

NLP پروژه درس

الف) یک مدل امبدینگ مثل distiluse-base-multilingual-cased-v2، روی داده‌های پروژه Fine-Tune کرده و عملکرد فرآیند بازیابی را بررسی نمایید. سپس میزان تشابه معنایی را با معیارهای Cosine Similarity و MRR محاسبه کنید. (۵)

(نمره)

مراحل انجام پروژه

مرحله‌ی اول:

فایل word را با کتابخانه‌ی مربوطه باز میکنیم تمام خطوط غیرخالی را جدا میکنیم و با یک گرامر خاص مثلاً اینجا در نظر گرفتیم که هر خطی که یک عدد بیاد و بعدش نقطه بیاد بعدی اسم غذاست غذاهارو پیدا میکنیم. البته روش‌های بهتری هم میشد استفاده بشه ولی خود داکیومنت هم مشکل داشت و از یک الگوی خاص پیروی نمیکرد که بشه راحت تمام غذاها و توضیحاتشون رو پیدا کرد.

بعد از پیدا کردن هر غذا یک دیکشنری جدید برای اون غذا میسازیم و از خط‌های بعد به عنوان توضیح اون غذا استفاده میکنیم و در انتهای غذا رو به لیست اضافه میکنیم.

```
from docx import Document
import re
import random

DATA_PATH = "./Guilan-Food.docx"
doc = Document(DATA_PATH)

lines = [p.text.strip() for p in doc.paragraphs if p.text.strip()]

foods = []
cur = None

for line in lines:
    m = re.match(r"\d+\.\s*(.+)$", line)
    if m:
        if cur:
            foods.append(cur)
        cur = {"title": m.group(1).strip(), "content": ""}
    elif cur:
        cur["content"] += line + "\n"
```

```
if cur:  
    foods.append(cur)
```

مرحله ی دوم:

محتوای هر غذا که شامل توضیحات و مواد اولیه و دستور پخت هست به سه بخش جداگانه تقسیم میکنیم و هر کدام از این محتوارو به غذای مربوطه اضافه میکنیم.

```
def split_sections(content: str):  
    ingredients, instructions, desc = "", "", ""  
    section = "desc"  
  
    for line in content.split("\n"):  
        if not line.strip():  
            continue  
        if line.startswith("مواد لازم"):  
            section = "ingredients"  
            continue  
        if line.startswith("طرز تهیه"):  
            section = "instructions"  
            continue  
  
        if section == "desc":  
            desc += line + "\n"  
        elif section == "ingredients":  
            ingredients += line + "\n"  
        elif section == "instructions":  
            instructions += line + "\n"  
  
    return {  
        "desc": desc.strip(),  
        "ingredients": ingredients.strip(),  
        "instructions": instructions.strip()  
    }  
  
for f in foods:  
    parts = split_sections(f["content"])  
    f.update(parts)  
  
print("بعد از split:")  
print("Title:", foods[0]["title"])  
print("Desc:", foods[0]["desc"][:60])
```

```

print("Ingredients:", foods[0]["ingredients"].split("\n")[:3])
print("Instructions:", foods[0]["instructions"][:60], "\n")

```

مرحله ی سوم:

در این بخش از پروژه، با استفاده از قالب‌های متنی (template) برای سه نوع اطلاعات هر غذا—مواد لازم، طرز تهیه و معرفی—مجموعه‌ای از پرسش‌پاسخ‌های مصنوعی و ساخت‌یافته تولید کردیم. برای هر غذا، اگر بخش مربوطه موجود بود، چندین سؤال منطقی با جایگذاری نام غذا در قالب‌ها ساخته شد و پاسخ آن‌ها مستقیماً از محتوای استخراج‌شده غذاها قرار گرفت. نتیجه‌ی این مرحله، دیتابیسی شامل نمونه‌های متنوع Q&A فارسی درباره غذاهای گیلانی است که می‌تواند برای ارزیابی، جستجو یا آموزش مدل‌های زبانی مورد استفاده قرار گیرد.

```

TEMPLATES_ING = [
    "{title} چیست؟ مواد لازم",
    "چه موادی نیاز است؟ {title} برای درست کردن",
    "با چه چیزهایی درست می‌شود؟ {title}"
]

TEMPLATES_INS = [
    "چطور است؟ طرز تهیه",
    "چگونه است؟ {title} روش پخت",
    "را درست کرد؟ {title} چطور می‌توان"
]

TEMPLATES_DESC = [
    "چیست؟",
    "توضیح بده {title} درباره",
    "چه نوع غذایی است؟ {title}"
]

#dataset
pairs = []
for idx, food in enumerate(foods):
    if food["ingredients"]:
        for q in TEMPLATES_ING:
            pairs.append({
                "query": q.format(title=food["title"]),
                "doc_id": idx,
                "answer": food["ingredients"],
                "section": "ingredients"
            })

```

```

if food["instructions"]:
    for q in TEMPLATES_INS:
        pairs.append({
            "query": q.format(title=food["title"]),
            "doc_id": idx,
            "answer": food["instructions"],
            "section": "instructions"
        })
if food["desc"]:
    for q in TEMPLATES_DESC:
        pairs.append({
            "query": q.format(title=food["title"]),
            "doc_id": idx,
            "answer": food["desc"],
            "section": "desc"
        })

```

مرحله ی چهارم:

برای هر غذا اگر بخش مواد لازم داشت محتوای آن به `corpus` اضافه میشے و همین کار برای طرز تهیه و توضیحات اولیه هم انجام میشے. و یه ایدی هم اضافه میشے و کل بدنه ی دیتاست ساخته میشے.

```

corpus = []
corpus_map = []
for idx, f in enumerate(foods):
    if f["ingredients"]:
        corpus.append(f["ingredients"])
        corpus_map.append((idx, "ingredients"))
    if f["instructions"]:
        corpus.append(f["instructions"])
        corpus_map.append((idx, "instructions"))
    if f["desc"]:
        corpus.append(f["desc"])
        corpus_map.append((idx, "desc"))

```

مرحله ی پنجم:

تو این مرحله با استفاده از یک مدل از قبل اموزش دیده ی `SentenceTransformer` تمام متن های موجود در دیتاستی که ساختیم را به بردار های عددی تبدیل کردیم که میتوانند معنی و محتوای جملات را در فضای برداری نمایش دهند.

بعد بردار هارو استخراج میکنیم و بعد یه faiss index میسازیم برای جست و جو بر اساس inner product معادل هست وقتی بردارها نرمال سازی شده باشن. Similarity

```
model = SentenceTransformer("sentence-transformers/distiluse-base-multilingual-cased-v2")

corpus_emb = model.encode(corpus, convert_to_numpy=True,
normalize_embeddings=True)

dim = corpus_emb.shape[1]
index = faiss.IndexFlatIP(dim)
index.add(corpus_emb)
```

مرحله ی ششم:

در این بخش از پروژه، با هدف ارزیابی کیفیت بازیابی معنایی (semantic retrieval)، تابعی طراحی کردیم که برای هر پرسش از مجموعه‌ی پرسش-پاسخ‌ها، ابتدا embedding پرسش را تولید می‌کند، سپس با استفاده از FAISS، شبیه‌ترین بخش‌های متنی (مانند مواد لازم، طرز تهیه یا توضیحات غذاها) را از میان کل متن‌ها بازیابی می‌کند. با مقایسه‌ی بخش بازیابی شده با پاسخ صحیح، معیارهایی مانند MRR (میانگین رتبه معکوس)، Hit@k (درصد موفقیت در k پاسخ اول) و میانگین شباهت کسینوسی بین پاسخ مدل و پاسخ واقعی محاسبه می‌شود. این ارزیابی به ما نشان می‌دهد که سیستم تا چه حد توانایی دارد تا پاسخ درست را از بین متون مختلف غذاها پیدا کند.

```
def evaluate(model, pairs, index, corpus_emb, corpus_map,
k_list=(1,3,5,10)):
    rr, sims = [], []
    hits = {k: 0 for k in k_list}

    for ex in tqdm(pairs, desc="Evaluating"):
        q_emb = model.encode([ex["query"]], convert_to_numpy=True,
normalize_embeddings=True)
        D, I = index.search(q_emb, max(k_list)) #most similiar indexes to
query --> I

        rank = None #rank with D
        for r, doc_id in enumerate(I[0], start=1):
            if corpus_map[doc_id][0] == ex["doc_id"] and
corpus_map[doc_id][1] == ex["section"]:
                rank = r
                break

    rr.append(1/rank if rank else 0.0)
```

```

        for k in k_list:
            if rank and rank <= k:
                hits[k] += 1

        top1_id = I[0][0]
        ans_emb = model.encode([ex["answer"]], convert_to_numpy=True,
normalize_embeddings=True)
        cos = float((ans_emb @ corpus_emb[top1_id])[0])
        sims.append(cos)

    results = {
        "MRR": float(np.mean(rr)),
        "Hit@k": {k: hits[k] / len(pairs) for k in k_list},
        "CosineSimilarity(avg)": float(np.mean(sims)) #check cosine sim
between real ans and model ans
    }

```

مرحله ی هفتم:

در این بخش، مدل اولیه‌ی چندزبانه‌ی SentenceTransformer را که به صورت عمومی آموزش دیده بود، با استفاده از داده‌های پرسش-پاسخ مرتبط با غذاهای گیلانی، روی دامنه خاص پروژه فاین‌تیون کردیم. ابتدا از پرسش‌ها و پاسخ‌های استخراج شده از فایل متنی غذاها، مجموعه‌ای از نمونه‌های تمرینی ساختیم و آن‌ها را به فرمت مناسب برای آموزش مدل embedding تبدیل کردیم. سپس با استفاده از تابع MultipleNegativesRankingLoss و در طی چند epoch، مدل را آموزش دادیم تا یاد بگیرد که پاسخ درست هر سؤال باید از سایر پاسخ‌ها متمایز باشد. در نتیجه، حالا مدلی در اختیار داریم که نه فقط به زبان فارسی مسلط است، بلکه دقیقاً برای درک سوالات مرتبط با مواد اولیه، طرز تهیه و معرفی غذاهای گیلانی تنظیم شده و انتظار می‌رود در بازیابی معنایی، دقیقاً بالاتری داشته باشد.

```

from sentence_transformers import SentenceTransformer, InputExample,
losses
from torch.utils.data import DataLoader
#dataset
train_examples = []
for ex in pairs:
    train_examples.append(InputExample(texts=[ex["query"], ex["answer"]]))

import random
random.shuffle(train_examples)
val_size = int(0.15 * len(train_examples))
val_data = train_examples[:val_size]
train_data = train_examples[val_size:]

train_loader = DataLoader(train_data, batch_size=16, shuffle=True)

```

```

ft_model = SentenceTransformer("sentence-transformers/distiluse-base-
multilingual-cased-v2")

loss = losses.MultipleNegativesRankingLoss(ft_model)

num_epochs = 3
warmup_steps = int(len(train_loader) * num_epochs * 0.1)

ft_model.fit(
    train_objectives=[(train_loader, loss)],
    epochs=num_epochs,
    warmup_steps=warmup_steps,
    show_progress_bar=True,
    use_amp=True
)

```

مرحله ی هشتم:

در این مرحله، عملکرد مدل فاین‌تیون شده را با مدل پایه (قبل از فاین‌تیون) مقایسه کردیم تا مشخص شود آموزش روی داده‌های اختصاصی غذایی گیلانی چه تأثیری در دقت بازیابی معنایی داشته است. ابتدا تمام متن‌های مرجع (**corpus**) را مدل جدید به تبدیل کردیم، سپس یک ایندکس جدید FAISS ساختیم و از طریق همانتابع ارزیابی قبلی، متريک‌هایی مانند **MRR** و **Hit@k** و ميانگين شباهت كسينوسي را برای مدل جدید محاسبه کردیم. با مقایسه‌ی اين نتایج با نسخه اولیه‌ی مدل، می‌توان به صورت دقیق بررسی کرد که آیا مدل یاد گرفته ارتباط معنایی دقیق‌تری بین پرسش‌ها و پاسخ‌ها برقرار کند یا نه. این ارزیابی نهایی، نشان‌دهنده‌ی ارزش واقعی فرآيند فاین‌تیون در پروژه است.

```

ft_corpus_emb = ft_model.encode(corpus, convert_to_numpy=True,
normalize_embeddings=True)

index_ft = faiss.IndexFlatIP(ft_corpus_emb.shape[1])
index_ft.add(ft_corpus_emb)

ft_metrics = evaluate(ft_model, pairs, index_ft, ft_corpus_emb,
corpus_map)

print("Baseline:", metrics)
print("Fine-Tuned:", ft_metrics)

```

خروجی حالت اول:

```
Evaluating: 100% [██████████] | 81/81 [00:01<00:00, 80.65it/s] Baseline metrics:  
{'MRR': 0.2955173427395649, 'Hit@k': {1: 0.1728395061728395, 3: 0.32098765432098764, 5: 0.4691358024691358, 10: 0.6790123456790124}, 'CosineSimilarity(avg)': 0.54508773724973}
```

خروجی فاین تیون شده:

Fine-Tuned: {'MRR': 0.9094650205761315, 'Hit@k': {1: 0.8271604938271605, 3: 1.0, 5: 1.0, 10: 1.0}, 'CosineSimilarity(avg)': 0.9000939399371912}

مقاسه:

"فاین تیون باعث شده که مدل از حالت یک "مدل عمومی" به یک "مدل متخصص در پاسخ دهی به سوالات مربوط به غذاهای گیلانی" تبدیل شده."

هم از نظر رتبه‌بندی، هم از نظر دقت معنایی، مدل پیشرفت بسیار چشم‌گیری داشته و این نشون می‌ده که فرآیند آموزش شخصی‌سازی شده موفق بوده.

ب) چند مدل امبدینگ مثل multilingual-e5-base و هر مدل مناسب دیگری ، روی داده های پروژه Fine-Tune کرده و عملکرد فرآیند بازیابی را بررسی و مقایسه ای از مدل ها ارائه کنید. برای بهبود کیفیت فضای امبدینگ روش های مختلف-Processing را پیاده سازی نمایید. جهت ارتقای سرعت و کارایی فرآیند بازیابی بردارها، استفاده از پایگاه های داده برداری مانند LanceDB، Chroma .FAISS و سایر پایگاه داده های مناسب نیز توصیه می شود. در انتها تحلیل نتایج بر اساس معیارهای Hit@k، Recall ،Precision ،MRR ،Cosine Similarity و ملاک ارزیابی شما در این بخش خواهد بود (نمره ۷۵).

کد بخش ب و ج

برای انجام این بخش از پروژه از فایل **Guilan-Food.docx** که شامل دستور پخت غذاهاست، متون رو استخراج کردیم (لیست غذا + مواد لازم + طرز تهیه) و دادهها رو به شکل سؤال ← پاسخ یا عنوان غذا ← دستور کامل تبدیل کردیم. و بعد مرحله پیش پردازش رو روش انجام دادیم.

سپس باید امبدینگ و فاین تیون کردن رو انجام بدیم. اول با مدل های مختلف که یکی از اونها خواسته مسئله multilingual e5-base است امبدینگ رو محاسبه کردیم. سپس روی داده ها Fine-Tune انجام دادیم تا امبدینگ ها تخصصی برای غذاهای گیلان بشن.

مرحله بعدی استفاده از کتابخونه FAISS برای ذخیره‌سازی و بازیابی سریع تره که بدین منظور داده‌های برداری رو داخل پایگاه ذخیره کردیم و جستجو (Similarity Search) را با Cosine Similarity و معیاری گفته شده انجام دادیم. معیارهای ارزیابی به صورت زیر هستند:

- Cosine Similarity
- MRR (Mean Reciprocal Rank)
- Precision
- Recall
- Hit@k

سپس عملکرد مدل‌ها رو با هم مقایسه کردیم و اعلام کردیم کدام مدل امبدینگ برای داده‌های گیلان بهتر جواب داده.

سپس RAG رو روی بهترین مدل پیاده سازی کردیم.

هر کدام از این مراحلو جدا جدا و توی کد در ادامه توضیح دادیم:

نصب کتابخونه‌ها:

```
!pip install faiss-cpu sentence-transformers transformers
```

آپلود فایل داده شده و استخراج متن:

```
!pip install python-docx pandas

from google.colab import files
import docx

uploaded = files.upload()
doc = docx.Document("Guilan-Food.docx")
#extracting text from file
text = "\n".join([para.text for para in doc.paragraphs if
para.text.strip() != ""])
```

استخراج سوال جواب از متن:

```
import pandas as pd

foods = []
current_title = None
current_content = []

for line in text.split("\n"):
```

```

if len(line.strip()) == 0:
    continue
#if the line is short it is a food's name
if len(line.split()) < 4 and not line.strip().isdigit():
    if current_title and current_content:
        foods.append({
            "title": current_title.strip(),
            "content": "\n".join(current_content).strip()
        })
    current_title = line
    current_content = []
else:
    current_content.append(line)

#last food is added
if current_title and current_content:
    foods.append({
        "title": current_title.strip(),
        "content": "\n".join(current_content).strip()
    })

qa_pairs = []
for f in foods:
    content = f["content"]

    #extracting ingredients
    ingredients = ""
    steps = content
    if "مواد لازم" in content:
        parts = content.split("مواد لازم")
        if len(parts) == 2:
            ingredients = parts[0].replace("مواد لازم", "").strip()
            steps = "طرز تهیه " + parts[1].strip()

    qa_pairs.append({
        "title": f["title"],
        "question": f"طرز تهیه {f['title']} چیست؟",
        "answer": steps,           #how to cook
        "ingredients": ingredients #ingredients
    })

df = pd.DataFrame(qa_pairs)
print("داده ها استخراج شد")
print(df.head(3))

```

این بخش از کد متن خامی که از فایل DOCX گرفتیم رو تبدیل میکنه به ساختاری که برای retrieval مناسب باشه:

- title: اسم غذا
- question: یک پرسش استاندارد (مثلًا "طرز تهیه میرزا قاسمی چیست؟")
- answer: متن کامل دستور پخت همون غذا
- ingrediants: مواد لازم برای تهیه اون غذا
- و همه اینا رو داخل یک DataFrame (df) میذاریم.

برای انجام این کار متن به بخش‌هایی تقسیم میشے و متن فایل خط به خط بررسی میشے. اگر خط کوتاه باشه (مثلًا فقط اسم غذا) اون رو به عنوان title ذخیره میکنه. اگر خط طولانی باشه (مثلًا مواد لازم و دستور پخت) به عنوان content ذخیره میشے. همینطور مواد لازم رو هم ذخیره میکنه. هر غذا به شکل یک دیکشنری ذخیره میشے و در نهایت همه اینا میرن داخل یک جدول پانداس:

title	question	ingredients	answer
میرزا قاسمی	طرز تهیه میرزا قاسمی چیست؟	بادمجان، گوجه، سیر، تخم مرغ...	بادمجان‌ها را کباب کنید...
باقلاء قاتوق	طرز تهیه باقلاء قاتوق چیست؟	باقلاء، شوید، سیر، تخم مرغ...	ابتدا باقلاء را بپزید...

خروجی این سلول همون دیتابست Q&A هست که بعداً می‌دیم به preprocessing

:preprocessing مرحله

```
import re

#defining stopwords
persian_stopwords = set([
    "اگر", "اما", "تا", "یا", "و", "آن", "این", "برای", "با", "را", "که", "به", "از", "شود", "شد", "یک"
])

def normalize_persian(text):
    #converting arabic words to persian
    text = re.sub("ي", "ی", text)
    text = re.sub("ك", "ک", text)
    return text
```

```

def normalize_numbers(text):
    #converting numbers
    persian_digits = "۰۱۲۳۴۵۶۷۸۹"
    english_digits = "0123456789"
    trans = str.maketrans("".join(persian_digits),
"").join(english_digits))
    return text.translate(trans)

def clean_text(text):
    if not isinstance(text, str):
        return ""

    #normalization
    text = normalize_persian(text)
    text = normalize_numbers(text)

    #removing spaces
    text = re.sub(r"\s+", " ", text).strip()

    #removing signs
    text = re.sub(r"[^\w\s\ء-\ى]", " ", text)

    #tokenizing
    tokens = text.split()

    #removing stopwords
    tokens = [t for t in tokens if t not in persian_stopwords]

    return " ".join(tokens)

#applying on the data
df["clean_answer"] = df["answer"].apply(clean_text)
df["clean_question"] = df["question"].apply(clean_text)

print("داده ها آماده شد (نسخه پیشرفته)")
print(df[["question","clean_question","answer","clean_answer"]].head())

```

در پری پراسینگ این تغییرات رو دادیم:

- نرمالسازی حروف
- نرمالسازی اعداد
- حذف علائم اضافی

Stopwords •

مدل ها و فاین تیون کردنشون:

آماده سازی داده برای Fine-Tuning

```
from sentence_transformers import InputExample
from torch.utils.data import DataLoader

#preparing data for fine tuning
train_examples = []
for i, row in df.iterrows():
    train_examples.append(
        InputExample(texts=[row["clean_question"], row["clean_answer"]]))
print("training data:", len(train_examples))
```

اینجا داده های clean_answer و clean_question رو به فرمت مناسب sentence-transformers در میاریم. کتابخونه sentence-transformers برای آموزش (Fine-Tune) نیاز داره بدونه کدوم متن ها به هم مرتبط هستن. این ارتباط معمولاً به شکل جفت متن (text pairs) داده می شه.

- یک متن ورودی (مثالاً سؤال)
- یک متن مرتبط (مثالاً جواب)

برای همین داخل این کتابخونه چیزی داریم به اسم InputExample. هر نمونه‌ی آموزشی باید به شکل زیر باشه:

InputExample(texts = [امتن ۱، متن ۲])

اینطوری مدل یاد می گیره که بردار سؤال و جواب هم دیگه نزدیک باشن و وقتی بعداً سؤال جدیدی بدیم، بردارش می ره نزدیک به جواب درست داخل فضای امبدینگ. پس از ستون های clean_answer و clean_question دیتا فریم استفاده می کنیم. برای هر ردیف یک InputExample می سازیم و این لیست از InputExample ها می شه ورودی آموزش مدل.

تابع :Fine-Tune

```
from sentence_transformers import SentenceTransformer, losses

def fine_tune_model(model_name, train_examples, output_path, epochs=3,
batch_size=8):
    print(f"finetune model: {model_name}")

    #base model
    model = SentenceTransformer(model_name)

    #DataLoader
    train_dataloader = DataLoader(train_examples, shuffle=True,
batch_size=batch_size)

    #loss: MultipleNegativesRankingLoss
    train_loss = losses.MultipleNegativesRankingLoss(model)

    #train
    model.fit(
        train_objectives=[(train_dataloader, train_loss)],
        epochs=epochs,
        warmup_steps=10
    )

    #saving the model
    model.save(output_path)
    print(f"model is saved at: {output_path}")

    return model
```

یک تابع می‌نویسیم که هر مدلی رو بگیره، روی داده‌ها فاین‌تیون کنه، و خروجی مدل فاین‌تیون شده رو ذخیره کنه. مدل‌های آماده بارگذاری می‌شن. این مدل‌ها قبلاً روی دیتاست‌های بزرگ آموزش دیدن، ولی هنوز تخصصی برای سؤال–جواب غذاهای گیلان نیستن.

داده‌هایی که در قالب `InputExample` ساختیم ریخته می‌شن داخل `DataLoader`. ترتیب داده‌ها با استفاده از `shuffle=True` تصادفی می‌شه برای جلوگیری از `overfitting` و در هر بار آموزش، ۸ جفت سؤال–جواب به مدل داده می‌شه. با استفاده از تابع `خطا` سؤال باید به جواب درستش نزدیک بشه و از جواب‌های اشتباه دور بشه. کل داده‌ها ۳ بار به مدل نشون داده می‌شن (`epoch=3`). با تنظیم `warmup_steps=10` به مدل میگیم چند گام اول آرام‌تر یادگیری رو انجام بده تا پایدارتر باشه.

Fine-Tune همه مدل‌ها:

```
models_to_test = [
    "intfloat/multilingual-e5-base",
    "sentence-transformers/distiluse-base-multilingual-cased-v2",
    "sentence-transformers paraphrase-multilingual-MiniLM-L12-v2",
    "sentence-transformers/all-mpnet-base-v2",
    "BAAI/bge-base-en-v1.5"
]

fine_tuned_models = {}

for model_name in models_to_test:
    output_path = f"fine_tuned_{model_name.replace('/', '_')}"
    model = fine_tune_model(model_name, train_examples, output_path,
epoch=3)
    fine_tuned_models[model_name] = model
```

استفاده از :FAISS

ما الان یه عالمه متن (جواب غذاها) داریم. هر متن رو با یک مدل امبدینگ به یک بردار با یه تعداد بعد تبدیل کردیم. وقتی کاربر سؤال می‌پرسه سؤال هم به بردار تبدیل میشه. حالا باید بین هزاران بردار جواب‌ها، نزدیکترین رو پیدا کنه. جستجوی عادی توی بردارهای بزرگ با ابعاد زیاد خیلی گرونه و کند میشه. اگه بخوای برای هر سؤال همه‌ی بردارها رو یکی یکی مقایسه کنی، برای دیتاست بزرگ اصلاً عملی نیست. FAISS (Facebook AI Similarity Search) یک کتابخونه از فیسبوکه که برای:

- ذخیره‌سازی بردارها
- جستجوی سریع در بین بردارها
- محاسبه شباهت (Cosine, Inner Product, L2 distance)

ساخته شده.

تابع ساخت ایندکس:

```
import faiss
import numpy as np

def build_faiss_index(embeddings):
    d = embeddings.shape[1]
```

```

index = faiss.IndexFlatIP(d)           #Inner Product
faiss.normalize_L2(embeddings)         #normalization for cosine
index.add(embeddings)                 #adding vectors to index
return index

```

هر امبدینگ یک بردار با یک طولی هست. خط اول تعداد بُعد بردارها رو می‌گیره و خط دوم یک ایندکس برداری ساخته میشه. ۲

برای مقایسه شباهتها از Inner Product استفاده می‌کنیم. وقتی بردارها رو نرمال‌سازی کنیم، میشه همون Inner Product هر بردار به طول ۱ (norm = 1) تبدیل میشه. این باعث میشه Inner Product بین دو بردار دقیقاً برابر بشه. همهی بردارها (امبدینگ‌های جواب‌های غذاها) به ایندکس اضافه میشن. از این به بعد، FAISS می‌تونه Cosine Similarity را برای گردونه FAISS return index یک شیء یاد کنه. برمی‌گردیک‌ترین بردارها رو پیدا کنیم. ذخیره شدن. بعداً وقتی سؤال کاربر رو امبدینگ کنیم، می‌تونیم با کد زیر نزدیک‌ترین بردارها رو پیدا کنیم.

```
D, I = index.search(query_vec, k)
```

در واقع میایم همهی جواب‌های غذاها رو روی یک نقشه‌ی چندبُعدی می‌چینیم بعد می‌تونیم هر سؤال جدید رو هم روی همون نقشه بذاریم و بپرسیم نزدیک‌ترین جواب‌ها کدوما هستن.

تابع ارزیابی با همه معیارها:

```

def evaluate_retrieval(df, model, index, k=3):
    reciprocal_ranks = []
    precision_scores = []
    recall_scores = []
    hit_scores = []
    cosine_scores = []

    for i in range(len(df)):
        query = df.iloc[i]["clean_question"]
        true_answer = df.iloc[i]["clean_answer"]

        #question embedding
        query_vec = model.encode([query], convert_to_numpy=True)
        faiss.normalize_L2(query_vec)

        #search
        D, I = index.search(query_vec, k)
        retrieved_answers = [df.iloc[idx]["clean_answer"] for idx in I[0]]

        #saving highest cosine similarity

```

```

cosine_scores.append(D[0][0])

#MRR
rr = 0
for rank, ans in enumerate(retrieved_answers, start=1):
    if ans == true_answer:
        rr = 1 / rank
        break
reciprocal_ranks.append(rr)

#Precision@k, Recall@k, Hit@k
if true_answer in retrieved_answers:
    precision_scores.append(1.0)
    recall_scores.append(1.0)
    hit_scores.append(1.0)
else:
    precision_scores.append(0.0)
    recall_scores.append(0.0)
    hit_scores.append(0.0)

return {
    "MRR": np.mean(reciprocal_ranks),
    f"Precision@{k)": np.mean(precision_scores),
    f"Recall@{k)": np.mean(recall_scores),
    f"Hit@{k)": np.mean(hit_scores),
    "Avg Cosine Sim": np.mean(cosine_scores)
}

```

این تابع کیفیت سیستم جستجو (Retriever) را با معیارهای زیر حساب می‌کند:

- MRR (Mean Reciprocal Rank)
- Precision@k
- Recall@k
- Hit@k
- Cosine Similarity میانگین

چند لیست خالی ساخته میشے تا برای هر سؤال مقادیر رو ذخیره کنه. تک تک سطرهای دیتا فریم رو برمی‌داره و سؤال به بردار تبدیل میشے (query_vec). با FAISS بین همه جوابها جستجو می‌کنه و نزدیکترین k جواب رو میده. اندیس جواب‌ها و امتیاز شباهت هاست. شباهت نزدیکترین جواب (رتبه ۱) ذخیره میشے.

- اگه جواب درست بین top-k باشه، وگرنه

- اگه جواب درست توی top-k باشه، وگرنه • (چون فقط یک جواب درست داریم).
- اگه جواب درست اومنده باشه، وگرنه •.

برای همه سؤال‌ها مقدایر محاسبه می‌شود و میانگین گرفته می‌شود و یک دیکشنری خروجی میدهد.

اجرای ارزیابی:

```
results = []

for model_name, model in fine_tuned_models.items():
    print(f"evaluating model: {model_name}")

    embeddings = model.encode(df["clean_answer"].tolist(),
convert_to_numpy=True, show_progress_bar=True)
    index = build_faiss_index(embeddings)

    metrics = evaluate_retrieval(df, model, index, k=3)
    metrics["model"] = model_name + " (fine-tuned)"
    results.append(metrics)

results_df = pd.DataFrame(results)
print("results:")
print(results_df)
```

ما مدل‌های فین‌تاون را Fine-Tune کردیم. یه تابع نوشتم برای ساخت ایندکس FAISS. یه تابع هم نوشتم برای محاسبه متريک‌ها. حالا اينجا مياد همه‌ی اينا رو به هم وصل می‌کنه و واقعاً روی مدل‌ها اجرا می‌کنه. تک‌تک مدل‌های فاین‌تاون شده رو می‌گيره. تمام جواب‌های ديتاپریم (مثلاً دستور غذاها) رو به بردار تبدیل می‌کنه. همه‌ی بردارها رو می‌ریزه توی پایگاه FAISS برای جستجوی سریع. با استفاده از همون تابع، معیارهای MRR، Precision@3، Recall@3، Hit@3 و Cosine Similarity رو حساب می‌کنه. نتیجه رو به همراه اسم مدل ذخیره می‌کنه. نتایج همه‌ی مدل‌ها رو توی یک جدول (DataFrame) نشون می‌دهد.

نتیجه:

	MRR	Precision@3	Recall@3	Hit@3	Avg	Cosine Sim	model
0	0.803030	0.909091	0.909091	0.909091		0.855092	
1	0.780303	0.954545	0.954545	0.954545		0.314796	
2	0.454545	0.681818	0.681818	0.681818		0.707634	
3	0.356061	0.545455	0.545455	0.545455		0.803841	
4	0.568182	0.636364	0.636364	0.636364		0.843606	

	model
0	intfloat/multilingual-e5-base (fine-tuned)

```
1 sentence-transformers/distiluse-base-multilingual...
2 sentence-transformers/paraphrase-multilingual-...
3 sentence-transformers/all-mpnet-base-v2 (fine-...
4 BAAI/bge-base-en-v1.5 (fine-tuned)
```

بهترین مدل:

```
#choosing best model according to MRR
best_model_name = results_df.sort_values("MRR",
ascending=False).iloc[0]["model"]
print("best model:", best_model_name)

#best models loading from file
base_name = best_model_name.replace(" (fine-tuned)", "")
best_model_path = f"fine_tuned_{base_name.replace('/', '_')}"

#loading the best model
from sentence_transformers import SentenceTransformer
best_model = SentenceTransformer(best_model_path)
```

خروجی:

```
best model: intfloat/multilingual-e5-base (fine-tuned)
```

ساخت ایندکس faiss برای بهترین مدل:

```
#embedding for all answers
best_embeddings = best_model.encode(df["clean_answer"].tolist(),
convert_to_numpy=True, show_progress_bar=True)

#index
index = build_faiss_index(best_embeddings)
print("produced faiss index:", index.ntotal, "بردار")
```

وقتی بهترین مدل انتخاب شد، باید تمام متن‌های دیتابست (یعنی جواب‌غذاها) را به بردار تبدیل کنیم و بعد آن‌ها را داخل FAISS ذخیره کنیم. ایندکس FAISS همون پایگاه داده‌ایه که بعداً توی RAG و Retrieval استفاده می‌کنیم. همه جواب‌های دیتابریم cosine similarity (clean_answer) را می‌گیریم. با مدل انتخاب‌شده اونا را به بردار تبدیل می‌کنیم. یه ایندکس FAISS از نوع index نشون می‌ده چند تا می‌سازه. اول بردارها را نرم‌السازی می‌کنیم. بعد همه بردارها را توی ایندکس اضافه می‌کنیم. یعنی میاد همه‌ی جواب‌های بردار (یعنی چند تا جواب) داخل ایندکس ذخیره شدن که باید برابر با تعداد ردیف‌های دیتابریم باشد. یعنی میاد همه‌ی جواب‌های دیتابست را به بردار تبدیل می‌کنیم و داخل یه موتور جستجو (FAISS) می‌ذاره تا بعداً بشه روی اون‌ها سریع و دقیق سرج کرد.

بارگذاری مدل مولد:

```
from transformers import pipeline

#generator model
generator = pipeline("text2text-generation",
model="csebuetnlp/mT5_multilingual_XLSum")

print("generator done")
```

از این مدل استفاده کردیم چون چند زبانه است.

دو بخش دارد:

- اسناد مرتبط رو میاره. Retriever
- همون اسناد + سؤال رو میگیره و یه پاسخ جدید میسازه. Generator

اینجا داریم generator رو آماده میکنیم. pipeline یکی از قابلیت‌های راحت transformers هست. به جای اینکه خودمون مدل رو بارگذاری و توکنایز کنیم، همه‌چیز رو آماده میکنه. اینجا یک مدل زبان بزرگ بارگذاری میکنیم. google/flan-t5-base مدلیه که برای پرسش و پاسخ و تولید متن کوتاه خوب عمل میکنه. حالا میتوانیم هر prompt رو به generator بدهیم و متن جواب تولید کنیم.

rag تابع تعريف:

```
def retrieve_only(query, retriever_model, index, df, k=3):
    #question to embedding
    query_vec = retriever_model.encode([query], convert_to_numpy=True)
    faiss.normalize_L2(query_vec)

    #search in faiss
    D, I = index.search(query_vec, k)

    retrieved_contexts = [df.iloc[idx]["answer"] for idx in I[0]]

    return retrieved_contexts[0], retrieved_contexts
```

این سلول تابعی می‌سازه که کل فرآیند RAG (Retrieval-Augmented Generation) رو توی یه قدم انجام بدنه:

- جستجو توی FAISS با کمک بهترین مدل امبدینگ.
- اضافه کردن متن های بازیابی شده به سؤال.
- دادن سؤال + context به مدل مولد برای تولید پاسخ نهایی.

توی بازیابی سؤال به بردار تبدیل میشه. FAISS جستجو می کنه و k تا جواب نزدیک رو پیدا می کنه. متن واقعی جواب های نزدیک (Top-k) از دیتا فریم استخراج میشه. این متن ها میشن اطلاعات زمینه (Context). یه رشته ساخته میشه شامل خود سؤال، متنون بازیابی شده، دستور به مدل که پاسخ کوتاه بده. پرامپت به مدل مولد داده میشه. خروجی مدل یک متن جواب طبیعیه. خروجی تابع دو چیز برمی گردونه. پاسخ نهایی تولید شده توسط Generator و متن های بازیابی شده.

حالا باید ذکر کنیم که علیرغم اینکه روش رگ اینه ولی ما از روش retriver only استفاده میکنیم که جواب درست تولید کنه. تو حالت generator جواب های پرت میداد.

ج) برای بهترین مدل QA، یک واسط کاربری طراحی نمایید که در آن کاربر بتواند با وارد کردن متن سوال، پاسخ تولید شده توسط را مشاهده نماید. استفاده از Fast-API، Streamlit، Gradio و سایر ابزار های مناسب، مجاز است. (۰.۲۵ نمره)

رابط کاربری برای بهترین مدل با استفاده از gradio ساخته میشه.

```
!pip install gradio
import gradio as gr

def qa_interface(query, k):
    answer, contexts = retrieve_only(query, best_model, index, df,
k=int(k))

    retrieved_texts = "\n\n".join(
        [f" {c[:400]} ..." for i, c in enumerate(contexts)])
)

    return str(answer), retrieved_texts

with gr.Blocks(theme=gr.themes.Soft()) as demo:
    gr.Markdown("## 📄 پرسش و پاسخ دستور غذاهای گیلانی (Retriever Only)")

    query = gr.Textbox(label="سؤال خود را وارد کنید")
    query.append(": میزرا قاسمی چیست؟")
    k = gr.Slider(minimum=1, maximum=5, value=3, step=1, label="Top-k")
```

```

run_btn = gr.Button("جستجو")
answer_box = gr.Textbox(label="پاسخ Retrieve-only", lines=5)
contexts_box = gr.Textbox(label="متن بازیابی شده", lines=10)

run_btn.click(fn=qa_interface, inputs=[query, k], outputs=[answer_box, contexts_box])

demo.launch(share=True)

```

خروجی به صورت زیر است:

پرسش و پاسخ دستور غذاهای گیلانی (Retriever Only)

سوال خود را وارد کنید

ظرف تهیه میرزا قاسمی چیست؟

Top-k

1 5

جستجو

پاسخ Retrieve-only

برای تهیه میرزا قاسمی، ایندا با ماده‌مان را بشویید و روی شعله گاز منقل با داخل فر کنایی کنید تا پوست آنها کاملاً سوید و داخلشان نرم شود. در حین کباب کردن، با ماده‌مان را بچراحته کنید. سپس نکارید با ماده‌مان ها کمی خنک شوند و پوست سوچته آنها را جدا کنید. گوشت داخل با ماده‌مان را با چاقو ساطوری کنید، نه خلی ریز و نه خلی داشته. در مرحله بعد، گوجه فرنگی‌ها را بشویید و پوست آنها را نکرد. برای راحتتر پوست گوجه فرنگی‌ها را زنده شده را اضافه کنید. گوچه فرنگی‌ها را جدا نمایند و سیر له شده را زنده شده را اضافه کنید. گمی تفت دهد. هر ده دقیقه یک مقدار سیر مخلوط کرده و مراوه باشید که شود. سیر مخلوط کرده خود را 10-5 دقیقه را در آن اضافه کنید و لف دهید تا آب از ماده‌مان بگذرد. با این مرحله، گمی نمک، ماهی سیرمه و زرگونه شده را به نایه اضافه کنید و نمک اضافه کنید. با این مرحله، گمی نمک، ماهی سیرمه و زرگونه شده را به نایه اضافه کنید و نمک اضافه کنید. گوچه فرنگی‌ها را جدا کنید و خوش‌شدن را انجام دهید. سپس اینها را مواد دیگر مخلوط کرده و هم نزدیک کنید. مراوه باشید تا آن شستید. اجازه دهد تا طعم‌ها به جوین ترکب شوند. سپس در وسط تابه یک گوچه ایجاد کرده و اندیمه‌ها را در آن شستید. اجازه دهد تا مقدار مخصوص خود را از چاقو ساطوری کنید. میرزا قاسمی معمولاً با نایه اضافه کرده و مراوه باشید و آنرا استفاده کنید تا اذای خوشمزه‌تر شود. بربری) و همراه با سیر ترش، زیتون پزوره ده و سدری خوردن سرو معده‌دار. اجازه دهد تا طعم نهاد. برای طعم نهاد، حنتماً با ماده‌مان را به حاوی کنایی کنید تا طعم دودی از آنها در غذا نمایان شود. همچنان از گوجه فرنگی‌ها ریسیده و آنرا استفاده کنید تا اذای خوشمزه‌تر شود.

متن بازیابی شده

متن 1:
برای تهیه میرزا قاسمی، ایندا با ماده‌مان را بشویید و روی شعله گاز منقل با داخل فر کنایی کنید تا پوست آنها کاملاً سوید و داخلشان نرم شود. در حین کباب کردن، با ماده‌مان را بچراحته کنید. سپس نکارید با ماده‌مان ها کمی خنک شوند و پوست سوچته آنها را جدا کنید. گوشت داخل با ماده‌مان را با چاقو ساطوری کنید، نه خلی ریز و نه خلی داشته. در مرحله بعد، گوجه فرنگی‌ها را بشویید و پوست آنها را نکرد. برای راحتتر پوست گوجه فرنگی‌ها را زنده شده را اضافه کنید. گوچه فرنگی‌ها را جدا نمایند و سیر له شده را زنده شده را اضافه کنید. گمی تفت دهد. هر ده دقیقه یک مقدار سیر مخلوط کرده و مراوه باشید که شود. سیر مخلوط کرده خود را 10-5 دقیقه را در آن اضافه کنید و لف دهید تا آب از ماده‌مان بگذرد. با این مرحله، گمی نمک، ماهی سیرمه و زرگونه شده را به نایه اضافه کنید و نمک اضافه کنید. با این مرحله، گمی نمک، ماهی سیرمه و زرگونه شده را به نایه اضافه کنید و نمک اضافه کنید. گوچه فرنگی‌ها را جدا کنید و خوش‌شدن را انجام دهید. سپس اینها را مواد دیگر مخلوط کرده و هم نزدیک کنید. مراوه باشید تا آن شستید. اجازه دهد تا طعم‌ها به جوین ترکب شوند. سپس در وسط تابه یک گوچه ایجاد کرده و اندیمه‌ها را در آن شستید. اجازه دهد تا مقدار مخصوص خود را از چاقو ساطوری کنید. میرزا قاسمی معمولاً با نایه اضافه کرده و مراوه باشید و آنرا استفاده کنید تا اذای خوشمزه‌تر شود.

متن 2:
ایندا گوشت چرخ‌کرده را با کمی نمک، فلفل و زرچوبه ورز دهد و به صورت کوتنه‌بزدگاهی کوچک، درآورید. در یک قایمه، گردیوی آسیاب شده را با حرارت ملایم کمی تفت دهد. سپس رب، آنرا اضافه کرده و کمی تفت دهد. حال آنرا به تزیین اضافه کنید و اجازه دهد تا مخلوط کمی غلظت شود. کوتنه‌بزدگاهی را به آرامی به حوصله اضافه کنید و حدو ۳۰ دقیقه پزد. سپس سری‌های خوش‌شده را اضافه کرده و نگارید تا خوب باشد.

متن 3:
کلاب: ۱. فاشق علاجی‌روی
پودر: ۲/۲. فاشق علاجی‌روی
پودر نارگیل: به مقدار لازم
کره با روغن: ۱-۲ چوچ علاجی‌روی
پسته با پادام خرد شده (اختصاری): به مقدار لازم

Use via API · Built with Gradio · Settings