Aphelion Algorithmic Superiority



Presented by Johnathan Overton, Khalid Abdulkadir, and Dylan Olsen

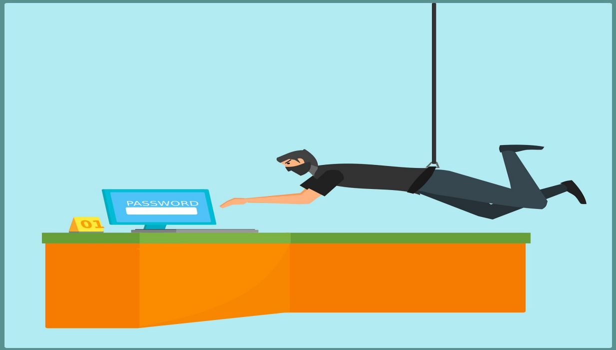
Motivation and Summary

Day traders and hedge funds alike are increasingly ditching their old methods and turning to machine learning to do the trading for them, and it's obvious why. Having an AI enabled bot do the work for you removes emotion from the equation, while providing the best returns possible using a high-level algorithm.

We collaborated to create an algorithmic trading bot using Amazon SageMaker’s notebook. The bot uses Alpaca’s paper trading API to send and receive data to make a prediction on whether to buy or sell a stock.

Data Techniques

- Data source: Alpaca’s paper trading api
  - Familiarity and ease of use/integration
  - We wanted to test our model on the open market by utilizing our alpaca paper trading accounts
- Reasoning for data selection
  - We selected TSLA stock data to train our model since Tesla stock has a history of high volatility and volume, and we wanted to incorporate that into our training.
- Collection, exploration, and cleaning process
  - Structure and Filter the DataFrame
  - Model, Fit, and Predict Data
  - Visualize and Evaluate Model
  - Initialize and Execute Trading Commands



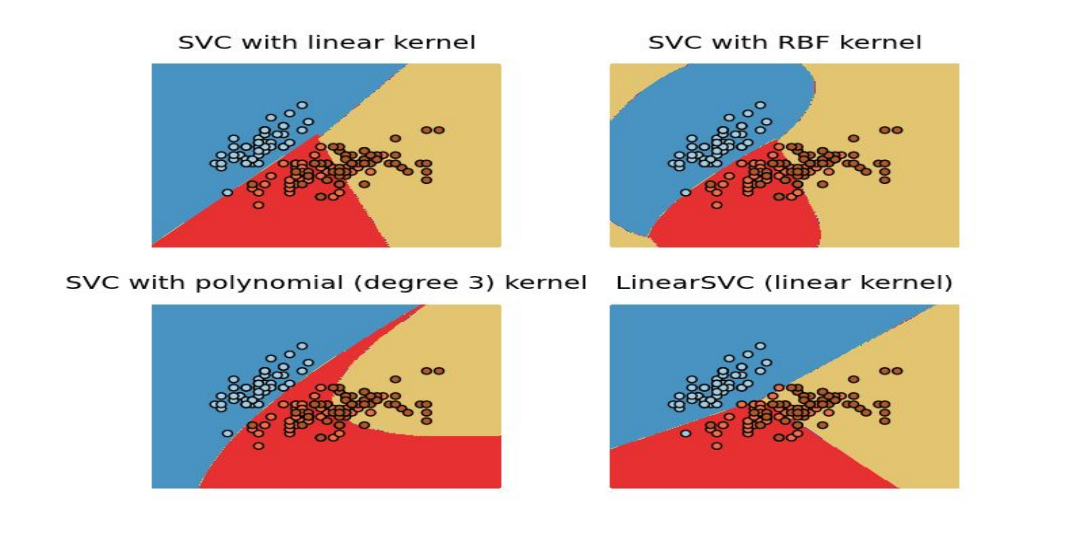
Approach

- Technologies used
  - Alpaca
  - AWS SageMaker
  - SVM/RandomForest
- Breakdown of tasks and roles
  - Johnathan: The bulk of programming and evaluation of the model.
  - Dylan: Filled the gaps and supplied the team with good vibes.
  - Khalid: alternative Random Forest model, co wrote this presentation
- Challenges
  - High recall
  - Overfitting Training Data
  - Env file proved difficult to load into Amazon SageMaker
    - Stack OverFlow provided a solution using your own API private keys
  - Git pushes/pulls

Model Summary

The first model used to predict the Tesla data was a Support Vector Classifier (SVC) model.

The support vector machine (SVM) is a data classification technique that has been recently shown to outperform other machine learning techniques when applied to stock market forecasting. In a possible particular state observation or outcome can be generated which is associated symbol of observation of probability distribution. The RBF kernel is popular because of its similarity to the K-Nearest Neighbor Algorithm. We chose the SVC model with linear kernel. In the future, we may implement a model represented by the RBF kernel.



Data Cleanup

The data used for our model is Tesla stock metrics spanning from 2016 - 2022 (8 years of data).

Using the python package, alpaca-trade-api, you must use api.get_bars(ticker, timeframe, start_date, end_date).df to structure the DataFrame properly.

[70]:
```
# Create Tesla DateFrame
tesla_data = api.get_bars(tesla, timeframe, start_date, end_date).df
tesla_data.head()
```

[70]:

timestamp	open	high	low	close	volume	trade_count	vwap
2016-01-04 05:00:00+00:00	230.77	231.38	219.00	223.41	6827146	69015	223.588147
2016-01-05 05:00:00+00:00	226.29	226.89	220.00	223.43	3186752	31300	223.139332
2016-01-06 05:00:00+00:00	220.00	220.05	215.98	219.04	3779128	33011	217.791187
2016-01-07 05:00:00+00:00	214.24	218.44	213.67	215.65	3554251	33417	216.042799
2016-01-08 05:00:00+00:00	218.56	220.44	210.77	211.00	3628058	32682	214.595420

Data Cleanup (cont.)

1. Preprocess Data
    a. Filter the data to only the 'close' column and rename it to 'TSLA'. Then,
    b. Index the new filtered DataFrame with Date as timestamp.date
    c. Create an Actual Returns (pct_change) column alongside TSLA's closing price.
    d. Developed customized SMA_Fast and SMA_Slow windows
    e. Created new columns for the DataFrame for the respective SMA's.

```
# Plot Tesla Closing Prices
tesla_data = tesla_data.filter(['close'])
tesla_data.rename(columns={'close':'TSLA'}, inplace = True)
tesla_data.index = tesla_data.index.map(lambda timestamp : timestamp.date)
tesla_data.plot()
```

```
# Create SMA windows
SMA_fast = 5
SMA_slow = 11
```

```
# Create new column called "Actual Returns" and add percent change to the column.
tesla_data['actual_returns'] = tesla_data['TSLA'].pct_change()
tesla_data
```

```
# Create two new columns for SMA's, and apply rolling mean
tesla_data['slow_SMA'] = tesla_data['TSLA'].rolling(window=SMA_slow).mean()
tesla_data['fast_SMA'] = tesla_data['TSLA'].rolling(window=SMA_fast).mean()
```

Model Training

2 . Model Setup

a. Set Features (X) as the ‘fast_SMA’, and ‘slow_SMA’ columns
b. Initialize signal column to identify targets (y)
c. Develop for loop in Actual Returns column
d. Locate targets from signals

```
# Initialize signal column
tesla_data['signal'] = 0.0
```

```
# Features for X variable are the SMA columns without nulls and shifted one row.
X = tesla_data[['fast_SMA', 'slow_SMA']].shift().dropna().copy()
```

```
# Make a copy of signal column for y values
y = tesla_data['signal'].copy()
```

```
# For loop to iterate over each data point in the actual returns column
for i in tesla_data['actual_returns']:
    if i > 0:
        z = True
    else:
        z = False
```

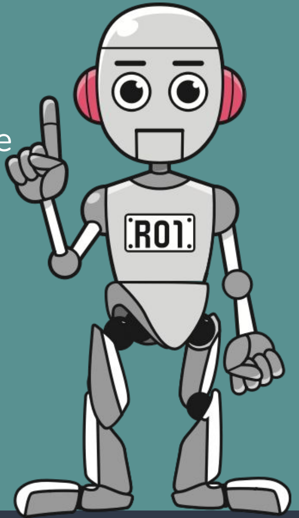
```
# locate actual returns greater than or equal to 0
tesla_data.loc[(tesla_data['actual_returns'] >= 0), 'signal'] = 1
```

```
# locate actual returns less than 0
tesla_data.loc[(tesla_data['actual_returns'] < 0), 'signal'] = -1
```

Model Training (cont.)

3. Fit and Predict the Model

   a. Identify the desired training and testing data
   b. Initiate StandardScaler package
   c. Fit and transform the training and testing data.
   d. Import Support Vector Classifier from the Support Vector Machines package
   e. Fit and Predict the scaled training data




```
# Identify data to use for training
X_train = X.loc[training_begin:training_end]
y_train = y.loc[training_begin:training_end]
```

```
# Identify the data used for testing

X_test = X.loc[training_end:]
y_test = y.loc[training_end:]
```

```
# Initiate StandardScaler Model, Fit, Transform
scaler = StandardScaler()
X_scaler = scaler.fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
# Initiate SVC model from svm package, fit and predict using trained data.
svm_model = svm.SVC()
svm_model = svm_model.fit(X_train_scaled, y_train)
training_signal_predictions = svm_model.predict(X_train_scaled)
training_signal_predictions
```

Model Evaluation

The initial classification report had a recall of 100%, but a precision of 0%. Likely, a result of overfitting training data.

```
[23]: # Print the classification report
      training_report = classification_report(y_train, training_signal_predictions)
      print(training_report)

              precision    recall  f1-score   support

        -1.0       1.00      0.00      0.00       599
         1.0       0.53      1.00      0.69       661

    accuracy                           0.53      1260
   macro avg       0.76      0.50      0.35      1260
weighted avg       0.75      0.53      0.36      1260
```

By adjusting the **training** data from 72 months to 48 months. We balanced the amount of training and testing sample data. In return, we were successfully able to achieve an:

Overall Precision: 75%

Overall Recall: 53%

Discussion

Discuss your findings. Was the model sufficient for the predictive task? If not, why not?

This model predicts stock movements on a paper trading account, which makes it a low-risk application. Because of this, our model's precision of 75% and recall of 53% are acceptable.

What inferences or general conclusions can you draw from your model performance?

There is certainly room for improvement, future upgrades to the model could include adding volatility and volume metrics to our data points, and checking to see if they improve our precision in any significant way.

Postmortem

Discuss any difficulties that arose, and how you dealt with them.

In cases where stock was at sub-penny prices, trade could not be conducted. We created a rounding function inside of the trade execution to ensure alleviation of the error.

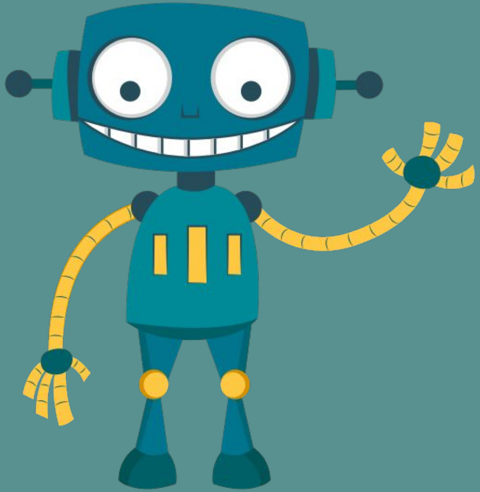
Discuss any additional questions or problems that came up but you didn't have time to answer: What would you research next if you had two more weeks?

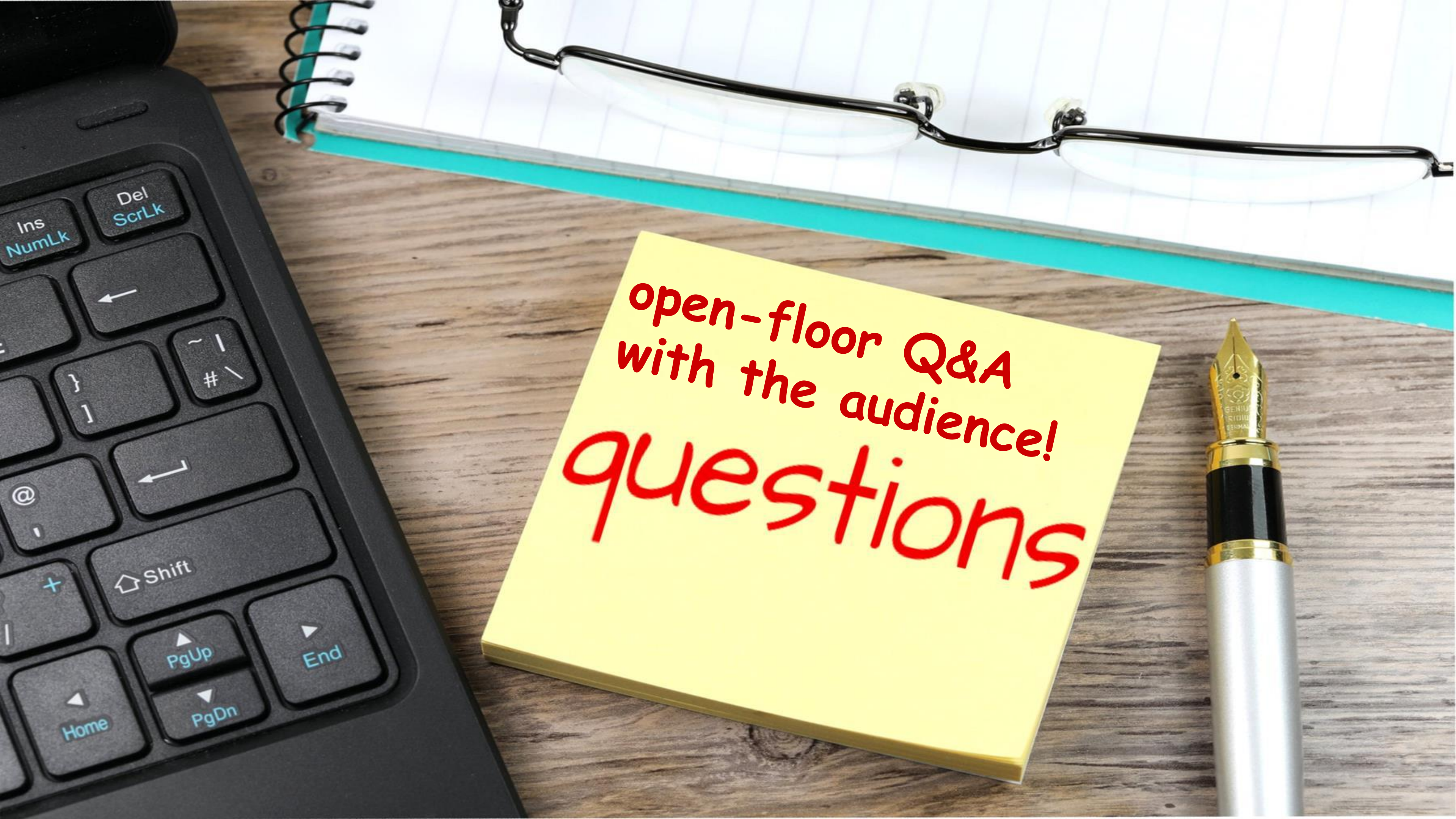
In the future we could add columns to the training/testing data with a snapshot of the last week of stock data included in each row, this could include metrics such as volatility over the past week, the week’s highs/lows, volume, etc. This would be a more comprehensive dataset for the model as opposed to the only the closing and SMA.

Links

- https://github.com/Johove83/Aphelion_Algorithmic_Superiority

Jonathan Dylan Khalid




open-floor Q&A
with the audience!
questions