CLEP Documentation

Release 0.0.2

Vinay Bharadhwaj, Daniel Domingo-Fernández and Charles Taple

Oct 19, 2020

CONTENTS:

Release notes : https://github.com/hybrid-kg/clep/releases

CHAPTER

ONE

CLEP: A HYBRID DATA- AND KNOWLEDGE- DRIVEN FRAMEWORK FOR GENERATING PATIENT REPRESENTATIONS.

CLEP has three main subgroups: `sample_scoring`, `embedding`, `classify`.

1. The `sample_scoring` module generates a score for every patient-feature pair.

2. The `embedding` module overlays the patients on the prior knowledge in-order generate a new KG, whose embedding is generated using KGE models from PyKEEN(Ali, *et al.*,2020).

3. The `classify` module classifies the generated embedding model (or any data that is passed to it) using generic classification models.

1.1 Welcome to CLEP's documentation!

Release notes : https://github.com/hybrid-kg/clep/releases

1.1.1 CLEP: A Hybrid Data- and Knowledge- Driven Framework for Generating Patient Representations.

CLEP has three main subgroups: `sample_scoring`, `embedding`, `classify`.

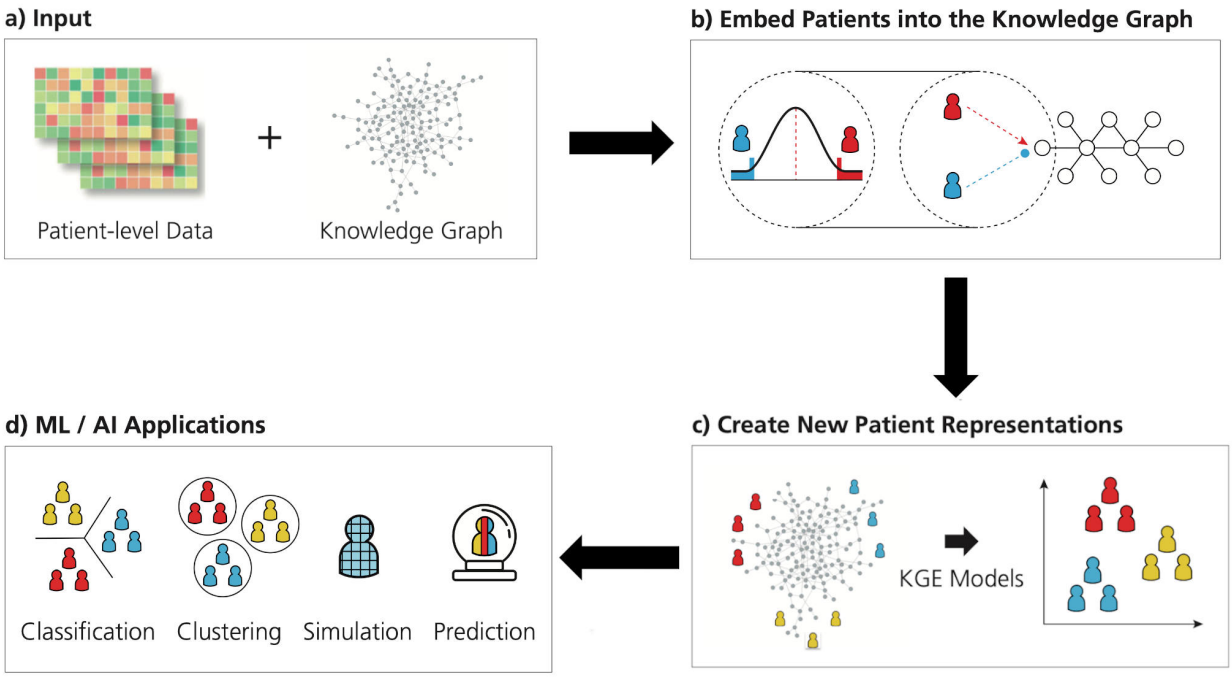
1. The `sample_scoring` module generates a score for every patient-feature pair.

2. The `embedding` module overlays the patients on the prior knowledge in-order generate a new KG, whose embedding is generated using KGE models from PyKEEN(Ali, *et al.*,2020).

3. The `classify` module classifies the generated embedding model (or any data that is passed to it) using generic classification models.

1.1.2 General info

CLEP is a framework that contains novel methods for generating patient representations from any patient level data and its corresponding prior knowledge encoded in a knowledge graph. The framework is depicted in the graphic below



1.1.3 Installation

In-order to install CLEP, an installation of R is **required** with a copy of Limma. Once they are installed, you can install CLEP package from pypi.

```
# Use pip to install the latest release
$ python3 -m pip install clep
```

You may instead want to use the development version from Github, by running

```
$ python3 -m pip install git+https://github.com/hybrid-kg/clep.git
```

For contributors, the repository can be cloned from GitHub and installed in editable mode using:

```
$ git clone https://github.com/hybrid-kg/clep.git
$ cd clep
$ python3 -m pip install -e .
```

1.1.4 Dependency

- Python 3.6+
- Installation of R

Mandatory

- Numpy
- Scipy
- Pandas
- Matplotlib
- rpy2 (for limma)
- Limma package from bioconductor

For API information to use this library, see the *Developmental Guide*.

1.1.5 Issues

If you have difficulties using CLEP, please open an issue at our GitHub repository.

1.1.6 Acknowledgements

Graphics

The CLEP logo and framework graphic was designed by Carina Steinborn.

1.1.7 Disclaimer

CLEP is a scientific software that has been developed in an academic capacity, and thus comes with no warranty or guarantee of maintenance, support, or back-up of data.

1.2 How to use CLEP

1.2.1 Sample Scoring

There are 4 main way to score the patient-feature pairs,

1. Linear model fitting using **Limma**
2. ssGSEA
3. Z-Score
4. Radical Searching (eCDF based)

To carry out sample scoring use,

```
$ clep sample-scoring radical-search --data <DATA_FILE> --design <DESIGN_FILE> \
--control Control --threshold 2.5 --control_based --ret_summary --out <OUTPUT_DIR>
```

Data Format

The format of a standard data file should look like,

	Sample_1	Sample_2	Sample_3
HGNC_ID_1	0.354	2.568	1.564
HGNC_ID_2	1.255	1.232	0.26452
HGNC_ID_3	3.256	1.5	1.5462

The format of a design file, for the data given above should look like,

FileName	Target
Sample_1	Abnormal
Sample_2	Abnormal
Sample_3	Control

1.2.2 Knowledge Graph Generation

A patient-feature knowledge graph (KG) can be generated using 3 methods,

1. Based on **pathway overlaps** (needs ssGSEA as the scoring functions)
2. Based on user-provided knowledge graph
3. Based on the overlap of multiple user-provided knowledge graph (needs the use of either ssGSEA, if each KG represents a distinct pathway, or any other appropriate 3rd party scoring function)

To carry out KG generation use,

```
$ clep embedding generate-network --data <SCORED_DATA_FILE> --method interaction_
↪network \
--ret_summary --out <OUTPUT_DIR>
```

Data Format

The format of a knowledge graph file for the data given above should be a modified version of edgelist, as shown below,

Source	Relation	Target
HGNC_ID_1	association	HGNC_ID_2
HGNC_ID_2	decreases	HGNC_ID_3
HGNC_ID_3	increases	HGNC_ID_1

1.2.3 Knowledge Graph Embedding

For the generation of an embedding use,

```
$ clep embedding kge --data <NETWORK_FILE> --design <DESIGN_FILE> \
--model_config <MODEL_CONFIG.json> --train_size 0.8 --validation_size 0.1 --out
↪<OUTPUT_DIR>
```

Data Format

The config file for the KGE model must contain the model name, and other optimization parameters, as shown in the template below,

```
{
  "model": "RotatE",
  "model_kwargs": {
    "automatic_memory_optimization": true
  },
  "model_kwargs_ranges": {
    "embedding_dim": {
      "type": "int",
      "low": 6,
      "high": 9,
      "scale": "power_two"
    }
  },
  "training_loop": "slcwa",
  "optimizer": "adam",
  "optimizer_kwargs": {
    "weight_decay": 0.0
  },
  "optimizer_kwargs_ranges": {
    "lr": {
      "type": "float",
      "low": 0.0001,
      "high": 1.0,
      "scale": "log"
    }
  },
  "loss_function": "NSSALoss",
  "loss_kwargs": {},
  "loss_kwargs_ranges": {
    "margin": {
      "type": "float",
      "low": 1,
      "high": 30,
      "q": 2.0
    },
    "adversarial_temperature": {
      "type": "float",
      "low": 0.1,
      "high": 1.0,
      "q": 0.1
    }
  },
  "regularizer": "NoRegularizer",
```

(continues on next page)

(continued from previous page)


```
  "regularizer_kwargs": {},
  "regularizer_kwargs_ranges": {},
  "negative_sampler": "BasicNegativeSampler",
  "negative_sampler_kwargs": {},
  "negative_sampler_kwargs_ranges": {
    "num_negs_per_pos": {
      "type": "int",
      "low": 1,
      "high": 50,
      "q": 1
    }
  },
  "create_inverse_triples": false,
  "evaluator": "RankBasedEvaluator",
  "evaluator_kwargs": {
    "filtered": true
  },
  "evaluation_kwargs": {
    "batch_size": null
  },
  "training_kwargs": {
    "num_epochs": 1000,
    "label_smoothing": 0.0
  },
  "training_kwargs_ranges": {
    "batch_size": {
      "type": "int",
      "low": 8,
      "high": 11,
      "scale": "power_two"
    }
  },
  "stopper": "early",
  "stopper_kwargs": {
    "frequency": 25,
    "patience": 4,
    "delta": 0.002
  },
  "n_trials": 100,
  "timeout": 129600,
  "metric": "hits@10",
  "direction": "maximize",
  "sampler": "random",
  "pruner": "nop"
}
```


For more details on the configuration, check out PyKEEN

1.2.4 Classification

The classification of any provided data, can be carried out using any of the 5 different machine learning models,

1. **Logistic regression** with l2 regularization
2. Logistic regression with **elastic net** regularization
3. Support Vector Machines
4. Random forest
5. Gradient boosting

The classification also requires the input of the following optimizers,

1. Grid search
2. Random search
3. Bayesian search

For the carrying out the classification use,

```
$ clep classify --data <EMBEDDING_FILE> --model elastic_net --optimizer grid_search \
--out <OUTPUT_DIR>
```

Data Format

The format of the input file for classification should look like,

	Component_1	Component_2	Component_3	label
Sample_1	0.48687	-1.5675	1.74140	0
Sample_2	-1.48840	5.26354	-0.4435	1
Sample_3	-0.41461	4.6261	8.104	0

For more information on the command line interface, please refer *Command Line Interface*.

1.2.5 Programmatic Access

CLEP implements an API through which developers can utilise each module available in the CLEP framework. An example for the usage of the API functions in shown below.

```
import os
import pandas as pd
from clep.classification import do_classification

model = "elastic_net" # Classification Model
optimizer = "grid_search" # Optimization function for the classification model
out = os.getcwd() # Output directory
cv = 10 # Number of cross-validation folds
metrics = ['roc_auc', 'accuracy', 'f1_micro', 'f1_macro', 'f1'] # Metrics to be␣
↪analysed in cross-validation
randomize = False # If the labels in the data must be permuted

data_df = pd.read_table(data, index_col=0)

results = do_classification(data_df, model, optimizer, out, cv, metrics, randomize)
```

For more information on the available API functions, please refer *Developmental Guide*.

1.3 Command Line Interface

CLEP commands.

1.3.1 clep

Run clep.

```
clep [OPTIONS] COMMAND [ARGS]...
```

classify

Perform machine-learning classification.

```
clep classify [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--out <out>
Required Path to the output folder

--model <model>
Required Choose a classification model

Options logistic_regression|elastic_net|svm|random_forest|gradient_boost

--optimizer <optimizer>
Required Optimizer used for classifier.

Options grid_search|random_search|bayesian_search

--cv <cv>
Number of cross validation steps

Default 5

-m, --metrics <metrics>
Metrics that should be tested during cross validation (comma separated)

Options explained_variance|r2|max_error|neg_median_absolute_error|neg_mean_absolute_error|neg_mean_squared_error|

--randomize
Randomize sample labels to test the stability of and effectiveness of the machine learning algorithm

embedding

List Vectorization methods available.

```
clep embedding [OPTIONS] COMMAND [ARGS]...
```

evaluate

Perform Evaluation of the Embeddings.

```
clep embedding evaluate [OPTIONS]
```

Options

--data <data>
Required Path to a set of binned files

--label <label>
Required Label for the set of binned files

generate-network

Generate Network for the given data.

```
clep embedding generate-network [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--out <out>
Required Path to the output folder

--method <method>
The method used to generate the network

Default interaction_network

Options pathway_overlaplinteraction_networklinteraction_network_overlap

--kg <kg>
Path to the Knowledge Graph file in tsv format if Interaction Network method is chosen

--gmt <gmt>
Path to the gmt file if Pathway Overlap method is chosen

--network_folder <network_folder>
Path to the folder containing all the knowledge graph files if Interaction Network Overlap method is chosen

--intersect_thr <intersect_thr>
Threshold to make edges in Pathway Overlap method

Default 0.1

-rs, --ret_summary
Flag to indicate if the edge summary for patients must be created.

Default False

--jaccard_thr <jaccard_thr>
Threshold to make edges in Interaction Network Overlap method

Default 0.1

kge

Perform knowledge graph embedding.

```
clep embedding kge [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--design <design>
Required Path to tab-separated experiment design file

--out <out>
Required Path to the output folder

--all_nodes
Use this tag to return all nodes (not just patients)

Default False

-m, --model_config <model_config>
Required The configuration file for the model used for knowledge graph embedding in JSON format

--train_size <train_size>
Size of the training data for the knowledge graph embedding model

Default 0.8

--validation_size <validation_size>
Size of the validation data for the knowledge graph embedding model

Default 0.1

sample-scoring

List Single Sample Scoring methods available.

```
clep sample-scoring [OPTIONS] COMMAND [ARGS]...
```

limma

Limma-based Single Sample Scoring

```
clep sample-scoring limma [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--design <design>
Required Path to tab-separated experiment design file

--out <out>
Required Path to the output folder

--alpha <alpha>
Family-wise error rate

Default 0.05

--method <method>
Method used for testing and adjustment of P-Values

Default fdr_bh

--control <control>
Annotated value for the control samples (must start with an alphabet)

Default Control

radical-search

Radical Searching based Single Sample Scoring

```
clep sample-scoring radical-search [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--design <design>
Required Path to tab-separated experiment design file

--out <out>
Required Path to the output folder

--control <control>
Annotated value for the control samples (must start with an alphabet)

Default Control

--threshold <threshold>
Percentage of samples considered as 'extreme' on either side of the distribution

Default 2.5

-rs, --ret_summary
Flag to indicate if the edge summary for patients must be created.

Default False

-cb, --control_based
Run Radical Searching where the scoring is based on the control population instead of entire dataset

ssgsea

ssGSEA based Single Sample Scoring

```
clep sample-scoring ssgsea [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--design <design>
Required Path to tab-separated experiment design file

--out <out>
Required Path to the output folder

--gs <gs>
Required Path to the .gmt geneset file

z-score

Z-Score based Single Sample Scoring

```
clep sample-scoring z-score [OPTIONS]
```

Options

--data <data>
Required Path to tab-separated gene expression data file

--design <design>
Required Path to tab-separated experiment design file

--out <out>
Required Path to the output folder

--control <control>
Annotated value for the control samples (must start with an alphabet)

Default Control

--threshold <threshold>
Threshold for choosing patients that are 'extreme' w.r.t. the controls. If the z_score of a gene is greater than this threshold the gene is either up or down regulated.

Default 2.0

1.4 Developmental Guide

1.4.1 Core Module APIs

Sample Scoring

clep.sample_scoring.limma.**do_limma**()
Perform data manipulation before limma based SS scoring.

Parameters

- **data** – Dataframe containing the gene expression values
- **design** – Dataframe containing the design table for the data
- **alpha** – Family-wise error rate
- **method** – Method used family-wise error correction
- **control** – label used for representing the control in the design table of the data

Returns Dataframe containing the Single Sample scores from limma

clep.sample_scoring.ssgsea.**do_ssgsea**()
Run single sample GSEA (ssGSEA) on filtered gene expression data set.

Parameters

- **filtered_expression_data** – filtered gene expression values for samples
- **gene_set** – .gmt file containing gene sets
- **output_dir** – output directory
- **processes** – Number of processes
- **max_size** – Maximum allowed number of genes from gene set also the data set
- **min_size** – Minimum allowed number of genes from gene set also the data set

Returns ssGSEA results in respective directory

clep.sample_scoring.z_score.**do_z_score**()
Carry out Z-Score based single sample DE analysis.

Parameters

- **data** – Dataframe containing the gene expression values
- **design** – Dataframe containing the design table for the data
- **control** – label used for representing the control in the design table of the data
- **threshold** – Threshold for choosing patients that are “extreme” w.r.t. the controls.

Returns Dataframe containing the Single Sample scores using Z_Scores

clep.sample_scoring.radical_search.**do_radical_search**()
Identify the samples with extreme feature values either based on the entire dataset or control population.

Parameters

- **data** – Dataframe containing the gene expression values
- **design** – Dataframe containing the design table for the data
- **threshold** – Threshold for choosing patients that are “extreme” w.r.t. the controls

- **control** – label used for representing the control in the design table of the data
- **control_based** – The scoring is based on the control population instead of entire dataset

Returns Dataframe containing the Single Sample scores using radical searching

KG Generation

clep.embedding.network_generator.**do_graph_gen**()
Generate patient-feature network given the data using a certain network generation method.

Parameters

- **data** – Dataframe containing the patient-feature scores
- **network_gen_method** – Method to generate the patient-feature network
- **gmt** – Optional field for the path to the gmt file containing the pathway data
- **intersection_threshold** – Threshold to make edges in Pathway Overlap method
- **kg_data** – Optional field for the knowledge graph in edgelist format stored in a pandas dataframe
- **folder_path** – Optional field for the path to a folder containing multiple knowledge graphs
- **jaccard_threshold** – Threshold to make edges in Interaction Network Overlap method
- **summary** – Flag to indicate if the summary of the patient-feature network must be returned

Returns Dataframe containing patient-feature network, and optionally the summary of the patient-feature network

KG Embedding

clep.embedding.kge.**_weighted_splitter**()
Split the given edgelist into training, validation and testing sets on the basis of the ratio of relations.

Parameters

- **edgelist** – Edgelist in the form of (Source, Relation, Target)
- **train_size** – Size of the training data
- **validation_size** – Size of the training data

Returns Tuple containing the train, validation & test splits

clep.embedding.kge.**do_kge**()
Carry out KGE on the given data.

Parameters

- **edgelist** – Dataframe containing the patient-feature graph in edgelist format
- **design** – Dataframe containing the design table for the data
- **out** – Output folder for the results
- **model_config** – Configuration file for the KGE models, in JSON format.
- **return_patients** – Flag to indicate if the final data should contain only patients or even the features

- **train_size** – Size of the training data for KGE ranging from 0 - 1
- **validation_size** – Size of the validation data for KGE ranging from 0 - 1. It must be lower than training size

Returns Dataframe containing the embedding from the KGE

Classification

clep.classification.classify.**do_classification**()

Perform classification on embeddings generated from previous step.

Parameters

- **data** – Dataframe containing the embeddings
- **model_name** – model that should be used for cross validation
- **optimizer_name** – Optimizer used to optimize the classification
- **out_dir** – Path to the output directory
- **validation_cv** – Number of cross validation steps
- **scoring_metrics** – Scoring metrics tested during cross validation
- **rand_labels** – Boolean variable to indicate if labels must be randomized to check for ML stability
- **args** – Custom arguments to the estimator model

Returns Dictionary containing the cross validation results

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

Symbols

C

D

M