237 lines - 9 Removals

```
"""
preprocessing_pipeline_steps: Implementation of pip
eline steps to extract examples from testcases and
 construct a dataset.
"""

import logging
import re
import typing


from pipeline_framework import PipelineState, Pipel
ineStep









class ValidateTestcaseFilesPipelineStep(PipelineSte
p):
    step_key = "validate_testcase_files"

    def input_ready(self, input: dict) -> bool:
        return "testcases" in input.keys()

    def execute(self, input: dict, output: dict) ->
None:
        import preprocessing

        output_testcases = []
        for testcase in input["testcases"]:
            testcase["Valid"] = preprocessing.valid
ate_testcase_files(testcase)
            output_testcases += [testcase]

        output["testcases"] = output_testcases

    def output_ready(self, output: dict) -> typing.
Optional[bool]:
        return "testcases" in output.keys() and all
(
            [
                "Valid" in testcase.keys() and test
case["Valid"]
                for testcase in output["testcases"]
            ]
        )


class ExtractExamplesPipelineStep(PipelineStep):
    step_key = "extract_examples"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        emit_primary_good_function: bool = True,
```

259 lines + 36 Additions

```
"""
preprocessing_pipeline_steps: Implementation of pip
eline steps to extract examples from testcases and
 construct a dataset.
"""

import logging
import re
import typing
import warnings

from pipeline_framework import PipelineState, Pipel
ineStep

# warnings.filterwarnings('ignore',
#                         message="Could not find s
ymbol __stack_chk_fail in symbol translation table
 for disassembly")
# logging.basicConfig(filename='warnings.log', leve
l=logging.WARNING)
# warnings.filterwarnings('ignore', category=UserWa
rning)


class ValidateTestcaseFilesPipelineStep(PipelineSte
p):
    step_key = "validate_testcase_files"

    def input_ready(self, input: dict) -> bool:
        return "testcases" in input.keys()

    def execute(self, input: dict, output: dict) ->
None:
        import preprocessing

        output_testcases = []
        for testcase in input["testcases"]:
            testcase["Valid"] = preprocessing.valid
ate_testcase_files(testcase)
            output_testcases += [testcase]

        output["testcases"] = output_testcases

    def output_ready(self, output: dict) -> typing.
Optional[bool]:
        return "testcases" in output.keys() and all
(
            [
                "Valid" in testcase.keys() and test
case["Valid"]
                for testcase in output["testcases"]
            ]
        )


class ExtractExamplesPipelineStep(PipelineStep):
    step_key = "extract_examples"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        emit_primary_good_function: bool = True,
```

```
        label_granularity: typing.Optional[str] = None,
        context_granularity: typing.Optional[str] = None,
    ):
        super().__init__(config, state, logger)

        import preprocessing

        self.extract_fn = lambda testcase: preprocessing.extract_examples(
            testcase,
            label_granularity=preprocessing.Granularity[label_granularity]
            if label_granularity is not None
            else None,
            context_granularity=preprocessing.Granularity[context_granularity]
            if context_granularity is not None
            else None,
            emit_primary_good_function=emit_primary_good_function,
            logger=self.logger,
        )


    def input_ready(self, input: dict) -> bool:
        return (
            "cwes" in input.keys()
            and "testcases" in input.keys()
            and all(
                [
                    "Valid" in testcase.keys() and testcase["Valid"]
                    for testcase in input["testcases"]
                ]
            )
        )

    def execute(self, input: dict, output: dict) -> None:
        output_examples = []
        for testcase in input["testcases"]:
            output_examples += self.extract_fn(testcase)
```

```
        label_granularity: typing.Optional[str] = None,
        context_granularity: typing.Optional[str] = None,
    ):
        super().__init__(config, state, logger)

        import preprocessing

        if self.state.malware:

            self.extract_fn = lambda testcase: preprocessing.extract_examples_malware(
                testcase,
                label_granularity=preprocessing.Granularity[label_granularity]
                if label_granularity is not None
                else None,
                context_granularity=preprocessing.Granularity[context_granularity]
                if context_granularity is not None
                else None,
                emit_primary_good_function=emit_primary_good_function,
                logger=self.logger
            )
        else:
            self.extract_fn = lambda testcase: preprocessing.extract_examples(
                testcase,
                label_granularity=preprocessing.Granularity[label_granularity]
                if label_granularity is not None
                else None,
                context_granularity=preprocessing.Granularity[context_granularity]
                if context_granularity is not None
                else None,
                emit_primary_good_function=emit_primary_good_function,
                logger=self.logger
            )


    def input_ready(self, input: dict) -> bool:
        return (
            "cwes" in input.keys()
            and "testcases" in input.keys()
            and all(
                [
                    "Valid" in testcase.keys() and testcase["Valid"]
                    for testcase in input["testcases"]
                ]
            )
        )

    def execute(self, input: dict, output: dict) -> None:
        output_examples = []
        for testcase in input["testcases"]:
            try:
```

```
                output_examples += self.extract_fn(testcase)
            except Exception as ex:
                print(ex)

        self.logger.info(
            f"Extracted {len(output_examples)} examples from {len(input['testcases'])} testcases"
        )

        output["examples"] = output_examples

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return "examples" in output.keys()


class LabelExamplesPipelineStep(PipelineStep):
    step_key = "label_examples"

    def input_ready(self, input: dict) -> bool:
        return "examples" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import preprocessing

        label_strategy = preprocessing.LabelStrategy[self.config["label_strategy"]]

        output_examples = []
        for example in input["examples"]:
            example["Label"] = preprocessing.label_example(example, label_strategy)
            output_examples += [example]

        output["examples"] = output_examples

        if label_strategy == preprocessing.LabelStrategy.BINARYCLASSIFICATION:
            output["label_encoding"] = {"Good": 0, "Bad": 1}
        else:
            raise NotImplementedError("Labelstrategy not implemented!")

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return (
            "examples" in output.keys()
            and all(["Label" in example.keys() for example in output["examples"]])
            and "label_encoding" in output.keys()
        )


class RemoveDuplicateExamplesPipelineStep(PipelineStep):
    step_key = "remove_duplicate_examples"

    def input_ready(self, input: dict) -> bool:
        return "examples" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import numpy as np

        # Some accounting
        observed_representations = set()
```

```
        kept_examples = 0

        # Shuffle examples to avoid bias from ordering
        shuffled_input_examples = list(input["examples"])
        rs = np.random.RandomState(self.config["seed"])
        rs.shuffle(shuffled_input_examples)

        # Mark which examples are kept
        for example in input["examples"]:
            # Descramble examples to reinstate non-unique representations
            normalized_text_representation = re.sub(
                r"!lc[0-9]+:", "", example["Example"]
            )

            # Use the power of python's set implementation to check for uniqueness
            if normalized_text_representation not in observed_representations:
                observed_representations |= {normalized_text_representation}

                # Mark example as kept
                example["Keep"] = True
                kept_examples += 1
            else:
                example["Keep"] = False

        self.logger.info(
            f"Duplicate examples removed: {len(input['examples']) - kept_examples} of {len(input['examples'])}. Remaining: {kept_examples}"
        )

        # Preserve the order of the examples
        output["examples"] = [ex for ex in input["examples"] if ex["Keep"]]
        assert len(output["examples"]) == kept_examples

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return "examples" in output.keys()


class AssignSplitToExamplesPipelineStep(PipelineStep):
    step_key = "assign_split_to_examples"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        validation_fraction: float,
        test_fraction: float,
    ):
        super().__init__(config, state, logger)

        self.validation_fraction = validation_fraction
        self.test_fraction = test_fraction
```

```
        kept_examples = 0

        # Shuffle examples to avoid bias from ordering
        shuffled_input_examples = list(input["examples"])
        rs = np.random.RandomState(self.config["seed"])
        rs.shuffle(shuffled_input_examples)

        # Mark which examples are kept
        for example in input["examples"]:
            # Descramble examples to reinstate non-unique representations
            normalized_text_representation = re.sub(
                r"!lc[0-9]+:", "", example["Example"]
            )

            # Use the power of python's set implementation to check for uniqueness
            if normalized_text_representation not in observed_representations:
                observed_representations |= {normalized_text_representation}

                # Mark example as kept
                example["Keep"] = True
                kept_examples += 1
            else:
                example["Keep"] = False

        self.logger.info(
            f"Duplicate examples removed: {len(input['examples']) - kept_examples} of {len(input['examples'])}. Remaining: {kept_examples}"
        )

        # Preserve the order of the examples
        output["examples"] = [ex for ex in input["examples"] if ex["Keep"]]
        assert len(output["examples"]) == kept_examples

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return "examples" in output.keys()


class AssignSplitToExamplesPipelineStep(PipelineStep):
    step_key = "assign_split_to_examples"

    def __init__(
        self,
        config: dict,
        state: PipelineState,
        logger: logging.Logger,
        validation_fraction: float,
        test_fraction: float,
    ):
        super().__init__(config, state, logger)

        self.validation_fraction = validation_fraction
        self.test_fraction = test_fraction
```

```
    def input_ready(self, input: dict) -> bool:
        return "examples" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import numpy as np

        output_examples = list(input["examples"])

        # Count examples exactly
        validation_count: int = int(
            np.ceil(self.validation_fraction * len(output_examples))
        )
        test_count: int = int(np.ceil(self.test_fraction * len(output_examples)))
        train_count = len(output_examples) - (validation_count + test_count)

        # Make sure no split is below zero unless the user wants it
        assert validation_count > 0 or self.validation_fraction == 0.0
        assert test_count > 0 or self.test_fraction == 0.0
        assert train_count > 0

        # Shuffle examples
        rs = np.random.RandomState(self.config["seed"])
        rs.shuffle(output_examples)

        for i, output_example in enumerate(output_examples):
            if i <= train_count:
                output_example["Split"] = "Training"
            elif i <= (train_count + validation_count):
                output_example["Split"] = "Validation"
            else:
                output_example["Split"] = "Test"

        self.logger.info(
            f"Assigned {len(output_examples)} examples to {train_count} training, {validation_count} validation and {test_count} test examples"
        )

        output["examples"] = output_examples

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return "examples" in output.keys() and all(
            ["Split" in example.keys() for example in output["examples"]]
        )


__all__ = [
    "ValidateTestcaseFilesPipelineStep",
    "ExtractExamplesPipelineStep",
    "LabelExamplesPipelineStep",
    "RemoveDuplicateExamplesPipelineStep",
    "AssignSplitToExamplesPipelineStep",
]
```


```
    def input_ready(self, input: dict) -> bool:
        return "examples" in input.keys()

    def execute(self, input: dict, output: dict) -> None:
        import numpy as np

        output_examples = list(input["examples"])

        # Count examples exactly
        validation_count: int = int(
            np.ceil(self.validation_fraction * len(output_examples))
        )
        test_count: int = int(np.ceil(self.test_fraction * len(output_examples)))
        train_count = len(output_examples) - (validation_count + test_count)

        # Make sure no split is below zero unless the user wants it
        assert validation_count > 0 or self.validation_fraction == 0.0
        assert test_count > 0 or self.test_fraction == 0.0
        assert train_count > 0

        # Shuffle examples
        rs = np.random.RandomState(self.config["seed"])
        rs.shuffle(output_examples)

        for i, output_example in enumerate(output_examples):
            if i <= train_count:
                output_example["Split"] = "Training"
            elif i <= (train_count + validation_count):
                output_example["Split"] = "Validation"
            else:
                output_example["Split"] = "Test"

        self.logger.info(
            f"Assigned {len(output_examples)} examples to {train_count} training, {validation_count} validation and {test_count} test examples"
        )

        output["examples"] = output_examples

    def output_ready(self, output: dict) -> typing.Optional[bool]:
        return "examples" in output.keys() and all(
            ["Split" in example.keys() for example in output["examples"]]
        )


__all__ = [
    "ValidateTestcaseFilesPipelineStep",
    "ExtractExamplesPipelineStep",
    "LabelExamplesPipelineStep",
    "RemoveDuplicateExamplesPipelineStep",
    "AssignSplitToExamplesPipelineStep",
]
```