ДОДАТОК А

Код програми

Лістинг А.1 - код файлу RuleLearner.py:

```
import string
import Orange
from Orange.classification.rules import
LaplaceAccuracyEvaluator, EntropyEvaluator
from owlready2 import *
import argparse
from sklearn.metrics import classification_report

def create_ontology(str_iri):
    onto = get_ontology(str_iri)
    return onto

def save_ontology_to_file(onto, filename):
    onto.save(filename)

def check_if_numerical_range(string):
    res = re.findall("[A-Za-z]", string)
    return not res

# Some operators like ≤ or range - (e.g. 322 - 531) require
additional atoms to be added to the rule
# For example range (l - r) is represented as (v - l) * (r -
v) >= 0 (where v is the value that we are checking),
# because in the case of range negation we just need to change
the operator to <= 0. Also we do include the left
# boundary of the range, so we need to make sure that the
right boundary is not included, for this we add the last
# additional atom - notEqual r/l != value
def get_numerical_atoms(rule_atom, instance_var, current_var,
```

```
negate=False):
    rule_atom_value = rule_atom[1]
    rule_atom = "has" + rule_atom[0] + "(" + "?" +
instance_var + ", " + "?" + current_var[0] + ")"
    additional_atoms = [rule_atom]
    if rule_atom_value[0] == "≥":
        if negate:
            additional_atoms.append("lessThan(" + "?" +
current_var[0] + ", " + rule_atom_value[2:] + ")")
        else:
            additional_atoms.append("greaterThanOrEqual(" +
"?" + current_var[0] + ", " + rule_atom_value[2:] + ")")
    elif rule_atom_value[0] == "<":
        if negate:
            additional_atoms.append("greaterThanOrEqual(" +
"?" + current_var[0] + ", " + rule_atom_value[2:] + ")")
        else:
            additional_atoms.append("lessThan(" + "?" +
current_var[0] + ", " + rule_atom_value[2:] + ")")
    else:
        subtract_res_var1 = get_next_letter(current_var)
        subtract_res_var2 = get_next_letter(subtract_res_var1)
        multiply_res_var2 = get_next_letter(subtract_res_var2)
        left_range_value, right_range_value =
rule_atom_value.split(" - ")
        additional_atoms.append(
            "subtract(" + "?" + subtract_res_var1[0] + ", " +
left_range_value + ", " + "?" + current_var[0] + ")")
        additional_atoms.append(
            "subtract(" + "?" + subtract_res_var2[0] + ", ?" +
current_var[0] + ", " + right_range_value + ")")
        additional_atoms.append(
            "multiply(" + "?" + multiply_res_var2[0] + ", ?" +
subtract_res_var1[0] + ", " + "?" + subtract_res_var2[0] +
")")
```

```
        if negate:
            additional_atoms.append("lessThanOrEqual(" + "?" +
multiply_res_var2[0] + ", " + "0" + ")")
            additional_atoms.append("notEqual(" + "?" +
subtract_res_var1[0] + ", " + "0" + ")")
        else:
            additional_atoms.append("greaterThanOrEqual(" +
"?" + multiply_res_var2[0] + ", " + "0" + ")")
            additional_atoms.append("notEqual(" + "?" +
subtract_res_var2[0] + ", " + "0" + ")")
        current_var[0] = multiply_res_var2[0]
    return additional_atoms

def get_categorical_atoms(rule_atom, instance_var,
current_var, negate=False):
    rule_atom_value = rule_atom[1]
    rule_atom = "has" + rule_atom[0] + "(" + "?" +
instance_var + ", " + "?" + current_var[0] + ")"
    additional_atoms = [rule_atom]
    if negate:
        additional_atoms.append("notEqual(" + "?" +
current_var[0] + ", \"" + rule_atom_value + "\")")
    else:
        additional_atoms.append("equal(" + "?" +
current_var[0] + ", \"" + rule_atom_value + "\")")
    return additional_atoms

def convert_rules_to_swrl_rules(rules):
    owl_data_properties = []
    owl_classes = []
    owl_rules = []
    for rule in rules:
        rule_string = str(rule).replace("IF", "")
        rule_parts = rule_string.split("THEN")
        rule_body = rule_parts[0].split("AND")
```

```
        rule_head = rule_parts[1]
        rule_atoms = []
        instance_var = "a"
        current_var = [instance_var]
        current_var = get_next_letter(current_var)
        for rule_atom in rule_body:
            rule_atom = rule_atom.strip()
            rule_atom = re.sub('[()]', '', rule_atom)
            if rule_atom == "TRUE":
                continue
            elif rule_atom.find("==") == -1:  # value negation
case (e.g. != blue)
                rule_atom = rule_atom.split("!=")
                rule_atom[0] = re.sub(' ', '_', rule_atom[0])
                rule_atom[0] = re.sub('-', '_', rule_atom[0])
                if check_if_numerical_range(rule_atom[1]):
                    additional_atoms =
get_numerical_atoms(rule_atom, instance_var, current_var,
negate=True)
                else:  # categorical value
                    additional_atoms =
get_categorical_atoms(rule_atom, instance_var, current_var,
negate=True)
            else:
                rule_atom = rule_atom.split("==")
                rule_atom[0] = re.sub(' ', '_', rule_atom[0])
                rule_atom[0] = re.sub('-', '_', rule_atom[0])
                if check_if_numerical_range(rule_atom[1]):
                    additional_atoms =
get_numerical_atoms(rule_atom, instance_var, current_var)
                else:  # categorical value
                    additional_atoms =
get_categorical_atoms(rule_atom, instance_var, current_var)
            rule_atom = ", ".join(additional_atoms)
            current_var = get_next_letter(current_var)
```

```
            rule_atoms.append(rule_atom)

        rule_body = ", ".join(rule_atoms)
        if len(rule_body) == 0:
            continue

        rule_head = rule_head.strip().replace(" ",
"_").split("=")
        if not re.findall("[^0-9]", rule_head[1]):
            rule_head[1] = rule_head[0] + "_" + rule_head[1]
        rule_head = rule_head[1] + "(" + "?" + instance_var +
")"

        owl_rule = rule_body + " -> " + rule_head
        owl_rules.append(owl_rule)
    return owl_rules

def create_numerical_prop(name, domain_class):
    prop = types.new_class(name, (DataProperty,
FunctionalProperty,))
    prop.domain = domain_class
    prop.range = float

def create_string_prop(name, domain_class):
    prop = types.new_class(name, (DataProperty,
FunctionalProperty,))
    prop.domain = domain_class
    prop.range = str

def create_class(name, parent_class):
    new_class = types.new_class(name, (parent_class,))

def create_ontology_entities_from_rules(ontology, owl_rules,
data):
    created_entities = set()
```

```
    with ontology:
        target_class =
types.new_class(data.domain.class_var.name.replace(" ", "_"),
(Thing,))
        for rule in owl_rules:
            rule_body, rule_head = rule.split(" -> ")
            rule_body_atoms = rule_body.split("), ")
            for i, val in enumerate(rule_body_atoms):
                if val.startswith("has"):
                    prop_name = re.findall("(has.*)\(",
val)[0]
                    if prop_name in created_entities:
                        continue
                    if
rule_body_atoms[i+1].startswith("equal") or
rule_body_atoms[i+1].startswith("notEqual"):
                        create_string_prop(prop_name,
target_class)
                    else:
                        create_numerical_prop(prop_name,
target_class)
                    created_entities.add(prop_name)
            example_class = re.findall("(.*)\(", rule_head)[0]
            if example_class in created_entities:
                continue
            else:
                created_entities.add(rule_head)
            create_class(example_class, target_class)

def add_rules_to_ontology(ontology, owl_rules):
    with ontology:
        for rule in owl_rules:
            rule_imp = Imp()
            rule_imp.set_as_rule(rule)
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	93 с.
	Інші документи	
2.	Презентаційні матеріали	плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Доценко К.О.			Поєднання нейронних мереж з моделями штучного інтелекту, що мають пояснення знань
Перевірив		Вітько О.В.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-20-1	122
ХНУРЕ кафедра ШІ	