Dimensionality Reduction

Contents

Algorithms for decomposition and dimensionality reduction.

class **anomalib.models.components.dimensionality_reduction.PCA(*n_components*)**

Bases: DynamicBufferMixin

Principle Component Analysis (PCA).

Parameters:

n_components (*float*) – Number of components. Can be either integer number of components or a ratio between 0-1.

Example

```
>>> import torch
>>> from anomalib.models.components import PCA
```

Create a PCA model with 2 components:

```
>>> pca = PCA(n_components=2)
```

Create a random embedding and fit a PCA model.

```
>>> embedding = torch.rand(1000, 5).cuda()
>>> pca = PCA(n_components=2)
>>> pca.fit(embedding)
```

Apply transformation:

```
>>> transformed = pca.transform(embedding)
>>> transformed.shape
torch.Size([1000, 2])
```

fit(*dataset*)

Fits the PCA model to the dataset.

Parameters:

dataset (*torch.Tensor*) – Input dataset to fit the model.

Return type:

`None`

Example

```
>>> pca.fit(embedding)
>>> pca.singular_vectors
tensor([9.6053, 9.2763], device='cuda:0')
```

```
>>> pca.mean
tensor([0.4859, 0.4959, 0.4906, 0.5010, 0.5042], device='cuda:0')
```

fit_transform(*dataset*)

Fit and transform PCA to dataset.

Parameters:

dataset (*torch.Tensor*) – Dataset to which the PCA if fit and transformed

Return type:

`Tensor`

Returns:

Transformed dataset

Example

```
>>> pca.fit_transform(embedding)
>>> transformed_embedding = pca.fit_transform(embedding)
>>> transformed_embedding.shape
torch.Size([1000, 2])
```

forward(*features*)

Transform the features.

Parameters:

features (*torch.Tensor*) – Input features

Return type:

Tensor

Returns:

Transformed features

Example

```
>>> pca(embedding).shape
torch.Size([1000, 2])
```

inverse_transform(*features*)

Inverses the transformed features.

Parameters:

features (*torch.Tensor*) – Transformed features

Return type:

Tensor

Returns:

Inverse features

Example

```
>>> inverse_embedding = pca.inverse_transform(transformed_embedding)
>>> inverse_embedding.shape
torch.Size([1000, 5])
```

transform(*features*)

Transform the features based on singular vectors calculated earlier.

Parameters:

features (*torch.Tensor*) – Input features

Return type:

`Tensor`

Returns:

Transformed features

Example

```
>>> pca.transform(embedding)
>>> transformed_embedding = pca.transform(embedding)
```

```
>>> embedding.shape
torch.Size([1000, 5])
#
>>> transformed_embedding.shape
torch.Size([1000, 2])
```

class **anomalib.models.components.dimensionality_reduction.SparseRandomProjection(*eps=0.1*, *random_state=None*)**

Bases: `object`

Sparse Random Projection using PyTorch operations.

Parameters:

- **eps** (*float, optional*) – Minimum distortion rate parameter for calculating Johnson-Lindenstrauss minimum dimensions. Defaults to `0.1`.
- **random_state** (*int | None, optional*) – Uses the seed to set the random state for sample_without_replacement function. Defaults to `None`.

Example

To fit and transform the embedding tensor, use the following code:

```
import torch
from anomalib.models.components import SparseRandomProjection

sparse_embedding = torch.rand(1000, 5).cuda()
model = SparseRandomProjection(eps=0.1)
```

Fit the model and transform the embedding tensor:

```
model.fit(sparse_embedding)
projected_embedding = model.transform(sparse_embedding)

print(projected_embedding.shape)
# Output: torch.Size([1000, 5920])
```

fit(*embedding*)

Generate sparse matrix from the embedding tensor.

Parameters:

embedding (*torch.Tensor*) – embedding tensor for generating embedding

Returns:

Return self to be used as

```
>>> model = SparseRandomProjection()
>>> model = model.fit()
```

Return type:

(SparseRandomProjection)

transform(*embedding*)

Project the data by using matrix product with the random matrix.

Parameters:

embedding (*torch.Tensor*) – Embedding of shape (n_samples, n_features) The input data to project into a smaller dimensional space

Returns:

Sparse matrix of shape

(n_samples, n_components) Projected array.

Return type:

projected_embedding (torch.Tensor)

Example

```
>>> projected_embedding = model.transform(embedding)
>>> projected_embedding.shape
torch.Size([1000, 5920])
```

Previous
Feature Extractors

Next
Normalizing Flows