Developing Scalable Kingfisher

Capstone Project Report

Tahsin Mostafa

Supervision: Dr. Osmar R. Zaiane

1 Background

Association rule mining [1] is the task of extracting co-occurrence relationships between items in a dataset. The typical algorithms for discovering these associations require very difficult-to-set parameters, such as support and confidence. In some applications where association rules are used, these parameters may have little meaning, making them even more difficult to set or interpret.

A less-known algorithm for mining association rules is the Kingfisher [2]. It does not have the aforementioned parameters but relies on statistical significance. One issue with the original implementation of the Kingfisher algorithm is that it essentially stores each row in the dataset as a vector of 0s and 1s, where 0 means an absence of an attribute-value pair in that dataset row, and 1 means a presence. An attribute-value pair can look like: (Age, 20-25), for example, and these pairs are represented by tokens like '12', '25', etc in a dataset. The binary representation used can be problematic in terms of run-time and memory requirements of the algorithm when there are large numbers of attribute-value pairs in the dataset (think about discretizing continuous attributes, for example); as a result, the algorithm is unusable for large datasets, particularly with large dimensionality.

2 Objective

The objective is to make the Kingfisher algorithm more scalable, modifying the implementation to use a more efficient representation of the dataset- where each row is represented only by the tokens representing the attribute-value pairs that are present in each data row. We call this the itemset representation. This should greatly reduce the run-time and memory requirements and make it feasible for the algorithm to be used on large datasets. What makes Kingfisher efficient is the branch-and-bound optimization strategy used. The project also requires the investigation of whether this optimization strategy can still be adapted to the new representation. Further, the project entails verifying the correctness of the algorithm and conducting experiments to make sure it is scalable for large datasets.

The developed algorithm has the potential of improving several works related to developing associative classifiers [3], [4], explainability of black-box machine learning models [5], among others.

3 Implementation

We use the original kingfisher software's source code as our code base. Then, we identify where the software initializes and stores the contents of the dataset in a data structure, and all the other areas where the software uses the data structure, and decide to modify those parts of the software, keeping other functions and operations the same. The modified software can be found here. In addition to some small changes in the original code base regarding file imports and function definitions, the

following are the major changes that we implement to make the software use the new representation of the dataset.

3.1 Dataset initialization

In the original software, the `initattrmatr` function in the `bitmatrice.c` file is used to initialize the dataset by essentially creating a 2D matrix. The matrix is a dynamically allocated array of `bitvector` pointers of size k, where k represents the total number of attribute-value pairs. A `bitvector` is an alias for `unsigned long long int` in the software. Each of the k rows in the matrix is allocated memory for a `bitvector` that can hold n bits, where n is the total number of rows in the dataset. Using bit operations, the appropriate bits in the matrix are set to 1 in order to indicate the presence of an attribute-value pair in a particular dataset row.

In the modified software, a dynamically allocated array of size n is used. The number of attribute-value pairs that are present in each row is counted and each of the n cells in the array is allocated memory to hold only those number of attribute-value pairs that are present in the corresponding row in the dataset. Finally, the tokens representing these attribute-value pairs are stored in the array by parsing the dataset.

3.2 Counting frequency of 1-item sets

In the original software, the `generate1sets` function in the `kingfisher.c` file is used to generate frequent 1-item sets. 1-item sets are just the individual attribute-value pairs. As a part of the function, the 2D matrix created from the dataset is used to count the frequency of each 1-item set in the whole dataset. The program goes over k rows and n columns of the matrix, checks if each bit is 1 and increases the frequency value in the index representing that attribute-value pair/1-item set in a frequency array.

In the modified software, in the same C file and function, in order to count the number of 1-item sets, the program only goes over the attribute-value pairs present in a particular row of the dataset and increases the frequency value for those attribute-value pairs in the frequency array, and then does this for all dataset rows.

3.3 Counting frequency of t-item sets

In the original software, the `frset` function in the `kingfisher.c` file is used to count the number of times a given t-item set appears together in all the rows of the dataset. T-item sets are combinations of attribute-value pairs of length t. For example, a 2-item set may look like: {(Age, 20-25), (Income, 30K-40K)} and may be represented as $\{12, 45\}$. Given a t-item set, the program does a `bitwise-and` operation among all the `bitvectors` corresponding to the attribute-value pairs in the t-item set. Then, the program stores the result in a temporary `bitvector` of size n, and counts the number of 1 bits in that `bitvector` to count the frequency of the given t-item set in the whole dataset.

In the modified software, in the same C file and function, the program goes through each cell of the array storing the dataset, and for each attribute-value pair in the t-item set, performs a binary search to locate the attribute-value pair. Recall that each cell of the main array points to an array containing only the attribute-value pairs present in the corresponding dataset row. If all t attribute-value pairs are found in the array of a cell, the frequency counter is incremented.

4 Correctness

In order to determine the correctness of the modified software (itemset representation), we benchmark it against the original software (binary representation) using datasets from the frequent itemset mining dataset repository [1]. In particular we use the `mushroom`, `chess`, and `retail` [6] datasets, in addition to several synthetic datasets that we create for our scalability experiments described in

[1] http://fimi.uantwerpen.be/data/

Section 5. We find that the rules generated for all the datasets are identical for both the softwares. The outputs for each software for the `mushroom`, `chess` and `retail` datasets can be found in the following links: itemset, binary.

5 Scalability

As a first step to analyze the scalability of the itemset representation, we measure the execution time of the software using the `clock` function in C- starting from initialization of all the parameters till the end of the search process of finding the best rules.

We notice that the two major parameters that affect the execution time of the software are the row length and the total number of attribute-value pairs, k in the dataset. Moreover, from some rough experimentation, we notice that the itemset representation is more efficient in terms of execution time compared to the binary representation for datasets that have a high ratio of k to row length. On the other hand, it is less efficient than the binary representation for datasets with a low ratio of k to row length.

In order to better understand our observations, we decide to compare the execution time of the softwares using the binary and itemset representations on datasets with varying values of row length and k.

5.1 Experiment Setup

We conduct all the experiments in this project in a machine with Intel Core i7 (2.8 GHz and 4 cores) and 16 GB RAM. Moreover, we only set the values of four input parameters of the software while conducting the experiments; they are: the executable file for the software for the itemset and binary representations, the dataset file for different row lengths, the value of k, and lastly the threshold for the goodness measure which we fix to be -1500 for all datasets; all other settings of the software are kept at default.

5.2 Synthetic Data Generation

The attribute-value pairs are represented as integer tokens in the datasets. Each generated dataset has 8124 rows and a maximum integer of 119 as in the `mushroom` dataset. The row length of each of the generated datasets vary from 10 to 50. All rows in a particular generated dataset have the same length. The `mushroom` dataset has 23 integers in each row. We use the same values of the `mushroom` dataset as the base in the generated datasets; wherever we need to trim the row length we discard the required number of trailing integers in each row, and wherever we need to increase the row length we choose a number between 1 and maximum integer, making sure that each row is sorted in ascending order and there are no duplicates in a row (just like the original `mushroom` dataset). The generated datasets can be found in the datasets folder.

5.3 Results and Discussion

In Figure 1, each row in the dataset has a length of 20. While running the softwares with the binary and itemset representations, we pass different values of k as input, and calculate the execution times for each representation. Note that we calculate the execution time for each representation and each k over three runs, average the times, and plot that; we follow this procedure for all the execution time v.s. k graphs plotted in this section. We observe that the itemset representation becomes more efficient over the binary representation when k is about 155 thousand when row length is 20.

In Figure 2, each row in the dataset has a length of 30, and we notice that the point where the itemset representation becomes more efficient is when k is about 1020 thousand (1 million and 20 thousand). The point where the itemset representation becomes more efficient has a much larger k value when the row length is 30 compared to when the row length is 20 in a dataset. These observations give us an idea that increased row length hurts the performance of the itemset representation more than the binary representation, while increased k hurts the performance of the

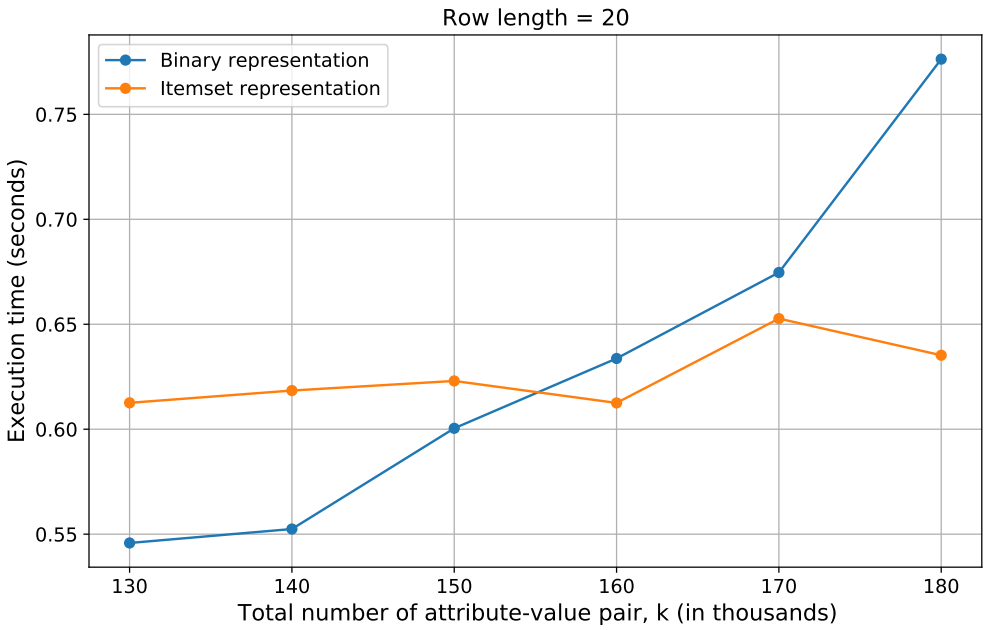

Figure 1: Execution time v.s. k

binary representation more. And thus, as we increase row length, the itemset representation requires more k to outperform the binary representation.

Moreover, we observe that the binary representation curve is steeper than the itemset representation curve. This is because as k increases, the binary representation allocates memory for all the attribute-value pairs and has to do all the operations like frequency counting for all those attribute-value pairs. On the other hand, the itemset representation only allocates memory for the attribute-value pairs present in each row of the dataset and needs to only traverse over those while doing operations like frequency counting. Therefore, the increase in k affects the execution time of the itemset representation much less.

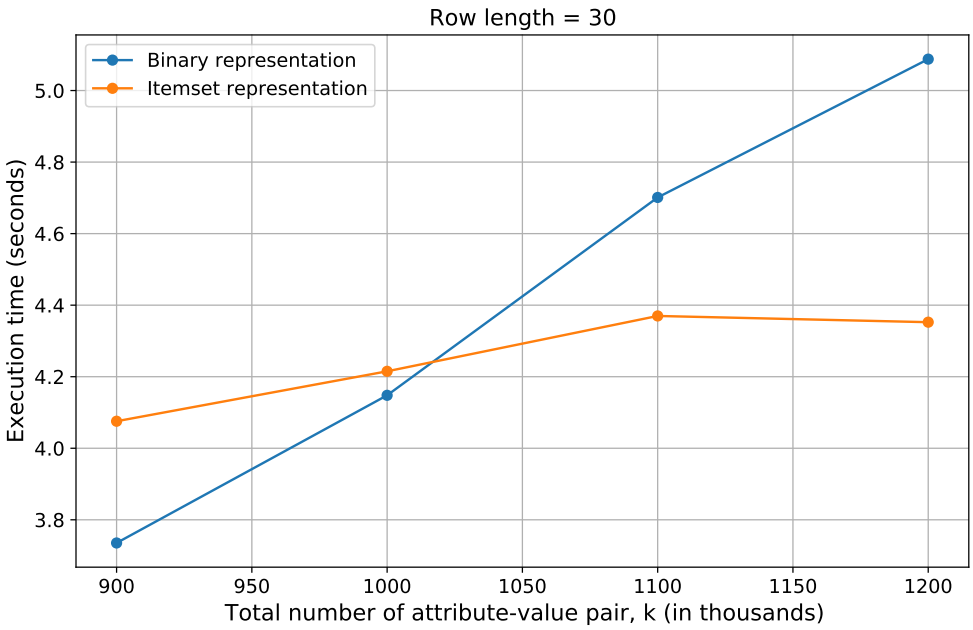

Figure 2: Execution time v.s. k

In Figure 3, we plot the value of k at which the itemset representation outperforms the binary representation for different values of row length. We can understand from this graph that the increase in row length has a big effect on the execution time of the itemset representation because the rate at which the value of k needs to increase (to outperform binary) is high.

In order to understand the relationship between k and row length for which the itemset representation becomes more efficient, we plot a bar plot of ratio of k (at which itemset outperforms binary) to row length in Figure 4. Given a dataset's row length and total number of attribute-value pairs, Figures 3 and 4 can give us a rough idea as to which representation will be better in terms of execution time for the dataset.

6 Conclusion

In this project, we have implemented the itemset representation of the kingfisher software and investigated its correctness and scalability. Through our experiments, we have gained an understanding of when the itemset representation is more efficient to use over the binary representation in terms of the relationship between the row length and total number of attribute-value pairs in a dataset. The github repository of the project can be found here.

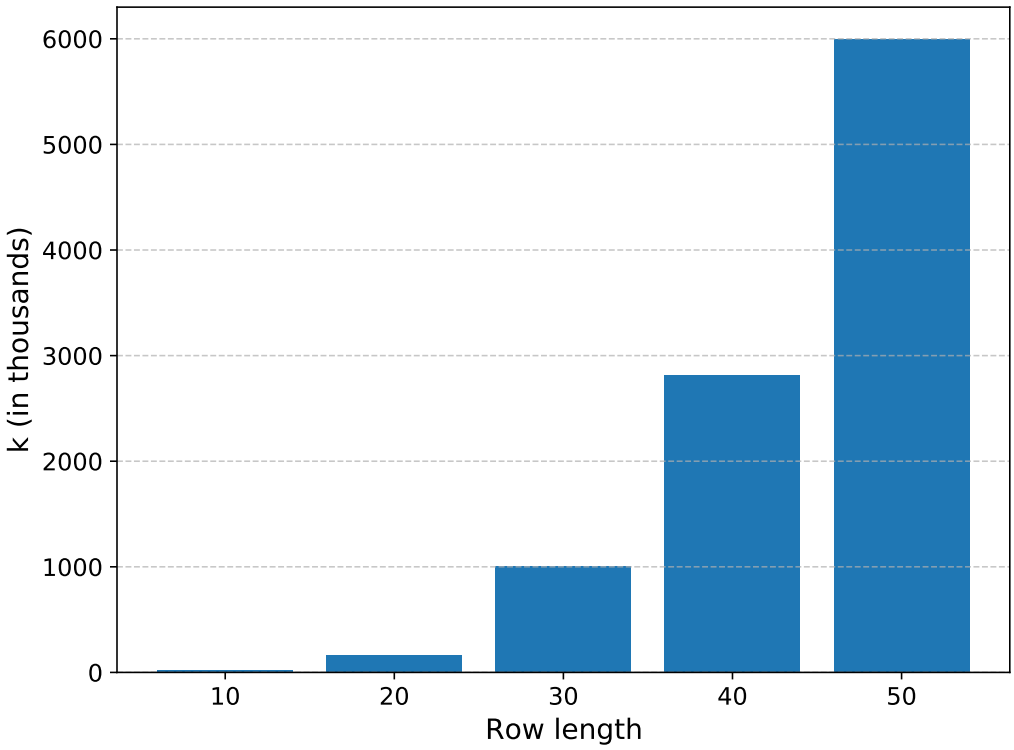

Figure 3: k (at which itemset outperforms binary) v.s. Row length

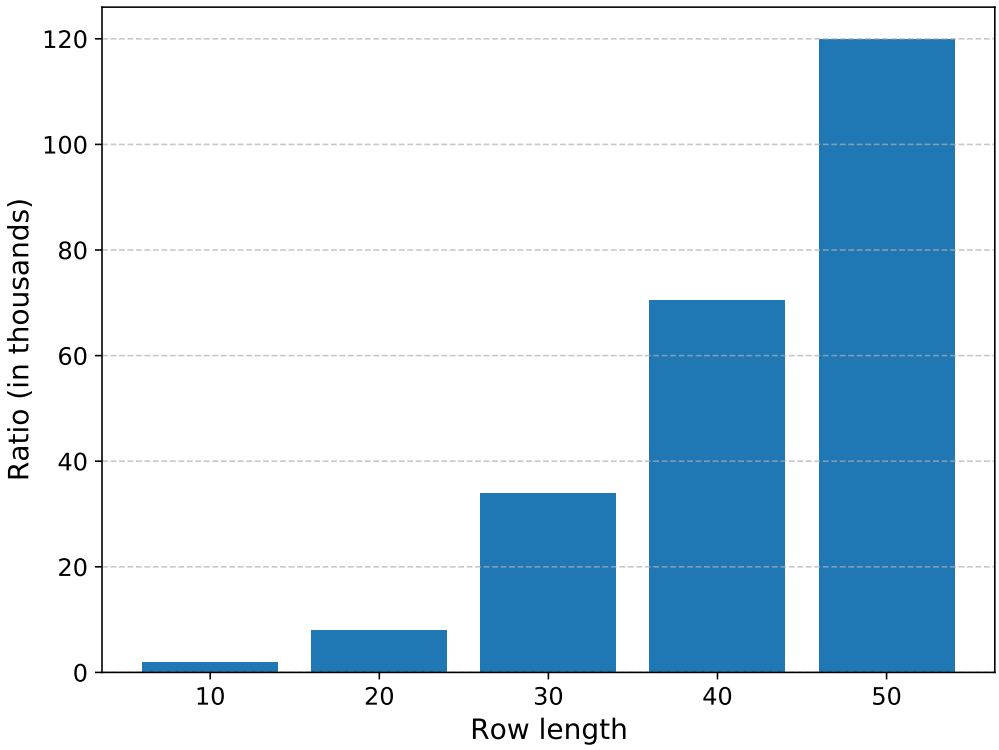

Figure 4: Ratio of k (at which itemset outperforms binary) to row length v.s. Row length

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In: *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499. ISBN: 1558601538.

[2] Wilhelmiina Hämäläinen. "Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures". In: *Knowledge and Information Systems* 32 (2011), pp. 383–414. URL: https://api.semanticscholar.org/CorpusID:3338191.

[3] Jundong Li and Osmar R. Zaiane. "Exploiting statistically significant dependent rules for associative classification". In: *Intell. Data Anal.* 21.5 (Jan. 2017), pp. 1155–1172. ISSN: 1088-467X. DOI: 10.3233/IDA-163141. URL: https://doi.org/10.3233/IDA-163141.

[4] Jundong Li and Osmar Zaiane. "Associative Classification with Statistically Significant Positive and Negative Rules". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 633–642. ISBN: 9781450337946. DOI: 10.1145/2806416.2806524. URL: https://doi.org/10.1145/2806416.2806524.

[5] Mohammad Motallebi, Md Tanvir Alam Anik, and Osmar R. Zaıane. "Explaining Decisions of Black-Box Models Using BARBE". In: *Database and Expert Systems Applications: 34th International Conference, DEXA 2023, Penang, Malaysia, August 28–30, 2023, Proceedings, Part II*. Penang, Malaysia: Springer-Verlag, 2023, pp. 82–97. ISBN: 978-3-031-39820-9. DOI: 10.1007/978-3-031-39821-6_6. URL: https://doi.org/10.1007/978-3-031-39821-6_6.

[6] Tom Brijs et al. "Using Association Rules for Product Assortment Decisions: A Case Study". In: *"Knowledge Discovery and Data Mining", pages = "254-260", year = "1999".*