Predicting Stock Prices Using Machine Learning

9 mins read Author Katherine (Yi) Li Updated May 13th, 2022

The stock market is known for being volatile, dynamic, and nonlinear. Accurate stock price prediction is extremely challenging because of multiple (macro and micro) factors, such as politics, global economic conditions, unexpected events, a company's financial performance, and so on.

But, all of this also means that there's a lot of data to find patterns in. So, financial analysts, researchers, and data scientists keep exploring analytics techniques to detect stock market trends. This gave rise to the concept of algorithmic trading, which uses automated, pre-programmed trading strategies to execute orders.

READ ALSO

7 Applications of Reinforcement Learning in Finance and Trading

In this article, we'll be using both traditional quantitative finance methodology and machine learning algorithms to predict stock movements. We'll go through the following topics:

- Stock analysis: fundamental vs. technical analysis
- Stock prices as time-series data and related concepts
- Predicting stock prices with Moving Average techniques
- Introduction to LSTMs
- Predicting stock prices with an LSTM model
- Final thoughts on new methodologies, such as ESN

Disclaimer: this project/article is not intended to provide financial, trading, and investment advice. No warranties are made regarding the accuracy of the models. Audiences should conduct their due diligence before making any investment decisions using the methods or code presented in this article.

Stock analysis: fundamental analysis vs. technical analysis

When it comes to stocks, fundamental and technical analyses are at opposite ends of the market analysis spectrum.

- Fundamental analysis (you can read more about it here):
  - Evaluates a company's stock by examining its intrinsic value, including but not limited to tangible assets, financial statements, management effectiveness, strategic initiatives, and consumer behaviors; essentially all the basics of a company.
  - Being a relevant indicator for long-term investment, the fundamental analysis relies on both historical and present data to measure revenues, assets, costs, liabilities, and so on.
  - Generally speaking, the results from fundamental analysis don't change with short-term news.
- Technical analysis (you can read more about it here):
  - Analyzes measurable data from stock market activities, such as stock prices, historical returns, and volume of historical trades; i.e. quantitative information that could identify trading signals and capture the movement patterns of the stock mark

- Due to its short-term nature, technical analysis results are easily influenced by news.
- Popular technical analysis methodologies include moving average (MA), support and resistance levels, as well as trend lines and channels.

For our exercise, we'll be looking at technical analysis solely and focusing on the Simple MA and Exponential MA techniques to predict stock prices. Additionally, we'll utilize LSTM (Long Short-Term Memory), a deep learning framework for time-series, to build a predictive model and compare its performance against our technical analysis.

As stated in the disclaimer, stock trading strategy is not in the scope of this article. I'll be using trading/investment terms only to help you better understand the analysis, but this is not financial advice. We'll be using terms like:

- trend indicators: statistics that represent the trend of stock prices,
- medium-term movements: the 50-day movement trend of stock prices.

Stock prices as time-series data

Despite the volatility, stock prices aren't just randomly generated numbers. So, they can be analyzed as a sequence of discrete-time data; in other words, time-series observations taken at successive points in time (usually on a daily basis). Time series forecasting (predicting future values based on historical values) applies well to stock forecasting.

Because of the sequential nature of time-series data, we need a way to aggregate this sequence of information. From all the potential techniques, the most intuitive one is MA with the ability to smooth out short-term fluctuations. We'll discuss more details in the next section.

RELATED ARTICLES

👉 Time Series Forecasting – Data, Analysis, and Practice
👉 Anomaly Detection in Time Series

Dataset analysis

For this demonstration exercise, we'll use the closing prices of Apple's stock (ticker symbol AAPL) from the past 21 years (1999-11-01 to 2021-07-09). Analysis data will be loaded from Alpha Vantage, which offers a free API for historical and real-time stock market data.

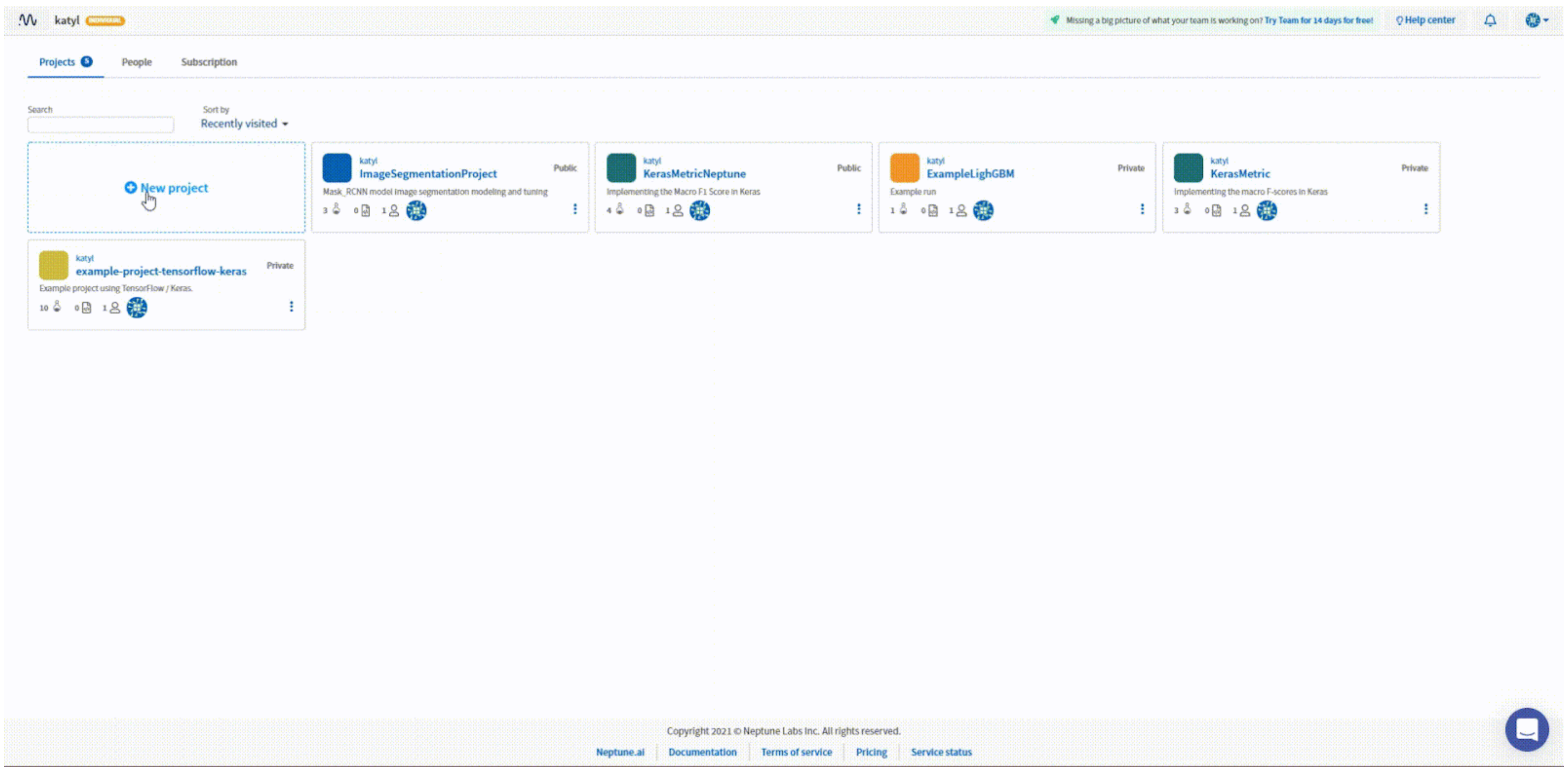
To get data from Alpha Vantage, you need a free API key; a walk-through tutorial can be found here. Don't want to create an API? No worries, the analysis data is available in my Github repo as well. If you feel like exploring other stocks, code to download the data is accessible in this Github repo as well. Once you have the API, all you need is the ticker symbol for the particular stock.

For model training, we'll use the oldest 80% of the data, and save the most recent 20% as the hold-out testing set.

```
#### Train-Test split for time-series ####
test_ratio = 0.2
training_ratio = 1 - test_ratio

train_size = int(training_ratio * len(stockprices))
test_size = int(test_ratio * len(stockprices))
print("train_size: " + str(train_size))
print("test_size: " + str(test_size))

train = stockprices[:train_size][['Date', 'Close']]
test = stockprices[train_size:][['Date', 'Close']]
```

With regard to model training and performance comparison, Neptune makes it convenient for users to track everything model-related, including hyper-parameter specification and evaluation plots. This complete guide provides step-by-step instructions on how to set up and configure your Neptune projects with Python.

Now, let's create a project for this particular exercise and name it "**StockPrediction**".


katyl
Missing a big picture of what your team is working on? Try Team for 14 days for free!
Help center
Projects 5
People
Subscription
Search
Sort by
Recently visited
New project
katyl
ImageSegmentationProject
Public
Mask_RCNN model image segmentation modeling and tuning
katyl
KerasMetricNeptune
Public
Implementing the Macro F1 Score in Keras
katyl
ExampleLighGBM
Private
Example run
katyl
KerasMetric
Private
Implementing the macro F-scores in Keras
katyl
example-project-tensorflow-keras
Private
Example project using TensorFlow / Keras.
Copyright 2021 © Neptune Labs Inc. All rights reserved.
Neptune.ai
Documentation
Terms of service
Pricing
Service status

Evaluation metrics and helper functions

Since stock prices prediction is essentially a regression problem, the RMSE (Root Mean Squared Error) and MAPE (Mean Absolute Percentage Error %) will be our current model evaluation metrics. Both are useful measures of forecast accuracy.

$$MAPE = \frac{1}{N} * \sum_{t=1}^{N} \left| \frac{At - Ft}{At} \right|$$

$$RMSE = \sqrt{\frac{1}{N} * \sum_{t=1}^{N} (At - Ft)^2}$$

, *where N = the number of time points, At = the actual / true stock price, Ft = the predicted / forecast value.*

RMSE gives the differences between predicted and true values, whereas MAPE (%) measures this difference relative to the true values. For example, a MAPE value of 12% indicates that the mean difference between the predicted stock price and the actual stock price is 12%.

Next, let's create several helper functions for the current exercise.

- Split the stock prices data into training sequence X and the next output value Y,

```
def extract_seqX_outcomeY(data, N, offset):
    """
    Split time-series into training sequence X and outcome value Y
    Args:
        data - dataset
        N - window size, e.g., 50 for 50 days of historical stock prices
        offset - position to start the split
    """
    X, y = [], []

    for i in range(offset, len(data)):
        X.append(data[i-N:i])
        y.append(data[i])

    return np.array(X), np.array(y)
```

- Calculate the RMSE and MAPE (%),

```
#### Calculate the metrics RMSE and MAPE ####
def calculate_rmse(y_true, y_pred):
    """
    Calculate the Root Mean Squared Error (RMSE)
    """
    rmse = np.sqrt(np.mean((y_true-y_pred)**2))
    return rmse

def calculate_mape(y_true, y_pred):
    """
    Calculate the Mean Absolute Percentage Error (MAPE) %
    """
    y_pred, y_true = np.array(y_pred), np.array(y_true)
    mape = np.mean(np.abs((y_true-y_pred) / y_true))*100
    return mape
```

- Calculate the evaluation metrics for technical analysis and log to Neptune (with arg. logNeptune = True),

```
    rmse = calculate_rmse(np.array(stockprices[train_size:]['Close']),
np.array(stockprices[train_size:][var]))
    ### MAPE
    mape = calculate_mape(np.array(stockprices[train_size:]['Close']),
np.array(stockprices[train_size:][var]))

    ## Log images to Neptune new version
    if logNeptune:
        npt_exp['RMSE'].log(rmse)
        npt_exp['MAPE (%)'].log(mape)

    return rmse, mape
```

- Plot the trend of the stock prices and log the plot to Neptune (with arg. with arg. logNeptune = True),

```
def plot_stock_trend(var, cur_title, stockprices=stockprices, logNeptune=True,
logmodelName='Simple MA'):
    ax = stockprices[['Close', var,'200day']].plot(figsize=(20, 10))
    plt.grid(False)
    plt.title(cur_title)
    plt.axis('tight')
    plt.ylabel('Stock Price ($)')

    ## Log images to Neptune new version
    if logNeptune:
        npt_exp[f'Plot of Stock Predictions with
{logmodelName}'].upload(neptune.types.File.as_image(ax.get_figure()))
```

Predicting stock price with Moving Average (MA) technique

MA is a popular method to smooth out random movements in the stock market. Similar to a sliding window, an MA is an average that moves along the time scale/periods; older data points get dropped as newer data points are added.

Commonly used periods are 20-day, 50-day, and 200-day MA for short-term, medium-term, and long-term investment respectively.

Two types of MA are most preferred by financial analysts: Simple MA and Exponential MA.

Simple MA

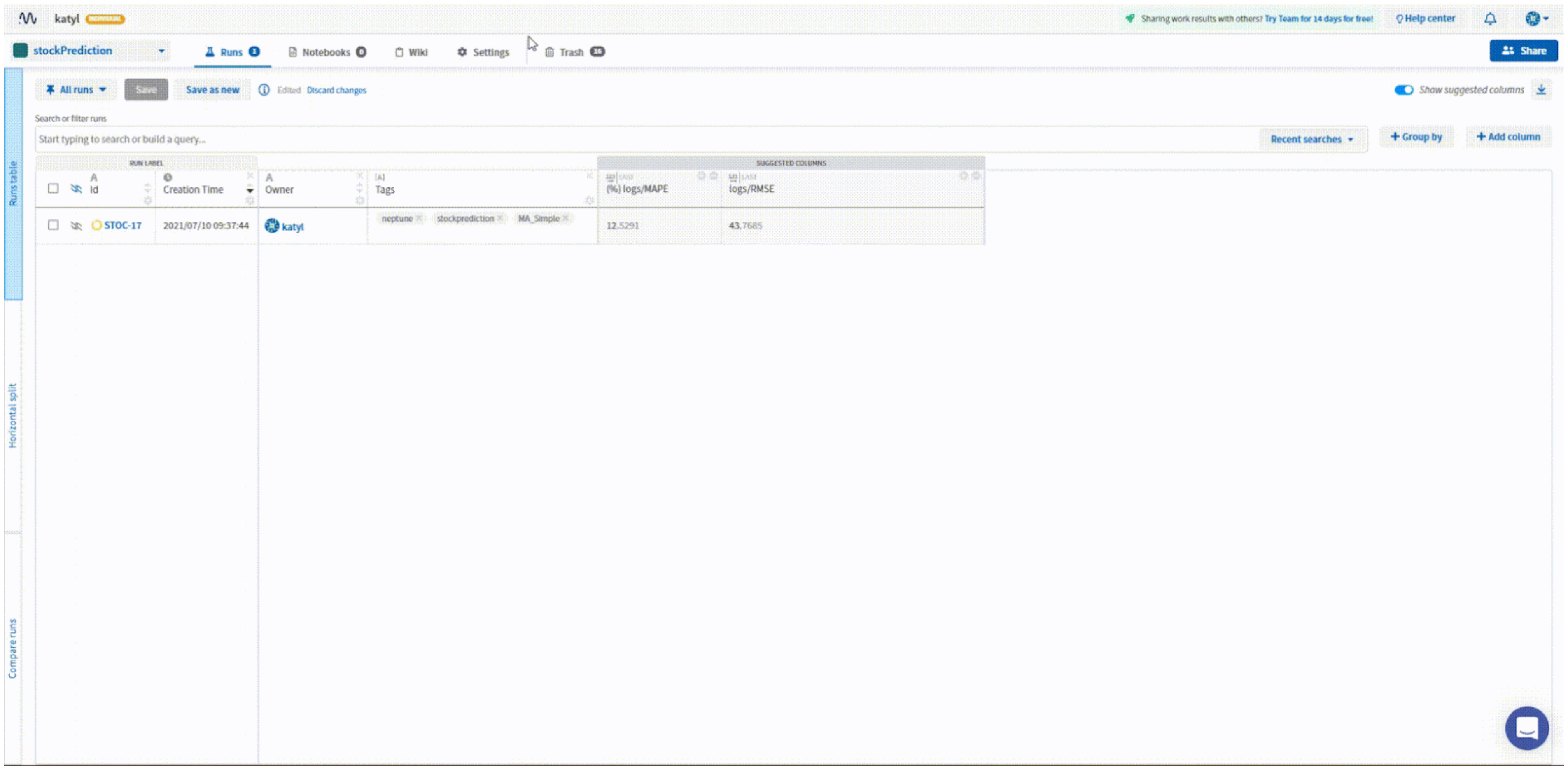
SMA, short for Simple Moving Average, calculates the average of a range of stock (closing) prices over a specific number of periods in that range. The formula for SMA is:

$SMA = \frac{P1 + P2 + ... + Pn}{N}$, *where Pn = the stock price at time point n, N = the number of time points.*

For this exercise of building an SMA model, we'll use the Python code below to compute the 50-day SMA. We'll also add a 200-day SMA for good measure.

```
# Create an experiment and log the model in Neptune new version
npt_exp = neptune.init(
        api_token=os.getenv('NEPTUNE_API_TOKEN'),
        project=myProject,
        name='SMA',
        description='stock-prediction-machine-learning',
        tags=['stockprediction', 'MA_Simple', 'neptune'])

window_var = str(window_size) + 'day'

stockprices[window_var] = stockprices['Close'].rolling(window_size).mean()
### Include a 200-day SMA for reference
stockprices['200day'] = stockprices['Close'].rolling(200).mean()

### Plot and performance metrics for SMA model
plot_stock_trend(var=window_var, cur_title='Simple Moving Averages',
logmodelName='Simple MA')
rmse_sma, mape_sma = calculate_perf_metrics(var=window_var, logmodelName='Simple
MA')

### Stop the run after logging for new version
npt_exp.stop()
```

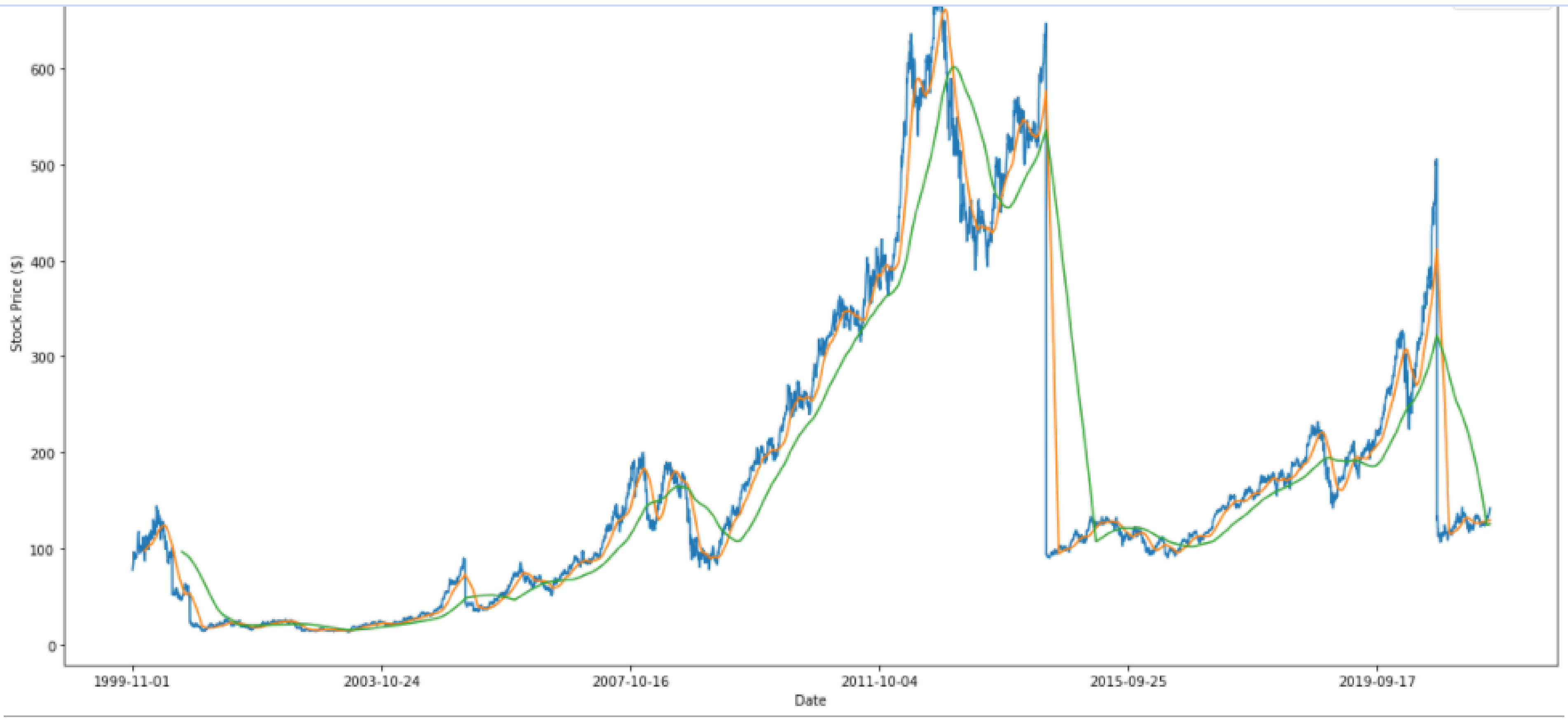
In our Neptune project, we'll see the performance metrics on the testing set; RMSE = 43.79, and MAPE = 12.53%.



Id	Creation Time	Owner	Tags	(%) logs/MAPE	logs/RMSE
STOC-17	2021/07/10 09:37:44	katyl	neptune, stockprediction, MA_Simple	12.5291	43.7685

[See in the app](#)

In addition, the trend chart below shows the 50-day, 200-day SMA predictions compared with the true stock closing values.



It’s not surprising to see that the 50-day SMA is a better trend indicator than the 200-day SMA in terms of (short-to-) medium movements. Both indicators, nonetheless, seem to give smaller predictions than the actual values.

Exponential MA

Different from SMA, which assigns equal weights to all historical data points, EMA, short for Exponential Moving Average, applies higher weights to recent prices, i.e., tail data points of the 50-day MA in our example. The magnitude of the weighting factor depends on the number of time periods. The formula to calculate EMA is:

$EMA = P_t * k + EMA_{t-1} * (1 - k)$,

where Pt = the price at time point t,

EMAt-1 = EMA at time point t-1,

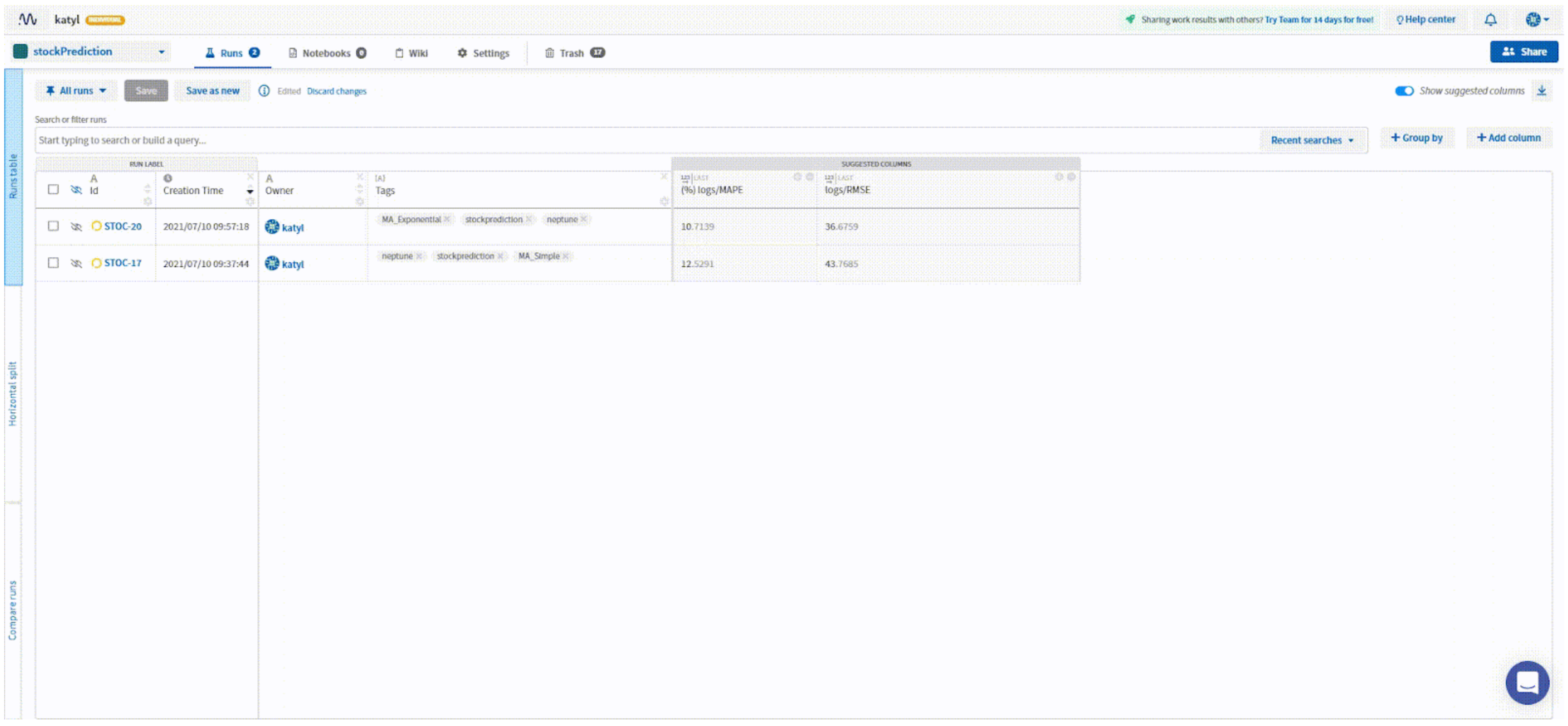
N = number of time points in EMA,

and weighting factor k = 2/(N+1).

One advantage of the EMA over SMA is that EMA is more responsive to price changes, which makes it useful for short-term trading. Here’s a Python implementation of EMA:

```
npt_exp = neptune.init(
        api_token=os.getenv('NEPTUNE_API_TOKEN'),
        project=myProject,
        name='EMA',
        description='stock-prediction-machine-learning',
        tags=['stockprediction', 'MA_Exponential', 'neptune'])

###### Exponential MA
window_ema_var = window_var+'_EMA'
# Calculate the 50-day exponentially weighted moving average
    stockprices[window_ema_var] = stockprices['Close'].ewm(span=window_size,
adjust=False).mean()
    stockprices['200day'] = stockprices['Close'].rolling(200).mean()

### Plot and performance metrics for EMA model
plot_stock_trend(var=window_ema_var, cur_title='Exponential Moving Averages',
logmodelName='Exp MA')
    rmse_ema, mape_ema = calculate_perf_metrics(var=window_ema_var,
logmodelName='Exp MA')
### Stop the run after logging for new version
npt_exp.stop()
```

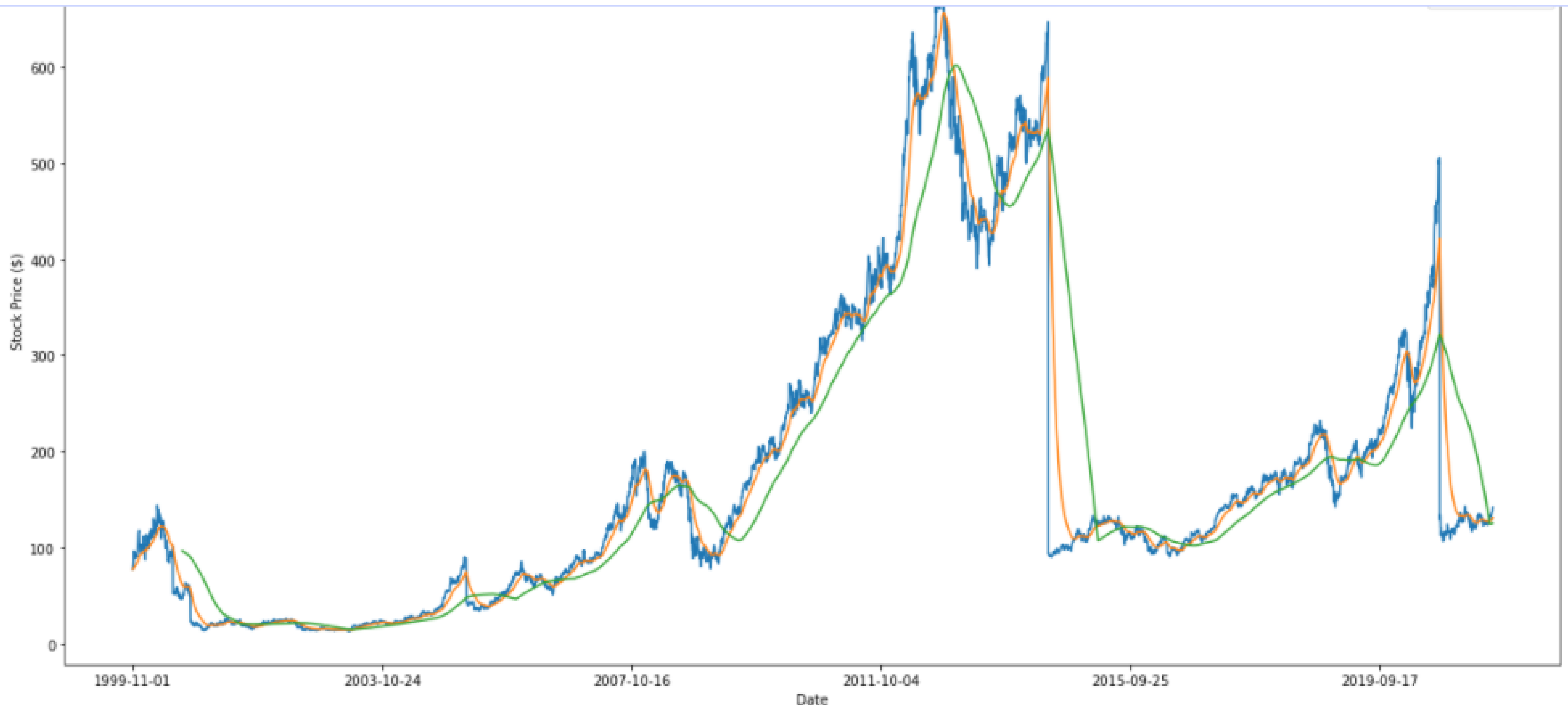
Examining the performance metrics tracked in Neptune, we have RMSE = 36.68, and MAPE = 10.71%, which is an improvement from SMA's 43.79 and 12.53% for RMSE and MAPE, respectively.



RUN LABEL				SUGGESTED COLUMNS	
Id	Creation Time	Owner	Tags	(%) logs/MAPE	logs/RMSE
STOC-20	2021/07/10 09:57:18	katyl	MA_Exponential, stockprediction, neptune	10.7139	36.6759
STOC-17	2021/07/10 09:37:44	katyl	neptune, stockprediction, MA_Simple	12.5291	43.7685

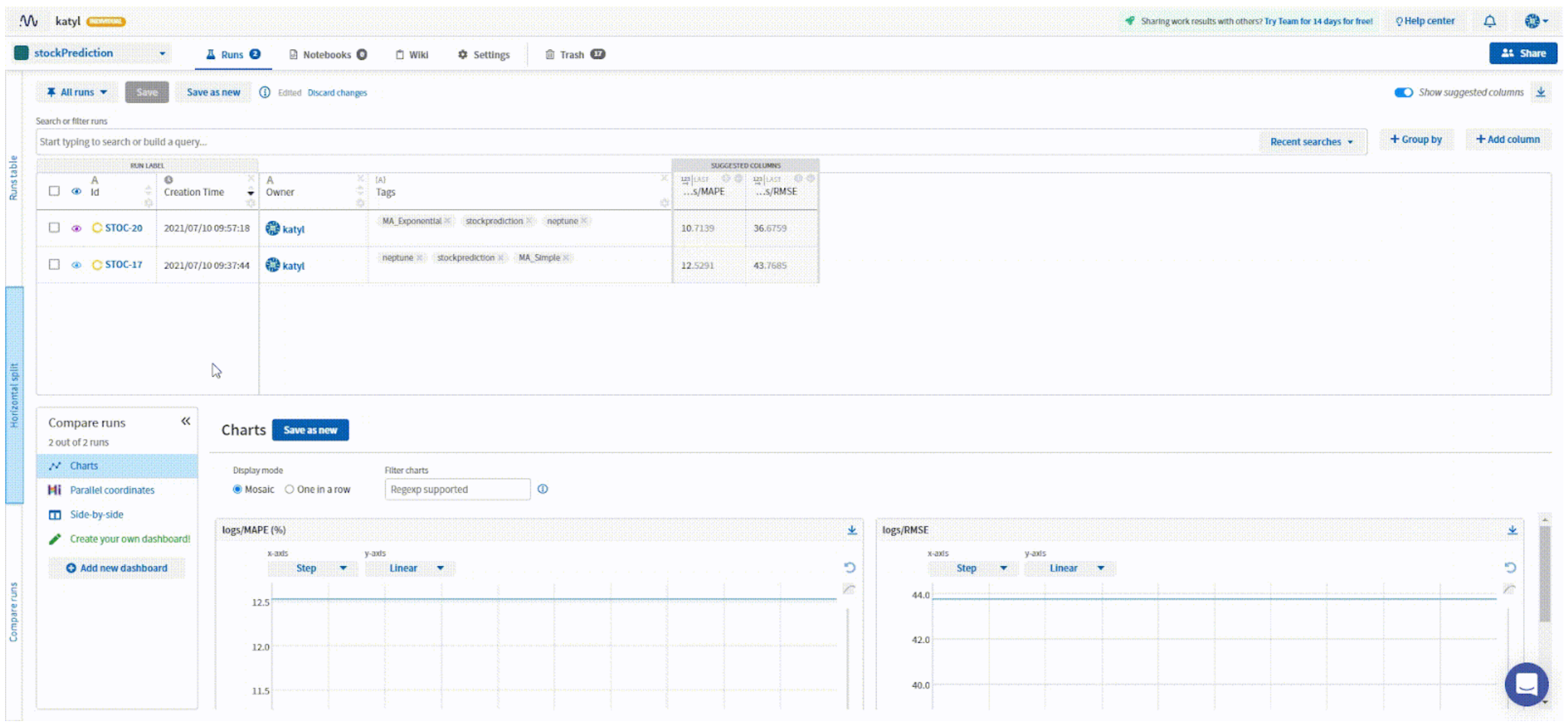
[See in the app](https://neptune.ai)

The trend chart generated from this EMA model also implies that it outperforms the SMA.



Comparison of the SMA and EMA prediction performance

The clip below shows a comparison of SMA and EMA side-by-side in Neptune; the blue and pink lines are SMA and EMA, respectively.



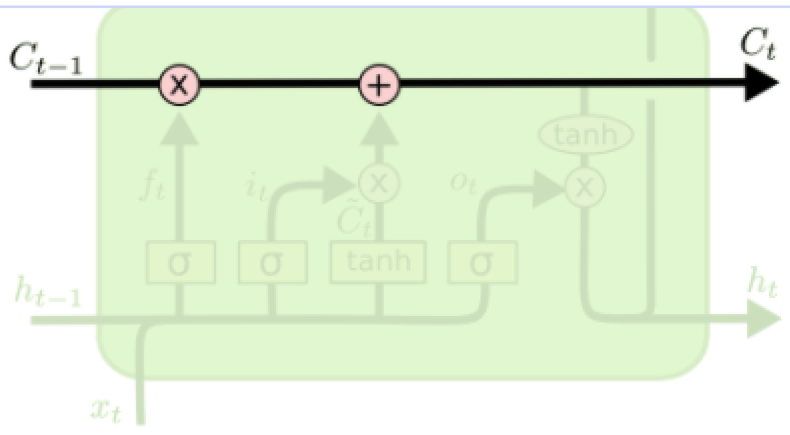
Id	Creation Time	Owner	Tags	...s/MAPE	...s/RMSE
STOC-20	2021/07/10 09:57:18	katyl	MA_Exponential, stockprediction, neptune	10.7139	36.6759
STOC-17	2021/07/10 09:37:44	katyl	neptune, stockprediction, MA_Simple	12.5291	43.7685

See in the app

Introduction to LSTMs for the time-series data

Now, let's move on to the LSTM model. LSTM, short for Long Short-term Memory, is an extremely powerful algorithm for time series. It can capture historical trend patterns, and predict future values with high accuracy.

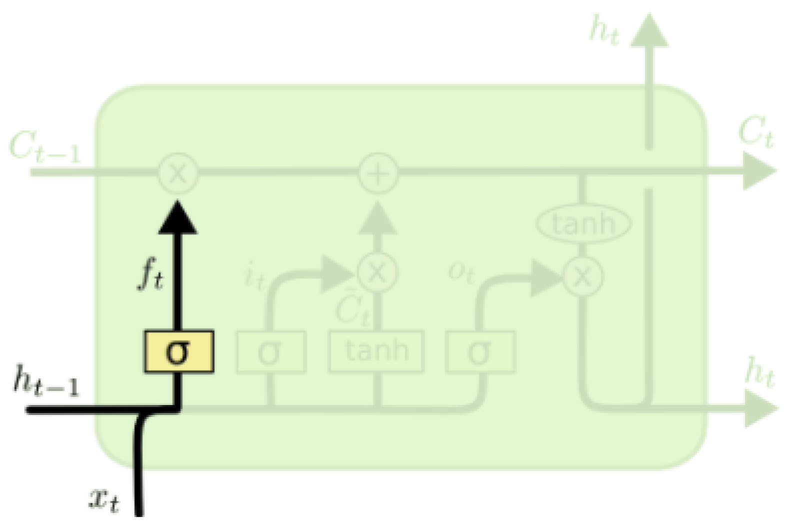
In a nutshell, the key component to understand an LSTM model is the Cell State (*Ct*), which represents the internal short-term and long-term memories of a cell.



Source

To control and manage the cell state, an LSTM model contains three gates/layers. It's worth mentioning that the "gates" here can be treated as filters to let information in (being remembered) or out (being forgotten).

- Forget gate:

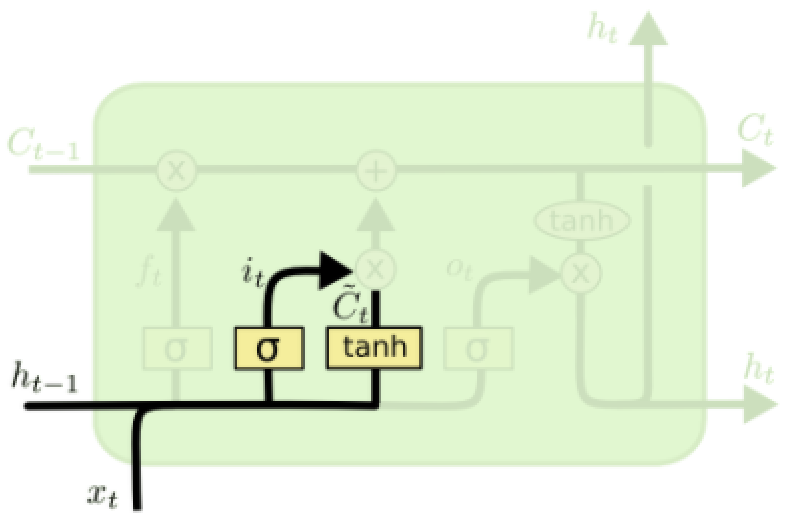


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Source

As the name implies, forget gate decides which information to throw away from the current cell state. Mathematically, it applies a sigmoid function to output/returns a value between [0, 1] for each value from the previous cell state (*Ct-1*); here '1' indicates "completely passing through" whereas '0' indicates "completely filtering out"

- Input gate:

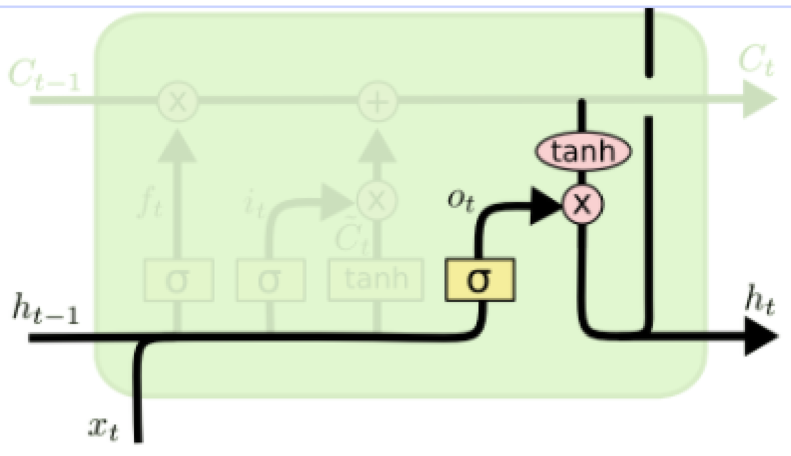


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Source

It's used to choose which new information gets added and stored in the current cell state. In this layer, a sigmoid function is implemented to reduce the values in the input vector (*it*), and then a tanh function squashes each value between [-1, 1] (*Ct*). Element-by-element matrix multiplication of *it* and *Ct* represents new information that needs to be added to the current cell state.

- Output gate:



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Source

The output gate is implemented to control the output flowing to the next cell state. Similar to the input gate, an output gate applies a sigmoid and then a tanh function to filter out unwanted information, keeping only what we've decided to let through.

For a more detailed understanding of LSTM, you can check out this document.

Knowing the theory of LSTM, you must be wondering how it does at predicting real-world stock prices. We'll find out in the next section, by building an LSTM model and comparing its performance against the two technical analysis models: SMA and EMA.

Predicting stock prices with an LSTM model

First, we need to create a Neptune experiment dedicated to LSTM, which includes the specified hyper-parameters.

```
layer_units, optimizer = 50, 'adam'
    cur_epochs = 15
    cur_batch_size = 20

    cur_LSTM_pars = {'units': layer_units,
                     'optimizer': optimizer,
                     'batch_size': cur_batch_size,
                     'epochs': cur_epochs
                     }

# Create an experiment and log the model in Neptune new version
npt_exp = neptune.init(
        api_token=os.getenv('NEPTUNE_API_TOKEN'),
        project=myProject,
        name='LSTM',
        description='stock-prediction-machine-learning',
        tags=['stockprediction', 'LSTM','neptune'])
npt_exp['LSTMPars'] = cur_LSTM_pars
```

Next, we scale the input data for LSTM model regulation and split it into train and test sets.

```
scaled_data = scaler.fit_transform(stockprices[['Close']])
    scaled_data_train = scaled_data[:train.shape[0]]

# We use past 50 days’ stock prices for our training to predict the 51th day's
closing price.
X_train, y_train = extract_seqX_outcomeY(scaled_data_train, window_size,
window_size)
```

A couple of notes:

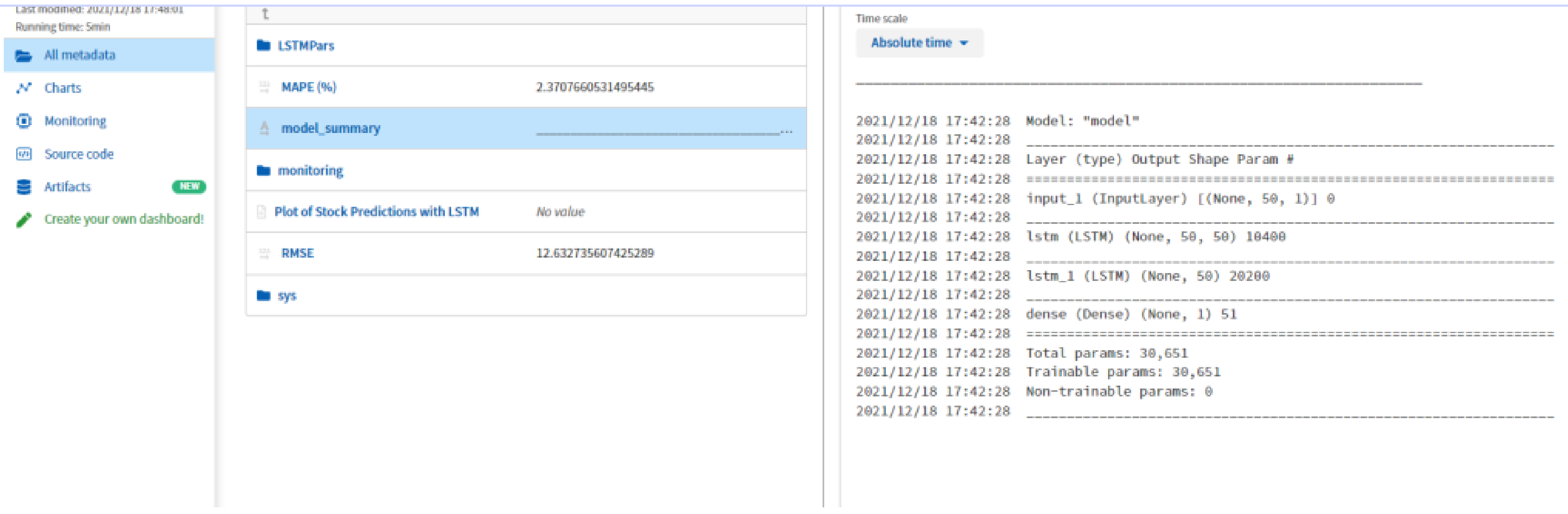
- we use the *StandardScaler*, rather than the *MinMaxScaler* as you might have seen before. The reason is that stock prices are ever-changing, and there are no true min or max values. It doesn’t make sense to use the *MinMaxScaler*, although this choice probably won’t lead to disastrous results at the end of the day;
- stock price data in its raw format can’t be used in an LSTM model directly; we need to transform it using our pre-defined `*extract_seqX_outcomeY*` function. For instance, to predict the 51st price, this function creates input vectors of 50 data points prior and uses the 51st price as the outcome value.

Moving on, let’s kick off the LSTM modeling process. Specifically, we’re building an LSTM with two hidden layers, and a ‘linear’ activation function upon the output. Also, this model is logged in Neptune.

```
### Build a LSTM model and log model summary to Neptune ###
def Run_LSTM(X_train, layer_units=50, logNeptune=True, NeptuneProject=None):
    inp = Input(shape=(X_train.shape[1], 1))

    x = LSTM(units=layer_units, return_sequences=True)(inp)
    x = LSTM(units=layer_units)(x)
    out = Dense(1, activation='linear')(x)
    model = Model(inp, out)

    # Compile the LSTM neural net
    model.compile(loss = 'mean_squared_error', optimizer = 'adam')

    ## !!! log to Neptune, e.g., set NeptuneProject = npt_exp (new version)
    if logNeptune:
       model.summary(print_fn=lambda x: NeptuneProject['model_summary'].log(x))

    return model

model = Run_LSTM(X_train, layer_units=layer_units, logNeptune=True,
NeptuneProject=npt_exp)

history = model.fit(X_train, y_train, epochs=cur_epochs, batch_size=cur_batch_size,
                    verbose=1, validation_split=0.1, shuffle=True)
```

Model hyper-parameters and summary have been logged into Neptune.

Privacy - Terms



LSTMPars	
MAPE (%)	2.3707660531495445
model_summary	______________________________...
monitoring	
Plot of Stock Predictions with LSTM	*No value*
RMSE	12.632735607425289
sys	

```
________________________________________________________________

2021/12/18 17:42:28  Model: "model"
2021/12/18 17:42:28  _________________________________________________________________
2021/12/18 17:42:28  Layer (type) Output Shape Param #
2021/12/18 17:42:28  =================================================================
2021/12/18 17:42:28  input_1 (InputLayer) [(None, 50, 1)] 0
2021/12/18 17:42:28  _________________________________________________________________
2021/12/18 17:42:28  lstm (LSTM) (None, 50, 50) 10400
2021/12/18 17:42:28  _________________________________________________________________
2021/12/18 17:42:28  lstm_1 (LSTM) (None, 50) 20200
2021/12/18 17:42:28  _________________________________________________________________
2021/12/18 17:42:28  dense (Dense) (None, 1) 51
2021/12/18 17:42:28  =================================================================
2021/12/18 17:42:28  Total params: 30,651
2021/12/18 17:42:28  Trainable params: 30,651
2021/12/18 17:42:28  Non-trainable params: 0
2021/12/18 17:42:28  _________________________________________________________________
```

See in the app

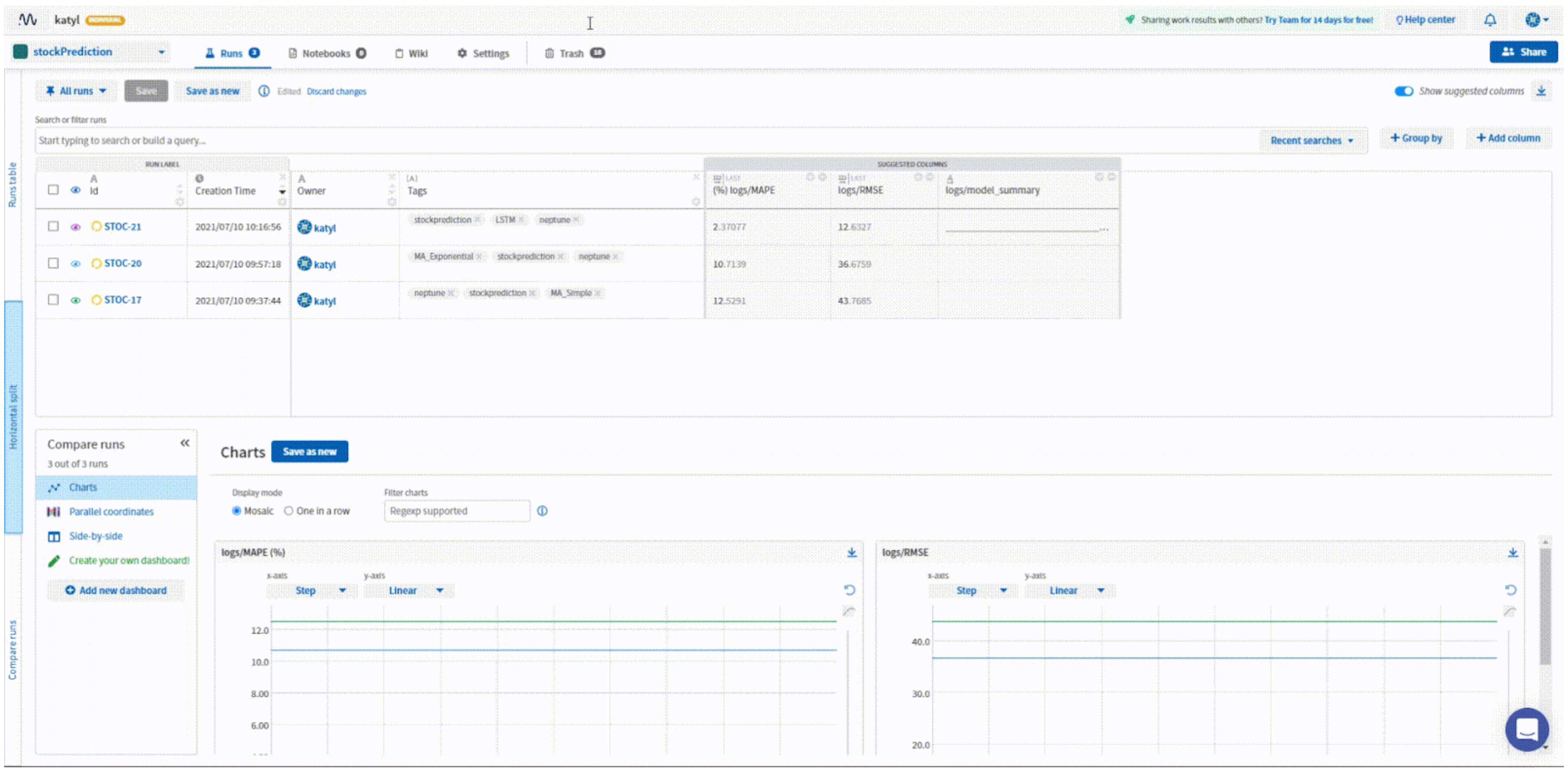
Once the training completes, we'll test the model against our hold-out set.

```
# predict stock prices using past window_size stock prices
def preprocess_testdat(data=stockprices, scaler=scaler, window_size=window_size,
test=test):
    raw = data['Close'][len(data) - len(test) - window_size:].values
    raw = raw.reshape(-1,1)
    raw = scaler.transform(raw)

    X_test = []
    for i in range(window_size, raw.shape[0]):
        X_test.append(raw[i-window_size:i, 0])

    X_test = np.array(X_test)

    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
    return X_test

X_test = preprocess_testdat()

predicted_price_ = model.predict(X_test)
predicted_price = scaler.inverse_transform(predicted_price_)

# Plot predicted price vs actual closing price
test['Predictions_lstm'] = predicted_price
```

Time to calculate the performance metrics and log them to Neptune.

```
rmse_lstm = calculate_rmse(np.array(test['Close']),
np.array(test['Predictions_lstm']))
mape_lstm = calculate_mape(np.array(test['Close']),
np.array(test['Predictions_lstm']))

### Neptune new version
npt_exp['RMSE'].log(rmse_lstm)
npt_exp['MAPE (%)'].log(mape_lstm)

### Plot prediction and true trends and log to Neptune
def plot_stock_trend_lstm(train, test, logNeptune=True):
    fig = plt.figure(figsize = (20,10))
    plt.plot(train['Date'], train['Close'], label = 'Train Closing Price')
    plt.plot(test['Date'], test['Close'], label = 'Test Closing Price')
    plt.plot(test['Date'], test['Predictions_lstm'], label = 'Predicted Closing
Price')
    plt.title('LSTM Model')
    plt.xlabel('Date')
    plt.ylabel('Stock Price ($)')
    plt.legend(loc="upper left")

## Log image to Neptune new version
    if logNeptune:
        npt_exp['Plot of Stock Predictions with
LSTM'].upload(neptune.types.File.as_image(fig))

plot_stock_trend_lstm(train, test)

### Stop the run after logging for new version
npt_exp.stop()
```

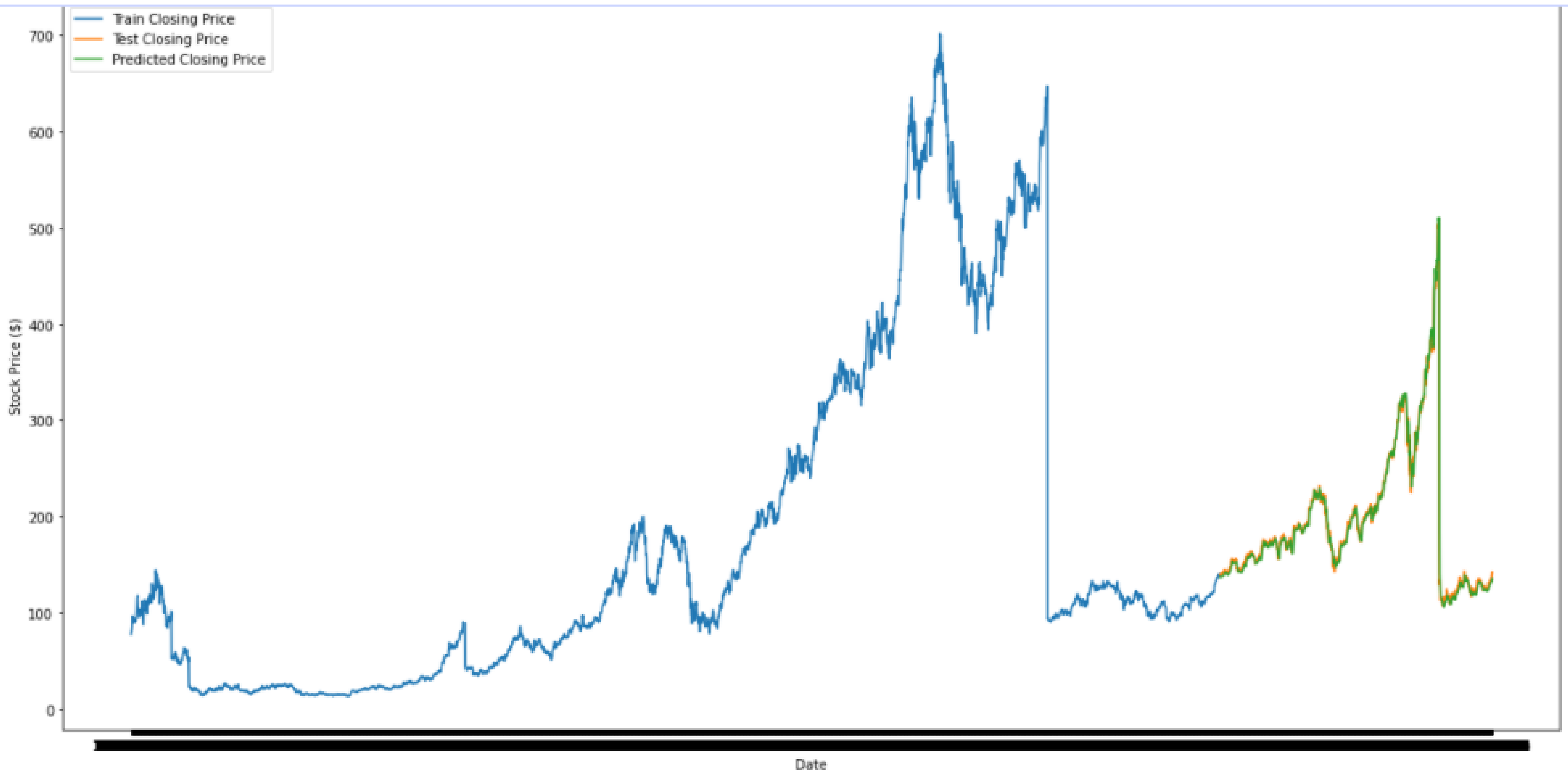
In Neptune, it's amazing to see that our LSTM model achieved an RMSE = 12.63 and MAPE = 2.37%; a tremendous improvement from the SMA and EMA models!



Id	Creation Time	Owner	Tags	(%) logs/MAPE	logs/RMSE	logs/model_summary
STOC-21	2021/07/10 10:16:56	katyl	stockprediction, LSTM, neptune	2.37077	12.6327	
STOC-20	2021/07/10 09:57:18	katyl	MA_Exponential, stockprediction, neptune	10.7139	36.6759	
STOC-17	2021/07/10 09:37:44	katyl	neptune, stockprediction, MA_Simple	12.5291	43.7685	

See in the app

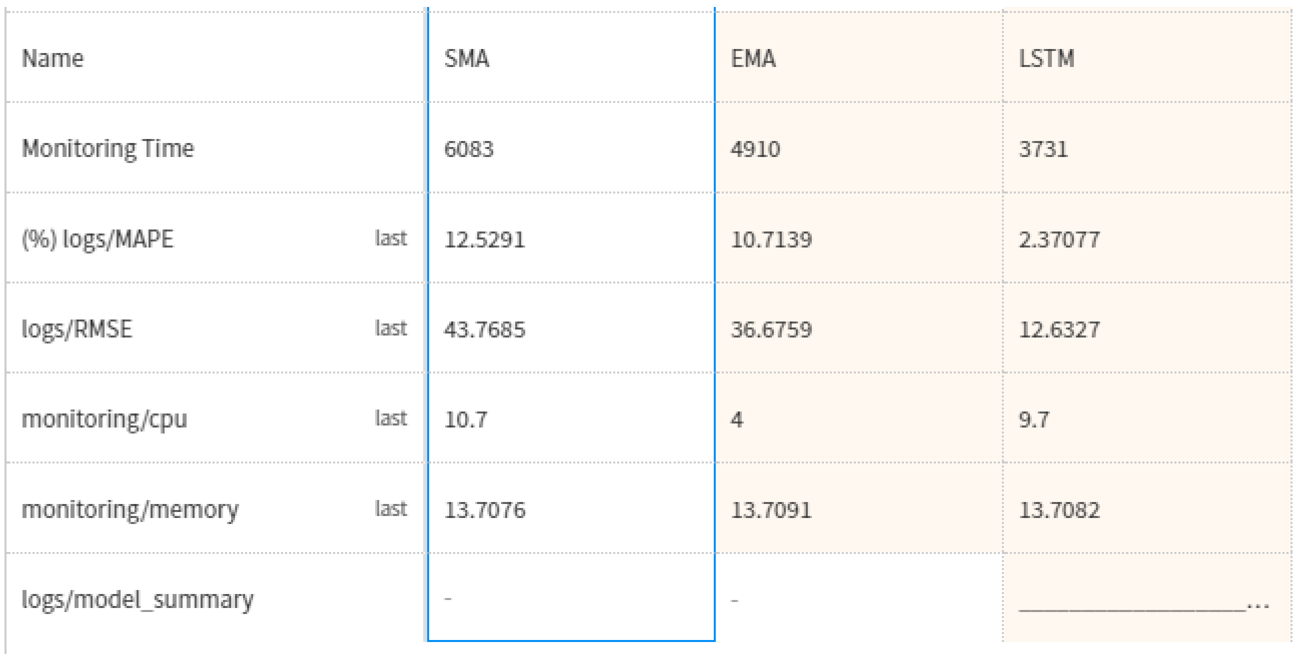
Privacy - Terms



Comparison of SMA, EMA, and LSTM models

Would the LSTM take up lots of computational and memory resources? After all, it's a more complicated algorithm than the traditional technical analysis models like SMA or EMA. If the LSTM model does indeed need much more resources to run, it would be a challenge to scale up, right?

Luckily, Neptune automatically monitors this information for us.



Name		SMA	EMA	LSTM
Monitoring Time		6083	4910	3731
(%) logs/MAPE	last	12.5291	10.7139	2.37077
logs/RMSE	last	43.7685	36.6759	12.6327
monitoring/cpu	last	10.7	4	9.7
monitoring/memory	last	13.7076	13.7091	13.7082
logs/model_summary		-	-	________________...

See in the app

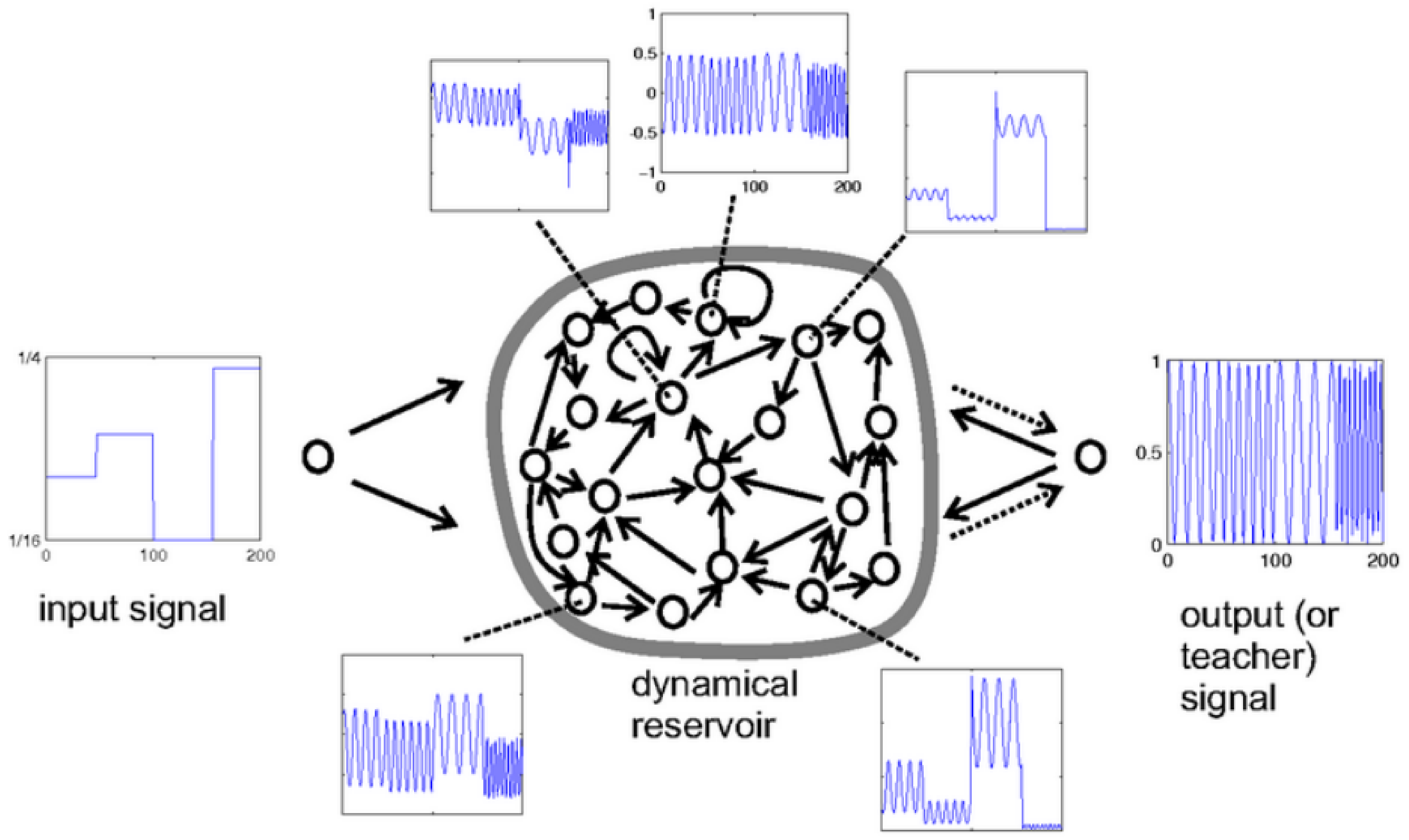
As we can see, the CPU and memory usages among these three models are fairly close; the LSTM model doesn't consume more resources than the MA models.

Final thoughts on new methodologies

We've seen the advantage of LSTMs in the example of predicting Apple stock prices compared to traditional MA models. Be careful about making generalizations to other stocks, because, unlike other stationary time series, stock market data is less-to-none seasonal and more chaotic.

In our example, Apple, as one of the biggest tech giants, has not only established a mature business model and management, its sales figures also benefit from the release of innovative products or services. Both contribute to the lower implied volatility of Apple stock, making the predictions relatively easier for the LSTM model in contrast to different, high-volatility stocks.

this hidden layer is referred to as the 'reservoir' designed to capture the non-linear history information of input data.



Schema of an Echo State Network (ESN)

At a high level, an ESN takes in a time-series input vector and maps it to a high-dimensional feature space, i.e. the dynamical reservoir (neurons aren't connected like a net but rather like a reservoir). Then, at the output layer, a linear activation function is applied to calculate the final predictions.

If you're interested in learning more about this methodology, check out the original paper by Jaeger and Haas.

In addition, it would be interesting to incorporate sentiment analysis on news and social media regarding the stock market in general, as well as a given stock of interest. Another promising approach for better stock price predictions is the hybrid model, where we add MA predictions as input vectors to the LSTM model. You might want to explore different methodologies, too.

Hope you enjoyed reading this article as much as I enjoyed writing it! Full code/scripts can be found in my Github repo here, and our current Neptune project is available here for your reference.

Katherine (Yi) Li

Data Scientist | Data Science Writer
A data enthusiast specializing in machine learning and data mining. Programming, coding and delivering data-driven insights are her passion. She believes that knowledge increases upon sharing; hence she writes about data science in hope of inspiring individuals who are embarking on a similar data science career.

Follow me on

READ NEXT

ML Experiment Tracking: What It Is, Why It Matters, and How to Implement It

10 mins read | Author Jakub Czakon | Updated July 14th, 2021

"… We were developing an ML model with my team, we ran a lot of experiments and got promising results…

…unfortunately, we couldn't tell exactly what performed best because we forgot to save some model parameters and dataset versions…

…after a few weeks, we weren't even sure what we have actually tried and we needed to re-run pretty much everything"

– unfortunate ML researcher.

And the truth is, when you develop ML models you will run a lot of experiments.

Those experiments may:

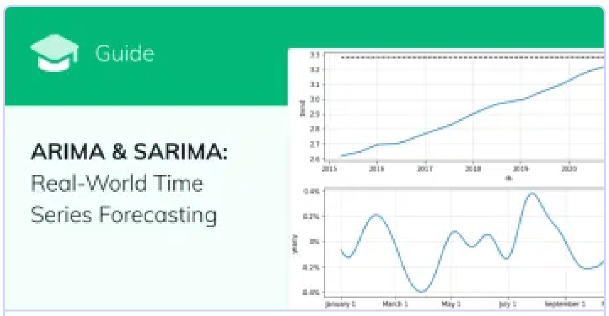
- use different models and model hyperparameters
- use different training or evaluation data,
- run different code (including this small change that you wanted to test quickly)
- run the same code in a different environment (not knowing which PyTorch or Tensorflow version was installed)

And as a result, they can produce completely different evaluation metrics.

Keeping track of all that information can very quickly become really hard. Especially if you want to organize and compare those experiments and feel confident that you know which setup produced the best result.

This is where ML experiment tracking comes in.

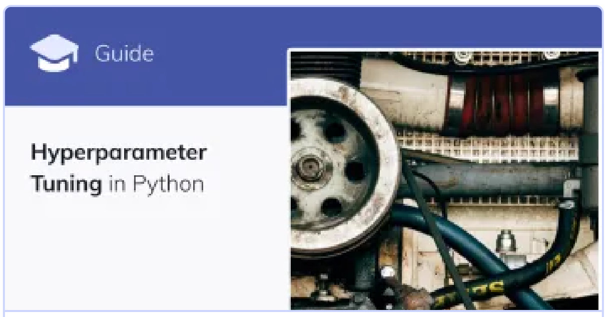
Continue reading ->



ARIMA & SARIMA: Real-World Time Series Forecasting

by Aayush Bajaj

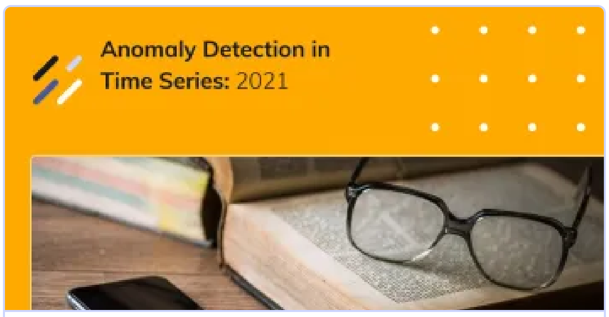
Read more



Hyperparameter Tuning in Python: a Complete Guide

by Shahul ES, Aayush Bajaj

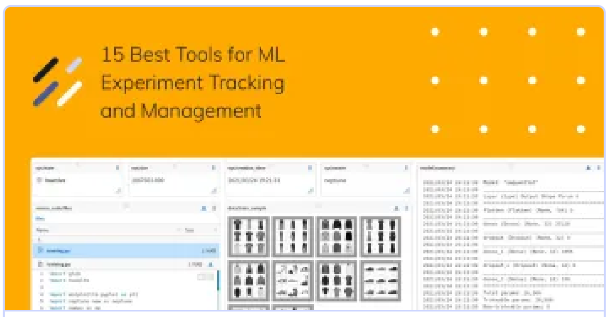
Read more



Anomaly Detection in Time Series

by Aayush Bajaj

Read more



15 Best Tools for ML Experiment Tracking and Management

by Patrycja Jenkner

Read more

Privacy - Terms

neptuneblog

Top MLOps articles from our blog in your inbox every month.

Type your email here Get Newsletter

GDPR compliant. Privacy policy.

Neptune is a metadata store for MLOps, built for research and production teams that run a lot of experiments.

Product

Overview
Experiment Tracking
Model Registry
ML Metadata Store
Notebooks in Neptune
Get Started
Python API
R Support
Pricing
Roadmap
Service Status

Legal

Terms of service
Privacy policy

Resources

Neptune Blog
Neptune Docs
Neptune Integrations
ML Experiment Tracking
ML Model Management
MLOps
ML Project Management

Competitor Comparison

Neptune vs Weights & Biases
Neptune vs MLflow
Neptune vs TensorBoard
Other Comparisons
ML Experiment Tracking Tools
Best MLflow Alternatives
Best TensorBoard Alternatives

Company

About us
Careers We are hiring!

Privacy - Terms