

290 lines - 48 Removals

```

1 """
2     evaluation_pipeline_steps: Implementations of p
3     ipeline steps to calculate eval metrics and print t
4     ables.
5 """
6
7 import logging
8
9 import numpy as np
10
11 import preprocessing
12
13 from pipeline_framework import PipelineState, Pipel
14 ineStep
15
16 class EvaluatePredictionsPipelineStep(PipelineSte
17 p):
18     step_key = "evaluate_predictions"
19
20     def __init__(
21         self, config: dict, state: PipelineState, l
22 ogger: logging.Logger, split: str
23     ):
24         super().__init__(config, state, logger)
25
26         self.split = split
27         self.label_strategy = preprocessing.LabelSt
28 rategy[self.config["label_strategy"]]
29
30         self.evaluation_yaml_path = (
31             self.state.run_base_dir / f"evaluation_
32 {self.split}.yaml"
33         )
34
35     def input_ready(self, input: dict) -> bool:
36         return "examples" in input.keys() and any(
37             ["Prediction" in example.keys() for exa
38 mple in input["examples"]]
39         )
40
41     def execute(self, input: dict, output: dict) ->
42 None:
43     # Collect relevant data
44     split_examples = [ex for ex in input["examp
45 les"] if ex["Split"] == self.split]
46
47     # Overall evaluation
48     overall_result = self._evaluate_subset(spli
49 t_examples, input["label_encoding"])
50
51     # Per CWE
52     weakness_ids = [cwe["WeaknessID"] for cwe i
53 n input["cwes"]]
54     per_cwe_result = self._evaluate_subsets(
55         split_examples,
56         input["label_encoding"],
57         set(weakness_ids),
58         lambda ex, subset: ex["Testcase"]["Weak
59 ness"]["WeaknessID"] == subset,
60     )
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

298 lines + 77 Additions

```

1 """
2     evaluation_pipeline_steps: Implementations of p
3     ipeline steps to calculate eval metrics and print t
4     ables.
5 """
6
7 import logging
8
9 import numpy as np
10
11 import preprocessing
12
13 from pipeline_framework import PipelineState, Pipel
14 ineStep
15
16 class EvaluatePredictionsPipelineStep(PipelineSte
17 p):
18     step_key = "evaluate_predictions"
19
20     def __init__(
21         self, config: dict, state: PipelineState, l
22 ogger: logging.Logger, split: str
23     ):
24         super().__init__(config, state, logger)
25
26         self.split = split
27         self.label_strategy = preprocessing.LabelSt
28 rategy[self.config["label_strategy"]]
29
30         self.evaluation_yaml_path = (
31             self.state.run_base_dir / f"evaluation_
32 {self.split}.yaml"
33         )
34
35     def input_ready(self, input: dict) -> bool:
36         return "examples" in input.keys() and any(
37             ["Prediction" in example.keys() for exa
38 mple in input["examples"]]
39         )
40
41     def execute(self, input: dict, output: dict) ->
42 None:
43     # Collect relevant data
44     split_examples = [ex for ex in input["examp
45 les"] if ex["Split"] == self.split]
46
47     # Overall evaluation
48     overall_result = self._evaluate_subset(spli
49 t_examples, input["label_encoding"])
50
51     if not self.state.malware:
52         # Per CWE
53         weakness_ids = [cwe["WeaknessID"] for c
54 we in input["cwes"]]
55         per_cwe_result = self._evaluate_subsets
56 (
57     split_examples,
58     input["label_encoding"],
59     set(weakness_ids),
60     lambda ex, subset: ex["Testcase"]
61 ["Weakness"]["WeaknessID"] == subset,
62 )
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

48         # Per Flow Variant
49         flow_variants = set(
50             [testcase["FlowVariant"] for testcase i
n input["testcases"]]
51         )
52         per_flow_variant_result = self._evaluate_su
bsets(
53             split_examples,
54             input["label_encoding"],
55             flow_variants,
56             lambda ex, subset: ex["Testcase"]["Flow
Variant"] == subset,
57         )

```

```

58
59         bvdetector_subsets = {
60             "MC": {
61                 120,
62                 124,
63                 126,
64                 127,
65                 129,
66                 134,
67                 170,
68                 415,
69                 416,
70                 590,
71                 761,
72                 785,
73                 805,
74                 806,
75                 822,
76                 824,
77                 843,
78             },
79             "NH": {190, 191, 194, 195, 196, 197, 36
9, 682, 839},
80         }
81         bvdetector_subsets["MC&NH"] = bvdetector_su
bsets["MC"].union(
82             bvdetector_subsets["NH"]
83         )
84

```

```

85         bvdetector_subset_result = self._evaluate_s
ubsets(
86             split_examples,
87             input["label_encoding"],
88             bvdetector_subsets.keys(),
89             lambda ex, subset: ex["Testcase"]["Weak
ness"]["WeaknessID"]
90             in bvdetector_subsets[subset],
91         )
92

```

```

93         evaluation = {
94             "Overall": overall_result,
95             "PerCWE": per_cwe_result,
96             "PerFlowVariant": per_flow_variant_resu
lt,
97             "BVDetector": bvdetector_subset_result,
98         }

```

```

49         # Per Flow Variant
50         flow_variants = set(
51             [testcase["FlowVariant"] for testca
se in input["testcases"]]
52         )
53         per_flow_variant_result = self._evaluat
e_subsets(
54             split_examples,
55             input["label_encoding"],
56             flow_variants,
57             lambda ex, subset: ex["Testcase"]
["FlowVariant"] == subset,
58         )
59

```

```

60         bvdetector_subsets = {
61             "MC": {
62                 120,
63                 124,
64                 126,
65                 127,
66                 129,
67                 134,
68                 170,
69                 415,
70                 416,
71                 590,
72                 761,
73                 785,
74                 805,
75                 806,
76                 822,
77                 824,
78                 843,
79             },
80             "NH": {190, 191, 194, 195, 196, 19
7, 369, 682, 839},
81         }
82         bvdetector_subsets["MC&NH"] = bvdetecto
r_subsets["MC"].union(
83             bvdetector_subsets["NH"]
84         )
85

```

```

86         bvdetector_subset_result = self._evalua
te_subsets(
87             split_examples,
88             input["label_encoding"],
89             bvdetector_subsets.keys(),
90             lambda ex, subset: ex["Testcase"]
["Weakness"]["WeaknessID"]
91             in bvdetector_subsets[subset],
92         )
93

```

```

94         evaluation = {
95             "Overall": overall_result,
96             "PerCWE": per_cwe_result,
97             "PerFlowVariant": per_flow_variant_
result,
98             "BVDetector": bvdetector_subset_res
ult,
99         }
100         else:
101             evaluation = {
102                 "Overall": overall_result,
103                 "PerCWE": overall_result,
104                 "PerFlowVariant": overall_result,
105                 "BVDetector": overall_result,

```

```

99
100     self.logger.info(f"Evaluation: {evaluation}")
101
102     # Save to YAML
103     with open(self.evaluation_yaml_path, "w") as evaluation_yaml:
104         import yaml
105
106         yaml.dump(evaluation, evaluation_yaml)
107
108     if "evaluation" in output.keys():
109         output["evaluation"][self.split] = evaluation
110     else:
111         output["evaluation"] = {self.split: evaluation}
112
113     def _evaluate_subsets(
114         self,
115         examples: list[dict],
116         label_encoding: dict,
117         subsets: set[str],
118         subset_filter,
119     ):
120         subsets_results = {}
121
122         for subset in subsets:
123             subset_examples = [ex for ex in examples if subset_filter(ex, subset)]
124             subsets_results[subset] = self._evaluate_subset(
125                 subset_examples, label_encoding
126             )
127
128         return subsets_results
129
130     def _evaluate_subset(self, examples: list[dict], label_encoding: dict):
131         if self.label_strategy == preprocessing.LabelStrategy.BINARYCLASSIFICATION:
132             return self._evaluate_subset_binary(examples, label_encoding)
133         else:
134             return self._evaluate_subset_multiclass(examples, label_encoding)
135
136     def _evaluate_subset_binary(self, examples: list[dict], label_encoding: dict):
137         confusion_matrix = self._calculate_confusion_matrix(examples, label_encoding)
138
139         true_positives = float(confusion_matrix[1, 1])
140         false_positives = float(confusion_matrix[0, 1])
141         false_negatives = float(confusion_matrix[1, 0])
142         true_negatives = float(confusion_matrix[0, 0])
143
144         total_examples = (
145             true_positives + false_positives + false_negatives + true_negatives
146         )
147
148         # Accuracy
149         if total_examples > 0:

```

```

106     }
107
108     self.logger.info(f"Evaluation: {evaluation}")
109
110     # Save to YAML
111     with open(self.evaluation_yaml_path, "w") as evaluation_yaml:
112         import yaml
113
114         yaml.dump(evaluation, evaluation_yaml)
115
116     if "evaluation" in output.keys():
117         output["evaluation"][self.split] = evaluation
118     else:
119         output["evaluation"] = {self.split: evaluation}
120
121     def _evaluate_subsets(
122         self,
123         examples: list[dict],
124         label_encoding: dict,
125         subsets: set[str],
126         subset_filter,
127     ):
128         subsets_results = {}
129
130         for subset in subsets:
131             subset_examples = [ex for ex in examples if subset_filter(ex, subset)]
132             subsets_results[subset] = self._evaluate_subset(
133                 subset_examples, label_encoding
134             )
135
136         return subsets_results
137
138     def _evaluate_subset(self, examples: list[dict], label_encoding: dict):
139         if self.label_strategy == preprocessing.LabelStrategy.BINARYCLASSIFICATION:
140             return self._evaluate_subset_binary(examples, label_encoding)
141         else:
142             return self._evaluate_subset_multiclass(examples, label_encoding)
143
144     def _evaluate_subset_binary(self, examples: list[dict], label_encoding: dict):
145         confusion_matrix = self._calculate_confusion_matrix(examples, label_encoding)
146
147         true_positives = float(confusion_matrix[1, 1])
148         false_positives = float(confusion_matrix[0, 1])
149         false_negatives = float(confusion_matrix[1, 0])
150         true_negatives = float(confusion_matrix[0, 0])
151
152         total_examples = (
153             true_positives + false_positives + false_negatives + true_negatives
154         )
155
156         # Accuracy
157         if total_examples > 0:

```

```

149         accuracy = (true_positives + true_negat
150 ives) / (
151             true_positives + false_positives +
152             false_negatives + true_negatives
153         )
154     else:
155         accuracy = None
156
157     # Precision
158     if (true_positives + false_positives) > 0:
159         precision = true_positives / (true_posi
160 tives + false_positives)
161     else:
162         precision = None
163
164     # Recall, TPR, FNR
165     if (true_positives + false_negatives) > 0:
166         recall = true_positives / (true_positiv
167 es + false_negatives)
168     else:
169         recall = None
170
171     true_positive_rate = true_positives /
172 (true_positives + false_negatives)
173     false_negative_rate = false_negatives /
174 (false_negatives + true_positives)
175     else:
176         recall = None
177         true_positive_rate = None
178         false_negative_rate = None
179
180     # FPR, TNR
181     if (true_negatives + false_positives) > 0:
182         false_positive_rate = false_positives /
183 (false_positives + true_negatives)
184     else:
185         true_negative_rate = true_negatives /
186 (true_negatives + false_positives)
187     else:
188         false_positive_rate = None
189         true_negative_rate = None
190
191     # F1
192     if precision is not None and recall is not
193 None:
194         if (precision + recall) > 0:
195             f1 = 2 * precision * recall / (prec
196 ision + recall)
197         else:
198             f1 = 0.0
199         else:
200             f1 = None
201
202     return {
203         "Accuracy": accuracy,
204         "Precision": precision,
205         "Recall": recall,
206         "F1": f1,
207         "TotalExamples": int(total_examples),
208         "TotalPositives": int(true_positives +
209 false_negatives),
210         "TotalNegatives": int(true_negatives +
211 false_positives),
212         "TruePositiveRate": true_positive_rate,
213         "FalsePositiveRate": false_positive_rat
214 e,
215         "FalseNegativeRate": false_negative_rat
216 e,
217         "TrueNegativeRate": true_negative_rate,
218     }

```

```

157         accuracy = (true_positives + true_negat
158 ives) / (
159             true_positives + false_positives +
160             false_negatives + true_negatives
161         )
162     else:
163         accuracy = None
164
165     # Precision
166     if (true_positives + false_positives) > 0:
167         precision = true_positives / (true_posi
168 tives + false_positives)
169     else:
170         precision = None
171
172     # Recall, TPR, FNR
173     if (true_positives + false_negatives) > 0:
174         recall = true_positives / (true_positiv
175 es + false_negatives)
176     else:
177         recall = None
178
179     true_positive_rate = true_positives /
180 (true_positives + false_negatives)
181     false_negative_rate = false_negatives /
182 (false_negatives + true_positives)
183     else:
184         recall = None
185         true_positive_rate = None
186         false_negative_rate = None
187
188     # FPR, TNR
189     if (true_negatives + false_positives) > 0:
190         false_positive_rate = false_positives /
191 (false_positives + true_negatives)
192     else:
193         true_negative_rate = true_negatives /
194 (true_negatives + false_positives)
195     else:
196         false_positive_rate = None
197         true_negative_rate = None
198
199     # F1
200     if precision is not None and recall is not
201 None:
202         if (precision + recall) > 0:
203             f1 = 2 * precision * recall / (prec
204 ision + recall)
205         else:
206             f1 = 0.0
207         else:
208             f1 = None
209
210     return {
211         "Accuracy": accuracy,
212         "Precision": precision,
213         "Recall": recall,
214         "F1": f1,
215         "TotalExamples": int(total_examples),
216         "TotalPositives": int(true_positives +
217 false_negatives),
218         "TotalNegatives": int(true_negatives +
219 false_positives),
220         "TruePositiveRate": true_positive_rate,
221         "FalsePositiveRate": false_positive_rat
222 e,
223         "FalseNegativeRate": false_negative_rat
224 e,
225         "TrueNegativeRate": true_negative_rate,
226     }

```

```

201
202     def _evaluate_subset_multiclass(self, examples:
list[dict], label_encoding: dict):
203         confusion_matrix = self._calculate_confusio
n_matrix(examples, label_encoding)
204
205         total_examples = float(confusion_matrix.sum
())
206         if total_examples > 0:
207             accuracy = float(confusion_matrix.diago
nal().sum()) / total_examples
208         else:
209             return None
210
211         return {"Accuracy": accuracy, "TotalExample
s": total_examples}
212
213     def _calculate_confusion_matrix(self, examples:
list[dict], label_encoding: dict):
214         label_count = len(label_encoding.keys())
215         confusion_matrix = np.zeros((label_count, l
abel_count), dtype=np.int64)
216
217         for example in examples:
218             prediction = example["Prediction"]
219             label = example["Label"]
220
221             label_idx = label_encoding[label]
222             prediction_idx = label_encoding[predict
ion]
223
224             confusion_matrix[label_idx, prediction_
idx] += 1
225
226         return confusion_matrix
227
228
229 class PrintEvaluationTablePipelineStep(PipelineSte
p):
230     step_key = "print_evaluation_table"
231
232     def __init__(
233         self,
234         config: dict,
235         state: PipelineState,
236         logger: logging.Logger,
237         collection: str,
238         split: str,
239         keys: list[str],
240     ):
241         super().__init__(config, state, logger)
242
243         self.collection = collection
244         self.keys = keys
245         self.split = split
246
247     def input_ready(self, input: dict) -> bool:
248         return (
249             "evaluation" in input.keys()
250             and self.split in input["evaluation"].k
eys()
251             and self.collection in input["evaluatio
n"][self.split].keys()
252         )
253
254     def execute(self, input: dict, output: dict) ->
None:

```

```

209
210     def _evaluate_subset_multiclass(self, examples:
list[dict], label_encoding: dict):
211         confusion_matrix = self._calculate_confusio
n_matrix(examples, label_encoding)
212
213         total_examples = float(confusion_matrix.sum
())
214         if total_examples > 0:
215             accuracy = float(confusion_matrix.diago
nal().sum()) / total_examples
216         else:
217             return None
218
219         return {"Accuracy": accuracy, "TotalExample
s": total_examples}
220
221     def _calculate_confusion_matrix(self, examples:
list[dict], label_encoding: dict):
222         label_count = len(label_encoding.keys())
223         confusion_matrix = np.zeros((label_count, l
abel_count), dtype=np.int64)
224
225         for example in examples:
226             prediction = example["Prediction"]
227             label = example["Label"]
228
229             label_idx = label_encoding[label]
230             prediction_idx = label_encoding[predict
ion]
231
232             confusion_matrix[label_idx, prediction_
idx] += 1
233
234         return confusion_matrix
235
236
237 class PrintEvaluationTablePipelineStep(PipelineSte
p):
238     step_key = "print_evaluation_table"
239
240     def __init__(
241         self,
242         config: dict,
243         state: PipelineState,
244         logger: logging.Logger,
245         collection: str,
246         split: str,
247         keys: list[str],
248     ):
249         super().__init__(config, state, logger)
250
251         self.collection = collection
252         self.keys = keys
253         self.split = split
254
255     def input_ready(self, input: dict) -> bool:
256         return (
257             "evaluation" in input.keys()
258             and self.split in input["evaluation"].k
eys()
259             and self.collection in input["evaluatio
n"][self.split].keys()
260         )
261
262     def execute(self, input: dict, output: dict) ->
None:

```

```

255         evaluation_of_collection = input["evaluation
n"][self.split][self.collection]
256         subsets = list(sorted(evaluation_of_collect
ion.keys()))
257
258         table_tuples = []
259
260         # Compose table header
261         table_tuples += [("Subset",) + tuple(self.k
eys)]
262         table_tuples += [tuple(["-"] * (len(self.ke
ys) + 1))]
263
264         # Compose table rows
265         for subset in subsets:
266             row = [str(subset)]
267             for key in self.keys:
268                 if isinstance(evaluation_of_collect
ion[subset][key], float):
269                     row += (f"{evaluation_of_collec
tion[subset][key]:0.4f}",)
270                 else:
271                     row += (str(evaluation_of_colle
ction[subset][key]),)
272
273             table_tuples.append(row)
274
275         # Convert table to strings
276         table_strings = []
277         for table_tuple in table_tuples:
278             table_strings.append(
279                 "| " + " | ".join([f"{str(el):15s}"
for el in table_tuple]) + " |"
280             )
281
282         # Print table strings to logger
283         self.logger.info(
284             f"Evaluation table of split {self.spli
t} (collection {self.collection}): \n"
285             + "\n".join(table_strings)
286         )
287
288
289 __all__ = ["EvaluatePredictionsPipelineStep", "Prin
tEvaluationTablePipelineStep"]
290

```

```

263         evaluation_of_collection = input["evaluation
n"][self.split][self.collection]
264         subsets = list(sorted(evaluation_of_collect
ion.keys()))
265
266         table_tuples = []
267
268         # Compose table header
269         table_tuples += [("Subset",) + tuple(self.k
eys)]
270         table_tuples += [tuple(["-"] * (len(self.ke
ys) + 1))]
271
272         # Compose table rows
273         for subset in subsets:
274             row = [str(subset)]
275             for key in self.keys:
276                 if isinstance(evaluation_of_collect
ion[subset][key], float):
277                     row += (f"{evaluation_of_collec
tion[subset][key]:0.4f}",)
278                 else:
279                     row += (str(evaluation_of_colle
ction[subset][key]),)
280
281             table_tuples.append(row)
282
283         # Convert table to strings
284         table_strings = []
285         for table_tuple in table_tuples:
286             table_strings.append(
287                 "| " + " | ".join([f"{str(el):15s}"
for el in table_tuple]) + " |"
288             )
289
290         # Print table strings to logger
291         self.logger.info(
292             f"Evaluation table of split {self.spli
t} (collection {self.collection}): \n"
293             + "\n".join(table_strings)
294         )
295
296
297 __all__ = ["EvaluatePredictionsPipelineStep", "Prin
tEvaluationTablePipelineStep"]
298

```