Karol Ziolo: 20224449

**Data Mining: Association Rules**

1. **Exercise 1**

In the first exercise we were supposed to examine the code provided by Chat GPT and find possible improvements.

The first thing that was immediately spotted was the fact that the algorithm is going through the whole dataset while it is looking for candidate itemset. This is completely unnecessary as we want to calculate frequencies for only frequent itemsets. There are couple of optimization techniques. The first one is the Dynamic Itemset Counting which interrupts after every M transaction and starts to generate larger candidates if possible. The second one is partition which partitions database and mines each part separately. The last one would be Sampling which runs apriori algorithm on subsample of the database.

The second issue which also makes the code very inefficient is that apriori function iterates through all possible itemsets. It doesn’t take into account the fact that subsets are monotone. It means that this algorithm counts the support of itemset even though its subsets are infrequent which actually contradicts the possibility that the original itemset is frequent. Therefore, to get rid of that issue the code should omit all infrequent itemsets so that it wouldn’t consider any further sets including them. I solved that issue by applying some changes to the apriori function. I called **apriori\_upgraded** in **excercise1.py** file. I run both functions and the one from the **mlxtend.frequent\_patterns** package on two datasets created by ChatGPT and **retail** datasetfrom <http://fimi.uantwerpen.be/data/>. The outcomes are presented in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| Transactions  (nr of transactions) | Time Chat  GPT apriori | Time of  Upgraded apriori | Time of  mlxtend apriori |
| N=10 | 543 μs | 82 μs | 2028 μs |
| N=15 | 49 s | 758 μs | 2483 μs |
| N=88,162 | over 50 min | 78 s | 5 s |

1. **Exercise 2**

For the second task we were provided with the functions that were supposed to generate the recommendation system. To rank the rules the average confidence was used. Therefore, the aim for that task was to use these functions and compare other ranking methods. To do that I decided to use **apriori** and **association\_rules** functions from the **mlxtend.frequent\_patterns** package to find the association rules.

Before implementing the code, I needed to collect the data and convert it to proper format. I decided to use the **retail** datasetfrom <http://fimi.uantwerpen.be/data/>. The code required to provide the one hot encoded dataframe, so I used the **MultiLabelBinarizer** class from **sklearn.preprocess** package. I also decided to remove all one-item baskets because they don’t give any useful information. Afterwards, I needed to generate a test data that I would use for evaluation purposes. I used **train\_test\_split** function and then from each test basket I randomly selected random number of items which would be treated as an output that should be recommended by the model.

Using training data, the frequent itemsets have been calculated using apriori algorithm. The minimum support was set to 0.005 as the higher ones provided significantly less rules. Afterwards, using association\_rules function the rules have been generated. This function required to provide the metric, its minimum threshold and number of top recommendations that would be returned by model. For each model it was decided to provide top 5 results. At the beginning of the analysis, I decided to investigate confidence, lift, and support as metrics for ranking method. Additionally, the hybrid score was calculated which was equally weighted sum of each previously mentioned metrics. The outputs of the evaluation function provided by Chat GPT were precision, recall and F1-score. The figure below represents the performance comparison of all four statistics depending on the thresholds.

A picture containing text, diagram, line, font

Description automatically generated

Figure 1: Comparison of evaluation statistics.

Firstly, let’s focus on the lift. The precision was gradually increasing as the threshold increased. However, in terms of the recall, after threshold reached value 1.5 the scores started to drop down. Additionally, it is noticeable that F1-score doesn’t have a continuous pattern. It reaches its peak when threshold is 1.5. In case of confidence, we can notice that the precision follows pattern of the sigmoid function. However, the recall and F1-score follow similar pattern together as they are starting to drop after the threshold reaches 0.6. The next statistic that was compared was the support. And here we can notice that the most optimal result in terms of all statistics together was while the threshold has been set between 0.04 and 0.08. The last but not least was the hybrid score. In terms of precision, we can notice a significant rise after threshold reaches 0.55. However, the recall decreases gradually through all investigated thresholds and drops do zero again while threshold reaches 0.6. The F1-score follows similar pattern.

Afterwards, I decided to investigate more why I obtained such results. Therefore, based on the graphs I selected the most optimal thresholds for each metric. But before, let’s focus what evaluation statistics represent. Precision measures ability to recommend the relevant products with high degree of accuracy. Recall, on the other hand, measures the proportion of relevant products that are recommended out of all the relevant products that could have been recommended. However, the F1-score takes into account both precision and recall. We need to consider that we are working with retail dataset, therefore we would be focused on recommending most of relevant products to customers. Therefore, we would be slightly more interested in recall score. However, we cannot completely avoid precision as we want to remain customer satisfaction. Ultimately, it was decided to select following thresholds: lift (1.5), confidence (0.6), support (0.05) and hybrid (0.3). For these thresholds the following statistics have been generated:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Number of rules** |
| **Lift** | 0.1324 | 0.0935 | 0.1096 | 410 |
| **Confidence** | 0.4349 | 0.0718 | 0.1232 | 349 |
| **Support** | 0.1801 | 0.0870 | 0.1173 | 32 |
| **Hybrid** | 0.4431 | 0.0599 | 0.1055 | 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 of the package and my computer 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 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.

1. **Exercise 3**

In the last exercise we were supposed to try out the Non-Derivable-Itemsets implementation. 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:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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:

|  |  |  |  |
| --- | --- | --- | --- |
| **Min Support** | **Precision** | **Recall** | **F1-Score** |
| 85 | 0.3661 | 0.1172 | 0.1776 |
| 125 | 0.3697 | 0.1136 | 0.1738 |
| 175 | 0.3757 | 0.1676 | 0.1644 |
| 250 | 0.3809 | 0.1025 | 0.1616 |
| 425 | 0.3843 | 0.0926 | 0.1493 |

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.

A blue line graph with white text

Description automatically generated with low confidence

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 15%.

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.