Predicting Stock Performance with Natural Language Deep Learning

December 4, 2017   👁 31,089

Overview

We recently worked with a financial services partner to develop a model to predict the future stock market performance of public companies in categories where they invest. The goal was to use select text narrative sections from publicly available earnings release documents to predict and alert their analysts to investment opportunities and risks. We developed a deep learning model using a one-dimensional convolutional neural network (https://en.wikipedia.org/wiki/Convolutional_neural_network) (a 1D CNN) based on text extracted from public financial statements from these companies to make these predictions. We used Azure Machine Learning Workbench (http://azureml.azureedge.net/content/apphome/index.html) to explore the data and develop the model. We modeled our solution using the

Keras (https://keras.io/) deep learning Python framework with a Theano (http://www.deeplearning.net/software/theano/) backend. Our results demonstrate how a deep learning model trained on text in earnings releases and other sources could provide a valuable signal to an investment decision maker.

The Challenge

When reviewing investment decisions, a firm needs to utilize all possible information, starting with publicly available documents like 10-K (https://en.wikipedia.org/wiki/Form_10-K) reports. However, reviewing public earnings release documents is time-intensive and the resulting analysis can be subjective. Moreover, the written sections of an earnings release require the most review time and are often the most subjective to interpretation. A thorough analysis of the investment opportunity of a business would also include a review of other companies in the industry to understand relative performance. Our challenge was to build a predictive model that could do a preliminary review of these documents more consistently and economically, allowing investment analysts to focus their follow-up analysis time more efficiently and resulting in better investment decisions.

For this project, we sought to prototype a predictive model to render consistent judgments on a company's future prospects, based on the written textual sections of public earnings releases extracted from 10k releases and actual stock market performance. We leveraged natural language processing (NLP) pre-processing and deep learning against this source text. In the end, we sought a model that was easy to operationalize, use and maintain over time. While there are broader potential applications of processing public earnings release narratives to predict future stock value, for the purposes of this project we focused just on generating predictions that could better inform further human analysis by our partner.

Our Approach

Tooling, Pre-Processing and Initial NLP Exploration

We began our work in Python with Azure Machine Learning Workbench (https://docs.microsoft.com/en-us/azure/machine-learning/preview/), exploring our data with the aid of the integrated Jupyter Notebook. We used the base AML Workbench Python libraries, including NLTK, (http://www.nltk.org/book/) and added some additional packages and NLP tools including the Gensim library. (https://radimrehurek.com/gensim/) All scripts and sample data are available in this GitHub repo (https://github.com/SingingData/StockPerformanceClassification), including Jupyter Notebooks (https://github.com/SingingData/StockPerformanceClassification/tree/master/JupyterNotebooks)for each of the steps, from filtering source data to pre-processing, running and evaluating the model. To make things easier, you'll find a list of the Python packages and utilities to install on top of the base Azure Machine Learning Workbench Python installation listed in the readme (https://github.com/SingingData/StockPerformanceClassification/blob/master/README.md).

As input, we gathered a text corpus of two years of earnings release information for thousands of public companies worldwide. We extracted as source the sections 1, 1A, 7 and 7A (https://en.wikipedia.org/wiki/Form_10-K)from each company's 10k — the business discussion, management overview, and disclosure of risks and market risks. We also gathered the stock price of each of the companies on the day of the earnings release and the stock price four weeks later. We categorized the public companies by industry category.

We pre-processed the text, converting to UTF-8, removing punctuation, stop words (https://en.wikipedia.org/wiki/Stop_words), and any character strings less than 2 characters. The pre-processing Jupyter Notebooks are on GitHub (Source Text Filtering (https://github.com/SingingData/StockPerformanceClassification/blob/master/JupyterNotebooks/Step%201%20-%20Text%20Preprocessing%20Filter%20Source%20Data%20-%203%20Class%20Model.ipynb) and Text Cleaning (https://github.com/SingingData/StockPerformanceClassification/blob/master/JupyterNotebooks/Step%202%20-%20Text%20Preprocessing%20Clean%20Text%20-%20for%203%20Class%20Model.ipynb)). Many of the techniques we used are described in detail in the NLTK in Python (http://www.nltk.org/book/) book. Below is an example of cleaned text, which in this case is a sample of a management overview from one earnings release. A number of text document samples (https://github.com/SingingData/StockPerformanceClassification/blob/master/Data/sampletext.txt) are available on GitHub.

```
1  business overview acad overview we are biopharmaceutical company focused discovery development commercialization small molecule drugs treatm
   ent central nervous system disorders we currently have six clinical programs several additional programs discovery development our most adva
   nced program we are conducting phase iii studies pimavanserin treatment parkinson disease psychosis we have reported positive results phase
   ii trial our program pimavanserin co therapy schizophrenia we also have completed enrollment phase iib trial our program acp stand alone tre
   atment schizophrenia addition we have completed proof concept clinical study pimavanserin treatment sleep maintenance insomnia healthy older
    adults we have retained worldwide
```

An important consideration in our approach was our limited data sample of less than 35,000 individual text document samples across industries, with much smaller sample sizes within an industry. Within biotechnology, we had 943 text document samples. As a result of the sample limitations, our project results should be viewed as simply a proof of concept to be validated and improved with additional samples.

Another factor was the large amounts of industry-specific vocabularies contained in each of the text documents. To better understand the variation within the corpus, we cleaned the text the help of NLP methods and libraries including NLTK (http://www.nltk.org/) and Gensim (https://radimrehurek.com/gensim/). When inspecting the source text from public company releases with an LDA (https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation) topic model analysis, we found that there was a large amount of vocabulary variation between industry vocabularies, and much less variability within industries. This finding led us to prototype our performance classification model based on single industries, rather than across them, in order to reduce the amount of less meaningful variation noise.

Representing Documents as Word Vectors

In order to take advantage of NLP deep learning, we needed to obtain numerical representation for our text. Specifically, we needed vector representations for each of our documents. Natural language, by its nature, has localized spatial correlations between words. For example, if you find the word 'sunny', you may be more likely to find the word 'weather' in the same sentence than another less closely related word. Word vector models represent these relationships numerically. In particular, word embedding (https://en.wikipedia.org/wiki/Word_embedding) is a technique wherein word pairs can be represented based on the Euclidian distance between them which can encode the semantic differences and similarities. These distances can be represented by vector differences.

There were two options for creating word embeddings. We could create custom embeddings based on our corpus of source texts, or we could leverage a pre-trained model based on a much larger corpus of text. Given the limited size of our sample, we looked to leveraged pre-trained word vectors. In our case, we used GloVe pre-trained models (https://nlp.stanford.edu/projects/glove/). These pre-trained models were trained on aggregate global word-word co-occurrence from a variety of very large datasets. The result is a vector that represents the linear substructure of the word vector space. We used the GloVe pre-trained model of all of Wikipedia's 2014 data (http://nlp.stanford.edu/data/glove.6B.zip), a six billion token, 400,000-word vocabulary vector model, chosen for its broad domain coverage and less colloquial nature. This pre-trained set of word vectors allowed us to vectorize our document set and prepare it for deep learning toolkits.
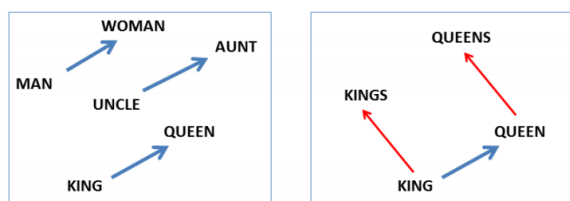
(Mikolov et al., NAACL HLT, 2013)

Word Vector Example

Although this pre-trained model has a vast 400,000-word vocabulary, it still has limitations as it relates to our text corpus. For example, in technology-driven industries, there is a highly specialized, domain-specific vocabulary which may not be represented in the pre-trained word model. These vocabulary terms might be predictive of performance, but when we used these pre-trained word models, out-of-vocabulary words would all get the same word vector values which reduce their predictive value. For some industries, this vocabulary changes over time as new technologies, compounds or products are developed. As a result, the word vector of these changing words might need to be different at different periods of time. The presence of the newest technology vocabulary might also have predictive value. In addition, the corporate earnings release statements are rendered with a particular subtle patois (https://en.wikipedia.org/wiki/Patois) not fully reflected in the Glove model pre-trained on Wikipedia articles.
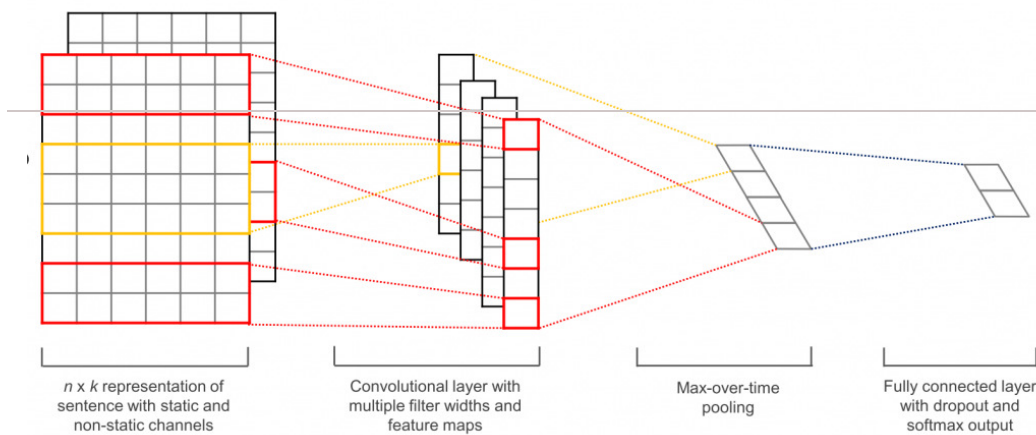
Research is emerging on new methods for dealing with out of vocabulary words for small vocabularies, and the temporal dimension of vocabulary words. This post (http://ruder.io/word-embeddings-2017/index.html#fnref:15) sums up important recent NLP research which promises to solve these issues in the future.

Classification Labels and Prediction Goal

We created three equally sized classification bins of high, medium and low performance based on the performance of the stock between the date of the release and four weeks later. The performance was calculated as the percentage change in the stock value in that time and applied some normalization for overall stock market changes. These low, medium and high 4-week performance classifications were the labels in our model. We modeled our prototype on just one industry, the biotechnology industry, which had the most abundant within-industry sample. Our project goal was to discern whether we could outperform chance accuracy of 33.33%.

Convolutional Neural Net Model with Keras

With our documents represented by a series of embeddings, we were able to take advantage of a convolutional neural network (https://en.wikipedia.org/wiki/Convolutional_neural_network) (CNN) model to learn the classifications. CNNs can be well suited to document modeling, as they can find small (and then large) syntactic structures across the training set through convolutional and max pooling steps, building a fuller model of the source corpus (read more about CNNs with NLP (http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/)). Below is a depiction of a one layer CNN. With our limited sample of source documents and very limited timespan of our data points, we chose the simpler 1D CNN, rather than using an LSTM (https://en.wikipedia.org/wiki/Long_short-term_memory) model for this project.

n x k representation of sentence with static and non-static channels	Convolutional layer with multiple filter widths and feature maps	Max-over-time pooling	Fully connected layer with dropout and softmax output

We used a 1D CNN in Keras using our custom word embeddings. See this excellent Keras example (https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html) for a 1D CNN architecture using custom word embeddings, like those pre-trained Glove model word vectors. In our model design, we started from the Keras reference as our architectural base and refined from there.

We chose a 10,000-word sequence as the maximum. For those documents with more than 10,000 words, we truncated the remaining text. For those documents with fewer than 10,000 words, we padded the sequence at the end with zeroes. We mapped each of our words in each sequence to one of the Glove embedding vocabulary items and used its 300 value numerical representation. For each document sample, we had a 10,000 x 300 sequence representation. Below is an excerpt of building the embedding matrix from this script (https://github.com/SingingData/StockPerformanceClassification/blob/master/JupyterNotebooks/Step%203%20-%20Run%201DCNN%20Stock%20Text%20Classification%20to%203%20Classes.ipynb).

```
1  ###############################
2  #Build the embedding matrix
3  ###############################
4
5  print('Building Embedding Matrix...')
6  embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
7  for word, i in word_index.items():
8      embedding_vector = embeddings_index.get(word)
9      if embedding_vector is not None:
10         # words not found in embedding index will be all-zeros.
11         embedding_matrix[i] = embedding_vector
12
13 embedding_layer = Embedding(len(word_index) + 1,
14                             EMBEDDING_DIM,
15                             weights=[embedding_matrix],
16                             input_length=MAX_SEQUENCE_LENGTH,
17                             trainable=False)
```

For the model itself, we employed the ADAM optimizer (https://keras.io/optimizers/), the Lecun (http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf) initializer (https://keras.io/initializers/), and we used exponential linear unit ('elu') (https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) activation (https://keras.io/activations/) function. We applied dropout in training (15% to inner layers, and 45% to the final layer), and the Keras early stopping (https://keras.io/callbacks/) feature to prevent over-fitting. Also, we stepped down the learning rate from the initial model to improve the test results to .00011. We discovered the model was very sensitive to initializer choices, with the Lecun model offering much better learning than other all other initializers available in Keras (https://keras.io/initializers/). After testing all the optimizer options in Keras (https://keras.io/optimizers/), we found that both ADAM and RMSprop optimizers performed much better than other optimizers, with ADAM performing slightly better. When we used the 'elu' function, the model trained less erratically than with the Relu, Prelu or Leaky Relu (https://en.wikipedia.org/wiki/Rectifier_(neural_networks))activation functions, and reached higher accuracy. We stepped down batch size to a modest size of 33 to improve learning.

In order to improve the model, we augmented the data in the original text with the title of the section from the 10-K report. We appended this text to the start of the document sample. Thesaurus-based data augmentation in NLP is discussed in more depth in this forum discussion (http://forums.fast.ai/t/data-augmentation-for-nlp/229).

Some of the hyperparameter choices and the model architecture are detailed below. Also, see the complete Jupyter Notebook (https://github.com/SingingData/StockPerformanceClassification/blob/master/JupyterNotebooks/Step%203%20-%20Run%201DCNN%20Stock%20Text%20Classification%20to%203%20Classes.ipynb) and this practical guide to troubleshooting and tuning your neural network (https://pcc.cs.byu.edu/2017/10/02/practical-advice-for-building-deep-neural-networks/).

```python
########################
#Select Model Parameters
########################
MAX_SEQUENCE_LENGTH = 10000 #Max sequence of 10k words from each sample
MAX_NB_WORDS = 400000 #Using the full Glove Vocabulary
EMBEDDING_DIM = 300 #Each word in the sequence represented by 300 values
VALIDATION_SPLIT = 0.2  #Train/Test Split
LEARNING_RATE = .00011
BATCH_SIZE = 33
DROPOUT_RATE = 0.45  #Dropout applied to last layer
INNERLAYER_DROPOUT_RATE = 0.15  #Dropout applied to inner layers

##############
#1D CNN DESIGN
##############
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')

embedded_sequences = embedding_layer(sequence_input)

x = Conv1D(128, 5, activation='elu', kernel_initializer='lecun_uniform')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Dropout(INNERLAYER_DROPOUT_RATE)(x)

x = Conv1D(128, 5, activation='elu', kernel_initializer='lecun_uniform')(x)
x = MaxPooling1D(5)(x)
x = Dropout(INNERLAYER_DROPOUT_RATE)(x)

x = Conv1D(128, 5, activation='elu', kernel_initializer='lecun_uniform')(x)
x = MaxPooling1D(35)(x)   # global max pooling

x = Flatten()(x)
x = Dense(100, activation='elu', kernel_initializer='lecun_uniform')(x) # best initializers: #glorot_normal #VarianceScaling #lecun_uniform
x = Dropout(DROPOUT_RATE)(x)

preds = Dense(len(labels_index), activation='softmax')(x) #no initialization in output layer

model = Model(sequence_input, preds)
```

```
Layer (type)                  Output Shape            Param #
=================================================================
input_1 (InputLayer)          (None, 25000)           0

embedding_1 (Embedding)       (None, 25000, 300)      9686400

conv1d_1 (Conv1D)             (None, 24996, 128)      192128

max_pooling1d_1 (MaxPooling1  (None, 4999, 128)       0

conv1d_2 (Conv1D)             (None, 4995, 128)       82048

max_pooling1d_2 (MaxPooling1  (None, 999, 128)        0

conv1d_3 (Conv1D)             (None, 995, 128)        82048

max_pooling1d_3 (MaxPooling1  (None, 28, 128)         0

flatten_1 (Flatten)           (None, 3584)            0

dense_1 (Dense)               (None, 128)             458880

dropout_1 (Dropout)           (None, 128)             0

dense_2 (Dense)               (None, 3)               387
=================================================================
Total params: 10,501,891
Trainable params: 815,491
Non-trainable params: 9,686,400

Test score: 0.954990378798
Test accuracy: 0.51598174768
```

The Result

Our prototype model results, while modest, suggest there is a useful signal available on future performance classification in at least the biotechnology industry based on the target text from the 10-K. The resulting statistics are listed below, including the statistics by class. For our model, '0' represents low performance, '1' represents middle performance and '2' represents high performance (see model evaluation notebook (https://github.com/SingingData/StockPerformanceClassification/blob/master/JupyterNotebooks/Step%204%20-%20Model%20Evaluation%203%20Classes%20Reporting.ipynb)).
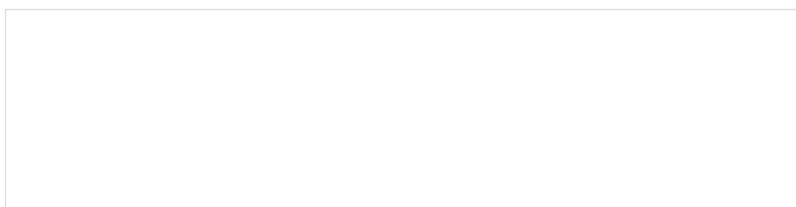
```
 1  Overall Statistics:
 2
 3  Accuracy: 0.502732240437
 4  95% CI: (0.42803368164557776, 0.57734086516071881)
 5  P-Value [Acc > NIR]: 0.133021698692
 6  Kappa: 0.25479930192
 7
 8
 9  Class Statistics:
10
11  Classes                                      0          1          2
12  Population                                 183        183        183
13  P: Condition positive                       61         62         60
14  N: Condition negative                      122        121        123
15  Test outcome positive                       84         39         60
16  Test outcome negative                       99        144        123
17  TP: True Positive                           38         29         25
18  TN: True Negative                           76        111         88
19  FP: False Positive                          46         10         35
20  FN: False Negative                          23         33         35
21  TPR: (Sensitivity, hit rate, recall)  0.622951   0.467742   0.416667
22  TNR=SPC: (Specificity)                0.622951   0.917355   0.715447
23  PPV: Pos Pred Value (Precision)       0.452381    0.74359   0.416667
24  NPV: Neg Pred Value
```

The confusion matrix (https://en.wikipedia.org/wiki/Confusion_matrix) below details the prediction comparing the true class of the sample, and the predicted class. The true label is on the vertical axis, and the predicted label coming from our model is on the horizontal axis. The top grid is the absolute count, and the bottom grid is the percentage. The visualization shows that our model performs best at predicting the true label of the low performing stocks, in the upper left.

For investment firms, predicting likely under-performers may be the most valuable prediction of all, allowing them to avoid losses on investments that will not fare well. Chance would have given us a 33.3% accuracy for any one classification. In this model, we are seeing 62% accuracy for predicting the under-performing company based on the sample 10-K text.

Confusion matrix, without normalization

Normalized confusion matrix
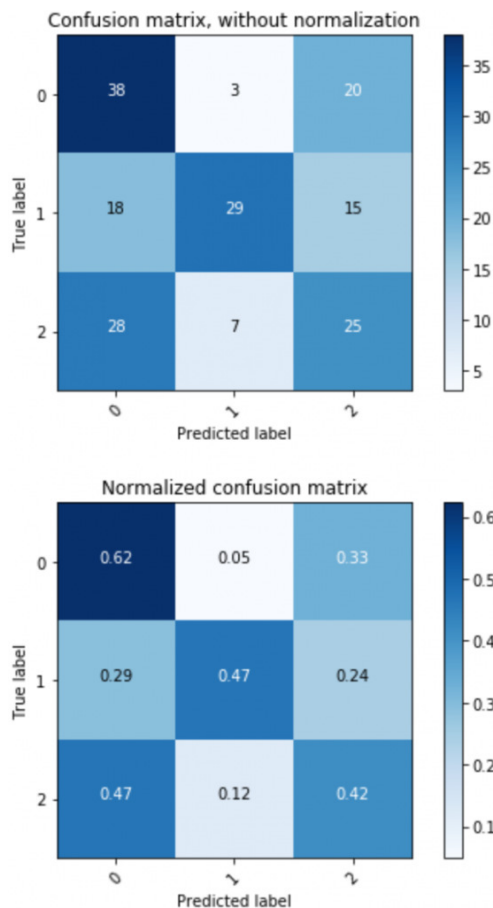
The history of model training and testing is below, trained for 24 epochs.
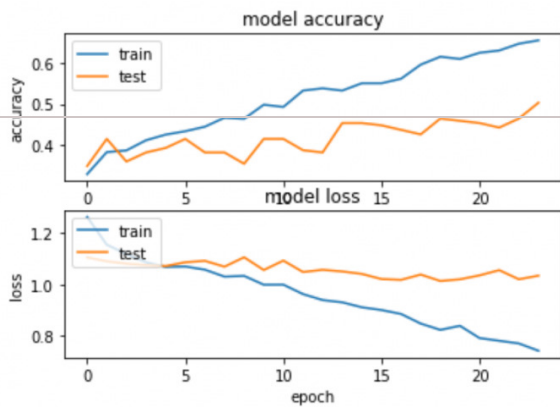
Next Steps

This initial result suggests that that deep learning models trained on text in earnings releases and other sources could prove a viable mechanism to improve the quality of the information available to those making investment decisions, particularly in avoiding investment losses. While the model needs to be improved with more samples, refinements of domain-specific vocabulary, and text augmentation, it suggests that providing this signal as another decision input for investment analyst would improve the efficiency of the firm's analysis work.

Our partner will look to improve the model with more samples and to augment them with additional information taken from the earnings releases and additional publications and a larger sample of companies. They will also explore alternative model architectures including LSTM to better understand the sequential nature of the publication and performance information. In addition, they will look to replicate this model for different industries and operationalize the model with Azure Machine Learning Workbench, allowing auto-scaling and custom model management for many clients.

Overall, this prototype validated additional investment by our partner in natural language based deep learning to improve efficiency, consistency, and effectiveness of human reviews of textual reports and information. Please feel free to reach out in comments below or directly via Twitter @SingingData (https://twitter.com/SingingData).

Featured Image by pixabay.com (https://pixabay.com/en/words-adjectives-feelings-1034410/)

Categories

Azure App Services

Big Data

Blockchain

Bots

Cognitive Services

Containers

DevOps

Frameworks

Internet Of Things

Machine Learning

Uncategorized

Follow us on Social Media

f   𝕏

Authors

Patty Ryan (https://www.microsoft.com/developerblog/author/patty-ryan/)