Instituto Infnet

GRADUAÇÃO EM BANCO DE DADOS

Projeto de Bloco: Ciência de Dados Aplicada [24E3_5]

WANDERSON RAFAEL MENDONÇA BATISTA

TESTE DE PERFORMANCE – TP4

PROF. DIEGO DA SILVA RODRIGUES

RIO DE JANEIRO, 2024

Configuração do Ambiente de Desenvolvimento:

Configure seu ambiente de desenvolvimento, incluindo Git para controle de versão e preparação para deploy. Lembre-se de seguir a estrutura do CRISP-DM para organizar seu projeto de forma eficiente e escalável.

https://github.com/wanderaf/ProjetoDeBloco

1. **Identificação e Escolha do Modelo LLM (Local):**

- **Critérios de Seleção: Pesquise e selecione o modelo de linguagem natural mais adequado para a sua aplicação, considerando os seguintes critérios:**
- **Desempenho: Avalie a precisão, a capacidade de resposta e o tipo de tarefa para a qual o modelo foi treinado (ex: GPT, BERT, T5).**
- **Custo Computacional Local: Verifique os recursos de hardware necessários para rodar o modelo no ambiente local (como GPU, memória RAM, etc.).**
- **Acessibilidade Local: Garanta que o modelo escolhido pode ser carregado localmente, utilizando bibliotecas como Transformers da HuggingFace para download e execução do modelo em sua própria máquina.**

O modelo escolhido para a aplicação foi o t5-small, da arquitetura T5, devido à sua capacidade de realizar tarefas de transformação de texto com precisão e eficiência, como previsão baseada em padrões históricos. Esse modelo apresenta requisitos computacionais moderados, sendo adequado para execução em CPUs com pelo menos 8 GB de RAM ou em GPUs para maior desempenho, garantindo viabilidade em ambientes locais com recursos limitados. Além disso, sua acessibilidade é facilitada pela integração com a biblioteca Hugging Face Transformers, permitindo carregamento e execução local de forma independente, com suporte a otimizações como cache de recursos para maior eficiência operacional.

2. **Integração de LLM com FastAPI no Ambiente Local:**

Execução Local: Utilize um modelo da HuggingFace ou um modelo treinado localmente para realizar tarefas de processamento de linguagem natural (como geração de texto, resumo automático, classificação de sentimento, etc.) e integre-o ao seu backend FastAPI, sem necessidade de conexão com a nuvem.

Rota FastAPI: Implemente uma rota em FastAPI que se conecte ao modelo LLM rodando localmente para processar dados textuais fornecidos pela aplicação. Exemplo de rota:

POST /processar_texto: Rota que recebe um texto enviado pelo usuário e retorna a análise ou processamento realizado pelo modelo de linguagem, como um resumo ou análise de sentimento.

Execução Local do HuggingFace: Baixe o modelo diretamente em seu ambiente de desenvolvimento local e utilize as funções da biblioteca Transformers para carregar e executar o modelo sem depender de serviços externos.

Modelo HuggingFace

```python
# Carregar o modelo HuggingFace local
tokenizer = AutoTokenizer.from_pretrained("t5-small")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-small")

# Função para gerar texto com o modelo
def generate_response(input_text: str) -> str:
    inputs = tokenizer(input_text, return_tensors="pt", max_length=512, truncation=True)
    outputs = model.generate(inputs["input_ids"], max_length=512, num_beams=5, early_stopping=True)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

```python
# Configurar o pipeline do HuggingFace para geração de texto
try:
    tokenizer = AutoTokenizer.from_pretrained("distilgpt2")
    model = AutoModelForCausalLM.from_pretrained("distilgpt2")
    text_generator = pipeline("text-generation", model=model, tokenizer=tokenizer)
    print("Modelo de geração de texto carregado com sucesso.")
except Exception as e:
    print(f"Erro ao carregar o modelo: {e}")
    text_generator = None
```

```python
@app.post("/chat/converse", response_model=ChatResponse)
async def converse(chat_request: ChatRequest):
    """
    Endpoint para responder perguntas específicas com base nos dados do MongoDB.
    """
    try:
        pergunta = chat_request.question.lower().strip()

        # Processar a pergunta com o modelo HuggingFace
        processed_question = generate_response(f"Interprete a pergunta: {pergunta}")

        resposta = ""

        # Normalização para nomes de unidades e procedimentos
        def normalize(text):
            return text.lower().strip()

        # Verificar qual pergunta está sendo feita e buscar os dados correspondentes
        if "procedimentos apresentados" in processed_question:
            unidade_nome = normalize(processed_question.split("na unidade")[-1])
            query = {"FANTASIA": {"$regex": f".*{unidade_nome}.*", "$options": "i"}}
            print(f"Consulta MongoDB: {query}")  # Log da consulta
            dados = list(collection_sia_analise.find(query, {"_id": 0}))
            print(f"Resultados encontrados: {dados}")  # Log dos resultados

            if not dados:
                return ChatResponse(
                    question=chat_request.question,
                    response=f"Nenhuma unidade encontrada com o nome '{unidade_nome}'."
                )

            total_procedimentos = sum(item.get("TOTAL_PA_QTDPRO", 0) for item in dados)
            resposta = f"A unidade '{unidade_nome}' apresentou um total de {total_procedimentos} procedimentos."

        elif "procedimentos aprovados" in processed_question:
            unidade_nome = normalize(processed_question.split("na unidade")[-1])
            query = {"FANTASIA": {"$regex": f".*{unidade_nome}.*", "$options": "i"}}
            print(f"Consulta MongoDB: {query}")  # Log da consulta
            dados = list(collection_sia_analise.find(query, {"_id": 0}))
            print(f"Resultados encontrados: {dados}")  # Log dos resultados

            if not dados:
                return ChatResponse(
                    question=chat_request.question,
                    response=f"Nenhuma unidade encontrada com o nome '{unidade_nome}'."
                )

            total_aprovados = sum(item.get("TOTAL_PA_QTDAPR", 0) for item in dados)
            resposta = f"A unidade '{unidade_nome}' aprovou um total de {total_aprovados} procedimentos."

        elif "procedimentos do tipo" in processed_question and "foram realizados na unidade" in processed_question:
            partes = processed_question.split("foram realizados na unidade")
            procedimento_nome = normalize(partes[0].split("do tipo")[-1])
            unidade_nome = normalize(partes[-1])
            query = {
                "FANTASIA": {"$regex": f".*{unidade_nome}.*", "$options": "i"},
                "IP_DSCR": {"$regex": f".*{procedimento_nome}.*", "$options": "i"}
            }
            print(f"Consulta MongoDB: {query}")  # Log da consulta
            dados = list(collection_sia_analise.find(query, {"_id": 0}))
            print(f"Resultados encontrados: {dados}")  # Log dos resultados

            if not dados:
                return ChatResponse(
                    question=chat_request.question,
                    response=f"Nenhuma unidade ou procedimento encontrado com os critérios fornecidos."
                )

            total_procedimentos = sum(item.get("TOTAL_PA_QTDPRO", 0) for item in dados)
            resposta = (
                f"Foram realizados {total_procedimentos} procedimentos do tipo '{procedimento_nome}' na unidade '{unidade_nome}'."
            )

        elif "historico de procedimentos realizados na unidade" in processed_question:
            unidade_nome = normalize(processed_question.split("na unidade")[-1])
            query = {"FANTASIA": {"$regex": f".*{unidade_nome}.*", "$options": "i"}}
            print(f"Consulta MongoDB: {query}")  # Log da consulta
            dados = list(collection_sia_analise.find(query, {"_id": 0}))
            print(f"Resultados encontrados: {dados}")  # Log dos resultados
```

```
            if not dados:
                return ChatResponse(
                    question=chat_request.question,
                    response=f"Nenhuma unidade encontrada com o nome '{unidade_nome}'."
                )

            historico = {}
            for item in dados:
                ano_mes = item.get("PA_MVMR", "Desconhecido")
                historico[ano_mes] = historico.get(ano_mes, 0) + item.get("TOTAL_PA_QTDPRO", 0)

            resposta = "Histórico de procedimentos realizados:\n"
            for periodo, total in sorted(historico.items()):
                resposta += f"- {periodo}: {total} procedimentos\n"

        else:
            resposta = "Desculpe, não entendi sua pergunta. Por favor, reformule."

        return ChatResponse(
            question=chat_request.question,
            response=resposta
        )

    except Exception as e:
        print(f"Erro no chat: {e}")
        raise HTTPException(status_code=500, detail="Erro interno no servidor.")
```

Get -> Busca nome das unidades através de palavras chaves

```
@app.get("/unidades/", response_model=List[dict])
async def consultar_unidades(pa_coduni: str = Query(..., description="Texto para busca (PA_CODUNI ou parte de FANTASIA)")):
    """
    Consulta unidades de saúde na coleção SIA_ANALISE pelo PA_CODUNI ou parte do nome FANTASIA.
    - Se `pa_coduni` corresponder a parte do PA_CODUNI, retorna todos os registros correspondentes.
    - Se `pa_coduni` corresponder a parte do FANTASIA, retorna todos os registros correspondentes.
    """
    try:
        # Construir o filtro de busca dinâmico
        query_filter = {
            "$or": [
                {"PA_CODUNI": {"$regex": pa_coduni, "$options": "i"}},  # Busca parcial no PA_CODUNI
                {"FANTASIA": {"$regex": pa_coduni, "$options": "i"}}    # Busca parcial no FANTASIA
            ]
        }

        # Buscar dados no MongoDB
        resultados = collection_sia_analise.find(query_filter, {"_id": 0, "PA_CODUNI": 1, "FANTASIA": 1})

        # Garantir unicidade dos resultados
        dados_unicos = list({(res["PA_CODUNI"], res["FANTASIA"]) for res in resultados if "PA_CODUNI" in res and "FANTASIA" in res})

        # Verificar se há registros encontrados
        if not dados_unicos:
            return {"message": "Nenhum registro encontrado para a consulta."}

        # Formatar como lista de dicionários
        resposta_formatada = [{"PA_CODUNI": item[0], "FANTASIA": item[1]} for item in dados_unicos]

        return resposta_formatada

    except Exception as e:
        # Log detalhado do erro para depuração
        print(f"Erro ao consultar unidades: {e}")
        raise HTTPException(status_code=500, detail="Erro interno no servidor.")
```

Post -> Acrescenta novo registro no banco

```python
@app.post("/unidades/")
async def adicionar_unidade(unidade: UnidadeData):
    """
    Endpoint para adicionar manualmente um registro na coleção SIA_ANALISE.
    """
    try:
        # Converter o objeto Pydantic para um dicionário
        unidade_dict = unidade.dict()

        # Inserir o registro no MongoDB
        resultado = collection_sia_analise.insert_one(unidade_dict)

        # Verificar se a inserção foi bem-sucedida
        if not resultado.inserted_id:
            raise HTTPException(status_code=500, detail="Erro ao inserir o registro no banco de dados.")

        return {"message": "Registro inserido com sucesso!", "id": str(resultado.inserted_id)}

```

3. **Manipulação das Respostas da API:**

Response Models: Crie Response Models em FastAPI para garantir que as respostas da API sejam consistentes e estruturadas, utilizando modelos de resposta claros e bem definidos.

Exemplo: A rota /processar_texto deve retornar um JSON estruturado com informações detalhadas, como o texto processado, o tipo de análise realizada (resumo, classificação, etc.) e os resultados gerados.

Validação das Respostas: Assegure-se de que todos os retornos da API estejam validados, garantindo que os dados enviados e recebidos estejam no formato correto e sejam compreensíveis para o cliente.

Foram realizadas modificações em relação ao código apresentado na questão anterior afim de aplicar o Response Models:

```python
@app.post("/chat/converse", response_model=ChatResponse)
async def converse(chat_request: ChatRequest):
    """
    Endpoint para responder perguntas específicas com base nos dados do MongoDB.
    """
    try:
        pergunta = chat_request.question.lower()

        # Processar a pergunta com o modelo HuggingFace
        processed_question = generate_response(f"Interprete a pergunta: {pergunta}")

        print(f"Processed Question: {processed_question}")  # Log para depuração

        # Mapear tipo de análise baseado na pergunta
        analysis_type = ""
        results = []

        if "procedimentos apresentados" in pergunta or "total apresentados" in pergunta:
            analysis_type = "procedimentos_apresentados"
        elif "procedimentos aprovados" in pergunta or "total aprovados" in pergunta:
            analysis_type = "procedimentos_aprovados"
        elif "procedimentos do tipo" in pergunta:
            analysis_type = "procedimentos_tipo"
        elif "histórico de procedimentos" in pergunta:
            analysis_type = "historico_procedimentos"
        else:
            return ChatResponse(
                question=chat_request.question,
                processed_question=ProcessedQuestionResponse(
                    processed_question=processed_question,
                    analysis_type="unknown"
                ),
                results=[]
            )

        # Extrair nome da unidade de saúde
        unidade_match = re.search(r"unidade\s(.+)", pergunta)
        unidade_nome = unidade_match.group(1).strip() if unidade_match else None

        if not unidade_nome:
            return ChatResponse(
                question=chat_request.question,
                processed_question=ProcessedQuestionResponse(
                    processed_question=processed_question,
                    analysis_type=analysis_type
                ),
                results=[]
            )
```

```python

        # Buscar no MongoDB com base no tipo de análise
        query = {"FANTASIA": {"$regex": unidade_nome, "$options": "i"}}
        dados = list(collection_sia_analise.find(query, {"_id": 0}))

        if not dados:
            return ChatResponse(
                question=chat_request.question,
                processed_question=ProcessedQuestionResponse(
                    processed_question=processed_question,
                    analysis_type=analysis_type
                ),
                results=[]
            )

        if analysis_type == "procedimentos_apresentados":
            total_procedimentos = sum(item.get("TOTAL_PA_QTDPRO", 0) for item in dados)
            results.append(QueryResult(unidade=unidade_nome, total=total_procedimentos))
        elif analysis_type == "procedimentos_aprovados":
            total_aprovados = sum(item.get("TOTAL_PA_QTDAPR", 0) for item in dados)
            results.append(QueryResult(unidade=unidade_nome, total=total_aprovados))

        return ChatResponse(
            question=chat_request.question,
            processed_question=ProcessedQuestionResponse(
                processed_question=processed_question,
                analysis_type=analysis_type
            ),
            results=results
        )

    except Exception as e:
        print(f"Erro no chat: {e}")
        raise HTTPException(status_code=500, detail="Erro interno no servidor.")
```

4. **Tratamento Robusto de Erros na API:**

Exceções HTTP: Implemente um tratamento robusto de erros na API, utilizando exceções HTTP específicas em FastAPI para lidar com problemas que possam ocorrer durante as requisições (como falha ao carregar o modelo, problemas com os dados de entrada, ou timeouts).

Exemplo: Se o modelo LLM não estiver carregado corretamente ou ocorrer um erro na manipulação dos dados, a aplicação deve retornar um código de erro adequado (ex: 503 Serviço Indisponível) e uma mensagem explicativa.

Melhoria da Experiência do Usuário: Garanta que os erros sejam tratados de forma clara e informativa, proporcionando feedback útil para o usuário final e facilitando o diagnóstico de possíveis problemas.

Exemplos de tratamento

```
166         except Exception as e:
167             raise HTTPException(
168                 status_code=503,
169                 detail=f"Erro ao processar a pergunta no modelo LLM: {e}"
170             )
```

```
189     try:
190         unidade_match = re.search(r"unidade\s(.+)", pergunta)
191         unidade_nome = unidade_match.group(1).strip() if unidade_match else None
192         if not unidade_nome:
193             raise ValueError("Nome da unidade não encontrado na pergunta.")
194     except Exception as e:
195         raise HTTPException(
196             status_code=400,
197             detail=f"Erro ao identificar a unidade de saúde: {e}"
198         )
```

```
213         except Exception as e:
214             raise HTTPException(
215                 status_code=500,
216                 detail=f"Erro ao consultar o banco de dados: {e}"
217             )
```