Index

1. Description and function of the .py files in the github:

1. Filename: Mask.py

- **Input:** Test data and blackbox
- **Goal:** Fit a mask to an input data sample and a black-box function, such that the mask can be used to perturb the input sample and evaluate the impact on the output of the black-box function.
- **Method in Brief:**
    1. **Step 1:** Initialize Attributes
    2. **Step 2:** Setup Optimizer
    3. **Step 3:** Run loop for given n epochs
    4. **Step 4:** Update Mask coefficient at each iteration using optimizer
    5. **Step 5:** Store important metrics
    6. **Step 6:** Return optimized mask
- **Attributes**
    1. **Perturbation:**
    2. **Device:**
    3. **Verbose:**

4. **Random seed:**
5. **Deletion Mode:** A Flag which is true if mask should identify the most important deletion
6. **eps(float):** Small number for Stability
7. **Mask tensors:** Tensor containing mask coefficient
8. **T(int):** No. of Time Steps
9. **N_features(int):** No. of features
10. **Y_target(torch.tensor)**: Black box prediction
11. **hist(torch.tensor):** History tensor containing the strong metrics at different epochs during optimization
12. **task(str):** Classf. or Reg.

- **Methods:**
  Has only method .fit(). Which has different arguments
- **Arguments of .fit() function:**

1. **X:** Input Matrix (as a T * N torch tensor)
2. **f:** Black Box Function
3. **Loss Function**:
4. **N_epochs:**
5. **Keep Ratio:** Fraction of elements in X that should be kept by the mask(a in paper)
6. **Initial Mask Coefficient:** Initial Value For the mask Coefficient called lambda 0.
7. **Size Regression Factor Init:** Initial coefficient for the regulator part of the total loss.
8. **Size Regression factor Dilation:** Ratio between the final and the initial Initial size regulation factor.(Called delta a in paper)
9. **Time regression factor:** Regulation factor for the variation in time
10. **Learning Rate:** For SGD
11. **Momentum:** For SGD

2. Filename: Perturbation.py:

1. **Goal:** Apply perturbation to an input data sample based on the mask
2. **Different Classes:** The parent class is the perturbation class. Other Classes like **FadeMovingAverage, Gaussian Blur, Fade MA Past Window, Fade Reference** are inherited/Subclasses of the Parent Class.

- **Class Perturbation:**
  - **Methods: 1.** “apply” and **2**. “apply extremal”
  - **Attributes: 1**. Mask_tensor,**2**. eps
  - **“apply” method:**

Apply perturbation on the input based on input mask tensor

- **“apply extremal”:**
  Apply perturbation on the input based on input extremal tensor i.e. Set of masks and the perturbation that is applied to each mask.

3. **Class Gaussian Blur:**

- **Attributes: One extra att. than parent class sigma max**
- **Method:**
- **Step1: Call Parent Class’s method “apply” or “apply extremal”**
- **Step2: Convert the mask into a tensor containing the width of each Gaussian perturbation. For each feature and each time, it computes the coefficients for the Gaussian perturbation. The perturbation is obtained by replacing each input by the linear combination weighted by Gaussian coeff.**

4. **Class Fade MA:**
   Attribute : Same as parent class

- **Method:**
- **Step1: Call Parent Class’s method “apply” or “apply extremal”**
- **Step2: Compute Moving Average of each feature and concatenate it to create tensor with X’s shape.The perturbation is then calculated as an affine combination of the input and the previous tensor, weighted by the mask.**

3. Filename: Mask Group.py:

- **Attributes: 1. Perturbation Object, 2. Device, 3. Random seed, and 4.Verbosity**
- **Method**
- **The main method in this class is the fit method,**
- **Arguments:**
  Takes in inputs such as the data, black-box function, list of areas for the masks, loss function, number of epochs, initial mask coefficient, and various regularization factors for the loss function. The method then fits the masks to the input data using the specified loss function and optimizer, and stores the results in the class's attributes.

4. Baseline.py:

- **Goal:** Fit a mask to an input data sample and a black-box function, such that the mask can be used to perturb the input sample and evaluate the impact on the output of the black-box function.
- **The methods:**
  1.Perturbation methods (FO, FP),

2. Integrated Gradient (IG),
3. Shapley methods (GradShap, SVS).

- **Process in Brief:**
- Each class takes in the black-box function as an input, and has an "attribute" method that takes in input data and produces feature importance scores. The feature importance scores can be normalized if specified.
- These classes use the captum library for the implementation of these attribution methods, and are mainly used for the rate time and feature experiment. The code also has a function normalize which normalizes the data.

5. Filename: Mimic.py:

- **This code runs an experiment to explain the predictions of a black-box model trained on the MIMIC dataset**
- **The experiment uses the FIT method to fit a mask to the input data, which highlights the most important features for the model's predictions**
- **The code first sets the device to use (GPU if available, otherwise CPU), loads the data, and loads a pre-trained model.**
- **The code then defines the black-box function as the loaded model**
- **and creates a perturbation operator and a mask saliency map.**
- **It then loops through the test data and for each sample, it fits the mask to the data using the FIT method, with specified parameters such as loss function, target, and regularization factors.**
- **The mask is then used to create a saliency map, which highlights the most important features for the model's predictions. The code also save the true label and inputs of the test set.**

6. Filename: experiments.py

1. Class 1: Experiment

- **This is a Python class called "Experiment" that is defined as an abstract base class (ABC) using the "ABC" metaclass from the "abc" module.**
- **The class has several methods including an init method that initializes some class variables, such as train_loader, valid_loader, test_loader and data.**
- **It also initializes the device to be used as "cuda" if a CUDA-enabled GPU is available, otherwise it defaults to "cpu".**
- **The class also has an abstract method called "run" that raises a runtime error if it is not implemented by a subclass.**
- **The class also has a method called "train" that takes in two arguments, n_epochs and learn_rt.**

- **The method trains the model using the train_loader, valid_loader and the optimizer. It also calls another function called "test" which is used to evaluate the performance on held-out test set.**
- **The train method also has a condition to check the value of learn_rt, if it is true, it calls a different function called "train_model_rt" or "train_model_rt_rg" depending on the value of data. Finally, the method prints the generator training time.**

2. Class 2 baseline :

- **This is a Python class called "Baseline" that inherits from the "Experiment" class defined previously.**
- **The class has an init method that initializes several class variables, such as feature_size, experiment and data. It also creates an instance of LR model and assigns it to self.model and moves it to device.**
- **The class also has a method called "run" that takes in a single argument train.**

If train is True, the method calls the parent class's train method passing n_epochs = 250, otherwise it checks if a checkpoint file exists, if so, it loads the model parameters from the checkpoint file and calls the test method to evaluate the performance.
If the checkpoint file does not exist, it raises a runtime error.

2. My Code Description:

- **Step 1 : Call Gaussian Blur perturbation and stored it in pert variable**

```
pert = GaussianBlur(device)
```

Q. Is there any alternative option to declare the pert variable?

- Yes.The pert variable is made using different classes in the file perturbation.py .Different Classes like **FadeMovingAverage, Fade MA Past Window, Fade Reference** are the other inherited/Subclasses of the Parent Class which we can use as a alternative to **Gaussian Blur**
- **Step 2: Declare the mask variable:**

```
mask = MaskGroup(pert, device)
```

Q. Is there any alternative option to declare the mask variable?

- Yes.we can declare the mask variable by calling the **Mask** class object.Here is an example:

```
# Fit the mask:
mask = Mask(pert, device, task="classification", verbose=False, deletion_mode=True)
```

- **Step 3: Mention different values of area :**

areas = [0.1, .25,0.35,0.45,0.55,0.65,0.66,0.665,0.678,.663,0.64]

- **Step 4: Fit the mask:**

```
mask.fit(torch.tensor(X_test.values, dtype=torch.float), f=lasso_output,
loss_function=mse_loss,area_list=areas,
size_reg_factor_init=0.01,learning_rate=2.5,
size_reg_factor_dilation=10000, initial_mask_coeff=0.5, n_epoch=250,
momentum=1.0, time_reg_factor=2)
```

- **Arguments of .fit() function:**
- **X:** Input Matrix (as a T * N torch tensor)
- **f:** Black Box Function
- **Loss Function**:
- **N_epochs:**
- **Keep Ratio:** Fraction of elements in X that should be kept by the mask(a in paper)
- **Initial Mask Coefficient:** Initial Value For the mask Coefficient called lambda 0.
- **Size Regression Factor Init:** Initial coefficient for the regulator part of the total loss.
- **Size Regression factor Dilation:** Ratio between the final and the initial Initial size regulation factor.(Called delta a in paper)
- **Time regression factor:** Regulation factor for the variation in time
- **Learning Rate:** For SGD
- **Momentum:** For SGD

- **Step 5: Plot the Mask**:
- `masks = mask.get_extremal_mask(threshold=0.3)`
- `masks.plot_mask(smooth= False, sigma=2.0)`

Q.Is there any option to plot the mask in a better way?

- Yet to be explored

3.Results on Different datasets:

- **Data 1 : Time series starter (data kaggle)**: https://www.kaggle.com/datasets/podsyp/time-series-starter-dataset/code
- **Used model :** https://www.kaggle.com/code/sudhanshu2198/time-series-analysis-using-linear-regression
- **Data Description:**

Basic configurations were directly constructed through `DeterministicProcess` module from statsmodel. These include a **constant, a time trend , and either a seasonal or a Fourier component**.

The process requires an index, which is the index of the full-sample (or in-sample).

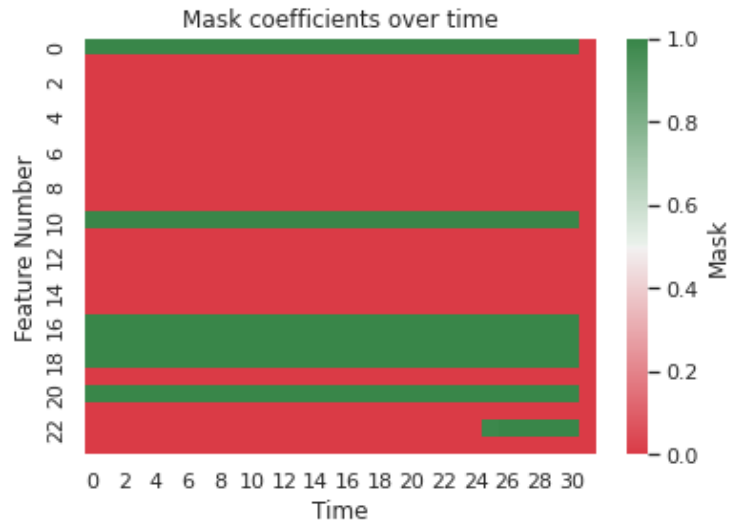
First, we initialize a deterministic process with a constant, a linear time trend, seasonal terms and Fourier terms.

- **Target Column of prediction:**
- The model was made to predict the revenue.The original data was like below

	Period	Revenue	Sales_quantity	Average_cost	The_average_annual_payroll_of_the_region
0	01.01.2015	1.601007e+07	12729.0	1257.763541	30024676.0
1	01.02.2015	1.580759e+07	11636.0	1358.507000	30024676.0
2	01.03.2015	2.204715e+07	15922.0	1384.697024	30024676.0
3	01.04.2015	1.881458e+07	15227.0	1235.606705	30024676.0

- **Github link:** https://github.com/cloudcraftz/cc-xai/blob/main/Dynamask_on_kaggle_data_2.ipynb

Mask obtained from Dynamask:



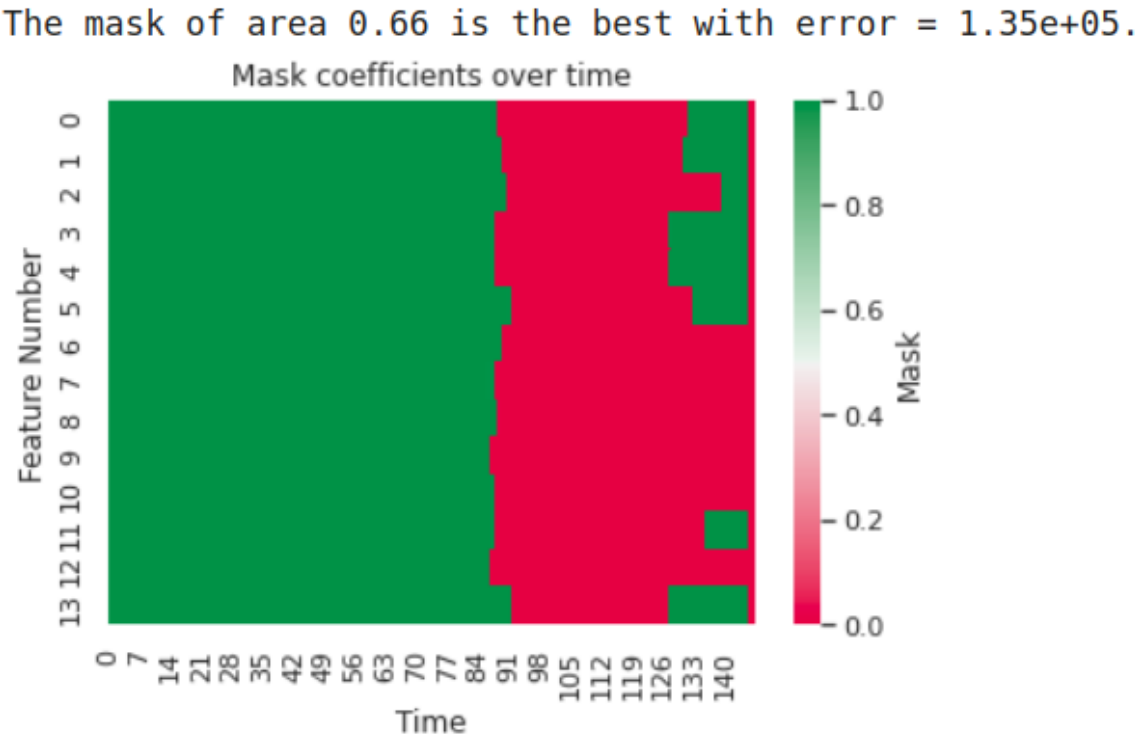
- **Features Selected by dynamask:**
    1. Feature no 0,10,16,18,20 has long term dependencies
    2. Feature no 22 has dependencies from time steps 24 to 30

- **Data 2: Bike Data set**
- **Data Description:**
- This dataset contains daily counts of rented bicycles from the bicycle rental company along with weather and seasonal information.

- The goal is to predict how many bikes will be rented depending on the weather and the day.

dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801

- **Target Column of prediction:** The Count of the rented bikes
- **Dynamask output:**



- **Features Selected by dynamask:**

0. instant: record index

1. season : season (1:winter, 2:spring, 3:summer, 4:fall)

2.yr : year (0: 2011, 1:2012)

3. mnth : month (1 to 12)

4.hr : hour (0 to 23)

5.holiday : weather day is holiday or not (extracted from [Web Link])

11. Wind Speed

13.No. of Registered users

Data 3 : Financial Distress Data
Data Description :

```
data
```

	Time	Financial Distress	x1	x2	x3	x4	x5	x6	x7	x8	...	x80_28	x80_29	x80_30	x80_31	x80_32	x80_33	x80_34	x80_35	x80
0	1	0.010636	1.2810	0.022934	0.87454	1.21640	0.060940	0.188270	0.52510	0.018854	...	0	0	0	0	0	0	0	0	
1	2	-0.455970	1.2700	0.006454	0.82067	1.00490	-0.014080	0.181040	0.62288	0.006423	...	0	0	0	0	0	0	0	0	
2	3	-0.325390	1.0529	-0.059379	0.92242	0.72926	0.020476	0.044865	0.43292	-0.081423	...	0	0	0	0	0	0	0	0	
3	4	-0.566570	1.1131	-0.015229	0.85888	0.80974	0.076037	0.091033	0.67546	-0.018807	...	0	0	0	0	0	0	0	0	
4	1	1.357300	1.0623	0.107020	0.81460	0.83593	0.199960	0.047800	0.74200	0.128030	...	0	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3667	10	0.438020	2.2605	0.202890	0.16037	0.18588	0.175970	0.198400	2.22360	1.091500	...	0	0	0	0	0	0	0	0	

- **Target Column of prediction: “Financial Distress”**

- **Features Selected by dynamask:**
- **No irrelevant features found**
- **Problem: In my view this type of output would occur in datasets for which Relevant features are already selected from Raw data where both Irrelevant and Relevant Features are present**
- **Wayout:**
- **Plots like above are binary representations of the whole time series data as (0:Irrelevant/Red) or (1:Relevant/Green).**
- **Plots like below has different shades of Red and Green color which is a better representation than representing as only red and green.**
- **(This graph is present in page no 7 of the original paper(2.4. Masks and information theory))**

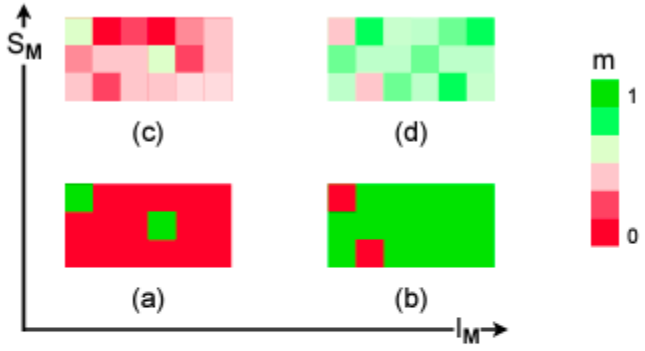


Figure 3. Mask information and entropy. For a given subsequence and a given mask, the information increases when more features are relevant ($a \rightarrow b$ or $c \rightarrow d$). The entropy increases when the sharpness of the saliency map decreases ($a \rightarrow c$ or $b \rightarrow d$). Masks with high entropy appear less legible, especially for long sequences.

- **This type of representation could help us to better understand the data.**

Data 4 : House Property Data

https://www.kaggle.com/datasets/htagholdings/property-sales

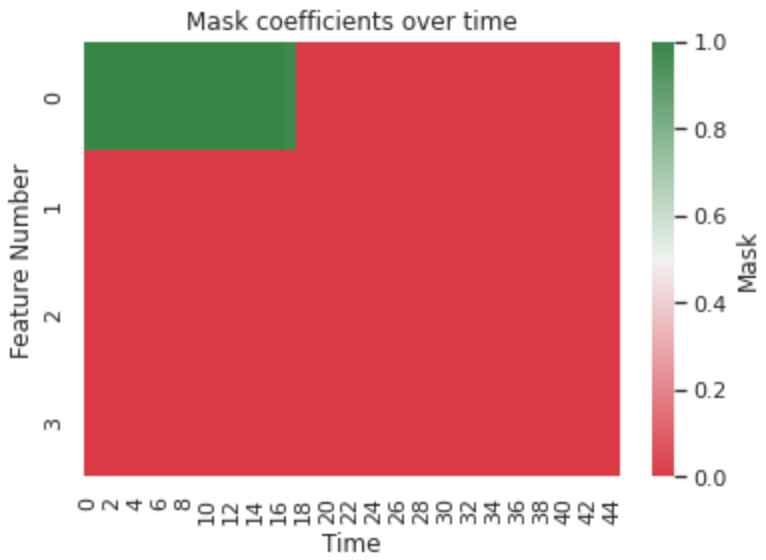
- **Model used:** https://www.kaggle.com/code/aklimarimi/tutorial-irregular-time-series
- **Data Description:**
- **Target Column of prediction: "Price" column.Original data was looking like below**

datesold	postcode	price	propertyType	bedrooms
2007-02-28	2756.500000	12.874395	0.000000	3.500000
2007-03-31	2905.333333	12.731008	0.000000	3.333333
2007-04-30	2706.000000	13.395763	0.000000	3.666667
2007-05-31	2904.000000	12.733577	0.000000	3.000000

```
X_test
```

datesold	propertyType	bedrooms	year	month
2015-11-30	1.0	3.0	2015	11
2015-12-31	1.0	3.0	2015	12
2016-01-31	1.0	3.0	2016	1
2016-02-29	1.0	3.0	2016	2

- **Github link:** https://github.com/cloudcraftz/cc-xai/blob/main/Dynamask_on_kaggle%20data%201(house_property_data).ipynb



- **Features Selected by dynamask:**
- **Only feature no 0 (Property type) from timesteps 0 to 18 are salient**
- **Xtest from 0 to 24 timesteps is shown below**

```
X_test[0:24]
```

datesold	propertyType	bedrooms	year	month
2015-11-30	1.0	3.0	2015	11
2015-12-31	1.0	3.0	2015	12
2016-01-31	1.0	3.0	2016	1
2016-02-29	1.0	3.0	2016	2
2016-03-31	1.0	3.0	2016	3
2016-04-30	1.0	3.0	2016	4
2016-05-31	1.0	3.0	2016	5
2016-06-30	1.0	3.0	2016	6
2016-07-31	1.0	3.0	2016	7
2016-08-31	1.0	3.0	2016	8
2016-09-30	1.0	3.0	2016	9
2016-10-31	1.0	3.0	2016	10
2016-11-30	1.0	3.0	2016	11
2016-12-31	1.0	3.0	2016	12
2017-01-31	1.0	3.0	2017	1
2017-02-28	1.0	3.0	2017	2
2017-03-31	1.0	3.0	2017	3
2017-04-30	1.0	3.0	2017	4