CS230

Using Deep Learning to Predict the Stock Market

Evan Rosenman
Department of Statistics, Stanford University
rosenman@stanford.edu

Abstract

We consider the problem of using deep learning to predict 10-day market-residualized returns for stocks. From a Kaggle competition, we source performance data on 3,800 assets over a six-year span; we also have data on relevant news articles for 30% of the asset-date pairs. We leverage a scoring scheme provided by the Kaggle competition, which prioritizes stability of performance across time. We explore loss functions whose minimization best maximizes this scoring function, finding that cross-entropy performs best. We then explore feed-forward and recurrent neural network architectures. Our best performer is two-layer feed-forward neural net leveraging the market and news data, though sequence models show much promise.

1 Introduction

It is estimated that 85% of U.S. stock market trades are performed by algorithms (Nelson et al., 2017). Among investors, there are substantial financial incentives to improve these algorithms such that they can better identify assets most likely to increase in value. The proliferation of financial news yields enormous potential predictive value (Soon, 2010) if it can be appropriately digested and featurized.

The New York-based hedge fund Two Sigma, recognizing the value of incorporating these data, has posted a Kaggle challenge asking contestants to "use the content of news analytics to predict stock price performance" (Kaggle, 2018). We use the data and scoring function provided for this challenge, but modify the definition of the training and test set. We also explore algorithms that both do and do not include features derived from contemporaneous news data.

Our problem is formulated as follows: for each asset-day pair of interest, the input to our algorithm is a set of covariates related to the stock's performance (prior returns, trading volume, etc.) and, optionally, a set of additional covariates summarizing news content related to the stock on the given day (sentiment, length, etc.). Our goal is then to predict a value in $[-1, 1]$ summarizing our confidence in the direction of the subsequent ten-day market-residualized return on the asset. A value near 1 signifies high confidence in a positive market-residualized return and a value near -1 signifies high confidence in a negative market-residualized return. We use a variety of neural network architectures to accomplish this task.

2 Related work

The problem of predicting stock market movements has substantial precedent in the machine learning literature. Work from the early 2000s (Huang et al., 2005; Tay and Cao, 2001; Kim, 2003) made heavy use of support vector machines. This work frequently relied on careful featurization of market data. For example, Tay and Cao (2001) provided lagged relative difference in price values based on a

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

five-day window. Tay's approach outperformed a neural network, but lacked the capacity to scale to wide datasets with less structured features.

More modern approaches have relied on neural networks. Jabin (2014) achieved high accuracy predicting the Indian stock market using a very simple feed-forward network providing the prior four days' returns as separate features. More recently, researchers have more explicitly made use of the time-series nature of stock returns by fitting sequence models. Chen et al. (2015) constructed a dataset of daily features like prior closing price and trade volume. They use an LSTM framework to model the sequential nature of the data, and found that such an approach improved predictions on the Chinese stock market. Similar approaches were found to be promising for predicting returns in the Brazilian (Nelson et al., 2017) and U.S. (Di Persio and Honchar, 2016) stock market.

A final series of papers seeks to incorporate data outside the market for the purpose of enhancing predictions. If properly fit and trained, such models would be expected to outperform because when "significant events (e.g., a bankruptcy of a global financial firm) [happen] such events would first appear in the news, not in the past numerical information (e.g., last five days' stock prices)" (Yoshihara et al., 2014).

Schumaker and Chen (2009) analyzed the effect of breaking news on extremely narrow stock price windows, seeking to examine the effect twenty minutes after the news broke. They were able to get good performance using simple bag-of-words and rules-based approaches to convert text data into features, but made use of a SVM, rather than a deep net, for prediction. Siah and Myers (2015) fit a GRU to the Nikkei returns, and instead featurized articles solely based on their sentiment scores.

Yoshihara et al. (2014) take a somewhat more modern approach, fitting an RNN to Japanese stock price data while making use of relevant news articles on each day. They convert the articles into one-hot feature vectors for the presence and absence of each word, and then use these features alongside other market features as inputs to the RNN.

3 Dataset and Features

The Two Sigma Kaggle competition provides two distinct datasets for training. The first is a market returns database which contains financial market information for 3,800 assets over the period from 2007 to 2016. The dataset contains 4.1 million rows and 16 columns. Each row in this database is uniquely identified by an asset-date pair. Among the columns are 11 market-based predictors, such as stock price and trading volume on the date, and market-residualized returns over prior 10 days. The data also contains market-residualized returns over the subsequent 10 days, which we seek to predict.

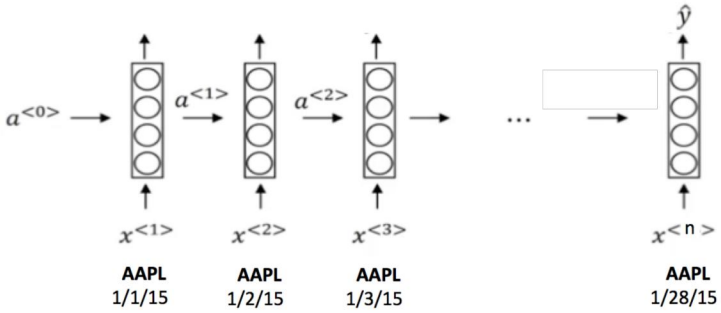
The second dataset is a news database, sourced from Thomson Reuters. Each row is a unique news article. The articles are tagged with the relevant assets (e.g., those that are discussed in the article). Each row also contains metadata about the article, including scores for sentiment, news novelty, and relevance to the asset. The database contains 9.3 million rows and 35 columns.

Data was pre-processed in a few key ways. The final two years of the dataset (2015 and 2016) were chosen as our test set, with the preceding years initially chosen as the training set. Due to CPU and RAM limitations on Kaggle kernels, reduction in the size of the training set was necessary to achieve stability. Initial experiments demonstrated that removing the first few years of the training set actually improved test error. This is hypothesized to be due to the fact that the financial crisis induced a dramatic dissimilarity between the training and test sets. This resulted in a final training set containing 1.9 million rows and a final test set containing 900,000 rows.

In experiments including the news predictors, only the numeric covariates were utilized. Features were derived by joining the datasets by asset and date (i.e. all news articles mentioning an asset on a given date were joined to the relevant asset-date pair) and aggregating over the articles by computing the mean, minimum, maximum, and standard deviation of the metrics. This yielded a dataset with 94 total predictors. Approximately 30% of rows had at least one relevant news article.

The training predictors and test predictors were centered. Null values were then filled with zero (the equivalent of mean-filling prior to normalization). Lastly, the data was scaled to have standard deviation one. For most of the models, the label we sought to predict was actually the sign of the market-residualized returns (described in the next section), so these values were also generated in the pre-processing stage.

Figure 1: Visual representation of how the data is turned into a length n sequence for a sequence model. We suppose the stock is Apple and the window starts on January 1, 2015



Code from public kernels contributed by Bruno G. do Amaral (do Amaral, 2018) and "Dieter" Dieter (2018) was heavily utilized in writing the pre-processing code.

4 Methods

4.1 Models

We explored three broad classes of models. The first was a simple logistic regression model using all of our features. This provided a baseline for performance.

The second class of models were feed-forward neural networks. I explored fully connected networks with either one or two hidden layers. Denote as x_i the covariate vector for a single example i in the training set (where either $x_i \in \mathbb{R}^{11}$ for the market data only, or $x_i \in \mathbb{R}^{94}$ for the news and market data). For a single hidden layer neural network, the relevant equations are:

$$\begin{aligned} a_i^{[1]} &= \sigma\left(W^{[1]}x_i + b^{[1]}\right) \\ \hat{y}_i &= \sigma\left(W^{[2]}a_i^{[1]} + b^{[2]}\right) \end{aligned} \tag{1}$$

where $\sigma(\cdot)$ is our activation function (chosen to be the sigmoid), $a_i^{[1]}$ is the value of the hidden layer for example i, $\hat{y}_i$ is the prediction for example i, and $W^{[1]}$, $W^{[2]}$, $b^{[1]}$, and $b^{[2]}$ are all trainable parameters. Parameters were trained via back-propagation. These models are able to flexibly incorporate a large amount of information, but they do not explicitly incorporate the sequential nature of the data.

The third class of models – sequence models – do incorporate this structure. I specifically explored Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), and Long Short-Term Memory Units (LSTMs). In these models, we take a sliding window of consecutive data points for each asset and treat this as a single training example. A schematic is provided in Figure 1, supposing Apple ("AAPL") is the relevant stock (stock i) and the window is of length n, starting on January 1, 2015. We denote the vector corresponding to Apple's covariates on the m^{th} day of the sequence as $x_i^{<m>}$.

The final prediction $\hat{y}$ corresponds to the label associated with the final date of the window. In our models, we generate this prediction by taking the final hidden state of the recurrent model and providing it to one more feed-forward layer. The equations are precisely those provided in Equations 1, with the final hidden state $a_i^{<n>}$ replacing the covariate vector x_i.

4.2 Losses

To make sure my results were relevant to the problem at hand, I opted to use the scoring function provided by the Kaggle competition. Recall that the user generates a prediction $\hat{y}_{ti} \in [-1, 1]$ for asset i at time t, representing confidence in the direction of the market-residualized return of the asset over the subsequent 10 days. For r_{ti} the true residualized return and u_{ti} a binary variable representing whether that particular asset-day pair is included in evaluation, the scoring proceeds as follows:

- For each day, compute
$$p_t = \sum_i \hat{y}_{ti} r_{ti} u_{ti}$$
- Across days, compute:
$$\text{score} = \frac{\bar{p}_t}{std(p_t)} \tag{2}$$
where $std(\cdot)$ is the standard deviation across days.

The denominator penalizes highly variant p_t values. This is a sensible strategy in that a hugely negative p_t value could bankrupt a firm before it realized future gains from higher subsequent p_t values. However, this poses a challenge as the score does not decompose into components corresponding to each example x_i. The expectation of the gradient with respect to a randomly selected sample is also not equal to the true gradient. So methods like stochastic or minibatch gradient ascent cannot be implemented in this context. Thus, we explore the minimization of several loss functions to serve as a proxy for maximizing this scoring scheme.

5 Results

5.1 Choosing a Proxy Loss

Per the discussion above, the choice of a "proxy loss function" was treated as a hyperparameter. For clarity, we will distinguish between the final confidence value $\hat{y}_{ti}$ and the value outputted by the network, which we denote v_{ti}. I chose a simple baseline model – a one-layer neural net with 32 hidden nodes, using market data only as the predictors – and trained it using each of the following loss functions:

- cross-entropy loss. The final layer of the neural net was modified to output a value $\hat{v}_{ti} \in [0, 1]$ and the loss was computed as $-\sum_{i,t} \mathrm{I}(r_{ti} > 0) \log(\hat{v}_{ti}) + (1 - \mathrm{I}(r_{ti} > 0)) \log(1 - \hat{v}_{ti})$ where $\mathrm{I}(\cdot)$ is an indicator function and the sum runs over days and assets in the training data.
- hinge loss. In this case, the final layer of the neural net was modified to output a value $\hat{v}_{ti} \in [-1, 1]$ and the loss was computed as: $\sum_{i,t} \max(0, 1 - I(r_{ti} > 0)\hat{v}_{ti})$
- squared error loss. In this case, the final layer of the neural net was modified to output a value $\hat{v}_{ti} \in \mathbb{R}$ and the loss was computed as: $\sum_{i,t} (\hat{v}_{ti} - r_{ti})^2$
- "score numerator" loss. In this case, the final layer of the neural net was modified to output a value $\hat{v}_{ti} \in [-1, 1]$ and the loss was computed as: $-\sum_{i,t} \hat{v}_{ti} r_{ti}$
  Note that this is the negative of the numerator in the score function

Training was run for 25 epochs with an Adam Optimizer and a learning rate of 0.001. Models were then evaluated on the set (with predictions being converted to the range $[-1, 1]$ either via clipping or by a shift and dilation) and the score was calculated according to 2. Results are provided in Table 1

Table 1: Scoring function values on the test set

Cross-Entropy	**Hinge**	**Squared Error**	**Score Numerator**
0.436	0.375	0.250	0.395

As cross-entropy performed the best as a proxy loss, it was used for the model selection phase. Note that this means our final learning algorithm seeks to predict the probability that the market residualized return r_{ti} is greater than 0. This final probability $\hat{v}_{ti}$ can easily be converted into confidence values $\hat{y}_{ti} \in [-1, 1]$ via the transformation:
$$\hat{y}_{ti} = 2 \cdot (\hat{v}_{ti} - 0.5)$$

5.2 Choosing a Model

All models were trained with an Adam Optimizer used to minimize the cross-entropy loss function. Mini-batches contained 64 data points apiece. Twenty-five training epochs were used, and training cross-entropy losses were manually checked for approximate convergence.

Some modest tuning was done on the number of hidden nodes (between 32, 64, and 128), the learning rate (between 0.01 and 0.001, the sequence length for the sequence models (between 10 and 50), and the dropout parameter (between none, 0.2 and 0.5). A learning rate of 0.001 was found to work best in all cases, and length 50 sequences always outperformed length 10 sequences. For brevity, we provide below only a selection of the best-performing models of each type. I acknowledge that the reported test scores may be biased upward because the tuning was done directly on the test set rather than on a development set.

Results can be found in Table 2. The best performing model, when measured by the score on the test set, is a deep neural network with two hidden layers, making use of the market and news features. With proper regularization, this model appears capable of gaining additional insights from the news content.

Two sequence models, trained using market features only, are close behind in test set performance. It's notable that GRUs and LSTMs do appear to be learning additional structure from the time series representation of the data. Their dramatic outperformance of the basic RNN makes sense, as we find that the longer (length 50) sequences yield better insights. Notably, these models also achieve much better training scores than the feed-forward network. It's plausible that more aggressive regularization schemes could thus boost their test scores to outperform a feed-forward network.

Note that instability and memory limitations on Kaggle kernels prevented training a sequence model making use of the news features, but these would also be expected to outperform.

We note also that the 1-hidden-layer feed-forward neural net wasn't able to achieve a performance gain from the addition of the news features. Though several hyperparameter values were tried, it would require more careful tuning to actually make use of these features effectively in a simple feed-forward neural network. The relative under-population of these features may be a culprit.

Lastly, note that all of these methods outperform a simple logistic regression, indicating that deep learning approaches are able to identify more meaningful structure in the data.

Table 2: Summary and performance measures of best-performing models of each type

Model	**Features**	**Num Hidden Nodes**	**Dropout**	**Sequence Length**	**Training Score**	**Test Score**
2-hidden-layer feed-forward neural net	All	128/16	0.2	N/A	0.594	*0.507*
GRU	Market features only	32	0.2	50	*1.840*	0.488
LSTM	Market features only	32	0.2	50	1.770	0.483
1-hidden-layer feed-forward neural net	Market features only	32	None	50	0.675	0.482
RNN	Market features only	32	None	50	1.086	0.448
1-hidden-layer feed-forward neural net	All	32	None	N/A	0.745	0.430
Logistic regression	All	N/A	N/A	N/A	0.383	0.368

6 Future Work

Our best performing algorithm was a feed-forward neural network, but the results strongly indicate that a properly designed sequence model, incorporating the news features, would outperform. The stability of Kaggle kernels was a major hindrance throughout this project and prevented both inclusion of the news features in a sequence model and extensive exploration of regularization methods for the sequence models. With better computing resources, this is an obvious next direction.

The under-population of the news features also appeared to pose a challenge to proper model training. With more time, I'd like to explore an ensemble-type approach with separate models fit to stock-date pairs that do or do not have relevant contemporaneous news stories. I'd also be interested to directly leverage the headlines of the articles by making use of the words' associated GloVe vectors.

Code

I have made all my kernels public and they can be accessed at `https://www.kaggle.com/rosenman/kernels`.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Chen, K., Zhou, Y., and Dai, F. (2015). A lstm-based method for stock returns prediction: A case study of china stock market. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2823–2824. IEEE.

Di Persio, L. and Honchar, O. (2016). Artificial neural networks approach to the forecast of stock market price movements. *International Journal of Economics and Management Systems*, 1.

Dieter (2018). Market data nn baseline.

do Amaral, B. G. (2018). A simple model - using the market and news data.

Huang, W., Nakamori, Y., and Wang, S.-Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, 32(10):2513–2522.

Jabin, S. (2014). Stock market prediction using feed-forward artificial neural network. *growth*, 99(9).

Kaggle (2018). Two sigma: Using news to predict stock movements.

Kim, K.-j. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319.

Nelson, D. M., Pereira, A. C., and de Oliveira, R. A. (2017). Stock market's price movement prediction with lstm neural networks. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1419–1426. IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Schumaker, R. P. and Chen, H. (2009). Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems (TOIS)*, 27(2):12.

Siah, K. W. and Myers, P. (2015). Stock market prediction through technical and public sentiment analysis.

Soon, Y. C. (2010). News which moves the market: Assessing the impact of published financial news on the stock market.

Tay, F. E. and Cao, L. (2001). Application of support vector machines in financial time series forecasting. *omega*, 29(4):309–317.

Yoshihara, A., Fujikawa, K., Seki, K., and Uehara, K. (2014). Predicting stock market trends by recurrent deep neural networks. In *Pacific rim international conference on artificial intelligence*, pages 759–769. Springer.