FOLD-R++ & FOLD-RM – Efficient Algorithms For Automated Inductive Learning

Firanol Abdisa[1][a], Joaquín Arias[1][b]
[1]*Research Centre for Artificial Intelligence and Information Technology (CETINIA), Rey Juan Carlos University, Mostoles, Spain*
[2] *Department of Artificial Intelligence, Rey Juan Carlos University, Mostoles, Spain*
{*Firanol Abdisa, Joaquín Arias*}*@urjc.es*

Keywords: Inductive learning, Answer set programming (ASP), Explainable models, Scalability, Model interpretability, multi-category classification.

Abstract: FOLD-RM and FOLD-R++ stand out as advanced automated inductive learning algorithms designed to extract default rules from heterogeneous data, encompassing both numerical and categorical variables. These algorithms produce transparent models through answer set programming (ASP) and normal logic program (NLP) rule sets for catagorical classification tasks. Through comparative evaluations, FOLD-RM and FOLD-R++ demonstrate competitiveness against leading algorithms such Random Forest. Noteworthy is their superior performance over Random Forest on specific datasets, particularly those of substantial size. Additionally, these algorithms prioritize model interpretability, providing easily understandable explanations for predictions, distinguishing them from conventional opaque models. These advancements position FOLD-RM and FOLD-R++ as invaluable tools for practitioners seeking effective and interpretable solutions in the realm of inductive learning applications. In this paper we will see how those two algorithms works to decide whether a certain student obtain a place or not in public university. In this case as our student dataset has only 128 rows, Random Forest performs with better accuracy but lack of transparency/explanation.

1 INTRODUCTION

The success of machine learning has led to a proliferation of Artificial Intelligence (AI) applications. However, these systems face a critical limitation in explaining decisions to human users, a challenge addressed by Explainable AI (XAI) [5]. The complexity of models generated by statistical machine learning methods makes it difficult for users to understand and verify underlying rules, further compounded by the inability to justify predictions for new data samples.

Inductive Logic Programming (ILP) [2] emerges as a solution to the model interpretability problem, involving the search for logic programming clauses deducing training examples. ILP algorithms, employing top-down or bottom-up approaches, find top-down methods better suited for large-scale or noisy datasets. Algorithms FOLD [3,4] tackle these challenges, and FOLD-R++ and the proposed FOLD-RM extend their capabilities to handle mixed features and enhance efficiency. FOLD-R++ introduces optimizations, including prefix sum computation, to improve efficiency and scalability. Both FOLD-R++ and FOLD-RM focus on explainable models in the form of logic programs, with FOLD-RM extending this capability to multi-category classification tasks.

Experimental results indicate comparable performance to traditional machine learning algorithms, FOLD-R++, FOLD-RM with superior execution efficiency. This paper addresses the interpretability gap in machine learning models, offering clear and efficient solutions for classification on students' dataset and decide whether that certain student obtain a place or not in a public university [1].

2 BACKGROUND

Inductive Logic Programming (ILP) [2], is a subset of machine learning dedicated to crafting human-understandable logic programming clauses. The central challenge involves formulating a background theory (B) as an extended logic program, encompassing positive and negative examples, and defining a hypothesis language equipped with a refinement operator. The objective is to derive a set of clauses ensuring consistency and accurate inference.

Default Logic, introduced by Reiter (1980), serves as a non-monotonic logic system designed for formalizing commonsense reasoning. This concept seamlessly integrates into normal logic programs, particularly in the context of default rules.

[a] https://orcid.org/0000-0000-0000-0000
[b] https://orcid.org/0000-0000-0000-0000

FOLD-R and FOLD-R++, specializing in ILP, are adept at learning default rules represented as answer set programs. Currently, these programs take the form of stratified normal logic programs, offering advantages in distinguishing exceptions and noise. While the current emphasis is on stratified programs, there is an envisioned extension to non-stratified answer set programs.

Both ILP[2] and default rules are pivotal in tackling classification problems, spanning binary and multi-category scenarios. Binary classification involves categorizing elements into two groups based on a defined rule, as seen in determining a patient's disease status. Multi-category classification expands this scope, classifying instances into three or more classes, such as predicting an animal's habitat. The algorithms prioritize efficiency and interpretability, yielding models that are not only accurate but also easily interpretable by humans.

3 THE FOLD-R++ ALGORITHM

FOLD-R++ stands out as a notable advancement over the FOLD-R algorithm, introducing three pivotal enhancements. Firstly, it expands the scope by allowing negated literals not just in the exception part but also in the default (positive) segment of the rule, adding depth to the representation. Secondly, the incorporation of the prefix sum algorithm accelerates computational processes, boosting overall efficiency. Thirdly, the introduction of the ”ratio” hyperparameter provides users with the flexibility to regulate the nesting level of exceptions, adding a nuanced control mechanism.

Presented Algorithm 2, the FOLD-R++ algorithm produces a set of default rules encoded as a normal logic program. The learning process involves selecting the optimal literal based on information gain, updating the rule set, and covering positive examples. The algorithm iteratively refines its rules by considering examples covered by the learned default literals. The learning cycle concludes when information gain hits zero or the number of negative examples falls below the defined ratio threshold. Subsequently, FOLD-R++ proceeds to learn exceptions by recursively swapping positive and negative examples.

The ratio parameter within Algorithm 2 signifies the equilibrium between training examples involved in exceptions and those solely implied by the default conclusion part of the rule. This parameter empowers users to finely adjust the nesting level of exceptions.

One noteworthy advantage of FOLD-R++ is its capacity to reduce the number of positive examples covered by a rule, thereby enhancing interpretability by generating a more concise set of rules and literals. The introduction of the exception ratio parameter further hones the algorithm's efficiency. For instance, in the student dataset, setting the ratio parameter to 0.5 significantly trims down the number of generated rules from 28 to just 13, achieving this in a fraction of the time compared to the absence of the ratio parameter.

Moreover, while FOLD and FOLD-R limited the use of negated literals in default theories for aesthetic reasons, FOLD-R++ embraces their inclusion in the default part of generated rules, recognizing their potential optimal contribution to information gain. Although operating as a greedy algorithm, FOLD-R++ presents a notable improvement over its earlier versions.

3.1 Efficient Literal Selection

The process of selecting literals in Shakerin's FOLD-R [3,5] algorithm, encapsulated in the SPECIALIZE function, involves determining the best literal by weighing information gain for learning defaults, similar to the original FOLD algorithm[3,5]. FOLD-R exhaustively explores all possible splits for numeric features, selecting the literal with the highest information gain. In contrast, FOLD-R++ adopts a more efficient approach using prefix sums to compute information gain based on classification categories.

FOLD-R++ classifies features into two types: categorical and numerical. Categorical features, including numerical values, are treated uniformly, generating equality and inequality literals. For numerical features, FOLD-R++ attempts numeric interpretation, converting to categorical if needed, and produces additional numerical comparison literals. Mixed-type features are treated as numerical.

In FOLD-R++, information gain for a literal is calculated efficiently using the prefix sum technique. The simplified information gain function IG, employing true positive, false negative, true negative, and false positive counts, speeds up the calculation. This approach requires only one classification round for a feature, even with mixed types of values, contributing to the enhanced efficiency of the FOLD-R++ algorithm in literal selection.

Algorithm 1: FOLD-R++ Information Gain

Input: tp, fn, tn, fp
Output: Information Gain

```
1.  function IG(tp, fn, tn, fp)
2.  if fp + fn > tp + tn then
3.  return $\infty$
4.  end if
5.  return 1 / (tp + fp + tn + fn) *
(F(tp, fp)+F(fp, tp)+F(tn, fn)+F(fn, tn))
6.  end function
7.  function F(a, b)
8.  if a = 0 then
9.  return 0
10. end if
```

```
11. return a * log2(a / (a + b))
12. end function
```

In the context of student placement, the ”FOLD-R++ Information Gain” algorithm is like a tool that helps the decision-making process by calculating the information gain associated with asking a specific question (literal) about a student. Here’s a simplified explanation: The Information Gain Tool:

1. Evaluate Questions (Literals): Think of this as a tool that evaluates how valuable a particular question (literal) is in terms of gaining information about whether a student will obtain a place or not.

2. Input Counts of Examples: The tool takes as input the counts of different examples related to the question: true positives (tp), false negatives (fn), true negatives (tn), and false positives (fp). These counts represent how many students fall into different categories based on the question.

3. Calculate Information Gain: It uses these counts to calculate the information gain, which is a measure of how much uncertainty or entropy is reduced by asking the specific question. Higher information gain implies that the question is more useful in predicting student placement.

4. Avoid Impossible Scenarios: If the sum of false positives and false negatives is greater than the sum of true positives and true negatives, it means there might be an issue with the question. In such cases, the information gain is set to negative infinity to avoid misleading results.

5. Information Gain Formula: The information gain is computed using a formula that involves four terms (F function). These terms represent the contribution of true positives, false positives, true negatives, and false negatives to the overall information gain.

6. Information Gain Contribution: Each term in the formula represents how much each category contributes to reducing uncertainty. For instance, if a category has zero examples, it doesn’t contribute to information gain.

The tool provides a numerical value representing the information gain associated with asking a specific question. This value helps the algorithm decide which questions are more effective in predicting whether a student will obtain a place or not. Higher information gain indicates that a question provides more clarity and better contributes to the decision-making process.

Algorithm 2: FOLD-R++ Algorithm
Input:

- E^+: Positive examples
- E^-: Negative examples

Global Parameters:

- target
- B (background knowledge)
- ratio (exception ratio)

Output: $R = \{r_1, \ldots, r_n\}$: a set of default rules with exceptions

```
1.  function Fold rpp(E+, E-, Lused)
2.  R ← Ø
3.  while |E+| > 0 do
4.      r ← learn rule(E+, E-, Lused)
5.      Etp ← covers (r, E+)
6.      if |Etp| = 0 then
7.          break
8.      end if
9.      E+ ← E+ \ Etp
10.     R ← R  {r}
11. end while
12. return R
13. end function

14. function Learn rule(E+, E-, Lused)
15. L ← Ø
16. while true do
17.     l ← find best literal(E+, E-, Lused)
18.     L ← L  {l}
19.     r ← set default(r, L)
20.     E+ ← covers(r, E+)
21.     E- ← covers(r, E-)
22.     if l is invalid or |E-| ≤ $|E+| * ratio$ then
23.         if l is invalid then
24.             r ← set default(r, L \ {l})
25.         else
26.         AB ← fold rpp(E-, E+, Lused + L)
27.         r ← set exception(r, AB)
28.         end if
29.         break
30.     end if
31. end while
32. return r
33. end function
```

Function 1: `Fold++ (E+, E-, Lused)`

Step 1: Initialization

```
1.  function Fold rpp(E+, E-, Lused) e
2.      R ← Ø
```

Description: Define the `Fold rpp` function with positive examples `E+`, negative examples `E-`, and used literals `Lused`. Initialize an empty set `R` to store the learned rules.

Context (Student Placement):

- `E+`: Students who successfully got placed.
- `E-`: Students who didn’t get placed.
- `Lused`: Initially empty, it will track the literals used in the rules.
- `R`: Initially empty, the set of rules.

Step 2: Main Loop

```
3.  while |E+| > 0 do
```

Description: Start a loop that continues until there are no more positive examples.

Context (Student Placement): Continue the loop until there are positive examples (students who are not yet placed).

Step 3: Learn Rule

```
4.  r ← learn rule(E+, E-, Lused)
```

Description: Call the `learn rule` function to generate a rule (`r`) based on the current positive and negative examples and the used literals.

Step 4: True Positive Examples

```
5.  Etp ← covers(r, E+)
```

Description: Determine the set of true positive examples (`Etp`) covered by the learned rule (`r`).

Context (Student Placement): Determine the set of true positive examples (`Etp`) covered by the learned rule (`r`). These are students predicted to be successfully placed.

Step 5: Check True Positive Examples

```
6.  if |Etp| = 0 then
7.      break
8.  end if
```

Description: If there are no true positive examples covered by the rule, exit the loop.

Step 6: Update Positive Examples

```
9.  E+ ← E+ \ Etp
```

Description: Remove the true positive examples (`Etp`) from the remaining positive examples, as they are already successfully placed.

Step 7: Update Rule Set

```
10. R ← R  {r}
```

Description: Add the learned rule (`r`) to the rule set (`R`), representing a condition for student placement.

Step 8: End of Loop

```
11. end while
```

Description: End the loop when there are no more positive examples, indicating that rules have been learned for all students.

Step 9: Return Result

```
12. return R
13. end function
```

Description: Return the set of learned rules (`R`) that provide insights into the conditions under which students get successfully placed.

Function 2: `Learn rule(E+, E-, Lused)`

- **Step 1: Start with Empty Rules:** Imagine having an empty set of rules to predict if a student will obtain a place.
- **Step 2: Pick the Best Hints:** Think of the function going through hints or features of students, like large family size, minimum rent, and more. It chooses the most helpful hints one at a time, creating a set of rules (default literals) for predicting.
- **Step 3: Adjust and Update:** As it goes through each hint, the function adjusts its rules, learning from examples of students who did or didn't get a place.
- **Step 4: Check for Common Sense:** It checks if the chosen hints make sense. If something seems off, it corrects it by updating the rules.
- **Step 5: Special Cases:** If there are tricky cases where the normal rules don't fit perfectly, it handles them separately (exceptions).
- **Step 6: Repeat Until Clear:** It repeats this process, making rules, adjusting, and handling exceptions until it's confident in predicting if a student will obtain a place.
- **Step 7: Result:** Finally, it gives us a set of rules that, based on different hints, can predict if a student is likely to get a place or not. So, it's like a smart teacher adjusting their rules based on different students' situations to predict who will obtain a place.

Function 3: `Find best literal(E+, E-, Lused)`

- **Step 1: Imagine the Counselor:** Think of the function as a counselor trying to guide the teacher in selecting the most informative question (literal) to ask about a student.
- **Step 2: Explore Different Questions:** The counselor explores various questions (literals) related to students' characteristics, such as family size, enrollment centers, and more.
- **Step 3: Evaluate Information Gain:** For each question, the counselor evaluates how much new information the question provides about whether a student will obtain a place.
- **Step 4: Suggest the Best Question:** The counselor suggests the question (literal) that gives the most valuable information to the teacher in predicting student placement.
- **Step 5: Consider Previous Choices:** It also takes into account what questions (literals) the teacher has already asked (`Lused`) to avoid redundancy.
- **Step 6: Repeat for Each Feature:** The process is repeated for different features, helping the teacher systematically choose the best questions to build accurate rules.

- **Result:** The function provides the teacher with the most beneficial questions (literals) to include in the rules, ensuring efficient and effective learning.

Essentially, it assists the algorithm in selecting the most relevant and informative features to make precise predictions about student placement.

The FOLD-R++ algorithm iteratively learns rules to predict outcomes based on positive and negative examples, updating the rule set at each iteration. The loop continues until there are no more positive examples. The result is a set of rules that provide insights into the patterns within the dataset, with a focus on interpretability and effectiveness.

3.2 Explainability

In the context of student placement, explain ability plays a crucial role, particularly in scenarios such as determining eligibility for courses or recommending career paths. Like domains like loan or credit card approval, having transparent rules for decision-making is essential. Inductive logic programming, as exemplified by the FOLD-R++ algorithm, offers explicit rules to elucidate the process of generating predictions. FOLD-R++ is adept at efficiently justifying its predictions by producing normal logic programs. Notably, these logic programs align seamlessly with the s(CASP) goal-directed answer set programming system, enhancing the overall transparency and interpretability of the decision-making process in the realm of student placement

The "Student data" is a classical classification task that contains 128 records. We treat 60% of the data as training examples and 40% as testing examples. The task is to learn the whether a specific student obtain a place or not based on features such as large_family,renta_minima_insercion, sibling_enroll_center,same_education_district, come_non_bilingual,want_bilingual_section('2nd ESO'), b1_certificate, OBTAIN_PLACE.

FOLD-R++ generates the following program that contains only 7 rules. The rules looks as follows:

Generated Logic Program

```
1. obtain_place(X,'yes'):-
     renta_minima_insercion(X,N1),
     N1>0.0, not ab2(X), not ab3(X).

2. obtain_place(X,'yes'):-
     large_family(X,N0), N0>0.0,
     not ab4(X).

3. obtain_place(X,'yes'):-
     large_family(X,N0),N0=<0.0,
     renta_minima_insercion(X,N1),
     N1>0.0, N1=<1.0, sibling_enroll
     _center(X,N2),N2=<0.0,
     same_education_district(X,N3),
     N3=<1.0, come_non_bilingual(X,N4),
     N4=<1.0, want_bilingual_section
     ('2nd_eso\)(X,N5), N5=<0.0.

4. ab1(X):-
      renta_minima_insercion(X,N1),
      N1=<1.0, sibling_enroll_
      center(X,N2), N2=<1.0,
      same_education_district(X,N3),
      N3>0.0, N3=<1.0, come_non_
      bilingual(X,N4), N4=<1.0.

5. ab2(X) :-
      large_family(X,N0), N0=<0.0,
      sibling_enroll_center(X,N2), N2>0.0,
      come_non_bilingual(X,N4), N4>0.0,
      want_bilingual_section('2nd_eso\)
      (X,N5),N5=<0.0, not ab1(X).

6. ab3(X) :-
      large_family(X,N0), N0=<0.0,
      renta_minima_insercion(X,N1),
      N1=<1.0, sibling_enroll_
      center(X,N2), N2=<0.0,
      same_education_district(X,N3),
      N3>0.0, N3=<1.0,
      come_non_bilingual(X,N4), N4>0.0,
      N4=<1.0, want_bilingual_section
      ('2nd_eso\)(X,N5), N5=<0.0.
7. ab4(X) :-
      same_education_district(X,N3),
      N3>0.0, come_non_bilingual(X,N4),
      N4>0.0.
```

Evaluation Metrics

The above program achieves the following metrics:

- Accuracy: 0.8571
- Precision: 0.89
- Recall: 0.9167
- F1 Score: 0.8991

Example Query

Given a new data sample, the predicted answer for this data sample using the above logic program can be efficiently produced by the s(CASP) system [2].

Since s(CASP) is query-driven, an example query such as `?- obtain_place(X,'yes').` which checks whether a student obtains a place or not.

The s(CASP) system will generate the proof tree in English, i.e., in a more human understandable form [6]. The justification tree generated for the person with ID a specific student as shown below:

4 FOLD-RM ALGORITHM

The FOLD-RM algorithm, an extension derived from FOLD-R++, is specifically tailored for multi-category classification tasks and reducing number of rules, offering a generalized approach beyond the binary classification focus of FOLD-R++. Outlined in Algorithm 4, this algorithm iteratively learns rules for each category, aiming to create a rule set that effectively classifies examples based on the diverse categories present in the dataset. The learning process initiates by pinpointing a target literal representing the category with the highest occurrence in the existing training set (line 4). Subsequent steps involve splitting the training set into positive and negative examples based on this target literal (line 5) and learning a rule to cover the designated category using the learn rule function from the FOLD-R++ Algorithm (line 6). After covering the target category, the algorithm updates the training set by excluding already covered examples (line 11), and the rule head is adjusted to align with the target literal (line 12). It is noteworthy that FOLD-RM produces an ordered answer set program, where rules are sequentially evaluated.

In contrast to FOLD-R++, the output of FOLD-RM adopts an ordered answer set program format, ensuring that rules are checked sequentially, and if a rule applies, subsequent rules are not considered. The determination of the target predicate for each rule is based on the label value with the highest frequency among the remaining training examples. Predicates in both the rule body and head adhere to a structured format, each having two arguments. The first argument pertains to the data record, while the second argument either signifies the predicted label (for the target predicate) or extracts the corresponding feature value. Abnormal predicates, responsible for handling exceptional cases, are characterized by having only one argument—the data record itself.

Input: E: examples, B: background knowledge, ratio: exception ratio

Output: $R = \{r_1, \ldots, r_n\}$: a set of defaults rules with exceptions

```
1. function FOLD-RM(E)
2. R ← Ø
3. while |E| > 0 do
4. l ← MOST(E)  l:
5. E+, E− ← SPLIT BY LITERAL(E, l)
6. r ← LEARN RULE(E+, E−, Ø)
7. EFN ← covers(r, E+, false)
8. if |EFN| = |E+| then
9. break
10. end if
11. E ← E+  EFN
12. r ← add head(r, l)
13. R ← R  {r}
14. end while
15. return R
16. end function
17. function MOST(E)
18. for e  E do
19. count[label(e)] ← count[label(e)] + 1
20. end for
21. labelmost ← FIND MOST(count)
22. return literal(index(label), =, labelmost)
23. end function
24. function SPLIT BY LITERAL(E, l)
25. E+, E− ← Ø, Ø
26. for e  E do
27. if EVALUATE(e, l) is true then
28. E+ ← E+  {e}
29. Else
30. E− ← E−  {e}
31. end if
32. end for
33. return E+, E−
34. end function
```

1. Initialization:

Input: Examples (E), Background Knowledge (B), Exception Ratio (ratio)
Output: Set of default rules with exceptions (R)

2. Iterative Rule Learning:

Initialize an empty set of rules: $R = \emptyset$.
Repeat until all examples are covered:

a. Most Popular Target Literal (`MOST(E)`): Find the most popular target literal (label) among the examples.
   This is a label that occurs most frequently among the students.

b. Splitting Examples (`SPLIT BY LITERAL (E, l)`): Split examples into positive (E+) and negative (E-) based on the most popular literal.
   E+ contains examples where the most popular literal is true, and E- contains examples where it is false.

c. Learn Rule (`LEARN RULE (E+, E-, Ø)`): Use the positive (E+) and negative (E-) examples to learn a rule (r).
   The rule is learned iteratively, focusing on the most popular literal as the target.

d. Handling False Negatives: Check for false negatives (EFN) covered by the rule.
   If all examples are covered (no false negatives), exit the loop.

e. Update Examples (`E ← E+  EFN`) and Rule Set (`R ← R` {`r`}): Remove the already covered examples from the set of examples. Add the learned rule to the set of rules.

3. Output:

Return the set of learned rules (R).

The algorithm starts by identifying patterns in the dataset, focusing on the most popular target literal. This literal represents the label that occurs most frequently, giving importance to what most students have in common. It learns rules by iteratively splitting the examples based on the most popular literal and creating rules that differentiate between positive and negative cases. The algorithm handles false negatives efficiently. If there are cases not covered by the rule, it identifies and handles them separately.

The process is iterative, refining the rules with each iteration to become more accurate in predicting student placement. The output is a set of rules that, based on different criteria and features (represented by literals), can predict whether a student is likely to obtain a place or not.

FOLD-RM adapts its rules based on the most common features in the dataset, refines them through iterations, and provides a set of rules for predicting student placement. It is a flexible and adaptive algorithm that tailors its rules to the characteristics of the dataset.

The "Student data" is a classical classification task that contains 128 records. We treat 60% of the data as training examples and 40% as testing examples. The task is to learn the whether a specific student obtain a place or not based on features such as large family, renta minima insercion, sibling enroll center, same education district, come non bilingual, want bilingual section('2nd ESO'), b1 certificate,OBTAIN PLACE. FOLD-R++ generates the following program that contains only 6 rules:

```
1. obtain_place(X,'yes'):-
     renta_minima_insercion(X,N1),
     N1>0.0, not ab1(X).
2. obtain_place(X,'no'):-
     large_family(X,N0), N0=<0.0.
3. obtain_place(X,'yes'):-
     renta_minima_insercion(X,N1),
     N1=<0.0, not ab2(X).
4. obtain_place(X,'no'):-
     large_family(X,N0), N0=<1.0.
5. ab1(X):-
     large_family(X,N0), N0>0.0,
     sibling_enroll_center(X,N2),
     N2>0.0, come_non_bilingual(X,N4),
     N4>0.0, want_bilingual_section
     ('2nd_eso\)(X,N5), N5=<0.0.
6. ab2(X):-
     sibling_enroll_center(X,N2),
     N2=<0.0, want_bilingual_section
     ('2nd_eso\)(X,N5), N5=<0.0.
```

5 RANDOM FOREST

The algorithmic steps of a Random Forest classifier for student placement prediction can be outlined as follows:

1. **Load Dataset:** Import the required libraries, including pandas for data manipulation and scikit-learn for machine learning. Load the student dataset from a CSV file.
2. **Split Data:** Separate the dataset into features (X) and the target variable (y), where X represents the student characteristics, and y indicates whether a student obtains a place or not. Split the data into training and testing sets using the `train_test_split` function.
3. **Random Forest Classifier:** Create a Random Forest classifier with specified parameters (100 trees in this case) using the `RandomForestClassifier` class from scikit-learn. Train the classifier on the training set using the `fit` method.
4. **Make Predictions:** Use the trained classifier to make predictions on the test set with the `predict` method.
5. **Evaluate Performance:** Compute various performance metrics, including accuracy, precision, recall, and F1 score, using scikit-learn's metrics functions (`accuracy_score`, `precision_score`, `recall_score`, `f1_score`).
6. **Timing Information:** Measure and report the time taken for both training and making predictions (inference) using the `time` library. The training time and inference time provide insights into the computational efficiency of the model.
7. **Display Results:** Print the computed metrics and timing information to the console, providing a summary of the model's performance.

Random Forest is a robust ensemble learning algorithm widely used for predictions, excelling in handling complex datasets and minimizing overfitting. Its strength lies in capturing intricate relationships and handling various types of features. However, the ensemble nature can reduce interpretability, and computational costs may increase with a large number of trees. Striking a balance between accuracy and interpretability is crucial when considering Random Forest for prediction tasks.

6 EXPERIMENT

In our comprehensive experiment evaluating three distinct models—FOLD-R++, FOLD-RM, and Random Forest—for student placement prediction, each model demonstrated unique strengths and characteristics. FOLD-R++, implemented in Python, showcased enhanced efficiency over its predecessor, FOLD-R, leveraging prefix-sum for information gain computation. The algorithm provided interpretable rules, contributing to transparent decision-making for student placement. FOLD-RM, also implemented in Python, excelled in handling mixed-type data directly, eliminating the need for one-hot encoding. Its succinct and interpretable rules, along with impressive efficiency, marked it as a viable candidate for student placement predictions.

Random Forest, on the other hand, exhibited perfect accuracy with all metrics—precision, recall, and F1 score—reaching 1.00. This model, though less interpretable, demonstrated exceptional efficiency with training and inference times at 0.0364 seconds and 0.0018 seconds, respectively. While Random Forest may be ideal for scenarios demanding maximum accuracy, FOLD-R++ and FOLD-RM offered a balance between accuracy, interpretability, and efficiency.

FOLD-R++ Results:

Accuracy (85.61): FOLD-R++ achieved a high accuracy, indicating that the model made correct predictions for the majority of students in the dataset. Precision (89): The precision, which measures the accuracy of positive predictions, was also high, indicating that when FOLD-R++ predicted a student would obtain a place, it was accurate 89% of the time. Recall (91): The recall, representing the ability of the model to capture all relevant instances, was high. FOLD-R++ successfully identified 91 of students who obtained a place. F1 Score (90): The F1 score, which balances precision and recall, was strong at 90.

FOLD-RM Results: Accuracy (88.52): FOLD-RM achieved a good level of accuracy compared to FOLD-R++ and Random Forest. Precision (86.21): The macro precision, considering all classes, was 86.21, indicating the ability to make accurate positive predictions across different scenarios. Recall (87.52): The macro recall, considering all classes, was 87.52, showcasing FOLD-RM's ability to capture relevant instances. F1 Score (87.23): The macro F1 score, providing a balanced measure, was 87.23.

Random Forest Results: Accuracy (100): Random Forest achieved perfect accuracy, indicating that every student's placement was correctly predicted in the test set. Precision (10Recall (100): Random Forest captured all students who obtained a place, demonstrating perfect recall. F1 Score (100): The F1 score, balancing precision and recall, was ideal at 100

Choosing the appropriate model depends on the specific requirements of the application, with Random Forest providing high predictive accuracy and FOLD-R++ offering interpretability. FOLD-RM strikes a balance between simplicity and performance. The overall comparison highlighted that FOLD-R++ and FOLD-RM generated interpretable rules, providing insights into the decision-making process, whereas Random Forest excelled in achieving flawless accuracy. FOLD-R++ demonstrated efficiency improvements over its predecessor, and FOLD-RM.

Random Forest excelled in terms of accuracy, precision, recall, and F1 score. FOLD-R++ demonstrated strong interpretability and competitive performance. FOLD-RM showed good performance with a focus on simplicity and fewer rules.

Metrics	FOLD-R++	FOLD-RM	Random Forest
Accuracy	85.61%	88.52%	100%
Precision	89%	86.21%	100%
Recall	91%	87.52%	100%
F1 Score	90%	87.23%	100%
#Rules	7	6	-

Table 1: Comparison of Metrics

Random forest looks a good performer in our case because our student dataset is small and simple.

The choice among these models depends on specific priorities, such as accuracy, interpretability, or efficiency, catering to the nuanced needs of student placement scenarios. Further considerations for generalization and model optimization were suggested, emphasizing the significance of our experiment in aiding stakeholders in making informed choices for student placement decisions.

7 CONCLUSION

In the context of student placement, our study introduced two innovative algorithms, FOLD-R++ and FOLD-RM, tailored to address the specific challenges of predicting whether a student will obtain a place or not. FOLD-R++, designed for binary classification, excels in generating transparent and efficient models, showcasing comparable performance to popular classifiers like Random Forest while offering superior training efficiency. This algorithm emphasizes the importance of explainability providing clear insights into predictions through its normal logic program representation.

On the other hand, FOLD-RM, specialized for multi-category classification, proves to be scalable and efficient in scenarios where students may fall into different placement categories. Noteworthy is its ability to forego data encoding, simplifying the preparation process. FOLD-RM achieves competitive performance in terms of accuracy, precision, recall, and F1 score when compared to well-known models like Random Forest and MLP, standing out for its efficiency and interpretability.

In conclusion, these algorithms contribute significantly to student placement tasks, offering efficient and interpretable solutions for predicting whether a student will obtain a place or not. The advancements showcased in this study open avenues for the development of transparent machine learning models, providing valuable insights into the decision-making process related to student placements.

8 FUTURE WORK

Future work for the implementation of FOLD-R++ and FOLD-RM in student placement holds significant potential across various dimensions. Firstly, there is a critical need to optimize the scalability of these algorithms, exploring parallelization and distributed computing to ensure efficiency as student datasets expand. Fine-tuning hyperparameters and experimenting with additional or transformed features are essential steps for improving predictive accuracy, while comparative analyses with other machine learning models can offer valuable insights into the strengths and weaknesses of FOLD-R++ and FOLD-RM in educational contexts.

Moreover, future efforts should prioritize enhancing model interpretability, addressing imbalanced data challenges, and considering the practical deployment of models in educational settings. Collaboration with educational experts, the incorporation of temporal features, and exploration of adaptive models for evolving learning environments are key areas for further exploration. These endeavors aim to contribute to the continual improvement of transparent, effective, and ethically grounded machine learning models for informed student placement decisions.

9 REFERENCES

1. Joaquín Arias1, Mar Moreno-Rebato1, Jose A. Rodriguez-García1, and Sascha Ossowski: Modeling Administrative Discretion Using Goal-Directed Answer Set Programming
2. Muggleton, S.: Inductive logic programming. New Gen. Comput. 8(4) (Feb 1991)
3. Shakerin, F.: Logic Programming-based Approaches in Explainable AI and Natural Language Processing. Ph.D. thesis (2020), Department of Computer Science, The University of Texas at Dallas
4. Shakerin, F., Salazar, E., Gupta, G.: A new algorithm to automate inductive learning of default theories. TPLP 17(5-6), 1010–1026 (2017)
5. Gunning, D.: Explainable artificial intelligence (XAI) (2015), https://www.darpa.mil/program/explainable-artificial-intelligence
6. Arias, J., Carro, M., Chen, Z., Gupta, G.: Justifications for goal-directed constraint answer set programming. In: Proceedings 36th International Conference on Logic Programming (Technical Communications). EPTCS, vol. 325, pp. 59–72 (202