

# CSC421: Artificial Intelligence

## Assignment 4

Rafael Solorzano

June 27, 2018

## 1 Introduction

In this assignment logic programming using logpy as well as a standard python is performed for a family tree logic. Further. Hidden Markov Models in python using hmmlearn is performed, to examine different states of healthy or unhealthy as well as moves performed according to this hidden states.

## 2 Logic Programming

### 2.1 logpy

Logic programming to determine parents, grandparents and children of different Star Wars characters was performed using logpy. First, the parent relationship is created as facts in which we state ("Parent", "Children") the relationships created for this problem is shown in Figure 1. Once the relationships have been created we can query direct results from them, we query "Who is the parent of Luke Skywalker" as well as "Who are the children of Darth Vader" the code for this queries is shown in Figure 2. Finally, in order to query "Who is the grandparent of Kylo Ren?" we first create a an auxiliary grandparent function, and then we query using this new function, the code for this is shown in Figure 3. The results of all these functions is shown in Figure 4.

```
from kanren import run, Relation, facts, var, conde
parent = Relation()
facts (parent, ( "Darth Vader","Luke Skywalker"),
              ("Darth Vader", "Leia Organa"),
              ("Leia Organa","Kylo Ren"),
              ("Han Solo", "Kylo Ren"))
```

Figure 1: Facts created for the parent relationship using logpy..

```

x=var()

#Query ONE parent of Luke Skywalker
print("The parent of Luke Skywalker is:")
print(run(1,x,parent(x,"Luke Skywalker")))

#Query TWO children of Darth Vader
#If we do 1, only the last one is returned
print("The children of Darth Vader are:")
print(run(2,x,parent("Darth Vader",x)))

```

Figure 2: Query for parent of Luke and children of Darth Vader.

```

#Help, function to return
#grandparent of
def grandparent(x,z):
    y = var()
    return conde((parent(x,y),parent(y,z)))

print("The grandparent of Kylo Ren is:")
print(run(1,x,grandparent(x,'Kylo Ren')))

```

Figure 3: Grandparent function, and query for grandparent of Kylo Ren.

```

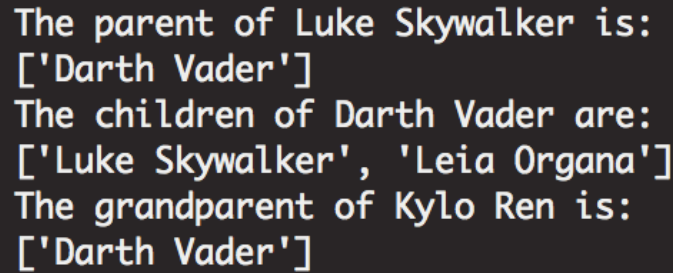
[rafa@Rafaels-MBP : ~/Documents/Git
The parent of Luke Skywalker is:
('Darth Vader',)
The children of Darth Vader are:
('Leia Organa', 'Luke Skywalker')
The grandparent of Kylo Ren is:
('Darth Vader',)

```

Figure 4: Result of all queries performed in Figures 2 and 3.

## 2.2 Standard Python

The same problem with the same queries is also performed using Standard Python. We use a dictionary of lists to establish the facts, this is created in the form "Parent" : ["Child 1" ... "Child n"]. Storing the facts this way, allows us to query for direct children, as well as using a function we can query for every parent of a child, by checking if a child is listed under a parent. Finally, in order to perform the grandparent check, we get the parents of the grandson first, further we get the parents of the parents of the grandson, in the same manner as in parent function. The entire code using Standard Python is shown in Figure 6, and its result is shown in Figure 5.



```
The parent of Luke Skywalker is:
['Darth Vader']
The children of Darth Vader are:
['Luke Skywalker', 'Leia Organa']
The grandparent of Kylo Ren is:
['Darth Vader']
```

Figure 5: Results from the Standard Python Implementation.

Contrasting the two methods, the logic programming allows for easier and faster queries as well as more robust results. Using logpy or Prolog allow for easy and robust logic programming, with very strong query processing. Whereas, with standard python using dictionaries we have to define every single function of what we want to achieve, and provide precise results of how to do this. For this particular problem as it is very small set, doing it wit logpy yielded with a lot less code, and only one extra function being needed, whereas for standard Python we required a separate function for each problem that we later can call to get the results.

```
from collections import defaultdict

#Create a dictionary of lists for parent – children relation
#Each key is the parent, the list is the children
family_parents = defaultdict(list)

#Add children of Darth Vader
family_parents["Darth Vader"].append("Luke Skywalker")

#Add Children of Darth Vader
family_parents["Darth Vader"].append("Leia Organa")

#Add Children of Leia
family_parents["Leia Organa"].append("Kylo Ren")

#Add Children of Han
family_parents["Han Solo"].append("Kylo Ren")

def get_parents(name):
    parents = []
    for k in family_parents:
        if name in family_parents[k]:
            parents.append(k)
    return parents

def get_children(name):
    return family_parents[name]

def get_grandparent(name):
    parents = get_parents(name)
    grandparents = []
    for i in parents:
        g = get_parents(i)
        for j in g:
            if j not in grandparents:
                grandparents.append(j)
    return grandparents

print("The parent of Luke Skywalker is:")
luke_parent = get_parents("Luke Skywalker")
print(luke_parent)
print("The children of Darth Vader are:")
darth_children = get_children("Darth Vader")
print(darth_children)
print("The grandparent of Kylo Ren is:")
kylo_grandparent = get_grandparent("Kylo Ren")
print(kylo_grandparent)
```

Figure 6: Standard Python implementation of section 2.1.

### 3 Hidden Markov Models

In order to generate random sequence of healthy, and injured an approach similar to the card deck in Monopoly is used. A list of different states labeled 1 for Healthy and 0 for injured is created. For example the healthy matrix would contain 7 1's and 3 0's to represent 0.7 probability of staying healthy and 0.3 of going injured. Further, to generate random sequences of dribble, pass, or shoot given the current state is doing in the same manner as for healthy injured with the difference that 2 represents dribble, 3 represents pass, and 4 represents shoot. The code for the generation of health and injured sequence is shown in Figure 7, whereas the code for the generation of moves sequence is shown in Figure 8. Finally, a sample printout of each function is shown in Figure 9.

```
def generate_random_health_injured_sequence(total_states=100):
    sequence=[]
    state = 1
    healthy_matrix = [1,1,1,1,1,1,1,0,0,0]
    unhealthy_matrix = [1,1,1,1,1,0,0,0,0,0]
    random.shuffle(healthy_matrix)
    random.shuffle(unhealthy_matrix)
    #Always start healthy
    sequence.append(state)
    for i in range(total_states-1):
        if state == 1:
            state = np.random.choice(healthy_matrix)
        elif state ==0:
            state = np.random.choice(unhealthy_matrix)
        sequence.append(state)
    return sequence
```

Figure 7: Sequence of random Healthy and Injured generation.

```
def generate_random_move_sequence(total_states=100):
    hidden = generate_random_health_injured_sequence(total_states)
    sequence = []
    state=1
    healthy_matrix = [2,2,3,4,4,4,4,4,4,4]
    unhealthy_matrix = [2,2,2,3,3,3,3,3,3,4]
    random.shuffle(healthy_matrix)
    random.shuffle(unhealthy_matrix)
    move = np.random.choice(healthy_matrix)
    sequence.append(move)
    for i in range(total_states-1):
        #skip first one as it has been already appended
        state = hidden[i+1]
        if state == 1:
            move = np.random.choice(healthy_matrix)
        elif state == 0:
            move = np.random.choice(unhealthy_matrix)
        sequence.append(move)
    return hidden, sequence
```

Figure 8: Sequence of random moves sequence generation.

```

rafa@Rafael's-MBP : ~/Documents/GitHub/CSC421/A4 python3 markov.py
Generating 300 Random Healthy Injured Sequence:
[1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
Generating 300 random move sequence
as well as its hidden Healthy-Injured Sequence
Hidden Sequence:

[1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0,
0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
Move Sequence:

[3, 2, 3, 4, 2, 3, 2, 2, 2, 2, 4, 4, 4, 2, 4, 4, 3, 2, 4, 4, 3, 3, 2, 3, 4, 4, 4,
, 2, 3, 4, 3, 4, 4, 2, 3, 2, 4, 3, 4, 4, 2, 4, 4, 3, 4, 4, 4, 4, 3, 2, 3, 4, 4,
2, 3, 4, 3, 4, 4, 3, 4, 4, 4, 4, 4, 2, 2, 3, 4, 3, 4, 2, 3, 3, 4, 4, 4, 2, 3, 4,
4, 3, 3, 3, 4, 3, 3, 3, 3, 4, 2, 4, 3, 3, 2, 4, 4, 4, 2, 2, 4, 3, 3, 3, 3, 2, 4,
, 4, 4, 4, 2, 4, 2, 4, 4, 4, 3, 4, 4, 4, 3, 3, 4, 2, 3, 3, 3, 2, 3, 2, 4, 4, 3,
2, 4, 4, 2, 4, 4, 4, 2, 3, 4, 4, 3, 3, 3, 3, 4, 4, 2, 4, 3, 4, 3, 3, 3, 4, 3, 4,
2, 4, 4, 3, 2, 2, 4, 4, 4, 4, 4, 3, 2, 4, 4, 2, 4, 2, 4, 2, 3, 3, 3, 4, 4, 4, 2,
, 2, 2, 4, 4, 2, 2, 3, 2, 3, 4, 4, 2, 4, 4, 4, 3, 3, 4, 4, 2, 3, 4, 2, 3, 2, 4,

```

Figure 9: A sample sequence generation result.

After we have created the sequence of observations, we create a Hidden Markov Model using `hmmlearn` with Multinomial Hidden Markov Model. First, we create the model object to which we assign the initial probabilities, this are 1.0 and 0.0 as the initial state is always healthy not chosen by random. Second, we define the transition matrix from this states. Third, the emission probability of the different moves is specified. The code for the creation of this model is shown in Figure 10.

```
#----- HMM -----

model = hmm.MultinomialHMM(n_components=2,n_iter=100)

#The Initial state, as it is assumed that the player starts healthy
model.startprob_ = np.array([1.0,0.0])

#Represents the change from Healthy to Injured
model.transmat_ = np.array([[0.3,0.7],
                             [0.5,0.5]])

#Represents how likely each possible move of
#dribble, pass, or shot is given its condition
#of either healthy or injured
model.emissionprob_ = np.array([[0.2,0.1,0.7],
                                 [0.3,0.6,0.1]])
```

Figure 10: Creation of the Multinomial HMM model.

Once we have the model we can fit the generated sequence of moves into our model. After fitting we can sample the model to generate a random sample from it. The code of fit and the printout of the sample results is shown in Figure 11 and 12 respectively.

```
#Append both observations
#and store their lengths
X=np.append(seq_2,hidden)
lengths=[len(seq_2),len(hidden)]

#Fit the observations
model.fit(X.reshape(-1,1),lengths)
#Get 300 samples from the model
print(model.sample(300))
```

Figure 11: Code to fit the sequences into the model.

[illegible]

Figure 12: The random generation of samples from the code in Figure 11.

The predict function of the HMM model will predict the sequence of hidden states, that was used to generate our sequence of observations. It was decided to run this predict function 10 times with different observations and report the results in order to provide how many errors the predict function produced in different runs. The code that does the predict is shown in Figure 13. The error results obtained for each run were:

- 97
- 108
- 173
- 194
- 120
- 107
- 177
- 111
- 202
- 117



```
#Predict the hidden states
prediction = model.predict(np.array(seq_2).reshape(-1,1))
print(prediction)

print("Errors generated by the prediction: ")
correct = np.sum(np.array(hidden) == prediction)
print(300-correct)
print("Correct results:")
print(correct)
```

Figure 13: The predict function.