Глава 9. Оценка RAG количественно и с использованием визуализаций

Это продолжение перевода книги Кит Борн. Раскрытие потенциала данных с помощью генеративного ИИ и технологии RAG. Оценка играет важную роль в создании и поддержке систем генерации, дополненной поиском (RAG). Во время построения пайплайна оценка помогает выявлять области для улучшения, оптимизировать производительность системы и измерять влияние улучшений. После развертывания системы RAG оценка обеспечивает ее эффективность, надежность и производительность. В этой главе будут рассмотрены следующие темы:

- Оценка на этапе создания приложения RAG
- Оценка приложения RAG после развертывания
- Стандартизированные структуры оценки
- Эталонные данные
- Лаборатория кода 9.1: ragas
- Дополнительные методы оценки для систем RAG

Предыдущая глава   Содержание   Следующая глава

Код для этой главы размещен в репозитории GitHub.

Начнем с обсуждения того, как оценка может помочь на начальных этапах построения системы RAG.

Оценка на этапе создания приложения RAG

Оценка играет важную роль на протяжении всего процесса разработки конвейера RAG. Постоянно оценивая систему в процессе ее создания, вы можете выявлять области, требующие улучшений, оптимизировать производительность системы и измерять влияние модификаций.

Оценка необходима для понимания компромиссов и ограничений различных подходов в рамках конвейера RAG. Конвейеры RAG часто требуют выбора между различными техническими решениями, такими как хранилище векторов, алгоритм поиска и модель генерации текста. Каждый из этих компонентов может существенно влиять на общую производительность системы. Систематическая оценка различных комбинаций этих компонентов позволяет получить ценные инсайты о том, какие подходы дают наилучшие результаты для конкретных задач и доменов.

Например, вы можете экспериментировать с различными моделями эмбеддингов, такими как локальные модели с открытым исходным кодом, которые можно загрузить бесплатно, или API облачных сервисов, которые взимают плату за каждую конвертацию текста в эмбеддинг. Возможно, вам нужно понять, лучше ли облачный сервис бесплатной модели стоит ли это преимущество дополнительных расходов. Аналогичным образом можно оценивать производительность различных моделей генерации текста, таких как ChatGPT, Llama и Claude.

Итеративный процесс оценки помогает принимать обоснованные решения об архитектуре и компонентах конвейера RAG. Учитывая такие факторы, как эффективность, масштабируемость и способность к обобщению, вы можете точно настроить систему для достижения оптимальной производительности, минимизируя вычислительные затраты и обеспечивая ее устойчивость в различных сценариях.

Оценка необходима для понимания компромиссов и ограничений различных подходов в рамках конвейера RAG. Однако она также может быть полезной после развертывания системы, о чем мы поговорим далее.

Оценка приложения RAG после развертывания

После развертывания системы генерации, дополненной поиском (RAG), оценка остается важнейшим аспектом обеспечения ее постоянной эффективности, надежности и производительности. Непрерывный мониторинг и анализ развернутого конвейера RAG необходимы для поддержания его качества и выявления возможных проблем или ухудшений со временем.

Существует множество причин, по которым производительность системы RAG может снижаться после развертывания. Например, данные, используемые для поиска, могут устареть или стать нерелевантными по мере появления новой информации. Модель генерации текста может испытывать трудности с адаптацией к меняющимся пользовательским запросам или изменениям в

целевой области. Кроме того, инфраструктура, лежащая в основе системы, такая как аппаратные или программные компоненты, может столкнуться с проблемами производительности или сбоями.

Представьте ситуацию, когда вы работаете в компании, занимающейся управлением финансовыми активами, и у вас есть приложение на базе RAG, которое помогает пользователям понять наиболее вероятные факторы, влияющие на их финансовый портфель. Источники данных могут включать все анализы, опубликованные крупными финансовыми фирмами за последние пять лет и охватывающие все финансовые активы, представленные вашими клиентами.

Однако на финансовых рынках крупные (макроэкономические) события могут значительно повлиять на портфели, что не было учтено в данных за последние пять лет. Крупные катастрофы, политическая нестабильность или даже региональные события, связанные с отдельными акциями, могут кардинально изменить их динамику. Для вашего приложения RAG это означает изменения в ценности данных, предоставляемых конечному пользователю. Со временем эта ценность может быстро снизиться без надлежащих обновлений. Пользователи могут начать задавать вопросы о конкретных событиях, с которыми приложение RAG не сможет справиться, например: «Какое влияние ураган 5-й категории, который только что произошел, окажет на мой портфель в следующем году?» Однако при регулярных обновлениях и мониторинге, особенно с учетом более свежих отчетов о последствиях урагана, эти проблемы, скорее всего, будут успешно решены.

Для минимизации таких рисков крайне важно постоянно мониторить систему RAG, особенно в общих точках отказа. Постоянно оценивая эти критически важные компоненты конвейера RAG, вы сможете проактивно выявлять и устранять любые ухудшения производительности. Это может включать обновление корпуса поиска актуальными данными, тонкую настройку модели генерации текста на новых данных или оптимизацию инфраструктуры системы для обработки увеличенной нагрузки или устранения узких мест в производительности. Кроме того, важно установить механизм обратной связи, позволяющий пользователям сообщать о любых проблемах или предлагать улучшения.

Активно собирая и учитывая отзывы пользователей, вы сможете непрерывно совершенствовать систему RAG, чтобы она лучше соответствовала их потребностям. Это также может включать мониторинг использование интерфейса, время отклика, релевантность и полезность сгенерированных ответов с точки зрения пользователя. Проведение опросов пользователей, анализ логов взаимодействия и отслеживание показателей удовлетворенности пользователей могут предоставить ценную информацию о том, насколько хорошо система RAG выполняет свою задачу.

То, как вы используете эту информацию, во многом зависит от типа приложения RAG, но в целом это наиболее распространенные области, за которыми следят для обеспечения непрерывного улучшения приложений RAG.

Оценка помогает стать лучше

Почему оценка так важна? Если вы не измеряете состояние системы, будет сложно понять, что именно улучшило (или ухудшило) производительность системы RAG. Без объективной точки сравнения трудно понять, что пошло не так, когда возникают проблемы. Это мог быть механизм поиска? Промт? Ответы большой языковой модели (LLM)? Эти вопросы поможет прояснить качественная система оценки.

Оценка обеспечивает систематический и объективный способ измерения производительности конвейера, выявления областей для улучшения и отслеживания влияния любых изменений или оптимизаций, которые вы вносите. Без надежной структуры оценки становится сложно понять, как развивается система RAG и в каких аспектах она требует доработки.

Внедряя оценку как неотъемлемую часть процесса разработки, вы сможете непрерывно совершенствовать и оптимизировать конвейер RAG. На ранних этапах разработки системы RAG вы уже начинаете принимать решения о том, какие технические компоненты будете рассматривать. На этом этапе у вас еще нет установленных компонентов, поэтому вы не можете оценивать код, но вы все же можете использовать стандартизированные структуры оценки, чтобы сузить круг рассматриваемых вариантов. Давайте обсудим эти стандартизированные структуры оценки для наиболее распространенных элементов систем RAG.

Стандартизированные структуры оценки

Технические компоненты системы RAG включают модель для создания эмбеддингов, векторное хранилище, векторный поиск и большую языковую модель (LLM). При рассмотрении различных вариантов для каждого из этих компонентов доступен ряд стандартизированных метрик, которые помогают сравнивать их друг с другом. Вот некоторые из распространенных метрик для каждой категории.

Бенчмарки моделей эмбеддингов

Масштабный бенчмарк текстовых эмбеддингов (Massive Text Embedding Benchmark, MTEB) и рейтинги MTEB для задач поиска оценивают производительность моделей эмбеддингов на различных задачах поиска по разным наборам данных. Рейтинги MTEB ранжирует модели по их средней производительности в рамках множества задач, связанных с эмбеддингами и поиском. Рейтинги доступен по ссылке.

При посещении этой страницы вы можете выбрать вкладки Retrieval и Retrieval w/Instructions для просмотра рейтингов моделей эмбеддингов, ориентированных на задачи поиска. Чтобы оценить каждую из моделей, представленных в Рейтинге, их выдача тестируются на различных наборах данных, охватывающих широкий круг областей, таких как:

- Поиск аргументов (ArguAna)
- Поиск фактов о климате (ClimateFEVER)
- Поиск дублирующих вопросов (CQADupstackRetrieval)
- Поиск сущностей (DBPedia)
- Извлечение фактов и их проверка (FEVER)
- Финансовые вопросы-ответы (FiQA2018)
- Многошаговые вопросы-ответы (HotpotQA)
- Ранжирование документов и отрывков текста (MSMARCO)
- Проверка фактов (NFCorpus)
- Вопросы-ответы в открытой области знаний (NQ)
- Обнаружение дублирующих вопросов (QuoraRetrieval)
- Поиск научных документов (SCIDOCS)
- Проверка научных утверждений (SciFact)
- Поиск аргументов (Touche2020)
- Поиск информации, связанной с COVID-19 (TRECCOVID)

Рейтинги ранжирует модели эмбеддингов по их средней производительности на этих задачах, предоставляя всесторонний обзор их сильных и слабых сторон. Вы также можете нажать на любую метрику, чтобы упорядочить модели по этой метрике. Например, если вас интересует метрика, больше ориентированная на финансовые вопросы-ответы, обратите внимание на модель, получившую наивысшую оценку по набору данных FiQA2018.

Бенчмарки для векторных хранилищ и векторного поиска

ANN-Benchmarks — инструмент для бенчмаркинга, оценивающий производительность алгоритмов поиска приближенных ближайших соседей (Approximate Nearest Neighbor, ANN), которые подробно обсуждались в главе 8. ANN-Benchmarks измеряет точность поиска, скорость и использование памяти различными инструментами векторного поиска. Среди протестированных инструментов — упомянутые ранее Facebook AI Similarity Search (FAISS), Approximate Nearest Neighbors Oh Yeah (ANNOY) и Hierarchical Navigable Small Worlds (HNSW).

Benchmarking IR (BEIR) — еще один полезный ресурс для оценки векторных хранилищ и алгоритмов поиска. BEIR предлагает разнородный бенчмарк для нулевого обучения (zero-shot) моделей информационного поиска в различных областях, включая вопросы-ответы, проверку фактов и поиск сущностей. Нулевое обучение (zero-shot), как будет подробнее обсуждено в главе 13, означает обработку вопросов или пользовательских запросов без предварительных примеров, что является частой ситуацией в RAG. BEIR предоставляет стандартизированную структуру оценки и включает популярные наборы данных, такие как:

- MSMARCO: крупномасштабный набор данных, созданный на основе реальных запросов и ответов, предназначенный для оценки моделей глубокого обучения в задачах поиска и вопросов-ответов.

- HotpotQA: набор данных вопросов-ответов с естественными многошаговыми вопросами, обеспечивающий сильную поддержку фактов и позволяющий создавать более объяснимые системы вопросов-ответов.
- CQADupStack: бенчмарк для исследований в области вопросов-ответов в сообществах (community question-answering, cQA), взятый из 12 подфорумов Stack Exchange и аннотированный информацией о дублирующих вопросах.

Эти и другие наборы данных, включенные в бенчмарк BEIR, охватывают широкий спектр областей и задач информационного поиска, что позволяет оценить производительность вашей системы поиска в разных контекстах и сравнить ее с современными методами.

Бенчмарки для больших языковых моделей (LLM)

Artificial Analysis LLM Performance Leaderboard — это обширный ресурс для оценки как открытых, так и проприетарных языковых моделей, таких как ChatGPT, Claude и Llama. Он измеряет их производительность в широком спектре задач. Для качественного сравнения используются несколько суб-рейтингов:

- Общие способности: Chatbot Arena.
- Логика и знания: Massive Multitask Language Understanding (MMLU) и Multi-turn Benchmark (MT Bench).

Также учитываются скорость работы и стоимость, что позволяет анализировать баланс между этими параметрами. Рейтинг предоставляет целостное представление о возможностях моделей, ранжируя их на основе производительности в различных задачах. Доступ к нему можно получить по ссылке: https://artificialanalysis.ai/.

Помимо общего, существуют специализированные рейтинги, сосредоточенные на определенных аспектах производительности LLM. Например, Artificial Analysis LLM Performance Leaderboard оценивает такие технические характеристики моделей, как скорость вывода, использование памяти и масштабируемость. Среди используемых метрик:

- Пропускная способность (количество токенов, обрабатываемых в секунду).
- Задержка (время на генерацию ответа).
- Объем памяти (memory footprint).
- Эффективность масштабирования.

Эти показатели помогают понять вычислительные требования и характеристики производительности различных моделей LLM.

Open LLM Leaderboard отслеживает производительность открытых языковых моделей в задачах понимания и генерации текста. Среди используемых бенчмарков:

- AI2 Reasoning Challenge (ARC) для сложных научных рассуждений.
- HellaSwag для рассуждений на основе здравого смысла.
- MMLU для оценки производительности в специфичных доменах.
- TruthfulQA для генерации правдивых и информативных ответов.
- WinoGrande для рассуждений о здравом смысле через неоднозначности в местоимениях.
- Grade School Math 8K (GSM8K) для способностей в математических рассуждениях.

Использование стандартизированных оценок и контрольных наборов данных предоставляет ценный начальный пункт для сравнения производительности компонентов вашего конвейера RAG. Учитывая результаты таких оценок наряду с другими факторами, такими как вычислительная эффективность и простота интеграции, вы можете сузить выбор и принять более обоснованные решения в отношении компонент приложения RAG.

Однако важно отметить, что, несмотря на полезность этих стандартизированных метрик для начального выбора компонентов, они могут не полностью отражать производительность вашего конвейера RAG с его уникальными входными и выходными данными. Чтобы действительно понять, насколько хорошо ваша система RAG работает в вашем случае, необходимо создать собственную систему оценки.

Далее следует обсудить один из самых важных и часто упускаемых аспектов оценки RAG — ваши эталонные данные.

Эталонные данные

Это данные, которые представляют собой идеальные ответы, ожидаемые в случае, если ваша система генерации, дополненной поиском (RAG), работает на пике производительности. В качестве практического примера можно рассмотреть систему RAG, позволяющую задавать вопросы о последних исследованиях в области ветеринарной онкологии для собак, при этом источником данных служат последние научные статьи по этой теме, размещенные на PubMed. В данном случае эталонными данными будут вопросы и ответы, которые можно задать и получить на основе этих данных. Эти вопросы должны быть реалистичными и соответствовать интересам вашей целевой аудитории, а ответы — представлять собой идеальные результаты, ожидаемые от языковой модели. Несмотря на то, что такая оценка может быть частично субъективной, наличие набора эталонных данных для сравнения входных и выходных данных системы RAG является критически важным шагом для анализа влияния внесенных изменений и повышения эффективности системы.

Как использовать эталонные данные?

Эталонные данные служат контрольной точкой для измерения производительности систем RAG. Сравнивая ответы, сгенерированные системой RAG, с эталонными данными, можно оценить, насколько хорошо система извлекает релевантную информацию и формирует точные и последовательные ответы. Эталонные данные помогают количественно оценить эффективность различных подходов в RAG и выявить области, требующие улучшений.

Создание эталонных данных

Создание эталонных данных вручную может быть трудоемким процессом. Если у вашей компании уже есть набор данных с идеальными ответами на конкретные запросы или задания, это может стать ценным ресурсом. Однако если таких данных нет, существуют альтернативные методы их получения.

Аннотирование вручную

Можно нанять аннотаторов, которые вручную создадут идеальные ответы на набор запросов или заданий. Этот подход обеспечивает высокое качество эталонных данных, но может быть затратным и требовать значительного времени, особенно для масштабных оценок.

Экспертные знания

В некоторых областях у вас может быть доступ к экспертам, которые способны предоставлять эталонные ответы, основываясь на своем опыте. Это особенно полезно в специализированных или технических областях, где важна точная информация.

Один из подходов, который может облегчить этот процесс, — использование правил для генерации. В рамках такого подхода для определенных областей или задач можно задавать набор правил или шаблонов для создания синтетических эталонных данных, а затем привлекать экспертов для заполнения этих шаблонов. Используя знания в предметной области и заранее определенные паттерны, можно создавать ответы, соответствующие ожидаемому формату и содержанию.

Например, если вы разрабатываете чат-бота для поддержки клиентов, связанных с мобильными телефонами, шаблон может выглядеть так: «Для решения [проблема] вы можете попробовать [решение]». Эксперты могут заполнить различные комбинации проблем и решений, где, например, проблема — это «быстрая разрядка батареи», а решение — «уменьшение яркости экрана и закрытие фоновых приложений». После заполнения шаблон генерирует итоговый ответ, например: «Для решения проблемы [быстрая разрядка батареи] вы можете попробовать [уменьшение яркости экрана и закрытие фоновых приложений]».

Краудсорсинг

Платформы, такие как Amazon Mechanical Turk и Figure Eight, позволяют распределить задачу создания эталонных данных среди большого числа работников. При условии предоставления четких инструкций и механизмов контроля качества можно получить разнообразный набор ответов.

Синтетические эталонные данные

В случаях, когда получение реальных эталонных данных затруднено или невозможно, генерация синтетических эталонных данных может стать жизнеспособной альтернативой. Этот процесс включает использование существующих языковых моделей или других техник для автоматической генерации правдоподобных ответов.

Некоторые подходы:

- Модели, дообученные на специальных данных. Вы можете дообучить языковую модель на небольшом наборе качественных ответов. Обучив модель на примерах идеальных ответов, она сможет генерировать аналогичные ответы для новых запросов. Сгенерированные таким образом ответы могут использоваться как синтетические эталонные данные.
- Методы на основе поиска. Если у вас есть большой корпус высококачественного текстового материала, вы можете использовать методы поиска для нахождения релевантных фрагментов или предложений, которые наиболее точно соответствуют запросу или заданию. Эти найденные фрагменты могут служить заменой эталонных данных.

Получение эталонных данных может быть сложным этапом в создании системы RAG, но как только эти данные будут получены, они станут прочной основой для эффективной оценки RAG. В следующем разделе мы рассмотрим практический пример, в котором генерируем синтетические эталонные данные, а затем интегрируем платформу оценки в систему RAG, чтобы оценить, как гибридный поиск, использованный в предыдущей главе, повлиял на результаты.

Лаборатория кода 9.1: ragas

Оценка генерации, дополненной поиском (Retrieval-Augmented Generation Assessment, ragas) — это платформа оценки, разработанная специально для RAG. В этом практикуме мы реализуем ragas в коде, создадим синтетические эталонные данные и установим набор метрик, которые можно интегрировать в систему RAG. Но системы оценки существуют для того, чтобы что-то оценивать, не так ли? Что именно мы будем оценивать в этом практикуме?

В главе 8 мы представили гибридный поиск для этапа извлечения. Сначала мы реализуем метод поиска на основе плотных векторных представлений (semantic-based search) и затем используем ragas для оценки гибридного поиска.

Прежде чем углубляться в использование ragas, важно отметить, что это проект, находящийся в активной разработке. Часто выпускаются новые функции и изменения в API, поэтому обратитесь к документации на официальном сайте, когда будете изучать примеры кода.

Этот практикум мы начнем с места, на котором остановились в предыдущей главе, где добавили EnsembleRetriever из LangChain. Начнем с установки новых пакетов:

```
%pip install ragas==0.1.20
%pip install tqdm==4.66.5 -q --user
%pip install matplotlib==3.9.2
```

Мы устанавливаем *ragas* (предмет практикума), *tqdm*, используемый в ragas (популярная библиотека Python для создания индикаторов прогресса и отображения информации о процессе выполнения итеративных задач), *matplotlib* (библиотека, широко используется для построения графиков в Python, мы будем использовать ее для визуализации результатов оценки).

Далее необходимо добавить несколько импортов, связанных с установленными пакетами:

```
import tqdm as notebook_tqdm
import pandas as pd
import matplotlib.pyplot as plt
from datasets import Dataset
from ragas import evaluate
from ragas.testset.generator import TestsetGenerator
from ragas.testset.evolutions import simple, reasoning, multi_context
from ragas.metrics import (
   answer_relevancy,
   faithfulness,
   context_recall,
   context_precision,
   answer_correctness,
   answer_similarity
)
```

Итак, tqdm обеспечит платформу ragas возможностью отображать индикаторы прогресса во время выполнения ресурсоемких процессов. Мы будем использовать популярную библиотеку pandas для анализа данных, загружая их в DataFrame как часть анализа. Импорт matplotlib.pyplot даст нам возможность добавлять визуализации (графики). Также мы импортируем Dataset. Эта библиотека, разработанная и поддерживаемая Hugging Face, предоставляет интерфейс для доступа и обработки данных в задачах, связанных с естественным языком (NLP).

Наконец, мы импортируем несколько пакетов из библиотеки ragas. Рассмотрим их подробнее.

from ragas import evaluate: функция evaluate принимает набор данных в формате ragas, а также опциональные параметры (метрики, языковые модели, эмбеддинги и другие конфигурации), и выполняет оценку конвейера RAG. Функция evaluate возвращает объект Result, содержащий оценки по каждой метрике, что позволяет анализировать производительность конвейеров RAG.

from ragas.testset.generator import TestsetGenerator: класс TestsetGenerator используется для генерации синтетических эталонных наборов данных для оценки конвейеров RAG. Он принимает набор документов и генерирует пары *вопрос-ответ* вместе с соответствующими контекстами. Важная особенность TestsetGenerator — возможность настройки распределения тестовых данных, указывая пропорции различных типов вопросов (например, простых, многоконтекстных или требующих рассуждений) с помощью параметра distributions. Он поддерживает генерацию тестовых наборов с использованием загрузчиков документов LangChain и LlamaIndex.

from ragas.testset.evolutions import simple, reasoning, multi_context: эти импорты представляют различные типы эволюции вопросов, используемые в процессе генерации тестового набора данных. Они помогают создать разнообразный и комплексный тестовый набор, охватывающий различные типы вопросов, встречающиеся в реальных сценариях:

- Simple evolution: генерирует простые вопросы на основе предоставленных документов.
- Reasoning evolution: создает вопросы, требующие навыков рассуждения.
- Multi_context evolution: генерирует вопросы, для ответа на которые требуется информация из нескольких связанных разделов или фрагментов.

from ragas.metrics import (метрики): этот импорт добавляет различные метрики оценки, предоставляемые библиотекой ragas. Дополнительные метрики будут рассмотрены в конце практикума. Эти метрики оценивают различные аспекты производительности конвейера RAG, связанные с извлечением и генерацией, а также со всеми этапами конвейера в целом.

Настройка языковых моделей (LLM) и моделей эмбеддингов

Теперь мы усовершенствуем способ работы с нашими языковыми моделями и сервисами эмбеддингов. С появлением ragas мы добавляем больше сложности в количество используемых LLM и хотим лучше управлять ими, заранее задав инициализацию для обеих категорий: сервиса эмбеддингов и сервисов LLM. Вот пример кода:

```
# LLMs/Embeddings
embedding_ada = "text-embedding-ada-002"
model_gpt35="gpt-3.5-turbo"
model_gpt4="gpt-4o-mini"

embedding_function = OpenAIEmbeddings(model=embedding_ada, openai_api_key=openai.api_key)
llm = ChatOpenAI(model=model_gpt35, openai_api_key=openai.api_key, temperature=0.0)
generator_llm = ChatOpenAI(model=model_gpt35, openai_api_key=openai.api_key, temperature=0.0)
critic_llm = ChatOpenAI(model=model_gpt4, openai_api_key=openai.api_key, temperature=0.0)
```

Обратите внимание, что, хотя мы по-прежнему используем только один сервис эмбеддингов, теперь у нас есть две LLM для вызова. Основная цель состоит в том, чтобы определить главную LLM, которую мы будем использовать для генерации (переменная llm), и две дополнительные LLM, назначенные для процесса оценки (generator_llm и critic_llm).

У нас есть преимущество использования более продвинутой модели — ChatGPT-4o-mini, которую мы назначим в качестве критической LLM (critic_llm). Теоретически это означает, что она может быть более эффективной при оценке входных данных, которые мы ей передаем. Это может быть не всегда так, или вы можете использовать LLM, специально дообученную для задачи оценки. В любом случае,

разделение LLM на специализированные назначения демонстрирует, как разные модели могут быть использованы для различных целей в системе RAG.

Теперь удалим строку кода, где ранее инициализировался объект LLM:

```
llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)
```

Далее добавим новый конвейер RAG для выполнения поиска по схожести (аналог того, что мы изначально выполняли с плотными эмбеддингами):

```
rag_chain_similarity = RunnableParallel(
    {"context": dense_retriever,
     "question": RunnablePassthrough()
}).assign(answer=rag_chain_from_docs)
```

Чтобы сделать вещи более понятными, мы обновим название гибридного конвейера DRAG следующим образом:

```
rag_chain_hybrid = RunnableParallel(
    {"context": ensemble_retriever,
     "question": RunnablePassthrough()
}).assign(answer=rag_chain_from_docs)
```

Обратите внимание на изменение имени переменной: раньше она называлась rag_chain_with_source, а теперь — rag_chain_hybrid, что подчеркивает аспект гибридного поиска.

Теперь обновим исходный код для отправки пользовательского запроса. На этот раз мы будем использовать как версию поиска по схожести, так и гибридного поиска.

Создание версии для поиска по схожести:

```
# Question - Submitted to the similarity / dense vector search
user_query = "What are Google's environmental initiatives?"
result = rag_chain_similarity.invoke(user_query)
retrieved_docs = result['context']

print(f"Original Question to Similarity Search: {user_query}\n")
print(f"Relevance Score: {result['answer']['relevance_score']}\n")
print(f"Final Answer:\n{result['answer']['final_answer']}\n\n")
print("Retrieved Documents:")
for i, doc in enumerate(retrieved_docs, start=1):
    print(f"Document {i}: Document ID: {doc.metadata['id']} source: {doc.metadata['source']}")
    print(f"Content:\n{doc.page_content}\n")
```

Создание версии для гибридного поиска:

```
# Question - Submitted to the hybrid / multi-vector search
user_query = "What are Google's environmental initiatives?"
result = rag_chain_hybrid.invoke(user_query)
retrieved_docs = result['context']

print(f"Original Question to Dense Search: {user_query}\n")
print(f"Relevance Score: {result['answer']['relevance_score']}\n")
print(f"Final Answer:\n{result['answer']['final_answer']}\n\n")
print("Retrieved Documents:")
for i, doc in enumerate(retrieved_docs, start=1):
    print(f"Document {i}: Document ID: {doc.metadata['id']} source: {doc.metadata['source']}")
    print(f"Content:\n{doc.page_content}\n")
```

Основное различие между этими двумя наборами кода заключается в использовании разных конвейеров RAG: rag_chain_similarity и rag_chain_hybrid.

Результаты

Сначала посмотрим на вывод поиска по схожести:

> Google's environmental initiatives include empowering individuals to take action, working together with partners and customers, operating sustainably, achieving net-zero carbon emissions, water stewardship, and promoting a circular economy. They have implemented sustainability features in products like Google Maps, Google Nest thermostats, and Google Flights to help individuals make more sustainable choices. Google also supports various environmental organizations and initiatives, such as the iMasons Climate Accord, ReFED, and The Nature Conservancy, to accelerate climate action and address environmental challenges. Additionally, Google is involved in public policy advocacy and is committed to reducing its environmental impact through its operations and value chain.

Затем — вывод гибридного поиска:

> Google's environmental initiatives include empowering individuals to take action, working together with partners and customers, operating sustainably, achieving net-zero carbon emissions, focusing on water stewardship, promoting a circular economy, engaging with suppliers to reduce energy consumption and greenhouse gas emissions, and reporting environmental data. They also support public policy and advocacy for low-carbon economies, participate in initiatives like the iMasons Climate Accord and ReFED, and support projects with organizations like The Nature Conservancy. Additionally, Google is involved in initiatives with the World Business Council for Sustainable Development and the World Resources Institute to improve well-being for people and the planet. They are also working on using technology and platforms to organize information about the planet and make it actionable to help partners and customers create a positive impact.

Определение того, какой результат лучше, может быть субъективным. Однако если вы вернетесь к коду из главы 8, то увидите, что механизм извлечения для каждого из этих конвейеров возвращает разные наборы данных для использования LLM при формировании ответа на пользовательский запрос. Эти различия отражаются в приведенных ответах: каждый из них содержит немного другую информацию и подчеркивает разные аспекты этих данных.

На данный момент мы настроили систему RAG так, чтобы она могла использовать два разных конвейера RAG: один сосредоточен на использовании только поиска по схожести/плотных эмбеддингов, а другой — на гибридном поиске. Это создает основу для применения ragas в нашем коде, чтобы установить более объективный подход к оценке результатов, которые мы получаем с помощью каждого из этих конвейеров.

Генерация синтетических эталонных данных

Поскольку у нас нет готовых эталонных данных, мы используем библиотеку ragas для генерации синтетических данных.

Библиотека ragas активно использует API LLM. Анализ, который предоставляет ragas, основан на оценке с помощью LLM, что означает, что каждый раз, когда создается или оценивается эталонный пример, вызывается LLM (иногда многократно для одной метрики), что приводит к расходам на API. Если вы генерируете 100 эталонных примеров, включая вопросы и ответы, а затем запускаете шесть различных метрик оценки, число вызовов API LLM существенно увеличивается, доходя до тысяч. Рекомендуется использовать этот инструмент умеренно, пока вы не получите четкое представление о частоте вызовов. Эти вызовы API могут привести к значительным расходам! На момент написания этой главы мои расходы составляли около $2,5 за полный запуск практикума с 10 примерами и 6 метриками. Если у вас больший набор данных или параметр test_size настроен на генерацию более 10 примеров, расходы значительно возрастут.

Мы начинаем с создания экземпляра генератора, который будем использовать для генерации эталонного набора данных:

```
# generator with openai models
generator = TestsetGenerator.from_langchain(
    generator_llm,
    critic_llm,
    embedding_function
)
# Create a list of Document objects from the chunks
```

```
documents = [Document(page_content=chunk) for chunk in splits]
#### FOR FOLLOWING CODE: Uncomment and run once to generate source for test dataset! ####
# generate testset -
testset = generator.generate_with_langchain_docs(
    documents,
    test_size=10,
    distributions={
        simple: 0.5,
        reasoning: 0.25,
        multi_context: 0.25
    }
)
# comparison dataframe
testset_df = testset.to_pandas()
# save dataframes to CSV files in the specified directory
testset_df.to_csv(os.path.join('testset_data.csv'), index=False)
print("testset DataFrame saved successfully in the local directory.")
```

В коде мы используем как generator_llm, так и critic_llm, а также embedding_function. Будьте осторожны! Это три разных API, которые могут привести к значительным расходам. В коде мы также берем наши ранее созданные сплиты данных и подготавливаем их для более эффективной работы с ragas. Каждый фрагмент в splits считается строкой, представляющей часть документа. Класс Document из библиотеки LangChain предоставляет удобный способ представления документа с его содержимым.

Функция testset использует метод generator_with_langchain_docs объекта генератора для создания синтетического теста. Эта функция принимает список документов в качестве входных данных. Параметр test_size задает желаемое количество генерируемых тестовых вопросов (в данном случае 10). Параметр distributions определяет распределение типов вопросов: 50% составляют простые вопросы, 25% — вопросы на логическое рассуждение, а оставшиеся 25% — вопросы с использованием нескольких контекстов. Затем мы преобразуем testset в DataFrame библиотеки pandas, который можно использовать для просмотра результатов, и сохраняем его в виде файла. Учитывая вышеупомянутые расходы, сохранение данных на этом этапе в CSV-файл, который будет храниться в вашей файловой системе, дает дополнительное удобство: вы можете выполнить этот код только один раз!

Теперь загрузим сохраненный набор данных и посмотрим на него:

```
# pull data from saved testset, rather than generating above
### load dataframs from CSV file
saved_testset_df = pd.read_csv(os.path.join('testset_data.csv'))
print("testset DataFrame loaded successfully from local directory.")
saved_testset_df.head(5)
```

Вывод должен выглядеть примерно так:

	question	contexts	ground_truth	evolution_type	metadata	episode_done
0	How does Google Flights provide carbon emissio...	['When individuals search in Google Flights, t...	Google Flights provides carbon emissions estim...	simple	[{}]	True
1	How is Google supporting the Nature Conservanc...	['iMasons Climate AccordGoogle is a founding m...	Google supported three of the Nature Conservan...	simple	[{}]	True
2	How has Google demonstrated its support for st...	['We've consistently supported strong climate ...	Google has consistently supported strong clima...	simple	[{}]	True
3	What innovative water stewardship solutions ar...	['multiple offices around the world, we've ach...	The new Bay View campus, opened in 2022, is on...	simple	[{}]	True
4	How much water have Google's contracted waters...	['In addition to focusing on responsible water...	271 million gallons of water	simple	[{}]	True

Рис. 9.1. DataFrame, показывающий синтезированные эталонные данные

В этом наборе данных вы видите вопросы и ответы (ground_truth), которые были сгенерированы экземпляром generator_llm, созданным ранее. Теперь у вас есть эталонные данные! LLM попытается создать 10 различных пар "вопрос-ответ" для эталона, но в некоторых случаях могут возникнуть ошибки, которые ограничат генерацию. Это приведет к меньшему количеству эталонных примеров,

чем было задано в переменной test_size. В данном случае удалось сгенерировать 7 примеров вместо 10. Для более тщательной проверки системы RAG вам, вероятно, потребуется больше 10 примеров. Однако в этом простом примере мы примем 7 примеров, чтобы снизить расходы на API.

Далее подготовим набор данных для проверки сходства:

```
# Convert the DataFrame to a dictionary
saved_testing_data = saved_testset_df.astype(str).to_dict(orient='list')
# Create the testing_dataset
saved_testing_dataset = Dataset.from_dict(saved_testing_data)
# Update the testing_dataset to include only these columns -
# "question", "ground_truth", "answer", "contexts"
saved_testing_dataset_sm = saved_testing_dataset.remove_columns(["evolution_type", "episode_done"])
```

Здесь мы выполняем дополнительное преобразование данных для обеспечения совместимости форматов с другими частями кода (в данном случае для ввода в ragas). Мы преобразуем DataFrame saved_testset_df в формат словаря с помощью метода to_dict() с параметром orient='list', предварительно преобразовав все столбцы в строки методом astype(str). Полученный словарь saved_testing_data используется для создания объекта Dataset под названием saved_testing_dataset с помощью метода from_dict() из библиотеки datasets. Затем создается новый набор данных под названием saved_testing_dataset_sm, представляющий небольшую часть данных, содержащую только нужные столбцы.

Мы удаляем столбцы evolution_type и episode_done с помощью метода remove_columns(). Давайте посмотрим на это, добавив код в отдельную ячейку:

```
saved_testing_dataset_sm
```

Результат:

```
Dataset({
   features: ['question', 'contexts', 'ground_truth', 'metadata'],
   num_rows: 10
})
```

Если у вас больше эталонных примеров, переменная num_rows будет это отражать, но остальная часть останется такой же. Объект Dataset показывает *features*, представляющие переданные столбцы, а также указывает, что у нас есть семь строк данных.

Далее мы настроим функцию для запуска цепочек RAG, которые мы передаем ей, а затем добавим форматирование, позволяющее работать с ragas:

```
# Function to generate answers using the RAG chain
def generate_answer(question, ground_truth, rag_chain):
   result = rag_chain.invoke(question)
   return {
      "question": question,
      "answer": result["answer"]["final_answer"],
      "contexts": [doc.page_content for doc in result["context"]],
      "ground_truth": ground_truth
   }
```

Этот блок кода определяет функцию generate_answer(), которая принимает вопрос, эталонные данные и rag_chain в качестве входных данных. Функция гибкая, так как принимает любую из цепочек, которые мы ей передаем, что будет полезно при анализе как цепочек на основе сходства, так и гибридных цепочек. Сначала функция вызывает rag_chain с вопросом. Далее возвращает вопрос, итоговый ответ, контекст и эталонные данные.

Теперь мы подготовим наборы данных для работы с ragas:

```
testing_dataset_similarity = saved_testing_dataset_sm.map(
   lambda x: generate_answer(x["question"],
   x["ground_truth"], rag_chain_similarity),
```

```
  remove_columns=saved_testing_dataset_sm.column_names
)
testing_dataset_hybrid = saved_testing_dataset_sm.map(
  lambda x: generate_answer(x["question"],
  x["ground_truth"], rag_chain_hybrid),
  remove_columns=saved_testing_dataset_sm.column_names
)
```

Здесь мы создали два новых набора данных, testing_dataset_similarity и testing_dataset_hybrid, применяя функцию generate_answer() к каждой строке saved_testing_dataset_sm для каждой из наших цепочек RAG (цепочка сходства и гибридная цепочка) с помощью метода map(). В качестве аргумента rag_chain используются rag_chain_similarity и rag_chain_hybrid для соответствующего наборов данных. Первоначальные столбцы saved_testing_dataset_sm удаляются с помощью remove_columns=saved_testing_dataset_sm.column_names.

И, наконец, запускаем ragas на двух наборах данных. Вот код для применения ragas к нашей цепочке RAG на основе сходства:

```
# Similarity search score
score_similarity = evaluate(
  testing_dataset_similarity,
  metrics=[
    faithfulness,
    answer_relevancy,
    context_precision,
    context_recall,
    answer_correctness,
    answer_similarity
  ]
)
similarity_df = score_similarity.to_pandas()
similarity_df
```

Здесь мы применяем ragas для оценки testing_dataset_similarity с использованием функции evaluate() из библиотеки ragas. Оценка выполняется с использованием указанных метрик, включая достоверность (faithfulness), релевантность ответа (answer_relevancy), точность контекста (context_precision), полноту контекста (context_recall), корректность ответа (answer_correctness) и сходство ответа (answer_similarity). Результаты оценки сохраняются в переменной score_similarity, которая затем преобразуется в DataFrame pandas, similarity_df, с помощью метода to_pandas().

То же самое сделаем с гибридным набором данных:

```
# similarity search score
score_hybrid = evaluate(
  testing_dataset_hybrid,
  metrics=[
    faithfulness,
    answer_relevancy,
    context_precision,
    context_recall,
    answer_correctness,
    answer_similarity
  ]
)
hybrid_df = score_hybrid.to_pandas()
hybrid_df
```

Как только вы достигнете этой точки, использование ragas завершено! Мы провели оценку двух цепочек с использованием ragas, и в этих двух DataFrame, similarity_df и hybrid_df, содержатся все наши метрики. Осталось проанализировать данные, предоставленные ragas.

Анализ результатов ragas

В оставшейся части практикума мы займемся форматированием и сохранением данных. Этот код можно будет использовать повторно для извлечения данных из .csv файлов, что избавит вас от необходимости снова запускать весь потенциально дорогостоящий процесс оценки.

Начнем с установки переменных:

```
# Analysis that consolidates everything into easier-to-read scores
# key columns to compare
key_columns = [
    'faithfulness',
    'answer_relevancy',
    'context_precision',
    'context_recall',
    'answer_correctness',
    'answer_similarity'
]

# mean scores for each key column in similarity_df
similarity_means = similarity_df[key_columns].mean()

# mean scores for each key column in hybrid_df
hybrid_means = hybrid_df[key_columns].mean()

# comparison dataframe
comparison_df = pd.DataFrame({'Similarity Run': similarity_means, 'Hybrid Run': hybrid_means})

# difference between the means
comparison_df['Difference'] = comparison_df['Similarity Run'] - comparison_df['Hybrid Run']

# save dataframes to CSV files in the specified directory
similarity_df.to_csv(os.path.join('similarity_run_data.csv'), index=False)
hybrid_df.to_csv(os.path.join('hybrid_run_data.csv'), index=False)
comparison_df.to_csv(os.path.join('comparison_data.csv'), index=True)

print("Dataframes saved successfully in the local directory.")
```

В этом коде мы сначала определяем список key_columns, содержащий названия столбцов, которые будут использоваться для сравнения. Затем вычисляем средние значения для каждого ключевого столбца в similarity_df и hybrid_df с помощью метода mean() и сохраняем их в переменные similarity_means и hybrid_means.

Далее создаем новый DataFrame comparison_df, который сравнивает средние значения метрик для цепочек сходства и гибридных. Столбец Difference добавляется в comparison_df и вычисляется как разница между средними значениями для двух типов поиска. Наконец, мы сохраняем DataFrame similarity_df, hybrid_df и comparison_df как файлы формата CSV. Это позволит нам работать с этими файлами в будущем, не возвращаясь к повторной генерации данных.

Также имейте в виду, что это всего лишь один из способов проведения анализа. Здесь вы можете проявить творческий подход и настроить код для анализа аспектов, которые важны именно в вашей системе RAG. Например, вы можете сосредоточиться исключительно на улучшении механизмов поиска. Или, если вы работаете с данными, которые поступают из развернутой системы в режиме реального времени, у вас, вероятно, не будет эталонных данных, и вы захотите сосредоточиться на метриках, которые могут работать без эталонных данных (подробнее об этой концепции см. в разделе "Основные идеи разработчиков ragas" далее в этой главе).

Теперь мы извлечем сохраненные файлы, чтобы завершить анализ, а затем выведем его результаты для каждого этапа системы RAG по двум различным цепочкам:

```
### load dataframes from CSV files
```

```
sem_df = pd.read_csv(os.path.join('similarity_run_data.csv'))
rec_df = pd.read_csv(os.path.join('hybrid_run_data.csv'))
comparison_df = pd.read_csv(os.path.join('comparison_data.csv'), index_col=0)

print("Dataframes loaded successfully from the local directory.")

# Analysis that consolidates everything into easier to read scores
print("Performance Comparison:")
print("\n**Retrieval**:")
print(comparison_df.loc[['context_precision', 'context_recall']])
print("\n**Generation**:")
print(comparison_df.loc[['faithfulness', 'answer_relevancy']])
print("\n**End-to-end evaluation**:")
print(comparison_df.loc[['answer_correctness', 'answer_similarity']])
```

Этот код генерирует набор метрик, которые мы будем изучать дальше. Сначала мы загружаем DataFrame из CSV файлов, созданных в предыдущем блоке кода. Затем применяем анализ, который объединяет данные в более удобочитаемые показатели.

Далее мы продолжаем использовать переменные, определенные в предыдущем блоке кода, чтобы построить графики с помощью библиотеки matplotlib:

```
# plotting - create subplots for each category with increased spacing
fig, axes = plt.subplots(3, 1, figsize=(12, 18), sharex=False)
bar_width = 0.35
categories = ['Retrieval', 'Generation', 'End-to-end evaluation']
metrics = [
    ['context_precision', 'context_recall'],
    ['faithfulness', 'answer_relevancy'],
    ['answer_correctness', 'answer_similarity']
]
```

Здесь мы создаем графики для каждой категории с увеличенным интервалом между ними.

Следующим шагом является итерация по каждой из этих категорий и построение соответствующих метрик:

```
# iterate over each category and plot the corresponding metrics
for i, (category, metric_list) in enumerate(zip(categories, metrics)):
    ax = axes[i]
    x = range(len(metric_list))

    # plot bars for Similarity Run (hex color #D51900)
    similarity_bars = ax.bar(x, comparison_df.loc[metric_list, 'Similarity Run'], width=bar_width,
label='Similarity Run', color='#D51900', hatch='///')

    # add values to Similarity Run bars
    for bar in similarity_bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.1%}', ha='center', va='bottom', fontsize=10)

    # plot bars for Hybrid Run (hex color #992111)
    hybrid_bars = ax.bar([i + bar_width for i in x], comparison_df.loc[metric_list, 'Hybrid Run'],
width=bar_width, label='Hybrid Run', color='#992111', hatch='\\\\\\')

    # add values to Hybrid Run bars
    for bar in hybrid_bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.1%}', ha='center', va='bottom', fontsize=10)
```

```
    ax.set_title(category, fontsize=14, pad=20)
    ax.set_xticks([i + bar_width / 2 for i in x])
    ax.set_xticklabels(metric_list, rotation=45, ha='right', fontsize=12)

    # move the legend to the bottom right corner
    ax.legend(fontsize=12, loc='lower right', bbox_to_anchor=(1, 1))
```

Большая часть этого кода сосредоточена на форматировании визуализаций, включая построение столбцов для цепочек сходства и гибридных цепочек, а также добавление значений столбцов. Мы добавляем цвет столбцам и даже используем штриховку для улучшения доступности для людей с нарушением зрения.

Нам осталось внести несколько улучшений в визуализацию:

```
# Add overall labels and title
fig.text(0.04, 0.5, 'Scores', va='center', rotation='vertical', fontsize=14)
fig.suptitle('Performance Comparison', fontsize=16)

# adjust the spacing between subplots and increase the top margin
plt.tight_layout(rect=[0.05, 0.03, 1, 0.95])
plt.subplots_adjust(hspace=0.6, top=0.92)
plt.show()
```

Мы добавили подписи и заголовок, настроили интервалы между элементами и увеличили верхнее поле. Наконец, с помощью plt.show() отображаем визуализацию в интерфейсе ноутбука.

RAG выполняет две основные задачи: поиск и генерация. При оценке системы RAG вы можете разделить анализ на эти две категории. Давайте сначала обсудим оценку поиска.

Оценка поиска

Ragas предоставляет метрики для оценки каждого этапа конвейера RAG отдельно. Для поиска ragas предлагает две метрики: context precision и context recall:

```
Dataframes loaded successfully from the local directory.
Performance Comparison:

**Retrieval**:
                   Similarity Run  Hybrid Run  Difference
context_precision        0.656827    0.664257   -0.007430
context_recall           1.000000    0.966667    0.033333
```

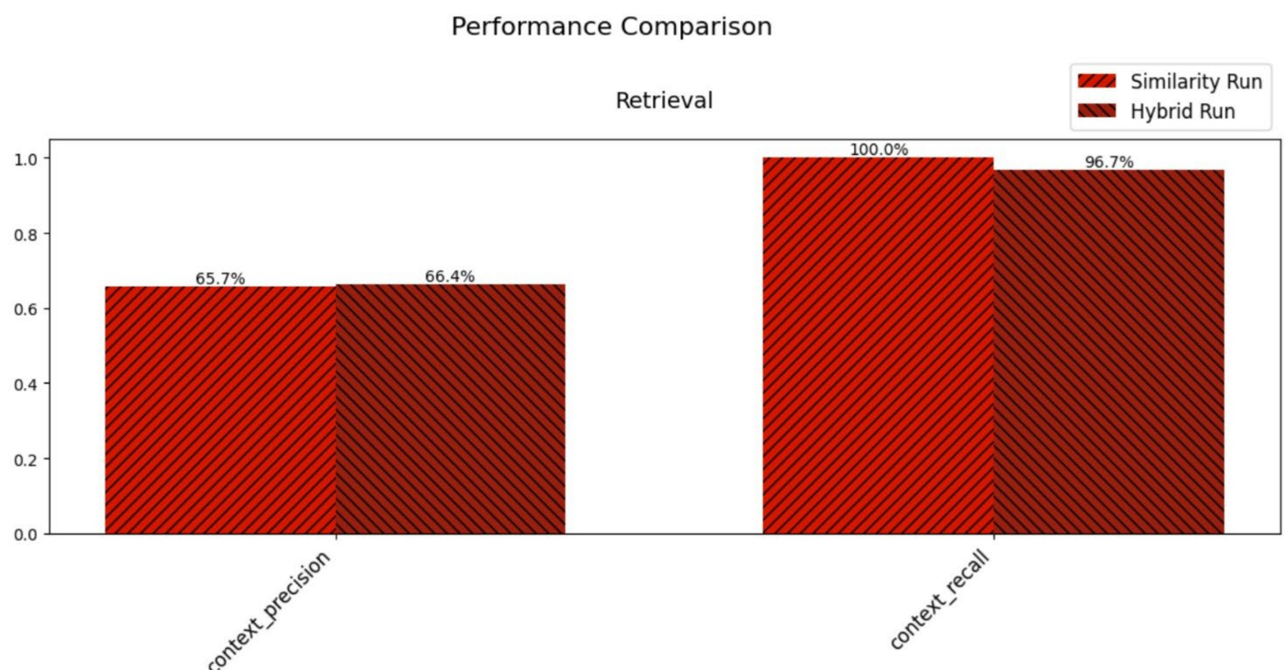

Рис. 9.2. Сравнение производительности поиска на основе сходства и гибридного

Оценка поиска направлена на анализ точности и релевантности найденных документов. Мы выполняем ее с помощью ragas, используя две метрики, описанные на сайте ragas:

- *context_precision:* соотношение сигнала к шуму в найденном контексте. context_precision оценивает, находятся ли все элементы, релевантные эталону, на более высоких позициях в результатах поиска. Идеально, если все релевантные фрагменты появляются в верхних позициях. Метрика вычисляется на основе вопроса, эталонного ответа и контекста. Значения варьируются от 0 до 1, где более высокие значения означают лучшую точность.
- *context_recall:* способен ли поиск найти всю релевантную информацию, необходимую для ответа на вопрос? context_recall измеряет, насколько найденный контекст соответствует аннотированному ответу, принимаемому за эталон. Метрика вычисляется на основе эталона и найденного контекста. Значения также находятся в диапазоне от 0 до 1, где более высокие значения указывают на лучшую производительность.

Если у вас есть опыт работы в области традиционной науки о данных или информационного поиска, вы, вероятно, знакомы с терминами точность и полнота (precision и recall) и можете задуматься, имеют ли эти метрики отношение к данным терминам. Метрики context precision и context recall, используемые в ragas, концептуально схожи с традиционными метриками точности и полноты.

В традиционном понимании точность измеряет долю релевантных элементов среди всех найденных, а полнота — долю найденных элементов среди всех релевантных. Аналогично, context precision оценивает релевантность найденного контекста, анализируя, находятся ли элементы, релевантные эталону, на более высоких позициях, а context recall измеряет, насколько найденный контекст охватывает релевантную информацию, необходимую для ответа на вопрос.

Однако есть некоторые ключевые отличия.

Традиционные метрики точности и полноты обычно рассчитываются на основе бинарного решения (релевантно или нерелевантно) для каждого элемента, тогда как context precision и context recall в ragas учитывают ранжирование и соответствие найденного контекста эталонному ответу.

Метрики context precision и context recall специально разработаны для оценки производительности поиска в задачах вопросов-ответов, принимая во внимание специфические требования поиска релевантной информации для ответа на заданный вопрос.

При анализе результатов нужно помнить, что мы используем небольшой набор данных для эталонной проверки. Более того, исходный набор данных, на основе которого работает вся система RAG, также невелик, что может повлиять на результаты. Поэтому не стоит слишком полагаться на конкретные числа, которые вы видите здесь. Однако эти результаты показывают, как можно использовать ragas для проведения анализа и визуализации результатов оценки.

Далее мы рассмотрим аналогичный анализ для этапа генерации.

Оценка генерации

Оценка генерации измеряет уместность ответа, сгенерированного системой, при предоставленном контексте. Для этапа генерации ragas предлагает две метрики:

- *faithfulness:* насколько фактически точен сгенерированный ответ? Эта метрика измеряет фактологическую согласованность сгенерированного ответа относительно предоставленного контекста. Она рассчитывается на основе ответа и найденного контекста. Значения варьируются от 0 до 1, где более высокие значения указывают на лучшую точность.
- *answer_relevancy:* насколько релевантен сгенерированный ответ заданному вопросу? Answer relevancy сосредоточена на оценке того, насколько уместен сгенерированный ответ по отношению к исходному запросу. Низкие оценки присваиваются ответам, которые являются неполными или содержат избыточную информацию. Высокие оценки указывают на лучшую релевантность. Метрика вычисляется на основе вопроса, контекста и ответа.

```
**Generation**:
                 Similarity Run  Hybrid Run  Difference
faithfulness           0.907143    0.870833    0.036310
answer_relevancy       0.949047    0.944186    0.004861
```

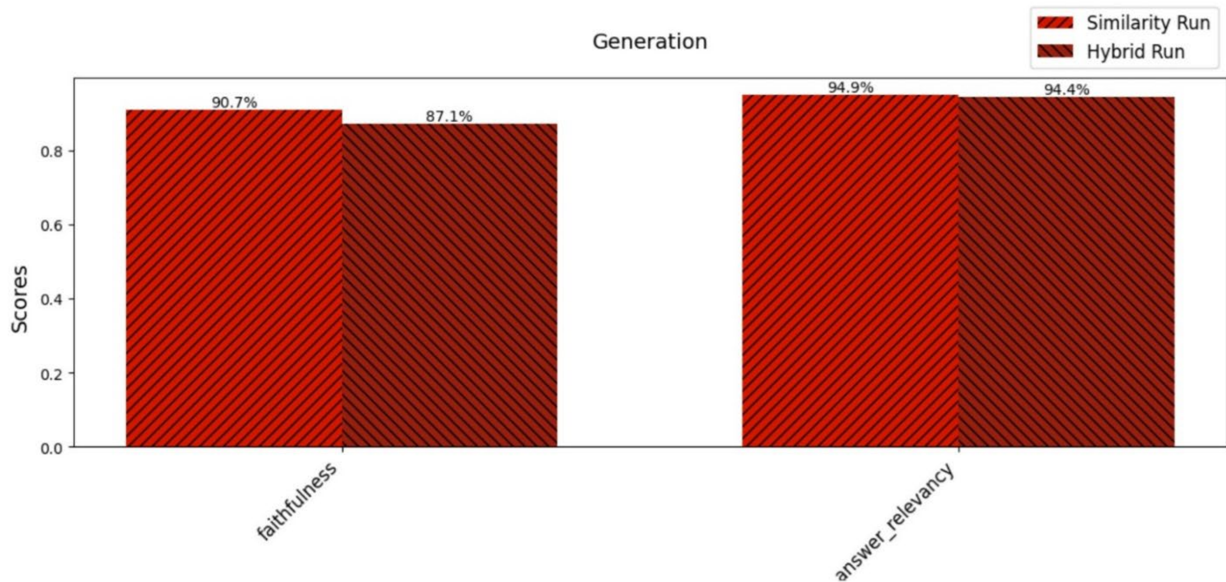

Рис. 9.3. Сравнение производительности генерации для поиска на основе сходства и гибридного

Мы использовали небольшой набор данных для эталонной проверки и анализа, что, вероятно, делает эти результаты менее надежными. Однако вы можете увидеть, как эти результаты формируют основу для предоставления информативного представления о том, что происходит на этапе генерации в системе RAG.

Это подводит нас к следующему набору комплексных метрик — для оценки всей системы, которые мы обсудим далее.

Комплексная оценка системы

Помимо метрик для оценки этапов конвейера RAG по отдельности, ragas предоставляет метрики для всей системы RAG, называемые end-to-end оценкой:

```
**End-to-end evaluation**:
                   Similarity Run  Hybrid Run  Difference
answer_correctness       0.835962    0.651976    0.183987
answer_similarity        0.935687    0.930801    0.004886
```

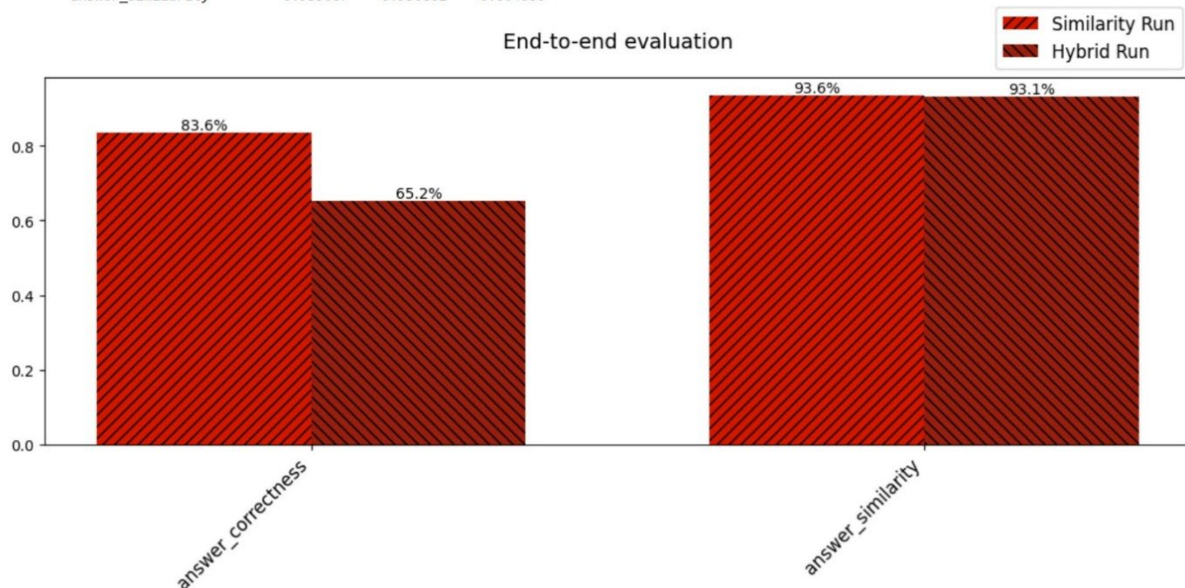

Рис. 9.4. Комплексное сравнения производительности системы RAG для поиска на основе сходства и гибридного

Комплексные метрики предназначены для оценки производительности всей системы, позволяя оценить общий пользовательский опыт при использовании конвейера:

- *answer_correctness:* оценивает точность сгенерированного ответа в сравнении с эталоном. Эта метрика измеряет, насколько точен сгенерированный ответ по отношению к эталонному. Оценка основывается на эталоне и ответе, а значения варьируются от 0 до 1. Более высокие значения указывают на лучшее соответствие между сгенерированным ответом и эталоном.
- *answer_similarity:* оценивает семантическое сходство между сгенерированным ответом и эталоном.

Чтобы не усложнять практикум, мы исключили несколько дополнительных метрик. Давайте обсудим их далее.

Иные метрики оценки конвейера RAG

Такие метрики предполагают анализ эффективности отдельных этапов конвейера для получения более глубокого понимания их работы и выявления областей для улучшения. Мы уже рассмотрели две метрики для этапов поиска и генерации, и вот еще несколько, доступных на платформе ragas:

Context relevancy (релевантность контекста). Эта метрика оценивает релевантность найденного контекста на основе вопроса и контекстов.

Context entity recall (полнота сущностей в контексте). Метрика измеряет полноту найденного контекста, основываясь на количестве сущностей, присутствующих как в данных ground_truth, так и в контекстах, относительно числа сущностей, присутствующих только в данных ground_truth. Проще говоря, это мера того, какая доля сущностей из ground_truth была найдена. Эта метрика особенно полезна в задачах, основанных на фактах, например, справочные службы для туристов или исторические вопросы-ответы. Она помогает оценить механизм поиска сущностей путем сравнения с данными ground_truth, так как в задачах, где сущности имеют значение, контексты должны их охватывать.

Aspect critique (анализ по аспектам). Метрика предназначена для оценки ответов на основе заранее определенных аспектов, таких как безопасность или корректность. Пользователи также могут определять собственные аспекты для оценки в соответствии с конкретными критериями. Результаты анализа по аспектам являются бинарными, указывая, соответствует ли ответ заданному аспекту или нет. Эта оценка выполняется на основе ответа.

При подготовки этой главы мы имели возможность поговорить с одним из основателей ragas, Шахулом Эс, чтобы получить дополнительные инсайты о платформе и рекомендациях по ее более эффективному использованию для разработки и оценки RAG. Ragas — молодая платформа, но, как вы могли видеть в ходе практикума, она уже имеет прочную основу метрик, которые можно использовать для оценки системы RAG. Это также означает, что у ragas есть значительный потенциал для роста, и платформа, специально разработанная для реализации RAG, будет продолжать развиваться. Шахул поделился полезными советами и рекомендациями, которые мы кратко изложим в следующем разделе.

Инсайты от основателя ragas

Генерация синтетических данных. Первая проблема, с которой обычно сталкиваются при оценке RAG, — это недостаток тестовых эталонных данных. Основная цель ragas — создание алгоритма, который может генерировать тестовый набор данных, охватывающий широкий спектр типов вопросов. После использования ragas для синтеза эталонных данных полезно пересмотреть сгенерированные данные и исключить любые вопросы, которые не соответствуют задаче.

Метрики обратной связи. На данный момент в разработке платформы особое внимание уделяется включению различных вариантов обратной связи от пользователей. Это могут быть явные метрики (например, ошибки) и неявные (уровень удовлетворенности, лайки/дизлайки и другие механизмы). Любое взаимодействие с пользователем потенциально может быть источником неявной обратной связи. Хотя такие данные могут быть *шумными* (с точки зрения анализа), они все равно полезны при правильном использовании.

Метрики с эталоном и без. Шахул разделил метрики на те, которые требуют эталонных данных (reference metrics), и метрики, не зависящие от эталона (reference-free metrics). Команда ragas делает акцент на разработке метрик без эталона. Для многих областей, где сбор эталонных данных затруднен, это особенно важно, так как позволяет проводить хотя бы частичную оценку.

Достоверность (faithfulness) и релевантность ответа (answer relevance) Шахул отметил как метрики без эталона.

Оценка в условиях развертывания. Метрики без эталона также идеально подходят для оценки системы в условиях развертывания, где наличие эталонных данных маловероятно.

Эти ключевые инсайты демонстрируют текущие возможности ragas, и интересно будет наблюдать за дальнейшим развитием платформы, которая поможет совершенствовать системы RAG. Последнюю версию документации по ragas можно найти здесь.

На этом мы завершаем практикум по оценке с использованием ragas. Однако ragas — не единственный инструмент для оценки RAG; существует множество других подходов! Далее мы обсудим некоторые из них.

Дополнительные методы оценки

Мы обсудим некоторые из наиболее популярных методов, которые можно использовать для оценки производительности системы RAG после получения или генерации эталонных данных.

BLEU (Bilingual Evaluation Understudy)

BLEU измеряет совпадение n-грамм между сгенерированным ответом и эталонным ответом. Оценка указывает на степень сходства между ними. В контексте RAG BLEU может быть использован для оценки качества сгенерированных ответов путем их сравнения с эталонными. Рассчитывая совпадение n-грамм, BLEU оценивает, насколько близко сгенерированные ответы соответствуют эталонным по выбору слов и формулировке. Однако важно отметить, что BLEU больше сосредоточен на поверхностном сходстве и может не отражать семантическое значение или релевантность сгенерированных ответов.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE оценивает качество сгенерированного ответа, сравнивая его с эталоном с точки зрения полноты (recall). Он измеряет, насколько большая часть эталона была включена в сгенерированный ответ. Для оценки RAG ROUGE может использоваться для анализа охвата и полноты сгенерированных ответов. Рассчитывая полноту между сгенерированными и эталонными ответами, ROUGE оценивает, насколько хорошо сгенерированные ответы охватывают ключевую информацию и детали, присутствующие в эталонных. ROUGE особенно полезен, если эталонные ответы более длинные или детализированные, поскольку он фокусируется на совпадении информации, а не на точных совпадениях слов.

Семантическое сходство

Косинусное сходство или семантическое текстовое сходство (STS), могут быть использованы для оценки семантической релевантности между сгенерированным ответом и эталоном. Эти метрики учитывают значение и контекст, выходя за рамки точного совпадения слов. В контексте RAG метрики семантического сходства позволяют оценить семантическую связность и релевантность сгенерированных ответов. Сравнивая семантические представления сгенерированных ответов и эталонов, эти метрики анализируют, насколько хорошо сгенерированные ответы отражают основное значение и контекст эталонных ответов. Метрики семантического сходства особенно полезны, если сгенерированные ответы используют различные слова или формулировки, но при этом передают то же самое значение, что и эталон.

Оценка человеком

Хотя автоматизированные метрики предоставляют количественную оценку, оценка человеком остается важным аспектом для анализа связности, беглости и общего качества сгенерированных ответов по сравнению с эталоном. В контексте RAG оценка человеком включает участие экспертов, которые оценивают сгенерированные ответы на основе различных критериев. Эти критерии могут включать:

- Релевантность ответа по отношению к вопросу.
- Фактическую точность.
- Ясность ответа.
- Общую связность.

Эксперты могут предоставлять качественную обратную связь и инсайты, которые автоматические метрики могут упускать, например:

- Уместность тона ответа.
- Наличие несоответствий или противоречий.
- Общий пользовательский опыт.

Оценка человеком может дополнять автоматические метрики, обеспечивая более всесторонний и тонкий анализ производительности системы RAG.

Комбинация методов оценки

При оценке системы RAG часто бывает полезно использовать комбинацию методов оценки для получения целостного представления о производительности системы. Каждый метод имеет свои сильные и слабые стороны, а использование нескольких метрик позволяет провести более надежный и всесторонний анализ. Также важно учитывать специфические требования и цели вашего приложения RAG при выборе подходящих методов оценки. Некоторые приложения могут отдавать приоритет фактической точности, в то время как для других важнее беглость и связность сгенерированных ответов. Согласовывая методы оценки с вашими конкретными потребностями, вы сможете эффективно оценить производительность системы RAG и выявить области для улучшения.

Ссылки

MSMARCO: https://microsoft.github.io/msmarco/
HotpotQA: https://hotpotqa.github.io/
CQADupStack: http://nlp.cis.unimelb.edu.au/resources/cqadupstack/
186 Evaluating RAG Quantitatively and with Visualizations
Chatbot Arena: https://chat.lmsys.org/?leaderboard
MMLU: https://arxiv.org/abs/2009.03300
MT Bench: https://arxiv.org/pdf/2402.14762