CSCI 5582
Artificial Intelligence

Lecture 22

Jim Martin

Today 11/16

- Finish up ILP/FOIL
- Break
  - HW questions
- Quiz review
  - Probabilistic sequence processing
  - Supervised ML
    - Learning Classifiers
      - DTs, DLs, Naïve Bayes, Ensembles, SVMs
    - Concept Learning
      - Version Spaces, FOIL

Relational Learning and Inductive Logic Programming

- Fixed feature vectors are a very limited representation of objects.
- Examples or target concept may require relational representation that includes multiple entities with relationships among them.
- First-order predicate logic is a more powerful representation for handling such relational descriptions.

ILP Example

- Learn definitions of family relationships given data for primitive types and relations.

  brother(A,C), parent(C,B) -> uncle(A,B)

  husband(A,C), sister(C,D), parent(D,B) -> uncle(A,B)
- Given the relevant predicates and a database populated with positive and negative examples
- By database I mean sets of tuples for each of the relevant relations

FOIL
First-Order Inductive Logic

- Top-down sequential covering algorithm to learn first order theories.
- Background knowledge provided extensionally (ie. A model)
- Start with the most general rule possible. (T --> P(x))
- Specialize it on demand…
- Specializations of a clause include adding all possible literals one at a time to the antecedent…
  - A -> P
  - B -> P
  - C -> P…

  Where A, B and C are predicates already in the domain theory.

  We're working top-down from the most general hypothesis so what's driving things?

FOIL

- At a high level.
  - Start with the most general H
  - Repeatedly constructs clauses that cover a subset of the positive examples and none of the negative examples.
  - Then remove the covered positive examples
  - Constructs another clause
  - Repeat until all the positive examples are covered.

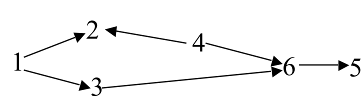
FOIL

- Constructing candidate clauses
  - Any predicate (negated or not), args are variables
    - But every predicate must contain a variable from an earlier literal or the head of the clause (antecedent)
  - Equality constraints on variables
  - Some arithmetic

FOIL Training Data

- Background knowledge consists of complete set of tuples for each background predicate for this universe.
- Example: Consider learning a definition for the target predicate `path` for finding a path in a directed acyclic graph.

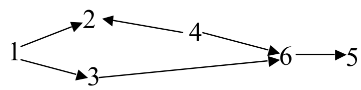
```
edge(X,Y) -> path(X,Y)
edge(X,Z)^ path(Z,Y) -> path(X,Y)
```




```
edge: {<1,2>,<1,3>,<3,6>,<4,2>,<4,6>,<6,5>}
path: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
       <4,2>,<4,6>,<4,5>,<6,5>}
```

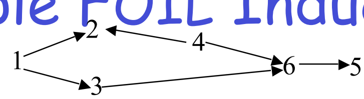
FOIL Negative Training Data

- Negative examples of target predicate can be provided directly, or generated indirectly by making a *closed world assumption*.
  - Every pair of constants <X,Y> not in positive tuples for `path` predicate.



Negative `path` tuples:
{<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Sample FOIL Induction

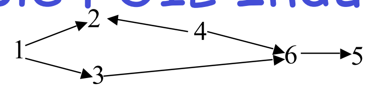


Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Start with clause:
`T -> path(X,Y)`
Possible literals to add:
`edge(X,X),edge(Y,Y),edge(X,Y),edge(Y,X),edge(X,Z),`
`edge(Y,Z),edge(Z,X),edge(Z,Y),path(X,X),path(Y,Y),`
`path(X,Y),path(Y,X),path(X,Z),path(Y,Z),path(Z,X),`
`path(Z,Y),X=Y,`
plus negations of all of these.

Sample FOIL Induction



Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

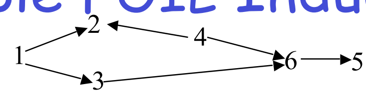
`edge(X,X) -> path(X,Y)`

Covers 0 positive examples

Covers 6 negative examples

Not a good literal to try.

Sample FOIL Induction



Pos: {<1,2>,<1,3>,<1,6>,<1,5>,<3,6>,<3,5>,
<4,2>,<4,6>,<4,5>,<6,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

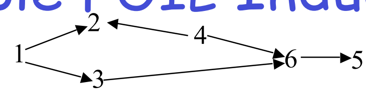
`edge(X,Y) -> path(X,Y)`

Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

`edge(X,Y) -> path(X,Y)`

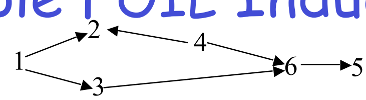
Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Remove covered positive tuples.

Sample FOIL Induction



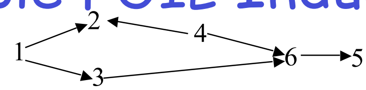
Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Start new clause

`T -> path(X,Y)`

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

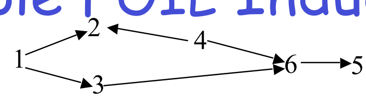
`edge(X,Y) -> path(X,Y)`

Covers 0 positive examples

Covers 0 negative examples

Not a good literal.

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,<2,1>,<2,2>,<2,3>,<2,4>,<2,5>,<2,6>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,<5,1>,
<5,2>,<5,3>,<5,4>,<5,5>,<5,6>,<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

Test:

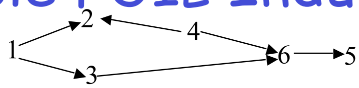
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples

Covers 14 of 26 negative examples

Eventually chosen as best possible literal

Sample FOIL Induction



Pos: {<1,6>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,
<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

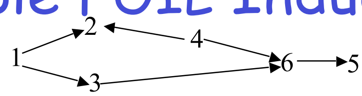
Test:
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples   Covers 14 of 26 negative examples

Eventually chosen as best possible literal

Negatives still covered, remove uncovered examples.

Sample FOIL Induction



Pos: {<1,6,2>,<1,6,3>,<1,5>,<3,5>,
<4,5>}

Neg: {<1,1>,<1,4>,
<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,
<6,1>,<6,2>,<6,3>,
<6,4>,<6,6>}

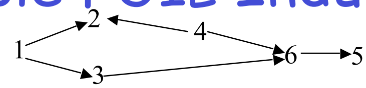
Test:
`Edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples   Covers 14 of 26 negative examples

Eventually chosen as best possible literal

Negatives still covered, remove uncovered examples.
Expand tuples to account for possible Z values. <X,Y,Z>

Sample FOIL Induction



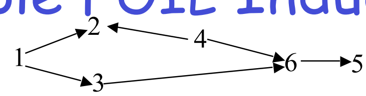
Pos: `{<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5>,`
`<4,5>}`

Neg: `{<1,1>,<1,4>,`
`<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,`
`<6,1>,<6,2>,<6,3>,`
`<6,4>,<6,6>}`

Test:
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples
Covers 14 of 26 negative examples
Eventually chosen as best possible literal
Negatives still covered, remove uncovered examples.
Expand tuples to account for possible `Z` values. <X,Y,Z>

Sample FOIL Induction



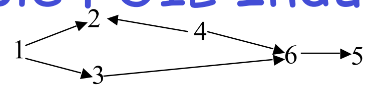
Pos: `{<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>,`
`<4,5>}`

Neg: `{<1,1>,<1,4>,`
`<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,`
`<6,1>,<6,2>,<6,3>,`
`<6,4>,<6,6>}`

Test:
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples
Covers 14 of 26 negative examples
Eventually chosen as best possible literal
Negatives still covered, remove uncovered examples.
Expand tuples to account for possible `Z` values. <X,Y,Z>

Sample FOIL Induction



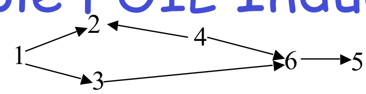
Pos: `{<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>,`
`<4,5,2>, <4,5,6>}`

Neg: `{<1,1>,<1,4>,`
`<3,1>,<3,2>,<3,3>,<3,4>,<4,1>,<4,3>,<4,4>,`
`<6,1>,<6,2>,<6,3>,`
`<6,4>,<6,6>}`

Test:
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples
Covers 14 of 26 negative examples
Eventually chosen as best possible literal
Negatives still covered, remove uncovered examples.
Expand tuples to account for possible `Z` values. <X,Y,Z>

Sample FOIL Induction



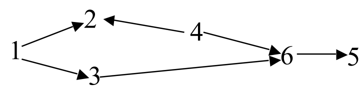
Pos: `{<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>,`
`<4,5,2>, <4,5,6>}`

Neg: `{<1,1,2>,<1,1,3>,<1,4,2>,<1,4,3>,<3,1,6>,<3,2,6>,`
`<3,3,6>,<3,4,6>,<4,1,2>,<4,1,6>,<4,3,2>,<4,3,6>`
`<4,4,2>,<4,4,6>,<6,1,5>,<6,2,5>,<6,3,5>,`
`<6,4,5>,<6,6,5>}`

Test:
`edge(X,Z) -> path(X,Y)`

Covers all 4 positive examples
Covers 14 of 26 negative examples
Eventually chosen as best possible literal
Negatives still covered, remove uncovered examples.
Expand tuples to account for possible `Z` values. <X,Y,Z>

Sample FOIL Induction



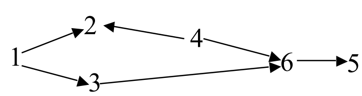
Pos: {<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>, <4,5,2>, <4,5,6>}

Neg: {<1,1,2>,<1,1,3>,<1,4,2>,<1,4,3>,<3,1,6>,<3,2,6>, <3,3,6>,<3,4,6>,<4,1,2>,<4,1,6>,<4,3,2>,<4,3,6> <4,4,2>,<4,4,6>,<6,1,5>,<6,2,5>,<6,3,5>, <6,4,5>,<6,6,5>}

Continue specializing clause:

```
edge(X,Z) -> path(X,Y)
```


Sample FOIL Induction



Pos: {<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>, <4,5,2>, <4,5,6>}

Neg: {<1,1,2>,<1,1,3>,<1,4,2>,<1,4,3>,<3,1,6>,<3,2,6>, <3,3,6>,<3,4,6>,<4,1,2>,<4,1,6>,<4,3,2>,<4,3,6> <4,4,2>,<4,4,6>,<6,1,5>,<6,2,5>,<6,3,5>, <6,4,5>,<6,6,5>}

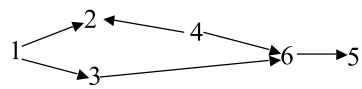
Test:

```
edge(X,Z) ^ edge(Z,Y) -> path(X,Y)
```

Covers 3 positive examples

Covers 0 negative examples

Sample FOIL Induction



Pos: {<1,6,2>,<1,6,3>,<1,5,2>,<1,5,3>,<3,5,6>,
<4,5,2>, <4,5,6>}

Neg: {<1,1,2>,<1,1,3>,<1,4,2>,<1,4,3>,<3,1,6>,<3,2,6>,
<3,3,6>,<3,4,6>,<4,1,2>,<4,1,6>,<4,3,2>,<4,3,6>
<4,4,2>,<4,4,6>,<6,1,5>,<6,2,5>,<6,3,5>,
<6,4,5>,<6,6,5>}

Test:

```
edge(X,Z) ^ path(Z,Y) -> path(X,Y)
```

Covers 4 positive examples   Covers 0 negative examples

Eventually chosen as best literal; completes clause.

Definition complete, since all original <X,Y> tuples are covered (by way of covering at least one of each positive <X,Y,Z> tuple.)

More Realistic Applications

- Classifying chemical compounds as mutagenic (cancer causing) based on their graphical molecular structure and chemical background knowledge.
- Classifying web documents based on both the content of the page and its links to and from other pages with particular content.
  - A web page is a university faculty home page if:
    - It contains the words "Professor" and "University", and
    - It is pointed to by a page with the word "faculty", and
    - It points to a page with the words "course" and "exam"

Rule Learning and ILP Summary

- There are effective methods for learning symbolic rules from data using greedy sequential covering and top-down or bottom-up search.
- These methods have been extended to first-order logic to learn relational rules and recursive Prolog programs.
- Knowledge represented by rules is generally more interpretable by people, allowing human insight into what is learned and possible human approval and correction of learned knowledge.

Break

- HW questions?
  - Elizabeth?

Break

- The next quiz will be on 11/28.
- It will cover the ML material and the probabilistic sequence material.
- The readings for this quiz are:
  - Chapter 18
  - Chapter 19
  - Chapter 20: 712-718
  - HMM chapter posted on the web

Quiz Topics

- Sequence processing
- Classifiers
- Concept Learning

Probabilistic Sequence Processing

- Reading: Assigned Chapter
  - Know the three problems
    - P(observations | model)
    - Argmax P(state sequence | observations, model)
    - Argmax P(model parameter | observations, model structure)

Probabilistic Sequence Processing

- The basis for the techniques
  - Independence assumptions
  - For a first order HMM
    1. ?
    2. ?

Probabilistic Sequence Processing

- The basis for the techniques
  - Independence assumptions
  - For a first order HMM
    1. Current state depends only on the previous
    2. ?

Probabilistic Sequence Processing

- The basis for the techniques
  - Independence assumptions
  - For a first order HMM
    1. Current state depends only on the previous
    2. Current output depends only on current state

Probabilistic Sequence Processing

- Know the computations
- Know the algorithms

Classifiers

- Chapter 18
- Basic induction task
  - DT, DL, Naïve Bayes
  - Ensembles (bagging, boosting)
  - For SVMS, just focus on the two main ideas (not the math)
- Learning theory
  - What are the key elements?

Classifiers

Theory: three main things we discussed:

1. ?
2. ?
3. ?

CSCI 5582 Fall 2006

Classifiers

Theory: three main things we discussed:

1. Size of the hypothesis space
2. ?
3. ?

CSCI 5582 Fall 2006

Classifiers

Theory: three main things we discussed:

1. Size of the hypothesis space
2. Number of training examples
3. ?

Classifiers

Theory: three main things we discussed:

1. Size of the hypothesis space
2. Number of training examples
3. Occam

Classifiers

Practice:

1. Training and testing
2. Learning as search
    1. Managing choices
    2. Making choices
    3. Termination conditions

Training Set

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0±10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30±60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0±10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10±30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0±10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0±10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0±10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10±30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0±10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30±60*	*Yes*

Concept Learning

- Chapter 19:
- Focus on what the task is
- How it's different from classifier induction
  - Version space learning
  - How FOIL works