Analysis of SEC reports using word embedding

Abstract

A Word2Vec model was trained from scratch from a dataset of SEC filings, with the aim of performing topic modeling to predict future price movements of public stocks. While the Word2Vec vectors were not used in the final project, they acquired significant semantic information and we discuss possible improvements that could have made this approach viable.

1. Introduction

1.1. Motivation and approach

The SEC requires U.S. firms to file regular reports and makes these reports available through their EDGAR database. We sought to analyze the natural language content of these reports in order to predict price movements of a firm's stock shortly after the firm files a report. Previous research has used sentiment scores based on computing the frequency of words of a given sentiment - for example, the frequency of positive or negative words. These rely on hand-crafted word lists which must be tailored to the specific subject at hand (for example, a list of positive words for political speech will not work as well for analysis of business reporting: consider the term "security" which is positive in the political context but merely denotes a financial asset in the business context, as in "mortgage-backed securities") [**?**].

In addition to sentiment scores, we extend this frequency-based approach using custom word lists, defined by first computing semantic word embeddings from our corpus of SEC filings, then clustering the word vectors. Since each word list corresponds to a collection of semantically similar words, we call them topics, and we call the resulting frequency scores topic scores. The topic scores are of interest both for explanatory power in modeling returns around a filing date, and for use in a trading strategy based on predicting stock price movements after a firm submits an SEC report. They have the advantage of being derived from the text corpus automatically (without hand-crafting the word lists). Other recent research has done more advanced topic modeling, called text factor analysis, for which embedding and clustering is the first step [**?**]. We want to see if a simplified frequency-based approach to text factor analysis can recover a significant portion of its explanatory power.

1.2. The use of word embedding

Our hypothesis is that adding these topic scores to the sentiment scores will improve the predictive power of sentiment scores alone. To test this, we need the best topic scores possible, so it is prerequisite that the word embedding capture as much semantic meaning as possible. We tried two approaches for word embedding: a custom Word2Vec model trained from scratch on the SEC filing corpus, and the pretrained GloVe vectors trained on Wikipedia and the English Gigaword newswire datasets. We thought that the from-scratch model might be able to capture some of the peculiarities of the language contained in SEC reports, and thus "overfit" to their particular mode of corporate speech (which in this context would be a good thing). Ultimately this did not happen, and we decided to use the GloVe vectors based on heuristic and quantitative analysis of the quality of the topic clusters generated from both sets of vectors. However, the from-scratch approach may be viable as well, given some improvements discussed below.

2. Data

2.1. Parsing reports

First, we web-scraped a selection of reports from the SEC EDGAR database. We scraped:

- 10-K and 10-Q reports
- filed between 2004-01-01 and 2014-01-01
- filed by firms which were S&P 500 constituents at some time between 1995-01-01 and 2018-01-01

Second, we extracted the Management's Discussion and Analysis section from each report. While most of an SEC report is boilerplate content that doesn't change much from year to year or quarter to quarter, the MD&A section is more informal and contains the managers' opinions on threats to the firm, recent news, and the future outlook of the firm generally. It tends to be more variable and opinionated than the rest of the report. Therefore, we predicted that most of the semantic and topic loading information that would be useful in predicting future price movements would be concentrated in the MD&A section.

Finally, we processed the extracted text by removing punctuation/numbers and lemmatizing words, converting each filing into a clean list of words. This constituted our text corpus. From this we computed the most frequently occurring 20,000 words based on a random sample from the corpus and appended the token <UNK> to represent all other words. This constituted our 20,001-word vocabulary (universe of words) to be embedded. We describe the construction of the training dataset for the model after describing the structure of the model below.

2.2. Other considerations

At first we intended to parse 8-K reports as well, which are unscheduled reports filed when a firm experiences a major event that may affect shareholders. Naturally, these could have a significant effect on the firm's stock price. However we found that it was much more difficult to parse these reports, as there is no particularly valuable section to extract (like the MD&A section for 10-K and 10-Q reports) and the forms can have vastly different contents and organization. There were also far more 8-K reports than any other kind, leading to more data than we could reasonably process. For these reasons we decided to restrict to 10-Q and 10-K reports only.

3. The Word2Vec model

The Word2Vec model is a neural network that learns word embeddings, which are mappings of words into a (relatively) low-dimensional vector space.

We used the CBOW (continuous-bag-of-words) variant, with a context window of 5. Given a list of V words, and an embedding dimension of E, the CBOW model consists of two layers with learnable parameters:

1. the embed layer: fully-connected, (V, E)
2. the decode layer: fully-connected, (E, V).

To build the training dataset, we load a processed filing, slide a context window of length 5 over the entire filing, and convert each 5-tuple $(w_1, w_2, w_3, w_4, w_5)$ of consecutive words into a $4 \times V$ array, where the rows are the one-hot encodings of the four context words w_1, w_2, w_4, w_5. The one-hot encoding of the middle word w_3 is used as the target for the cross-entropy loss computation. This process is repeated for all processed filings to generate a $4 \times V \times N$ array of inputs and a $V \times N$ array of targets, where N is the number of 5-tuples of consecutive words in the text corpus.

Thus, the model learns to predict a word based on the surrounding context words by passing the context words through an embedding layer and taking their average. The idea is that a model which can accurately make these predictions must have encoded some significant semantic knowledge in the embed layer.

Note that to be completely precise we should say that the model consists of four weight-sharing embed layers whose output is simply averaged then fed to the decode layer. Figure 1, borrowed from [**?**], shows this architecture.

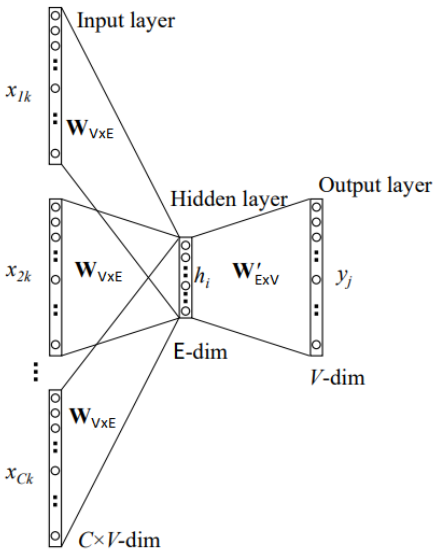

Figure 1. CBOW Word2Vec architecture

4. Implementation and potential improvements

4.1. Implementation

The model was implemented using PyTorch, with a vocabulary of size $V = 20001$ and embedding dimension $E = 100$. We verified the implementation using techniques demonstrated in earlier projects for this course: first verifying the appropriate loss for a completely untrained model with random initial weights, then overfitting the model to a very small dataset and verifying that the model achieved near-perfect accuracy on this small dataset.

Although a total of 63 million training samples were built from the corpus, a random sample of 20 million was used due to computational limits. Training the final model consisted of two epochs of batched stochastic gradient descent using the Adam optimization algorithm. While training could have continued further (the validation loss was still decreasing and the model did not show evidence of overfitting), we decided the result would probably not improve upon the pretrained GloVe vectors so we stopped developing the Word2Vec model.

One improvement we made was to incorporate data subsampling. Data subsampling is done while constructing the training dataset. Instead of using every word in the filing, we first delete some words based on the frequency of the words in the corpus. If the frequency of a word is f, then

the probability of keeping that word is

$$\left(\sqrt{\frac{f}{0.001}}+1\right)\frac{0.001}{f}$$

As the frequency of a word increases, the probability to keep it decreases. There is a frequency threshold below which the word will never be deleted, and there is a minimum probability to keep a word (no matter how frequent), both determined by the constant 0.001 in the formula above. Using data subsampling showed a decrease in the time required to fully train the model on a smaller dataset, so we rebuilt the dataset to incorporate subsampling and used the subsampled dataset for the final model.

4.2. Potential improvements and sources of error

An improvement which is well-known for Word2Vec models, but which we did not try, is negative sampling [**?**]. Rather than updating the parameters for every word vector in every loss computation, instead a random sample of false or "negative" words are drawn and updated. This drastically reduces the time to compute the loss function since only around 20 negative samples need to be used, and yields a corresponding speedup in the backpropagation as well. This would have allowed us to train for more epochs and on more data. However, based on the results so far, we judged that even with this improvement, the GloVe vectors would probably still be superior for our project, so we did not implement negative sampling.

While examining the GloVe vectors, we found that many of the "words" included stopwords, punctuation marks, and other oddities, whereas we specifically filtered out these types of tokens. It's possible that we should have included these in our vocabulary in order to train the best vectors, even though we would not perform topic modeling on these tokens themselves. Similarly, it may have been better to keep all forms of words (rather than lemmatizing them) and increase the size of our vocabulary. This would have increased the computation time even more, and we were already at our limit for the project. But, given more computational power and a commitment to the custom Word2Vec approach, we would have made this modification.

There were other possible choices for models. For example, we could have increased the size of the context window above 5. Or, we could have used the Skip-Gram model, a variant of Word2Vec which predicts context words from the target word, and has been shown to outperform the CBOW variant at many tasks [**?**]. Again, either of these changes would have increased computation time as well as development time and we did not implement them.

Finally, another option would be to use the pretrained GloVe vectors in the embed layer, then train on our corpus to fine-tune them to our specific project. Given more time to develop the model we would certainly try this, as fine-tuning has been shown to produce significant increases in performance for many different types of models in many contexts, and is relatively inexpensive computationally.

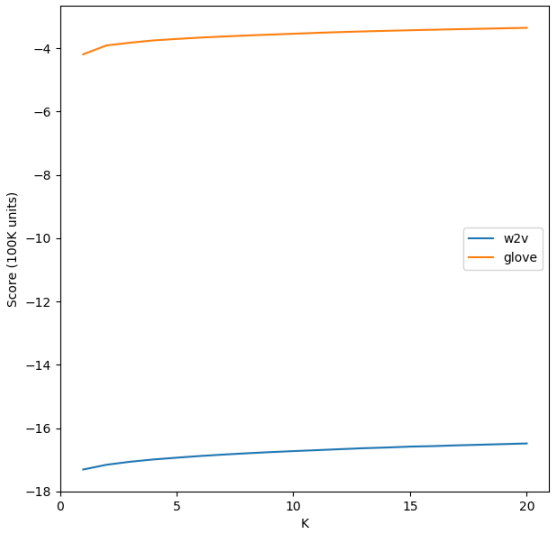

Figure 2. Negative average distance scores for K-means clusters with varying K

5. Results

5.1. Comparison of clusters

We concluded that the GloVe vector clusters were superior to the custom Word2Vec clusters based on

1. how well the K-means algorithm was able to cluster the two sets of word vectors, as measured by the average distance from points to the centroids of their respective clusters
2. visual inspection of the clusters by using PCA to project and plot them in two dimensions, and
3. inspection of the high-frequency words within topics.

Figure 2 shows the scores (negative average distance, so higher is better) of the K-means clusters, with varying K, for both the GloVe and custom Word2Vec vectors. Note that the vectors are 100-dimensional for both sets of vectors, so it is reasonable to compare distances. Clearly the algorithm is able to produce better clusters for the GloVe vectors by this metric, for every value of K.

Figure 3 shows the plots of the first 3 clusters, projected to 2 dimensions using PCA. It is visually clear that the clusters achieve better separation from each other for the GloVe vectors, and the clusters for the custom Word2Vec vectors look almost like they were obtained from random noise.

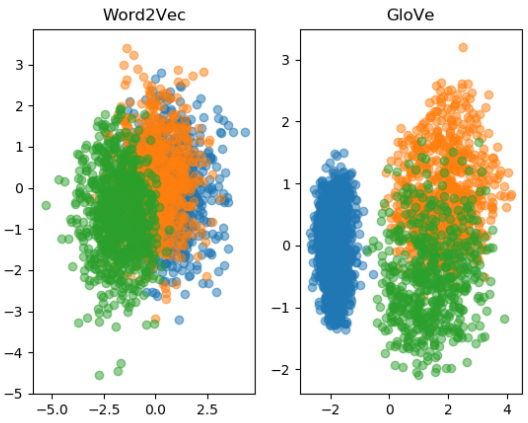

Figure 3. 2-D projections of topic clusters

GloVe Topic 8	GloVe Topic 12	Word2Vec Topic 3	Word2Vec Topic 5
Chemical Manufacturing	**Accounting**	**Financial Growth**	**International Finance**
manufacture	amortization	increase	partially
herein	allowance	quarter	profit
copper	contractual	year	annual
recoverable	receivables	operate	materially
synthetic	receivable	primarily	section
liquid	intangible	compare	maintenance
residual	gaap	decrease	plant
depletion	payable	share	save
chemicals	equivalents	lower	negatively
quantities	unrealized	also	mine
chemical	unsecured	earn	canada
ore	collateral	growth	conversion
wafer	amortize	decline	previous
emission	impairments	u.s.	canadian
intermediate	maturities	percent	utilization
nickel	severance	average	california
fiber	contingencies	gross	reportable
accretion	marketable	margin	pacific
fabrication	indebtedness	generally	start
liquids	ebitda	generate	industries

Figure 4. Frequent words from topic clusters. We judged the highlighted words as strongly related to the topic.

Indeed, when examining the high-frequency words in each cluster, we found that most of the GloVe clusters had collected similar words together with a clear shared meaning or "topic". In Figure 4 we give two examples from each vector set. For each topic we have written our guess for the meaning of the topic and highlighted the words which are strongly related to that meaning. GloVe topic 8 seems to contain words associated with chemical manufacturing and materials. GloVe topic 12 seems to contain words related to accounting, financial items, and payment terms. These two topics were chosen because they proved to be significant in the regressions we computed later in the project. All but 2 of the high-frequency words for these topics are strongly related to the topics' meanings, in our judgment.

However, it is much more difficult to assign clear interpretations to the Word2Vec topics in Figure 4. Word2Vec topic 3 seems to contain words related to reporting of financial growth, but also contains rather weakly related words like "u.s.", "also", and "generally". Word2Vec topic 5 is the most difficult to interpret. Our best guess is international finance, but it seems to have collected words related to physical resources and assets as well, like "maintenance", "plant", and "mine". So the subject would would have to be expanded to "international finance/location of physical assets". This clearly lacks the clarity of the GloVe topic interpretations.

In general, we found that the Word2Vec topic clusters were either too vague to interpret, or seemed to have collected several weakly related subjects together, while the GloVe topic clusters had a single, clear subject interpretation.

Therefore, all three methods of evaluation confirmed that the GloVe vectors generated superior clusters for the purpose of topic modeling.

5.2. Conclusion

Ultimately we decided to use the pretrained GloVe vectors instead of our from-scratch Word2Vec embedding. However, the results above show that the Word2Vec model did learn a significant amount of semantic information from the corpus which was reflected in the topic clusters. While the results were inferior to the pretrained GloVe vectors, we believe that the from-scratch approach could be made competitive by incorporating the improvements suggested above without prohibitively expanding the computational cost.