Word2Vec Model Instantiation and Hyperparameter Comparison in Parallel

Matthew Hoffman

December 2019

1 Introduction

Word2Vec is a simple neural network useful for contextualizing language vocabularies and representing words in high vector spaces. There exist two models for Word2Vec: **Continuous Bag-Of-Words**, and **Skip-gram**. In both architectures, the network uses a target item (i.e. word) and the context around it (i.e. the sentence) to produce distributed representations. In CBOW, a target context is used to determine a word that would fits, while in SG, a given word is used to consider a context it would fit in.

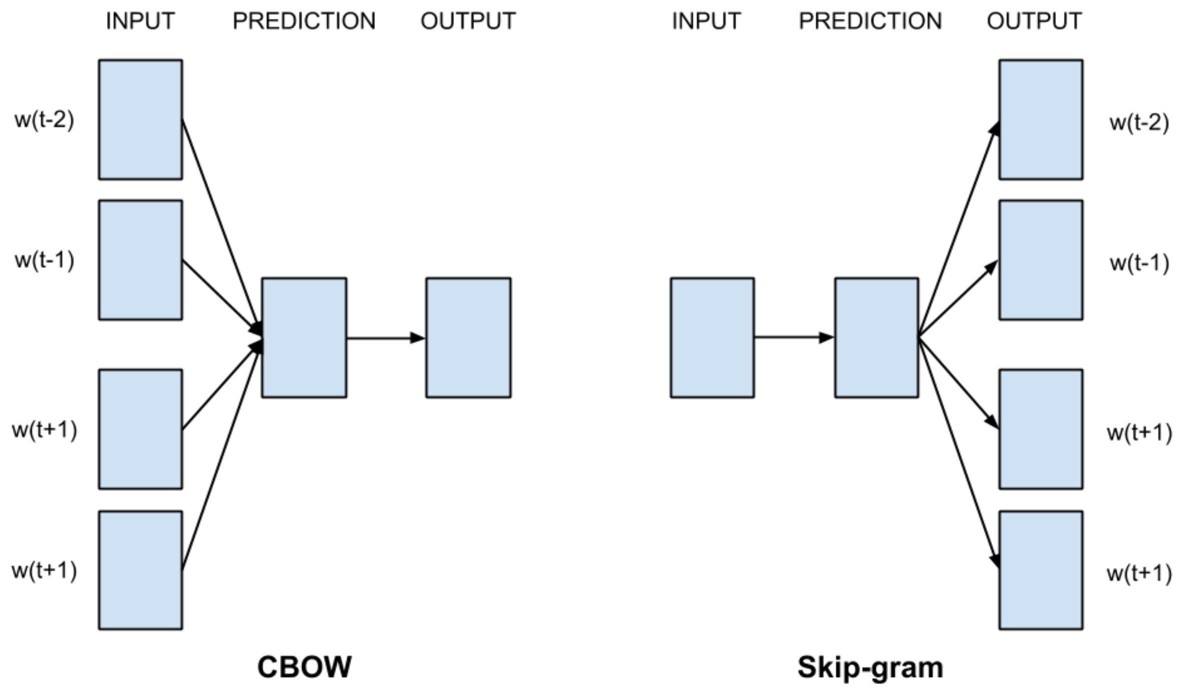


Figure 1: Image displaying the difference between CBOW and SG.

Like many neural networks, the outcome of Word2Vec is malleable by hyperparameters provided prior to training. Word2Vec uses window size to determine how many training pairs to create, i.e. how many context words surrounding the target must be used in training.

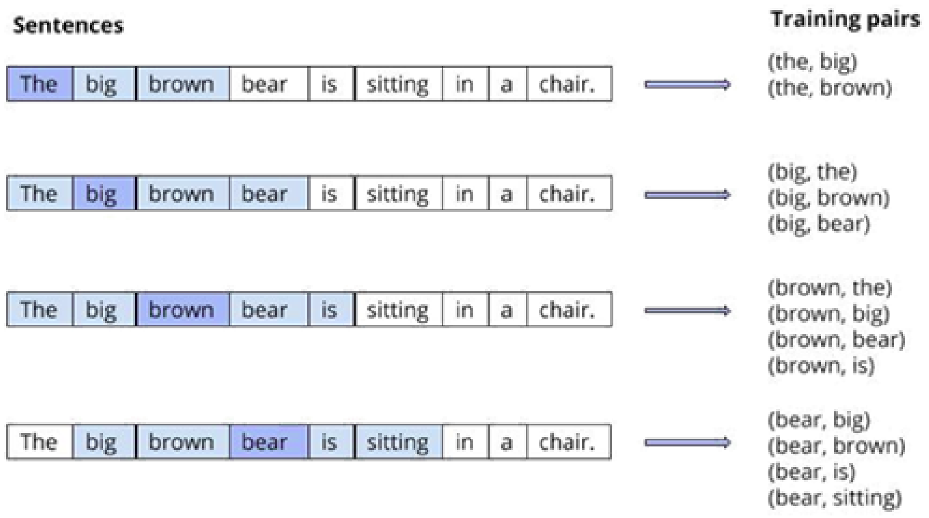


Figure 2: Demonstration of how training pairs are dependent on window size, e.g. 2 in the image above.

Additionally, a parameter of dimensionality is required for the final embedded vectors. PCA and other dimension reduction techniques can be used to depict these multi-dimension vectors as 2D graphs, but the vectors are in actuality as large as the hyperparameter provided.

While further training can improve the performance of neural networks, predefined constraints such as these can cause issues that cannot be remedied easily. Also, the creation of these models is both computationally expensive and memory intensive, limiting the opportunities of working with multiple at the same time.

The purpose of this paper is to solve the problems associated with working with multiple models simultaneously by creating a system to work with separate Word2Vectors models in parallel, extract valuable features, and to examine the final differences in the word-vectors under different hyperparameters, while improving both time and memory usage.

2 Program Pipeline

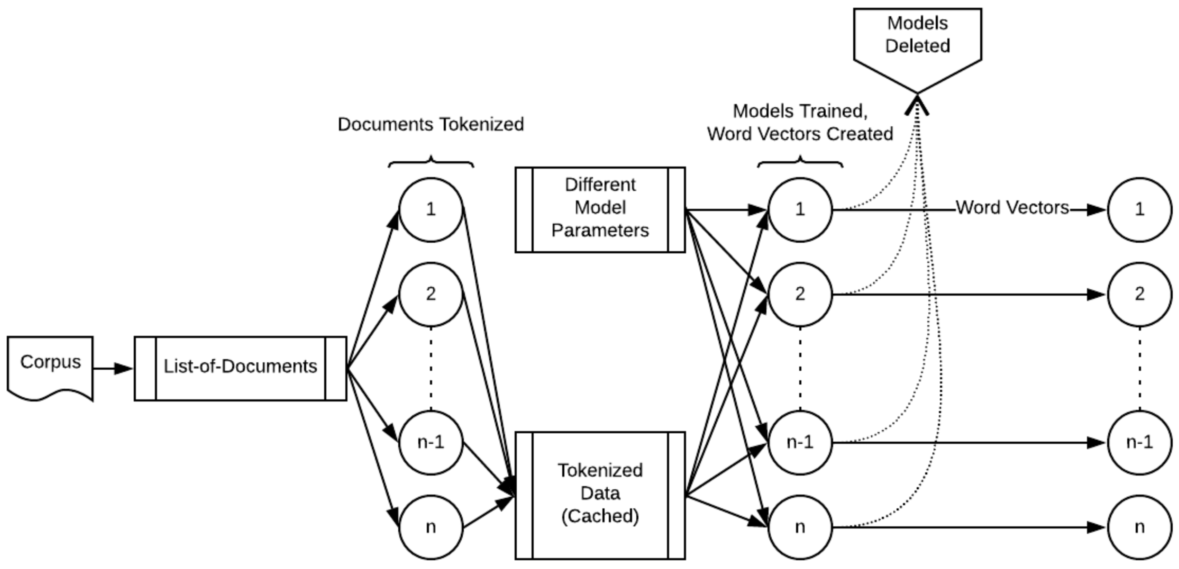
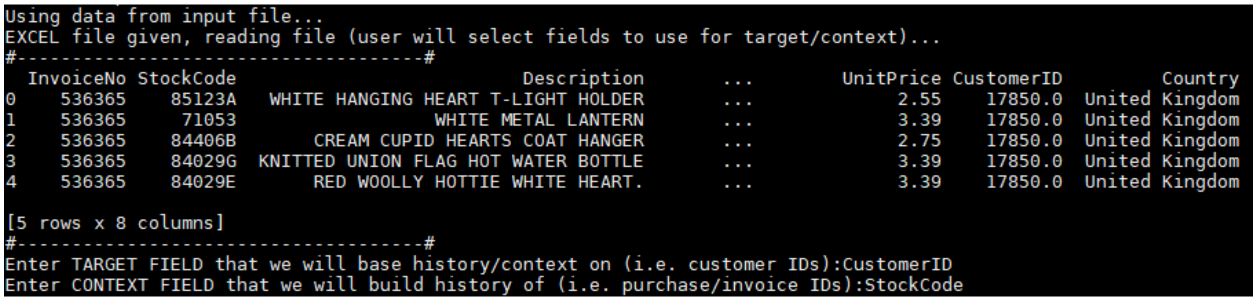


Figure 3: Graphic of the parallel pipeline used to create and compare models.

2.1 Data Preparation

The model was designed to take advantage of **PySpark** parallelism in several ways; first, data is preprocessed by either tokenizing or parsing through for useable information, dependent on the form of input. The default dataset for the program is the *Twenty Newsgroups* provided through Sklearn [5], but can run with any text file.

The system is designed to create associations between any data prepared in a list-structure, not solely natural language. For example, the program can also run with Excel sheets, generating sentences based on user specified fields for the target/contexts. Like word-associations, this can be used with product IDs to create recommendations based on histories of purchases (using these list-of-purchases as the context for Word2Vec).



```
Using data from input file...
EXCEL file given, reading file (user will select fields to use for target/context)...
#-------------------------------------#
  InvoiceNo StockCode                          Description       ...       UnitPrice CustomerID         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER       ...            2.55    17850.0  United Kingdom
1    536365     71053                  WHITE METAL LANTERN       ...            3.39    17850.0  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER       ...            2.75    17850.0  United Kingdom
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE       ...            3.39    17850.0  United Kingdom
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.       ...            3.39    17850.0  United Kingdom

[5 rows x 8 columns]
#-------------------------------------#
Enter TARGET FIELD that we will base history/context on (i.e. customer IDs):CustomerID
Enter CONTEXT FIELD that we will build history of (i.e. purchase/invoice IDs):StockCode
```

Figure 4: Output and prompts given an Excel database; the program asks the user to specify the target and context fields (displaying an *online retail dataset*[4])

.

2.2 Model Generation

Once the associations have been created out of the given corpus (word pairs, product histories, etc.), tuples are delivered to each worker specifying the hyperparameters for each model to be created: dimensionality, window size, and Word2Vector framework (CBOW or SG). The previously prepared data is cached, such that all workers have access. This is extremely beneficial with reference to window size, as word pairs are created up to the largest window size; therefore, smaller windows reuse combinations. Each driver generates a single Word2Vector model using **Gensim**[6].

2.3 Feature Extraction and Comparisons

Collecting each model would be extremely expensive with regards to memory; therefore, transformations take place on the workers to condition the data into useable forms. For example, by first extracting the vocabularies (i.e. the comprehensive word vectors) from the models, and by applying Uniform Manifold Approximation, a cluster graph can be generated representing each model for the vocabulary provided, and a plot can be returned for each. The models are discarded intermittently during this process.

3 Results Analysis

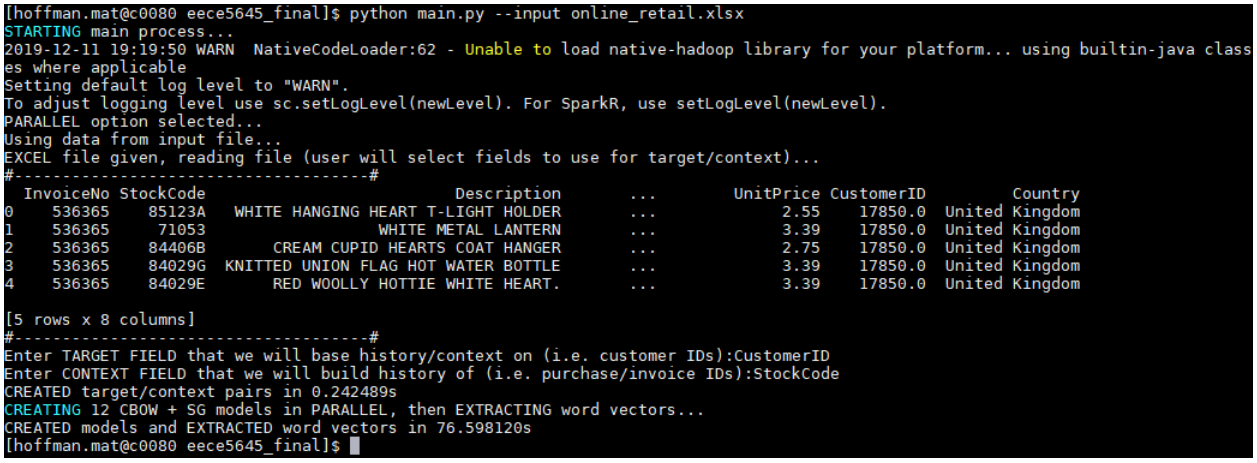
3.1 Serial versus Parallel Performance

The program is created to run in either parallel or serial, as specified by command-line arguments. This way, comparisons can be made as to how performance varies between the two processes.

With respect to preprocessing and data preparation, parallel computation vastly outperforms serial for larger datasets. While testing creating target/context pairs for the *online retail dataset* excel sheet (building histories of purchases based on user IDs), the parallel pipeline completed in an average 0.25s for 4372 unique users across 20 workers, while the serial pipeline completed the same task in 2m13s. As explained later, this illustrates that the parallel processes are altogether more efficient during this step.

Next, the pipelines were tasked with generating 12 Word2Vector models (with differing hyperparameters), extracting word vectors, and performing Uniform Manifold Approximations on each of them with the same dataset. The parallel version completed this task in 1m16.6s. Since each driver works with only one model, only 12 drivers worked at this task out of the 20 available, meaning that. In comparison, the serial pipeline completed in 4m21s, at about 21.82s/iteration.

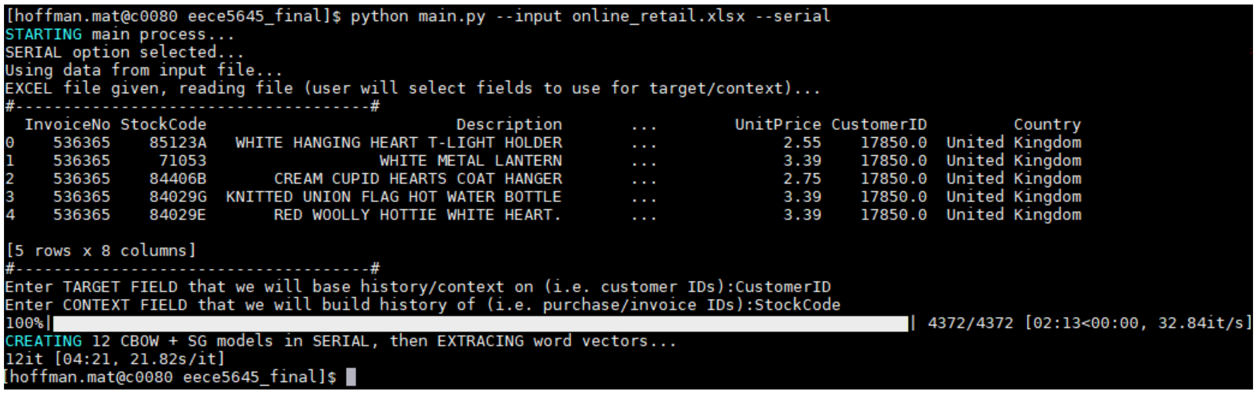
While performing completely-serial processes in parallel would only increase performance proportionally to the amount of workers utilized, the benefit of this parallel model is that shared tasks complete tasks drastically faster, greater than a constant-growth as expected from adding workers to separate processes. Here, it was demonstrated that the joint tasks were not only completed faster, but more efficiently, in parallel.



```
[hoffman.mat@c0080 eece5645_final]$ python main.py --input online_retail.xlsx
STARTING main process...
2019-12-11 19:19:50 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java class
es where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
PARALLEL option selected...
Using data from input file...
EXCEL file given, reading file (user will select fields to use for target/context)...
#-------------------------------------#
  InvoiceNo StockCode                          Description       ...       UnitPrice CustomerID         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER       ...            2.55    17850.0  United Kingdom
1    536365     71053                  WHITE METAL LANTERN       ...            3.39    17850.0  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER       ...            2.75    17850.0  United Kingdom
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE       ...            3.39    17850.0  United Kingdom
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.       ...            3.39    17850.0  United Kingdom

[5 rows x 8 columns]
#-------------------------------------#
Enter TARGET FIELD that we will base history/context on (i.e. customer IDs):CustomerID
Enter CONTEXT FIELD that we will build history of (i.e. purchase/invoice IDs):StockCode
CREATED target/context pairs in 0.242489s
CREATING 12 CBOW + SG models in PARALLEL, then EXTRACTING word vectors...
CREATED models and EXTRACTED word vectors in 76.598120s
[hoffman.mat@c0080 eece5645_final]$
```

Figure 5: Output of program with parallel pipeline on *online retail dataset* generating 12 different models.



```
[hoffman.mat@c0080 eece5645_final]$ python main.py --input online_retail.xlsx --serial
STARTING main process...
SERIAL option selected...
Using data from input file...
EXCEL file given, reading file (user will select fields to use for target/context)...
#-------------------------------------#
  InvoiceNo StockCode                          Description       ...       UnitPrice CustomerID         Country
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER       ...            2.55    17850.0  United Kingdom
1    536365     71053                  WHITE METAL LANTERN       ...            3.39    17850.0  United Kingdom
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER       ...            2.75    17850.0  United Kingdom
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE       ...            3.39    17850.0  United Kingdom
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.       ...            3.39    17850.0  United Kingdom

[5 rows x 8 columns]
#-------------------------------------#
Enter TARGET FIELD that we will base history/context on (i.e. customer IDs):CustomerID
Enter CONTEXT FIELD that we will build history of (i.e. purchase/invoice IDs):StockCode
100%|██████████████████████████████████████████████████████████████████████████████████████████| 4372/4372 [02:13<00:00, 32.84it/s]
CREATING 12 CBOW + SG models in SERIAL, then EXTRACING word vectors...
12it [04:21, 21.82s/it]
[hoffman.mat@c0080 eece5645_final]$
```

Figure 6: Output of program with serial pipeline on the *online retail dataset* generating 12 different models.

Finally, performance was examined on a trivially small data set, to examine when serial stops outperforming parallel. Surprisingly, the two work virtually the same. This is unexpected considering that the initial upstart for parallel processing is often overshadowed by simpler serial approaches.

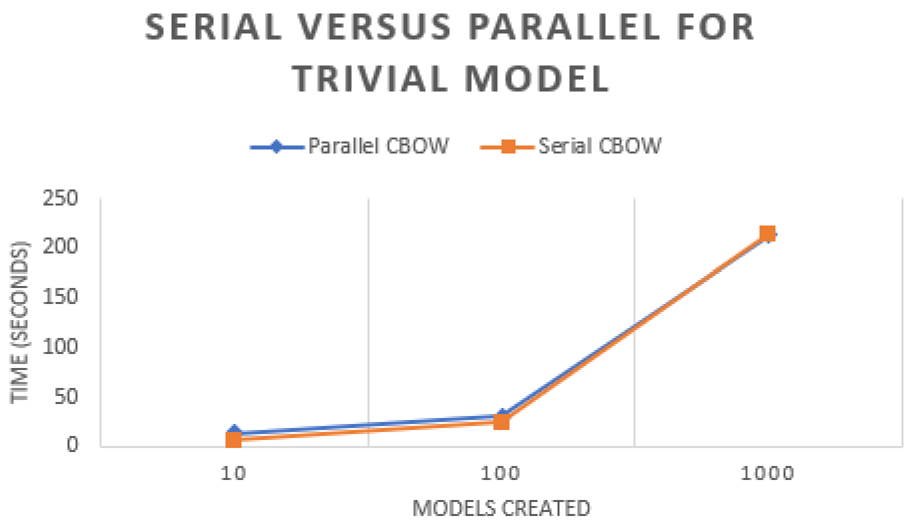
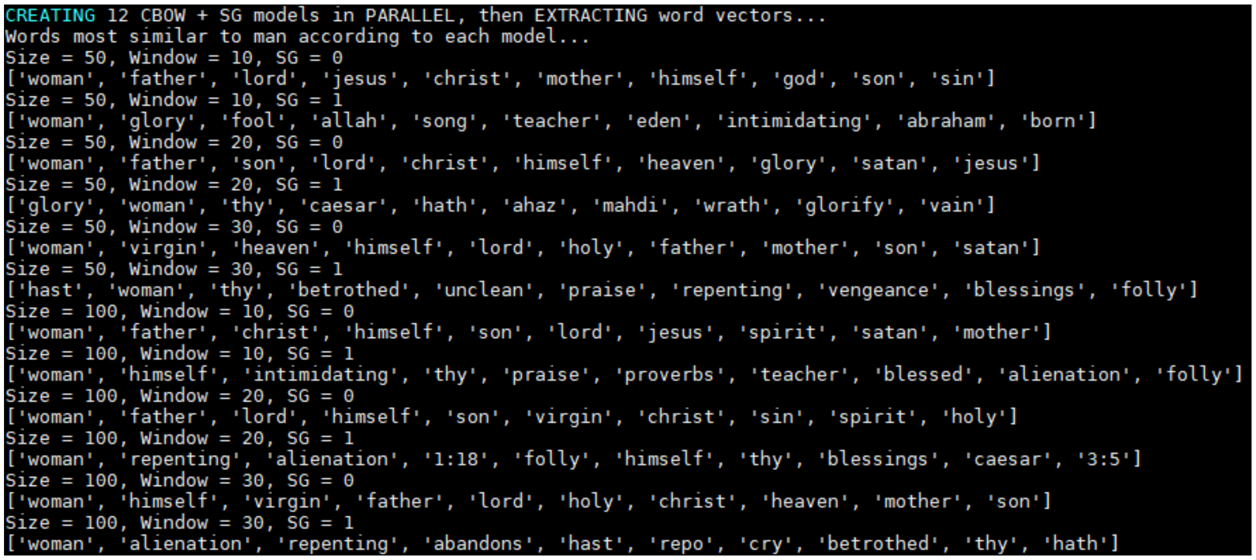


Figure 7: Near exact performance comparison between parallel and serial generation of CBOW Word2Vec models for the sentence *"the quick brown fox jumps over the lazy dog"*.

3.2 Word2Vector Features / Applications

The Word2Vector models had approximately the same results whether they were produced in serial or parallel, as intended. This was verified by examining the top-n most similar terms to sample words from the *Twenty Newsgroups* dataset, as well as by visually comparing plots created with the same hyperparameters through the different pipelines.



```
CREATING 12 CBOW + SG models in PARALLEL, then EXTRACTING word vectors...
Words most similar to man according to each model...
Size = 50, Window = 10, SG = 0
['woman', 'father', 'lord', 'jesus', 'christ', 'mother', 'himself', 'god', 'son', 'sin']
Size = 50, Window = 10, SG = 1
['woman', 'glory', 'fool', 'allah', 'song', 'teacher', 'eden', 'intimidating', 'abraham', 'born']
Size = 50, Window = 20, SG = 0
['woman', 'father', 'son', 'lord', 'christ', 'himself', 'heaven', 'glory', 'satan', 'jesus']
Size = 50, Window = 20, SG = 1
['glory', 'woman', 'thy', 'caesar', 'hath', 'ahaz', 'mahdi', 'wrath', 'glorify', 'vain']
Size = 50, Window = 30, SG = 0
['woman', 'virgin', 'heaven', 'himself', 'lord', 'holy', 'father', 'mother', 'son', 'satan']
Size = 50, Window = 30, SG = 1
['hast', 'woman', 'thy', 'betrothed', 'unclean', 'praise', 'repenting', 'vengeance', 'blessings', 'folly']
Size = 100, Window = 10, SG = 0
['woman', 'father', 'christ', 'himself', 'son', 'lord', 'jesus', 'spirit', 'satan', 'mother']
Size = 100, Window = 10, SG = 1
['woman', 'himself', 'intimidating', 'thy', 'praise', 'proverbs', 'teacher', 'blessed', 'alienation', 'folly']
Size = 100, Window = 20, SG = 0
['woman', 'father', 'lord', 'himself', 'son', 'virgin', 'christ', 'sin', 'spirit', 'holy']
Size = 100, Window = 20, SG = 1
['woman', 'repenting', 'alienation', '1:18', 'folly', 'himself', 'thy', 'blessings', 'caesar', '3:5']
Size = 100, Window = 30, SG = 0
['woman', 'himself', 'virgin', 'father', 'lord', 'holy', 'christ', 'heaven', 'mother', 'son']
Size = 100, Window = 30, SG = 1
['woman', 'alienation', 'repenting', 'abandons', 'hast', 'repo', 'cry', 'betrothed', 'thy', 'hath']
```

Figure 8: Top 10 most similar words to 'man' using *Twenty Newsgroups* models (SG = 0 means that this is a CBOW model). In 10 of the 12 models, 'woman' is the closest term.

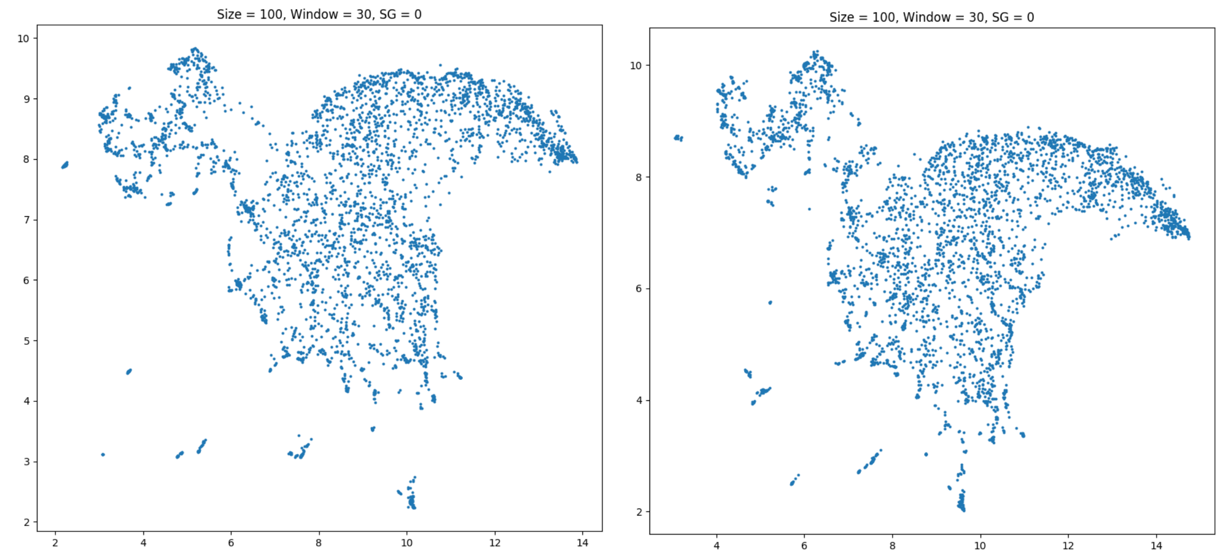


Figure 9: Comparison of plots created by the serial pipeline (left) and the parallel pipeline (right) with the same hyperparameters (SG = 0 means that it is a CBOW model) using the *online retail dataset*. Clusters represent similar purchases, and close nodes represents recommendable purchases.

Using the distance between vectors as shown in Figure 8, it is also possible to generate products similar to one selected, based on the history of other users' purchases. This is accomplished in the same way that "man" is used to find similar terms in Figure 7, although the term in question for this instance would be the Product ID.

4 Conclusion

This paper explored the concept of parallelism in Word2Vec model generation and comparison. The program developed showed it was extremely easy to work with multiple models using PySpark, with virtually no cons as compared to a serial pipeline. Using PySpark, the parallel computing pipeline vastly outperformed serial in preprocessing (tokenization/pair generation), model creation, and feature extractions (transformations, term similarity computation, etc.). This is accomplished by sharing terms across models with different parameters, such as window sizes (i.e. sharing word pairs), and allocating training of models to individual workers.

References

[1] Prateek Joshi and Analytics Vidhya. *Build a Recommendation System Using word2vec in Python*. Aug. 2019. URL: https://www.analyticsvidhya.com/blog/2019/07/how-to-build-recommendation-system-word2vec-python/.

[2] Sumedh Kadam. *Python: Word Embedding using Word2Vec*. May 2018. URL: https://www.geeksforgeeks.org/python-word-embedding-using-word2vec/.

[3] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[4] *Online Retail Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/online+retail.

[5] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2825–2830. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2078195.

[6] 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.