

RoCELib

Robust Counterfactual Explanation Library

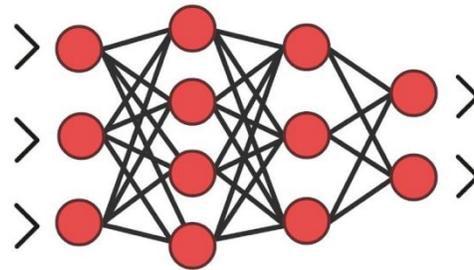
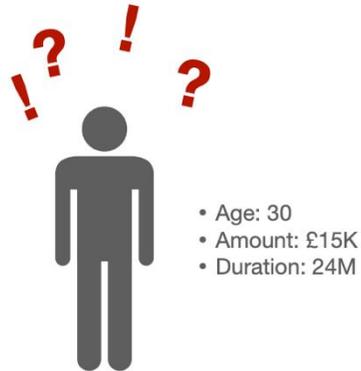
Ayush Patel, Lorenzo Evans, Shlok Shah,
James Stadler, Ivan Artiukhov, Seth Mack

Group 6

What is a Counterfactual
Explanation? (CE)

Why do we need counterfactual explanations?

- Explainable AI = problem
- Difficult to explain the outcomes of Neural Networks



Loan denied

DNNs are black boxes!

- User wants to know why loan has been denied so they can try change the outcome
- GDPR regulations – require meaningful explanations for automated decision making

What is a counterfactual explanation?

Original instance



- Age: 30
- Amount: £15K
- Duration: 24M

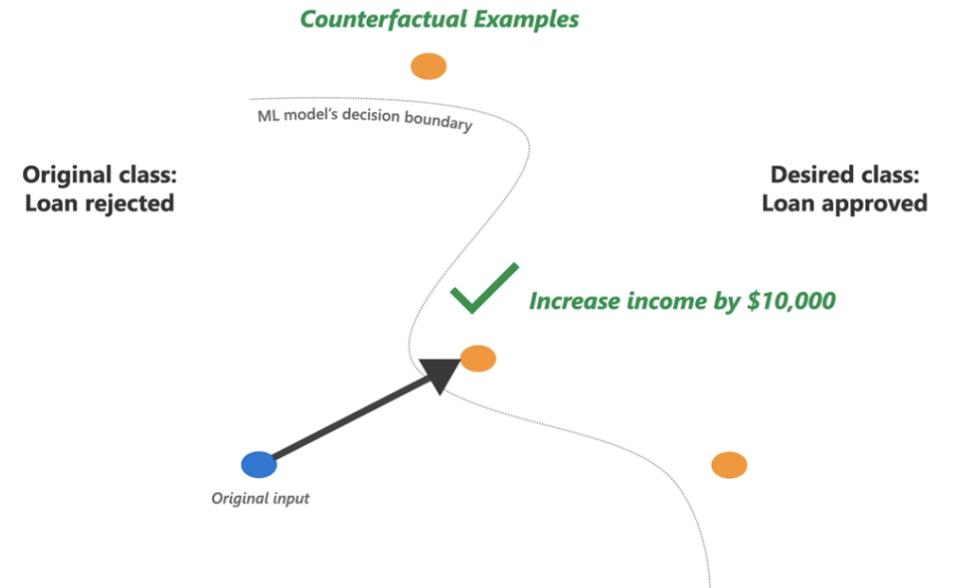
Loan denied

Counterfactual explanation



- Age: 30
- Amount: **£10K**
- Duration: 24M

The application would have been accepted
had you asked for £10K instead of £15K



- Counterfactual explanation: a what-if scenario that describes how a different outcome could have occurred, had the input been different.
- We can use Counterfactual Explanations to help explain the outcome of these black box models

Importance of Robustness of CEs

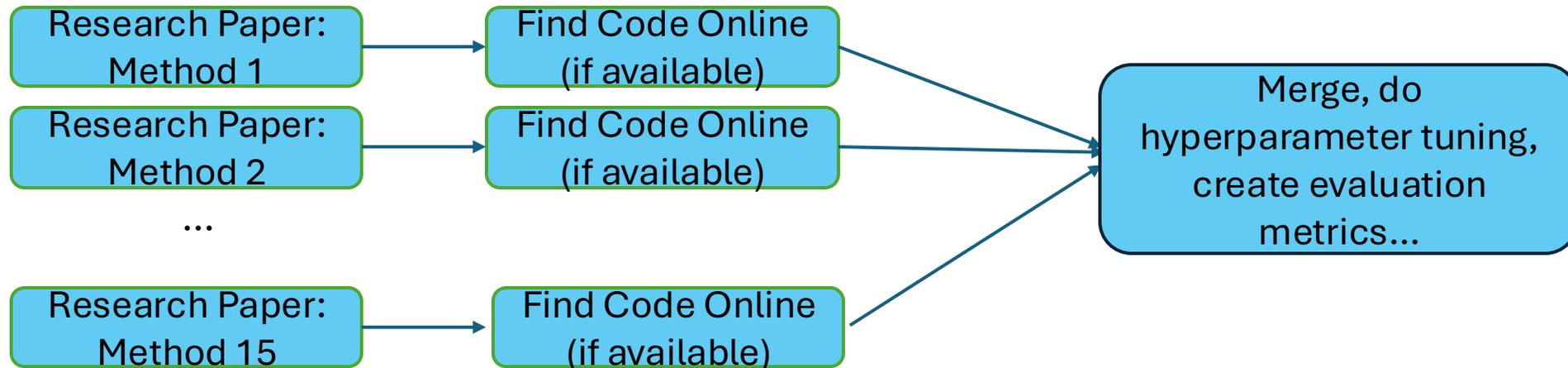
- Many **methods** for generating CEs
- Lots of ways to evaluate how 'good' these Counterfactual Explanations are
- Distance commonly used
- One way is measuring the **Robustness** of CEs
- Robustness = explanation remains consistent and reliable even when small changes are made to the input data
- Evaluating robustness is important:
 - Trustworthy interpretation
 - Fairness

Different Notions of Robustness

1. Input perturbations
2. Model perturbations
3. Model multiplicity
4. Noisy execution

Problem: Counterfactual Explanation research

- Many papers outlining different ways to generate CEs
- Very painful process to compare even 2 methods let alone the whole research space



- Very few ways to benchmark these methods currently
- Most libraries that do exist focus purely on evaluating the distance of CEs produced
- There is very little emphasis on evaluating the **Robustness** of Counterfactual Explanations

Introducing...

RoCELib!

Open-Source Python Library

Covers 15+ CE generation methods

Focus on robustness evaluation – 8+ metrics

Easy to extend

Glossary

A generation method is an algorithm used to produce counterfactual explanations

Robustness evaluation assesses the stability and reliability of counterfactual explanations

Initial Codebase

- The result of a UROP - bare bones implementation
- Issues included:
 - Very hard to understand how the library works and to use it
 - Filled with bugs
 - Not enough robustness metrics for evaluation

User Requirement Gathering

- Discussed with our supervisors about their needs, created a list of key features to add
- We were able to categorise the list of features into the desires of 3 potential stakeholders:

ML Engineer



- Ability to import trained models
- Generating Counterfactual explanations easily
- Digestible way of comparing methods

CE Researcher



- Extensibility: Adding own recourse method & evaluation metric

Library Maintainer



- Code Maintainability
- CI/CD
- Testing

Meet Sarah...

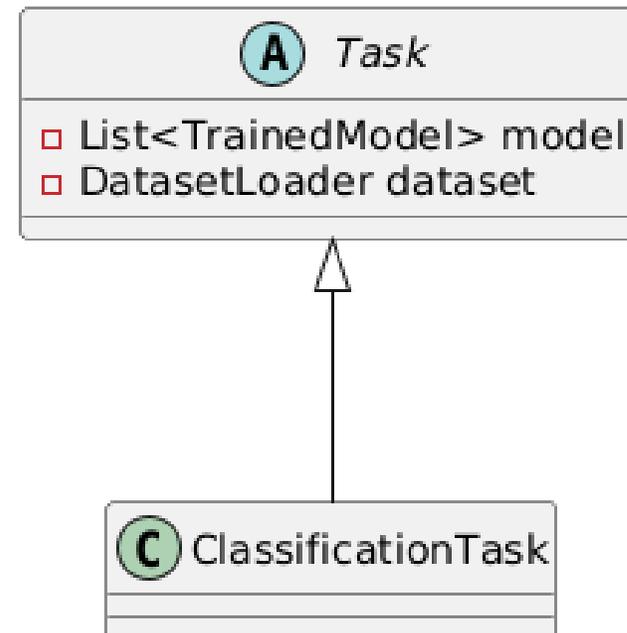
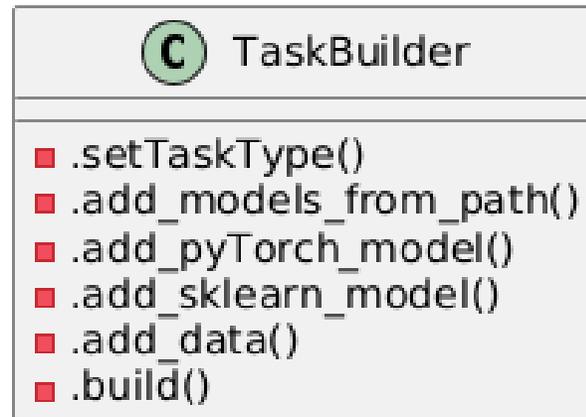


- Researcher who has developed a model for predicting outcomes of loan applications
- Needs to ensure robustness of her counterfactual explanations
- Rather than spending weeks cloning researchers' code and testing her model, she uses RoCELib...

Sarah's Need 1: Ability to import trained models

- Sarah wants to assess CE robustness across a set of models
- So needs to add multiple models to the library

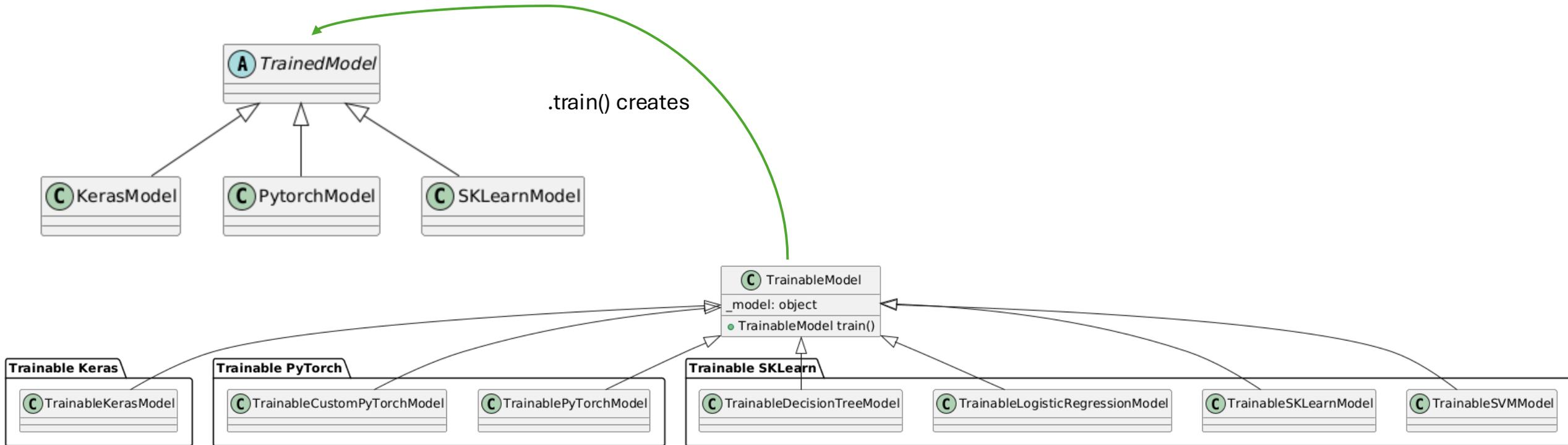
Task Redesign



Model Redesign

TrainedModel = Trained Model Imported from Sarah's computer

TrainableModel = Model we create explicitly within the library



Task Builder

C TaskBuilder
<ul style="list-style-type: none">■ .setTaskType()■ .add_models_from_path()■ .add_pyTorch_model()■ .add_sklearn_model()■ .add_keras_model()■ .add_data()

```
ct = (TaskBuilder()  
    .add_model_from_path("model.pkl")  
    .add_pytorch_model(34, [8], 1, dl)  
    .add_sklearn_model("svm")  
    .add_data(dl).build()  
)
```

- Makes Tasks **configurable**
- Allows the addition of **multiple models**
- Task creation in one line (**simplified user interaction**)

Sarah's Need 2: Generating Counterfactual explanations easily

Sarah wants to:

- **Generate CEs easily**, in a single line
- Evaluate CE generation methods using a **combination of metrics**

Original code:

- required you to **instantiate multiple classes** for each recourse method and evaluation metric
- **restricted data to Pandas** data frames only

Easy Generation of CEs

- Can now generate in one line with **any combination** of recourse methods and evaluation metrics using **strings**

```
ces = ct.generate(["MCE", "BinaryLinearSearch", "RNCE"])  
evals = ct.evaluate(["MCE", "RNCE"], ["DeltaRobustnessEvaluator", "Distance", "Validity"])
```

- Can **specify preferred datatype**

```
ces = ct.generate(["MCE"], "Tensor")
```

Sarah's need 3: Digestible way of comparing methods



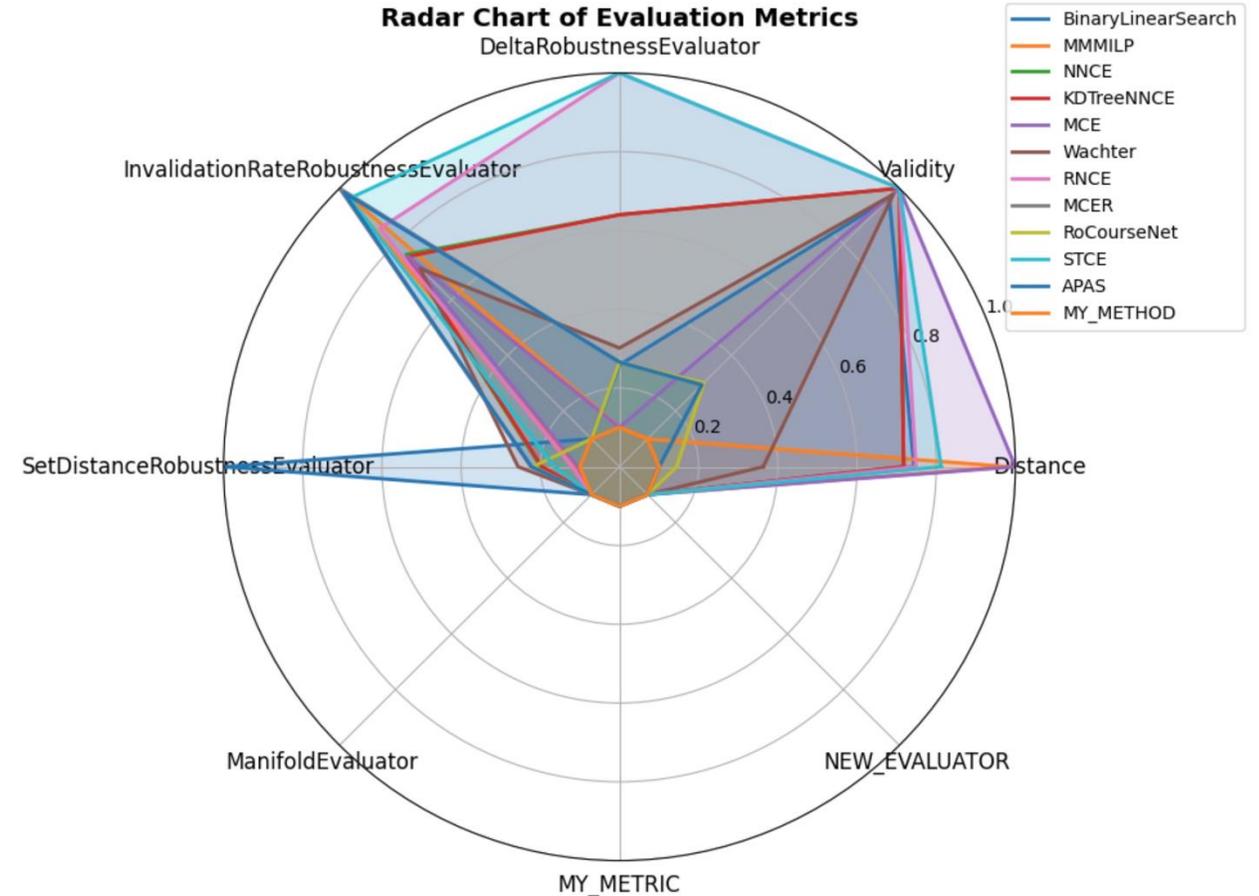
Doesn't have complete understanding of arbitrary metric values



Need to quickly choose the best one

Visualisation + exportation

- Radar graph allows quick comparison of metrics you compare about more
- Can quickly digest the method comparisons
- Results get exported to a .zip folder
- Cares about DeltaRobustness, STCE most well-rounded



Sarah's requirements

- Importing trained models



- Easy counterfactual generation



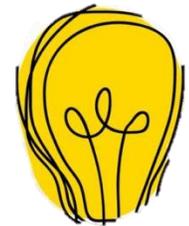
- Availability of visual methods comparison



Meet Kevin...



- Researcher who has an idea for a new counterfactual generation method
- He wants to evaluate the validity and robustness of his method against other generators in the research space
- He decides to use RoCELib...



Kevin's Need: Library Extensibility

Kevin **requires** the library to be **extensible** so that he can add his own:

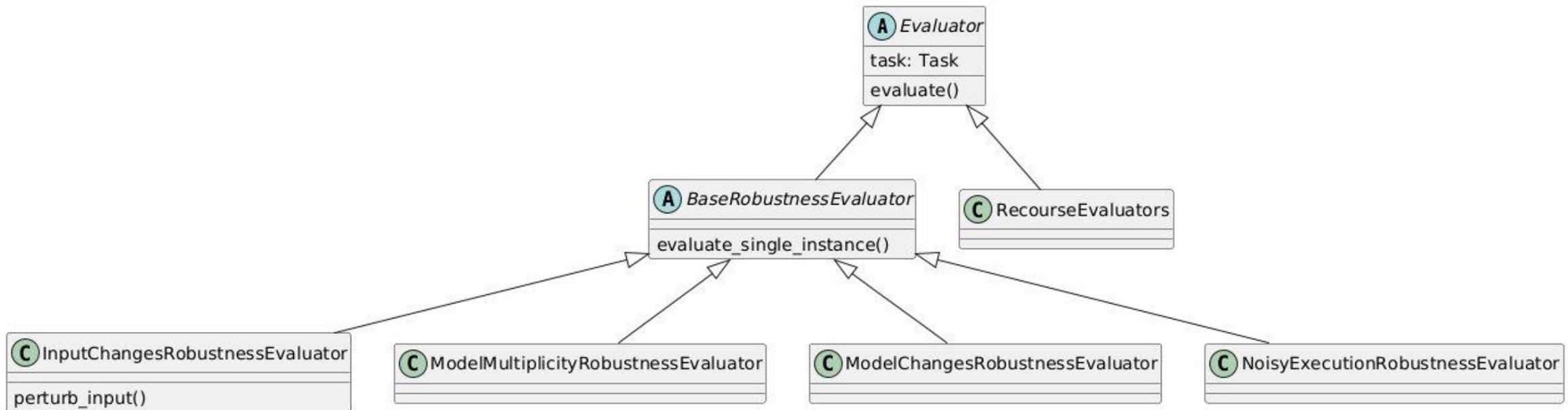
- recourse methods
- evaluation metric

However, the old codebase:

- Contains **tight coupling**
- Requires changes to underlying library to add recourse methods & metrics

Extensibility: Easy Addition of New Recourse Methods & Evaluation Metrics

- Can now easily add a new recourse method without changing the library
- User simply implements a single abstract method



Kevin's requirements

- Adding CE generation methods



- Adding evaluation metric



Deployment

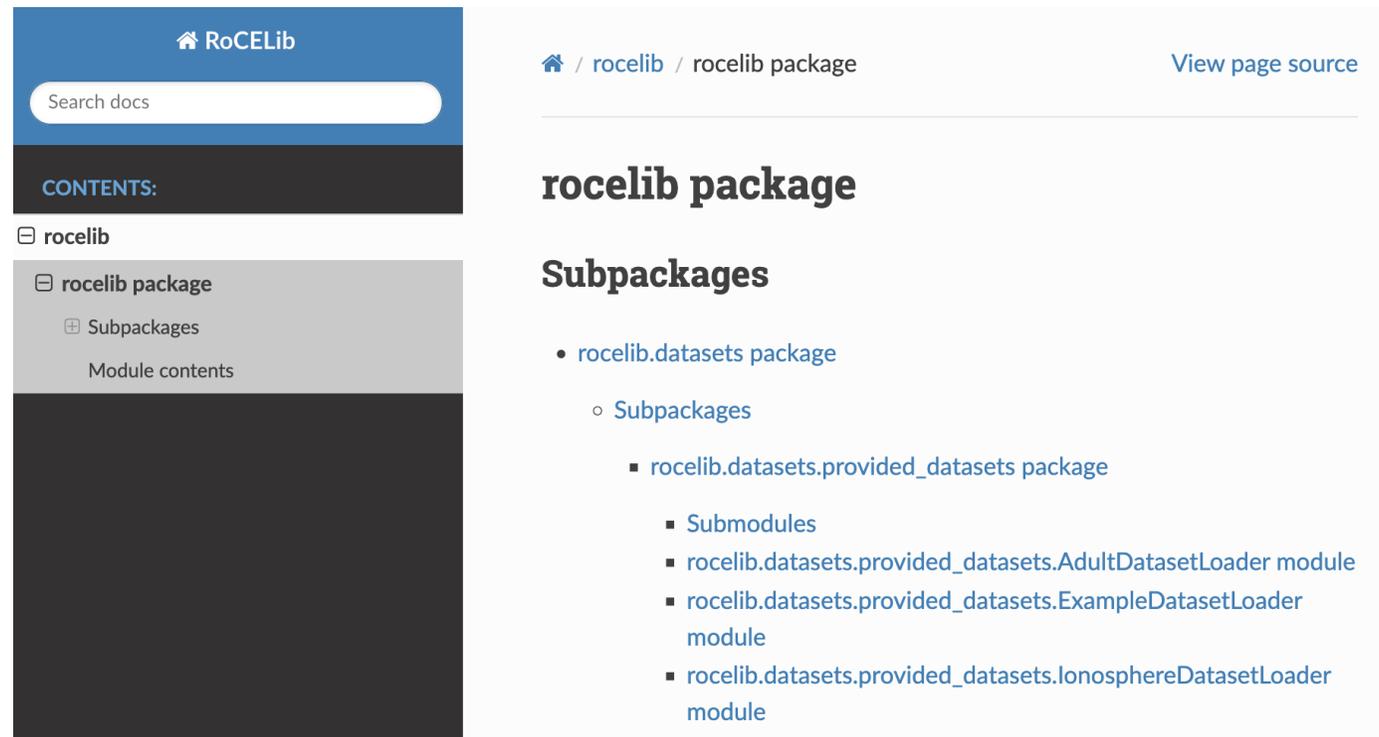
User Need: Being able to set-up library

- Little emphasis on usability
 - Lack of documentation
 - No packaging
- Difficult adding to existing project and understanding codebase

Documentation

<https://jstadi.github.io/RoCELib/rocelib.html>

- All functions documented
- Exhaustive Notebooks explaining how to use library, hosted on Google Colab



The screenshot displays the documentation page for the RoCELib package. On the left is a dark sidebar with a search bar and a 'CONTENTS:' section. The main content area on the right shows the breadcrumb path 'rocelib / rocelib package', the title 'rocelib package', and a 'Subpackages' section listing 'rocelib.datasets package' and its subpackages, including 'rocelib.datasets.provided_datasets package' with its submodules.

RoCELib

Search docs

CONTENTS:

- rocelib
 - rocelib package
 - Subpackages
 - Module contents

rocelib / rocelib package [View page source](#)

rocelib package

Subpackages

- rocelib.datasets package
 - Subpackages
 - rocelib.datasets.provided_datasets package
 - Submodules
 - rocelib.datasets.provided_datasets.AdultDatasetLoader module
 - rocelib.datasets.provided_datasets.ExampleDatasetLoader module
 - rocelib.datasets.provided_datasets.IonosphereDatasetLoader module

Packaging



The screenshot shows the PyPI project page for RoCELib 0.2.3. The header includes a search bar, navigation links for Help, Sponsors, Log in, and Register. The main content area features the project name 'RoCELib 0.2.3' with a 'Latest version' badge and a 'Released: Mar 6, 2025' date. A code block shows the command 'pip install RoCELib'. Below this is a description: 'RoCELib is an open-source Python library designed for benchmarking the robustness of counterfactual explanation (CE) methods.' The page is divided into a 'Navigation' sidebar with links for 'Project description', 'Release history', and 'Download files', and a 'Project description' main section. The 'Project description' section includes a 'Verified details' badge and a paragraph of text about the library's purpose and development.

```
pip install RoCELib
```

Packaging and deployment automated through GitHub actions

Evaluation: Deployment



Maintainability

Meet Howard...

- Maintainer/Contributor of library
- He wants to be able to contribute some new research he has made to the library for others to use
- Needs good developer ecosystem

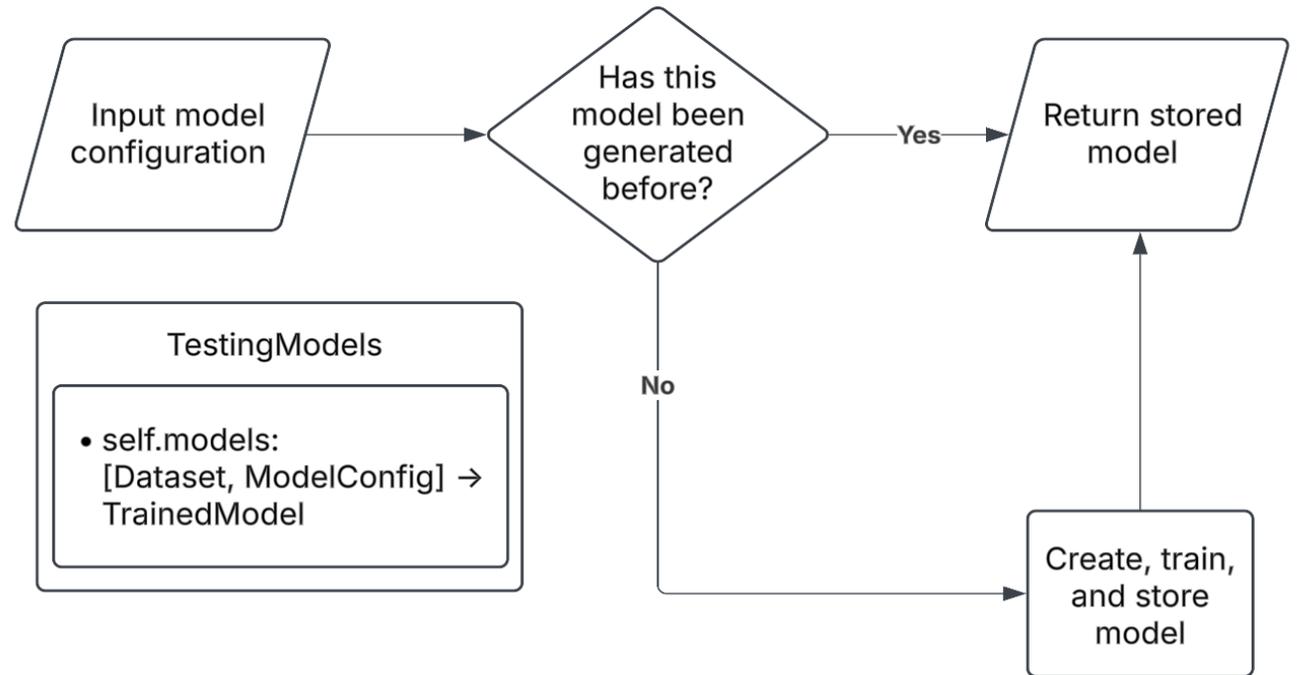


Problem: Code Maintainability

- Software engineering best practices not followed in original codebase
 - Poor testing
 - No CI/CD Pipeline
- Needs good automation to prevent keeping package updated becoming a chore

Testing

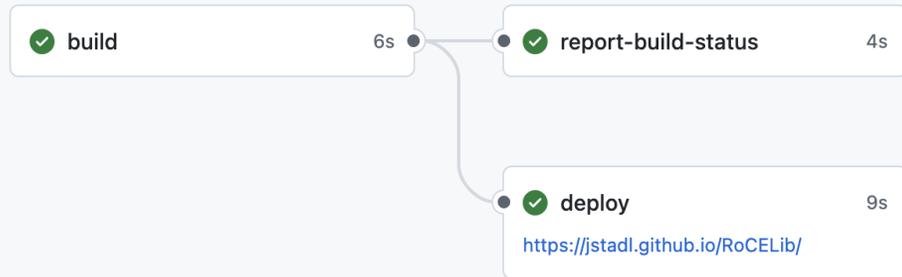
- Original tests were slow, and often not testing the right thing
- Singleton testing infrastructure for efficiency gains
- Original code: 33 tests, 13:05 to run, 66% coverage
- Our code: 75 tests, 8:09 to run, 78% coverage



CI/CD setup

pages-build-deployment

on: dynamic



Summary

Jobs

Build

Run details

Usage

Workflow file

Build

succeeded 17 minutes ago in 10m 42s

- > Set up job
- > Run actions/checkout@v4
- > Set up Python 3.10
- > Cache pip dependencies
- > Install dependencies
- > Lint with flake8
- > Test with pytest
- > Post Cache pip dependencies
- > Post Set up Python 3.10
- > Post Run actions/checkout@v4
- > Complete job

publish-to-pypi.yml

on: push



Evaluation: Maintainability



Project Management

Project Management

- Four 2-week sprints
- Bi-weekly meetings
- Scrum Master
- Feedback → pivot (classification task)
- At the end of each sprint we created a backlog

The image displays a Kanban board with four columns representing different stages of work:

- TO DO:** Contains a '+ Create issue' button.
- IN PROGRESS (3 items):**
 - packaging (SCRUM-25) assigned to SJ
 - Adding new types of visualisation (SCRUM-26) assigned to SS
 - Make generation and evaluation user friendly (SCRUM-22) assigned to SM
- DONE (5 items):**
 - Implement MM metrics (SCRUM-18) assigned to PA
 - Redesign evaluation hierarchy (SCRUM-19) assigned to AI
 - Implement IC evaluator (SCRUM-20) assigned to SS
 - Allow user to add multiple models to a Task (SCRUM-21) assigned to PA
 - Implement NE metric (SCRUM-23) assigned to EL
- BACKLOG (2 items):**
 - Do hyperparameter tuning (SCRUM-27) assigned to an unnamed user
 - Add some more evaluation methods based on the research papers (SCRUM-28) assigned to an unnamed user

Each item in the 'IN PROGRESS', 'DONE', and 'BACKLOG' columns has a checked checkbox, a Scrum ID, and a circular assignee icon. The 'DONE' column also features a green checkmark icon next to the count '5'.

Project Evaluation

Overall evaluation

Redesigned Architecture

- Simpler and more intuitive structure
- More extensible

Added Key Features

- Importing models
- Easy CE generation
- More robust metrics
- Visualisations and output

Much More Maintainable

- Solid CI/CD and actions set up
- Much improved testing

Easier for Users to Use

- Tutorial notebooks
- Auto-deployed documentation

RoCELib

Robust Counterfactual Explanation Library

Questions?

Metrics Implemented

- Problem: Doesn't have enough robustness metrics to evaluate on
 - Noisy Execution robustness metric: Invalidation Rate Robustness
 - Input change robustness metric: VaR robustness
- Very quick to add a new metric due to redesigned classes

Error Handling

- Original code: User exposed to low level errors
- **SS of pandas dimension error**

Fixes Implemented:

- Improved exception handling in PyTorch, Keras and SkLearn model imports.
- Clearer error messages for file format mismatches and missing files.
- Validation to ensure only supported model structures are loaded.

Result: Faster debugging and troubleshooting.

- More robust and user-friendly library.