On “Aggregation Detection in CSV Files”

Seminar: Modern Database Systems for Machine Learning and Knowledge Discovery 2022/2023

JAKOB HORNUNG

As part of their work on structure detection in CSV files, Jiang et al. address a more specific problem of detection of aggregations. The term aggregation is used to describe an arithmetic connection between a set of numbers and a single number. In their article “Aggregation Detection in CSV Files” the authors present an approach for detecting sums, averages, differences, divisions, and relative changes within the numeric cells of files that also contain non-numeric data. The objective of the present work is to outline this approach. The approach is composed of three stages, in which the aggregation candidates are determined row- and column-wise and pitted against each other using various heuristics. Experiments have resulted in precision and recall of over 0.95 for averages, divisions, and relative changes for 90% of the files examined. For an unseen dataset, comparable results are obtained. The so far unique approach leads to significantly better results with real data than a baseline where all permutations are checked row by row and column by column. However, the discussion of the results also shows that there is still room for improvement.

1 INTRODUCTION

In recent years, the hunger for data has increased steadily. The goal is often knowledge discovery through automated data processing. In what follows, we will consider a paper that is concerned with data preparation. It deals with the specific case of structure detection in comma-separated values (CSV) files. These text files typically represent tabular data by separating individual rows by separators into cells. Often the semantics is clearly specified by the creating or processing applications. However, the structure of CSV files is not always easy to recognize in practice, especially when they were exported from spreadsheets that contain not only simple tables. Jiang et al. refer to such files as *verbose*, formally defined as CSV files whose raw values serve various purposes, such as data, metadata, group headers, or notes and appear in various positions [2]. While there is other work that addresses structure detection in CSV files, Jiang et al. extend their approach to include a unique non-keyword-based detection of aggregations. Their work, entitled “Aggregation Detection in CSV Files” [1], is the subject of this paper. Note that since all of the following work is based on [1], citations are not explicitly given after each paragraph.

An aggregation denotes an arithmetic connection between a set of numbers and a single number. The single number – the aggregator – can be evaluated by applying an aggregation function to the set of numbers – the range. Jiang et al. consider in their paper the aggregations sum, difference, average, division, and relative change. In what follows, we will look at how the authors realized their approach to detecting aggregations in CSV files. We will then examine how well they succeeded. To do this, we will take a closer look at the evaluation methods and the results obtained.

2 AGGREGATION DETECTION APPROACH

Before discussing the aggregation detection approach in detail, we will first introduce some preliminary considerations. The heuristics-based approach itself is divided into three stages.

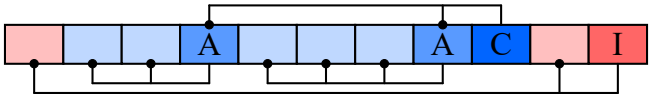

Fig. 1. Aggregator-aggregatee patterns: adjacent (A), cumulative (C) and interrupt (I)

2.1 Formalization and characterization of aggregations

The formal definitions of the aggregations considered are as follows:

- Sum $A = \sum_{i=1}^{n} B_i$,
- difference $A = B - C$,
- average $A = \frac{1}{n} \sum_{i=1}^{n} B_i$,
- division $A = B/C$,
- relative change $A = (C-B)/B$,

where A is the aggregator and B and C are the range elements. In practice, however, there is the complication that floating point numbers in CSV files can only be represented with finite precision, and, in addition, unknown rounding errors often occur. Therefore, error terms must be included in the equations for detection. However, the actual error is unknown. It should also be noted that when rounding to a specific decimal place, the relative error varies depending on the magnitude of the number. When accounting for rounding errors, a tradeoff must be made between not detecting aggregations caused by an error level that is too low and generating false positives due to an error level that is too high.

Only aggregations within rows or within columns are considered. The procedure is identical in each case. Jiang et al. classify three aggregation patterns covered by their approach. They distinguish between adjacent, cumulative and interrupt aggregator-aggregatee patterns. In the adjacent pattern, the aggregator is right next to its range. The aggregator in the cumulative pattern considers cells that are themselves adjacent aggregators. Therefore, a cumulative aggregator cannot be completely adjacent to its range, but only to its range combined with the range elements of its range. Only sums and differences are detected according to this pattern. Finally, the interrupt pattern is considered. Here, spaces can occur between aggregator and individual range elements, but only if they contain other aggregations. Figure 1 exemplifies the three patterns.

An additional point to be considered with CSV files is the number format used. Jiang et al. describe different combinations of digit group separators and decimal separators. For example, the same number can be represented both by 12345, 67 (digit group separator: *none*/decimal separator: *comma*) and by 12 345.67 (*space*/*dot*). The authors differentiate between the formats *space*/*comma*, *space*/*dot*, *comma*/*dot*, *none*/*comma* and *none*/*dot*. For the detection of aggregations, the correct interpretation of the numbers is required. Jiang et al. choose an approach that checks to which regular expression of a number format most cells match. They then transform the numbers into the normalized format *none*/*dot*. These serve as input to the workflow covered below.

2.2 Reflection of the elaborated workflow

The naïve approach to finding aggregations would be to consider for each cell all possible combinations of cells from the same row/column for an aggregation. See section 3 for why Jiang et al. do not believe this is appropriate. The authors themselves employ a heuristic approach. Aggregations are searched only in a certain neighborhood and pruning rules are applied. The approach called AggreCol is divided into three stages – individual aggregation detection, collective aggregation detection and supplemental aggregation detection – which are elaborated below. All procedures described for rows are applied analogously for columns.

Individual aggregation detection. Independent detection is performed for each of the five treated aggregation types in this stage. In addition to the error level for the aggregation equations, a second variable input parameter is specified with the coverage threshold. The coverage describes the ratio of lines in which an aggregation pattern was found to the total number of lines. It therefore specifies that the same pattern must occur in several lines in order to be detected. A customized strategy is implemented for commutative aggregations. For these, the order of the range elements does not matter – in our case, this is true for the sum and the average. Also, unlike differences, divisions, and relative changes, commutative aggregations are not limited to just two range elements but can have any number. For these an *adjacency list strategy* is applied. Commutativity allows to apply a greedy approach. For each possible aggregator candidate c_j, the possible range elements c_k with $k > j$ of the same row are iteratively added to the adjacency list starting from the closest cell. After each iteration, the aggregation equation is checked to determine if it is satisfied for the selected error level. If it is satisfied, the adjacency list corresponds to an aggregation range and the iteration is terminated. The same is done for $k < j$ to find range elements on the other side of the possible aggregator. In this approach, Jiang et al. set the minimum number of range elements to two so that false positives would not be detected for identical values in two adjacent cells. One problem with the adjacency list approach is that it may break out of the iteration loop too early. The aggregation equation may coincidentally be satisfied for fewer range elements than the actual aggregation includes. To counteract this, for each aggregation detected, it is examined if the same pattern can be found in nearby rows. If this is the case, the aggregation candidate is included. Finally, the coverage must be satisfied, i.e., a certain proportion of the rows must have an identical pattern for these aggregations to be considered.

For non-commutative aggregations – in this case difference, division, and relative change – the order of the range elements influences the result. As stated earlier, these three aggregations each have exactly two range elements. For these, a *sliding window strategy* is used. For each possible aggregator candidate c_j, all permutations within the window of size w are traversed. Once again, the procedure is performed for both sides of c_j. A candidate is added if the aggregation equation holds, and the coverage is satisfied.

All aggregation candidates are now grouped by their patterns. A group contains all aggregation candidates where the type, the column of the aggregator and the number and columns of the range elements match. A sufficiency score is defined as the number of candidates in the group divided by the total numeric cells in the column. For groups that share one aggregator and for groups that share the same range, only the one with the higher sufficiency score is considered in each case. The remaining groups are prioritized according to how many aggregation candidates they include and how small their average error is. By iterating over the prioritized groups, lower ranked groups that cannot coexist with the analyzed group are eliminated. This is the case when one of the following pruning rules applies:

- Directional disagreement: A possible aggregator is included in two groups of the same type but with range elements on different sides.
- Complete inclusion: A possible aggregator and part of its range elements are included in the range of an aggregation candidate of another group.
- Mutual inclusion: The possible aggregators of two aggregation candidates are mutually within the range of the other candidate.

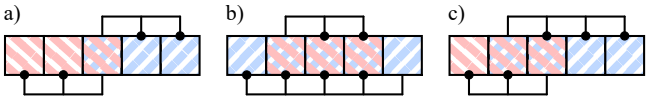

Fig. 2. Pruning rules: a) directional disagreement, b) complete inclusion and c) mutual inclusion

Figure 2 visualizes all three rules. The result of the individual aggregation detection stage are aggregation candidates (for rows and columns) for each aggregation type.

Collective aggregation detection. Contrary to its title, this stage does not actually involve any detection of aggregation candidates. Instead, further pruning is applied with the aggregation types no longer considered separately. Except for the division, which is excluded at this stage. Similar to the first phase, all groups – containing aggregation candidates of multiple rows with the same type and pattern – are ranked first. The criteria are the number of range elements of the group and secondarily the number of detected aggregations in a group. The complete inclusion and mutual inclusion rules from the previous stage (see Figure 2b and c) are applied across all aggregation types. If one rule applies, the lower ranked group is eliminated. Additionally, another case is considered. A cell must not be an aggregator of two aggregation candidates where the ranges (partially) overlap. However, this is allowed for disjoint ranges.

Supplemental aggregation detection. The goal of this phase is to additionally identify interrupt aggregations (see Figure 1). These have not been considered in the approach described so far. The underlying idea is to apply the individual aggregation detection approach, but with slightly modified CSV files as input. Each of the files results from systematic removal of certain already detected aggregation cells. Aggregator columns of the detected non-cumulative aggregation types are generally excluded from the constructed files, as they cannot be used as range elements. For the modified files, the individual detectors are executed in sequence. If new aggregations are discovered for at least one aggregation type, the complete procedure described so far for this stage, including file modifications, is repeated. If no aggregation detector finds new aggregations, the

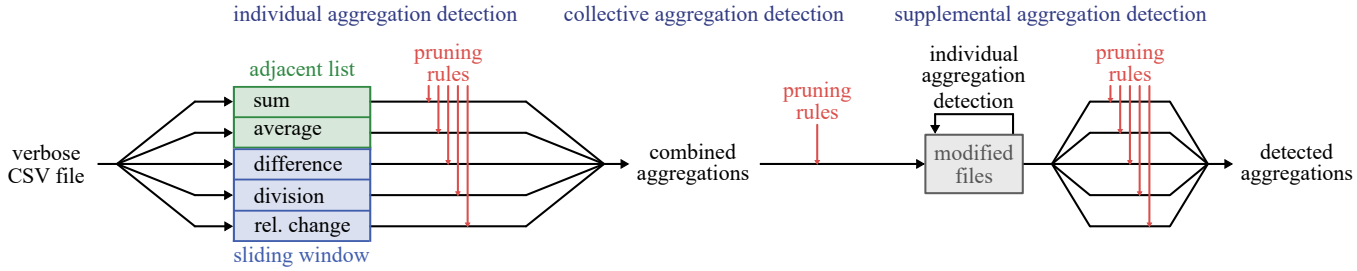

Fig. 3. Overview of the three stages of AggreCol namely individual/collective/supplemental aggregation detection

algorithm terminates. Finally, the same pruning rules as for the individual aggregation detection phase are applied. Figure 3 provides an overall view of all three stages.

3 EVALUATION

The challenge for Jiang et al. in evaluating AggreCol is that there are no comparable solutions. Therefore, no benchmarking can be performed. In the following, we will look at what evaluation methods are used and what results the evaluation provides.

3.1 Introduction of the evaluation methods

Two datasets of CSV files, each compiled from different but disjoint sources, form the basis for the empirical analysis. The authors have implemented a tool to manually annotate aggregations for all CSV files. These annotations include for each aggregation its type and the positions of aggregator cell and range elements. Non-numeric cells were included, for example ‘-’ as 0 or ‘+1.4 Points’ as 1.4. All files considered contain at least one aggregation, resulting in 185 and 81 files for the two datasets, respectively. The first dataset was used to develop the approach presented and to determine the input parameters. The second dataset is used for validation to show that no overfitting has occurred. As described in subsection 2.1, the transformation of the number format is performed initially.

A detected aggregation and the actual aggregation match if and only if the type, aggregator, and range elements match. Errors include missing an actual aggregation, or a detected aggregation being incorrect. Jiang et al. use the metrics precision P and recall R for the evaluation, where

$$P = \frac{n_{\text{correct}}}{n_{\text{correct}} + n_{\text{incorrect}}} \quad \text{and} \quad R = \frac{n_{\text{correct}}}{n_{\text{correct}} + n_{\text{missed}}}$$

with n_i corresponding to the absolute number of results with classification i. Thus, P indicates the number of correctly detected aggregations among all detected ones, while R includes the number of correctly detected aggregations among all actually correct ones. The authors specify that P or R becomes equal one whenever the denominator in the equations is zero. There is a trade-off between P and R. One can favor P by making the detection very restrictive to avoid false positives, or favor R by making the detection overly sensitive to avoid false negatives. Therefore, the F_1-score is introduced, which incorporates both metrics by forming the harmonic mean.

There are three input parameters in AggreCol that affect results: 1. The error level in the aggregation equations, 2. the aggregation coverage, which tells what percentage of all rows/columns must be covered by aggregations of the same pattern, and 3. the window size in the sliding window strategy. Jiang et al. set the window size to ten. The other two parameters are determined empirically based on the first dataset. Coverage is fixed at 0.7 by experimental maximization of the F_1-score. Also, the error level is set using the F_1-score. Obviously, a higher error level – if there occur indeed (rounding) errors in the files – leads to more true positives, but also to more false positives. The authors show that for all aggregation types considered, a different error level optimizes the F_1-score.

In the following, the methodology of the experiments will be briefly described. The results follow in subsection 3.2. Since a difference can be converted to a sum by moving the minuend to the aggregator side, Jiang et al. include differences in the sum metrics of the results. As a first evaluation of AggreCol, precision, recall and the F_1-score for the resulting aggregations are considered according to the individual, collective and supplemental aggregation detection stages per aggregation type. This is to capture the influence of each phase on the aggregations found. Moreover, the result after the third stage corresponds to the decisive overall result for the first dataset. However, the authors note that the distribution of aggregations across the files considered is not uniform. While many files have very few (at least one) aggregations, there are a few files with a large number of (at most 1651) aggregations. An evaluation based on the total number of aggregations over all files bears the risk that the detection is optimized for the few files with many aggregations. To counteract this bias, Jiang et al. introduce file-level effectiveness as an additional metric. Here, precision and recall are calculated per file and the proportion of files for which a certain threshold is exceeded is considered. This is the metric that is then used to evaluate AggreCol against the second dataset, which has not yet been studied. By comparing the results of the two datasets from dissimilar sources, it is possible to determine if there is an overfitting in favor of the first dataset. As mentioned before, AggreCol is the first approach that aims to detect aggregations in verbose CSV files. Therefore, the results can only be compared to the baseline. This examines for the given error levels the satisfaction of the aggregation equations for all cells as aggregators, each with all possible permutations within the same row/column as range elements. This corresponds to a time complexity of $O(n^3)$ for types with exactly two range elements or $O(n \cdot 2^{n-1})$ otherwise. In their setup, the authors have set a time limit for execution of 5 minutes per file.

3.2 Results

Examining the results after all three stages of AggregCol, the majority of the total aggregates detected correspond to those of the first stage. The precision for the considered aggregation types increases

after the phase of collective aggregation detection, while the recall remains the same or decreases negligible. It can be concluded that in this phase, where only pruning is performed, mainly false positives are removed. After the supplementary aggregation detection stage, there is both minimal deterioration and improvement for the three metrics across the aggregation types. Only for the average there is a significant change of the F_1-score by this phase of +0.039. Overall, F_1-scores of 0.782, 0.693, 0.860, and 0.844 are achieved for the sum, average, division, and relative change respectively across aggregations of all files.

At file-level, precision and recall above 0.95 are achieved for 90% of the files for average, division, and relative change. The authors argue that detection works well for many files, while some files contain aggregation patterns that are not covered. The results for the sum seem less promising. Precision and recall > 0.95 are only achieved for 79.2% and 61.6% of the files, respectively. There is a particular need for improvement in this respect, since sums account for about 70% of all aggregations in the dataset studied. For the unseen dataset, a qualitatively similar distribution of precision and recall values can be seen at the file-level. The proportion of precision and recall values above 0.95 decreases for most aggregation types, but not to an extent that AggreCol can be considered exclusively suitable for the first dataset.

For the comparison with the baseline, only the files of the unseen dataset are considered for which both approaches terminate within the time limit. With AggreCol, the limit was never exceeded. For each type, AggreCol achieves an F_1-score of > 0.95 for more than 60% of the files (pulled down by the sum), while the baseline only achieves this for a maximum of 35% of the files. The baseline approach produces a large number of false positives, which has a negative impact on precision. It turns out that the heuristics used in AggreCol not only provide efficiency benefits. The constraints where range elements are allowed also correspond more closely to aggregation patterns in real files than is achieved by checking all permutations.

Jiang et al. further investigate reasons for false positives/negatives. As described earlier, a fixed error level affects both of these errors. In addition, false positives occur for sums if many small numbers – especially zeros and ones – are present. Blank cells interpreted as zeros, or 1/0 used in the sense of true/false, increase the occurrence of this phenomenon. False negatives occur when interrupt patterns are not interrupted by other aggregations, range elements are outside the window width, the directional disagreement rule according to Figure 2 is violated in practice, or sum ranges end with 0.

3.3 Discussion and proposals

The previous subsection outlined where the authors themselves still see sources of error. A few remarks on the approach and the evaluation of AggreCol from an external perspective follow at this point. Clearly, the heuristics described in subsection 2.2 made assumptions that favored certain aggregation patterns and excluded others altogether. The results do not provide a complete indication of how well the approach is generally suited. On the positive side, the datasets were compiled from disjoint sources. However, it is fundamentally difficult to assess when exactly a dataset is a representative sample for *verbose* CSV files. A concern is the manual annotation of the actual aggregations in the CSV files. Here, deviations from the original files are possible. It would be conceivable to use Excel, Tableau or other files with aggregations and then export them (possibly with intentional rounding errors) as CSV files. Regarding the results, it is unclear why differences are included in the metrics of the sum. The authors justify this with the fact that their equations can be rearranged arithmetically. However, this seems to be irrelevant in this context, since they are detected by AggreCol using different strategies. Besides, the results of the file-level evaluation are considered biased. The authors define $P := 1$ for an aggregation type and a file if this type was not detected in this file, and $R := 1$ if it does not occur in it. Thus, for the proportion of files in which there are no aggregations of the examined type, $R > 0.95$ is obtained in each case. Similarly, $P > 0.95$ holds for the fraction of files in which there were no detections for that type. The results would be more meaningful if for P only those files were considered in which the aggregation type under investigation was detected at least once, and for R only those in which the type occurs.

It has been shown that the error level influences the false positive/negative rate. Here it would be conceivable to define an individual error level not only for each aggregation type, but also automatically for each file. Starting points could be the magnitudes of the occurring numbers and also the number format (e.g., number of decimal places). Jiang et al. distinguish their approach from evaluating cells by keywords, such as "total" and "all" for sum. They argue that matching keywords does not provide reliable results. Not all aggregators feature such labels, and false positives do occur. However, it would be possible to incorporate this information into the approach presented. Suppose a column label contains a keyword for an aggregation type. In this case, the range restriction for range elements could be made less restrictive when searching for a possible aggregator of that type in that column.

4 CONCLUSION

We have discussed an initial solution approach that deals with the detection of aggregations – namely sum, average, difference, division, and relative change – in CSV files. This heuristics-based workflow performs significantly better than a baseline that tries all permutations, both in terms of runtime and precision. The results indicate that the approach achieves high precision and recall for aggregation patterns of many files, except for the sum. Nevertheless, it must be mentioned that the results are not completely informative. Suggestions were made on how to present the results in a more meaningful way and how to enhance the workflow. In any case, Jiang et al. describe that they achieved improved results in their higher-level research area – structure detection in verbose CSV files – using the presented aggregation detection method.

REFERENCES

[1] Lan Jiang, Gerardo Vitagliano, Mazhar Hameed, and Felix Naumann. 2022. Aggregation Detection in CSV Files. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*. OpenProceedings.org, 2:207–2:219. https://doi.org/10.48786/edbt.2022.10

[2] Lan Jiang, Gerardo Vitagliano, and Felix Naumann. 2021. Structure Detection in Verbose CSV Files. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*. OpenProceedings.org, 193–204. https://doi.org/10.5441/002/edbt.2021.18