CS245 Homework 1

July 31, 2024

Name: Kelly Furuya

1 Problem 1: The Apriori Algorithm

1.1 Given a minimum support of 3, apply the Apriori algorithm to the above transaction dataset to find frequent itemsets

For this question, the *mlxtend* Python package was used to implement the Apriori algorithm using the code below that is based on the examples provided in the User Guide of the package's documentation.

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
import pandas as pd


T_list = [["Python", "Machine Learning"],
          ["Introduction to AI", "JavaScript", "C++"],
          ["Python", "Machine Learning", "Data Science", "Introduction to AI", "
                                                JavaScript"],
          ["Mathematics", "Machine Learning"],
          ["Python", "Data Science", "Introduction to AI", "Machine Learning"],
          ["Python", "Data Science"],
          ["Mathematics", "Machine Learning", "Data Science"],
          ["Python", "Mathematics"],
          ["Python", "Machine Learning", "Introduction to AI"],
          ["Introduction to AI", "Mathematics", "C++"]]

te = TransactionEncoder()
te_transform = te.fit(T_list).transform(T_list)
T_df = pd.DataFrame(te_transform, columns = te.columns_)

freq_itemsets = apriori(T_df, min_support = 0.3, use_colnames = True)
freq_itemsets
```

The code then provides the frequent itemsets with a minimum support of 0.3 shown in Table 1.

Index	Support	Itemsets
0	0.4	(Data Science)
1	0.5	(Introduction to AI)
2	0.6	(Machine Learning)
3	0.4	(Mathematics)
4	0.6	(Python)
5	0.3	(Data Science, Machine Learning)
6	0.3	(Data Science, Python)
7	0.3	(Introduction to AI, Machine Learning)
8	0.3	(Introduction to AI, Python)
9	0.4	(Python, Machine Learning)
10	0.3	(Introduction to AI, Python, Machine Learning)

Table 1: Frequent itemsets with a minimum support of 3

This means there are a total of 11 frequent items/itemsets for the list of transactions. We can see that Python and Machine Learning were the most frequent, each with a support of 6. It is then not surprising that (Python, Machine Learning), has the highest support out of the 2-itemsets and the only frequent 3-itemset also contains these two items.

If we were to perform the Apriori algorithm by hand, we would first loop through the transactions to the find the supports for each unique item to determine which are frequent. In this case, all of the items except JavaScript and C++ are frequent. We can then begin to recursively form all potential 2-itemset candidates from the remaining frequent items, and then scan through the transactions again to find the supports of these new candidates. After checking the supports, we would find that only [Python, Machine Learning], [Python, Data Science], [Machine Learning, Introduction to AI], and [Machine Learning, Data Science] are the only frequent 2-itemsets. With these 2-itemsets, we then try to form 3-itemsets that are not supersets of any of the infrequent 2-itemsets. We can only create two in this case, [Python, Machine Learning, Data Science] and [Python, Machine Learning, Introduction to AI], of which only the latter is frequent. This would then give us the same list of requent itemsets as Table 1.

1.2 Generate association rules based on the frequent itemsets with a minimum confidence level of 60 percent

Again, *mlxtend* and its User Guide are used to generate association rules with a minumum confidence level of 60 percent.

```
from mlxtend.frequent_patterns import association_rules

association_rules(freq_itemsets, metric="confidence", min_threshold = 0.6)
```

The code then provides the following association rules in Table 2.

Index	**Antecedents**	**Conseqeunts**	**Support**	**Confidence**
0	(Data Science)	(Machine Learning)	0.3	0.75
1	(Data Science)	(Python)	0.3	0.75
2	(Introduction to AI)	(Machine Learning)	0.3	0.6
3	(Introduction to AI)	(Python)	0.3	0.6
4	(Python)	(Machine Learning)	0.4	0.67
5	(Machine Learning)	(Python)	0.4	0.67
6	(Introduction to AI, Python)	(Machine Learning)	0.3	1.0
7	(Introduction to AI, Machine Learning)	(Python)	0.3	1.0
8	(Python, Machine Learning)	(Introduction to AI)	0.3	0.75
9	(Introduction to AI)	(Python, Machine Learning)	0.3	0.6

Table 2: Association rules with a confidence of at least 0.6

If we wanted to determine these association rules by hand, we would first look at the frequent itemset list. We can see that there are five 2-itemsets and one 3-itemset. Then we find the confidence between each item in each of the 2-itemsets in both directions. Once all of the 2-itemsets have been checked, we then check for association rules for the 3-itemset where either the antecedent or consequent is a 2-itemset and the other is a 1-itemset. The supports from the list of frequent itemsets can be used to speed up the confidence calculations and will result in the same rules shown in Table 2.

1.3 Discuss the key strengths and limitation so fthe Apriori algorithm

When compared to brute force methods that require multiple scans of the database to check every potential candidate, the Apriori algorithm can reduce the number of scans by eliminating potential candidates without scanning the database for their support. It accomplishes this by making use of the fact that if an item or itemset is not frequent, then all of its supersets will also not be frequent, thus eliminating them as potential candidates. For large or complex databases, scans can be expensive in terms of both resources and time, meaning any reduction on the number of scans can be beneficial, and the relative ease of this algorithm makes it simpler to implement and use than others.

Consequently, a downside of the Apriori algorithm is that, in the worst case scenario, it can still require a large number of scans if the itemsets are large or if too many itemsets are marked as frequent. It also stiill takes up a consider amount of computing power by itself in order to calculate potential candidates for larger itemsets which can quickly grow exponentially as the dataframe or itemsets get larger. Another downside is that the entire database must be scanned for the algorithm to complete.

2 Problem 2: FP-Tree and FP-Growth

2.1 Construct the FP-Tree for the given transaction dataset. Describe the process and provide a visual representation of the final tree.

To generate an FP-Tree, there are two steps. The step is to scan to find the frequent items and their supports so we can then list them in descending order to create the F-list. The second step is to form the tree itself. The tree starts with the root and has each immediate child ordered from left to right in descending order. The database is then scanned again, ordering each frequent item in each transaction with respect to the F-list. If the first item of the transaction is not already directly linked to the root, it is added as a new child with a counter of 1 and a connection is formed to the root. If it is already directlly linked, the counter for that item is incremented. The rest of the transaction is then examined item by item, either incrementing a counter if a corresponding connection already exists between the current and prior item in the FP-tree or a new node and connection are formed. This process is reapeated for all transactions. The last step is to take the header table which contains the F-list and form side-links to and between each corresponding node.

The resulting FP-Tree from the data given is shown below in Figure 1.

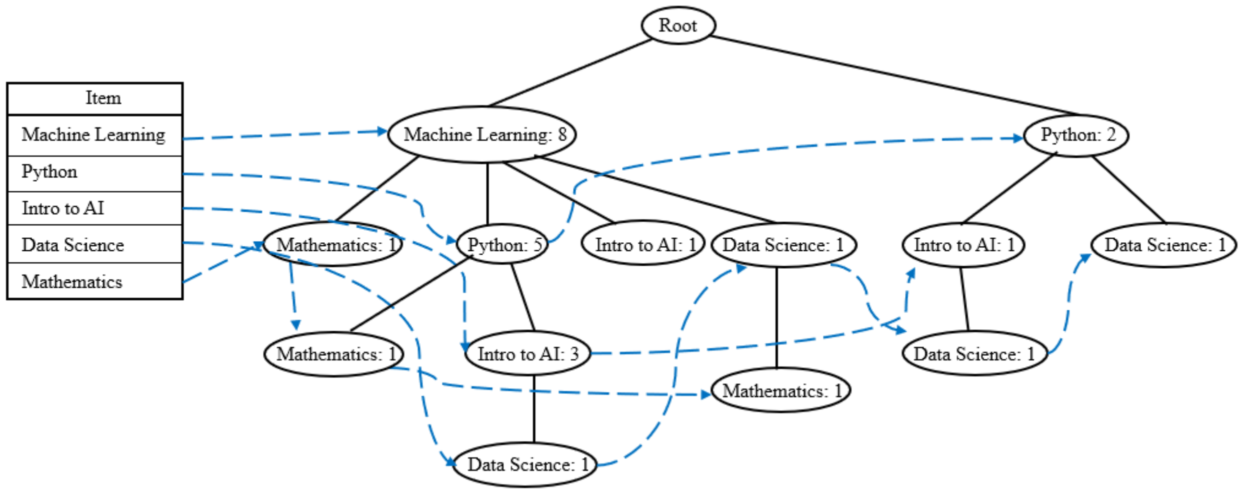

Figure 1: FP-Tree with header table

2.2 Based on the FP-Tree, use the FP-Growth algorithm to generate the frequent itemsets

The first step is to find patterns that contain the item with the lowest support, Mathematics, in this case. The FP-Tree shows us that there are three Mathematics nodes so there are three patterns, [Machine Learning, Mathematics], [Machine Learning, Python, Mathematics], and [Machine Learning, Data Science, Mathematics]. All three patterns only appear once. This means that only the only frequent pattern is Mathematics iteslf, so now we need to recursively mine using the local frequent items. If we count the support of [Machine Learning, Mathematics] and [Python, Mathematics], regardless of the tree structure, we can see that [Machine Learning, Mathematics] has a support of 3 and [Python, Mathematics] has a support of 1. This means that the list of frequent itemset list, currently, is [[Mathematics], [Machine Learning, Mathematics]].

The next step is to look for patterns in the tree that contain Data Science but not Mathematics. These patterns are [Machine Learning, Python, Intro to AI, Data Science], [Machine Learning, Data Science], [Python, Intro to AI, Data Science], and [Python, Data Science]. Once again, none of these patterns are frequent, meaning we can now move on to recursively mining using the local frequent items. [Machine Learning, Data Science] has a support of 2, [Python, Data Science] has a support of 3, [Intro to AI, Data Science] has a support of 2, [Machine Learning, Python, Data Science] has a support of 1, [Machine Learning, Intro to AI, Data Science] has a support of 1, and [Python, Intro to AI, Data Science] has a support of 2. This means the frequent itemset list is now

[[Mathematics], [Machine Learning, Mathematics], [Data Science], [Python, Data Science]].

Next, we look for patterns with Intro to AI but not Mathematics or Data Science. The patterns in the tree are [Machine Learning, Python, Intro to AI] with a support of 3, [Machine Learning, Intro to AI] with a support of 1, and [Python, Intro to AI] with a support of 1. Only the first pattern is frequent so now we recursively mine. [Machine Learning, Intro to AI] has a support of 4 and [Python, Intro to AI] has a support of 4 which means the frequent itemset list is now [[Mathematics], [Machine Learning, Mathematics], [Data Science], [Python, Data Science], [Intro to AI], [Machine Learning, Python, Intro to AI], [Machine Learning, Intro to AI], [Python, Intro to AI]].

Now we look for patterns with Python that don't have Intro to AI, Data Science, or Mathematics. The only pattern is [Machine Learning, Python] with a support of 5.This means the list is now [[Mathematics], [Machine Learning, Mathematics], [Data Science], [Python, Data Science], [Intro to AI], [Machine Learning, Python, Intro to AI], [Machine Learning, Intro to AI], [Python, Intro to AI], [Python], [Machine Learning, Python]].

Lastly, since Machine Learning is the most frequent item on the list, it therefore does not have a parent above it anywhere in the tree aside from the Root. This means that the only pattern we add to the list is [Machine Learning].

The final list is shown below in Table 3 in descending support order.

Frequent Patterns	**Support**
Machine Learning	8
Python	7
Machine Learning, Python	5
Intro to AI	5
Data Science	4
Machine Learning, Intro to AI	4
Python, Intro to AI	4
Mathematics	3
Machine Learning, Python, Intro to AI	3
Machine Learning, Mathematics	3
Python, Data Science	3

Table 3: Frequent Item list using the FP-Tree and FP-Growth algorithm

2.3 Compare the key strengths and limitations of the FP-Growth algorithm and the Apriori algorithm

The FP-Growth algorithm, in most cases, will find all frequent patterns in fewer scans than the Apriori algorithm since it only requires 2 total scans while the Apriori algorithm can require as many scans as the length of the longest frequent pattern. This means that time and resources can be saved from fewer scans.

The tree structure itself also allows us to quickly and easily search through and find frequent patterns since the supports are saved at each node. Apriori still requires a scan to determine the support of any new superset candidates. This also means that the Apriori algorithm requires us to continually access the database while FP-Growth stores all the relevant information in a much more compact way that is less resource intensive to access and look through. This compactness allows the algorithm to generally scale nicely for both large and small patterns.

Despite generally scaling well, the FP-Growth algorithm can end up taking up too much memory as the tree grows if there is a large number of unique frequent items. The larger tree requires more and more working memory to search through, especially if there are many unique frequent items and the minimum support is very low, requiring a scan of very long branches.

Another noteable downside of the FP-Growth algorithm is that the construction and use of tree can be less intuative than the Apriori algorithm. Simple algorithms are generally easier to implement and troubleshoot than more complex algorithms.

3 Problem 3: Constraint-based Frequent Pattern Mining

3.1 Generate frequent itemssets from the given dataset that comply with the budget and timeslot constraint using either the Apriori or FP-Growth algorithms. Assume that only classes that start at the same time introduce conflicts.

We can use the same code from Problem 1 with slight adjustments to the transaction list in order to use the Apriori algorithm to find the frequent itemset without any constraints, shown in Table 4.

Index	**Support**	**Itemsets**
0	0.4	(Data Science)
1	0.5	(Introduction to AI)
2	0.8	(Machine Learning)
3	0.3	(Mathematics)
4	0.7	(Python)
5	0.3	(Data Science, Python)
6	0.4	(Introduction to AI, Machine Learning)
7	0.4	(Introduction to AI, Python)
8	0.3	(Machine Learning, Mathematics)
9	0.5	(Python, Machine Learning)
10	0.3	(Introduction to AI, Python, Machine Learning)

Table 4: Frequent itemsets with a minimum support of 3

Next we want to see which classes have a time conflict. The given times indicate the the only conflicst are [Python, JavaScript, C++] for 8am and [Mathematics, Data Science] for 9am. Both JavaScript and C++ are infrequent items and can therefore be ignored in this instance. There are also no frequent itemsets with Mathematics and Data Science in Table 4, meaning we can also ignore this time constraint.

Lastly, we will look at the budget constraint and see which, if any, itemsets in Table 4 violate the budget of $3800. Table 5 shows each itemset along with the total cost of all the textbooks.

Index	**Support**	**Itemsets**	**Total Cost**
0	0.4	(Data Science)	$500
1	0.5	(Introduction to AI)	$1000
2	0.8	(Machine Learning)	$3000
3	0.3	(Mathematics)	$750
4	0.7	(Python)	$200
5	0.3	(Data Science, Python)	$700
6	0.4	(Introduction to AI, Machine Learning)	$4000
7	0.4	(Introduction to AI, Python)	$1200
8	0.3	(Machine Learning, Mathematics)	$3750
9	0.5	(Python, Machine Learning)	$3200
10	0.3	(Introduction to AI, Python, Machine Learning)	$4200

Table 5: Frequent itemsets with the total cost of their textbooks

From Table 5, we can see that two itemsets violate the budget constraint, [Introduction to AI, Machine Learning] and [Introduction to AI, Python, Machine Learning], as they cost $4000 and $4200, respectively. Table 6 shows the final frequent itemset list along with their prices.

Index	**Support**	**Itemsets**	**Total Cost**
0	0.4	(Data Science)	$500
1	0.5	(Introduction to AI)	$1000
2	0.8	(Machine Learning)	$3000
3	0.3	(Mathematics)	$750
4	0.7	(Python)	$200
5	0.3	(Data Science, Python)	$700
6	0.4	(Introduction to AI, Python)	$1200
7	0.3	(Machine Learning, Mathematics)	$3750
8	0.5	(Python, Machine Learning)	$3200

Table 6: Frequent itemsets with the total cost of their textbooks

If we were to use the Apriori algorithm ourselves by hand, we would eliminate each itemset that violates either constraint as we go, reducing the number of condidates we must search through. For example, once we reach the second scan of the database, we can begin to eliminate potential candidates based on whether or not the 2-itemsets violate the time or budget constraints before moving on to find 3-itemset candidates. In this case, the code provides a list of all the frequent itemsets without taking into account the constraints so it becomes a simple matter of removing the violations.

3.2 Discuss whether the budget constraint mentioned above is anti-monotone or not. Justify your response and explain why it is useful to consider.

The budget constraint is an anti-monotone constraint. For example, the 2-itemset [Machine Learning, Introduction to AI] has a total cost of $4000 which is more than the $3800 budget. This means that any other itemsets containing both of these are also going to violate the budget constraint as adding more classes will only increase the total cost of textbooks further. In other words, for any combination of classes whose combined book cost violates the budget constraint, any superset of these combinations will also violate the budget constraint, making it an anti-monotone constraint.

Recognizing that any given constraint is anti-monotone can be useful in reducing the time and resources needed to find all the frequent itemsets. If we already know that a certain pattern violates the constraint, then there is no need to check any supersets of that pattern as they will all also violate the constraint. For example, if we already know that the itemset [Machine Learning, Introduction to AI] already violates the budget constraint, then we could look at the itemset [Introduction to AI, Python, Machine Learning] and already know that it will also violate the budget constraint without adding up the price of all the textbooks.

3.3 Describe a constraint that could be applied to this dataset that will be either (a) anti-monotone if you answered monotone for the previous question, or (b) monotone if your previous answer was anti-monotone. Explain why this type of constraint is useful to consider.

An example of a monotone constraint would be that the least expensive textbook must be less than \$500. If an itemset contains a textbook that is less than \$500, then any superset of that itemset will also meet that constraint. Even if a cheaper book is added to the itemset, the constraint is still satsified and not violated.