Karol Ziolo: 20224449

Data Mining: Association Rules

Exercise 1

In the initial exercise, our objective was to thoroughly examine the code provided by Chat GPT and identify potential areas for improvement. As I delved into the code, the first issue that immediately caught my attention was the algorithm's unnecessary traversal of the entire dataset while searching for candidate itemsets. This inefficiency became evident because our primary interest lied in calculating frequencies only for frequent itemsets, rendering the exhaustive search through the entire dataset redundant. To rectify this, I explored several optimization techniques that could be applied to enhance the algorithm's performance.

One technique that emerged as a potential solution is Dynamic Itemset Counting. This approach involves interrupting the counting process after every M transaction and pivoting to generating larger candidates if feasible. By adopting this strategy, we can reduce the computational burden of traversing the entire dataset repeatedly and focus on extracting meaningful patterns from the frequent itemsets.

Another optimization technique worth considering is partitioning the database. This entails dividing the dataset into smaller, more manageable partitions and mining each partition separately. By doing so, we can distribute the workload and exploit parallel processing capabilities, potentially leading to significant speed improvements. This partition-based approach allows for more efficient mining of frequent itemsets, as it enables us to tackle the problem in a divide-and-conquer manner.

Additionally, Sampling presents another avenue for optimizing the algorithm's performance. By employing the Apriori algorithm on a subsample of the database, we can obtain a representative subset of transactions that retains the essential patterns while reducing the overall dataset's size. This technique leverages the concept of statistical sampling to provide approximate results while alleviating the computational requirements associated with processing the complete dataset.

Moving on to the second issue that undermines the code's efficiency, the apriori function's treatment of all possible itemsets without considering the monotonicity of subsets poses a significant drawback. Currently, the algorithm counts the support of an itemset, even if its subsets are infrequent, thereby contradicting the likelihood of the original itemset being frequent. To address this concern, we endeavored to enhance the apriori function by implementing modifications that exclude infrequent itemsets from consideration. By omitting these irrelevant itemsets, we eliminate unnecessary computations and streamline the search process for frequent itemsets, thereby improving the overall efficiency of the code.

To validate the effectiveness of my improvement, I executed original code, the modified one and the pre-existing apriori function from the mlxtend.frequent_patterns package on two datasets generated by ChatGPT, as well as the retail dataset available at http://fimi.uantwerpen.be/data/ . By comparing the outcomes of these experiments, I sought to

assess the impact of my optimization on the algorithm's performance in terms of runtime. The comprehensive results of these evaluations are presented in the table below.

Table 1: Comparison of the apriori algorithms.

Nr of transactions / Support Threshold	Time Chat GPT apriori	Time of Upgraded apriori	Time of mlxtend apriori
N=10 / min_sup=0.3	543 µs	82 µs	2028 µs
N=15 / min_sup=0.3	49 s	758 µs	2483 µs
N=88,162 / min_sup=0.01	over 50 min	78 s	5 s

Exercise 2

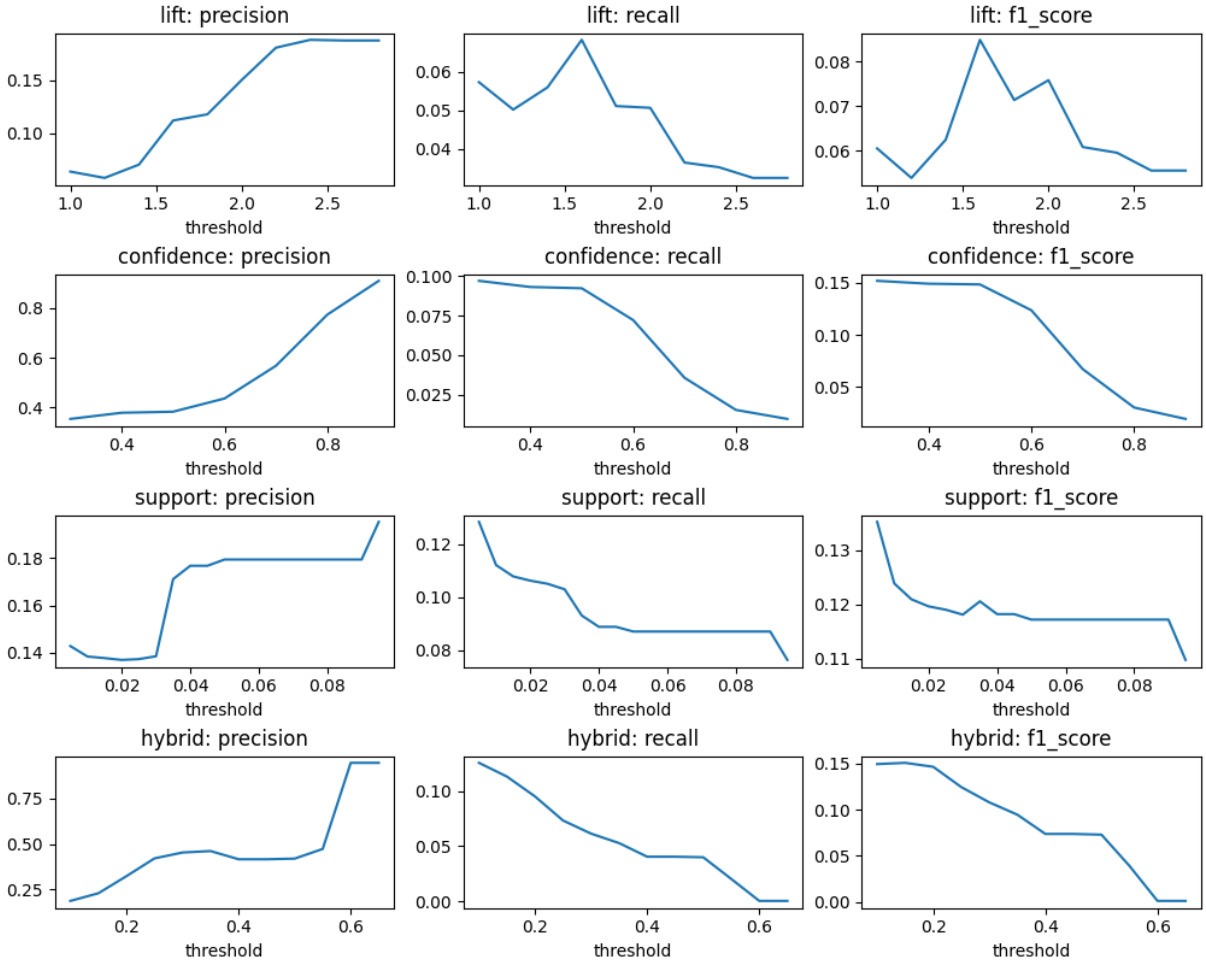
In the second task, we were given a set of functions designed to generate a recommendation system. The existing approach utilized the average confidence metric to rank the rules. Our objective for this task was to explore alternative ranking methods and compare their performance. To accomplish this, I opted to utilize the apriori and association_rules functions available in the mlxtend.frequent_patterns package to discover association rules.

Before implementing the code, it was necessary to gather the appropriate data and convert it into the required format. I again selected the retail dataset from http://fimi.uantwerpen.be/data/ for this purpose. The code expected a one-hot encoded dataframe as input, so I utilized the MultiLabelBinarizer class from the sklearn.preprocessing package to perform the conversion. Additionally, I made the decision to exclude all baskets containing only a single item, as they did not provide any meaningful information for the recommendation system. Subsequently, I needed to generate a test dataset for evaluation purposes. I accomplished this by employing the train_test_split function and randomly selecting a varying number of items from each test basket, which would serve as the expected output to be recommended by the model.

Using the training data, I proceeded to calculate the frequent itemsets using the apriori algorithm. To strike a balance between generating a sufficient number of rules and computational efficiency, I set the minimum support threshold to 0.005. Higher support values yielded significantly fewer rules. Subsequently, the association_rules function was employed to generate the rules. This function required the specification of a ranking metric, its minimum threshold, and the number of top recommendations to be returned by the model. For consistency, I chose to provide the top 5 results for each model. Initially, I focused on investigating the confidence, lift, and support metrics as potential ranking methods. Additionally, I computed a hybrid score, which was an equally weighted sum of the previously mentioned metrics, to provide a comprehensive evaluation criterion. The evaluation function provided by Chat GPT yielded precision, recall, and F1-score as outputs. The performance comparison of all four statistics, depending on their respective thresholds, is depicted in the

figure below. This analysis serves to illuminate the effectiveness of the different ranking methods and aids in selecting the most appropriate approach for the recommendation system.

Figure 1: Comparison of evaluation statistics.



Let's begin by examining the results for the lift metric. As the threshold increased, we observed a gradual increase in precision. However, when it comes to recall, the scores started to decline once the threshold reached a value of 1.5. The F1-score, on the other hand, did not exhibit a consistent pattern. It reached its peak at a threshold of 1.5. Moving on to the confidence metric, we noticed that precision followed an exponential-like pattern. In contrast, both recall and F1-score exhibited an opposite pattern, with a decline starting at a threshold of 0.5. Next, we analyzed the support statistic. It became apparent that the most optimal results, considering all the statistics together, were obtained when the threshold was set between 0.04 and 0.08. Lastly, we considered the hybrid score. In terms of precision, a significant increase was observed once the threshold reached 0.55. However, recall gradually decreased across all investigated thresholds and dropped to zero when the threshold reached 0.6. The F1-score followed a similar pattern.

To gain a better understanding of these results, I delved into the graphs and identified the most optimal thresholds for each metric. But before doing so, let's clarify the meaning of the evaluation statistics. Precision measures the ability to accurately recommend relevant products. Recall, on the other hand, assesses the proportion of relevant products recommended

out of all the relevant products that could have been recommended. The F1-score takes into account both precision and recall. Given that we are dealing with a retail dataset, our primary objective is to provide customers with a large number of relevant product recommendations. In this context, the consequences of delivering poor recommendations are not as severe as they would be in the case of a medical dataset. Therefore, we have a slightly higher interest in the recall score. However, we cannot disregard precision entirely, as we also aim to maintain customer satisfaction.

Ultimately, I decided to select the following thresholds: lift (1.5), confidence (0.6), support (0.05), and hybrid (0.3). With these thresholds in place, the following statistics were generated:

Table 2: Evaluation scores comparison.

	Precision	Recall	F1-score	Number of rules
Lift	0.1308	0.0933	0.1089	410
Confidence	0.4323	0.0707	0.1215	349
Support	0.1779	0.0865	0.1164	32
Hybrid	0.4564	0.0623	0.1096	48

First thing which is noticeable is that Confidence and Hybrid achieved the best results. However, we can notice that results aren't perfect. There are couple of reasons that are responsible for that. Firstly, we have significantly smaller number of rules in comparison to the number of product types, which was 16,470. It might be caused by the minimal support in apriori function. I selected 0.005 which means that item to be considered as a frequent one must occur 426 times in transactions. By taking into account number of product types, the number of rules doesn't surprise. Possible solution for that would be decreasing minimal support, however due to limitations I couldn't investigate deeper that issue in this section.

Besides minimal support, there are also other attributes that impacts the results. One of that is the number of recommendations the model returns. As we increase it, the chance of recommending correct item gets higher, which have a positive impact on the true positives and negative one on the false negatives. Therefore, the recall might get higher. On the other hand, the number of false positives increases which have negative impact on the precision.

At the end, it should be also pointed out that testing phase is also not perfect. The aim of the recommender system is to expose recommended products to customer and then investigate whether he decided to buy any of them. Here, the testing phase used finite baskets where testing items were randomly selected from the basket. We are omitting customer behavior which is crucial in that case.

Exercise 3

In the last exercise we were supposed to try out the Non-Derivable-Itemsets implementation. The Non-Derivable-Itemsets implementation finds sets of items in a dataset that cannot be derived or obtained from any other sets of items. It filters out itemsets that can be derived from larger sets, focusing on unique and independent patterns. This analysis helps identify significant itemsets that stand on their own without depending on other itemsets, revealing interesting associations or patterns in the data. Ultimately, it allowed to investigate itemsets

with lower support. Firstly, I decided to experiment with the following support thresholds: 85, 125, 175, 250 and 450 transactions. I also decided to set the IEdepth parameter to 5. The table below represents the number of itemsets meeting condition for each threshold:

Table 3: Number of itemsets for each minimal support.

Threshold: 85	Threshold: 125	Threshold: 175	Threshold: 250	Threshold: 450
8,116	4,613	2,735	1,559	624

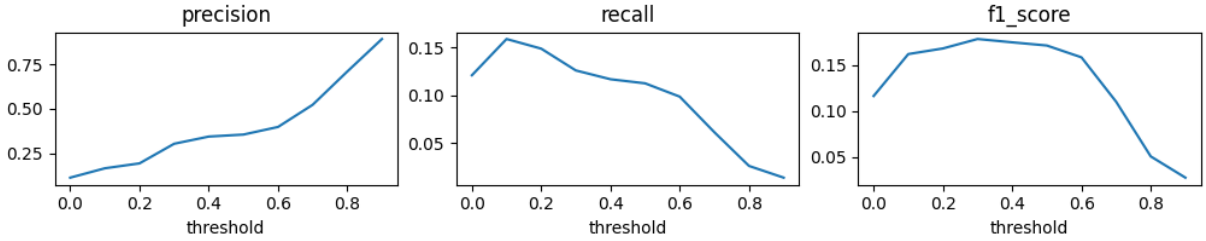
To examine the performance of the recommendation system I used confidence metric with threshold equal to 0.5. The table below represents the results I obtained:

Table 4: Performance results for different support thresholds.

Min Support	Precision	Recall	F1-Score
85	0.3572	0.3572	0.1726
125	0.3631	0.1108	0.1698
175	0.3682	0.1051	0.1635
250	0.3729	0.1005	0.1584
425	0.3799	0.0916	0.1477

Based on the results, we can conclude that itemsets with low support should be taken into account for building recommendation system for that dataset. Therefore, I also decided to investigate the confidence threshold for minimal support equal to 85.

Figure 2: Confidence thresholds comparison.



Based on the graphs, we can notice that confidence threshold set to 0.3 performs the best. We have slightly lower precision, but we get higher recall. Ultimately, we obtain the highest F1-score across every examined model, which was over 17%.

To conclude, if I was supposed to select my final model, I would choose the last one. That is the one with minimal support equal to 85 transactions and confidence with 0.3 threshold. However, we should keep in mind that not all possible combinations have been tried out. There are other parameters that could improve the performance of the recommender system.

Project Structure:

```
├── task1
│   ├── apriori_upgraded.py          - modified apriori
│   ├── functions_GPT.py             - functions provided by ChatGPT
│   ├── retail.dat                   - retail dataset
│   └── run_apriori.ipynb            - main script for algorithm comparisons
├── task2
│   ├── functions.py                 - helper functions
│   ├── hybrid_evaluation.py         - functions for hybrid metric analysis
│   ├── ranking_investigation.ipynb  - main script for ranging evaluation
│   ├── test_baskets.pickle          - test consumer baskets (used also in 3)
│   ├── test_output.pickle           - output baskets (used also in 3)
│   └── train_baskets.pickle         - training baskets (used also in 3)
└── task3
    ├── functions.py                 - helper functions
    ├── ndi 2                        - contains ndi C++ scripts
    ├── output1.txt                  - ndi output with 85 min support
    ├── output2.txt                  - ndi output with 125 min support
    ├── output3.txt                  - ndi output with 175 min support
    ├── output4.txt                  - ndi output with 250 min support
    ├── output5.txt                  - ndi output with 450 min support
    └── run_ndi.ipynb                - main script for ndi analysis
```