University. Autogluon. [Online] <https://arxiv.org/abs/2003.06505>.

Udacity. MLND Stocks Project Description. [Online] <https://bit.ly/3sUY53e>.

Wikipedia. Moving Average. [Online] https://en.wikipedia.org/wiki/Moving_average.

—. Root-mean-square deviation. [Online] https://en.wikipedia.org/wiki/Root-mean-square_deviation.

—. Stock Market Prediction. [Online] https://en.wikipedia.org/wiki/Stock_market_prediction.