COL761/COL7361/AIL7026 Data Mining
Assignment 1: Frequent Itemset Mining
Team T123

Kislaya Shrestha 2025SIY7573
Palaash Goel 2025AIZ8825
Anoop Parashar 2025MCS2118

Assignment 1

Part 1

Task 1

1. Introduction

We have been given a dataset named WebDocs. It contains 1.7 million documents and is 5 GB in size. The dataset was built from the web collection. It contains 1,692,082 transactions and 5,267,656 distinct items. The maximum transaction length is 71,472.

We have to use 2 data mining algorithms to find frequent item sets and association rules. The first is the Apriori algorithm, developed by Agrawal and Srikant in 1994. The second is the FP-growth algorithm, developed by Jiawei Han in 2000.

1.1 Frequent Itemset Mining and Association Rule Mining

In this field, the task is to find items that appear together in a transaction. Its application is mostly in the field of commerce, where customer buying patterns are studied. Our task is to efficiently find rules of implication in the transaction database, so that we can efficiently provide the best use of resources in the logistics of the products.

1.2 Apriori Algorithm

It was developed by Agrawal and Srikant in 1994. It follows the apriori principle "If an itemset is frequent, its subset will also be frequent." Here, we start with a single item and find the frequent items for a given minimum support value. Support is the fraction of transactions that contain both the antecedent and consequent. Using the frequent items obtained, we pair them to create a candidate set and check it for infrequent subsets. Next, we count the candidate set and keep it if its count is more than the minimum support value provided.

1.3 FP-Growth

FP-growth, or frequent pattern growth, uses a divide-and-conquer approach. It begins by compressing the database of frequent items into an FP-tree, which preserves the itemset associations. The compressed database is then split into multiple conditional databases, each linked to a specific frequent item or "pattern fragment," and each is mined separately. For each pattern fragment, only its relevant data sets are analyzed. This method can significantly reduce the size of the data sets that need to be searched and limit the number of patterns examined, improving overall efficiency.

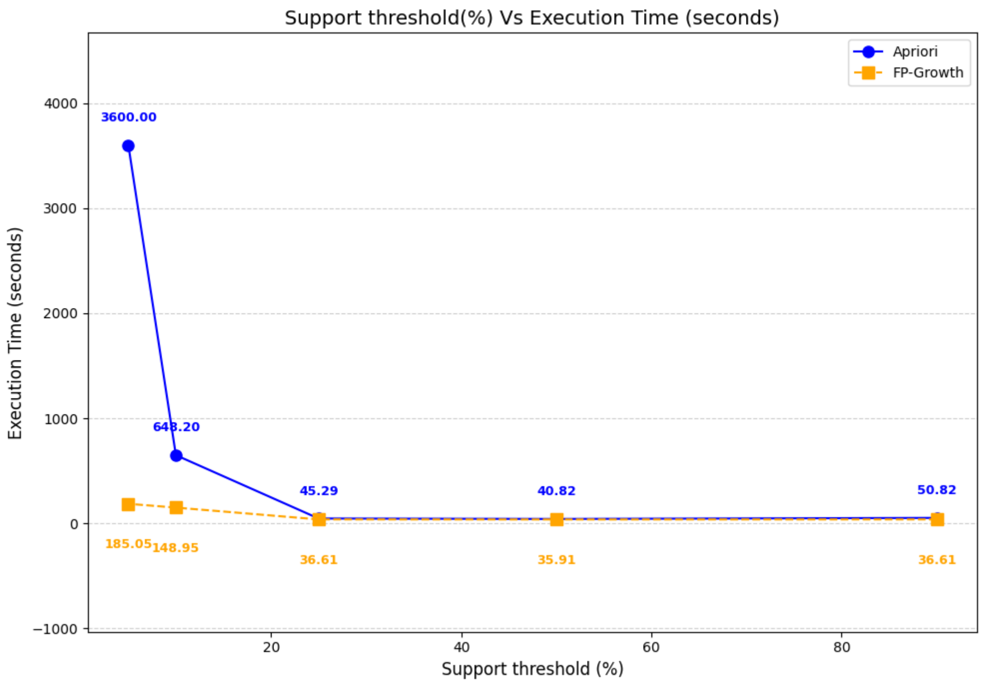
1.4 Dataset

English-language dataset built from a spidered collection of 1.7 million HTML documents. It is a preprocessed dataset with HTML tags removed, stopwords filtered, and stemming applied. The dataset size is 1.48 GB. It contains 1,692,082 transactions with 5,267,656 distinct items, with a maximum transaction length of 71,472 items.

2. Runtime Results for Task 1

Min Support	Apriori Execution Time	FP-Growth Execution Time
90%	50.82 s	36.61 s
50%	40.82 s	35.91 s
25%	45.29 s	36.61 s
10%	648.20 s	148.95 s
5%	3600 s (timeout)	185.05 s

3. Plot for Task 1



4. Observations for Task 1

- FP-growth outperforms Apriori in speed, especially at low support thresholds.
- The execution time of Apriori increases exponentially as the minimum support value decreases.
- The Apriori curve is steep, while the FP-growth curve remains relatively flat when comparing minimum support to execution time.

- Apriori is effective only when the minimum support is high.
- For both Apriori and FP-Growth, it is observed that the execution time exhibits an inverse proportionality to the minimum support threshold. Specifically, as the support threshold increases, the execution time decreases.
- For threshold values below 10%, a significant increase in the execution time of the Apriori algorithm is observed. The execution time becomes exceedingly high at a minimum support of 5%, resulting in a timeout.
- Speedup of FP-growth compared with Apriori.

Min Support	Apriori Execution Time	FP-Growth Execution Time	Speedup
90%	50.82 s	36.61 s	1.39
50%	40.82 s	35.91 s	1.14
25%	45.29 s	36.61 s	1.24
10%	648.20 s	148.95 s	4.35
5%	3600 s (timeout)	185.05 s	19.45

- We observe that the speedup of FP-growth over Apriori increases as the minimum support decreases.
- The execution time of the FP-growth algorithm increases as the minimum support decreases.

5. Analysis of Observations for Task 1

- **Observation:** FP-Growth is faster than Apriori at lower support values.

  Analysis: This is because FP-growth requires two scans of the transactions for frequency-based ordering and one scan for lexicographical ordering to create the FP-tree. After that, it mines the FP-tree to find the frequent itemsets and the association rules. In contrast, Apriori has to execute multiple scans of the database:

  - First, it finds the number of frequent items of size k=1 by counting the support.
  - It then performs multiple scans until no new frequent itemsets are identified, checking all k+1 candidates for infrequent subsets or support counting.

- **Observation:** The execution time of Apriori increases exponentially as the minimum support decreases.

  Analysis: Our dataset contains a very large number of unique items (1,692,082). At lower minimum support (10%), a substantial portion of these items becomes frequent, significantly increasing the size of itemsets at k=1.

  For example, even if only 200 items are frequent at k=1, Apriori generates C(200,2)=19,900 candidates at k=2, C(200,3)=1,313,400 candidates at k=3, and C(200,4)=64,684,950 candidates at k=4.

  Each of these millions of candidates must be checked for infrequent subsets and scanned against 1,692,082 transactions to compute support. This results in massive CPU and I/O overhead. As the value of k increases, longer frequent itemsets are generated, causing multiple database scans. Consequently, Apriori runtime rises sharply (648 s at 10% and timeout at 5%).

- **Observation:** The Apriori curve is steep, whereas the FP-growth curve is flat for minimum support versus execution time.

  Analysis: The reason lies in how each algorithm handles frequent itemsets. At a low minimum support level, the number of frequent itemsets increases significantly. In the Apriori algorithm, during the candidate generation phase, each candidate itemset requires subset checking and support counting, which necessitates multiple scans of the entire database. This leads to higher execution time, resulting in a steep curve.

  In contrast, the FP-growth algorithm begins with a construction phase for the FP-tree, which requires scanning the database twice for frequency-based ordering and once for lexicographic ordering. After constructing the FP-tree, the algorithm generates the conditional pattern base and builds a conditional FP-tree, then mines that tree recursively. As a result, even when there is a sharp increase in the number of frequent itemsets, the I/O cost remains low. This contributes to a flatter execution time curve for the FP-growth algorithm.

- **Observation:** The FP-growth algorithm shows an increase in execution time as the minimum support decreases.

  Analysis: With low minimum support, the number of frequent itemsets increases. This increases the size of the FP-tree, making it less compact. A low support threshold causes more frequent itemsets to be mined from the tree. This increases the size of the conditional pattern bases, the conditional FP-tree, and the time for recursive FP-tree mining. This sudden increase in the number of frequent itemsets increases the execution time.

Part1

Task 2

Sub Task 3

Comparison between original data and synthetic data

Objective

The goal of Task 2 was to construct a synthetic transactional dataset (around 15k transactions with universal itemset between 25 to 40) such that the runtime trends of Apriori and FP-Growth resemble to reference plot provided in the assignment.

The expected:

- Apriori should take significantly longer at low and medium support thresholds (5–50%) because we get more frequents item for low and medium support thresholds
- FP-Growth should remain much faster across thresholds because it does not generate candidates like apriori
- Both algorithms should behave same as high support (90%) because very few frequent patterns exist

Synthetic Dataset Generation

We have created our synthetic dataset by taking the universal_itemset and num_transactions as the input. Each transaction consists of the following :

- Highly frequent overlapping item groups
- Moderately frequent item groups
- Rare noise items

The script divides the item universe into four parts:

- freq_items1 (most frequent quarter)
- freq_items2 (second quarter)
- freq_items3 (third quarter)
- Remaining items used as noise

- freq_items1 = items[:num_items//4]
- freq_items2 = items[num_items//4:2*num_items//4]
- freq_items3 = items[2*num_items//4:3*num_items//4]

In our dataset, each transaction includes frequent items on the basis of the probability:

- if random.random() < 0.9: # group 1
- if random.random() < 0.8: # group 2
- if random.random() < 0.6: # group 3

Additionally, the loop repeats 5 times:

- for _ in range(5):   ...

It ensures that transactions with large, frequent subsets result in **:**

- Many items fall within the support thresholds of 5% to 50%.%
- This Apriori algorithm generates high candidate sets, where Fpgrowth operates more efficiently, which is also reflected in its runtime.

This aligns with the expected plot trend, where Apriori runtime remains nearly constant and very high until the support threshold increases significantly.

FP-Growth Remains Fast

FP-Growth is efficient because it avoids explicitly generating candidates. Instead, it relies on the following:

• FP-tree

• Mining frequent patterns directly via conditional pattern bases.

Our dataset exhibits significant transaction overlap due to common item groups, resulting in:

- High prefix sharing
- Compact FP-tree representation
- Much lower mining cost

- Thus, FP-Growth consistently performs quickly, even when Apriori becomes very slow. This explains why, in all our plots, FP-Growth remains near the bottom, with runtime increasing only slightly at low support levels.

Effect of Support Threshold

5%

At 5% minsup we can observe explosion in candidates and number of frequent itemsets
as a result Apriori is taking too much time but fpgrowth is doing better as expected

10%, 25%,50%

now same behavior can be observed for minsups 10% , 25% and 50%.

90%

- At **high support (90%)**, very few itemsets remain frequent:
- Only the most dominant items survive
- Candidate generation becomes trivial for Apriori
- FP-Growth conditional mining also becomes trivial

Hence both algorithms drop sharply at 90%, consistent with the reference plot.

Effect of Increasing Item Universe Size

We experimented with different values of num_items while keeping transactions fixed at 15,000:

for experimenting we tried with 4 different universal Itemsets values

25,30,35,40. but as it is mentioned that we have make report for **30 we have attached**

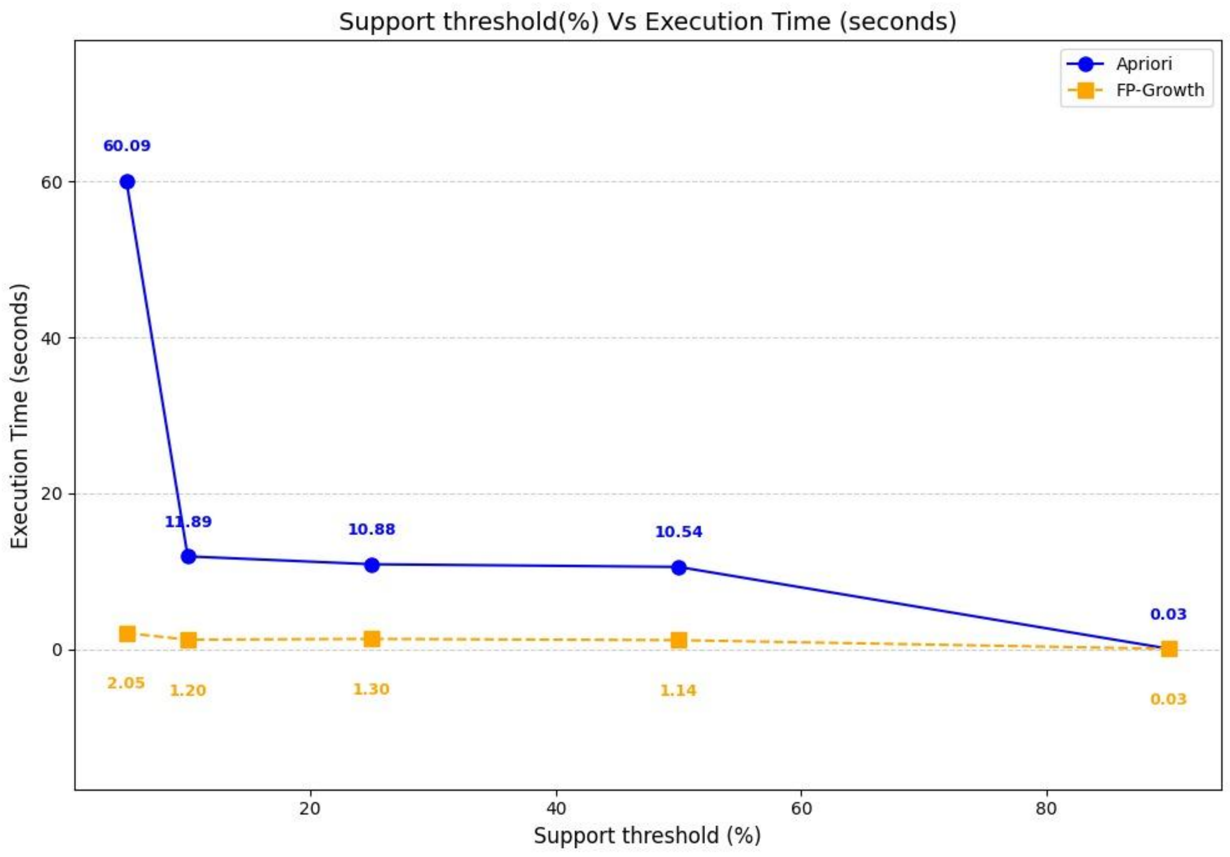
the plot for universal items =30. but we observed following outcomes as well

25 items

- at universal itemsets=25 less number of frequent itemsets
- Apriori runtime is high at 5%, but drops more smoothly.
- Candidate explosion is limited due to smaller universe.

30 items

- Runtime curve resembles the reference plot more closely.
- Enough frequent combinations exist to keep Apriori expensive up to 50%.



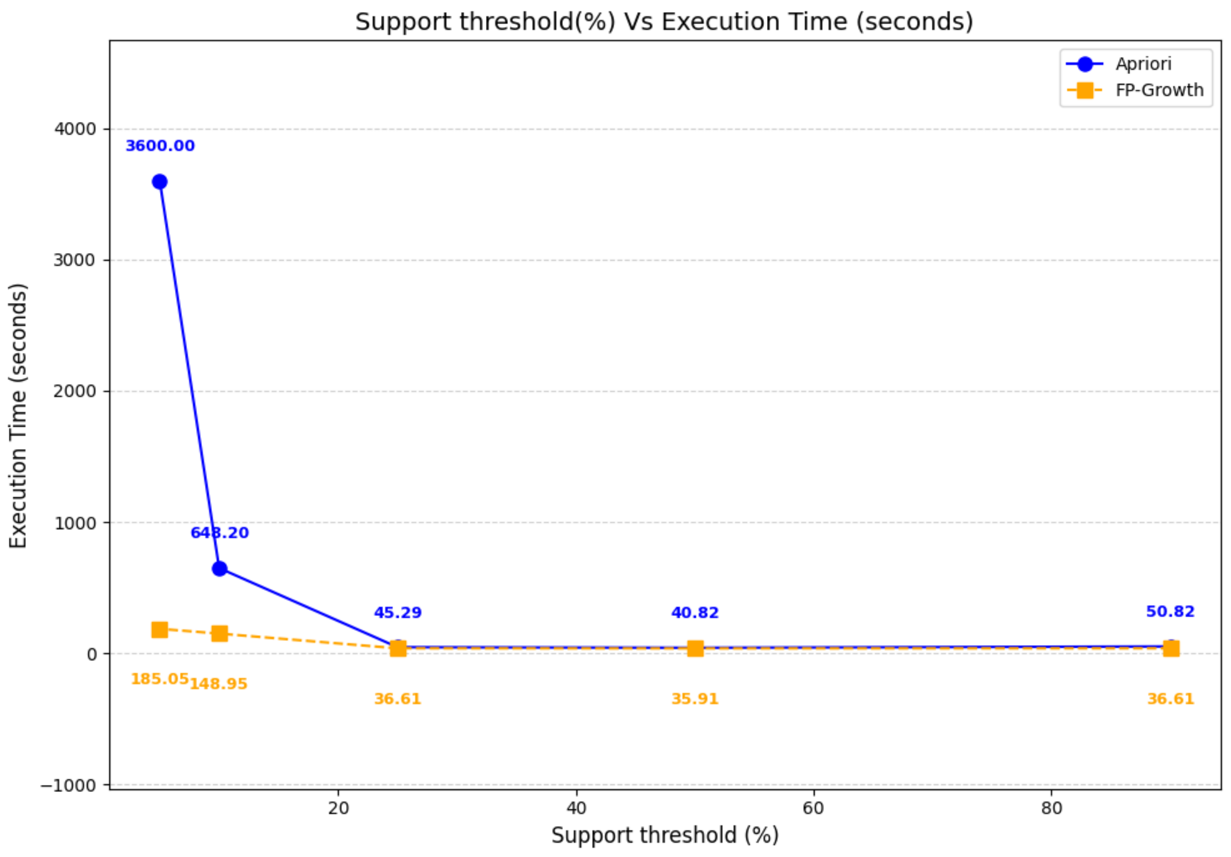
Plot Observed for Universal Itemset=30

35-40 items

- Apriori produces extremely large output files.
- Number of frequent itemsets becomes enormous at low supports.
- Candidate space grows very high
- specifically for 40 Universal Itemset it generated enormous amount of frequent itemsets

Comparison with Original Dataset (webdocs)

The webdocs.dat dataset is an English-language dataset built from a spidered collection of 1.7 million HTML documents. It is very dense, with many documents sharing common terms, resulting in strong frequent patterns and numerous mid-support frequent itemsets. This density makes the dataset especially challenging for algorithms like Apriori.



Plot Observed for the WebDocs.dat

In its original form, Apriori suffers from significant candidate explosion at low and medium support levels, because many item combinations meet the minimum support threshold. This causes excessive candidate generation, multiple database scans, and high computational costs. Conversely, FP-Growth performs much better on this dataset because it compresses transactions into an FP-tree, avoiding explicit candidate generation.

Our synthetic dataset was purposefully created to replicate these real-world properties, including highly frequent overlapping item groups, repeated inclusion of frequent subsets, and controlled noise items, to ensure:

• Transactions are highly dense and overlapping.

• Numerous frequent itemsets appear at mid-support levels.

• The dataset causes a combinatorial explosion similar to the original data.

Our plot for the synthetic dataset decreases first and then plateaus above the FP-growth's curve and decrease after 50%. For the FP-growth, the curve does not produce a steep ascent or descent in the graph.

It mostly remains the same.

This is due to the fact that our synthetic dataset at low and medium support values contains many overlapping frequent item groups. This causes a large number of itemsets to become frequent, leading Apriori to generate a huge number of candidates. As a result, Apriori spends most of its time performing subset checking and repeated database scans. This keeps its runtime high and relatively flat (plateau) over a wide range of support values.

As the minimum support increases beyond around 50%, many of these frequent combinations no longer satisfy the support threshold. The number of frequent itemsets drops sharply, reducing candidate generation and database scans. Consequently, Apriori's execution time decreases after 50%.

For FP-Growth, the curve does not show steep ascent or descent because the algorithm does not rely on candidate generation. Instead, it constructs an FP-tree and mines patterns directly from it. Due to strong overlap among transactions in the synthetic dataset, the FP-tree remains compact and highly shared across different support values. Therefore, FP-Growth's runtime remains mostly stable, producing a nearly flat curve, with only minor variations at lower supports.

Consequently, the runtime behavior of Apriori and FP-Growth on our generated dataset closely mirrors their performance on the original webdocs dataset. Apriori suffers from significant runtime overhead due to excessive candidate generation, whereas FP-Growth benefits more from compression with the FP-tree.

Therefore:

• Apriori performs poorly on both datasets at low and medium supports.

• FP-Growth consistently outperforms Apriori thanks to its pattern-growth method.

• The overall runtime patterns on the generated dataset replicate those observed on the original dataset.

This comparison demonstrates that our generated dataset is a realistic approximation of the webdocs dataset and effectively reproduces similar runtime characteristics.

Conclusion of the Comparision

Our dataset generation approach effectively replicates the qualitative runtime patterns of the reference plot:

- Apriori remains very slow at low to mid support levels because of candidate explosion.
- FP-Growth stays fast thanks to FP-tree compression.
- Both algorithms become quicker at very high support.
- Increasing the number of items raises the combinatorial complexity, especially noticeable at 40 items.

- The apriori's runtime decreases with lower support, plateaus for most min support values, and ultimately decreases again.
- The FP-growth's runtime remains the same most of the time and generates a plateau.

Therefore, the generated dataset meets the assignment requirement of mimicking the Apriori versus FP-Growth runtime behaviour shown in the provided plot.

References :

- Han, Jiawei, Kamber, Micheline, & Pei, Jian. *Data Mining: Concepts and Techniques*. 3rd ed. Morgan Kaufmann Publishers, University of Illinois at Urbana–Champaign; Simon Fraser University, 2011.
- OpenAI. *ChatGPT: Conversational AI Model*. OpenAI, San Francisco, 2022. Available at: https://openai.com/chatgpt
- DeepSeek AI. *DeepSeek: Large Language Model for Advanced Reasoning*. DeepSeek, Beijing, 2024. Available at: https://deepseek.com
- Anthropic. *Claude: Constitutional AI Assistant*. Anthropic, San Francisco, 2023. Available at: https://www.anthropic.com/claude