torch$_{k}means$Documentation

Release latest

Jonas K. Falkner

Dec 04, 2022

CONTENTS

This is the documentation of **torch_kmeans**.

CHAPTER

ONE

TORCH_KMEANS

PyTorch implementations of KMeans, Soft-KMeans and Constrained-KMeans

torch_kmeans features implementations of the well known k-means algorithm as well as its soft and constrained variants.

All algorithms are completely implemented as PyTorch modules and can be easily incorporated in a PyTorch pipeline or model. Therefore, they support execution on GPU as well as working on (mini-)batches of data. Moreover, they also provide a scikit-learn style interface featuring

```
model.fit(), model.predict() and model.fit_predict()
```

functions.

-> view official github

1.1 Highlights

- Fully implemented in PyTorch.
- GPU support like native PyTorch.
- PyTorch script JIT compiled for most performance sensitive parts.
- **Works with mini-batches of samples:**
  - each instance can have a different number of clusters.
- **Constrained Kmeans works with cluster constraints like:**
  - a max number of samples per cluster or,
  - a maximum weight per cluster, where each sample has an associated weight.
- SoftKMeans is a fully differentiable clustering procedure and can readily be used in a PyTorch neural network model which requires backpropagation.
- Unit tested against the scikit-learn KMeans implementation.
- GPU execution enables very fast computation even for large batch size or very high dimensional feature spaces (see speed comparison)

1.2 Installation

Simply install from PyPI

```
pip install torch-kmeans
```

1.3 Contents

1.3.1 torch_kmeans

PyTorch implementations of KMeans, Soft-KMeans and Constrained-KMeans

torch_kmeans features implementations of the well known k-means algorithm as well as its soft and constrained variants.

All algorithms are completely implemented as PyTorch modules and can be easily incorporated in a PyTorch pipeline or model. Therefore, they support execution on GPU as well as working on (mini-)batches of data. Moreover, they also provide a scikit-learn style interface featuring

```
model.fit(), model.predict() and model.fit_predict()
```

functions.

-> view official documentation

Highlights

- Fully implemented in PyTorch. (PyTorch and Numpy are the only package dependencies!)
- GPU support like native PyTorch.
- PyTorch script JIT compiled for most performance sensitive parts.
- **Works with mini-batches of samples:**
  - each instance can have a different number of clusters.
- **Constrained Kmeans works with cluster constraints like:**

  - a max number of samples per cluster or,
  - a maximum weight per cluster, where each sample has an associated weight.

- SoftKMeans is a fully differentiable clustering procedure and can readily be used in a PyTorch neural network model which requires backpropagation.
- Unit tested against the scikit-learn KMeans implementation.
- GPU execution enables very fast computation even for large batch size or very high dimensional feature spaces (see speed comparison)

Installation

Simply install from PyPI

```
pip install torch-kmeans
```

Usage

Pytorch style usage

```
import torch
from torch_kmeans import KMeans

model = KMeans(n_clusters=4)

x = torch.randn((4, 20, 2))   # (BS, N, D)
result = model(x)
print(result.labels)
```

Scikit-learn style usage

```
import torch
from torch_kmeans import KMeans

model = KMeans(n_clusters=4)

x = torch.randn((4, 20, 2))   # (BS, N, D)
model = model.fit(x)
labels = model.predict(x)
print(labels)
```

or

```
import torch
from torch_kmeans import KMeans

model = KMeans(n_clusters=4)

x = torch.randn((4, 20, 2))   # (BS, N, D)
labels = model.fit_predict(x)
print(labels)
```

Examples

You can find more examples and usage in the detailed example notebooks.

1.3.2 License

The MIT License (MIT)

Copyright (c) 2022 Jonas K. Falkner

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.3.3 Contributors

- Jonas K. Falkner <jokofa@gmail.com>

1.3.4 torch_kmeans

torch_kmeans package

class torch_kmeans.**KMeans**(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *normalize: ~typing.Optional[~typing.Union[str, bool]] = None*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, ***kwargs*)

Bases: `Module`

Implements k-means clustering in terms of pytorch tensor operations which can be run on GPU. Supports batches of instances for use in batched training (e.g. for neural networks).

Partly based on ideas from:

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://github.com/overshiki/kmeans_pytorch

Parameters

- **init_method** (*str*) – Method to initialize cluster centers ['rnd', 'k-means++'] (default: 'rnd')

- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **normalize** (*Optional[Union[str, bool]]*) – String id of method to use to normalize input. one of [‘mean’, ‘minmax’, ‘unit’]. None to disable normalization. (default: None).
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++']

NORM_METHODS = ['mean', 'minmax', 'unit']

property is_fitted: bool

True if model was already fitted.

property num_clusters: Union[int, Tensor, Any]

Number of clusters in fitted model. Returns a tensor with possibly different numbers of clusters per instance for whole batch.

forward(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → *ClusterResult*

torch.nn like forward pass.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
ClusterResult tuple

Return type
ClusterResult

fit(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → Module

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
KMeans model

Return type
Module

predict(*x: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- ****kwargs** – additional kwargs for assignment procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

fit_predict(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → LongTensor

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

training: bool

class torch_kmeans.**ConstrainedKMeans**(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *n_priority_trials_before_fall_back: int = 5*, *raise_infeasible: bool = True*, ***kwargs*)

Bases: *KMeans*

Implements constrained k-means clustering. Priority implementation is based on the method of

Paper:
Geetha, S., G. Poonthalir, and P. T. Vanathi. "Improved k-means algorithm for capacitated clustering problem." INFOCOMP Journal of Computer Science 8.4 (2009)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: ['rnd', 'topk', 'k-means++', 'ckm++'] (default: 'rnd')
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: 123).
- **n_priority_trials_before_fall_back** (*int*) – Number of trials trying to assign samples to constrained clusters based on priority values before falling back to assigning the node with the highest weight to a cluster which can still accommodate it or the dummy cluster otherwise. (default: 5)
- **raise_infeasible** (*bool*) – if set to False, will only display a warning instead of raising an error (default: True)
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++', 'topk', 'ckm++']

NORM_METHODS = []

predict(*x: Tensor*, *weights: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **weights** (*Tensor*) – normalized weight for each sample (BS, N)
- ****kwargs** – additional kwargs for assignment procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

training: bool

class torch_kmeans.**SoftKMeans**(*init_method: str = 'rnd'*, *num_init: int = 1*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.CosineSimilarity'>*, *p_norm: int = 1*, *normalize: str = 'unit'*, *tol: float = 1e-05*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *temp: float = 5.0*, ***kwargs*)

Bases: *KMeans*

Implements differentiable soft k-means clustering. Method adapted from https://github.com/bwilder0/clusternet to support batches.

Paper:
Wilder et al., "End to End Learning and Optimization on Graphs" (NeurIPS'2019)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: ['rnd', 'topk'] (default: 'rnd')
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers. If >1 selects the best configuration before propagating through fixpoint (default: 1).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: CosineSimilarity).
- **p_norm** (*int*) – norm for lp distance (default: 1).
- **normalize** (*str*) – id of method to use to normalize input. (default: 'unit').
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- **temp** (*float*) – temperature for soft cluster assignments (default: 5.0).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class torch_kmeans.**LpDistance**(***kwargs*)

Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Optional[Tensor] = None*) → Tensor

Compute the batched p-norm distance between each pair of the two collections of row vectors.

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Optional[Tensor]*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Computes the pairwise distance between vectors v1, v2 using the p-norm

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.**DotProductSimilarity**(***kwargs*)

Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.**CosineSimilarity**(***kwargs*)

Bases: *DotProductSimilarity*

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class torch_kmeans.**ClusterResult**(*labels: LongTensor*, *centers: Tensor*, *inertia: Tensor*, *x_org: Tensor*, *x_norm: Tensor*, *k: LongTensor*, *soft_assignment: Optional[Tensor] = None*)

Bases: tuple

Named and typed result tuple for kmeans algorithms

Parameters

- **labels** (*LongTensor*) – label for each sample in x
- **centers** (*Tensor*) – corresponding coordinates of cluster centers

- **inertia** (*Tensor*) – sum of squared distances of samples to their closest cluster center
- **x_org** (*Tensor*) – original x
- **x_norm** (*Tensor*) – normalized x which was used for cluster centers and labels
- **k** (*LongTensor*) – number of clusters
- **soft_assignment** (*Optional[Tensor]*) – assignment probabilities of soft kmeans

Create new instance of ClusterResult(labels, centers, inertia, x_org, x_norm, k, soft_assignment)

labels: LongTensor
Alias for field number 0

centers: Tensor
Alias for field number 1

inertia: Tensor
Alias for field number 2

x_org: Tensor
Alias for field number 3

x_norm: Tensor
Alias for field number 4

k: LongTensor
Alias for field number 5

soft_assignment: Optional[Tensor]
Alias for field number 6

Subpackages

torch_kmeans.clustering package

class torch_kmeans.clustering.**ConstrainedKMeans**(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *n_priority_trials_before_fall_back: int = 5*, *raise_infeasible: bool = True*, ***kwargs*)

Bases: *KMeans*

Implements constrained k-means clustering. Priority implementation is based on the method of

Paper:
Geetha, S., G. Poonthalir, and P. T. Vanathi. "Improved k-means algorithm for capacitated clustering problem." INFOCOMP Journal of Computer Science 8.4 (2009)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: ['rnd', 'topk', 'k-means++', 'ckm++'] (default: 'rnd')

- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: 123).
- **n_priority_trials_before_fall_back** (*int*) – Number of trials trying to assign samples to constrained clusters based on priority values before falling back to assigning the node with the highest weight to a cluster which can still accommodate it or the dummy cluster otherwise. (default: 5)
- **raise_infeasible** (*bool*) – if set to False, will only display a warning instead of raising an error (default: True)
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++', 'topk', 'ckm++']

NORM_METHODS = []

predict(*x: Tensor*, *weights: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **weights** (*Tensor*) – normalized weight for each sample (BS, N)
- ****kwargs** – additional kwargs for assignment procedure

Returns

batch tensor of cluster labels for each sample (BS, N)

Return type

LongTensor

training: bool

class torch_kmeans.clustering.**KMeans**(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *normalize: ~typing.Optional[~typing.Union[str, bool]] = None*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, ***kwargs*)

Bases: Module

Implements k-means clustering in terms of pytorch tensor operations which can be run on GPU. Supports batches of instances for use in batched training (e.g. for neural networks).

Partly based on ideas from:

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://github.com/overshiki/kmeans_pytorch

Parameters

- **init_method** (*str*) – Method to initialize cluster centers [‘rnd’, ‘k-means++’] (default: ‘rnd’)
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **normalize** (*Optional[Union[str, bool]]*) – String id of method to use to normalize input. one of [‘mean’, ‘minmax’, ‘unit’]. None to disable normalization. (default: None).
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++']

NORM_METHODS = ['mean', 'minmax', 'unit']

property is_fitted: bool

True if model was already fitted.

property num_clusters: Union[int, Tensor, Any]

Number of clusters in fitted model. Returns a tensor with possibly different numbers of clusters per instance for whole batch.

forward(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → *ClusterResult*

torch.nn like forward pass.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)

- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
ClusterResult tuple

Return type
ClusterResult

fit(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → Module

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
KMeans model

Return type
Module

predict(*x: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- ****kwargs** – additional kwargs for assignment procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

fit_predict(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → LongTensor

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

training: bool

class torch_kmeans.clustering.SoftKMeans(*init_method: str = 'rnd'*, *num_init: int = 1*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.CosineSimilarity'>*, *p_norm: int = 1*, *normalize: str = 'unit'*, *tol: float = 1e-05*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *temp: float = 5.0*, ***kwargs*)

Bases: *KMeans*

Implements differentiable soft k-means clustering. Method adapted from https://github.com/bwilder0/clusternet to support batches.

Paper:
Wilder et al., "End to End Learning and Optimization on Graphs" (NeurIPS'2019)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: ['rnd', 'topk'] (default: 'rnd')
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers. If >1 selects the best configuration before propagating through fixpoint (default: 1).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: CosineSimilarity).
- **p_norm** (*int*) – norm for lp distance (default: 1).
- **normalize** (*str*) – id of method to use to normalize input. (default: 'unit').
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- **temp** (*float*) – temperature for soft cluster assignments (default: 5.0).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class torch_kmeans.clustering.KNN(*k: int*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *normalize: ~typing.Optional[~typing.Union[str, bool]] = None*, ***kwargs*)

Bases: Module

Implements k nearest neighbors in terms of pytorch tensor operations which can be run on GPU. Supports mini-batches of instances.

Parameters

- **k** (*int*) – number of neighbors to consider
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **normalize** (*Optional[Union[str, bool]]*) – String id of method to use to normalize input. one of ['mean', 'minmax', 'unit']. None to disable normalization. (default: None).

Initializes internal Module state, shared by both nn.Module and ScriptModule.

NORM_METHODS = ['mean', 'minmax', 'unit']

forward(*x: Tensor*, *k: Optional[int] = None*, *same_source: bool = True*) → KNeighbors

torch.nn like forward pass.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[int]*) – optional number of neighbors to use
- **same_source** (*bool*) – flag if each sample itself should be included as its own neighbor (default: True)

Returns
KNeighbors tuple

Return type
KNeighbors

fit(*x: Tensor*, *k: Optional[int] = None*, ***kwargs*) → KNeighbors

Compute k nearest neighbors for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[int]*) – optional number of neighbors to use
- ****kwargs** – additional kwargs for fitting procedure

Returns
KNeighbors tuple

Return type
KNeighbors

training: bool

Submodules

torch_kmeans.clustering.constr_kmeans module

class torch_kmeans.clustering.constr_kmeans.**ConstrainedKMeans**(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *n_priority_trials_before_fall_back: int = 5*, *raise_infeasible: bool = True*, ***kwargs*)

Bases: *KMeans*

Implements constrained k-means clustering. Priority implementation is based on the method of

Paper:

Geetha, S., G. Poonthalir, and P. T. Vanathi. "Improved k-means algorithm for capacitated clustering problem." INFOCOMP Journal of Computer Science 8.4 (2009)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: ['rnd', 'topk', 'k-means++', 'ckm++'] (default: 'rnd')
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: 123).
- **n_priority_trials_before_fall_back** (*int*) – Number of trials trying to assign samples to constrained clusters based on priority values before falling back to assigning the node with the highest weight to a cluster which can still accommodate it or the dummy cluster otherwise. (default: 5)
- **raise_infeasible** (*bool*) – if set to False, will only display a warning instead of raising an error (default: True)
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++', 'topk', 'ckm++']

NORM_METHODS = []

predict(*x: Tensor*, *weights: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **weights** (*Tensor*) – normalized weight for each sample (BS, N)
- ****kwargs** – additional kwargs for assignment procedure

Returns

batch tensor of cluster labels for each sample (BS, N)

Return type

LongTensor

training: bool

torch_kmeans.clustering.kmeans module

class torch_kmeans.clustering.kmeans.KMeans(*init_method: str = 'rnd'*, *num_init: int = 8*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *tol: float = 0.0001*, *normalize: ~typing.Optional[~typing.Union[str, bool]] = None*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, ***kwargs*)

Bases: Module

Implements k-means clustering in terms of pytorch tensor operations which can be run on GPU. Supports batches of instances for use in batched training (e.g. for neural networks).

Partly based on ideas from:

- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://github.com/overshiki/kmeans_pytorch

Parameters

- **init_method** (*str*) – Method to initialize cluster centers [‘rnd’, ‘k-means++’] (default: ‘rnd’)
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers (default: 8).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **normalize** (*Optional[Union[str, bool]]*) – String id of method to use to normalize input. one of [‘mean’, ‘minmax’, ‘unit’]. None to disable normalization. (default: None).

- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

INIT_METHODS = ['rnd', 'k-means++']

NORM_METHODS = ['mean', 'minmax', 'unit']

property is_fitted: bool

True if model was already fitted.

property num_clusters: Union[int, Tensor, Any]

Number of clusters in fitted model. Returns a tensor with possibly different numbers of clusters per instance for whole batch.

forward(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → *ClusterResult*

torch.nn like forward pass.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
ClusterResult tuple

Return type
ClusterResult

fit(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → Module

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
KMeans model

Return type
Module

predict(*x: Tensor*, ***kwargs*) → LongTensor

Predict the closest cluster each sample in X belongs to.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- ****kwargs** – additional kwargs for assignment procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

fit_predict(*x: Tensor*, *k: Optional[Union[LongTensor, Tensor, int]] = None*, *centers: Optional[Tensor] = None*, ***kwargs*) → LongTensor

Compute cluster centers and predict cluster index for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[Union[LongTensor, Tensor, int]]*) – optional batch of (possibly different) numbers of clusters per instance (BS,)
- **centers** (*Optional[Tensor]*) – optional batch of initial centers to use (BS, K, D)
- ****kwargs** – additional kwargs for initialization or cluster procedure

Returns
batch tensor of cluster labels for each sample (BS, N)

Return type
LongTensor

training: bool

torch_kmeans.clustering.knn module

class torch_kmeans.clustering.knn.**KNN**(*k: int*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.LpDistance'>*, *p_norm: int = 2*, *normalize: ~typing.Optional[~typing.Union[str, bool]] = None*, ***kwargs*)

Bases: Module

Implements k nearest neighbors in terms of pytorch tensor operations which can be run on GPU. Supports mini-batches of instances.

Parameters

- **k** (*int*) – number of neighbors to consider
- **distance** (*BaseDistance*) – batched distance evaluator (default: LpDistance).
- **p_norm** (*int*) – norm for lp distance (default: 2).
- **normalize** (*Optional[Union[str, bool]]*) – String id of method to use to normalize input. one of [‘mean’, ‘minmax’, ‘unit’]. None to disable normalization. (default: None).

Initializes internal Module state, shared by both nn.Module and ScriptModule.

NORM_METHODS = ['mean', 'minmax', 'unit']

forward(*x: Tensor*, *k: Optional[int] = None*, *same_source: bool = True*) → KNeighbors

torch.nn like forward pass.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[int]*) – optional number of neighbors to use
- **same_source** (*bool*) – flag if each sample itself should be included as its own neighbor (default: True)

Returns
KNeighbors tuple

Return type
KNeighbors

fit(*x: Tensor*, *k: Optional[int] = None*, ***kwargs*) → KNeighbors

Compute k nearest neighbors for each sample.

Parameters

- **x** (*Tensor*) – input features/coordinates (BS, N, D)
- **k** (*Optional[int]*) – optional number of neighbors to use
- ****kwargs** – additional kwargs for fitting procedure

Returns
KNeighbors tuple

Return type
KNeighbors

training: bool

torch_kmeans.clustering.soft_kmeans module

class torch_kmeans.clustering.soft_kmeans.**SoftKMeans**(*init_method: str = 'rnd'*, *num_init: int = 1*, *max_iter: int = 100*, *distance: ~torch_kmeans.utils.distances.BaseDistance = <class 'torch_kmeans.utils.distances.CosineSimilarity'>*, *p_norm: int = 1*, *normalize: str = 'unit'*, *tol: float = 1e-05*, *n_clusters: ~typing.Optional[int] = 8*, *verbose: bool = True*, *seed: ~typing.Optional[int] = 123*, *temp: float = 5.0*, ***kwargs*)

Bases: *KMeans*

Implements differentiable soft k-means clustering. Method adapted from https://github.com/bwilder0/clusternet to support batches.

Paper:
Wilder et al., "End to End Learning and Optimization on Graphs" (NeurIPS'2019)

Parameters

- **init_method** (*str*) – Method to initialize cluster centers: [‘rnd’, ‘topk’] (default: ‘rnd’)
- **num_init** (*int*) – Number of different initial starting configurations, i.e. different sets of initial centers. If >1 selects the best configuration before propagating through fixpoint (default: 1).
- **max_iter** (*int*) – Maximum number of iterations (default: 100).
- **distance** (*BaseDistance*) – batched distance evaluator (default: CosineSimilarity).
- **p_norm** (*int*) – norm for lp distance (default: 1).
- **normalize** (*str*) – id of method to use to normalize input. (default: ‘unit’).
- **tol** (*float*) – Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence. (default: 1e-4)
- **n_clusters** (*Optional[int]*) – Default number of clusters to use if not provided in call (optional, default: 8).
- **verbose** (*bool*) – Verbosity flag to print additional info (default: True).
- **seed** (*Optional[int]*) – Seed to fix random state for randomized center inits (default: True).
- **temp** (*float*) – temperature for soft cluster assignments (default: 5.0).
- ****kwargs** – additional key word arguments for the distance function.

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

torch_kmeans.utils package

class torch_kmeans.utils.**LpDistance**(***kwargs*)

Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Optional[Tensor] = None*) → Tensor

Compute the batched p-norm distance between each pair of the two collections of row vectors.

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Optional[Tensor]*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Computes the pairwise distance between vectors v1, v2 using the p-norm

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.utils.DotProductSimilarity(**kwargs)

Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.utils.CosineSimilarity(**kwargs)

Bases: *DotProductSimilarity*

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class torch_kmeans.utils.ClusterResult(*labels: LongTensor*, *centers: Tensor*, *inertia: Tensor*, *x_org: Tensor*, *x_norm: Tensor*, *k: LongTensor*, *soft_assignment: Optional[Tensor] = None*)

Bases: tuple

Named and typed result tuple for kmeans algorithms

Parameters

- **labels** (*LongTensor*) – label for each sample in x
- **centers** (*Tensor*) – corresponding coordinates of cluster centers
- **inertia** (*Tensor*) – sum of squared distances of samples to their closest cluster center
- **x_org** (*Tensor*) – original x
- **x_norm** (*Tensor*) – normalized x which was used for cluster centers and labels
- **k** (*LongTensor*) – number of clusters
- **soft_assignment** (*Optional[Tensor]*) – assignment probabilities of soft kmeans

Create new instance of ClusterResult(labels, centers, inertia, x_org, x_norm, k, soft_assignment)

labels: LongTensor

Alias for field number 0

centers: Tensor
Alias for field number 1

inertia: Tensor
Alias for field number 2

x_org: Tensor
Alias for field number 3

x_norm: Tensor
Alias for field number 4

k: LongTensor
Alias for field number 5

soft_assignment: Optional[Tensor]
Alias for field number 6

Submodules

torch_kmeans.utils.distances module

class torch_kmeans.utils.distances.**LpDistance**(***kwargs*)
Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Optional[Tensor] = None*) → Tensor
Compute the batched p-norm distance between each pair of the two collections of row vectors.

Parameters
- **query_emb** (*Tensor*) –
- **ref_emb** (*Optional[Tensor]*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor
Computes the pairwise distance between vectors v1, v2 using the p-norm

Parameters
- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.utils.distances.**DotProductSimilarity**(***kwargs*)
Bases: BaseDistance

Initializes internal Module state, shared by both nn.Module and ScriptModule.

compute_mat(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

pairwise_distance(*query_emb: Tensor*, *ref_emb: Tensor*) → Tensor

Parameters

- **query_emb** (*Tensor*) –
- **ref_emb** (*Tensor*) –

Return type
Tensor

training: bool

class torch_kmeans.utils.distances.**CosineSimilarity**(***kwargs*)

Bases: *DotProductSimilarity*

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

torch_kmeans.utils.utils module

class torch_kmeans.utils.utils.**ClusterResult**(*labels: LongTensor*, *centers: Tensor*, *inertia: Tensor*, *x_org: Tensor*, *x_norm: Tensor*, *k: LongTensor*, *soft_assignment: Optional[Tensor] = None*)

Bases: tuple

Named and typed result tuple for kmeans algorithms

Parameters

- **labels** (*LongTensor*) – label for each sample in x
- **centers** (*Tensor*) – corresponding coordinates of cluster centers
- **inertia** (*Tensor*) – sum of squared distances of samples to their closest cluster center
- **x_org** (*Tensor*) – original x
- **x_norm** (*Tensor*) – normalized x which was used for cluster centers and labels
- **k** (*LongTensor*) – number of clusters
- **soft_assignment** (*Optional[Tensor]*) – assignment probabilities of soft kmeans

Create new instance of ClusterResult(labels, centers, inertia, x_org, x_norm, k, soft_assignment)

labels: LongTensor

Alias for field number 0

centers: Tensor

Alias for field number 1

inertia: Tensor
Alias for field number 2

x_org: Tensor
Alias for field number 3

x_norm: Tensor
Alias for field number 4

k: LongTensor
Alias for field number 5

soft_assignment: Optional[Tensor]
Alias for field number 6

torch_kmeans.utils.utils.**rm_kwargs**(*kwargs: Dict*, *keys: List*)
Remove items corresponding to keys specified in 'keys' from kwargs dict.

Parameters

- **kwargs** (*Dict*) –
- **keys** (*List*) –

1.4 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

L

M

N

P

R

S

T

X