Computational Models of Argument
C. Reed et al. (Eds.)

doi:10.3233/FAIA240306

Review Argumentation at Scale

Davide CEOLIN [a,1] and Laura Sofie OOTES [b]
[a] *Centrum Wiskunde & Informatica, Amsterdam, The Netherlands*
[b] *Netherlands eScience Center, Amsterdam, The Netherlands*
ORCiD ID: Davide Ceolin https://orcid.org/0000-0002-3357-9130, Laura Sofie Ootes https://orcid.org/0000-0002-2800-8309

Abstract. Product reviews represent a valuable source of information for both (potential) customers and sellers. Usually, reviews come in pairs (score, motivation), where the motivation is a piece of unstructured text explaining the score given to a product. For reviews, this setting is ideal to combine a quantitative assessment of a product with a qualitative explanation. Aggregating the numerical scores might be uninformative while parsing large quantities of text might be challenging.

Automated argument analysis can help in this process, and we previously developed an argument-based quality analysis pipeline that helps identify the most significant items from a corpus of reviews. Given that the pipeline is effective but time-consuming, this work sets out to improve its computational efficiency. Next to optimisation by conventional methods, we investigate the effect of reducing the number of text chunks that are used to build the argumentation graph.

We find that conventional methods significantly improve the computation time, which allows us to analyse much larger datasets of real-world reviews. When the number of tokens is scaled down, accuracy remains similar compared to the original version of the pipeline. However, we find that this does not necessarily result in a computation time reduction.

Keywords. Product Reviews, Computational Argumentation Pipelines, Scalability

1. Introduction

Product reviews are a valuable source of information both for customers and sellers. By sharing their opinions online, users allow others to acquire information about products and services without having to experience them first-hand. Reviews often come in pairs of a numerical score, which we consider to be a claim about the product's quality, and a textual description containing arguments supporting the score given. This formulation allows reviewers to express their opinions naturally but complicates the analysis of the reviews' quality. One limitation of product reviews is that, since they are subjective, their quality is diverse. The informal definition of quality is fitness for purpose, but then specific fitness function and the purpose are rather subjective. Reviewers could use the same score to highlight different issues regarding the same product or could judge the severity (and importance) of the same issue on different scales. This problem has been addressed in different manners, but the most popular solutions focus on the quantitative part of re-

[1] Corresponding Author: Davide Ceolin, davide.ceolin@cwi.nl

views: aggregating the scores of multiple reviewers and weighing their reliability on the number of reviews received [1,2]. While this is a reasonable approach to try to leverage the wisdom of the crowds, it cannot discern the quality of reviews. This quantitative approach assumes that for a large enough set of reviews, the resulting average would be representative of the quality of the product. However, the quality of such reviews can significantly vary the size of this 'ideally large' set: a small set of high-quality reviews might be good enough; a large dataset of low-quality ones might not. Analyzing the arguments in product reviews can provide insight into their quality, as we demonstrated in previous work [3,4].

Argument mining and reasoning is a time-consuming effort. In this work, we investigate how to improve the computational efficiency of the previously introduced argument-based information quality assessment pipeline. Two methods are considered that we refer to as lossless and lossy optimization. For lossless optimization, we analyze to what extent the performance of the pipeline can be improved using conventional methods that do not affect the created argumentation graph itself (**Q1**). We compare computation times to the original pipeline and test if lossless optimization allows for analyzing a larger set of reviews. Secondly, we investigate the use of lossy optimization, in which - in addition to the adjustments made for lossless optimization - the number of tokens (phrase chunks) is reduced by introducing a threshold to prioritize more informative tokens. This selection method affects the constructed argumentation graph and we investigate if this improves the computation time without compromising accuracy (**Q2**). The focus of using the lossy method lies in the possibility of extending the validation of the pipeline by scaling it up to implement specific improvements in the future.

The rest of the paper is structured as follows. Section 2 introduces related work. Section 3 describes the methods proposed. Section 4 describes the datasets analyzed. Section 5 presents and discusses the results obtained, and Section 6 concludes.

2. Related Work

The use of pipelines is a common approach in Natural Language Processing. Frameworks like Stanford CoreNLP [5], GATE [6], and more recent solutions like Spacy [7] and DeepNL [8] combine a sequence of components to incrementally extrapolate (structured) information from text. The focus of these pipelines is on their ability to extract information from text. The scalability of NLP pipelines is instead more central to works like [9] and [10]. Focusing on argument pipelines, Lenz et al. [11] propose a pipeline that is complementary to ours: while we mine and reason on arguments, with their web service-based pipeline they also allow creating and visualizing arguments. Snaith et al. [12], and Wang et al.[13] propose solutions for scaling up argument pipelines. These approaches were a source of inspiration for our work, but they focus on argument mining. We cover both the mining and reasoning part, and we specialize our pipeline on product reviews. Regarding the analysis of arguments in reviews, our previous work introduces the pipelines that are scaled up here [3,4]. There we implement the theoretical framework by Baroni et al. [14]. They provide effective representation solutions to handle the combinatorial effects of their algorithm, while we address its computational consequences. Aakhus et al. [15] also provide a theoretical argumentation framework but their focus is on representation only. Relevant to this topic are also the works of Schneider and

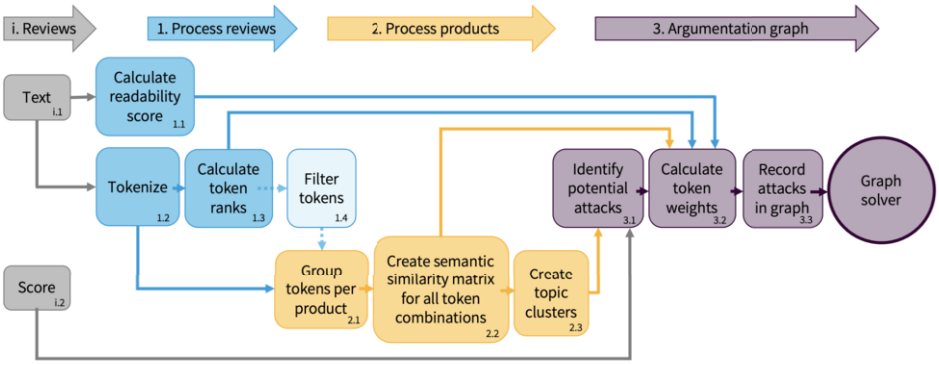

Figure 1. Pipeline overview. The input consists of the review texts and scores, and arrows indicate where the data enters the pipeline (i, grey). We identify three phases: 1. processing individual reviews (blue), 2. processing reviews per product (orange), and 3. constructing and solving the argumentation graph (purple). The steps per phase are indicated by numbered blocks, and the arrows show how the output of one step is used as input for the next steps. Note that step 1.4, filtering of tokens, is only applied in lossy computations.

Wyner [16], Gabbriellini and Santini [17], and Anang and Masaioshi [18], who study the mining of arguments from reviews and differ from our approach in that we focus on the reasoning of such arguments. In another of their works, Gabbriellini and Santini [19] align with Dung's abstract argumentation framework [20] to reason on the arguments identified, although their study is limited in size compared to ours. Schlosser [21] shows that the perceived usefulness of reviews does not depend on a balance between positive and negative comments, thus stressing the importance of in-depth argument analysis.

3. Method

The argumentation pipeline[2] was previously introduced [3,4] in terms of argumentation graph construction and argumentation reasoning; we recall it below to explain the proposed improvements. The argumentation graph is constructed from the outputs of the analysis of individual reviews and that of all reviews per product. Figure 1 shows the computational overview of the pipeline in which we separate the review analysis (1. *process reviews*) and product analysis (2. *process products*) from the argumentation graph construction and reasoning (collectively referred to as 3. *argument graph* phase).

3.1. Pipeline

Argument Graph Construction The input of the pipeline is a set of reviews regarding several products. The goal of the argument graph construction component is to identify a graph for each product representing arguments implying attacks (or potentially supports) between reviews. Attacks between reviews imply disagreement, and the goal of the pipeline is to identify the most reliable review for each product. The graph for each product resulting from this component is compliant with the framework of Baroni et al. [14]. This means that it will explicitly encode attacks (edges) between reviews (nodes), while

[2]See: https://github.com/davideceolin/FAReviews

supports are only implicitly derived: according to the framework of Caminada and Baroni, reviews/nodes support other reviews/nodes when they attack their attackers.

We do not specifically look for arguments in the pipeline, but we treat each phrase chunk in a review as potentially (part of) an argument: a review text explains the review score. The higher the token centrality is in the text (as indicated by the TextRank measure [22]), the higher the chances that the token represents an argument explaining the score. Any pair of reviews presenting a different review score will then potentially attack each other. To avoid obtaining an unbearable number of edges in our graph we apply argument (chunk) clustering and argument weighing. Two reviews attack each other when they present two arguments that are semantically close enough. To this aim, we cluster all the tokens identified for one product. When two tokens belong to the same cluster, and the reviews have a different review score, we compute a weight for each of them, taking into account the importance of the token, i.e., their TextRank value, in the review and the readability of the review (which represents one of the aspects of quality of a review). We then record only attacks that go from the token with a higher weight to the lower one. Computationally, the following steps are taken (see Figure 1). We denote each step as per their number in this figure. The pipeline takes in a data file containing reviews for several products (i). Each review contains a review text (i.0) and a review score (i.1) for the product. First, the individual reviews are analyzed. For each review text a text readability score[3] is calculated (1.1) using a Spacy pipeline[4] [7]. Each review text is tokenized into chunks (1.2), and the tokens are given a rank score (1.3) using PyTextRank [23]. This package uses the TextRank measure, indicating the importance of a token (how informative it is) within the text. In the case of the proposed lossy method, tokens are then filtered based on their TextRank measure (1.4). Then, given all tokens of a product (2.1), the semantic distance between each token combination is calculated using the Word Mover's Distance [24]. A matrix of Word Mover's Distances between the different tokens of a product is built (2.2) using the gensim wmdistance package [25]. Based on the Word Mover's Distances, topic clusters are determined and each token is assigned to a cluster (2.3). Clustering is done using K-Means [26]. Based on a tokens' cluster in combination with the score of the review (i.2) it belongs to, potential attacks are identified (3.1). The review text readability (1.1), token importance (rank, 1.2), and semantic distances between tokens (2.2) are used for argument weighing between the potential attacks (3.2). The computed information of each product is recorded in the argumentation graph (3.3).

Argument Reasoning Once an argument graph is constructed, we apply the framework of Baroni et al. to identify which reviews to accept (*in*) or reject (*out*). A review that cannot be accepted nor rejected, will be classified as undecided (*undec*). This framework is implemented using a Prolog solver [27] (*graph solver* in Figure1).

3.2. Scalability Improvements

In this section, the methods proposed to optimize the computational performance of the pipeline are described. We profiled the original code to identify its computational bottleneck. The *process products* and *argument graph* were found to take significantly longer than the *process reviews* phase, and the number of reviews per product plays a significant

[3] See: https://github.com/mholtzscher/spacy_readability

[4] We used the en_core_web_md model, available at https://github.com/explosion/spacy-models

role in computation time. In terms of individual components of the code, the semantic distance matrix calculation was found to take up significant execution time. To address the bottlenecks we propose to apply the most apparent methods for time improvement.

Q1 - Parallelization The datasets analyzed contain reviews regarding multiple products. The pipeline we adopt takes as input all reviews regarding each product at the time. The previous version of the code already exploited this peculiarity to parallelize part of the pipeline. In the *process products* phase, the creation of the token matrix and topic clusters (steps 2.2 and 2.3 in Figure 1) was already executed parallel on a product level.

In this work, we apply parallelization also in the *process reviews* phase of the pipeline, in which the analyses are done on the review level. We use Python's multiprocessing package to create a pool of t workers that each analyze a chunk of the given reviews in parallel. Since no data is shared between reviews, we consider this an effective method to improve performance. In the *argumentation graph* phase parallelization is also introduced (purple in Figure1). In this phase, the reviews are processed and combined per product to build and solve the product graph. Since (the data of) different products are fully independent from one another, parallelization on a product level is suitable here. The parallelization is implemented in the Python code that identifies potential attacks, calculates token weights, builds the graphs, and then calls the Prolog server to solve the graph. We use here the ability of the Prolog server package to receive multiple requests concurrently. In this work, we use for all parallel processes 12 CPU cores.

Q1 - Work Load Distribution Although parallelization was already applied in the *process products* phase of the pipeline, we further optimize the process in this phase by considering how work is divided among workers. A function calculating the matrix and clusters (2.2 in Figure 1) for a product is applied to a Python Pandas [28] DataFrame in which each row contains a product identifier. This dataframe is split up into n_w sub-dataframes, where n_w indicates the number of workers. Each worker then applies this function to its assigned sub-dataframe. The function (which takes in the full set of reviews), calculates for the given product the corresponding matrix and clusters based on all tokens of the reviews of that product. Since each product has a different amount of reviews, and the number of tokens per review depends on the length of the review text, the calculation time for each product is different. To fully optimize the parallelization in this phase, one has to consider how the product dataframe is split. In this work, we take the total number of tokens per product into account and split the products such that each worker receives a similar dataset in terms of the products with n_t unique number of tokens. That is, we give each worker a similar amount of large and small products to distribute the load (reasonably) equally over the different workers.

Q1 - Matrix Optimization In the *process products* phase of the pipeline, we build for each product a matrix with the Word Mover's Distance between each pair of tokens identified in the reviews and use this matrix to create topic clusters (steps 2.2 and 2.3). The computation time of the matrix is dependent on its size, which is set by the total unique number of tokens for a product. Since the Word Mover's Distance is calculated between the pairs of tokens, the matrix has n_t^2 number of elements. However, this matrix is symmetric, as the Word Mover's Distance between tokens $n_{t,1}$ and $n_{t,2}$ is equal to that of $n_{t,2}$ and $n_{t,1}$. We can therefore improve the matrix creation process by leveraging its symmetry, reducing the number of elements to be calculated to $(n_t^2 - n_t)/2$.

Table 1. Properties of the different data files analyzed.

Dataset	N_{reviews}	N_{products}	$\max\left(\frac{N_{\text{reviews}}}{\text{product}}\right)$	$\max\left(\frac{N_{\text{tokens}}}{\text{product}}\right)$
Amazon Fashion 5	3,108	31	377	2,342
Luxury & Beauty 5	30,313	1,581	769	8,865
Industrial & Scientific 5	72,968	5,334	910	7,520
Amazon Fashion	883,636	186,189	4,384	36,525

Q1 - Graph Construction Optimization; dataframe handling Reviews are handled using Pythons' Pandas DataFrames. In the *argumentation graph* phase of the pipeline, we identified that (in addition to parallelization) we could increase the performance by leveraging Pandas inherent optimizations. Panda is optimized for vectorized operations on full columns in its dataframes. Instead of iterating over rows, which introduces significant overhead, in the new version of the code operations are performed on entire columns in the dataframes. Given the dataframe sizes, such a relatively small change could have an impact on the total execution time.

Q2 - Token Centrality Thresholding One assumption of our pipeline is that the TextRank centrality of textual tokens is an indicator of their importance in the review and of their strength as arguments. This means that the lowest-ranked tokens are potentially irrelevant. We, therefore, introduce a threshold for token centrality: tokens below that threshold are ignored from *process reviews* phase onwards ('Filter tokens' in Figure 1). This directly affects the size of the similarity matrix and the number of nodes in the argument graph. Additionally, clustering will be based on fewer tokens, which may affect the identified topic clusters and the identified potential attacks. This may affect the computation time, but can also affect the outcome of which is the most relevant argument.

We test the system performance under several different values for such threshold to observe the possible computational gain and monitor potential performance loss in terms of the accuracy of the computation. This is the only lossy improvement we introduce.

4. Dataset Description

We evaluate the performance of the pipeline over three datasets that are part of the datasets published by McAuley et al. [29,30]. We use the Amazon Fashion 5, Luxury and Beauty 5, and Industrial and Scientific 5 datasets composed of reviews of products belonging to three different categories (referred to as AF5, LB5, and IS5 respectively from here on). These so-called 5-core data sets are a subset of the full data in which users and items have at least 5 reviews each. We additionally analyze the full Amazon Fashion dataset (from hereon AF), to which these constraints are not applied. Each dataset contains both the textual description and numerical score of the reviews, as well as metadata (product ID, Reviewer ID, review timestamp, number of upvotes). Reviews can only receive positive feedback from the readers. Figure 2 show statistics of the datasets, after removing duplicate reviews. For the 5-core data sets, this causes a number of products to only have one review, despite the properties of the original datasets. The distribution of the number of reviews per product varies, and while the AF5 dataset is split in two (about half of the product received a low number of reviews, the other half hundreds of reviews), the other two datasets follow a similar long-tail distribution. The number of to-

Figure 2. Top: distribution of the number of reviews per product for the different datasets. Bottom: distribution of the number of reviews per product for the different datasets.

kens per review follows a bell-shaped distribution for the LB5 and the IS5 datasets, while the AF5 dataset is less regular, likely because of its limited size. The AF5 is particularly small compared to the other two datasets, and the whole AF dataset (see Table 1). The two smallest datasets, AF5 and LB5, are selected to test the lossless optimization method against the baseline that is the original script. These two datasets are significantly different from each other in terms of the number of reviews, number of products, and maximum number of reviews per product. The results for the lossless optimization method are also obtained for the other datasets, but not tested against the original code, because of the sheer computational resources that would be required. We use the AF5, LB5, and IS5 datasets to test the lossy optimization method as a baseline for the lossless code.

5. Results

Here the results obtained are presented, both in terms of computational time and accuracy. All results were obtained by running the analysis on DAS-6 (The Distributed ASCI Supercomputer 6) [31], on one of the standard nodes. We used 12 CPUs, creating a pool of 12 multiprocessing workers for each script. We discuss the results below.

5.1. Computation Time

Lossless Improvements Table 2 shows the computational gain obtained by introducing lossless optimization for the AF5 and LB5 datasets. All components benefit from these improvements in terms of computation time, with otimization factors of 6.9 and 20.9 respectively. For both datasets, a computational time improvement (5.7 and 12.0) is obtained for the *process reviews* phase, where parallelization was introduced. The absolute time of this phase is small in comparison to the analysis time for the full pipeline though.

Table 2. Comparison of the run times of the original scripts and new, lossless optimized scripts for the AF5 and LB5 datasets. The run times and optimization factors are given for the individual scripts and for the total. The comparison was made using 12 CPUs in the original and new scripts.

Dataset	Script	Method	t_{original} (s)	t_{new} (s)	Opt. factor
AF 5	Total		2,366	341	6.9
	Process reviews	parallelization	17	3	5.7
	Process products	work load distribution, matrix optimization	935	254	3.7
	Argumentation graph	parallelization, dataframe handling	1,414	82	17.2
LB 5	Total		199,724	9,569	20.9
	Process reviews	parallelization	456	38	12.0
	Process products	work load distribution, matrix optimization	8,486	4,656	1.8
	Argumentation graph	parallelization, dataframe handling	190,782	4,874	39.1

Table 3. Properties and run times of a sample splits of the AF dataset. To reduce memory usage, the dataset was split into 18 different sets run separately. The splits each contain reviews of 10,343 or 10,344 different products. For each split, the properties of the products and reviews are given, the total run time, and run times per phase; process reviews (t_pr), process products (t_pp), and argumentation graph (t_ag).

Split ID	N_{reviews}	t_{total} (s)	t_{pr}(s)	t_{pp} (s)	t_{ag} (s)	$\max\left(\frac{N_{\text{reviews}}}{\text{product}}\right)$	$\max\left(\frac{N_{\text{tokens}}}{\text{product}}\right)$
0	51,534	32,737	62	9,240	23,433	4,384	36,525
5	49,205	23,179	61	3,051	20,065	2,206	18,998
10	48,647	23,647	59	1,659	21,927	1,680	12,565
15	47,364	20,807	58	1,663	19,085	1,418	8,915
Total	883,636	411,983	1,077	65,116	345,759		

The optimization factors for the *process products* phase (3.7 and 1.8), obtained by matrix optimization and improved workload distribution, are relatively small compared to the overall optimization factor. Yet it does have a significant effect on the pipeline since this phase takes up a large fraction of the total computation time. The optimization for the (argumentation graph) phase, based on parallelization and improved dataframe handling, is the largest; a factor of 17.2 for the AF5 dataset, and 39.1 for LB5. The optimization of this phase provides another significant contribution to overall time improvement.

Next, the optimized code is used to analyze the full AF dataset, containing 883,636 reviews of 186,180 products. To make this larger dataset manageable in terms of memory usage, we split it into 18 batches of 10,343 or 10,344 products, resulting in batches of 50,000 reviews each. The different batches do have a large difference in the maximum number of reviews per product and the maximum number of tokens per product. Table 3 shows the computation time obtained. The execution time of the batches takes $4.8 - 9.3$ hours each. Comparing the execution times for the different batches shows that while most time is spent on the *argumentation graph* phase of the pipeline, there is a large spread in the duration of the *process products* phase.

Lossy Improvement The computation times for the AF5, LB5, and IS5 datasets with lossy improvement, i.e., with a TextRank threshold are shown in Figure 3 (left axis). The threshold reduces the number of tokens selected from each review in the *process reviews* phase of the pipeline for further analysis. The goal of these graphs is to observe

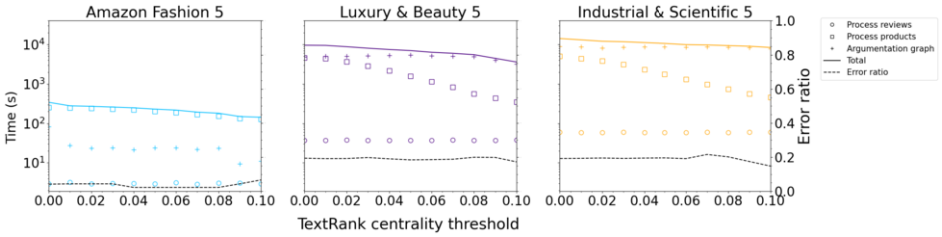

Figure 3. Computation time of each component of the pipeline for the AF5, LB5, and IS5 datasets as a function of the TextRank threshold (left axis). The right axis shows the error rate (expressed as the ratio of upvotes of the reviews classified as 'out' over the total count of upvotes) for different values for the TextRank threshold.

the execution time when different thresholds for the TextRank centrality measure are set. When the threshold is set to 0, the execution time coincides with that reported in Table 2.

The threshold is applied at the end of the *process reviews* phase (step 1.4 in Figure1), and hence only affects the computations from thereon. For all data sets, a decline in computation time for the *process products* phase for increased TextRank threshold can be observed, accompanied by a similar trend for the total computation time. The *argumentation graph* phase takes up a significant part of the overall computation time for the LB5 and IS5 datasets. For the AF5 dataset, the computation time is larger if there is no threshold (i.e. for threshold 0.0), and somewhat smaller for thresholds of 0.09 and 0.10 while being relatively stable for thresholds in between. For the LB5 dataset, this phase shows some variation in computation time for TextRank thresholds between 0.0 and 0.8, and then steeply declines. For the IS5 dataset, the computation time is relatively stable with increasing TextRank threshold during this full phase.

Discussion - Computation Time We found that lossless optimization of the different components of the pipeline has led to significant improvement in the computation times. For both the AF5 and LB5 datasets a moderate improvement is obtained for the *process reviews* phase, consisting of the readability score computation, tokenization, and ranking (Figure 1, steps 1.1-1.3). We optimized by parallelized processing of batches of reviews. In this work, we used 12 CPU cores for the calculations, which matches the optimization factor for the LB5 dataset. For the AF5 dataset the optimization factor is smaller, which can be explained by the relatively large overhead for the small amount of data to analyze.

The optimization factors for the *process products* phase (Figure 1, steps 2.1-2.3) are relatively small. This phase of the code was previously parallelized on product level, hence the number of cores used does not affect the optimization factor. We tried to optimize the parallel computations on this full phase by improving the workload distribution. Further time improvement was obtained from optimizing the calculation of the token matrix of a product. The matrix calculation time depends on the size of the matrix and hence the number of tokens from all reviews of a product. An improvement of a factor 2 can be obtained for the matrix calculation (section 3.2). Considering that the matrix calculation is only part of this phase, the obtained optimization factors of 1.8 - 3.7 for this full phase (from both optimization methods) are in line with the expectations.

The optimization factors for the *argumentation graph* phase are the largest among all phases, which has a big impact on the overall analysis time improvement. The improvement factor is larger than the number of workers (12) used for the parallelization, showing that the improved dataframe handling also plays a significant role. This espe-

Table 4. Count of the number of upvotes received by the reviews that the pipeline classifies as accepted (in), rejected (out), or undecided (undec).

Dataset	in	out	undec	Error
Amazon Fashion 5	3	70	1,591	4%
Amazon Fashion	51,117	88,641	323,457	19%

cially affects the LB5 dataset. It has products with large amounts of reviews per product, resulting in bigger dataframes, so the efficient use of these becomes more important.

The analysis of the full AF dataset shows a difference of almost a factor 2 between the different batches it was split. The batches are balanced in the number of products and moderately in terms of the total number of reviews, but not based on the number of reviews and tokens per product. Despite the improvement in computation time, analyzing such a large data set still takes up considerable time. The computation time can be further reduced by using more CPUs, such that more products can be analyzed in parallel. Even more computational gain could be obtained when transforming the code to offload computations to a GPU, especially considering the parallelizability of this code. However, one should consider the availability of GPU resources, the cost of a GPU and its energy use, and the amount of CO_2 emission.

Thresholding (taking effect after the *process reviews* phase) reduces the computation time of the *process products* with increasing threshold. This is as expected since the number of tokens that need to be processed reduces. This affects the computation time for the Word Mover distances matrix (step 2.4), as the number of elements to be calculated is smaller. The use of a threshold also yields an unexpected result. There is no clear trend in computation time for the *argumentation graph* phase over the inspected TextRank threshold range $0.0 - 0.1$. For the AF5 dataset, there is only a significant decline between thresholds $0.0 - 0.01$, and $0.08 - 0.09$. For the LB5 dataset there is a decline between $0.08 - 0.10$. Outside these ranges and for the full threshold range for the IS5 dataset the computation times are relatively stable. Inspection of graph solver computation times for a few individual products shows that there is no obvious relation between the number of tokens for a specific product, based on the TextRank threshold. Hence this behavior is not due to a particular configuration of the dataset, but to the topology of the argumentation graph resulting from ignoring the tokens with low TextRank centrality: despite being less connected, the resulting graph could be harder to solve. Overall, introducing a threshold for the TextRank centrality of tokens can be computationally beneficial, but this is not guaranteed. This strategy requires careful calibration.

5.2. Prediction Accuracy

Lossless Prediction For the lossless method, we focus on the use of the optimized code for the analysis of large review datasets and inspect the accuracy based on these. Table 4 shows for the AF5 and AF data sets the number of upvotes received by the reviews that our pipeline classifies as accepted (in), rejected (out), or undecided (undec). Since the pipeline classifies as undecided reviews by default, unless they attack or they are attacked, we compute the error as the percentage of upvotes of reviews classified. For the larger AF5 dataset, the error rate is 4% while 19% for the AF data.

Lossy Prediction For the lossy method (using the TextRank threshold) we test the accuracy of the three smaller datasets to check whether its consequences are domain-dependent. The right axis of figure 3 shows relatively stable error rates for all thresholds.

Discussion - Prediction Accuracy The code optimization described above allows us to analyze the whole AF dataset, which is about 250 times larger than the dataset we analyzed before. In the larger dataset, the error is significantly higher than in the smaller one. However, the error rate of the larger dataset is in line with that of the datasets from other domains (Figure 3). Here, we do not evaluate the performance of the pipeline per se but explore the possibility of scaling it up to provide a deeper evaluation and these results demonstrate this possibility. Concerning lossy improvements, we observe no significant accuracy loss (Figure 3). This aspect is negligible when deciding whether to implement this improvement, at least based on the three datasets analyzed.

6. Conclusion

We investigate the possibility of scaling up an existing argument-based pipeline to predict the quality of product reviews. We study two questions: **Q1** focuses on the significance of improvements derived from lossless optimization, and **Q2** on the suitability of lossy optimization as a safe means to improve performance. Regarding **Q1**, thanks to the lossless optimizations we could analyze an extensive dataset of reviews (about 250 times larger than the dataset originally tested), and datasets from two additional domains (IS5 and LB5). We observed that the error rate of the larger AF dataset is higher than that of the original AF5 dataset. However, the error rates of the IS5 and the LB5 datasets are similar to that. So, it is the original AF5 dataset that yields a particularly favorable performance. In the future, we plan to study the impact of the choice of different components on these results, e.g., by testing different readability measures. Concerning **Q2**, we must be more careful. Based on the datasets we analyzed, the introduction of a threshold for the TextRank value is safe, i.e., it does not cause a significant loss in accuracy. The gain in computation time is more subtle; for part of the pipeline, the threshold introduces a significant gain in computation speed. However, when solving the argumentation graph we obtain an improvement only in some cases. As such, to safely use this improvement, one should first test its consequences on the specific dataset considered. In the future, we will investigate the precise reason for the possible increase in computation time. Also, we intend to evaluate the introduction of other threshold types, e.g., on the readability value. This implies testing different values for such thresholds and checking the behavior of different metrics concerning thresholding. Lastly, while the Prolog solver we used was optimized for the task at hand, we plan to test the performance of other reasoners as well.

Acknowledgments Work was supported by the Netherlands eScience Center under grant number NLESC.SSI.2021b.011.

References

[1] Zhang Z. Weighing Stars: Aggregating Online Product Reviews for Intelligent E-commerce Applications. IEEE Intelligent Systems. 2008;23(5):42-9.

[2] Decker R, Trusov M. Estimating aggregate consumer preferences from online product reviews. International Journal of Research in Marketing. 2010;27(4):293-307.

[3] Ceolin D, Primiero G, Wielemaker J, Soprano M. Assessing the Quality of Online Reviews Using Formal Argumentation Theory. In: Web Engineering. Springer; 2021. p. 71-87.

[4] Ceolin D, Primiero G, Soprano M, Wielemaker J. Transparent assessment of information quality of online reviews using formal argumentation theory. Information Systems. 2022;110:102107.

[5] Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S, McClosky D. The Stanford CoreNLP Natural Language Processing Toolkit. In: ACL (System Demonstrations). ACL; 2014. p. 55-60.
[6] Cunningham H, Maynard D, Bontcheva K, Tablan V. GATE: an Architecture for Development of Robust HLT applications. In: Proceedings of the 40th Annual Meeting of the ACL. ACL; 2002. p. 168-75.
[7] Honnibal M, Montani I, van Landeghem S, A B. spaCy: Industrial-strength Natural Language Processing in Python.; 2020.
[8] Attardi G. DeepNL: a Deep Learning NLP pipeline. In: Proceedings of the 1st Workshop on Vector Space Modeling for NLP. ACL; 2015. p. 109-15.
[9] Agerri R, Artola X, Beloki Z, Rigau G, Soroa A. Big data for Natural Language Processing: A streaming approach. Knowledge-Based Systems. 2015;79:36-42.
[10] Kocaman V, Talby D. Spark NLP: Natural Language Understanding at Scale. Software Impacts. 2021;8:100058.
[11] Lenz M, Sahitaj P, Kallenberg S, Coors C, Dumani L, Schenkel R, et al. Towards an Argument Mining Pipeline Transforming Texts to Argument Graphs. In: COMMA 2020. vol. 326; 2020. p. 263 270.
[12] Snaith M, Devereux J, Lawrence J, Reed C. Pipelining Argumentation Technologies. In: COMMA 2010. vol. 216;. p. 447 453.
[13] Wang H, Huang Z, Dou Y, Hong Y. Argumentation Mining on Essays at Multi Scales. In: Proceedings of the 28th International Conference on Computational Linguistics. ACL; 2020. p. 5480-93.
[14] Baroni P, Caminada M, Giacomin M. Abstract argumentation frameworks and their semantics. In: Handbook of Formal Argumentation. College Publications; 2018. p. 159-236.
[15] Aakhus M, Lewiński M. Advancing Polylogical Analysis of Large-Scale Argumentation: Disagreement Management in the Fracking Controversy. Argumentation. 2017;31:179-207.
[16] Schneider J, Wyner AZ. Identifying Consumers' Arguments in Text. In: Workshop on Semantic Web and Information Extraction at EKAW; 2012. .
[17] Gabbriellini S, Santini F. From Reviews to Arguments and from Arguments Back to Reviewers' Behaviour. In: Agents and Artificial Intelligence. Springer; 2017. p. 56-72.
[18] Kunaefi A, Aritsugi M. Extracting Arguments Based on User Decisions in App Reviews. IEEE Access. 2021;9:45078-94.
[19] Gabbriellini S, Santini F. A Micro Study on the Evolution of Arguments in Amazon.com's Reviews. In: PRIMA 2015: Principles and Practice of Multi-Agent Systems. Springer; 2015. p. 284-300.
[20] Dung PM. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence. 1995;2:321 358.
[21] Schlosser AE. Can including pros and cons increase the helpfulness and persuasiveness of online reviews? The interactive effects of ratings and arguments. J of Consumer Psych. 2011;21(3):226-39.
[22] Mihalcea R, Tarau P. TextRank: Bringing Order into Text. In: EMNLP 2004. ACL; 2004. p. 404-11.
[23] Paco N. PyTextRank, a Python implementation of TextRank for phrase extraction and summarization of text documents.; 2016.
[24] Kusner M, Sun Y, Kolkin N, Weinberger K. From Word Embeddings To Document Distances. In: ICML 2015. vol. 37. PMLR; 2015. p. 957-66.
[25] Řehůřek R, Sojka P. Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. ELRA; 2010. p. 45-50.
[26] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. JMLR. 2011;12(85):2825-30.
[27] Wielemaker J, Schrijvers T, Triska M, Lager T. SWI-Prolog. Theory and Practice of Logic Programming. 2012;12(1-2):67-96.
[28] Reback J, jbrockmendel, McKinney W, den Bossche JV, Augspurger T, Roeschke M, et al.. pandas-dev/pandas: Pandas 1.4.2. Zenodo; 2022.
[29] He R, McAuley J. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In: WWW 2016. Intl. WWW Conf. Steering Committee; 2016. p. 507–517.
[30] McAuley J, Targett C, Shi Q, van den Hengel A. Image-Based Recommendations on Styles and Substitutes. In: SIGIR 2015. ACL; 2015. p. 43–52.
[31] Bal H, Epema D, de Laat C, van Nieuwpoort R, Romein J, Seinstra F, et al. A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term. Computer. 2016;49(5):54-63.
[32] Noor K, Hunter A. Analysing Product Reviews Using Probabilistic Argumentation. vol. 326 of COMMA 2020. IOS Press; 2020. p. 295 306.