STA323

Big Data Analysis Software and Application (Hadoop or Spark)

Report on Project 2

12112627 李乐平

Question 1. Questioning and Answering System.

Design and build a generative question answering system. The training data set is from SQuAD v2 Dataset (also located in the server path /shareddata/data/project2).

(1). [2 points] Write python code to process data. Use the context and question as input, and the answer as output. Use the official validation set as test set, and split the original training set into training set and validation set (5000 samples for valid set, the rest for train set). Prepare the data according to the requirements of model training (Can refer to the original T5 and Flan-T5 paper for data format). You can use either Pyspark or pure python code.

Answer:

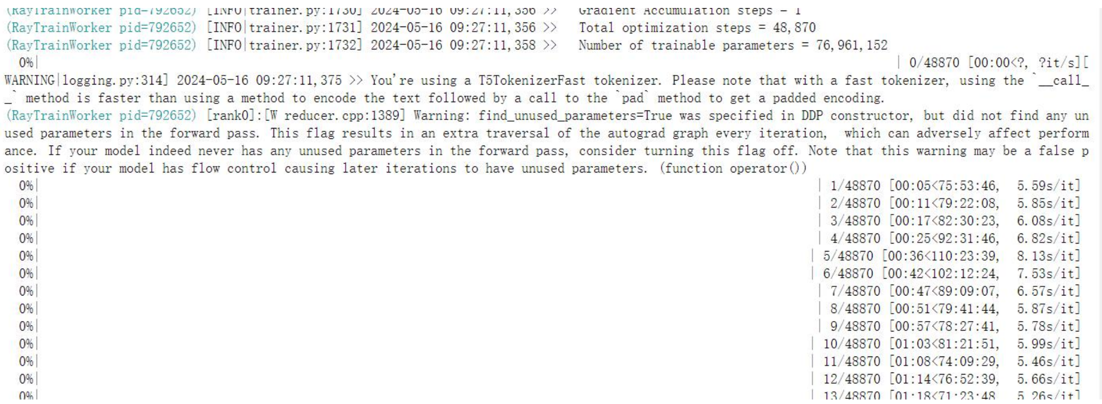
根据论文中提到的数据格式，对于 SQuAD 数据集，采用 question: {} context: {}的格式拼接作为输入，然后划分验证集即可。

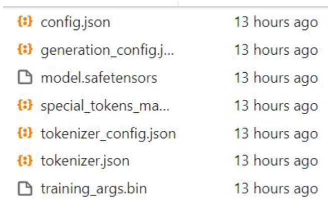
```
n = 5000
df = pd.read_parquet("./data/Task1/squad_v2/squad_v2/train-00000-of-00001.parquet")
train_df, validation_df = train_test_split(df, test_size=n / len(df), random_state=42)

# Process each row:
    def preprocess_squad_batch(
        examples,
        question_column: str,
        context_column: str,
        answer_column: str,
    ) -> Tuple[List[str], List[str]]:
        questions = examples[question_column]
        contexts = examples[context_column]
        answers = [eval(row) for row in examples[answer_column]]

        def generate_input(_question, _context):
            return " ".join(["question:", _question.lstrip(), "context:", _context.lstrip()])

        inputs = [generate_input(question, context) for question, context in zip(questions, contexts)]
        targets = [answer["text"][0] if len(answer["text"]) > 0 else "" for answer in answers]
        return inputs, targets
```

(2). [5 points] Write the bash script and Ray-train python code to train the QA model by further funetuning the Flan-T5-small model. As no GPU is available in the server, you can use pytorch-cpu to debug your code, train the model for a few hours and save a checkpoint. Note that the validation set is used for designing the hyperparameters and selecting the model checkpoint. You can also refer to the training examples in the huggingface repo. You can also rent the GPU server in AutoDL.

Answer:

详见代码。代码运行后能成功保存一个 checkpoint。代码参考了 github 上的官方样例，主要修改的地方有将主函数的调用改为由 ray train 唤起、数据集的处理和加载、验证集的分割、测试度量的加载以及诸多的环境与路径配置的问题。





(3). [4 points] Deploy the finetuned QA model with Spark-NLP, and answer the questions of the test set in a streaming processing manner using Kafka. You can search the spark-nlp model here. Note that the deployed model is not the original Flan-T5-small model.

Answer:

首先编写 producer 程序向 kafka 中输入.parquet 文件中的信息。

```
import numpy as np
import pandas as pd
from kafka import KafkaProducer
import json

df = pd.read_parquet("./data/Task1/squad_v2/squad_v2/validation-00000-of-00001.parquet")

def convert_to_serializable(obj):
    if isinstance(obj, np.ndarray):
        return obj.tolist()
    return obj

producer = KafkaProducer(
    bootstrap_servers="localhost:9092",
    value_serializer=lambda v: json.dumps(v, default=convert_to_serializable).encode('utf-8')
)

topic_name = "p2"
cnt = 0
for _, row in df.iterrows():
    producer.send(topic_name, row.to_dict())
    producer.flush()

producer.close()
```

配置好 Kafka，并启动 sparknlp。从 Kafka 会话中读进测试集后将输入格式化。需要处理 Huggingface 模型使其变成 sparknlp 能够加载的模型，最后加载模型并启动流水线即可。

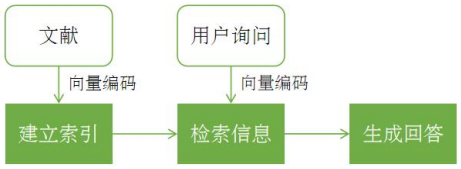
```
import sparknlp
from pyspark.ml import Pipeline
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StructType, StringType, IntegerType

spark = SparkSession.builder.appName("Spark NLP") \
```

```
    .config(
        "spark.jars",
        "./kafka-clients-3.5.0.jar,./spark-sql-kafka-0-10_2.12-3.5.0.jar, \
        ./spark-token-provider-kafka-0-10_2.12-3.5.0.jar, \
        ./commons-pool2-2.12.0.jar") \
    .config("spark.sql.shuffle.partitions", "3") \
    .config("spark.executorEnv.PYSPARK_PYTHON","/root/anaconda3/bin/python") \
    .config("spark.executor.memory", "8g") \
    .config("spark.driver.memory", "8g") \
    .config("spark.log.level", "ERROR") \
    .master("local[*]") \
    .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \
    .config("spark.kryoserializer.buffer.max", "2000M") \
    .config("spark.driver.maxResultSize", "0") \
    .config("spark.jars.packages", "com.johnsnowlabs.nlp:spark-nlp_2.12:5.3.3") \
    .getOrCreate()

from sparknlp.base import *
from sparknlp.annotator import *
import pyspark.sql.functions as F

p2_df = spark.readStream.format("kafka")\
    .option("kafka.bootstrap.servers", "localhost:9092")\
    .option("subscribe", "p2") \
    .option("startingOffsets", "earliest") \
    .load()

df = df.withColumn(
    'text', F.format_string('question: %s context: %s', F.col('question'), F.col('context'))
)

p2_df.writeStream \
    .format("memory") \
    .outputMode("append") \
    .queryName("validation_set") \
    .option("checkpointLocation", "./checkpoint") \
    .start()

schema = "id STRING, title STRING, context STRING, question STRING, answers STRUCT<text: ARRAY<STRING>,
answer_start: ARRAY<INT>>"
parsed_df = df \
    .withColumn("parsed_value", from_json(col("value"), schema)) \
    .select("parsed_value.*")

document_assembler = DocumentAssembler() \
    .setInputCol("text") \
    .setOutputCol("documents")

MODEL_NAME = "output/debug_seq2seq_squad/checkpoint-500"

EXPORT_PATH = f"output/onnx/flan_t5_finetuned"

!optimum-cli export onnx --task text2text-generation-with-past --model {MODEL_NAME} {EXPORT_PATH}

local_model_path = "./output/onnx/flan_t5_finetuned"

t5 = T5Transformer \
    .loadSavedModel(local_model_path, spark) \
    .setUseCache(True) \
    .setInputCols(["documents"]) \
    .setOutputCol("outputs")
parsed_df = parsed_df.withColumn('text', F.format_string('question: %s context: %s', F.col('question'),
F.col('context')))

results = pipeline.fit(parsed_df).transform(parsed_df)
```

```
+-----------------------------------------------------------------+----------------------------------------------------------------------------+
|question                                                         |outputs                                                                     |
+-----------------------------------------------------------------+----------------------------------------------------------------------------+
|In what country is Normandy located?                             |[{document, 0, 5, France, {sentence -> 0}, []}]                             |
|From which countries did the Norse originate?                    |[{document, 0, 26, Denmark, Iceland and Norway, {sentence -> 0}, []}]       |
|What century did the Normans first gain their separate identity? |[{document, 0, 11, 10th century, {sentence -> 0}, []}]                      |
|What is France a region of?                                      |[{document, 0, 7, Normandy, {sentence -> 0}, []}]                           |
|What type of major impact did the Norman dynasty have on modern Europe?|[{document, 0, 31, political, cultural and military, {sentence -> 0}, []}]|
+-----------------------------------------------------------------+----------------------------------------------------------------------------+
```

(4). [2 points] Read a survey paper about retrieval-augmented generation and illustrate one possible method that can support open-domain question answering with system diagram and pipeline introduction. The code is not required for this task.

Answer:



基于这篇对大型语言模型（LLM）的检索增强生成（RAG）的调查论文，支持开放领域问题回答的一种可能方法是 Naive RAG 方法。Naive RAG 是一种早期的方法，它将外部知识集成到 LLM 中，以增强其性能，特别是在知识密集型任务中。它遵循传统的流程，包括三个主要阶段：索引、检索和生成。

流程

1. 索引：文献的有效内容被提取、处理，并转换为统一格式。内容被分割成 LLM 可以处理的较小块。然后将这些块编码为向量表示，并存储在向量数据库中以进行有效的检索。
2. 检索：在收到用户询问后，系统使用与索引阶段相同的编码模型将查询转换为向量。然后计算查询向量与数据库中文档向量之间的相似度分数，检索出与查询最相似的前 K 个块。
3. 生成：检索到的文档块和原始查询组合成一个全面的提示。LLM 使用此提示生成响应，利用其内在知识和检索到的文档中包含的信息。

Naive RAG 方法可以通过为 LLM 提供相关的外部信息，有效减少错误内容的生成，从而增强对开放域问题生成的答案的准确性和可信度。当然，经过更新换代，也有了更加先进的 Advanced RAG 方法、Modular RAG 方法等。其中，Advanced RAG 引入了预检索、后检索等策略；而 Modular RAG 则引入了更多的模块如搜索、融合（RAG-Fusion）等，同时还引入了更加灵活的检索模式。

Question 2. Startup Analyses.

[4 points] Select one startup you are interested from the following candidates. Suppose you are an investor who is interested in the startup. Write an analyses report about the startup and their product. More than one page.

Answer: 选定的初创公司：ClickHouse。

公司优势和特色

ClickHouse 是一款高性能的分析性数据库管理系统，专为处理大规模数据和执行复杂查询而优化。其主要优势在于其创新的列存储模型和优化的查询处理能力。相比于传统的行存储数据库，ClickHouse 通过将数据以列的方式存储在磁盘上，实现了更快的查询速度和更高的压缩率。此外，ClickHouse 支持水平扩展，可以在数据增长的同时轻松扩展集群规模，为用户提供了出色的扩展性和性能表现。

产品详述和分析

ClickHouse 的架构设计旨在提供高性能的分析能力、可扩展性和容错性。其主要组件包括服务器、存储、查询处理器、客户端和 ZooKeeper。服务器负责接收和处理来自客户端应用程序的查询，并将结果返回给客户端，同时管理数据存储、执行查询以及处理集群中节点间的数据复制。存储层负责在磁盘上存储数据，并通过分片将数据分布在集群中的多个节点上，同时负责数据压缩、分区和索引以优化查询性能。查询处理器负责解析和优化查询，生成执行计划以最小化数据读取和处理时间。客户端是用户与 ClickHouse 交互的接口，可以是命令行工具、SQL 客户端或应用程序编程接口(API)。ZooKeeper 是一个分布式协调服务，用于管理集群元数据和协调集群中节点间的数据复制，负责管理集群配置、数据同步和故障切换。ClickHouse 还支持多种复制模式，包括异步和同步复制，以确保即使集群中的一个或多个节点失败，数据也是可用的。

目标市场的未来趋势

随着数据驱动的业务模式在各行各业的普及，分析性数据库的需求将会持续增长。ClickHouse 作为一款高性能、可扩展的分析型数据库，将在未来的市场中占据重要地位。特别是在处理大数据集、实时数据分析以及时间序列数据分析等方面，ClickHouse 将具有更广阔的应用前景。随着企业对数据洞察力的需求不断增加，ClickHouse 将成为许多企业的首选解决方案，助力他们更好地理解数据并做出明智的决策。

公司面临的威胁

尽管 ClickHouse 拥有出色的性能和功能，但仍面临一些潜在威胁。首先，数据库市场竞争激烈，存在许多其他类似的分析型数据库产品，如 Apache Druid 和 Amazon Redshift 等，它们也具有类似的性能和功能。其次，ClickHouse 在处理事务性工作负载方面的能力相对有限，可能无法满足某些需要频繁读写操作的应用场景。此外，ClickHouse 的复杂设置和配置可能对某些用户构成障碍，尤其是对于缺乏数据库管理经验的用户来说。

关于公司的公开报导

从 ClickHouse CEO Aaron Katz 的讲话中我们可以得知，ClickHouse 的优势在于速度和简单性，并且 ClickHouse Cloud 将这一优势推向了一个新的高度，使企业能够以其他市场解决方案成本的一小部分启动服务并分析数据。在短短几个月内，ClickHouse Cloud 的公测版本已经吸引了 100 多家客户和数千名新用户，涵盖了开发人员、数据分析师、市场营销等业务的关键领域，在这些领域，数据被分析和存储。Twilio 的首席产品官 Eyal Manor 表示，ClickHouse Cloud 符合他们希望赋予开发人员能力的愿望，使他们能够在几天内从概念到实时分析用例的交付中实现，并大大加快了产品创新周期。此外，ClickHouse 还获得了来自领先技术投资者 Thrive Capital 的新一轮投资，进一步验证了其市场机会、团队和商业模式。这笔资金将支持进一步投资于技术，并允许 ClickHouse 继续建设其世界领先的软件工程团队。通过这些信息，我们可以看出 ClickHouse 在持续改进产品功能、提高用户体验方面的不懈努力，以及其在市场发展和商业拓展方面的积极态度。「新闻链接」

小结

综上分析，对 ClickHouse 的投资前景可以持乐观态度。作为一款高性能、易于使用的在线分析型列式数据库管理系统，ClickHouse 在处理大规模数据和执行复杂查询方面表现出色。其创新的产品功能和强大的生态系统使其成为企业数据处理和分析的理想选择。

ClickHouse Cloud 的推出进一步拓展了公司的市场份额，并为用户提供了灵活、高效的云端解决方案。在成功的公测阶段吸引了众多客户的同时，ClickHouse 还得到了来自 Thrive Capital 的新一轮投资，进一步验证了其商业模式和市场潜力。

基于以上分析，投资者可以密切关注 ClickHouse，并考虑长期持有该公司的股票。随着数据驱动型业务的持续增长，以及对实时分析和大数据处理需求的不断增加，ClickHouse 有望在未来取得更大的成功。同时，投资者也应该关注公司的技术创新和市场拓展动态，以及行业竞争格局的变化，以便及时调整投资策略。

Question 3. Paper Reading.

[4 points] Read one of the following papers and write a report more than one page.

Answer: 选定：Spark SQL: Relational Data Processing in Spark

What is the problem addressed in the paper?

Spark SQL 致力于解决原生的 Spark RDD 无法实现关系型数据处理的问题，同时旨在提供基于 DBMS 的高性能服务。Spark SQL 还尝试支持图处理、机器学习等高级扩展任务。

Is this a new problem?

这不是一个全新的问题，而是基于已有技术的扩展尝试。

或者说，这是将已有的技术理念在自己的技术框架下实现并应用的一次尝试。

What is the scientific hypothesis that the paper is trying to verify?

论文试图验证的假设包括：

① DataFrame API 可以在 Spark 程序中执行关系型操作，同时保持懒加载的特性，以便进行关系型优化。

② Catalyst 优化器 可以通过 Scala 的模式匹配来表达可组合规则，并且是图灵完备的，这使得它能够支持广泛的数据源和算法。

③ Spark SQL 可以与 Spark 的其他组件（如机器学习库）无缝集成，并且比纯 Spark 代码在可表达为 SQL 的计算中更快、更节省内存。

What are the key related works and who are the key people working on this topic?

Spark SQL 是由一步步技术迭代而来的，其有众多的相关工作。其最接近的前身是 R.S. Xin 等人的 Shark。相对于基于相同引擎开发的，具有相似功能的 Shark，Spark SQL 提供了更友好的用户接口。而 M. Isard 和 Y. Yu 的 DryadLINQ 则是 Spark SQL 设计的灵感来源，相对而言，Spark SQL 提供了更加接近数据科学库的 DataFrame 接口，同时还提供了接入不同数据源、不同数据格式的 API。

Catalyst 优化器则与 G. Graefe 和 D. DeWitt 的 EXODUS 和 G. Graefe 的 Cascades 优化器框架相似。

Spark SQL 还支持在大型集群上运行的高级分析算法，其依赖 M. Zaharia 等人提出的支持迭代算法的平台和 J. E. Gonzalez 等人、Y. Low 等人各自提出的支持图分析的框架。

What is the key of the proposed solution in the paper?

针对如何在 Apache Spark 中整合关系型数据处理和 Spark 的函数式编程 API 的问题，论文提出的核心解决方法是 Spark SQL 模块，它通过以下两个主要贡献来桥接关系型处理和程序化处理：

① DataFrame API：这是一个声明式的 API，允许在 Spark 的内置分布式集合和外部数据源上执行关系操作。这个 API 类似于 R 中广泛使用的数据框概念，但是它延迟执行操作以便进行关系优化。

② Catalyst 优化器：这是一个高度可扩展的优化器，使用 Scala 编程语言的特性构建，使得添加可组合规则、控制代码生成和定义扩展点变得容易。Catalyst 使得 Spark SQL 能够支持各种数据源和算法，包括机器学习领域的数据类型和查询联合到外部数据库。

How are the experiments designed?

论文主要组织了 2 方面的性能测试，包括 SQL 查询的性能和 Spark 程序的性能。

对于 SQL 查询的性能，作者进行了扫描、聚合、联合、UDF 等 4 个方面的实验，将 Spark 与 Shark 和 Impala 进行了性能的对比。在所有方面 Spark SQL 的性能均优于 Shark。在数据量较小，选择性较强时，Spark SQL 的性能不如 Impala，但随着处理的数据量增大，Spark SQL 的性能超过了 Impala。

对于 Spark 程序的性能，作者组织了基于 DataFrame API 的程序与使用 Python API 和 Scala API 的原生 Spark 程序的性能对比。结果显示，对于同一个查询，DataFrame API 的性能是 Python API 的 12 倍，同时也优于 Scala API。

此外，作者还进行了一个简单的流水线性能比较试验。比较了 DataFrame API 与 Spark + Scala 对于单词统计任务（统计出现最频繁的单词）的效率。实验表明 DataFrame API 的效率是 Spark + Scala 的约 2 倍。

What datasets are built/used for quantitative evaluation? Is the code open sourced?

在 SQL 性能比较部分，作者提到其使用了 110GB 的数据集，其使用列式压缩的 Parquet 格式存储。

而对于 Spark 程序的性能比较部分，作者提到其使用了一个有 10 亿形如(a, b)整数对的数据集，a 共有 10 万个不同值。

对于流水线比较部分，则使用了一个包含 10 亿条信息，使用 HDFS 存储的合成数据集，每条信息包含 10 个单词，均是从英语词典上抓下来的。

Spark SQL 是著名开源项目 Spark 的一部分，可以轻易地找到其开源代码。但是其数据集并未见开源，至少从论文中无法轻易地找到。

Is the scientific hypothesis well supported by evidence in the experiments?

从实验设计的角度而言，个人认为论文提出的假设不能很好地被实验支持。一方面，其进行的实验过于简单，没有构建实际的应用场景进行性能实验。另一方面，其仅使用了列式存储的数据集而未涉及传统的行式存储数据集，实验不够全面。

What are the contributions of the paper?

论文引入了 Spark SQL 这个新模块，它在 Apache Spark 中提供了与关系处理深度整合的功能。Spark SQL 通过引入 DataFrame API 扩展了 Spark，使得用户可以使用声明式方法进行关系处理，带来诸如自动优化等好处。它支持广泛的功能，包括半结构化数据、查询联合和机器学习数据类型。Spark SQL 基于可扩展的优化器 Catalyst，并通过 Scala 编程语言嵌入实现了这些功能，使得添加优化规则、数据源和数据类型变得简单。用户反馈和基准测试表明，Spark SQL 大幅简化了编写混合关系和过程处理的数据流水线，并且相对于之前的 SQL-on-Spark 引擎，提供了显著的性能提升。

当然，虽然其支持 DataFrame 和 SQL 的大部分语法，但是实现和用法不尽相同，给我们带来了额外的学习压力。

What should/could be done next?

接下来可以探索基于 Spark SQL 的一些应用。作者在文中列举了 2 个例子，包括通用在线聚合、计算基因组学等。

Appendix.

使用的大模型有 ChatGPT 和 Copilot（原 New Bing）。主要的 Prompt 如下：

[文章内容]
请阅读上述文章并结合你已有的知识，使用中文写一篇关于 ClickHouse 的报告。
在这里，你需要使用流畅而准确的中文来编写报告。报告应包含 5 个部分，分别围绕主题
"The key strength and niche of the company and their products."
"The detail description and analyses of the product."
"The future trend of the targeted market."
"Are there any threats to the company?"
"What do you learn from the public talks or interviews of the entrepreneur/company?"
展开。

[另一篇文章内容]
请阅读这篇公开文章，并基于此重写报告的第 5 部分"从企业家/公司公开演讲或采访中学到的东西"

请你根据先前的分析报告，用流畅的中文给出关于 ClickHouse 的投资建议，作为报告的结尾

由于 Copilot 的 bug 比较多，会话并没有得到保存，因此诸多 prompt 没有得到记录。但总体而言大致都是类似 任务（完成阅读）+具体问题的 prompt。

失败的情况：

在不包含关键词「流畅」时，大模型会生成标志性的分点式回答。

在输入内容过长或会话进行多轮后，大模型会遗忘最早的输入。

在使用微软的 Copilot 时，其似乎不太听得懂人话，无论用中文还是英文输入，其大部分输出均在我的意料之外，如果不是需要读 pdf 还是别使用 Copilot 了……同时，其似乎倾向于只阅读论文的第一部分而忽视后面的部分，也很令人火大。