Association Rule Learning with Applications to Market Basket Analysis

Project Report – APMA 940: Mathematics of Data Science

Javier A. Almonacid*

April 18, 2023

Abstract

Market basket analysis is a popular association rule learning technique for analyzing large data sets of transactional data. The main idea is to first look for combinations of items that occur together frequently in transactions (frequent itemset generation), to then identify association rules over these combinations. Using this information businesses can, for instance, make decisions about potential discounts on a certain product or how to place these items in the store. In an online setting, this is typically seen in "recommended products based on your purchase" sections.

In this work, we describe the association rule learning problem applied to market basket analysis. We review two methods for frequent itemset generation: Apriori and FP-Growth. In addition, we discuss metrics that are commonly used to describe the usefulness of the results. Finally, we perform market basket analysis on two sets of data, a small one coming from supermarket transactions and a larger one from an online wholesale retailer.

1 Introduction

Unsupervised learning is a type of learning in which data is not labelled, so it is usually of interest to find patterns, structures, and relationships in the data on its own. Clustering, dimensionality reduction, matrix completion, and the generation of association rules are just some examples of unsupervised learning [13]. The latter one will be the main focus of this work.

Association rule learning has its roots in the work by Agrawal, Imieliński, and Swami in 1993 [1]. Here, using a set of transactions generated by point-of-sale systems in a supermarket, the authors were able to obtain association rules between items in the database. An example of such rule is, for example, "90% of customers that purchase bread and butter also purchase milk". The process of obtaining association rules from transactional databases would then be known as *market basket analysis* (MBA). However, it did not take long for researchers to realize that this same technique could be used in other areas as well. Nowadays, this same idea can be found in diverse applications such as intrusion detection in cybersecurity [5], microbial association networks [8], analysis of microblogging online texts (such as *tweets*) [7], and ad-targeting in social media [11], to name a few.

A large part of the work on devising association rules goes towards the discovery of frequently occurring events, patterns, or items in the data set, which are then used to generate these rules.

*Department of Mathematics, Simon Fraser University, Burnaby, BC V5A 1S6, Canada.

TID	Items
1	Bread, Milk
2	Bread, Diapers, Beer, Eggs
3	Milk, Diapers, Beer, Cola
4	Bread, Milk, Diapers, Beer
5	Bread, Milk, Diapers, Cola

Table 2.1: Example of a transactional database containing 5 transactions and 6 items. Each transaction is uniquely identified by a transaction ID (TID).

This task is commonly known as "frequent itemset mining" and was an active research area in the mid 90s and early 2000s [12]. The first algorithm was precisely introduced in the work by Agrawal et al. [1], which today is known as *Apriori*. This first approach was designed to work on a set of binary attributes (called items) and a database of transactions where each transaction was represented as a binary vector. To find frequent itemsets (i.e. sets of items), the method follows a level-wise breadth first methodology where candidates are formed from already generated frequent itemsets. This reduces considerably the search space (for a list of n items, there can be up to $2^n - 1$ itemsets, which is a prohibitively large number for stores that carry thousands of items). However, the number of passes over the data increases with the number of items in the frequent itemsets.

Although several improvements were proposed in the years after the introduction of the original Apriori method [1] (see, for instance, [2, 4, 15]), none of these would reduce significantly the number of times the database needed to be read. In 2000, Han et al. [9] proposed a new algorithm called *FP-Growth* using a novel frequent pattern tree structure for storing compressed information about patterns. In this case, the database is read twice: first to count how frequent single items are and then to find frequent itemsets by traversing the resulting tree structure. This is nowadays a key algorithm to compare with. Some other algorithms worth mentioning are ECLAT (Equivalent CLAss Transformation) [18] in which the frequent itemsets are identified by intersecting sets of transaction IDs and dECLAT (diff ECLAT) [19] which keeps track of the differences between transactions. In terms of sequential algorithms, Apriori, FP-Growth, ECLAT, and dECLAT represent the most important algorithms in the field of frequent itemset mining. Moreover, as market basket analysis continues to be used in both brick-and-mortar and online stores, advances in high-performance computing throughout the last two decades now allow analysts to process even larger amounts of data [12].

In this work, we first describe the association rule learning problem in the context of MBA and the two steps involved in its solution: frequent itemset mining (FIM) and rule generation. Then, we describe two algorithms for FIM: Apriori and FP-Growth. Finally, we perform two sets of MBAs. The first one is done on a small database of transactions at a supermarket [3], whereas the second one is performed on a much larger and realistic data set from an online retailer, whose many of its customers are wholesalers [6].

2 Association rule analysis

Let us begin by establishing some definitions that will be used throughout this project. Most of the content in this section has been extracted from [10, 16].

Consider $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ to be a set of binary attributes called *items*. Let $\mathcal{T}$ be a database containing m uniquely identified transactions in as in Table 2.1.

Definition 2.1 (Association rule)**.** Let $X, Y \subset \mathcal{I}$ be sets of items (also called *itemsets*) such

that $X \cap Y = \emptyset$. An *association rule* is a statement of the form

$$X \Rightarrow Y$$

where X and Y are respectively the *antecedent* and *consequent* of the rule.

Several metrics have been developed to measure the importance of a rule in a database $\mathcal{T}$, ranging from simple quotients to more involved quantities [17]. Unlike in supervised learning where there is a clear measure of success using the training data, the effectiveness of unsupervised learning algorithms is evaluated in more heuristic and application-dependent terms. Therefore, rather than finding the "best" outcome of an algorithm, we will be interested in learning valuable information from data. Here are some of the simplest and most commonly used metrics.

Definition 2.2 (Support of an itemset)**.** Let $X \subset \mathcal{I}$ be an itemset. We define the *absolute support* of X as the number of transactions that contain the items in the itemset X, that is,

$$\text{supp}_0(X) := \left|\left\{T \in \mathcal{T} : T \supseteq X\right\}\right| \in [0, \infty).$$

Then, we define the *support* of X as:

$$\text{supp}(X) := \frac{\text{supp}_0(X)}{m},$$

where m is the total number of transactions in the database.

Definition 2.3 (Metrics)**.** Let $X, Y \subset \mathcal{I}$, $X \cap Y = \emptyset$. Then, for the association rule $X \Rightarrow Y$ we define the following metrics.

1. *Support* (introduced in [1]), which measures how frequently the itemset $X \cup Y$ appears in the database:
$$\text{supp}(X \Rightarrow Y) = \text{supp}(X \cup Y) \in [0, 1].$$

2. *Confidence* (introduced in [2]), which measures the probability that a transaction containing the antecedent X will also contain the consequent Y:
$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \Rightarrow Y)}{\text{supp}(X)} = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} \in [0, 1].$$

3. *Lift* (introduced in [4]), which measures how much more often the antecedent and consequent of the rule occur together than we would expect if they were statistically independent:
$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \Rightarrow Y)}{\text{supp}(X)\text{supp}(Y)} = \frac{\text{conf}(X \Rightarrow Y)}{\text{supp}(Y)} \in [0, \infty).$$
Thus, if $\text{lift}(X \Rightarrow Y) = 1$ then X and Y are independent.

4. *Leverage* (introduced in[14]), which measures the difference between the observed frequency of X and Y appearing together and the frequency that would be expected if A and C were completely independent:
$$\text{lev}(X \Rightarrow Y) = \text{supp}(X \Rightarrow Y) - \text{supp}(X)\text{supp}(Y) \in [-1, 1].$$
Thus, if $\text{lev}(X \Rightarrow Y) = 0$ then X and Y are independent of each other.

5. *Conviction* (introduced in [4]), which measures how dependent the consequent is on the antecedent:
$$\text{conv}(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \Rightarrow Y)} \in [0, \infty].$$
Therefore, the higher the conviction value is, the more dependent the consequent is on the antecedent, and thus it is closely related to the concept of lift above.

For example, consider the rule {Beer} ⇒ {Diapers} from the data in Table 2.1. We have that supp({Beer} ⇒ {Diapers}) = 0.6, meaning that Beer and Diapers appeared together in 60% of the transactions. Then, conf({Beer} ⇒ {Diapers}) = 1, which means that 100% of the times Beer was purchased, Diapers were purchased as well. Moreover, lift({Beer} ⇒ {Diapers}) = 1, lev({Beer} ⇒ {Diapers}) = 0.12, and conv({Beer} ⇒ {Diapers}) = ∞. This suggests that it is highly likely that Beer and Diapers *will* be purchased together. As a consequence, a price analyst could suggest increasing the price of Diapers to boost revenue, or a store manager could suggest placing these items far apart in the store to boost the sales of other products.

The *market basket analysis* problem then consists in finding all association rules R in a transactional database $\mathcal{T}$ such that

$$\text{supp}(R) \geq minsupp \quad \text{and} \quad \text{conf}(R) \geq minconf, \tag{2.1}$$

where $minsupp, minconf \in (0, 1)$ are the corresponding support and confidence thresholds.

A brute-force approach to learn association rules could be to compute the support and confidence of every possible rule. However, this is prohibitively expensive because, for a data set containing d items, there are $3^d - 2^{d+1} + 1$ possible rules (cf. [16, Equation 6.3]). A more efficient approach is to divide this problem into two tasks:

1. **Frequent itemset generation** where we want to learn combinations of items $\mathcal{K} \subset \mathcal{I}$ that have support above *minsupp*. We call these sets *frequent itemsets*. This tends to be the most computationally expensive part of association rule learning.
2. **Rule generation** where we want to extract all rules with confidence above the $minconf$ threshold using only the itemsets found in the previous step.

Before continuing with the description of these two tasks, let us briefly emphasize that the items here are *binary* attributes, i.e. they are either purchased or not. Therefore, we will not be interested in, for instance, *how many* of each item have been purchased or *how much* money customers spent on each of them. Creating association rules involving these rather continuous variables is known as *quantitative association rule learning*. This type of problem can be solved by splitting the continuous range into several intervals which can then be used to define binary versions of the continuous variables. We refer the reader to [16, Chapter 7] for more information on this topic.

3 Frequent itemset generation

Two well-known algorithms to find frequent itemsets (and patterns in general) are *Apriori* and *FP-Growth*. These represent the basis of many other algorithms in the frequent itemset mining literature [12].

3.1 Apriori

One of the first algorithms to find frequent itemsets was proposed by Agrawal, Imieliński, and Swami in 1993 [1], which was later formalized in [2] with the name *Apriori*. To overcome the curse of dimensionality (recall that a set of n items can generate up to $2^n - 1$ frequent itemsets), the main idea in this method is to discard multiple itemsets at once based on the following property.

Lemma 3.1 (Anti-monotonicity)**.** Let $\mathcal{L}, \mathcal{K}$ be itemsets such that $\mathcal{L} \subseteq \mathcal{K}$. Then $\text{supp}(\mathcal{L}) \geq \text{supp}(\mathcal{K})$.

Algorithm 3.1: Apriori algorithm (cf. [16, Algorithm 6.1]).

Data: $\mathcal{T}$, a database of transactions; *minsupp*, the minimum support threshold.
Result: $\mathcal{L}$, frequent itemsets in $\mathcal{T}$.

```
1  k = 1;
2  F_k = {{i} | i ∈ I and supp({i}) ≥ minsupp}   (Find all frequent 1-itemsets);
3  while F_k ≠ ∅ do
4      k = k + 1;
5      C_k = apriori_gen(F_{k-1})   (Candidate generation);
6      foreach transaction T ∈ T do
7          D_T = {C ∈ C_k : C ⊆ T}   (Identify all candidates that belong to T);
8          foreach candidate itemset c ∈ D_T do
9              supp_0(c) = supp_0(c) + 1   (Increment absolute support by 1);
10     F_k = {c | c ∈ C_k and supp(c) ≥ minsupp}   (Extract the frequent k-itemsets);
11 return L = ⋃ F_k;
```

This means that, if an itemset is frequent, then all of its subsets must also be frequent. Therefore, many candidate itemsets can be discarded without the need to compute their support (this is called *support-based pruning*). The core of this algorithm (which is shown in full in Algorithm 3.1) can be explained using the example database in Table 2.1. Let us assume that the absolute minimum support threshold is 3 (i.e. $minsupp = 3/5 = 0.6$). We also refer to itemsets with k items as *k-itemsets*.

Item	supp_0	Frequent?
Beer	3	Yes
Bread	4	Yes
Cola	2	No
Diapers	4	Yes
Milk	4	Yes
Eggs	1	No

Table 3.1: Candidate 1-itemsets.

1. Consider every item as a candidate 1-itemset and count their support. Discard infrequent items.

Itemset	supp_0	Frequent?
{Beer, Bread}	2	No
{Beer, Diapers}	3	Yes
{Beer, Milk}	2	No
{Bread, Diapers}	3	Yes
{Bread, Milk}	3	Yes
{Diapers, Milk}	3	Yes

Table 3.2: Candidate 2-itemsets.

2. Generate candidate 2-itemsets using only the frequent 1-itemsets because, by Lemma 3.1, all supersets of the infrequent 1-itemsets must be infrequent. Then, count the support of these 2-itemsets and discard infrequent ones.

Itemset	supp_0	Frequent?
{Beer, Diapers, Milk}	3	Yes

Table 3.3: Candidate 3-itemsets.

3. Continue generating k-itemsets from frequent $k-1$ itemsets, discarding those that are deemed infrequent.

The pseudocode in Algorithm 3.1 contains the function `aprori_gen` which generates the candidate itemsets. Instead of tackling the problem using a brute-force solution, one common approach to generate a candidate k-itemset is to merge two frequent $(k-1)$-itemsets, but only if their first $k-2$ items are identical. This only requires an additional pruning step to ensure that the remaining $k-2$ subsets of the candidate are frequent [16].

In general, there are several factors that can affect the computational complexity of the Apriori algorithm, which also show the overall complexity of the MBA problem:

TID	Items
1	a, b
2	b, c, d
3	a, c, d, e
4	a, c, e
5	a, b, c

TID	Items
6	a, b, c, d
7	a
8	a, b, c
9	a, b, d
10	b, c, e

Table 3.4: Example database including 10 transactions and 5 items.

1. Support threshold: lowering the support threshold *minsupp* leads to more itemsets deemed frequent.

2. Number of items: as the number of items increase, more space in memory is needed to store the support counts (improved versions of the Apriori algorithm use hash trees for this task).

3. Average transaction width: this is problematic when the database is not sparse (i.e. the ratio of items bought in a transaction vs. the total amount of items is not very low). This can increase the maximum size of the frequent itemset, and therefore the algorithm requires more iterations to finish.

4. Number of transactions: because the Apriori algorithm makes repeated passes over the database, a larger number of transactions will clearly increase its run time.

Hardware limitations typically restrict improvements on how fast the data can be read. Instead, one may try to reduce as much as possible the amount of times the data needs to be read. This is the main idea in the *FP-Growth* algorithm, which we talk about next.

3.2 FP-Growth

This alternative approach proposed by Han et al. in 2000 [9] consists in encoding the database into a compact tree structure so that frequent itemsets can be extracted from this tree (which is stored in memory) rather than from the database itself (stored on disk). Let us describe how the algorithm works by using the example database in Table 3.4. Most of the material in this section will be extracted from [9, 16]. In particular, the figures are extracted from [16, Section 6.6].

3.2.1 Constructing the FP-tree

Consider the database in Table 3.4. The algorithm to construct the FP-tree is described as follows.

Item	Support
a	8
b	7
c	6

Item	Support
d	5
e	3

Table 3.5: Support count for each item.

1. Scan the database to determine the support of each item. Discard infrequent items. Because we will assume an (absolute) support of 2, all items in this database are deemed frequent.

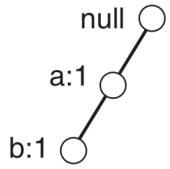

Figure 3.1: After reading TID=1.

2. Start a second pass over the data. After reading the first transaction (TID=1), the nodes labelled a and b are created, and a path $null \rightarrow a \rightarrow b$ is created. Their frequency counts are also counted (in this case, 1 each).

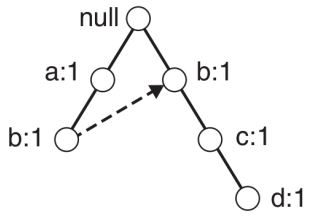

Figure 3.2: After reading TID=2.

3. After reading the second transaction (TID=2), a new set of nodes are created along with the path $b \rightarrow c \rightarrow d$. Each item has a frequency of 1 along this path. A pointer is created between common items (in this case, item b).

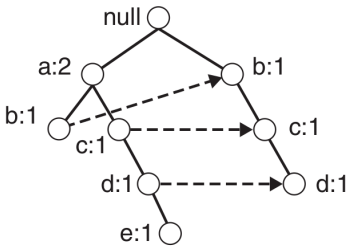

Figure 3.3: After reading TID=3.

4. For the next transaction $\{a, c, d, e\}$ (TID=3), because its first element is a, there is no need to create another node for a. Instead, the path $null \rightarrow a \rightarrow c \rightarrow d \rightarrow e$ is created. The frequency count is included in these nodes. For node a, this count is increased by one. Pointers are created between common items with previous transactions (in this case, items c and d).

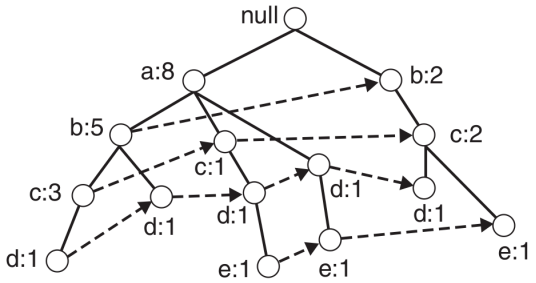

Figure 3.4: After reading TID=10.

5. The process continues until all transactions have been read. The resulting FP-tree is shown to the left.

3.2.2 Generating frequent itemsets

To generate frequent itemsets, the FP-tree is explored in a bottom-up fashion, starting with itemsets ending in e first, then d, c, b, and a. Every itemset has been mapped to a different path in the tree, so to generate the itemsets whose last item is, for instance, e (e.g. $\{e\}$, $\{d, e\}$, $\{c, e\}$ and so on) we have to focus only on those paths that end in e.

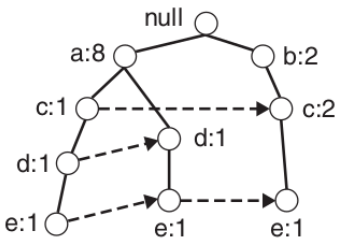

Figure 3.5: Prefix paths ending in e.

1. First, gather all paths containing the node e. These initial paths are called *prefix paths* (see the figure to the left).

2. Decide whether $\{e\}$ is a frequent itemset. From Table 3.5, the support of $\{e\}$ is 3. Because we are assuming a minimum (absolute) support of 2, the itemset $\{e\}$ is deemed frequent.

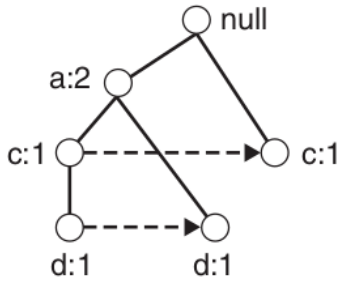

Figure 3.6: Conditional FP-tree for e.

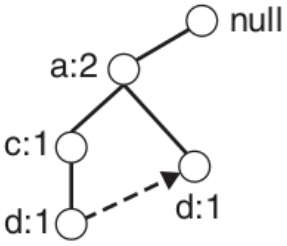

Figure 3.7: Prefix paths ending in de.

Figure 3.8: Conditional FP-tree for de.

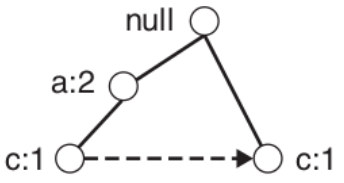

Figure 3.9: Prefix paths ending in ce.

3. Construct the *conditional FP-tree* for this itemset as follows:

3.1. Update the support counts along the prefix paths. For instance, the rightmost path in Figure 3.5: $null \rightarrow b : 2 \rightarrow c : 2 \rightarrow e : 1$, includes the transaction $\{b, c\}$ that does not contain e. Hence, the support count for b and c must be decreased by one unit.

3.2. Remove the node e. This can be safely done as the support counts have been properly updated and the subproblems for itemsets ending in de, ce, be and ae no longer need information about the node e.

3.3. Remove infrequent items. For instance b only appears once and now has support 1, which is below the minimum support of 2. Thus, node b is removed. The result of these last three steps is shown in Figure 3.6.

4. Gather the prefix paths ending in d from the conditional FP-tree for e (Figure 3.6). This is shown in the figure to the left. By adding the frequency counts associated to the d nodes we obtain the support count for $\{d, e\}$, which is equal to 2. Therefore, $\{d, e\}$ is a frequent itemset.

5. Construct the conditional FP-tree for $\{d, e\}$ as in step 3. Update the support counts (not needed in this particular case). Remove the d nodes. Then, remove infrequent items (in this case, node c). Because the conditional FP-tree has only one node, a, whose support is 2, the algorithm extracts the frequent itemset $\{a, d, e\}$. Move to the next subproblem.

6. Gather the prefix paths ending in c from the conditional FP-tree for e (Figure 3.6) as in step 4. Then proceed as in step 5. Only $\{c, e\}$ is found to be frequent.

Ending in	Frequent itemsets
e	$\{e\}$, $\{d, e\}$, $\{a, d, e\}$, $\{c, e\}$, $\{a, e\}$
d	$\{d\}$, $\{c, d\}$, $\{b, c, d\}$, $\{a, c, d\}$, $\{b, d\}$, $\{a, b, d\}$, $\{a, d\}$
c	$\{c\}$, $\{b, c\}$, $\{a, b, c\}$, $\{a, c\}$
b	$\{b\}$, $\{a, b\}$
a	$\{a\}$

Table 3.6: Frequent itemsets generated by the FP-Growth algorithm.

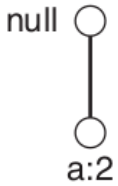

Figure 3.10: Prefix paths ending in ae.

7. There are no prefix paths ending in be from the conditional FP-tree for e (Figure 3.6). Thus, we gather the prefix paths for ae (shown in the left). $\{a, e\}$ is the only frequent itemset remaining.

The process is repeated for d, c, b, and a in the same fashion. We show all the frequent itemsets generated by the algorithm in Table 3.6

In general, FP-Growth runs about an order of magnitude faster than Apriori [12]. However, when the database is large (and especially when it contains a significant number of items), the space in memory required to store the information about the FP-trees can be large as well. This is part of the reason why, even though the Apriori algorithm is about 30 years old (at the time of writing this document), it has not fallen out of fashion completely.

4 Rule generation

Each frequent k-itemset $\mathcal{K}$ can generate $2^k - 2$ association rules (ignoring those that have empty antecedents or consequents). These rules can be generated by partitioning $\mathcal{K}$ into two non-empty subsets X and $\mathcal{K} \setminus X$ such that $X \Rightarrow \mathcal{K} \setminus X$. Note that these rules already satisfy the support condition in (2.1) as they are generated from a frequent itemset. Therefore, rules must only be pruned according to their confidence. This can be done efficiently by exploiting the following property.

Lemma 4.1. If a rule $X \Rightarrow \mathcal{K} \setminus X$ does not satisfy the confidence threshold, then any rule $X' \Rightarrow \mathcal{K} \setminus X'$ with $X' \subseteq X$ must not satisfy the threshold as well.

To see why this is true, observe that

$$\text{conf}(X' \Rightarrow \mathcal{K} \setminus X') = \frac{\text{supp}(X' \cup (\mathcal{K} \setminus X'))}{\text{supp}(X')} = \frac{\text{supp}(\mathcal{K})}{\text{supp}(X')}.$$

Because $X' \subseteq X$, the anti-monotonicity property (Lemma 3.1) yields $\text{supp}(X') \geq \text{supp}(X)$. Therefore,

$$\text{conf}(X' \Rightarrow \mathcal{K} \setminus X') \leq \frac{\text{supp}(\mathcal{K})}{\text{supp}(X)} = \frac{\text{supp}(X' \cup (\mathcal{K} \setminus X))}{\text{supp}(X)} = \text{conf}(X \Rightarrow \mathcal{K} \setminus X).$$

In particular, the Apriori algorithm uses a level-wise approach to generate these rules, where each level corresponds to the number of items in the rule consequent. This allows us to discard rules according to Lemma 4.1. For example, if a rule $\{b, c, d\} \Rightarrow \{a\}$ does not satisfy the minimum confidence threshold, then the following rules can be discarded as well: $\{c, d\} \Rightarrow \{a, b\}$, $\{b, c\} \Rightarrow \{a, c\}$, $\{b, c\} \Rightarrow \{a, d\}$, $\{d\} \Rightarrow \{a, b, c\}$, $\{c\} \Rightarrow \{a, b, d\}$, $\{b\} \Rightarrow \{a, c, d\}$ (see Figure 4.1).

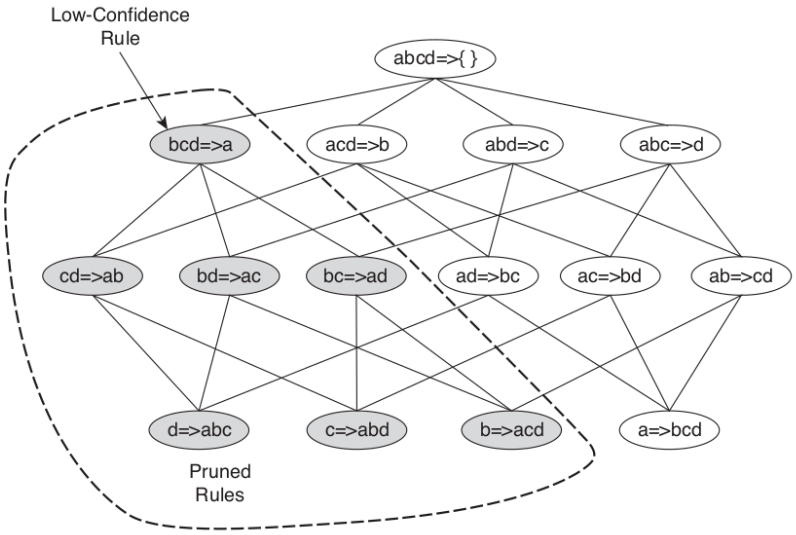

Figure 4.1: Confidence-based pruning of association rules. Source: [16, Figure 6.15].

M_{jk}	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Table 5.1: Binary representation of the transactional database shown in Table 2.1.

5 Examples of market basket analysis

We now present two examples of MBA. First, we perform the analysis on a rather small data set obtained from transactions in a supermarket. Then, we use a more realistic data set from an online retailer.

All the experiments were coded in Jupyter notebooks using primarily the `pandas` and `mlxtend` libraries. The latter conveniently provides the functions `mlxtend.frequent_patterns.apriori` and `mlxtend.frequent_patterns.fpgrowth` to generate frequent itemsets. The output can then be used with the function `mlxtend.frequent_patterns.association_rules` to generate association rules. Note that the `apriori` and `fpgrowth` functions require the transactions to be in a binary representation. That is, for a database with n items (i.e. $\mathcal{I} = \{i_k\}_{k=1}^{n}$) and m transactions (i.e. $\mathcal{T} = \{t_j\}_{j=1}^{m}$), the database is transformed into an m-by-n matrix $\boldsymbol{M} = (M_{jk})$ such that

$$M_{jk} = \begin{cases} 1 & \text{if item } i_k \text{ was purchased in transaction } t_j, \\ 0 & \text{otherwise.} \end{cases}$$

An example of this representation is shown in Table 5.1 using the example from Table 2.1. In what follows, we assume that the databases have already been encoded into their binary formats (the reader is invited to review the accompanying Jupyter notebooks to learn more about the preprocessing stage).

Support	Itemset
0.252046	(Eggs)
0.243863	(White Bread)
0.219313	(2pct. Milk)
0.201309	(Potato Chips)
0.201309	(98pct. Fat Free Hamburger)
0.194763	(Hot Dogs)
0.180033	(Sweet Relish)
0.171849	(Onions)
0.165303	(Toothpaste)
0.163666	(Cola)

Table 5.2: Top 10 of the most-frequent itemsets in the supermarket database.

5.1 Example 1: supermarket data set

The first database in consideration consists of 255 items and 1,361 transactions originating from point-of-sale transactions at a small supermarket [3]. After removing single-item transactions (recall that we need at least two items to construct a valid association rule), we are left with 611 transactions.

First, we learn frequent itemsets with $minsupp = 0.4$ using the Apriori algorithm. A total of 1,647 frequent itemsets are found, with the 10 most frequent shown in Table 5.2. Then we generate association rules with $minconf = 0.9$. The high minimum confidence threshold leads to only 20 rules, which are shown in Table 5.4. In this case, the store manager might be interested in those rules that have higher lift and conviction values, since these denote rules that did not happen by chance. The first rule in particular tells us that White Bread, Bananas, Wheat Bread and 2pct. Milk appeared together in 4.42% of the transactions, and that 96.4% of the times that White Bread, Bananas, and Wheat Bread were purchased, 2pct. Milk was also purchased. Similarly, from line 8 in Table 5.2, we have that Bananas, 2pct. Milk, Toothpaste, and White Bread were purchased together in 4.26% of the transactions and that 96.3% of the times that Bananas, 2pct. Milk, and Toothpaste were purchased, White Bread was purchased as well.

5.2 Example 2: online retail data set

For this example, we perform an MBA on the "Online Retail" database [6] which contains 541,909 instances of transactions occurring between December 1-9, 2010, at a UK-based and registered non-store online retailer. Prior to any preprocessing, there is a total of 25,900 transactions and 4,223 items. Then, the following tasks are performed:

1. Removing cancellations from the database (number of transactions reduced to 22,064 and items to 4,207).
2. Removing rows that include missing values (number of transactions further reduced to 18,536 and items to 3,877).
3. Removing transactions from countries other than the United Kingdom, as they only represent 9% of the total number of transactions.

After all these operations, the database that we will work with contains 16,649 transactions and 3,844 items, which represent respectively about 64% and 91% of the original amount of data.

Support	Itemset
0.121358	(WHITE HANGING HEART T-LIGHT HOLDER)
0.093197	(JUMBO BAG RED RETROSPOT)
0.090466	(REGENCY CAKESTAND 3 TIER)
0.084417	(ASSORTED COLOUR BIRD ORNAMENT)
0.082986	(PARTY BUNTING)
0.072841	(LUNCH BAG RED RETROSPOT)
0.064971	(SET OF 3 CAKE TINS PANTRY DESIGN)
0.064646	(LUNCH BAG BLACK SKULL.)
0.061004	(PAPER CHAIN KIT 50'S CHRISTMAS)
0.060939	(NATURAL SLATE HEART CHALKBOARD)

Table 5.3: Top 10 of the most-frequent itemsets in the online retail database.

First, we learn frequent itemsets using the FP-Growth algorithm with $minsupp = 0.03$. The algorithm finds 108 frequent itemsets, with the 10 most frequent shown in Table 5.3. Then, we generate association rules with $minconf = 0.2$. This leads to a total of 8 rules which are shown in Table 5.5. These are all rules that have relatively high lift values, which might be of interest to the owners of the store. However, we note that all the rules involve variations of the same item (such as colours or patterns). This makes sense because a wholesale reseller who purchases items from this retailer might be interested in buying several variations of the same product. Therefore, the rules that we just found might not be the most informative ones.

Let us now find association rules that contain the most sold item: "WHITE HANGING HEART T-LIGHT HOLDER" (see Table 5.3). To do this, we will have to lower the minimum support threshold so that we obtain more rules. Indeed, considering now $minsupp = 0.015$ and $minconf = 0.2$ the algorithm finds 518 frequent itemsets and 280 rules, of which 8 rules contain "WHITE HANGING HEART T-LIGHT HOLDER" as the consequent. These rules are displayed in Table 5.6. In particular, we infer from the first rule (line 9) that "CANDLEHOLDER PINK HANGING HEART" and "WHITE HANGING HEART T-LIGHT HOLDER" appeared together in 1.53% of the transactions. Moreover, 74.4% of the times the first article was purchased, the second one was also purchased.

As a closing note for this section, we emphasize that the experiments above are quite simplistic and are motivated primarily by academic reasons. In reality, the rules could be further filtered to account for different product attributes (such as manufacturer, brand, style, colour, or size) using stock keeping unit (SKU) numbers.

	Antecedent X	Consequent Y	supp(X)	supp(Y)	Support	Confidence	Lift	Conviction
0	(White Bread, Bananas, Wheat Bread)	(2pct. Milk)	0.045827	0.219313	0.044190	0.964286	4.396855	21.859247
1	(Potato Chips, Eggs, Apples)	(2pct. Milk)	0.044190	0.219313	0.042553	0.962963	4.390824	21.078560
2	(Eggs, Pepperoni Pizza - Frozen, Toothpaste)	(2pct. Milk)	0.042553	0.219313	0.040917	0.961538	4.384328	20.297872
3	(White Bread, Potato Chips, Eggs, Toothpaste)	(2pct. Milk)	0.047463	0.219313	0.044190	0.931034	4.245239	11.319967
4	(Wheat Bread, Eggs, White Bread, Toothpaste)	(2pct. Milk)	0.044190	0.219313	0.040917	0.925926	4.221946	10.539280
5	(Eggs, Popcorn Salt, Cola)	(2pct. Milk)	0.044190	0.219313	0.040917	0.925926	4.221946	10.539280
6	(Eggs, Popcorn Salt, Toothpaste)	(2pct. Milk)	0.049100	0.219313	0.044190	0.900000	4.103731	7.806874
7	(Wheat Bread, Potato Chips, Toothpaste)	(2pct. Milk)	0.049100	0.219313	0.044190	0.900000	4.103731	7.806874
8	(Bananas, 2pct. Milk, Toothpaste)	(White Bread)	0.044190	0.243863	0.042553	0.962963	3.948794	20.415712
9	(Potato Chips, 98pct. Fat Free Hamburger, W...	(White Bread)	0.047463	0.243863	0.044190	0.931034	3.817866	10.963993
10	(Hot Dogs, 2pct. Milk, Toothpaste)	(White Bread)	0.044190	0.243863	0.040917	0.925926	3.796918	10.207856
11	(Eggs, Tomatoes, Sugar Cookies)	(White Bread)	0.052373	0.243863	0.047463	0.906250	3.716233	8.065466
12	(Eggs, 98pct. Fat Free Hamburger, Wheat Bread)	(White Bread)	0.049100	0.243863	0.044190	0.900000	3.690604	7.561375
13	(Bananas, 2pct. Milk, Wheat Bread)	(White Bread)	0.049100	0.243863	0.044190	0.900000	3.690604	7.561375
14	(Sweet Relish, Plain Bagels)	(Eggs)	0.044190	0.252046	0.040917	0.925926	3.673641	10.097381
15	(Tomatoes, 2pct. Milk, Sugar Cookies)	(Eggs)	0.044190	0.252046	0.040917	0.925926	3.673641	10.097381
16	(Potato Chips, Tomatoes, 2pct. Milk)	(Eggs)	0.044190	0.252046	0.040917	0.925926	3.673641	10.097381
17	(White Bread, Tomatoes, Sugar Cookies)	(Eggs)	0.052373	0.252046	0.047463	0.906250	3.595576	7.978178
18	(White Bread, Popcorn Salt, 2pct. Milk)	(Eggs)	0.049100	0.252046	0.044190	0.900000	3.570779	7.479542
19	(Popcorn Salt, 2pct. Milk, Toothpaste)	(Eggs)	0.049100	0.252046	0.044190	0.900000	3.570779	7.479542

Table 5.4: Generated association rules $X \Rightarrow Y$ ordered by their lift values ($minsupp = 0.4$, $minconf = 0.9$).

	Antecedent X	Consequent Y	supp(X)	supp(Y)	Support	Confidence	Lift	Conviction
0	(GREEN REGENCY TEACUP AND SAUCER)	(ROSES REGENCY TEACUP AND SAUCER)	0.039802	0.043900	0.030957	0.777778	17.717202	4.302452
1	(ROSES REGENCY TEACUP AND SAUCER)	(GREEN REGENCY TEACUP AND SAUCER)	0.043900	0.039802	0.030957	0.705185	17.717202	3.256952
2	(JUMBO BAG PINK POLKADOT)	(JUMBO BAG RED RETROSPOT)	0.052680	0.093197	0.032908	0.624691	6.702899	2.416152
3	(LUNCH BAG PINK POLKADOT)	(LUNCH BAG RED RETROSPOT)	0.055086	0.072841	0.030632	0.556080	7.634188	2.088574
4	(LUNCH BAG BLACK SKULL.)	(LUNCH BAG RED RETROSPOT)	0.064646	0.072841	0.031478	0.486922	6.684737	1.807051
5	(LUNCH BAG RED RETROSPOT)	(LUNCH BAG BLACK SKULL.)	0.072841	0.064646	0.031478	0.432143	6.684737	1.647164
6	(LUNCH BAG RED RETROSPOT)	(LUNCH BAG PINK POLKADOT)	0.072841	0.055086	0.030632	0.420536	7.634188	1.630668
7	(JUMBO BAG RED RETROSPOT)	(JUMBO BAG PINK POLKADOT)	0.093197	0.052680	0.032908	0.353105	6.702899	1.464412

Table 5.5: Generated association rules $X \Rightarrow Y$ ordered by their lift values ($minsupp = 0.03$, $minconf = 0.2$).

	Antecedent X	Consequent Y	supp(X)	supp(Y)	Support	Confidence	Lift	Conviction
9	(CANDLEHOLDER PINK HANGING HEART)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.020552	0.121358	0.015284	0.743671	6.127912	3.427789
27	(RED HANGING HEART T-LIGHT HOLDER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.041623	0.121358	0.027836	0.668750	5.510557	2.652504
220	(HEART OF WICKER LARGE)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.051314	0.121358	0.018600	0.362484	2.986901	1.378228
223	(WOODEN PICTURE FRAME WHITE FINISH)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.055411	0.121358	0.019836	0.357981	2.949796	1.368561
251	(WOODEN FRAME ANTIQUE WHITE)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.050858	0.121358	0.016259	0.319693	2.634299	1.291538
277	(NATURAL SLATE HEART CHALKBOARD)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.060939	0.121358	0.016975	0.278549	2.295264	1.217881
278	(HEART OF WICKER SMALL)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.060094	0.121358	0.016584	0.275974	2.274050	1.213550
279	(PARTY BUNTING)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.082986	0.121358	0.017495	0.210815	1.737134	1.113354

Table 5.6: Generated association rules $X \Rightarrow Y$ that contain Y = (WHITE HANGING HEART T-LIGHT HOLDER) as the consequent ordered by their lift values ($minsupp = 0.015$, $minconf = 0.2$).

6 Conclusions

In this project, we have described the association rule learning problem using market basket analysis. Moreover, given that a big part of this task goes towards finding frequent itemsets, we have described the two most common approaches to solve this problem: the Apriori and FP-Growth methods. While the latter tends to run faster than the former, in the end it is just a trade-off between speed and memory requirements, specially when working with large commercial databases. Our rather small numerical experiments ran in a matter of seconds in a standard laptop, so neither of the algorithms showed an advantage over the other. Nevertheless, this also reveals how accessible is to perform these tasks using simple Python commands. We believe that market basket analysis is a simple way to showcase the many complexities that can arise in unsupervised learning, which in the end leads to a better understanding of this important area in machine learning.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, *Mining association rules between sets of items in large databases*, in Proceedings of the 1993 ACM SIGMOD international conference on Management of data, 1993, pp. 207–216.

[2] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules*, in Proc. 20th int. conf. very large data bases, VLDB, vol. 1215, Santiago, Chile, 1994, pp. 487–499.

[3] M. Barksy, *CSCI 485: Data mining 2012, lab 7: Rules in market basket data*. Department of Computer Science, Vancouver Island University (2012). Available at http://csci.viu.ca/~barskym/teaching/DM2012/labs/LAB7/PartII.html. Accessed on April 12, 2023.

[4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, *Dynamic itemset counting and implication rules for market basket data*, in Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 1997, pp. 255–264.

[5] A. L. Buczak and E. Guven, *A survey of data mining and machine learning methods for cyber security intrusion detection*, IEEE Communications surveys & tutorials, 18 (2015), pp. 1153–1176.

[6] D. Chen, S. L. Sain, and K. Guo, *Data mining for the online retail industry: A case study of rfm model-based customer segmentation using data mining*, Journal of Database Marketing & Customer Strategy Management, 19 (2012), pp. 197–208. Available at http://archive.ics.uci.edu/ml/datasets/Online+Retail. Accessed on April 12, 2023.

[7] J. A. Diaz-Garcia, M. D. Ruiz, and M. J. Martin-Bautista, *Non-query-based pattern mining and sentiment analysis for massive microblogging online texts*, IEEE Access, 8 (2020), pp. 78166–78182.

[8] K. Faust and J. Raes, *Microbial interactions: from networks to models*, Nature Reviews Microbiology, 10 (2012), pp. 538–550.

[9] J. Han, J. Pei, and Y. Yin, *Mining frequent patterns without candidate generation*, ACM sigmod record, 29 (2000), pp. 1–12.

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Springer Series in Statistics, Springer New York : Imprint: Springer, 2nd ed., 2009.

[11] S.-H. Liao, R. Widowati, and Y.-C. Hsieh, *Investigating online social media users' behaviors for social commerce recommendations*, Technology in Society, 66 (2021), p. 101655.

[12] J. M. Luna, P. Fournier-Viger, and S. Ventura, *Frequent itemset mining: A 25 years review*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 9 (2019), p. e1329.

[13] K. P. Murphy, *Machine learning: a probabilistic perspective*, MIT press, 2012.

[14] G. Piatetsky-Shapiro, *Discovery, analysis, and presentation of strong rules*, Knowledge discovery in database, (1991), pp. 229–248.

[15] A. Savasere, E. R. Omiecinski, and S. B. Navathe, *An efficient algorithm for mining association rules in large databases*, tech. rep., Georgia Institute of Technology, 1995.

[16] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, Pearson Addison Wesley, 1st ed., 2006.

[17] X. Yan, C. Zhang, and S. Zhang, *Confidence metrics for association rule mining*, Applied Artificial Intelligence, 23 (2009), pp. 713–737.

[18] M. J. Zaki, *Scalable algorithms for association mining*, IEEE transactions on knowledge and data engineering, 12 (2000), pp. 372–390.

[19] M. J. Zaki and K. Gouda, *Fast vertical mining using diffsets*, in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, 2003, pp. 326–335.