Stock Price Prediction using LSTM, CNN, CNN-LSTM and ARIMA

David Ocepek

Abstract—Stock price forecasting is a long-term area of interest for the financial world with many new models being proposed on a monthly basis. In this paper we conducted a comparative analysis of LSTM, CNN, CNN-LSTM and ARIMA on five tech stock data sets obtained from YAHOO! Finance. We built upon our analysis by using ARIMA to remove seasonality and trend from our time series and analyzed the room for improvement that hybrid models, such as ARIMA-CNN can offer. We used our models to predict the next day opening price of each stock and quantified the stability of our results using cross-validation on a rolling basis. Our results show that ARIMA is overall the best model for short-term time series forecasting compared to non-hybrid models, furthermore it appears that we can improve ARIMA models by using CNNs or CNN-LSTMs to reduce residual errors of the ARIMA model.

Index Terms—Stock price forecasting, ARIMA model, convolutional neural networks(CNN), Long Short-term memory networks(LSTM), convolutional long short-term memory(CNN-LSTM), hybrid models

1 Introduction

AUTOMATED stock trading systems are commonly used in the financial sector to maximize profits by optimizing stock prediction models. For example, in 2016 more than 80% of transactions in the Forex market were directed by robots [1]. Because of widespread automatization even small improvements in model prediction accuracy would yield large profits for financial institutions. This poses an interesting problem for data scientists because stock price forecasting is notoriously difficult due high levels of noise present in financial time series, a side-effect of market efficiency.

In our paper we focused on next-day opening price prediction using multiple neural network models. As the baseline we used the statistical ARIMA model. In addition, we conducted a comparative analysis of ARIMA-neural network hybrid models. We explained and verified our results using the LIME algorithm.

Our contributions are minor and mainly in the domain of result reproducibility and improvement of hybrid model understanding by performing additional analysis in some less-analyzed areas such as quantification of model uncertainty with cross-validation on a rolling basis and cross-model comparison with LIME.

1.1 Related Work

Stock price forecasting is an actively researched field and a lot of work has been done in recent years to optimize stock price prediction. Simpler models like ARIMA have lately been used to detrend and deseasonalize financial time series.

In 2017, Selvin et al. [2] used AR,ARMA and ARIMA to improve RNN, LSTM and CNN model predictions of closing prices of multiple NSE listed companies. In 2019 Lei Ji et al. [3] used the ARIMA model to improve and analize carbon forecasting of CNN-LSTM. In 2019 Hoseinzade et al. [4] used 3D CNNs to combine features from multiple markets to improve next-day direction of movement prediction of the S&P 500, NASDAQ, DJI, NYSE and RUSSELL indices. In 2020 Yan Wang et al. [5] used an ARIMA XGBoost model to predict the opening price of 10 chinese stocks. In 2021 Qi Tang et al. [6] used Wavelet Transform (WT) and Singular Spectrum Analysis (SSA) to denoise the Dow Jones Industrial Average Index. This resulted in improved neural network prediction models. In 2021 Xuan Ji et al. [7] proposed using word2vec to build text feature vectors from social media and combining extracted features with stock market features as inputs to SSA smoothed LSTMs.

2 METHODOLOGY

2.1 Data sets

We used the stock data sets of Apple (AAPL), Amazon (AMZN), Facebook (FB), Google (GOOG) and Microsoft (MSFT) obtained from Yahoo! Finance [8]. Each dataset had the following daily information: Opening price, Closing price, Highest and Lowest stock price, Volume of stocks traded and Adjusted Closing price, which is the closing price adjusted for the companies actions such as paying of dividends.

2.2 Models

2.2.1 ARIMA

The auto-regressive integrated moving average (ARIMA) [9] is a statistical model commonly used as a baseline for most time series models, as it is relatively fast and has high prediction accuracy for short-term forecasting.

ARIMA is comprised of two parts: auto-regression (AR) and moving averages (MA). Prior to their usage the time series is differentiated to make it stationary, the (I) as in integrated, representing differentiation.

ARIMA is formally defined in Equation 1

$$\begin{aligned} y_t^{'} = c + \phi_1 y_{t-1}^{'} + \dots + \phi_p y_{t-p}^{'} + \\ \theta_1 \epsilon_{t-1} + \dots + \theta_q \epsilon_{t-q} + \epsilon_t, \end{aligned} \tag{1}$$

where lagged values are represented as y_t and lagged prediction errors are denoted as ϵ_t, while ϕ_i and θ_j represent the learned model parameters.

In our paper we used ARIMA(1,1,1).

2.2.2 LSTM

Long-short term memory (LSTM) neural network is a commonly used model for time series forecasting, mainly due to its ability to discover long term dependencies in sequential data. The structure of a LSTM cell is show in Figure 1. A LSTM cell is comprised of three gates: input, output and forget. The LSTM cell has an internal state C_t, which can be modified by the input gate and is forwarded to the next layer by the output gate; the forget gate learns how much of the previous state the cell should forget. For the activation function we used the sigmoid and tanh activation function.

The learning process of a LSTM cell is comprised of three stages. In the first stage part of the information is eliminated by the forget gate.

$$f_t = \sigma(b_f + W_f x_t + U_f h_{t-1}), \tag{2}$$

where x_t represents the input of the current cell, h_{t-1} represents the output of the previous cell and σ represents the sigmoid activation function, while W and U are model weights and b is bias. The second stage updates the status of information in the cell. It is formally defined in Equation 3

$$\begin{aligned} i_t &= \sigma(b_i + W_i x_t + U_i h_{t-1}), \\ \tilde{C}_t &= tanh(b_c + W_c x_t + U_c h_{t-1}), \\ C_t &= f_t \times C_{t-1} + i_t \times \tilde{C}_t. \end{aligned} \tag{3}$$

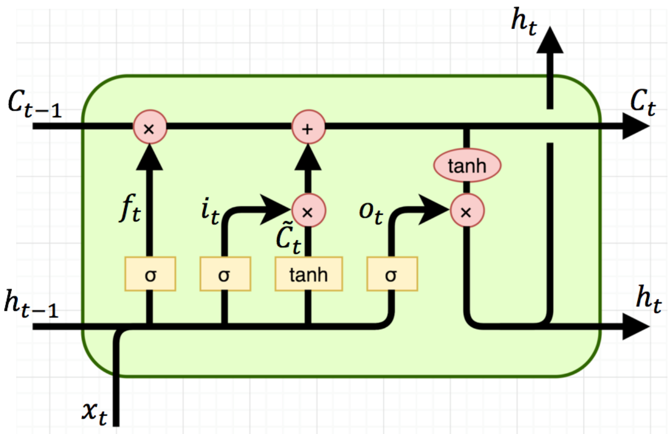

Fig. 1. **LSTM Cell.** x_t is cell input, C_t is the current cell state, h_t cell output.

In the third stage we use the sigmoid function to check which part O_t of the cell state our neuron outputs, then that part is preprocessed with the tanh activation function and outputted. The third stage is described in Equation 4

$$\begin{aligned} O_t &= \sigma(b_o + W_o x_t + U_o h_{t-1}), \\ h_t &= O_t \times tanh(C_t). \end{aligned} \tag{4}$$

For our paper we used a many-to-one stacked stateless LSTM-ANN neural network.

We used three LSTM layers followed by one ANN layer; all layers were comprised of 100 neurons. For regularization a dropout rate of 0.2 was used and a weight decay of $l1 = 0.00001$, $l2 = 0.000001$. We did not use Batch normalization for our LSTM model. We normalized our data and used a lag window of size 60. We trained our LSTM model for 200 epochs with pre-emptive stopping.

2.2.3 CNN

For our second model we used a stacked 2D CNN-ANN. CNN models are commonly used to extract and combine hierarchical data from time series features [4].

A convolutional layer is comprised of a filter applied to the input matrix $[v_{i,j}]$. Each filter utilizes a shared set of weights to perform convolution, which are updated during training. The filter output is feed into an activation unit. We used ReLU as our activation unit of choice for our CNN layers.

As our CNN input we created an image where the first dimension represented our j stock features and the second the i time lags. In the first layer we combined our features with a 1d convolution layer with kernel size of six, in the second layer we combined our lags with a 1d convolution with kernel size three.

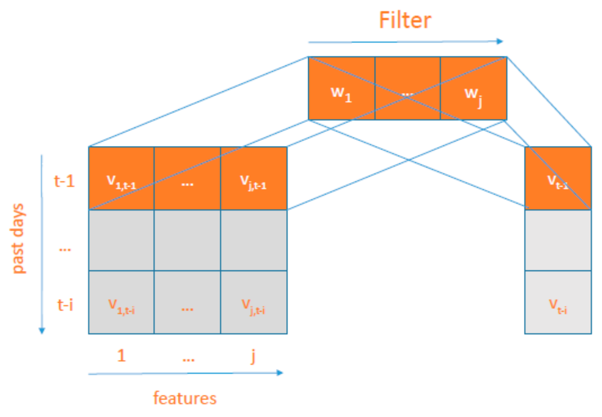

Fig. 2. **2D CNN.** Extracting and combining data from features and lags.

The formal transformation preformed by our convolutional layers in given by Equation 5

$$v_{i,j} = \delta(\Sigma_{k=0}^{F-1}\Sigma_{m=0}^{F-1} w_{k,m} V_{i+k,j+m}), \tag{5}$$

where V is the input image, F is the symmetric filter size, w are the filter weights and δ is the activation function.

We used a max polling layer to sub sample our data, in order to reduce the computational cost of our learning process as well as reduce overfitting.

Our CNN is structured as follows: the first two layers are 1d Conv layers, followed by a maxpool layer, followed by 3 ANN layers with 100, 200 and 200 neurons respectively, after each layer we used batch normalization. For regularization a dropout rate of 0.2 was used and a weight decay of $l1 = 0.0000001$, $l2 = 0.0000001$. We standardized our data and used a lag window of size 18. We trained our CNN model for 200 epochs with pre-emptive stopping.

2.2.4 CNN-LSTM

Our third and final model is the CNN-LSTM-ANN model which uses both CNN and LSTM layers. The Conv layers are used to capture hierarchical nonlinear data, while the LSTM layers are used to capture long-term dependencies in our data. The CNN-LSTM has been shown to outperform both LSTM and CNN [3].

Unlike in the previous model the Conv layers are used exclusively to extract spatial temporal data and not to extract inter-feature information. The CNN-LSTM structure is shown in figure 3.

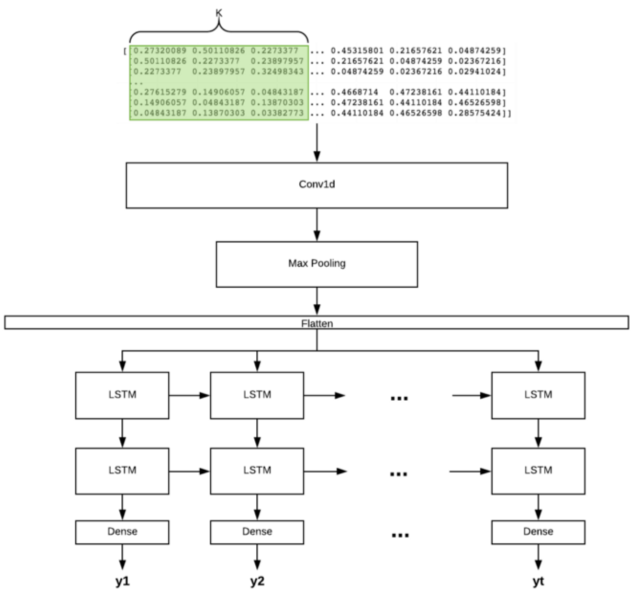

Fig. 3. **CNN-LSTM.** Extracted data from CNN is used in LSTM.

Our CNN-LSTM model was constructed as follows: our first two layers were 1d Conv layers with kernel size three wrapped in the *TimeDistributed* layer, following were two LSTM layers with 100 neurons each, followed by three ANN layers with 100 neurons each. For regularization a dropout rate of 0.2 was used and a weight decay of $l1 = 0.0000001$, $l2 = 0.0000001$. We standardized our data and used a lag window of size 18. We trained our CNN-LSTM model for 500 epochs with preemptive stopping.

2.3 Deseasonilizing & detrending

Most training models work better when the time series is stationarized and the data is deseasonalized and detrended. We used the ARIMA model to remove the trend and seasonality of our time series. For each data set we iteratively predicted the next-day values with ARIMA and then subtracted the result from our data gaining residuals. In order to test the stationarity of our time series we used the Augmented Dickey-Fuller (ADF) statistical test, achiving a p-value of 0.01 on all residual dataset. We then used our models to predict the residuals. We compared $RMSE$ to the original ARIMA $RMSE$; the $RMSE$ of our models residuals is the same as our $RMSE$ on the original datasets, therefore we can compare them.

2.4 Model Explainability

In order to explain our results we used Locally Interpretable Model-Agnostic Explanations [10] (LIME). LIME perturbs our sample data point x in order to get perturbations $z^{'}$ which are then evaluated by an interpretable model g which approximates our model $f(z)$, finally values $g(z^{'})$ are used to approximate our feature value z. A model is considered interpretable if we can visually inspect our results and verify our assumptions about the model. The current python implementation of LIME uses lines to explain models. This does require our model to be locally linear. The formal definition of LIME is given in Equation 6

$$\begin{aligned}\xi(x) &= \underset{g \in G}{\arg\min} \mathcal{L}(f, g, \pi_x) + \Omega(g),\\ \mathcal{L}(f, g, \pi_x) &= \Sigma \pi_x(z) d(f(z), g(z^{'})),\end{aligned} \tag{6}$$

where π_x is probability of sampling point z and d is the cosine distance.

We aggregated the explanation to get marginal explanations for both features and lags.

2.5 Evaluation

We evaluated our models using cross-validation on a rolling basis. Figure 4 show the core idea of cross-validation on a rolling basis. We picked 10 six year intervals shifted by half a year. In each interval the first five years were used for training and validation while the last year was used for model testing. We used mean absolute percentage error (MAPE) for cross-model comparison and root mean squared error (RMSE) for an interpretable quantification of error. For quantification of uncertainty standard deviation of absolute percentage error (SDAPE) was used, furthermore standard deviation for cross-validation was also calculated.

$$RMSE = \sqrt{\frac{1}{N}\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2}$$
$$MAPE = \frac{1}{N}\Sigma_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}| \quad (7)$$
$$SDAPE = \sqrt{\frac{1}{N}\Sigma_{i=1}^{N}(\frac{y_i - \hat{y}_i}{y_i} - MAPE)}$$

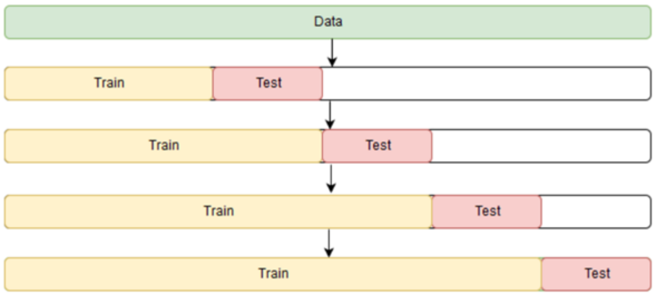

Fig. 4. **Cross-validation on a rolling basis.** We trained our model on consecutive five year time periods and tested it on the following year time period.

2.6 Implementation details

Our code was implemented in python. We used Keras to build all our neural network models and statsmodels for ARIMA and our Dickey-Fuller test. For LIME we used the python LIME library and for result visualizations we used plotnine.

3 Results

3.1 General results

In Figure 5 we compared models based on mean MAPE and SDAPE. We see that ARIMA

TABLE 1
MAPE and RMSE of models.

ALGORITHM	MAPE	SDAPE	RMSE	SDMSE
ARIMA	0.009	0.003	8.02	4.07
CNN	0.024	0.009	17.26	8.64
CNN-LSTM	0.018	0.010	13.96	8.45
LSTM	0.020	0.010	16.12	8.96

scored on average the best accross all datasets. CNN appears to have scored consistently the worst. From Figure 5 and Table 1 we conclude

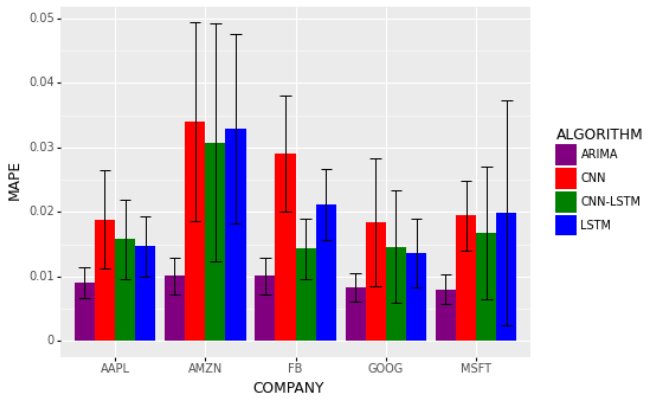

Fig. 5. **MAPE across datasets.** The height of the bars represents mean MAPE with the lines representing SDAPE.

that ARIMA is statistically the best scoring prediction model, while CNN is the worst. We note that our results for CNN-LSTM and LSTM are inconclusive since despite CNN-LSTM scoring on average better than LSTM it did so inconsistently with LSTM being better on AAPL and GOOG. In term of mean stability ARIMA is the most stable model, while the other models have comparable mean stability.

3.2 Hybrid model comparison

We removed the ARIMA predicted values from the datasets and trained our models on the residuals. While this process changes MAPE significantly, it leaves RMSE intact therefore in the following section we will analyze RMSE change in comparison with the original ARIMA. This process also stationarized all five of our time series.

TABLE 2
RMSE of residuals.

	AAPL	AMZN	FB	GOOG	MSFT
ARIMA	0.64	22.11	2.59	13.52	1.26
CNN	0.57	20.27	2.54	11.93	1.11
CNN-LSTM	0.57	20.41	2.63	12.51	1.13
LSTM	0.65	24.48	2.92	14.47	1.30

From Table 6 and Figure 2 we see that the best model in ARIMA residual prediction is CNN, with CNN-LSTM being second, while LSTM consistently increases the residual error. We note that this is not because of incorrect

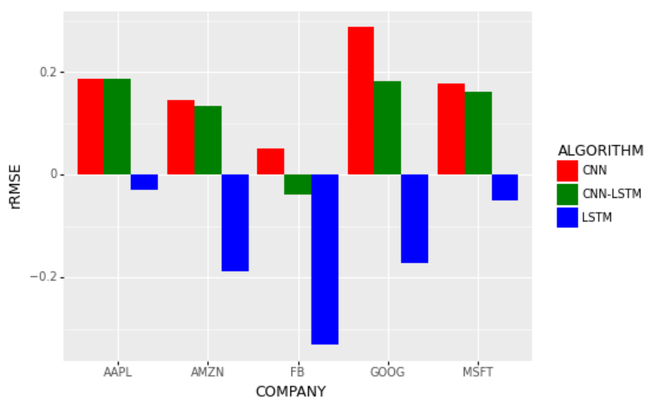

Fig. 6. **RMSE normalized.** The following histogram represents the reduction in RMSE relative to ARIMA.

hyperparameter tunning as the LSTM model displayed extremely erratic prediction patterns and failed to predict certain parts of the model entirely as can be observed from the results in Apendix A.

3.3 Model Explainability

In this section we used LIME to analyze our models and compare the importance they assign to features and to lags as well as how this changes when the models are trained on residuals. As shown in Figure 7 all three mod-

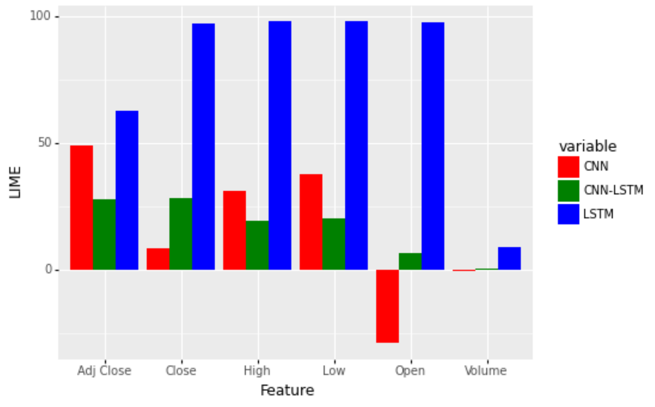

Fig. 7. **LIME of features.** Histogram shows marginal feature importance.

els assign almost no importance to Volume, this is inline with Wang et al [11] who also found volume to be a poor predictor of stock growth. LSTM assign importance nearly uniformly across all other variables, while CNN-LSTM appears to favour Adj Close and Close, this being more inline with our intuition as we would assume that the closing price being closer to the next-day opening price should be a better predictor than other prices. While CNN does appear to assign high importance to Adj Close it assign little to Close and negative to Open, this is definitely unexpected as the Opening price should still be positively correlated with the next day price as such this might be the cause of CNNs poor performance. We note that CNN assigns higher feature importance to Low compared to High, indicating perhaps that the lowest daily stock price is a better predictor of stock value than the highest daily stock price, however this is questionable considering CNNs poor prediction score.

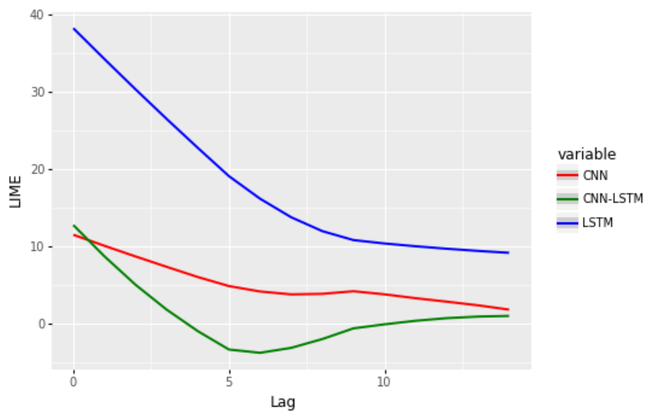

Fig. 8. **LIME of lags.** Smoothed marginal importance of each lag; lag n represent lag at $t-(n+1)$.

All three models appear to assign higher importance to earlier lags and lower importance to later lags, with CNN and CNN-LSTM approaching zero as lags grow above 12. This is inline with our empirical results achieved during parameter tunning, where we found that both CNN and CNN-LSTM performed poorly with bigger lag windows, whereas LSTMs were shown to be much more resilient; considering LSTM scored similarly to CNN-LSTM we conclude that LSTM might have been assigning to much importance to later lags. Another interesting observation is the CNN-LSTM inflection around lag five. It appears that CNN-LSTM assumed that lags between three and eight are anti-correlated with the next day stock value.

In Figure 9 we compare LIME of residual

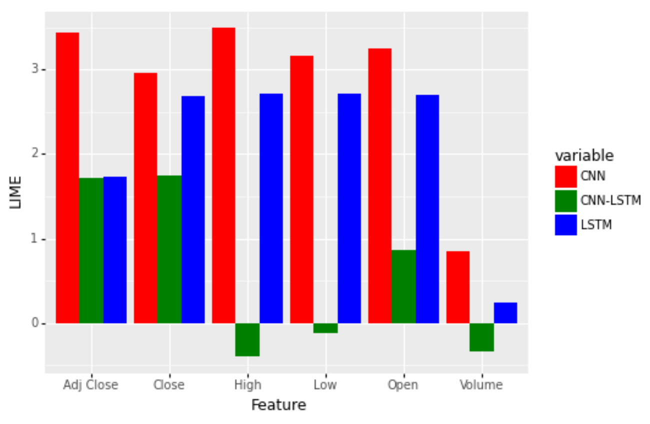

Fig. 9. **Lime of residual features.** Histogram shows marginal feature importance.

features. Despite CNN being the best predictor and LSTM the worst, the overall distribution of importance appears to be similar for LSTM and CNN, with the clear exception that CNN adds far more importance to Adj Close, wheres LSTM appears to distribute feature importance almost uniformly.

CNN-LSTM by distribution completely diverges from LSTM and CNN. It assigns near-zero importance to most features besides Close, Adj Close and Open. It is possible that CNN-LSTM found another important relationship in our data different from the first two, however further analysis is necessary to make a conclusion.

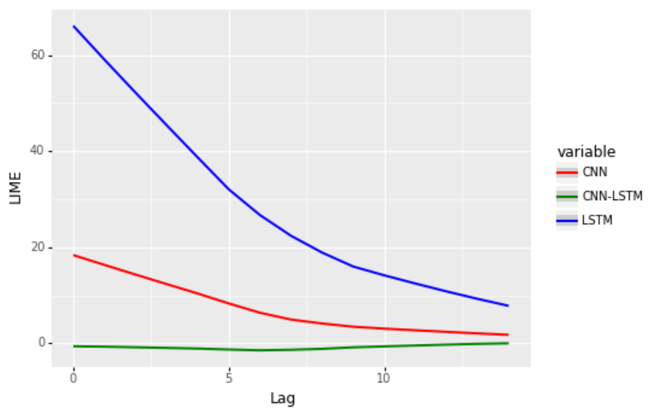

Fig. 10. **LIME of residual lags.** Smoothed marginal importance of each lag; lag n represent lag at $t-(n+1)$.

From Figure 10 we conclude that CNN-LSTM was most likely unable to properly learn from the dataset as its lag importance is close to zero for all lags. Figure 10 also provides an explanation why LSTM underperformed in comparison to CNN; ARIMA removed a significant amount of long term dependencies, considering the uniformly high overall lag importance assigned by LSTM we conclude it improperly modeled the temporal features of our time series. Another noteworthy observation is that both LSTM and CNN have very similair lag graphs for both residuals and the original dataset, while this is expected in the case of CNN it is far less expected in the instance of LSTM.

4 Conclusion

We have on the problem of next-day stock prediction compared LSTM, CNN and CNN-LSTM and found that they consistently underperformed ARIMA both in mean prediction accuracy as well as in prediction stability, which is inline with other similar research [3].

On the problem of residual training CNN was the best predictor. This finding is important as it is not inline with conventional wisdom to use either LSTM or CNN-LSTM to model time series residuals [3]. It also implies that ARIMA is able to remove a significant portion of long term dependencies. All models were analyzed by LIME, which we showed to be a very powerful tool for model analysis, as it was able to explain the poor residual predictions of LSTM as well as discover CNN-LSTMs failure to properly model residuals.

Acknowledgments

The authors would like to thank the professors and assistants of the Faculty of Computer Science for providing us with the knowledge necessary to write this paper.

References

[1] A. Bigiotti and A. Navarra, “Optimizing automated trading systems,” in *The 2018 International Conference on Digital Science*. Springer, 2018, pp. 254–261.
[2] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, “Stock price prediction using lstm, rnn and cnn-sliding window model,” in *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE, 2017, pp. 1643–1647.

[3] L. Ji, Y. Zou, K. He, and B. Zhu, “Carbon futures price forecasting based with arima-cnn-lstm model,” *Procedia Computer Science*, vol. 162, pp. 33–38, 2019.

[4] E. Hoseinzade and S. Haratizadeh, “Cnnpred: Cnn-based stock market prediction using a diverse set of variables,” *Expert Systems with Applications*, vol. 129, pp. 273–285, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419301915

[5] Y. Wang and Y. Guo, “Forecasting method of stock market volatility in time series data based on mixed model of arima and xgboost,” *China Communications*, vol. 17, no. 3, pp. 205–221, 2020.

[6] Q. Tang, R. Shi, T. Fan, Y. Ma, and J. Huang, “Prediction of financial time series based on lstm using wavelet transform and singular spectrum analysis,” *Mathematical Problems in Engineering*, vol. 2021, 2021.

[7] X. Ji, J. Wang, and Z. Yan, “A stock price prediction method based on deep learning technology,” *International Journal of Crowd Science*, 2021.

[8] “U.S. Securities and Exchange Comission,” 2021. [Online]. Available: https://www.sec.gov/edgar.shtml

[9] R. Hyndman, *Forecasting : principles and practice*. Melbourne: OTexts, 2018.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, “”why should i trust you?”: Explaining the predictions of any classifier,” 2016.

[11] Y.-F. Wang, “Mining stock price using fuzzy rough set system,” *Expert Systems with Applications*, vol. 24, no. 1, pp. 13–23, 2003.