UML_Bert

October 24, 2025

[21]:
```
%reload_ext autoreload
%autoreload 2
```

[22]:
```
from kret_studies import *
from kret_studies.notebook import *
from kret_studies.complex import *

logger = get_notebook_logger()
```

```
/Users/Akseldkw/coding/kretsinger/data/nb_log.log
```

[23]:
```
from uml_project.data.constants import *
```

1 SETUP

Architecture: * Encoder * word_embedding_model generates word embeddings from text. * in BERT, sentence embeding is based on 1) token embedding (words) 2) segment embedding 3) position embedding (words order) * spits out sentence embeddings

- Pooler
  - FC layer
  - performs task given embedding from ecnoder

Training: train encoder with different pooler dimensions and select best encodere for this task.

Note: Ctrl+F ABOBA to change dims

2 DATA

2.1 IMDB

This is sentiment analysis, will be using later.

[24]:
```
df_imdb_train = pd.read_parquet(HF_DIR / "imdb/train.parquet")
```

[25]:
```
df_imdb_test = pd.read_parquet(HF_DIR / "imdb/test.parquet")
```

[26]:
```
df_imdb_train.describe()
```

[26]:
```
              label
count  25000.00000
mean       0.50000
std        0.50001
min        0.00000
25%        0.00000
50%        0.50000
75%        1.00000
max        1.00000
```

[27]:
```
df_imdb_train["text"][0]
```

[27]:
```
'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy
that surrounded it when it was first released in 1967. I also heard that at
first it was seized by U.S. customs if it ever tried to enter this country,
therefore being a fan of films considered "controversial" I really had to see
this for myself.<br /><br />The plot is centered around a young Swedish drama
student named Lena who wants to learn everything she can about life. In
particular she wants to focus her attentions to making some sort of documentary
on what the average Swede thought about certain political issues such as the
Vietnam War and race issues in the United States. In between asking politicians
and ordinary denizens of Stockholm about their opinions on politics, she has sex
with her drama teacher, classmates, and married men.<br /><br />What kills me
about I AM CURIOUS-YELLOW is that 40 years ago, this was considered
pornographic. Really, the sex and nudity scenes are few and far between, even
then it\'s not shot like some cheaply made porno. While my countrymen mind find
it shocking, in reality sex and nudity are a major staple in Swedish cinema.
Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex
scenes in his films.<br /><br />I do commend the filmmakers for the fact that
any sex shown in the film is shown for artistic purposes rather than just to
shock people and make money to be shown in pornographic theaters in America. I
AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and
potatoes (no pun intended) of Swedish cinema. But really, this film doesn\'t
have much of a plot.'
```

I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of films considered “controversial” I really had to see this for myself.The plot is centered around a young Swedish drama student named Lena who wants to learn everything she can about life. In particular she wants to focus her attentions to making some sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues in the United States. In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has sex with her drama teacher, classmates, and married men.What kills me about I AM CURIOUS-YELLOW is that 40 years ago, this was considered pornographic. Really, the sex and nudity scenes are few and far between, even then it’s not shot like some cheaply made porno. While my countrymen mind find it shocking, in reality sex and nudity are a major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.I do commend the

filmmakers for the fact that any sex shown in the film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and potatoes (no pun intended) of Swedish cinema. But really, this film doesn't have much of a plot.

We are not doing sentiment analysis yet.

2.2 Sentence-compression

[28]:
```
dataset_name = "embedding-data/sentence-compression"
dataset = load_dataset(dataset_name)

print("Dataset structure before splitting and transforming:", dataset)

# Assuming the dataset has a 'train' split, split it into train, validation,␣
 ↪and test
if "train" in dataset:
    print("Splitting the 'train' split into train, validation, and test (80/10/
 ↪10).")
    # Split train into train and test (80/20)
    train_test_split = dataset["train"].train_test_split(test_size=0.2, seed=42)
    train_split = train_test_split["train"]
    test_validation_split = train_test_split["test"]

    # Split the 20% test set into validation and test (50/50, resulting in 10%␣
 ↪validation and 10% test of original)
    validation_test_split = test_validation_split.train_test_split(test_size=0.
 ↪5, seed=42)
    validation_split = validation_test_split["train"]
    test_split = validation_test_split["test"]

    # Create a new DatasetDict with the splits
    dataset_splits = DatasetDict({"train": train_split, "validation":␣
 ↪validation_split, "test": test_split})

    print("\nDataset structure after splitting:", dataset_splits)

    # Define a function to transform each example
    def transform_example(example):
        if isinstance(example["set"], list) and len(example["set"]) == 2:
            return {
                "sentence1": example["set"][0],
                "sentence2": example["set"][1],
                "label": 1.0,  # Add the label with value 1
            }
        else:
            # Return None or handle cases that don't match the expected format
```

```
            return None

    # Apply the transformation to each split
    transformed_dataset_splits = DatasetDict()
    for split_name, split_dataset in dataset_splits.items():
        transformed_dataset_splits[split_name] = split_dataset.
 ↪map(transform_example, remove_columns=["set"])

    dataset_splits = transformed_dataset_splits

    print("\nDataset structure after transforming:", dataset_splits)
    print("\nNumber of examples in train split:", len(dataset_splits["train"]))
    print("Number of examples in validation split:",␣
 ↪len(dataset_splits["validation"]))
    print("Number of examples in test split:", len(dataset_splits["test"]))

    # Save the splits locally
    save_path = "./sentence-compression-dataset"
    dataset_splits.save_to_disk(save_path)
    print(f"\nDataset splits saved to {save_path}")

else:
    print("Dataset does not contain a 'train' split. Cannot perform splitting.")
```

```
README.md: 0.00B [00:00, ?B/s]

sentence-compression_compressed.jsonl.gz:   0%|          | 0.00/14.2M [00:00<?, ?
 ↪B/s]

Generating train split:   0%|          | 0/180000 [00:00<?, ? examples/s]

Dataset structure before splitting and transforming: DatasetDict({
    train: Dataset({
        features: ['set'],
        num_rows: 180000
    })
})
Splitting the 'train' split into train, validation, and test (80/10/10).

Dataset structure after splitting: DatasetDict({
    train: Dataset({
        features: ['set'],
        num_rows: 144000
    })
    validation: Dataset({
        features: ['set'],
        num_rows: 18000
    })
    test: Dataset({
```

```
        features: ['set'],
        num_rows: 18000
    })
})
```

```
Map:   0%|          | 0/144000 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/18000 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/18000 [00:00<?, ? examples/s]
```

```
Dataset structure after transforming: DatasetDict({
    train: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 144000
    })
    validation: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 18000
    })
    test: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 18000
    })
})

Number of examples in train split: 144000
Number of examples in validation split: 18000
Number of examples in test split: 18000
```

```
Saving the dataset (0/1 shards):   0%|          | 0/144000 [00:00<?, ? examples/
↪s]
```

```
Saving the dataset (0/1 shards):   0%|          | 0/18000 [00:00<?, ? examples/s]
```

```
Saving the dataset (0/1 shards):   0%|          | 0/18000 [00:00<?, ? examples/s]
```

```
Dataset splits saved to ./sentence-compression-dataset
```

[29]:
```
print("Dataset structure:", dataset_splits)

# Display information about the training split
keys = ["train", "validation", "test"]
for key in keys:
    if key in dataset_splits:
        print(f"\n{key} split info:")
        print(dataset_splits[key])
        print("\nFeatures:", dataset_splits[key].features)
        print(f"\nNumber of examples in {key} split:", len(dataset_splits[key]))
```

```
        # Display the first few examples from the training split
        print("\nFirst 5 examples from the training split:")
        for i in range(min(5, len(dataset_splits[key]))):
            print(f"Example {i}: {dataset_splits[key][i]}")
```

```
Dataset structure: DatasetDict({
    train: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 144000
    })
    validation: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 18000
    })
    test: Dataset({
        features: ['sentence1', 'sentence2', 'label'],
        num_rows: 18000
    })
})

train split info:
Dataset({
    features: ['sentence1', 'sentence2', 'label'],
    num_rows: 144000
})

Features: {'sentence1': Value('string'), 'sentence2': Value('string'), 'label':
Value('float64')}

Number of examples in train split: 144000

First 5 examples from the training split:
Example 0: {'sentence1': 'A Michigan man has pleaded guilty to persuading
mothers in several states to sexually assault their young children and send him
images.', 'sentence2': 'Mich. man pleads guilty to persuading mothers to
sexually assault their young children', 'label': 1.0}
Example 1: {'sentence1': "Isipathana made it three in a row when they recorded a
convincing 50-12 victory over St.Joseph's College in the Cup Final of the Zahira
Rugby Sevens held on Sunday at the Royal College Sports Complex organized by
Zahira College Group of Sixties.", 'sentence2': 'Isipathana make it three in a
row', 'label': 1.0}
Example 2: {'sentence1': 'A mother accused of abducting her 4-year-old daughter
turned herself in at the Prince William County Adult Detention Center early
Thursday.', 'sentence2': 'Mother accused of abducting daughter turns herself
in', 'label': 1.0}
Example 3: {'sentence1': "In this post-``Bridesmaids'' autumn, the comedy
``What's Your Number?'' looks awfully familiar:", 'sentence2': "'What's Your
```

```
Number?':", 'label': 1.0}
Example 4: {'sentence1': 'Oh God, I am not back.', 'sentence2': 'I am not
back:', 'label': 1.0}

validation split info:
Dataset({
    features: ['sentence1', 'sentence2', 'label'],
    num_rows: 18000
})

Features: {'sentence1': Value('string'), 'sentence2': Value('string'), 'label':
Value('float64')}

Number of examples in validation split: 18000

First 5 examples from the training split:
Example 0: {'sentence1': 'The government of the central Bié province improved
the power supply in the local chief communes.', 'sentence2': 'Government
improves power supply in communes', 'label': 1.0}
Example 1: {'sentence1': 'A Box Elder woman whose son was killed in Afghanistan
described him as a loving family man, a true friend, an outdoorsman and a
hero.', 'sentence2': "Box Elder woman's son killed in Afghanistan", 'label':
1.0}
Example 2: {'sentence1': 'Samsung has unveiled what is undoubtedly its largest
Android smartphone to date, the Galaxy Mega.', 'sentence2': 'Samsung unveils its
largest Android smartphone Galaxy Mega', 'label': 1.0}
Example 3: {'sentence1': "New Delhi The Delhi High Court has rejected the
government's plea that cabinet papers containing the deliberations of the
ministers cannot be disclosed under the RTI Act even after a decision has been
taken by it on an issue.", 'sentence2': 'Cabinet papers can be disclosed under
RTI Act:', 'label': 1.0}
Example 4: {'sentence1': "Petr Cech has admitted that Chelsea might have been
lucky to beat Barcelona in last week's Champions League semi-final first leg.",
'sentence2': "Petr Cech: ``we may have been lucky''", 'label': 1.0}

test split info:
Dataset({
    features: ['sentence1', 'sentence2', 'label'],
    num_rows: 18000
})

Features: {'sentence1': Value('string'), 'sentence2': Value('string'), 'label':
Value('float64')}

Number of examples in test split: 18000

First 5 examples from the training split:
Example 0: {'sentence1': 'Stoke City defender Danny Collins has joined
```

```
Nottingham Forest for an undisclosed fee after signing a three-year contract.',
'sentence2': 'Stoke City defender Danny Collins joins Nottingham Forest',
'label': 1.0}
Example 1: {'sentence1': 'The city of Ellsworth has begun offering vehicle
registration renewals, saving many residents a trip to the Bureau of Motor
Vehicles after paying their excise tax at City Hall.', 'sentence2': 'City
offering vehicle registration renewals', 'label': 1.0}
Example 2: {'sentence1': 'Share prices on Bursa Malaysia ended the morning
session mixed but sentiment remained positive, said dealers.', 'sentence2':
'Share prices end morning session mixed', 'label': 1.0}
Example 3: {'sentence1': "The Foundation stone laying ceremony of 'Urbania
Homes' super luxury apartments was held at 1st Cross, Shivbaug Road, Kadri here
on Wednesday August 17.", 'sentence2': 'Foundation stone laid for Urbania Homes
super luxury apartments', 'label': 1.0}
Example 4: {'sentence1': 'Atlas Energy Inc. reported that average daily
production in its Appalachian segment, which includes Marcellus shale natural
gas activity primarily in Pennsylvania, increased 21 percent from first-quarter
rates to about 55 million cubic feet per day.', 'sentence2': 'Atlas Energy
reports production increase', 'label': 1.0}
```

2.3 Scientific x Taylor

[ ]:

3 Model

[31]:
```python
# -------------------------
# Model creation helper
# -------------------------
def build_model(base_model_name: str, target_dim: int) -> SentenceTransformer:
    """
    Build a SentenceTransformer where we append a Dense projection after pooling
    to obtain exactly `target_dim` output dimensions.
    """
    # Transformer (encoder)
    word_embedding_model = models.Transformer(base_model_name,␣
 ↪max_seq_length=128)

    # Mean pooling (or use cls pooling if you prefer)
    pooling_model = models.Pooling(
        word_embedding_model.get_word_embedding_dimension(),
        pooling_mode_mean_tokens=True,  # sentence embedding = mean of word␣
 ↪embeddings in sentence, that's rule of thumb for sentence similarity but if␣
 ↪we want to do classification prob cls is better
        pooling_mode_cls_token=False,  # instead of cls or max use mean here;␣
 ↪ABOBA: can vary and see changes
        pooling_mode_max_tokens=False,
```

```
    )

    # The pooler (projector)
    dense = models.Dense(
        in_features=pooling_model.get_sentence_embedding_dimension(),
        out_features=target_dim,  # ABOBA: vary output dim
        activation_function=nn.Tanh(),  # paper used typical pooler activations;
 ↪ Tanh is common
    )

    model = SentenceTransformer(modules=[word_embedding_model, pooling_model,␣
 ↪dense], device=DEVICE)
    return model


# -------------------------
# Data loaders (contrastive / STS demo)
# -------------------------
def load_sts_train_eval():  # paper used sts for evaluation
    """
    Load STS-B dataset (train/validation/test) from 'glue' or 'stsb_multi_mt'.
    We use this both for training demo and evaluation (small-scale).
    Replace with large contrastive dataset for better training (e.g.,␣
 ↪NLI+hard-negatives).
    """
    ds = load_dataset("glue", "stsb")  # ABOBA: @alena add Aksel's datasets
    train = ds["train"]
    val = ds["validation"]
    test = ds["test"]

    # Prepare SentenceTransformers InputExample format for regression (score in␣
 ↪[0,1])
    def to_examples(split):
        examples = []
        for row in split:
            s1 = row["sentence1"]
            s2 = row["sentence2"]
            score = float(row["label"]) / 5.0  # STS-B scores 0..5 -> normalize␣
 ↪to 0..1
            examples.append(InputExample(texts=[s1, s2], label=score))
        return examples

    return to_examples(train), to_examples(val), to_examples(test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[31], line 4
```

```
      1 # -------------------------
      2 # Model creation helper
      3 # -------------------------
----> 4 def build_model(base_model_name: str, target_dim: int) ->␣
 ↪SentenceTransformer:
      5     """
      6     Build a SentenceTransformer where we append a Dense projection afte␣
 ↪pooling
      7     to obtain exactly `target_dim` output dimensions.
      8     """
      9     # Transformer (encoder)

NameError: name 'SentenceTransformer' is not defined
```

[ ]:
```
# -------------------------
# Training functions
# -------------------------
def freeze_encoder_only(model: SentenceTransformer):
    # SentenceTransformer stores modules in model._modules (OrderedDict-like).␣
 ↪The transformer is index 0.
    # Simpler: freeze parameters in modules that are instances of models.
 ↪Transformer
    for module in model._modules.values():
        if isinstance(module, models.Transformer):
            for p in module.parameters():
                p.requires_grad = False


def unfreeze_all(model: SentenceTransformer):
    for (
        p
    ) in (
        model.parameters()
    ):  # note .parameters() is built-in nn.Module from whihc␣
 ↪SentenceTransformer and its submodules inherit
        p.requires_grad = True


def train_pooler_then_finetune(model: SentenceTransformer, train_examples,␣
 ↪val_examples, out_dir: str):
    # Step A: train pooler only (encoder frozen)
    freeze_encoder_only(model)
    train_dataloader = torch.utils.data.DataLoader(train_examples,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
    # Use CosineSimilarityLoss for contrastive-style or MSELoss for regression␣
 ↪(STS)
```

```
    loss_fct = losses.CosineSimilarityLoss(
        model
    )  # ABOBA: try dfferent distances; use losses.MultipleNegativesRankingLoss␣
↪for regression
    # regression objective (STS) -> losses.SoftTargetLoss or losses.
↪SentenceLabelLoss
    # sentence-transformers doesn't provide direct regression loss we can wrap␣
↪a MSE by computing cosine and matching target score. For simplicity, use␣
↪CosineSimilarityLoss and treat high score -> similar.
    evaluator = evaluation.EmbeddingSimilarityEvaluator.from_input_examples(
        val_examples, name="sts-val"
    )  # note this benchmark compares against human-annotated similarity scores;
↪ ABOBA: we can't self-annotate sim for Swift or Verma so we can't get␣
↪encoder error

    model.fit(
        train_objectives=[(train_dataloader, loss_fct)],
        evaluator=evaluator,
        epochs=EPOCHS_POOLER,
        warmup_steps=100,
        output_path=os.path.join(out_dir, "stepA_pooler_only"),
        optimizer_params={"lr": POOLER_LR},
    )

    # Step B: unfreeze encoder and finetune whole model
    unfreeze_all(model)
    # Recreate dataloader (sentence-transformers expects InputExamples in an␣
↪in-memory list)
    train_dataloader = torch.utils.data.DataLoader(train_examples,␣
↪batch_size=BATCH_SIZE, shuffle=True)
    loss_fct2 = losses.MultipleNegativesRankingLoss(
        model
    )  # good objective for contrastive training (requires positive pairs)
    model.fit(
        train_objectives=[(train_dataloader, loss_fct2)],
        evaluator=evaluator,
        epochs=EPOCHS_FINETUNE,
        warmup_steps=100,
        output_path=os.path.join(out_dir, "stepB_finetune"),
        optimizer_params={"lr": FINETUNE_LR},
    )
```

[ ]:
```
# -------------------------
# Evaluation helpers
# -------------------------
def evaluate_sts(model: SentenceTransformer, examples):
```

```
    """Compute Pearson & Spearman on STS-style examples using␣
 ↪sentence-transformers evaluator utilities."""
    evaluator = evaluation.EmbeddingSimilarityEvaluator.
 ↪from_input_examples(examples, name="sts-eval")
    return evaluator(model)


def compute_pca_explained_variance(embeddings: np.ndarray, n_components: int =␣
 ↪50) -> Tuple[np.ndarray, np.ndarray]:
    pca = PCA(n_components=n_components)
    pca.fit(embeddings)
    explained = pca.explained_variance_ratio_
    cum = np.cumsum(explained)
    return explained, cum


def participation_ratio(singular_values: np.ndarray) -> float:
    """
    Participation ratio = (sum_i s_i^2)^2 / sum_i s_i^4
    When s_i are singular values of embedding matrix (or eigenvalues).
    Higher -> more dimensions effectively used.
    """
    s2 = singular_values**2
    num = (s2.sum()) ** 2
    den = (s2**2).sum()
    if den == 0:
        return 0.0
    return num / den
```

[ ]:
```
"""
This is reproduction the two-step training idea from:
"On the Dimensionality of Sentence Embeddings" (Wang et al., EMNLP Findings␣
 ↪2023).
that paper didn't include code
"""

import os
from typing import Tuple
import numpy as np
import torch
from torch import nn
from datasets import load_dataset, DatasetDict, load_from_disk  # Import␣
 ↪load_from_disk
from sentence_transformers import SentenceTransformer, models, losses,␣
 ↪InputExample, evaluation
from sentence_transformers.readers import STSBenchmarkDataReader
from sklearn.decomposition import PCA
```

```python
from sklearn.metrics import mean_squared_error
import evaluate

# -------------------------
# Config
# -------------------------
BASE_MODEL = "sentence-transformers/all-MiniLM-L6-v2"  # ABOBA: small, fast␣
 ↪baseline; swap as desired; can use BERT
TARGET_DIM = 32  # ABOBA desired embedding dimensionality (experiment with 32,␣
 ↪64, 128...)
BATCH_SIZE = 64
POOLER_LR = 2e-4
FINETUNE_LR = 2e-5
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
EPOCHS_POOLER = 2  # step A epochs (pooler only)
EPOCHS_FINETUNE = 2  # step B epochs (unfreeze and train)
SAVE_DIR = "./lowdim_model"
CUSTOM_DATASET_PATH = "./sentence-compression-dataset"
os.makedirs(SAVE_DIR, exist_ok=True)


# -------------------------
# Model creation helper
# -------------------------
def build_model(base_model_name: str, target_dim: int) -> SentenceTransformer:
    """
    Build a SentenceTransformer where we append a Dense projection after pooling
    to obtain exactly `target_dim` output dimensions.
    """
    # Transformer (encoder)
    word_embedding_model = models.Transformer(base_model_name,␣
 ↪max_seq_length=128)

    # Mean pooling (or use cls pooling if you prefer)
    pooling_model = models.Pooling(
        word_embedding_model.get_word_embedding_dimension(),
        pooling_mode_mean_tokens=True,  # sentence embedding = mean of word␣
 ↪embeddings in sentence, that's rule of thumb for sentence similarity but if␣
 ↪we want to do classification prob cls is better
        pooling_mode_cls_token=False,  # instead of cls or max use mean here;␣
 ↪ABOBA: can vary and see changes
        pooling_mode_max_tokens=False,
    )

    # The pooler (projector)
    dense = models.Dense(
        in_features=pooling_model.get_sentence_embedding_dimension(),
```

```
        out_features=target_dim,  # ABOBA: vary output dim
        activation_function=nn.Tanh(),  # paper used typical pooler activations;
 ↪ Tanh is common
    )

    model = SentenceTransformer(modules=[word_embedding_model, pooling_model,␣
 ↪dense], device=DEVICE)
    return model


# -------------------------
# Data loaders (contrastive / STS demo)
# -------------------------
# def load_sts_train_eval(): # paper used sts for evaluation
#     """
#     Load STS-B dataset (train/validation/test) from 'glue' or 'stsb_multi_mt'.
#     We use this both for training demo and evaluation (small-scale).
#     Replace with large contrastive dataset for better training (e.g.,␣
 ↪NLI+hard-negatives).
#     """
#     ds = load_dataset("glue", "stsb") # ABOBA: @alena add Aksel's datasets
#     train = ds["train"]
#     val = ds["validation"]
#     test = ds["test"]

#     # Prepare SentenceTransformers InputExample format for regression (score␣
 ↪in [0,1])
#     def to_examples(split):
#         examples = []
#         for row in split:
#             s1 = row["sentence1"]
#             s2 = row["sentence2"]
#             score = float(row["label"]) / 5.0  # STS-B scores 0..5 ->␣
 ↪normalize to 0..1
#             examples.append(InputExample(texts=[s1, s2], label=score))
#         return examples

#     return to_examples(train), to_examples(val), to_examples(test)


def load_custom_dataset(dataset_path: str):
    """
    Load a custom dataset from a local path or Hugging Face Hub.
    Assumes the dataset is saved in a format loadable by datasets (e.g.,␣
 ↪parquet, json, csv).
    Assumes the dataset has 'sentence1', 'sentence2', and 'label' columns.
```

```
    Labels are assumed to be similarity scores.
    If only a 'train' split is available, it will be split into train,␣
↪validation, and test sets.
    """
    ds = None
    if dataset_path and os.path.exists(dataset_path):
        try:
            # Try loading from a directory (e.g., saved with dataset.
↪save_to_disk)
            ds = load_from_disk(dataset_path)
            print(f"Loaded dataset from disk: {dataset_path}")
        except Exception as e_disk:
            print(f"Could not load dataset from disk: {e_disk}")
            try:
                # Try loading from common file formats (csv, json, parquet)
                if dataset_path.endswith(".csv"):
                    ds = load_dataset("csv", data_files=dataset_path)
                elif dataset_path.endswith(".json"):
                    ds = load_dataset("json", data_files=dataset_path)
                elif dataset_path.endswith(".parquet"):
                    ds = load_dataset("parquet", data_files=dataset_path)
                else:
                    print(f"Unsupported file format for custom dataset:␣
↪{dataset_path}")
                    return None
                print(f"Loaded dataset from file: {dataset_path}")
            except Exception as e_file:
                print(f"Could not load dataset from file: {e_file}")
                return None
    elif dataset_path:
        # Try loading from Hugging Face Hub if dataset_path is not a local path
        try:
            ds = load_dataset(dataset_path)
            print(f"Loaded dataset from Hugging Face Hub: {dataset_path}")
        except Exception as e_hub:
            print(f"Could not load dataset from Hugging Face Hub: {e_hub}")
            return None
    else:
        print("No dataset path provided.")
        return None

    if ds is None:
        return None

    # Ensure ds is a DatasetDict for consistent handling
    if not isinstance(ds, DatasetDict):
        if "train" in ds.features:  # Check if it's a single split Dataset
```

```
            ds = DatasetDict({"train": ds})
            print("Wrapped single dataset split in a DatasetDict.")
        else:
            print("Loaded dataset is not in a recognized format (Dataset or␣
↪DatasetDict with 'train' split).")
            return None

    def to_examples(split):
        examples = []
        # Check if required columns exist
        if (
            "sentence1" not in split.column_names
            or "sentence2" not in split.column_names
            or "label" not in split.column_names
        ):
            print("Custom dataset must contain 'sentence1', 'sentence2', and␣
↪'label' columns.")
            return []
        for row in split:
            s1 = row["sentence1"]
            s2 = row["sentence2"]
            score = float(row["label"])  # Assume label is already a float␣
↪similarity score
            examples.append(InputExample(texts=[s1, s2], label=score))
        return examples

    custom_data = {}
    if "train" in ds:
        # If only train split, split it
        if "validation" not in ds and "test" not in ds:
            print("Splitting the 'train' split into train, validation, and test␣
↪(80/10/10).")
            ds_split = ds["train"].train_test_split(test_size=0.2, seed=42)
            ds_validation_test = ds_split["test"].train_test_split(test_size=0.
↪5, seed=42)
            custom_data["train"] = to_examples(ds_split["train"])
            custom_data["validation"] = to_examples(ds_validation_test["train"])
            custom_data["test"] = to_examples(ds_validation_test["test"])
        else:
            custom_data["train"] = to_examples(ds["train"])
            if "validation" in ds:
                custom_data["validation"] = to_examples(ds["validation"])
            if "test" in ds:
                custom_data["test"] = to_examples(ds["test"])

    return custom_data
```

```
# -------------------------
# Training functions
# -------------------------
def freeze_encoder_only(model: SentenceTransformer):
    # SentenceTransformer stores modules in model._modules (OrderedDict-like).␣
 ↪The transformer is index 0.
    # Simpler: freeze parameters in modules that are instances of models.
 ↪Transformer
    for module in model._modules.values():
        if isinstance(module, models.Transformer):
            for p in module.parameters():
                p.requires_grad = False


def unfreeze_all(model: SentenceTransformer):
    for (
        p
    ) in (
        model.parameters()
    ):  # note .parameters() is built-in nn.Module from whihc␣
 ↪SentenceTransformer and its submodules inherit
        p.requires_grad = True


def train_pooler_then_finetune(model: SentenceTransformer, train_examples,␣
 ↪val_examples, out_dir: str):
    # Step A: train pooler only (encoder frozen)
    freeze_encoder_only(model)
    train_dataloader = torch.utils.data.DataLoader(train_examples,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
    # Use CosineSimilarityLoss for contrastive-style or MSELoss for regression␣
 ↪(STS)
    loss_fct = losses.CosineSimilarityLoss(
        model
    )  # ABOBA: try dfferent distances; use losses.MultipleNegativesRankingLoss␣
 ↪for regression
    # regression objective (STS) -> losses.SoftTargetLoss or losses.
 ↪SentenceLabelLoss
    # sentence-transformers doesn't provide direct regression loss we can wrap␣
 ↪a MSE by computing cosine and matching target score. For simplicity, use␣
 ↪CosineSimilarityLoss and treat high score -> similar.
    evaluator = evaluation.EmbeddingSimilarityEvaluator.from_input_examples(
        val_examples, name="sts-val"
```

```
    )  # note this benchmark compares against human-annotated similarity scores;
 ↪ ABOBA: we can't self-annotate sim for Swift or Verma so we can't get␣
 ↪encoder error

    model.fit(
        train_objectives=[(train_dataloader, loss_fct)],
        evaluator=evaluator,
        epochs=EPOCHS_POOLER,
        warmup_steps=100,
        output_path=os.path.join(out_dir, "stepA_pooler_only"),
        optimizer_params={"lr": POOLER_LR},
    )

    # Step B: unfreeze encoder and finetune whole model
    unfreeze_all(model)
    # Recreate dataloader (sentence-transformers expects InputExamples in an␣
 ↪in-memory list)
    train_dataloader = torch.utils.data.DataLoader(train_examples,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
    loss_fct2 = losses.MultipleNegativesRankingLoss(
        model
    )  # good objective for contrastive training (requires positive pairs)
    model.fit(
        train_objectives=[(train_dataloader, loss_fct2)],
        evaluator=evaluator,
        epochs=EPOCHS_FINETUNE,
        warmup_steps=100,
        output_path=os.path.join(out_dir, "stepB_finetune"),
        optimizer_params={"lr": FINETUNE_LR},
    )


# -------------------------
# Evaluation helpers
# -------------------------
def evaluate_sts(model: SentenceTransformer, examples):
    """Compute Pearson & Spearman on STS-style examples using␣
 ↪sentence-transformers evaluator utilities."""
    evaluator = evaluation.EmbeddingSimilarityEvaluator.
 ↪from_input_examples(examples, name="sts-eval")
    return evaluator(model)


def compute_pca_explained_variance(embeddings: np.ndarray, n_components: int =␣
 ↪50) -> Tuple[np.ndarray, np.ndarray]:
    pca = PCA(n_components=n_components)
```

```
    pca.fit(embeddings)
    explained = pca.explained_variance_ratio_
    cum = np.cumsum(explained)
    return explained, cum


def participation_ratio(singular_values: np.ndarray) -> float:
    """
    Participation ratio = (sum_i s_i^2)^2 / sum_i s_i^4
    When s_i are singular values of embedding matrix (or eigenvalues).
    Higher -> more dimensions effectively used.
    """
    s2 = singular_values**2
    num = (s2.sum()) ** 2
    den = (s2**2).sum()
    if den == 0:
        return 0.0
    return num / den


# -------------------------
# Main: build, train, evaluate
# -------------------------
def main():
    print("Device:", DEVICE)
    # 1) Build model with custom pooler/projection to TARGET_DIM
    model = build_model(BASE_MODEL, TARGET_DIM)
    print("Model built. Output dim:", model.get_sentence_embedding_dimension())

    train_examples = []
    val_examples = []
    test_examples = []

    # Load custom dataset if specified
    custom_data = load_custom_dataset(CUSTOM_DATASET_PATH)
    if custom_data:
        print(
            f"Loaded custom dataset: train={len(custom_data.get('train', []))}␣
 ↪val={len(custom_data.get('validation', []))} test={len(custom_data.
 ↪get('test', []))}"
        )
        # Use custom data for training, validation, and testing if available
        train_examples = custom_data.get("train", [])
        val_examples = custom_data.get("validation", [])
        test_examples = custom_data.get("test", [])
        print("Using custom dataset for training, validation, and evaluation.")
    else:
```

```
        # 2) Load STS train/eval (demo) if no custom dataset is provided
        train_examples, val_examples, test_examples = load_sts_train_eval()
        print(f"Loaded STS: train={len(train_examples)} val={len(val_examples)}␣
↪test={len(test_examples)}")
        print("Using STS-B dataset for training, validation, and evaluation.")

    if not train_examples or not val_examples or not test_examples:
        print(
            "Error: Training, validation, and/or test datasets are empty or␣
↪could not be loaded. Cannot proceed with training and evaluation."
        )
        return

    # 3) Train: two-step
    train_pooler_then_finetune(model, train_examples, val_examples, SAVE_DIR)

    # 4) Evaluate on test
    print("Evaluating on test set...")
    evaluate_sts(model, test_examples)

    # 5) Dump embeddings for intrinsic-dim analysis (take a subset for speed)
    sentences = [ex.texts[0] for ex in test_examples][:1000]  # first sentence␣
↪of pairs, subset
    embeddings = model.encode(sentences, show_progress_bar=True,␣
↪convert_to_numpy=True)
    print("Embeddings shape:", embeddings.shape)

    # PCA info
    explained, cum = compute_pca_explained_variance(embeddings,␣
↪n_components=min(embeddings.shape[1], 50))
    print("Explained variance ratios (first 10):", explained[:10])
    print("Cumulative variance (first 10):", cum[:10])

    # Participation ratio (use SVD of covariance)
    cov = np.cov(embeddings, rowvar=False)
    eigvals = np.linalg.eigvalsh(cov)  # ascending
    eigvals = eigvals[eigvals > 0]
    pr = participation_ratio(np.sqrt(eigvals))  # pass singular values ~␣
↪sqrt(eigvals)
    print(f"Participation ratio (approx): {pr:.3f}")

    # Save final model
    model.save(os.path.join(SAVE_DIR, "final_lowdim_model"))
    print("Saved model to", os.path.join(SAVE_DIR, "final_lowdim_model"))
```

```
if __name__ == "__main__":
    main()
```

```
Device: cuda

config.json:   0%|          | 0.00/612 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/90.9M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/350 [00:00<?, ?B/s]

vocab.txt: 0.00B [00:00, ?B/s]

tokenizer.json: 0.00B [00:00, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]

Model built. Output dim: 32
Loaded dataset from disk: ./sentence-compression-dataset
Loaded custom dataset: train=144000 val=18000 test=18000
Using custom dataset for training, validation, and evaluation.

Computing widget examples:   0%|          | 0/1 [00:00<?, ?example/s]

/usr/local/lib/python3.12/dist-packages/notebook/notebookapp.py:191:
SyntaxWarning: invalid escape sequence '\/'
  | |_| | '_ \/ _` / _` |  _/ -_)

<IPython.core.display.Javascript object>

wandb: Logging into wandb.ai. (Learn how to deploy a W&B server
locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here:
https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter:

 ..........

wandb: WARNING If you're specifying your api key in code,
ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY
environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
wandb: Currently logged in as: alenachan121
(alenachan121-columbia-university) to https://api.wandb.ai.
Use `wandb login --relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.12/dist-
packages/sentence_transformers/evaluation/EmbeddingSimilarityEvaluator.py:195:
ConstantInputWarning: An input array is constant; the correlation coefficient is
not defined.
  eval_pearson, _ = pearsonr(labels, scores)
/usr/local/lib/python3.12/dist-
packages/sentence_transformers/evaluation/EmbeddingSimilarityEvaluator.py:196:
ConstantInputWarning: An input array is constant; the correlation coefficient is
not defined.
  eval_spearman, _ = spearmanr(labels, scores)

<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3251801477.py in <cell line: 0>()
    317
    318 if __name__ == "__main__":
--> 319     main()

/tmp/ipython-input-3251801477.py in main()
    288
    289     # 3) Train: two-step
--> 290     train_pooler_then_finetune(model, train_examples, val_examples,␣
 ↪SAVE_DIR)
    291
    292     # 4) Evaluate on test

/tmp/ipython-input-3251801477.py in train_pooler_then_finetune(model,␣
 ↪train_examples, val_examples, out_dir)
    217     train_dataloader = torch.utils.data.DataLoader(train_examples,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
    218     loss_fct2 = losses.MultipleNegativesRankingLoss(model)  # good␣
 ↪objective for contrastive training (requires positive pairs)
--> 219     model.fit(
    220         train_objectives=[(train_dataloader, loss_fct2)],
    221         evaluator=evaluator,

/usr/local/lib/python3.12/dist-packages/sentence_transformers/fit_mixin.py in␣
 ↪fit(self, train_objectives, evaluator, epochs, steps_per_epoch, scheduler,␣
 ↪warmup_steps, optimizer_class, optimizer_params, weight_decay,␣
 ↪evaluation_steps, output_path, save_best_model, max_grad_norm, use_amp,␣
 ↪callback, show_progress_bar, checkpoint_path, checkpoint_save_steps,␣
 ↪checkpoint_save_total_limit, resume_from_checkpoint)
```

```
    406                 resume_from_checkpoint = None
    407
--> 408         trainer.train(resume_from_checkpoint=resume_from_checkpoint)
    409
    410     @staticmethod

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in train(self,␣
 ↪resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
   2323                 hf_hub_utils.enable_progress_bars()
   2324         else:
-> 2325             return inner_training_loop(
   2326                 args=args,
   2327                 resume_from_checkpoint=resume_from_checkpoint,

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in␣
 ↪_inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial,␣
 ↪ignore_keys_for_eval)
   2672                     )
   2673                     with context():
-> 2674                         tr_loss_step = self.training_step(model, inputs␣
 ↪num_items_in_batch)
   2675
   2676                     if (

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in␣
 ↪training_step(***failed resolving arguments***)
   4069                     kwargs["scale_wrt_gas"] = False
   4070
-> 4071                 self.accelerator.backward(loss, **kwargs)
   4072
   4073             return loss.detach()

/usr/local/lib/python3.12/dist-packages/accelerate/accelerator.py in␣
 ↪backward(self, loss, **kwargs)
   2732             self.lomo_backward(loss, learning_rate)
   2733         else:
-> 2734             loss.backward(**kwargs)
   2735
   2736     def set_trigger(self):

/usr/local/lib/python3.12/dist-packages/torch/_tensor.py in backward(self,␣
 ↪gradient, retain_graph, create_graph, inputs)
    645                 inputs=inputs,
    646             )
--> 647         torch.autograd.backward(
    648             self, gradient, retain_graph, create_graph, inputs=inputs
    649         )
```

```
/usr/local/lib/python3.12/dist-packages/torch/autograd/__init__.py in␣
 ↪backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables,␣
 ↪inputs)
    352     # some Python versions print out the first line of a multi-line␣
 ↪function
    353     # calls in the traceback and some print out the last line
--> 354     _engine_run_backward(

    355         tensors,
    356         grad_tensors_,

/usr/local/lib/python3.12/dist-packages/torch/autograd/graph.py in␣
 ↪_engine_run_backward(t_outputs, *args, **kwargs)
    827         unregister_hooks =␣
 ↪_register_logging_hooks_on_whole_graph(t_outputs)
    828     try:
--> 829         return Variable._execution_engine.run_backward(  # Calls into␣
 ↪the C++ engine to run the backward pass

    830             t_outputs, *args, **kwargs
    831         )  # Calls into the C++ engine to run the backward pass

KeyboardInterrupt:
```