跟龙哥学真AI

Embedding技术与实践

语义理解中Embedding意义

```
import torch

embedding = torch.nn.Embedding(num_embeddings=4, embedding_dim=3)
words = [ [2, 3], [1, 2] ]
embed = embedding(torch.LongTensor(words))
print(embed)
print(embed.size())


'''
tensor([[[ 0.4015,  0.3967,  0.8442],
         [ 1.7117,  0.2838,  0.1853]],

        [[-2.2015,  0.3107,  0.9713],
         [ 0.4015,  0.3967,  0.8442]]], grad_fn=<EmbeddingBackward0>)
torch.Size([2, 2, 3])
'''
```

Word2Vec

最著名的密集表示方法之一是word2vec，由Google于2013年提出的论文：effective Estimation of Word Representations in Vector Space https://arxiv.org/abs/1301.3781

代码示例

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import torch.utils.data as Data

dtype = torch.FloatTensor
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


# 模型的相关参数
batch_size = 8
embedding_size = 2  # 词向量的维度是2
```

```python
C = 2  # window size

def prepare_data():
    # 文本预处理
    sentences = ["longge like dog", "longge like cat", "longge like animal",
                "dog cat animal", "banana apple cat dog like", "dog fish milk
like",
                "dog cat animal like", "longge like apple", "apple like", "longge
like banana",
                "apple banana longge movie book music like", "cat dog hate", "cat
dog like"]

    word_sequence = " ".join(sentences).split()  # ['longge', 'like', 'dog',
'longge', 'like', 'cat', 'animal',...]
    vocab = list(set(word_sequence))  # build words vocabulary，去重
    word2idx = {w: i for i, w in enumerate(vocab)}

    voc_size = len(vocab)

    # 数据预处理
    skip_grams = []
    print(word2idx)
    for idx in range(C, len(word_sequence) - C):
        center = word2idx[word_sequence[idx]]  # 中心词

        context_idx = list(range(idx - C, idx)) + list(range(idx + 1, idx + C +
1))  # 中心词左边的2个词+中心词右边的两个词
        context = [word2idx[word_sequence[i]] for i in context_idx]
        for w in context:
            skip_grams.append([center, w])  # 中心词和每个周围词组成一个训练样本


    def make_data(skip_grams):
        input_data = []
        output_data = []
        for i in range(len(skip_grams)):
            # input_data转换为one-hot形式，output_data合成一个list
            input_data.append(np.eye(voc_size)[skip_grams[i][0]])
            output_data.append(skip_grams[i][1])
        return input_data, output_data


    print(skip_grams)
    input_data, output_data = make_data(skip_grams)
    print(input_data)
    print(output_data)
    input_data, output_data = torch.Tensor(input_data),
torch.LongTensor(output_data)
    dataset = Data.TensorDataset(input_data, output_data)
    loader = Data.DataLoader(dataset, batch_size, True)

    return vocab,voc_size, loader


# 构建模型
class Word2Vec(nn.Module):
    def __init__(self):
        super(Word2Vec, self).__init__()
        self.W_in = nn.Parameter(torch.randn(voc_size,
embedding_size).type((dtype)))
        self.W_out = nn.Parameter(torch.randn(embedding_size,
voc_size).type((dtype)))

    def forward(self, X):
        # X : [batch_size, voc_size] one-hot
        # torch.mm only for 2 dim matrix, but torch.matmul can use to any dim
        hidden_layer = torch.matmul(X, self.W_in)  # hidden_layer : [batch_size,
embedding_size]
        output_layer = torch.matmul(hidden_layer, self.W_out)  # output_layer :
[batch_size, voc_size]
        return output_layer



def train_model():

    model = Word2Vec().to(device)     #定义模型
    criterion = nn.CrossEntropyLoss().to(device)   #交叉熵损失函数
    optimizer = optim.Adam(model.parameters(), lr=1e-4)    #优化器
    # 训练
    for epoch in range(2000):
        for i, (batch_x, batch_y) in enumerate(loader):
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device)
            pred = model(batch_x)
            loss = criterion(pred, batch_y)
            if (epoch + 1) % 1000 == 0:
                print(epoch + 1, i, loss.item())
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    # 将每个词在平面直角坐标系中标记出来，看看各个词之间的距离
    for i, label in enumerate(vocab):
        W, WT = model.parameters()
        # W是词向量矩阵
        x, y = float(W[i][0]), float(W[i][1])
        plt.scatter(x, y)
        plt.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points',
ha='right', va='bottom')
    plt.show()


#准备好数据集
vocab,voc_size,loader=prepare_data()

#开始训练
train_model()
```

使用llamaIndex 进行embedding微调

环境准备

```
#conda创建  python=3.10版本的虚环境
conda create -n rag_embedding python=3.10
#激活conda创建的名字叫rag_embedding的虚环境
conda activate rag_embedding


#torch安装
conda install pytorch==2.3.1 torchvision==0.18.1 torchaudio==2.3.1 pytorch-
cuda=12.1 -c pytorch -c nvidia


#安装依赖
#如果用openai模型来生成微调数据集
pip install llama-index-llms-openai
pip install llama-index-embeddings-openai

#如果是本地部署ollama
pip install llama-index-llms-ollama

#安装依赖
pip install llama-index-finetuning
pip install llama-index-embeddings-huggingface
pip install -U llama-index    #安装最新的llama-index
```

文档地址：https://docs.llamaindex.ai/en/stable/use_cases/fine_tuning/

准备数据集

使用　中华人民共和国证券法(2019修订).pdf　文件作为语料，下载地址

https://www.modelscope.cn/datasets/longgeai3x3/ragdata/files

将文档生成QA数据对 用作微调数据集

```
import json

from llama_index.core import SimpleDirectoryReader
from llama_index.core.node_parser import SentenceSplitter
from llama_index.core.schema import MetadataMode


import os
from llama_index.llms.openai import OpenAI
from llama_index.llms.ollama import Ollama
```

```python
from llama_index.finetuning import generate_qa_embedding_pairs
from llama_index.core.evaluation import EmbeddingQAFinetuneDataset


from llama_index.finetuning import SentenceTransformersFinetuneEngine
from llama_index.core.evaluation import EmbeddingQAFinetuneDataset


BASE_DIR="G:/ai课程/rag/finetuning/data/"

TRAIN_FILES = [BASE_DIR+"中华人民共和国证券法(2019修订).pdf"]
VAL_FILES = [BASE_DIR+"中华人民共和国证券法(2019修订).pdf"]

TRAIN_CORPUS_FPATH = BASE_DIR+"train_corpus.json"
VAL_CORPUS_FPATH = BASE_DIR+"val_corpus.json"


def load_corpus(files, verbose=False):
    if verbose:
        print(f"Loading files {files}")

    reader = SimpleDirectoryReader(input_files=files)
    docs = reader.load_data()

    if verbose:
        print(f"Loaded {len(docs)} docs")

    parser = SentenceSplitter()
    nodes = parser.get_nodes_from_documents(docs, show_progress=verbose)

    if verbose:
        print(f"Parsed {len(nodes)} nodes")

    return nodes

def mk_dataset():
    train_nodes = load_corpus(TRAIN_FILES, verbose=True)
    val_nodes = load_corpus(VAL_FILES, verbose=True)

    #ollama本地模型
    ollm = Ollama(model="qwen2:7b-instruct-q4_0", request_timeout=120.0)


    train_dataset = generate_qa_embedding_pairs(
        llm=ollm, nodes=train_nodes
    )
    val_dataset = generate_qa_embedding_pairs(
        llm=ollm, nodes=val_nodes
    )

    train_dataset.save_json(TRAIN_CORPUS_FPATH)
    val_dataset.save_json(VAL_CORPUS_FPATH)


mk_dataset()
```

运行嵌入微调

加载训练集和验证集，使用SentenceTransformer微调，llama_index工具链全支持

```
from llama_index.finetuning import SentenceTransformersFinetuneEngine
from llama_index.core.evaluation import EmbeddingQAFinetuneDataset

def finetune_embedding_model():
    #加载训练集和验证集
    train_dataset = EmbeddingQAFinetuneDataset.from_json(TRAIN_CORPUS_FPATH)
    val_dataset = EmbeddingQAFinetuneDataset.from_json(VAL_CORPUS_FPATH)
    finetune_engine = SentenceTransformersFinetuneEngine(
        train_dataset,   #训练集
        model_id="BAAI/bge-small-zh-v1.5",   #底模
        model_output_path="zhengquan",
        val_dataset=val_dataset,  #验证集
    )

    finetune_engine.finetune()   #直接微调

    embed_model = finetune_engine.get_finetuned_model()


    print(embed_model)


finetune_embedding_model()
```

评估微调模型

定义 eval 函数

```
from llama_index.core import VectorStoreIndex
from llama_index.core.schema import TextNode
from tqdm.notebook import tqdm
import pandas as pd


#评估命中率，这儿只计算没次检索有没有命中，命中率的计算用pandas函数计算出来
def evaluate(
    dataset,
    embed_model,
    top_k=5,
    verbose=False,
):
    corpus = dataset.corpus
    queries = dataset.queries
    relevant_docs = dataset.relevant_docs

    #把评估集语料，用我们要评估的模型embedding好，让后用评估的 query来检索
    nodes = [TextNode(id_=id_, text=text) for id_, text in corpus.items()]
    index = VectorStoreIndex(
        nodes, embed_model=embed_model, show_progress=True
    )
    retriever = index.as_retriever(similarity_top_k=top_k)

    eval_results = []
    for query_id, query in tqdm(queries.items()):
        retrieved_nodes = retriever.retrieve(query)  #让后用评估的 query来检索
        retrieved_ids = [node.node.node_id for node in retrieved_nodes]   #生成一
个检索出来的 文档id 列表
        expected_id = relevant_docs[query_id][0]   #从验证数据集找到查询对应的验证文档
id
        is_hit = expected_id in retrieved_ids  # 看验证文档id是否在 检索出的相关文档列
表里，是否命中

        eval_result = {
            "is_hit": is_hit,
            "retrieved": retrieved_ids,
            "expected": expected_id,
            "query": query_id,
        }
        eval_results.append(eval_result)
    return eval_results
```

选项 2：我们使用`InformationRetrievalEvaluator`来自 sentence_transformers。

```
from sentence_transformers.evaluation import InformationRetrievalEvaluator
from sentence_transformers import SentenceTransformer
from pathlib import Path

#用sentence_transformers带的功能评估
def evaluate_st(
    dataset,
    model_id,
    name,
):
    corpus = dataset.corpus
    queries = dataset.queries
    relevant_docs = dataset.relevant_docs

    #建立的评估器
    evaluator_st = InformationRetrievalEvaluator(
        queries, corpus, relevant_docs, name=name
    )
    model = SentenceTransformer(model_id)   #要评估的embedding模型
    output_path = "results/"
    Path(output_path).mkdir(exist_ok=True, parents=True)
    return evaluator_st(model, output_path=output_path)   #用前面建立的这个
```

评估OpenAI的嵌入模型

```
from llama_index.embeddings.openai import OpenAIEmbedding
import pandas as pd

#创建openai 的 embedding模型
ada = OpenAIEmbedding()
ada_val_results = evaluate(val_dataset, ada)  #对验证集评估

df_ada = pd.DataFrame(ada_val_results)

hit_rate_ada = df_ada["is_hit"].mean()  #计算了依稀 0,1的平均值，也能当做命中率
print(hit_rate_ada)
```

评估BAAI/bge-small-zh-v1.5模型

```
val_dataset = EmbeddingQAFinetuneDataset.from_json(VAL_CORPUS_FPATH)
bge_val_results = evaluate_st(val_dataset, "BAAI/bge-small-zh-v1.5", name="bge-
zh")    #评估开源embedding模型
```

评估自己微调的模型

```
val_dataset = EmbeddingQAFinetuneDataset.from_json(VAL_CORPUS_FPATH)
val_results_finetuned =evaluate_st(val_dataset, "zhengquan", name="finetuned")
 #评估自己微调的模型
```

结果摘要

```
df_st_bge = pd.read_csv(
    "results/Information-Retrieval_evaluation_bge_results.csv"
)
df_st_finetuned = pd.read_csv(
    "results/Information-Retrieval_evaluation_finetuned_results.csv"
)

df_st_bge["model"] = "bge"
df_st_finetuned["model"] = "fine_tuned"
df_st_all = pd.concat([df_st_bge, df_st_finetuned])
df_st_all = df_st_all.set_index("model")
print(df_st_all)
```

利用autotrain来微调embedding模型

项目地址：https://github.com/huggingface/autotrain-advanced

autotrain文档地址：https://huggingface.co/docs/autotrain/index

环境准备

```
#conda创建  python=3.10版本的虚环境
conda create -n autotrain python=3.10
#激活conda创建的名字叫autotrain的虚环境
conda activate autotrain


#torch安装
conda install pytorch==2.3.1 torchvision==0.18.1 torchaudio==2.3.1 pytorch-
cuda=12.1 -c pytorch -c nvidia


#安装autotrain-advanced
pip install autotrain-advanced
```

安装git 和git-lfs

git下载地址：https://git-scm.com/downloads

如果是windows系统，则不需要单独安装git-lfs，git安装包有带

git-lfs下载地址：https://github.com/git-lfs/git-lfs/releases

git-lfs安装教程：https://github.com/git-lfs/git-lfs/wiki/Installation

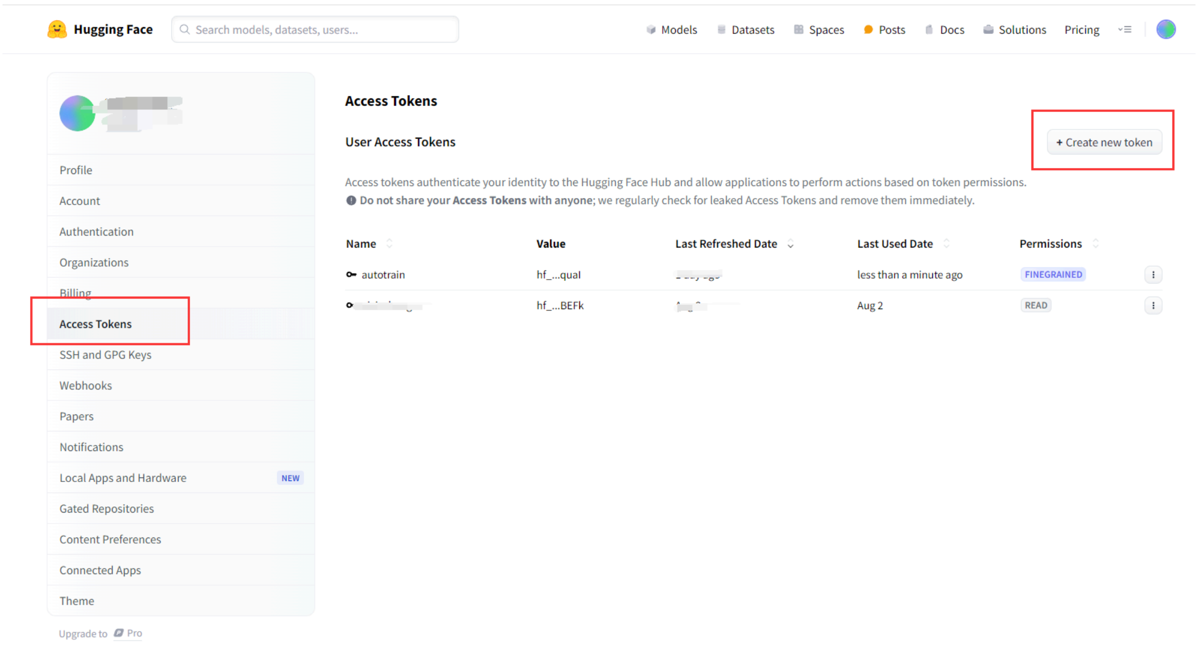
git-lfs用来下载大文件

HF_TOKEN

注册登录huggingface

获取HF_TOKEN ，在这个页面：https://huggingface.co/settings/tokens

添加环境变量HF_TOKEN



Hugging Face | Search models, datasets, users... | Models | Datasets | Spaces | Posts | Docs | Solutions | Pricing

[redacted]

- Profile
- Account
- Authentication
- Organizations
- Billing
- Access Tokens
- SSH and GPG Keys
- Webhooks
- Papers
- Notifications
- Local Apps and Hardware NEW
- Gated Repositories
- Content Preferences
- Connected Apps
- Theme

Upgrade to Pro

Access Tokens

User Access Tokens + Create new token

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions.
Do not share your Access Tokens with anyone; we regularly check for leaked Access Tokens and remove them immediately.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
autotrain	hf_...qual	[redacted]	less than a minute ago	FINEGRAINED
[redacted]	hf_...BEFk	[redacted]	Aug 2	READ

这一步没做会报错误：

```
{"detail":"Invalid or expired token"}
```

启动

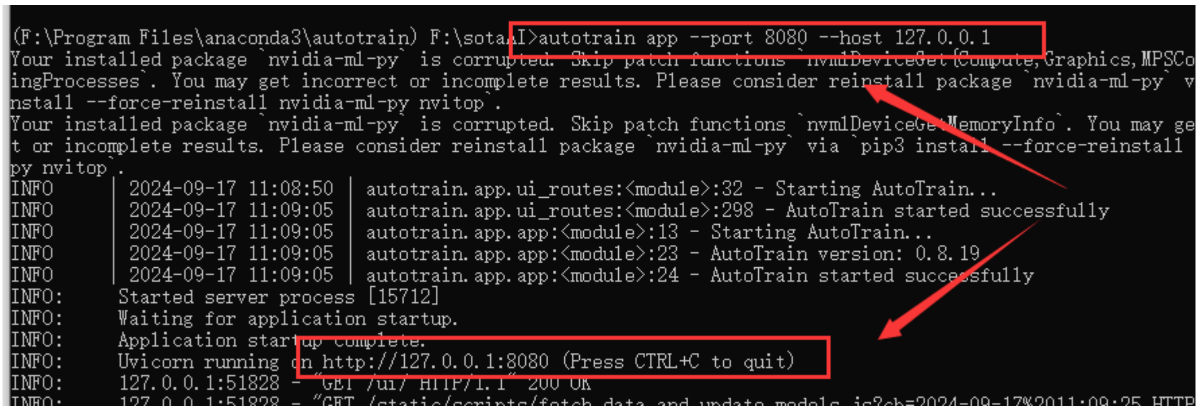
可以用下面的命令启动autorain webui界面

```
autotrain app --port 8080 --host 127.0.0.1
```

也可以用下面的命令来使用autotrain

```
autotrain --config <path_to_config_file>
```

启动后cmd界面：



```
(F:\Program Files\anaconda3\autotrain) F:\sotaAI>autotrain app --port 8080 --host 127.0.0.1
Your installed package `nvidia-ml-py` is corrupted. Skip patch functions `nvmlDeviceGet{Compute,Graphics,MPSCo
ingProcesses`. You may get incorrect or incomplete results. Please consider reinstall package `nvidia-ml-py` v
nstall --force-reinstall nvidia-ml-py nvitop`.
Your installed package `nvidia-ml-py` is corrupted. Skip patch functions `nvmlDeviceGetMemoryInfo`. You may ge
t or incomplete results. Please consider reinstall package `nvidia-ml-py` via `pip3 install --force-reinstall
py nvitop`.
INFO     | 2024-09-17 11:08:50 | autotrain.app.ui_routes:<module>:32 - Starting AutoTrain...
INFO     | 2024-09-17 11:09:05 | autotrain.app.ui_routes:<module>:298 - AutoTrain started successfully
INFO     | 2024-09-17 11:09:05 | autotrain.app.app:<module>:13 - Starting AutoTrain...
INFO     | 2024-09-17 11:09:05 | autotrain.app.app:<module>:23 - AutoTrain version: 0.8.19
INFO     | 2024-09-17 11:09:05 | autotrain.app.app:<module>:24 - AutoTrain started successfully
INFO:     Started server process [15712]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8080 (Press CTRL+C to quit)
INFO:     127.0.0.1:51828 - "GET /ui/ HTTP/1.1" 200 OK
INFO:     127.0.0.1:51828 - "GET /static/scripts/fetch_data_and_update_models.js?cb=2024-09-17%2011:09:25 HTTP
```

复制粘贴就可以在浏览器打开

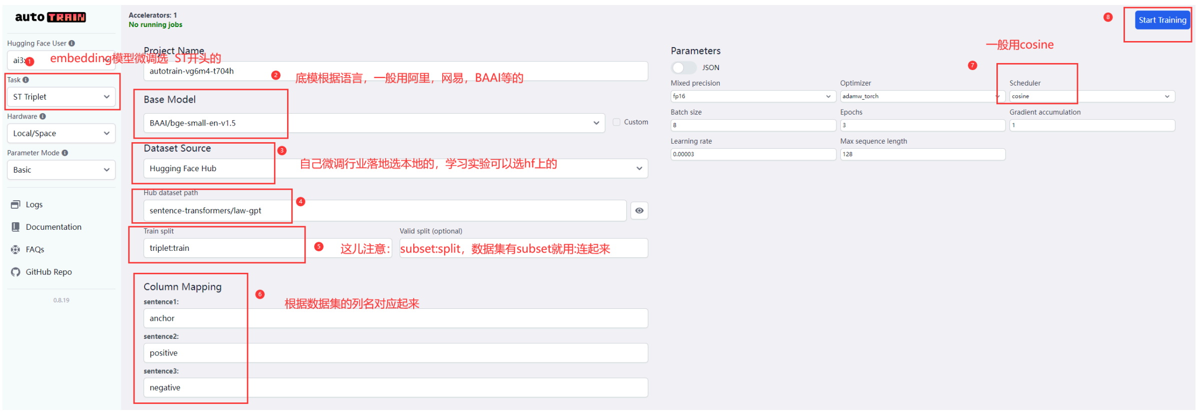
训练

实时autotrain微调embedding模型，选择ST开头的

教程：https://huggingface.co/blog/abhishek/finetune-custom-embeddings-autotrain

SentenceTransformer微调

设置可以参考下图


Accelerators: 1
No running jobs

Hugging Face User: ai3x3
Task: ST Triplet
Hardware: Local/Space
Parameter Mode: Basic

Project Name: autotrain-vg6m4-t704h
Base Model: BAAI/bge-small-en-v1.5
Dataset Source: Hugging Face Hub
Hub dataset path: sentence-transformers/law-gpt
Train split: triplet:train
Valid split (optional):

Column Mapping
sentence1: anchor
sentence2: positive
sentence3: negative

Parameters
Mixed precision: fp16
Optimizer: adamw_torch
Scheduler: cosine
Batch size: 8
Epochs: 3
Gradient accumulation: 1
Learning rate: 0.00003
Max sequence length: 128

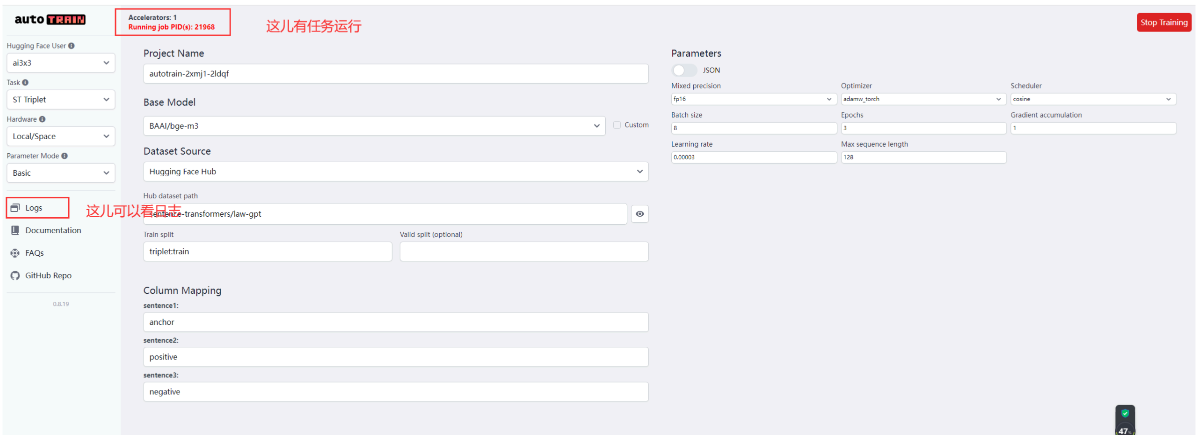
1 embedding模型微调选 ST开头的
2 底模根据语言，一般用阿里，网易，BAAI等的
3 自己微调行业落地选本地的，学习实验可以选hf上的
4
5 这儿注意：subset:split，数据集有subset就用:连起来
6 根据数据集的列名对应起来
7 一般用cosine
8

注意：

llamaindex中 generate_qa_embedding_pairs生成的数据集，autotrain是没法用的，格式不兼容

在train split栏目，如果线上数据集有subset，则采用subset:split的格式填写，如上图

训练过程中，界面上可以看日志，以及确认有没有任务运行


Accelerators: 1
Running job PID(s): 21968

这儿有任务运行

Hugging Face User: ai3x3
Task: ST Triplet
Hardware: Local/Space
Parameter Mode: Basic

这儿可以看日志

Project Name: autotrain-2xmj1-2ldqf
Base Model: BAAI/bge-m3
Dataset Source: Hugging Face Hub
Hub dataset path: sentence-transformers/law-gpt
Train split: triplet:train
Valid split (optional):

Column Mapping
sentence1: anchor
sentence2: positive
sentence3: negative

Parameters
Mixed precision: fp16
Optimizer: adamw_torch
Scheduler: cosine
Batch size: 8
Epochs: 3
Gradient accumulation: 1
Learning rate: 0.00003
Max sequence length: 128

使用config.yml训练

也可以使用配置文件，比如新建一个config.yml

```yaml
task: sentence-transformers:triplet
base_model: microsoft/mpnet-base
project_name: autotrain-st-triplet
log: tensorboard
```

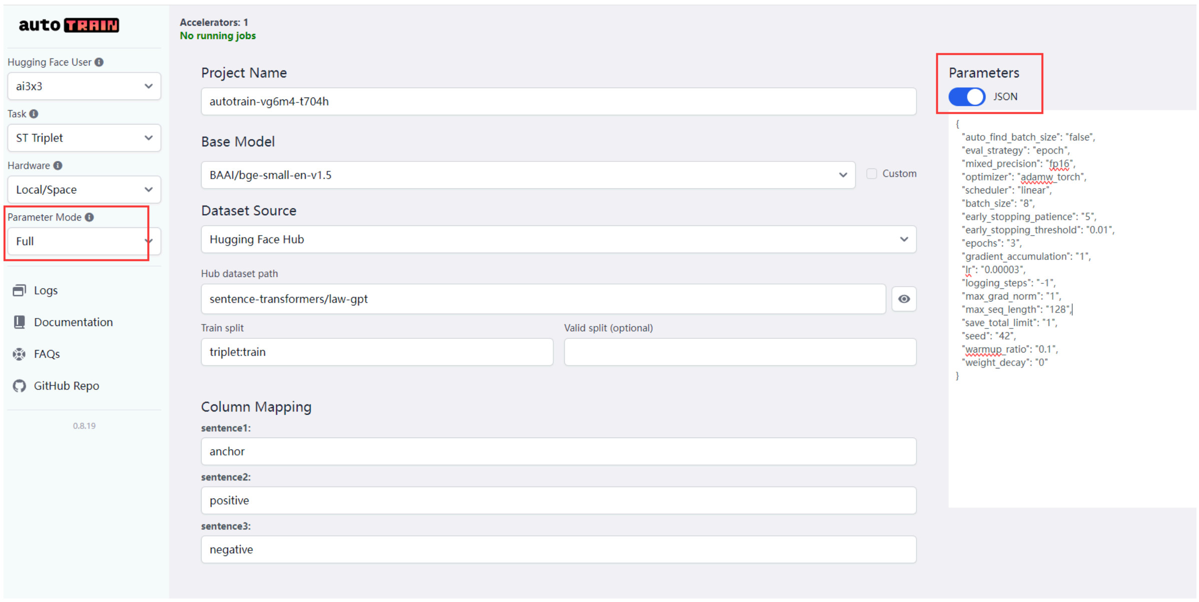
```
backend: local

data:
  path: sentence-transformers/all-nli
  train_split: triplet:train
  valid_split: triplet:dev
  column_mapping:
    sentence1_column: anchor
    sentence2_column: positive
    sentence3_column: negative

params:
  max_seq_length: 512
  epochs: 5
  batch_size: 8
  lr: 2e-5
  optimizer: adamw_torch
  scheduler: linear
  gradient_accumulation: 1
  mixed_precision: fp16

hub:
  username: ${HF_USERNAME}
  token: ${HF_TOKEN}
  push_to_hub: true
```

params参数，可以在webui界面复制出来



Accelerators: 1
No running jobs

Hugging Face User: ai3x3
Task: ST Triplet
Hardware: Local/Space
Parameter Mode: Full

Project Name: autotrain-vg6m4-t704h
Base Model: BAAI/bge-small-en-v1.5 ☐ Custom
Dataset Source: Hugging Face Hub
Hub dataset path: sentence-transformers/law-gpt
Train split: triplet:train
Valid split (optional):

Column Mapping
sentence1: anchor
sentence2: positive
sentence3: negative

Parameters
JSON

```
{
  "auto_find_batch_size": "false",
  "eval_strategy": "epoch",
  "mixed_precision": "fp16",
  "optimizer": "adamw_torch",
  "scheduler": "linear",
  "batch_size": "8",
  "early_stopping_patience": "5",
  "early_stopping_threshold": "0.01",
  "epochs": "3",
  "gradient_accumulation": "1",
  "lr": "0.00003",
  "logging_steps": "-1",
  "max_grad_norm": "1",
  "max_seq_length": "128",
  "save_total_limit": "1",
  "seed": "42",
  "warmup_ratio": "0.1",
  "weight_decay": "0"
}
```

让后使用命令

```
autotrain --config <path_to_config_file>
```

使用MRR评测

llamaindex文档：https://docs.llamaindex.ai/en/stable/api_reference/

语料库我们都将使用欧洲人工智能法案：https://artificialintelligenceact.eu/，这个语料库是世界上第一个关于人工智能的法律框架，支持24种语言版本，这使得比较不同语言族的数据检索的准确性成为可能

环境准备

这儿embedding评估环境和前面llamaindex进行embedding微调的一样，我们复习一遍

```
#conda创建  python=3.10版本的虚环境
#conda create -n rag_embedding python=3.10
#激活conda创建的名字叫rag_embedding的虚环境
conda activate rag_embedding



#torch安装
#conda install pytorch==2.3.1 torchvision==0.18.1 torchaudio==2.3.1 pytorch-cuda=12.1 -c pytorch -c nvidia



#安装依赖
#如果用openai模型来生成微调数据集
pip install llama-index-llms-openai
pip install llama-index-embeddings-openai

#如果是本地部署ollama
pip install llama-index-llms-ollama

#安装依赖
pip install llama-index-finetuning
pip install llama-index-embeddings-huggingface
pip install -U llama-index    #要按照最新的llama-index

pip install llama-index-readers-web



#评测模型有量化
pip install accelerate
pip install https://github.com/jllllll/bitsandbytes-windows-webui/releases/download/wheels/bitsandbytes-0.41.2.post2-py3-none-win_amd64.whl
```

生成自定义Q/A数据集

如下面的代码所示：

```
from llama_index.readers.web import SimpleWebPageReader
from llama_index.core.node_parser import SentenceSplitter

#法案文档下载地址
#https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689
language = "EN"
url_doc = "https://eur-lex.europa.eu/legal-content/"+language+"/TXT/HTML/?
uri=CELEX:32024R1689"

documents = SimpleWebPageReader(html_to_text=True).load_data([url_doc])

parser = SentenceSplitter(chunk_size=1000)    #这儿每指定 chunk_overlap 重叠部分大小
nodes = parser.get_nodes_from_documents(documents, show_progress=True)
```

下载欧盟24种官方语言的欧盟人工智能法案链接(https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689)

生成每个块假设问题（reverse HyDE）的提示词模版：

```
prompts={}
prompts["EN"] = """\
Context information is below.

---------------------
{context_str}
---------------------

Given the context information and not prior knowledge, generate only questions
based on the below query.

You are a Teacher/ Professor. Your task is to setup {num_questions_per_chunk}
questions for an upcoming quiz/examination.
The questions should be diverse in nature across the document. Restrict the
questions to the context information provided."
"""
```

然后可以通过调用LlamaIndex库中的`generate_qa_embedding_pairs`来生成问题:

```
#from llama_index.llms import OpenAI
from llama_index.finetuning import generate_qa_embedding_pairs
from llama_index.llms.ollama import Ollama

#使用ollama服务
ollm = Ollama(model="qwen2:7b-instruct-q4_0", request_timeout=120.0)

qa_dataset = generate_qa_embedding_pairs(
    llm=ollm,     #OpenAI(model="gpt-3.5-turbo-0125",additional_kwargs=
{'seed':42}) ,这儿使用ollama本地部署的模型
    nodes=nodes,
    qa_generate_prompt_tmpl = prompts[language],
    num_questions_per_chunk=2
)
```

openai embedding模型评价

闭源模型主要是openai的 embedding模型：https://platform.openai.com/docs/models/embeddings

openai embedding 模型信息：https://openai.com/index/new-embedding-models-and-api-updates/

llamaindex embedding模型微调文档：https://docs.llamaindex.ai/en/stable/examples/finetuning/embeddings/finetune_embedding

mrr评估函数

完整代码

openai embedding模型 多种语言 批量mrr评测的完整代码

```
from llama_index.readers.web import SimpleWebPageReader
from llama_index.core.node_parser import SentenceSplitter

from llama_index.core import VectorStoreIndex
from llama_index.core.schema import TextNode
from tqdm.notebook import tqdm

from llama_index.embeddings.openai import OpenAIEmbedding
import pandas as pd
import numpy as np
#from llama_index.llms import OpenAI
from llama_index.finetuning import generate_qa_embedding_pairs
from llama_index.llms.ollama import Ollama

from llama_index.core.evaluation import (
    generate_question_context_pairs,
    EmbeddingQAFinetuneDataset,
)

#准备qa对的数据集
def prepare_data():
    #法案文档下载地址
    #https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689
    language = "EN"
    url_doc = "https://eur-lex.europa.eu/legal-content/"+language+"/TXT/HTML/?
uri=CELEX:32024R1689"

    documents = SimpleWebPageReader(html_to_text=True).load_data([url_doc])

    parser = SentenceSplitter(chunk_size=1000)    #这儿每指定 chunk_overlap 重叠部分
大小
    nodes = parser.get_nodes_from_documents(documents, show_progress=True)


    prompts={}
    prompts["EN"] = """\
    Context information is below.
    ---------------------
    {context_str}
    ---------------------

    Given the context information and not prior knowledge, generate only
questions based on the below query.

    You are a Teacher/ Professor. Your task is to setup {num_questions_per_chunk}
questions for an upcoming quiz/examination.
    The questions should be diverse in nature across the document. Restrict the
questions to the context information provided."
    """



    #使用ollama服务
    ollm = Ollama(model="qwen2:7b-instruct-q4_0", request_timeout=120.0)

    qa_dataset = generate_qa_embedding_pairs(
        llm=ollm,    #OpenAI(model="gpt-3.5-turbo-0125",additional_kwargs=
{'seed':42}) ,这儿使用ollama本地部署的模型
        nodes=nodes,
        qa_generate_prompt_tmpl = prompts[language],
        num_questions_per_chunk=2
    )

    return qa_dataset




#自定义的mrr评估函数
def evaluate(dataset, embed_model, insert_batch_size=1000, top_k=5):
    # 数据集中的语料，查询，与查询相关的文档
    corpus = dataset.corpus
    queries = dataset.queries
    relevant_docs = dataset.relevant_docs

    # 用要评估的模型生成 index
    nodes = [TextNode(id_=id_, text=text) for id_, text in corpus.items()]
    index = VectorStoreIndex(
        nodes, embed_model=embed_model)
    retriever = index.as_retriever(similarity_top_k=top_k)   #检索返回相似的topk

    # Prepare to collect evaluation results
    eval_results = []

    # 对每个查询检索
    for query_id, query in tqdm(queries.items()):
        # 检索出相关文档  topk
        retrieved_nodes = retriever.retrieve(query)
        retrieved_ids = [node.node.node_id for node in retrieved_nodes]   #相关文
档 id  list

        # 相关文档的 id
        expected_id = relevant_docs[query_id][0]
        is_hit = expected_id in retrieved_ids  # 看文档是否命中

        # 命中了就计算 mrr，没命中就是0
        if is_hit:
            rank = retrieved_ids.index(expected_id) + 1
            mrr = 1 / rank
        else:
            mrr = 0
        eval_results.append(mrr)

    return np.average(eval_results)   #求了平均值



#对多个模型和语言进行评估
def embedding_evaluation():

    embeddings_model_spec = {
    }

    embeddings_model_spec['OAI-Large-256']={'model_name':'text-embedding-3-
large','dimensions':256}
    embeddings_model_spec['OAI-Large-3072']={'model_name':'text-embedding-3-
large','dimensions':3072}
    embeddings_model_spec['OAI-Small']={'model_name':'text-embedding-3-
small','dimensions':1536}
    embeddings_model_spec['OAI-ada-002']={'model_name':'text-embedding-ada-
002','dimensions':None}

    results = []

    languages = ["EN", "FR", "CS", "HU"]

    # Loop through all languages
    for language in languages:

        # Load dataset
        file_name=language+"_dataset.json"


        qa_dataset = EmbeddingQAFinetuneDataset.from_json(file_name)

        # Loop through all models
        for model_name, model_spec in embeddings_model_spec.items():

            #闭源模型和开源模型 ,模型加载方式不一样
            embed_model = OpenAIEmbedding(model=model_spec['model_name'],
                                          dimensions=model_spec['dimensions'])

            # Assess embedding score (in terms of MRR)
            score = evaluate(qa_dataset, embed_model)

            results.append([language, model_name, score])

    df_results = pd.DataFrame(results, columns = ["Language" ,"Embedding model",
"MRR"])
```

```
    print(df_results)



qa_dataset=prepare_data()
embedding_evaluation()

#龙 哥抖音号：龙 哥紫 貂智能
```

性能总结

OpenAI embedding模型的性能，详见官方数据：https://openai.com/blog/new-embedding-models-and-api-updates

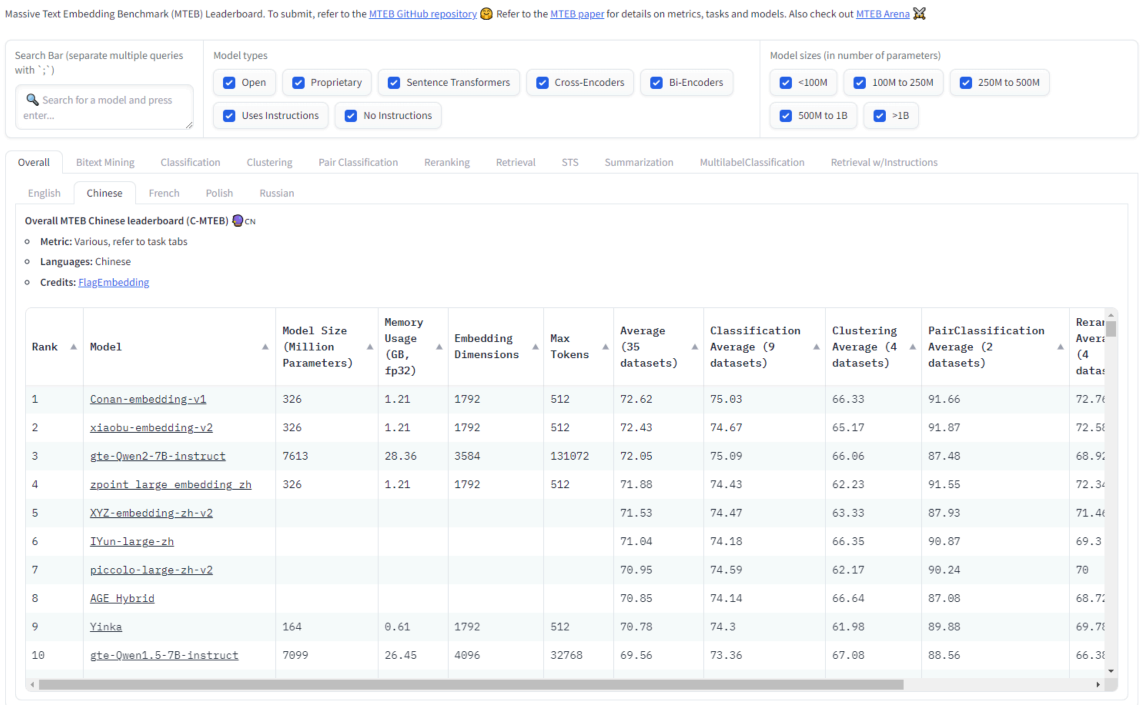
开源embedding模型的评估

了解最新发布的开源embedding模型可以到

https://huggingface.co/spaces/mteb/leaderboard

选择开源embedding模型

- gte-large-en-v1.5(https://huggingface.co/Alibaba-NLP/gte-large-en-v1.5) ：支持高达**8192**的上下文长度，同时进一步增强了模型性能。模型建立在`transformer++` encoder 主干网络（BERT + RoPE + GLU）之上
- ML-E5-large (https://huggingface.co/intfloat/multilingual-e5-large-instruct)： 微软的另一个E5模型，旨在更好地处理多语言数据。它从[xlm-roberta-large](xlm-roberta-large)初始化，并在多语言数据集的混合上进行训练。它比E5-Mistral小得多(10倍)，但上下文大小也小得多(514)。
- bge-large-en-v1.5(https://huggingface.co/BAAI/bge-large-en-v1.5)：该模型由北京人工智能研究院设计，是他们最先进的多语言数据embedding模型，支持100多种工作语言。截至2024年2月22日，它还没有进入MTEB排行榜。
- Nomic- embed（https://huggingface.co/nomic-ai/nomic-embed-text-v1）：该模型声称性能优于OpenAI Ada-002和text- embeding3 -small，而大小仅为0.55GB。该模型是第一个开放数据和开源训练代码。
- BGE-M3(https://huggingface.co/BAAI/bge-m3)：该模型由北京人工智能研究院设计，是他们最先进的多语言数据embedding模型，支持100多种工作语言。截至2024年2月22日，它还没有进入MTEB排行榜。


Massive Text Embedding Benchmark (MTEB) Leaderboard. To submit, refer to the MTEB GitHub repository 🤗 Refer to the MTEB paper for details on metrics, tasks and models. Also check out MTEB Arena ⚔️

Overall MTEB Chinese leaderboard (C-MTEB) CN
- Metric: Various, refer to task tabs
- Languages: Chinese
- Credits: FlagEmbedding

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (35 datasets)	Classification Average (9 datasets)	Clustering Average (4 datasets)	PairClassification Average (2 datasets)	Rera Avera (4 datas
1	Conan-embedding-v1	326	1.21	1792	512	72.62	75.03	66.33	91.66	72.7
2	xiaobu-embedding-v2	326	1.21	1792	512	72.43	74.67	65.17	91.87	72.5
3	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	72.05	75.09	66.06	87.48	68.9
4	zpoint_large_embedding_zh	326	1.21	1792	512	71.88	74.43	62.23	91.55	72.3
5	XYZ-embedding-zh-v2					71.53	74.47	63.33	87.93	71.4
6	IYun-large-zh					71.04	74.18	66.35	90.87	69.3
7	piccolo-large-zh-v2					70.95	74.59	62.17	90.24	70
8	AGE_Hybrid					70.85	74.14	66.64	87.08	68.7
9	Yinka	164	0.61	1792	512	70.78	74.3	61.98	89.88	69.7
10	gte-Qwen1.5-7B-instruct	7099	26.45	4096	32768	69.56	73.36	67.08	88.56	66.3

完整代码

```
from llama_index.readers.web import SimpleWebPageReader
from llama_index.core.node_parser import SentenceSplitter

from llama_index.core import VectorStoreIndex
from llama_index.core.schema import TextNode
from tqdm.notebook import tqdm

from llama_index.embeddings.openai import OpenAIEmbedding
import pandas as pd
import numpy as np
import time
#from llama_index.llms import OpenAI
from llama_index.finetuning import generate_qa_embedding_pairs
from llama_index.llms.ollama import Ollama

import torch
from transformers import AutoTokenizer,AutoModel
from sentence_transformers import SentenceTransformer
from llama_index.embeddings.huggingface import HuggingFaceEmbedding

from llama_index.core.evaluation import (
    generate_question_context_pairs,
    EmbeddingQAFinetuneDataset,
)

#准备qa对的数据集
def prepare_data():
    #法案文档下载地址
    #https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689
    language = "EN"
    url_doc = "https://eur-lex.europa.eu/legal-content/"+language+"/TXT/HTML/?
uri=CELEX:32024R1689"

    documents = SimpleWebPageReader(html_to_text=True).load_data([url_doc])

    parser = SentenceSplitter(chunk_size=1000)   #这儿每指定 chunk_overlap 重叠部分
大小
    nodes = parser.get_nodes_from_documents(documents, show_progress=True)


    prompts={}
    prompts["EN"] = """\
    Context information is below.

    ---------------------
    {context_str}
    ---------------------

    Given the context information and not prior knowledge, generate only
questions based on the below query.

    You are a Teacher/ Professor. Your task is to setup {num_questions_per_chunk}
questions for an upcoming quiz/examination.
    The questions should be diverse in nature across the document. Restrict the
questions to the context information provided."
    """


    #使用ollama服务
    ollm = Ollama(model="qwen2:7b-instruct-q4_0", request_timeout=120.0)

    qa_dataset = generate_qa_embedding_pairs(
        llm=ollm,    #OpenAI(model="gpt-3.5-turbo-0125",additional_kwargs=
{'seed':42}) ,这儿使用ollama本地部署的模型
        nodes=nodes,
        qa_generate_prompt_tmpl = prompts[language],
        num_questions_per_chunk=2
    )
    qa_dataset.save_json("EN_dataset.json")
    return qa_dataset


#自定义的mrr评估函数
def evaluate(dataset, embed_model, insert_batch_size=1000, top_k=5):
    # 数据集中的语料，查询，与查询相关的文档
    corpus = dataset.corpus
    queries = dataset.queries
    relevant_docs = dataset.relevant_docs

    # 用要评估的模型生成 index
    nodes = [TextNode(id_=id_, text=text) for id_, text in corpus.items()]
    index = VectorStoreIndex(
        nodes, embed_model=embed_model)
    retriever = index.as_retriever(similarity_top_k=top_k)   #检索返回相似的topk

    # Prepare to collect evaluation results
    eval_results = []

    # 对每个查询检索
    for query_id, query in tqdm(queries.items()):
        # 检索出相关文档  topk
        retrieved_nodes = retriever.retrieve(query)
        retrieved_ids = [node.node.node_id for node in retrieved_nodes]   #相关文
档 id  list

        # 相关文档的 id
        expected_id = relevant_docs[query_id][0]
        is_hit = expected_id in retrieved_ids  # 看文档是否命中

        # 命中了就计算 mrr，没命中就是0
        if is_hit:
            rank = retrieved_ids.index(expected_id) + 1
            mrr = 1 / rank
        else:
            mrr = 0
        eval_results.append(mrr)

    return np.average(eval_results)   #求了平均值



#ebedding模型评估
def embedding_evaluation():

    embeddings_model_spec = {
    }

    embeddings_model_spec['gte-large-en']={'model_name':'Alibaba-NLP/gte-large-
en-v1.5','max_length':32768, 'pooling_type':'last_token',
                                            'normalize': True, 'batch_size':1,
'kwargs': {'load_in_4bit':True, 'bnb_4bit_compute_dtype':torch.float16}}
    embeddings_model_spec['ML-E5-large']={'model_name':'intfloat/multilingual-e5-
large','max_length':512, 'pooling_type':'mean',
                                        'normalize': True, 'batch_size':1,
'kwargs': {'device_map': 'cuda', 'torch_dtype':torch.float16}}
    embeddings_model_spec['bge-large-en']={'model_name':'BAAI/bge-large-en-
v1.5','max_length':8192, 'pooling_type':'cls',
                                    'normalize': True, 'batch_size':1, 'kwargs':
{'device_map': 'cuda', 'torch_dtype':torch.float16}}
    embeddings_model_spec['Nomic-Embed']={'model_name':'nomic-ai/nomic-embed-
text-v1','max_length':8192, 'pooling_type':'mean',
                                        'normalize': True, 'batch_size':1,
'kwargs': {'device_map': 'cuda', 'trust_remote_code' : True}}
    embeddings_model_spec['BGE-M3']={'model_name':'BAAI/bge-
m3','max_length':8192, 'pooling_type':'cls',
                                 'normalize': True, 'batch_size':1, 'kwargs':
{'device_map': 'cuda', 'torch_dtype':torch.float16}}

    results = []

    #languages = ["EN", "FR", "CS", "HU"]
    languages=["EN"]
    # Loop through all models
    for model_name, model_spec in embeddings_model_spec.items():

        print("Processing model : "+str(model_spec))

        # Get model
        #tokenizer = AutoTokenizer.from_pretrained(model_spec['model_name'])
        #embed_model =
AutoModel.from_pretrained(model_spec['model_name'],trust_remote_code=True,
**model_spec['kwargs'])
        #embed_model = SentenceTransformer(model_spec['model_name'],
trust_remote_code=True)

        #model_name=model_spec['model_name']
        device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

        #闭源模型和开源模型 ,模型加载方式不一样
        embed_model = HuggingFaceEmbedding(model_name=model_spec['model_name'],
trust_remote_code=True,device=device)


        # Loop through all languages
        for language in languages:

            # Load dataset
            file_name=language+"_dataset.json"
            qa_dataset = EmbeddingQAFinetuneDataset.from_json(file_name)

            start_time_assessment=time.time()

            # Assess embedding score (in terms of hit rate at k=5)
            #score = evaluate(qa_dataset,  embed_model, model_spec['normalize'],
model_spec['max_length'], model_spec['pooling_type'])
            score = evaluate(qa_dataset,embed_model)

            # Get duration of score assessment
            duration_assessment = time.time()-start_time_assessment

            results.append([language, model_name, score, duration_assessment])

    df_results = pd.DataFrame(results, columns = ["Language" ,"Embedding model",
"MRR", "Duration"])
    print(df_results)



qa_dataset=prepare_data()

embedding_evaluation()


#龙 哥抖音号：龙 哥紫 貂智能
```

以MRR表示的结果准确性报告如下：

```
  Language Embedding model       MRR   Duration

                                                            | 0/1 [00:00<?, ?it/s]
0        EN    gte-large-en  0.404409  41.133034
1        EN     ML-E5-large  0.420323  28.296962
2        EN    bge-large-en  0.395161  27.659480
3        EN     Nomic-Embed  0.367258  20.260832
```

使用MTEB来评估

RAG 评测数据集建设尚处于初期阶段，缺乏针对特定领域和场景的专业数据集

mteb简介

目前最权威的检索榜单是 `HuggingFace MTEB` （Massive Text Embedding Benchmark）

论文：https://arxiv.org/pdf/2210.07316

说明：https://huggingface.co/blog/mteb

项目：https://github.com/embeddings-benchmark/mteb

榜单：https://huggingface.co/spaces/mteb/leaderboard

C-MTEB

论文：https://arxiv.org/pdf/2309.07597

我们用mteb来评测一下之前微调的 zhengquan 模型

环境准备

```
#和前面的embedding微调共用一个虚拟环境
conda activate rag_embedding

pip install mteb
```

使用入门

https://github.com/embeddings-benchmark/mteb

```
from mteb import MTEB
from sentence_transformers import SentenceTransformer

# Define the sentence-transformers model name
model_name = "zhengquan"
```

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



#评测结果生成保存在output_folder指定的文件夹下，这儿也就是results 文件夹下


'''
{
  "dataset_revision": "0fd18e25b25c072e09e0d92ab615fda904d66300",
  "evaluation_time": 16.388249158859253,
  "kg_co2_emissions": null,
  "mteb_version": "1.14.15",
  "scores": {
    "test": [
      {
        "accuracy": 0.6078896103896104,
        "f1": 0.5925685324177159,
        "f1_weighted": 0.5925685324177159,
        "hf_subset": "default",
        "languages": [
          "eng-Latn"
        ],
        "main_score": 0.6078896103896104,
        "scores_per_experiment": [
          {
            "accuracy": 0.6152597402597403,
            "f1": 0.6021413335785741,
            "f1_weighted": 0.6021413335785742
          },

        ......

'''
```

会生成一个`results/zhengquan/Banking77Classification.json`文件

可以通过将结果添加到Hub上任何模型的README.md元数据中，将其提交到排行榜。

运行 https://github.com/embeddings-benchmark/mteb/blob/main/mteb/create_meta.py 来生成元数据：

```
python create_meta.py results/zhengquan
```

具体可参考：https://huggingface.co/blog/mteb

也可以使用官方提供的 CLI

```
mteb --available_tasks     #获取支持的评测任务

mteb -m my_embedding \      #要评估的embedding模型
    -t Banking77Classification  \   #评测任务
    --output_folder results \    #评测结果文件夹
    --verbosity 3
```

任务列表

任务列表文档：https://github.com/embeddings-benchmark/mteb/blob/main/docs/tasks.md

任务选择

MTEB支持指定数据集，可以通过下面的形式

按task_type任务类型（例如"聚类"或"分类"）

```
evaluation = MTEB(task_types=['Clustering', 'Retrieval'])      # 选择clustering 和
retrieval 任务
```

按类别划分, 例如"句子到句子 "S2S" (sentence to sentence)， 段落到段落"P2P" (paragraph to paragraph)

```
evaluation = MTEB(task_categories=['S2S'])  # 选择 sentence2sentence datasets
```

按照文本语言

```
evaluation = MTEB(task_langs=["en", "de"]) # 选择  "en", "de" or "en-de" datasets
```

还可以针对数据集选择语言：

```
from mteb.tasks import AmazonReviewsClassification, BUCCBitextMining

evaluation = MTEB(tasks=[
        AmazonReviewsClassification(langs=["en", "fr"]) # Only load "en" and "fr"
subsets of Amazon Reviews
        BUCCBitextMining(langs=["de-en"]), # Only load "de-en" subset of BUCC
])
```

可为某些任务集合提供预设

```
from mteb import MTEB_MAIN_EN
evaluation = MTEB(tasks=MTEB_MAIN_EN, task_langs=["en"])
```

有的数据集有多个split，评测会比较耗时，可以指定split，来减少评测时间，比如下面的就指定了只用test split。

```
evaluation.run(model, eval_splits=["test"])
```

自定义评测模型

可以参考仓库 https://github.com/embeddings-benchmark/mteb。

```
class MyModel():
    def __init__(self, use_gpu=True):
        '''
        providers = ['CUDAExecutionProvider'] if use_gpu else
['CPUExecutionProvider']
        sess_options = ort.SessionOptions()
        self.predictor = ort.InferenceSession(
            model_path, sess_options=sess_options, providers=providers)
        self.tokenizer = AutoTokenizer.from_pretrained(tokenizer_path)
        '''
        #加载模型


    def encode(self, sentences, batch_size=32, **kwargs):
        """
        Returns a list of embeddings for the given sentences.
        Args:
            sentences (`List[str]`): List of sentences to encode
            batch_size (`int`): Batch size for the encoding

        Returns:
            `List[np.ndarray]` or `List[tensor]`: List of embeddings for the
given sentences
        """

        '''
        all_embeddings = []
        batch_count = math.ceil(len(sentences) / batch_size)

        for i in tqdm(range(batch_count)):
            # 按batch
            sub_sentences = sentences[i * batch_size : min(len(sentences), (i +
1) * batch_size)]
            features = self.tokenizer(sub_sentences, max_seq_len=128,
                                    pad_to_max_seq_len=True,
truncation_strategy="longest_first")
            vecs = self.predictor.run(None, features.data)
            all_embeddings.extend(vecs[0])
        return all_embeddings
        '''

        #使用自定义的模型生成embedding


        pass
```

```
model = MyModel()
evaluation = MTEB(tasks=["Banking77Classification"])
evaluation.run(model)
```

如果针对query和corpus需要使用不同的encode方法，可以独立提供`encode_queries` and `encode_corpus`两个方法。

```
class MyModel():
    def encode_queries(self, queries, batch_size=32, **kwargs):
        """
        Returns a list of embeddings for the given sentences.
        Args:
            queries (`List[str]`): List of sentences to encode
            batch_size (`int`): Batch size for the encoding

        Returns:
            `List[np.ndarray]` or `List[tensor]`: List of embeddings for the
given sentences
        """
        pass

    def encode_corpus(self, corpus, batch_size=32, **kwargs):
        """
        Returns a list of embeddings for the given sentences.
        Args:
            corpus (`List[str]` or `List[Dict[str, str]]`): List of sentences to
encode
                or list of dictionaries with keys "title" and "text"
            batch_size (`int`): Batch size for the encoding

        Returns:
            `List[np.ndarray]` or `List[tensor]`: List of embeddings for the
given sentences
        """
        pass
```

自定义评测Task（数据集）

要添加一个新任务，你需要实现一个从与任务类型相关的`AbsTask`继承的新类（例如，对于重排任务是`AbsTaskReranking`）

支持的任务类型：https://github.com/embeddings-benchmark/mteb/tree/main/mteb/abstasks

文档：https://github.com/embeddings-benchmark/mteb/blob/main/docs/adding_a_dataset.md

自定义Retrieval任务：

```
from datasets import load_dataset, DatasetDict
from mteb import AbsTaskRetrieval


def load_retrieval_data(hf_hub_name, eval_splits):
    eval_split = eval_splits[0]
    dataset = load_dataset(hf_hub_name)
    qrels = load_dataset(hf_hub_name + '-qrels')[eval_split]

    corpus = {e['id']: {'text': e['text']} for e in dataset['corpus']}
    queries = {e['id']: e['text'] for e in dataset['queries']}
    relevant_docs = defaultdict(dict)
    for e in qrels:
        relevant_docs[e['qid']][e['pid']] = e['score']

    corpus = DatasetDict({eval_split:corpus})
    queries = DatasetDict({eval_split:queries})
    relevant_docs = DatasetDict({eval_split:relevant_docs})
    return corpus, queries, relevant_docs


class T2Retrieval(AbsTaskRetrieval):
    @property
    def description(self):
        return {
            'name': 'T2Retrieval',
            'hf_hub_name': 'C-MTEB/T2Retrieval',
            'reference': 'https://arxiv.org/abs/2304.03679',
            'description': 'T2Ranking: A large-scale Chinese Benchmark for
Passage Ranking',
            'type': 'Retrieval',
            'category': 's2p',          #任务分类
            'eval_splits': ['dev'],     #数据集split
            'eval_langs': ['zh'],       #语言
            'main_score': 'ndcg_at_10', #评测指标
        }

    def load_data(self, **kwargs):
        if self.data_loaded:
            return

        self.corpus, self.queries, self.relevant_docs =
load_retrieval_data(self.description['hf_hub_name'],

 self.description['eval_splits'])
        self.data_loaded = True

model = SentenceTransformer("myembedding")
evaluation = MTEB(tasks=[T2Retrieval()])
evaluation.run(model)
```

自定义Rerank任务：

```python
from mteb import MTEB
from mteb.abstasks.AbsTaskReranking import AbsTaskReranking
from sentence_transformers import SentenceTransformer


class MindSmallReranking(AbsTaskReranking):
    @property
    def description(self):
        return {
            "name": "MindSmallReranking",
            "hf_hub_name": "mteb/mind_small",
            "description": "Microsoft News Dataset: A Large-Scale English Dataset
for News Recommendation Research",
            "reference": "https://www.microsoft.com/en-
us/research/uploads/prod/2019/03/nl4se18LinkSO.pdf",
            "type": "Reranking",
            "category": "s2s",
            "eval_splits": ["validation"],
            "eval_langs": ["en"],
            "main_score": "map",
        }

model = SentenceTransformer("myembedding")
evaluation = MTEB(tasks=[MindSmallReranking()])
evaluation.run(model)
```