Journal of Artificial Intelligence and Capsule Networks (ISSN: 2582-2012)
www.irojournals.com/aicn/

An Optimized Machine Learning Based Rule Engine Architecture

Umesh Bhandari
SAP Labs India Pvt. Ltd., India
E-mail: umesh.bhandari@gmail.com

Abstract

Rules are segments of knowledge which are generally conveyed as, “when some conditions are evaluated as true, then perform some tasks". A rule engine is basically an advanced software system responsible for rules evaluation and execution. While it is easy to add rules, problems arise when their numbers tend to explode exponentially over time due to new business scenario needs. It eventually becomes more complex when an enterprise information system, having configuration model powered by a declarative rule engine, needs to be maintained. Performance significantly degrades when there are thousands of rules, since the engine must figure out every time which rule should be fired when large number of facts arrive for processing, thus rapidly choking up the system. Objective of this paper is to use machine learning techniques to optimize declarative business rules system when the number of rules increases creating performance degradation and complexity issues.

Keywords: Rule engine, rete algorithm, machine learning

1. Introduction

Organizations employ business processes to achieve, align and optimize their business goals. These business processes are implemented as business rules. These rules are implemented inside a Business Rule Management System (BRMS) to define, maintain, and execute business decision logic independent of tight coupling with actual code. This provides complex businesses flexibility to adapt and update their business policies and rules with dynamically changing business requirements.

Knowledge representation is domain related with representation, expression, and processing of knowledge. They are used by BRMS to map knowledge into a knowledge repository and against which data can be processed to derive business decisions.

DOI: https://doi.org/10.36548/jaicn.2022.4.006
Received: 08.11.2022, received in revised form: 07.12.2022, accepted: 23.12.2022, published: 03.01.2023

Rule engine is an integral unit of BRMS and is responsible for execution of business rules. It provides the capability to define, validate, maintain business rules and inter-rules relation management as well. Rule engine systems are generally declarative in nature and consists of an inference engine responsible for processing of rules and facts at a very high scale.

Machine Learning (ML) consists of how computers use different algorithms to carry out certain tasks and improve over the period by learning from data. Machine learning provides different algorithms and techniques to process data and provide meaningful information e.g., clustering of data, identifying patterns and predictive analysis.

Over the years, different research has evolved and improved constraint programming-based engines instead of rule engines [9][10][11][12] to solve rule-based business problems. This research paper proposes an idea of optimizing rule engines using machine learning techniques. It addresses the main limitations in rule engines based on the Rete [1] algorithm and presents an optimized rule engine framework that enables much faster rule processing.

The proposal introduces the following key aspects.

- Use Machine Learning -based clustering algorithm to create rule groups to improve rule load time.
- Use Machine Learning -based recommendation model for ordering of rules to speed up their execution.

2. Rule Engine Architecture

BRMS consists of a rule repository in which rules are stored. A standard rule engine has three major components [2]:

- Production Memory (PM)
- Working Memory (WM)
- Inference Engine

Fig. 1. shows the standard rule engine architecture which consists of all the above components along with rule repository and rule evaluation queue.

Rule engine processing starts by loading the rules from repository to PM for evaluation. From PM, these rules are pushed into an evaluation queue for processing to the

inference engine. Input from users or business processes called facts which contain actual scenario event are handled by WM which forwards them for evaluation against the rules

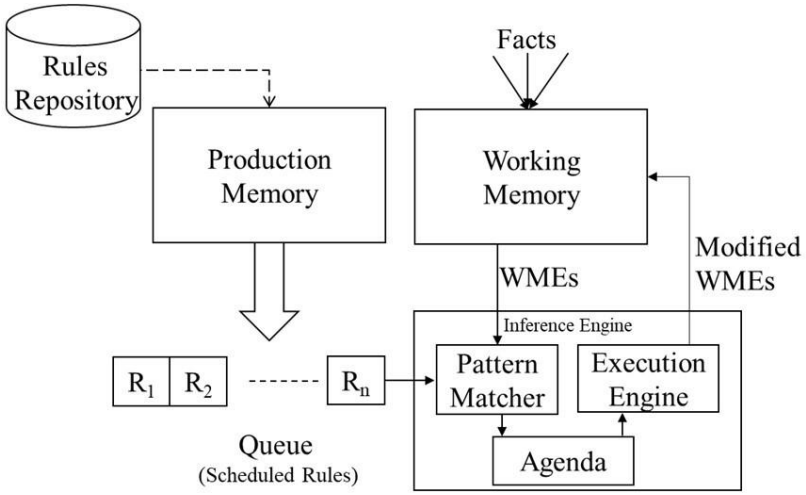

Figure 1. Rule Engine Architecture

The inference engine has three components:

i) Pattern Matcher ii) Agenda iii) Execution Engine

The rule engine performs the following sequence of events cycle.

- Initially input facts from users are loaded into WM as Working Memory Elements (WME).
- These WMEs are matched against the rules of PM in pattern matcher.
- In pattern matcher, rules for which condition parts match with the sub-elements of current WME are instantiated and added into the agenda.
- Agenda chooses an instantiation in case of multiple instantiations (conflict set) using conflict resolving mechanism.
- The execution engine then executes the action part of chosen instantiation.
- The above steps of inference cycle will be repeated until no rule instance is left for processing in agenda.

The rules evaluation processing with a sample use case has been focused and discussed. Below example shows the collection containing sample of rules demonstrating shopping offers on an e-commerce website. The action of the rule is performed when the evaluation of the rule condition is true.

Example: The following rules are considered.

Rule 1: If it is Christmas time, Sony gives special discount of 20% on their products.

Rule 2: On launch of a new camera, provide premium sale offer for prime members.

Rule 3: On arrival of summer season, announce discount offers on ACs.

Rule 4: On buying gadgets of price greater than Rs 35000 from Nikon, offer free prime membership to customer.

Rule 5: On buying camera of price greater than Rs 20000 from Nikon, offer free insurance to buyer.

Rule 6: Sony offers discount on fancy camera bag on purchase of camera.

It's assumed that the below Facts (F) have come for processing.

F1: Buyer: Bob,

F2: Bob buys a camera,

F3: Seller: Nikon,

F4: Price should be up to Rs 30000.

In this example of rule processing, all rules will be processed one by one sequentially and depending on the evaluation against the coming facts, they would return either true or false. The facts containing info like 'Company: Nikon', 'Buy a camera' and 'Price should be up to Rs 30000' match with the condition of Rule 5, so it would be matched as true, whereas the rest will be false.

So, pattern matching is a core functionality of a rule engine. Rete is a very popular and efficient pattern matching algorithm. There are other algorithms as well which perform better in certain scenarios [3] but Rete remains the preferred algorithm overall.

Even though Rete is very popular, it has few shortcomings as well. Rete requires that all rules are loaded in production memory even before the pattern matching starts, due to which considerable amount of memory is required for large number of rules [4]. This large number of rule loading also increases rule load time [5]. This loading of rule sets results in scalability issues in rule engines based on Rete [6]. As the number of rules grows, it also impacts the rule matching time with facts, as each fact needs to be matched which eventually

impacts its suitability for large-scale environments. As the number of facts entering Rete network grows, the propagation becomes slower in Rete tree [2]. The proposed architecture tries to address the above set of problems in Rete.

In the example, rules 1, 3, and 6 are not required to be loaded for processing, since there is no matching fact corresponding to them i.e., neither it is Christmas time nor summers and customer does not plan to buy Sony products. This forms the basis of the first proposal that only required rules for processing should be loaded. If the rules can be divided into groups based on certain criteria and process, i.e., only certain rules instead of all, then it will significantly reduce the rule load and processing time as well.

The second proposal of this paper is the machine learning based recommendation model for the ordering of rules. The idea is to process rules first which would result in reducing the complete rule execution time; considering sooner a rule starts, faster the corresponding actions will be completed, or further rules may be triggered as a part of its action chain. The recommendation model will try to recommend next rules to be processed which has high probability of matching against the fact. This is what the ML -based rule recommendation feature plans to target.

3. Proposed Architecture

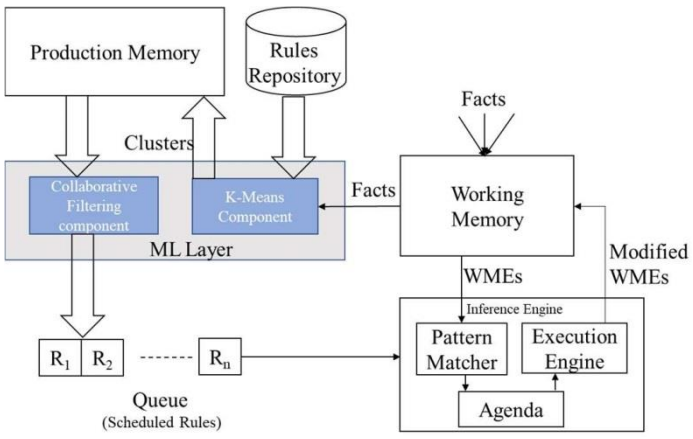

Figure 2. Proposed Rule Engine Architecture

Fig. 2 depicts the proposed modified rule engine architecture. A new machine learning layer is introduced in the rule engine to modify its processing.

Following are the main points of machine learning based proposal:

- Use K-means algorithm to create rule groups to load only required ones to expedite their loading in memory.

- Use collaborative filtering-based recommendation model for the ordering of rules to speed up their execution

3.1 Proposed architecture workflow

The proposed rule engine performs the following sequence of events cycle.

- Instead of initially loading rules from repository to PM, the rules are sent to K-means component before being loaded to the PM.
- Facts from WM are passed to K-means component as well for processing. The K-means component uses rules from repository and facts to form rule clusters, and these clusters are then passed to the PM.
- The rules from PM are processed in collaborative filtering component which re-orders rules as per the recommendation before moving to evaluation queue.
- Rest of the processing steps of inference engine i.e., pattern matching and agenda execution, remain as existing until no rule instance is left for processing in agenda.

There is a strong correlation between the two proposed techniques which is explained below. The K-means based grouping technique empowers the rule recommendation model. The following two sections contains the detailed explanation of the proposal.

3.2 Machine learning-based groups

Machine Learning (ML) provides many algorithms for clustering, prediction, and analysis of data. This section focuses on machine learning -based clustering algorithm to implement the proposed solution. Clustering is a method of unsupervised learning and is a common technique for statistical data analysis used in many fields. ML Clustering is a technique for grouping input data points based on certain features or attributes. Ideally, data points that are in the same cluster should have similar properties and/or features, whereas data points in different groups should have highly dissimilar properties and/or features. K-means clustering is very simple, fast, and popular unsupervised clustering algorithm. Its simplicity makes it easy to understand and implement as most of the ML libraries provide direct support for the same.

In this process of creating clusters of rules, decision should also be taken based on what features/attributes of a rule the clusters would be created. The rules are grouped in a cluster, based on how similar they are to each other. The data point information available in a rule is basically the fact variable in the condition which is used to identify similarity between

rules. A cluster will contain rules having identical fact in condition part of the rule and each rule will be assigned to only a single cluster. Assignment of rule to a unique cluster group ensures that when a matching fact comes, only rules of that cluster are loaded in PM and affected.

Let's consider there is a collection of rules (R1, R2……Rn) and a collection of predicates (p1, p2……. pn) used to construct its condition part. The basic idea is to scan all rules and identify all fact variables from the condition part of the rules and then run k-means algorithm on these predicates as features, and form clusters of rules based on it.

Considering the 6 rules shown in the example, the following steps are performed in this phase.

- Scan all rules and identify all fact variables. The possible fact variables are:
    1. Christmas time
    2. Purchase
    3. Sony
    4. Camera
    5. Nikon
    6. Summer
    7. Customer
    8. Price (worth)

- Most of the libraries providing K-means algorithm require numeric data points for processing; so, transformation of fact variables to numeric values is required. This is a general ML practice of cleaning, massaging, transforming data to make it most effective for algorithms for processing. For simplicity purpose, assignment of numeric values starting from 1 and so on to fact variables is done.

- After assigning numeric values, the example rules would contain fact variables in the form of numeric array, where each array value point to the corresponding fact variable as shown below.

    Rule1 - [1,3]
    Rule2 - [4,5]
    Rule3 - [6]
    Rule4 - [2,5,7,8]

Rule5 - [2,4,5,7,8]

Rule6 - [2,3,4]

- Now these array data points of rules are passed as input to K-means to create clusters. Here for the explanation purpose, cluster count K=3 is assumed.

- After processing, K-means provides clusters of rules as shown in the image below.

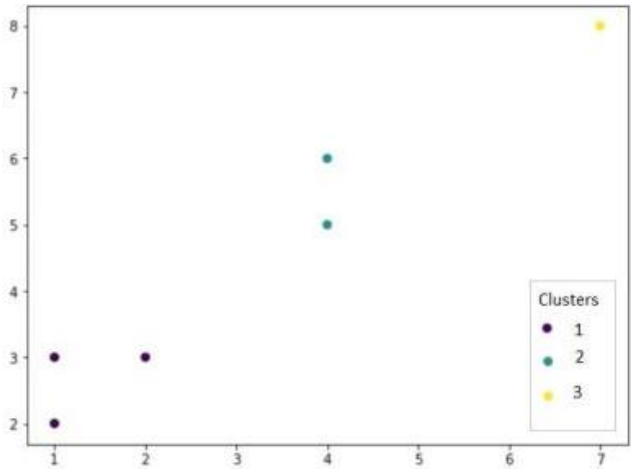

Figure 3. K-means rule clusters for the example

As can be seen, cluster 1 contains 3 rules [Rule 2, Rule 4, Rule 5], whereas cluster 2 contains 2 rules [Rule 1, Rule 6] and finally cluster 3 contains 1 rule [Rule 3]. This shows K-means as the effective and efficient way to rule clusters based on the similarity of fact variables provided. The only tricky part of using K-means clustering is choosing value for K initially as it is not known how many actual number of clusters would be formed beforehand. While K-means is very simple, fast, and effective, if one still wants to avoid specifying K initially, then there are other ML clustering algorithms available where it is not required e.g., Hierarchical clustering.

While defining a rule, an optional attribute <category> can be a part of fact. This category ID can be also used as a basis for defining clusters e.g., Use brands like Nikon, Sony as category. Although this method can be very helpful in forming a very clustering strategy, however besides category, there would be other overlapping fact variables as well. Therefore, it is suggested to use it along with the K-means algorithm.

3.3 Machine earning-based rule ordering recommendation

As mentioned earlier, the main objective for this proposed technique is to identify the order of rule evaluation. The goal is to utilize rule matching done using overlapping

predicates as the similarity metric. ML technique called ‘Item based collaborative filtering’ is used to identify this correlation score. ‘Item based collaborative filtering’ is used by many recommender systems e.g., If a user has purchased an item, then what should be recommended for the next purchase based on similarity. Clusters created earlier in clustering phase are used in this phase. Following are the steps performed in this phase:

- In each cluster, identify rule which has the highest number of predicates such as other rule members of cluster, and it is called as ‘alpha’ rule.
- For each cluster using the alpha rule, the correlation index score for other rules in same cluster is identified. ML technique called ‘Item based collaborative filtering’ is used to identify this correlation score.
    - For each rule, load info rows containing their predicate data points.
    - Create a matrix view for each rule predicate corresponding to every other rule predicate, where each matrix cell denotes whether the predicate is present or not in rule row or column.
    - Compute the pairwise correlation of predicate vector of alpha rule with every other rule. Many ML libraries provide direct function to calculate correlation score.
    - Order rules based on the correlation score with alpha rule.
- The alpha rule is the first rule in the queue for processing, from a cluster. For processing each cluster against the coming facts, the alpha rule is picked from it and evaluated first.
- If alpha rule is evaluated as true, the next rule from its cluster is picked which is next in order i.e., which has highest similarity and is put forward in evaluation queue, else next alpha rule from another cluster will be evaluated.

The idea behind this ML based rule recommendation is, if two rules have commonality and if one of them is evaluated as true, then the other has higher chances to be evaluated as true. So, it is recommended for earlier processing and moving ahead in the rule queue. Most ML based algorithm require their models to be trained first against past/training data to predict result for test data; but in this case, as training data is not available for rule engines, most of the ranking algorithms were not fit for this use case.

4. Results and Discussion

The proposed solution was implemented using JRuleEngine [7] which is a JSR-94 specification based open source java rule engine. Rules were created in xml format as per specification consumed by JRuleEngine. Deeplearning4j [8] a java-based ML library, was used for implementing K-means clustering and collaborative filtering.

A sample ruleset data consisting of 20 rules which were modelled from day-to-day purchase offers on ecommerce sites was generated. For analysis, browsing and shopping sample sessions on e-commerce websites were collected for few users. For same sessions, purchase recommendations given by Amazon and Flipkart were analyzed. After examining, 10 rules and 5 fact sets were derived from Amazon samples, and similarly from Flipkart samples, other 10 rules and 5 fact sets were derived. This practical approach of rule set generation gave a real-life and suitable use case to showcase the impact of proposal used for analysis.

These experimental data were used to evaluate the ML based proposal. Test system had Intel Core i5 processor and 16 GB RAM. The initial processing was required to generate clusters, and rule loading time was of millisecond range. The following table data shows the comparative analysis of rules loaded in memory in normal implementation compared to ML based clustering solution.

Table 1. Comparison of normal vs ML cluster-based rule loading

Number of rules loaded in memory										
Fact Group	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Normal loading	20	20	20	20	20	20	20	20	20	20
ML cluster-based loading	17	17	17	12	17	13	17	17	12	17

The result in Table I clearly shows that the ML clustering-based rule loading approach outperforms normal rule loading implementation as the number of rules loaded in memory is decreased resulting in lesser memory usage and faster processing. In the analysis, 20 rules were stored in the rule repository for processing by the rule engine. G1, G2 ….G10 depict different fact sets against which the rules are matched. As can be seen in normal loading of rules in Rete algorithm, 20 rules are loaded in production memory every time irrespective of whether eventually they match with the facts or not; while in proposed ML cluster-based loading, only the required matching rules i.e., ranging from 12 to 17 are loaded

in memory for processing thus reducing rules loaded in memory. Importantly, the outcome actions for both implementation with each fact sets are not changed, showing new approach can be used without any change in behavior of rule engine.

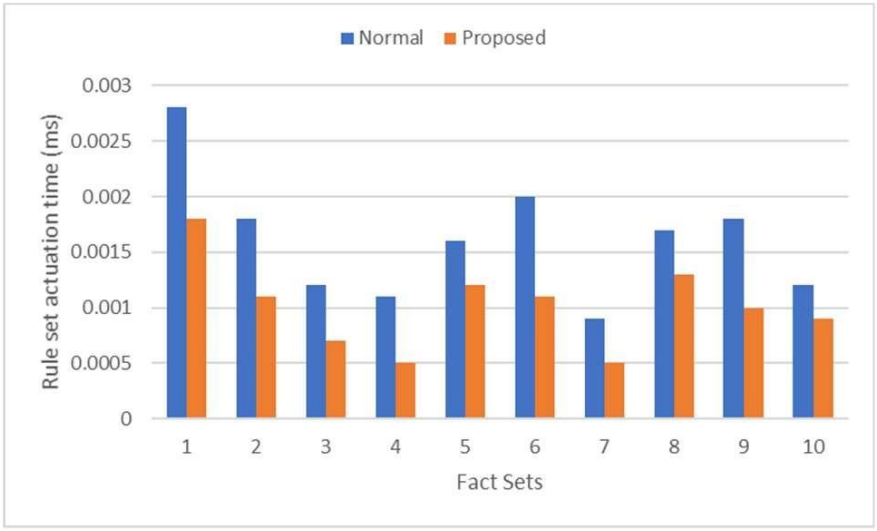

Figure 4. Comparison of actuation time for normal and proposed solution

The time from when evaluation of incoming user facts starts against the rules for matching, till the time when the last rule is evaluated as true is executed, is called the actuation time. Actuation time for normal and proposed solution is calculated and logged in the proposed solution built using JRuleEngine. Fig. 4 shows a comparison graph of the rule set actuation time of the normal implementation and the proposed ML-based clustering and recommender solution for each of the 10 fact sets. It can be observed that, using ML -based approach the rule actuation time is reduced by expediting the overall rule processing. The experimental results demonstrate the performance improvement making the proposal a promising solution for real-life use cases with large number of rule sets and continuously coming fact sets. So, it can be inferred that, over multiple iterations and large set of rules and facts, ML -based approach would outperform default implementation.

5. Conclusion

The performance of rule engine is critical for overall performance of an enterprise information system. Over the years, there have been many improvements of solving rule-based business problems using constraint programming- based engines instead of rule engines, but there has been hardly any discussion on improving rule engines by leveraging Machine Learning (ML) capabilities. In this paper, an ML -based technique is proposed to optimize the performance of a business rule engine. In the first part of approach, fact-based

ML clustering technique to load rules selectively is proposed. In the second part, ML -based prediction technique for ordering of rules for evaluation is proposed. The analysis of the experimental results shows that the proposed solution can be seamlessly integrated in a classical rule-based engine, and it would also open new avenues for further study in proposed direction. Considering the ML techniques used in today's technological world where plethora of frameworks and documentations are available, the proposed approach can be implemented and integrated in any Rete -based rule engine in a generic way.

References

[1] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artif. Intell., vol. 19, no. 1, pp. 17–37, Sep. 1982, doi: 10.1016/0004-3702(82)90020-0.

[2] H. Dong, J. Fan, and L. Zhang, "An improved rete algorithm based on double hash filter and node indexing for distributed rule engine," IEICE Trans. Inf. Syst., vol. E96-D, no. 12, pp. 2635–2644, 2013, doi: 10.1587/transinf.E96.D.2635.

[3] Y. W. Wang and E. N. Hanson, "A performance comparison of the Rete and TREAT algorithms for testing database rule conditions," Proc. - Int. Conf. Data Eng., pp. 88–97, 1992, doi: 10.1109/icde.1992.213202.

[4] F. Ongenae et al., "Towards computerizing intensive care sedation guidelines: Design of a rule-based architecture for automated execution of clinical guidelines," BMC Med. Inform. Decis. Mak., vol. 10, no. 1, 2010, doi: 10.1186/1472-6947-10-3.

[5] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A scalable rule engine architecture for service execution frameworks," Proc. - 2016 IEEE Int. Conf. Serv. Comput. SCC 2016, pp. 689–696, 2016, doi: 10.1109/SCC.2016.95.

[6] D. Liu, T. Gu, and J.-P. Xue, "Rule Engine based on improvement Rete algorithm," Dec. 2010, doi: 10.1109/icacia.2010.5709916.

[7] "JRuleEngine - OpenSource Java Rule Engine." http://jruleengine.sourceforge.net/ (accessed May 16, 2021).

[8] "Deeplearning4j." https://deeplearning4j.org/ (accessed May 16, 2021).

[9] J. Feldman, "Representing and Solving Rule-Based Decision Models with Constraint Solvers," in Rule-Based Modeling and Computing on the Semantic Web, Springer Berlin Heidelberg, 2011, pp. 208–221.

[10] R. C. Fernandes et al., "A rule-based system proposal to aid in the evaluation and decision-making in external beam radiation treatment planning," arXiv, pp. 1–20, 2018.

[11] T. Wang, "Machine Learning for Constraint Programming," 2019.

[12] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. A. Ciré, "Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization," CoRR, vol. abs/2006.0, 2020, [Online]. Available: https://arxiv.org/abs/2006.01610.