STOCK VOLATILITY FORECASTING REPORT

This is a report on analyzing and forecasting the stock price volatility based on data using ARIMA models, Recurrent Neural Networks (RNNs), Convolutional Neural Network (CNNs) and Generative Adversarial Network (GAN). For each, the best estimate for the volatility can be found in section 3.2, the generalized LSTM model. The project is based on one of my own deep learning projects and Using the latest advancements in deep learning to predict stock price movements (only for the GAN model). More details can be found in the source code and the Google colab notebooks.

1. DATA ANALYSIS AND CLEANING

This report uses a stock prices dataset provided by SIG. This dataset contains 6 different stocks a, b, c, d, e and f. Data were collected every 1 minute, beginning from dar 1, $9:30$ to day 362, $16:00$ pm. In total, we have 98353 rows (minutes) and 8 columns (day number, time string, and 6 stocks. It's important to make sure that we have a clean dataset, thus we did the following steps.

1.1. **Basic Manipulation.**

We first explored the dataset with standard pandas functionality: data dimensions, components, find null values, plot to identify abnormal behaviors, etc. The dataset is quite regular, most days contain data for the full 391 minutes from $9:30$ am to $4:00$ pm. Every month contains 21 days. However, the following flaws are identified and handled in the project, more observations on the dataset can be found in the Google Colab Notebook.

- Prices of a drops abnormally at some random places.
- Prices of a drops abnormally at some random places.
- Day 327 doesn't have the full number of minutes, the data stops at $1:00$ pm that day, it only contains 211 data points.

1.2. **Data cleaning.**

The times of the price drops in stock a and stock d seem to be random and appear on 93 different days. The price drops of stock a and d happened at the exactly same places. Abnormal values of a are all 0.0's, abnormal values of b are all 1.0's. It seems that these drops are recording errors. We set those abnormal values in the prices of stock a and stock d as NaN, which makes it compatible for our later use of the windows functionality of pandas.

For the same consideration, We insert NaN into the time slots after $1:00$ pm on day 327. We can still do all the statistics the same as the other days, the only difference is that we have less data to compute the volatility on day 327.

1.3. **Data transforms.**

We define the quantity (daily volatility measured in annualized monthly percent return) that we're interested in and related concepts.

Definition 1.1 (Annualized monthly percent return)**.** the annualized monthly percent return $A_m(t)$ at time t is given by

$$\begin{aligned} A_m(t) &= \left(\frac{\text{Price at time a month later than } t}{\text{Price at time } t}\right)^{\frac{1\,\text{year}}{1\,\text{month}}} - 1 \\ &= \left(\frac{p(t+t_m)}{p(t)}\right)^{12} - 1 \end{aligned},$$

where $p(t)$ denote the price at time t, we use minute as the unit, the same as in our dataset, t_m denote a month in unit of the dataset, in our case, $t_m = 391 \times 21 = 8211$.

Definition 1.2 (Daily volatility of annualized monthly percent return)**.** the daily volatility of annualized monthly percent return $\sigma_{d,m}(t)$ at time t is defined to be the sample deviation of the set $\{A_m(x)|t \le x \le t+t_d\}$,

$$\sigma_{d,m}(t) = \sqrt{\left[\frac{1}{t_d}\sum_{j=0}^{t_d}\left(A_m(t+j) - \frac{1}{t_d+1}\sum_{i=0}^{t_d}A_m(t+i)\right)^2\right]},$$

where t_d denote the number of minutes in a day, for us $t_d = 391$.

With the model assumptions that

- We're interested in the statistics on percent returns after holding an asset for a month (but we can always change this if we want later).
- Volatility is a constant in a day.

We use the rolling window functionality of pandas to compute the volatility of all the stocks. We find that if we compute with the full volatility data, it's very slow to train the deep learning models, thus we compute the daily volatility once, this means we view the volatility in a day as a constant. Plots and more detail analysis can be found in the Google Colab Notebooks.

2. Statistical analysis and forecasting

Though it seems that there's no obvious correlation among the 6 stocks, and some of them even behave like "noises". But for the completeness of the report, we compute several different correlations in order to test the validity of our observations (i.e. no two stocks are apparently correlated), and to chose source and target stocks for later ARIMA and deep learning models.

2.1. **Correlation and trend analysis.**

We find that the volatility of stock b and stock c have the highest correlation (0.864241) among all the pairs. stock b and stock f, stock c and stock f also have high positive correlation. All other pairs have relatively low correlations. By the dynamic time wrapping analysis, The change in the volatility of stock b slightly leads that of stock a. These observations explains our later choices of the source and target stocks in the deep learning models. For example, it's now reasonable to use stock a and stock f as targets and the other stocks as source for forecasting.

2.2. **ARIMA models for predicting volatility.**

We visualize our data using time series decomposition to decompose our data into three main components: trend, seasonality, and noise. It shows the obvious seasonality, for example, for stock a, we can find an alternating pattern, namely, the volatility of a alternate between high values and low values in a period of roughly $3-4$ months.

We also train an ARIMA (Autoregressive Integrated Moving Average) model to forecast the future Volatility. To get optimal output, we implement a grid search method to get the optimal parameters for the ARIMA mode. For example, the optimal ARIMA parameters for stock a are $(1,0,1)\times(0,0,1,12)$. Then we fit ARIMA models to predict next month's volatility. For example, the predicted values for stock a for the next month are given by

0.06660579 0.07960862 0.08916176 0.09618043 0.10133702 0.10512555 0.10790897

0.10995395 0.11145638 0.11256022 0.1133712 0.11396703 0.11440478

0.1147264 0.11496269 0.11513629 0.11526384 0.11535754 0.11542639 0.11547697 0.11551413.

Since it's **not our best model**, we don't include too many details in the report, plots and analysis can be found in the Google colab Notebooks.

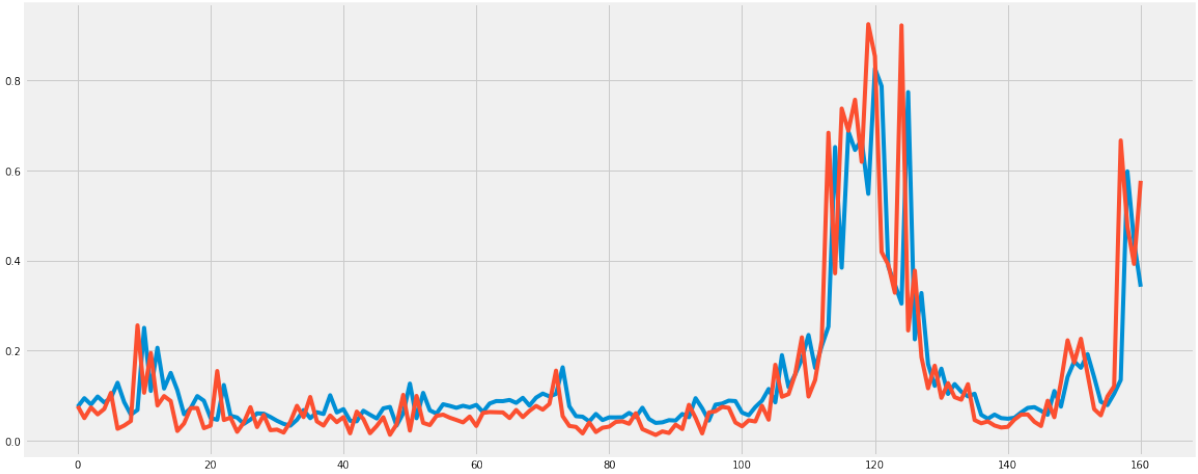

Figure 1. Single-feature LSTM: Training performance on stock a

3. Deep learning models

Recurrent Neural Networks (RNNs) are good fits for time-series analysis because RNNs keep record of historic data and they're designed to capture patterns developing through time. However, vanilla RNNs have major disadvantages such as the vanishing gradient problem. LSTM (Long-Short Term Memory) is a variation of vanilla RNNS, it overcomes the vanishing gradient (or gradient explode) problem by clipping gradients if they exceed some constant bounds. They also handle the relative importance of more and less recent samples well. We developed several LSTM models from single-step, single-feature to multi-step, multi-feature forecasting. **The best results in our projects are realized by the generalized multi-step, multi-feature LSTM models**. Building details can be found in the Google colab notebooks or the source code.

3.1. **Basic model: single-step, single-feature forecasting with LSTM.** We only trained the model for 100 epochs, feel free to modify the code to any number as long as we have enough computing power. Here are some results we find during the experiments that LSTM with Adam or RMSprop optimizers work better than the SGD optimizer in this project. Each model fits the training dataset very well. For example, the training data, predicted data compared with the actual data for the volatility of stock a are shown in the Figure 1 and Figure 2. The prediction captures the range and characteristics of the real data. Most importantly, the model predict the large rises or drops in the volatility before they happen correctly many times, which is very important in real investments.

3.2. **Generalized model: multi-step, multi-feature forecasting with LSTM.** We build a multi-step, multi-feature LSTM model. That means we can use several-day's data to predict multiple-day's volatility values and multiple features in the future spontaneously. For example, we can use last 12-day's data of a, b, c, f to predict next three-day's values of a and f. The results are shown in Figure 3 and Figure 4. To read the figures, the long line is the real data, each short line segment is a 3-day prediction: the start, middle, end point of the line segment means the prediction for the next 1, 2, 3 day's volatility respectively. Though the dataset is not big enough, we still successfully capture several features in the prediction. For example, the model predictions shows similar trend as the real data, e.g. from the predicted f data above, the predicted values are more or less in the most correct range and goes in the same direction as the real data.

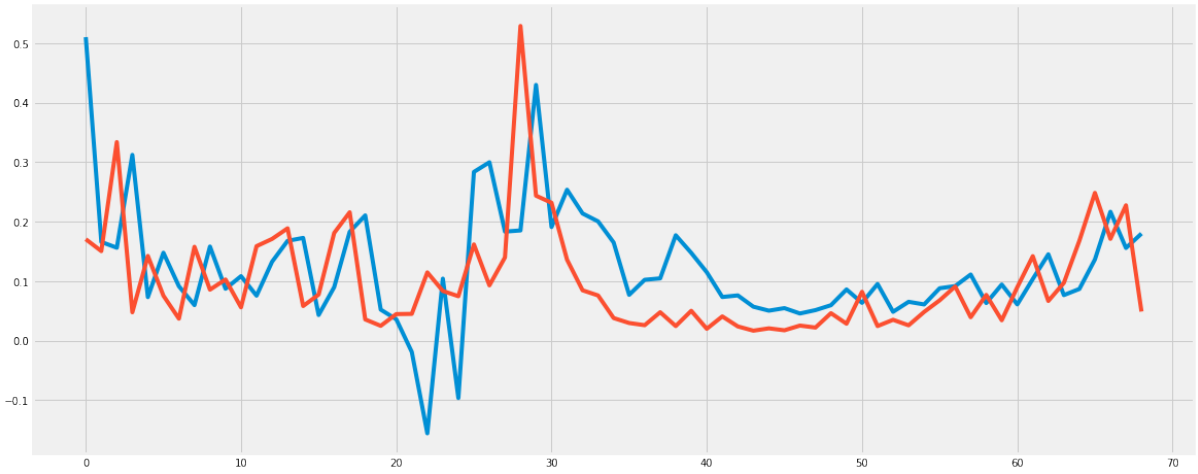

Figure 2. Single-feature LSTM: Forecasting performance on stock a

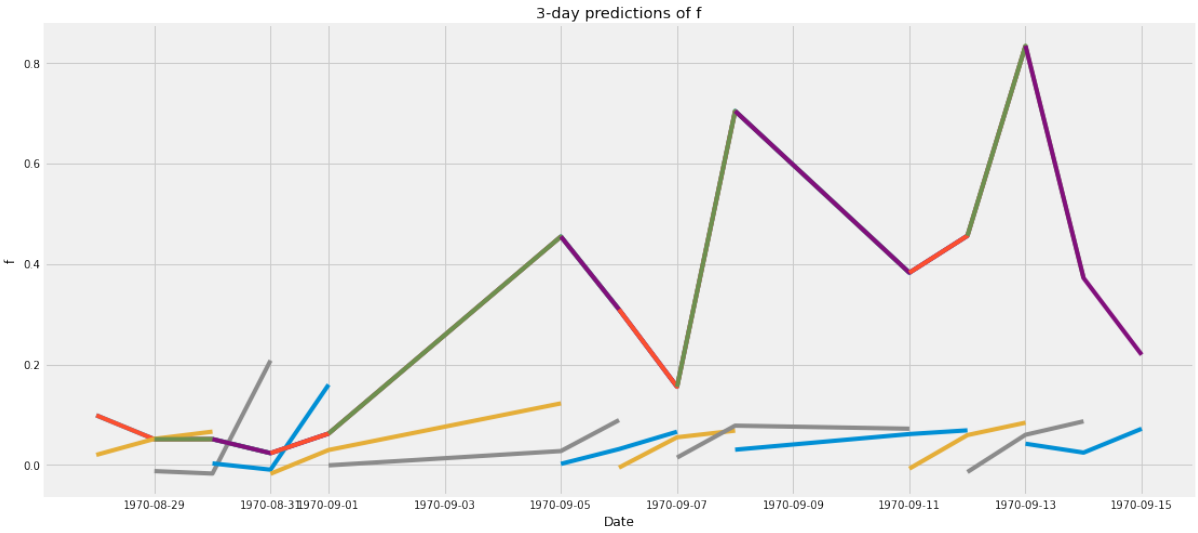

Figure 3. Multiple-feature LSTM: Forecasting performance on stock a

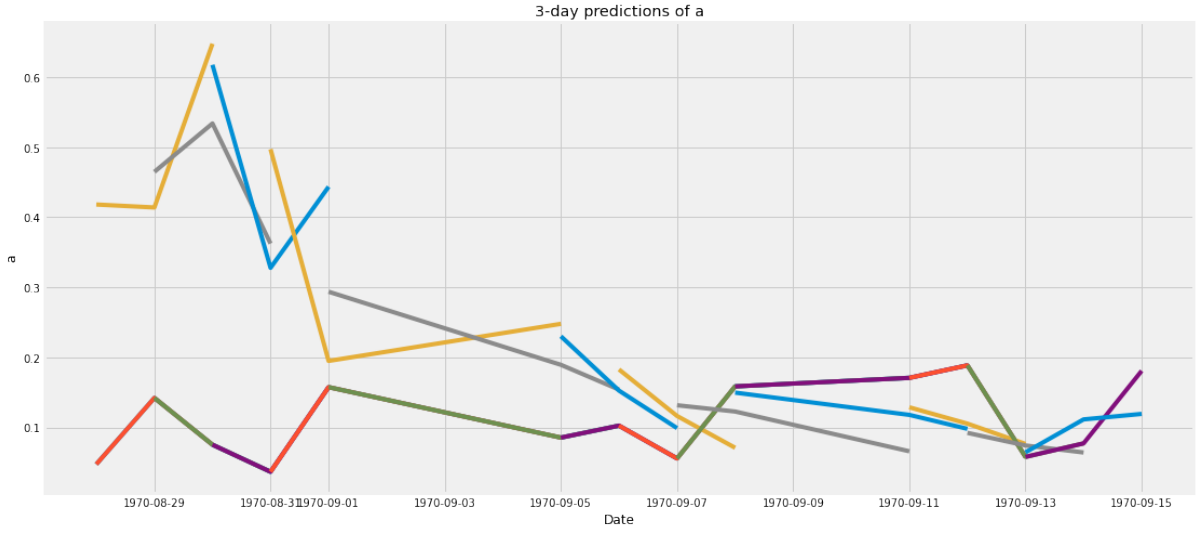

Figure 4. Multiple-feature LSTM: Forecasting performance on stock f

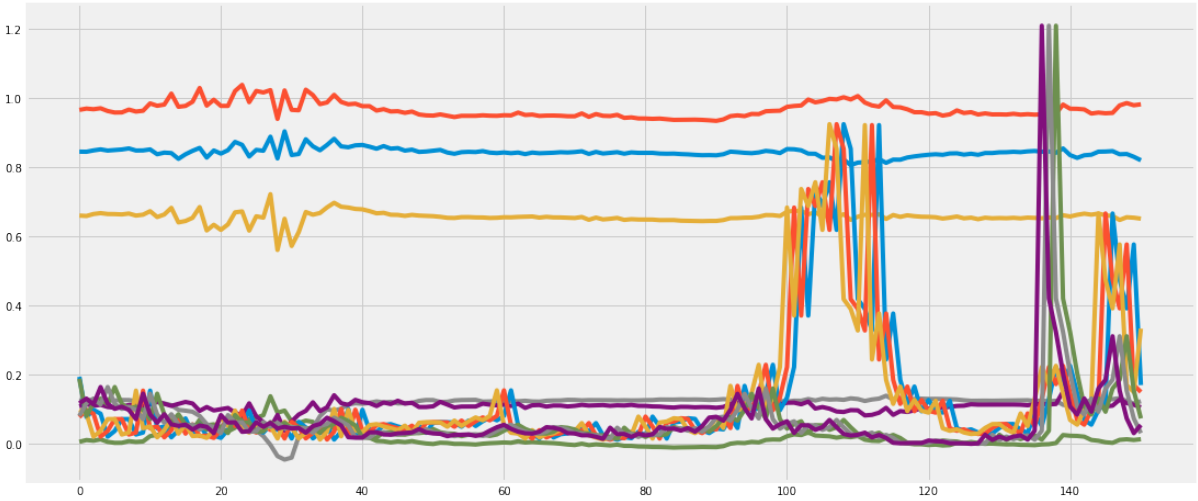

Figure 5. Multiple-feature GAN: Forecasting performance on stock a

All 3-day predictions are liquid, which means the models successfully capture the local behavior of the volatility, which is also in agreement with our model assumptions.

3.3. **Advanced model: Generative Adversarial Network (GAN) with RNN and CNN.** Generative Adversarial Networks (GAN) have been a successful model in generating realistic images, paintings, and videos. The idea that GANs can to used to predict time-series data is new and experimental. We know GANs are powerful in learning characteristics of data, our model is based on the assumptions that

- Values of a feature has certain patterns and behavior (characteristics).
- The future values of a feature should follow more or less the same pattern or behavior (unless it starts operating in a totally different way, or the economy drastically changes.

We use LSTM as a time-series generator and a 1-dimensional CNN as a discriminator. The implementation details can be found in the Google Colab Notebooks and the source code. One training epoch is shown in Figure 5. Our LSTM generator is not pre-trained, which means that the GAN model learns from scratch, though we didn't get results as good as the previous models, but this experimental model shows potentials, as we can see the GAN model successfully learns the correct range. It also learns the most drastic characteristics of the data, namely, the sudden huge changes.

4. Conclusions and Next steps

We applied Arima models and built 3 different deep learning models to make future predictions. LSTM models work well both in the single-feature forecasting and multi-feature forecasting. More conclusions and remarks can be found in the Google Colab Notebooks. We address the aspects relevant to the requirements in the data exercise statement

- The accuracy rate for going up and going down from the LSTM models are all above 80%. The highest value accuracy rate is 53%.
- The higher the sample frequency, the harder to train the deep learning models, we chose to work with the daily volatility data, the code can be modified easily to include all other possible sample frequencies.
- The relative importance of more and less recent samples is handled by the LSTM models, which emphasize more on more recent samples than less recent ones.