**CMPE 480 INTRODUCTION TO ARTIFICIAL INTELLIGENCE**

**FINAL ANSWERS**

1. male (Emrah)

male (Onat)

male (Cemal)

female (Selin)

female (Nurgül)

female (Sibel)

loves (Emrah, Selin)

loves (Selin, Emrah)

loves (Selin, Nurgül)

loves (Selin, Onat)

loves (Nurgül, Cemal)

loves (Nurgül, Onat)

loves (Nurgül, Sibel)

loves (Cemal, Emrah)

loves (Cemal, Selin)

loves (Onat, Selin)

loves (Onat, Nurgül)

∀X ∀Y (loves (X,Y) ∧ loves (Y,X) → gocine (X,Y))

∀X ∀Y (gocine (X,Y) ∧ ((male (X) ∧ female (Y)) ∨ (female (X) ∧ male (Y))) → marry (X,Y))

b) male (emrah).

male (onat).

male (cemal).

female (selin).

female (nurgül).

female (sibel).

loves (emrah, selin).

loves (selin, emrah).

loves (selin, nurgül).

loves (selin, onat).

loves (nurgül, cemal).

loves (nurgül, onat).

loves (nurgül, sibel).

loves (cemal, emrah).

loves (cemal, selin).

loves (onat, selin).

loves (onat, nurgül).

gocine (X,Y) :– loves (X,Y), loves (Y,X).

marry (X,Y) :– gocine (X,Y), male (X), female (Y).

marry (X,Y) :– gocine (X,Y), female (X), male (Y).

c) goal: marry (X,Y).

marry (X,Y) :– gocine (X,Y), male (X), female (Y)

gocine (A,B) :– loves (A,B), loves (B,A) {variables must be distinct}

Substitutions: A/X, B/Y, so we obtain:

marry (X,Y) :– male (X), female (Y), loves (X,Y), loves (Y,X)

male (emrah)

Substitutions: X/emrah, so we obtain:

marry (emrah,Y) :– female (Y), loves (emrah,Y), loves (Y,emrah)

female (selin)

Substitutions: Y/selin, so we obtain:

marry (emrah,selin) :– loves (emrah,selin), loves (selin,emrah)

loves (emrah, selin)

Substitutions: none, we obtain:

marry (emrah,selin) :– loves (selin, emrah)

loves (selin,emrah)

Substitutions: none, we obtain:

marry (emrah,selin)

…

Output is:

X=emrah, Y=selin

X=onat, Y=selin

X=onat, Y=nurgül

X=selin, Y=emrah

X=selin, Y=onat

X=nurgül, Y=onat

1. *Domain-independent, important attributes:*

The classical domain-independent attributes which are important in knowledge representation are *isa* and *instance* attributes. These attributes automatically support property inheritance They are represented in different ways in different formalisms.

We can give the following example for semantic network representation:

instance isa

In logic, we can represent as follows:

instance (Ali, Pitcher)

isa (Pitcher, Baseball Player)

or

Pitcher (Ali)

∀x Pitcher (x) 🡪 Baseball Player (x)

1. *Inverse of an attribute:*

An inverse attribute of an attribute enables us to interpret a relation in both ways. In logical representation, this is done automatically. For instance,

team (Ali, GS)

can be interpreted in both directions: Team of Ali is GS or GS is the team for Ali.

In semantic networks, we can use two different attributes as shown below:

team

team-member

1. *Attribute hierarchy:*

Like classes and values, attributes may also form a hierarchy. The example below shows an hierarchical relation (isa relation) between two attributes:

isa

Such relations may be important during reasoning. For example, given a request such as “Display the physical sizes of ...” in a system, we can understand that “height” is a “physical size” by making use of the above relationship and display the height data.

1. *Single-valued vs. multi-valued attributes:*

A single-valued attribute may take a single value at a time, whereas a multi-valued attribute may take more than one value. In the case of a single-valued attribute, it is not allowed to assert more than one value at a time in the knowledge base. For example, height of a person may be a single-valued attribute. When another height value is tried to be inserted for a particular person in the KB, a contradiction signal can be given by the system.

1. *Low-level and high-level primitives:*

Low-level primitives are used to represent the data in a more detailed level, whereas high-level primitives are used to represent in a more abstract way. For instance, the sentence “John spotted Sue” can be represented in high-level as

spot (agent(John), object(Sue))

or in low-level as

see (agent(John), object(Sue), timespan(briefly)).

In the first case, we can answer questions like “Who spotted Sue?” easily, but we cannot answer questions like “Did John see Sue?” without knowing that “spotting” implies “seeing”.

The basic advantage of low-level primitives is that we can represent similar but different things (e.g. “see”, “spot”, “look”, etc.) in the same way (using “see” only). This means that we use few number of primitives (operators), thus few inference rules. This makes reasoning in a system much easier. Remember the conceptual dependency notation. It assumes that every action can be represented with just 10-15 primitives (ptrans, attend, etc.).

The disadvantage of low-level primitives is that it uses more storage. For instance, in the representation of the data “John spotted Sue” and “Sue spotted Mary” with the above “see” primitive, “timespan(briefly)” argument is stored twice. Another disadvantage is that it is more difficult to convert from natural language to the representation, since more detail should be extracted from the data.

1. *Script identification:*

The scripts can be indexed by the significant words they contain. For instance, in the “restaurant” script, we can index it with the words like “steak”, “order”, “bill”. Then, when these words occur in the given text, we can access that script (although the word “restaurant” may not appear in the text).

But, there is a problem that a word may point to more than one script. For instance, the word “steak” may indicate both “restaurant” and “supermarket” scripts. In this case, we can use the important words in the text and take the intersection of the structures they point to. For instance, if the word “steak” indicates “restaurant” and “supermarket” scripts and the word “bill” indicates “restaurant” and “shopping” scripts, the intersection is the “restaurant” script and we access it.

Another problem is that the intersection may be empty. For instance, in the text “John rode his bicycle to Steak and Ale ... He paid the bill ...”, the intersection of the words “steak”, “bill” and “bicycle” will be empty, since the word “bicycle” is not related to restaurants. In this case, we can give an order to the words and begin from the most important word. If we first use “steak” and then “bill”, the “restaurant” script will be identified, without the need to consider the word “bicycle”.

1. **Training:**

We will use the following formula to identify the feature that will be used to split the tree:

where *n* is the number of branches after the tree is split, *k* is the number of classes (outputs), *Si* is the set of data that belong to split (branch) *i*, and *Sij* is the set of data that belong to split *i* and class *j*.

Step 1:

For feature “*status*”: (The parent node shows the feature and the set of data that belong to that node; the branches show the possible values of the feature; the children show the data that belong to each feature value and the corresponding output values.)

status

{1,2,3,4,5,6,7,8}

faculty staff student

{1, 3, 6} {2, 5, 8} {4, 7}

N Y Y N N N Y Y

For feature “*floor*”:

floor

{1,2,3,4,5,6,7,8}

three four five

{1, 2, 7} {3, 4, 8} {5, 6}

N N Y Y Y N N Y

For feature “*department*”:

department

{1,2,3,4,5,6,7,8}

CMPE EE

{3, 5, 6, 8} {1, 2, 4, 7}

Y N Y N N N Y Y

For feature “*size*”:

size

{1,2,3,4,5,6,7,8}

small medium large

{2, 7} {3, 5, 8} {1, 4, 6}

N Y Y N N N Y Y

Thus, we choose the feature “*status*” to split the tree, since it has the lowest entropy. Thus, the tree is:

status

faculty staff student

Step 2 (Branch “status=faculty”):

For feature “*floor*”:

status

{1,2,3,4,5,6,7,8}

faculty staff student

{1, 3, 6} {2, 5, 8} {4, 7}

floor

three four five

{1} {3} {6}

N Y Y

Since no ambiguity remains, Entropy=0

(i.e. )

So, we do not need to test the other features.

Step 2 (Branch “status=staff”):

For feature “*floor*”:

status

{1,2,3,4,5,6,7,8}

faculty staff student

{1, 3, 6} {2, 5, 8} {4, 7}

floor

three four five

{2} {8} {5}

N N N

Since no ambiguity remains, Entropy=0. So, we do not need to test the other features.

Step 2 (Branch “status=student”):

For feature “*floor*”:

status

{1,2,3,4,5,6,7,8}

faculty staff student

{1, 3, 6} {2, 5, 8} {4, 7}

floor

three four

{7} {4}

Y Y

Since no ambiguity remains, Entropy=0. So, we do not need to test the other features.

Therefore, the tree learnt is as follows:

status

faculty staff student

floor floor floor

three four five three four five three four

N Y Y N N N Y Y

**Testing:**

For each data, beginning from the root of the tree, we will follow the tree and arrive at an output. So, we obtain the following results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | Status | Floor | Department | Size | Recycling Bin?  (correct output) | Recycling Bin?  (test output) |
| 1 | student | three | CMPE | small | Yes | Yes |
| 2 | staff | five | CMPE | medium | No | No |
| 3 | faculty | four | CMPE | medium | No | Yes |
| 4 | student | four | EE | small | Yes | Yes |

As we can see, 3 out of 4 outputs are correct. So, the success of the system is 75%.

1. There are four features and two classes (Yes, No).

**Training:**

We learn the prior and likelihood probabilities:

P(class=yes) = 4/8

P(class=no) = 4/8

P(status=faculty | class=yes) = 2/4 P(status=faculty | class=no) = 1/4

P(status=staff | class=yes) = 0/4 P(status=staff | class=no) = 3/4

P(status=student | class=yes) = 2/4 P(status=student | class=no) = 0/4

P(floor=three | class=yes) = 1/4 P(floor=three | class=no) = 2/4

P(floor=four | class=yes) = 2/4 P(floor=four | class=no) = 1/4

P(floor=five | class=yes) = 1/4 P(floor=five | class=no) = 1/4

P(department=CMPE | class=yes) = 2/4 P(department=CMPE | class=no) = 2/4

P(department=EE | class=yes) = 2/4 P(department=EE | class=no) = 2/4

P(size=small | class=yes) = 1/4 P(size=small | class=no) = 1/4

P(size=medium | class=yes) = 1/4 P(size=medium | class=no) = 2/4

P(size=large | class=yes) = 2/4 P(size=large | class=no) = 1/4

**Testing:**

*Test data 1:*

P(class=yes | status=student, floor=three, department=CMPE, size=small)

= P(class=yes) \* P(status=student | class=yes) \* P(floor=three | class=yes) \*

P(department=CMPE | class=yes) \* P(size=small | class=yes)

= 4/8 \* 2/4 \* 1/4 \* 2/4 \* 1/4

= 16/2048

P(class=no | status=student, floor=three, department=CMPE, size=small)

= P(class=no) \* P(status=student | class=no) \* P(floor=three | class=no) \*

P(department=CMPE | class=no) \* P(size=small | class=no)

= 4/8 \* 0/4 \* 2/4 \* 2/4 \* 1/4

= 0/2048

Since P(class=yes | testdata1) > P(class=no | testdata1), we decide answer “Yes” for this data.

*Test data 2:*

P(class=yes | status=staff, floor=five, department=CMPE, size=medium)

= P(class=yes) \* P(status=staff | class=yes) \* P(floor=five | class=yes) \*

P(department=CMPE | class=yes) \* P(size=medium | class=yes)

= 4/8 \* 0/4 \* 1/4 \* 2/4 \* 1/4

= 0/2048

P(class=no | status=staff, floor=five, department=CMPE, size=medium)

= P(class=no) \* P(status=staff | class=no) \* P(floor=five | class=no) \*

P(department=CMPE | class=no) \* P(size=medium | class=no)

= 4/8 \* 3/4 \* 1/4 \* 2/4 \* 2/4

= 48/2048

Since P(class=no | testdata2) > P(class=yes | testdata2), we decide answer “No” for this data.

*Test data 3:*

P(class=yes | status=faculty, floor=four, department=CMPE, size=medium)

= P(class=yes) \* P(status=faculty | class=yes) \* P(floor=four | class=yes) \*

P(department=CMPE | class=yes) \* P(size=medium | class=yes)

= 4/8 \* 2/4 \* 2/4 \* 2/4 \* 1/4

= 32/2048

P(class=no | status=faculty, floor=four, department=CMPE, size=medium)

= P(class=no) \* P(status=faculty | class=no) \* P(floor=four | class=no) \*

P(department=CMPE | class=no) \* P(size=medium | class=no)

= 4/8 \* 1/4 \* 1/4 \* 2/4 \* 2/4

= 16/2048

Since P(class=yes | testdata3) > P(class=no | testdata3), we decide answer “Yes” for this data.

*Test data 4:*

P(class=yes | status=student, floor=four, department=EE, size=small)

= P(class=yes) \* P(status=student | class=yes) \* P(floor=four | class=yes) \*

P(department=EE | class=yes) \* P(size=small | class=yes)

= 4/8 \* 2/4 \* 2/4 \* 2/4 \* 1/4

= 32/2048

P(class=no | status=student, floor=four, department=EE, size=small)

= P(class=no) \* P(status=student | class=no) \* P(floor=four | class=no) \*

P(department=EE | class=no) \* P(size=small | class=no)

= 4/8 \* 0/4 \* 1/4 \* 2/4 \* 1/4

= 0/2048

Since P(class=yes | testdata4) > P(class=no | testdata4), we decide answer “Yes” for this data.

So, we arrive at the following results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | Status | Floor | Department | Size | Recycling Bin?  (correct output) | Recycling Bin?  (test output) |
| 1 | student | three | CMPE | small | Yes | Yes |
| 2 | staff | five | CMPE | medium | No | No |
| 3 | faculty | four | CMPE | medium | No | Yes |
| 4 | student | four | EE | small | Yes | Yes |

As we can see, 3 out of 4 outputs are correct. So, the success of the system is 75%.

*Note*: The posterior probability in some of the test data is zero, since the likelihood probability P(feature=value|class)=0 for a feature. This occurs if there is no data in the training set with the corresponding feature value for that class. Normally, in such a case, a technique called “smoothing” is applied to prevent the final posterior probability being zero.