Date of publication SEP-30, 2025, date of current version MAR-02, 2025.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.19.3.XX

Association Rule Mining in SQL to Improve Demand Forecasting using LSTM

PANDEGA ABYAN ZUMARSYAH[1], FARIZA EKA AULIA[1]
[1]Department of Electrical and Information Engineering, Universitas Gadjah Mada, Yogyakarta, Indonesia

Corresponding author: Fariza Eka Aulia (e-mail: farizaekaaulia@mail.ugm.ac.id).

ABSTRACT Association Rule Mining (ARM) and demand forecasting are vital in business intelligence, especially in commerce. ARM algorithms can be computationally expensive and require data migration to environments like R or Python. Meanwhile, demand forecasting often needs separate models for each product and is sensitive to data quality. To address ARM challenges, we implemented Apriori and Eclat algorithms, well-known itemsets generation algorithms, along with a rules generation algorithm in PostgreSQL. Additionally, we developed multivariate Long Short-Term Memory (LSTM) models with grouped data based on itemsets generation. Evaluation on the Online Retail II dataset (~one million rows) showed that our SQL ARM implementation achieved low overhead time (3.3s) and acceptable processing times (Eclat: 3s, rules generation: 0.1s). Our method is comparable to the state-of-the-art R implementations (overhead: 4.4s, Eclat: 0.66s, rules generation: 0.4s). For demand forecasting, the multivariate LSTM with grouped data reduced training time from 280.1s to 122.1s, improved Mean Squared Error from 1.045 to 0.882, and Mean Absolute Error from 0.471 to 0.433.

KEYWORDS Business Intelligence; Market Basket Analysis; Association Rule Mining; Apriori; Eclat; SQL; Forecasting; Demand Forecasting; Long Short-Term Memory; Multivariate.

I. INTRODUCTION

Predictive analytics is an important part of business intelligence that is useful in transforming raw data into meaningful information for improving organizational performance [1], [2]. It includes association rule mining (ARM) and demand forecasting [3], which is useful for commerce and other business cases.

ARM is useful for Market Basket Analysis to discover patterns of online purchasing behaviors of customers. Such information can be used for promotion, product placement, or other things [4]Meanwhile, demand forecasting is useful for predicting demand needs based on historical purchase data. As business develops, the growing sales data should be used to prepare supplies efficiently. Customers may be more likely to move to other competitors if their supply needs are unmet.

ARM allows decision makers to find a product set X that is frequently purchased, then find another product set Y that has a high probability of being related to X. Various algorithms have been proposed for ARM, such as Apriori [5], FP-tree [6], and Eclat [7].

These algorithms have been used to solve many cases in commerce. Hamdani et al. [8] used the Apriori algorithm with RapidMiner software to calculate minimum support and confidence values. The algorithm generates association rules that reflect purchasing activities. Fajrianti et al. [9] analyzed historical data from visitors and customers using the Apriori algorithm. From the analysis, user preferences can be identified, allowing the creation of accurate product recommendations. Experiments show that ARM can generate precise recommendations with a confidence value of 76.92%. Hameed et al. [10] conduct research to help small online gift store retailers that have not yet implemented a recommendation system using association rules to provide personalized product recommendations to their customers. The results highlight the top 10 frequent itemsets and their support values, as well as a list of association rules, showing that some items are often bought together. Sipahutar et al. [11] developed a website application using the ARM with the Apriori algorithm to identify patterns of relationships between products purchased by customers. The system, built using PHP and 4,820 transaction data entries, analyzed the Antoni store dataset to derive association rules. Another research done by Wang et al. [12] to address the issues of inefficient mining and insufficient rules caused by the large and complex data, an improved method combining K-Means and Eclat algorithms is proposed.

The search space for ARM is exponential in the number of items in the database. This can lead to high computational costs and long processing times, especially with large datasets [5].

Since the Apriori algorithm needs to scan the database many times, large datasets will make it significantly slower and more memory-intensive [13]. The Eclat algorithm does not require multiple scans. However, its performance can degrade with large datasets due to the need to intersect arrays for candidate generation [14]. In addition, the implementation of ARM sometimes requires data migration from a database to environments such as R [15] or Python [16].

Meanwhile, many operations are known to be much more efficient when performed using SQL directly on the database. There are studies implementing linear algebra operations [17], Einstein summation [18], and even a full machine learning workflow [19] in SQL. These studies demonstrated that SQL offers near-data processing, portability, and efficient computation compared to other programming languages. With this consideration, an optimized implementation of ARM using SQL can be a useful alternative.

Related to demand forecasting, deep learning has been widely used. Deep learning is a subset of artificial intelligence that mimics the human brain using neural networks. It has become prevalent in various fields such as healthcare, visual recognition, text analytics, and cybersecurity [20].

Several studies have used deep learning forecasting to predict demand using historical data. Chandriah et al. [21] proposes research using Recurrent Neural Networks (RNN) and Long-Short Term Memory (LSTM) with a modified Adam optimizer to predict spare parts demand. Meanwhile, Li et al. [22] compares Light Gradient Boosting Machine (LGBM) and LSTM sales prediction models using real sales data. Models forecast product sales for stores over the next three days. Another study by Taha et al. [23] compares the quality of two methods, Seasonal Autoregressive Integrated Moving Average (SARIMA) and LSTM. Their study uses actual retail sales data from an Austrian retailer. Both models produced reasonable to good results, with LSTM performing better for products with stable demand and SARIMA showing better results for products with seasonal behavior. In addition, Zhang et al. [24] combine LSTM with attention mechanism that achieves better performance but needs more computational resources than traditional methods.

Demand or time series forecasting is highly sensitive to the quality of the data. Missing values, outliers, and noise can significantly affect the accuracy of forecasts. Thus, data pre-processing is crucial [25]. Overfitting is also a common problem in time series forecasting, especially when using complex models such as neural networks [26]. Furthermore, demand forecasting may need separate models for different products or items, increasing the complexity and computational cost. Selecting an appropriate model for time series forecasting can also be challenging since there are numerous models with their own strengths and weaknesses. Choosing the wrong model can lead to poor performance [27].

In this study, LSTM is chosen simply because it is widely used in recent studies and performs well in many cases [21], [22], [23], [24]. However, instead of using single product data, we propose the usage of grouped product data. Thus, the task can be considered multivariate time series forecasting and LSTM can be used for such task [28].

The grouping is based on the results of ARM algorithms. We believe this approach can reduce the effect of bad data quality, avoid overfitting, and reduce computational time and cost. A study also showed that multivariate forecasting performs better than univariate ones [29].

The contributions of this study are as follows:

- Implementation of ARM algorithms, including Apriori and Eclat, in SQL
- Evaluation of ARM implementations in R, Python, and SQL
- Development, training, and evaluation of LSTM models for demand forecasting of multiple products
- Usage of week sequence and grouped product based on ARM results for improving the performance of LSTM models

II. MATERIAL AND METHODS

A. DATASET

The dataset used in this study is Online Retail II, which is available at the UCI Machine Learning Repository [30]. It contains transactions from a UK-registered online retail between 2009-12-01 and 2011-12-09. The business primarily offers a variety of unique gifts, with the majority of customers being wholesalers.

The dataset consists of two CSV files with the same columns. When combined, the dataset contains 1,067,371 rows, excluding the header. Each row represents a transaction of a product and contains the following columns [30]:

- *InvoiceNo*: A unique 6-digit number assigned to each transaction, where a leading “C” denotes a cancellation.
- *StockCode*: A unique, most likely 5-digit number identifying each product.
- *Description*: The name of the product.
- *Quantity*: The number of units of the product purchased in the transaction.
- *InvoiceDate*: The date and time when the transaction occurred in the format “M/d/yyyy H:mm”.
- *UnitPrice*: The cost per unit of the product in pounds sterling (£).
- *CustomerID*: A unique 5-digit number identifying the customer.
- *Country*: The country where the customer is located.

Some processing steps were taken to clean the dataset:

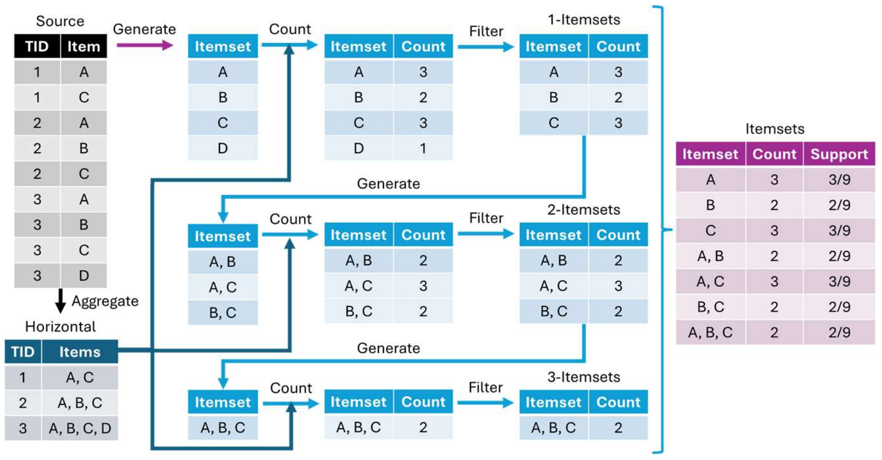
1. Remove rows where the *InvoiceNo* contains the letter "A" or "C".
2. Remove rows with missing (NaN) values in the *Description*.
3. Remove rows where the *Quantity* is less than 0.
4. Convert the *InvoiceDate* column to a more standard datetime format “yyyy-MM-dd HH:mm:ss”.

The cleaned dataset contains 1,018,784 rows. In this study, only some columns are used: *InvoiceNo*, *StockCode*, *Quantity*, and *InvoiceDate*.

B. ARM ALGORITHMS

The ARM process can be divided into two main steps: frequent itemsets generation and rules generation. For the former, two popular algorithms are Apriori [5] and Eclat [7]. Both take a minimum support threshold (*min_supp*) as input. In ARM, the support of an itemset is the proportion of transactions that contain the itemset [16]:

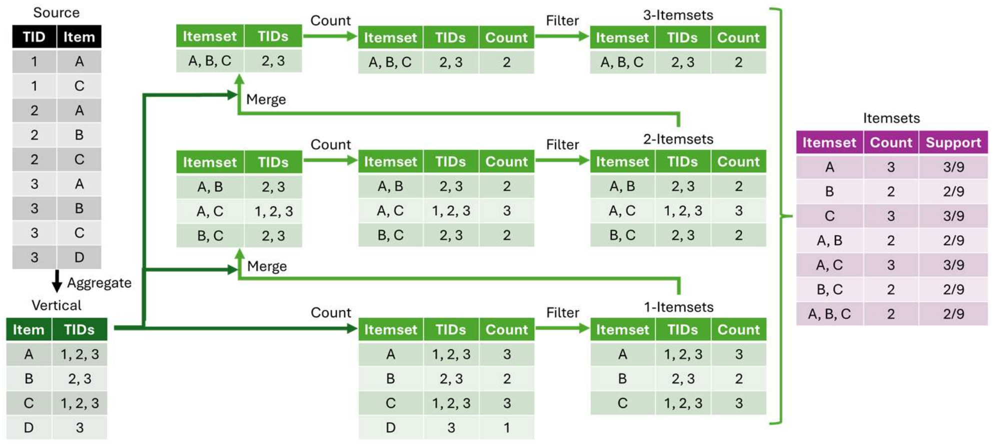
$$\text{supp}(X) = \frac{\text{number of transactions with } X}{\text{total number of transactions}}. \quad (1)$$



TID	Items
1	A, C
2	A, B, C
3	A, B, C, D

Itemset	Count	Support
A	3	3/9
B	2	2/9
C	3	3/9
A, B	2	2/9
A, C	3	3/9
B, C	2	2/9
A, B, C	2	2/9

Figure 1. Flow of Apriori Algorithm.

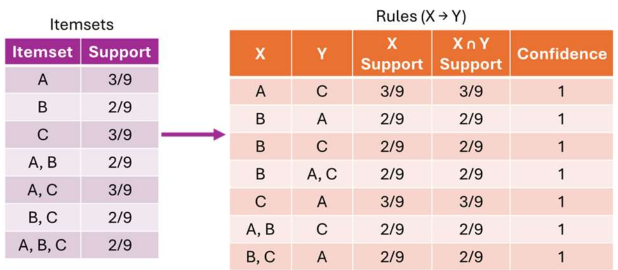


Item	TIDs
A	1, 2, 3
B	2, 3
C	1, 2, 3
D	3

Figure 2. Flow of Eclat Algorithm.

Fig. 1 shows the flow of the Apriori algorithm. Recent studies have used similar flows [31], [32]. In this case, *TID* refers to the transaction ID while *Item* refers to the product. In Apriori, the source data is aggregated to create a horizontal view. Candidates for the 1-itemsets are generated directly from the source data. Then, the candidates are counted using the horizontal view and filtered based on *min_supp*, creating the frequent 1-itemsets. Candidates for the *k*-itemsets are generated from the frequent (*k-1*)-itemsets. Candidates are counted and filtered as before.

Fig. 2 shows the flow of the Eclat algorithm. There are also recent studies using similar flows [33], [34]. The flow starts by aggregating the source data to create a vertical view. Candidates for the 1-itemsets are simply the vertical view. Since the candidates contain TIDs, counting can be done easily. The candidates are filtered based on *min_supp*, creating the frequent 1-itemsets. Candidates for the *k*-itemsets are created by merging the frequent (*k-1*)-itemsets and the vertical view in some way. Candidates are counted and filtered as before.



Itemset	Support
A	3/9
B	2/9
C	3/9
A, B	2/9
A, C	3/9
B, C	2/9
A, B, C	2/9

X	Y	X Support	X ∩ Y Support	Confidence
A	C	3/9	3/9	1
B	A	2/9	2/9	1
B	C	2/9	2/9	1
B	A, C	2/9	2/9	1
C	A	3/9	3/9	1
A, B	C	2/9	2/9	1
B, C	A	2/9	2/9	1

Figure 3. Association Rules Generation from Itemsets

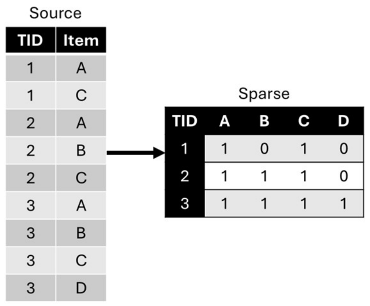
The generated frequent itemsets are used to generate association rules as in Fig. 3. A rule ($X \rightarrow Y$) is a pair of itemsets, antecedent (X) and consequent (Y), where the antecedent implies the consequent. The rules generation takes a minimum confidence threshold (*min_conf*) as input. In ARM, confidence is the conditional probability of the consequent given the antecedent [16]:

$$\mathrm{conf}(X \rightarrow Y) = P(Y|X) = \frac{\mathrm{supp}(X \cap Y)}{\mathrm{supp}(X)}. \qquad (2)$$

In addition, there are other metrics that can be used, such as lift, leverage, conviction, and Zhang [16].

C. ARM EXISTING IMPLEMENTATIONS

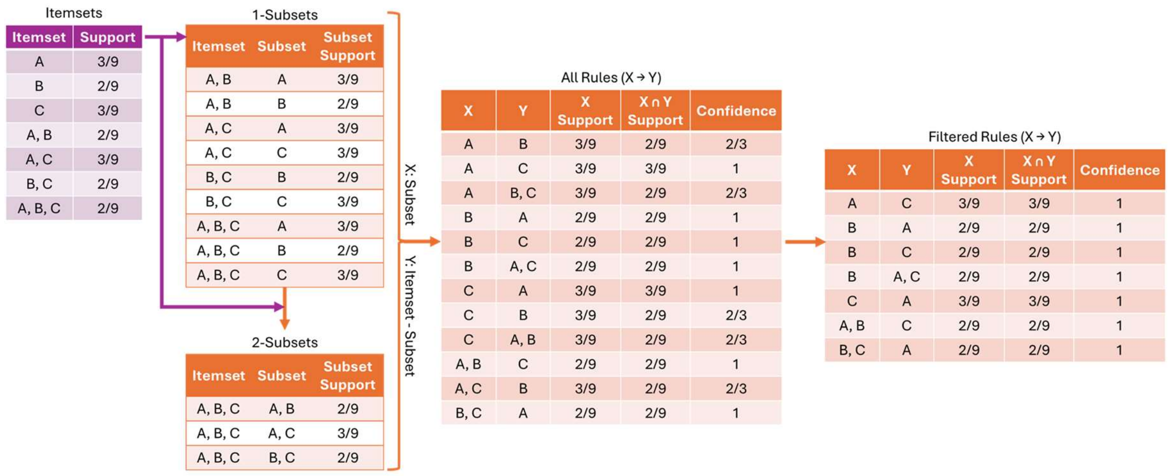
There are many implementations of ARM in various programming languages. One of the most powerful is the *arules* package in R [15]. It provides a wide range of functions for ARM, including the Apriori and Eclat algorithms.



TID	Item
1	A
1	C
2	A
2	B
2	C
3	A
3	B
3	C
3	D

TID	A	B	C	D
1	1	0	1	0
2	1	1	1	0
3	1	1	1	1

Figure 4. Sparse Binary Matrix from Source Data



Itemsets

Itemset	Support
A	3/9
B	2/9
C	3/9
A, B	2/9
A, C	3/9
B, C	2/9
A, B, C	2/9

1-Subsets

Itemset	Subset	Subset Support
A, B	A	3/9
A, B	B	2/9
A, C	A	3/9
A, C	C	3/9
B, C	B	2/9
B, C	C	3/9
A, B, C	A	3/9
A, B, C	B	2/9
A, B, C	C	3/9

2-Subsets

Itemset	Subset	Subset Support
A, B, C	A, B	2/9
A, B, C	A, C	3/9
A, B, C	B, C	2/9

All Rules (X → Y)

X	Y	X Support	X ∩ Y Support	Confidence
A	B	3/9	2/9	2/3
A	C	3/9	3/9	1
A	B, C	3/9	2/9	2/3
B	A	2/9	2/9	1
B	C	2/9	2/9	1
B	A, C	2/9	2/9	1
C	A	3/9	3/9	1
C	B	3/9	2/9	2/3
C	A, B	3/9	2/9	2/3
A, B	C	2/9	2/9	1
A, C	B	3/9	2/9	2/3
B, C	A	2/9	2/9	1

Filtered Rules (X → Y)

X	Y	X Support	X ∩ Y Support	Confidence
A	C	3/9	3/9	1
B	A	2/9	2/9	1
B	C	2/9	2/9	1
B	A, C	2/9	2/9	1
C	A	3/9	3/9	1
A, B	C	2/9	2/9	1
B, C	A	2/9	2/9	1

Figure 5. Flow of Association Rules Generation.

For Python, one of the most popular libraries is *mlxtend* [16]. It consists of useful tools for various data science tasks, including ARM. However, it does not provide an Eclat implementation. There is another library called *pyECLAT* [35], but it is not as optimized as *mlxtend*.

Itemset generation in all three libraries takes a sparse matrix as input. The matrix is a binary matrix where rows represent transactions and columns represent items. Thus, the source data must be transformed into a format as shown in Fig. 4.

D. ARM SQL IMPLEMENTATIONS

While there are some SQL implementations of ARM, it is not as many and powerful as the R ones. Thus, we develop our own SQL implementation of Apriori, Eclat, and rules generation. It is done in PostgreSQL, a powerful open-source relational database management system. To improve performance, the source data elements are transformed to integers and the *intarray* extension is used to handle arrays of integers.

Listing 1 shows the SQL implementation of Apriori based on Fig. 1. The implementation uses common table expressions (CTE) to create modular and readable code.

Listing 1. SQL Implementation of Apriori

```
WITH item_counts AS (
SELECT item, count(*) AS cnt
FROM source_data
GROUP BY item HAVING count(*) > %L
), horizontal_data AS (
SELECT sd.tid, array_agg(sd.item) AS items
FROM source_data sd
INNER JOIN item_counts ic ON sd.item = ic.item
GROUP BY sd.tid
), itemsets_1 AS (
SELECT ARRAY[item] AS itemset, cnt
FROM item_counts
), itemsets_2 AS (
SELECT
  (s.itemset || ic.item) AS itemset, count(*) AS cnt
FROM itemsets_1 s
INNER JOIN item_counts ic ON ic.item > s.itemset[1]
INNER JOIN horizontal_data hd
  ON hd.items @> (s.itemset || ic.item)
GROUP BY s.itemset, ic.item HAVING count(*) > %L
)
...
```

In Listing 1, CTE *item_counts* is a helper CTE that counts the occurrences of each item in the source data. CTE *horizontal_data* is the horizontal view created using array aggregation. Joining with *item_counts* filters out infrequent items. CTE *sets_1* simply generates the *1*-itemsets from *item_counts*. The next CTEs generate the *k*-itemsets from the (*k-1*)-itemsets. Joining with *item_counts* expands the itemsets and generates the candidates. Meanwhile, joining with *horizontal_data*, followed by the use of GROUP BY and HAVING statements, counts itemsets occurrences and filters out infrequent itemsets. The itemsets CTEs are then combined using UNION to generate the frequent itemsets. Regarding the symbols, "%L" is a placeholder for *min_supp*. Operation "A || B" concatenates arrays A and B while operation "A @> B" checks if array A contains array B.

Listing 2 shows the SQL implementation of Eclat based on Fig. 2. CTE *vertical_data* is the vertical view created using array aggregation. CTE *itemsets* is a recursive CTE containing base and recursive terms. The base one simply generates the *1*-itemsets from *vertical_data*. The recursive one generates the *k*-itemsets from the (*k-1*)-itemsets. The (*k-1*)-itemsets are joined with *vertical_data* to expand and generate the candidates. The candidates are directly filtered using the WHERE statement.

Listing 2. SQL Implementation of Eclat

```
WITH RECURSIVE vertical_data AS (
SELECT item, array_agg(tid) AS tids
FROM source_data
GROUP BY item HAVING count(*) > %L
), itemsets AS (
SELECT ARRAY[item] AS itemset, tids
FROM vertical_data
UNION ALL
SELECT (i.itemset || p.item) AS itemset,
  (i.tids & p.tids) AS tids
FROM itemsets i
INNER JOIN vertical_data vd
  ON vd.item > i.itemset[icount(i.itemset)]
WHERE icount(i.tids & vd.tids) > %L
  AND icount(i.itemset) < %L
)
SELECT itemset, icount(tids) AS cnt
FROM itemsets
```

In Listing 2, the first and second "%L" are placeholders for *min_supp* while the third "%L" is a placeholder for the maximum number of items in an itemset (*max_len*). Operation "A & B" returns the intersection of arrays A and B. The intarray function *icount()* from *intarray* returns the number of elements in an array.

Compared to the Apriori implementation, the Eclat implementation is better due to various reasons:

- The vertical view usually has fewer rows than the horizontal view, resulting in faster processing.

- Stored TIDs can be used to directly count itemset occurrences, instead of using JOIN and GROUP BY statements.
- Filtering using the WHERE statement is simpler and more efficient than using the HAVING statement.
- The absence of GROUP BY and HAVING allows the use of recursive CTE, which is simpler and more efficient.

After generating the frequent itemsets, the next step is to generate the association rules. Listing 3 shows the SQL implementation with Fig. 5 illustrating the flow. It takes *itemsets* and *min_conf* as inputs. First, the function *unnest()* is used to create CTE *unnested* by unnesting *itemset* column of *itemsets*. Next, CTE *subsets* contains subsets of itemsets that are generated recursively. The base term creates the *1*-subsets via a simple transformation of *unnested* and looking up the support of the subsets from *itemsets*. The recursive term generates the *k*-subsets by joining the (*k-1*)-subsets with *unnested*. There is also a lookup of the support of the subsets. The association rules are generated by obtaining the antecedent, consequent, support of antecedent, support of union, and confidence. Finally, the rules are filtered based on *min_conf*. It should be noted that metrics other than confidence can be easily calculated using SQL statements.

Listing 3. SQL Implementation of Association Rules Generation

```sql
WITH RECURSIVE unnested AS (
SELECT itemset, supp, unnest(itemset) AS item
FROM itemsets
), subsets AS (
SELECT itemset, supp, ARRAY[item] AS subset,
   (SELECT supp FROM itemsets
     WHERE itemset = ARRAY[item]) AS sub_supp
FROM unnested
UNION ALL
SELECT u.itemset, u.supp, s.subset || u.item,
   (SELECT supp FROM itemsets
     WHERE itemset = (s.subset || u.item))
FROM subsets s
INNER JOIN unnested u
   ON s.itemset = u.itemset
     AND icount(s.subset) < icount(s.itemset) - 1
     AND u.item > s.subset[icount(s.subset)]
)
SELECT
   subset AS ante, (itemset - subset) AS cons,
   subset_supp AS ante_supp, supp AS union_supp,
   (supp/subset_supp) AS confidence
FROM subsets
WHERE icount(itemset) > 1 AND (supp/subset_supp) > %L
```

E. FORECASTING NETWORKS

Fig. 6 is the neural networks model for the forecasting. It takes *F*-features *S*-steps data as input and outputs *G*-features 1-step forecast. The values of *F*, *S, and G* are based on the scenario. There are four LSTM layers with 64 units each. The last output of the last layer is fed into two dense layers with 32 and *G* units, respectively. The activation function is "tanh" for the LSTM layers and *dense_act* for the dense layers. All layers are subject to L1 regularization with a value of *l1_reg*. LSTM layers are also subject to dropout with a value of *drop_rate*.

The model is compiled using the Adam optimizer [36] with the learning rate *lr*. The loss function and the primary metric are "mean_squared_error" (MSE). The variables *dense_act*, *l1_reg*, *drop_rate*, and *lr* are subject to hyperparameter tuning.

F. FORECASTING SCENARIOS

In this study, forecasting is performed on the weekly quantity of some products. Six high-selling products are selected. The weekly quantity is preferred over the daily and monthly quantities because the daily quantity contains many zero values while the monthly quantity contains too few data points. Since the first and last weeks may be incomplete, they are removed, resulting in 104 weeks. Thus, the used data contains six sequences of 104 data points each.

In this study, there are four forecast scenarios. All use a sequence length of 12 weeks (*S=12*). The first uses simple univariate models where it takes a sequence (*F=1*) from a product and returns a value as a forecast of that sequence (*G=F=1*). In this scenario, each product is forecast separately using different models. It is not efficient in terms of computation and the number of models.

The second scenario addresses that issue by using multivariate models. It takes multiple sequences from a group of products (F>1) and returns multiple values as the forecast for each sequence (*G=F>1*). In this scenario, the groupings are based on the results of itemsets generation. Products in the same itemset with high support are grouped together. The number of models is reduced and the computation can be more efficient.

The third and fourth scenarios are similar to the first and second, respectively, but with the addition of a week sequence. Thus, the number of input features is increased by one (*F=G+1*). The week sequence *v* is generated from sinusoidal encoding of the week number *w* that goes from 1 to 104:

$$v = \sin(\pi w/26 - 5). \qquad (3)$$

The encoding is designed so that the end of quarter *q* is represented by *q* times 90 degrees angle. With the addition of the week sequence, the model is expected to capture the seasonality of the data.

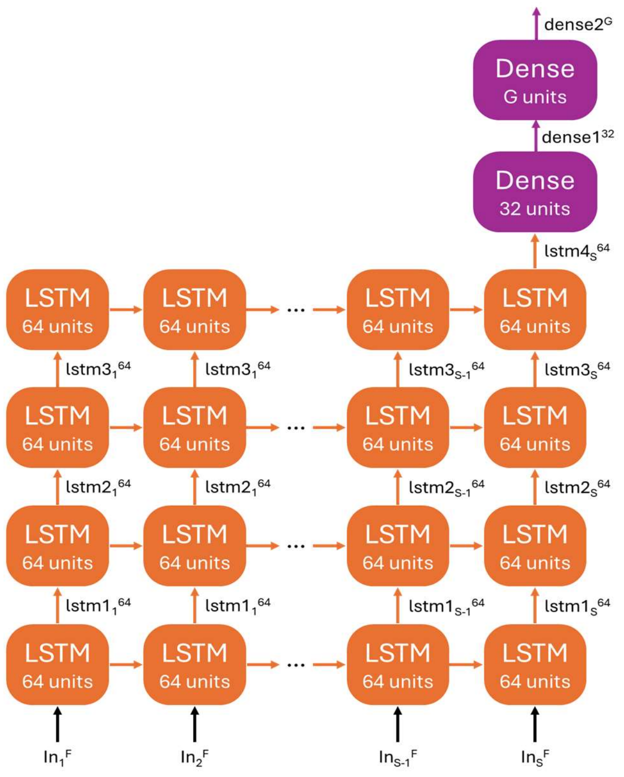
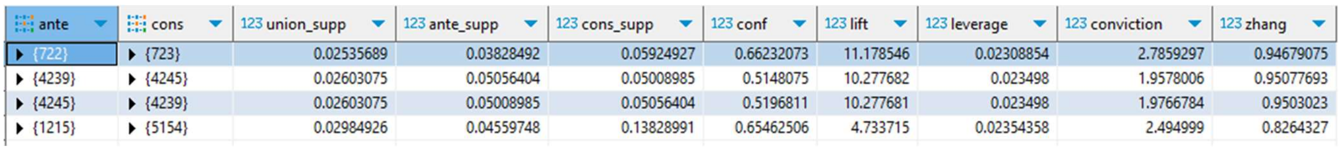


Figure 6. Neural Networks Architecture for Forecasting



ante	cons	union_supp	ante_supp	cons_supp	conf	lift	leverage	conviction	zhang
{722}	{723}	0.02535689	0.03828492	0.05924927	0.66232073	11.178546	0.02308854	2.7859297	0.94679075
{4239}	{4245}	0.02603075	0.05056404	0.05008985	0.5148075	10.277682	0.023498	1.9578006	0.95077693
{4245}	{4239}	0.02603075	0.05008985	0.05056404	0.5196811	10.277681	0.023498	1.9766784	0.9503023
{1215}	{5154}	0.02984926	0.04559748	0.13828991	0.65462506	4.733715	0.02354358	2.494999	0.8264327

Figure 7. Results and Metrices of Rules Generation using SQL

G. FORECASTING IMPLEMENTATIONS

Forecasting is done using Python with a minor involvement of SQL queries. *NumPy* [37] and *pandas* [38] are used for data manipulation and processing. *Tensorflow Keras* [39] is used to build, train, and evaluate the neural networks model.

Some processing steps need to be done before the data are used for training and evaluation:

1. Each sequence should be split into input and output sequences using the sliding window technique. In this case, a sequence of length 104 is split into (104 - 12) pairs of input sequences of length 12 and output sequences of length 1.
2. Each split input sequences are scaled so that the minimum and maximum values are 0 and 1, respectively. The output sequences should be scaled accordingly.
3. Pairs of scaled input and output sequences are split into train, validation, and test sets with a ratio of 80%, 10%, and 10%, respectively.

Grid search is used to find the best hyperparameters for the model. It is implemented manually using validation MSE as the metric. The hyperparameters and their ranges are as follows:

- *dense_act*: [“relu”, None]
- *l1_reg*: [0.0001, 0.00001, 0.000001]
- *drop_rate*: [0.2, 0.4]
- *lr*: [0.001, 0.0001, 0.00001]

The best hyperparameters are used to train the final models using the train set. The final models are evaluated using the test set. Besides MSE, there are Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Mean Squared Logarithmic Error (MSLE) to compare models predictions ($\hat{y}$) and true values (y):

$$MAE = \frac{1}{\mathrm{n}} \sum |y - \hat{y}| \tag{4}$$

$$MAPE = \frac{100}{\mathrm{n}} \sum \left| \frac{y - \hat{y}}{y} \right| \tag{5}$$

$$MSLE = \frac{1}{\mathrm{n}} \sum (\log(1 + y) - \log(1 + \hat{y}))^2. \tag{6}$$

III. RESULTS

A. ARM PERFORMANCE

Table 1. Processing Time Comparison of Multiple ARM Implementations in Cleaned Dataset with Parameters: *min_supp* = 0.025, *max_len* = 5, *min_conf* = 0.5

Language	Package / Tools	OH Time	Apriori Time	Eclat Time	Rules Time
Python	*mlxtend* [16] & *PyECLAT* [35]	92 s	1.93 s	too long	0.01 s
R	*arules* [15]	4.4 s	0.7 s	0.66 s	0.4 s
SQL	-	3.3 s	780 s	3 s	0.1 s

Table 1 shows the processing times of the ARM implementations in R, Python, and SQL. Processing times can be divided into four parts: data preparation or overhead (OH), Apriori, Eclat, and rules generation. Data preparation in R and Python is sparse matrix generation (excluding data migration) while in SQL is materialized view creation. The materialized view is the simplified version of the source data with two indexed integer columns, one representing transactions while the other representing items.

SQL has the fastest overhead time (3.3 s) since the operation is simpler. Python overhead time (92 s) is much slower than R time (4.4 s) despite obtaining the same output. In terms of overhead space, the created materialized view only takes 50 MB while the sparse matrix takes 188 MB in Python and 65 MB in R. However, the sparse matrix in R can be compressed to only 4.4 MB as transaction data. It should be noted that the materialized view uses disk space while the sparse matrix uses memory space. Moreover, the materialized view has a general structure that can be used for other purposes.

Itemsets generation in R is the fastest (0.66 s), followed by Python (1.93 s) and SQL (3 s). The SQL implementation of Apriori is much slower than that of Eclat. It is expected since the implementation is not efficient, as explained before. The Python implementation of Eclat is too slow to be measured. When checking the PyECLAT source code, we found that it tries to generate all possible itemsets, which is very inefficient.

Rules generation in Python is the fastest (0.01 s), followed by SQL (0.1 s) and R (0.4 s). Since only less than 200 itemsets are used as input, the rules generation is fast. However, the rules generation in SQL and R is slower than expected. The SQL one may be slow since it generates many possible subsets before filtering them. It should be noted that rules generation in Python and SQL also includes the calculation of various metrics, not only support and confidence as in Listing 3. Fig. 7 shows some of the SQL-generated rules with their metrics.

B. FORECASTING PERFORMANCE

As previously mentioned, the results of itemsets generation are used to group products for forecasting in scenarios 2 and 4. There selected itemsets as in Fig. 7 are {722, 723}, {4239, 4245}, and {1215, 5154}. Thus, there are six items for the I/III scenario and three groups of two items for the II/IV scenario. Fig. 8 shows the weekly quantity of two products in one of the groups. It can be seen that the sequences are very noisy and hard to predict.

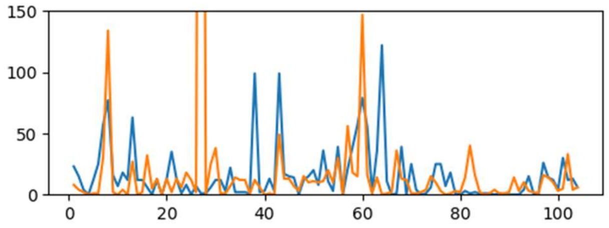


Figure 8. Sequence Samples of Two Products in the Same Group, x-axis is quantity while y-axis is week

Table 2. Performance Comparison of Multiple Scenarios

Metrics	Scenario I	Scenario II	Scenario III	Scenario IV
train duration	280.1 s	**122.6 s**	468.4 s	176.3 s
train MSE	0.21007	0.58666	**0.13754**	0.41921
val MSE	0.20773	0.20782	**0.16838**	0.16840
test MSE	1.04456	**0.88170**	0.98381	0.92671
test MAE	0.47079	**0.43305**	0.48488	0.43471
test MAPE	422.336	389.610	386.394	**326.256**
test MSLE	0.13476	**0.11648**	0.15214	0.12705

Table 2 shows the performance comparison of the four forecasting scenarios. The train duration is the total time taken to train all models, six for the I/III scenario and three for the II/IV scenario. All other metrics correspond to the best models that are obtained from the grid search and evaluated in train, validation, and test sets. Train and validation metrics are not discussed here. They are not as valid as test metrics since the models have already "seen" the train and validation sets.

Since the data is quite challenging, the errors can be considered high. However, the obtained metrics can still be used to compare the scenarios.

Product grouping can reduce the number of models and computation time. In this case, the training time is reduced by more than half. Moreover, it can be seen that product grouping can improve performance in all test metrics.

The addition of a week sequence can also improve performance, but at the cost of increased computation time. In this case, the training time is increased by about 40 - 70 percent. Comparing I with III scenarios, week sequence addition gives better MSE and MAPE but slightly worse MAE and MSLE. Comparing II with IV scenarios, week sequence addition only improves MAPE while the other metrics are worse. It can be concluded that adding the week sequence increases computation time but does not guarantee better performance.

IV. CONCLUSIONS

In this study, we develop SQL implementations of ARM algorithms, including Apriori, Eclat, and rules generation. The implementations are done in PostgreSQL by utilizing the extension 'intarray', CTE, materialized view, and recursive query. We also develop LSTM models to forecast the weekly quantity of products. The models are built using Tensorflow Keras with grid search for hyperparameter tuning. Forecasting is done in four scenarios: single product, grouped products, single product with week sequence, and grouped products with week sequence. The grouping is based on the results of itemsets generation. Our methods are evaluated on a challenging dataset with around one million rows named Online Retail II.

Evaluation shows that our SQL implementations of ARM have the lowest overhead time (3.3s) compared to R (4.4s) and Python (92s). The R implementation of itemsets generation is the fastest (Apriori: 0.7s, Eclat: 0.66s), but Python (Apriori: 1.93s) and SQL (Eclat: 3s) are not far behind. The current Python implementation of Eclat is too slow to be measured, while the SQL implementation of Apriori is hard to optimize. Rules generation in Python is the fastest (0.01s), followed by SQL (0.1s) and R (0.4s). It can be concluded that SQL is a viable alternative for ARM implementations.

Related to forecasting, grouping products reduces the number of models and reduces training time by more than half. Grouping products also improves performance on all test metrics. Meanwhile, week sequence addition significantly increases training time by about 40 - 70 percent, but does not guarantee better performance.

In the future, SQL implementations of ARM can be further optimized. SQL can perform better than R or Python, but current SQL implementations are relatively less mature. Meanwhile, the forecasting scenario of grouped products can be further evaluated using other datasets and other models.

References

[1] D. Jangam and A. R. Deshpande, “Business analytics using predictive algorithms,” *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, issue 8s, pp. 595–609 2023, https://doi.org/10.17762/ijritcc.v11i8s.7242.

[2] A. Gunasekaran *et al.*, “Big data and predictive analytics for supply chain and organizational performance,” *Journal of Business Research*, vol. 70, pp. 308–317, 2017, https://doi.org/10.1016/j.jbusres.2016.08.004.

[3] B. Tierney, *Predictive Analytics using Oracle Data Miner: Develop & use Data Mining Models in ORACLE DATA MINER, SQL & PL/SQL*. New York: McGraw-Hill Education, 2014.

[4] B. Christian and K. Rudolf, “Induction of association rules: Apriori implementation,” *Proceedings of the Compstat 2002*, Berlin, Germany: Physica, Heidelberg, 2002, pp. 395–400. https://doi.org/10.1007/978-3-642-57489-4_59.

[5] R. Agrawal, R. Srikant, “Fast algorithms for mining association rules,” *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, Santiago, Chile, September 1994, pp. 487–499. [Online]. Available at: http://www.vldb.org/conf/1994/P487.PDF.

[6] H. Jiawei and P. Jian, “Mining frequent patterns without candidate generation,” *ACM SIGMOD Record*, vol. 29, no. 2, pp. 1–12, 2000, https://doi.org/10.1145/335191.335372.

[7] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000, https://doi.org/10.1109/69.846291.

[8] S. M. Hamdani, “Application of association rule method using apriori algorithm to find sales patterns case study of indomaret tanjung anom,” *Brilliance: Research of Artificial Intelligence*, vol. 1, no. November 2021, pp. 54–66, 2021, https://doi.org/10.47709/brilliance.v1i2.1228.

[9] S. Fajrianti, O. M. Prabowo, R. R. Nurmalasari, and R. Ramdani, “Application of the association rule in online stores using apriori algorithm for product recommendations at promotional events,” *Proceedings of the 2023 9th International Conference on Wireless and Telematics (ICWT)*, Jul. 2023, pp. 1–5. https://doi.org/10.1109/ICWT58823.2023.10335465.

[10] V. A. Hameed, M. E. Rana, and L. H. Enn, “Apriori algorithm based association rule mining to enhance small-scale retailer sales,” *Proceedings of the 2023 IEEE 6th International Conference on Big Data and Artificial Intelligence (BDAI)*, Jul. 2023, pp. 187–191. https://doi.org/10.1109/BDAI59165.2023.10256952.

[11] S. Y. K. Sipahutar, A. A. Panjaitan, D. P. Sitanggang, and I. Fitriyaningsih, “Implementation of association rules with apriori algorithm in determining customer purchase patterns,” *Proceedings of the 2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM)*, Oct. 2022, pp. 1–6. https://doi.org/10.1109/ICOSNIKOM56551.2022.10034921.

[12] L. Wang, Y. Guo, and Y. Guo, “An improved eclat algorithm based association rules mining method for failure status information and remanufacturing machining schemes of retired products,” *Proceedings of the 16th CIRP Conference on Intelligent Computation in Manufacturing Engineering, CIRP ICME'22*, Italy: Procedia CIRP, 2023, pp. 572–577. https://doi.org/10.1016/j.procir.2023.06.098.

[13] C. Borgelt, “An implementation of the FP-growth algorithm,” *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, 2010, pp. 1-5. https://doi.org/10.1145/1133905.1133907.

[14] G. G., “Efficiently Using Prefix-trees in Mining Frequent Itemsets,” *presented at the FIMI'03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations*, Melbourne, Florida, USA, 2003. [Online]. Available at: https://www.researchgate.net/publication/220845998_Efficiently_Using_Prefix-trees_in_Mining_Frequent_Itemsets.

[15] M. Hahsler, B. Grün, and K. Hornik, "Arules - A Computational Environment for Mining Association Rules and Frequent Item Sets," *J. Stat. Soft.*, vol. 14, no. 15, 2005, https://doi.org/10.18637/jss.v014.i15.
[16] S. Raschka, "MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack," *JOSS*, vol. 3, no. 24, p. 638, 2018, https://doi.org/10.21105/joss.00638.
[17] M. Blacher, J. Giesen, S. Laue, J. Klaus, and V. Leis, "Machine learning, linear algebra, and more: Is SQL all you need?," *Proceedings of the Conference on innovative data systems research*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249872850.
[18] M. Blacher, J. Klaus, C. Staudt, S. Laue, V. Leis, and J. Giesen, "Efficient and portable Einstein summation in SQL," *Proc. ACM Manag. Data*, vol. 1, no. 2, p. 121:1-121:19, 2023, https://doi.org/10.1145/3589266.
[19] M. Schule, H. Lang, M. Springer, A. Kemper, T. Neumann, and S. Gunnemann, "In-database machine learning with SQL on GPUs," in *Proceedings of the 33rd International Conference on Scientific and Statistical Database Management*, in SSDBM '21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 25–36. https://doi.org/10.1145/3468791.3468840.
[20] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN COMPUT. SCI.*, vol. 2, no. 6, p. 420, 2021, https://doi.org/10.1007/s42979-021-00815-1.
[21] K. K. Chandriah and R. V. Naraganahalli, "RNN / LSTM with modified Adam optimizer in deep learning approach for automobile spare parts demand forecasting," *Multimed Tools Appl*, vol. 80, no. 17, pp. 26145–26159, 2021, https://doi.org/10.1007/s11042-021-10913-0.
[22] Z. Li and N. Zhang, "Short-term demand forecast of e-commerce platform based on ConvLSTM network," *Computational Intelligence and Neuroscience*, vol. 2022, no. 1, p. 5227829, 2022, https://doi.org/10.1155/2022/5227829.
[23] F. Taha, D. Farzaneh, B. Patrick, and U. Chibuzor, "Predictive analytics for demand forecasting – A comparison of SARIMA and LSTM in retail SCM," *Procedia Computer Science*, vol. 200, no. 2022, pp. 993–1003, 2022, https://doi.org/10.1016/j.procs.2022.01.298.
[24] X. Zhang, P. Li, X. Han, Y. Yang, and Y. Cui, "Enhancing time series product demand forecasting with hybrid attention-based deep learning models," *IEEE Access*, vol. 12, pp. 190079–190091, 2024, https://doi.org/10.1109/ACCESS.2024.3516697.
[25] C. Chatfield, *Time-Series Forecasting (1st ed.)*, 1st ed. Chapman and Hall/CRC, 2000. https://doi.org/10.1201/9781420036206.
[26] P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, vol. 50, no. 17, pp. 159–175, 2003, https://doi.org/10.1016/S0925-2312(01)00702-0.
[27] J. Robin and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. OTexts, 2018. [Online]. Available at: https://otexts.org/fpp2/.
[28] B. Nguyen-Thai, V. Le, N.-D. T. Tieu, T. Tran, S. Venkatesh, and N. Ramzan, "Learning evolving relations for multivariate time series forecasting," *Appl Intell*, vol. 54, no. 5, pp. 3918–3932, 2024, https://doi.org/10.1007/s10489-023-05220-0.
[29] Q. Zhao, G. Yang, K. Zhao, J. Yin, W. Rao, and L. Chen, "Multivariate time-series forecasting model: Predictability analysis and empirical study," *IEEE Trans. Big Data*, vol. 9, no. 6, pp. 1536–1548, 2023, https://doi.org/10.1109/TBDATA.2023.3288693.
[30] Daqing Chen, "Online Retail II." UCI Machine Learning Repository, 2012. https://doi.org/10.24432/C5CG6D.
[31] K. Dahdouh, A. Dakkak, L. Oughdir, and A. Ibriz, "Association rules mining method of big data for e-learning recommendation engine," *Proceedings of the Advanced Intelligent Systems for Sustainable Development (AI2SD'2018)*, vol. 915, M. Ezziyyani, Ed., in Advances in Intelligent Systems and Computing, vol. 915., Cham: Springer International Publishing, 2019, pp. 477–491. https://doi.org/10.1007/978-3-030-11928-7_43.
[32] Z. Zhao, Z. Jian, G. S. Gaba, R. Alroobaea, M. Masud, and S. Rubaiee, "An improved association rule mining algorithm for large data," *Journal of Intelligent Systems*, vol. 30, no. 1, pp. 750–762, 2021, https://doi.org/10.1515/jisys-2020-0121.
[33] H. Lan, X. Ma, L. Ma, and W. Qiao, "Pattern investigation of total loss maritime accidents based on association rule mining," *Reliability Engineering & System Safety*, vol. 229, p. 108893, 2023, https://doi.org/10.1016/j.ress.2022.108893.
[34] X. Wang, X. Huang, Y. Zhang, X. Pan, and K. Sheng, "A data-driven approach based on historical hazard records for supporting risk analysis in complex workplaces," *Mathematical Problems in Engineering*, vol. 2021, pp. 1–15, 2021, https://doi.org/10.1155/2021/3628156.
[35] J. R. Dias, *jeffrichardchemistry/pyECLAT: pyECLAT*. (Sep. 08, 2020). Zenodo. https://doi.org/10.5281/ZENODO.4019037.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Jan. 29, 2017, *arXiv*: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.
[37] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, https://doi.org/10.1038/s41586-020-2649-2.
[38] The pandas development team, *pandas-dev/pandas: Pandas*. (Apr. 10, 2024). Zenodo. https://doi.org/10.5281/ZENODO.3509134.
[39] TensorFlow Developers, *TensorFlow*. (Jun. 18, 2024). Zenodo. https://doi.org/10.5281/ZENODO.4724125.

PANDEGA ABYAN ZUMARSYAH received his Bachelor in Electrical Engineering and Master's in Information Technology from Universitas Gadjah Mada (UGM), Indonesia. Currently, he is pursuing his PhD in Electrical Engineering from UGM. His current research focuses on medical image processing, while his fields of interest include web development, signal processing, data analytics, and machine learning. Email: pandegaabyanzumarsyah@mail.ugm.ac.id

FARIZA EKA AULIA currently pursuing master's degree in information technology at Department of Electrical and Information Engineering Universitas Gadjah Mada Yogyakarta, Indonesia. Main research interests include Data Mining and Machine Learning. Email: farizaekaaulia@mail.ugm.ac.id