Intelligent Systems 2020: 5th practical assignment

Logical Agents

Your name: Seeun Park

Your VUnetID: spk760

If you do not provide your name and VUnetID we will not accept your submission.

Learning objectives

At the end of this exercise you should be able to work with some basic fuzzy concepts, and implement and evaluate a simple probabilistic approach to playing Schnapsen.

Preliminaries

In this worksheet we will build a Fuzzy Logic knowledge-base and implement a simple probabilistic strategy for an agent to play Schnapsen.

Practicalities

Follow this Notebook step-by-step.

Of course, you can do the exercises in any Programming Editor of your liking. But you do not have to. Feel free to simply write code in the Notebook. Please use your studentID+Assignment5.ipynb as the name of the Notebook.

Note: unlike the courses dedicated to programming we will not evaluate the style of the programs. But we will, however, test your programs on other data that we provide, and your program should give the correct output to the test-data as well.

As was mentioned, the assignment is graded as pass/fail. To pass you need to have either a full working code or an explanation of what you tried and what didn't work for the tasks that you were unable to complete (you can use multi-line comments or a text cell).

Calculating Fuzzyvalues for a Formula (KnowledgeBase)

The first task is to implement part of the algorithm to calculate a fuzzy value for a complex formula:

> f(¬A) = 1 - f(A)
> f(A v B) = max(f(A);f(B))
> f(A & B) = min(f(A);f(B))
> f(A -> B) = min(1;1 – f(A) + f(B))

We will do this in a very simplistic way, which should still give you an additional idea on how this can be done. The main restriction is that we will only calculate the value of formulas in Clause Normal Form. This means we do not have to implement the rule for implication, and we do have a very simple structure of formulas:

> (L11 v L12 v L13 v .... v L1n)
> & (L21 v L22 v L23 v .... v L2n)
> & ...

where each of the L's is a Literal, or in other words an Atom, or a negation of an Atom. So, all that needs to be done is to be able to:

1) assign a fuzzy value V to an atom, or (1 - V) value in case it is a negation.
2) calculate the fuzzy value of a disjunction of values for Literals, and
3) calculate the fuzzy value of a conjuction of values for disjuctions.

There are two steps that need to be taken and they will have to be done at two different places in the fuzzykb.py file. Open this file in a Python editor of your choice. First, we will have to define the value of a negated atom.

Task 1A: Fuzzy negation

Go to the definition of the NegFuzzySymbol class. Here in line 52, given a fuzzy value "value", the value of a negation has to be defined. Replace the line "???" with the correct code, and add the snippet to the following cell:

1 - value

Task 1B: Conjunction and Disjunction

The task is now to: 1) define the fuzzy value of Clause based on the values of the symbols (you get those by calling symbol.value() for all symbols in a clause)
2) define the fuzzy value of the Knowlegde base (i.e. the conjunction of all the clauses)

Go to the definition of the fuzzyvalue() function. Here in line 120 replace the lines "???" with the correct code, and add the snippet you wrote to the following cell:

minvalue = [] for clause in self._clauses: clausevalue = [] for symbol in clause: clausevalue.append(symbol.value()) minvalue.append(max(clausevalue)) return min(minvalue)

Once you have succesfully finished the implementations of these methods, you should be able to run the following code.

In [1]:
```python
import sys
from bots.kbbot.fuzzykb import fuzzyKB, FuzzySymbol

# Define our symbols
A = FuzzySymbol('A',0.3)
B = FuzzySymbol('B',0.8)
C = FuzzySymbol('C',0.6)

# Create a new knowledge base
kb = fuzzyKB()
```

```
# Add clauses
kb.add_clause(A, B, C)
kb.add_clause(~A, B)
kb.add_clause(~B, C)
kb.add_clause(B, ~C)

# Calculate the fuzzy value of the knowledge base (the conjunction of all the clauses).
print("f(kb):", kb.fuzzyvalue())
```

```
f(kb): 0.6
```

Task 2

The next task is to model the properties of Schnapsen cards in a simple Fuzzy Representation, with the following propositions:

> HC = HighCard
>
> VHC = VeryHighCard
>
> Avg = AverageCard
>
> Exp = ExpensiveCard
>
> Chp = CheapCard

Now let us build a fuzzy knowledge-base with values for a Queen (there is not one correct answer, but try to model this as close as possible to the true value and cost of a Queen in Schnapsen.

a) First, let us reset the knowledge base, and define our fuzzy variables for a Queen:

In [2]:
```
kb = fuzzyKB()
# Define our symbols
HC = FuzzySymbol('A',0.2)
VHC = FuzzySymbol('A',0.1)
Avg = FuzzySymbol('A',0.4)
Low = FuzzySymbol('A',0.8)
Chp = FuzzySymbol('A',0.4)
```

b) Let us define now a set of cards, and check to what degree our queen is a member of this set. We define the set of cards as a set that are

> 1. are either low, high or very high
> 2. neither very high, nore average
> 3. either not low or cheap, AND
> 4. neither cheap, nor high

In [3]:
```
#Add clauses
kb.add_clause(Low, HC, VHC)
kb.add_clause(~VHC, ~Avg)
kb.add_clause(~Low, Chp)
kb.add_clause(~Chp, ~HC)
```

Now run the script and report the output in the cell after the code.

In [4]:
```
# Calculate the fuzzy value of the knowledge base (the conjunction of all the clauses).
print("f(kb):", kb.fuzzyvalue())
```

```
f(kb): 0.4
```

Explain in your own words what does this output mean:

The value 0.4 shows how the card will respond to the queen.

A Probability-based Bot.

Now it is time to move to a real rational agent that plays Schnapsen, and to implement a probabilistic bot. The idea of this bot is described on the Canvas page for this assignment and in the probability.py file.

Tasks 3:

You will have to finish the implementation of a bot that uses probabilistic reasoning to determine its next move. All you have to do is fill in the missing code at the lines marked with "???". At these spots, we explain what you will have to do, but we strongly recommend that you also have a careful look at the entire bot, and the documentation of the code to get the overall idea.

But first, let us not forget to initialise it:

In [5]:
```python
import sys, random

from api import State, engine, util
```

Now you need to start coding. In the directory /bots, you will have to make a subdirectory called probability, and copy the provided file probability.py into this new directory.

The next step is to finish the implemenation of this file. Open it in your favourite Python Editor and fill in the gaps.

First, in line 101, you will need to implement the probability that the opponent has a problemCard.

In line 107, you will have to update the maximal probability value and the chosen move accordingly.

Now we can run a game between rand and your new bot, to check whether everything works fine.

In [12]:
```python
# Choose your first player
player1 = "rand"
player2 = "probabilitybully"
# Decide in which phase you want to start the game.
startphase = 1
# Decide whether you want verbose output or not
verbose=True

#And here you run a game on the engine.
engine.play(util.load_player(player1),util.load_player(player2), state=State.generate(p
```

```
player1: <bots.rand.rand.Bot object at 0x00000163F6B842B0>
player2: <bots.probabilitybully.probabilitybully.Bot object at 0x00000163F6B84040>
*   Player 2 plays: AH
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 0, pending: 0
The trump suit is: H
Player 1's hand: KD JD KH AS 10S
Player 2's hand: 10C KC AH JH QS
```

```
There are 10 cards in the stock
Player 2 has played card: A of H

*   Player 1 plays: 10S
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 21, pending: 0
The trump suit is: H
Player 1's hand: JC KD JD KH AS
Player 2's hand: 10C KC JH KS QS
There are 8 cards in the stock

*   Player 2 plays: 10C
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 21, pending: 0
The trump suit is: H
Player 1's hand: JC KD JD KH AS
Player 2's hand: 10C KC JH KS QS
There are 8 cards in the stock
Player 2 has played card: 10 of C

*   Player 1 plays: JD
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 33, pending: 0
The trump suit is: H
Player 1's hand: AC JC KD KH AS
Player 2's hand: KC QC JH KS QS
There are 6 cards in the stock

*   Player 2 plays: KS
*   Player 2 melds a marriage between KS and QS
The game is in phase: 1
Player 1's points: 0, pending: 0
Player 2's points: 33, pending: 20
The trump suit is: H
Player 1's hand: AC JC KD KH AS
Player 2's hand: KC QC JH KS QS
There are 6 cards in the stock
Player 2 has played card: K of S

*   Player 1 plays: AS
The game is in phase: 1
Player 1's points: 15, pending: 0
Player 2's points: 33, pending: 20
The trump suit is: H
Player 1's hand: AC JC KD QD KH
Player 2's hand: KC QC 10H JH QS
There are 4 cards in the stock

*   Player 1 plays: QD
The game is in phase: 1
Player 1's points: 15, pending: 0
Player 2's points: 33, pending: 20
The trump suit is: H
Player 1's hand: AC JC KD QD KH
Player 2's hand: KC QC 10H JH QS
There are 4 cards in the stock
Player 1 has played card: Q of D

*   Player 2 plays: 10H
The game is in phase: 1
Player 1's points: 15, pending: 0
Player 2's points: 66, pending: 0
```

```
The trump suit is: H
Player 1's hand: AC JC KD KH JS
Player 2's hand: KC QC 10D JH QS
There are 2 cards in the stock

Game finished. Player 2 has won, receiving 2 points.
```

Out[12]:
```
(2, 2)
```

Provide in the following cell the code that you have written to make the probabilistic bot work:

probability = 1 - ((u-pc)/u) * (((u-1) - pc)/u-1) * (((u-2) - pc)/u-2) * (((u-3) - pc)/u-3) * (((u-4) - pc)/u-4) # All that is left to do is to check whether the new probability is higher than the earlier value # (to calculate the maximum). If it is, we set probability to be the new maximal probability, and the # current move our new chosen move. if (move[0] is not None and probability > maxProbability): maxProbability = probability chosen_move = move #print("Played MaxProbability Card ", maxProbability, "played", chosen_move) return chosen_move

An alternative might be to run games from the command-line in your terminal. To run a game between rand and probability just run:

> python play.py -1 rand -2 probability

or check out line python play.py -h for more details. Alternatively you can run a tournament with

> python tournament.py -p rand,probability
> (and again check for help with: python tournament.py -h).

Of course, you can also run the tournament in this notebook with the code that we provided in the last assignment if you prefer it that way.

Task 4

Run a tournament against some of the other bots, e.g. rand, kbbot or rdeep. Describe your findings in the next cell.

Playing 10 games: Played 1 out of 10 games (10%): [1, 0] Played 2 out of 10 games (20%): [2, 0] Played 3 out of 10 games (30%): [3, 0] Played 4 out of 10 games (40%): [3, 1] Played 5 out of 10 games (50%): [3, 3] Played 6 out of 10 games (60%): [3, 6] Played 7 out of 10 games (70%): [3, 7] Played 8 out of 10 games (80%): [3, 8] Played 9 out of 10 games (90%): [5, 8] Played 10 out of 10 games (100%): [5, 9] Results: bot : 5 points bot : 9 points -- rand loses against probability -- Playing 10 games: Played 1 out of 10 games (10%): [2, 0] Played 2 out of 10 games (20%): [4, 0] Played 3 out of 10 games (30%): [4, 1] Played 4 out of 10 games (40%): [6, 1] Played 5 out of 10 games (50%): [8, 1] Played 6 out of 10 games (60%): [9, 1] Played 7 out of 10 games (70%): [9, 2] Played 8 out of 10 games (80%): [9, 3] Played 9 out of 10 games (90%): [11, 3] Played 10 out of 10 games (100%): [11, 5] Results: bot : 11 points bot : 5 points -- rdeep won against probability -- Playing 10 games: Played 1 out of 10 games (10%): [3, 0] Played 2 out of 10 games (20%): [3, 2] Played 3 out of 10 games (30%): [3, 5] Played 4 out of 10 games (40%): [6, 5] Played 5 out of 10 games (50%): [6, 8] Played 6 out of 10 games (60%): [6, 11] Played 7 out of 10 games (70%): [7, 11] Played 8 out of 10 games (80%): [7, 14] Played 9 out of 10 games (90%): [7, 15] Played 10 out of 10 games (100%): [7, 16] Results: bot : 7 points bot : 16 points -- kbbot lost against probability 8 times. --

Utility

Unless we are very much mistaken, your new probability bot will not perform very well. One reason for this is that our probabilistic strategy has a serious flaw: the Aces and 10s have a high probability of not having a higher card of the same suit, so that our strategy will pick valuable cards in phase 1. This is a high-gain, but also a high-risk strategy, as a reasonably good opponent would trump those cards and win valuable points.

One possible solution is to combine the probability of a card being easily beaten with the costs it takes to loose such a card. We do this by introducing a notion of utily, which simply devides the probability of being good by the costs of a potential loss of the played card.

Task 5

Now you need to do a bit of coding again. In the directory /bots, you will have to make a subdirectory called utility, and copy the provided utility.py file into this new directory.

The next step is to finish the implemenation of this file. Open it in your favourite Python Editor and fill in the gaps.

First, you will need to copy the solutions from probability.py to utility.py in lines 97 and 115, but now also a single line in 108.

Provide the written lines of code in the following cell.

probability = 1 - ((u-pc)/u) * (((u-1) - pc)/u-1) * (((u-2) - pc)/u-2) * (((u-3) - pc)/u-3) * (((u-4) - pc)/u-4) # Now we have the probability, but still want to normalise it with the costs of losing the card. # E.g. playing a 10 is a higher risk of loosing points as compared with playing a Jack. # Let score be an array with values for the various cards. The modulo method will assign a rank to each # card, and each of these ranks is given a value from the score array. score = [11, 10, 4, 3, 2] if (move[0] is not None): rank = move[0] % 5 points = score[rank] utility = probability / points else: utility = 0.0 # Now we check if the utility of the current option (move) is higher than the highest of the previous # options. If so, we choose this as our new maximal utility, and this move the chosen move. if (move[0] is not None and utility > maxUtility): maxUtility = utility chosen_move = move # print("Played MaxUtility Move: ", maxUtility, "played", chosen_move) return chosen_move

Task 6

Now it is time to evaluate your two new bots: utility and probability. Run a number of tournaments in the next cell. Summarise what you did, and what the results were.

Playing 10 games: Played 1 out of 10 games (10%): [1, 0] Played 2 out of 10 games (20%): [1, 1] Played 3 out of 10 games (30%): [1, 2] Played 4 out of 10 games (40%): [1, 4] Played 5 out of 10 games (50%): [2, 4] Played 6 out of 10 games (60%): [5, 4] Played 7 out of 10 games (70%): [5, 6] Played 8 out of 10 games (80%): [5, 7] Played 9 out of 10 games (90%): [7, 7] Played 10 out of 10 games (100%): [7, 10] Results: bot : 7 points bot : 10 points -- I ran 10 games for probability against utility. Probability seems to have the winning hand, since two of the games tied and two games were lost against utility. -- Playing 10 games: Played 1 out of 10 games (10%): [1, 0] Played 2 out of 10 games (20%): [1, 1] Played 3 out of 10 games (30%): [3, 1] Played 4 out of 10 games (40%): [3, 3] Played 5 out of 10 games (50%): [3, 4] Played 6 out of 10 games (60%): [3, 5] Played 7 out of 10 games (70%): [4, 5] Played 8 out of 10 games (80%): [5, 5] Played 9 out of 10 games (90%): [5, 7] Played 10 out of 10 games (100%): [6, 7] Results: bot : 6 points bot : 7 points -- Utility won 5 times and tied with rand 2 times, thus lost to rand by 3 times. Through this we can hypothesize that bully will be able to win utility or tie with utility because bully has a better strategy than rand. -- Playing 10 games: Played 1 out of 10 games (10%): [2, 0] Played 2 out of 10 games (20%): [5, 0] Played 3 out of 10 games (30%): [5, 1] Played 4 out of 10 games (40%): [5, 2] Played 5 out of 10 games (50%): [8, 2] Played 6 out of 10 games (60%): [9, 2] Played 7 out of 10 games (70%): [9, 5] Played 8 out of 10 games (80%): [11, 5] Played 9 out of 10 games (90%): [11, 6] Played 10 out of 10 games (100%): [14, 6] Results: bot : 14 points bot : 6 points -- Bully won all 10 games against utility. Therefore, we can assume that utility has no chance against rdeep. -- Playing 10 games: Played 1 out of 10 games (10%): [2, 0] Played 2 out of 10 games (20%): [2, 1] Played 3 out of 10 games (30%): [4, 1] Played 4 out of 10 games (40%): [7, 1] Played 5 out of 10 games (50%): [9, 1] Played 6 out of 10 games (60%): [9, 2] Played 7 out of 10 games (70%): [11, 2] Played 8 out of 10 games (80%): [14, 2] Played 9 out of 10 games (90%): [16, 2] Played 10 out of 10 games (100%): [19, 2] Results: bot : 19 points bot : 2 points -- As assumed from when running the games with bully against utility, the results clearly display that rdeep won against utility. -- Playing 10 games: Played 1 out of 10 games (10%): [0, 1] Played 2 out of 10 games (20%): [0, 3] Played 3 out of 10 games (30%): [0, 6] Played 4 out of 10 games (40%): [0, 8] Played 5 out of 10 games (50%): [2, 8] Played 6 out of 10 games (60%): [2, 9] Played 7 out of 10 games (70%): [2, 11] Played 8 out of 10 games (80%): [5, 11] Played 9 out of 10 games (90%): [6, 11] Played 10 out of 10 games (100%): [6, 14] Results: bot : 6 points bot : 14 points -- kbbot on the other hand lost against all the games to utility showing that suggests that its strategy is weaker than rand. -- Conclusion: When looking at the games that were run in task 4 probability won against rand therefore, in comparison to utility utility has a weaker strategy than probability. However, probability lost against kbbot 2 times whereas utility won against kbbot for all the games. Therefore, we can see that the strategy can be stronger depending on what type of strategy the opponent uses. Despite the fact that probability wins more often against utility.

Task 7

Again, unless we are much mistaken, the results will be disappointing for both bots. The final task is to change one of the bots and try to improve it. In the next cell, describe what changes you attempted to make, add the code that you have changed or added, and report on the tournaments you ran (what did you do, and what were the results).

Note: do not despair, this is not a simple task and chances are that you will not manage to improve performance much. But still, describe what you have tried.

Results: player1: player2: * Player 2 plays: AH The game is in phase: 1 Player 1's points: 0, pending: 0 Player 2's points: 0, pending: 0 The trump suit is: H Player 1's hand: KD JD KH AS 10S Player 2's hand: 10C KC AH JH QS There are 10 cards in the stock Player 2 has played card: A of H * Player 1 plays: 10S The game is in phase: 1 Player 1's points: 0, pending: 0 Player 2's points: 21, pending: 0 The trump suit is: H Player 1's hand: JC KD JD KH AS Player 2's hand: 10C KC JH KS QS There are 8 cards in the stock * Player 2 plays: 10C The game is in phase: 1 Player 1's points: 0, pending: 0 Player 2's points: 21, pending: 0 The trump suit is: H Player 1's hand: JC KD JD KH AS Player 2's hand: 10C KC JH KS QS There are 8 cards in the stock Player 2 has played card: 10 of C * Player 1 plays: JD The game is in phase: 1 Player 1's points: 0, pending: 0 Player 2's points: 33, pending: 0 The trump suit is: H Player 1's hand: AC JC KD KH AS Player 2's hand: KC QC JH KS QS There are 6 cards in the stock * Player 2 plays: KS * Player 2 melds a marriage between KS and QS The game is in phase: 1 Player 1's points: 0, pending: 0 Player 2's points: 33, pending: 20 The trump suit is: H Player 1's hand: AC JC KD KH AS Player 2's hand: KC QC JH KS QS There are 6 cards in the stock Player 2 has played card: K of S * Player 1 plays: AS The game is in phase: 1 Player 1's points: 15, pending: 0 Player 2's points: 33, pending: 20 The trump suit is: H Player 1's hand: AC JC KD QD KH Player 2's hand: KC QC 10H JH QS There are 4 cards in the stock * Player 1 plays: QD The game is in phase: 1 Player 1's points: 15, pending: 0 Player 2's points: 33, pending: 20 The trump suit is: H Player 1's hand: AC JC KD QD KH Player 2's hand: KC QC 10H JH QS There are 4 cards in the stock Player 1 has played card: Q of D * Player 2 plays: 10H The game is in phase: 1 Player 1's points: 15, pending: 0 Player 2's points: 66, pending: 0 The trump suit is: H Player 1's hand: AC JC KD KH JS Player 2's hand: KC QC 10D JH QS There are 2 cards in the stock Game finished. Player 2 has won, receiving 2 points. (2, 2)I implied the strategy to get the move with the highest rank available from all suits and replaced it with the random choice being played. It led to winning rand however, losing against rdeep and bully. class Bot: __randomize = True def __init__(self,

```
randomize=True, depth=5): self.__max_depth = depth def get_move(self, state): # type: (State) -> tuple[int, int] # All
legal moves moves = state.moves() chosen_move = moves[0] # If it is not my turn, then it does not make sense to
play the Probabilistic Strategy. # In that case, we simply play a random move. This could be done better, but it not #
the purpose of this bot. for index, move in enumerate(moves): if move[0] is not None and move[0] % 5 <=
chosen_move[0] % 5: chosen_move = move return chosen_move
```