

بسمه تعالی



گزارش تمرین دوم

استاد

دکتر احسان‌الدین عسگری

نویسندگان

محمد حسین اسلامی (۴۰۱۱۰۵۵۳۷)

ارشیا ایزدیاری (۴۰۱۱۰۵۶۵۶)

صادق محمدیان (۴۰۱۱۰۹۴۷۷)

دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

بهار - تابستان ۱۴۰۴

مقدمه

در این تمرین که دنباله‌ای از تمرین اول بود، می‌خواهیم تا با داده‌های بدست آمده یک مدل retrieval طراحی داده و آن را آموزش دهیم. در این تمرین که بیشتر به دنبال اهداف آموزشی هستیم، قصد داریم تا با استفاده از داده‌های بدست آمده و آموزش مدل زبانی، روش‌های مختلف برای طراحی یک مدل retrieval در نظر بگیریم. در این تمرین، در ابتدا ما داده‌های مدنظر را به صورتی که می‌خواهیم آماده کرده و سپس مدل‌های مختلفی برای انجام کار retrieve در نظر می‌گیریم. سپس با ۵۰ سوال، مدل‌های خود را ارزیابی کرده و در نهایت نتایج را با هم مقایسه می‌کنیم.

بخش اول: آماده‌سازی داده‌های آموزش

در بخش می‌خواهیم تا داده‌های خامی که از تمرین قبل به ما رسیده است را همان‌طور که صورت تمرین خواسته آماده کنیم. داده‌های اولیه، تعدادی فایل json بوده که توسط گروه‌های مختلف در تمرین یک بدست آمده‌اند. در ابتدا با استفاده از کد داده‌ها را به قالب استاندارد گفته شده در تمرین یک تبدیل کردیم و با merge کردن تمام فایل‌ها، یک فایل کلی از داده‌ها بدست می‌آوریم که تمام اطلاعات غذا در آن قرار دارد.

حال که تمام داده‌ها در یک جا جمع شده‌اند، ما با استفاده از یک کد ساده، مدل Gemma و یک پرامپت اولیه، هر entry در داده‌ها را به مدل داده و یک پاراگراف روان و مرتبط با آن داده از مدل خروجی می‌گیریم. این پاراگراف قرار است ادامه در فرایند آموزش استفاده شود. سپس بر اساس آن پاراگراف، سؤالاتی مربوط به همان متن تولید می‌شوند. با استفاده از مدل زبانی، این سؤالات به شکلی طرح می‌شوند که جواب در خود متن باشد. حال که هم پاراگراف‌ها و هم سؤالات آموزشی برای هر پاراگراف را آماده داریم، 50 سوال برای بخش ارزیابی هر مدل طراحی می‌کنیم تا در ادامه از آن‌ها استفاده کنیم.

وقتی که داده‌های تولیدی آماده شدند، ما آن‌ها را به فرمتی که برای مدلی که می‌خواهیم آن را آموزش دهیم آماده می‌کنیم. ساختار داده‌ها به این شکل است که در یک فایل، یک لیست کلی از تمام پاراگراف‌ها داریم. از این لیست در ادامه در مدل‌هایی که آموزش نیاز ندارند نیز استفاده می‌کنیم. در فایلی دیگر، یک لیست بزرگ از دیکشنری‌هایی آماده کردیم که در هر دیکشنری دو کلید قرار دارد. کلید اول، یک سوال آموزشی از سری سؤالاتی بود که طراحی کردیم. کلید دوم نیز پاراگراف مربوط به آن سوال است. برای آموزش، که توضیحات بیشتری در مورد آن در ادامه خواهیم داشت، هر کدام از اعضای لیست ذکر شده در فایل بالا به عنوان یک داده label دار استفاده می‌شود که می‌توان آن را یک جفت متن و سوال مربوط به آن در نظر گرفت.

در ادامه، به توضیح کدهای پیاده‌سازی شده برای تولید متن از داده‌های خام می‌پردازیم. همان‌طور که قبلاً نیز به آن اشاره شد، برای تولید متون و سؤالات از Gemma استفاده کردیم. در ابتدا ما یک پیش prompt برای مدل زبانی آماده کردیم. این prompt، با هر درخواست به سمت مدل زبانی می‌رود تا مدل هر دفعه بداند سوال پرسیده شده چیست و چه کار باید بکند. در این prompt برای مدل توضیح داده شده که در ادامه قرار است تا داده ساختاری را دریافت کند و از فیلدهای درون آن، متنی روان بدست بیاورد.

در ابتدا مدل را تعریف کرده و آن را با پارامترهای مناسب initialize می‌کنیم:

```
API_KEY = "Hello There!"
if not API_KEY:
    raise RuntimeError("Set the GEMINI_API_KEY environment variable.")

genai.configure(api_key=API_KEY)
model = genai.GenerativeModel("gemma-3-27b-it")
GEN_CONFIG = {"max_output_tokens": 500, "temperature": 0.7}

# --- Load data ---
with INPUT_JSON.open("r", encoding="utf-8") as f:
    foods = json.load(f)

if not isinstance(foods, list):
    raise ValueError("foods.json must contain a list of food entries.")
```

در ادامه، هر کدام از داده‌ها به همراه prompt پایین به مدل زبانی داده می‌شوند متون به صورت دسته‌های ۵۰ تایی ساخته می‌شوند.

```
PROMPT_PREFIX = (
    """take a look at this data structure.
    I want you to write a paragraph in persian using the information given below.
    Please make sure the paragraph is smooth and like a normal paragraph.
    Use the information in each field.
    Make sure that you include all the ingredients and instructions.
    Do not miss them out. You can ingnore the image links as they are not important.
    There should not be any english in the response."""
)
```

حال به سراغ پردازش تمام داده‌ها می‌رویم و تک‌تک آن‌ها را به مدل می‌دهیم. در کنار داده‌های خام، prompt اولیه نیز به مدل داده می‌شود.

```
# --- Process entries in batches ---
results = []
total = len(foods)

batch = foods[1000:]
for idx, food in enumerate(batch):
    title = food.get("title", f"Untitled {idx}")
    entry_text = json.dumps(food, ensure_ascii=False)
    full_prompt = PROMPT_PREFIX + entry_text

    print(f" - [{idx}] {title}")
    try:
        response = model.generate_content(full_prompt, generation_config=GEN_CONFIG)
        text = response.text
    except GoogleAPIError as e:
        print(f" [ERROR] Failed for {title}: {e.message}")
        text = None

    results.append([
        "title": title,
        "response": text,
    ])
```

سوالات نیز به شکلی دقیقاً مشابه با روش بالا تولید می‌شوند و فقط قسمت **prompt** آن متفاوت می‌شود.

حال که داده‌های آموزش و ارزیابی آماده هستند، وقت آن است که به معرفی مدل‌ها بپردازیم.

بخش دوم: معرفی مدل‌ها و ارزیابی آن‌ها

در این بخش به معرفی سه مدل خواهیم پرداخت که در ادامه می‌خواهیم آن‌ها را با یک دیگر مقایسه بکنیم. سه مدل مدنظر موارد زیر می‌باشند:

- مدل آماری **tf-idf** برای **retrieval**
- مدل **GLOT500** بدون آموزش قبلی (**zero-shot**)
- مدل **GLOT500** با آموزش (**fine-tune** شده)

مدل آماری **tf-idf**:

در این بخش، به پیاده‌سازی مدل آماری **tf-idf** می‌پردازیم. این روش، راهی برای پیدا کردن یک **representation** برای متون می‌باشد. برای پیاده‌سازی این مدل، ما از ماژول آماده‌ای که کتابخانه **sklearn** در اختیار ما قرار می‌دهد استفاده می‌کنیم. در ابتدا ما یک **instance** از کلاس **TfidfVectorizer** می‌گیریم و حداکثر بعد را برای بردارهای تولیدی مشخص می‌کنیم.

در ادامه، تمام پاراگراف‌های تولیدی را به **instance** ساخته شده می‌دهیم. هدف ما این است تا تعدادی بردار تولید کنیم که بتوانند متون را به صورت برداری نشان دهند. در ادامه، تمام سوالات داده شده را نیز با همان **instance** به بردار تبدیل می‌کنیم. همان‌طور که می‌توان دید، ما در تلاش هستیم تا متون و سوالات خود را به یک فضای برداری برده تا بتوانیم در آینده و با استفاده روشی شباهت بین دو بردار را پیدا کنیم. در این مدل، به صورت خاص از **cosine similarity** برای محاسبه شباهت دو بردار استفاده کردیم. در واقع ما در تلاش هستیم تا متون و سوالاتی که به هم مرتبط هستند و ممکن است جواب سوال را در متن مشخص شده مشاهده کرد به وسیله **project** کردن در یک فضای برداری پیدا کنیم.

در ادامه به ازای هر سوال، سه متنی که بیشترین شباهت را با سوال ما داشته گزارش می‌کنیم. همان‌طور که قابل مشاهده است، این روش، بسیار روش ساده و پرسرعتی می‌باشد اما به دلیل آن که از قابلیت‌های مختلف زبانی و ارتباطی که اجزای یک جمله و یا حتی متن با یکدیگر دارند استفاده نمی‌کند روش ضعیفی محسوب می‌شود. در واقع، در این روش، خروجی‌های بدست آمده بسیار وابسته به ظاهر متون و سوالات است و اطلاعات کمی را از لحاظ معنایی در متون مشخص شده خواهیم دید. به همین دلیل، هر چه قدر سوالات سخت‌تر و پیچیده‌تر شوند، عملکرد این مدل ضعیف‌تر خواهد شد.

در نهایت می‌توانید پیاده‌سازی ساده این مدل را عکس زیر مشاهده کنید. البته، کد مدل در کنار این گزارش ضمیمه شده است.

```

import json
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

with open("../data_extractor/corpus.json", "r", encoding="utf-8") as f:
    corpus = json.load(f)

with open("../data_extractor/questions.json", "r", encoding="utf-8") as f:
    eval_questions = json.load(f)

print(f"Loaded {len(eval_questions)} evaluation questions.")

vectorizer = TfidfVectorizer(max_features=50000)
tfidf_corpus = vectorizer.fit_transform(corpus)

tfidf_q = vectorizer.transform(eval_questions)
sims = cosine_similarity(tfidf_q, tfidf_corpus)
top3_tfidf = sims.argsort(axis=1)[:, -3:][:, :-1]

output_file = "tfidf_results.txt"

with open(output_file, "w", encoding="utf-8") as f:
    for i, question in enumerate(eval_questions):
        f.write(f"سوال {i + 1}: {question}\n")
        f.write(f"پاسخهای پیشنهادی:\n")
        for rank, para_idx in enumerate(top3_tfidf[i]):
            para_text = corpus[para_idx]
            similarity = sims[i][para_idx]
            f.write(f"    {rank + 1}. (Similarity: {similarity:.4f})\n")
            f.write(f"        {para_text}\n")
        f.write("\n" + "-" * 60 + "\n\n")

```

مدل زبانی GLOT500:

مدل GLOT500 یک مدل زبانی بزرگ چندزبانه است که برای پشتیبانی از حدود ۵۰۰ زبان مختلف، از جمله زبان‌های کم‌منابع، طراحی شده است. این مدل با استفاده از مجموعه داده‌های متنوع و باکیفیت، به گونه‌ای آموزش دیده که بتواند در طیف وسیعی از وظایف زبانی مانند ترجمه، خلاصه‌سازی، پاسخ به پرسش و درک متن عملکرد مناسبی داشته باشد. از ویژگی‌های مهم GLOT500 می‌توان به پوشش وسیع زبانی و حفظ دقت در زبان‌های کم‌منابع اشاره کرد. در این تمرین ما از این مدل برای بازیابی جواب سوالات خود استفاده کردیم. این مدل زبانی در دو تنظیمات zero-shot و fine-tune شده در این تمرین استفاده شده که در ادامه نحوه استفاده از آن، آموزش و در نهایت انجام ارزیابی آن را توضیح خواهیم داد.

مستندات اجرا

در این قسمت نوتبوک nlp-2 توضیح داده خواهد شد.

بررسی گام به گام فرایند آموزش

ابتدا به قسمت لود دیتا می‌پردازیم. برای راحتی یک کلاس در نظر گرفته شده است به مانند زیر

```
class QAPairsDataset(Dataset):
    """
    A simple Dataset that returns (question, passage) pairs.
    Expects a list of dicts: {"question": str, "passage": str}.
    """
    def __init__(self, data): self.data = data
    def __len__(self): return len(self.data)
    def __getitem__(self, idx): return self.data[idx]
```

در ادامه نیز یک کلاس برای فرایند MLM میسازیم به مانند زیر

```
class MLMDataset(Dataset):
    """
    Dataset for MLM: returns masked inputs and labels for Masked Language Modeling.
    """
    def __init__(self, texts, tokenizer, max_length=128, mask_prob=0.15):
        self.texts = texts
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.mask_prob = mask_prob

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        enc = self.tokenizer(
            text,
            truncation=True,
            max_length=self.max_length,
            return_special_tokens_mask=True
        )
        input_ids = enc['input_ids']
        labels = input_ids.copy()
        for i in range(len(input_ids)):
            if random.random() < self.mask_prob and enc['special_tokens_mask'][i] == 0:
                input_ids[i] = self.tokenizer.mask_token_id
            else:
                # Use -100 to signal CrossEntropyLoss.ignore_index, so that
                # unmasked positions are ignored in the MLM loss computation.
                labels[i] = -100
        return {
            'input_ids': torch.tensor(input_ids),
            'labels': torch.tensor(labels)
        }
```

در این قسمت در تابع `__getitem__` 15 درصد توکن ها ماسک گذاشته خواهند شد همانگونه که در کامنت توضیح داده شده است برای ماسک گذاشتن صرفاً کافی است مقدار لیبل این مقادیر را برابر 100- بگذاریم که یک قرارداد است و هنگام فرایند آموزش این توکن ها در نظر گرفته نمی شوند و لاسی برای آن ها محاسبه نمی شود. در قسمت بعدی دو تابع `collate_qa` و `collate_mlm` قرار دارند این تابع ها وظیفه توکنایز کردن و پد گذاشتن در دیتا را بر عهده دارند.

برای این تمرین مطابق تصویر زیر از یک Bi-Encoder-Retriever استفاده کردیم که از یک ترنسفورمر مشترک بین سوالات و متن ها استفاده می کند این مدل انکودر ها به صورت جدا سوالات و متن هارا انکود می کنن بر خلاف cross-encoder ها که به ازای هر جفت سوال-متن یک ترنسفورمر ران می کنند که از نظر هزینه زمانی بسیار بالا تر است. در نهایت خروجی ها به صورت mean-pool شده باز گردانده می شود. دقت کنید که ترنسفورمر هایی مانند XLM-RoBERTa خروجی را به صورت یک آرایه $L \times D$ باز می گردانند برای اینکه ما خروجی را به فرمت یک بردار یک بعدی در بیاوریم نیاز به یک فرایند pooling داریم که اینجا mean-pooling انتخاب شده است. دقت کنید که توکن هایی که برای پدینگ اضافه شده اند را به حساب نمیآوریم.

```
def mean_pooling(token_embeddings, attention_mask):
    """
    Mean-pool token embeddings, masking out padding tokens.
    """
    mask = attention_mask.unsqueeze(-1).float()
    summed = torch.sum(token_embeddings * mask, dim=1)
    counts = torch.clamp(mask.sum(dim=1), min=1e-9)
    return summed / counts

class BiEncoderRetriever(nn.Module):
    """
    Bi-encoder model using a shared transformer for both questions and passages.
    """
    def __init__(self, model_name="cis-lmu/glot500-base"):
        super().__init__()
        self.encoder = AutoModel.from_pretrained(model_name)

    def forward(self, input_ids, attention_mask):
        outputs = self.encoder(
            input_ids=input_ids,
            attention_mask=attention_mask,
            return_dict=True
        )
        return mean_pooling(outputs.last_hidden_state, attention_mask)
```

در قسمت بعدی لاس مورد نیاز برای contrastive Training را تعریف می کنیم. در این لاس ابتدا امبدینگ سوالات و متن ها نرمالایز شده سپس با استفاده از ضرب داخلی میزان شباهت آن ها سنجیده می شود. هدف این است که امبدینگ سوالات به متن های مربوط به خود نزدیک شود و از متن های دیگر دور شود. دقت کنید که خروجی تابع matmul یک ماتریس است که درایه i و j آن شباهت بین سوال i ام و متن j ام را می سنجد. سپس از کراس آنروپی استفاده می کنیم که در آن لیبیل درست برای سوال i در واقع متن شماره i است.

```
def info_nce_loss(q_embeddings, p_embeddings, temperature=0.05):
    """
    Compute InfoNCE contrastive loss between question and passage embeddings.

    Args:
        q_embeddings (Tensor): Question embeddings [B, D].
        p_embeddings (Tensor): Passage embeddings [B, D].
        temperature (float): Scaling factor for logits.

    """
    q_norm = nn.functional.normalize(q_embeddings, dim=1)
    p_norm = nn.functional.normalize(p_embeddings, dim=1)
    sims = torch.matmul(q_norm, p_norm.t()) / temperature
    labels = torch.arange(sims.size(0), device=sims.device)
    return nn.CrossEntropyLoss()(sims, labels)
```


در قسمت بعد به تابع `train` می‌رسیم که به دلیل طولانی بودن کد آن در اینجا قرار داده نمی‌شود برای دیدن آن می‌توانید به ژوپیتِر نوتبوک رجوع کنید.


در تصویر زیر کانفیگ برای آموزش مدل را مشاهده می‌کنید:

```
train_path      = '/kaggle/input/training-data/training_data.json'      # QA training data
questions_path  = '/kaggle/input/evaluation/questions.json'             # List of questions
passages_path   = '/kaggle/input/evaluation/passages.json'              # Retrieval corpus
model_name      = 'cis-lmu/glot500-base'
output_dir      = './retriever_ckpt'
epochs          = 6
batch_size      = 8
learning_rate   = 2e-5
max_length      = 500
warmup_steps    = 1000
do_mlm          = True
mlm_epochs      = 2
mlm_batch_size  = 1
device          = 'cuda' if torch.cuda.is_available() else 'cpu'
```


پس از ران کردن مدل با توجه به اینکه سرعت کاهش لاس کاهش یافت و ریزالت مدل آموزش دیده شده خوب بود نتیجه گرفتیم که 6 اپیک برای آموزش مدل کافی است. Batch-size را به علت محدودیت vram برابر با 8 گذاشتیم. لرنینگ ریت را نیز پس از تست چندین عدد مختلف و مشاهده لاس برابر $2e-5$ قرار دادیم. دقت کنید که مدل رتریور محدودیت 512 توکن برای یک متن را دارد پس ما max_length را برابر با 500 قرار دادیم. در نهایت نیز یک linear_scheduler با هزار قدم warm-up برای تنظیم لرنینگ ریت در نظر گرفتیم. برای فرایند mlm هم با توجه به تحلیل لاس و ریزالت مدل نتیجه گرفتیم که 2 اپیک کافی است. (حتی یک اپیک نیز احتمالا کافی بوده است).

در تصویر زیر لاس هنگام فرایند mlm را مشاهده می‌کنیم:

model.safetensors: 100%  1.58G/1.58G [00:06<00:00, 565MB/s]

MLM Epoch 1/2: 100%  9143/9143 [58:24<00:00, 2.58it/s]

MLM Epoch 1/2 - Loss: 1.0447

MLM Epoch 2/2: 100%  9143/9143 [58:28<00:00, 2.67it/s]

MLM Epoch 2/2 - Loss: 0.6468

در تصویر زیر نیز لاس هنگام فرایند آموزش را مشاهده می کنیم:

Contrast Epoch 1/6: 100%  1143/1143 [14:57<00:00, 1.36it/s]

Contrastive Epoch 1/6 - Loss: 0.7086

Contrast Epoch 2/6: 100%  1143/1143 [14:56<00:00, 1.27it/s]

Contrastive Epoch 2/6 - Loss: 0.1928

Contrast Epoch 3/6: 100%  1143/1143 [14:57<00:00, 1.32it/s]

Contrastive Epoch 3/6 - Loss: 0.1167

Contrast Epoch 4/6: 100%  1143/1143 [15:00<00:00, 1.27it/s]

Contrastive Epoch 4/6 - Loss: 0.0915

Contrast Epoch 5/6: 100%  1143/1143 [14:57<00:00, 1.33it/s]

Contrastive Epoch 5/6 - Loss: 0.0674

Contrast Epoch 6/6: 100%  1143/1143 [14:59<00:00, 1.27it/s]

Contrastive Epoch 6/6 - Loss: 0.0573

Training complete.

همانگونه که مشاهده می کنید بنظر لاس همگرا شده است.

مشکل مدل پایه:

هنگام لود مدل پایه از hugging-face با وارنینگ زیر روبرو میشویم

```
Some weights of XLMRobertaModel were not initialized from the model checkpoint at cis-lmu/glot500-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

دلیل این اخطار این است که مدل glot-500 نسبت به فرمت hugging-face چندین وزن کمتر دارد و hugging-face آن وزن‌ها را با عدد‌های رندوم پر کرده است پس علنا انتظار داریم این مدل بسیار بد عمل کند زیرا چندین وزن آن رندوم است. همانگونه که در متن نوشته شده است انتظار داریم با آموزش این مدل عملکرد آن بسیار پیشرفت کند.

در نهایت نیز به فرایند evaluation میرسیم. برای این بخش 50 سوال که توسط انسان تهیه شده است لود میشود (این سوالا در فایل question.json ضمیمه شده است). و متن‌های قسمت آموزش را نیز به صورت مستقل از سوالات خود بار دیگر لود می‌کنیم (این متن‌ها در فایل passages.json ضمیمه شده است)

در نهایت به کمک تابع زیر این متن‌ها و سوالات را به کمک مدل (پایه یا آموزش دیده شده) انکود می‌کنیم.

```
def retrieve_embeddings(model, tokenizer, texts, batch_size=16):
    embs=[]
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        enc = tokenizer(batch, padding=True, truncation=True,
                        max_length=max_length, return_tensors='p
t')
        for k in enc: enc[k] = enc[k].to(device)
        with torch.no_grad(): out = model(**enc)
        embs.append(mean_pooling(out.last_hidden_state, enc['atte
ntion_mask'])).cpu())
    return torch.cat(embs, 0)
```

سپس top-3 متن‌هایی که امبدینگ آن‌ها به هر امبدینگ هر سوال نزدیک هست را خروجی می‌دهیم. دقت کنید که خروجی‌های مدل‌ها و tf-idf در فایل results.csv قرار دارد.

در قسمت بعدی به بخش tf-idf می‌رسیم.

```
# -----  
# TF-IDF Baseline  
# -----  
vectorizer = TfidfVectorizer(max_features=50000)  
corpus_tfidf = vectorizer.fit_transform(passages)  
q_tfidf = vectorizer.transform(questions)  
sims_tfidf = cosine_similarity(q_tfidf, corpus_tfidf)  
top3_tfidf = sims_tfidf.argsort(axis=1)[:, -3:][:, :-1]
```

در این قسمت از تابع TfidfVectorizer که در کتاب خوانه sklearn وجود دارد استفاده می‌کنیم. و 3 متن نزدیک به هر سوال را انتخاب می‌کنیم.

بخش امتیازی

در این قسمت می خواهیم عملیات duplicate-detection را به کمک انکودر آموزش دیده شده انجام دهیم. کد آن به شرح زیر است:

```
import faiss

# Build index for passage embeddings
pass_emb_ft_np = pass_emb_ft.cpu().detach().numpy().astype('float32')
faiss.normalize_L2(pass_emb_ft_np)
dim = pass_emb_ft_np.shape[1]
index = faiss.IndexFlatIP(dim)
index.add(pass_emb_ft_np)

# Search each passage's nearest neighbor (k=2 includes itself)
D, I = index.search(pass_emb_ft_np, 2)
duplicates = []
threshold = 0.98
for i in range(len(passages)):
    neighbor = I[i,1]
    score = float(D[i,1])
    if neighbor != i and score > threshold:
        duplicates.append((i, neighbor, score))

# Save duplicates to CSV
dup_csv = 'duplicates.csv'
with open(dup_csv, 'w', newline='', encoding='utf-8-sig') as f:
    writer = csv.writer(f)
    writer.writerow(['idx1', 'idx2', 'score', 'text1', 'text2'])
    for i,j,score in duplicates:
        writer.writerow([i, j, f'{score:.4f}', passages[i], passages[j]])
print(f"Saved duplicates to {dup_csv}")
```

در این قسمت از کتاب خانه faiss استفاده کرده ایم که برای فرایند پیدا کردن نزدیک ترین امبدینگ ها در اردر زمانی خوب کمک کننده است. در نهایت برای هر متن نزدیک ترین متن به آن را پیدا می کنیم و اگر درصد شباهت آن ها بیشتر از 90 درصد بود آن ها را به عنوان duplicate تلقی می کنیم. دقت کنید که از ضرب داخلی برای تعیین میزان شباهت استفاده می کنیم. در نهایت خروجی ها در فایل deulicates.csv محیا شده است. در این فایل دو متن و ایندکس آن ها و درصد شباهت آن ها را گزارش کردیم.

برای مثال در سطر 487 این فایل دو متن 1547 و 230 قرار با درصد شباهت 96 درصد قرار گرفته اند.

این دو متن به شرح زیر اند:

متن 1547:

برویشین ترش، غذای محلی خوشمزه‌ای از استان کردستان و شهر کردستان است که با موادی ساده و در دسترس تهیه می‌شود. برای پخت این آش لذیذ، به ۵۰۰ گرم بلغور، ۱۰۰ گرم لپه، ۲۰۰ گرم زردآلو، نیم پیمانه سرکه، یک عدد پیاز، سه قاشق غذاخوری روغن و مقداری زردچوبه، فلفل و نمک نیاز دارید. ابتدا پیاز را خرد کرده و در روغن سرخ می‌کنیم. سپس سه پیمانه آب به پیاز داغ اضافه می‌کنیم و بلغور، لپه و زردآلو را به آن می‌افزاییم. بعد از اینکه مواد پختند، سرکه، نمک، فلفل و زردچوبه را به آش اضافه کرده و آن را برای صرف در وعده‌های نهار یا شام آماده می‌کنیم. این غذا به عنوان یک غذای اصلی و مقوی، بسیار محبوب است.

متن 230:

برویشین ترش، یکی از آش‌های خوشمزه و سنتی استان کردستان و شهر سنندج است که معمولاً در موقعیت‌های روزمره و دورهمی‌های خانوادگی تهیه می‌شود. برای پخت این آش، به مواد زیر نیاز دارید: ۵۰۰ گرم بلغور، ۱۰۰ گرم لپه، ۲۰۰ گرم زردآلو، نیم پیمانه سرکه، یک عدد پیاز، سه قاشق غذاخوری روغن و همچنین زردچوبه، فلفل و نمک به میزان لازم. ابتدا پیاز را خرد کرده و با روغن در قابلمه سرخ می‌کنیم. سپس سه پیمانه آب به پیاز داغ اضافه می‌کنیم. در این مرحله بلغور، لپه و زردآلو را به قابلمه می‌افزاییم و اجازه می‌دهیم بپزند. پس از پخته شدن مواد، سرکه، نمک، فلفل و زردچوبه را اضافه کرده و آش را به صرف می‌رسانیم. این آش با طعم ترش و شیرین خود، یک انتخاب عالی برای یک وعده غذایی مقوی و لذیذ است.

همانگونه که مشاهده می‌کنید این دو متن بسیار به هم شبیه هستند. و مدل به درستی داپلیکیت‌ها را پیدا کرده است.

ارزیابی تحلیلی:

در ادامه، قرار است تا با توجه به نتایج labeling و همچنین روش‌هایی که برای پیاده‌سازی مدل‌ها استفاده شد، نتایج بدست آمده را تحلیل کرده و براساس آن‌ها به یک نتیجه نهایی اما نه کلی برسیم. در این بخش، برخی از نکات قابل توجه و دلایل وجود آن‌ها را بررسی می‌کنیم:

- همان‌طور که قابل حدس بود، با سخت شدن سوالات، عملکرد مدل‌ها افت کرد. در این جا منظور از سختی، پیچیدگی و تعداد اجزای سوال و همچنین منطق و هدف سوال را شامل می‌شود. برخی از سوالات، با درگیر کردن مقدار بسیار کمی داده، خواسته کمی داشته و به دنبال هیچ‌گونه reasoning ای نیستند. از سوی دیگر برخی سوالات، با وارد کردن داده‌های زیاد، سوال را پیچیده کرده که این موضوع مدل را به چالش می‌کشد. یکی از نکات قابل توجهی که در مورد مدل‌های طراحی شده می‌توان به آن اشاره کرد، فرار آن‌ها در برخی از سوالات بود. در داده‌های ما، برخی از پاراگراف‌ها،

به دلیل نبود اطلاعات کامل در داده‌های خام، به درستی شکل نگرفتند و مدل زبانی تصمیم گرفته تا صرفاً عدم توانایی در تولید متن را اطلاع دهد. مثال آن در زیر قابل مشاهده است:

```
متاسفانه، اطلاعات ارائه شده برای نوشتن یک پاراگراف کامل در مورد نان کرویسی "output2":  
بسیار ناقص است. تنها عنوان "\نان کرویسی\" و محل تهیه آن، یعنی استان هرمزگان و شهر  
برای تهیه این نان و (ingredients) پندربانی با مختصات جغرافیایی مشخص شده اند. هیچ موادی  
برای پخت آن ذکر نشده است. بنابراین، نمیتوان یک پاراگراف (instructions) هیچ دستورالعملی  
اگر اطلاعات مربوط به مواد لازم و مراحل تهیه نان\n\n.توصیفی و کامل در مورد این نان نوشت  
کرویسی را در اختیارم بگذارید، خوشحال می‌شوم یک پاراگراف روان و طبیعی به زبان فارسی  
در\n\n.برای استان بنویسم
```

در فرایند evaluation، برخی سوالات به دلیل منطق و reasoning پشت آن‌ها، مدل‌ها را به چالش کشیدند. این چالش در حدی بود که مدل‌ها، مخصوصاً tf-idf، تصمیم گرفتند تا پاراگراف‌های ناقص را به عنوان جواب انتخاب کنند زیرا نتوانسته بودند هیچ موردی که به صورت سوال نزدیک باشد پیدا کنند. این مورد به ما می‌گوید در شرایطی که مدل نتواند جوابی پیدا کند، ترجیح می‌دهد تا عدم ناتوانی را اعلام کند به جای آن که جوابی را تحت هر شرایطی به استفاده‌کننده اعلام کند. این مورد در مدل آموزش دیده‌شده کمتر دیده شد زیرا طبیعتاً این مدل از قدرت reasoning بیشتری برخوردار بود و بر روی داده‌های مشابه آموزش دیده شده بود.

- یکی از شرایطی که باعث به چالش کشیدن تمام مدل‌ها شد، وجود سوالاتی بود که جواب آن‌ها در چندین متن دیده می‌شد. به عنوان مثال، یکی از سوالات بخش ارزیابی، خواسته بود تا بین آش‌های دو استان مقایسه انجام دهد. این گونه سوالات، به دلیل وجود جواب آن‌ها در چند پاراگراف، ما را به نتیجه مطلوب نمی‌رسانند. در این شرایط مدل می‌تواند متوجه وجود برخی کلمات کلیدی مانند آش و یا نام استان شود اما در جواب خود فقط می‌تواند به یک بخش به صورت خاص بپردازد.
- یکی از موارد قابل توجه، عدم توانایی مدل‌ها در جواب به سوالاتی بود که با استفاده از کلمات خود مفهوم خاصی را می‌رسانند و دارای بعد reasoning هستند. این سوالات، خواسته خود را به صورت واضح بیان نمی‌کنند اما به دلیل ارتباط بین کلمات استفاده شده در آن‌ها، ما می‌توانیم منظور سوال را به خوبی متوجه شویم. این مشکل خود را به طور جدی در مدل tf-idf و zero-shot نشان می‌دهد. به عنوان مثال، یکی از سوالات ما، خواستار غذاهایی شد که در آن‌ها برنج استفاده نشده است. در این حالت، مدل tf-idf صرفاً به دلیل وجود کلمه برنج، پاراگراف‌هایی را پیشنهاد داد که در آن برنج استفاده شده است (در واقع کم پیش می‌آید دستور پختی بگوید از ماده غذایی‌ای استفاده نمی‌کند). در سوی دیگر، مدلی که آموزش دیده بود، به وجود کلمه دقت بیشتری کرده بود و در نتیجه نتایج بهتری را برای ما بازایی کرد. یکی از نکات مثبت این مدل، توجه به روابط بین کلمات است که باعث عملکرد بهتر آن نسبت به بقیه مدل‌ها شده است.
- از دیگر مشاهداتی که می‌توان آن را نتیجه نکته قبلی در نظر گرفت، نزدیک شدن عملکرد مدل‌ها به یکدیگر در سوالات ساده می‌باشد. در این سوالات، که معمولاً منطقی پشت آن‌ها نیست و با یک کلمه کلیدی هدف سوال مشخص می‌شود، مدل‌های tf-idf و آموزش داده شده نتیجه حدوداً یکسانی داشتند. در این سوالات، مدل tf-idf به صرف دیدن کلمه کلیدی، پاراگراف مدنظر را پیدا کرده و آن را خروجی می‌دهد. در این گونه سوالات، استفاده از یک مدل آماری مانند

مدل tf-idf ای که استفاده کردیم، خواسته ما را با منابع بسیار کمتر و با سرعت بیشتری خروجی می‌دهد و در نتیجه گزینه بهتری برای استفاده در این سناریوها می‌باشد.

- در نهایت به یک دسته سوال دیگر و تفاوت عملکرد بین مدل آموزش‌دیده و دو مدل دیگر می‌پردازیم. در این دسته از سوالات، داده‌های زیادی در سوال داده شده و این داده‌ها هرکدام ممکن است یک فضای جدیدی را در سوال ایجاد بکنند. در این سوالات، علی‌رغم آن که مدل tf-idf کلمات کلیدی را به خوبی تشخیص می‌داد، نمی‌توانست تمام ابعاد سوال را یکجا در نظر گرفته و پاراگرافی که نزدیک‌ترین اشتراک را داشت به عنوان خروجی بدهد. در واقع، وجود برخی کلمات تکراری و یا اضافه باعث می‌شد تا مدل از رسیدن به چند هدف خود دور شود. از طرف دیگر، مدل آموزش‌دیده، با کنار هم قرار دادن تمام داده‌ها سعی می‌کرد تا ارتباط بین آن‌ها را درک کرده و متنی پیدا کند که این ارتباط را در خود جا داده است.

مقایسه مدل آماری و مدل آموزش‌دیده شده

با توجه به نکات بالا، ما می‌توانیم به یک جمع‌بندی در مورد مقایسه بین مدل آماری و مدل آموزش‌دیده برسیم. همان‌طور که دیدیم، در سوالاتی که فضای سوال به راحتی از خود صورت قابل درک است و داده‌های درون سوال مقدار کمی دارند، مدل آماری پا به پای مدل آموزش‌دیده، جواب‌های درست را به ما تحویل می‌دهد. به دلیل آن که روش‌های آماری اکثراً بر پایه شمارش هستند، با داشتن شواهد دقیق، ساده و مشخص به راحتی می‌توانند بازیابی را انجام دهند. در نتیجه، در کارهایی که سوالات ساده می‌باشند، استفاده از این مدل، به دلیل سرعت، سادگی و استفاده کم از منابع، نسبت به مدل آموزش‌دیده پیشنهاد می‌شود. از سوی دیگر، در بالا به سوالاتی پرداختیم که مفهوم آن‌ها در ظاهر دیده نمی‌شود. در این موارد، مدل‌های آماری، به دلیل نداشتن evidence کافی قادر به بازیابی درست نیستند. در نتیجه ما به مدل‌هایی نیاز داریم که بتوانند ارتباط بین کلمات و مفاهیم پنهان را به خوبی تشخیص دهند. مدل زبانی آموزش‌دیده، به دلیل منابع بالایی که با آن آموزش‌دیده و همچنین آموزش مجددی که بر روی داده‌های خاصی صورت گرفته است (fine-tuning) از قدرت خوبی برای پیدا کردن ارتباطات و مفاهیم برخوردار است و در صورت داشتن سوالات پیچیده، حتی اگر سرعت آن کمتر و هزینه آن بیشتر باشد نسبت به مدل‌های آماری ترجیح داده می‌شود.

تمامی نتایج بالا از تحلیل و ارزیابی انسانی در هنگام لیبل‌گذاری صورت گرفته است که با پیش‌فرض ما همخوانی دارد.

مقایسه مدل پایه و مدل آموزش‌دیده شده

همانگونه که در بخش مستندات نشان دادیم در مدل پایه یک سری از پارامترها به صورت رندوم پر شده‌اند که این باعث می‌شود دقت این مدل بدون هیچگونه آموزش بسیار بد عمل کند. بر خلاف این موضوع هنگامی که مدل را آموزش می‌دهیم آن لایه‌های رندوم مطابق خواسته و تسک ما fine-tune خواهد شد و انتظار یک جهش عملکردی بسیار بالا را در مدل آموزش‌دیده شده داریم. این موضوع خود را در نتایج بدست آمده نمایش داده و همچنین به وضوح در هنگام label زدن امری مشهود بود.

در فرآیند ارزیابی مدل ها از ۵۰ سوال استفاده کردیم که ۲۵ سوال اول سطح دشواری بیشتری داشتند و ۲۵ سوال دوم آسان تر بودند. در ۲۵ سوال اول سوالاتی وجود دارد که متن آن ها نیاز به تحلیل دارد و مدل باید توانایی تحلیل داشته باشد. ۲۵ سوال دوم آسان تر هستند و متن سوالات نیاز به تحلیل خاصی ندارد و بعضا با دادن مواد اولیه غذا به دنبال آن می گردیم و متن سوال بدون دشواری است.

پس از آماده شدن نتایج مدل ها که از هر مدل ۳ خروجی برای هر سوال آماده شد و در مجموع ۹ خروجی برای هر سوال آماده شد. به منظور اینکه عدالت در لیبیل زدن رعایت شود باید اینکه هر خروجی از چه مدلی است از دید ارزیاب پنهان بماند. برای این منظور از نفر سوم که در ارزیابی دخالتی ندارد خواسته شد تا خروجی ها از ترتیب اولیه جایگشت دهد تا مدلی که خروجی ها از آن آمده اند نامعلوم باشد و ارزیابی ها تا حد امکان فاقد سوگیری باشد.

قبل از اعمال جایگشت :

```
tfidf1,tfidf2,tfidf3,zero1,zero2,zero3,ft1,ft2,ft3
```

ترتیب جدید بعد از اعمال جایگشت :

```
tfidf2,zero2,tfidf1,zero1,ft2,zero3,ft3,ft1,tfidf3
```

سپس دود نفر دیگر بدون دانستن ترتیب اقدام به ارزیابی خروجی ها برای هر سوال کردند. در ارزیابی براساس میزان ارتباط خروجی ها با سوال پرسیده شده عددی بین ۱ تا ۹ اختصاص داده شد. به مرتبط ترین پاسخ خروجی عدد ۱ و به نامرتب ترین خروجی عدد ۹ اختصاص داده شده است. و در مجموع ما ۱۰۰ داده ارزیابی شده داریم که هر کدام ۹ خروجی دارند که براساس میزان ارتباط و درستی خروجی به آنها عددی بین ۱ تا ۹ اختصاص داده شده است.

معیارهای مورد استفاده :

برای ارزیابی کیفی و مقایسه عملکرد مدل های مختلف در بازایی پاسخ های مرتبط، از سه معیار استفاده کرده ایم:

- میانگین رتبه
- Top-K count
- Worst Rank count

معیار میانگین رتبه

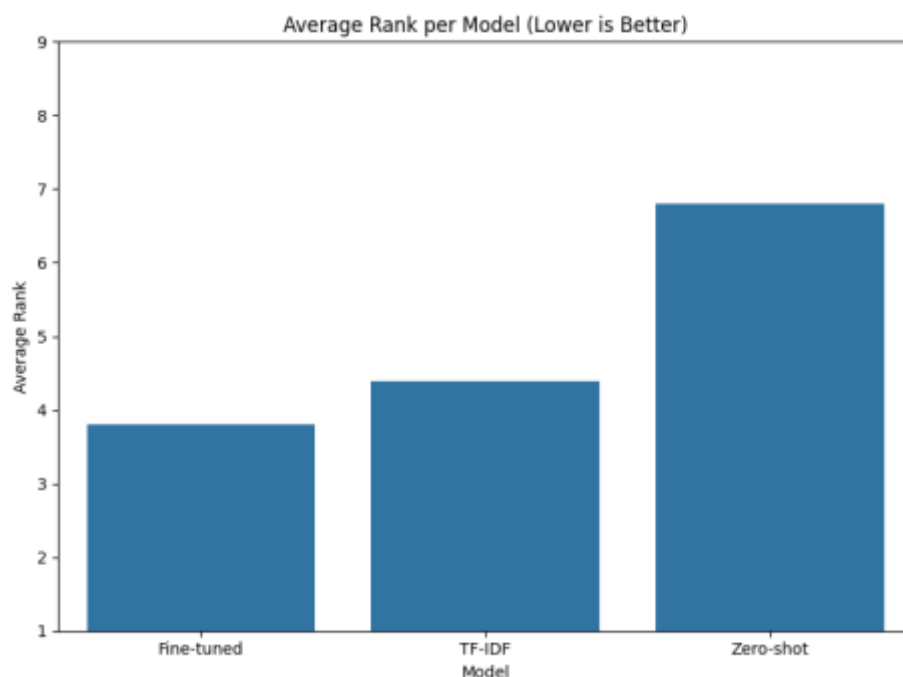
برای هر مدل، میانگین رتبه های مربوط به خروجی های آن محاسبه شده است. این معیار تصویری کلی از میانگین کیفیت پاسخ های تولید شده توسط هر مدل ارائه می دهد. میانگین رتبه پایین تر نشان دهنده عملکرد بهتر مدل است. مدلهایی که به صورت پایدار پاسخ های مرتبط تری تولید می کنند، میانگین رتبه ی پایین تری دارند.

برای مدل M ، اگر تعداد کل پاسخ‌های آن N باشد و رتبه‌ی هر پاسخ با r_i نمایش داده شود، آنگاه:

$$AverageRank(M) = \frac{1}{N} \sum_{i=1}^N r_i$$

نتیجه این معیار برای ۳ مدل تست شده و نمودار آن به صورت زیر است:

Model Name	Fine-tuned	TF-IDF	Zero-shot
AverageRank	3.81	4.38	6.80



همانطور که مشاهده می‌کنید طبق این معیار که نشان دهنده میانگین عملکرد مدل‌ها است، مدل فاین تیون شده بهترین عملکرد را دارد و در جایگاه بعدی TF-IDF قرار می‌گیرد.

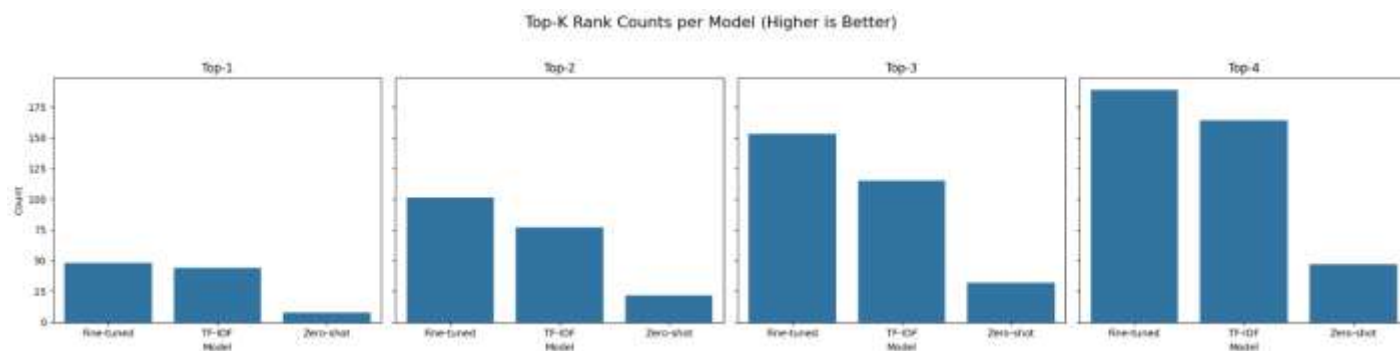
معیار Top-k count

معیار Top-K Rank Count مشخص می‌کند که هر مدل چند بار توانسته یکی از K پاسخ برتر باشد. این معیار نشان می‌دهد که پاسخ‌های قابل قبول هر مدل چقدر تکرار می‌شوند، حتی اگر همیشه رتبه‌ی اول را کسب نکنند. برای مثال، در معیار Top-1، بررسی می‌شود که هر مدل چند بار پاسخ رتبه ۱ (بهترین پاسخ) را داده است. در Top-3، پاسخ‌هایی با رتبه‌های ۱، ۲ یا ۳ در نظر گرفته می‌شوند. این معیار مناسب برای تحلیل پایداری مدل و مکمل خوبی برای میانگین رتبه مدل است.

$$TopKCount(M, K) = \sum_{i=1}^N 1(r_i \leq K)$$

نتیجه ی این معیار برای ۳ مدل تست شده و نمودار به صورت زیر است:

Model Name	Fine-tuned	TF-IDF	Zero-shot
Top1Count	48	44	8
Top2Count	101	77	22
Top3Count	153	115	32
Top4Count	189	164	47



همانطور که در نمودارهای مربوط به Top-1 تا Top-4 دیده می شود که مدل Fine-tuned در تعداد زیادی از سوالات در بین بهترین پاسخ ها قرار گرفته است و بعد از آن مدل TF-IDF در جایگاه بعدی قرار می گیرد.

معیار Worst Rank Count

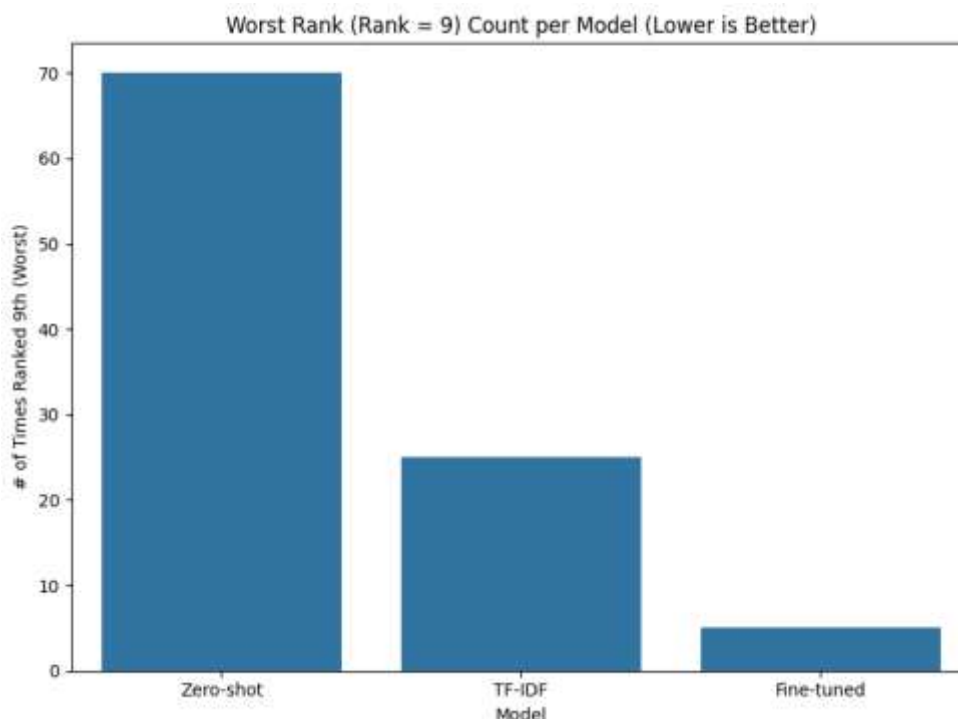
این معیار برای سنجش ریسک عملکرد ضعیف یک مدل استفاده می شود. معیار Worst Rank Count بررسی می کند که مدل چند بار خروجی ای تولید کرده که از دید ارزیاب بدترین پاسخ ممکن (رتبه = ۹) بوده است. حتی اگر مدلی در میانگین خوب باشد، ولی زیاد دچار خطاهای فاحش شود (یعنی زیاد در رتبه ۹ قرار بگیرد)، نمی توان به آن اعتماد کامل داشت. این معیار به ما کمک می کند چنین مدل هایی را شناسایی کنیم.

اگر مقدار WorstRankCount برای مدلی زیاد باشد، نشان دهنده ی آن است که مدل اغلب پاسخ های بی کیفیت یا نامربوط تولید کرده است. در مقابل، مقدار پایین برای این معیار نشانه ی پایداری و خطای کم در عملکرد مدل است و مکمل خوبی برای معیارهای میانگین و Top-K است و مجموعه ای معیار های مناسب برای بررسی مدل ها هستند.

$$WorstRankCount(M) = \sum_{i=1}^N 1(r_i = 9)$$

نتیجه این معیار برای ۳ مدل تست شده و نمودار آن به صورت زیر است:

Model Name	Fine-tuned	TF-IDF	Zero-shot
WorstRankCount	5	25	70



استفاده‌ی همزمان از سه معیار میانگین رتبه و Top-K و تعداد رتبه‌های بدترین باعث ارزیابی جامع و دقیق عملکرد مدل‌ها می‌شود؛ زیرا میانگین رتبه تصویری کلی از کیفیت پاسخ‌ها ارائه می‌دهد، Top-K نشان می‌دهد مدل چند بار در بین بهترین پاسخ‌ها بوده (پایداری عملکرد)، و تعداد رتبه‌های ۹ مواردی را مشخص می‌کند که مدل کاملاً ضعیف عمل کرده و این ترکیب کمک می‌کند نقاط قوت و ضعف مدل‌ها را همزمان بررسی کنیم و ارزیابی متعادل‌تری داشته باشیم.

نتایج این سه معیار مکمل یکدیگر هستند و نشان می‌دهند که مدل Fine-tuned **بهترین عملکرد** را داشته است، در حالی که مدل TF-IDF عملکرد قابل قبولی دارد اما با درصد بیشتری از پاسخ‌های ضعیف، و مدل Zero-shot نسبت به سایر مدل‌ها عملکرد ضعیف‌تری از خود نشان داده است.

چالش‌ها:

در انجام این پروژه ما به چالش‌هایی برخوردیم که در ادامه برخی از آن‌ها را معرفی می‌کنیم:

- عدم وجود ساختار یکسان در داده‌های خام اولی باعث شد تا لازم باشد preprocessing قبل از شروع پروژه انجام شود. داده‌های خام ساختار یکسانی نداشتند و نمی‌توانستیم با صرفاً یک کد، تمام آن‌ها را بررسی کنیم.

- هذیان‌گویی مدل‌های زبانی generative گاهی باعث می‌شد تا پاراگراف‌های تولیدی ساختار مدنظر ما را نداشته باشند و ما مجبور به گرفتن مجدد خروجی و یا تغییر prompt می‌شدیم. طراحی prompt مناسب که بتواند منظور را در متنی کوتاه به مدل برساند کاری بود که به سادگی انجام نشد.
- عملیات لیبل‌گذاری و گرفتن خروجی و داده‌هایی که بتوان از آن‌ها در قسمت ارزیابی استفاده کرد نیز خود کاری وقت‌گیر و چالشی بود زیرا بررسی 9 * 50 متن به خودی خود کار طولانی‌ای می‌باشد.
- زمان طولانی آموزش مدل زبانی خود از دیگر چالش‌هایی بود که باعث می‌شد ما هر تغییری را با دقت انجام دهیم تا نیاز به آموزش مجدد نباشد.

تجربه اجرای پروژه:

این پروژه که برای تمام اعضای تیم تجربه جدیدی بود، ما را با مفاهیم جدیدی آشنا کرد. در این پروژه ما با انواع چالش‌ها که شامل آماده‌سازی داده‌های خام و کار با مدل زبانی‌ای که fine-tuning بر روی آن انجام نشده بود روبه‌رو شدیم. این پروژه باعث شد تا به خوبی با مفهوم مدل‌های retrieval آشنا شده و روش‌های مختلف برای پیاده‌سازی آن را ببینیم. همچنین توانستیم تا این روش‌ها را با هم مقایسه کرده و مزایا و معایب هر کدام را در سناریوهای مختلف مشاهده کنیم. همچنین در این پروژه توانستیم توالی خوبی از قسمت قبل ببینیم و کاربرد مرحله قبل را به خوبی درک کنیم.