[![Logo](../../_static/logo.png)](../../index.html)

Getting Started

* [Installation](../installation.html)
  * [Install with pip](../installation.html#install-with-pip)
  * [Install with Conda](../installation.html#install-with-conda)
  * [Install from Source](../installation.html#install-from-source)
  * [Editable Install](../installation.html#editable-install)
  * [Install PyTorch with CUDA support](../installation.html#install-pytorch-with-cuda-support)
* [Quickstart](../quickstart.html)
  * [Sentence Transformer](../quickstart.html#sentence-transformer)
  * [Cross Encoder](../quickstart.html#cross-encoder)
  * [Next Steps](../quickstart.html#next-steps)

Sentence Transformer

* [Usage](../sentence_transformer/usage/usage.html)
  * [Computing Embeddings](../../examples/applications/computing-embeddings/README.html)
    * [Initializing a Sentence Transformer Model](../../examples/applications/computing-embeddings/README.html#initializing-a-sentence-transformer-model)
    * [Calculating Embeddings](../../examples/applications/computing-embeddings/README.html#calculating-embeddings)
    * [Prompt Templates](../../examples/applications/computing-embeddings/README.html#prompt-templates)

* [Input Sequence Length](../../examples/applications/computing-embeddings/README.html#id1)
* [Multi-Process / Multi-GPU Encoding](../../examples/applications/computing-embeddings/README.html#multi-process-multi-gpu-encoding)
* [Semantic Textual Similarity](../sentence_transformer/usage/semantic_textual_similarity.html)
* [Similarity Calculation](../sentence_transformer/usage/semantic_textual_similarity.html#similarity-calculation)
* [Semantic Search](../../examples/applications/semantic-search/README.html)
* [Background](../../examples/applications/semantic-search/README.html#background)
* [Symmetric vs. Asymmetric Semantic Search](../../examples/applications/semantic-search/README.html#symmetric-vs-asymmetric-semantic-search)
* [Manual Implementation](../../examples/applications/semantic-search/README.html#manual-implementation)
* [Optimized Implementation](../../examples/applications/semantic-search/README.html#optimized-implementation)
* [Speed Optimization](../../examples/applications/semantic-search/README.html#speed-optimization)
* [Elasticsearch](../../examples/applications/semantic-search/README.html#elasticsearch)
* [Approximate Nearest Neighbor](../../examples/applications/semantic-search/README.html#approximate-nearest-neighbor)
* [Retrieve & Re-Rank](../../examples/applications/semantic-search/README.html#retrieve-re-rank)

* [Examples](../../examples/applications/semantic-search/README.html#examples)
* [Retrieve & Re-Rank](../../examples/applications/retrieve_rerank/README.html)
* [Retrieve & Re-Rank Pipeline](../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)
* [Retrieval: Bi-Encoder](../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)
* [Re-Ranker: Cross-Encoder](../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)
* [Example Scripts](../../examples/applications/retrieve_rerank/README.html#example-scripts)
* [Pre-trained Bi-Encoders (Retrieval)](../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrieval)
* [Pre-trained Cross-Encoders (Re-Ranker)](../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)
* [Clustering](../../examples/applications/clustering/README.html)
* [k-Means](../../examples/applications/clustering/README.html#k-means)
* [Agglomerative Clustering](../../examples/applications/clustering/README.html#agglomerative-clustering)
* [Fast Clustering](../../examples/applications/clustering/README.html#fast-clustering)
* [Topic Modeling](../../examples/applications/clustering/README.html#topic-modeling)
* [Paraphrase Mining](../../examples/applications/paraphrase-mining/README.html)
* [`paraphrase_mining()`](../../examples/applications/paraphrase-mining/README.html#sentence_transformers.util.paraphrase_mining)
* [Translated Sentence Mining](../../examples/applications/parallel-sentence-mining/README.html)

* [Margin Based Mining](../../examples/applications/parallel-sentence-mining/README.html#margin-based-mining)
    * [Examples](../../examples/applications/parallel-sentence-mining/README.html#examples)
  * [Image Search](../../examples/applications/image-search/README.html)
    * [Installation](../../examples/applications/image-search/README.html#installation)
    * [Usage](../../examples/applications/image-search/README.html#usage)
    * [Examples](../../examples/applications/image-search/README.html#examples)
  * [Embedding Quantization](../../examples/applications/embedding-quantization/README.html)
    * [Binary Quantization](../../examples/applications/embedding-quantization/README.html#binary-quantization)
    * [Scalar (int8) Quantization](../../examples/applications/embedding-quantization/README.html#scalar-int8-quantization)
    * [Additional extensions](../../examples/applications/embedding-quantization/README.html#additional-extensions)
    * [Demo](../../examples/applications/embedding-quantization/README.html#demo)
    * [Try it yourself](../../examples/applications/embedding-quantization/README.html#try-it-yourself)
  * [Speeding up Inference](../sentence_transformer/usage/efficiency.html)
    * [PyTorch](../sentence_transformer/usage/efficiency.html#pytorch)
    * [ONNX](../sentence_transformer/usage/efficiency.html#onnx)
    * [OpenVINO](../sentence_transformer/usage/efficiency.html#openvino)
    * [Benchmarks](../sentence_transformer/usage/efficiency.html#benchmarks)
  * [Creating Custom Models](../sentence_transformer/usage/custom_models.html)
    * [Structure of Sentence Transformer

Models](../sentence_transformer/usage/custom_models.html#structure-of-sentence-transformer-models)

* [Sentence Transformer Model from a Transformers Model](../sentence_transformer/usage/custom_models.html#sentence-transformer-model-from-a-transformers-model)

* [Pretrained Models](../sentence_transformer/pretrained_models.html)

* [Original Models](../sentence_transformer/pretrained_models.html#original-models)

* [Semantic Search Models](../sentence_transformer/pretrained_models.html#semantic-search-models)

* [Multi-QA Models](../sentence_transformer/pretrained_models.html#multi-qa-models)

* [MSMARCO Passage Models](../sentence_transformer/pretrained_models.html#msmarco-passage-models)

* [Multilingual Models](../sentence_transformer/pretrained_models.html#multilingual-models)

* [Semantic Similarity Models](../sentence_transformer/pretrained_models.html#semantic-similarity-models)

* [Bitext Mining](../sentence_transformer/pretrained_models.html#bitext-mining)

* [Image & Text-Models](../sentence_transformer/pretrained_models.html#image-text-models)

* [INSTRUCTOR models](../sentence_transformer/pretrained_models.html#instructor-models)

* [Scientific Similarity Models](../sentence_transformer/pretrained_models.html#scientific-similarity-models)

* [Training Overview](../sentence_transformer/training_overview.html)

* [Why Finetune?](../sentence_transformer/training_overview.html#why-finetune)

* [Training Components](../sentence_transformer/training_overview.html#training-components)

* [Dataset](../sentence_transformer/training_overview.html#dataset)

* [Dataset Format](../sentence_transformer/training_overview.html#dataset-format)

* [Loss Function](../sentence_transformer/training_overview.html#loss-function)

* [Training Arguments](../sentence_transformer/training_overview.html#training-arguments)

* [Paraphrase Data](../../examples/training/paraphrases/README.html)
  * [Pre-Trained Models](../../examples/training/paraphrases/README.html#pre-trained-models)
* [Quora Duplicate Questions](../../examples/training/quora_duplicate_questions/README.html)
  * [Training](../../examples/training/quora_duplicate_questions/README.html#training)
  * [MultipleNegativesRankingLoss](../../examples/training/quora_duplicate_questions/README.html#multiplenegativesrankingloss)
  * [Pretrained Models](../../examples/training/quora_duplicate_questions/README.html#pretrained-models)
* [MS MARCO](../../examples/training/ms_marco/README.html)
  * [Bi-Encoder](../../examples/training/ms_marco/README.html#bi-encoder)
* [Matryoshka Embeddings](../../examples/training/matryoshka/README.html)
  * [Use Cases](../../examples/training/matryoshka/README.html#use-cases)
  * [Results](../../examples/training/matryoshka/README.html#results)
  * [Training](../../examples/training/matryoshka/README.html#training)
  * [Inference](../../examples/training/matryoshka/README.html#inference)
  * [Code Examples](../../examples/training/matryoshka/README.html#code-examples)
* [Adaptive Layers](../../examples/training/adaptive_layer/README.html)
  * [Use Cases](../../examples/training/adaptive_layer/README.html#use-cases)
  * [Results](../../examples/training/adaptive_layer/README.html#results)
  * [Training](../../examples/training/adaptive_layer/README.html#training)
  * [Inference](../../examples/training/adaptive_layer/README.html#inference)
  * [Code Examples](../../examples/training/adaptive_layer/README.html#code-examples)
* [Multilingual Models](../../examples/training/multilingual/README.html)
  * [Extend your own models](../../examples/training/multilingual/README.html#extend-your-own-models)
  * [Training](../../examples/training/multilingual/README.html#training)

* [Datasets](../../examples/training/multilingual/README.html#datasets)
* [Sources for Training Data](../../examples/training/multilingual/README.html#sources-for-training-data)
* [Evaluation](../../examples/training/multilingual/README.html#evaluation)
* [Available Pre-trained Models](../../examples/training/multilingual/README.html#available-pre-trained-models)
* [Usage](../../examples/training/multilingual/README.html#usage)
* [Performance](../../examples/training/multilingual/README.html#performance)
* [Citation](../../examples/training/multilingual/README.html#citation)
* [Model Distillation](../../examples/training/distillation/README.html)
* [Knowledge Distillation](../../examples/training/distillation/README.html#knowledge-distillation)
* [Speed - Performance Trade-Off](../../examples/training/distillation/README.html#speed-performance-trade-off)
* [Dimensionality Reduction](../../examples/training/distillation/README.html#dimensionality-reduction)
* [Quantization](../../examples/training/distillation/README.html#quantization)
* [Augmented SBERT](../../examples/training/data_augmentation/README.html)
* [Motivation](../../examples/training/data_augmentation/README.html#motivation)
* [Extend to your own datasets](../../examples/training/data_augmentation/README.html#extend-to-your-own-datasets)
* [Methodology](../../examples/training/data_augmentation/README.html#methodology)
* [Scenario 1: Limited or small annotated datasets (few labeled sentence-pairs)](../../examples/training/data_augmentation/README.html#scenario-1-limited-or-small-annotated-datasets-few-labeled-sentence-pairs)
* [Scenario 2: No annotated datasets (Only unlabeled sentence-pairs)](../../examples/training/data_augmentation/README.html#scenario-2-no-annotated-datasets-only-unlabeled-sentence-pairs)

* [Domain Adaptation vs. Unsupervised Learning](../../examples/domain_adaptation/README.html#domain-adaptation-vs-unsupervised-learning)
    * [Adaptive Pre-Training](../../examples/domain_adaptation/README.html#adaptive-pre-training)
    * [GPL: Generative Pseudo-Labeling](../../examples/domain_adaptation/README.html#gpl-generative-pseudo-labeling)
  * [Hyperparameter Optimization](../../examples/training/hpo/README.html)
    * [HPO Components](../../examples/training/hpo/README.html#hpo-components)
    * [Putting It All Together](../../examples/training/hpo/README.html#putting-it-all-together)
    * [Example Scripts](../../examples/training/hpo/README.html#example-scripts)
  * [Distributed Training](../sentence_transformer/training/distributed.html)
    * [Comparison](../sentence_transformer/training/distributed.html#comparison)
    * [FSDP](../sentence_transformer/training/distributed.html#fsdp)

Cross Encoder

* [Usage](../cross_encoder/usage/usage.html)
  * [Retrieve & Re-Rank](../../examples/applications/retrieve_rerank/README.html)
    * [Retrieve & Re-Rank Pipeline](../../examples/applications/retrieve_rerank/README.html#retrieve-re-rank-pipeline)
    * [Retrieval: Bi-Encoder](../../examples/applications/retrieve_rerank/README.html#retrieval-bi-encoder)
    * [Re-Ranker: Cross-Encoder](../../examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder)
    * [Example Scripts](../../examples/applications/retrieve_rerank/README.html#example-scripts)
    * [Pre-trained Bi-Encoders (Retrieval)](../../examples/applications/retrieve_rerank/README.html#pre-trained-bi-encoders-retrie

val)

* [Pre-trained Cross-Encoders (Re-Ranker)](../../examples/applications/retrieve_rerank/README.html#pre-trained-cross-encoders-re-ranker)
  * [Pretrained Models](../cross_encoder/pretrained_models.html)
    * [MS MARCO](../cross_encoder/pretrained_models.html#ms-marco)
    * [SQuAD (QNLI)](../cross_encoder/pretrained_models.html#squad-qnli)
    * [STSbenchmark](../cross_encoder/pretrained_models.html#stsbenchmark)
    * [Quora Duplicate Questions](../cross_encoder/pretrained_models.html#quora-duplicate-questions)
    * [NLI](../cross_encoder/pretrained_models.html#nli)
    * [Community Models](../cross_encoder/pretrained_models.html#community-models)
  * [Training Overview](../cross_encoder/training_overview.html)
  * [Training Examples](../cross_encoder/training/examples.html)
    * [MS MARCO](../../examples/training/ms_marco/cross_encoder_README.html)
      * [Cross-Encoder](../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder)
      * [Cross-Encoder Knowledge Distillation](../../examples/training/ms_marco/cross_encoder_README.html#cross-encoder-knowledge-distillation)

Package Reference

  * [Sentence Transformer](sentence_transformer/index.html)
    * [SentenceTransformer](sentence_transformer/SentenceTransformer.html)
      * [SentenceTransformer](sentence_transformer/SentenceTransformer.html#id1)
      *

[SentenceTransformerModelCardData](sentence_transformer/SentenceTransformer.html#sentencetransformermodelcarddata)
      * [SimilarityFunction](sentence_transformer/SentenceTransformer.html#similarityfunction)
    * [Trainer](sentence_transformer/trainer.html)
      * [SentenceTransformerTrainer](sentence_transformer/trainer.html#sentencetransformertrainer)
    * [Training Arguments](sentence_transformer/training_args.html)
      * [SentenceTransformerTrainingArguments](sentence_transformer/training_args.html#sentencetransformertrainingarguments)
    * [Losses](sentence_transformer/losses.html)
      * [BatchAllTripletLoss](sentence_transformer/losses.html#batchalltripletloss)
      * [BatchHardSoftMarginTripletLoss](sentence_transformer/losses.html#batchhardsoftmargintripletloss)
      * [BatchHardTripletLoss](sentence_transformer/losses.html#batchhardtripletloss)
      * [BatchSemiHardTripletLoss](sentence_transformer/losses.html#batchsemihardtripletloss)
      * [ContrastiveLoss](sentence_transformer/losses.html#contrastiveloss)
      * [OnlineContrastiveLoss](sentence_transformer/losses.html#onlinecontrastiveloss)
      * [ContrastiveTensionLoss](sentence_transformer/losses.html#contrastivetensionloss)
      * [ContrastiveTensionLossInBatchNegatives](sentence_transformer/losses.html#contrastivetensionlossinbatchnegatives)
      * [CoSENTLoss](sentence_transformer/losses.html#cosentloss)
      * [AnglELoss](sentence_transformer/losses.html#angleloss)
      * [CosineSimilarityLoss](sentence_transformer/losses.html#cosinesimilarityloss)
      * [DenoisingAutoEncoderLoss](sentence_transformer/losses.html#denoisingautoencoderloss)
      * [GISTEmbedLoss](sentence_transformer/losses.html#gistembedloss)

    * [CachedGISTEmbedLoss](sentence_transformer/losses.html#cachedgistembedloss)
    * [MSELoss](sentence_transformer/losses.html#mseloss)
    * [MarginMSELoss](sentence_transformer/losses.html#marginmseloss)
    * [MatryoshkaLoss](sentence_transformer/losses.html#matryoshkaloss)
    * [Matryoshka2dLoss](sentence_transformer/losses.html#matryoshka2dloss)
    * [AdaptiveLayerLoss](sentence_transformer/losses.html#adaptivelayerloss)
    * [MegaBatchMarginLoss](sentence_transformer/losses.html#megabatchmarginloss)
    * [MultipleNegativesRankingLoss](sentence_transformer/losses.html#multiplenegativesrankingloss)
    * [CachedMultipleNegativesRankingLoss](sentence_transformer/losses.html#cachedmultiplenegativesrankingloss)
    * [MultipleNegativesSymmetricRankingLoss](sentence_transformer/losses.html#multiplenegativessymmetricrankingloss)
    * [CachedMultipleNegativesSymmetricRankingLoss](sentence_transformer/losses.html#cachedmultiplenegativessymmetricrankingloss)
    * [SoftmaxLoss](sentence_transformer/losses.html#softmaxloss)
    * [TripletLoss](sentence_transformer/losses.html#tripletloss)
  * [Samplers](sentence_transformer/sampler.html)
    * [BatchSamplers](sentence_transformer/sampler.html#batchsamplers)
    * [MultiDatasetBatchSamplers](sentence_transformer/sampler.html#multidatasetbatchsamplers)
  * [Evaluation](sentence_transformer/evaluation.html)
    * [BinaryClassificationEvaluator](sentence_transformer/evaluation.html#binaryclassificationevaluator)
    *

[EmbeddingSimilarityEvaluator](sentence_transformer/evaluation.html#embeddingsimilarityevaluator)

*

[InformationRetrievalEvaluator](sentence_transformer/evaluation.html#informationretrievalevaluator)

    * [NanoBEIREvaluator](sentence_transformer/evaluation.html#nanobeirevaluator)

    * [MSEEvaluator](sentence_transformer/evaluation.html#mseevaluator)

*

[ParaphraseMiningEvaluator](sentence_transformer/evaluation.html#paraphraseminingevaluator)

    * [RerankingEvaluator](sentence_transformer/evaluation.html#rerankingevaluator)

    * [SentenceEvaluator](sentence_transformer/evaluation.html#sentenceevaluator)

    * [SequentialEvaluator](sentence_transformer/evaluation.html#sequentialevaluator)

    * [TranslationEvaluator](sentence_transformer/evaluation.html#translationevaluator)

    * [TripletEvaluator](sentence_transformer/evaluation.html#tripletevaluator)

  * [Datasets](sentence_transformer/datasets.html)

    * [ParallelSentencesDataset](sentence_transformer/datasets.html#parallelsentencesdataset)

    * [SentenceLabelDataset](sentence_transformer/datasets.html#sentencelabeldataset)

*

[DenoisingAutoEncoderDataset](sentence_transformer/datasets.html#denoisingautoencoderdataset)

    * [NoDuplicatesDataLoader](sentence_transformer/datasets.html#noduplicatesdataloader)

  * [Models](sentence_transformer/models.html)

    * [Main Classes](sentence_transformer/models.html#main-classes)

    * [Further Classes](sentence_transformer/models.html#further-classes)

  * [quantization](sentence_transformer/quantization.html)

*

[`quantize_embeddings()`](sentence_transformer/quantization.html#sentence_transformers.quantization.quantize_embeddings)

*
[`semantic_search_faiss()`](sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_faiss)
*
[`semantic_search_usearch()`](sentence_transformer/quantization.html#sentence_transformers.quantization.semantic_search_usearch)
  * [Cross Encoder](cross_encoder/index.html)
    * [CrossEncoder](cross_encoder/cross_encoder.html)
      * [CrossEncoder](cross_encoder/cross_encoder.html#id1)
      * [Training Inputs](cross_encoder/cross_encoder.html#training-inputs)
    * [Evaluation](cross_encoder/evaluation.html)
      * [CEBinaryAccuracyEvaluator](cross_encoder/evaluation.html#cebinaryaccuracyevaluator)
*
[CEBinaryClassificationEvaluator](cross_encoder/evaluation.html#cebinaryclassificationevaluator)
      * [CECorrelationEvaluator](cross_encoder/evaluation.html#cecorrelationevaluator)
      * [CEF1Evaluator](cross_encoder/evaluation.html#cef1evaluator)
      * [CESoftmaxAccuracyEvaluator](cross_encoder/evaluation.html#cesoftmaxaccuracyevaluator)
      * [CERerankingEvaluator](cross_encoder/evaluation.html#cererankingevaluator)
  * util
    * Helper Functions
      * `community_detection()`
      * `http_get()`
      * `is_training_available()`
      * `mine_hard_negatives()`
      * `normalize_embeddings()`
      * `paraphrase_mining()`
      * `semantic_search()`

    * `truncate_embeddings()`
  * Model Optimization
    * `export_dynamic_quantized_onnx_model()`
    * `export_optimized_onnx_model()`
    * `export_static_quantized_openvino_model()`
  * Similarity Metrics
    * `cos_sim()`
    * `dot_score()`
    * `euclidean_sim()`
    * `manhattan_sim()`
    * `pairwise_cos_sim()`
    * `pairwise_dot_score()`
    * `pairwise_euclidean_sim()`
    * `pairwise_manhattan_sim()`

__[Sentence Transformers](../../index.html)

  * [](../../index.html)
  * util
  * [Edit on GitHub](https://github.com/UKPLab/sentence-transformers/blob/master/docs/package_reference/util.md)

* * *

utilïƒ•

`sentence_transformers.util` defines different helpful functions to work with
text embeddings.

Helper Functionsïƒ•

sentence_transformers.util.community_detection(_embeddings : [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") | ndarray_, _threshold : float = 0.75_, _min_community_size : int = 10_, _batch_size : int = 1024_, _show_progress_bar : bool = False_) -> list[list[int]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L1151-L1250)ïƒ•

Function for Fast Community Detection.

Finds in the embeddings all communities, i.e. embeddings that are close
(closer than threshold). Returns only communities that are larger than
min_community_size. The communities are returned in decreasing order. The
first element in each list is the central point in the community.

Parameters:

* **embeddings** ([_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _or_ _numpy.ndarray_) â€“ The input embeddings.

* **threshold** (_float_) â€“ The threshold for determining if two embeddings are close. Defaults to 0.75.

* **min_community_size** (_int_) â€“ The minimum size of a community to be considered. Defaults to 10.

* **batch_size** (_int_) â€“ The batch size for computing cosine similarity scores. Defaults to 1024.

* **show_progress_bar** (_bool_) â€“ Whether to show a progress bar during computation. Defaults to False.

Returns:

A list of communities, where each community is represented as a list of
indices.

Return type:

List[List[int]]

sentence_transformers.util.http_get(_url : str_, _path : str_) ->
None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L1033-L1067)ïƒ•

Downloads a URL to a given path on disk.

Parameters:

* **url** (_str_) â€“ The URL to download.

* **path** (_str_) â€“ The path to save the downloaded file.

Raises:

requests.HTTPError â€“ If the HTTP request returns a non-200 status code.

Returns:

None

sentence_transformers.util.is_training_available() ->

bool[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\util.py#L1488-L1493)ïƒ•

Returns True if we have the required dependencies for training Sentence Transformers models, i.e. Huggingface datasets and Huggingface accelerate.

sentence_transformers.util.mine_hard_negatives(_dataset : Dataset_, _model : [SentenceTransformer](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")_, _anchor_column_name : str | None = None_, _positive_column_name : str | None = None_, _corpus : list[str] | None = None_, _cross_encoder : [CrossEncoder](cross_encoder/cross_encoder.html#sentence_transformers.cross_encoder.CrossEncoder "sentence_transformers.cross_encoder.CrossEncoder") | None = None_, _range_min : int = 0_, _range_max : int | None = None_, _max_score : float | None = None_, _min_score : float | None = None_, _margin : float | None = None_, _num_negatives : int = 3_, _sampling_strategy : Literal['random', 'top'] = 'top'_, _as_triplets : bool = True_, _batch_size : int = 32_, _faiss_batch_size : int = 16384_, _use_faiss : bool = False_, _use_multi_process : list[str] | bool = False_, _verbose : bool = True_) -> Dataset[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L520-L1030)ïƒ•

Add hard negatives to a dataset of (anchor, positive) pairs to create (anchor, positive, negative) triplets or (anchor, positive, negative_1, â€¦, negative_n) tuples.

Hard negative mining is a technique to improve the quality of a dataset by adding hard negatives, which are texts that may appear similar to the anchor, but are not. Using hard negatives can improve the performance of models trained on the dataset.

This function uses a SentenceTransformer model to embed the sentences in the dataset, and then finds the closest matches to each anchor sentence in the dataset. It then samples negatives from the closest matches, optionally using a CrossEncoder model to rescore the candidates.

You can influence the candidate negative selection in various ways:

* **range_min** : Minimum rank of the closest matches to consider as negatives: useful to skip the most similar texts to avoid marking texts as negative that are actually positives.

* **range_max** : Maximum rank of the closest matches to consider as negatives: useful to limit the number of candidates to sample negatives from. A lower value makes processing faster, but may result in less candidate negatives that satisfy the margin or max_score conditions.

* **max_score** : Maximum score to consider as a negative: useful to skip candidates that are too similar to the anchor.

* **min_score** : Minimum score to consider as a negative: useful to skip candidates that are too dissimilar to the anchor.

* **margin** : Margin for hard negative mining: useful to skip candidates negatives whose similarity to the anchor is within a certain margin of the positive pair. A value of 0 can be used to enforce that

the negative is always further away from the anchor than the positive.

* **sampling_strategy** : Sampling strategy for negatives: â€œtopâ€• or â€œrandomâ€•. â€œtopâ€• will always sample the top n candidates as negatives, while â€œrandomâ€• will sample n negatives randomly from the candidates that satisfy the margin or max_score conditions.

Example

```
>>> from sentence_transformers.util import mine_hard_negatives
>>> from sentence_transformers import SentenceTransformer
>>> from datasets import load_dataset
>>> # Load a Sentence Transformer model
>>> model = SentenceTransformer("all-MiniLM-L6-v2")
>>>
>>> # Load a dataset to mine hard negatives from
>>> dataset = load_dataset("sentence-transformers/natural-questions", split="train")
>>> dataset
Dataset({
    features: ['query', 'answer'],
    num_rows: 100231
})
>>> dataset = mine_hard_negatives(
...     dataset=dataset,
...     model=model,
...     range_min=10,
```

```
...     range_max=50,
...     max_score=0.8,
...     margin=0.1,
...     num_negatives=5,
...     sampling_strategy="random",
...     batch_size=128,
...     use_faiss=True,
... )
Batches:
100%|â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–
ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ
–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ
â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 784/784 [00:43<00:00,
17.83it/s]
Batches:
100%|â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–
ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ
–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ
â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 784/784 [00:07<00:00,
99.60it/s]
Querying FAISS index:
100%|â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–
ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ
–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ
â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 784/784 [00:00<00:00, 884.99it/s]
```

Metric	Positive	Negative	Difference
Count	100,231	431,255	431,255

Mean 0.6866 0.4289 0.2804

Median 0.7010 0.4193 0.2740

Std 0.1125 0.0754 0.0999

Min 0.0303 0.1720 0.1001

25% 0.6221 0.3747 0.1991

50% 0.7010 0.4193 0.2740

75% 0.7667 0.4751 0.3530

Max 0.9584 0.7743 0.7003

Skipped 1289492 potential negatives (25.23%) due to the margin of 0.1.

Skipped 39 potential negatives (0.00%) due to the maximum score of 0.8.

Could not find enough negatives for 69900 samples (13.95%). Consider adjusting the range_max, range_min, margin and max_score parameters if you'd like to find more valid negatives.

```
>>> # Note: The minimum similarity difference is 0.1001 due to our margin of 0.1
>>> dataset
Dataset({
    features: ['query', 'answer', 'negative'],
    num_rows: 431255
})
>>> dataset[0]
{
    'query': 'when did richmond last play in a preliminary final',
     'answer': "Richmond Football Club Richmond began 2017 with 5 straight wins, a feat it had not achieved since 1995. A series of close losses hampered the Tigers throughout the middle of the season, including a 5-point loss to the Western Bulldogs, 2-point loss to Fremantle, and a 3-point loss to the Giants. Richmond ended the season strongly with convincing victories over Fremantle and St Kilda in the final two rounds, elevating the club to 3rd on the ladder. Richmond's first final of the season against the Cats at the MCG attracted a record qualifying final crowd of 95,028; the
```

Tigers won by 51 points. Having advanced to the first preliminary finals for the first time since 2001, Richmond defeated Greater Western Sydney by 36 points in front of a crowd of 94,258 to progress to the Grand Final against Adelaide, their first Grand Final appearance since 1982. The attendance was 100,021, the largest crowd to a grand final since 1986. The Crows led at quarter time and led by as many as 13, but the Tigers took over the game as it progressed and scored seven straight goals at one point. They eventually would win by 48 points â€“ 16.12 (108) to Adelaide's 8.12 (60) â€“ to end their 37-year flag drought.[22] Dustin Martin also became the first player to win a Premiership medal, the Brownlow Medal and the Norm Smith Medal in the same season, while Damien Hardwick was named AFL Coaches Association Coach of the Year. Richmond's jump from 13th to premiers also marked the biggest jump from one AFL season to the next.",

'negative': "2018 NRL Grand Final The 2018 NRL Grand Final was the conclusive and premiership-deciding game of the 2018 National Rugby League season and was played on Sunday September 30 at Sydney's ANZ Stadium.[1] The match was contested between minor premiers the Sydney Roosters and defending premiers the Melbourne Storm. In front of a crowd of 82,688, Sydney won the match 21Ã¢â‚¬â€œ6 to claim their 14th premiership title and their first since 2013. Roosters five-eighth Luke Keary was awarded the Clive Churchill Medal as the game's official man of the match."

}

>>> dataset.push_to_hub("natural-questions-hard-negatives", "triplet-all")

Parameters:

* **dataset** (_Dataset_) â€“ A dataset containing (anchor, positive) pairs.

* **model** ([_SentenceTransformer_](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ A SentenceTransformer model to use for embedding the sentences.

* **anchor_column_name** (_str_ _,__optional_) â€“ The column name in dataset that contains the anchor/query. Defaults to None, in which case the first column in dataset will be used.

* **positive_column_name** (_str_ _,__optional_) â€“ The column name in dataset that contains the positive candidates. Defaults to None, in which case the second column in dataset will be used.

* **corpus** (_List_ _[__str_ _]__,__optional_) â€“ A list containing documents as strings that will be used as candidate negatives in addition to the second column in dataset. Defaults to None, in which case the second column in dataset will exclusively be used as the negative candidate corpus.

* **cross_encoder** ([_CrossEncoder_](cross_encoder/cross_encoder.html#sentence_transformers.cross_encoder.CrossEncoder "sentence_transformers.cross_encoder.CrossEncoder") _,__optional_) â€“ A CrossEncoder model to use for rescoring the candidates. Defaults to None.

* **range_min** (_int_) â€“ Minimum rank of the closest matches to consider as negatives. Defaults to 0.

* **range_max** (_int_ _,__optional_) â€“ Maximum rank of the closest matches to consider as negatives. Defaults to None.

* **max_score** (_float_ _,__optional_) â€“ Maximum score to consider as a negative. Defaults to

None.

* **min_score** (_float_ _,__optional_) â€“ Minimum score to consider as a negative. Defaults to None.

* **margin** (_float_ _,__optional_) â€“ Margin for hard negative mining. Defaults to None.

* **num_negatives** (_int_) â€“ Number of negatives to sample. Defaults to 3.

* **sampling_strategy** (_Literal_ _[__" random"__,__" top"__]_) â€“ Sampling strategy for negatives: â€œtopâ€• or â€œrandomâ€•. Defaults to â€œtopâ€•.

* **as_triplets** (_bool_) â€“ If True, returns up to num_negatives (anchor, positive, negative) triplets for each input sample. If False, returns 1 (anchor, positive, negative_1, â€¦, negative_n) tuple for each input sample. Defaults to True.

* **batch_size** (_int_) â€“ Batch size for encoding the dataset. Defaults to 32.

* **faiss_batch_size** (_int_) â€“ Batch size for FAISS top-k search. Defaults to 16384.

* **use_faiss** (_bool_) â€“ Whether to use FAISS for similarity search. May be recommended for large datasets. Defaults to False.

* **use_multi_process** (_bool_ _|__List_ _[__str_ _]__,__optional_) â€“ Whether to use multi-GPU/CPU processing. If True, uses all GPUs if CUDA is available, and 4 CPU processes if itâ€™s not available. You can also pass a list of PyTorch devices like [â€œcuda:0â€•, â€œcuda:1â€•, â€¦] or [â€œcpuâ€•, â€œcpuâ€•, â€œcpuâ€•, â€œcpuâ€•].

* **verbose** (_bool_) â€“ Whether to print statistics and logging. Defaults to True.

Returns:

A dataset containing (anchor, positive, negative) triplets or (anchor,

positive, negative_1, â€¦, negative_n) tuples.

Return type:

Dataset

sentence_transformers.util.normalize_embeddings(_embeddings :

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\)")_) ->

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\)")[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\util.py#L264-L274)ïƒ•

Normalizes the embeddings matrix, so that each sentence embedding has unit

length.

Parameters:

embeddings (_Tensor_) â€“ The input embeddings matrix.

Returns:

The normalized embeddings matrix.

Return type:

Tensor

sentence_transformers.util.paraphrase_mining(_model, sentences: list[str], show_progress_bar: bool = False, batch_size: int = 32, query_chunk_size: int = 5000, corpus_chunk_size: int = 100000, max_pairs: int = 500000, top_k: int = 100, score_function: ~typing.Callable[[~torch.Tensor, ~torch.Tensor], ~torch.Tensor] = <function cos_sim>_) -> list[list[float | int]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L317-L359)ïƒ•

Given a list of sentences / texts, this function performs paraphrase mining.

It compares all sentences against all other sentences and returns a list with

the pairs that have the highest cosine similarity score.

Parameters:

* **model** ([_SentenceTransformer_](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ SentenceTransformer model for embedding computation

* **sentences** (_List_ _[__str_ _]_) â€“ A list of strings (texts or sentences)

* **show_progress_bar** (_bool_ _,__optional_) â€“ Plotting of a progress bar. Defaults to False.

* **batch_size** (_int_ _,__optional_) â€“ Number of texts that are encoded simultaneously by the model. Defaults to 32.

* **query_chunk_size** (_int_ _,__optional_) â€“ Search for most similar pairs for #query_chunk_size at the same time. Decrease, to lower memory footprint (increases run-time). Defaults to 5000.

* **corpus_chunk_size** (_int_ _,__optional_) â€“ Compare a sentence simultaneously against #corpus_chunk_size other sentences. Decrease, to lower memory footprint (increases run-time). Defaults to 100000.

* **max_pairs** (_int_ _,__optional_) â€“ Maximal number of text pairs returned. Defaults to 500000.

* **top_k** (_int_ _,__optional_) â€“ For each sentence, we retrieve up to top_k other sentences. Defaults to 100.

* **score_function** (_Callable_ _[__[__Tensor_ _,__Tensor_ _]__,__Tensor_ _]__,__optional_) â€“ Function for computing scores. By default, cosine similarity. Defaults to cos_sim.

Returns:

Returns a list of triplets with the format [score, id1, id2]

Return type:

List[List[Union[float, int]]]

sentence_transformers.util.semantic_search(_query_embeddings: ~torch.Tensor, corpus_embeddings: ~torch.Tensor, query_chunk_size: int = 100, corpus_chunk_size: int = 500000, top_k: int = 10, score_function: ~typing.Callable[[~torch.Tensor, ~torch.Tensor], ~torch.Tensor] = <function cos_sim>_) -> list[list[dict[str, int | float]]][[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transform

ers\\util.py#L440-L517)ïƒ•

This function performs a cosine similarity search between a list of query embeddings and a list of corpus embeddings. It can be used for Information Retrieval / Semantic Search for corpora up to about 1 Million entries.

Parameters:

* **query_embeddings** (_Tensor_) â€“ A 2 dimensional tensor with the query embeddings.

* **corpus_embeddings** (_Tensor_) â€“ A 2 dimensional tensor with the corpus embeddings.

* **query_chunk_size** (_int_ _,__optional_) â€“ Process 100 queries simultaneously. Increasing that value increases the speed, but requires more memory. Defaults to 100.

* **corpus_chunk_size** (_int_ _,__optional_) â€“ Scans the corpus 100k entries at a time. Increasing that value increases the speed, but requires more memory. Defaults to 500000.

* **top_k** (_int_ _,__optional_) â€“ Retrieve top k matching entries. Defaults to 10.

* **score_function** (_Callable_ _[__[__Tensor_ _,__Tensor_ _]__,__Tensor_ _]__,__optional_) â€“ Function for computing scores. By default, cosine similarity.

Returns:

A list with one entry for each query. Each entry is a list of dictionaries

with the keys â€˜corpus_idâ€™ and â€˜scoreâ€™, sorted by decreasing cosine

similarity scores.

Return type:

List[List[Dict[str, Union[int, float]]]]

sentence_transformers.util.truncate_embeddings(_embeddings : ndarray_, _truncate_dim : int | None_) -> ndarray[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L285-L314)ïƒ•

sentence_transformers.util.truncate_embeddings(_embeddings : [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_, _truncate_dim : int | None_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")

Truncates the embeddings matrix.

Parameters:

* **embeddings** (_Union_ _[__np.ndarray_ _,_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _]_) â€“ Embeddings to truncate.

* **truncate_dim** (_Optional_ _[__int_ _]_) â€“ The dimension to truncate sentence embeddings to. None does no truncation.

Example

```
>>> from sentence_transformers import SentenceTransformer
>>> from sentence_transformers.util import truncate_embeddings
>>> model = SentenceTransformer("tomaarsen/mpnet-base-nli-matryoshka")
>>> embeddings = model.encode(["It's so nice outside!", "Today is a beautiful day.", "He drove to work earlier"])
>>> embeddings.shape
(3, 768)
>>> model.similarity(embeddings, embeddings)
tensor([[1.0000, 0.8100, 0.1426],
        [0.8100, 1.0000, 0.2121],
        [0.1426, 0.2121, 1.0000]])
>>> truncated_embeddings = truncate_embeddings(embeddings, 128)
>>> truncated_embeddings.shape
>>> model.similarity(truncated_embeddings, truncated_embeddings)
```

tensor([[1.0000, 0.8092, 0.1987],

[0.8092, 1.0000, 0.2716],

[0.1987, 0.2716, 1.0000]])

Returns:

Truncated embeddings.

Return type:

Union[np.ndarray,

[torch.Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")]

Model Optimizationïƒ•

sentence_transformers.backend.export_dynamic_quantized_onnx_model(_model :

[SentenceTransformer](sentence_transformer/SentenceTransformer.html#sentence_transformers.S

entenceTransformer "sentence_transformers.SentenceTransformer")_, _quantization_config :

QuantizationConfig | Literal['arm64', 'avx2', 'avx512', 'avx512_vnni']_, _model_name_or_path : str_,

_push_to_hub : bool = False_, _create_pr : bool = False_, _file_suffix : str | None = None_) ->

None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transforme

rs\\backend.py#L117-L198)ïƒ•

Export a quantized ONNX model from a SentenceTransformer model.

This function applies dynamic quantization, i.e. without a calibration dataset. Each of the default quantization configurations quantize the model to int8, allowing for faster inference on CPUs, but are likely slower on GPUs.

See <https://sbert.net/docs/sentence_transformer/usage/efficiency.html> for more information & benchmarks.

Parameters:

* **model** ([_SentenceTransformer_](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ The SentenceTransformer model to be quantized. Must be loaded with backend=â€•onnxâ€•.

* **quantization_config** (_QuantizationConfig_) â€“ The quantization configuration.

* **model_name_or_path** (_str_) â€“ The path or Hugging Face Hub repository name where the quantized model will be saved.

* **push_to_hub** (_bool_ _,__optional_) â€“ Whether to push the quantized model to the Hugging

Face Hub. Defaults to False.

* **create_pr** (_bool_ _,__optional_) â€“ Whether to create a pull request when pushing to the Hugging Face Hub. Defaults to False.

* **file_suffix** (_str_ _|__None_ _,__optional_) â€“ The suffix to add to the quantized model file name. Defaults to None.

Raises:

* **ImportError** â€“ If the required packages optimum and onnxruntime are not installed.

* **ValueError** â€“ If the provided model is not a valid SentenceTransformer model loaded with backend=â€•onnxâ€•.

* **ValueError** â€“ If the provided quantization_config is not valid.

Returns:

None

sentence_transformers.backend.export_optimized_onnx_model(_model : [SentenceTransformer](sentence_transformer/SentenceTransformer.html#sentence_transformers.S

entenceTransformer "sentence_transformers.SentenceTransformer")_, _optimization_config : OptimizationConfig | Literal['O1', 'O2', 'O3', 'O4']_, _model_name_or_path : str_, _push_to_hub : bool = False_, _create_pr : bool = False_, _file_suffix : str | None = None_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\backend.py#L28-L114)ïƒ•

Export an optimized ONNX model from a SentenceTransformer model.

The O1-O4 optimization levels are defined by Optimum and are documented here:
<https://huggingface.co/docs/optimum/main/en/onnxruntime/usage_guides/optimization>

The optimization levels are:

* O1: basic general optimizations.

* O2: basic and extended general optimizations, transformers-specific fusions.

* O3: same as O2 with GELU approximation.

* O4: same as O3 with mixed precision (fp16, GPU-only)

See <https://sbert.net/docs/sentence_transformer/usage/efficiency.html> for
more information & benchmarks.

Parameters:

* **model** ([_SentenceTransformer_](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ The SentenceTransformer model to be optimized. Must be loaded with backend=â€•onnxâ€•.

* **optimization_config** (_OptimizationConfig_ _|__Literal_ _[__" O1"__,__" O2"__,__" O3"__,__" O4"__]_) â€“ The optimization configuration or level.

* **model_name_or_path** (_str_) â€“ The path or Hugging Face Hub repository name where the optimized model will be saved.

* **push_to_hub** (_bool_ _,__optional_) â€“ Whether to push the optimized model to the Hugging Face Hub. Defaults to False.

* **create_pr** (_bool_ _,__optional_) â€“ Whether to create a pull request when pushing to the Hugging Face Hub. Defaults to False.

* **file_suffix** (_str_ _|__None_ _,__optional_) â€“ The suffix to add to the optimized model file name. Defaults to None.

Raises:

* **ImportError** â€“ If the required packages optimum and onnxruntime are not installed.

* **ValueError** â€“ If the provided model is not a valid SentenceTransformer model loaded with backend=â€•onnxâ€•.

* **ValueError** â€“ If the provided optimization_config is not valid.

Returns:

None

sentence_transformers.backend.export_static_quantized_openvino_model(_model : [SentenceTransformer](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")_, _quantization_config : OVQuantizationConfig | dict | None_, _model_name_or_path : str_, _dataset_name : str | None = None_, _dataset_config_name : str | None = None_, _dataset_split : str | None = None_, _column_name : str | None = None_, _push_to_hub : bool = False_, _create_pr : bool = False_, _file_suffix : str = 'qint8_quantized'_) -> None[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\backend.py#L201-L309)ïƒ•

Export a quantized OpenVINO model from a SentenceTransformer model.

This function applies Post-Training Static Quantization (PTQ) using a
calibration dataset, which calibrates quantization constants without requiring

model retraining. Each default quantization configuration converts the model to int8 precision, enabling faster inference while maintaining accuracy.

See <https://sbert.net/docs/sentence_transformer/usage/efficiency.html> for more information & benchmarks.

Parameters:

* **model** ([_SentenceTransformer_](sentence_transformer/SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer")) â€“ The SentenceTransformer model to be quantized. Must be loaded with backend=â€•openvinoâ€•.

* **quantization_config** (_OVQuantizationConfig_ _|__dict_ _|__None_) â€“ The quantization configuration. If None, default values are used.

* **model_name_or_path** (_str_) â€“ The path or Hugging Face Hub repository name where the quantized model will be saved.

* **dataset_name** (_str_ _,__optional_) â€“ The name of the dataset to load for calibration. If not specified, the sst2 subset of the glue dataset will be used by default.

* **dataset_config_name** (_str_ _,__optional_) â€“ The specific configuration of the dataset to load.

* **dataset_split** (_str_ _,__optional_) â€“ The split of the dataset to load (e.g., â€˜trainâ€™, â€˜testâ€™). Defaults to None.

* **column_name** (_str_ _,__optional_) â€“ The column name in the dataset to use for calibration. Defaults to None.

* **push_to_hub** (_bool_ _,__optional_) â€“ Whether to push the quantized model to the Hugging Face Hub. Defaults to False.

* **create_pr** (_bool_ _,__optional_) â€“ Whether to create a pull request when pushing to the Hugging Face Hub. Defaults to False.

* **file_suffix** (_str_ _,__optional_) â€“ The suffix to add to the quantized model file name. Defaults to qint8_quantized.

Raises:

* **ImportError** â€“ If the required packages optimum and openvino are not installed.

* **ValueError** â€“ If the provided model is not a valid SentenceTransformer model loaded with backend=â€•openvinoâ€•.

* **ValueError** â€“ If the provided quantization_config is not valid.

Returns:

None

Similarity Metricsïƒ•

sentence_transformers.util.cos_sim(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L92-L108)ïƒ•

Computes the cosine similarity between two tensors.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Matrix with res[i][j] = cos_sim(a[i], b[j])

Return type:

Tensor

sentence_transformers.util.dot_score(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L128-L142)ïƒ•

Computes the dot-product dot_prod(a[i], b[j]) for all i and j.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Matrix with res[i][j] = dot_prod(a[i], b[j])

Return type:

Tensor

sentence_transformers.util.euclidean_sim(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L196-L210)ïƒ•

Computes the euclidean similarity (i.e., negative distance) between two tensors.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Matrix with res[i][j] = -euclidean_distance(a[i], b[j])

Return type:

Tensor

sentence_transformers.util.manhattan_sim(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")_) -> [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L162-L176)ïƒ•

Computes the manhattan similarity (i.e., negative distance) between two tensors.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Matrix with res[i][j] = -manhattan_distance(a[i], b[j])

Return type:

Tensor

sentence_transformers.util.pairwise_cos_sim(_a :

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")_, _b :

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")_) ->

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L111-L125)ïƒ•

Computes the pairwise cosine similarity cos_sim(a[i], b[i]).

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Vector with res[i] = cos_sim(a[i], b[i])

Return type:

Tensor

sentence_transformers.util.pairwise_dot_score(_a :

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")_, _b :

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")_) ->

[Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in

PyTorch v2.5\")[[source]](https://github.com/UKPLab/sentence-

transformers/blob/master/sentence_transformers\\util.py#L145-L159)ïƒ•

Computes the pairwise dot-product dot_prod(a[i], b[i]).

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Vector with res[i] = dot_prod(a[i], b[i])

Return type:

Tensor

sentence_transformers.util.pairwise_euclidean_sim(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L213-L227)ïƒ•

Computes the euclidean distance (i.e., negative distance) between pairs of
tensors.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Vector with res[i] = -euclidean_distance(a[i], b[i])

Return type:

Tensor

sentence_transformers.util.pairwise_manhattan_sim(_a : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_, _b : list | ndarray | [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)")_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\util.py#L179-L193)ïƒ•

Computes the manhattan similarity (i.e., negative distance) between pairs of tensors.

Parameters:

* **a** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The first tensor.

* **b** (_Union_ _[__list_ _,__np.ndarray_ _,__Tensor_ _]_) â€“ The second tensor.

Returns:

Vector with res[i] = -manhattan_distance(a[i], b[i])

Return type:

Tensor

[Previous](cross_encoder/evaluation.html "Evaluation")

* * *