COMP3411/9814 Artificial Intelligence
Term 1, 2021

Assignment 2 – Machine Learning
(DRAFT ONLY)

Due: Friday 23 April, 10pm

Marks: 20% of final assessment for COMP3411/9814 Artificial Intelligence

Part 1: Decision Trees

Question 1.1 (5 marks):

Consider the decision tree learning algorithm of Figure 7.7 and the data of Figure 7.1 from Poole & Mackworth [1], also presented below. Suppose, for this question, the stopping criterion is that all of the examples have the same classification. The tree of Figure 7.6 was built by selecting a feature that gives the maximum information gain. This question considers what happens when a different feature is selected.

Example	*Author*	*Thread*	*Length*	*Where_read*	*User_action*
e_1	*known*	*new*	*long*	*home*	*skips*
e_2	*unknown*	*new*	*short*	*work*	*reads*
e_3	*unknown*	*followup*	*long*	*work*	*skips*
e_4	*known*	*followup*	*long*	*home*	*skips*
e_5	*known*	*new*	*short*	*home*	*reads*
e_6	*known*	*followup*	*long*	*work*	*skips*
e_7	*unknown*	*followup*	*short*	*work*	*skips*
e_8	*unknown*	*new*	*short*	*work*	*reads*
e_9	*known*	*followup*	*long*	*home*	*skips*
e_{10}	*known*	*new*	*long*	*work*	*skips*
e_{11}	*unknown*	*followup*	*short*	*home*	*skips*
e_{12}	*known*	*new*	*long*	*work*	*skips*
e_{13}	*known*	*followup*	*short*	*home*	*reads*
e_{14}	*known*	*new*	*short*	*work*	*reads*
e_{15}	*known*	*new*	*short*	*home*	*reads*
e_{16}	*known*	*followup*	*short*	*work*	*reads*
e_{17}	*known*	*new*	*short*	*home*	*reads*
e_{18}	*unknown*	*new*	*short*	*work*	*reads*
e_{19}	*unknown*	*new*	*long*	*work*	? skips
e_{20}	*unknown*	*followup*	*short*	*home*	? skips

Figure 7.1: Examples of a user's preferences

a) Suppose you change the algorithm to always select the first element of the list of features. What tree is found when the features are in the order [*Author, Thread, Length, WhereRead*]? Does this tree represent a different function than that found with the maximum information gain split? Explain.

b) What tree is found when the features are in the order [*WhereRead, Thread, Length, Author*]? Does this tree represent a different function than that found with the maximum information gain split or the one given for the preceding part? Explain.

c) Is there a tree that correctly classifies the training examples but represents a different function than those found by the preceding algorithms? If so, give it. If not, explain why.

Question 1.2 (5 marks):

The goal is to take out-of-the-box models and apply them to a given dataset. The task is to analyse the data and build a model to predict whether income exceeds $50K/yr based on census data (also known as "Census Income" dataset).

Use the data set **Adult Data Set** from the Machine Learning repository [2].
Use the supervised learning methods discussed in the lectures, Decision Trees.

Do not code these methods: instead use the J48 implementation in Weka. Read the Weka documentation on Decision Trees, and the linked pages describing the parameters of the methods. You will need to download Weka, from he link above.

This question will help you master the workflow of model building. For example, you'll get to practice how to use the critical steps:

- Importing data
- Cleaning data
- Splitting it into train/test or cross-validation sets
- Pre-processing
- Transformations
- Feature engineering

Use the Weka documentation pages for instructions how to use J48. See also this WebCMS page.

The data are available here: http://archive.ics.uci.edu/ml/

machine-learning-databases/adult/

Part 2: Inductive Logic Programming

Duce is a program devised by Muggleton [3] to perform learning on a set of propositional clauses. The system includes 6 operators that transform pairs of clauses into a new set of clauses. Your job is to write Prolog programs to implement 5 of the 6 operators. The other one is given as an example to get you started.

Inter-construction. This transformation takes two rules, with *different* heads, such as

```
x <- [b, c, d, e]
y <- [a, b, d, f]
```

and replaces the with rules

```
x <- [c, e, z]
y <- [a, f, z]
z <- [b, d]
```

To make processing the rules easier, we represent the body of the clause by a list and we define the operator `<-` to mean "implied by" or "if".

A Prolog program to implement inter-construction is given as an example to give you hints about how to write the remaining 5 operators.

```
:- op(300, xfx, <-).

inter_construction(C1 <- B1, C2 <- B2, C1 <- Z1B, C2 <- Z2B, C <- B) :-
      C1 \= C2,
      intersection(B1, B2, B),
      gensym(z, C),
      subtract(B1, B, B11),
      subtract(B2, B, B12),
      append(B11, [C], Z1B),
      append(B12, [C], Z2B).
```

First, we define `<-` as an operator that will allow us to use the notation `x <- y`. The program uses Prolog built-in predicates to perform set operations on lists and to generate new symbols.

1. The first line the program assumes that the first two arguments are given as propositional clauses. The remaining three arguments are the output clauses, as in the example above.
2. inter_construction operates on two clauses that gave different heads, i.e. C1 \= C2.
3. We then find the interaction of the bodies of the clauses and call it, B.
4. **gensym** is a builtin predicate that creates a new name from a base name. So **gensym(z, C)** binds **C** to **z1**. Every subsequent call to gensym, with base z, will create names, z2, z3, …. Calling **reset_gensym** will restart the numbering sequence.
5. At this point the last argument `C <- B = z1<-[b, d]` because **C** is bound to **z1** and **B** is the intersection `[b, d]`.
6. The bodies **Z1B** and **Z2B**, are obtained by subtracting the intersection, **B**, from **B1** and **B2** and appending the single element **[C].** So we can run the program:

```
?- inter_construction(x <- [b, c, d, e], y <- [a, b, d, f], X, Y, Z).
X = x<-[c, e, z1],
Y = y<-[a, f, z1],
Z = z1<-[b, d].
```

obtaining the desired result.

You will write programs for the other five operators, defined below. All of these programs can be created from the builtin predicates used in the **inter-construction** code. So all you have to do is work out how to combine them. The only other builtin predicate you may need is to test if one list is a subset of another with **subset(X, Y)**, where **X** is a subset of **Y**.

You can assume that the inputs will always be valid clauses and the lists will always be non-empty, i.e. with at least one element.

Question 2.1 (2 marks):

Intera-construction. This transformation takes two rules, with the *same* head, such as

```
x <- [b, c, d, e]
x <- [a, b, d, f]
```

and replaces the with rules

```
x <- [b, d, z]
z <- [c, e]
z <- [a, f]
```

That is, we merge the two, **x**, clauses, keeping the intersection and adding a new predicate, **z**, that distributes the differences to two new clauses.

```
?- intra_construction(x <- [b, c, d, e], x <- [a, b, d, f], X, Y, Z).
X = x<-[b, c, d, e, z1],
Y = z1<-[c, e],
Z = z1<-[a, f].
```

Question 2.2 (2 marks):

Absorption. This transformation takes two rules, with the *different* heads, such as

```
x <- [a, b, c, d, e]
y <- [a, b, c]
```

and replaces the with rules

```
x <- [y, d, e]
y <- [a, b, c]
```

Note that the second clause is unchanged. This operator checks to see if the body of one clause is a subset of the other. If it is, the common elements can be removed from the larger clause and replaced by the head of the smaller one.

```
?- absorption(x <- [a, b, c, d,e], y <- [a, b, c], X, Y).
X = x<-[y, d, e],
Y = y<-[a, b, c].
```

Question 2.3 (2 marks):

Identification. This transformation applies to two rules that have the *same* head and one clause has *exactly one symbol* that does not appear in the other clause, for example:

```
x <- [a, b, c, d, e]
x <- [a, b, y]
```

can be replaced by rules

```
x <- [a, b, y]
y <- [c, d, e]
```

That is, we take the intersection out of each clause and if there is exactly one symbol left in one of the clauses, we identify it with the set of symbols from the other clause.

```
?- identification(x <- [a, b, c, d, e], x <- [a, b, y], X, Y).
X = x<-[a, b, y],
Y = y<-[c, d, e].
```

Question 2.4 (2 marks):

Dichotomisation. This transformation is different from the other in that it works on positive and negative examples of the same predicate. For example:

```
x <- [a, b, c, d]
not(x) <- [a, c, j, k]
```

can be replaced by rules

```
x <- [a, c, z]
not(x) <- [a, c, not(z)]
z <- [b, d]
not(z) <- [j, k]
```

This is similar to a decision tree splitting rule where we create a binary split into a branch with positive examples of x and branch with negative examples. Again, it depends on finding the intersection of the two input clauses, where one is a positive example, `x`, and one is a negative example, `not(x)`. Keep the intersection in each clause and create two new clauses, `z`, and `not(z)`, to distinguish the positive and negative examples.

```
?- dichotomisation(x <- [a,b,c,d], not(x) <- [a,c,j,k], A, B, C, D).
A = x<-[a, c, z1],
B = not(x)<-[a, c, not(z1)],
C = z1<-[b, d],
D = not(z1)<-[j, k].
```

Question 2.5 (2 marks):

Truncation. This is the simplest transformation. It takes two rules and simply drops the differences to leave just one rule. For example

```
x <- [a, b, c, d]
x <- [a, c, j, k]
```

are replaced by

```
x <- [a, c]
```

That is, the body of the new clause is just the intersection of the bodies of the input clauses.

```
?- truncation(x <- [a, b, c, d], x <- [a, c, j, k], X).
X = x<-[a, c].
```

The complete Duce algorithm performs a search, looking for combination of these operators that will lead to the greater compression over the database of examples. Here compression is measured by the reduction in the number of symbols in the database. You don't have to worry about the search algorithm, just implement the operators, as above.

References

1. Poole &Mackworth, Artificial Intelligence: Foundations of Computational Agents, Chapter 7, Supervised Machine Learning)
2. http://archive.ics.uci.edu/ml/datasets/Adult).
3. Muggleton, S. (1987). Duce, An oracle based approach to constructive induction. Proceedings of the International Joint Conference on Artificial Intelligence, 287–292.

Submission

This assignment must be submitted electronically.

Put your zID and your name at the top of every page of your submission!

give cs3411 assign2 part1.pdf part2.pl

Late submissions will incur a penalty of 10% per day, applied to the maximum mark.

Group submissions will not be allowed. By all means, discuss the assignment with your fellow students. But you must write (or type) your answers individually. **Do NOT copy anyone else's assignment, or send your assignment to any other student.**