Stock Market Prediction

Probabilistic Machine Learning | Arne Michael Schulze, Joshua Nerling

Table of Contents

Motivation

Our Vision

- Predicting difficult because of noise and volatility

Can be rewarding

Our Goal

1. Predicting Prices
2. Assessing how uncertain the are

Dataset – Google Stock

Period of Time

2004 - 2024

Used Features

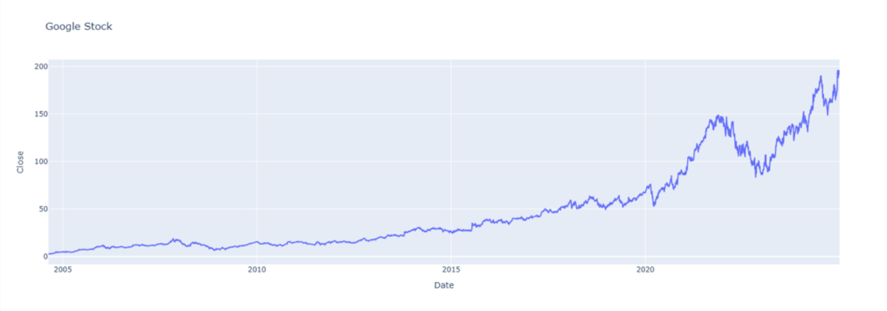
Column: Close

Source

Yahoo Finance

```
df = pd.read_csv('/content/GOOGL_historical_data.csv')
df.head()
```

	Date	Open	High	Low	Close	Volume
0	2004-08-20	2.515750	2.716741	2.503048	2.697563	456686856
1	2004-08-23	2.758334	2.826327	2.715994	2.724711	365122512
2	2004-08-24	2.770538	2.779504	2.579509	2.611887	304946748
3	2004-08-25	2.614129	2.689843	2.587231	2.640031	183772044
4	2004-08-26	2.613879	2.688597	2.606657	2.687601	141897960



Data processing

```
# === DATEN ===
# Extract the "Close" price column and reshape it to a 2D array (required by scaler and models)
close_data = df['Close'].values.reshape(-1, 1)

# Normalize the data to a range between 0 and 1 using MinMaxScaler
scaler = MinMaxScaler()
close_data_scaled = scaler.fit_transform(close_data)

# Define a training/testing split ratio (e.g., 80% training)
split_percent = 0.8
split_index = int(len(close_data_scaled) * split_percent)

# Create training and testing datasets
close_train = close_data_scaled[:split_index]
close_test = close_data_scaled[split_index:]

# Extract corresponding dates (for plotting later)
date_train = df['Date'][:split_index]
date_test = df['Date'][split_index:]

# Define how many past time steps to use as input (look-back window)
look_back = 10
```

Normalizing

Using MinMaxScaler()

Training/Testing Split

Using 0.8 split Ratio (80% Training)

Lookback

= 10

Methods – Used Models

LSTM

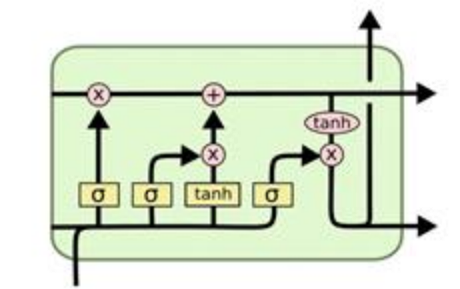


Illustration 1:
https://miro.medium.com/v2/resize:fit:1080/1*_2yXu6QhUihXUWEQn0bDkw.jpeg

Bayesian Linear Regression

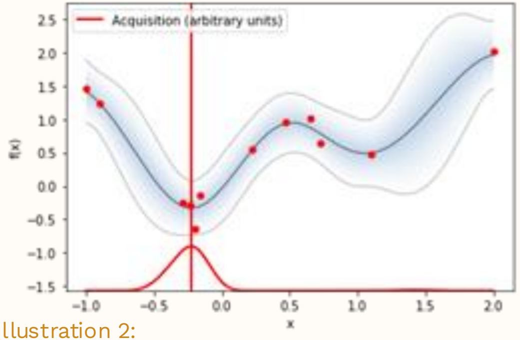


Illustration 2:
https://krasserm.github.io/img/2018-03-21/output_17_0.png

Methods – LSTM

Goal

Predicting future prices based on last 10 observed values using LSTM neural network

Preparation

- Converts time series into input/output sequences using a Keras Timeseries Generator

Input = window of "lookback" timesteps

Architecture LSTM

- LSTM layer with 10 Relu Units
- Adam Optimizer using Mean Squared Error as loss
- Trained on 100 epochs

Prediction

- Predicting on test data & inverse transform values to match the original scale again

Methods – LSTM

Goal

Predicting future prices based on last 10 observed values using LSTM neural network

Preparation

- Converts time series into input/output sequences using a Keras Timeseries Generator

💡 Input = window of "lookback" timesteps

Architecture LSTM

```
# Define a new LSTM model
model = Sequential()
model.add(LSTM(10, activation='relu', input_shape=(look_back, 1)))  # LSTM layer with 10 units
model.add(Dense(1))  # Single output neuron for regression
model.compile(optimizer='adam', loss='mse')  # Use Mean Squared Error as loss
model.fit(train_generator, epochs=100, verbose=1)  # Train the model on training data
model.save("lstm_model.h5")  # Save the trained model
print("✅ LSTM model trained and saved.")
```

Prediction

- Predicting on test data & inverse transform values to match the original scale again

Methods – Bayesian Regression

Goal

Use a probabilistic linear regression model to predict the next closing price

Preparation

Converts time series into input/output sequences using a helper function

```python
# Helper function to convert data into input/output format manually
def create_dataset(data, look_back):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:i+look_back, 0])  # Input: sequence of past values
        y.append(data[i + look_back, 0])  # Output: next value
    return np.array(X), np.array(y)
```

Methods – Bayesian Regression

Goal

Use a probabilistic linear regression model to predict the next closing price

Preparation

```
# Helper function to convert data into input/output format manually
def create_dataset(data, look_back):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:i+look_back, 0])  # Input: sequence of past values
        y.append(data[i + look_back, 0])  # Output: next value
    return np.array(X), np.array(y)
```

Architecture BR & Prediction

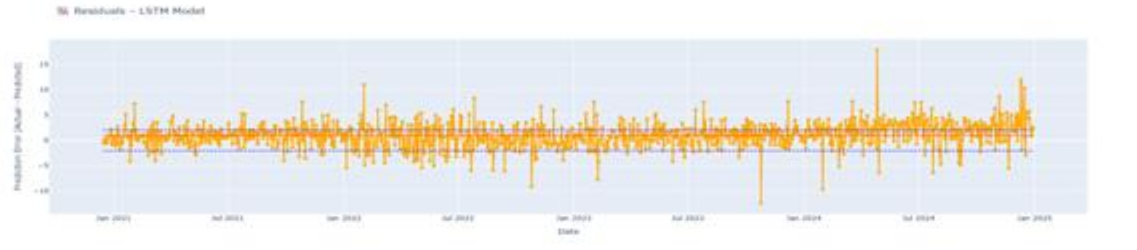
```
# Initialize and train a Bayesian Ridge Regression model
bayesian_model = BayesianRidge()
bayesian_model.fit(X_train, y_train)

# Make predictions and inverse-transform the results
bayesian_pred_scaled = bayesian_model.predict(X_test)
bayesian_pred = scaler.inverse_transform(bayesian_pred_scaled.reshape(-1, 1)).reshape(-1)

# Inverse-transform the actual target values for comparison
y_test_actual = scaler.inverse_transform(y_test.reshape(-1, 1)).reshape(-1)
date_test_bayes_adj = date_test[look_back:]

# Calculate error for Bayesian Regression
mse_bayes = mean_squared_error(y_test_actual, bayesian_pred)
print(f"📉 Mean Squared Error (Bayesian Regression): {mse_bayes:.4f}")
```

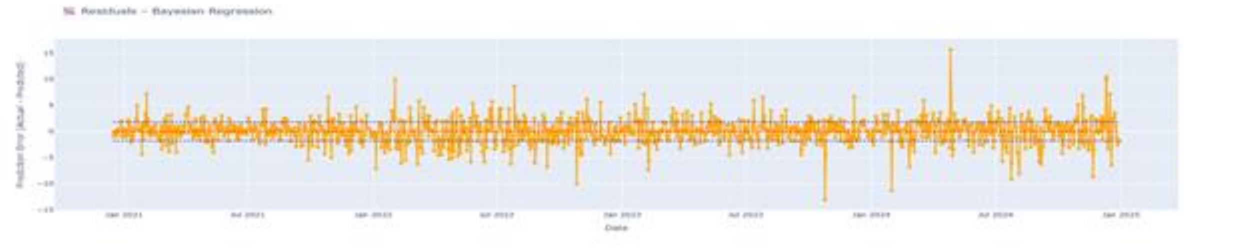
Results - LSTM



MSE LSTM: 7.4
RMSE LSTM: 2.72

Results - Bayesian Regression

MSE BR: 6.17
RMSE BR: 2.48



Roadmap: Next Steps

More Models

Implement models like XGBoost to evaluate trade-offs in accuracy & uncertainty

More Stocks

Test generalization on other stocks

Enrich Input

Add more variables to enhance precision and reduce model Variance

Webapp

Create a simple UI to improve usability

Discussion

What worked well?

Dataset was clean and easy to preprocess

LSTM produced reasonably accurate predictions

Challenges & Limitations

MSE doesn't reflect uncertainty in predictions

Relying on Closing data only may limit output

Open Questions

1. How can we better evaluate probabilistic forecasts beyond point-wise metrics?
2. Is it possible to combine LSTM and Bayesian models into a hybrid approach?

Thanks!

Do you have any questions?

as64nivi@studserv.uni-leipzig.de

cc80ahid@studserv.uni-leipzig.de