nlpatl Documentation

Release 0.0.2

Edward Ma

Dec 18, 2021

CONTENTS:

nlpgatl is a library for active learning in machine learning experiments. The goal of NLPatl is assisting user to build high quality labeled dataset. It built on top of transformers, scikit-learn and other machine learning package. It can be applied into both cold start scenario (no any labeled data) and limited labeled data scenario.

CHAPTER

ONE

LEARNING

1.1 nlpatl.learning.mismatch_farthest_learning

class nlpatl.learning.mismatch_farthest_learning.**MismatchFarthestLearning**(*clustering_sampling*, *embeddings*, *clustering*, *classification*, *embeddings_type=None*, *embeddings_model_config=None*, *clustering_model_config=None*, *classification_model_config=None*, *multi_label=False*, *name='mismatch_farthest_learning'*)

Bases: nlpatl.learning.learning.Learning

Applying mis-match first farthest traversal method apporach (with modification) to annotate the most valuable data points. You may refer to http://zhaoshuyang.com/static/documents/MAL2.pdf . Here is the pseudo:

1. [NLPatl] Convert raw data to features (Embeddings model)
2. [NLPatl] Train model and clustering data points (Clustering model)
3. [NLPatl] Estmiate the most valuable data points (Sampling)
4. [Human] Subject matter exepknrnts annotates the most valuable data points
5. [NLPatl] Train classification model (Classification model)
6. [NLPatl] Classify unlabeled data points and comparing the clustering model result according to the farthest mismatch data points
7. [Human] Subject matter exepknrnts annotates the most valuable data points
8. Repeat Step 2 to 7 until acquire enough data points or reach other exit criteria.

Parameters

- **clustering_sampling** (*str or function*) – Clustering sampling method for stage 1 exploration. Providing certified methods name (*nearest_mean*) or custom function.
- **embeddings** (str or nlpatl.models.embeddings.Embeddings) – Function for converting raw data to embeddings. Providing model name according to embeddings type. For

example, *multi-qa-MiniLM-L6-cos-v1* for *sentence_transformers*. bert-base-uncased` for *transformers*. *vgg16* for *torch_vision*.

- **embeddings_model_config** (*dict*) – Configuration for embeddings models. Optional. Ignored if using custom embeddings class
- **embeddings_type** (*str*) – Type of embeddings. *sentence_transformers* for text, *transformers* for text or *torch_vision* for image
- **clustering** (str or `nlpatl.models.clustering.Clustering`) – Function for clustering inputs. Either providing certified methods (*kmeans*) or custom function.
- **clustering_model_config** (*dict*) – Configuration for clustering models. Optional. Ignored if using custom clustering class
- **classification** (`nlpatl.models.classification.Classification`) – Function for classifying inputs. Either providing certified methods (*logistic_regression*, *svc*, *linear_svc*, *random_forest* and *xgboost*) or custom function.
- **classification_model_config** (*dict*) – Configuration for classification models. Optional. Ignored if using custom classification class
- **multi_label** (*bool*) – Indicate the classification model is multi-label or multi-class (or binary). Default is False.
- **name** (*str*) – Name of this learning.

clear_learn_data()
Clear all learn data points

educate(*index*, *x*, *x_features*, *y*)
Annotate data point. Only allowing annotate data point one by one. NOT batch.

Parameters

- **index** (*int*) – Index of data point.
- **x** (string, int, float or `np.ndarray`) – Raw data input. It can be text, number or numpy (for image).
- **x_features** (int, float or `np.ndarray`) – Data features
- **y** (*string, int, list of string (multi-label case) or list or int (multi-label case)*) – Label of data point

explore(*x*, *return_type='dict'*, *num_sample=10*)
Estimate the most valuable data points for annotation.

Parameters

- **x** (list of string, int or float or `np.ndarray`) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.

Returns The most valuable data points.

Return type `nlpatl.dataset.Dataset` objects or dict

explore_educate_in_notebook(*x*, *num_sample=5*, *num_sample_per_cluster=2*, *data_type='text'*)
Estimate the most valuable data points for annotation and annotate it in IPython Notebook. Executing *explore* function and *educate* function sequentially.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.
- **data_type** (*str*) – Indicate the data format for displying in IPython Notebook. Possible values are *text* and *image*.
- **num_sample_per_cluster** (*int*) –

get_learn_data()
Get all learn data points

Returns Learnt data points

Return type Tuple of index list of int, x (str or numpy.ndarray) x_features (numpy.ndarray) and y (numpy.ndarray)

learn(*x*, *y*, *include_learn_data=True*)
Train the classification model.

Parameters

- **x** (list of string, int or float or np.ndarray.) – Raw data inputs. It can be text, number or numpy.
- **y** (*bool*) – Label of data inputs
- **include_learn_data** (*bool*) – Train the model whether including human annotated data and machine learning self annotated data. Default is True.

1.2 nlpatl.learning.semi_supervised_learning

class nlpatl.learning.semi_supervised_learning.**SemiSupervisedLearning**(*sampling*, *embeddings*, *classification*, *embeddings_type=None*, *embeddings_model_config=None*, *classification_model_config=None*, *multi_label=False*, *self_learn_threshold=0.9*, *name='semi_supervised_learning'*)

Bases: nlpatl.learning.learning.Learning

Applying both active learning and semi-supervised learning apporach to annotate the most valuable data points. You may refer to https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0162075&type=printable . Here is the pseudo:

1. [NLPatl] Convert raw data to features (Embeddings model)
2. [NLPatl] Train model and classifing data points (Classification model)
3. [NLPatl] Estmiate the most valuable data points (Sampling)

4. [Human] Subject matter experts annotates the most valuable data points
5. [NLPatl] Retrain classification model
6. [NLPatl] Classify unlabeled data points and labeling those confidences are higher than *self_learn_threshold*
7. Repeat Step 2 to 6 until acquire enough data points.

Parameters

- **sampling** (*str or function*) – Sampling method for get the most valuable data points. Providing certified methods name (*most_confidence*, *entropy*, *least_confidence*, *margin*, *nearest_mean*, *fathest*) or custom function.
- **embeddings** (str or `nlpatl.models.embeddings.Embeddings`) – Function for converting raw data to embeddings. Providing model name according to embeddings type. For example, *multi-qa-MiniLM-L6-cos-v1* for *sentence_transformers*. bert-base-uncased` for *transformers*. *vgg16* for *torch_vision*.
- **embeddings_model_config** (*dict*) – Configuration for embeddings models. Optional. Ignored if using custom embeddings class
- **embeddings_type** (*str*) – Type of embeddings. *sentence_transformers* for text, *transformers* for text or *torch_vision* for image
- **classification** (`nlpatl.models.classification.Classification`) – Function for classifying inputs. Either providing certified methods (*logistic_regression*, *svc*, *linear_svc*, *random_forest* and *xgboost*) or custom function.
- **classification_model_config** (*dict*) – Configuration for classification models. Optional. Ignored if using custom classification class
- **self_learn_threshold** (*float*) – The minimum threshold for classifying probabilities. Data will be labeled automatically if probability is higher than this value. Default is 0.9
- **name** (*str*) – Name of this learning.
- **multi_label** (*bool*) –

`clear_learn_data()`
Clear all learn data points

`educate(`*index*, *x*, *x_features*, *y*`)`
Annotate data point. Only allowing annotate data point one by one. NOT batch.

Parameters

- **index** (*int*) – Index of data point.
- **x** (string, int, float or `np.ndarray`) – Raw data input. It can be text, number or numpy (for image).
- **x_features** (int, float or `np.ndarray`) – Data features
- **y** (*string, int, list of string (multi-label case) or list or int (multi-label case)*) – Label of data point

`explore(`*x*, *return_type='dict'*, *num_sample=10*`)`
Estimate the most valuable data points for annotation.

Parameters

- **x** (list of string, int or float or `np.ndarray`) – Raw data inputs. It can be text, number or numpy (for image).

- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.

Returns The most valuable data points.

Return type nlpatl.dataset.Dataset objects or dict

explore_educate_in_notebook(*x*, *num_sample=2*, *data_type='text'*)

Estimate the most valuable data points for annotation and annotate it in IPython Notebook. Executing *explore* function and *educate* function sequentially.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.
- **data_type** (*str*) – Indicate the data format for displying in IPython Notebook. Possible values are *text* and *image*.

get_learn_data()

Get all learn data points

Returns Learnt data points

Return type Tuple of index list of int, x (str or numpy.ndarray) x_features (numpy.ndarray) and y (numpy.ndarray)

get_self_learn_data()

Get all self learnt data points

Returns Self learnt data points

Return type Tuple of index list of int, x (numpy.ndarray) and y (numpy.ndarray)

learn(*x=None*, *y=None*, *include_learn_data=True*)

Train the classification model.

Parameters

- **x** (list of string, int or float or np.ndarray.) – Raw data inputs. It can be text, number or numpy.
- **y** (*bool*) – Label of data inputs
- **include_learn_data** (*bool*) – Train the model whether including human annotated data and machine learning self annotated data. Default is True.

1.3 nlpatl.learning.supervised_learning

class nlpatl.learning.supervised_learning.**SupervisedLearning**(*sampling*, *embeddings*, *classification*, *embeddings_type=None*, *embeddings_model_config=None*, *classification_model_config=None*, *multi_label=False*, *name='supervised_learning'*)

Bases: nlpatl.learning.learning.Learning

Applying typical active learning apporach to annotate the most valuable data points. Here is the pseudo:

1. [NLPatl] Convert raw data to features (Embeddings model)
2. [NLPatl] Train model and classifing data points (Classification model)
3. [NLPatl] Estmiate the most valuable data points (Sampling)
4. [Human] Subject matter experts annotates the most valuable data points
5. Repeat Step 2 to 4 until acquire enough data points.

Parameters

- **sampling** (*str or function*) – Sampling method for get the most valuable data points. Providing certified methods name (*most_confidence*, *entropy*, *least_confidence*, *margin*, *nearest_mean*, *fathest*) or custom function.
- **embeddings** (str or nlpatl.models.embeddings.Embeddings) – Function for converting raw data to embeddings. Providing model name according to embeddings type. For example, *multi-qa-MiniLM-L6-cos-v1* for *sentence_transformers*. bert-base-uncased` for *transformers*. *vgg16* for *torch_vision*.
- **embeddings_model_config** (*dict*) – Configuration for embeddings models. Optional. Ignored if using custom embeddings class
- **embeddings_type** (*str*) – Type of embeddings. *sentence_transformers* for text, *transformers* for text or *torch_vision* for image
- **classification** (nlpatl.models.classification.Classification) – Function for classifying inputs. Either providing certified methods (*logistic_regression*, *svc*, *linear_svc*, *random_forest* and *xgboost*) or custom function.
- **classification_model_config** (*dict*) – Configuration for classification models. Optional. Ignored if using custom classification class
- **multi_label** (*bool*) – Indicate the classification model is multi-label or multi-class (or binary). Default is False.
- **name** (*str*) – Name of this learning.

clear_learn_data()
Clear all learn data points

educate(*index*, *x*, *x_features*, *y*)
Annotate data point. Only allowing annotate data point one by one. NOT batch.

Parameters

- **index** (*int*) – Index of data point.

- **x** (string, int, float or np.ndarray) – Raw data input. It can be text, number or numpy (for image).
- **x_features** (int, float or np.ndarray) – Data features
- **y** (*string, int, list of string (multi-label case) or list or int (multi-label case)*) – Label of data point

explore(*x*, *return_type='dict'*, *num_sample=10*)

Estimate the most valuable data points for annotation.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.

Returns The most valuable data points.

Return type nlpatl.dataset.Dataset objects or dict

explore_educate_in_notebook(*x*, *num_sample=2*, *data_type='text'*)

Estimate the most valuable data points for annotation and annotate it in IPython Notebook. Executing *explore* function and *educate* function sequentially.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.
- **data_type** (*str*) – Indicate the data format for displying in IPython Notebook. Possible values are *text* and *image*.

get_learn_data()

Get all learn data points

Returns Learnt data points

Return type Tuple of index list of int, x (str or numpy.ndarray) x_features (numpy.ndarray) and y (numpy.ndarray)

learn(*x*, *y*, *include_learn_data=True*)

Train the classification model.

Parameters

- **x** (list of string, int or float or np.ndarray.) – Raw data inputs. It can be text, number or numpy.
- **y** (*bool*) – Label of data inputs
- **include_learn_data** (*bool*) – Train the model whether including human annotated data and machine learning self annotated data. Default is True.

1.4 nlpatl.learning.unsupervised_learning

class nlpatl.learning.unsupervised_learning.**UnsupervisedLearning**(*sampling*, *embeddings*, *clustering*, *embeddings_type=None*, *embeddings_model_config=None*, *clustering_model_config=None*, *multi_label=False*, *name='unsupervised_learning'*)

Bases: nlpatl.learning.learning.Learning

Applying unsupervised learning apporach to annotate the most valuable data points. You may refer to https://homepages.tuni.fi/tuomas.virtanen/papers/active-learning-sound.pdf. Here is the pseudo:

1. [NLPatl] Convert raw data to features (Embeddings model)
2. [NLPatl] Train model and clustering data points (Clustering model)
3. [NLPatl] Estmiate the most valuable data points (Sampling)
4. [Human] Subject matter experts annotates the most valuable data points
5. Repeat Step 2 to 4 until acquire enough data points.

Parameters

- **sampling** (*str or function*) – Sampling method for get the most valuable data points. Providing certified methods name (*most_confidence*, *entropy*, *least_confidence*, *margin*, *nearest_mean*, *fathest*) or custom function.
- **embeddings** (str or nlpatl.models.embeddings.Embeddings) – Function for converting raw data to embeddings. Providing model name according to embeddings type. For example, *multi-qa-MiniLM-L6-cos-v1* for *sentence_transformers*. bert-base-uncased` for *transformers*. *vgg16* for *torch_vision*.
- **embeddings_model_config** (*dict*) – Configuration for embeddings models. Optional. Ignored if using custom embeddings class
- **embeddings_type** (*str*) – Type of embeddings. *sentence_transformers* for text, *transformers* for text or *torch_vision* for image
- **clustering** (str or nlpatl.models.clustering.Clustering) – Function for clustering inputs. Either providing certified methods (*kmeans*) or custom function.
- **clustering_model_config** (*dict*) – Configuration for clustering models. Optional. Ignored if using custom clustering class
- **multi_label** (*bool*) – Indicate the classification model is multi-label or multi-class (or binary). Default is False.
- **name** (*str*) – Name of this learning.

clear_learn_data()
Clear all learn data points

educate(*index*, *x*, *x_features*, *y*)
Annotate data point. Only allowing annotate data point one by one. NOT batch.

Parameters

- **index** (*int*) – Index of data point.
- **x** (string, int, float or np.ndarray) – Raw data input. It can be text, number or numpy (for image).
- **x_features** (int, float or np.ndarray) – Data features
- **y** (*string, int, list of string (multi-label case) or list or int (multi-label case)*) – Label of data point

explore(*inputs*, *return_type='dict'*, *num_sample=2*)
Estimate the most valuable data points for annotation.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.
- **inputs** (*List[str]*) –

Returns The most valuable data points.

Return type nlpatl.dataset.Dataset objects or dict

explore_educate_in_notebook(*x*, *num_sample=2*, *data_type='text'*)
Estimate the most valuable data points for annotation and annotate it in IPython Notebook. Executing *explore* function and *educate* function sequentially.

Parameters

- **x** (list of string, int or float or np.ndarray) – Raw data inputs. It can be text, number or numpy (for image).
- **return_type** (*str*) – Data type of returning object. If *dict* is assigned. Return object is *dict*. Possible values are *dict* and *object*.
- **num_sample** (*int*) – Maximum number of data points for annotation.
- **data_type** (*str*) – Indicate the data format for displying in IPython Notebook. Possible values are *text* and *image*.

get_learn_data()
Get all learn data points

Returns Learnt data points

Return type Tuple of index list of int, x (str or numpy.ndarray) x_features (numpy.ndarray) and y (numpy.ndarray)

learn(*x*, *y*, *include_learn_data=True*)
Train the classification model.

Parameters

- **x** (list of string, int or float or np.ndarray.) – Raw data inputs. It can be text, number or numpy.
- **y** (*bool*) – Label of data inputs
- **include_learn_data** (*bool*) – Train the model whether including human annotated data and machine learning self annotated data. Default is True.

CHAPTER

TWO

MODEL

2.1 Classification

2.1.1 nlpatl.models.classification.sklearn_classification

sci-kit learn classification wrapper

class nlpatl.models.classification.sklearn_classification.**SkLearnClassification**(*model_name='logistic_regress*
model_config={},
name='sklearn_classification'

Bases: nlpatl.models.classification.classification.Classification

A wrapper of sci-kit learn classification class.

Parameters

- **model_name** (*str*) – sci-kit learn classification model name. Possible values are *logistic_regression*, *svc*, *linear_svc* and *random_forest*.
- **model_config** (*dict*) – Model paramateters. Refer to https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model
- **name** (*str*) – Name of this classification

```
>>> import nlpatl.models.classification as nmcla
>>> model = nmcla.SkLearnClassification()
```

predict_proba(*x*, *predict_config={}*)

Parameters

- **x** (*np.ndarray*) – Raw features
- **predict_config** (*dict*) – Model prediction paramateters. Refer to https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

Returns Feature and probabilities

Return type nlptatl.dataset.Dataset

train(*x*, *y*)

Parameters

- **x** (*np.ndarray*) – Raw features

- **y** (list of string, int or float or np.ndarray.) – Label of data inputs

2.1.2 nlpatl.models.classification.xgboost_classification

class nlpatl.models.classification.xgboost_classification.**XGBoostClassification**(*model_config={}*, *name='xgboost_classification'*)

Bases: *nlpatl.models.classification.sklearn_classification.SkLearnClassification*

A wrapper of xgboost classification class.

Parameters

- **model_config** (*dict*) – Model paramateters. Refer to https://xgboost.readthedocs.io/en/stable/python/python_api.html
- **name** (*str*) – Name of this classification

```
>>> import nlpatl.models.classification as nmcla
>>> model = nmcla.XGBoostClassification()
```

predict_proba(*x*, *predict_config={}*)

Parameters

- **x** (*np.ndarray*) – Raw features
- **predict_config** (*dict*) – Model prediction paramateters. Refer to https://xgboost.readthedocs.io/en/stable/python/python_api.html

Returns Feature and probabilities

Return type nlptatl.dataset.Dataset

train(*x*, *y*)

Parameters

- **x** (*np.ndarray*) – Raw features
- **y** (list of string, int or float or np.ndarray.) – Label of data inputs

2.2 Clustering

2.2.1 nlpatl.models.clustering.sklearn_clustering

class nlpatl.models.clustering.sklearn_clustering.**SkLearnClustering**(*model_name='kmeans'*, *model_config={}*, *name='sklearn_clustering'*)

Bases: nlpatl.models.clustering.clustering.Clustering

A wrapper of sci-kit learn clustering class.

Parameters

- **model_name** (*str*) – sci-kit learn clustering model name. Possible values are *kmeans*.
- **model_config** (*dict*) – Model paramateters. Refer to https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster

- **name** (*str*) – Name of this clustering

```
>>> import nlpatl.models.clustering as nmclu
>>> model = nmclu.SkLearnClustering()
```

predict_proba(*x*, *predict_config={}*)

Parameters

- **x** (*np.ndarray*) – Raw features
- **predict_config** (*dict*) – Model prediction paramateters. Refer to https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster

Returns Feature and probabilities

Return type nlptatl.dataset.Dataset

train(*x*)

Parameters **x** (*np.ndarray*) – Raw features

2.3 Embeddings

2.3.1 nlpatl.models.embeddings.sentence_transformers

class nlpatl.models.embeddings.sentence_transformers.**SentenceTransformers**(*model_name_or_path*, *batch_size=16*, *name='sentence_transformers'*)

Bases: nlpatl.models.embeddings.embeddings.Embeddings

A wrapper of transformers class.

Parameters

- **model_name_or_path** (*str*) – sentence transformers model name.
- **batch_size** (*int*) – Batch size of data processing. Default is 16
- **model_config** (*dict*) – Model paramateters. Refer to https://www.sbert.net/docs/pretrained_models.html
- **name** (*str*) – Name of this embeddings

```
>>> import nlpatl.models.embeddings as nme
>>> model = nme.SentenceTransformers()
```

convert(*x*)

Parameters **x** (*np.ndarray*) – Raw features

Returns Vectors of features

Return type np.ndarray

2.3.2 nlpatl.models.embeddings.transformers

class nlpatl.models.embeddings.transformers.**Transformers**(*model_name_or_path*, *batch_size=16*, *padding=False*, *truncation=False*, *nn_fwk=None*, *name='transformers'*)

Bases: nlpatl.models.embeddings.embeddings.Embeddings

A wrapper of transformers class.

Parameters

- **model_name_or_path** (*str*) – transformers model name.
- **batch_size** (*int*) – Batch size of data processing. Default is 16
- **padding** (*bool*) – Inputs may not have same size. Set True to pad it. Default is False
- **truncation** (*bool*) – Inputs may not have same size. Set True to truncate it. Default is False
- **nn_fwk** (*str*) – Neual network framework. Either pt (for PyTorch) or tf (for TensorFlow)
- **model_config** (*dict*) – Model paramateters. Refer to https://huggingface.co/docs/transformers/index
- **name** (*str*) – Name of this embeddings

```
>>> import nlpatl.models.embeddings as nme
>>> model = nme.Transformers()
```

convert(*x*)

Parameters **x** (*np.ndarray*) – Raw features

Returns Vectors of features

Return type np.ndarray

2.3.3 nlpatl.models.embeddings.torchvision

class nlpatl.models.embeddings.torchvision.**TorchVision**(*model_name_or_path*, *batch_size=16*, *model_config={'pretrained': True}*, *transform=None*, *name='torchvision'*)

Bases: nlpatl.models.embeddings.embeddings.Embeddings

A wrapper of torch vision class.

Parameters

- **model_name_or_path** (*str*) – torch vision model name. Possible values are *resnet18*, *alexnet* and *vgg16*.
- **batch_size** (*int*) – Batch size of data processing. Default is 16
- **model_config** (*dict*) – Model paramateters. Refer to https://pytorch.org/vision/stable/models.html
- **transform** – Preprocessing function
- **name** (*str*) – Name of this embeddings

```
>>> import nlpatl.models.embeddings as nme
>>> model = nme.TrochVision()
```

convert(*x*)

Parameters **x** (*np.ndarray*) – Raw features

Returns Vectors of features

Return type np.ndarray

CHAPTER

THREE

SAMPLING

3.1 Certainity Sampling

3.1.1 nlpatl.sampling.certainty.most_confidence

class nlpatl.sampling.certainty.most_confidence.**MostConfidenceSampling**(*threshold=0.85*, *name='most_confidence_sampling'*)

Bases: nlpatl.sampling.sampling.Sampling

Sampling data points if the confidence is higher than threshold. Refer to https://markcartwright.com/files/wang2019active.pdf

Parameters

- **threshold** (*float*) – Minimum probability of model prediction. Default value is 0.85
- **name** (*str*) – Name of this sampling

sample(*data*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data** (*<MagicMock id='140655954062016'>*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

3.2 Uncertainity Learning

3.2.1 nlpatl.sampling.uncertainty.least_confidence

class nlpatl.sampling.uncertainty.least_confidence.**LeastConfidenceSampling**(*name='least_confidence_sampling'*)

Bases: nlpatl.sampling.sampling.Sampling

Sampling data points according to the least confidence. Pick the lowest probabilies for the highest class.

Parameters **name** (*str*) – Name of this sampling

sample(*data*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data** (*<MagicMock id='140655953918608'>*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

3.2.2 nlpatl.sampling.uncertainty.entropy

class nlpatl.sampling.uncertainty.entropy.**EntropySampling**(*name='entropy_sampling'*)

Bases: nlpatl.sampling.sampling.Sampling

Sampling data points according to the entropy. Pick the highest N data points

Parameters **name** (*str*) – Name of this sampling

sample(*data*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data** (*<MagicMock id='140655953764304'>*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

3.2.3 nlpatl.sampling.uncertainty.margin

class nlpatl.sampling.uncertainty.margin.**MarginSampling**(*name='margin_sampling'*)

Bases: nlpatl.sampling.sampling.Sampling

Sampling data points according to the margin confidence. Pick the lowest probabilies difference between the highest class and second higest class.

Parameters **name** (*str*) – Name of this sampling

sample(*data*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data** (*<MagicMock id='140655953899968'>*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

3.2.4 nlpatl.sampling.uncertainty.mismatch

class nlpatl.sampling.uncertainty.mismatch.**MismatchSampling**(*name='mismatch_sampling'*)
Bases: nlpatl.sampling.sampling.Sampling

Sampling data points according to the mismatch. Pick the N data points randomly.

Parameters **name** (*str*) – Name of this sampling

sample(*data1*, *data2*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data1** (*Union[List[str], List[int], List[float], <MagicMock id='140655953612656'>]*) –
- **data2** (*Union[List[str], List[int], List[float], <MagicMock id='140655953590544'>]*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

3.3 Clustering Sampling

3.3.1 nlpatl.sampling.clustering.farthest

class nlpatl.sampling.clustering.farthest.**FarthestSampling**(*name='farthest_sampling'*)
Bases: nlpatl.sampling.sampling.Sampling

Sampling data points according to the distances of cluster centriod. Picking n farthest data points per number of cluster. http://zhaoshuyang.com/static/documents/MAL2.pdf

Parameters **name** (*str*) – Name of this sampling

sample(*data*, *groups*, *num_sample*)

Parameters

- **x** – Values of determine the sampling
- **num_sample** (*int*) – Total number of sample for labeling
- **data** (*<MagicMock id='140655953397456'>*) –
- **groups** (*<MagicMock id='140655954027952'>*) –

Returns Tuple of target indices and sampling values

Return type Tuple of numpy.ndarray, numpy.ndarray

See modindex for API.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

T

U

X