Predicting stock prices with LSTM Networks

Devashish Choudhary
choudharydev836@gmail.com

Abstract

In this project we explore a method of predicting stock price movements. We apply a long short term memory (LSTM) based recurrent neural network (RNN) to predict whether the S&P 500 will increase (up) or decrease (down) over the next trading month using the features volatility, return and trading volume as three sets of time sequence data. We use accuracy and the area under the (ROC) curve (AUC) as the metric to tune hyper-parameters and to measure the performance of our prediction. It turns out that the LSTM models perform much like the corresponding baseline linear classification models.

1 Introduction

Artificial Intelligence is changing virtually every aspect of our lives. Today's algorithms accomplish tasks that until recently only expert humans could perform. As it relates to finance, this is an exciting time to adopt a disruptive technology that will transform how investment decisions are made on a broad scale.

Models that explain the returns of individual stocks generally use company and stock characteristics, e.g., the market prices of financial instruments and companies' accounting data. These characteristics can also be used to predict expected stock returns out-of-sample. Most studies use simple linear models to form these predictions [1] or [2]. An increasing body of academic literature documents that more sophisticated tools from the Machine Learning (ML) and Deep Learning (DL) repertoire, which allow for nonlinear predictor interactions, can improve the stock return forecasts [3], [4] or [5].

The main goal of this project is to investigate whether modern DL techniques can be utilized to more efficiently predict the movements of the stock market. Specifically, we train LSTM-networks with time series price-volume data and compare their out-of-sample return predictability with the performance of simple logistic regressions (our baseline models).

2 Related work

The financial industry has been slow to integrate DL techniques into their systems. Moreover, successful financial models are usually proprietary, and it is therefore difficult to find compelling literature with state-of-the-art performance on our specific task. However, the academic community has started to acknowledge the advances in Financial Machine Learning. Consequently, there are increasingly more papers that attempt to apply DL methods to tackle the immensely complex problem of stock market prediction. One common theme amongst these works is the choice of an LSTM based recurrent neural network. For example, Gao applied an LSTM network to forecast the stock markets in six different industries in the US, using 359 stock features as input [6]. Nelson et al generated 175 features for each 15 minutes of stock data using an LSTM network to predict whether the stock price will go up or down [7]. A third study by Chen et al uses historical stock price data to predict whether a stock price will rise, fall or remain in a day, using an LSTM architecture as well [8].

Inspired by the above papers, this project also proposes a model based on an LSTM architecture to predict price movements using dominant predictive signals that include variations on momentum, volatility and liquidity as input. These input features have been shown to have widespread predictive power.

3 Dataset and Features

3.1 Raw Data

The dataset provides daily quotes for the S&P 500 going back to 1980 and is downloaded using the Yahoo Finance API. It comes with the “Open”, “High”, “Low”, “Close”, “Adj Close”, and “Volume” for each day. In total, the raw dataset consists of approximately 10,000 rows and 6 columns.

3.2 Features

The feature set includes rolling (i) logarithmic returns, (ii) trading volumes, and (iii) volatilities for the past $i \in \{1, 2, 3, 4k \mid 1 \leq k \leq 20\}$ trading week(s).

First, daily logarithmic returns can be computed as shown below:

$$(1) \quad \text{Logarithmic}\, returns = \ln \mathrm{B}\frac{\mathrm{CD}}{\mathrm{C_{DEF}}}\mathrm{G}$$

Since logarithmic returns are additive over time, we can simply sum over the corresponding number of daily returns to compute the rolling logarithmic returns for each of the past i trading weeks. This gives us 23 features.

Next, rolling volatilities for each of the past i trading weeks can be calculated as follows:

$$(2) \quad Volatilities = \sigma(daily\ return) * \mathrm{R}\overline{days\ in\ i\ week(s)}$$

This provides us with additional 23 features.

Lastly, we compute the rolling average trading volume for each of the past i trading weeks, which adds another 23 features to our feature set. This results in a total of 69 features.

3.3 Scaling

In the next step, all features are rescaled before training in order to (i) make the contours of the cost function more symmetrical in shape and thus easier to optimize, (ii) make training less sensitive to the scale of features and (iii) ensure regularization does not behave differently for different scaling. Normalization and Standardization are two well-known methods for rescaling data. We will try both techniques and choose the one that works best for our application.

3.3.1 Standardization

Standardization rescales the data to have a mean (μ) of 0 and a standard deviation (σ) of 1 (unit variance):

$$(3) \quad X_{V}^{(W)} = \frac{X^{(Y)} Z X[}{\backslash]}, \ where\ \bar{X} = \frac{-}{-}\sum_{Wfg}^{`} X^{(W)},\ and\ S_{X} = c\sqrt{\frac{-}{-}\sum_{Wfg}^{`} (X - \bar{X})^{e}}$$

3.3.2 Normalization

On the other hand, Normalization scales all numeric variables in the range [0,1]:

$$(4) \quad X_{hij`}^{(W)} = \frac{X^{(Y)} Z X_{kYl}}{X_{kmn} Z X_{kYl}},\ where\ X_{`op} = \max_{W\in\{g,\dots,`\}} (X^{(W)}),\ and\ X_{`Wh} = \min_{W\in\{g,\dots,`\}} (X^{(W)})$$

3.4 Separating Test Data

Since K-Fold Cross Validation fails in finance [9], we instead use walk-forward-cross-validation with a dev size of 10% for our model development (hyper-parameter tuning). Furthermore, to provide an unbiased evaluation of our final model we withhold the last 10% of our data as a separate test set. This set is only used once our model is completely trained (using the train and validation sets).[1]

3.5 Label Distribution

The labels are factors with levels Up and Down indicating whether the market had a positive or negative return on a given month (classification problem). In the training set the distribution is 62%-38% in favor of "Up", and for the test set it is 64%-36% also in favor of "Up".

4 Models

4.1 Baseline Models

To provide linear baselines for reference, simple logistic regressions are implemented. As described in Section 3.4, we use walk-forward-cross-validation for model development and then evaluate the performance on the hold out test set. As part of the optimization, the penalties Ridge, Lasso, Elastic Net and their corresponding hyperparameters, Alpha and Lambda, are tuned. Both grid search algorithms and random search algorithms are used to choose the optimal hyper-parameters. It turns out that the standardized data is best suited for this task.

[1] The details of this procedure can be found in the GitHub Repository

4.2 LSTM Networks

Given the recent success of RNNs in the area of sequential data and taking into account the temporal sequence of our data, we apply such an architecture to our prediction task. Specifically, we use an LSTM network, which is a special type of a recurring neural network that works much better than the standard version for many tasks. One of the appeals of LSTM`s is the idea that they are able to learn long-term dependencies. Our network structure is depicted in the following figure.

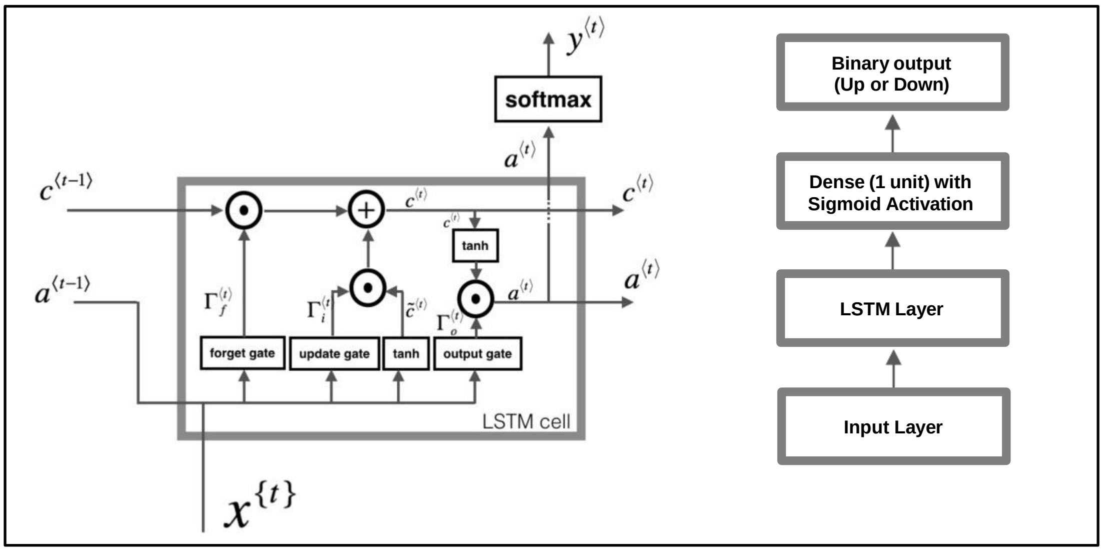
$y^{\langle t \rangle}$
softmax
$a^{\langle t \rangle}$
$c^{\langle t-1 \rangle}$
$c^{\langle t \rangle}$
$a^{\langle t-1 \rangle}$
$a^{\langle t \rangle}$
tanh
$\Gamma_f^{\langle t \rangle}$
$\Gamma_i^{\langle t \rangle}$
$\tilde{c}^{\langle t \rangle}$
$\Gamma_o^{\langle t \rangle}$
forget gate
update gate
tanh
output gate
LSTM cell
$x^{\{t\}}$
Binary output (Up or Down)
Dense (1 unit) with Sigmoid Activation
LSTM Layer
Input Layer

Figure 1: LSTM unit and Network Structure

The key to LSTM`s is the cell state, the horizontal line that runs through the top of the diagram. With only a few linear interactions, it is easy for information to simply flow through. However, the LSTM has the ability to remove or add information to the cell state, carefully controlled by three gates:

The first gate decides which information is discarded from the cell state. This decision is made by a sigmoid layer called the "*forget gate layer*". It considers $a^{\mathrm{stZ\_u}}$ and x^{stu} and outputs a number between 0 (forget) and 1 (keep) for each number in the cell state $c^{\mathrm{stZ\_u}}$. To update the cell state, we multiply $c^{\mathrm{stZ\_u}}$ by $\Gamma_{y}^{\mathrm{stu}}$.

The second gate decides which new information will be stored in the cell state. This decision is made by a sigmoid layer called the "*update gate layer*". It considers a vector of new candidate values $\tilde{c}^{\mathrm{stu}}$ created by a tanh layer and outputs a number between 0 (update) and 1 (no update) for each candidate value. To update the cell state we add $\tilde{c}^{\mathrm{stu}} * \Gamma^{\mathrm{stu}}$ to $c^{\mathrm{stZ\_u}}$.

Finally, the third gate decides which information will be outputted. This decision is made by a sigmoid layer called the "*output gate layer*". It considers the filtered (first step) and updated (second step) cell state c^{stu} which is put through a tanh to push the values to be between -1 and 1 and outputs a number between 0 (no output) and 1 (output). To compute the output a^{stu} we multiply $\Gamma_{\{}^{\mathrm{stu}}$ by $\tanh(c^{\mathrm{stu}})$[2] [10].

[2] In theory, LSTM models are meant to consider the entire sequence for our prediction task. In practice, however, we need to create batches to train a model with a backprogation algorithm. Since the gradients cannot backpropagate between batches, the LSTM model only remembers what happened within a batch. Thus, at the beginning of each batch the states are initialized and set to 0 (therefore not taking into account previous information). To circumvent this problem during training, Keras introduces a stateful flag that reminds the model of what happened in the previous batch by passing states from the previous batch to the next batch. A stateful LSTM model also makes a lot of sense during the prediction phase, since otherwise the LSTM model trained with batch time series of length T assumes that the hidden states are initialized to zero at every T steps.

4.3 Experiment Design

As aforementioned, we use walk-forward-cross-validation for model development and then evaluate the performance on the hold out test set. Due to the nature of our features (returns, volatilities, and trading volumes of the past week(s), we apply a stateful LSTM architecture with a batch size and a time step of 1 to our prediction problem. As part of the optimization, we search through different hyperparameters for each model, including dropout rate and the number of hidden units. Both grid search algorithms and random search algorithms are used to choose the optimal hyper-parameters.

4.4 Experiment Results

We have three sets of time sequence data: (1) volatility, (2) return and (3) trading volume. By combining these feature sets, 7 models are created and compared with their corresponding baseline model. As above mentioned, we use accuracy and AUC as our evaluation metrics. The following table summarizes the results for the various model contractions (both the confusion matrices and the ROC curves for all models can be found on Github):

Model	**Description**	**Dev/Test Acc. LSTM**	**Dev/Test Acc. Baseline**	**Test AUC LSTM**	**Test AUC Baseline**
1	Feature Set 1	61.02% / 64.38%	54.48% / 65.01%	0.52	0.52
2	Feature Set 2	55.62% / 65.01%	55.44% / 66.07%	0.52	0.54
3	Feature Set 3	56.49% / 65.12%	55.21% / 63.95%	0.54	0.48
4	Feature Sets 1 + 2	54.42% / 63.32%	52.06% / 60.04%	0.52	0.54
5	Feature Sets 1 + 3	59.34% / 65.96%	55.52% / 59.09%	0.46	0.53
6	Feature Sets 2 + 3	49.56% / 63.85%	54.19% / 65.54%	0.55	0.55
7	Feature Sets 1 + 2 + 3	53.13% / 65.54%	52.28% / 66.17%	0.53	0.50

Table 1 Summary of model results

From a trading perspective, we are interested in maximizing accuracy, as this tells us how well our model performs on both Up`s and Down`s. However, as described in Section 3.5, there is a significant imbalance both in the training (62% "Up`s") and the test data (64% Up`s). Therefore, we must take accuracy with a grain of salt and the AUC score is probably a better choice for our model performance as it is insensitive to class imbalance. As shown in table 1, all 7 models have a test AUC score close to 0.5, which is an unsatisfactory result. One possible reason is that our features do not contain enough information to predict future returns. Moreover, our input data is very noisy, and it is difficult for the classifier to generalize to other periods due to the non-stationarity of our data.

5. Conclusion and Future Work

From this project, we can conclude that stock price forecasting remains a difficult task even after simplifying the problem to a binary price trend and applying powerful deep learning models. Nonetheless, this information is useful in guiding future work, specifically in determining relevant feature sets and model architectures. We have shown that it might need more than Price/Volume/Volatility data in order to predict future returns. Potential future direction for this project may include:

- Enriching our dataset. We should include non-technical features such as fundamental companies' accounting data
- Changing our prediction objective. We could use a regression model instead of a classification model. Rather than prediction "Up" or "Down", we could sort our predictions in certain quantiles and try to predict bucket probability.
- Exploring new models. We should experiment with different models such as attention
- Experimenting with different time horizons. We can explore a variety of different window sizes (many high-frequency traders consume data in milliseconds)