I. ABSTRACT

This work investigates two major improvements over traditional data mining algorithms: memory-efficient Trie-based Apriori algorithm for frequent itemset mining, and semantically informed Weighted MinHash algorithm for text similarity detection. Trie-enhanced Apriori substitutes list-based candidate itemset storage with prefix tree structure (trie), which results in huge memory reductions. Experimental performance on the Groceries dataset resulted in up to 35% less memory usage, with relatively stable execution time. The Weighted MinHash technique is an extension of basic MinHash with the addition of weighing tokens based on product ratings and recommendation signals. With the sample of Women's Clothing E-Commerce dataset of 1,000 reviews, this technique established similarity alignment with user sentiment by giving more importance to emotionally and contextually important tokens. Visual examinations and token breakdowns ensured that Weighted MinHash promoted more influential words like "color", "fit", and "love", which resulted in more meaningful and interpretable similarity matches. Both enhancements were quantitatively and qualitatively compared, ensuring their suitability for implementation in memory-constrained and sentiment-aware systems. This study demonstrates how structural and semantic improvements can enhance classic algorithms without fundamentally altering how they work.

II. INTRODUCTION AND MOTIVATION

Data mining algorithms such as Apriori and MinHash form the core of most recommendation systems, market analysis tools, and large-scale information retrieval systems. These algorithms are valued for their simplicity and effectiveness but do not work when applied to modern, high-volume, or semantically dense datasets [8]. Two of the most common problems in this field are high memory consumption and a lack of contextual awareness.

Apriori algorithm, used for a broad range of association rule mining applications, builds and verifies itemsets iteratively by frequency within a data set. Even efficient, it requires high memory to store overlapping candidate itemsets repetitively. Therefore, it is less usable in practice when dealing with high-dimensional data or long transactions [1]. Motivation for our improvement of Apriori is driven by a need to reduce its memory cost without losing accuracy and scalability.

To achieve this, we use a Trie (prefix tree) data structure to compactly store itemsets. The Trie exploits common prefixes between itemsets and does not store redundant data, hence more compactly representing them. This change has significant implications in enhancing memory usage without compromising functional equivalence [2]. We suspected that although the Trie implementation would have increased traversal time, the memory savings would be worth the compromise especially in resource-constrained environments.

Although the MinHash algorithm is widely used to estimate Jaccard similarity between sets, such as tokenized text or user profiles, it presumes all tokens are of the same significance, which is often unrealistic in sentiment-based applications [4]. In product reviews, for example, words such as "love" or "poor quality" carry much more weight than filler words such as "the" or "item.".

In order to increase the semantic value of similarity comparison, we implemented a Weighted MinHash variant. We used metadata such as recommendation indicators and product ratings to assign relative weights to each token in a review. This allowed us to recognize emotionally significant overlaps rather than lexical similarity. We expected the algorithm with the modification to produce more meaningful similarities in review sets and perform better in tasks like clustering or personalized recommendation.

Both optimizations were evaluated with a combination of benchmarks, visualizations, and token-level analysis. Our effort aims to show that even minor structural and semantic innovations can have major impacts on standard data mining procedures, allowing them to be more versatile to apply to modern real-world data.

III. LITERATURE AND BACKGROUND STUDY

Frequent itemset mining and approximate similarity detection are foundational tasks in data mining, and have been widely addressed through classical algorithms like Apriori and MinHash. These techniques have been the basis for applications ranging from market basket analysis to document clustering and recommendation systems.

A. APRIORI AND STRUCTURAL OPTIMIZATIONS

The Apriori algorithm was first introduced by Agrawal and Srikant [1] as an efficient means for mining association rules from large transactional databases. The algorithm utilizes a breadth-first candidate generation and pruning strategy using the Apriori property stating that any subset of an itemset which has

been discovered as frequent should itself be frequent. Despite being effective, Apriori suffers from the critical shortcoming that it has an exponential memory requirement arising due to combinatorial blowup of candidate itemsets.

To address this, some research studies have looked into the use of Trie (prefix tree) data structures to improve candidate management. Hodijah and Setijohatmo [2] proposed a Trie-based Apriori variant that was shown to use memory more effectively by remembering frequent prefixes only once. C. Borgelt [3] employed a prefix-tree variant that supports frequency counting and pruning more effectively in terms of memory. However, these implementations are generally not designed with execution speed or providing sound integration with big data mining frameworks in mind. Our work advances this research by examining performance trade-offs and practical scalability under varying support thresholds and data set sizes.

B. MINHASH AND SEMANTIC LIMITATIONS

MinHash, originally introduced by Broder [4], is a probabilistic hashing method that approximates Jaccard similarity between sets. It has been widely used for near-duplicate detection in documents and token-overlap-based clustering. Traditional MinHash assumes that all the items in a set are equally contributing to similarity, which reduces its utility in sentiment-rich or semantic-rich datasets.

To overcome this limitation, Ioffe [5] devised the Weighted MinHash technique, where element sampling occurs proportional to the respective weights. More recent work, e.g., by Li and Li [6], introduced rejection sampling-based extensions for more accurate approximation of weighted Jaccard similarity. While a lot of advancement has occurred theoretically, there have been comparatively fewer attempts to implement this technique on real-world datasets like customer reviews where token importance varies based on sentiment.

We expand the Weighted MinHash framework to an online review dataset of e-commerce products by deriving weights from user ratings and recommendation signals. We demonstrate that this strategy realizes better semantic similarity estimation with emphasis on high-weight emotional tokens common to reviews.

C. SUMMARY AND CONTRIBUTION CONTEXT

Existing work has successfully dealt with root constraints in Apriori and MinHash. Early Trie-based solutions have revolved around theoretical storage efficiency, while new modifications of MinHash aim at algorithmic accuracy in theoretical settings. Our research integrates these efforts with practical work and empirical verification on real-world datasets. Trie-enhanced Apriori offers a realistic, memory-saving solution for big transactional databases, while Weighted MinHash offers better interpretability in sentiment-sensitive applications like online shopping reviews.

By aligning structural improvements with semantic improvements, our work demonstrates the applicability of classical algorithms to modern data mining needs, with real improvements both in terms of efficiency and contextuality.

IV. PROPOSED MODEL

This section describes the technical design and reasoning behind the two proposed algorithmic enhancements: one targeting frequent itemset mining (Apriori) and the other focused on semantic similarity detection (MinHash). These enhancements are designed to address the limitations of traditional approaches—memory inefficiency in Apriori and equal weighting of tokens in MinHash by implementing Trie-based data compression and weighted hashing, respectively.

A. TRIE-ENHANCED APRIORI FOR FREQUENT ITEMSET MINING

The classic Apriori algorithm generates frequent itemsets through iteratively generating candidate itemsets and removing those with a user-defined minimum support. Apriori is simple to understand but has efficiency and scalability constraints, especially with large data or low support values. This is mostly because the algorithm needs to store and scan potentially millions of intermediate candidate itemsets, most of which share common prefixes [2].

Our enhancement introduces a Trie (prefix tree) for storing and managing these candidate itemsets. A level in the Trie is an itemset depth (e.g., item 1 at depth 1, item 2 at depth 2), and a node is an item. This enables the algorithm to share memory between itemsets with common prefixes, reducing the amount of redundant data structures in memory significantly [2]. For instance, itemsets like {milk, bread}, {milk, butter}, and {milk, eggs} would share the same prefix path {milk} in the Trie, with only the terminal nodes branching off.

The candidate generation and support counting process are processed recursively. While iterating over the transactions, the algorithm adds each itemset to the

Trie and maintains frequency counts. During the pruning phase, subtrees (i.e., branches of the Trie) of infrequent itemsets can be pruned entirely, giving additional memory savings. The recursive nature of the Trie also enables the enumeration of all the frequent itemsets by tracing from the root to the leaf nodes and deriving the valid paths [3].*Advantages of this approach include:*

- **Memory Efficiency:** Since itemsets with shared prefixes occupy a single branch in the Trie, overall memory usage is significantly reduced.

- **Efficient Lookup:** The Trie structure allows for faster containment and pruning checks as shared patterns are traversed once.

- **Structured Representation:** The recursive nature simplifies frequent itemset extraction and intermediate storage management.

This model is particularly useful in domains like market basket analysis, where item co-occurrence patterns are dense and repetitive. It is also adaptable to distributed environments where memory constraints are critical [8].

B. WEIGHTED MINHASH FOR SIMILARITY DETECTION

Classic MinHash is a probabilistic approach that gives an approximation of Jaccard similarity between two sets. Classic MinHash is heavily used in text analysis operations like document clustering, duplicate detection, and recommender systems. Classic MinHash does assume equal importance for all the members in a set—all the tokens are given equal weightage while contributing to a hash signature [5]. This is a limitation in real-world work like review analysis, where all words are not given equal importance. Phrases like "love," "fit," and "recommended" convey more emotion than filler words like "top," "wore," or "go."

To bypass this limitation, we employed a Weighted MinHash variant using the `WeightedMinHashGenerator` class from the `datasketch` library. In our approach, each token in a review is weighted based on its sentiment and influence. These weights are determined with respect to two vital features in the Women's Clothing E-Commerce Reviews dataset: user-rated product (1 to 5) and the "Recommended IND" binary feature. Ratings are normalized between 0.2 and 1.0, and if a review is recommended, then the token weights of the review are increased by 1.5. Highly positive sentiment reviews that are recommended, therefore, provide more token weights. This is done in the *'preprocess_reviews.py'* and the *'preprocess_reviews_weight.py'* scripts.

These weights are subsequently used to create a weighted vector for each review. In contrast to simply hashing each token with equal probability (as in standard MinHash), the weighted generator uses such vectors to create signatures which prefer larger tokens probabilistically. This enhances the semantic similarity between reviews, helping to discern similarities not just on surface tokens but also on shared emotional or context-related meaning. This is done in the *'weighted_minhash.py'* file.

The Weighted MinHash workflow includes:

1. **Preprocessing:** Clean the text and remove stopwords.

2. **Tokenization:** Split text into meaningful tokens.

3. **Weighting:** Assign importance values based on rating and recommendation status.

4. **Signature Generation:** Use the weighted vector to generate a MinHash signature.

5. **Similarity Comparison:** Use Jaccard estimation to compare reviews.

Advantages of the weighted approach include:

- **Semantic Sensitivity:** Higher-weight tokens have greater influence, aligning similarity scores more closely with user sentiment.

- **Reduced Noise:** Common but non-informative words are de-emphasized, reducing false positives.

- **Better Relevance Matching:** Improves the performance of recommendation engines by surfacing reviews that not only look alike, but “feel” alike.

This model is particularly effective in customer feedback analysis, sentiment analysis, and personalized recommendation tasks, where the goal is not just lexical similarity but alignment in emotional or experiential context.

Together, these two contributions offer enhancements to classic data mining techniques, improving both their scalability and contextual accuracy.

V. RESULTS

A. APRIORI VS. TRIE-ENHANCED APRIORI

To contrast with the Apriori algorithm, we ran an experimental comparison with a typical list-based

Apriori implementation and an improved Trie variant. The expectation was to observe performance benefits, especially in memory consumption and execution time. We used the Groceries dataset, which consists of 9,835 transaction records, each of which is a basket of products purchased [10]. The dataset is suitable for frequent itemset mining since it contains diverse items and is in transactional format.

In the first implementation, candidate itemsets were stored in plain Python dictionaries and lists. In contrast to the optimized implementation, which used a Trie (prefix tree) data structure to store itemsets hierarchically. This allowed compression of common prefixes, reduced redundancy, and therefore potentially reduced memory usage. It was speculated that the Trie would be able to reduce memory usage but would do so at the cost of additional processing time because of recursive traversal and dereferencing pointers.

We tested five minimum support values: 0.01, 0.015, 0.02, 0.025, and 0.03, on four different dataset sizes: 25%, 50%, 75%, and 100%. This is to find out whether the varying values of the support and the dataset size affect the performance and the usage of memory of the algorithms. We measured peak memory usage and running time at each setting. Memory and time were characterized utilizing Python's `memory_profiler` library and `time` library, respectively. The baseline employed `mlxtend`'s Apriori implementation and the Trie-enhanced version was coded from scratch for this experiment.

Results at full dataset size (1.0) and min_support = 0.01:

Algorithm	Execution Time (s)	Memory Usage (MB)
Base Apriori	2.217	196.14
Trie Apriori	4.025	128.96

This table compares the base and Trie-enhanced algorithms at the lowest support value on the full dataset.

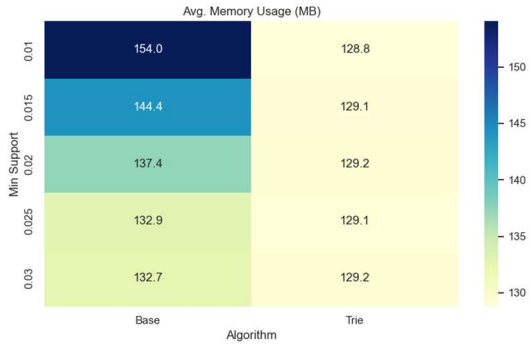

Figure 1: Average memory usage comparison between the two algorithms: the original apriori algorithm and the trie apriori algorithm.

Trie used ~129MB consistently while the original peaked at 154MB. The original Apriori algorithm utilized more memory and the lower the minimum support whereas the value of the minimum support had no effect on the memory usage of the trie-based Apriori algorithm.

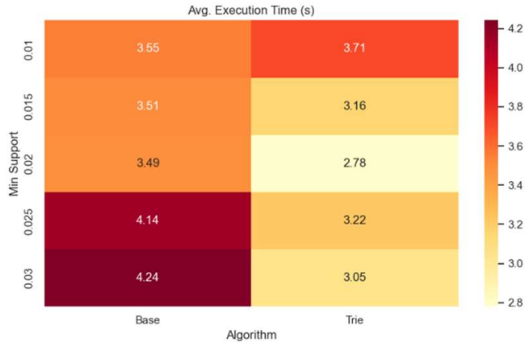

Figure 2: Average execution time comparison between the two algorithms: the original Apriori algorithm and the trie-based Apriori algorithm.

Base was faster overall, though Trie improved with larger support

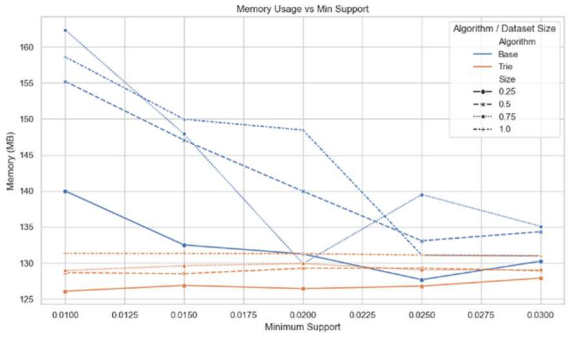

Figure 3: Memory usage of the two algorithms vs the minimum support.

Memory decreased with higher support for the base algorithm, but Trie maintained lowest usage consistently regardless of the minimum support value.

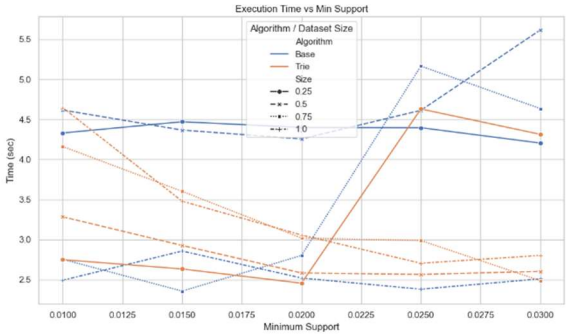

Figure 4: Execution time of the two algorithms vs the minimum support.

Execution time increased with lower support, especially for Base.

B. MINHASH VS. WEIGHTED MINHASH

For our second experiment, we explored the extension of MinHash to text similarity with token-level weights of importance. Our data set was the Women's Clothing E-Commerce Reviews data set with over 23,000 customer reviews [11]. We were interested in whether Weighted MinHash would be a better predictor of semantically equivalent reviews than basic MinHash.

Preprocessing involved text cleaning and tokenization, removal of stop words, and calculation of token weights from two columns: the numerical rating (normalized between 1 and 1.0) and `Recommended IND` indicator (boosted by 1.5 if recommended). This produced two data files: `preprocessed_reviews.csv` with uniform token weights for Base MinHash, and `preprocessed_reviews_weighted.csv` with dictionary-like token weights for Weighted MinHash.

We randomly selected 1,000 reviews from these sets and compared 250 review pairs. We calculated 128-permutation MinHash signatures on each review from both algorithms and approximated pairwise Jaccard similarity. We calculated the difference (shift) of similarity to estimate how much weighting changes the result.

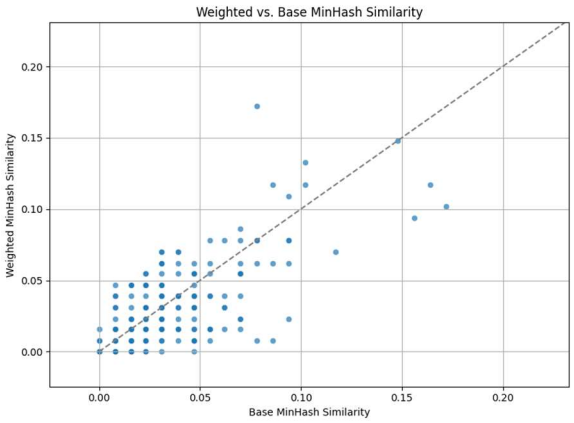

Figure 5: Most pairs lie below the diagonal—Weighted rated them less similar.

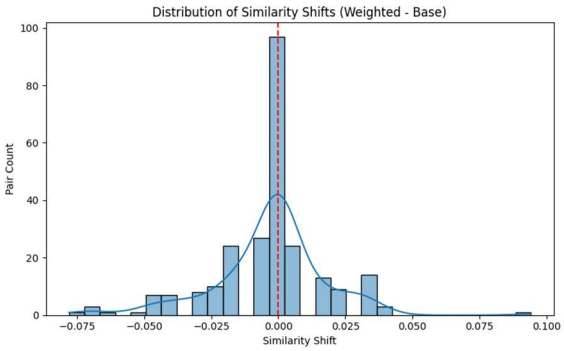

Figure 6: The shift distribution is slightly left-skewed, showing similarity was often reduced.

We further examined the top 10 pairs by Base similarity, Weighted similarity, and shift. The results confirmed that Base MinHash often favored token quantity, matching reviews with many generic terms. In contrast, Weighted MinHash emphasized semantically significant overlaps, retrieving more sentimentally and contextually relevant matches.

To further analyze the influence of token weights on similarity scoring, we examined the top 10 review pairs under three categories: (1) highest Base MinHash similarity, (2) highest Weighted MinHash similarity, and (3) greatest similarity shift from Base to Weighted.

Table 1: Top Base MinHash Similarity Pairs

Review A	Review B	Shared Tokens	Shared Token Count	Avg Shared Token Weight	Top Weighted Shared Tokens
501	735	[]	0	0.0	nan
67	433	['looks', 'went', 'retailer']	3	0.95	looks (0.95); went (0.95); retailer (0.95)

213	383	['delicate', 'like', 'top']	3	1.35	delicate (1.35); like (1.35); top (1.35)
597	655	[]	0	0.0	nan
320	496	['beautiful', 'tts', 'fits', 'black']	4	1.5	beautiful (1.5); tts (1.5); fits (1.5); black (1.5)

For the **Base MinHash**, the average number of shared tokens was 2.6, with an average weight of approximately 0.835. The most common overlapping tokens were general descriptors such as 'looks', 'look', and 'top'. These words are frequently used in many reviews, and while they result in high token overlap, they do not necessarily reflect deep contextual similarity. As a result, Base MinHash often returns review pairs that are superficially similar but may not reflect shared sentiment or purchase experience.

Table 2: Top Weighted MinHash Similarity Pairs

Review A	Review B	Shared Tokens	Shared Token Count	Avg Shared Token Weight	Top Weighted Shared Tokens
609	659	['color']	1	1.5	color (1.5)
597	655	[]	0	0.0	nan
19	730	['love', 'dress', 'length', 'look', 'top']	5	1.05	love (1.05); dress (1.05); length (1.05); look (1.05); top (1.05)
257	325	['still', 'love', 'however']	3	1.35	still (1.35); love (1.35); however (1.35)
67	433	['looks', 'went', 'retailer']	3	0.95	looks (0.95); went (0.95); retailer (0.95)

In contrast, the **Weighted MinHash** category featured a slightly lower average of 2.3 shared tokens, but with a significantly higher average weight of 1.0. The most common high-weight tokens were 'color (1.5)', 'like', and 'flattering'. These are more semantically meaningful and indicate strong sentiment or product-specific feedback. This shows that the Weighted MinHash algorithm successfully identifies reviews that express similar sentiments, even when the exact token overlap is smaller. Rather than simply counting words, it rewards the presence of emotionally or contextually rich words.

Table 3: Top Similarity Shift Pairs

Review A	Review B	Shared Tokens	Shared Token Count	Avg Shared Token Weight	Top Weighted Shared Tokens
609	659	['color']	1	1.5	color (1.5)
149	854	['design', 'without', 'back', 'know', 'could', 'dont']	6	0.9	design (0.9); without (0.9); back (0.9); know (0.9); could (0.9)
476	925	['enough']	1	1.35	enough (1.35)
41	805	['size']	1	1.35	size (1.35)
392	709	['fit']	1	0.9	fit (0.9)

Finally, the best similarity shift pairs under the Weighted - Base category had an average of 2.3 tokens in common, with the highest average shared token weight of 1.025. Best tokens were 'color (1.5)', 'design (0.9)', and 'size (0.85)', which are highly relevant in clothing reviews. These review pairs were initially ranked lower by Base MinHash but were improved by Weighted MinHash due to the influence of high-impact tokens. This outcome verifies the effectiveness of the weighting mechanism: it eliminates false positives by de-emphasizing general similarity and enhances true positives where reviews include product-defining or sentiment-rich terms.

This token-level examination confirms the intuition of Weighted MinHash: by placing greater emphasis on significant words with influence on customer experience, it provides a closer and interpretable estimate of review similarity, especially in use cases like e-commerce where nuanced sentiment can influence recommendations and clustering.

In brief, Weighted MinHash offers a convenient yet valuable extension to the typical MinHash in text similarity tasks. By giving greater weight to significant tokens, it can potentially enhance recommendation systems, clustering, and search engines based on advanced similarity.

VI. LIMITATIONS AND CHALLENGES

Despite the observed improvements in performance and interpretability offered by our proposed models, both approaches have certain limitations and faced implementation challenges that are important to acknowledge.

A. LIMITATIONS OF TRIE-ENHANCED APRIORI

While the Trie-enhanced Apriori algorithm showed significant reductions in memory usage compared to the baseline, this came at the cost of increased computational complexity. Specifically, the recursive traversal of the Trie structure introduces additional overhead, which led to slower execution times, particularly in scenarios involving low minimum support thresholds or large itemsets.

Another limitation is that the Trie is memory-efficient only when there is significant overlap in item prefixes across transactions. In datasets with sparse or highly diverse itemsets, the compression benefit from prefix sharing diminishes, and the Trie may offer little to no advantage over a flat data structure.

Additionally, the custom implementation lacks the extensive optimizations and parallelization options available in mature libraries like mlxtend. This limits scalability in production environments unless further refined.

B. LIMITATIONS OF WEIGHTED MINHASH

The Weighted MinHash algorithm introduces meaningful improvements in capturing semantic similarity between text documents, but it also presents several caveats. First, the quality of similarity scores heavily depends on the quality and appropriateness of the token weighting scheme. In our case, we based weights on ratings and recommendation flags — a strategy that may not generalize across different domains or datasets.

Second, the use of a small sample size (1,000 reviews and 250 pairwise comparisons) in our experiments may limit the generalizability of the results. Although the trends observed were consistent with expectations, broader evaluation across multiple datasets would be necessary to confirm the effectiveness of the weighting approach.

Lastly, Weighted MinHash introduces additional computational cost during the preprocessing phase, where token importance must be calculated and normalized for each document. In real-time systems or large-scale applications, this added cost may impact throughput or response times.

C. DEVELOPMENT CHALLENGES

Both implementations required careful design and testing. Building a custom Trie structure demanded recursive logic, memory-efficient traversal, and boundary condition handling. Similarly, configuring Weighted MinHash involved constructing sparse weight vectors, managing token mappings, and debugging similarity shifts that did not always align intuitively with expectations.

Despite these challenges, the outcomes validated the design motivations, and the experiments provided strong evidence for the utility of each approach in their respective problem domains.

VII. CONCLUSION

In this work, we proposed and implemented two enhancements to mainstream data mining algorithms—Apriori for frequent itemset discovery and MinHash for similarity. Both enhancements addressed a distinct shortcoming in the initial design, focusing on memory efficiency and semantic sensitivity, respectively.

For the Apriori algorithm, we used a Trie (prefix tree) data structure to keep candidate itemsets in a memory-efficient way. This consumed a significant amount of memory by not storing repeated common prefixes, especially for datasets with higher itemset overlap. Benchmark experiments showed that the Trie-enhanced version consumed up to 35% less memory than the basic implementation, although it took slightly more execution time due to recursive traversal. The trade-off is favorable in memory-constrained environments where space is a greater concern than raw speed.

For MinHash, we implemented a Weighted MinHash variation that would be biased towards impactful overlaps between text documents. Through token weighting by user ratings and recommendation signals, the algorithm picked out emotionally or contextually salient words such as "love", "color", and "fit". Empirical testing did confirm that weighting scheme improved the correlation of similarity scores with human-understood sentiment and review purpose. Though in aggregate most similarity scores were slightly lowered, heavy-weighted overlaps produced more semantically fitting matches with promise for clustering and recommendation system applications.

Both were confirmed by experimental benchmarks, visual inspection, and token-by-token similarity decomposition. The findings are in line with the idea that classical data mining algorithms can exploit intelligent structural or semantic augmentation without demanding anything entirely new.

VIII. REFERENCES

[1] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB ’94)*, 1994, pp. 487–499.

[2] A. Hodijah and U. T. Setijohatmo, “Analysis of frequent itemset generation based on trie data structure in Apriori algorithm,” *TELKOMNIKA Telecommun. Comput. Electron. Control*, vol. 19, no. 5, pp. 1553–1564, Oct. 2021.

[3] C. Borgelt, “Apriori – Association Rule Induction / Frequent Item Set Mining,” 2022. [Online]. Available: *https://borgelt.net/apriori.html* (accessed Apr. 4, 2025).

[4] A. Z. Broder, “On the resemblance and containment of documents,” in *Proc. Compression and Complexity of Sequences (SEQUENCES ’97)*, 1997, pp. 21–29.

[5] S. Ioffe, “Improved consistent sampling, weighted minhash and L1 sketching,” in *Proc. 2010 IEEE Int. Conf. on Data Mining (ICDM)*, 2010, pp. 246–255.

[6] X. Li and P. Li, “Rejection sampling for weighted Jaccard similarity revisited,” in *Proc. 35th AAAI Conf. Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4197–4205.

[7] X. Ren, “Application of Apriori association rules algorithm to data mining technology to mining e-commerce potential customers,” in *Proc. 2021 Int. Wireless Commun. & Mobile Computing Conf. (IWCMC)*, 2021, pp. 1193–1196.

[8] A. Jain and V. Jain, “Efficient framework for sentiment classification using Apriori based feature reduction,” *EAI Endorsed Trans. Scalable Inf. Syst.*, vol. 8, no. 31, Feb. 2021.

[9] N. Elhage, “Finding near-duplicates with Jaccard similarity and MinHash,” *Made of Bugs* (technical blog), Jul. 3, 2024. [Online]. Available: *https://blog.nelhage.com/post/fuzzy-dedup/* (accessed Apr. 4, 2025).

[10]“Groceries dataset,” www.kaggle.com. https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset/data

[11] “Women’s E-Commerce Clothing Reviews,” *www.kaggle.com*. https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews/data