Mining Big Data - Final Report

Investigating Paragraph Vectors

a1632538 Zachary Forman
a1646930 James Caddy

October 30, 2016

Introduction

We live in a world where enormous amounts of textual data exists, and even more is generated each day. This data presents both a major opportunity and major challenges, being plentiful and rich in information, but also high dimensional and loosely structured. We can view text as consisting of words, and can describe any collection of words as a document. In the past, textual documents were frequently viewed as just a collection of words, the so-called "Bag of Words" model [1]. Each word in such a model is typically just a number - the semantic connotations of words and documents discarded for the sake of efficiency.
In 2013, Mikolov *et al.* presented the `word2vec` algorithm [2]. This approach allowed for each word to be represented by a vector that contains a compact representation of the semantic meaning of that word. In 2014, Le and Mikolov extended this approach [3] with the `doc2vec` algorithm to allow learning vectors that summarize the semantic meaning of a document.
We seek to extend Le and Mikolov's work, reproducing their `doc2vec` results and further extending them to the domains of document similarity and understanding source code.

Contributions

This work makes the following contributions:

- Replication of a relatively recent paper [3] that makes some contentious claims [4].
- Answers the age-old question of what the *Citizen Kane* of video games is.
- Comparison of `doc2vec`'s suitability as a document similarity measure as compared to Jaccard similarity.
- Application of the `doc2vec` algorithm to a novel domain – source code.

Background

Word Vectors

The first meaningful attempt at learning vector representations was Rumelhart *et al.* 's 1986 contribution [5]. This work, while relatively primitive, paved the way for more sophisticated models, such as Bengio *et al.* 's 2003 work [6]. This work presents a model that is extremely similar to the current state of the art: Learn vectors for words based on errors seen while trying to predict surrounding context.
Mikolov *et al.* present various enhancements to this technique [2, 7], improving both the speed of training the word vectors and the quality of the learned representations. The algorithm presented in these papers is known as `word2vec`, and works as follows (see also Figure 1):

1. First, a set of sequential words is taken, and the central word set aside.
2. This text is then projected into the vector space, and summed.
3. The resulting vector is then used to predict the central word.
4. The error in the prediction is backpropagated to refine both the estimates of the word vectors and the prediction network.

Mikolov *et al.* show that these learnt word vectors not only retain semantic information, but also relational information; for example, the result of $\text{vec}(\text{king}) - \text{vec}(\text{man}) + \text{vec}(\text{woman}) \approx \text{vec}(\text{queen})$.

Paragraph Vectors

In 2014, Le and Mikolov presented a method of altering the `word2vec` algorithm to jointly train vector representations of documents in addition to word vectors [3]. This method is based on the idea of basically adding a "context vector", representing an individual document, to each optimization. The process of learning these document vectors is equivalent to that of learning word vectors in the `word2vec` algorithm, except an additional paragraph vector (representing the document a given text sample comes from) is added - see Figure 1 for a graphical representation of the difference. This modified `word2vec` algorithm is called `doc2vec`.
Further exploration by Lau *et al.* shows that the `doc2vec` algorithm provides many of the same interesting

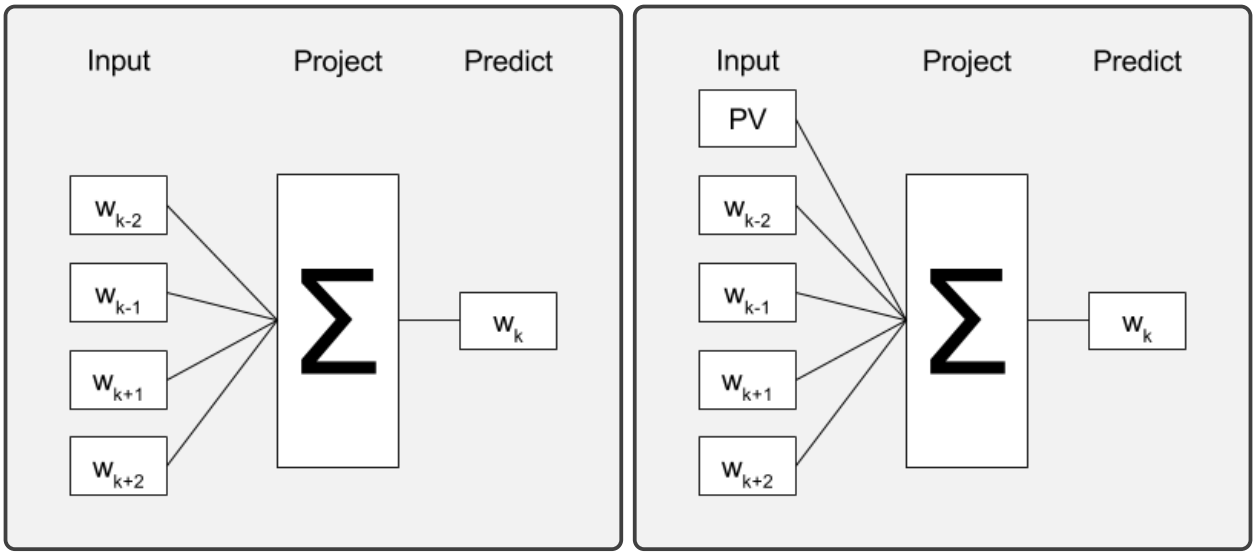

Figure 1: Graphical representation of the `word2vec` (left) and `doc2vec` (right) algorithms.

properties that `word2vec` can provide [8]. Not only do the document vectors capture semantic information, they *also* capture relations: famous singers have similar vectors, and using document vectors trained on wikipedia pages, "Lady Gaga" - "America" + "Japan" is closest to "Ayumi Hamasaki", a famous Japanese singer [9].

Data Collection

Data Sources

We used three sources of data. Firstly, we use Maas' IMDB set [10] for verification and English language documents which can be found at `http://ai.stanford.edu/~amaas/data/sentiment/`. Secondly, we used The Wikimedia Foundation's English Wikipedia data dump [11] as a source for English language documents. We downloaded the full wiki dump from `https://dumps.wikimedia.org/enwiki/20160820/`. Lastly, we used The Chromium Project's Chromium repository [12] as a source for source code documents. We cloned the repository at revision 4fe31bb06cf458234d7017950a8b2b82427487c8.

Data Scale

The IMDB dataset contains a total of 100,000 files. Half are labelled as either a positive or negative reviews, and the remainder are unlabelled. It is 346MB in size and contains 23 million words, from a vocabulary of 90 thousand words.
The English Wikipedia Dataset is a single XML file, 13GB compressed and 55GB uncompressed. It contains 16 million pages, however this includes redirects and meta-pages such as categories and portals, and therefore only 5 million of these are useful as documents. Excluding markup, this dataset contains approximately 4 billion words.
The Chromium codebase is 2.6GB in size, and contains 245 thousand files. Of these, 23 thousand are C and C++ files, which contain 4.8 million lines of code and 608 thousand lines of comments.

Data Format

The fundamental requirements for the `doc2vec` [3] algorithm to work are that each document can be associated with the words in the document, and that words in each document are just a series of words, i.e. no excess punctuation or markup.
This requirement can easily be met by storing our set of documents as a newline separated list of space separated tokens. The following example shows how a simple document set might be transformed into this file format:

{"My cow was sick, but now it is better.", "Apples are nice.", "Amazing results!"}

Convert all text to lowercase and remove all punctuation:

{“my cow was sick but now it is better”, “apples are nice”, “amazing results”}

Concatenate documents, separated by newlines:

```
my cow was sick but now it is better
apples are nice
amazing results
```

This format is simple to produce, simple to parse, and - given that the `doc2vec` algorithm needs to process every word anyway - is efficient to read. Perhaps more importantly, especially for large datasets like the Chromium dataset, this storage format has no storage requirements in excess of just the raw text in the files.

Data Processing

To transform the data into the format previously described, some data processing is required. All programs that we used to transform the datasets are available at `https://github.com/mbd-doc2vec-team/mbd-doc2vec/tree/master/data-processing`.

IMDB Dataset

The IMDB dataset needed minimal processing. The bulk of the work had already been performed by the authors of the dataset. Beyond what had already been done, punctuation, symbols and elements of markup were removed, and each document concatenated into a file as discussed in the previous section.

Wikipedia Dataset

The Wikipedia dataset required extensive pre-processing, with each document needing to be parsed from the XML, and the plain text extracted from the wiki markup. Additionally, “fake” documents (e.g. portals, category pages) were removed. Each wiki page was then tokenized and stored in a file as previously stated.

Chromium Dataset

The Chromium codebase required extensive pre-processing as well, with each C++ file being tokenized and having unicode characters (e.g. ¿) removed. Unlike the other two datasets, because punctuation and uppercase/lowercase distinctions are semantically significant in code, we retained punctuation and the original casing in tokens. Each comment and string literal was then tokenized as well, with punctuation being removed from these tokens only, and the resulting tokens being stored as discussed in the previous section. For an example of this process, the document

```
// Frobulates the bar, in an efficient manner.
void Frobulate(Bar &bar);
```

would be represented by a line in the file as follows

```
Frobulates the bar in an efficient manner void Frobulate ( Bar & bar ) ;
```

Experiments

Following are details as to how we carried out our experiments. Full source code to do this can be found at `https://github.com/mbd-doc2vec-team/mbd-doc2vec`, and full datasets can be obtained as listed in the “Data Sources” section of this paper.

IMDB Dataset

Preprocessing

- The entire set of 100,000 documents were processed as described in the previous sections, using the imdb_cleaner.sh and filecat.sh scripts available at `https://github.com/mbd-doc2vec-team/mbd-doc2vec/blob/master/data-processing/`

Training

- One of the co-authors of the original paper, Mikolov, provided some guidance and recommended the use of gensim's skip-gram model [13], which is trained for 20 iterations on the entire corpus of reviews.
- After each iteration, the order of the documents is randomized.
- The parameters we used are available at `https://github.com/mbd-doc2vec-team/mbd-doc2vec/blob/master/doc2vec/train_imdb_embedder.py`.

Evaluation

- The training document vectors are paired with their labels and a logistic regression is calculated to fit the training data.
- Prediction of sentiment using the testing document vectors takes place and the accuracy of the predicted labels versus the actual labels is reported.

Wikipedia Dataset

Preprocessing

- The Wikipedia XML dump was processed as discussed in the Data Processing section, using the parsing program available at `https://github.com/mbd-doc2vec-team/mbd-doc2vec/blob/master/data-processing/wiki_parser.cc`.

Training

- The gensim [13] `doc2vec` implementation was used to train document vectors for Wikipedia as well. We used document vectors of dimensionality 200, a window of 8, and summed vectors instead of concatenating them. We removed all words that occurred less than 8 times for being too rare.

Evaluation

- We evaluated the quality of the semantic modeling of the document vectors by qualitatively considering relationships between document vectors.

Chromium Dataset

Preprocessing

- The entire chromium repository was processed as described in the previous section, using the parsing program available at `https://github.com/mbd-doc2vec-team/mbd-doc2vec/blob/master/data-processing/source_tokenizer.cc`.

Training

- The gensim [13] `doc2vec` implementation was used to train document vectors for Chromium. We used document vectors of dimensionality 200, a window of 8, and summed vectors instead of concatenating them. We removed all tokens that occured less than 8 times for being too rare, much as for the wikipedia dataset.

Evaluation

- To evaluate `doc2vec` on source code, we first investigated how well `doc2vec`'s learned document vectors performed in finding clusters of related source files.

- Secondly, we evaluated the learned word vectors performed on tokens in source, to see if the semantic relationships found made sense in the context of the codebase.
- Finally, we compared the quality of the similarity metric given by the cosine distance of adjacent document vectors. We compare and contrast it with the Jaccard Similarity coefficient.

Results

IMDB Dataset

Through experimentation and switching from a neural network based approach to a logistic regression, the final best accuracy achieved whilst attempting to replicate the results was 89.4%.
Being unable to replicate the results of Le and Mikolov in [3], some research into the result was conducted. Many others were trying their hardest to replicate these results, but ultimately the co-author Mikolov, who was not involved in the experimentation found the issue.
This was reported in [14], where Mikolov states the 92.6% accuracy is achievable using hierarchical softmax "only when the training and test data are not shuffled. Thus, we consider this result to be invalid". We therefore confirm this non-replication.

Wikipedia Dataset

Experimentation with the Wikipedia dataset reveals some remarkable semantic connections. Table 1 shows articles with semantic contexts that most closely resemble the article describing *Lady Gaga*. Each of these pages is that of another famous singer.

Page name	Cosine similarity
lorde	0.578353
lana del rey	0.574220
nicki minaj	0.571770
kreesha turner	0.568376
katy perry	0.557113

Table 1: Wikipedia pages most similar to *Lady Gaga*

Another nice feature of representing the embedding as a vector is being able to find A in the relationship "X is to Y, as A is to B" by summing the vectors for X and B, and subtracting Y. A simple example of how this works is to query the model for "London is to England, as _____ is to France"

```
>>> model.docvecs.most_similar([london - england + france])
(u'paris', 0.47777268290519714)
```

That the model can learn such a relationship - without any knowledge of what a capital city is explicitly being provided to it - speaks to its representative power.
Citizen Kane is a film widely and highly regarded by critics, and as such it has become a part of popular culture to ask the question: "What compares to the critical acclaim of *Citizen Kane* in other types of entertainment?". To explore the Wikipedia model's embedding of semantics, the model was applied to answer the question, "What is the *Citizen Kane* of video games?", a question widely debated on the internet. The results of querying the model are displayed in Table 2.

Page name	Cosine similarity
chrono trigger	0.555336
day of the tentacle	0.528209
deus ex	0.507725
kid icarus	0.506333
sid meier's alpha centauri	0.502858

Table 2: Wikipedia pages most similar to *Citizen Kane* − *Film* + *Video Game*

Each of these games has received critical acclaim, with the "worst" of them, *Kid Icarus*, known for having a dedicated following.

Chromium Dataset

The final application of this document vector model was to investigate the semantic relationships it gathers when applied to source code files. The results were mixed. Major clusters include C♯ code, autofill code and the WebKit rendering engine, as can be seen in Figure 2. There are clearly many other clusters, but these were less well defined, and frequently had clusters that overlapped due to image size limitations. Reprojection into three dimensions might help with this problem, but runs into issues when attempting to present the results in a 2-d format, e.g. here. Otherwise, there aren't a lot of relations that can be seen in the overall distribution.

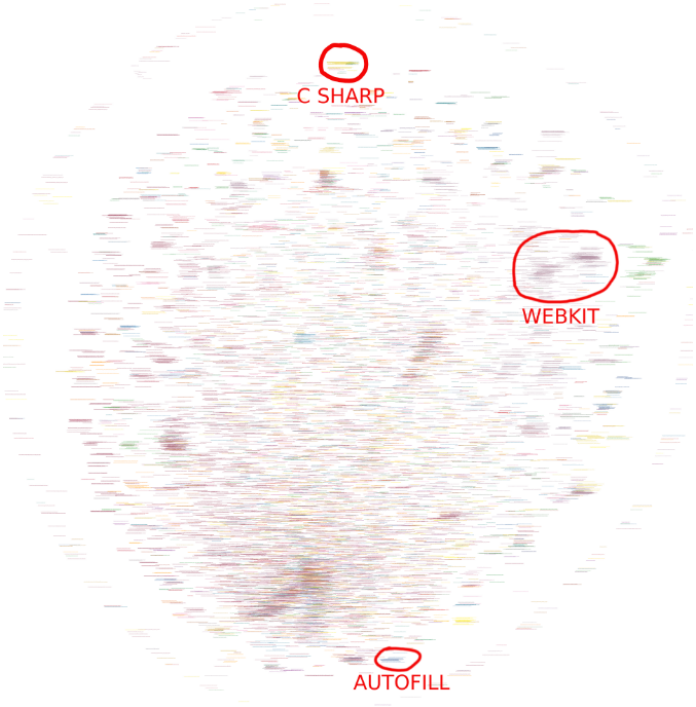

Figure 2: t-SNE representation of the Chromium codebase under a `doc2vec` transform.

Beyond these clusters, it's also interesting to see similarity patterns that form in the tokens, as can be seen in Table 3. It's clear that y, width, height, size and z are related to the idea of x as a coordinate.

Token	Cosine similarity
y	0.892718
width	0.798918
height	0.762949
size	0.721175
z	0.720715

Table 3: Tokens similar to x

We then randomly sampled several thousand chromium files and computed the Jaccard similarity and cosine similarity for each pair using the document vectors. We observed that in every instance where `doc2vec` based cosine distance indicated high semantic similarity, we found the two files to perform a similar role. In contrast, frequently files which had a high Jaccard similarity seemed quite dissimilar from the perspective of a software engineer. This relationship can be seen in Figure 3.
It's clear that Jaccard similarity fails completely to find semantic similarities on files with no words in common (see the left side of the graph), as well as overestimating semantic similarity in much of the codebase (consider that tokens like '+', 'for', 'while' will appear in almost every file, while a small handful of tokens, like comments and variable names, will define much of the semantic meaning of a given file). The cost of

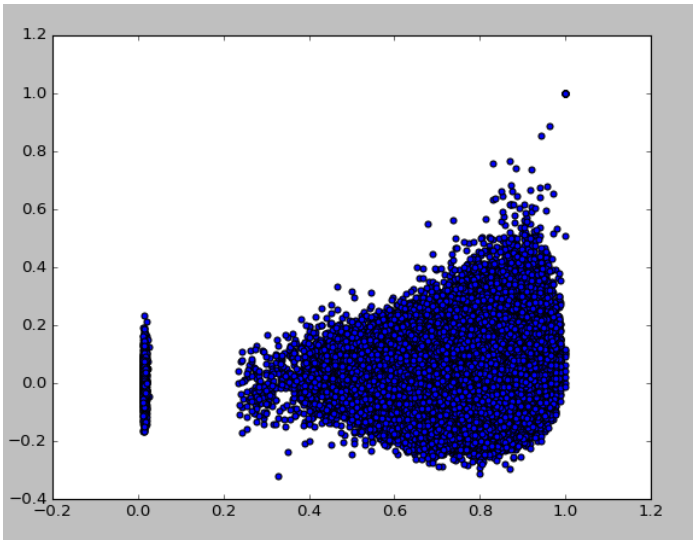

Figure 3: Jaccard similarity (x axis) vs Cosine similarity (y axis)

this improved quality of similarity metric lies in the increased time expenditure: The Jaccard similarity of a pair of documents can be computed in tens or hundreds of microseconds - or even faster using methods like minhash. In contrast, `doc2vec` takes order of 10s of milliseconds to infer a document vector for an unseen document. Given that the Jaccard similarity of two documents typically seems to serve as a useful upper bound for document similarity, however, it might be possible to opportunistically only apply the `doc2vec` based similarity metric to documents that are known to be somewhat similar, and achieve both high throughput and better evaluation of semantic similarity.

Conclusion

In this paper, we have examined the `doc2vec` algorithm. We find that

- Some of the claims made by Le and Mikolov in [3], specifically regarding accuracy achieved on the sentiment analysis tasks, are inaccurate, and reflect invalid experimental procedure as opposed to the actual performance of the `doc2vec` algorithm.
- The *Citizen Kane* of video games is *Chrono Trigger*.
- `doc2vec` with cosine similarity significantly improves upon Jaccard similarity in terms of evaluating the actual semantic similarity of two documents, but uses significantly more computing resources in order to do so.
- `doc2vec` can be applied to codebases and still produce valid, interesting similarity results.

Future Work

We identify several major directions for future work:

- More formal analysis of the actual speed and quality of `doc2vec` with cosine similarity is necessary. We suspect that use of a kd-tree might allow it to perform comparably well to minhashing with w-shingles for a sufficiently large set of prior documents.
- Investigating techniques like minhashing to see if they can successfully be augmented with a `doc2vec` based method to validate similar documents based on semantic similarity.
- Investigating whether combined corpora, e.g. training on a codebase and English language text, or on multiple different languages simultaneously, can lead to a general-purpose model that handles different categories of input.

References

[1] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. "Understanding bag-of-words model: a statistical framework". In: *International Journal of Machine Learning and Cybernetics* 1.1-4 (2010), pp. 43–52.

[2] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[3] Quoc V Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents." In: *ICML*. Vol. 14. 2014, pp. 1188–1196.

[4] *Google Groups*. URL: `https://groups.google.com/d/msg/word2vec-toolkit/q49firnoqro/bp--14e4unwj`.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[6] Yoshua Bengio et al. "A neural probabilistic language model". In: *journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[7] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[8] Jey Han Lau and Timothy Baldwin. "An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation". In: *arXiv preprint arXiv:1607.05368* (2016).

[9] Andrew M Dai, Christopher Olah, and Quoc V Le. "Document embedding with paragraph vectors". In: *arXiv preprint arXiv:1507.07998* (2015).

[10] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 142–150. URL: `http://www.aclweb.org/anthology/P11-1015`.

[11] Meta. *Data dump torrents — Meta, discussion about Wikimedia projects*. [Online; accessed 9-August-2016]. 2016. URL: `https://meta.wikimedia.org/w/index.php?title=Data_dump_torrents`.

[12] *Git repositories on chromium*. URL: `https://chromium.googlesource.com/`.

[13] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. `http://is.muni.cz/publication/884893/en`. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[14] Grégoire Mesnil et al. "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews". In: *arXiv preprint arXiv:1412.5335* (2014).