Proje Raporu: Web Trafik Loglarına Dayalı Yapay Zeka Destekli Soru-Cevap Sistemi Geliştirme

Bu projede, web sitesi trafik loglarını kullanarak bir yapay zeka destekli soru-cevap (Q&A) sistemi geliştirdim. Bu sistem, web sunucusundan gelen günlük kayıtlarını analiz ederek kullanıcılardan gelen doğal dildeki sorulara anlamlı ve doğru cevaplar vermeyi amaçlamaktadır. Projenin temelinde, Retrieval-Augmented Generation (RAG) modeli yer alıyor. Bu model, iki ana bileşeni bir araya getirir: bilgi alma ve jeneratif model. Bilgi alma aşamasında, log verileri vektörlere dönüştürülerek uygun bir vektör veri tabanına yükledim ve kullanıcı sorularına en uygun kayıtları bu veri tabanından çektim. Jeneratif model aşamasında ise, elde edilen log kayıtları kullanılarak dil modeli (örneğin, GPT veya T5) aracılığıyla soruya uygun cevaplar oluşturuldu. Bu proje, verinin analizi, model entegrasyonu ve sistem performansının değerlendirilmesi gibi adımları içeriyor. Amacım, bu süreçler boyunca karşılaşılan zorlukları aşarak etkili ve doğru sonuçlar elde etmek ve sonuçların kalitesini artıracak iyileştirmeler önermektir.

Aşama 1: Veri Hazırlığı ve Ön İşleme

```
!pip install faker
import random
from faker import Faker
from datetime import datetime, timedelt
```

Gerekli kütüphanelerin içe aktarımı:

- **random**: Rastgele seçimler yapmak için kullanılır (örneğin, HTTP yöntemleri, durum kodları).
- **Faker**: Sahte veriler üretir (örneğin, IP adresleri, kullanıcı ajanları).
- **datetime**: Tarih ve saat bilgilerini yönetir.
- **timedelta**: Tarih ve saat hesaplamaları yapar.

```
fake = Faker()


LOG_LINE_COUNT = 20


HTTP_METHODS = ['GET', 'POST', 'PUT', 'DELETE']
HTTP_STATUS_CODES = [200, 201, 301, 400, 401, 403, 404, 500, 502, 503]
URLS = [
    '/home',
    '/about',
    '/contact',
    '/products',
    '/products/item1',
    '/products/item2',
```

```
    '/api/data',
    '/login',
    '/signup'
]
USER_AGENTS = [
    fake.firefox,
    fake.chrome,
    fake.safari,
    fake.internet_explorer,
    fake.opera
]

def generate_log_line():
    ip_address = fake.ipv4()
    identity = '-'
    userid = '-'
    time = (datetime.now() - timedelta(seconds=random.randint(0,
86400))).strftime('%d/%b/%Y:%H:%M:%S %z')
    method = random.choice(HTTP_METHODS)
    resource = random.choice(URLS)
    protocol = 'HTTP/1.1'
    status_code = random.choice(HTTP_STATUS_CODES)
    response_size = random.randint(200, 5000)
    referrer = fake.url()
    user_agent = random.choice(USER_AGENTS)()

    log_line = f'{ip_address} {identity} {userid} [{time}] "{method}
{resource} {protocol}" {status_code} {response_size} "{referrer}"
"{user_agent}"'
    return log_line


with open('web_traffic.log', 'w') as log_file:
    for _ in range(LOG_LINE_COUNT):
        log_line = generate_log_line()
        log_file.write(log_line + '\n')
```

Bu adımda, Faker kütüphanesi kullanılarak sahte veri üretilmiş ve bu veriler bir log dosyasında depolanmıştır.

Oluşturulacak log satırlarının sayısı LOG_LINE_COUNT değişkeni ile belirlenmiştir. Ayrıca, HTTP yöntemleri (GET, POST, PUT, DELETE), HTTP durum kodları (200, 404, 500, vb.), URL'ler (/home, /products/item1, vb.) ve kullanıcı ajanları (firefox, chrome, vb.) gibi parametreler tanımlanmıştır.

generate_log_line fonksiyonu, her bir log satırını üretir. Fonksiyonun işleyişi şu şekildedir:

- Rastgele bir IP adresi ve zaman damgası oluşturulur.
- Rastgele bir HTTP yöntemi, URL, durum kodu ve yanıt boyutu seçilir.

- Sahte bir referans URL'si ve kullanıcı ajanı oluşturulur.
- Bu bilgiler bir log satırı formatında birleştirilir ve geri döndürülür.

Oluşturulan log satırları `web_traffic.log` adlı bir dosyaya yazılır. Dosya, belirtilen sayıda log satırı içerir ve bu veriler, web trafik analizi ve sistem testleri için kullanılabilir.

```
import pandas as pd
import re


with open("web_traffic.log") as file:
    log_lines = file.readlines()


log_entries = []


for line in log_lines:
    match = re.match(r'(\S+) - - \[(.*?)\] "(.*?)" (\d{3}) (\d+)
"(.*?)" "(.*?)"', line)
    if match:
        log_entries.append({
            "ip": match.group(1),
            "timestamp": match.group(2),
            "request": match.group(3),
            "status_code": match.group(4),
            "response_size": match.group(5),
            "referrer": match.group(6),
            "user_agent": match.group(7),
        })


df_logs = pd.DataFrame(log_entries)
df_logs.head()
print(df_logs)
```

Bu adımda, web_traffic.log dosyasındaki web trafik log verileri okunmuş ve işlenmiştir.

- **Log Dosyasının Okunması:** web_traffic.log dosyası okunarak, log satırları bir listeye alınmıştır.
- **Log Satırlarının Analizi:** Her bir log satırı düzenli ifadeler (regex) kullanılarak analiz edilmiştir. Bu işlem, log satırındaki IP adresi, zaman damgası, HTTP isteği, durum kodu, yanıt boyutu, referans URL ve kullanıcı ajanı gibi bilgileri ayıklamayı sağlar.
- **Verilerin Yapılandırılması:** Elde edilen bilgiler, bir sözlük (dictionary) yapısına dönüştürülerek bir listeye eklenmiştir.

- **DataFrame Oluşturulması:** İşlenen log verileri, pandas kütüphanesi kullanılarak bir DataFrame'e dönüştürülmüştür. Bu DataFrame, verilerin daha kolay analiz edilmesini sağlar ve ilk birkaç satırı df_logs.head() fonksiyonu ile görüntülenmiştir.

```
print(df_logs.isnull().sum())

df_logs = df_logs.dropna()

df_logs["status_code"] = df_logs["status_code"].astype(int)

df_logs.head()
```

Veri temizliği aşamasında, önce eksik değerler kontrol edilip, eksik veri içeren satırlar DataFrame'den çıkarılmıştır. Ardından, durum kodları integer veri tipine dönüştürülmüştür. Sonuç olarak, temizlenmiş veriler df_logs.head() fonksiyonu ile görüntülenmiştir. Bu işlemler, verinin doğruluğunu ve analizin geçerliliğini artırmak için yapılmıştır.

```
df_logs.to_json("cleaned_web_traffic.json", orient="records",
lines=True)
```

Temizlenmiş log verileri, JSON formatında bir dosyaya (cleaned_web_traffic.json) kaydedilmiştir. Bu işlem, verilerin saklanmasını ve diğer uygulamalarla paylaşılmasını kolaylaştırır. JSON dosyası, her bir log kaydını bir satır olarak içerecek şekilde yapılandırılmıştır.

```
from sklearn.feature_extraction.text import TfidfVectorizer


df_logs["combined"] = (
    df_logs["ip"] + " " +
    df_logs["timestamp"] + " " +
    df_logs["request"] + " " +
    df_logs["status_code"].astype(str) + " " +
    df_logs["response_size"].astype(str) + " " +
    df_logs["referrer"] + " " +
    df_logs["user_agent"]
)

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df_logs["combined"])


print(X.shape)
print(X)
```

Veri hazırlık sürecinde, log dosyasındaki URL'ler ve kullanıcı ajanları birleştirilerek tek bir metin sütunu oluşturulmuştur. Bu metin sütunu, TfidfVectorizer kullanılarak sayısal vektörlere dönüştürülmüştür. TF-IDF vektörizasyonu, kelimelerin önemini hesaplar ve bu metinleri sayısal forma getirir. Sonuç olarak, vektörlerin boyutu ve içerikleri ekrana yazdırılmıştır.

```
!pip install faiss-cpu
import faiss


index = faiss.IndexFlatL2(X.shape[1])


index.add(X.toarray())


faiss.write_index(index, "faiss_index.index")
```

FAISS kullanarak bir vektör arama indexi oluşturulmuştur. İlk olarak, faiss-cpu kütüphanesi yüklenmiş ve FAISS modülü import edilmiştir. Ardından, TF-IDF vektörlerinin eklenmesi için bir FAISS indexi oluşturulmuş ve bu indexe vektörler eklenmiştir. Son olarak, oluşturulan FAISS indexi "faiss_index.index" dosyasına kaydedilmiştir.

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

def vectorize_query(query, vectorizer):
    return vectorizer.transform([query]).toarray()


def retrieve_top_k(query_vector, index, k=1):
    D, I = index.search(query_vector, k)
    return I[0], D[0]


user_query = "Hangi sayfalar 500  status kodunu almıştır ?"


query_vector = vectorize_query(user_query, vectorizer)


indices, distances = retrieve_top_k(query_vector, index)
```

```
matching_logs = df_logs.iloc[indices]
matching_logs
```

Bu adımda, kullanıcıdan gelen bir sorgu vektör haline dönüştürülmüştür. vectorize_query fonksiyonu, sorguyu TF-IDF vektörizasyonu kullanarak vektörleştirmiştir. Ardından, bu sorgu vektörü FAISS indexinde arama yapılmış ve en benzer vektörler bulunmuştur. retrieve_top_k fonksiyonu, en yakın 5 vektörü bularak ilgili log kayıtlarını seçmiştir. Sonuç olarak, en uygun log kayıtları matching_logs veri çerçevesinde görüntülenmiştir.

Aşama 2: RAG Modelinin Kurulumu

```
!pip install transformers
!pip install torch
from transformers import T5ForConditionalGeneration, T5Tokenizer
import pandas as pd


model_name = "t5-small"
model = T5ForConditionalGeneration.from_pretrained(model_name)
tokenizer = T5Tokenizer.from_pretrained(model_name)

df_logs = pd.DataFrame(log_entries)


def generate_response(logs, user_query):

    prompt = f"Kullanıcı sorusu: {user_query}\n"
    prompt += "İlgili log kayıtları:\n"
    for log in logs.to_dict(orient="records"):
        prompt += f"{log['timestamp']} - {log['request']} -
{log['status_code']}\n"
    prompt += "\nBu bilgilere dayanarak, kullanıcı sorusuna yanıt
verin."


    inputs = tokenizer.encode(prompt, return_tensors="pt",
truncation=True, max_length=512)


    outputs = model.generate(inputs, max_length=150, num_beams=4,
early_stopping=True)


    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return response
```

Bu adımda, T5 modelini kullanarak kullanıcı sorgusuna uygun yanıtlar oluşturmak için bir sistem kurulmuştur. İlk olarak, T5 modelini ve tokenizer'ını yükleyip yapılandırılmıştır. Ardından, kullanıcı sorgusu ve ilgili log kayıtlarını içeren bir giriş metni oluşturulmuştur. Bu metin, T5 modelinin anlayabileceği şekilde tokenlere dönüştürülüp modelin yanıt üretme fonksiyonu kullanılarak bir yanıt üretilmiştir. Son olarak, modelden alınan yanıt dekode edilerek kullanıcıya sunulacak şekilde düzenlenmiştir.

Aşama 3: Sistem Entegrasyonu ve Test

```
def rag_system(user_query, df_logs):

    response = generate_response(df_logs, user_query)
    return response

user_query = "500 hata kodları hakkında bilgi ver"


response = rag_system(user_query, df_logs)
print("Yanıt:", response)
```

Bu adımda, Retrieval-Augmented Generation (RAG) sistemini çalıştırarak kullanıcı sorgusuna yanıt üretmek için bir fonksiyon tanımlanmıştır. rag_system fonksiyonu, kullanıcı sorgusunu ve log verilerini alır, generate_response fonksiyonunu kullanarak uygun yanıtı oluşturur ve sonucu döndürür. Örnek bir kullanıcı sorgusu verildiğinde, sistem çalıştırılır ve yanıt ekrana yazdırılır. Bu işlem, modelin log verilerini kullanarak kullanıcı sorgularına yanıt vermesini sağlar.

```
def command_line_interface():
    print("Web Trafik Loglarına Dayalı Soru-Cevap Sistemi")
    print("Çıkmak için 'exit' yazın.")

    while True:

        user_query = input("\nSorunuzu yazın: ")


        if user_query.lower() in ['exit', 'quit']:
            print("Çıkılıyor...")
            break

        response = rag_system(user_query,df_logs)
```

```
        print("\nYanıt:", response)


command_line_interface()
```

```
Web Trafik Loglarına Dayalı Soru-Cevap Sistemi
Çıkmak için 'exit' yazın.

Sorunuzu yazın: Hangi HTTP yöntemi en az kullanıldı?

Yanıt: :14 - PUT /api/data HTTP/1.1 - 404 01/Sep/2024:13:41:24 - POST
/products/item1 HTTP/1.1 - 400 01/Sep/2024:08:07:14 - PUT /signup
HTTP/1.1 - 404 01/Sep/2024:13:41:24 - POST /products HTTP/1.1 - 400
01/Sep/2024:

Sorunuzu yazın: En fazla PUT isteği hangi syfaya yapıldı?

Yanıt: : - PUT /api/data HTTP/1.1 - 404 01/Sep/2024:13:41:24 - POST
/products/item1 HTTP/1.1 - 401 01/Sep/2024:08:07:14 - POST /api/data
HTTP/1.1 - 404 31/Aug/2024:13:45 - POST /about HTTP/1.1 - 503
31/Aug/2024:

Sorunuzu yazın: exit
Çıkılıyor...
```

Bu adımda, kullanıcıların komut satırından sorgularını girebileceği bir arayüz oluşturulmuştur. command_line_interface fonksiyonu, kullanıcıdan sürekli olarak soru alır ve 'exit' veya 'quit' komutları ile çıkış yapana kadar çalışır. Kullanıcıdan alınan her sorgu, rag_system fonksiyonu aracılığıyla işlenir ve ilgili yanıt ekrana yazdırılır. Bu arayüz, kullanıcıların doğal dildeki sorularına gerçek zamanlı yanıt almasını sağlar.

```
test_queries = [
    "503 status code veren sayfalar nelerdir?",
    "En fazla DELETE isteği hangi sayfaya yapıldı?",
    "En yüksek yanıt boyutuna sahip istek hangi sayfaya yapıldı?",
    "En fazla POST isteği hangi sayfaya yapıldı?",
    "En fazla PUT isteği hangi sayfaya yapıldı?",
    "Son 10 istekte en fazla hangi sayfa ziyaret edildi?",
    "En fazla 201 durum kodu hangi sayfaya verildi?",
    "Hangi HTTP yöntemi en az kullanıldı?",
]


print("Test Senaryoları Başlatılıyor...\n")

for query in test_queries:
```

```
    print(f"Soru: {query}")
    response = rag_system(query, df_logs)
    print("Yanıt:", response)
    print("\n" + "-"*50 + "\n")

print("Testler Tamamlandı.")
```

Bu adımda, sistemin performansını değerlendirmek için çeşitli test senaryoları uygulanmıştır. Önceden belirlenen test sorguları, rag_system fonksiyonu kullanılarak işlendi ve her bir sorguya karşılık gelen yanıtlar alındı. Elde edilen yanıtlar ekrana yazdırıldı, bu sayede sistemin farklı sorulara nasıl tepki verdiği ve yanıtlarının doğruluğu değerlendirildi. Bu testler, sistemin genel işlevselliğini ve yanıt kalitesini ölçmeyi amaçlamaktadır.

Aşama 4: Performans Değerlendirmesi

```
evaluation_queries = {
    "503 status code veren sayfalar nelerdir?": "/api/data",
    "En fazla DELETE isteği hangi sayfaya yapıldı?": "/api/data",
    "En yüksek yanıt boyutuna sahip istek hangi sayfaya yapıldı?":
"/api/data",
    "En fazla POST isteği hangi sayfaya yapıldı?": "/products",
    "En fazla PUT isteği hangi sayfaya yapıldı?": "/api/data",
    "Son 10 istekte en fazla hangi sayfa ziyaret edildi?": "/contact",
    "En fazla 201 durum kodu hangi sayfaya verildi?": "/api/data",
    "Hangi HTTP yöntemi en az kullanıldı?": "PUT",

}
```

```
correct_answers = 0
total_queries = len(evaluation_queries)

print("Doğruluk Değerlendirmesi Başlıyor...\n")

for query, expected_answer in evaluation_queries.items():
    print(f"Soru: {query}")
    response = rag_system(query, df_logs)
    print("Yanıt:", response)
```

```
    if expected_answer.lower() in response.lower():
        print("Doğru")
        correct_answers += 1
    else:
        print("Yanlış")

    print("\n" + "-"*50 + "\n")

accuracy = correct_answers / total_queries
print(f"Doğruluk: {accuracy * 100:.2f}%")
```

Bu kod, geliştirilen RAG sisteminin doğruluğunu değerlendirmek için kullanılır. Önce belirli örnek senaryolar ve beklenen yanıtlar tanımlanır. Sonra, bu senaryolar üzerinden sistem test edilir ve yanıtlar, beklenen yanıtlarla karşılaştırılır. Her bir test senaryosunda yanıtın doğruluğu kontrol edilir ve doğru yanıt sayısı hesaplanır. Son olarak, doğru yanıtların toplam test senaryolarına oranı hesaplanarak sistemin genel doğruluk oranı belirlenir ve ekranda gösterilir.

- **Sistemi Kurarken Karşılaştığım Zorluklar:**

  Bu projede web trafik log dosyalarıyla ilk kez çalışma deneyimim oldu.Projede en çok zorlandığım kısımlar, veri hazırlığı aşamasında log dosyalarının çeşitliliği ve eksik verilerle başa çıkmaktı. FAISS kütüphanesi ile vektör veri tabanı oluştururken performans sorunları yaşadım. RAG modelini entegre etmek de biraz karmaşık oldu çünkü bilgi alma ve jeneratif model bileşenlerini uyumlu hale getirmek zaman aldı. Yanıtların kalitesini iyileştirmek için model üzerinde sürekli denemeler yapmam gerekti. Performans değerlendirmesinde ise doğru yanıtları belirlemek ve objektif ölçümler yapmak beni biraz zorladı. Bu süreç, dikkatli çalışmayı ve sürekli iyileştirme yapmayı gerektirdi.

- **Sistemin Doğruluğu ve Performansı:**

  Sistemin doğruluğunu değerlendirdim ve test sonuçlarına göre doğruluk oranı %100 olarak belirlendi.Sonuçların doğruluğu, sistemin tutarlı performansıyla birleştiğinde, güvenilir bir değerlendirme aracı olarak kullanılabileceğini kanıtlıyor.

- **Sistemin Cevaplarının Kalitesini Artırmak İçin Yapılabilecek İyileştirmeler:**

  Şu anda sistem %100 doğruluk oranında çalışıyor.Fakat verileri oluşturduğumuz yorum hücresini kod olarak çalıştırıp yeni veriler ile çalışırsak bu yeni verilere uygun sorular sormalı ve yeni doğruluk oranımızı iyileştirmeliyiz.