Seldonian FairML

Abdul Hannan Kanji

Aug 08, 2021

TABLE OF CONTENTS

CHAPTER

ONE

INTRODUCTION

Welcome to the documentation of FairML, a tool to build Machine Learning models and RL agents with certain desirable behavior.You can read more about this approach here.

Also, read more about a comprehensive quickstart guide at https://aisafety.cs.umass.edu.

CHAPTER

TWO

QUICKSTART

The best way is get started is to quickly jump into an example: Here is a Google Colab notebook to train a simple Logistic Regression model on the UCI Adult dataset.And here is a step-by-step tutorial.

2.1 Model class creation

Create a subclass of `seldonian.algorithm.SeldonianAlgorithm` class.

```
from seldonian.algorithm import *
class ExampleSeldonianModel(SeldonianAlgorithm):
    def __init__(self, *params, **kwargs ):
        example_model = Model()
        #initialize all the model parameters
        pass
```

Now that we have a basic model setup, we need to implement the abstract method of `SeldonianAlgorithm` class.

- `predict` - This is a basic prediction method that uses the *current* model parameters to predict the output targets.

```
from seldonian.algorithm import *
class ExampleSeldonianModel(SeldonianAlgorithm):
    def __init__(self, *params, **kwargs ):
        self.example_model = Model()
        #initialize all the model parameters
        pass
    def predict(self, X, **kwargs):
        # prediction based on teh model
        return self.example_model.predict(X)
```

- `data` returns the complete data and targets as a tuple back. This includes the safety as well as the candidate data.

```
from seldonian.algorithm import *
class ExampleSeldonianModel(SeldonianAlgorithm):
    def __init__(self, *params, **kwargs ):
        self.example_model = Model()
        #initialize all the model parameters
        pass
    def predict(self, X, **kwargs):
        # prediction based on teh model
        return self.example_model.predict(X)
```

(continues on next page)

(continued from previous page)

```
    def data(self):
        return X, y
```

- `fit` trains the model with the constraints.

```
from seldonian.algorithm import *
class ExampleSeldonianModel(SeldonianAlgorithm):
    def __init__(self, *params, **kwargs ):
        self.example_model = Model()
        #initialize all the model parameters
        pass
    def predict(self, X, **kwargs):
        # prediction based on teh model
        return self.example_model.predict(X)
    def data(self):
        return self.X, self.y
    def fit(self, *args, **kwargs):
        # fit model based under the constraint that g >0.
        pass
```

There are various examples of such constraint optimization problems implemented like the Lagrangian 2 player game as implemented in the `VanillaNN` class.

Or using a barrier when optimizing using a Black box optimization technique like `CMA-ES` or `scipy.optimize.minimize` class. You can find them under the `seldonian.seldonian` package.

- `_safetyTest` performs a the safety test using the safety set, or predicts the upper bound of the constraint `g(\theta)` during candidate selection (or in this case, `fit`).

```
from seldonian.algorithm import *
class ExampleSeldonianModel(SeldonianAlgorithm):
    def __init__(self, *params, **kwargs ):
        self.example_model = Model()
        #initialize all the model parameters
        pass
    def predict(self, X, **kwargs):
        # prediction based on teh model
        return self.example_model.predict(X)
    def data(self):
        return self.X, self.y
    def fit(self, *args, **kwargs):
        # fit model based under the constraint that g >0.
        pass
    def _safetyTest(self, predict, **kwargs):
        if predict:
            # predict the upper bound during candidate selection
            return 1 if passed_is_predicted else 0
            pass
        else:
            # run the actual safety test
            return 1 if passed else 0
            pass
        pass
```

2.2 Training

This is *all* you need to implement a Seldonian model. You also need some constraints that are basically function callables. Some examples of such constraints is present in the `seldonian.objectives` package. A sample run would look something like this -

```
constraints = [constraint1, constraint2,...] #list of function callables
seldonian_model = ExampleSeldonianModel(constriants, data, other_args)
X, y = data
seldonian_model.fit(X, y)
return seldonian_model if seldonian_model._safetyTest() else NSF # No solution found
# we now have a trained model you can now do your predictions on this model
```

CHAPTER

THREE

REFERENCE

3.1 Seldonian Algorithm

class `seldonian.algorithm.`**`SeldonianAlgorithm`**

Bases: `abc.ABC`

Abstract class which represents the basic functions of a Seldonian Algorithm. This class can be considered as a starting point for implementing your own Seldonian algorithm.

Read more about the Seldonian Approach in Preventing undesirable behavior of intelligent machines

abstract _safetyTest(***kwargs*)

Run the safety test on the trained model from the candidate selection part i.e. the *`fit()`* function. It is also used to predict the $g(\theta)$ value used in candidate selection.

:param kwargs Key value arguments sent to the subclass implementation of safety test. :return Depending on the implementation, it will either return *0* if it passes or *1* if it doesn't. Or, it will also return the $g(\theta)$ value if it does not pass the safety test. Use the `safetyTest()` method to get a boolean value.

abstract data()

Access the training data used by the model.

Returns Tuple (Training data, labels)

abstract fit(***kwargs*)

Abstract method that is used to train the model. Also, this is the **candidate selection** part of the Seldonian Algorithm.

Parameters **kwargs** – key value arguments sent to the fit function

Returns

abstract predict(*X*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

safetyTest(***kwargs*)

A wrapper for the `_safetyTest` method that return a `Boolean` indicating whether the model passed the safety test.

Parameters **kwargs** – Key-value arguments that is passed directly to `_safetyTest`.

Returns

- `True` if model passed the safety test.

- `False` if the model fails the safety test.

3.2 Seldonian Abstract classes

Use this as a base class to implement your own fair model using the Seldonian approach.

class seldonian.seldonian.**LogisticRegressionSeldonianModel**(*X*, *y*, *g_hats=[]*, *safety_data=None*, *test_size=0.5*, *verbose=True*, *hard_barrier=False*, *stratify=False*, *random_seed=0*)

Bases: *seldonian.algorithm.SeldonianAlgorithm*

Implements a Logistic Regression classifier using `scipy.optimize` package as the optimizer using the Seldonian Approach for training the model. Have a look at the scipy.optimize.minimize reference for more information. You can use any of the methods listen in the *method* input of this SciPy function as a parameter to the `fit()` method call.

__init__(*X*, *y*, *g_hats=[]*, *safety_data=None*, *test_size=0.5*, *verbose=True*, *hard_barrier=False*, *stratify=False*, *random_seed=0*)

Initialize self. See help(type(self)) for accurate signature.

_safetyTest(*theta=None*, *predict=False*, *ub=True*)

This is the mehtod that implements the safety test. for this model.

Parameters

- **theta** – Model parameters to be used to run the safety test. **Default** - `None`. If `None`, the current model parameters used.
- **predict** – **Default** - `False`. Indicate whether you want to predict the upper bound of $g(\theta)$ using the candidate set (this is used when running candidate selection).
- **ub** – returns the upper bound if `True`. Else, it returns the calculated value. **Default**- `True`.

Returns Returns the value $max\{0, g(\theta)|X\}$ if *predict* = `False` , else $max\{0, \hat{g}(\theta)|X\}$.

data()

Access the training data used by the model.

Returns Tuple (Training data, labels)

fit(*opt='Powell'*)

Abstract method that is used to train the model. Also, this is the **candidate selection** part of the Seldonian Algorithm.

Parameters **kwargs** – key value arguments sent to the fit function

Returns

predict(*X*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

class seldonian.seldonian.**PDISSeldonianPolicyCMAES**(*data*, *states*, *actions*, *gamma*, *threshold=2*, *test_size=0.4*, *multiprocessing=True*)

Bases: *seldonian.cmaes.CMAESModel*, *seldonian.algorithm.SeldonianAlgorithm*

__init__(*data*, *states*, *actions*, *gamma*, *threshold=2*, *test_size=0.4*, *multiprocessing=True*)

Initialize self. See help(type(self)) for accurate signature.

_safetyTest(*theta*, *predict=False*, *ub=False*, *est=None*)

Run the safety test on the trained model from the candidate selection part i.e. the `fit()` function. It is also used to predict the $g(\theta)$ value used in candidate selection.

:param kwargs Key value arguments sent to the subclass implementation of safety test. :return Depending on the implementation, it will either return *0* if it passes or *1* if it doesn't. Or, it will also return the $g(\theta)$ value if it does not pass the safety test. Use the `safetyTest()` method to get a boolean value.

predict(*X*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

class seldonian.seldonian.**SeldonianAlgorithmLogRegCMAES**(*X*, *y*, *g_hats=[]*, *safety_data=None*, *verbose=False*, *test_size=0.35*, *stratify=False*, *hard_barrier=False*, *random_seed=0*)

Bases: *seldonian.cmaes.CMAESModel*, *seldonian.algorithm.SeldonianAlgorithm*

Implements a Logistic Regression classifier with CMA-ES as the optimizer using the Seldonian Approach.

__init__(*X*, *y*, *g_hats=[]*, *safety_data=None*, *verbose=False*, *test_size=0.35*, *stratify=False*, *hard_barrier=False*, *random_seed=0*)

Initialize the model.

Parameters

- **X** – Training data to be used by the model.
- **y** – Training labels for the *X*
- **g_hats** – A list of all constraint on the model.
- **safety_data** – If you have a separate held out data to be used for the safety set, it should be specified here, otherwise, the data *X* is split according to *test_size* for this.
- **verbose** – Print out extra log statements
- **test_size** – ratio of the data *X* to e used for the safety set.
- **stratify** – Stratify the training data when splitting to train/safety sets.
- **hard_barrier** – Use a hard barrier while training the data using the BBO optimizer.

_safetyTest(*theta=None*, *predict=False*, *ub=True*)

Run the safety test on the trained model from the candidate selection part i.e. the `fit()` function. It is also used to predict the $g(\theta)$ value used in candidate selection.

:param kwargs Key value arguments sent to the subclass implementation of safety test. :return Depending on the implementation, it will either return *0* if it passes or *1* if it doesn't. Or, it will also return the $g(\theta)$ value if it does not pass the safety test. Use the `safetyTest()` method to get a boolean value.

data()

Access the training data used by the model.

Returns Tuple (Training data, labels)

predict(*X*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

class seldonian.seldonian.**SeldonianCEMPDISPolicy**(*data*, *states*, *actions*, *gamma*, *threshold=1.41537*, *test_size=0.4*, *verbose=False*, *use_ray=False*)

Bases: seldonian.algorithm.SeldonianAlgorithm

__init__(*data*, *states*, *actions*, *gamma*, *threshold=1.41537*, *test_size=0.4*, *verbose=False*, *use_ray=False*)

Initialize self. See help(type(self)) for accurate signature.

_safetyTest(*theta*, *predict=False*, *ub=False*)

Run the safety test on the trained model from the candidate selection part i.e. the fit() function. It is also used to predict the $g(\theta)$ value used in candidate selection.

:param kwargs Key value arguments sent to the subclass implementation of safety test. :return Depending on the implementation, it will either return *0* if it passes or *1* if it doesn't. Or, it will also return the $g(\theta)$ value if it does not pass the safety test. Use the safetyTest() method to get a boolean value.

data()

Access the training data used by the model.

Returns Tuple (Training data, labels)

fit(*method='Powell'*)

Abstract method that is used to train the model. Also, this is the **candidate selection** part of the Seldonian Algorithm.

Parameters **kwargs** – key value arguments sent to the fit function

Returns

predict(*X*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

class seldonian.seldonian.**VanillaNN**(*X*, *y*, *test_size=0.4*, *g_hats=[]*, *verbose=False*, *stratify=False*, *epochs=10*, *model=None*, *random_seed=0*)

Bases: seldonian.algorithm.SeldonianAlgorithm

Implement a Seldonian Algorithm on a Neural network.

__init__(*X*, *y*, *test_size=0.4*, *g_hats=[]*, *verbose=False*, *stratify=False*, *epochs=10*, *model=None*, *random_seed=0*)

Initialize a model with *g_hats* constraints. This class is an example of training a non-linear model like a neural network based on the Seldonian Approach.

Parameters

- **X** – Input data, this also includes the safety set.
- **y** – targets for the data X

- **test_size** – the fraction of X to be used for the safety test
- **g_hats** – a list of function callables that correspond to a constriant
- **verbose** – Set this to True to get some debug messages.
- **stratify** – set this to true if you want to do stratified sampling of safety set.
- **epochs** – number of epochs to run teh training of the model. Default: 10
- **model** – PyTorch model to use. Should be an instance of nn.Module. Defaults to a 2 layer model with a binary output.

_safetyTest(*predict=False*, *ub=True*)

Run the safety test on the trained model from the candidate selection part i.e. the *fit()* function. It is also used to predict the $g(\theta)$ value used in candidate selection.

:param kwargs Key value arguments sent to the subclass implementation of safety test. :return Depending on the implementation, it will either return *0* if it passes or *1* if it doesn't. Or, it will also return the $g(\theta)$ value if it does not pass the safety test. Use the safetyTest() method to get a boolean value.

data()

Access the training data used by the model.

Returns Tuple (Training data, labels)

fit(***kwargs*)

Abstract method that is used to train the model. Also, this is the **candidate selection** part of the Seldonian Algorithm.

Parameters **kwargs** – key value arguments sent to the fit function

Returns

predict(*X*, *pmf=False*)

Predict the output of the model on the the input X.

Parameters **X** – input data to be predicted by the model.

Returns output predictions for each sample in the input X

3.3 Sample constraint functions

class seldonian.objectives.**Constraint**

Bases: abc.ABC

seldonian.objectives.**ghat_recall_rate**(*A_idx*, *method='ttest'*, *threshold=0.2*)

Create a g_hat for the recall rate difference between :param A_idx subset versus the entire data.

Parameters

- **A_idx** –
- **method** –
- **threshold** – Recall rate should not be greater than this value.

Returns method that is to be sent to the Seldonian Algorithm and is used for calculating teh g_hat

seldonian.objectives.**ghat_tpr_diff**(*A_idx*, *method='ttest'*, *threshold=0.2*)

Create a $g(\theta)$ for the true positive rate difference between A_idx subset versus the entire data.

Parameters

- **A_idx** – index of the sensitive attribute in the X passed to the method returned by this function.
- **method** – The method used to calculate the upper bound. Currently supported values are:
  - *ttest* - Use student Student's t-distribution to calculate the confidence interval.
  - *hoeffdings* - Use the Hoeffdings inequality to caluclate the 95% confidence interval.
- **threshold** – TPR rate should not be greater than this value.

Returns method that is to be sent to the Seldonian Algorithm and is used for calculating the $g(\theta)$

seldonian.objectives.**ghat_tpr_diff_t**(*A_idx*, *method='ttest'*, *threshold=0.2*)

Pytorch version of the true positive rate difference version of *ghat_tpr_diff()*.

Create a $g(\theta)$ for the true positive rate difference between A_idx subset versus the entire data.

Parameters

- **A_idx** – index of the sensitive attribute in the X passed to the method returned by this function.
- **method** – The method used to calculate the upper bound. Currently supported values are:
  - *ttest* - Use student Student's t-distribution to calculate the confidence interval.
  - *hoeffdings* - Use the Hoeffdings inequality to caluclate the 95% confidence interval.
- **threshold** – TPR rate should not be greater than this value.

Returns method that is to be sent to the Seldonian Algorithm and is used for calculating the $g(\theta)$

3.4 CMA-ES optimizer implementation

class seldonian.cmaes.**CMAESModel**(*X*, *y*, *verbose=False*, *random_seed=0*, *theta=None*, *maxiter=None*)

Bases: abc.ABC

This library is an implementation of the paper Preventing undesirable behavior of intelligent machines.

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX