Appendix B

June 23, 2021

Maastricht University

Research project

Project report

Explainable AI: Learning Arguments

Appendix B: Notebook

Authors: Jonas Bei, David Pomerenke, Lukas Schreiner, Sepideh Sharbaf

Supervisors: Nico Roos, Pieter Collins

June 23, 2021

This notebook contains literate code for (a) reproducing the examples from Verheij 2017 and (b) showing what theories the different algorithms learn on these case models, as well as on some small artificial case models. Some of the examples are discussed in more detail in the "Qualitative Analysis" section of the report.

This file is provided as PDF for easy reading and archiving purposes, and as an iPython Notebook with interactively executable cells. The notebook needs to be executed in the same folder as the files `logic.py` and `learning.py`, and the steps for the installation of the dependencies need to be followed, as described in the `README.md` file. The output of the cells from their last execution is also visible without installing or executing anything.

[1]:
```
from logic import *
from learning import *
```

1 Legal case models

1.1 Presumption of innocence

(p. 137-138)

"Let `inn` denote that a suspect is innocent, and `gui` that he is guilty. Then the argument `(inn, ¬gui)` is properly presumptive, since `inn` $\not\models$ `gui`. The argument `(inn` $\wedge$ `¬gui, ¬gui)` is non-presumptive, since `inn` $\wedge$ `¬gui` $\models$ gui."

[2]:
```
assert Argument.fromStr('inn -> ¬gui').is_properly_presumptive
assert not Argument.fromStr('inn, ¬gui -> ¬gui').is_properly_presumptive
```

"Presumptive validity and defeasibility are illustrated using a case model. Consider the case model with two cases `inn` ∧ `¬gui` and `¬inn` ∧ `gui` ∧ `evi` with the first case preferred to the second (…). Here evi denotes evidence for the suspect's guilt."

[3]:
```
case_model = CaseModel.fromStr([
    (1, 'inn, ¬gui'),
    (0, '¬inn, gui, evi')
])
```

"Then the properly presumptive argument `(inn, ¬gui)` is presumptively valid with respect to this case model since the conclusion `¬gui` follows in the case `inn` ∧ `¬gui` that is a preferred case of the premise `inn`. The argument is conclusive since there are no other cases implying `inn`."

[4]:
```
argument = Argument.fromStr('inn -> ¬gui')
assert argument.is_presumptively_valid_in(case_model)
assert argument.is_conclusive_in(case_model)
```

"The argument `([], inn)`—in fact a presumption now that its premises are tautologous—is presumptively valid since `inn` follows in the preferred case `inn` ∧ `¬gui`. This shows that the example represents what is called the presumption of innocence, when there is no evidence. This argument is properly defeasible since in the other case of the argument's premises the conclusion does not follow."

[5]:
```
presumption_of_innocence = Argument.fromStr('-> inn')
assert presumption_of_innocence.is_a_presumption
assert presumption_of_innocence.is_presumptively_valid_in(case_model)
assert presumption_of_innocence.is_properly_defeasible_in(case_model)
```

"In fact, the argument `(evi, inn)` is not coherent since there is no case in which both `evi` and `inn` follow."

[6]:
```
assert not Argument.fromStr('evi -> inn').is_coherent_in(case_model)
```

"The argument `(evi, gui)` is presumptively valid, even conclusive."

[7]:
```
argument = Argument.fromStr('evi -> gui')
assert argument.is_presumptively_valid_in(case_model)
assert argument.is_conclusive_in(case_model)
```

"Continuing the example of the case model (…), we find the following. The circumstances `evi` defeat the presumptively valid argument (⊤, `inn`) since `(evi, inn)` is not presumptively valid. In fact, these circumstances are excluding since `(evi, inn)` is not coherent. The circumstances are also rebutting since the argument for the opposite conclusion `(evi, inn)` is presumptively valid.

[8]:
```
circumstances = [Fact.fromStr('evi')]
assert presumption_of_innocence.is_defeated_by_in(circumstances, case_model)
assert presumption_of_innocence.is_excluded_by_in(circumstances, case_model)
assert presumption_of_innocence.is_rebutted_by_in(circumstances, case_model)
```

1.1.1 Learning arguments

[9]:
```
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
inn  ¬gui <~
---

evi  gui <- ¬inn
evi  ¬inn <- gui
gui  ¬inn <- evi
inn <- ¬gui
¬gui <- inn
inn  ¬gui <~
---

evi  gui  inn <~
¬gui <~ inn
¬inn <~ evi
```

[10]:
```
patched_case_model = CaseModel.fromStr([
    (1, 'inn, ¬gui'),
    (1, 'inn, ¬gui'),
    (0, '¬inn, gui, evi')
])
print(Theory.learn_with_naive_search(patched_case_model))
print('---')
print(Theory.learn_with_pruned_search(patched_case_model))
print('---')
print(Theory.learn_with_hero(patched_case_model))
```

```
inn  ¬gui <~
---

evi  gui <- ¬inn
evi  ¬inn <- gui
gui  ¬inn <- evi
inn <- ¬gui
¬gui <- inn
inn  ¬gui <~
---

evi  inn  ¬gui <~
gui  ¬inn <~ evi
```

1.2 Lying witness

"In the cases, there is a witness testimony (wit) that the suspect was at the crime scene (sus). In Case 1, the witness was not misguided (¬mis), in Case 2 he was. In Case 1, the suspect was indeed at the crime scene; in Case 2, the witness was misguided and it is unspecified whether the suspect was at the crime scene or not. In the case model, Case 1 is preferred to Case 2 (…), representing that witnesses are usually not misguided." (p. 139)

[11]:
```
case_model = CaseModel.fromStr([
    (1, 'sus, ¬mis, wit'),
    (0, 'mis, wit')
])
```

"Since Case 1 is a preferred case of wit, the argument (wit, sus) is presumptively valid: the witness testimony provides a presumptively valid argument for the suspect having been at the crime scene. The argument's conclusion can be strengthened to include that the witness was not misguided. Formally, this is expressed by saying that (wit, sus ∧ ¬mis) is a presumptively valid argument." (p. 139)

[12]:
```
assert Argument.fromStr('wit -> sus').is_presumptively_valid_in(case_model)
assert Argument.fromStr('wit -> sus, ¬mis').
 ↪is_presumptively_valid_in(case_model)
```

"When the witness was misguided after all (mis), there are circumstances defeating the argument (wit, sus). This can be seen by considering that Case 2 is the only case in which wit ∧ mis follows, hence is preferred. Since sus does not follow in Case 2, the argument (wit ∧ mis, sus) is not presumptively valid. The misguidedness is not rebutting, hence undercutting since (wit ∧ mis, ¬sus) is not presumptively valid. The misguidedness is excluding since the argument (wit ∧ mis, sus) is not even coherent." (p. 139)

[13]:
```
argument = Argument.fromStr('wit -> sus')
circumstances = [Fact.fromStr('mis')]
assert argument.is_defeated_by_in(circumstances, case_model)
assert argument.is_undercut_by_in(circumstances, case_model)
assert argument.is_excluded_by_in(circumstances, case_model)
```

1.2.1 Learning arguments

[14]:
```
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
wit <-
sus   ¬mis <~
---

sus <- ¬mis
```

```
wit <~
¬mis <~ sus
sus   ¬mis <~
---

mis   sus   wit <~
¬mis <~ sus
```

1.3 Chaining arguments

"Arguments can typically be chained, namely when the conclusion of one is a premise of another. For instance when there is evidence (evi) that a suspect is guilty of a crime (gui), the suspect's guilt can be the basis of punishing the suspect (pun). For both steps there are typical defeating circumstances. The step from the evidence to guilt is blocked when there is a solid alibi (ali), and the step from guilt to punishing is blocked when there are grounds of justification (jus), such as force majeure. (…) In the case model, Case 1 is preferred to Case 2 and Case 3, modeling that the evidence typically leads to guilt and punishing, unless there are grounds for justification (Case 2) or there is an alibi (Case 3). Cases 2 and 3 are preferentially equivalent." (p. 139-140)

[15]:
```
case_model = CaseModel.fromStr([
    (1, 'pun, gui, evi'),
    (0, '¬pun, gui, evi, jus'),
    (0, '¬gui, evi, ali')
])
```

"In this case model, the following arguments are presumptively valid:

- Argument 1 (presumptively valid): (evi, gui)
- Argument 2 (presumptively valid): (gui, pun)
- Argument 3 (presumptively valid): (evi, gui ∧ pun)"

(p. 140)

[16]:
```
argument1 = Argument.fromStr('evi -> gui')
argument2 = Argument.fromStr('gui -> pun')
argument3 = Argument.fromStr('evi -> gui, pun')

assert argument1.is_presumptively_valid_in(case_model)
assert argument2.is_presumptively_valid_in(case_model)
assert argument3.is_presumptively_valid_in(case_model)
```

"The following arguments are not presumptively valid in this case model:

- Argument 4 (not presumptively valid): (evi ∧ ali, gui)
- Argument 5 (not presumptively valid): (gui ∧ jus, pun)"

(p. 141)

[17]:
```
argument4 = Argument.fromStr('evi, ali -> gui')
argument5 = Argument.fromStr('gui, jus -> pun')
assert not argument4.is_presumptively_valid_in(case_model)
```

```
assert not argument5.is_presumptively_valid_in(case_model)
```

"This shows that Arguments 1 and 2 are defeated by circumstances `ali` and `jus`, respectively:" (p. 141)

[18]:
```
assert argument1.is_defeated_by_in([Fact.fromStr('ali')], case_model)
assert argument2.is_defeated_by_in([Fact.fromStr('jus')], case_model)
```

"As expected, chaining the arguments fails under both of these defeating circumstances, as shown by the fact that these two arguments are not presumptively valid:

- Argument 6 (not presumptively valid): `(evi ∧ ali, gui ∧ pun)`
- Argument 7 (not presumptively valid): `(gui ∧ jus, gui ∧ pun)`"

(p. 141)

[19]:
```
argument6 = Argument.fromStr('evi, ali -> gui, pun')
argument7 = Argument.fromStr('gui, jus -> gui, pun')
assert not argument4.is_presumptively_valid_in(case_model)
assert not argument5.is_presumptively_valid_in(case_model)
```

"But the first step of the chain—the step to guilt—can be made when there are grounds for justification. Formally, this can be seen by the presumptive validity of this argument:

- Argument 8 (presumptively valid): `(evi ∧ jus, gui)`"

(p. 141)

[20]:
```
argument8 = Argument.fromStr('evi, jus -> gui')
assert argument8.is_presumptively_valid_in(case_model)
```

1.3.1 Learning arguments

[21]:
```
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
evi <-
gui   pun <~
---

ali <- ¬gui
evi <-
gui <- pun
gui   jus <- ¬pun
gui   ¬pun <- jus
¬gui <- ali
gui   pun <~
```

---

ali evi gui jus pun <~
¬gui <~ ali
¬pun <~ jus

1.4 DNA evidence

"We discuss an example, adapting our earlier treatment of the presumption of innocence. Consider a crime case where two pieces of evidence are found, one after another. In combination, they are considered to prove the suspect's guilt beyond a reasonable doubt. For instance, one piece of evidence is a witness who claims to have seen the suspect committing the crime (`evi`), and a second piece of evidence is DNA evidence matching the suspect's profile (`evi'`). The issue is whether the suspect is innocent (`inn`) or guilty (`gui`). Consider now a case model with four cases:

- Case 1: `inn` ∧ `¬gui` ∧ `¬evi`
- Case 2: `¬inn` ∧ `gui` ∧ `evi` ∧ `¬evi'`
- Case 3: `inn` ∧ `¬gui` ∧ `evi` ∧ `¬evi'`
- Case 4: `¬inn` ∧ `gui` ∧ `evi` ∧ `evi'`

Case 1 expresses the situation when no evidence has been found, hence the suspect is considered innocent and not guilty. In order to express that by default there is no evidence concerning someone's guilt, this case has highest preference. Cases 2 and 3 express the situation that the first piece of evidence is found. Case 2 expresses guilt, Case 3 innocence, still considered a possibility given only the first piece of evidence. In order to express that `evi` makes the suspect's guilt more plausible than his innocence, Case 2 has higher preference than Case 3. Case 4 represents the situation that both pieces of evidence are available, proving guilt. It has lowest preference. Summarizing the preference relation we have:

Case 1 > Case 2 > Case 3 > Case 4"

(p. 145-146)

[22]:
```
case_model = CaseModel.fromStr([
    (3, 'inn, ¬gui, ¬evi'),
    (2, "¬inn, gui, evi, ¬evi'"),
    (1, "inn, ¬gui, evi, ¬evi'"),
    (0, "¬inn, gui, evi, evi'"),
])
```

table 1 and 2

[23]:
```
argument1 = Argument.fromStr('-> inn')
argument2 = Argument.fromStr('-> gui')
argument3 = Argument.fromStr('evi -> inn')
argument4 = Argument.fromStr('evi -> gui')
argument5 = Argument.fromStr("evi, evi' -> inn")
argument6 = Argument.fromStr("evi, evi' -> gui")

assert argument1.is_coherent_in(case_model)
```

```
assert not argument1.is_conclusive_in(case_model)
assert argument1.is_presumptively_valid_in(case_model)

assert argument2.is_coherent_in(case_model)
assert not argument2.is_conclusive_in(case_model)
assert not argument2.is_presumptively_valid_in(case_model)

assert argument3.is_coherent_in(case_model)
assert not argument3.is_conclusive_in(case_model)
assert not argument3.is_presumptively_valid_in(case_model)

assert argument4.is_coherent_in(case_model)
assert not argument4.is_conclusive_in(case_model)
assert argument4.is_presumptively_valid_in(case_model)

assert not argument5.is_coherent_in(case_model)
assert not argument5.is_conclusive_in(case_model)
assert not argument5.is_presumptively_valid_in(case_model)

assert argument6.is_coherent_in(case_model)
assert argument6.is_conclusive_in(case_model)
assert argument6.is_presumptively_valid_in(case_model)
```

1.4.1 Learning arguments

[24]:
```
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
inn   ¬evi   ¬gui <~
---

evi <- ¬evi'
evi   gui <- ¬inn
evi   gui   ¬inn <- evi'
evi   ¬inn <- gui
inn <- ¬gui
inn   ¬gui <- ¬evi
¬evi' <- evi   inn
¬evi' <- evi   ¬gui
¬gui <- inn
gui   ¬evi'   ¬inn <~ evi
gui   ¬inn <~ evi   ¬evi'
gui   ¬inn <~ ¬evi'
inn   ¬evi   ¬gui <~
```

```
¬evi' <~ gui
¬evi' <~ ¬inn
---

evi   gui   inn   ¬evi' <~
¬gui <~ inn
¬inn <~ gui
```

1.5 Alfred Hitchcock's "To catch a thief"

"During the investigation, gradually a case model has been developed representing the arguments discussed in the example. (…) First the properties of the four main hypotheses are accumulated (…):" (p. 149)

[25]:
```
hypothesis1 = Case.fromStr(None, 'rob')
hypothesis2 = Case.fromStr(None, '¬rob, fou')
hypothesis3 = Case.fromStr(None, '¬rob, ¬fou, dau, jwl')
hypothesis4 = Case.fromStr(None, '¬rob, ¬fou, ¬dau, ¬jwl')
```

"Then these are conjoined with the maximally specific accumulated evidence that provide a coherent argument for them:" (p. 149)

[26]:
```
evidence1 = Case.fromStr(None, 'res, esc')
evidence2 = Case.fromStr(None, 'res, esc, fgt')
evidence3 = Case.fromStr(None, 'res, esc, fgt, pro, cau, con, fin')
evidence4 = Case.fromStr(None, 'res, esc, fgt, pro, cau, con')
```

"Cases 5–7 complete the case model. Case 5 is the hypothetical case that Robie is not the thief, that there is resemblance, and the Robie does not escape. In Case 6, Robie and Foussard are not the thieves, and there is no fight. In Case 7, Robie, Foussard and his daughter are not the thieves, and she is not caught in the act. Note that the cases are consistent and mutually exclusive."

[27]:
```
case1 = Case(4, hypothesis1.facts + evidence1.facts)
case2 = Case(3, hypothesis2.facts + evidence2.facts)
case3 = Case(1, hypothesis3.facts + evidence3.facts)
case4 = Case(0, hypothesis4.facts + evidence4.facts)
case5 = Case.fromStr(3, '¬rob, res, ¬esc')
case6 = Case.fromStr(2, '¬rob, ¬fou, res, esc, ¬fgt')
case7 = Case.fromStr(0, '¬rob, ¬fou, ¬dau, res, esc, fgt, pro, ¬cau')
case_model = CaseModel(
    [case1, case2, case3, case4, case5, case6, case7])
```

"(…) the argument from the evidential premises `res` ∧ `esc` to the hypothesis `rob` is presumptively valid in this case model since Case 1 is the only case implying the case made by the argument. It is not conclusive since also the argument from these same premises to `¬rob` is coherent." (p. 150)

[28]:
```
argument1 = Argument.fromStr('res, esc -> rob')
assert argument1.is_presumptively_valid_in(case_model)
assert not argument1.is_conclusive_in(case_model)
```

"The latter argument is not presumptively valid since all ¬rob-cases implying the premises (Cases 2–7) have lower preference than Case 1." (p. 150)

[29]:
```
argument2 = Argument.fromStr('res, esc -> ¬rob')
assert not argument2.is_presumptively_valid_in(case_model)
```

"The argument from res ∧ esc ∧ fgt to rob is incoherent as there is no case in which the premises and the conclusion follow." (p. 150)

[30]:
```
argument3 = Argument.fromStr('res, esc, fgt -> rob')
assert not argument3.is_coherent_in(case_model)
```

"Also arguments that do not start from evidential premises can be evaluated. For instance, the argument from the premise (not itself evidence) dau to jwl is conclusive since in the only case implying the premises (Case 3) the conclusion follows." (p. 150)

[31]:
```
argument4 = Argument.fromStr('dau -> jwl')
assert argument4.is_conclusive_in(case_model)
```

"Finally we find the conclusive argument from premises res ∧ esc ∧ fgt ∧ pro ∧ cau ∧ con ∧ jwl to conclusion ¬rob ∧ ¬fou ∧ dau ∧ jwl (only Case 3 implies the premises), hence also to dau." (p. 150)

[32]:
```
argument5 = Argument.fromStr('res, esc, fgt, pro, cau, con, jwl -> ¬rob, ¬fou,␣
↪dau, jwl')
assert argument5.is_conclusive_in(case_model)
```

1.5.1 Learning arguments

We cannot learn the arguments naively, because the case model is too big.

[33]:
```
# print(Theory.learn_with_naive_search(case_model))
print('---')
# print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
---
---

cau   con   esc   fgt   fin   jwl   pro   res   ¬dau   ¬fou   ¬rob <~
dau <~ fin
¬jwl <~ ¬dau
```

2 Artificial data

[37]:
```
case_model = CaseModel.fromStr([
    (2, 'y'),
    (2, 'y'),
```

```
    (2, 'y'),
    (1, 'a, ¬y'),
    (1, 'a, ¬y'),
    (0, 'a, b,  y'),
])
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
y <~
---

a <- ¬y
a  y <- b
b <- a  y
y <~
¬y <~ a
---

a  b  y <~
y <~ b
¬y <~ a
```

[38]:
```
case_model = CaseModel.fromStr([
    (2, 'a,  b,  c,  y'),
    (2, 'a,  b,  c,  y'),
    (2, 'a,  b,  c,  y'),
    (1, 'a,  b, ¬c, ¬y'),
    (1, 'a,  b, ¬c, ¬y'),
    (0, 'a, ¬b, ¬c,  y'),
])
print(Theory.learn_with_naive_search(case_model))
print('---')
print(Theory.learn_with_pruned_search(case_model))
print('---')
print(Theory.learn_with_hero(case_model))
```

```
a <-
b  c  y <~
---

a <-
b  y <- c
b  ¬c <- ¬y
```

```
c <- b   y
y   ¬c <- ¬b
¬b <- y   ¬c
¬y <- b   ¬c
b   c   y <~
¬y <~ a   ¬c
¬y <~ ¬c
---

a   b   c   y <~
¬b <~ y   ¬c
¬c <~ ¬b
¬c <~ ¬y
¬y <~ b   ¬c
```

[ ]: