

237 lines - 9 Removals

```

1 """
2 preprocessing_pipeline_steps: Implementation of pipeline steps to extract examples from testcases and
  construct a dataset.
3 """
4
5 import logging
6 import re
7 import typing
8
9 from pipeline_framework import PipelineState, PipelineStep
10
11
12 class ValidateTestcaseFilesPipelineStep(PipelineStep):
13     step_key = "validate_testcase_files"
14
15     def input_ready(self, input: dict) -> bool:
16         return "testcases" in input.keys()
17
18     def execute(self, input: dict, output: dict) -> None:
19         import preprocessing
20
21         output_testcases = []
22         for testcase in input["testcases"]:
23             testcase["Valid"] = preprocessing.validate_testcase_files(testcase)
24             output_testcases += [testcase]
25
26         output["testcases"] = output_testcases
27
28     def output_ready(self, output: dict) -> typing.Optional[bool]:
29         return "testcases" in output.keys() and all(
30             [
31                 "Valid" in testcase.keys() and testcase["Valid"]
32                 for testcase in output["testcases"]
33             ]
34         )
35
36
37 class ExtractExamplesPipelineStep(PipelineStep):
38     step_key = "extract_examples"
39
40     def __init__(
41         self,
42         config: dict,
43         state: PipelineState,
44         logger: logging.Logger,
45         emit_primary_good_function: bool = True,

```

259 lines + 36 Additions

```

1 """
2 preprocessing_pipeline_steps: Implementation of pipeline steps to extract examples from testcases and
  construct a dataset.
3 """
4
5 import logging
6 import re
7 import typing
8 import warnings
9
10 from pipeline_framework import PipelineState, PipelineStep
11
12 # warnings.filterwarnings('ignore',
13 #                          message="Could not find symbol __stack_chk_fail in symbol translation table
14 #                          for disassembly")
15 # logging.basicConfig(filename='warnings.log', level=logging.WARNING)
16 # warnings.filterwarnings('ignore', category=UserWarning)
17
18 class ValidateTestcaseFilesPipelineStep(PipelineStep):
19     step_key = "validate_testcase_files"
20
21     def input_ready(self, input: dict) -> bool:
22         return "testcases" in input.keys()
23
24     def execute(self, input: dict, output: dict) -> None:
25         import preprocessing
26
27         output_testcases = []
28         for testcase in input["testcases"]:
29             testcase["Valid"] = preprocessing.validate_testcase_files(testcase)
30             output_testcases += [testcase]
31
32         output["testcases"] = output_testcases
33
34     def output_ready(self, output: dict) -> typing.Optional[bool]:
35         return "testcases" in output.keys() and all(
36             [
37                 "Valid" in testcase.keys() and testcase["Valid"]
38                 for testcase in output["testcases"]
39             ]
40         )
41
42
43 class ExtractExamplesPipelineStep(PipelineStep):
44     step_key = "extract_examples"
45
46     def __init__(
47         self,
48         config: dict,
49         state: PipelineState,
50         logger: logging.Logger,
51         emit_primary_good_function: bool = True,

```

```

46     label_granularity: typing.Optional[str] = N
one,
47     context_granularity: typing.Optional[str] =
None,
48 ):
49     super().__init__(config, state, logger)
50
51     import preprocessing
52
53     self.extract_fn = lambda testcase: preproce
ssing.extract_examples(
54         testcase,
55         label_granularity=preprocessing.Granula
rity[label_granularity]
56         if label_granularity is not None
57         else None,
58         context_granularity=preprocessing.Granu
larity[context_granularity]
59         if context_granularity is not None
60         else None,
61         emit_primary_good_function=emit_primary
_good_function,
62         logger=self.logger,
63     )

```

```

64
65     def input_ready(self, input: dict) -> bool:
66         return (
67             "cwes" in input.keys()
68             and "testcases" in input.keys()
69             and all(
70                 [
71                     "Valid" in testcase.keys() and
testcase["Valid"]
72                     for testcase in input["testcase
s"]
73                 ]
74             )
75         )
76
77     def execute(self, input: dict, output: dict) ->
None:
78         output_examples = []
79         for testcase in input["testcases"]:
80             output_examples += self.extract_fn(test
case)

```

```

52     label_granularity: typing.Optional[str] = N
one,
53     context_granularity: typing.Optional[str] =
None,
54 ):
55     super().__init__(config, state, logger)
56
57     import preprocessing
58
59     if self.state.malware:
60         self.extract_fn = lambda testcase: prep
rocessing.extract_examples_malware(
61             testcase,
62             label_granularity=preprocessing.Gra
nularity[label_granularity]
63             if label_granularity is not None
64             else None,
65             context_granularity=preprocessing.G
ranularity[context_granularity]
66             if context_granularity is not None
67             else None,
68             emit_primary_good_function=emit_pri
mary_good_function,
69             logger=self.logger
70         )
71     else:
72         self.extract_fn = lambda testcase: prep
rocessing.extract_examples(
73             testcase,
74             label_granularity=preprocessing.Gra
nularity[label_granularity]
75             if label_granularity is not None
76             else None,
77             context_granularity=preprocessing.G
ranularity[context_granularity]
78             if context_granularity is not None
79             else None,
80             emit_primary_good_function=emit_pri
mary_good_function,
81             logger=self.logger
82         )

```

```

83
84     def input_ready(self, input: dict) -> bool:
85         return (
86             "cwes" in input.keys()
87             and "testcases" in input.keys()
88             and all(
89                 [
90                     "Valid" in testcase.keys() and
testcase["Valid"]
91                     for testcase in input["testcase
s"]
92                 ]
93             )
94         )
95
96     def execute(self, input: dict, output: dict) ->
None:
97         output_examples = []
98         for testcase in input["testcases"]:
99             try:

```

```

81
82         self.logger.info(
83             f"Extracted {len(output_examples)} exam
ples from {len(input['testcases'])} testcases"
84         )
85
86         output["examples"] = output_examples
87
88         def output_ready(self, output: dict) -> typing.
Optional[bool]:
89             return "examples" in output.keys()
90
91
92     class LabelExamplesPipelineStep(PipelineStep):
93         step_key = "label_examples"
94
95         def input_ready(self, input: dict) -> bool:
96             return "examples" in input.keys()
97
98         def execute(self, input: dict, output: dict) ->
None:
99             import preprocessing
100
101             label_strategy = preprocessing.LabelStrateg
y[self.config["label_strategy"]]
102
103             output_examples = []
104             for example in input["examples"]:
105                 example["Label"] = preprocessing.label_
example(example, label_strategy)
106                 output_examples += [example]
107
108             output["examples"] = output_examples
109
110             if label_strategy == preprocessing.LabelStr
ategy.BINARYCLASSIFICATION:
111                 output["label_encoding"] = {"Good": 0,
"Bad": 1}
112             else:
113                 raise NotImplementedError("Labelstrateg
y not implemented!")
114
115         def output_ready(self, output: dict) -> typing.
Optional[bool]:
116             return (
117                 "examples" in output.keys()
118                 and all(["Label" in example.keys() for
example in output["examples"]])
119                 and "label_encoding" in output.keys()
120             )
121
122
123     class RemoveDuplicateExamplesPipelineStep(PipelineS
tep):
124         step_key = "remove_duplicate_examples"
125
126         def input_ready(self, input: dict) -> bool:
127             return "examples" in input.keys()
128
129         def execute(self, input: dict, output: dict) ->
None:
130             import numpy as np
131
132             # Some accounting
133             observed_representations = set()
134
135             output_examples += self.extract_fn
(testcase)
136
137             except Exception as ex:
138                 print(ex)
139
140
141         self.logger.info(
142             f"Extracted {len(output_examples)} exam
ples from {len(input['testcases'])} testcases"
143         )
144
145         output["examples"] = output_examples
146
147         def output_ready(self, output: dict) -> typing.
Optional[bool]:
148             return "examples" in output.keys()
149
150
151     class LabelExamplesPipelineStep(PipelineStep):
152         step_key = "label_examples"
153
154         def input_ready(self, input: dict) -> bool:
155             return "examples" in input.keys()
156
157         def execute(self, input: dict, output: dict) ->
None:
158             import preprocessing
159
160             label_strategy = preprocessing.LabelStrateg
y[self.config["label_strategy"]]
161
162             output_examples = []
163             for example in input["examples"]:
164                 example["Label"] = preprocessing.label_
example(example, label_strategy)
165                 output_examples += [example]
166
167             output["examples"] = output_examples
168
169             if label_strategy == preprocessing.LabelStr
ategy.BINARYCLASSIFICATION:
170                 output["label_encoding"] = {"Good": 0,
"Bad": 1}
171             else:
172                 raise NotImplementedError("Labelstrateg
y not implemented!")
173
174         def output_ready(self, output: dict) -> typing.
Optional[bool]:
175             return (
176                 "examples" in output.keys()
177                 and all(["Label" in example.keys() for
example in output["examples"]])
178                 and "label_encoding" in output.keys()
179             )
180
181
182     class RemoveDuplicateExamplesPipelineStep(PipelineS
tep):
183         step_key = "remove_duplicate_examples"
184
185         def input_ready(self, input: dict) -> bool:
186             return "examples" in input.keys()
187
188         def execute(self, input: dict, output: dict) ->
None:
189             import numpy as np
190
191             # Some accounting
192             observed_representations = set()

```

```

134         kept_examples = 0
135
136         # Shuffle examples to avoid bias from order
137         shuffled_input_examples = list(input["examples"])
138         rs = np.random.RandomState(self.config["seed"])
139         rs.shuffle(shuffled_input_examples)
140
141         # Mark which examples are kept
142         for example in input["examples"]:
143             # Descramble examples to reinstate non-
144             unique representations
145             normalized_text_representation = re.sub(
146                 r"!lc[0-9]+:", "", example["Example"]
147             )
148             # Use the power of python's set implementation to check for uniqueness
149             if normalized_text_representation not in observed_representations:
150                 observed_representations |= {normalized_text_representation}
151
152             # Mark example as kept
153             example["Keep"] = True
154             kept_examples += 1
155         else:
156             example["Keep"] = False
157
158         self.logger.info(
159             f"Duplicate examples removed: {len(input['examples']) - kept_examples} of {len(input['examples'])}. Remaining: {kept_examples}"
160         )
161
162         # Preserve the order of the examples
163         output["examples"] = [ex for ex in input["examples"] if ex["Keep"]]
164         assert len(output["examples"]) == kept_examples
165
166         def output_ready(self, output: dict) -> typing.Optional[bool]:
167             return "examples" in output.keys()
168
169
170 class AssignSplitToExamplesPipelineStep(PipelineStep):
171     step_key = "assign_split_to_examples"
172
173     def __init__(
174         self,
175         config: dict,
176         state: PipelineState,
177         logger: logging.Logger,
178         validation_fraction: float,
179         test_fraction: float,
180     ):
181         super().__init__(config, state, logger)
182         self.validation_fraction = validation_fraction
183
184         self.test_fraction = test_fraction
185

```

```

156         kept_examples = 0
157
158         # Shuffle examples to avoid bias from order
159         shuffled_input_examples = list(input["examples"])
160         rs = np.random.RandomState(self.config["seed"])
161         rs.shuffle(shuffled_input_examples)
162
163         # Mark which examples are kept
164         for example in input["examples"]:
165             # Descramble examples to reinstate non-
166             unique representations
167             normalized_text_representation = re.sub(
168                 r"!lc[0-9]+:", "", example["Example"]
169             )
170             # Use the power of python's set implementation to check for uniqueness
171             if normalized_text_representation not in observed_representations:
172                 observed_representations |= {normalized_text_representation}
173
174             # Mark example as kept
175             example["Keep"] = True
176             kept_examples += 1
177         else:
178             example["Keep"] = False
179
180         self.logger.info(
181             f"Duplicate examples removed: {len(input['examples']) - kept_examples} of {len(input['examples'])}. Remaining: {kept_examples}"
182         )
183
184         # Preserve the order of the examples
185         output["examples"] = [ex for ex in input["examples"] if ex["Keep"]]
186         assert len(output["examples"]) == kept_examples
187
188         def output_ready(self, output: dict) -> typing.Optional[bool]:
189             return "examples" in output.keys()
190
191
192 class AssignSplitToExamplesPipelineStep(PipelineStep):
193     step_key = "assign_split_to_examples"
194
195     def __init__(
196         self,
197         config: dict,
198         state: PipelineState,
199         logger: logging.Logger,
200         validation_fraction: float,
201         test_fraction: float,
202     ):
203         super().__init__(config, state, logger)
204         self.validation_fraction = validation_fraction
205
206         self.test_fraction = test_fraction
207

```

```

186     def input_ready(self, input: dict) -> bool:
187         return "examples" in input.keys()
188
189     def execute(self, input: dict, output: dict) ->
None:
190         import numpy as np
191
192         output_examples = list(input["examples"])
193
194         # Count examples exactly
195         validation_count: int = int(
196             np.ceil(self.validation_fraction * len
(output_examples))
197         )
198         test_count: int = int(np.ceil(self.test_fra
ction * len(output_examples)))
199         train_count = len(output_examples) - (valid
ation_count + test_count)
200
201         # Make sure no split is below zero unless t
he user wants it
202         assert validation_count > 0 or self.validat
ion_fraction == 0.0
203         assert test_count > 0 or self.test_fraction
== 0.0
204         assert train_count > 0
205
206         # Shuffle examples
207         rs = np.random.RandomState(self.config["see
d"])
208         rs.shuffle(output_examples)
209
210         for i, output_example in enumerate(output_e
xamples):
211             if i <= train_count:
212                 output_example["Split"] = "Trainin
g"
213             elif i <= (train_count + validation_cou
nt):
214                 output_example["Split"] = "Validati
on"
215             else:
216                 output_example["Split"] = "Test"
217
218             self.logger.info(
219                 f"Assigned {len(output_examples)} exam
ples to {train_count} training, {validation_count} v
alidation and {test_count} test examples"
220             )
221
222             output["examples"] = output_examples
223
224     def output_ready(self, output: dict) -> typing.
Optional[bool]:
225         return "examples" in output.keys() and all(
226             ["Split" in example.keys() for example
in output["examples"]]
227         )
228
229     __all__ = [
230         "ValidateTestcaseFilesPipelineStep",
231         "ExtractExamplesPipelineStep",
232         "LabelExamplesPipelineStep",
233         "RemoveDuplicateExamplesPipelineStep",
234         "AssignSplitToExamplesPipelineStep",
235     ]
236 ]
237

```

```

208     def input_ready(self, input: dict) -> bool:
209         return "examples" in input.keys()
210
211     def execute(self, input: dict, output: dict) ->
None:
212         import numpy as np
213
214         output_examples = list(input["examples"])
215
216         # Count examples exactly
217         validation_count: int = int(
218             np.ceil(self.validation_fraction * len
(output_examples))
219         )
220         test_count: int = int(np.ceil(self.test_fra
ction * len(output_examples)))
221         train_count = len(output_examples) - (valid
ation_count + test_count)
222
223         # Make sure no split is below zero unless t
he user wants it
224         assert validation_count > 0 or self.validat
ion_fraction == 0.0
225         assert test_count > 0 or self.test_fraction
== 0.0
226         assert train_count > 0
227
228         # Shuffle examples
229         rs = np.random.RandomState(self.config["see
d"])
230         rs.shuffle(output_examples)
231
232         for i, output_example in enumerate(output_e
xamples):
233             if i <= train_count:
234                 output_example["Split"] = "Trainin
g"
235             elif i <= (train_count + validation_cou
nt):
236                 output_example["Split"] = "Validati
on"
237             else:
238                 output_example["Split"] = "Test"
239
240             self.logger.info(
241                 f"Assigned {len(output_examples)} exam
ples to {train_count} training, {validation_count} v
alidation and {test_count} test examples"
242             )
243
244             output["examples"] = output_examples
245
246     def output_ready(self, output: dict) -> typing.
Optional[bool]:
247         return "examples" in output.keys() and all(
248             ["Split" in example.keys() for example
in output["examples"]]
249         )
250
251     __all__ = [
252         "ValidateTestcaseFilesPipelineStep",
253         "ExtractExamplesPipelineStep",
254         "LabelExamplesPipelineStep",
255         "RemoveDuplicateExamplesPipelineStep",
256         "AssignSplitToExamplesPipelineStep",
257     ]
258 ]
259

```

