通过改进Embedding模型，将你的RAG上下文召回率提高95%

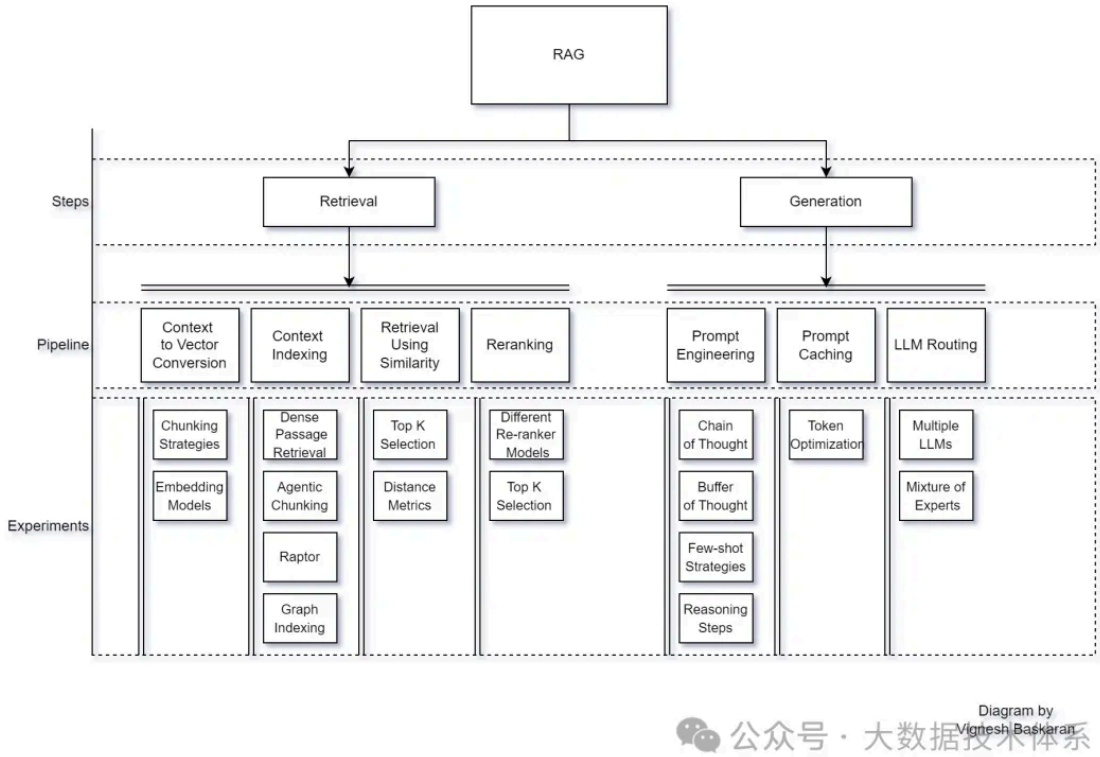
大数据技术体系 2024年10月23日 23:47 广东

通过改进嵌入模型，将你的RAG上下文召回率提高95%

检索增强生成（RAG）是一种将LLM（大型语言模型）集成到商业用例中的突出技术，它允许将专有知识注入LLM中。本文假设您已经了解RAG的相关知识，并希望提高您的RAG准确率。

让我们简要回顾一下这个过程。RAG模型包括两个主要步骤：检索和生成。在检索步骤中，涉及多个子步骤，包括将上下文文本转换为向量、索引上下文向量、检索用户查询的上下文向量以及重新排序上下文向量。一旦检索到查询的上下文，我们就进入生成阶段。在生成阶段，上下文与提示结合，然后发送给LLM以生成响应。在发送给LLM之前，可能需要进行缓存和路由步骤以优化效率。

对于每个管道步骤，我们将进行多次实验，以共同提高RAG的准确率。您可以参考以下图片，其中列出了在每个步骤中进行的实验（但不限于）。



开发者面临的一个主要问题是，在生产环境中部署应用程序时，准确性会有很大的下降。

“RAG在POC（原型）中表现最佳，在生产中最差。”这种挫败感在构建GenAI（通用人工智能）应用程序的开发者中很常见。

生成阶段已经通过一些提示工程得到了解决，但主要挑战是检索与用户查询相关且完整的上下文。这通过一个称为上下文召回率的指标来衡量，它考虑了为给定查询检索的相关上下文数量。检索阶段的实验目标是提高上下文召回率。

嵌入模型适配

在检索阶段进行的实验中，通过适配嵌入模型，可以显著地将您的上下文召回率提高+95%。

在适配嵌入模型之前，让我们了解其背后的概念。这个想法始于词向量，我们将训练模型理解单词的周围上下文（了解更多关于CBOW和Skipgram的信息）。在词向量之后，嵌入模型是专门设计来捕捉文本之间关系的神经网络。它们超越了单词级别的理解，以掌握句子级别的语义。嵌入模型使用掩码语言模型目标进行训练，其中输入文本的一定比例将被屏蔽以训练嵌入模型，以预测屏蔽的单词。这种方法使模型能够在使用数十亿个标记进行训练时理解更深层的语言结构和细微差别，结果生成的嵌入模型能够产生具有上下文感知的表示。这些训练好的嵌入模型旨在为相似的句子产生相似的向量，然后可以使用距离度量（如余弦相似度）来测量，基于此检索上下文将被优先考虑。

现在我们知道了这些模型是用来做什么的。它们将为以下句子生成相似的嵌入：

句子1：玫瑰是红色的

句子2：紫罗兰是蓝色的

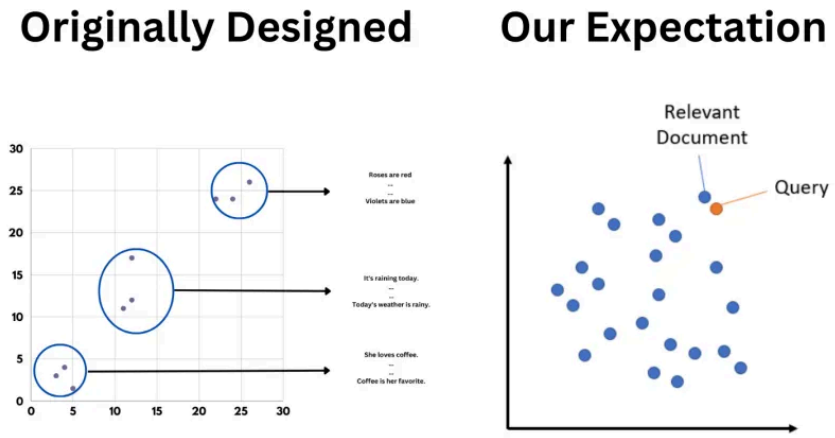
它们非常相似因为这两句都在谈论颜色。

对于RAG，查询和上下文之间的相似度分数应该更高，这样就能检索到相关的上下文。让我们看看以下查询和来自PubmedQA数据集的上下文。

查询：肿瘤浸润性免疫细胞特征及其在术前新辅助化疗后的变化能否预测乳腺癌的反应和预后？

上下文：肿瘤微环境免疫与乳腺癌预后相关。高淋巴细胞浸润与对新辅助化疗的反应相关，但免疫细胞亚群特征在术前和术后残余肿瘤中的贡献仍不清楚。我们通过对121例接受新辅助化疗的乳腺癌患者进行免疫组化分析，分析了术前和术后肿瘤浸润性免疫细胞（CD3、CD4、CD8、CD20、CD68、Foxp3）。分析了免疫细胞特征并与反应和生存相关。我们确定了三种肿瘤浸润性免疫细胞特征，它们能够预测对新辅助化疗的病理完全缓解（pCR）（B簇：58%，与A簇和C簇：7%相比）。CD4淋巴细胞的高浸润是pCR发生的主要因素，这一关联在六个公共基因组数据集中得到了验证。化疗对淋巴细胞浸润的影响，包括CD4/CD8比率的逆转，与pCR和更好的预后相关。对化疗后残余肿瘤中免疫浸润的分析确定了一个特征（Y簇），其主要特征是CD3和CD68浸润高，与较差的无病生存率相关。

查询和上下文看起来相似吗？我们是否在使用嵌入模型的方式中使用了它们的设计意图？显然，不是！



作者提供的左侧图像；右侧图像归功于：https://github.com/UKPLab/sentence-transformers/blob/master/docs/img/SemanticSearch.png，[Apache-2.0许可证](https://github.com/UKPLab/sentence-transformers#Apache-2.0-1-ov-file)

我们需要微调嵌入模型的原因是确保查询和相关的上下文表示更接近。为什么不从头开始训练呢？这是因为嵌入模型已经从数十亿个标记的训练中获得了对语言结构的理解，这些理解仍然可以加以利用。

微调嵌入模型

为了微调嵌入模型，我们需要包含类似预期用户查询和公司相关文档的数据集。我们可以利用语言模型（LLM）根据知识库文档生成查询。使用公司的知识库训练LLM就像提供了一个快捷方式，因为它允许嵌入模型在训练阶段本身访问上下文。

数据准备 - 训练和测试：

以下是数据准备步骤：

对于训练集：

1. 使用LLM从公司的知识库中挖掘所有可能的问题。
2. 如果知识库被分块，确保从所有块中挖掘问题。

对于测试集：

1. 从知识库中挖掘较少数量的问题。
2. 如果有，使用真实用户的问题。
3. 对训练集中的问题进行释义。
4. 结合并释义训练集和测试集中的问题。

我们中的大多数人都不会开发全领域的嵌入模型。我们创建的嵌入模型旨在在公司的知识库上表现更好。因此，使用公司的内部数据集训练嵌入模型并无害处。

对于本文，我们将使用Hugging Face上的"_qiaojin/PubMedQ"_数据集，它包含pubid、问题和上下文等列。pubid将用作问题ID。

```
from datasets import load_dataset
med_data = load_dataset("qiaojin/PubMedQA", "pqa_artificial", split="train")
med_data
```

```
DatasetDict({
    train: Dataset({
        features: ['pubid', 'question', 'context', 'long_answer', 'final_decision'],
        num_rows: 211269
    })
})
```


pubid 是一个唯一的ID，它指向行。我们将使用 pubid 作为问题ID。

为了训练嵌入模型，我们将使用sentence-transformer训练器进行训练，但你也可以使用huggingface训练器。此外，我们使用_MultipleNegativeRankingLoss_来微调我们的模型，但同样的效果也可以通过使用多种损失函数实现，例如_TripletLoss_、_ContrastiveLoss_等。但是，对于每种损失，所需的数据不同。例如，对于tripletloss，你需要（查询，正例上下文，负例上下文）对，而在MultipleNegativeRankingLoss中，你只需要（查询，正例上下文）对。对于给定的查询，除了正例上下文之外的所有上下文都将被视为负例。

在我们的PubMedQA数据集中，每一行的"question"列包含一个问题，"context"列包含适合该问题的上下文列表。因此，我们需要扩展上下文列表列，并创建包含相应上下文的新列的单独行。

```
dataset = med_data.remove_columns(['long_answer', 'final_decision'])

df = pd.DataFrame(dataset)
df['contexts'] = df['context'].apply(lambda x: x['contexts'])

# 展平上下文列表并重复问题
expanded_df = df.explode('contexts')

# 可选：如果需要，重置索引
expanded_df.reset_index(drop=True, inplace=True)

expanded_df = expanded_df[['question', 'contexts']]
splitted_dataset = Dataset.from_pandas(expanded_df).train_test_split(test_size=0.05)

expanded_df.head()
```

	question	contexts
0	Are group 2 innate lymphoid cells (ILC2s) in...	Chronic rhinosinusitis (CRS) is a heterogeneou...
1	Are group 2 innate lymphoid cells (ILC2s) in...	The aim of this study was to identify ILC2s in...
2	Are group 2 innate lymphoid cells (ILC2s) in...	A cross-sectional study of patients with CRS u...
3	Are group 2 innate lymphoid cells (ILC2s) in...	35 patients (40% female, age 48 ± 17 years) in...
4	Does vagus nerve contribute to the development...	Phosphatidylethanolamine N-methyltransferase (...
5	Does vagus nerve contribute to the development...	8-week old Pemt(-/-) and Pemt(+/+) mice were s...
6	Does vagus nerve contribute to the development...	HV abolished the protection against the HFD-in...
7	Does psammaplin A induce Sirtuin 1-dependent a...	Psammaplin A (PsA) is a natural product isolat...
8	Does psammaplin A induce Sirtuin 1-dependent a...	We tested cell proliferation, cell cycle progr...
9	Is methylation of the FGFR2 gene associated wi...	This study examined links between DNA methylat...

准备评估数据集：

现在，我们已经准备好了训练和测试数据集。接下来，让我们为评估准备数据集。对于评估，我们将使用LLM从上下文中挖掘问题，这样我们可以获得一个关于我们的嵌入模型改进效果的现实感受。从PubMedDataset中，我们将选择前250行，将上下文列表合并成每行一个字符串，然后发送给LLM进行问题挖掘。因此，对于每一行，LLM可能会输出大约10个问题。这样，我们将有大约2500个问题-上下文对用于评估。

```
from openai import OpenAI
from tqdm.auto import tqdm

eval_med_data_seed = med_data.shuffle().take(251)

client = OpenAI(api_key="<YOUR_API_KEY>")

prompt = """Your task is to mine questions from the given context.
Example question is also given to you.
You have to create questions and return as pipe separated values(|)

<Context>
{context}
</Context>

<Example>
{example_question}
</Example>
"""

questions = []
for row in tqdm(eval_med_data_seed):

    question = row["question"]
    context = "\n\n".join(row["context"]["contexts"])
    question_count = len(row["context"]["contexts"])
```

```
        message = prompt.format(question_count=question_count,
                                context=context,
                                example_question=question)

        completion = client.chat.completions.create(
            model="gpt-4o",
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {
                    "role": "user",
                    "content": message
                }
            ]
        )

        questions.append(completion.choices[0].message.content.split("|"))

    eval_med_data_seed = eval_med_data_seed.add_column("test_questions", questions)
    df = eval_med_data_seed.to_pandas()

    eval_data = Dataset.from_pandas(df.explode("test_questions"))
    eval_data.to_parquet("test_med_data2.parquet")
```

在我们开始训练之前，我们需要使用上面创建的评估数据集来准备评估器。

准备评估器：

sentence-transformer库提供了各种评估器，如_EmbeddingSimilarityEvaluator_、*BinaryClassificationEvaluator_和_InformationRetrievalEvaluator*。对于我们的特定用例，即训练用于RAG的嵌入模型，_InformationRetrievalEvaluator_是最合适的选择。此外，可以添加多个评估器并用于评分。

给定一组查询和大型语料库集，信息检索评估器将为每个查询检索最相似的top-k个文档。信息检索评估器将使用各种指标来评估模型，如Recall@k、Precision@k、MRR和Accuracy@K，其中k将是1、3、5和10。对于RAG，Recall@K指标是最重要的，因为它表明检索器可以成功检索多少个相关上下文。这一点至关重要，因为如果检索器可以检索到正确的上下文，生成的内容很可能会是准确的，即使我们有额外的非相关上下文。

```
    eval_context_id_map = {}

    for row in eval_data:
        contexts = row["context"]["contexts"]
        for context, context_id in zip(contexts, row["context_ids"]):
            eval_context_id_map[context_id] = context

    eval_corpus = {} # Our corpus (cid => document)
```

```
eval_queries = {}  # Our queries (qid => question)
eval_relevant_docs = {}  # Query ID to relevant documents (qid => set([relevant_cids

for row in eval_data:
    pubid = row.get("pubid")
    eval_queries[pubid] = row.get("test_questions")
    eval_relevant_docs[pubid] = row.get("context_ids")

    for context_id in row.get("context_ids"):
        eval_corpus[context_id] = eval_context_id_map[context_id]
```

_查询：将每个出版物的ID映射到其对应的问题。

_语料库：将每个上下文ID映射到上下文映射中的内容。

_相关文档：将每个出版物的ID关联到一个相关上下文ID的集合中。

在形成所有字典之后，我们可以从sentence_transformer包中创建一个InformationRetrievalEvaluator实例。

```
ir_evaluator = InformationRetrievalEvaluator(
    queries=eval_queries,
    corpus=eval_corpus,
    relevant_docs=eval_relevant_docs,
    name="med-eval-test",
)
```

模型训练：

最后，让我们来训练我们的模型。使用sentence-transformer训练器进行训练非常简单。只需设置以下训练配置参数：

1. eval_steps - 指定模型多久评估一次。
2. save_steps - 指定模型多久保存一次。
3. num_train_epochs - 训练的轮数。
4. per_device_train_batch_size - 在单个GPU的情况下，这是批大小。
5. save_total_limit - 指定允许的最大保存模型数量。
6. run_name - 因为日志将被发布在wandb.ai上，所以运行名称是必要的。

然后，我们将我们的参数、训练数据集、测试数据集、损失函数、评估器和模型名称传递给训练器。现在您可以坐下来放松，直到训练完成。

放松：你是个好人，亚瑟！

对于我们的训练数据，训练模型大约需要3个小时，这包括了测试数据集和评估数据集的推理时间。

```
# Load base model
model = SentenceTransformer("stsb-distilbert-base")
output_dir = f"output/training_mnrl-{datetime.now():%Y-%m-%d_%H-%M-%S}"

train_loss = MultipleNegativesRankingLoss(model=model)

# Training arguments
args = SentenceTransformerTrainingArguments(
    output_dir=output_dir, num_train_epochs=1, per_device_train_batch_size=64,
    eval_strategy="steps", eval_steps=250, save_steps=250, save_total_limit=2,
    logging_steps=100, run_name="mnrl"
)

# Train the model
trainer = SentenceTransformerTrainer(model=model,
                                     args=args,
                                     train_dataset=splitted_dataset["train"],
                                     eval_dataset=splitted_dataset["test"],
                                     loss=train_loss,
                                     evaluator=ir_evaluator)

trainer.train()
```

Step	Training Loss	Validation Loss	Med-eval-dev Cosine Accuracy@1	Med-eval-dev Cosine Accuracy@3	Med-eval-dev Cosine Accuracy@5	Med-eval-dev Cosine Accuracy@10	Med-eval-dev Cosine Precision@1	Med-eval-dev Cosine Precision@3	Med-eval-dev Cosine Precision@5	Med-eval-dev Cosine Precision@10	Med-eval-dev Cosine Recall@1	Med-eval-dev Cosine Recall@3	Med eval- Cosi Reca
250	0.553600	0.104097	0.918000	0.962500	0.974500	0.983500	0.918000	0.679333	0.456600	0.246800	0.318447	0.683064	0.75
500	0.398600	0.076128	0.930500	0.972000	0.984000	0.992500	0.930500	0.710333	0.473700	0.256100	0.322613	0.713482	0.78
750	0.351900	0.068458	0.950000	0.980500	0.989000	0.995000	0.950000	0.729833	0.488100	0.261650	0.329447	0.732704	0.80
1000	0.304800	0.054870	0.945000	0.980500	0.989000	0.994000	0.945000	0.736000	0.494200	0.266150	0.327638	0.738549	0.81
1250	0.281000	0.050285	0.948000	0.983500	0.989500	0.995000	0.948000	0.743500	0.498300	0.267900	0.329005	0.745883	0.82
1500	0.251100	0.045652	0.957500	0.989000	0.994000	0.996500	0.957500	0.761167	0.507500	0.272100	0.331672	0.763434	0.83
1750	0.261400	0.040136	0.960000	0.989000	0.994500	0.996500	0.960000	0.765000	0.510200	0.272300	0.333600	0.767463	0.84
2000	0.229700	0.036455	0.957500	0.989500	0.992000	0.996500	0.957500	0.769667	0.514700	0.274400	0.331255	0.771708	0.85
2250	0.212000	0.034385	0.965500	0.989000	0.994500	0.997000	0.965500	0.780500	0.522700	0.277500	0.334880	0.783247	0.86
2500	0.196100	0.034800	0.962500	0.989000	0.995500	0.998500	0.962500	0.783833	0.523200	0.278650	0.333225	0.786273	0.86
2750	0.203900	0.031889	0.961500	0.990000	0.997000	0.998500	0.961500	0.787000	0.526500	0.278800	0.333880	0.789214	0.86
3000	0.191500	0.029169	0.969500	0.991000	0.997000	0.998000	0.969500	0.791000	0.529900	0.280700	0.335797	0.793251	0.87
3250	0.187700	0.027539	0.966500	0.993500	0.996500	0.997500	0.966500	0.794167	0.533300	0.283100	0.335750	0.796523	0.87
3500	0.169000	0.028117	0.979500	0.995000	0.997000	0.998500	0.979500	0.801167	0.535800	0.282950	0.340505	0.803884	0.88
3750	0.161100	0.026305	0.977500	0.993000	0.997500	0.999000	0.977500	0.804833	0.538400	0.284450	0.339642	0.808109	0.88
4000	0.172900	0.024944	0.974500	0.994000	0.998000	0.999000	0.974500	0.803667	0.535300	0.284400	0.339309	0.806543	0.88
4250	0.154300	0.024085	0.976500	0.995000	0.998000	0.998500	0.976500	0.810333	0.539100	0.286400	0.339434	0.813098	0.88
4500	0.159200	0.021880	0.969500	0.995000	0.997000	0.998500	0.969500	0.808167	0.538200	0.286500	0.336505	0.810802	0.88
4750	0.146300	0.022793	0.976500	0.993500	0.998000	0.998000	0.976500	0.816667	0.543200	0.288100	0.339105	0.818865	0.89
5000	0.152400	0.020878	0.976500	0.995000	0.997000	0.999000	0.976500	0.811833	0.543400	0.287800	0.339355	0.814398	0.89
5250	0.146800	0.021243	0.978500	0.996000	0.998000	0.999000	0.978500	0.817500	0.546500	0.288850	0.339713	0.819678	0.90
5500	0.143800	0.021242	0.983000	0.997000	0.998000	0.999000	0.983000	0.822167	0.549500	0.289350	0.341105	0.824493	0.90
5750	0.134600	0.020660	0.979000	0.996500	0.999000	0.999000	0.979000	0.822833	0.548200	0.289600	0.340088	0.825448	0.90

Full results on the notebook attached at the end

结果

为了进行比较，让我们初始化两个模型的实例，一个带有训练好的权重，另一个带有未训练的权重。

```
untrained_pubmed_model = SentenceTransformer("stsb-distilbert-base")
trained_pubmed_model = SentenceTransformer("/kaggle/input/sentencetransformerpubmedm
```

```
ir_evaluator(untrained_pubmed_model)
ir_evaluator(trained_pubmed_model)
```

结果非常明显，每个指标都有惊人的提升。以下是关注指标的提升情况：

- recall@1 – 相比未训练模型提升了78.80%
- recall@3 – 相比未训练模型提升了137.92%
- recall@5 – 相比未训练模型提升了116.36%
- recall@10 – 相比未训练模型提升了95.09%

分析结果后，很明显，嵌入模型增强了上下文召回率，从而显著提高了RAG生成的整体准确性。然而，一个缺点是需要监控知识库中文档的增加，并定期重新训练模型。

这可以通过遵循标准的机器学习管道流程来实现，其中我们监控模型是否存在任何漂移，如果漂移超过某个阈值，就重新启动训练流程。

参考文献：

1. 领域自适应到专有数据自适应的想法来源于：GPL：用于密集检索无监督领域自适应的生成伪标签[1]
2. RAG评估 - https://www.pinecone.io/learn/series/vector-databases-in-production-for-busy-engineers/rag-evaluation/
3. SBERT训练 - https://sbert.net/examples/training/ms_marco/cross_encoder_README.html

引用链接

[1] GPL：用于密集检索无监督领域自适应的生成伪标签: *https://arxiv.org/pdf/2112.07577*

RAG 5

RAG · 目录

上一篇 · 最佳RAG技术？Anthropic的上下文检索与混合搜索