NaturallyDrifted

Akshita Singh

Aug 18, 2022

CONTENTS:

CHAPTER

ONE

BASE CLASSES

1.1 base module

class base.**detectorParent**(*data_ref: Optional[Union[ndarray, list]] = None*, *data_h0: Optional[Union[ndarray, list]] = None*, *data_h1: Optional[Union[ndarray, list]] = None*, *sample_dict: Optional[Dict] = None*, *test: Union[MMD, LSDD] = 'MMD'*, *sample_size: int = 500*, *windows: Optional[int] = 10*, *drift_type: Optional[Union[Sudden, Gradual]] = 'Sudden'*, *SBERT_model: str = 'bert-base-uncased'*, *embedding_model: Union[Doc2Vec, SBERT, USE] = 'Doc2Vec'*, *transformation: Optional[Union[UMAP, UAE]] = None*, *pval_thresh: int = 0.05*, *dist_thresh: int = 0.0009*, *iterations: int = 5*, *plot: bool = True*)

Bases: object

__init__(*data_ref: Optional[Union[ndarray, list]] = None*, *data_h0: Optional[Union[ndarray, list]] = None*, *data_h1: Optional[Union[ndarray, list]] = None*, *sample_dict: Optional[Dict] = None*, *test: Union[MMD, LSDD] = 'MMD'*, *sample_size: int = 500*, *windows: Optional[int] = 10*, *drift_type: Optional[Union[Sudden, Gradual]] = 'Sudden'*, *SBERT_model: str = 'bert-base-uncased'*, *embedding_model: Union[Doc2Vec, SBERT, USE] = 'Doc2Vec'*, *transformation: Optional[Union[UMAP, UAE]] = None*, *pval_thresh: int = 0.05*, *dist_thresh: int = 0.0009*, *iterations: int = 5*, *plot: bool = True*)

In this class, we define the base arguments and parameters that are required by Alibi detectors. Not all of these parameters are used by each detector.

Parameters

- **data_ref** (*np.ndarray, list*) – Dataset on which the original model is trained (ex: training dataset). We flag a drift with a reference to the distribution of this dataset.
- **data_h0** (*np.ndarray, list*) – This is an optional dataset that we can use as a sanity check for the efficacy of a drift detector. Generally, we use the same dataset as data_ref (or a stream that comes soon after). The lack of drift in data_h0 (with data_ref as our reference) is the necessary condition to decide the robustness of the drift detection method
- **data_h1** (*np.ndarray, list*) – This is the principal dataset on which we might see a drift (ex. deployment data). It can be just one sample (for sudden drifts) or stream of samples (for gradual drifts). Often, for pipelines, datasets come in batches, and each new batch can then be updated to the new data_h1.
- **sample_dict** (*dict*) – Dictionary with samples for reference and comparison data (or streams of comparison data). The user can directly input the dictionary as our dataset source if they would prefer to organize the data on their own.
- **sample_size** (*int*) – This parameter decides the number of samples from each of the above 3 datasets that we would like to work with. For instance, if the entire training data

is 100K sentences, we can use a sample_size = 500 to randomly sample 500 of those sentences.

- **test** (*str*) – Here, we specify the kind of drift detection test we want (KS, KLD, JSD, MMD, LSDD). Each of them is described in greater detail in the README.md.
- **drift_type** (*str*) – Drifts can vary depending on the time horizan and frequency at which we try to detect them. This parameter asks the user to specify the type of drift ("Sudden", "Gradual", "Online"). The details of each are in README.md
- **plot** (*bool*) – This parameter asks the user if they wish to see some of the plots of the results from the drift detection. Not every detector will result in relevant plot.
- **windows** (*int*) – This parameter is relevant for gradual drifts and helps break down the data into a certain number of buckets. These buckets can act like "batches" or "data streams". The idea behind this approach is that we are trying to localize drifts to a certain time frame and check for consistencies (or lack thereof) in detection. If data_h1 has 100K data points, and if we wish to detect drifts gradually over time, a proxy approach would be to break the data in sets of 5K points and then randomly sample from each set separately.
- **SBERT_model** (*str*) – This parameter is specific to the SBERT embedding models. If we choose to work with SBERT, we can specify the type of SBERT embedding out here. Ex. 'bert-base-uncased'
- **ert** (*int (optional)*) – Expected Run Time before a drift is detected. Alibi detect uses this approach for it's online drift detectors. If the average ERT for the reference data is significantly higher than the average run time for the drifted data, that might indicate a possible drift.
- **window_size** (*int*) – This parameter is used within Alibi's online detectors. It specifies the number of datapoints to include in one window.
- **n_run** (*int*) – This parameter is used within Alibi's online detectors and specifies the number of runs the detector must perform before we can get an average ERT.
- **n_bootstraps** (*int*) – This parameter is used within Alibi's online detectors
- **context_type** (*str*) – Context that we wish to ignore 1) sub-population: if we wish to ignore the relative change in sub-population of certain classes
- **embedding_model** (*str*) – This is the principle parameter of this class. It decided the kind of embedding the text goes through. The embeddings we consider thus far are: a) SBERT: A Python framework for state-of-the-art sentence, text and image embeddings. b) Universal Sentence Encoders: USE encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks c) Doc2Vec: a generalization of Word2Vec, which in turn is an algorithm that uses a neural network model to learn word associations from a large corpus of text

Return type
Nothing

1.2 sampling module

class sampling.**samplingData**(**args*, ***kwargs*)

Bases: *detectorParent*

__init__(**args*, ***kwargs*)

Takes in 2-3 datasets - the reference set, the h0 set (generally the same as reference set), and the possibly drifted set and then samples from them based on the kind of drift we are trying to flag. The h0 data is optional.

Return type
Dictionary with dataset (referemce, h0, h1) as key and a numpy array of data samples as values

random_sample(*data: Union[ndarray, list]*)

sample_data_gradual()

Takes in 2-3 datasets - the reference set, the h0 set (generally the same as reference set), and the possibly drifted set and then samples from them based on the kind of drift we are trying to flag. The h0 data is optional.

Return type
Dictionary with dataset (referemce, h0, h1) as key and a numpy array of data samples as values

sample_data_online()

Takes in 2-3 datasets - the reference set, the h0 set (generally the same as reference set), and the possibly drifted set and samples from each given the sample_size as decided by the user. The h1 (comparison data) is divded into buckets as decided by the number of windows

Return type
Dictionary with (0 (referemce), 1 (h0), 2,. . . , n (h1//window)) and numpy array of data samples as values

sample_data_sudden()

Takes in 2-3 datasets - the reference set, the h0 set (generally the same as reference set), and the possibly drifted set and samples from each given the sample_size as decided by the user. The h0 data is optional.

Return type
Dictionary with 2-3 keys (0 (referemce), 1 (h0), 2 (h1)) and numpy array of data samples as values

samples()

Takes in 2-3 datasets - the reference set, the h0 set (generally the same as reference set), and the possibly drifted set and then samples from them based on the kind of drift we are trying to flag. The h0 data is optional.

Return type
Dictionary with dataset (referemce, h0, h1) as key and a numpy array of data samples as values

CHAPTER

TWO

ALIBI DETECTORS

2.1 basicDetectors module

sudden and (basic) gradual drifts on text data from the following detectors - MMD and LSDD

class basicDetectors.**basicDetectors**(**args*, ***kwargs*)

Bases: *samplingData*, *detectorParent*

__init__(**args*, ***kwargs*)

In this class, we check for possible sudden drift in the data, using some of Alibi's methods. Sudden drifts are drifts we could see right after deployment. We can also use sudden drift techniques to try identifying drifts in a new batch of data (Ex. data being streamed weekly).

Returns

- *Lists and plots of relevant test statistics (p-values, distances) given the selected*
- *detector (MMD, LSDD etc)*

detector()

Here, we call the relevant drift detection method from Alibi Detect, given user input. The function uses reference samples and preprocessing from the previous function as arguments for the detection model development here.

Return type
A trained detection model (MMD, LSDD etc) as specified by the user input

embedData()

Call the samplingData class to construct samples from the input data provided by the user

Return type
Dictionary with samples for reference and comparison data (or streams of comparison data).

preprocess()

Here we process the text data in the following manner: 1) Embed it (generally, by using some kind of a Sentence Transformer) 2) Prepare a dimension reduction model for it that we can than feed into the main Alibi detector function

Return type
A dimesnion reduction/preprocessing model that the Alibi Detector can use (generally, an Untrained Autoencoder)

run()

Here, we run the detection model from the previous function, on the comparison data on which we want to check for a possible drift.

Return type
Lists and plots of relevant test statistics (p-values, distances) given the selected detector (MMD, LSDD etc)

`run_all()`

`sampleData()`
Call the samplingData class to construct samples from the input data provided by the user

Return type
Dictionary with samples for reference and comparison data (or streams of comparison data).

2.2 onlineDetectors module

online (calibrated gradual) drifts on text data from the following detectors - MMD and LSDD

`class onlineDetectors.onlineDetectors(*args, **kwargs)`
Bases: *samplingData*, *detectorParent*

`__init__(*args, **kwargs)`
Checks for possible drift in the dataset in an online fashion. Instead of detecting drifts for each new, non-overlapping window, this method tries to detect drift as soon as any new data arrives. This detector leverages a calibration method discussed in Cobb et all (2021). The detectors compute a test statistic during the configuration phase. Then, at test time, the test statistic is updated sequentially at a low cost. When no drift has occurred the test statistic fluctuates around its expected value, and once drift occurs the test statistic starts to drift upwards. When it exceeds some preconfigured threshold value, drift is detected.

Almost all offline drift detectors have their online counterparts.

Returns

- *1) Lists and plots of expected run times (OnlineMMD, OnlineLSDD etc).*
- *2) Plots of dynamic threshold pitted against the test statistic for that window*

`detector()`
Here, we call the relevant drift detection method from Alibi Detect, given user input. The function uses reference samples and preprocessing from the previous function as arguments for the detection model development here.

Return type
A trained detection model (MMD, LSDD etc) as specified by the user input

`preprocess()`
Here we process the text data in the following manner: 1) Embed it (generally, by using some kind of a Sentence Transformer) 2) Prepare a dimension reduction model for it that we can than feed into the main Alibi detector function

Return type
A dimesnion reduction/preprocessing model that the Alibi Detector can use (generally, an Untrained Autoencoder)

`run()`
Here, we run the detection model from the previous function, on the comparison data on which we want to check for a possible drift.

Returns

- *1) Lists and plots of expected run times (OnlineMMD, OnlineLSDD etc).*

- *2) Plots of dynamic threshold pitted against the test statistic for that window*

sampleData()

Call the samplingData class to construct samples from the input data provided by the user

Return type
Dictionary with samples for reference and comparison data (or streams of comparison data).

2.3 alibiDetectors module

class alibiDetectors.**alibiDetectors**(**args*, ***kwargs*)

Bases: *detectorParent*

__init__(**args*, ***kwargs*)

This is final wrapper class for all text related Alibi Detectors (basic detectors, online detectors, context aware detectors etc.). We can generally just directly call this one function and populate all the parameters (datasets, detection tests, drift types etc.) and get our test statistics.

Parameters

- **data_ref** (*np.ndarray, list*) – Dataset on which the original model is trained (ex: training dataset). We flag a drift with a reference to the distribution of this dataset.
- **data_h0** (*np.ndarray, list*) – This is an optional dataset that we can use as a sanity check for the efficacy of a drift detector. Generally, we use the same dataset as data_ref (or a stream that comes soon after). The lack of drift in data_h0 (with data_ref as our reference) is the necessary condition to decide the robustness of the drift detection method
- **data_h1** (*np.ndarray, list*) – This is the principal dataset on which we might see a drift (ex. deployment data). It can be just one sample (for sudden drifts) or stream of samples (for gradual drifts). Often, for pipelines, datasets come in batches, and each new batch can then be updated to the new data_h1.
- **sample_dict** (*dict*) – Dictionary with samples for reference and comparison data (or streams of comparison data). The user can directly input the dictionary as our dataset source if they would prefer to organize the data on their own.
- **sample_size** (*int*) – This parameter decides the number of samples from each of the above 3 datasets that we would like to work with. For instance, if the entire training data is 100K sentences, we can use a sample_size = 500 to randomly sample 500 of those sentences.
- **test** (*str*) – Here, we specify the kind of drift detection test we want (KS, KLD, JSD, MMD, LSDD). Each of them is described in greater detail in the README.md.
- **drift_type** (*str*) – Drifts can vary depending on the time horizan and frequency at which we try to detect them. This parameter asks the user to specify the type of drift ("Sudden", "Gradual", "Online"). The details of each are in README.md
- **plot** (*bool*) – This parameter asks the user if they wish to see some of the plots of the results from the drift detection. Not every detector will result in relevant plot.
- **windows** (*int*) – This parameter is relevant for gradual drifts and helps break down the data into a certain number of buckets. These buckets can act like "batches" or "data streams". The idea behind this approach is that we are trying to localize drifts to a certain time frame and check for consistencies (or lack thereof) in detection. If data_h1 has 100K data points, and if we wish to detect drifts gradually over time, a proxy approach would be to break the data in sets of 5K points and then randomly sample from each set separately.

- **SBERT_model** (*str*) – This parameter is specific to the SBERT embedding models. If we choose to work with SBERT, we can specify the type of SBERT embedding out here. Ex. 'bert-base-uncased'
- **ert** (*int (optional)*) – Expected Run Time before a drift is detected. Alibi detect uses this approach for it's online drift detectors. If the average ERT for the reference data is significantly higher than the average run time for the drifted data, that might indicate a possible drift.
- **window_size** (*int*) – This parameter is used within Alibi's online detectors. It specifies the number of datapoints to include in one window.
- **n_run** (*int*) – This parameter is used within Alibi's online detectors and specifies the number of runs the detector must perform before we can get an average ERT.
- **n_bootstraps** (*int*) – This parameter is used within Alibi's online detectors
- **context_type** (*str*) – Context that we wish to ignore 1) sub-population: if we wish to ignore the relative change in sub-population of certain classes

Returns

- *Lists and plots of relevant test statistics (p-values, distances) given the selected*
- *detector (MMD, LSDD etc) and drift type (Sudden, Gradual, Online)*

run()

CHAPTER

THREE

FEATURE LEVEL DETECTORS

3.1 baseModels module

class baseModels.**baseModels**(*data*, *sample_size*, *SBERT_model: Optional[str]*)

Bases: object

SBERT_model

This class sets the stage for the embedding models we choose to work with later

Return type
An embedding model

doc2vec_base(*vector_size: Optional[int] = 100*, *window: Optional[int] = 2*, *min_count: Optional[int] = 1*, *workers: Optional[int] = 4*)

Develops model for Doc2Vec embeddings

sbert_base()

Develops model for Sentence Transformer embeddings

3.2 embedding module

class embedding.**embedding**(**args*, ***kwargs*)

Bases: *samplingData*, *detectorParent*

__init__(**args*, ***kwargs*)

In this class, we turn the samples of text inputs into text embeddings, which we can then use to a) either construct distributions, or b) calculate drift on. There are many different kinds of text embeddings and encodings. In this class, we cover 3 umbrella embeddings (discussed below)

Return type
A dictionary containing the embeddings as decided by the choice of embedding model and drift detection test type

dim_reduction(*emb_dict: Optional[dict] = None*, *components: Optional[int] = 25*, *n_iters: Optional[int] = 7*)

Embeds text inherited from the sampling class.

Parameters

- **emb_dict** (*dictionary*) –
- **method** (*Dictionary of embeddings as returned by the embed_data*) –

- **component** (*int (optional)*) –
- **SVD** (*The number top components we want from PCA or*) –
- **n_iters** (*int*) –

Returns

- *a dictionary containing the embeddings as decided by the choice of embedding model and*
- *drift detection test type*

embed_data()

Embeds text inherited from the sampling class. The type of embedding (Doc2Vec, SBERT etc) is decided by the user

Return type
A dictionary containing the embeddings as decided by the choice of embedding model and drift detection test type

embed_data_iters()

Runs the embedding function "iterations" number of times, if the selected drift detection test is the KS Test. For KL and JS Divergence, the iterations are taken care of in the distributions class

Return type
A dictionary containing the embeddings as decided by the choice of embedding model and drift detection test type

final_embeddings()

Returns

- *a dictionary containing the embeddings as decided by the choice of embedding model and*
- *drift detection test type*

sampleData()

Call the samplingData class to construct samples from the input data provided by the user

Return type
Dictionary with samples for reference and comparison data (or streams of comparison data).

3.3 distributions module

class distributions.**distributions**(**args*, ***kwargs*)

Bases: *embedding*, *samplingData*, *detectorParent*

__init__(**args*, ***kwargs*)

In this class, we construct distributions out of the embeddings we got from the "embedding" class. This is an optional class and is only required if we are running a distribution dependent test such as KLD or JSD.

Returns

- *A dictionary containing the distributions as decided by the choice of embedding model and*
- *drift detection test type*

distributions_doc2vec()

Constructs distributions for Doc2Vec embeddings

Return type
A dictionary containing the distributions as decided by the choice of embedding model and drift detection test type

distributions_seneconders()

Constructs distributions for Sentence Transformer or Universal Sentence Encoder embeddings

Return type
a dictionary containing the distributions as decided by the choice of embedding model and drift detector

final_distributions()

Constructs distributions for the selected embeddings (Doc2Vec, SBERT, USE)

Return type
A dictionary containing the distributions as decided by the choice of embedding model and drift detector

kde()

3.4 myDetectors module

class myDetectors.**myDetectors**(**args*, ***kwargs*)

Bases: *distributions*, *embedding*, *samplingData*, *detectorParent*

__init__(**args*, ***kwargs*)

This class returns the final detection results based on the embeddings or distributions it inherits. Currently, the tests covered in this class are Kolmogorov–Smirnov test, Kullback–Leibler divergence, and Jensen-Shannon Divergence. Each test will return a different output based on the kind of embedding model we choose to work with.

Parameters

- **data_ref** (*np.ndarray, list*) – Dataset on which the original model is trained (ex: training dataset). We flag a drift with a reference to the distribution of this dataset.
- **data_h0** (*np.ndarray, list*) – This is an optional dataset that we can use as a sanity check for the efficacy of a drift detector. Generally, we use the same dataset as data_ref (or a stream that comes soon after). The lack of drift in data_h0 (with data_ref as our reference) is the necessary condition to decide the robustness of the drift detection method
- **data_h1** (*np.ndarray, list*) – This is the principal dataset on which we might see a drift (ex. deployment data). It can be just one sample (for sudden drifts) or stream of samples (for gradual drifts). Often, for pipelines, datasets come in batches, and each new batch can then be updated to the new data_h1.
- **sample_dict** (*dict*) – Dictionary with samples for reference and comparison data (or streams of comparison data). The user can directly input the dictionary as our dataset source if they would prefer to organize the data on their own.
- **sample_size** (*int*) – This parameter decides the number of samples from each of the above 3 datasets that we would like to work with. For instance, if the entire training data is 100K sentences, we can use a sample_size = 500 to randomly sample 500 of those sentences.
- **test** (*str*) – Here, we specify the kind of drift detection test we want (KS, KLD, JSD, MMD, LSDD). Each of them is described in greater detail in the README.md.

- **drift_type** (*str*) – Drifts can vary depending on the time horizan and frequency at which we try to detect them. This parameter asks the user to specify the type of drift ("Sudden", "Gradual", "Online"). The details of each are in README.md
- **plot** (*bool*) – This parameter asks the user if they wish to see some of the plots of the results from the drift detection. Not every detector will result in relevant plot.
- **windows** (*int*) – This parameter is relevant for gradual drifts and helps break down the data into a certain number of buckets. These buckets can act like "batches" or "data streams". The idea behind this approach is that we are trying to localize drifts to a certain time frame and check for consistencies (or lack thereof) in detection. If data_h1 has 100K data points, and if we wish to detect drifts gradually over time, a proxy approach would be to break the data in sets of 5K points and then randomly sample from each set separately.
- **SBERT_model** (*str*) – This parameter is specific to the SBERT embedding models. If we choose to work with SBERT, we can specify the type of SBERT embedding out here. Ex. 'bert-base-uncased'
- **embedding_model** (*str*) – This is the principle parameter of this class. It decided the kind of embedding the text goes through. The embeddings we consider thus far are: a) SBERT: A Python framework for state-of-the-art sentence, text and image embeddings. b) Universal Sentence Encoders: USE encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering, and other natural language tasks c) Doc2Vec: a generalization of Word2Vec, which in turn is an algorithm that uses a neural network model to learn word associations from a large corpus of text

Return type
Drift detection related test statistics and any relevant plots

`divergence_doc2vec()`

Calculated Kullback–Leibler or Jensen–Shannon Divergence for Doc2Vec embeddings

Return type
The distances as given by the KL or JS Divergence

`divergence_seneconders()`

Calculated Kullback–Leibler or Jensen–Shannon Divergence for SBERT/USE embeddings

Return type
The distances as given by the KL or JS Divergence

`js_divergence(`*p*, *q*`)`

Calculated the Jensen–Shannon Divergence for the 2 distributions p and q

Parameters

- **p** (*np.ndarray*) –
- **data** (*A numpy array containing the distributions of some*) –
- **q** (*np.ndarray*) –
- **data** –

Return type
The JS Divergence distance

`kl_divergence(`*p*, *q*`)`

Calculated the Kullback–Leibler Divergence for the 2 distributions p and q

Parameters

- **p** (*np.ndarray*) –

- **data** (*A numpy array containing the distributions of some*) –
- **q** (*np.ndarray*) –
- **data** –

Return type
The KL Divergence distance

ks_doc2vec()

Calculated Kolmogorov–Smirnov test for Doc2Vec embeddings

Return type
The p-values and distances as given by the Kolmogorov–Smirnov test

ks_sbert()

Calculated the Kolmogorov–Smirnov test test for SBERT embeddings.

Return type
The p-values and distances as given by the Kolmogorov–Smirnov test

run()

Calculates the drift detection metrics, as specified by the choice of embedding model and drift detection test.

Return type
Distances for KLD or JSD or P-values for KS (depending on choice of test)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

D

E

F

J

K

M

O

P

R

S