N-Gram Graphs for Text Classification: A Distributed Approach

Ioannis Kontopoulos[1], George Giannakopoulos[1], and Iraklis Varlamis[2]

[1] Institute of Informatics and Telecommunications, N.C.S.R. "Demokritos", Greece,
{ikon, ggianna}@iit.demokritos.gr
[2] Harokopio University of Athens, Department of Informatics and Telematics, Greece,
varlamis@hua.gr

Abstract. The power of n-gram models in capturing syntactic patterns using large text corpora made them the tool of choice for language modeling in machine translation, speech recognition, summarization and other tasks. N-grams have been frequently used for developing features for supervised classifiers and improved the performance of unigram features and recently n-gram graphs managed to capture significant language characteristics that go beyond mere vocabulary and grammar and have been established as a prominent representation for text classification. This work proposes a distributed implementation of the n-gram graph framework on Apache Spark, named ARGOT. The main operations of the n-gram graph framework (graph similarity, graph merging and update) have been redesigned in order to execute efficiently in the distributed environment. The implementation performance is evaluated on a demanding text classification task where the n-gram graphs are used for extracting features for a supervised classifier. The experimental results show the scalability of the proposed implementation to large text corpora and its ability to take advantage of a varying number of processing cores. Although the emphasis is on the scalability of algorithms, the classification algorithm also achieves state-of-the-art accuracy performance on a well-established multilingual dataset, which adds to the efficiency of the proposed approach.

Keywords: Distributed Processing; N-gram graphs; Text classification

1 Introduction

Text classification is a supervised machine learning technique for identifying predefined categories that are related to a specific document. Typically, a classification model is trained over a set of labeled documents and then predicts the label of previously unseen, unlabeled documents. Text classification is a popular research topic with many applications ranging from simple tasks such as spam filtering [2] to more complex tasks such as the analysis of social media content [1]. The continuous growth in the amount of text generated in social media imposes

the need for scalable techniques that take advantage of the available computing power to the maximum. Several distributed processing paradigms have been implemented to tackle exactly this need [6], [7], [8], [9], allowing parallel and distributed solutions for text mining tasks.

Parallel processing refers to the method where a large task is divided into smaller tasks, which are computed simultaneously in order to decrease the total execution time. From the dawn of multi-core machines, applications were developed to use each core and take advantage of as much computation power as available on a single, multi-core machine. However, with the constant increase of large scale data, monolithic solutions reach their limits and architectures that span many machines are the only solution. Distributed computing applications employ in tandem the processing cores of several machines in the same network or cluster thus allowing an – apparently infinite – upscale to related solutions.

The n-gram graph framework (*nGG* framework [4], [5]) is supported by a toolkit[3] for the creation and processing of n-gram graphs. It has been successfully employed in several NLP tasks ranging from summarization [22] to text classification [3] and indexing with excellent results. The application of the *nGG framework* in large corpora clearly sets the need for scalability.

This article proposes a newly developed framework, called ARGOT, which provides a distributed implementation of n-gram graph algorithms. The implementation is tested on a text classification task and handles the time consuming processes of feature extraction and graph merging in order to improve the overall scalability in large corpora. The ARGOT framework improves the scalability of the *nGG framework*, while retaining its state-of-the-art performance. The experimental evaluation studies the scalability of the solution and the effect of the dataset and features set size to the time performance of the implementation.

Section 2 that follows summarizes related work, whereas Section 3 explains the *nGG* Framework. Section 4 provides the implementation details of ARGOT and Section 5 the results of our experimental evaluation. Finally, Section 6 concludes this work and summarizes the next steps.

2 Related Work

Human-generated content grows by massive amounts every day. Thus, new architectures and tools are needed to process this data volume efficiently by using every available processing resource. Apache Hadoop[4] is a platform for distributed storage and distributed processing of very large datasets on computer clusters built from commodity hardware. Apache Hadoop uses MapReduce tasks to process the high volume of data. MapReduce is the programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. Apache Spark [5] is another cluster-computing framework used for in memory large-scale data processing.

[3] https://github.com/ggianna/JInsect
[4] http://hadoop.apache.org/
[5] http://spark.apache.org

Several techniques have been proposed in the literature for distributed text classification in Big Data applications which employ the aforementioned frameworks [15], [16], [17], [12]. An implementation of Naive Bayes text classification in a MapReduce model is presented in [14]. To adequately manage big data in associative classifiers (because of time complexity and memory constraints) a map reduce solution has been presented in [13]. In [10] the efficiency of Support Vector Machine and Naive Bayes classifiers is tested using a simple $Word2Vec$ model and algorithm on Apache Spark. In [11], again Apache Spark was used for common text classification and text mining techniques.

Apart from the distributed approach, graph-based techniques have been developed over the years for accurate text classification [20], [21]. Authors in [18] have employed an n-gram graph framework, which employs the neighboring n-grams as features for a text classifier. A graph-based approach has been introduced in [19] for encoding the relationships between the different terms in the text. In our work, we evaluate the n-gram graph framework in a text classification task, using the proposed distributed implementation.

The primary aim of this work is to provide a parallel and distributed implementation that boosts the scalability of the original *nGG framework*. Thus, the focus is not on the classification task itself, but on the properties that affect the scalability of the classification solution. These properties drive the redesign of existing operators (graph creation, graph merging, graph similarity calculation, etc.) in the distributed setup.

3 The N-gram Graph Framework

Before getting into the details of the proposed distributed processing implementation, it is recommended to elaborate on n-grams, the n-gram graph representation and the algorithms that make this representation applicable to a text classification task. Character or word n-gram graphs are used for representing texts. They use a sliding window that moves one character at a time as follows:
Input text: *Hello World!*
Output n-grams: *Hel,ell,llo,lo ,o W, Wo,Wor,orl,rld,ld!*

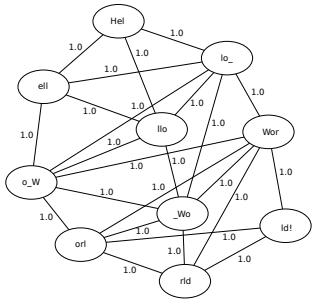


Fig. 1: The N-Gram Graph Representation of the "Hello World!" string.

The output n-grams are the graph vertices and are unique (e.g. if the trigram "Hel" is found multiple times in the text only one vertex is added to the graph). The edges of the n-gram graph connect consecutive n-grams (i.e. n-grams found within the same sliding window). For example, for a window of size 3, each trigram is connected to the next 3 trigrams to create unique undirected edges: *"Hel-ell", "Hel-llo", "Hel-lo ", "ell-llo", "ell-lo ", "ell-o W"* etc.

Each edge has a weight, which corresponds to the number of its occurrences in the text. So, a weight of 2, for the edge "Hel-ell" means that "Hel" is close to trigram "ell" (within the sliding window) twice. The respective graph for the "Hello World!" input text is shown in Figure 1.

3.1 Similarity Operators

A number of operators (e.g. merging, difference, similarity [4]) can be applied to single or pairs of n-gram graphs.Concerning similarity, ARGOT implements four different similarity operators for graph comparison.If $|G_i|$ is the size of the graph G_i (i.e. number of edges) then *Size Similarity*, compares graphs' sizes and is given by Equation 1.*Containment Similarity* expresses the proportion of edges e that graphs G_1 and G_2 share in common and is given by Equation 2. *Value Similarity* considers the weights of the common edges and the relative size of the compared graphs. For an edge $e \in G_1 \cap G_2$, the respective weights of e in graphs G_1 and G_2 are w_{1e} and w_{2e} and Value Similarity is given by Equation 3, where VR is the edge value ratio given by Equation 4. Finally, *Normalized Value Similarity* NVS, given by Equation 5, ignores the relative sizes of the compared graphs and focuses on the weights of common edges.

$$SS(G_1, G_2) = \frac{min(|G_1|, |G_2|)}{max(|G_1|, |G_2|)} \tag{1}$$

$$CS(G_1, G_2) = \frac{\sum_{e \in G_1 \cap G_2} 1}{min(|G_1|, |G_2|)} \tag{2}$$

$$VS(G_1, G_2) = \frac{\sum_{e \in G_1 \cap G_2} VR(e)}{max(|G_1|, |G_2|)} \tag{3}$$

$$VR(e) = \frac{min(w_{1e}, w_{2e})}{max(w_{1e}, w_{2e})} \tag{4}$$

$$NVS(G_1, G_2) = \frac{VS(G_1, G_2)}{SS(G_1, G_2)} \tag{5}$$

3.2 Graph Operators

Four additional binary operators are defined for the text classification task: i) union (or merge), ii) intersection, iii) inverse intersection and iv) minus or delta (all-not-in). The *Union* operator creates a new graph G_U, which contains all the edges of the two input graphs (i.e. G_1, G_2). The edge weights on G_U are

calculated using Equation 6, where L ($0 \leq L \leq 1$) is a balancing factor for edges that appear in both graphs.

$$w_{eU} = \begin{cases} w_{e1} & \text{if } e \in G_1 \text{ and } e \notin G_2 \\ w_{e2} & \text{if } e \notin G_1 \text{ and } e \in G_2 \\ L \cdot w_{e1} + (1 - L) \cdot w_{e2} & \text{if } e \in G_1 \cap G_2 \end{cases} \quad (6)$$

The *Union* operator can be defined for merging multiple graphs at a time. In this case, the merged graph has the union of all edges and new edge weights,for edges occurring in n documents, are given by Equation 7.

$$w_{eU} = \frac{\sum_{i=1}^{n} w_{ei}}{n} \quad (7)$$

The binary *intersect* operator creates a new graph G_I which contains only the common edges of the input graphs. Equation 6 is used for calculating the new edge weights. The *inverse intersect* operator creates a graph G_{II} containing the non-common edges of G_1, G_2. Finally, the *minus* (all-not-in) operator creates a graph G_M that contains the edges of G_1 that do not appear in G_2.

3.3 The Use of N-gram Graphs in Text Classification

The first step of the classification process is to train the classifier with the training instances. In the case of n-gram graphs (see Figure 2), separate class graphs are composed by merging –using the union operator– the training instances (n-gram graph representations of documents) of each class. In the classification step, each unlabeled instance (i.e. document) is first represented as n-gram graph. It is then compared with all the class graphs and the respective similarities (CS, VS and NVS) form the feature vector representation of the instance. As a result, in an N-class classification problem, each instance maps to a $3 \times N$ feature vector.

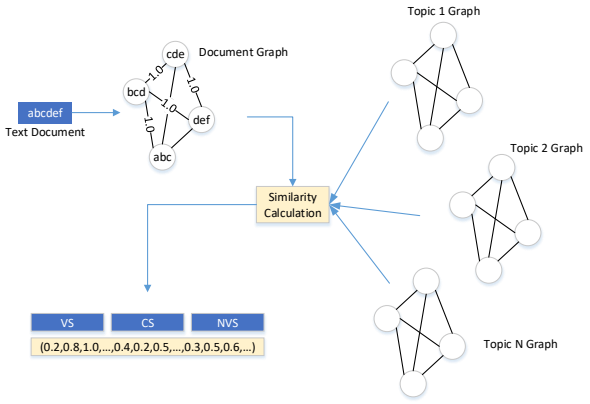


Fig. 2: Extracting the feature vector using the n-gram graph framework.

4 Distributed Implementation

Despite its performance [3], the n-gram graph classification requires a significant amount of time for training (i.e. for constructing the class graphs) and for creating the feature vectors (i.e. computing graph similarities). The time complexity of these processes depends both on the number of classes and the amount and size of training documents and sets a scalability challenge.

In order to address this challenge, we propose a distributed implementation of the classification algorithm that uses Apache Spark, a mature, fast and general engine for large-scale data processing. In this section, the distributed algorithms for the construction of the class graphs and the computation of graph similarities are explained. The code of these implementations and the framework developed, which is called ARGOT, is publicly available and can be found on a public repository[6].

Distributed class graph creation As illustrated in Figure 3, the process begins with the transformation of the training text documents to the respective n-gram graphs (nGGs). The documents, which can be located in a Hadoop (HDFS), Network (NFS) or any other distributed file system, are split into k partitions and each partition is processed independently. Each edge of the nGG consists of the connected vertex identifiers (ids) and the edge weight. The vertex ids of each edge are used as key and the edges are repartitioned based on the hash of this key. As a result, edges from different documents that correspond to the same vertex pairs are located in the same partition, thus allowing fast merging of the graphs and calculation of the new edge weights. As a result, the edges of an nGG are distributed across the processing nodes and after merging the training instances of the same class, we have a "distributed" class graph.

Distributed graph similarity computation The second step, as illustrated in Figure 4, is the extraction of similarity features, which is based on the computation of graph similarities between the class graphs and each unclassified document. The class graph can be huge (i.e., millions of edges) and so can be the time needed to compare each unclassified instance with the class graphs. To solve this problem, a graph similarity algorithm, similar to the semi-join technique used in distributed databases, has been implemented. At each comparison, the smaller document graph is broadcasted to every partition or cluster node, and the the partial overlap with the portion of the class graph in each node is computed. This implementation increases the algorithm's performance, since it reduces the communication overhead between nodes. Since the similarity measures are based on the overlapping edges only, we filter the partitions and take the edges that exist in both graphs. The resulting set of edges is too small (i.e., the number of the common edges per partition is equal or less than the number of edges of the small graph) and is collected back to the master node. The master node can then compute the similarities really fast.

[6] https://github.com/ioannis-kon/ARGOT

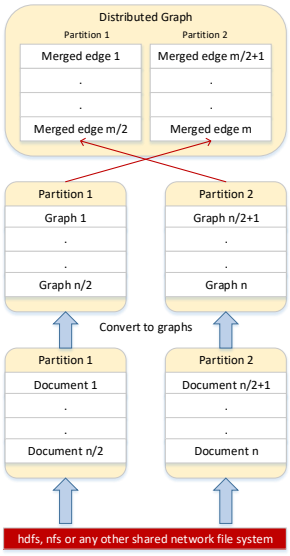


Fig. 3: The merging of multiple documents into one class-representative graph using two partitions.

5 Experimental Evaluation

The distributed implementation of the document classification algorithm was evaluated for its performance and scalability on a large-scale, real-world dataset, using two different hardware infrastructures. The experiments aim to study how several parameters (e.g. the total number of documents or classes) affect the algorithm performance.

The processing steps of the classification task, as described in [3], were repeated: i) on a single node with 24 cores (Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz) and 96 GB of RAM, running Debian 64-bit with Linux Kernel 3.16.0-4-amd64 and Java OpenJDK 1.8 and ii) on a cluster comprising 6 commodity

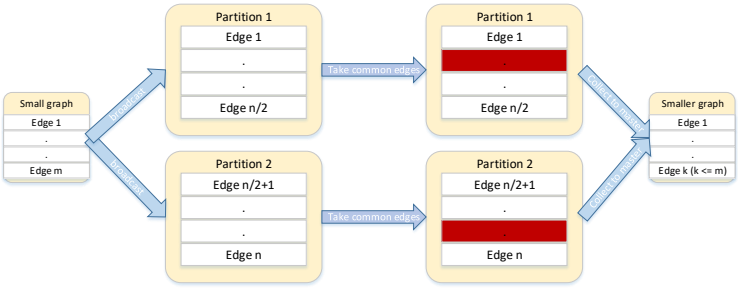


Fig. 4: Similarity computation between a small and a distributed graph.

machines, each with 4 cores (Intel(R) Core(TM) i5-3330S CPU @ 2.70GHz) and 8 GB of RAM, totaling in 24 cores and 48 GB of RAM, running OS X 10.10 (14A389) with Kernel Darwin 14.0.0 and Java OpenJDK 1.8, connected with 100-Mbit ethernet links. For the implementation of ARGOT, Scala 2.11.7 and Apache Spark 2.0.1 were used.

5.1 Dataset

The Reuters RCV2 corpus[7] is a collection of more than 480,000 news articles, categorized along a class hierarchy of 104 overlapping topics, written in 13 different languages and is very popular in text classification tasks [24], [25], [26]. In the experiments we used a subset of the Reuters corpus (only the top four, non overlapping, categories), comprising articles in four languages (see Table 1).

Table 1: Dataset statistics.

Characteristic	Value	Characteristic	Value
Classes	4	# of edges of class graph 1	5,772,038
Documents	172,115	# of edges of class graph 2	10,134,473
Total Characters ($\times 10^8$)	2.07	# of edges of class graph 3	3,708,371
Characters/Document	1,205.55	# of edges of class graph 4	8,251,410
Total Tokens ($\times 10^8$)	30.92	Total # of training instances	154,905
Tokens/Document	179.66	Total # of test instances	17,210
Av. Token Length	5.68	Total # of graph comparisons	688,460
		Total size of dataset	201.69MB
		Average size of each instance	1.2KB

5.2 Classification performance

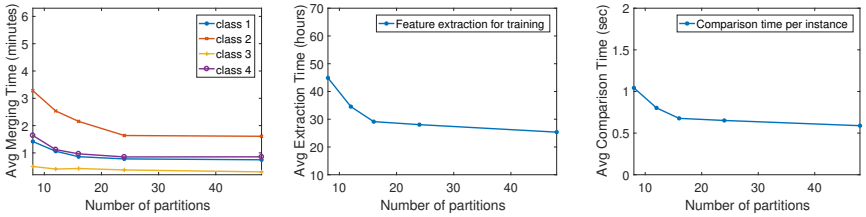
In order to evaluate the performance of ARGOT implementations, we run a full classification experiment as described in [3]. More specifically, we performed a 90%-10% training-test split of the dataset. The (four) class graphs are created by merging a randomly selected subset (90%) of the training documents. The document graphs of all documents in the training (and test) set are compared with all the class graphs using three similarity measures, thus creating a set of training (and test) instances each one comprising 12 features (3×4). Then, a Naive Bayes classifier is trained using the training instances and evaluated on the test instances The classification precision was 95.1% whereas the maximum precision reported in [23] was 94% in only a subset of the same data set, which shows that the n-gram graph approach, is language-independent and performs well in the classification task. However, this work focuses on the time performance and scalability of the distributed algorithms as discussed in the following.

5.3 Time performance evaluation

In order to test the scalability of the implemented algorithms, we conducted experiments on the complete dataset described in Section 5.1 and Table 1.

[7] http://trec.nist.gov/data/reuters/reuters.html

Number of partitions: ARGOT scalability was tested using an increasing number of partitions (from 8 to 48) on a single node machine and on a cluster of commodity machines. We repeat the experiments 10 times and report the average times. Figure 5a shows the average merging time per class, where the distributed merging function scales better for bigger classes (i.e., class 2). Figure 5b shows the average time for extracting features from the instances. It shows great scalability capacity since the algorithm reduced the extraction time by almost 50% from 8 to 48 partitions. The performance improvement is more obvious in Figure 5c, which shows the average time for extracting similarities per instance, which was reduced from 1.04 seconds to 0.58 seconds (i.e. 43% improvement), from 8 to 48 partitions.



(a) Average graph merging time per class.

(b) Average feature extraction time for training.

(c) Average comparison time per instance.

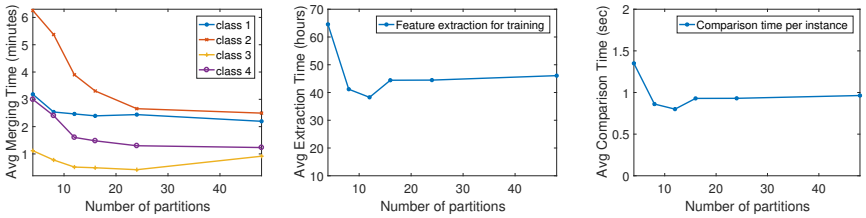
Fig. 5: ARGOT performance in a large-scale text classification task (single node).

Using the same dataset and partitions setup, we repeated the experiment on a distributed cluster[8], in order to test how data broadcasting and repartitioning affect performance. Figures 6a, 6b and 6c show the average merging time, the feature extraction time and the comparison time per instance, correspondingly. We can see that the performance drops when increasing the number of partitions (from 16 to 48) since the communication overhead is higher than the benefit from distributed processing. In larger class graphs and datasets the system is expected to perform better with more partitions. Finally, it is worth mentioning that in the single node experiment the peak memory usage was 41 GB, whereas on the cluster experiment was roughly 3 GB per machine.

Dataset size and number of classes: The second set of experiments examines how the implementation performs with an increasing total number of instances and the number of classes increases. Two subsets of the large dataset have been created, each comprising $10,000$ documents in total, from two and four classes respectively (see Table 2 for details). Figure 7a shows the number of instances per topic. Figures 7b and 7c show the number of instances per topic for the first and the second subset.

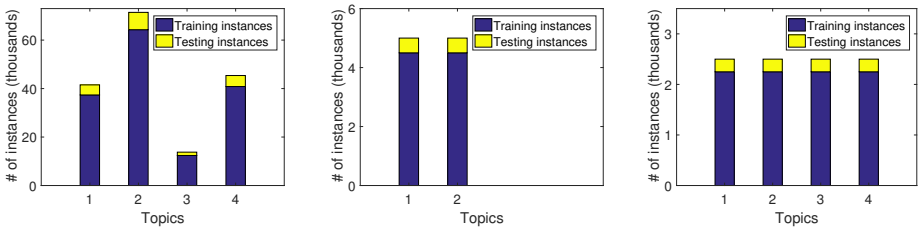
We run the same experiment with the same classification settings as before, but this time we used 2 to 24 partitions (2,4,8,12,16,24) and since these are only subsets we run them on the single node setup. Again, we repeated the experiment 10 times for each setting and took the average times.

[8] Cluster nodes are connected with 100-Mbit Ethernet links.



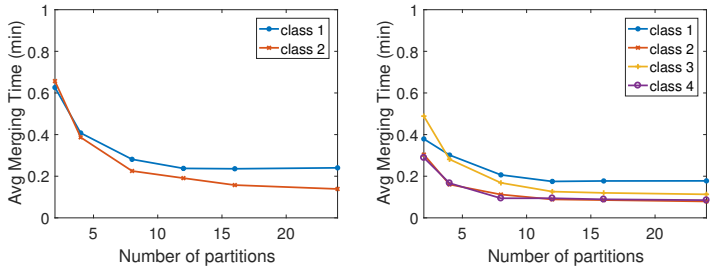
(a) Average graph merging time per class. (b) Average feature extraction time for training. (c) Average comparison time per instance.

Fig. 6: ARGOT performance in a large-scale text classification task (cluster mode).



(a) Main data set. (b) First subset. (c) Second subset.

Fig. 7: Number of instances per topic.

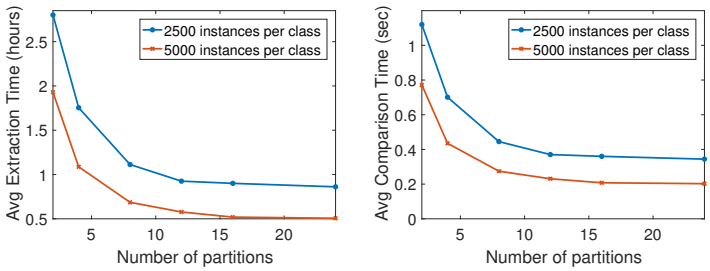


(a) 10,000 instances and two topics. (b) 10,000 instances and four topics.

Fig. 8: Average merging time per class.

Table 2: Statistics of the two subsets.

Characteristic	Subset 1	Subset 2
Total # of training instances	9,000	9,000
Total # of testing instances	1,000	1,000
Total # of graph comparisons	20,000	40,000
Average size of each instance	1.028KB	1.157KB



(a) Avg feature extraction time. (b) Avg comparison time.

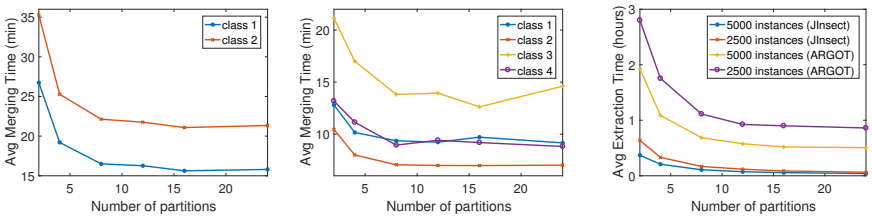
Fig. 9: Avg feature extraction time and avg comparison time per experiment.

Figures 8a and 8b show the average merging time per class per experiment. From these two figures we can conclude that the merging time of the graphs depends on the number of the documents in a topic. Figure 9a depicts the average feature extraction time per experiment. While having less documents per topic and the same number of total training instances the extraction time took longer. The difference is the number of graph comparisons, which are greater in the last case. From this we can infer that the feature extraction time depends on the number of topics. Finally, Figure 9b shows the average comparison time per instance. From the experiments conducted we can safely say that the more documents we have per class and the larger graphs we have, the scalability of the algorithms is great.

5.4 Time performance evaluation of multi-threaded implementation

To better evaluate ARGOT, we compared our newly developed framework with the current implementation of n-gram graph algorithms, JInsect. JInsect is not scalable to many machines but it can use all available processing threads of a computer (multi-threaded execution). First, we run JInsect using the two aforementioned subsets in the virtual machine. Figures 10a and 10b show the average merging time of graphs in the first subset and in the second subset, correspondingly. Figure 10c shows the average feature extraction time in both subsets for both (distributed and multi-threaded) implementations. We can see that ARGOT is faster in graph merging but slower in feature extraction. To further evaluate ARGOT against the current implementation, we run JInsect using the

large dataset. This time JInsect utilized the entire memory of the virtual machine, plus swap memory and did not manage to finish the experiment. From these experimental results we can infer that ARGOT can not only scale to multiple machines, but it can also handle larger data sets in contrast to the multi-threaded, single machine implementation.



(a) Average merging time (two topics). (b) Average merging time (four topics). (c) Average feature extraction time.

Fig. 10: JInsect (i.e. multi-threaded, single-machine) performance evaluation.

Based on the above, we can safely conclude that ARGOT is a prerequisite to apply n-gram graphs to a large scale setting, overcoming single-machine limitations. Finally, we should note that ARGOT yields the same results as JInsect in terms of classification performance.

6 Conclusions

In this work we presented ARGOT, a distributed implementation of the n-gram graph algorithms in text classification. We illustrated the details of the distributed implementation and demonstrated the scalability of ARGOT on a real-world, large-scale, multilingual dataset. We showed how the number of topics and the number of instances can affect the performance of the algorithms. We also presented the effect the broadcasting of data can have on the execution time of the experiment, when using a cluster of computers. We showed that if the number of instances per topic is large, thus the corresponding class graphs are huge, ARGOT can benefit from the larger number of CPUs in a cluster/machine. We also compared ARGOT to the established implementation of n-gram graphs and showed that ARGOT scales better when using big data in contrast to JInsect which did not manage to finish the experiment using the large data set. As a side-effect we demonstrated state-of-the-art performance on an established classification dataset for a single-label, multi-class and multilingual setting, though this is not the purpose of this paper.

References

1. S. Kinsella, A. Passant and J. G. Breslin. Topic Classification in Social Media Using Metadata from Hyperlinked Objects. In *Proceedings of 33rd European Conference on IR Research*, pages 201–206, 2011.
2. A. Kosmopoulos, G. Paliouras and I. Androutsopoulos. Adaptive spam filtering using only naive bayes text classifiers. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, pages 1–2, 2008.
3. G. Giannakopoulos, P. Mavridi, G. Paliouras, G. Papadakis and K. Tserpes. Representation models for text classification: a comparative analysis over three web document types. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, pages 9–12, 2012.
4. G. Giannakopoulos. Automatic summarization from multiple documents. *Department of Information and Communication Systems Engineering, University of the Aegean*, pages 49–66, 2009.
5. G. Giannakopoulos, V. Karkaletsis, G. Vouros and P. Stamatopoulos. Summarization system evaluation revisited: N-gram graphs. *ACM Trans. Speech Lang. Process.*, Vol. 5, pages 1–39, 2008.
6. Bifet, A., Maniu, S., Qian, J., Tian, G., He, C., and Fan, W. StreamDM: Advanced Data Mining in Spark Streaming. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference*, pages 1608–1611, 2015.
7. Fei, X., Li, X., and Shen, C. Parallelized text classification algorithm for processing large scale TCM clinical data with MapReduce. In *Information and Automation, 2015 IEEE International Conference*, pages 1983–1986, 2015.
8. Lu, G., Xia, Y., Wang, J., and Yang, Z. Research on Text Classification Based on TextRank. In *International Conference on Communications, Information Management and Network Security* 2016.
9. Han, R., Lu, X., and Xu, J. On big data benchmarking. *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware, Springer International Publishing*, pages 3–18, 2014.
10. Choi, M., Jin, R., and Chung, T. S. Document Classification Using Word2Vec and Chi-square on Apache Spark. In *International Conference on Computer Science and its Applications*, pages 867–872, 2016.
11. Ye, Z., Tafti, A. P., He, K. Y., Wang, K., and He, M. M. SparkText: Biomedical Text Mining on Big Data Framework. PloS one, 11(9), e0162721, 2016.
12. Semberecki, P., and Maciejewski, H. Distributed Classification of Text Documents on Apache Spark Platform. In *International Conference on Artificial Intelligence and Soft Computing*, pages 621–630, 2016.
13. Bechini, A., Marcelloni, F., and Segatori, A. A MapReduce solution for associative classification of big data. Information Sciences, 332, pages 33–55, 2016.
14. Zhou, L., Yu, Z., Lin, J., Zhu, S., Shi, W., Zhou, H., ... and Zeng, X. Acceleration of Naive-Bayes algorithm on multicore processor for massive text classification. In *Integrated Circuits (ISIC), 2014 14th International Symposium, IEEE*, pages 344–347, 2014.
15. Xu, K., Wen, C., Yuan, Q., He, X., and Tie, J. A MapReduce based parallel SVM for email classification. *Journal of Networks, 9(6)*, pages 1640–1648, 2014.
16. Santoso, J., Yuniarno, E. M., and Hariadi, M. Large Scale Text Classification Using Map Reduce and Naive Bayes Algorithm for Domain Specified Ontology Building. In *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference, IEEE*, Vol. 1, pages 428–432, 2015.

17. Gupta, A. Learning Apache Mahout Classification. Packt Publishing Ltd., 2015.
18. Angelova, R., and Weikum, G. Graph-based text classification: learn from your neighbors. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, ACM*, pages 485–492, 2006.
19. Malliaros, F. D., and Skianis, K. Graph-based term weighting for text categorization. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference*, pages 1473–1479, 2015.
20. Das, N., Ghosh, S., Gonalves, T., and Quaresma, P. Comparison of different graph distance metrics for semantic text based classification. Polibits, (49), pages 51–58, 2014.
21. Dasondi, V., Pathak, M., and Singh, N. P. An implementation of graph based text classification technique for social media. In *Colossal Data Analysis and Networking (CDAN), Symposium, IEEE*, pages 1–7, 2016.
22. Giannakopoulos, G. and Karkaletsis, V. N-gram Graphs: Representing Documents and Document Sets in Summary System Evaluation. In TAC 2009.
23. Fortuna, B. and Shawe-taylor, J. The use of machine translation tools for cross-lingual text mining. 2005.
24. Dasqupta, A., Drineas, P., Harb, B., Josifovski, V. and Mahoney M.W. Feature selection methods for text classification. In *Proceedings of the 13th ACM SIGKDD international*, pages 230–239, 2007.
25. Song, Y., Upadhyay, S., Peng, H. and Roth, D. Cross-lingual dataless classification for many languages. IJCAI International Joint Conference on Artificial Intelligence, pages 2901–2907, 2016.
26. Ferreira, D.C., Martins, A.F.T. and Almeida, M.S.C. Jointly Learning to Embed and Predict with Multiple Languages. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 230–239, 2016.