simple-rdr

October 26, 2023

1 Tugas IF4070 - Representasi Pengetahuan dan Penalaran

2 Implementasi Ripple Down Rules

3 Simple RDR

Simple RDR is a ripple down rules implemented in Python.

3.1 Setup

Assuming you've installed the latest version of Python (if not, guides for it are widely available), 1. ensure pip is installed by running `python -m ensurepip --upgrade`; 2. install the Python dependencies by running `pip install -r requirements.txt`.

[122]:
```
import random
```

[123]:
```
class Node:
    def __init__(
        self,
        precedent: str,
        antecedent: str,
        cornerstone: set[str],
        except_: "Node" = None,
        else_: "Node" = None,
        is_root: bool = False,
    ) -> None:
        self._precedent = precedent
        self._antecedent = antecedent
        self._cornerstone = cornerstone
        self._except = except_
        self._else = else_
        self._is_root = is_root

    def get_antecedent(self) -> str:
        return self._antecedent

    def get_cornerstone(self) -> set[str]:
        return self._cornerstone
```

```
    def get_except(self) -> "Node":
        return self._except

    def get_else(self) -> "Node":
        return self._else

    def set_except(self, except_: "Node") -> None:
        self._except = except_

    def set_else(self, else_: "Node") -> None:
        self._else = else_

    def match_precedent(self, case: set[str]) -> bool:
        if self._is_root:
            return True
        else:
            for statement in case:
                if statement.startswith("~"):
                    exec(f"{statement[1:]} = False")
                else:
                    exec(f"{statement} = True")

            try:
                eval(self._precedent)
                return True
            except NameError:
                return False
            finally:
                for statement in case:
                    if statement.startswith("~"):
                        exec(f"del {statement[1:]}")
                    else:
                        exec(f"del {statement}")
```

[124]:
```
class Tree:
    def __init__(self, root: "Node"):
        self._root = root

    def _traverse_tree(self, case: set[str]) -> None:
        previous_node = self._root
        current_node = self._root
        last_true = self._root

        while current_node:
            if next_node := current_node.match_precedent(case):
                last_true = current_node
```

```
                previous_node = current_node
                current_node = current_node.get_except()
            else:
                previous_node = current_node
                current_node = current_node.get_else()

        antecedent = last_true.get_antecedent()
        print(f"The system concludes that your case can be associated with the␣
↪following: {antecedent}.")
        print("Do you agree? (y/n)")

        while True:
            agreement = input(
                f"The system concludes that your case can be associated with␣
↪the following: {antecedent}.\nDo you agree? (y/n)"
            ).lower()
            print(f"You entered: {agreement}")
            if agreement in ("y", "n"):
                break
            else:
                print(f"Please input a valid option!")

        if agreement == "n":
            print("Please input a correct conclusion for this case!")
            conclusion = input("Please input a correct conclusion for this case!
↪")
            print(f"You entered: {conclusion}")

            if len(case - last_true.get_cornerstone()) == 0:
                new_precedent = random.choice(tuple(case))
            else:
                try:
                    new_precedent = random.choice(tuple(case - last_true.
↪get_cornerstone()))
                except:
                    new_precedent = "~" + random.choice(tuple(last_true.
↪get_cornerstone() - case))
            new_node = Node(new_precedent, conclusion, case)
            if next_node:
                previous_node.set_except(new_node)
            else:
                previous_node.set_else(new_node)

    def _traverse_tree_by_dataset(self, case: set[str], label: str) -> None:
        previous_node = self._root
        current_node = self._root
        last_true = self._root
```

```
        while current_node:
            if next_node := current_node.match_precedent(case):
                last_true = current_node
                previous_node = current_node
                current_node = current_node.get_except()
            else:
                previous_node = current_node
                current_node = current_node.get_else()

        if (label != last_true.get_antecedent()):
            if len(case - last_true.get_cornerstone()) == 0:
                new_precedent = random.choice(tuple(case))
            else:
                try:
                    new_precedent = random.choice(tuple(case - last_true.
↪get_cornerstone()))
                except:
                    new_precedent = "~" + random.choice(tuple(last_true.
↪get_cornerstone() - case))
            new_node = Node(new_precedent, label, case)
            if next_node:
                previous_node.set_except(new_node)
            else:
                previous_node.set_else(new_node)


    def start(self) -> None:
        print("Welcome to RDR Expert System!")
        print()

        while True:
            print("Enter your case, separated by a comma for each fact!")
            print("Example case: mammal, fly, ~swim")
            print("Input your case!")

            case = set(input("Input your case: ").split(", "))
            print(f"You entered: {case}")
            self._traverse_tree(case)

            while True:
                print()
                print("Would you like to evaluate a different case? (y/n)")
                continue_use = input("Would you like to evaluate a different␣
↪case? (y/n) ").lower()
                if continue_use in ("y", "n"):
                    break
```

```
                else:
                    print("Please input a valid option!")

            if continue_use == "y":
                continue
            else:
                break

    def fit(self, cases: list[set[str]], labels: list[str]) -> None:
        for i in range(len(cases)):
            # print(f"Training case {i+1}: {cases[i]} -> {labels[i]}")
            self._traverse_tree_by_dataset(cases[i], labels[i])

    def predict(self, case: set[str]) -> str:
        current_node = self._root
        last_true = self._root

        while current_node and (current_node.get_except() or current_node.
↪get_else()):
            if current_node.match_precedent(case):
                last_true = current_node
                current_node = current_node.get_except()
            else:
                current_node = current_node.get_else()
        else:
            if current_node.match_precedent(case):
                last_true = current_node

        return last_true.get_antecedent()
```

4 Dataset yang digunakan - Zoo.csv

sumber : https://www.kaggle.com/datasets/uciml/zoo-animal-classification?select=zoo.csv

4.1 Set up

[125]:
```
import pandas as pd

df = pd.read_csv("./dataset/zoo.csv")
df.head()
```

[125]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	\
0	aardvark	1	0	0	1	0	0	1	\
1	antelope	1	0	0	1	0	0	0	
2	bass	0	0	1	0	0	1	1	
3	bear	1	0	0	1	0	0	1	

```
4        boar     1         0     0     1         0         0         1

   toothed  backbone  breathes  venomous  fins  legs  tail  domestic  catsize
0        1         1         1         0     0     4     0         0        1  \
1        1         1         1         0     0     4     1         0        1
2        1         1         0         0     1     0     1         0        0
3        1         1         1         0     0     4     0         0        1
4        1         1         1         0     0     4     1         0        1

   class_type
0           1
1           1
2           4
3           1
4           1
```

4.2 Deskripsi Dataset

Dataset yang digunakan adalah dataset Zoo.csv yang berisi data mengenai 101 hewan unik yang terdiri dari 16 atribut. Atribut-atribut tersebut adalah sebagai berikut: | Atribut | Tipe Data | Keterangan | | — | — | — | | animal_name | String (unique) | Nama hewan | | hair | Boolean | Apakah hewan tersebut berbulu? | | feathers | Boolean | Apakah hewan tersebut berbulu? | | eggs | Boolean | Apakah hewan tersebut bertelur? | | milk | Boolean | Apakah hewan tersebut menyusui? | | airborne | Boolean | Apakah hewan tersebut terbang? | | aquatic | Boolean | Apakah hewan tersebut hidup di air? | | predator | Boolean | Apakah hewan tersebut predator? | | toothed | Boolean | Apakah hewan tersebut ber gigi? | | backbone | Boolean | Apakah hewan tersebut memiliki tulang belakang? | | breathes | Boolean | Apakah hewan tersebut bernafas? | | venomous | Boolean | Apakah hewan tersebut berbisa? | | fins | Boolean | Apakah hewan tersebut memiliki sirip? | | legs | Integer | Jumlah kaki hewan tersebut | | tail | Boolean | Apakah hewan tersebut memiliki ekor? | | domestic | Boolean | Apakah hewan tersebut jinak? | | catsize | Boolean | Apakah hewan tersebut berukuran besar? | | class_type | Integer | Tipe kelas hewan tersebut |

4.3 Preprocessing Dataset

preprocessing pada dataset dilakukan dengan mengabaikan kolom yang densitas datanya merupakan noise serta mengabaikan baris pada tiap kolom yang mengandung missing value.

[126]:
```
df.shape
```

[126]:
```
(101, 18)
```

[127]:
```
df.isnull().sum()
```

[127]:
```
animal_name    0
hair           0
feathers       0
eggs           0
milk           0
```

```
airborne       0
aquatic        0
predator       0
toothed        0
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
class_type     0
dtype: int64
```

Dikarenakan tidak ada kolom maupun baris yang mengandung missing value, maka tidak perlu dilakukan pengabaian data. Selanjutnya, untuk simplifikasi penggunaan dataset, kolom legs dan class_type akan dihapus. Perlu ditekankan bahwa pada dataset ini, kolom animal_name akan bertindak sebagai label dari data.

[128]:
```
df = df[["animal_name", "hair", "feathers", "eggs", "milk", "airborne",␣
 ↪"aquatic", "predator", "toothed", "backbone", "breathes", "venomous",␣
 ↪"fins", "tail", "domestic", "catsize"]]
df.head()
```

[128]:

	animal_name	hair	feathers	eggs	milk	airborne	aquatic	predator	\
0	aardvark	1	0	0	1	0	0	1	\
1	antelope	1	0	0	1	0	0	0	
2	bass	0	0	1	0	0	1	1	
3	bear	1	0	0	1	0	0	1	
4	boar	1	0	0	1	0	0	1	

	toothed	backbone	breathes	venomous	fins	tail	domestic	catsize
0	1	1	1	0	0	0	0	1
1	1	1	1	0	0	1	0	1
2	1	1	0	0	1	1	0	0
3	1	1	1	0	0	0	0	1
4	1	1	1	0	0	1	0	1

4.4 Implementasi RDR

Pada tahap ini, akan dilakukan 2 jenis implementasi, yaitu dengan manual (diinput oleh user) dan dengan menggunakan dataset yang telah disediakan.

4.4.1 Testing - with Dataset

[129]:
```
def read_dataset(df: pd.DataFrame) -> (list[set[str]], list[str]):
    dataset = []
    labels = []
    for _, row in df.iterrows():
        case = set()
        for column in df.columns:
            if row[column] == 1:
                case.add(column)
            elif type(row[column]) == str:
                labels.append(row[column])
        dataset.append(case)
    return dataset, labels
```

[130]:
```
dataset, labels = read_dataset(df)
for data in dataset:
    print(set(data))

for label in labels:
    print(label)
```

```
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'predator'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'toothed', 'hair', 'breathes', 'tail', 'domestic', 'milk', 'catsize',
'backbone'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'tail', 'domestic'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'domestic'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'backbone', 'breathes', 'feathers', 'airborne', 'eggs', 'tail', 'domestic'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'eggs', 'predator'}
{'aquatic', 'eggs', 'predator'}
{'aquatic', 'eggs', 'predator'}
{'backbone', 'breathes', 'airborne', 'eggs', 'predator', 'tail', 'feathers'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'fins', 'toothed', 'aquatic', 'tail', 'catsize', 'backbone', 'eggs',
'predator'}
{'fins', 'toothed', 'aquatic', 'breathes', 'tail', 'milk', 'catsize',
'backbone', 'predator'}
{'backbone', 'breathes', 'feathers', 'airborne', 'eggs', 'tail', 'domestic'}
```

```
{'backbone', 'breathes', 'aquatic', 'airborne', 'eggs', 'tail', 'feathers'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'backbone', 'breathes', 'catsize', 'airborne', 'eggs', 'tail', 'feathers'}
{'breathes', 'eggs'}
{'backbone', 'breathes', 'toothed', 'aquatic', 'eggs', 'predator'}
{'backbone', 'breathes', 'toothed', 'venomous', 'aquatic', 'eggs', 'predator'}
{'backbone', 'breathes', 'milk', 'toothed', 'hair', 'airborne', 'tail'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'toothed', 'hair', 'breathes', 'domestic', 'milk', 'catsize', 'backbone',
'predator'}
{'airborne', 'eggs', 'breathes'}
{'toothed', 'hair', 'breathes', 'tail', 'domestic', 'milk', 'catsize',
'backbone'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes'}
{'airborne', 'aquatic', 'breathes', 'tail', 'feathers', 'backbone', 'eggs',
'predator'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'tail'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'tail', 'domestic'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'tail'}
{'backbone', 'breathes', 'airborne', 'eggs', 'predator', 'tail', 'feathers'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'breathes', 'venomous', 'hair', 'airborne', 'eggs', 'domestic'}
{'hair', 'airborne', 'eggs', 'breathes'}
{'backbone', 'breathes', 'eggs', 'predator', 'tail', 'feathers'}
{'airborne', 'eggs', 'predator', 'breathes'}
{'backbone', 'breathes', 'airborne', 'eggs', 'tail', 'feathers'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'aquatic', 'eggs', 'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'toothed', 'hair', 'aquatic', 'breathes', 'tail', 'milk', 'catsize',
'backbone', 'predator'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'predator', 'tail'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'hair', 'airborne', 'eggs', 'breathes'}
{'backbone', 'breathes', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'catsize', 'aquatic', 'eggs', 'predator'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'predator', 'tail'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'backbone', 'catsize', 'breathes', 'eggs', 'tail', 'feathers'}
{'backbone', 'breathes', 'feathers', 'airborne', 'eggs', 'tail', 'domestic'}
{'aquatic', 'breathes', 'tail', 'feathers', 'catsize', 'backbone', 'eggs',
'predator'}
{'backbone', 'breathes', 'airborne', 'eggs', 'tail', 'feathers'}
```

```
{'fins', 'toothed', 'aquatic', 'tail', 'catsize', 'backbone', 'eggs',
'predator'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'predator', 'tail'}
{'backbone', 'toothed', 'venomous', 'breathes', 'eggs', 'predator', 'tail'}
{'hair', 'aquatic', 'breathes', 'tail', 'milk', 'catsize', 'backbone', 'eggs',
'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'domestic', 'milk', 'catsize',
'backbone'}
{'fins', 'toothed', 'aquatic', 'breathes', 'tail', 'milk', 'catsize',
'backbone', 'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'domestic', 'milk', 'catsize',
'backbone', 'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'toothed', 'hair', 'breathes', 'tail', 'domestic', 'milk', 'catsize',
'backbone'}
{'backbone', 'catsize', 'breathes', 'eggs', 'predator', 'tail', 'feathers'}
{'breathes', 'tail', 'predator', 'venomous'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'tail'}
{'fins', 'toothed', 'hair', 'aquatic', 'breathes', 'milk', 'catsize',
'backbone', 'predator'}
{'fins', 'toothed', 'hair', 'aquatic', 'breathes', 'tail', 'milk', 'catsize',
'backbone', 'predator'}
{'backbone', 'toothed', 'venomous', 'aquatic', 'predator', 'tail'}
{'aquatic', 'eggs', 'predator', 'venomous'}
{'airborne', 'aquatic', 'breathes', 'tail', 'feathers', 'backbone', 'eggs',
'predator'}
{'airborne', 'aquatic', 'breathes', 'tail', 'feathers', 'backbone', 'eggs',
'predator'}
{'backbone', 'toothed', 'breathes', 'eggs', 'predator', 'tail'}
{'breathes', 'eggs'}
{'backbone', 'fins', 'toothed', 'aquatic', 'eggs', 'tail'}
{'backbone', 'breathes', 'airborne', 'eggs', 'tail', 'feathers'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'tail'}
{'aquatic', 'eggs', 'predator'}
{'fins', 'toothed', 'venomous', 'aquatic', 'tail', 'catsize', 'backbone',
'eggs', 'predator'}
{'airborne', 'aquatic', 'breathes', 'tail', 'feathers', 'catsize', 'backbone',
'eggs'}
{'breathes', 'eggs'}
{'backbone', 'breathes', 'toothed', 'aquatic', 'eggs'}
{'backbone', 'catsize', 'breathes', 'eggs', 'tail'}
{'backbone', 'toothed', 'breathes', 'eggs', 'predator', 'tail'}
{'fins', 'toothed', 'aquatic', 'tail', 'catsize', 'backbone', 'eggs',
```

```
'predator'}
{'backbone', 'breathes', 'milk', 'toothed', 'hair', 'airborne', 'tail'}
{'backbone', 'milk', 'toothed', 'hair', 'breathes', 'tail'}
{'airborne', 'breathes', 'tail', 'feathers', 'catsize', 'backbone', 'eggs',
'predator'}
{'backbone', 'milk', 'toothed', 'catsize', 'hair', 'breathes', 'tail'}
{'breathes', 'venomous', 'hair', 'airborne', 'eggs'}
{'toothed', 'hair', 'breathes', 'tail', 'milk', 'catsize', 'backbone',
'predator'}
{'breathes', 'eggs'}
{'backbone', 'breathes', 'airborne', 'eggs', 'tail', 'feathers'}
aardvark
antelope
bass
bear
boar
buffalo
calf
carp
catfish
cavy
cheetah
chicken
chub
clam
crab
crayfish
crow
deer
dogfish
dolphin
dove
duck
elephant
flamingo
flea
frog
frog
fruitbat
giraffe
girl
gnat
goat
gorilla
gull
haddock
hamster
hare
```

hawk
herring
honeybee
housefly
kiwi
ladybird
lark
leopard
lion
lobster
lynx
mink
mole
mongoose
moth
newt
octopus
opossum
oryx
ostrich
parakeet
penguin
pheasant
pike
piranha
pitviper
platypus
polecat
pony
porpoise
puma
pussycat
raccoon
reindeer
rhea
scorpion
seahorse
seal
sealion
seasnake
seawasp
skimmer
skua
slowworm
slug
sole
sparrow
squirrel

```
starfish
stingray
swan
termite
toad
tortoise
tuatara
tuna
vampire
vole
vulture
wallaby
wasp
wolf
worm
wren
```

[131]:
```
root_node = Node("", "human", set(), is_root=True)
model = Tree(root_node)

model.fit(dataset, labels)
```

[132]:
```
testData = {'toothed', 'hair', 'breathes', 'milk', 'catsize', 'backbone',␣
 ↪'predator'}
print(model.predict(testData))
```

```
gorilla
```

4.4.2 Testing - Manual

[133]:
```
root_node = Node("", "human", set(), is_root=True)
model = Tree(root_node)

model.start()
```