Report-Team Zoro

JUSTIFICATION

- The Apriori algorithm is a key component of Associative rule mining, a process used to identify relationships between data items.
- This algorithm is used to identify all of the items that are associated with a given target item. It then creates a rule that states that if a given item is present, the associated items are also likely to be present. This rule can then be used to predict the presence of the target item based on the presence of the associated items.
- One reason I went for apriori is, we can code it up pretty easily compared to the FP-tree approach.
- However, the apriori algorithm is known for its drawbacks, especially when the length of L1 is large.
- But in our case, the length of L1 is $\approx$ 200. Which is very low. So I went for it. Hope this justification is sufficient.

Explanation

```
def check_subsets(cur, prev_size):
    for i in range(len(cur)):
        s = cur[:i] + cur[i+1:]
        s= tuple(s)
        if s not in l[prev_size]:
            return False
    return True

def apriori_gen(l):
    c = []
    L = list(l.keys())

    try:
        L = [[i for i in l] for l in L]
        prev_size = len(list(l.keys())[0])
    for i in range(len(L)):
        for j in range(i+1, len(L)):
            if L[i][:-1] == L[j][:-1]:
                if(check_subsets(L[i] + [L[j][-1]], prev_size)):
                    c.append(sorted(L[i] + [L[j][-1]]))
    return c
#l[1] contains 1-length frequent patterns.
for i in range(2,10):
    c = apriori_gen(l[i-1])
    l.append({})
    for item in c:
        for user in train:
            if set(item).issubset(user):
```

```
                if tuple(item) in l[i]:
                    l[i][tuple(item)] += 1
                else:
                    l[i][tuple(item)] = 1
    l[i] = {k:v for k,v in l[i].items() if v > support_count}
    print("Len of l" + str(i) , len(l[i]))
```

- First, we generated, L1 (1-length frequent patterns) set by taking only those movies which satisfies the minimum support count.
- using that as our starting point, we generated 2-length candidate set C2 from L1
    - before adding elements into C2, we make sure that we are only taking in candidates for which all the subsets lies in L1
    - We also make sure that there are no repetitions by checking that all the items are sorted and and combined only when the condition that
        - members l_1 and l_2 of L_{k-1} are joined if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$.
        - The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated.
        - The resulting k-candidate set formed by joining l_1 and l_2 is { $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$}
- Once we get C2, L2 is generated by scanning the database to get the support counts and considered only those whose support counts are greater the minimum support count.
- Once we have items of all lengths, We generated association rules of the from $X \implies Y$ where X is a single movie and Y is the set of movies from the items that are generated.
- I calculated the supports and confidences using the formulas
    - $support(A \implies B) = P(A \cup B)$
    - $confidence(A \implies B) = \frac{supportCount(A \cup B)}{supportCount(A)}$
- Generated the required text files accordingly, you can see them in the directory itself.
- For task -3, for every user in the train set I considered k-rules for getting the recommendation_set for each movie he watched(in the train set)
- We took union of recommendation_sets for all movies he watched, to create a rec_set
- Hit_set is the intersection of this rec_set with the test_set movies of the same user
- Then calculated precison and recall for each user, and took average for each K to plot the graph like we did in task-3
- Later, We have taken some random 10 users to draw graphs in task-4.

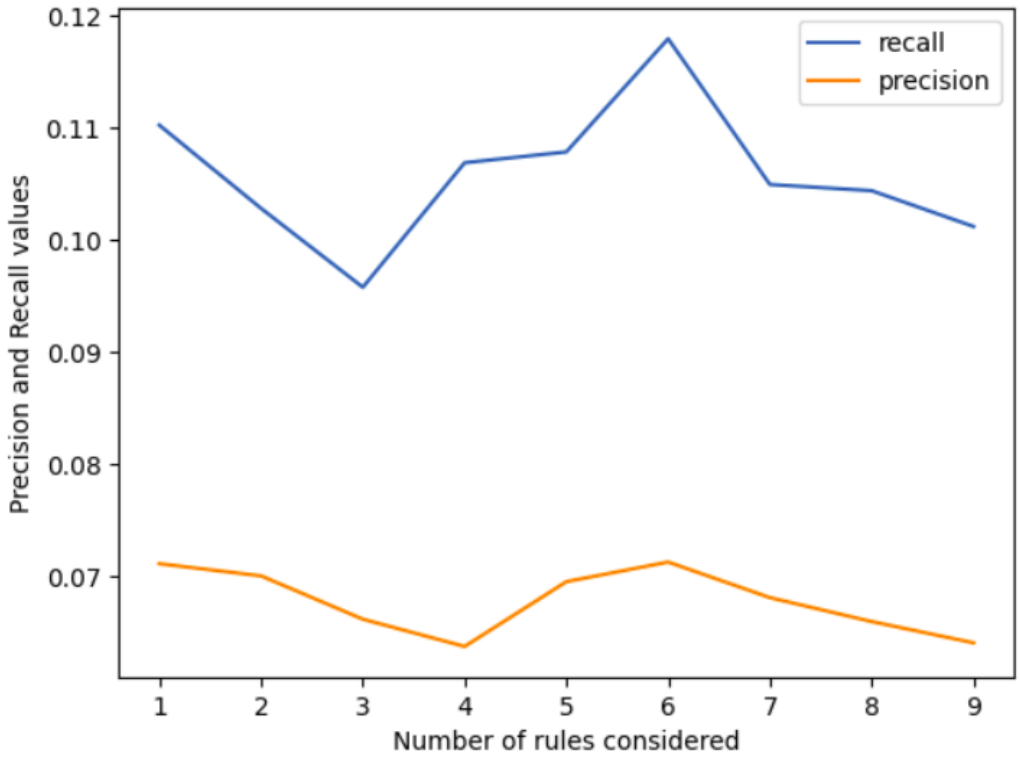
Optimizations

- We used efficient data structures such as sets, tuples and dictionaries.
- We used dictionary for each L_i so that it would be easy for us in the pruning steps.
- Whenever, there is a requirement to revisit the value, again and again, We precomputed the required values, so that there will be less number of scans required.
- I used sets for performing union and intersection operations, as inbuilt operators are well optimized.

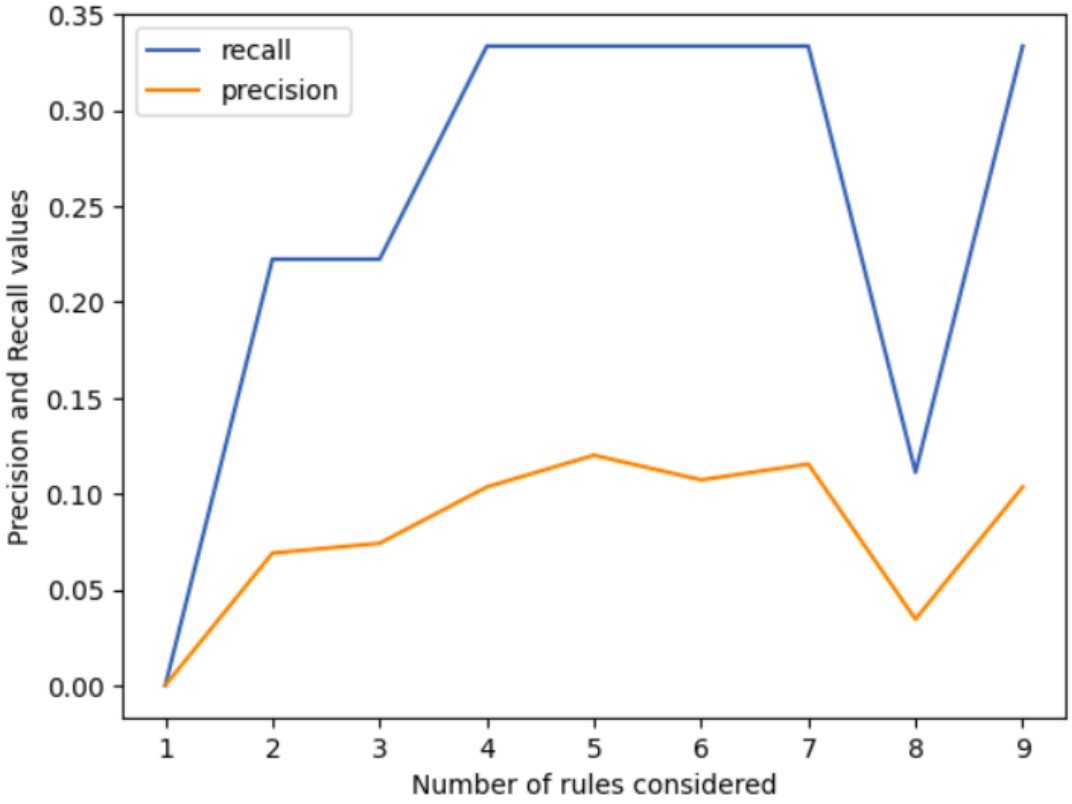
- I took min_support as 0.1 and min_confidence as 0.1

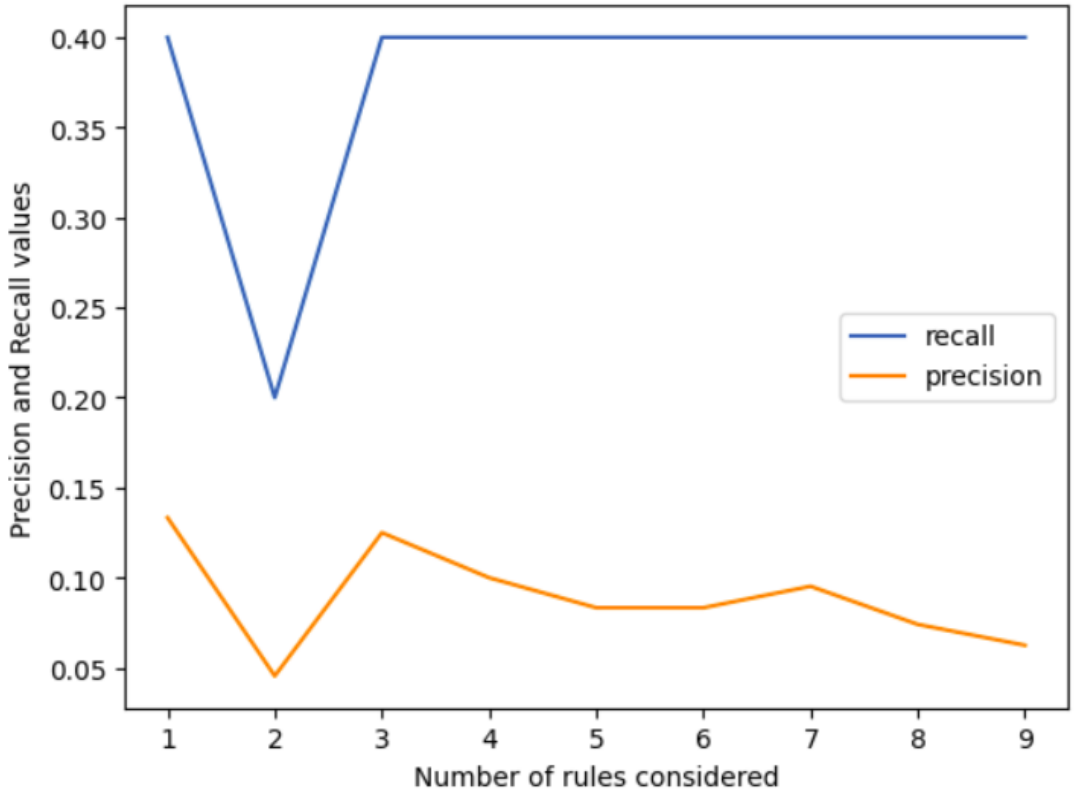
Task-3 (Average plot for all the users)

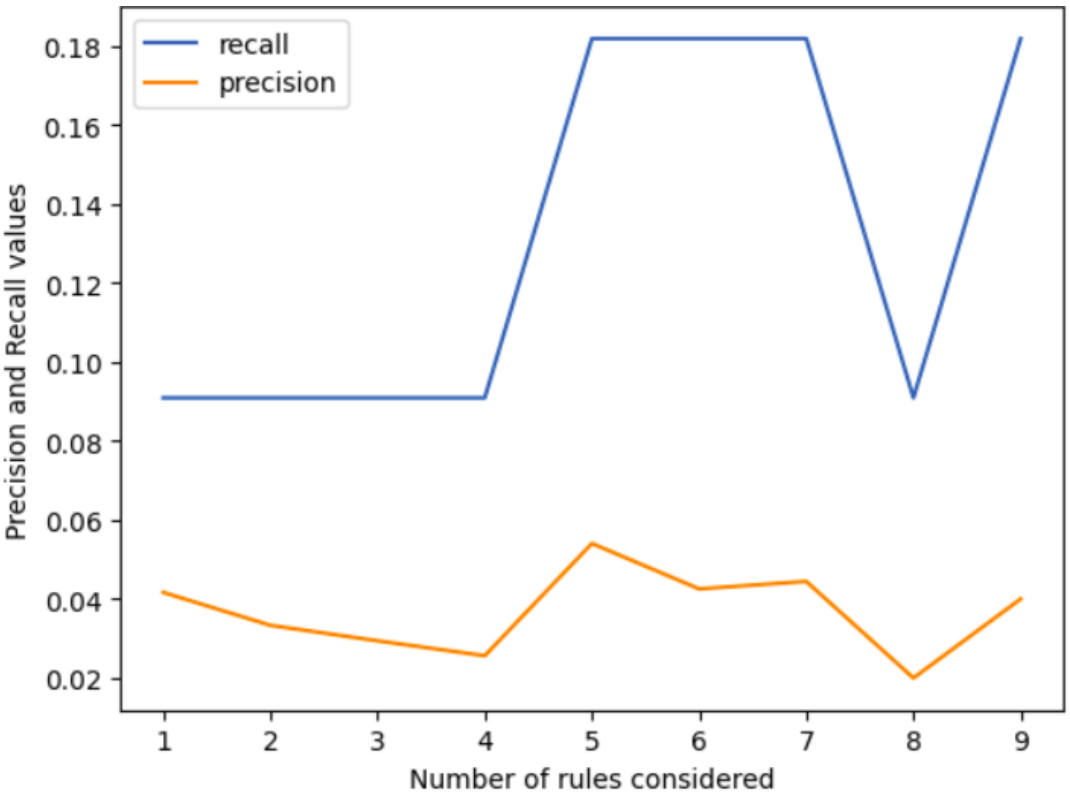
- All the explanation is provided above.

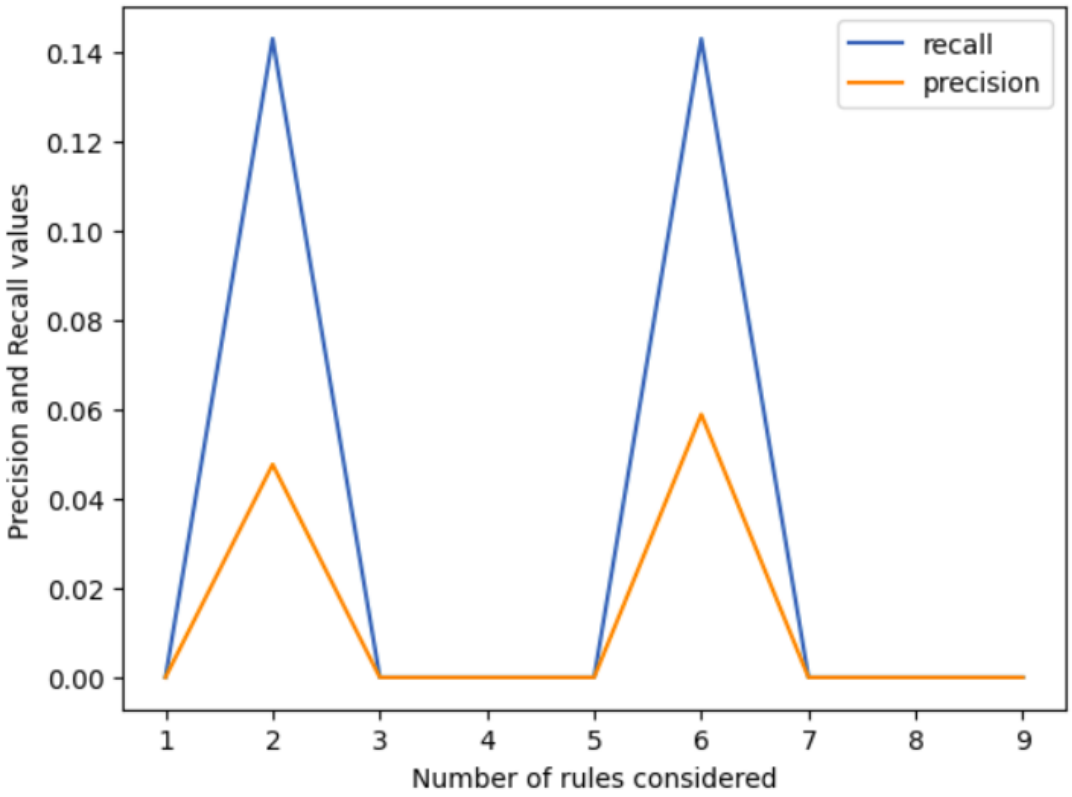


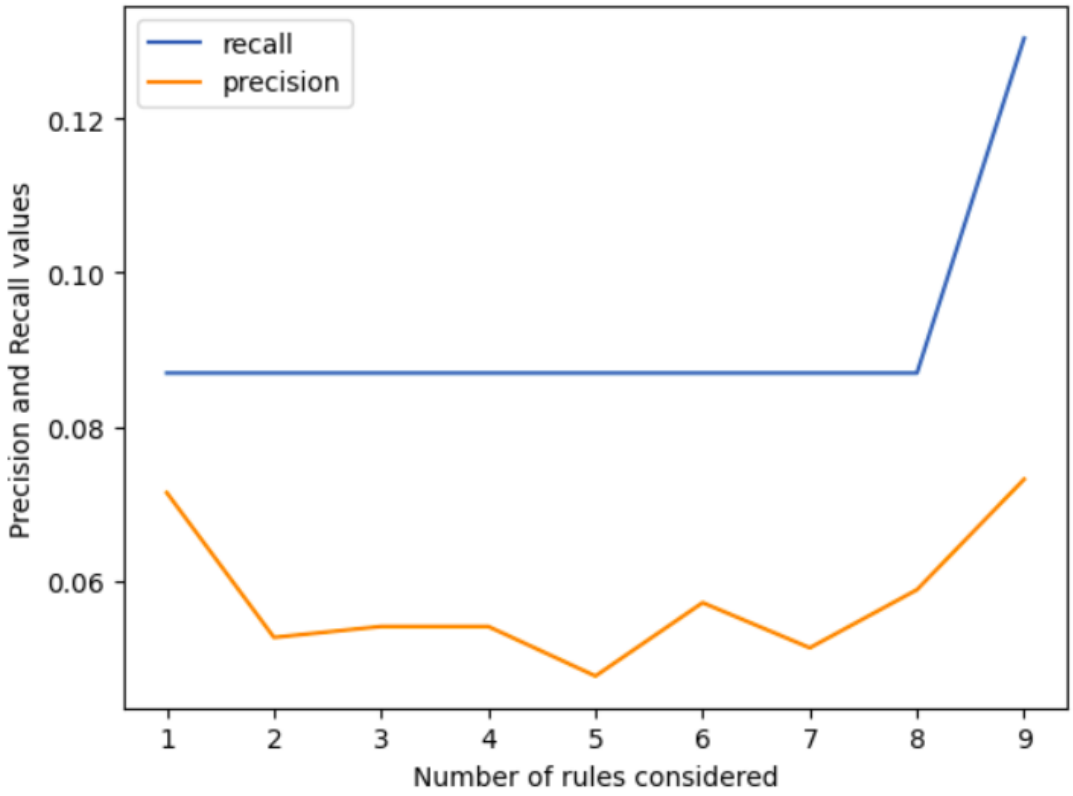
Task 4 (Plots for 10 random users)

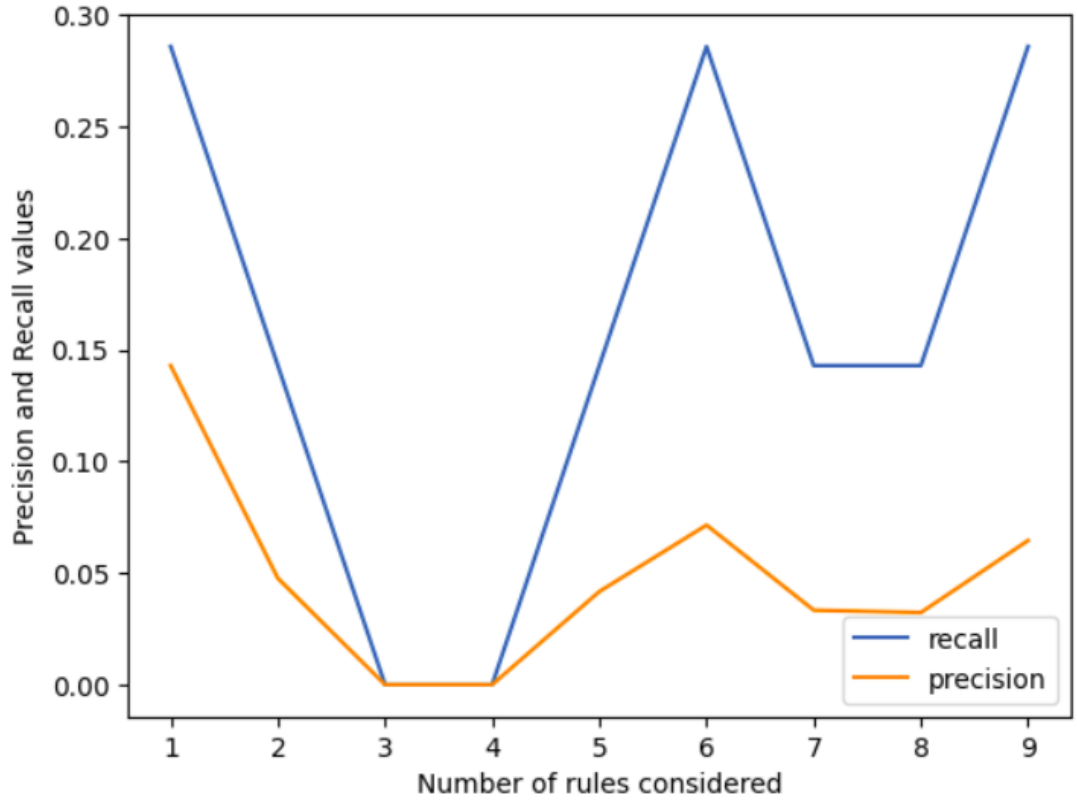


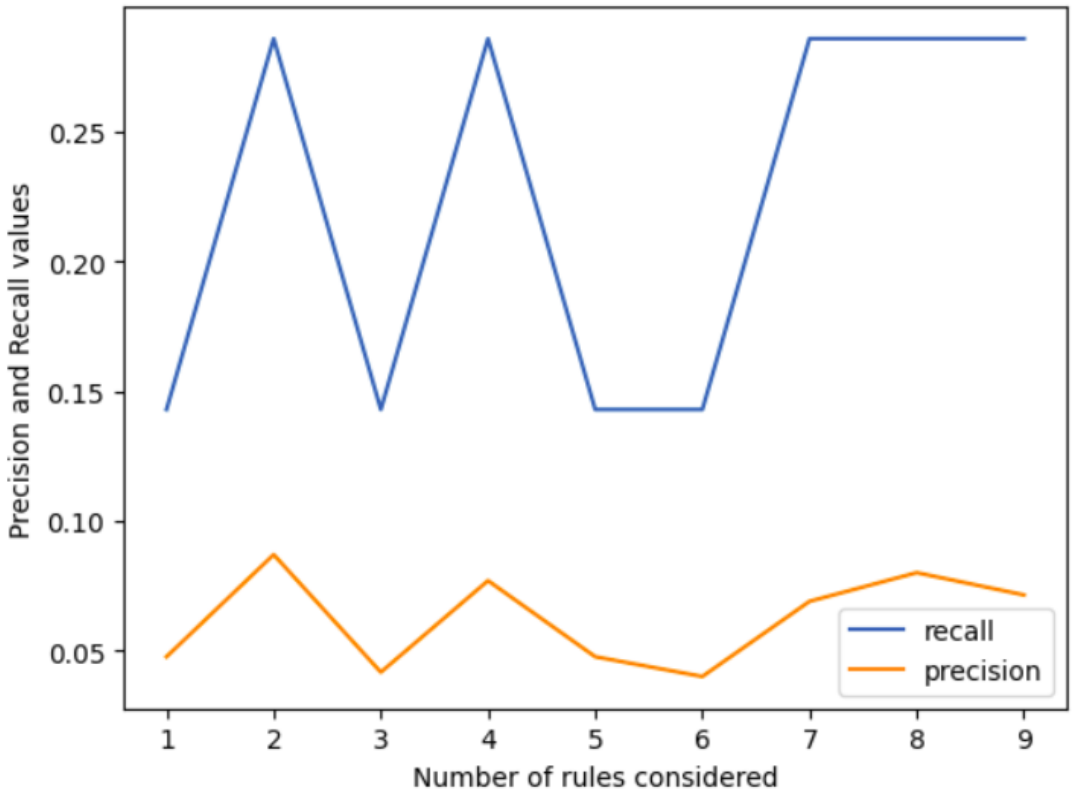

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Precision and Recall values
recall
precision
1
2
3
4
5
6
7
8
9
Number of rules considered

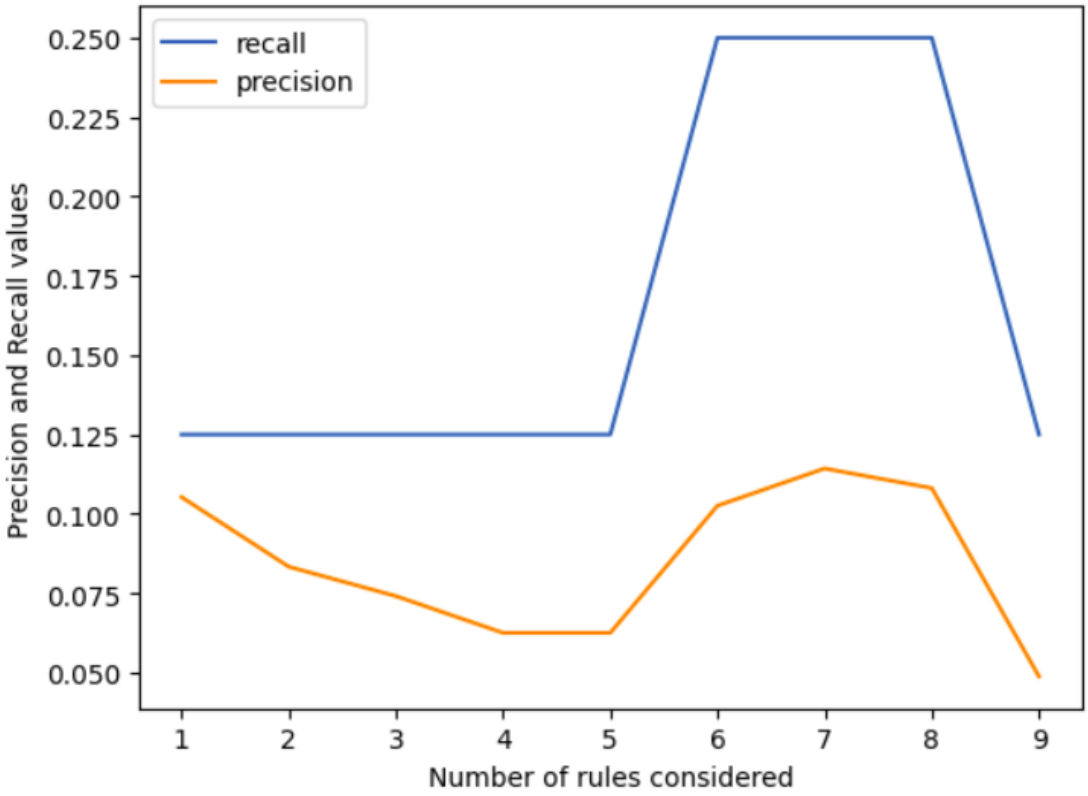

recall
precision
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
Precision and Recall values
1
2
3
4
5
6
7
8
9
Number of rules considered

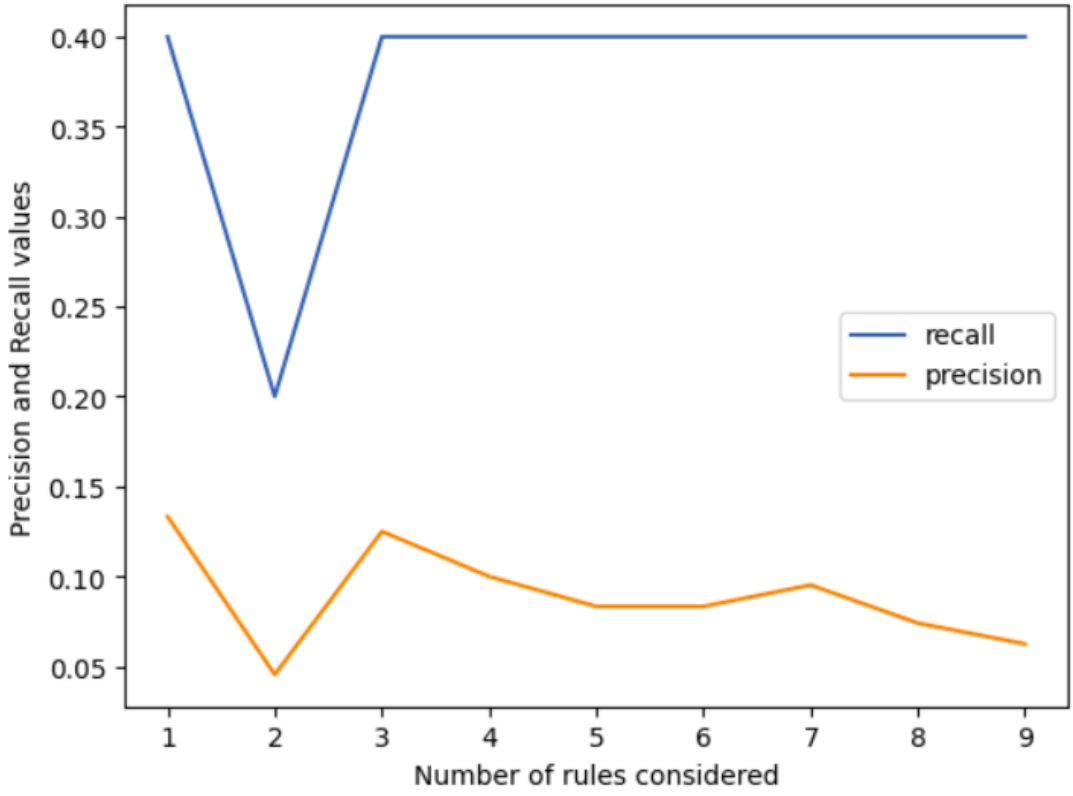

recall
precision
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Precision and Recall values
1
2
3
4
5
6
7
8
9
Number of rules considered

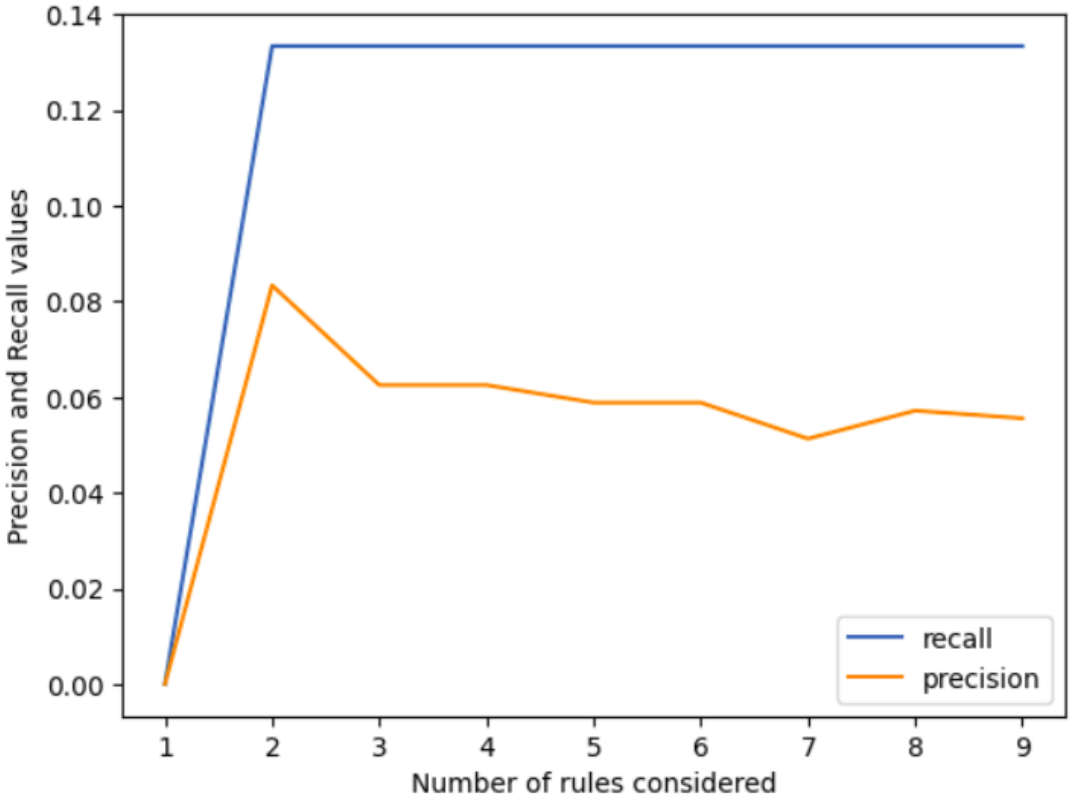

recall
precision
0.12
0.10
0.08
0.06
Precision and Recall values
1
2
3
4
5
6
7
8
9
Number of rules considered


0.30
0.25
0.20
0.15
0.10
0.05
0.00
Precision and Recall values
recall
precision
1
2
3
4
5
6
7
8
9
Number of rules considered


0.25
0.20
0.15
0.10
0.05
Precision and Recall values
recall
precision
1
2
3
4
5
6
7
8
9
Number of rules considered


recall
precision
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
Precision and Recall values
1
2
3
4
5
6
7
8
9
Number of rules considered


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
Precision and Recall values
recall
precision
1
2
3
4
5
6
7
8
9
Number of rules considered


0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Precision and Recall values
1
2
3
4
5
6
7
8
9
Number of rules considered
recall
precision