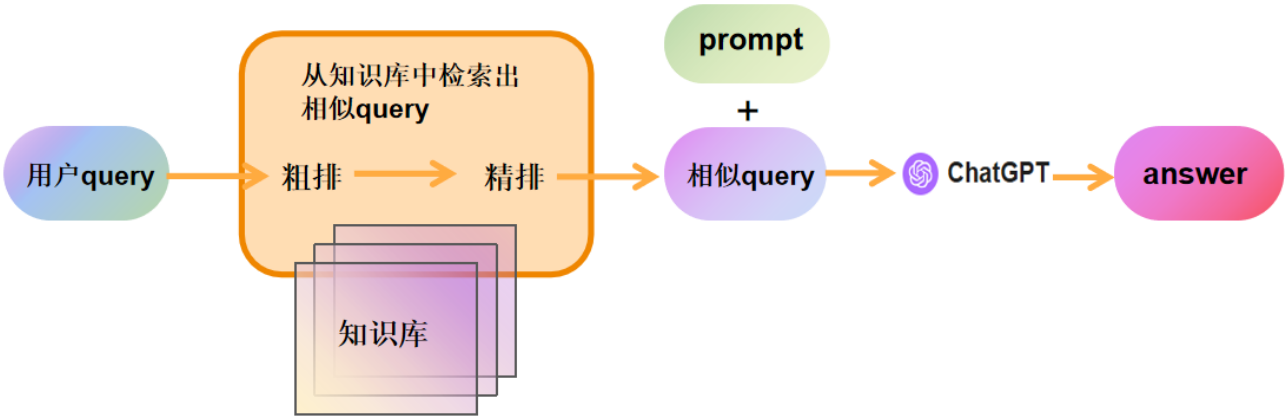
RAG入门

随着大模型的爆火，很多**垂域行业**都开始使用大模型来优化自己的业务，最典型的方法就是RAG（检索增强生成）了。简单来说就是利用检索技术，找出与用户问题相关性最高的段落，再让LLM基于以上段落，去回答用户的提问。是解决大模型落地以及幻觉问题的方法之一。

找到了一个非常适合rag入门的项目，来自CSDN：CSDN问答机器人 作者：ToTensor

CSDN问答机器人_csdn ai问答-CSDN博客

目的：构建一个问答机器人，以自动回答CSDN上用户提出的问题



1 构建知识库

知识库的数据来源:

1. 已采纳的问题: 376140
2. 编程语言的官方手册: 67015
3. 高质量的博客: 4779402

这里的`477w`博客, 是没有做结构化的量, 做完结构化后的量是`12959544`, 将近`1300w`的数据量了。

> 结构化：将博客按内容中的小标题拆分开。

使用PostgresSQL存储知识库:

Column	Type	Collation	Nullable	Default
id	integer		not null	
query	text			
inner_id	integer		not null	
query_vector	vector(384)			
meta	jsonb			'{}'::jsonb
create_at	timestamp without time zone			now()
update_at	timestamp without time zone			now()

```
字段说明:
    id: 博客id
    query: 博客标题
    inner_id: 小标题序号
    query_vector: 目标query的向量，这里是 query+head_title 的向量化后的结果
    meta: 主要用于存储小标题及小标题之间的内容
```

随机取的一条拆分后的博客的数据, 这是其中一个meta字段:

```
{
  "url": "博客链接",
  "tags": "debian,linux,ubuntu,vim,编辑器",
  "content": "xxxxxxxxx",
  "head_title": "安装ctags"
}

字段说明:
    tags: 博客标签
    content: 小标题与小标题之间的内容
    head_title: 小标题
```

`tags`字段存的是博客标签, 好处: 1、加速召回 2、在一定程度上提高召回准确率

原因: 通过传入博客标签, 我们将`query`库从全量数据缩小到单个标签的数据, 数据量减少, 速度当然变快, 准确率也有一定提升。

2 粗排

目的：用一个检索模型，快速过滤掉大量不相关或低相关性的回答候选，大大减少需要进一步处理的数据量。

我们怎么得到这个检索模型呢?

步骤：1、构建训练数据 2、训练模型

这里选择的检索模型是SBert也就是Sentence transformer。

1、构建数据集

训练一个无监督的语义相似度模型(如`SimCSE`), 使用该模型来两两计算相似度, 筛选相似度比较高的数据。

通过`SimCSE`计算相似度后的数据举例:

```
TypeScript生成随机数 jmeter随机数生成 0.89
TypeScript生成随机数 kotlin 生成随机数    0.88
TypeScript生成随机数 Javascript生成随机数 0.88
TypeScript生成随机数 ThreadLocalRandom生成随机数  0.88
TypeScript生成随机数 Java生成随机数SecureRandom   0.86
TypeScript生成随机数 wincc随机数的生成 0.86
```

相似度阈值设定在0.9, 筛选出来的数据:

```
python中分割字符串    python字符串分割
python中分割字符串    oracle分割字符串
python中分割字符串    将String字符串分割
python中分割字符串    boost 分割字符串
python中分割字符串    Arduino分割字符串
python中分割字符串    Linux Shell 分割字符串
```

可以看出, 这些筛选出来的数据, 就是我们所关心的部分文本相同, 但语义完全不同的数据，对这部分数据做人工标注。示例如下:

```
利用python发送qq邮件  使用python发送qq邮件  1
利用python发送qq邮件  使用java发送qq邮件    0
利用python发送qq邮件  用Java发送QQ邮件 0
利用python发送qq邮件  使用python发送邮件    1
利用python发送qq邮件  C#利用QQ信箱发送EMAIL 0
利用python发送qq邮件  使用python发邮件 1
利用python发送qq邮件  用Python发送邮件 1
```

其中，当某个技术词里面包括了另一个词时, 我们认为是相似的, 如:

```
利用python发送qq邮件
python 利用zmail库发送邮件
```

2、训练模型

微调`SBERT`模型了，这里直接贴代码吧，没什么难度，`sentence_transformers`库封装得太好了

```python
class TrainSBert:
    def __init__(self, config, options):
        self.model_name="sentence-transformers/all-MiniLM-L6-v2"
        self.data_path = "自己的标注数据路径"
        self.model = None
        self.model_base_dir = '模型保存base路径'
        self.model_dir = os.path.join(self.model_base_dir, self.model_name.split("/")[-1])
        if not os.path.exists(self.model_dir):
            os.makedirs(self.model_dir)
        self.evaluate_path = os.path.join(self.model_dir, "result.txt")

    def load(self):
        self.model = SentenceTransformer(self.model_name)

    def load_train_data(self):
        file_handle = open(self.data_path, 'r')
        train_data_list = []
        dev_sentences1, dev_sentences2, dev_labels = [], [], []
        count = 0
        for line in file_handle:
            item_list = line.strip().split("\t")
            sa = item_list[0]
            sb = item_list[1]
            label = float(item_list[2])
```

```
            count += 1
            if count <= 5000:
                dn = InputExample(texts=[sa, sb], label=label)
                train_data_list.append(dn)
            else:
                dev_sentences1.append(sa)
                dev_sentences2.append(sb)
                dev_labels.append(label)

        train_dataset = SentencesDataset(train_data_list, self.model)
        train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=32)
        return train_dataloader, dev_sentences1, dev_sentences2, dev_labels

    def train(self):
        self.load()
        train_dataloader, dev_sentences1, dev_sentences2, dev_labels =
self.load_train_data()

        train_loss = losses.CosineSimilarityLoss(self.model)
        evaluator = evaluation.EmbeddingSimilarityEvaluator(dev_sentences1, dev_sentences2,
dev_labels)
        self.model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=10,
warmup_steps=100,
          evaluator=evaluator, evaluation_steps=100, output_path= self.model_dir)
        self.model.evaluate(evaluator, self.evaluate_path)
```

训练完成后, 我们来看看效果:

```
s1: 二叉树的python实现 与 s2: Ribbon实现负载均衡 相似度为: 0.18773505091667175
s1: 二叉树的python实现 与 s2: 使用openFeign实现负载均衡 相似度为: -0.04088197648525238
s1: 二叉树的python实现 与 s2: Nginx负载均衡实现 相似度为: 0.018543850630521774
s1: 二叉树的python实现 与 s2: python 的二叉树实现 相似度为: 0.965272068977356
s1: 二叉树的python实现 与 s2: 请问下二叉树用python怎么实现，求求各位大佬了，小弟实在不会 相似度为:
0.8639361262321472
s1: 二叉树的python实现 与 s2: 二叉树的python实现 相似度为: 1.0
s1: 二叉树的python实现 与 s2: 二叉树的c++实现 相似度为: 0.21337147057056427
```

完美符合预期~

3 精排

目的：在粗排的基础上，对筛选出的相似的query进行更细致的评估。

这里使用的LTR模型是LGBMRanker。

> LTR（Learning to Rank）模型通过学习排序信息（即将相关结果排在不相关结果之前）来优化排序效果。

人工构造特征, 作为`LTR`模型的输入, 在这里, 构造了以下特征:

```
1、SBERT语义相似度
2、最长公共子序列
3、编辑距离
4、jaccard相似度
5、余弦相似度
6、皮尔逊相关性系数
7、欧式距离
8、KL散度
```

训练数据还是使用微调阶段构造的。

训练时的参数如下：

```
params = {
    "boosting_type": "gbdt",
    "max_depth": 5,
    "objective": "binary",
    "num_leaves": 64,
    "learning_rate": 0.05,
    "max_bin": 512,
    "subsample_for_bin": 200,
    "subsample": 0.5,
    "subsample_freq": 5,
    "colsample_bytree": 0.8,
    "reg_alpha": 5,
    "reg_lambda": 10,
    "min_split_gain": 0.5,
    "min_child_weight": 1,
    "min_child_samples": 5,
    "scale_pos_weight": 1,
    "group": "name:groupId",
    "metric": "auc",
}
```

4 prompt

```
假如你是一名资深的IT专家，请你结合以下参考资料和你现有的知识回答以下问题，尽量给出具体的解决方案，请将每一步都以清晰易懂的语言告诉我，请尽可能地展示代码，如果你没有把握解决该问题，只需要回答：我无法解决该问题，请不要试图编造假的答案来忽悠我，答案用markdown格式返回，以下是问题和参考资料：
问题：
{query}

参考资料：
{blog_content}
```

总结

优化的方向：

- 构建知识库的方法更加合理，考虑关联表拆分之类

- 增加`SBERT`微调数据集，可以用GPT之类的辅助生成数据集
- 粗排模型的优化
- 精排模型的优化