[![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)

gloss)

* [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)

sentence-pairs)](../../../examples/training/data_augmentation/README.html#scenario-2-no-annotated-datasets-only-unlabeled-sentence-pairs)

* [Training](../../../examples/training/data_augmentation/README.html#training)

* [Citation](../../../examples/training/data_augmentation/README.html#citation)

* [Training with Prompts](../../../examples/training/prompts/README.html)

* [What are Prompts?](../../../examples/training/prompts/README.html#what-are-prompts)

* [Why would we train with Prompts?](../../../examples/training/prompts/README.html#why-would-we-train-with-prompts)

* [How do we train with Prompts?](../../../examples/training/prompts/README.html#how-do-we-train-with-prompts)

* [Training with PEFT Adapters](../../../examples/training/peft/README.html)

* [Compatibility Methods](../../../examples/training/peft/README.html#compatibility-methods)

* [Adding a New Adapter](../../../examples/training/peft/README.html#adding-a-new-adapter)

* [Loading a Pretrained Adapter](../../../examples/training/peft/README.html#loading-a-pretrained-adapter)

* [Training Script](../../../examples/training/peft/README.html#training-script)

* [Unsupervised Learning](../../../examples/unsupervised_learning/README.html)

* [TSDAE](../../../examples/unsupervised_learning/README.html#tsdae)

* [SimCSE](../../../examples/unsupervised_learning/README.html#simcse)

* [CT](../../../examples/unsupervised_learning/README.html#ct)

* [CT (In-Batch Negative Sampling)](../../../examples/unsupervised_learning/README.html#ct-in-batch-negative-sampling)

* [Masked Language Model (MLM)](../../../examples/unsupervised_learning/README.html#masked-language-model-mlm)

* [GenQ](../../../examples/unsupervised_learning/README.html#genq)

* [GPL](../../../examples/unsupervised_learning/README.html#gpl)

* [Performance

Comparison](../../../examples/unsupervised_learning/README.html#performance-comparison)

* [Domain Adaptation](../../../examples/domain_adaptation/README.html)

* [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-retrieval)

* [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

    * [DenoisingAutoEncoderLoss](losses.html#denoisingautoencoderloss)
    * [GISTEmbedLoss](losses.html#gistembedloss)
    * [CachedGISTEmbedLoss](losses.html#cachedgistembedloss)
    * [MSELoss](losses.html#mseloss)
    * [MarginMSELoss](losses.html#marginmseloss)
    * [MatryoshkaLoss](losses.html#matryoshkaloss)
    * [Matryoshka2dLoss](losses.html#matryoshka2dloss)
    * [AdaptiveLayerLoss](losses.html#adaptivelayerloss)
    * [MegaBatchMarginLoss](losses.html#megabatchmarginloss)
    * [MultipleNegativesRankingLoss](losses.html#multiplenegativesrankingloss)
    * [CachedMultipleNegativesRankingLoss](losses.html#cachedmultiplenegativesrankingloss)
    * [MultipleNegativesSymmetricRankingLoss](losses.html#multiplenegativessymmetricrankingloss)
    * [CachedMultipleNegativesSymmetricRankingLoss](losses.html#cachedmultiplenegativessymmetricrankingloss)
    * [SoftmaxLoss](losses.html#softmaxloss)
    * [TripletLoss](losses.html#tripletloss)
  * [Samplers](sampler.html)
    * [BatchSamplers](sampler.html#batchsamplers)
    * [MultiDatasetBatchSamplers](sampler.html#multidatasetbatchsamplers)
  * Evaluation
    * BinaryClassificationEvaluator
    * EmbeddingSimilarityEvaluator
    * InformationRetrievalEvaluator
    * NanoBEIREvaluator
    * MSEEvaluator

*

[`export_static_quantized_openvino_model()`](../util.html#sentence_transformers.backend.export_static_quantized_openvino_model)

* [Similarity Metrics](../util.html#module-sentence_transformers.util)
* [`cos_sim()`](../util.html#sentence_transformers.util.cos_sim)
* [`dot_score()`](../util.html#sentence_transformers.util.dot_score)
* [`euclidean_sim()`](../util.html#sentence_transformers.util.euclidean_sim)
* [`manhattan_sim()`](../util.html#sentence_transformers.util.manhattan_sim)
* [`pairwise_cos_sim()`](../util.html#sentence_transformers.util.pairwise_cos_sim)
* [`pairwise_dot_score()`](../util.html#sentence_transformers.util.pairwise_dot_score)
* [`pairwise_euclidean_sim()`](../util.html#sentence_transformers.util.pairwise_euclidean_sim)
* [`pairwise_manhattan_sim()`](../util.html#sentence_transformers.util.pairwise_manhattan_sim)

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

* [](../../../index.html)
* [Sentence Transformer](index.html)
* Evaluation

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

* * *

Evaluationïƒ•

`sentence_transformers.evaluation` defines different classes, that can be used

to evaluate the model during training.

BinaryClassificationEvaluatorï•

_class _sentence_transformers.evaluation.BinaryClassificationEvaluator(_sentences1 : list[str]_, _sentences2 : list[str]_, _labels : list[int]_, _name : str = ''_, _batch_size : int = 32_, _show_progress_bar : bool = False_, _write_csv : bool = True_, _truncate_dim : int | None = None_, _similarity_fn_names : list[Literal['cosine', 'dot', 'euclidean', 'manhattan']] | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\BinaryClassificationEvaluator.py#L23-L364)ï•

Evaluate a model based on the similarity of the embeddings by calculating the accuracy of identifying similar and dissimilar sentences. The metrics are the cosine similarity, dot score, Euclidean and Manhattan distance The returned score is the accuracy with a specified metric.

The results are written in a CSV. If a CSV already exists, then values are appended.

The labels need to be 0 for dissimilar pairs and 1 for similar pairs.

Parameters:

* **sentences1** (_List_ _[__str_ _]_) â€“ The first column of sentences.

* **sentences2** (_List_ _[__str_ _]_) â€“ The second column of sentences.

* **labels** (_List_ _[__int_ _]_) â€“ labels[i] is the label for the pair (sentences1[i], sentences2[i]). Must be 0 or 1.

* **name** (_str_ _,__optional_) â€“ Name for the output. Defaults to â€œâ€•.

* **batch_size** (_int_ _,__optional_) â€“ Batch size used to compute embeddings. Defaults to 32.

* **show_progress_bar** (_bool_ _,__optional_) â€“ If true, prints a progress bar. Defaults to False.

* **write_csv** (_bool_ _,__optional_) â€“ Write results to a CSV file. Defaults to True.

* **truncate_dim** (_Optional_ _[__int_ _]__,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

* **similarity_fn_names** (_Optional_ _[__List_ _[__Literal_ _[__" cosine"__,__" dot"__,__" euclidean"__,__" manhattan"__]__]__]__,__optional_) â€“ The similarity functions to use. If not specified, defaults to the `similarity_fn_name` attribute of the model. Defaults to None.

Example

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import BinaryClassificationEvaluator
```

```
from datasets import load_dataset

# Load a model
model = SentenceTransformer('all-mpnet-base-v2')

# Load a dataset with two text columns and a class label column (https://huggingface.co/datasets/sentence-transformers/quora-duplicates)
eval_dataset = load_dataset("sentence-transformers/quora-duplicates", "pair-class", split="train[-1000:]")

# Initialize the evaluator
binary_acc_evaluator = BinaryClassificationEvaluator(
    sentences1=eval_dataset["sentence1"],
    sentences2=eval_dataset["sentence2"],
    labels=eval_dataset["label"],
    name="quora_duplicates_dev",
)
results = binary_acc_evaluator(model)
'''
Binary Accuracy Evaluation of the model on the quora_duplicates_dev dataset:
Accuracy with Cosine-Similarity:           81.60  (Threshold: 0.8352)
F1 with Cosine-Similarity:                 75.27  (Threshold: 0.7715)
Precision with Cosine-Similarity:          65.81
Recall with Cosine-Similarity:             87.89
Average Precision with Cosine-Similarity:  76.03
Matthews Correlation with Cosine-Similarity: 62.48
'''
```

print(binary_acc_evaluator.primary_metric)

=> "quora_duplicates_dev_cosine_ap"

print(results[binary_acc_evaluator.primary_metric])

=> 0.760277070888393

Base class for all evaluators. Notably, this class introduces the
`greater_is_better` and `primary_metric` attributes. The former is a boolean
indicating whether a higher evaluation score is better, which is used for
choosing the best checkpoint if `load_best_model_at_end` is set to `True` in
the training arguments.

The latter is a string indicating the primary metric for the evaluator. This
has to be defined whenever the evaluator returns a dictionary of metrics, and
the primary metric is the key pointing to the primary metric, i.e. the one
that is used for model selection and/or logging.

EmbeddingSimilarityEvaluatorïƒ•

_class _sentence_transformers.evaluation.EmbeddingSimilarityEvaluator(_sentences1 : list[str]_, _sentences2 : list[str]_, _scores : list[float]_, _batch_size : int = 16_, _main_similarity : str | [SimilarityFunction](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.similarity_functions.SimilarityFunction") | None = None_, _similarity_fn_names : list[Literal['cosine', 'dot', 'euclidean', 'manhattan']] | None = None_, _name : str = ''_, _show_progress_bar : bool = False_, _write_csv : bool = True_, _precision : Literal['float32', 'int8', 'uint8', 'binary', 'ubinary'] | None = None_, _truncate_dim : int | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transfor

mers\\evaluation\\EmbeddingSimilarityEvaluator.py#L23-L257)ïƒ•

Evaluate a model based on the similarity of the embeddings by calculating the Spearman and Pearson rank correlation in comparison to the gold standard labels. The metrics are the cosine similarity as well as euclidean and Manhattan distance The returned score is the Spearman correlation with a specified metric.

Example

```
from datasets import load_dataset
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator, SimilarityFunction

# Load a model
model = SentenceTransformer('all-mpnet-base-v2')

# Load the STSB dataset (https://huggingface.co/datasets/sentence-transformers/stsb)
eval_dataset = load_dataset("sentence-transformers/stsb", split="validation")

# Initialize the evaluator
dev_evaluator = EmbeddingSimilarityEvaluator(
    sentences1=eval_dataset["sentence1"],
    sentences2=eval_dataset["sentence2"],
```

```
    scores=eval_dataset["score"],
    name="sts_dev",
)
results = dev_evaluator(model)
'''
EmbeddingSimilarityEvaluator: Evaluating the model on the sts-dev dataset:
Cosine-Similarity :  Pearson: 0.8806 Spearman: 0.8810
'''
print(dev_evaluator.primary_metric)
# => "sts_dev_pearson_cosine"
print(results[dev_evaluator.primary_metric])
# => 0.881019449484294
```

Constructs an evaluator based for the dataset.

Parameters:

* **sentences1** (_List_ _[__str_ _]_) â€“ List with the first sentence in a pair.

* **sentences2** (_List_ _[__str_ _]_) â€“ List with the second sentence in a pair.

* **scores** (_List_ _[__float_ _]_) â€“ Similarity score between sentences1[i] and sentences2[i].

* **batch_size** (_int_ _,__optional_) â€“ The batch size for processing the sentences. Defaults to

16.

* **main_similarity** (_Optional_ _[__Union_ _[__str_ _,_[_SimilarityFunction_](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction") _]__]__,__optional_) â€“ The main similarity function to use. Can be a string (e.g. â€œcosineâ€•, â€œdotâ€•) or a SimilarityFunction object. Defaults to None.

* **similarity_fn_names** (_List_ _[__str_ _]__,__optional_) â€“ List of similarity function names to use. If None, the `similarity_fn_name` attribute of the model is used. Defaults to None.

* **name** (_str_ _,__optional_) â€“ The name of the evaluator. Defaults to â€œâ€•.

* **show_progress_bar** (_bool_ _,__optional_) â€“ Whether to show a progress bar during evaluation. Defaults to False.

* **write_csv** (_bool_ _,__optional_) â€“ Whether to write the evaluation results to a CSV file. Defaults to True.

* **precision** (_Optional_ _[__Literal_ _[__" float32"__,__" int8"__,__" uint8"__,__" binary"__,__" ubinary"__]__]__,__optional_) â€“ The precision to use for the embeddings. Can be â€œfloat32â€•, â€œint8â€•, â€œuint8â€•, â€œbinaryâ€•, or â€œubinaryâ€•. Defaults to None.

* **truncate_dim** (_Optional_ _[__int_ _]__,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

InformationRetrievalEvaluatorï*ƒ*•

_class _sentence_transformers.evaluation.InformationRetrievalEvaluator(_queries : dict[str, str]_, _corpus : dict[str, str]_, _relevant_docs : dict[str, set[str]]_, _corpus_chunk_size : int = 50000_, _mrr_at_k : list[int] = [10]_, _ndcg_at_k : list[int] = [10]_, _accuracy_at_k : list[int] = [1, 3, 5, 10]_, _precision_recall_at_k : list[int] = [1, 3, 5, 10]_, _map_at_k : list[int] = [100]_, _show_progress_bar : bool = False_, _batch_size : int = 32_, _name : str = ''_, _write_csv : bool = True_, _truncate_dim : int | None = None_, _score_functions : dict[str, Callable[[[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\")], [Tensor](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\")]] | None = None_, _main_score_function : str | [SimilarityFunction](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.similarity_functions.SimilarityFunction") | None = None_, _query_prompt : str | None = None_, _query_prompt_name : str | None = None_, _corpus_prompt : str | None = None_, _corpus_prompt_name : str | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\InformationRetrievalEvaluator.py#L23-L500)ïƒ•

This class evaluates an Information Retrieval (IR) setting.

Given a set of queries and a large corpus set. It will retrieve for each query
the top-k most similar document. It measures Mean Reciprocal Rank (MRR),
Recall@k, and Normalized Discounted Cumulative Gain (NDCG)

Example

```
import random
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import InformationRetrievalEvaluator
from datasets import load_dataset

# Load a model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Load the Touche-2020 IR dataset (https://huggingface.co/datasets/BeIR/webis-touche2020, https://huggingface.co/datasets/BeIR/webis-touche2020-qrels)
corpus = load_dataset("BeIR/webis-touche2020", "corpus", split="corpus")
queries = load_dataset("BeIR/webis-touche2020", "queries", split="queries")
relevant_docs_data = load_dataset("BeIR/webis-touche2020-qrels", split="test")

# For this dataset, we want to concatenate the title and texts for the corpus
corpus = corpus.map(lambda x: {'text': x['title'] + " " + x['text']}, remove_columns=['title'])

# Shrink the corpus size heavily to only the relevant documents + 30,000 random documents
required_corpus_ids = set(map(str, relevant_docs_data["corpus-id"]))
required_corpus_ids |= set(random.sample(corpus["_id"], k=30_000))
corpus = corpus.filter(lambda x: x["_id"] in required_corpus_ids)

# Convert the datasets to dictionaries
corpus = dict(zip(corpus["_id"], corpus["text"]))  # Our corpus (cid => document)
queries = dict(zip(queries["_id"], queries["text"]))  # Our queries (qid => question)
```

```
relevant_docs = {}  # Query ID to relevant documents (qid => set([relevant_cids])
for qid, corpus_ids in zip(relevant_docs_data["query-id"], relevant_docs_data["corpus-id"]):
    qid = str(qid)
    corpus_ids = str(corpus_ids)
    if qid not in relevant_docs:
        relevant_docs[qid] = set()
    relevant_docs[qid].add(corpus_ids)

        # Given queries, a corpus and a mapping with relevant documents, the InformationRetrievalEvaluator computes different IR metrics.
ir_evaluator = InformationRetrievalEvaluator(
    queries=queries,
    corpus=corpus,
    relevant_docs=relevant_docs,
    name="BeIR-touche2020-subset-test",
)
results = ir_evaluator(model)
'''
Information Retrieval Evaluation of the model on the BeIR-touche2020-test dataset:
Queries: 49
Corpus: 31923

Score-Function: cosine
Accuracy@1: 77.55%
Accuracy@3: 93.88%
Accuracy@5: 97.96%
Accuracy@10: 100.00%
```

Precision@1: 77.55%

Precision@3: 72.11%

Precision@5: 71.43%

Precision@10: 62.65%

Recall@1: 1.72%

Recall@3: 4.78%

Recall@5: 7.90%

Recall@10: 13.86%

MRR@10: 0.8580

NDCG@10: 0.6606

MAP@100: 0.2934

'''

print(ir_evaluator.primary_metric)

=> "BeIR-touche2020-test_cosine_map@100"

print(results[ir_evaluator.primary_metric])

=> 0.29335196224364596

Initializes the InformationRetrievalEvaluator.

Parameters:

* **queries** (_Dict_ _[__str_ _,__str_ _]_) â€“ A dictionary mapping query IDs to queries.

* **corpus** (_Dict_ _[__str_ _,__str_ _]_) â€“ A dictionary mapping document IDs to documents.

* **relevant_docs** (_Dict_ _[__str_ _,__Set_ _[__str_ _]__]_) â€“ A dictionary mapping query IDs to a set of relevant document IDs.

* **corpus_chunk_size** (_int_) â€“ The size of each chunk of the corpus. Defaults to 50000.

* **mrr_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for MRR calculation. Defaults to [10].

* **ndcg_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for NDCG calculation. Defaults to [10].

* **accuracy_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for accuracy calculation. Defaults to [1, 3, 5, 10].

* **precision_recall_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for precision and recall calculation. Defaults to [1, 3, 5, 10].

* **map_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for MAP calculation. Defaults to [100].

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

* **batch_size** (_int_) â€“ The batch size for evaluation. Defaults to 32.

* **name** (_str_) â€“ A name for the evaluation. Defaults to â€œâ€•.

* **write_csv** (_bool_) â€“ Whether to write the evaluation results to a CSV file. Defaults to True.

* **truncate_dim** (_int_ _,__optional_) â€“ The dimension to truncate the embeddings to. Defaults to None.

* **score_functions** (_Dict_ _[__str_ _,__Callable_ _[__[__Tensor_ _,__Tensor_ _]__,__Tensor_ _]__]_) â€“ A dictionary mapping score function names to score functions. Defaults to the `similarity` function from the `model`.

* **main_score_function** (_Union_ _[__str_ _,_[_SimilarityFunction_](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction") _]__,__optional_) â€“ The main score function to use for evaluation. Defaults to None.

* **query_prompt** (_str_ _,__optional_) â€“ The prompt to be used when encoding the corpus. Defaults to None.

* **query_prompt_name** (_str_ _,__optional_) â€“ The name of the prompt to be used when encoding the corpus. Defaults to None.

* **corpus_prompt** (_str_ _,__optional_) â€“ The prompt to be used when encoding the corpus. Defaults to None.

* **corpus_prompt_name** (_str_ _,__optional_) â€“ The name of the prompt to be used when encoding the corpus. Defaults to None.

NanoBEIREvaluatorïƒ•

_class _sentence_transformers.evaluation.NanoBEIREvaluator(_dataset_names: list[~typing.Literal['climatefever', 'dbpedia', 'fever', 'fiqa2018', 'hotpotqa', 'msmarco', 'nfcorpus', 'nq', 'quoraretrieval', 'scidocs', 'arguana', 'scifact', 'touche2020']] | None = None, mrr_at_k: list[int] = [10], ndcg_at_k: list[int] = [10], accuracy_at_k: list[int] = [1, 3, 5, 10], precision_recall_at_k: list[int] = [1, 3, 5, 10], map_at_k: list[int] = [100], show_progress_bar: bool = False, batch_size: int = 32, write_csv: bool = True, truncate_dim: int | None = None, score_functions: dict[str, ~typing.Callable[[~torch.Tensor, ~torch.Tensor], ~torch.Tensor]] | None = None, main_score_function: str | ~sentence_transformers.similarity_functions.SimilarityFunction | None = None, aggregate_fn: ~typing.Callable[[list[float]], float] = <function mean>, aggregate_key: str = 'mean', query_prompts: str | dict[str, str] | None = None, corpus_prompts: str | dict[str, str] | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\NanoBEIREvaluator.py#L72-L452)ïƒ•

This class evaluates the performance of a SentenceTransformer Model on the NanoBEIR collection of datasets.

The collection is a set of datasets based on the BEIR collection, but with a significantly smaller size, so it can be used for quickly evaluating the retrieval performance of a model before commiting to a full evaluation. The datasets are available on HuggingFace at <https://huggingface.co/collections/zeta-alpha-ai/nanobeir-66e1a0af21dfd93e620cd9f6> The Evaluator will return the same metrics as the InformationRetrievalEvaluator (i.e., MRR, nDCG, Recall@k), for each dataset and on average.

Example

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import NanoBEIREvaluator

model = SentenceTransformer('intfloat/multilingual-e5-large-instruct')

datasets = ["QuoraRetrieval", "MSMARCO"]
query_prompts = {
        "QuoraRetrieval": "Instruct: Given a question, retrieve questions that are semantically equivalent to the given question\nQuery: ",
    "MSMARCO": "Instruct: Given a web search query, retrieve relevant passages that answer the query\nQuery: "
}

evaluator = NanoBEIREvaluator(
    dataset_names=datasets,
    query_prompts=query_prompts,
)

results = evaluator(model)
'''
NanoBEIR Evaluation of the model on ['QuoraRetrieval', 'MSMARCO'] dataset:
Evaluating NanoQuoraRetrieval
```

Information Retrieval Evaluation of the model on the NanoQuoraRetrieval dataset:

Queries: 50

Corpus: 5046

Score-Function: cosine

Accuracy@1: 92.00%

Accuracy@3: 98.00%

Accuracy@5: 100.00%

Accuracy@10: 100.00%

Precision@1: 92.00%

Precision@3: 40.67%

Precision@5: 26.00%

Precision@10: 14.00%

Recall@1: 81.73%

Recall@3: 94.20%

Recall@5: 97.93%

Recall@10: 100.00%

MRR@10: 0.9540

NDCG@10: 0.9597

MAP@100: 0.9395

Evaluating NanoMSMARCO

Information Retrieval Evaluation of the model on the NanoMSMARCO dataset:

Queries: 50

Corpus: 5043

Score-Function: cosine

Accuracy@1: 40.00%

Accuracy@3: 74.00%

Accuracy@5: 78.00%

Accuracy@10: 88.00%

Precision@1: 40.00%

Precision@3: 24.67%

Precision@5: 15.60%

Precision@10: 8.80%

Recall@1: 40.00%

Recall@3: 74.00%

Recall@5: 78.00%

Recall@10: 88.00%

MRR@10: 0.5849

NDCG@10: 0.6572

MAP@100: 0.5892

Average Queries: 50.0

Average Corpus: 5044.5

Aggregated for Score Function: cosine

Accuracy@1: 66.00%

Accuracy@3: 86.00%

Accuracy@5: 89.00%

Accuracy@10: 94.00%

Precision@1: 66.00%

Recall@1: 60.87%

Precision@3: 32.67%

Recall@3: 84.10%

Precision@5: 20.80%

Recall@5: 87.97%

Precision@10: 11.40%

Recall@10: 94.00%

MRR@10: 0.7694

NDCG@10: 0.8085

'''

print(evaluator.primary_metric)

=> "NanoBEIR_mean_cosine_ndcg@10"

print(results[evaluator.primary_metric])

=> 0.8084508771660436

Initializes the NanoBEIREvaluator.

Parameters:

* **dataset_names** (_List_ _[__str_ _]_) â€“ The names of the datasets to evaluate on.

* **mrr_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for MRR calculation. Defaults to [10].

* **ndcg_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for NDCG calculation. Defaults to [10].

* **accuracy_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for accuracy calculation. Defaults to [1, 3, 5, 10].

* **precision_recall_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for precision and recall calculation. Defaults to [1, 3, 5, 10].

* **map_at_k** (_List_ _[__int_ _]_) â€“ A list of integers representing the values of k for MAP calculation. Defaults to [100].

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

* **batch_size** (_int_) â€“ The batch size for evaluation. Defaults to 32.

* **write_csv** (_bool_) â€“ Whether to write the evaluation results to a CSV file. Defaults to True.

* **truncate_dim** (_int_ _,__optional_) â€“ The dimension to truncate the embeddings to. Defaults to None.

* **score_functions** (_Dict_ _[__str_ _,__Callable_ _[__[__Tensor_ _,__Tensor_ _]__,__Tensor_ _]__]_) â€“ A dictionary mapping score function names to score functions. Defaults to {SimilarityFunction.COSINE.value: cos_sim, SimilarityFunction.DOT_PRODUCT.value: dot_score}.

* **main_score_function** (_Union_ _[__str_ _,_[_SimilarityFunction_](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction") _]__,__optional_) â€“ The main score function to use for evaluation. Defaults to None.

* **aggregate_fn** (_Callable_ _[__[__list_ _[__float_ _]__]__,__float_ _]_) â€“ The function to aggregate the scores. Defaults to np.mean.

* **aggregate_key** (_str_) â€“ The key to use for the aggregated score. Defaults to â€œmeanâ€•.

* **query_prompts** (_str_ _|__dict_ _[__str_ _,__str_ _]__,__optional_) â€“ The prompts to add to the queries. If a string, will add the same prompt to all queries. If a dict, expects that all datasets in dataset_names are keys.

* **corpus_prompts** (_str_ _|__dict_ _[__str_ _,__str_ _]__,__optional_) â€“ The prompts to add to the corpus. If a string, will add the same prompt to all corpus. If a dict, expects that all datasets in dataset_names are keys.

MSEEvaluatorïƒ•

_class _sentence_transformers.evaluation.MSEEvaluator(_source_sentences : list[str]_, _target_sentences : list[str]_, _teacher_model =None_, _show_progress_bar : bool = False_, _batch_size : int = 32_, _name : str = ''_, _write_csv : bool = True_, _truncate_dim : int | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\MSEEvaluator.py#L17-L146)ïƒ•

Computes the mean squared error (x100) between the computed sentence embedding
and some target sentence embedding.

The MSE is computed between ||teacher.encode(source_sentences) -

student.encode(target_sentences)||.

For multilingual knowledge distillation (<https://arxiv.org/abs/2004.09813>),
source_sentences are in English and target_sentences are in a different
language like German, Chinese, Spanishâ€¦

Parameters:

* **source_sentences** (_List_ _[__str_ _]_) â€“ Source sentences to embed with the teacher model.

* **target_sentences** (_List_ _[__str_ _]_) â€“ Target sentences to embed with the student model.

* **teacher_model** ([_SentenceTransformer_](SentenceTransformer.html#sentence_transformers.SentenceTransformer "sentence_transformers.SentenceTransformer") _,__optional_) â€“ The teacher model to compute the source sentence embeddings.

* **show_progress_bar** (_bool_ _,__optional_) â€“ Show progress bar when computing embeddings. Defaults to False.

* **batch_size** (_int_ _,__optional_) â€“ Batch size to compute sentence embeddings. Defaults to 32.

* **name** (_str_ _,__optional_) â€“ Name of the evaluator. Defaults to â€œâ€•.

* **write_csv** (_bool_ _,__optional_) â€“ Write results to CSV file. Defaults to True.

* **truncate_dim** (_int_ _,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

Example

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import MSEEvaluator
from datasets import load_dataset

# Load a model
student_model = SentenceTransformer('paraphrase-multilingual-mpnet-base-v2')
teacher_model = SentenceTransformer('all-mpnet-base-v2')

# Load any dataset with some texts
dataset = load_dataset("sentence-transformers/stsb", split="validation")
sentences = dataset["sentence1"] + dataset["sentence2"]

        # Given queries, a corpus and a mapping with relevant documents, the InformationRetrievalEvaluator computes different IR metrics.
mse_evaluator = MSEEvaluator(
    source_sentences=sentences,
    target_sentences=sentences,
```

```
    teacher_model=teacher_model,
    name="stsb-dev",
)
results = mse_evaluator(student_model)
'''
MSE evaluation (lower = better) on the stsb-dev dataset:
MSE (*100):  0.805045
'''
print(mse_evaluator.primary_metric)
# => "stsb-dev_negative_mse"
print(results[mse_evaluator.primary_metric])
# => -0.8050452917814255
```

Base class for all evaluators. Notably, this class introduces the `greater_is_better` and `primary_metric` attributes. The former is a boolean indicating whether a higher evaluation score is better, which is used for choosing the best checkpoint if `load_best_model_at_end` is set to `True` in the training arguments.

The latter is a string indicating the primary metric for the evaluator. This has to be defined whenever the evaluator returns a dictionary of metrics, and the primary metric is the key pointing to the primary metric, i.e. the one that is used for model selection and/or logging.

ParaphraseMiningEvaluatorï*ƒ•*

_class _sentence_transformers.evaluation.ParaphraseMiningEvaluator(_sentences_map : dict[str, str]_, _duplicates_list : list[tuple[str, str]] | None = None_, _duplicates_dict : dict[str, dict[str, bool]] | None = None_, _add_transitive_closure : bool = False_, _query_chunk_size : int = 5000_, _corpus_chunk_size : int = 100000_, _max_pairs : int = 500000_, _top_k : int = 100_, _show_progress_bar : bool = False_, _batch_size : int = 16_, _name : str = ''_, _write_csv : bool = True_, _truncate_dim : int | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\ParaphraseMiningEvaluator.py#L19-L272)ïƒ•

Given a large set of sentences, this evaluator performs paraphrase (duplicate)
mining and identifies the pairs with the highest similarity. It compare the
extracted paraphrase pairs with a set of gold labels and computes the F1
score.

Example

```
from datasets import load_dataset
from sentence_transformers.SentenceTransformer import SentenceTransformer
from sentence_transformers.evaluation import ParaphraseMiningEvaluator

# Load a model
model = SentenceTransformer('all-mpnet-base-v2')

# Load the Quora Duplicates Mining dataset
```

```
    questions_dataset = load_dataset("sentence-transformers/quora-duplicates-mining", "questions", split="dev")
    duplicates_dataset = load_dataset("sentence-transformers/quora-duplicates-mining", "duplicates", split="dev")

    # Create a mapping from qid to question & a list of duplicates (qid1, qid2)
    qid_to_questions = dict(zip(questions_dataset["qid"], questions_dataset["question"]))
    duplicates = list(zip(duplicates_dataset["qid1"], duplicates_dataset["qid2"]))

    # Initialize the paraphrase mining evaluator
    paraphrase_mining_evaluator = ParaphraseMiningEvaluator(
        sentences_map=qid_to_questions,
        duplicates_list=duplicates,
        name="quora-duplicates-dev",
    )
    results = paraphrase_mining_evaluator(model)
    '''
    Paraphrase Mining Evaluation of the model on the quora-duplicates-dev dataset:
    Number of candidate pairs: 250564
    Average Precision: 56.51
    Optimal threshold: 0.8325
    Precision: 52.76
    Recall: 59.19
    F1: 55.79
    '''
    print(paraphrase_mining_evaluator.primary_metric)
    # => "quora-duplicates-dev_average_precision"
```

print(results[paraphrase_mining_evaluator.primary_metric])

=> 0.5650940787776353

Initializes the ParaphraseMiningEvaluator.

Parameters:

* **sentences_map** (_Dict_ _[__str_ _,__str_ _]_) â€“ A dictionary that maps sentence-ids to sentences. For example, sentences_map[id] => sentence.

* **duplicates_list** (_List_ _[__Tuple_ _[__str_ _,__str_ _]__]__,__optional_) â€“ A list with id pairs [(id1, id2), (id1, id5)] that identifies the duplicates / paraphrases in the sentences_map. Defaults to None.

* **duplicates_dict** (_Dict_ _[__str_ _,__Dict_ _[__str_ _,__bool_ _]__]__,__optional_) â€“ A default dictionary mapping [id1][id2] to true if id1 and id2 are duplicates. Must be symmetric, i.e., if [id1][id2] => True, then [id2][id1] => True. Defaults to None.

* **add_transitive_closure** (_bool_ _,__optional_) â€“ If true, it adds a transitive closure, i.e. if dup[a][b] and dup[b][c], then dup[a][c]. Defaults to False.

* **query_chunk_size** (_int_ _,__optional_) â€“ To identify the paraphrases, the cosine-similarity between all sentence-pairs will be computed. As this might require a lot of memory, we perform a batched computation. query_chunk_size sentences will be compared against up to

corpus_chunk_size sentences. In the default setting, 5000 sentences will be grouped together and compared up-to against 100k other sentences. Defaults to 5000.

* **corpus_chunk_size** (_int_ _,__optional_) â€“ The corpus will be batched, to reduce the memory requirement. Defaults to 100000.

* **max_pairs** (_int_ _,__optional_) â€“ We will only extract up to max_pairs potential paraphrase candidates. Defaults to 500000.

* **top_k** (_int_ _,__optional_) â€“ For each query, we extract the top_k most similar pairs and add it to a sorted list. I.e., for one sentence we cannot find more than top_k paraphrases. Defaults to 100.

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

* **batch_size** (_int_ _,__optional_) â€“ Batch size for computing sentence embeddings. Defaults to 16.

* **name** (_str_ _,__optional_) â€“ Name of the experiment. Defaults to â€œâ€•.

* **write_csv** (_bool_ _,__optional_) â€“ Write results to CSV file. Defaults to True.

* **truncate_dim** (_Optional_ _[__int_ _]__,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

RerankingEvaluatorï*ƒ•*

_class _sentence_transformers.evaluation.RerankingEvaluator(_samples, at_k: int = 10, name: str = '', write_csv: bool = True, similarity_fct: ~typing.Callable[[~torch.Tensor, ~torch.Tensor], ~torch.Tensor] = <function cos_sim>, batch_size: int = 64, show_progress_bar: bool = False, use_batched_encoding: bool = True, truncate_dim: int | None = None, mrr_at_k: int | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\RerankingEvaluator.py#L23-L305)ïƒ•

This class evaluates a SentenceTransformer model for the task of re-ranking.

Given a query and a list of documents, it computes the score [query, doc_i]
for all possible documents and sorts them in decreasing order. Then,
[MRR@10](/cdn-cgi/l/email-protection#d19c8383f7f2e2e6eaf7f2e4e3eaf7f2e5e9eae0e1), [NDCG@10](/cdn-cgi/l/email-protection#a9e7edeaee8f8a9a9e928f8a9c9b928f8a9d91929899) and MAP
is compute to measure the quality of the ranking.

Parameters:

* **samples** (_list_) â€“ A list of dictionaries, where each dictionary represents a sample and has the following keys: \- â€˜queryâ€™: The search query. \- â€˜positiveâ€™: A list of positive (relevant) documents. \- â€˜negativeâ€™: A list of negative (irrelevant) documents.

* **at_k** (_int_ _,__optional_) â€“ Only consider the top k most similar documents to each query for the evaluation. Defaults to 10.

* **name** (_str_ _,__optional_) â€“ Name of the evaluator. Defaults to â€œâ€•.

* **write_csv** (_bool_ _,__optional_) â€“ Write results to CSV file. Defaults to True.

* **similarity_fct** (_Callable_ _[__[_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _,_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _]__,_[_torch.Tensor_](https://pytorch.org/docs/stable/tensors.html#torch.Tensor "\(in PyTorch v2.5\)") _]__,__optional_) â€“ Similarity function between sentence embeddings. By default, cosine similarity. Defaults to cos_sim.

* **batch_size** (_int_ _,__optional_) â€“ Batch size to compute sentence embeddings. Defaults to 64.

* **show_progress_bar** (_bool_ _,__optional_) â€“ Show progress bar when computing embeddings. Defaults to False.

* **use_batched_encoding** (_bool_ _,__optional_) â€“ Whether or not to encode queries and documents in batches for greater speed, or 1-by-1 to save memory. Defaults to True.

* **truncate_dim** (_Optional_ _[__int_ _]__,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

* **mrr_at_k** (_Optional_ _[__int_ _]__,__optional_) â€“ Deprecated parameter. Please use at_k instead. Defaults to None.

Base class for all evaluators. Notably, this class introduces the
`greater_is_better` and `primary_metric` attributes. The former is a boolean
indicating whether a higher evaluation score is better, which is used for
choosing the best checkpoint if `load_best_model_at_end` is set to `True` in
the training arguments.

The latter is a string indicating the primary metric for the evaluator. This
has to be defined whenever the evaluator returns a dictionary of metrics, and
the primary metric is the key pointing to the primary metric, i.e. the one
that is used for model selection and/or logging.

SentenceEvaluatorïƒ•

_class
_sentence_transformers.evaluation.SentenceEvaluator[[source]](https://github.com/UKPLab/senten
ce-
transformers/blob/master/sentence_transformers\\evaluation\\SentenceEvaluator.py#L10-L85)ïƒ•

Base class for all evaluators

Extend this class and implement __call__ for custom evaluators.

Base class for all evaluators. Notably, this class introduces the
`greater_is_better` and `primary_metric` attributes. The former is a boolean
indicating whether a higher evaluation score is better, which is used for

choosing the best checkpoint if `load_best_model_at_end` is set to `True` in
the training arguments.

The latter is a string indicating the primary metric for the evaluator. This
has to be defined whenever the evaluator returns a dictionary of metrics, and
the primary metric is the key pointing to the primary metric, i.e. the one
that is used for model selection and/or logging.

SequentialEvaluatorïƒ•

_class _sentence_transformers.evaluation.SequentialEvaluator(_evaluators:
~collections.abc.Iterable[~sentence_transformers.evaluation.SentenceEvaluator.SentenceEvaluator
],
main_score_function=<function
SequentialEvaluator.<lambda>>_)[[source]](https://github.com/UKPLab/sentence-
transformers/blob/master/sentence_transformers\\evaluation\\SequentialEvaluator.py#L12-L64)ïƒ•

This evaluator allows that multiple sub-evaluators are passed. When the model
is evaluated, the data is passed sequentially to all sub-evaluators.

All scores are passed to â€˜main_score_functionâ€™, which derives one final
score value

Initializes a SequentialEvaluator object.

Parameters:

* **evaluators** (_Iterable_ _[__SentenceEvaluator_ _]_) â€“ A collection of SentenceEvaluator objects.

* **main_score_function** (_function_ _,__optional_) â€“ A function that takes a list of scores and returns the main score. Defaults to selecting the last score in the list.

Example

```
evaluator1 = BinaryClassificationEvaluator(...)
evaluator2 = InformationRetrievalEvaluator(...)
evaluator3 = MSEEvaluator(...)
seq_evaluator = SequentialEvaluator([evaluator1, evaluator2, evaluator3])
```

TranslationEvaluatorïƒ•

_class _sentence_transformers.evaluation.TranslationEvaluator(_source_sentences : list[str]_, _target_sentences : list[str]_, _show_progress_bar : bool = False_, _batch_size : int = 16_, _name : str = ''_, _print_wrong_matches : bool = False_, _write_csv : bool = True_, _truncate_dim : int | None = None_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transfor

mers\\evaluation\\TranslationEvaluator.py#L21-L187)ïƒ•

Given two sets of sentences in different languages, e.g. (en_1, en_2, en_3â€¦)
and (fr_1, fr_2, fr_3, â€¦), and assuming that fr_i is the translation of
en_i. Checks if vec(en_i) has the highest similarity to vec(fr_i). Computes
the accuracy in both directions

Example

```
    from sentence_transformers import SentenceTransformer
    from sentence_transformers.evaluation import TranslationEvaluator
    from datasets import load_dataset

    # Load a model
    model = SentenceTransformer('paraphrase-multilingual-mpnet-base-v2')

    # Load a parallel sentences dataset
       dataset  =  load_dataset("sentence-transformers/parallel-sentences-news-commentary",  "en-nl",
split="train[:1000]")

    # Initialize the TranslationEvaluator using the same texts from two languages
    translation_evaluator = TranslationEvaluator(
        source_sentences=dataset["english"],
        target_sentences=dataset["non_english"],
```

```
    name="news-commentary-en-nl",
)
results = translation_evaluator(model)
'''
Evaluating translation matching Accuracy of the model on the news-commentary-en-nl dataset:
Accuracy src2trg: 90.80
Accuracy trg2src: 90.40
'''
print(translation_evaluator.primary_metric)
# => "news-commentary-en-nl_mean_accuracy"
print(results[translation_evaluator.primary_metric])
# => 0.906
```

Constructs an evaluator based for the dataset

The labels need to indicate the similarity between the sentences.

Parameters:

* **source_sentences** (_List_ _[__str_ _]_) â€“ List of sentences in the source language.

* **target_sentences** (_List_ _[__str_ _]_) â€“ List of sentences in the target language.

* **show_progress_bar** (_bool_) â€“ Whether to show a progress bar when computing

embeddings. Defaults to False.

* **batch_size** (_int_) â€“ The batch size to compute sentence embeddings. Defaults to 16.

* **name** (_str_) â€“ The name of the evaluator. Defaults to an empty string.

* **print_wrong_matches** (_bool_) â€“ Whether to print incorrect matches. Defaults to False.

* **write_csv** (_bool_) â€“ Whether to write the evaluation results to a CSV file. Defaults to True.

* **truncate_dim** (_int_ _,__optional_) â€“ The dimension to truncate sentence embeddings to. If None, the modelâ€™s current truncation dimension will be used. Defaults to None.

TripletEvaluatorïƒ•

_class _sentence_transformers.evaluation.TripletEvaluator(_anchors : list[str]_, _positives : list[str]_, _negatives : list[str]_, _main_similarity_function : str | [SimilarityFunction](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.similarity_functions.SimilarityFunction") | None = None_, _margin : float | dict[str, float] | None = None_, _name : str = ''_, _batch_size : int = 16_, _show_progress_bar : bool = False_, _write_csv : bool = True_, _truncate_dim : int | None = None_, _similarity_fn_names : list[Literal['cosine', 'dot', 'euclidean', 'manhattan']] | None = None_, _main_distance_function : str | [SimilarityFunction](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.similarity_functions.SimilarityFunction") | None = 'deprecated'_)[[source]](https://github.com/UKPLab/sentence-transformers/blob/master/sentence_transformers\\evaluation\\TripletEvaluator.py#L25-L265)ïƒ•

Evaluate a model based on a triplet: (sentence, positive_example, negative_example). Checks if `similarity(sentence, positive_example) < similarity(sentence, negative_example) + margin`.

Example

```
from sentence_transformers import SentenceTransformer
from sentence_transformers.evaluation import TripletEvaluator
from datasets import load_dataset

# Load a model
model = SentenceTransformer('all-mpnet-base-v2')

# Load a dataset with (anchor, positive, negative) triplets
dataset = load_dataset("sentence-transformers/all-nli", "triplet", split="dev")

# Initialize the TripletEvaluator using anchors, positives, and negatives
triplet_evaluator = TripletEvaluator(
    anchors=dataset[:1000]["anchor"],
    positives=dataset[:1000]["positive"],
    negatives=dataset[:1000]["negative"],
    name="all_nli_dev",
)
results = triplet_evaluator(model)
```

```
'''
TripletEvaluator: Evaluating the model on the all-nli-dev dataset:
Accuracy Cosine Similarity:        95.60%
'''
print(triplet_evaluator.primary_metric)
# => "all_nli_dev_cosine_accuracy"
print(results[triplet_evaluator.primary_metric])
# => 0.956
```

Initializes a TripletEvaluator object.

Parameters:

* **anchors** (_List_ _[__str_ _]_) â€“ Sentences to check similarity to. (e.g. a query)

* **positives** (_List_ _[__str_ _]_) â€“ List of positive sentences

* **negatives** (_List_ _[__str_ _]_) â€“ List of negative sentences

* **main_similarity_function** (_Union_ _[__str_ _,_[_SimilarityFunction_](SentenceTransformer.html#sentence_transformers.SimilarityFunction "sentence_transformers.SimilarityFunction") _]__,__optional_) â€“ The similarity function to use. If not specified, use cosine similarity, dot product, Euclidean, and Manhattan similarity. Defaults to None.

* **margin** (_Union_ _[__float_ _,__Dict_ _[__str_ _,__float_ _]__]__,__optional_) â€“ Margins for various similarity metrics. If a float is provided, it will be used as the margin for all similarity metrics. If a dictionary is provided, the keys should be â€˜cosineâ€™, â€˜dotâ€™, â€˜manhattanâ€™, and â€˜euclideanâ€™. The value specifies the minimum margin by which the negative sample should be further from the anchor than the positive sample. Defaults to None.

* **name** (_str_) â€“ Name for the output. Defaults to â€œâ€•.

* **batch_size** (_int_) â€“ Batch size used to compute embeddings. Defaults to 16.

* **show_progress_bar** (_bool_) â€“ If true, prints a progress bar. Defaults to False.

* **write_csv** (_bool_) â€“ Write results to a CSV file. Defaults to True.

* **truncate_dim** (_int_ _,__optional_) â€“ The dimension to truncate sentence embeddings to. None uses the modelâ€™s current truncation dimension. Defaults to None.

* **similarity_fn_names** (_List_ _[__str_ _]__,__optional_) â€“ List of similarity function names to evaluate. If not specified, evaluate using the `model.similarity_fn_name`. Defaults to None.

[Previous](sampler.html "Samplers") [Next](datasets.html "Datasets")

* * *

Built with [Sphinx](https://www.sphinx-doc.org/) using a

[theme](https://github.com/readthedocs/sphinx_rtd_theme) provided by [Read the

Docs](https://readthedocs.org).