Data Mining Assignment 2: Recommendations using non-derivable association rules

Viktor Hura

June 27, 2023

Introduction

In this report I will describe my solutions and findings for the second assignment for Data Mining. This report will go over the dataset I used throughout the assignment. Then followed by my solutions and results for the 3 assigned sub-tasks. And finally I will provide some general conclusions and reference the code used in this assignment.

Dataset

For the dataset I chose to use the "Online Retail II" dataset from the UC Irvine Machine Learning Repository [4]:

"Online Retail II data set contains all the transactions occurring for a UK-based and registered, non-store online retail between 01/12/2009 and 09/12/2011"

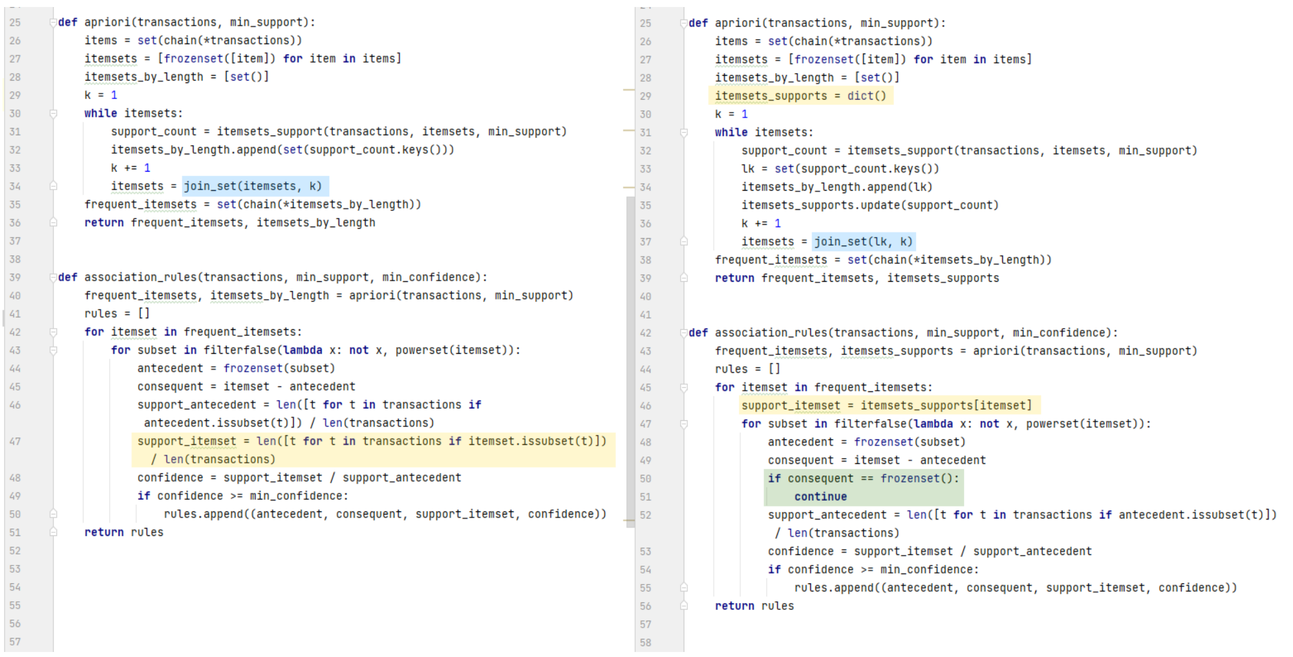
I grouped the items by a common invoice number to transform the dataset into a transactional dataset. Then I took all the transactions of more than 10 items and sampled 20% of them randomly to be used in the hold-out test set.

For this test set, I transformed these transactions into recommendation ground truths for testing the different systems. I do this by taking each transaction from this test set and randomly sampling 70% of the items to be used as "history" for a non existent user. The remaining items are then used as the ground truth for any recommendation made using this history.

This is based on the assumption that the remaining items would be a relevant recommendation for this imaginary user, since they appeared together in the same transaction. I chose the arbitrary value of 70% to give the recommender more history and thus more potential association rules being applicable. It goes without saying that it would have been desirable to have real user data in this dataset instead.

Finally I am left with a transactions dataset and this "users" dataset that can be used to evaluate the recommenders in later sections. The training set consists of 48576 transactions and the "users" set contains 5052 rows. This equates to a train/test split of about 90/10.

1 Apriori



```python
def apriori(transactions, min_support):
    items = set(chain(*transactions))
    itemsets = [frozenset([item]) for item in items]
    itemsets_by_length = [set()]
    k = 1
    while itemsets:
        support_count = itemsets_support(transactions, itemsets, min_support)
        itemsets_by_length.append(set(support_count.keys()))
        k += 1
        itemsets = join_set(itemsets, k)
    frequent_itemsets = set(chain(*itemsets_by_length))
    return frequent_itemsets, itemsets_by_length


def association_rules(transactions, min_support, min_confidence):
    frequent_itemsets, itemsets_by_length = apriori(transactions, min_support)
    rules = []
    for itemset in frequent_itemsets:
        for subset in filterfalse(lambda x: not x, powerset(itemset)):
            antecedent = frozenset(subset)
            consequent = itemset - antecedent
            support_antecedent = len([t for t in transactions if
             antecedent.issubset(t)]) / len(transactions)
            support_itemset = len([t for t in transactions if itemset.issubset(t)])
              / len(transactions)
            confidence = support_itemset / support_antecedent
            if confidence >= min_confidence:
                rules.append((antecedent, consequent, support_itemset, confidence))
    return rules
```

```python
def apriori(transactions, min_support):
    items = set(chain(*transactions))
    itemsets = [frozenset([item]) for item in items]
    itemsets_by_length = [set()]
    itemsets_supports = dict()
    k = 1
    while itemsets:
        support_count = itemsets_support(transactions, itemsets, min_support)
        lk = set(support_count.keys())
        itemsets_by_length.append(lk)
        itemsets_supports.update(support_count)
        k += 1
        itemsets = join_set(lk, k)
    frequent_itemsets = set(chain(*itemsets_by_length))
    return frequent_itemsets, itemsets_supports


def association_rules(transactions, min_support, min_confidence):
    frequent_itemsets, itemsets_supports = apriori(transactions, min_support)
    rules = []
    for itemset in frequent_itemsets:
        support_itemset = itemsets_supports[itemset]
        for subset in filterfalse(lambda x: not x, powerset(itemset)):
            antecedent = frozenset(subset)
            consequent = itemset - antecedent
            if consequent == frozenset():
                continue
            support_antecedent = len([t for t in transactions if antecedent.issubset(t)])
             / len(transactions)
            confidence = support_itemset / support_antecedent
            if confidence >= min_confidence:
                rules.append((antecedent, consequent, support_itemset, confidence))
    return rules
```

Figure 1: Differences between Chat-GPT(left) and my improved implementation(right) of Apriori algorithm

For the first task, I improved the code generated by Chat-GPT to correctly implement the Apriori algorithm, you can see some of the differences in the figure above.

The main problem that I found is that candidate itemsets were never replaced by L_k, so never pruned (shown in blue). Thus the search space was larger than it ought to be.

Another issue is that the support of the frequent itemset was being calculated redundantly for each of its subsets (highlighted in yellow). The support of the frequent itemset can be computed once and used for calculating all the confidences of the potential rules. And in fact, this support is already calculated during apriori algorithm.
So I extended Apriori to also return a dictionary containing the supports for all the frequent itemsets such that we don't have to scan the database needlessly again.

Finally there were some minor improvements, such as changing powerset and join set functions to use built-in itertools algorithms as they were generating redundant combinations and also filtering out the empty set from powerset (green).
See the code repository for all the details on these implementations.

2 Association rule based recommendations

In this section I used association rules generated by the Apriori algorithm to build a recommender. Then I tested the performance of this recommender and the impact of different ranking metrics.

For practical purposes, in this section I have made use of the "Efficient-Apriori"[1] python implementation of Apriori. This is because this implementation claims to make use of some pruning optimisations, found in the original paper. Thus it reports to be much faster [2], which I also could verify empirically.

For all tests I used a relative minimum support value of 0.0025, which corresponds to 121 transactions that an itemset must occur in to be considered relevant. For the minimum confidence value, I used 0.60, also for all tests. This generated 13251 association rules.

I tested 6 different metrics: Confidence, Lift, Conviction, Rule Power Factor, Jaccard Coefficient and Imbalance Ratio.
I also tested different ways of combining these metrics, given an item recommendations and its corresponding rules, namely taking the average of the metrics of the rules (avg), taking the maximum of the metrics (max) and finally taking the weighted maximum (wmx). Here the maximum score is weighted by $1 - \frac{1}{\#itemrules}$, such that items that have many rules in their favor, will have a better weight.

Below are the results:

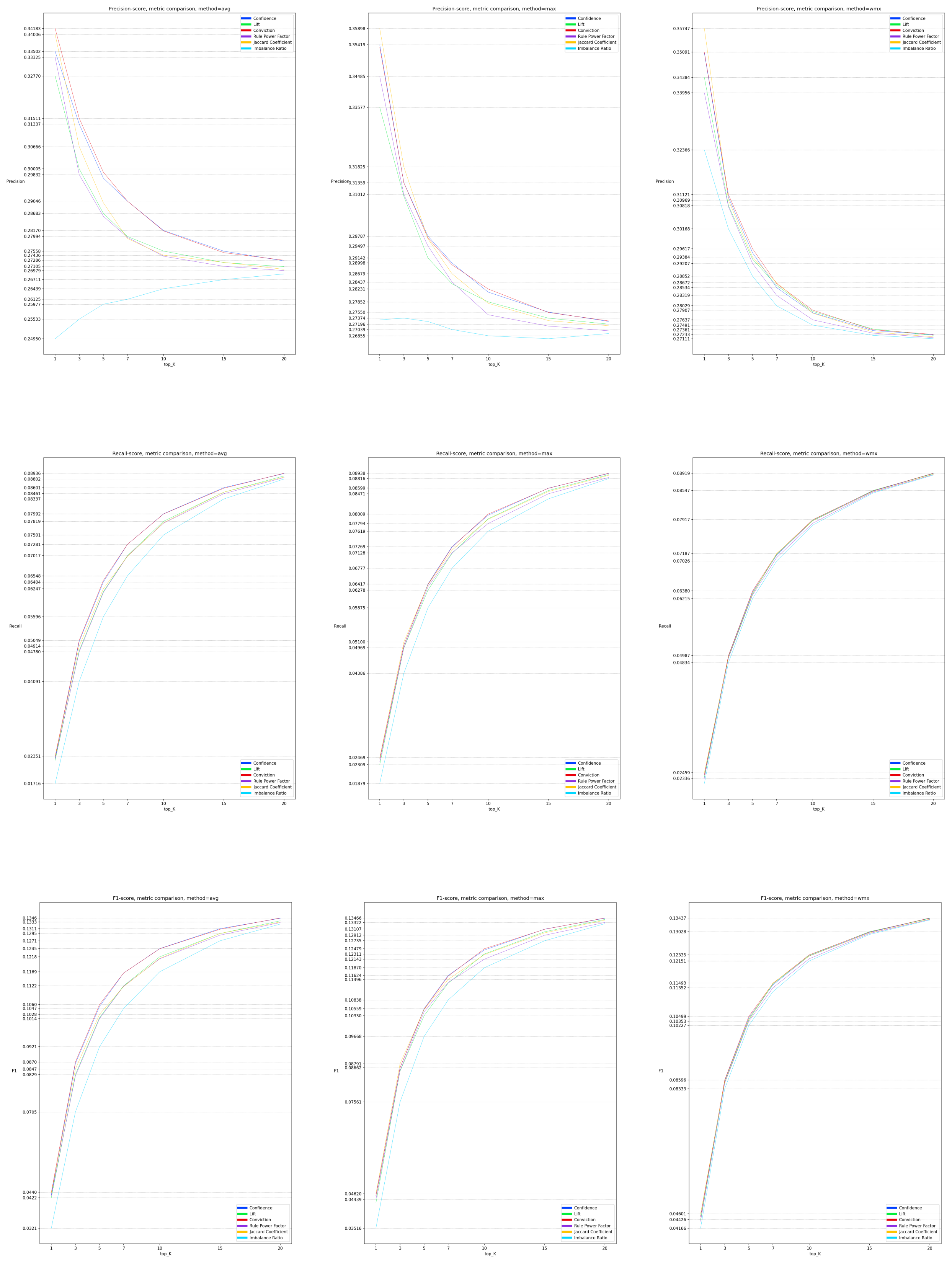


Figure 2: Comparison of the Accuracy, Precision and F1 scores for the different metrics and different methods to combine them.

As expected, increasing the amount of recommendations that we make, will increase the recall but lower our precision but the F1 score increases still.

In general, looking at all the results, we can see that the differences are quite small, this is likely because the recommendations are already not that good as evident by the highest F1 score achieved being merely 0.134.

But if I were to make conclusions based on these minor differences, you could say that taking the max of the metric is slightly better than the average and weighted max is slightly worse.
But we can also see that when using weighted max to combine metrics, the chosen metric itself matters less as the lines almost overlap each other. I suspect this is because this weight term has a larger impact on the ranking than the metric itself and this method is in effect similar to simply ranking by amount of rules that support a recommendation.

As for the metrics themselves, the differences are mostly small but we see an outlier when looking at the precision of Imbalance Ratio. It seems like this metric does not perform that well on small K values. Except for when using weighted max, where it seems to perform about the same as the other metrics, giving more credibility to the theory that the weight term is having the most effect on the ranking.
Overall Confidence and Conviction seem to be slightly better than the other metrics in most cases.

3 Non-Derivable-Itemsets based association rules

For this final part, I used the Non-Derivable-Itemsets [3] implementation to generate association rules based on these Non-Derivable Itemsets instead of the frequent datasets generated by Apriori algorithm.

For testing, I kept the same minimum support and used the max confidence metric for ranking recommendations. I also tested with different minimum confidence thresholds, below are the results:

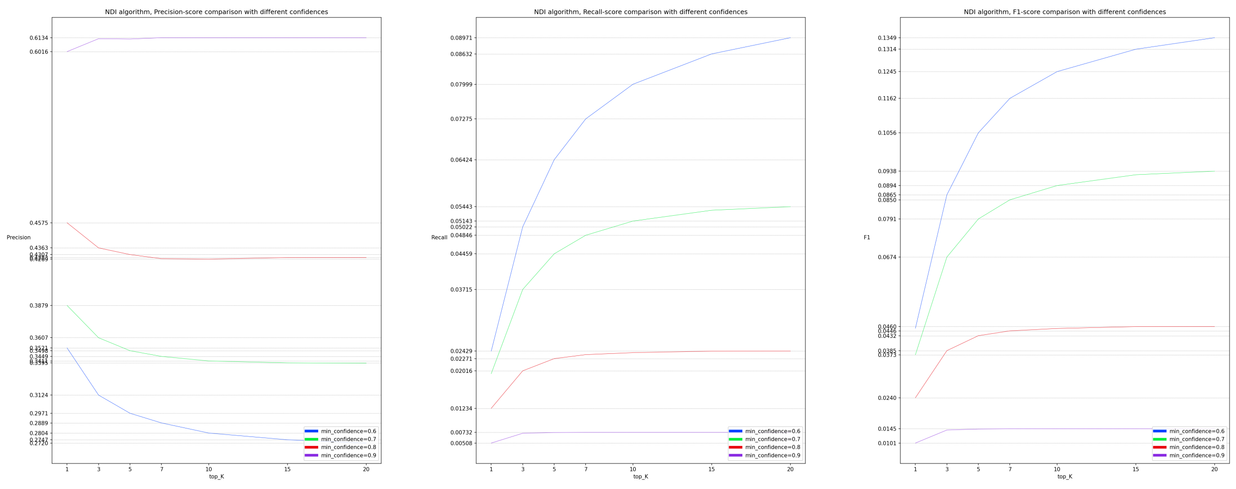


Figure 3: Precision, Recall and F1 of recommendations based on Non-Derivable-Itemsets and different confidence thresholds

We notice that increasing the confidence threshold, will improve the precision of the recommendation as the rules are of better quality, reaching as high as 0.6 precision. But the recall will suffer and the F1 score suffers as well.

As for using NDI versus frequent itemsets, at a first glance the performance of the recommendations seem as good as equal when considering the same minimum confidence and minimum support.
But this is quite remarkable, as NDI produces less itemsets and thus less rules for the same minimum support. This means that despite having less association rules, these rules appear to capture all the relevant patterns.

The benefit of NDI becomes clear when looking at the speed of the algorithm compared to Apriori. Because it considers less itemsets, it is faster to compute while empirically still achieving the same performance. This efficiency gain would allows us to lower the minimum support threshold further and still be as fast as Apriori while producing more association rules, thus likely better recommendations.

Conclusions

While the performance of these association rule based recommenders was not very impressive on the particular dataset that I used, it was nevertheless insightful to learn how these types of recommenders work and how the parameters affect them. In addition it was very interesting to see how NDI compares to Apriori and to experience the benefits of this approach in practice.

Code

The relevant code and data used for this assignment can be found in this github repository.

References

[1] Efficient-Apriori, June 2023. URL: `https://github.com/tommyod/Efficient-Apriori`.

[2] Major speedup (˜100x) on large inputs by tommyod · Pull Request #40 · tommyod/Efficient-Apriori, June 2023. URL: `https://github.com/tommyod/Efficient-Apriori/pull/40`.

[3] Toon Calders and Bart Goethals. Non-derivable itemset mining. *Data Min. Knowl. Disc.*, February 2007. `doi:10.1007/s10618-006-0054-6`.

[4] Daqing Chen. Online Retail II. UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5CG6D.