Vietnam National University, Ho Chi Minh City
University of Information Technology
Falcuty of Computer Science

Final Project Report

One Embedder, Any Task: Instruction-Finetuned Text Embeddings

Instructor: Dr. Quy Nguyen Thi
Class: CS222.O11.KHTN

Student name	*Student ID*
Tran Hoang Bao Ly	21521109
Le Thi Lien	21522282
Le Thanh Minh	21520063
Huynh Pham Duc Lam	21521050

Ho Chi Minh City, December 2023

Contents

1 Introduction

1.1 Abstraction

We present INSTRUCTOR, a novel approach for generating text embeddings based on task instructions. In this method, each text input is embedded along with accompanying instructions that elucidate the specific use case, such as task details and domain descriptions. Unlike previous encoders, which tend to be more task-specific, INSTRUCTOR serves as a unified embedder capable of producing text embeddings tailored to various downstream tasks and domains, requiring no additional training. Our methodology involves annotating instructions for 330 diverse tasks and training INSTRUCTOR on this multitask dataset using a contrastive loss. Evaluation of INSTRUCTOR encompasses 70 embedding evaluation tasks, including unseen tasks during training, spanning classification, information retrieval, semantic textual similarity, and text generation assessments. Despite having significantly fewer parameters than the leading model, INSTRUCTOR attains state-of-the-art performance, boasting an average improvement of 3.4% across the 70 diverse datasets compared to previous benchmarks. Our analysis indicates that INSTRUCTOR demonstrates robustness to changes in instructions, and fine-tuning based on instructions helps address the challenge of training a single model on diverse datasets.

1.2 Introduction article

One Embedder, Any Task: Instruction-Finetuned Text Embeddings [1] is the paper was approved at ACL 2023. You can find their paper here. Their code was published at github.

In this article, we will guild you performing experiment with Instructer. Include training and evaluation. Our work was published at Github. We explained this paper in presentation. You can find it here.

2 Experiment

Our experiment follow structure of paper include training and evaluation on a wide range of 35 tasks

2.1 Installation

In case you use local machine, we recommend you using virtual environment with Anaconda:

```
conda env create -n instructor python=3.7
```

If you run experiment on Google Colab or Kaggle Notebook. You can skip create virtual environment with Anaconda.

```
# remember clone from our repo instead of author's repo
git clone https://github.com/4ursmile/instructor-embedding
pip install -r requirements.txt
pip install InstructorEmbedding # for Python API
```

Now active virtual environment.

```
conda activate instructor
```

2.2 Inference

Inference phase can be easily perform with Python API. First, Downloading model from Huggingface:

```
from InstructorEmbedding import INSTRUCTOR
model = INSTRUCTOR('hkunlp/instructor-large')
```

Second, prepare data for inference. Your data must follow this structure [instruction, sentence]. For example:

```
sentences = [
    ['Represent the Medicine sentence for clustering: ','Dynamical Scalar Degree of Freedom in Horava-Lifshitz Gravity'],
    ['Represent the Medicine sentence for clustering: ','Comparison of Atmospheric Neutrino Flux Calculations at Low Energies'],
    ['Represent the Medicine sentence for clustering: ','Fermion Bags in the Massive Gross-Neveu Model'],
    ['Represent the Medicine sentence for clustering: ',"QCD corrections to Associated t-tbar-H production at the Tevatron"],
    ['Represent the Medicine sentence for clustering: ','A New Analysis of the R Measurements: Resonance Parameters of the Higher,  Vector States of Charmonium']
]
# list sentences as batch
```

The encode function will encode sentences with instructions into 768d vector embedding

```
model.encode( sentences,
              batch_size: int = 32,
```

```
            show_progress_bar: bool = None,
            output_value: str = 'sentence_embedding',
            convert_to_numpy: bool = True,
            convert_to_tensor: bool = False,
            device: str = None,
            normalize_embeddings: bool = False)
```

- sentences: The sentences to be embedded. It should be in the format of [["instruction prompt 0", "text to be embedded 0], ["instruction prompt 1", "text to be embedded 1], ...].
- batch_size (default: 32): The batch size used for the computation. It determines the number of sentences processed together in each batch.
- show_progress_bar (default: None): If set to True, it displays a progress bar while encoding sentences, providing a visual indication of the encoding progress.
- output_value (default: 'sentence_embedding'): Specifies the desired output type. The default value 'sentence_embedding' returns sentence embeddings. Setting it to 'token_embeddings' returns wordpiece token embeddings. Setting it to None returns all output values.
- convert_to_numpy (default: True): If set to True, the output is a list of numpy vectors. If set to False, the output is a list of PyTorch tensors.
- convert_to_tensor (default: False): If set to True, the function returns a stacked tensor as a single output. This parameter overrides any setting specified by convert_to_numpy.
- device (default: None): Specifies the torch.device to use for the computation. If not specified, the function uses the default device.
- normalize_embeddings (default: False): If set to True, the returned vectors will have a length of 1, indicating that they are normalized. In this case, similarity search would use the faster dot-product (util.dot_score), instead of cosine similarity.

You can use Instructor for many downstream task for example clustering task:

```
import sklearn.cluster
sentences = [['Represent the Medicine sentence for
   clustering: ','Dynamical Scalar Degree of Freedom in
   Horava-Lifshitz Gravity'],
```

```
            ['Represent the Medicine sentence for
    clustering: ','Comparison of Atmospheric Neutrino Flux
     Calculations at Low Energies'],
            ['Represent the Medicine sentence for
    clustering: ','Fermion Bags in the Massive Gross-Neveu
     Model'],
            ['Represent the Medicine sentence for
    clustering: ',"QCD corrections to Associated t-tbar-H
    production at the Tevatron"],
            ['Represent the Medicine sentence for
    clustering: ','A New Analysis of the R Measurements:
    Resonance Parameters of the Higher,  Vector States of
    Charmonium']]
embeddings = model.encode(sentences)
clustering_model = sklearn.cluster.MiniBatchKMeans(
    n_clusters=2)
clustering_model.fit(embeddings)
cluster_assignment = clustering_model.labels_
print(cluster_assignment)
```

2.3 Training

2.3.1 Training objective

The training process for INSTRUCTOR involves framing a diverse set of tasks as a text-to-text problem, specifically aimed at distinguishing between good and bad candidate outputs $y \in \{y^+, y_i^-\}$ when provided with an input x. Each training sample is represented by the tuple (x, I_x, y, I_y), where I_x and I_y denote the instructions associated with x and y, respectively. To illustrate, in the context of a retrieval task, x serves as a query, and the good/bad instances of y represent relevant/irrelevant documents from a given document collection.

2.3.2 Datasets

Fortunately, For training Instructor, Author of Instructor paper created Multitask Embeddings Data with Instructions (MEDI) so we don't need to collect create datasets by ourselves. MEDI constructed by comprising a compilation of 330 datasets sourced from Super-NI (Super-Natural Instructions) [2], sentence-transformer embedding training data, KILT [3], and MedMCQA [4], Their dataset encompasses a diverse array of domains and tasks. In cases where positive and negative pairs are not initially available, they systematically generate and organize them into a standardized format:

```
[
    {'query': ['Represent the Wikipedia question for
    retrieving relevant documents;', 'big little lies
```

season 2 how many episodes’], ’pos’: [’Represent the Wikipedia document for retrieval;’, ’Big Little Lies (TV series) series garnered several accolades. It received 16 Emmy Award nominations and won eight, including Outstanding Limited Series and acting awards for Kidman, Skarsg rd, and Dern. The trio also won Golden Globe Awards in addition to a Golden Globe Award for Best Miniseries or Television Film win for the series. Kidman and Skarsg rd also received Screen Actors Guild Awards for their performances. Despite originally being billed as a miniseries, HBO renewed the series for a second season. Production on the second season began in March 2018 and is set to premiere in 2019. All seven episodes are being written by Kelley’], ’neg’: [’Represent the Wikipedia document for retrieval;’, ’Little People, Big World final minutes of the season two-A finale, "Farm Overload". A crowd had gathered around Jacob, who was lying on the ground near the trebuchet. The first two episodes of season two-B focus on the accident, and how the local media reacted to it. The first season of "Little People, Big World" generated solid ratings for TLC (especially in the important 18 49 demographic), leading to the show\’s renewal for a second season. Critical reviews of the series have been generally positive, citing the show\’s positive portrayal of little people. Conversely, other reviews have claimed that the show has a voyeuristic bend’], ’ task_name’: ’NQ’}
3 {’query’: [’Represent the Wikipedia question for retrieving relevant documents;’, ’who sang waiting for a girl like you’], ’pos’: [’Represent the Wikipedia document for retrieval;’, ’Waiting for a Girl Like You Waiting for a Girl Like You "Waiting for a Girl Like You" is a 1981 power ballad by the British-American rock band Foreigner. The distinctive synthesizer theme was performed by the then-little-known Thomas Dolby, and this song also marked a major departure from their earlier singles because their previous singles were mid to upper tempo rock songs while this song was a softer love song with the energy of a power ballad. It was the second single released from the album "4" (1981) and was co-written by Lou Gramm and Mick Jones. It has become one of the band\’s most’], ’neg’: [’ Represent the Wikipedia document for retrieval;’, ’ Waiting for a Girl Like You held off the number 1 spot by Olivia Newton-John\’s single "Physical" for nine consecutive weeks, and then by Hall & Oates\’ "I Can\’ t Go for That (No Can Do)" for a tenth week on January 30, 1982. Because of its chart longevity, it ended up being the number 19 song on the Top 100 singles of 1982. The song was the band\’s biggest hit until "I

```
    Want to Know What Love Is" hit number 1 in 1985. The
    song lists at number 100 on ""Billboard"\'s Greatest
    Songs of All Time". Waiting for a Girl Like You "
    Waiting for a Girl'], 'task_name': 'NQ'}
]
```

Each individual instance in the dataset is comprised of a query, a positive pair, a negative pair, and a task identifier. This task identifier plays a crucial role in maintaining coherence within training batches by ensuring that data within the same batch originates from the same task. For those interested, the MEDI data is accessible for download at: here.

2.3.3 Backbone

In our approach, We build INSTRUCTOR based on single encoder architecture [2] we employ GTR models as the backbone encoder, with specific configurations such as GTR-Base for INSTRUCTOR-Base, GTR-Large for INSTRUCTOR, and GTR-XL for INSTRUCTOR-XL. These GTR models are initially initialized from T5 models, which undergo pretraining on a web corpus, followed by fine-tuning on datasets focused on information search tasks. This sequential process enables the GTR models to capture and refine information pertinent to the specified tasks, enhancing their effectiveness in subsequent applications.

For some unidentified reasons. GTR model can't download from Huggingface API. So we must download it manually. Use following command:

```
git clone https://huggingface.co/sentence-transformers/
    gtr-t5-large
#or
git clone https://huggingface.co/sentence-transformers/
    sentence-t5-large
#replace "large" with "base" or "xl" if you want.
```

2.3.4 Training

Firstly, you need to download MEDI dataset from here. We provided script below to download and setup dataset for you.

```
# install gdown
pip install gdown
#download MEDI dataset as zip
gdown 1vZ5c2oJNonGOvXzppNg5mHz24O6jcc52 -O medi-data.zip
#unzip dataset
mkdir ./data
unzip medi-data.zip -d ./data
```

```
#Create logs directory for trained model and training
   result.
mkdir logs
```

For training, The Authors provide python script for training INSTRUCTOR.

```
python train.py --model_name_or_path sentence-
   transformers/gtr-t5-large --output_dir logs --
   cache_dir data/medi-data --max_source_length 512 --
   num_train_epochs 10 --save_steps 500 --cl_temperature
   0.1 --warmup_ratio 0.1 --learning_rate 2e-5 ----
   overwrite_output_dir
```

Below is training arguments

- model_name_or_path: Is the model name or path to save pre-trained model for backbone. We have download 2 model gtr-t5-large and sentence-t5-large above.
- output_dir: The directory to store the trained models(checkpoints) for evaluation.
- cache_dir: The directory to cache downloaded models and data. The downloaded MEDI data(medi-data.json) should be put under the directory –cache_dir.
- cl_tempeture: Temperature for contrastive loss.
- All the other arguments are standard Huggingface's transformers training arguments, such as –overwrite_output_dir, –num_train_epochs, –learning_rate. For details, refer to transformers. You can playaround with it.

Change backbone model from large to base or xl to get instructor-large, instructor-base or instructor-xl.

2.4 Evaluation

In this section we will evaluate INSTRUCTOR on a wide range of 70 downstream tasks over the three benchmarks: MTEB, Billboard, and prompt retrieval.

2.4.1 MTEB

MTEB: Massive Text Embedding Benchmark [5] stands as a comprehensive evaluation benchmark designed to offer a thorough assessment of the performance of current embedding models. The primary goal is to provide a holistic perspective and uncover universal text

embeddings that demonstrate effectiveness across a diverse array of tasks. By encompassing a wide range of evaluation scenarios, MTEB aims to contribute to a deeper understanding of the capabilities and limitations of various embedding models, fostering the development of more versatile and robust text embeddings. includes 56 datasets over 7 diverse task categories. Below is detail:

```
TASK_LIST_CLASSIFICATION = [
    "AmazonCounterfactualClassification",
    "AmazonPolarityClassification",
    "AmazonReviewsClassification",
    "Banking77Classification",
    "EmotionClassification",
    "ImdbClassification",
    "MassiveIntentClassification",
    "MassiveScenarioClassification",
    "MTOPDomainClassification",
    "MTOPIntentClassification",
    "ToxicConversationsClassification",
    "TweetSentimentExtractionClassification",
]

TASK_LIST_CLUSTERING = [
    "ArxivClusteringP2P",
    "ArxivClusteringS2S",
    "BiorxivClusteringP2P",
    "BiorxivClusteringS2S",
    "MedrxivClusteringP2P",
    "MedrxivClusteringS2S",
    "RedditClustering",
    "RedditClusteringP2P",
    "StackExchangeClustering",
    "StackExchangeClusteringP2P",
    "TwentyNewsgroupsClustering",
]

TASK_LIST_PAIR_CLASSIFICATION = [
    "SprintDuplicateQuestions",
    "TwitterSemEval2015",
    "TwitterURLCorpus",
]

TASK_LIST_RERANKING = [
    "AskUbuntuDupQuestions",
    "MindSmallReranking",
    "SciDocsRR",
    "StackOverflowDupQuestions",
]

TASK_LIST_RETRIEVAL = [
    "ArguAna",
    "ClimateFEVER",
```

```
    "CQADupstackAndroidRetrieval",
    "CQADupstackEnglishRetrieval",
    "CQADupstackGamingRetrieval",
    "CQADupstackGisRetrieval",
    "CQADupstackMathematicaRetrieval",
    "CQADupstackPhysicsRetrieval",
    "CQADupstackProgrammersRetrieval",
    "CQADupstackStatsRetrieval",
    "CQADupstackTexRetrieval",
    "CQADupstackUnixRetrieval",
    "CQADupstackWebmastersRetrieval",
    "CQADupstackWordpressRetrieval",
    "DBPedia",
    "FEVER",
    "FiQA2018",
    "HotpotQA",
    "MSMARCO",
    "NFCorpus",
    "NQ",
    "QuoraRetrieval",
    "SCIDOCS",
    "SciFact",
    "Touche2020",
    "TRECCOVID",
]

TASK_LIST_STS = [
    "BIOSSES",
    "SICK-R",
    "STS12",
    "STS13",
    "STS14",
    "STS15",
    "STS16",
    "STS17",
    "STS22",
    "STSBenchmark",
    "SummEval",
]
```

Example to evaluation SentenceTransformer model.
Install MTEB:

```
pip install mteb
```

Evaluation model on 1 task:

```
from mteb import MTEB
from sentence_transformers import SentenceTransformer

# Define the sentence-transformers model name
model_name = "average_word_embeddings_komninos"

model = SentenceTransformer(model_name)
```

```
evaluation = MTEB(tasks=["Banking77Classification"])
results = evaluation.run(model, output_folder=f"results/{model_name}")
```

For simple evaluation. We added several bash scripts and evaluation scripts for setup and evaluate all tasks.

First, setup for evaluation phase. Using below command:

```
!python eval_setup.py
```

Above command will setup for and evaluation task included MTEB.

Script for MTEB evaluation:

```
python mteb_eval.py --model large --task retrieval
```

Below is arguments:

- model: choose in ['base', 'large', 'xl']. which INSTRUCTOR model you want to evaluate.
- task: choose in ['all', 'classification', 'clustering', 'pair_classification', 'reranking', 'retrieval', 'sts']. which categories you want to evaluate. default 'all' meaning all 56 tasks will be evaluated.

MTEB benchmark with save its result in './evaluation/MTEB/results'. evaluation on MTEB take large amount of time, so we have only evaluated on some tasks of each catagories.

Before go to report, we will introduce metrics on each catagories:

- In the context of retrieval, the objective is to identify the most pertinent documents from a corpus D = p1, p2...pn for a given query q. To achieve this, an embedding model is employed to convert both the query q and the documents p1...pn into fixed-sized vectors. Subsequently, the similarity between the query q and each document pi is assessed using the cosine similarity metric applied to their respective embeddings. The performance of the retrieval process is quantified using NDCG@10 (Normalized Discounted Cumulative Gain at rank position 10). This metric provides a normalized measure of the relevance of the top 10 retrieved documents, taking into account both the relevance and position of each document in the ranking. A higher NDCG@10 score indicates better performance in terms of retrieving highly relevant documents within the top 10 positions.
- In the reranking process, the objective is to arrange a list of documents according to their relevance to a given query. For a query q and a set of documents D = p1, p2...pn, the embedding model generates embeddings for both the query and the documents. Subsequently, these embeddings are utilized to establish a ranking of

the documents, determined by their cosine similarities with the query. To assess the performance of reranking, the Mean Average Precision (MAP) is employed as a standard metric. MAP calculates the average precision across various recall levels, providing a comprehensive measure of the precision-recall trade-off. In the context of reranking, a higher MAP score indicates superior performance in accurately prioritizing relevant documents throughout the entire list.

- In the realm of clustering, the primary objective is to organize similar documents into coherent groups or clusters. To achieve this, the encoder is employed to map each document in a given set into an embedding. Subsequently, the k-means clustering algorithm is applied to partition the embedded documents, creating distinct clusters based on similarities in their embeddings. To evaluate the effectiveness of the clustering process, the v-measure is utilized as a performance metric. The v-measure assesses the quality of clustering results by considering both homogeneity and completeness, providing a balanced measure that is independent of the permutations of clustering labels. A higher v-measure indicates more accurate and meaningful clustering, reflecting the ability of the algorithm to group similar documents together while avoiding mixing dissimilar ones.

- Pair classification tasks involve predicting a binary label for a pair of texts. One specific example of this task is paraphrasing identification, where the objective is to determine whether two sentences are paraphrases of each other. In the context of a given sentence pair (t1, t2), the embedding model is employed to encode each sentence, t1 and t2, individually. Subsequently, the cosine similarity between the two embeddings is utilized to make predictions about the binary label. For evaluating the performance of pair classification tasks, the average precision score is employed as a metric. This score assesses the precision of the model in correctly identifying positive instances within the pair classification task. A higher average precision score signifies better performance in accurately classifying text pairs, especially in tasks such as paraphrasing identification.

- Semantic Textual Similarity (STS) tasks involve assessing the degree of similarity between two sentences. When presented with a sentence pair (t1, t2), the embedding model is utilized to map each sentence, t1 and t2, into separate embeddings. The similarity between t1 and t2 is then gauged by measuring the cosine similarity

between their respective embeddings. For evaluating the performance of STS tasks, Spearman's rank correlation is employed as the evaluation metric. Spearman's rank correlation measures the correlation between the similarity scores produced by the model and the human judgments of similarity. A higher Spearman's rank correlation indicates a stronger alignment between the model's assessments and human perceptions of sentence similarity, thus reflecting the model's effectiveness in capturing semantic textual similarity.

- Classification is a widely employed method for assessing the effectiveness of a model. In this evaluation approach, each example in the classification dataset is processed by utilizing the embedding of the input text as features for a classifier. During training, the classifier is trained on the provided training data, while the embeddings of the sentences remain fixed or frozen. The evaluation metric is determined by reporting the classification accuracy on the test set. This accuracy metric reflects the model's ability to correctly classify instances in the test set and serves as a key indicator of its performance in the context of the specific classification task.

Table 1 and table 2 are paper's result and our result on MTEB.

Our experimental result on MTEB was approximate 0-2% with paper's result. Some task Instructor-large with 335M parameters have better result on Instructor-xl with 1.5B parameters. This result may be because the size of the model is too large, has not yet converged, or the larger model is not suitable for some tasks.

Categories	Task	Instructor-large	Instructor-xl
Retrieval	SciFact	64.30	64.60
	NFcorpus	34.10	36.00
	ArguAna	57.10	55.70
	CQADupstackWebmastersRetrieval	46.40	45.10
	FEVER	72.70	70.00
	HotpotQA	55.20	55.90
Reranking	AskUbuntuDupQuestions	64.30	65.40
	StackOverflowDupQuestions	52.20	52.50
	SciDocsRR	82.00	79.50
	MindSmallReranking	31.70	31.80
Clustering	BiorxivClusteringS2S	31.30	30.60
	MedrxivClusteringS2S	32.00	30.80
	ArxivClusteringP2P	43.20	42.50
	RedditClustering	63.70	63.40
Pair classification	SprintDuplicateQuestions	93.10	94.90
	TwitterSemEval2015	77.40	78.00
STS	STS12	76.30	75.30
	STS13	88.20	87.40
	STS14	81.90	81.90
	STS15	89.00	88.90
	STS16	85.50	85.40
	BIOSSES	84.40	84.20
	SICK-R	81.30	81.70
	STSBenchmark	86.90	86.60
Classification	Banking77Classification	78.50	82.70
	AmazonReviewsClassification	47.90	43.00
	EmotionClassification	52.70	53.20
	AmazonPolarityClassification	91.50	86.50

Table 1: Paper's result on MTEB

Categories	Task	Instructor-large	Instructor-xl
Retrieval	SciFact	63.59	63.71
	NFcorpus	33.72	35.39
	ArguAna	56.93	55.36
	CQADupstackWebmastersRetrieval	45.60	44.53
	FEVER	72.34	69.23
	HotpotQA	54.80	55.35
Reranking	AskUbuntuDupQuestions	63.75	64.59
	StackOverflowDupQuestions	52.10	51.97
	SciDocsRR	81.22	78.56
	MindSmallReranking	31.62	31.76
Clustering	BiorxivClusteringS2S	30.75	29.79
	MedrxivClusteringS2S	31.53	30.32
	ArxivClusteringP2P	42.84	41.85
	RedditClustering	62.92	63.25
Pair classification	SprintDuplicateQuestions	92.89	94.87
	TwitterSemEval2015	76.80	77.38
STS	STS12	75.39	74.95
	STS13	87.37	86.54
	STS14	81.21	81.35
	STS15	88.43	88.61
	STS16	84.63	85.25
	BIOSSES	83.60	83.26
	SICK-R	81.06	80.87
	STSBenchmark	86.63	86.40
Classification	Banking77Classification	77.73	82.31
	AmazonReviewsClassification	47.84	42.36
	EmotionClassification	52.46	52.65
	AmazonPolarityClassification	91.12	85.71

Table 2: Our result on MTEB.

2.4.2 Automatic Evaluation for Generation

Just as the MTEB employs summarization evaluation, the Billboard benchmark [6] is utilized to extend the application of INSTRUCTOR for automatic evaluations in three additional text generation tasks: MSCOCO image captioning [6], CNN/DailyMail news summarization [7], and WMT21 Chinese-to-English translation [8]. In this evaluation process, the cosine similarity between the generated text and each reference text is calculated, and the maximum similarity score across all available references is considered. The performance of all embedding models is then assessed by measuring the Pearson correlation with human judgments, providing a quantitative measure of how well the generated text aligns with human-perceived quality across these diverse text generation tasks.

To evaluate on Billboard benchmark just simply run below command:

```
python billboard.py --model large
```

Below is arguments:

- model: choose in ['base', 'large', 'xl']. which INSTRUCTOR model you want to evaluate.

This script will print benchmark result on console.
Table 3 and table 4 are paper's result and our result on Billboard.

Categories	Task	Instructor-large	Instructor-xl
Billboard	mscoco	41.60	39.70
	cnn summary	30.30	31.90
	machine translation	38.90	30.60

Table 3: Paper's result on Billboard benchmark.

Categories	Task	Instructor-large	Instructor-xl
Billboard	mscoco	41.57	39.55
	cnn summary	30.28	31.86
	machine translation	38.84	30.55

Table 4: Our result on Billboard benchmark.

Our experimental result on Billboard benchmark [6] was approximate with paper's result. Some task Instructor-large with 335M parameters have better result on Instructor-xl with 1.5B parameters. This result may be because the size of the model is too large, has not yet converged, or the larger model is not suitable for some tasks.

2.4.3 Promt retrieval

Su el at.(2022)[9] Present the prompt retrieval task, where the objective is to retrieve a small set of in-context learning examples (demonstrations) from annotated examples given a test instance. The embedding model is employed to encode all annotated examples and identify the most similar examples to the test instance based on cosine similarity. Following the methodology outlined by [9], the retrieved examples are then utilized for in-context learning on GPT-J [**eleutherai/gpt-j-6b**] across 11 diverse downstream tasks. These tasks include classification, multiple choice, and text-to-SQL, among others, and are distinct from those included in MEDI, thus representing zero-shot settings. To compare various embedding methods, the average performance on these downstream tasks is measured using the metrics introduced earlier for each task. This comparative analysis provides insights into the effectiveness of different embedding methods in the context of in-context learning for a range of diverse tasks.]

To evaluate on Promt retrieval just simply run below command:

```
python promt_retrieval.py --model large
```

Below is arguments:

- model: choose in ['base', 'large', 'xl']. which INSTRUCTOR model you want to evaluate.

Promt retrieval benchmark with save its result in './evaluation/MTEB/ouputs/result_summary.json'. evaluation on Promt retrieval take large amount of time and disk space for GPT-J model, so we have only evaluated on some tasks of each catagories.

Table 5 and table 6 are paper's result and our result on Promt retrieval.

Categories	Task	Instructor-large	Instructor-xl
Promt retrieval	RTE	58.80	59.30
	SST-5	53.80	60.10
	Amazon	38.00	48.00
	Dbpedia_14	93.00	94.00

Table 5: Paper's result Promt retrieval.

Our experimental result on MTEB was approximate 0-2% with paper's result.

Categories	Task	Instructor-large	Instructor-xl
Promt retrieval	RTE	57.81	59.03
	SST-5	53.38	59.71
	Amazon	37.92	47.21
	Dbpedia_14	92.50	93.34

Table 6: Our result Promt retrieval.

3 Conclusion

In this article, we conducted experiments utilizing INSTRUCTOR, a novel approach for generating text embeddings based on task instructions, in conjunction with the MEDI dataset. Our findings align with the results presented in the experimental section of the paper titled "One Embedder, Any Task: Instruction-Finetuned Text Embeddings" by Su et al. (2023) [1]. Through a comprehensive series of experiments, we observed that INSTRUCTOR consistently outperformed existing methods, demonstrating slightly better performance and achieving a state-of-the-art status on various text embedding benchmarks. These results substantiate the efficacy of INSTRUCTOR in producing text embeddings tailored to specific tasks through the integration of task instructions.

While INSTRUCTOR brings about a significant improvement in baseline GTR performance, a limitation in our study was the constraint on computation resources, allowing us to use only four negative examples during the model finetuning process. It's worth noting that negative examples are recognized for their pivotal role in contrastive learning, as highlighted by Robinson et al. (2021)[10]. We anticipate that future research endeavors will address this constraint by scaling up the number of negatives used during finetuning and exploring diverse methods for effectively mining hard negatives.

Moreover, due to computational limitations, we were unable to extend multitask instruction finetuning to GTR-XXL, a model with 4.8 billion parameters. This presents an avenue for future exploration, and we hope that with increased computational resources, researchers can delve into the application of multitask instruction finetuning to larger models, such as GTR-XXL, to further understand its impact and potential improvements.

References

[1] Hongjin Su et al. *One Embedder, Any Task: Instruction-Finetuned Text Embeddings*. 2023. arXiv: 2212.09741 [cs.CL].

[2] Jianmo Ni et al. *Sentence-T5: Scalable Sentence Encoders from Pre-trained Text-to-Text Models*. 2021. arXiv: 2108.08877 [cs.CL].

[3] Fabio Petroni et al. *KILT: a Benchmark for Knowledge Intensive Language Tasks*. 2021. arXiv: 2009.02252 [cs.CL].

[4] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. "MedMCQA: A Large-scale Multi-Subject Multi-Choice Dataset for Medical domain Question Answering". In: *Proceedings of the Conference on Health, Inference, and Learning*. Ed. by Gerardo Flores et al. Vol. 174. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 248–260. URL: https://proceedings.mlr.press/v174/pal22a.html.

[5] mteb. *mteb (Massive Text Embedding Benchmark)*. https://huggingface.co/mteb. (Accessed on 07/25/2023).

[6] Jungo Kasai et al. *Bidimensional Leaderboards: Generate and Evaluate Language Hand in Hand*. 2022. arXiv: 2112.04139 [cs.CL].

[7] Alexander R. Fabbri et al. *SummEval: Re-evaluating Summarization Evaluation*. 2021. arXiv: 2007.12626 [cs.CL].

[8] Loıc Barrault et al. "Findings of the 2020 Conference on Machine Translation (WMT20)". In: *Proceedings of the Fifth Conference on Machine Translation*. Ed. by Loıc Barrault et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 1–55. URL: https://aclanthology.org/2020.wmt-1.1.

[9] Hongjin Su et al. *Selective Annotation Makes Language Models Better Few-Shot Learners*. 2022. arXiv: 2209.01975 [cs.CL].

[10] Joshua Robinson et al. *Contrastive Learning with Hard Negative Samples*. 2021. arXiv: 2010.04592 [cs.LG].