D-SAIL

Release 0.1

D-SAIL

Sep 09, 2021

GETTING STARTED

Welcome to the documentation of Distributed Secure AI Learning!

The code is available at https://github.com/XavierLessage/D-SAIL.

CHAPTER

ONE

API

Welcome to the API. The main packages and modules are available in the 'Quick access' below.

All functions are defined in details further down the page.

1.1 Quick access

dicom_converter
dicom_pseudonymizer
federated_learning

1.1.1 dicom_converter

Modules

dicom_converter.add_metadata	Created on Tue Sep 7 10:06:08 2021
dicom_converter.classify_data	Created on Tue Sep 7 13:58:39 2021
dicom_converter.utils	

dicom_converter.add_metadata

Created on Tue Sep 7 10:06:08 2021

@author: eloyen

Functions

add_label_in_dcm(dcmFile, label, tag)	Add the label metadata in the DICOM file
go_through_folder(folderPath, label, tag)	Go trough the folder to add the label tag metadata

dicom_converter.classify_data

Created on Tue Sep 7 13:58:39 2021

@author: eloyen

Functions

classify_in_labelled_folders(inputFolder, ...)	Classify the IMAGES vs METADATA in folders according to the label tag
get_tag_from_json(json_path, tag)	Get tag value from .json fiel containing DICOM metadata

dicom_converter.utils

Modules

dicom_converter.utils.cat_to_dataset	
dicom_converter.utils.dicom_to_img	Created on Wed Sep 1 15:44:11 2021
dicom_converter.utils.hospital_split	Splits a large dataset (grouped by category) into small datasets with custom individual sizes

dicom_converter.utils.cat_to_dataset

Functions

cat_to_dataset([datapath, ...])	Divides a directory containing only contain folders with each category to classify into 'train', 'valid' and 'test' folders contain each category for ML with custom splitting percentages trainset_percentage + validset_percentage + testset_percentage should be = 1 !
dataset_to_cat([datapath])	Groups a directory containing a 'train', 'test' and 'valid' (code is case sensitive) folders for ML with category sub folders into category folders.

dicom_converter.utils.dicom_to_img

Created on Wed Sep 1 15:44:11 2021

Module with functions used to convert DICOM files (.dcm) to .png/.bmp and .json files and from .png/.bmp to DICOM.

The function 'compress_to_png' calls executables from OpenJPEG (https://www.openjpeg.org/) available at 'https://github.com/uclouvain/openjpeg/releases/tag/v2.4.0'.

Functions

compress_to_png(file_path, software_root[, ...])	Compresses an image to a .png with a specified 'compress_ratio'
decompose_dicom(file_path, output_path[, ...])	Divides dicom file into a .json file with the dicom metadata and a .'img_format' file containing the image.
dicom_from_img_or_json(file_path, output_folder)	Creates a dicom from a .json file or a .png/.bmp file
get_tag_from_json(json_path, tag)	Get tag value from .json fiel containing DICOM metadata
img_from_dicom(ds)	Extract array from dicom dataset 'dcm' with [0,256] pixel intensities.

dicom_converter.utils.hospital_split

Splits a large dataset (grouped by category) into small datasets with custom individual sizes

Functions

hospital_split([number_of_datasplits, ...])	

Parameters

1.1.2 dicom_pseudonymizer

Modules

dicom_pseudonymizer.anonymizer
dicom_pseudonymizer.utils

dicom_pseudonymizer.anonymizer

Functions

anonymize(input_path, output_path, ...)	Read data from input path (folder or file) and launch the anonymization.
generate_actions_dictionary(map_action_tag)	Generate a new dictionary which maps actions function to tags
main([defined_action_map])	

dicom_pseudonymizer.utils

Modules

dicom_pseudonymizer.utils.dicom_fields	Tags anonymized in DICOM standard Documentation for groups meaning can be found in default associated actions.
dicom_pseudonymizer.utils.format_tag	Utility for printing the tags in the original hex format.
dicom_pseudonymizer.utils.simple_dicomanonymizer	

dicom_pseudonymizer.utils.dicom_fields

Tags anonymized in DICOM standard Documentation for groups meaning can be found in default associated actions.
http://dicom.nema.org/dicom/2013/output/chtml/part15/chapter_E.html#table_E.1-1

This code was taken and adapted from https://github.com/KitwareMedical/dicom-anonymizer

dicom_pseudonymizer.utils.format_tag

Utility for printing the tags in the original hex format.

This code was taken and adapted from https://github.com/KitwareMedical/dicom-anonymizer

Functions

hex_to_string(x)	Convert a tag number to it's original hex string.
tag_to_hex_strings(tag)	Convert a tag tuple to a tuple of full hex number strings.

dicom_pseudonymizer.utils.simple_dicomanonymizer

Functions

anonymize_dataset(dataset[, ...])	Anonymize a pydicom Dataset by using anonymization rules which links an action to a tag
anonymize_dicom_file(in_file, out_file[, ...])	Anonymize a DICOM file by modifying personal tags
clean(dataset, tag)	C - clean, that is replace with values of similar meaning known not to contain identifying information and consistent with the VR
delete(dataset, tag)	X - remove
delete_element(dataset, element)	Delete the element from the dataset.
delete_or_empty(dataset, tag)	X/Z - X unless Z is required to maintain IOD conformance (Type 3 versus Type 2)
delete_or_empty_or_replace(dataset, tag)	X/Z/D - X unless Z or D is required to maintain IOD conformance (Type 3 versus Type 2 versus Type 1)
delete_or_empty_or_replace_UID(dataset, tag)	X/Z/U* - X unless Z or replacement of contained instance UIDs (U) is required to maintain IOD conformance (Type 3 versus Type 2 versus Type 1 sequences containing UID references)
delete_or_replace(dataset, tag)	X/D - X unless D is required to maintain IOD conformance (Type 3 versus Type 1)
empty(dataset, tag)	Z - replace with a zero length value, or a non-zero length value that may be a dummy value and consistent with the VR
empty_element(element)	Clean element according to the element's VR: - SH, PN, UI, LO, CS: value will be set to '' - DA: value will be replaced by '00010101' - TM: value will be replaced by '000000.00' - UL: value will be replaced by 0 - SQ: all subelement will be called with "empty_element"
empty_or_replace(dataset, tag)	Z/D - Z unless D is required to maintain IOD conformance (Type 2 versus Type 1)
generate_actions(tag_list, action[, options])	Generate a dictionary using list values as tag and assign the same value to all
get_private_tag(dataset, tag)	Get the creator and element from tag
get_private_tags(anonymization_actions, dataset)	Extract private tag as a list of object with creator and element
initialize_actions()	Initialize anonymization actions with DICOM standard values
keep(dataset, tag)	K - keep (unchanged for non-sequence attributes, cleaned for sequences)
regexp(options)	Apply a regexp method to the dataset
replace(dataset, tag)	D - replace with a non-zero length value that may be a dummy value and consistent with the VR
replace_UID(dataset, tag)	U - replace with a non-zero length UID that is internally consistent within a set of Instances Lazy solution : Replace with empty string
replace_and_keep_correspondence(dataset, tag)	P - addition to pseudonimize the code and keep a lookup table.

continues on next page

Table 13 – continued from previous page

replace_element(element)	Replace element's value according to it's VR: - DA: cf replace_element_date - TM: replace with '000000.00' - LO, SH, PN, CS: replace with 'Anonymized' - UI: cf replace_element_UID - IS: replace with '0' - FD, FL, SS, US: replace with 0 - ST: replace with '' - SQ: call replace_element for all sub elements - DT: cf replace_element_date_time
replace_element_UID(element)	Keep char value but replace char number with random number The replaced value is kept in a dictionary link to the initial element.value in order to automatically apply the same replaced value if we have an other UID with the same value
replace_element_date(element)	Replace date element's value with '00010101'
replace_element_date_time(element)	Replace date time element's value with '00010101010101.000000+0000'

1.1.3 federated_learning

Modules

federated_learning.client
federated_learning.server

federated_learning.client

Modules

federated_learning.client.client
federated_learning.client.dsail

federated_learning.client.client

Functions

main([arch, lr, epochs, bs, device, port, ...])

federated_learning.client.dsail

Modules

federated_learning.client.dsail.differential_privacy
federated_learning.client.dsail.federated_learning
federated_learning.client.dsail.utils

federated_learning.client.dsail.differential_privacy

Classes

DPCallback(alphas, noise_multiplier, ...)

federated_learning.client.dsail.federated_learning

Classes

FLClient(learn, lr, ep, apply_dp, alphas, ...)

federated_learning.client.dsail.utils

Functions

get_imbalance_weights(ds)
save_matrix(learn, path)
save_roc(learn, path)
set_seed(dls, seed)

Classes

ImbalancedDatasetSampler(dataset, indices, ...)	Samples elements randomly from a given list of indices for imbalanced dataset

federated_learning.server

Modules

federated_learning.server.server

federated_learning.server.server

Functions

main([fraction_fit, min_fit_clients, ...])

Classes

SaveModelStrategy(*args, **kwargs)

1.2 dicom_converter

1.2.1 add_metadata

Created on Tue Sep 7 10:06:08 2021

@author: eloyen

dicom_converter.add_metadata.**add_label_in_dcm**(*dcmFile*, *label*, *tag*)
Add the label metadata in the DICOM file

Parameters

dcmFile [string] /.../dicominfo.json

label [short string] ex: '0', '1'

tag [tuple of two elements] DICOM tag, must be in hexagonal format. ex: (0x10,0x20)

Returns

dcmFile [string] dcmFile with the new added metadata

dicom_converter.add_metadata.**go_through_folder**(*folderPath*, *label*, *tag*)
Go trough the folder to add the label tag metadata

Parameters

folderPath [string] /…/dicoms/

label [short string] ex: '0', '1'

tag [tuple of two elements] DICOM tag, must be in hexagonal format. ex: (0x10,0x20)

Returns

None.

1.2.2 classify_data

Created on Tue Sep 7 13:58:39 2021

@author: eloyen

dicom_converter.classify_data.**classify_in_labelled_folders**(*inputFolder*, *labelTag*, *outputDir*)

Classify the IMAGES vs METADATA in folders according to the label tag

Parameters

inputFolder [string] /…/dicoms/

labelTag [tuple of two elements] DICOM tag, must be in hexagonal format. ex: (0x10,0x20)

outputDir [string] /…/outputs

Returns

——-

None.

dicom_converter.classify_data.**get_tag_from_json**(*json_path*, *tag*)

Get tag value from .json fiel containing DICOM metadata

Parameters

json_path [string] /…/dicominfo.json

tag [tuple of two elements] DICOM tag, must be in hexagonal format. ex: (0x10,0x20)

Returns

value [Value stored in tag]

1.2.3 utils

cat_to_dataset

dicom_converter.utils.cat_to_dataset.**cat_to_dataset**(*datapath=''*, *trainset_percentage=0.7*, *validset_percentage=0.2*, *testset_percentage=0.1*, *seed=3*)

Divides a directory containing only contain folders with each category to classify into 'train', 'valid' and 'test' folders contain each category for ML with custom splitting percentages trainset_percentage + validset_percentage + testset_percentage should be = 1 !

Parameters

datapath [string] Datapath directory should only contain folders with each category to classify. Each category fodler must only contain image files of that category.. The default is "".

trainset_percentage [float] Training set percentage (trainset_percentage + validset_percentage + testset_percentage should be = 1). The default is 0.7.

validset_percentage [float] Validation set percentage (trainset_percentage + validset_percentage + testset_percentage should be = 1). The default is 0.2.

testset_percentage [float] Test set percentage (trainset_percentage + validset_percentage + testset_percentage should be = 1). The default is 0.1.

seed [integer] Random seed. The default is 3.

Returns

None.

dicom_converter.utils.cat_to_dataset.**dataset_to_cat**(*datapath=''*)

Groups a directory containing a 'train', 'test' and 'valid' (code is case sensitive) folders for ML with category sub folders into category folders.

Parameters

datapath [string] Datapath directory should only contain a 'train', 'test' and 'valid' (code is case sensitive) folders. Each fodler should only contain folders with each category to classify. Each category fodler must only contain image files of that category. The default is "".

Returns

None.

dicom_to_img

Created on Wed Sep 1 15:44:11 2021

Module with functions used to convert DICOM files (.dcm) to .png/.bmp and .json files and from .png/.bmp to DICOM.

The function 'compress_to_png' calls executables from OpenJPEG (https://www.openjpeg.org/) available at 'https://github.com/uclouvain/openjpeg/releases/tag/v2.4.0'.

dicom_converter.utils.dicom_to_img.**compress_to_png**(*file_path*, *software_root*, *compress_ratio=1*)

Compresses an image to a .png with a specified 'compress_ratio'

Parameters

file_path [string] /.../filename.png

software_root [string] Path to the folder containing the openjpeg .exe programs

compress_ratio [int, optional] Best if multiple of 8. The default is 1.

Returns

None.

dicom_converter.utils.dicom_to_img.**decompose_dicom**(*file_path*, *output_path*, *img_format='bmp'*, *removeImgInJson=False*)

Divides dicom file into a .json file with the dicom metadata and a .'img_format' file containing the image.

Parameters

file_path [string] /.../filename.dcm

output_path [string] /.../foldername/

img_format [string, optional] Image file format : bmp, png, ... The default is 'bmp'.

removeImgInJson [True/False, optional] Removes PixelData from dicom metadata. The default is False.

Returns

None.

dicom_converter.utils.dicom_to_img.**dicom_from_img_or_json**(*file_path*, *output_folder*, *metadata_path=None*, *randomizeName=False*, *verbose=False*)

Creates a dicom from a .json file or a .png/.bmp file

Parameters

file_path [string] /…/filename.png|.json

output_path [string] /…/foldername/

metadata_path [string, optional] path to reference dicom file. The default is dcm_file_path.

randomizeName [True/False] Creates a random patientID, optional

verbose [True/False, optional] Raises warnings. The default is False.

Returns

None.

dicom_converter.utils.dicom_to_img.**get_tag_from_json**(*json_path*, *tag*)

Get tag value from .json fiel containing DICOM metadata

Parameters

json_path [string] /…/dicominfo.json

tag [tuple of two elements] DICOM tag, must be in hexagonal format. ex: (0x10,0x20)

Returns

value [Value stored in tag]

dicom_converter.utils.dicom_to_img.**img_from_dicom**(*ds*)

Extract array from dicom dataset 'dcm' with [0,256] pixel intensities.

Parameters

dcm [FileDataset object of pydicom.dataset module]

Returns

d [array] Image array of the dicom dataset

hospital_split

Splits a large dataset (grouped by category) into small datasets with custom individual sizes

dicom_converter.utils.hospital_split.**hospital_split**(*number_of_datasplits=3*, *split_percentages=array([0.5, 0.3, 0.2])*, *seed=3*, *datapath=''*)

Parameters

number_of_datasplits [integer] Number of data splits (hospitals needed). The default is 3.

split_percentages [array] split percentage by dataset (hospital) the number of components must be equal to 'number_of_datasplits'. Be careful that the sum of them is exactly = 1. The default is np.array([0.5,0.3,0.2]).

seed [integer] Random seed used. The default is 3.

datapath [string] Datapath directory should only contain folders with each category to classify. Each category fodler must only contain image files of that category. The default is "".

Returns

None.

1.3 dicom_pseudonymizer

1.3.1 anonymizer

dicom_pseudonymizer.anonymizer.**anonymize**(*input_path: str*, *output_path: str*, *lookup_path: str*, *anonymization_actions: dict*, *delete_private_tags: bool*, *rename_files: bool*) → None

Read data from input path (folder or file) and launch the anonymization.

Parameters

input_path [str] Path to a folder or to a file. If set to a folder, then cross all over subfiles and apply anonymization.

output_path [str] Path to a folder or to a file.

csv_path [str] Path to lookup table csv path.

anonymization_actions [dict] List of actions that will be applied on tags.

delete_private_tags [bool] Whether to delete private tags.

rename_files [bool] Whether to remane output files with pseudo.

Returns

None.

dicom_pseudonymizer.anonymizer.**generate_actions_dictionary**(*map_action_tag*, *defined_action_map={}*) → dict

Generate a new dictionary which maps actions function to tags

Parameters

map_action_tag [dict] Link actions to tags

defined_action_map [dict] Link action name to action function

Returns

generated_map [dict.] The map of actions.

dicom_pseudonymizer.anonymizer.**main**(*defined_action_map={}*)

1.3.2 utils

dicom_fields

Tags anonymized in DICOM standard Documentation for groups meaning can be found in default associated actions. http://dicom.nema.org/dicom/2013/output/chtml/part15/chapter_E.html#table_E.1-1

This code was taken and adapted from https://github.com/KitwareMedical/dicom-anonymizer

format_tag

Utility for printing the tags in the original hex format.

This code was taken and adapted from https://github.com/KitwareMedical/dicom-anonymizer

dicom_pseudonymizer.utils.format_tag.**hex_to_string**(*x: hex*)

Convert a tag number to it's original hex string. E.g. if a tag has the hex number 0x0008, it becomes 8, and we then convert it back to 0x0008 (as a string).

Parameters

x [hex] The hex number to be converted.

Returns

s [str] The hex tag converted to hex number string.

dicom_pseudonymizer.utils.format_tag.**tag_to_hex_strings**(*tag: tuple*)

Convert a tag tuple to a tuple of full hex number strings.

E.g. (0x0008, 0x0010) is evaluated as (8, 16) by python. So we convert it back to a string '(0x0008, 0x0010)' for pretty printing.

Parameters

tag [tuple] The tuple to be converted from hex numbers to hex number strings.

Returns

s [tuple] The hex tag converted to hex number string.

simple_dicomanonymizer

dicom_pseudonymizer.utils.simple_dicomanonymizer.**anonymize_dataset**(*dataset: pydicom.dataset.Dataset*, *extra_anonymization_rules: Optional[dict] = None*, *delete_private_tags: bool = True*) → None

Anonymize a pydicom Dataset by using anonymization rules which links an action to a tag

Parameters

dataset [FileDataset object of pydicom.dataset module] Dataset to be anonymized

extra_anonymization_rules [dict] Rules to be applied on the dataset

delete_private_tags [bool] Define if private tags should be delete or not

Returns

None.

dicom_pseudonymizer.utils.simple_dicomanonymizer.**anonymize_dicom_file**(*in_file: str*, *out_file: str*, *lookup_file: Optional[str] = None*, *extra_anonymization_rules: Optional[dict] = None*, *delete_private_tags: bool = True*, *rename_files: bool = False*) → None

Anonymize a DICOM file by modifying personal tags

Conforms to DICOM standard except for customer specificities.

Parameters

in_file [str] File path or file-like object to read from

out_file [str] File path or file-like object to write to

lookup_file [str] File path to the lookup table.

extra_anonymization_rules [str] Add more tag's actions

delete_private_tags [bool] Define if private tags should be delete or not

Returns

d: dict The generated dictionary with the action to be applied.

dicom_pseudonymizer.utils.simple_dicomanonymizer.**clean**(*dataset*, *tag*)
C - clean, that is replace with values of similar meaning known not to contain identifying information and consistent with the VR

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete**(*dataset*, *tag*)
X - remove

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete_element**(*dataset*, *element*)
Delete the element from the dataset. If VR's element is a date, then it will be replaced by 00010101

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete_or_empty**(*dataset*, *tag*)
X/Z - X unless Z is required to maintain IOD conformance (Type 3 versus Type 2)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete_or_empty_or_replace**(*dataset*, *tag*)
X/Z/D - X unless Z or D is required to maintain IOD conformance (Type 3 versus Type 2 versus Type 1)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete_or_empty_or_replace_UID**(*dataset*, *tag*)
X/Z/U* - X unless Z or replacement of contained instance UIDs (U) is required to maintain IOD conformance (Type 3 versus Type 2 versus Type 1 sequences containing UID references)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**delete_or_replace**(*dataset*, *tag*)
X/D - X unless D is required to maintain IOD conformance (Type 3 versus Type 1)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**empty**(*dataset*, *tag*)
Z - replace with a zero length value, or a non-zero length value that may be a dummy value and consistent with the VR

dicom_pseudonymizer.utils.simple_dicomanonymizer.**empty_element**(*element*)
Clean element according to the element's VR: - SH, PN, UI, LO, CS: value will be set to '' - DA: value will be replaced by '00010101' - TM: value will be replaced by '000000.00' - UL: value will be replaced by 0 - SQ: all subelement will be called with "empty_element"

dicom_pseudonymizer.utils.simple_dicomanonymizer.**empty_or_replace**(*dataset*, *tag*)
Z/D - Z unless D is required to maintain IOD conformance (Type 2 versus Type 1)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**generate_actions**(*tag_list: list*, *action*, *options: Optional[dict] = None*) → dict
Generate a dictionary using list values as tag and assign the same value to all

Parameters

tag_list [list] List of tags which will have the same associated actions

action [function] Define the action that will be use. It can be a callable custom function or a name of a pre-defined action from simpledicomanonymizer

options [dict] Define options tht will be affected to the action (like regexp)

Returns

d: dict The generated dictionary with the action to be applied.

dicom_pseudonymizer.utils.simple_dicomanonymizer.**get_private_tag**(*dataset*, *tag*)
Get the creator and element from tag

Parameters

dataset [FileDataset object of pydicom.dataset module] Dicom dataset

tag [tuple] Tag from which we want to extract private information

Returns

d [dict] Dictionary with creator of the tag and tag element (which contains element + offset)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**get_private_tags**(*anonymization_actions: dict*, *dataset: pydicom.dataset.Dataset*) → List[dict]
Extract private tag as a list of object with creator and element

Parameters

anonymization_actions [dict] List of tags associated to an action.

dataset [FileDataset object of pydicom.dataset module] Dicom dataset which will be anonymize and contains all private tags

Returns

d [Array of object] Array with private tags

dicom_pseudonymizer.utils.simple_dicomanonymizer.**initialize_actions**() → dict
Initialize anonymization actions with DICOM standard values

Parameters

None.

Returns

d: dict Dict object which map actions to tags

dicom_pseudonymizer.utils.simple_dicomanonymizer.**keep**(*dataset*, *tag*)
K - keep (unchanged for non-sequence attributes, cleaned for sequences)

dicom_pseudonymizer.utils.simple_dicomanonymizer.**regexp**(*options: dict*)
Apply a regexp method to the dataset

Parameters

options [dict]

Contains two values:

- find: which string should be found
- replace: string that will replace the found string

Returns

s [string] The string with the regexp applied.

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace**(*dataset*, *tag*)
D - replace with a non-zero length value that may be a dummy value and consistent with the VR

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_UID**(*dataset*, *tag*)
U - replace with a non-zero length UID that is internally consistent within a set of Instances Lazy solution : Replace with empty string

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_and_keep_correspondence**(*dataset*, *tag*)
P - addition to pseudonimize the code and keep a lookup table. If used, it should be called when tag (0x0010, 0x0020) (PatientID) is encountered. It also replaces implicitly the tag (0x0008, 0x0050) (AccessionNumber). A lookup table (csv file) is create with columns: 'old_patient_id', 'new_patient_id', 'old_accession_number', 'new_accession_number'

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_element**(*element*)
Replace element's value according to it's VR: - DA: cf replace_element_date - TM: replace with '000000.00' - LO, SH, PN, CS: replace with 'Anonymized' - UI: cf replace_element_UID - IS: replace with '0' - FD, FL, SS, US: replace with 0 - ST: replace with '' - SQ: call replace_element for all sub elements - DT: cf replace_element_date_time

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_element_UID**(*element*)
Keep char value but replace char number with random number The replaced value is kept in a dictionary link to the initial element.value in order to automatically apply the same replaced value if we have an other UID with the same value

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_element_date**(*element*)
Replace date element's value with '00010101'

dicom_pseudonymizer.utils.simple_dicomanonymizer.**replace_element_date_time**(*element*)
Replace date time element's value with '00010101010101.000000+0000'

1.4 federated_learning

1.4.1 client

federated_learning.client.client.**as_tensor**(*data*, *dtype=None*, *device=None*) → Tensor
Convert the data into a *torch.Tensor*. If the data is already a *Tensor* with the same *dtype* and *device*, no copy will be performed, otherwise a new *Tensor* will be returned with computational graph retained if data *Tensor* has `requires_grad=True`. Similarly, if the data is an `ndarray` of the corresponding *dtype* and the *device* is the cpu, no copy will be performed.

Args:

data (array_like): Initial data for the tensor. Can be a list, tuple, NumPy `ndarray`, scalar, and other types.

dtype (torch.dtype, optional): the desired data type of returned tensor. Default: if None, infers data type from data.

device (torch.device, optional): the desired device of returned tensor. Default: if None, uses the current device for the default tensor type (see torch.set_default_tensor_type()). device will be the CPU for CPU tensor types and the current CUDA device for CUDA tensor types.

Example:

```
>>> a = numpy.array([1, 2, 3])
>>> t = torch.as_tensor(a)
>>> t
tensor([ 1,  2,  3])
>>> t[0] = -1
>>> a
array([-1,  2,  3])

>>> a = numpy.array([1, 2, 3])
>>> t = torch.as_tensor(a, device=torch.device('cuda'))
>>> t
tensor([ 1,  2,  3])
>>> t[0] = -1
>>> a
array([1,  2,  3])
```

federated_learning.client.client.**main**(*arch: str <Architecture to use> = 'resnet18'*, *lr: int <Learning Rate> = 0.0006000000000000001*, *epochs: int <number of epochs for training> = 5*, *bs: int <Batch size to use> = 64*, *device: str <Which device to use> = 'cuda:0'*, *port: int <The port used for federated learning> = 8080*, *apply_dp: <Learning rate> = True*, *alphas: range <Alphas> = range(2, 32)*, *noise_multiplier: int <Noise injected in DP> = 0.5*, *max_grad_norm: int <Maximum Gradient Norm when clipping> = 1.0*, *delta: int <Delta> = 1e-05*, *matrix_path: str <Pass a value to save the confusion matrix> = None*, *csv_path: str <Pass a value to store the logs in csv> = None*, *roc_path: str <Pass a value to store the ROC-AUC curve> = None*, *data_path: str <datapath to use> = '../../../../Hospitals/H0'*, *seed: int <Pass a value to set seed> = 42*)

dsail.differential_privacy

class federated_learning.client.dsail.differential_privacy.**DPCallback**(*alphas*, *noise_multiplier*, *max_grad_norm*, *delta*, *device*)

Bases: fastai.callback.core.Callback

after_epoch()

before_step()

dsail.federated_learning

class federated_learning.client.dsail.federated_learning.**FLClient**(*learn*, *lr*, *ep*, *apply_dp*, *alphas*, *noise_multiplier*, *max_grad_norm*, *delta*, *device*, *csv_path*, *data_path*, *matrix_path*, *roc_path*)

Bases: object

evaluate(*parameters*, *config*)

fit(*parameters*, *config*)

get_parameters()

set_parameters(*parameters*)

dsail.utils

class federated_learning.client.dsail.utils.**ImbalancedDatasetSampler**(*dataset*, *indices: Optional[list] = None*, *num_samples: Optional[int] = None*, *callback_get_label: Optional[Callable] = None*)

Bases: Generic[torch.utils.data.sampler.T_co]

Samples elements randomly from a given list of indices for imbalanced dataset

Parameters

indices: list a list of indices

num_samples: int number of samples to draw

callback_get_label: Callable a callback-like function which takes two arguments - dataset and index

federated_learning.client.dsail.utils.**get_imbalance_weights**(*ds*)

federated_learning.client.dsail.utils.**save_matrix**(*learn*, *path*)

federated_learning.client.dsail.utils.**save_roc**(*learn*, *path*)

federated_learning.client.dsail.utils.**set_seed**(*dls*, *seed*)

1.4.2 server

class federated_learning.server.server.**SaveModelStrategy**(**args*, ***kwargs*)
Bases: flwr.server.strategy.fedavg.FedAvg

aggregate_fit(*rnd: int*, *results*, *failures*)
Aggregate fit results using weighted average.

federated_learning.server.server.**main**(*fraction_fit: float <The fraction of available client used for training> = 1.0*, *min_fit_clients: int <The minimum number of clients used to start training> = 3*, *min_available_clients: int <The minimum number of clients used to start server> = 3*, *min_eval_clients: int <The minimum number of clients used for evaluation> = 3*, *num_rounds: int <The number of rounds of training> = 3*, *save_path: str <Set a path to save weights> = None*)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

d

f

INDEX

R

S

T