# Selecting the Foremost Rule of First Order Logic using FOIL Algorithm

**[1]Vinay. S, [2]Prof. B.I.Khodanpur**

[1]PG. Student, [2]Associate Professor
Information Science and Engineering,
Dayananda Sagar College of Engineering, Bangalore, India

*Abstract*: **INDUCTIVE LOGIC PROGRAMMING is a research area, which uses a both the Machine Learning and the inductive programming. ILP uses the background knowledge for learning. ILP can be implemented using the FOIL (First Order inductive Logic) algorithm. FOIL is an Inductive Logic Programming Algorithm to get first order rules to elucidate the patterns involved during a domain of data FOIL is an function free Horn clauses. Using the positive examples, negative examples, background information and the rules FOIL algorithm would calculate the foil gain for the respective rules. For a unique rule foil gain would be maximum when compared to other foil gains. Rule for which gain is maximum is considered to be a best rule.**

***Index Terms*: FOIL, FOIL_GAIN, First Order Logic**

## I. INTRODUCTION

Learning a general theory from specific examples, commonly called induction, has been a topic of inquiry for centuries. It is often seen as a main source of scientific knowledge. Suppose we a r given a large number of patients records from a hospital, consisting of properties of each patient, including symptoms and diseases. We want to some general rules concerning which symptoms indicate which diseases. Hospitals records provide example from which we can find clauses as to what those rules are. Consider measles, a virus disease. If every patient in the hospital who has a fever and has a redspots suffers from measles. We could infer the general rule.

RULE1: "If someone has a fever and redspot then he has measles"

RULE2: "If someone has a measles then he has redspot"

These inferences are cases of induction. Accordingly they have predictive power: they can be used to make predictions about the feature patient which the same symptoms or diseases.

Usually when we want to learn something, we do not start from scratch: most often we already have some background knowledge relevant to the learning task. For instance, he has a fever but not an redspots. Then the previous rule1 cannot be used. Rule 2 and rule 3 cannot be used too. Now suppose if we see the background knowledge, we see that this person has the same addresses as another patients be suffer from measles. Since we know that measles is a rather contagious disease, we can infer that a also has measles. The fact that measles is contagious is not something that is expressed . Instead, it is a piece of knowledge we already had. Nevertheless the piece of background knowledge combined with the examples allow us to induce the general rule.

RULE 3: "If x has fever and y has measles, x and y lives in a same address then x has a measles".

This rule can then be combined with rule2 to predict that x will get redspots

## II. RELATED WORK.

The current paper presents a brief overview of Inductive Logic Programming (ILP) systems. ILP algorithms are of extraordinary interest for machine Learning, on the grounds that the vast majority of them offer viable techniques for expanding the introductions utilized in algorithm that that supervised learning undertakings[1]. The augmentation of the FOIL to address the records with continuous variable and information with missing values allows it to be implemented to a wider variety of troubles, which includes traditional characteristic-value classification issues. This paper reports the results of evaluating FOIL to C4.5 policies to a attribute-price problems. on the domains attempted, FOIL is computationally realistic and in comparable resulting accuracy[2].we reveal how distinctive formsof history are regularly integrated withaninductive approach for generating functionfr eeHornclause policies. furthermore,we evaluate, both theoretically and empirically, the impact that these types of know-how wear the fee and accuracy of mastering. finally, we demonstrate that a hybrid rationalization-based and inductive gaining knowledge of technique can advantageously use an approximate domain theory, even when this concept is incorrect and incomplete[3]. FOIL is a first-order learning systemhat uses statistics in a group of members of the family to assemble theories expressed in a dialect of Prolog. This paper offers an outline of the primary ideas and strategies used inside the contemporary version of the device, such as latest additions. We present examples of duties tackled with the aid of FOIL and of systems that adapt and expand its approach[4]. FOIL is an Inductive logic Programming algorithm to find out first order policies to giveanexplanationfor the styles concerned in asite of knowledge. Domains as records Retrievalor records Extracti onarehandicapsforFOIL dueto the big amount of statistics it wishes manage toplot the guidelines. Current solutions to issues in th ose domainnames are restricted todevising advert hoc area dependent inductive algorithms that use a much less-expressive formalism to code regulations[5]. FOIL is an Inductive Logic Programming Algorithm to get first order rules explain the patterns involved during a domain of data. Domains with a huge amount of information are handicaps for FOIL due to the explosion of the search of space to devise the rules. Current solutions to problems in these domains are restricted to devising ad hoc domain dependent inductive algorithms that use a less-expressive formalism to code rules. We work on optimizing FOIL learning process to affect such complex domain problems while retaining expressiveness.[6]

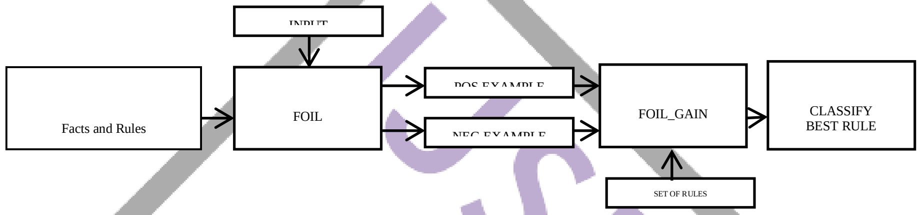
## III. PROPOSED SYSTEM

In first order learning, training data comprises a target predicate, which is defined by a collection of positives and negatives examples according to whether they satisfy the target predicate or not. Therefore, a set of support predicates is defined either extensionally, similarly to what was previously made with the target predicate or intentionally, by means of a set of rules. The goal is to learn a set of logic rules that explain the target predicate in terms of itself and the support predicates. FOIL is an algorithm of machine learning that evaluates first order rules. It uses separate-and-conquer method rather than divide-and conquer, focusing on creating a single rule at a time and removing any positive examples covered by each learnt rule. Then, it is invoked again to learn a second rule based on the remaining training examples. It is called a sequential covering algorithm because it sequentially learns a set of rules that together cover the full set of positive examples. In order to learn each rule, it follows a top-down approach, starting with the most general header rule, and guided by a greedy search, is adding new literals to the rule, until it does not satisfy any negative example belonging to the target predicate. The set of rules is ready when none positive examples of the target predicate remain to be satisfied.

## IV. PROBLEM STATEMENT

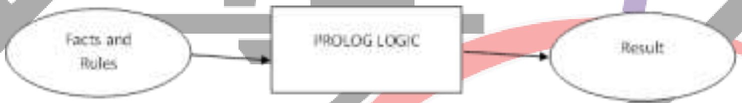
The problem statement states that, Whenever an set of Rules are given and we have to find out which is the best rule from the set and find out suitable elements which fits into respective rules.

## V. SYSTEM ARCHITECTURE



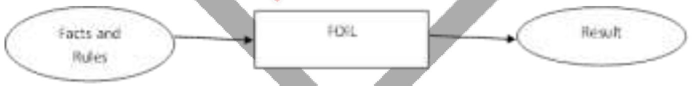
The characteristics of developed project can be described from the designed specification which mainly consists of architecture diagram modules of the project components Initially Facts and Rules would be present, which consists of real world data which includes all the target predicates. We would provide an Input to the foil algorithm which would help us to discriminate between the positive and negative examples. We would give the set of rules for which we are trying to find out FOIL_GAIN considering the positive and negative examples. Then considering the maximum foil gain we will decide which the best rule form the rest.

***Level 0 for Prolog system***



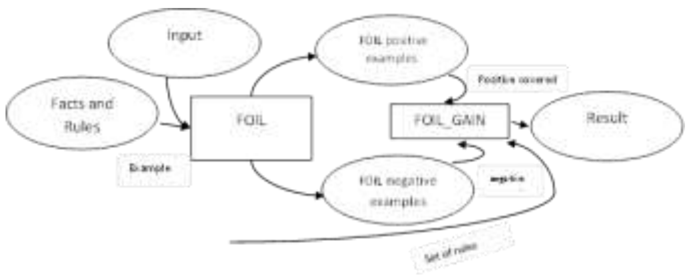
The facts and the rules would be known to the user. The user has to just ask a question to the prolog compiler. The prolog compiler would give an output as true or false depending on the background knowledge which it has. If we ask the question which is predicate itself it would give us possible outcome values. The values which are generated cannot be stored in any of the data structure.

***Level 0 for FOIL***



The level 0 of the FOIL algorithm is as shown in above figure. The background knowledge is given to the FOIL and FOIL algorithm would give us the output as result.

***Level 1 for FOIL***



Level 1 DFD is as shown in the figure initially the foil takes an Facts and rules with input as an input to foil, The foil would classify the input into two sub division as positive and negative examples. Considering positive and negative examples with an set of rules as an input to the FOIL_GAIN algorithm. The foil_gain would be generated for the particular rule. Considering the foil_gain system will decide which is the best rule.

## VI. RESULTS

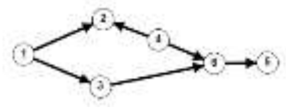
***Results of Prolog 1***

```
edge(1,2).
edge(1,3).
edge(3,6).
edge(4,2).
edge(4,6).
edge(6,5).
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z),path(Z,Y).
% c:/Users/VINAY/Desktop/mtech PROJECT/Prolog/Prolog/path1.pl compiled 0.00 sec, 8 clauses
?- path(4,2).
true .
?- path(6,5).
true
?- path(1,3).
true.
?- path(3,5).
true
?- path(1,5).
true
?- path(2,4).
false.
?- path(2,6).
false.
```

***Results of Prolog2***

```
parents(pam,bob).
parents(tom,bob).
parents(tom,liz).
parents(bob,ann).
parents(bob,pat).
parents(pat,jim).
ancestors(X,Y) :- parents(X,Y).
ancestors(X,Y) :- parents(X,Z),parents(Z,Y).
ancestors(X,Y) :- parents(X,Z),ancestors(Z,Y).
%c:/Users/VINAY/Documents/Prolog/fam.pl compiled 0.02 sec, 9 clauses
?- ancestors(pam,bob).
true .
?- ancestors(pam,ann).
true .
?- ancestors(pam,pat).
true .
?- ancestors(pam,jim).
true
```

***Results of FOIL1***



***INPUT:***

edges=[(1,2),(1,3),(4,2),(6,5),(4,6),(3,6)]

Rules: path(X,Y) :- edge(X,Y)

path(X,Y) :- edge(X,Z),path(Z,Y)

***OUTPUT:***

Generating all the possible paths using an edges.

```
paths= [(1, 2), (1, 3), (4, 6), (4, 5), (1, 5), (1, 6), (3, 6), (4, 2), (6, 5), (3, 5)]
all_possible_outcomes= [(1, 1), (1, 4), (1, 3), (1, 6), (1, 2), (1, 5), (4, 1), (4, 4), (4, 3), (4, 6), (4, 2), (4, 5), (3, 1), (3, 4), (3, 3), (3, 6), (3, 2), (3, 5), (6, 1), (6, 4), (6, 3), (6, 6), (6, 2), (6, 5), (2, 1), (2, 4), (2, 3), (2, 6), (2, 2), (2, 5), (5, 1), (5, 4), (5, 3), (5, 6), (5, 2), (5, 5)]
neg= [(1, 1), (1, 4), (4, 1), (4, 4), (4, 3), (3, 1), (3, 4), (3, 3), (3, 2), (6, 1), (6, 4), (6, 3), (6, 6), (6, 2), (2, 1), (2, 4), (2, 3), (2, 6), (2, 2), (2, 5), (5, 1), (5, 4), (5, 3), (5, 6), (5, 2), (5, 5)] #neg is generated by removing pos tuples from 'all' list
pos= [(1, 2), (1, 3), (4, 6), (4, 5), (1, 5), (1, 6), (3, 6), (4, 2), (6, 5), (3, 5)] #paths which are generated are called positives.
```

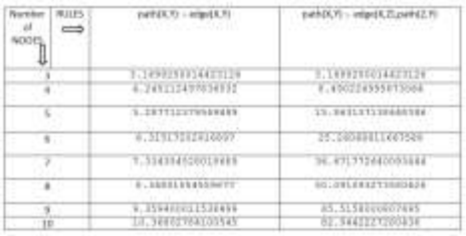
- For rule: path(X,Y) :- edge(X,Y)

positives ones which are covered: [(1, 2), (1, 3), (4, 6), (3, 6), (4, 2), (6, 5)]

```
negative ones which are covered: []
pos= [(4, 5), (1, 5), (1, 6), (3, 5)]
neg= [(1, 1), (1, 4), (4, 1), (4, 4), (4, 3), (3, 1), (3, 4), (3, 3), (3, 2), (6, 1), (6, 4), (6, 3), (6, 6), (6, 2), (2, 1), (2, 4), (2, 3), (2, 6), (2, 2), (2, 5), (5, 1), (5, 4), (5, 3), (5, 6), (5, 2), (5, 5)]
number of positive examples covered 6
number of negative examples covered 0
FOIL_GAIN= 11.087981439329699
path(X,Y) :- edge(X,Z)
positives ones which are covered: [(4, 5), (1, 5), (1, 6), (3, 5)]
negatives ones which are covered: [(1, 1), (1, 4), (4, 1), (4, 4), (4, 3), (6, 1), (6, 4), (6, 3), (6, 6), (6, 2), (3, 1), (3, 4), (3, 3), (3, 2)]
pos= [(4, 5), (1, 5), (1, 6), (3, 5)]
neg= [(1, 1), (1, 4), (4, 1), (4, 4), (4, 3), (3, 1), (3, 4), (3, 3), (3, 2), (6, 1), (6, 4), (6, 3), (6, 6), (6, 2), (2, 1), (2, 4), (2, 3), (2, 6), (2, 2), (2, 5), (5, 1), (5, 4), (5, 3), (5, 6), (5, 2), (5, 5)]
number of positive examples covered 4
number of negative examples covered 14
FOIL_GAIN= -1.2877123795494505
#extending  the z values..
pos= [(1, 5, 2), (1, 6, 2), (1, 5, 3), (1, 6, 3), (4, 5, 2), (4, 5, 6), (3, 5, 6)]
neg= [(1, 1, 2), (1, 4, 2), (1, 1, 3), (1, 4, 3), (4, 1, 2), (4, 4, 2), (4, 3, 2), (6, 1, 5), (6, 4, 5), (6, 3, 5), (6, 6, 5), (6, 2, 5), (4, 1, 6), (4, 4, 6), (4, 3, 6), (3, 1, 6), (3, 4, 6), (3, 3, 6), (3, 2, 6)]
Rule: path(X,Y) :- edge(X,Z),path(Z,Y)
positives ones which are covered: [(1, 5, 3), (1, 6, 3), (4, 5, 6), (3, 5, 6)]
negatives ones which are covered: []
pos= [(1, 5, 2), (1, 6, 2), (1, 5, 3), (1, 6, 3), (4, 5, 2), (4, 5, 6), (3, 5, 6)]
neg= [(1, 1, 2), (1, 4, 2), (1, 1, 3), (1, 4, 3), (4, 1, 2), (4, 4, 2), (4, 3, 2), (6, 1, 5), (6, 4, 5), (6, 3, 5), (6, 6, 5), (6, 2, 5), (4, 1, 6), (4, 4, 6), (4, 3, 6), (3, 1, 6), (3, 4, 6), (3, 3, 6), (3, 2, 6)]
number of positive examples covered 4
number of negative examples covered 0
FOIL_GAIN= 14.7839752524396
for the rule path(X,Y) :- edge(X,Y) the foil gain is: 11.087981439329699
for the rule path(X,Y) :- edge(X,Z),path(Z,Y) the foil gain is: 14.7839752524396
```

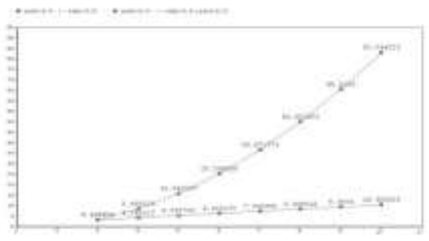
**For different number of Nodes**

```
3nodes=[(1,2),(2,3)]
4e nodes =[(1,2),(2,3),(3,4)]
5 nodes =[(1,2),(2,3),(3,4),(4,5)]
6 nodes =[(1,2),(2,3),(3,4),(4,5),(5,6)]
7 nodes =[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7)]
8 nodes =[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8)]
9 nodes =[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9)]
10 nodes =[(1,2),(2,3),(3,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10)]
```



| Number of NODES ⇩ | RULES ⇨ | path(X,Y) :- edge(X,Y) | path(X,Y) :- edge(X,Z),path(Z,Y) |
|---|---|---|---|
| 3 | | 3.1699250014423126 | 3.1699250014423126 |
| 4 | | [illegible] | [illegible] |
| 5 | | 5.2877123795494485 | [illegible] |
| 6 | | [illegible] | [illegible] |
| 7 | | [illegible] | [illegible] |
| 8 | | [illegible] | [illegible] |
| 9 | | [illegible] | [illegible] |
| 10 | | [illegible] | [illegible] |

**foil gain for number of nodes**

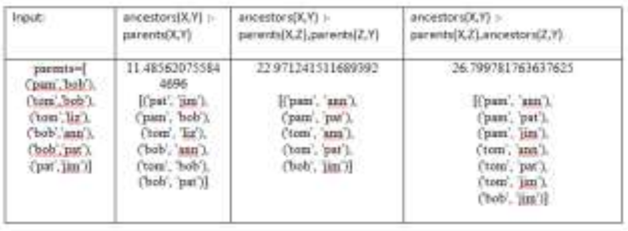


**Foil gain comparison with different Rules**

From the above table there are different number of nodes for the graph and the rules (path(X,Y):- edge(X,Y) and the path(X,Y):- edge(X,Z),path(Z,Y)). The foil_gain is calculated for the each node wrt to its rules and it has been tabulated on the table.
The above table has been represented in form of graph, in the Y-axis we have number of nodes and in the X-axis we have foil_gain. For the rule path(X,Y) :- edge(X,Y) it is an linear curve, whereas for the path(X,Y) :- edge(X,Z),path(Z,Y) the curve is exponential.

From the graph we can conclude that the foil_gain increases with increase in number of nodes in a graph. In the initial stage, If the number of nodes are less then, the foil_gain for the different rules would almost be same because the number of positive elements covered for both the rules would be same, However increase in the number of nodes, the gain would drastically differ from each other because the number of positive covered by second rule would be more when compared it with first rule . In general the rules with single target predicate, the foil_gain would be relatively less (linearly) as the number of nodes increases. If the target predicates are more than one, then the foil_gain would increase exponentially as the number of node increases.

***Results of FOIL-2***



| Input: | ancestors(X,Y) :- parents(X,Y) | ancestors(X,Y) :- parents(X,Z),parents(Z,Y) | ancestors(X,Y) :- parents(X,Z),ancestors(Z,Y) |
|---|---|---|---|
| parents=[ ('pam','bob'), ('tom','bob'), ('tom','liz'), ('bob','ann'), ('bob','pat'), ('pat','jim')] | 11.485620755844696 [('pat', 'jim'), ('pam', 'bob'), ('tom', 'liz'), ('bob', 'ann'), ('tom', 'bob'), ('bob', 'pat')] | 22.971241511689392 [('pam', 'ann'), ('pam', 'pat'), ('tom', 'ann'), ('tom', 'pat'), ('bob', 'jim')] | 26.799781763637625 [('pam', 'ann'), ('pam', 'pat'), ('pam', 'jim'), ('tom', 'ann'), ('tom', 'pat'), ('tom', 'jim'), ('bob', 'jim')] |

**foil gain for the different rules**

In this section, there are 3 different rules which determine the ancestor's property. For the each rule foil gain is found and also found what are all the possibilities values of X and Y which fits into the ancestors.

## VII. CONCLUSION

In this project, we have successfully implemented and deployed the foil algorithm. We have found out the foil_gain for the different set of input and even for different set of rules. When there is an ambiguity in selecting the best rule foil_gain metric helps us to pick the best rule out of all the other rules and in order to find the foil_gain we would have generated a set of elements which fits rule. The problem of finding out the relations in a complex family has been effectively solved using foil_gain algorithm.

## VIII. ACKNOWLEDGMENT

I WOULD LIKE TO THANK MY GUIDE, **PROF. B I KHODANDPUR**, PROFESSOR, DEPARTMENT OF INFORMATION SCIENCE, DSCE, FOR HIS CONSTANT GUIDANCE THAT HELPED ME IN COMPLETING THE PROJECT PHASE SUCCESSFULLY.

## REFERENCES

[1] Overview of Inductive Logic Programming (ILP) Systems Svetla Boytcheva Department of Information Technologies, Faculty of Mathematics and Informatics, Sofia University, Bulgaria
[2] FIRSTORDER LEARNING AND ZEROTH ORDER LEARNING R.M.CAMERON-JONES and J.R.QUINLAN,Basser department of computer science the university of sydney,nws Australia.
[3] The Utility of Knowledge in Inductive Learning MICHAEL PAZZANI, Department of Information & Computer Science, University of California, Irvine, lrvine, CA 92717-3425
[4] Induction of Logic Programs: FOIL and Related Systems J. R. QUINLAN University of Sydney, Sydney, Australia 2006. R. M. CAMERON-JONES University of Tasmania, Launceston, Australia 7250.
[5] On improving FOIL Algorithm Patricia Jimeenez, J.L. Arjona, J.L. Alvarez The Distributed Group – Onuba Escuela Polit´ecnica Superior. Crta. Huelva - La R´abida. Palos de la Frontera 21071
[6] Optimising FOIL by new scoring functions, Patricia Jimenez Universidad de Sevilla, José Luid Arjona Universidad de Huelva, José Luid Arjona Universidad de Huelva