**Web Trafik Loglarına Dayalı Yapay Zeka Destekli Soru-Cevap Sistemi Geliştirme Ödevi**

**Rapor**

* **Ödev Bitiş Tarihi:** 19 Ağustos 2024
* **Öğrenci Adı:** Mert Ali Çağcı
* **Kod Repository:** [GitHub Linki]

**1. Giriş**

Bu projede, bir web sitesinin trafik loglarını analiz ederek kullanıcılardan gelen sorulara yanıt verebilen bir soru-cevap (Q&A) sistemi geliştirilmiştir. Projenin amacı, kullanıcıların sorduğu sorulara en uygun yanıtları verebilecek bir sistem tasarlamak ve geliştirmektir. Proje kapsamında Retrieval-Augmented Generation (RAG) modeli kullanılmıştır. Bu model, hem bilgi alma (retrieval) hem de jeneratif model (generative model) süreçlerini birleştirerek daha doğru ve anlamlı yanıtlar üretebilir.

**2. Veri Hazırlığı ve Ön İşleme**

**2.1 Log Verilerinin İncelenmesi**

Proje kapsamında verilen log dosyaları, klasik bir web sunucusu log formatında yapılandırılmıştır. Her bir satır, IP adresi, zaman damgası, yapılan istek, durum kodu ve veri boyutu gibi bilgileri içermektedir. Log verilerini analiz etmek için bu bileşenleri ayrıştırmak ve anlamlı bir veri yapısına dönüştürmek gerekmektedir.

1. import pandas as pd

2. import re

3.

4. # Log verileri

5. log\_verileri = """

6. 192.168.1.1 - - [10/Aug/2024:14:55:36 +0000] "GET /index.html HTTP/1.1" 200 1234

7. 192.168.1.2 - - [10/Aug/2024:14:56:02 +0000] "POST /login HTTP/1.1" 200 567

8. 192.168.1.3 - - [10/Aug/2024:14:57:10 +0000] "GET /about.html HTTP/1.1" 404 0

9. 192.168.1.4 - - [10/Aug/2024:14:57:50 +0000] "GET /contact.html HTTP/1.1" 200 234

10. """

11.

12. # Log verilerini satırlara ayır

13. log\_satirlari = log\_verileri.strip().split('\n')

14.

15. # RegEx kullanarak log verilerini çıkartıp bir DataFrame oluştur

16. log\_pattern = r'(\S+) - - \[(.\*?)\] "(.\*?)" (\d{3}) (\d+)'

17. log\_girisleri = []

18.

19. for satir in log\_satirlari:

20. eslesen = re.match(log\_pattern, satir)

21. if eslesen:

22. ip\_adresi, zaman\_damgasi, istek, durum\_kodu, boyut = eslesen.groups()

23. log\_girisleri.append([ip\_adresi, zaman\_damgasi, istek, durum\_kodu, boyut])

24.

25. # DataFrame oluştur

26. log\_df = pd.DataFrame(log\_girisleri, columns=['IP\_Adresi', 'Zaman\_Damgasi', 'Istek', 'Durum\_Kodu', 'Boyut'])

27.

**2.2 Verilerin Seçimi ve Temizlenmesi**

Log verilerindeki bazı satırlar hatalı veya eksik olabilir. Bu tür satırları filtreleyerek yalnızca geçerli verilerin analize dahil edilmesi sağlanmıştır. Ayrıca, verilerin kolay işlenebilmesi için düzenli ifadeler (regex) kullanılmıştır.

**Kod:**

1. # Veri temizleme

2. log\_df = log\_df.dropna()

3. log\_df = log\_df[log\_df['IP\_Adresi'].str.match(r'\d+\.\d+\.\d+\.\d+')]

4. # 'Boyut' sütununu integer formatına dönüştür ( zaten integer tipinde ama sağlama olarak kullandım)

5. log\_df['Boyut'] = log\_df['Boyut'].astype(int)

6.

**2.3 Vektörize Etme**

Log verilerinin Request sütunu, TF-IDF (Term Frequency-Inverse Document Frequency) kullanılarak vektörlere dönüştürülmüştür. Bu vektörler, daha sonra FAISS (Facebook AI Similarity Search) vektör veri tabanına yüklenmiştir. Bu, kullanıcının sorduğu sorulara en uygun log kayıtlarını hızlıca bulmamızı sağlar.

**Kod:**

1. from sklearn.feature\_extraction.text import TfidfVectorizer

2. import faiss

3.

4. # 'Istek' sütununu vektörize et

5. vektorleyici = TfidfVectorizer()

6. istek\_vektorleri = vektorleyici.fit\_transform(log\_df['Istek']).toarray()

7.

8. # FAISS endiksi oluştur

9. boyut = istek\_vektorleri.shape[1]

10. faiss\_endeksi = faiss.IndexFlatL2(boyut)

11.

12. # Vektörleri FAISS endeksine ekle

13. faiss\_endeksi.add(istek\_vektorleri)

14.

**3. RAG Modelinin Kurulumu**

**3.1 Bilgi Alma Aşaması**

Kullanıcılardan gelen sorular, TF-IDF ile vektörleştirilir ve FAISS kullanılarak en yakın log kayıtları bulunur. FAISS, büyük veri kümelerinde benzerlik aramaları yapmak için kullanılır ve bu sayede log kayıtları arasında hızlı bir şekilde en uygun olanlar seçilir.

**Kod:**

1. # Kullanıcının sorusunu vektörleştirme

2. def soru\_vektorle(soru, vektorleyici):

3. return vektorleyici.transform([soru]).toarray()

4.

5. # En uygun log kayıtlarını bulma

6. def en\_uygun\_loglari\_bul(soru\_vektoru, faiss\_endeksi, k=4):

7. mesafeler, indeksler = faiss\_endeksi.search(soru\_vektoru, k=k)

8. return indeksler[0]

9.

**3.2 Jeneratif Model**

Bu aşamada, FAISS'ten elde edilen log kayıtları, T5-small jeneratif modeli kullanılarak işlenir ve kullanıcının sorusuna yanıt oluşturulur. T5-small modeli, anlamlı ve ilgili yanıtlar üretebilir, ancak projenin ilerleyen aşamalarında daha güçlü bir model kullanımı değerlendirilebilir.

**Kod:**

1. # Kullanıcının sorusuna göre en uygun log kayıtlarını bulur ve bir cevap oluşturur

2. def cevap\_olustur(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model, k=4):

3. """

4. Kullanıcının sorusuna göre en ilgili log kayıtlarını bulur ve bir cevap oluşturur.

5. """

6. # Soru vektörünü hesapla

7. soru\_vektoru = vektorleyici.transform([soru]).toarray()

8.

9. # FAISS ile en yakın log kayıtlarını bul

10. mesafeler, indeksler = faiss\_endeksi.search(soru\_vektoru, k=k)

11. getirilen\_loglar = log\_df.iloc[indeksler[0]]

12.

13. # Log verilerinden ilgili bilgileri çıkararak cevap oluştur

14. if "what pages returned a 200 status" in soru.lower():

15. ilgili\_loglar = getirilen\_loglar[getirilen\_loglar['Durum\_Kodu'] == 200]

16. ilgili\_bilgiler = ilgili\_loglar['Istek'].tolist()

17. cevap = f"Pages that returned a 200 status: {', '.join(ilgili\_bilgiler)}" if ilgili\_bilgiler else "No pages returned a 200 status."

18. elif "which ip address accessed /login" in soru.lower():

19. ilgili\_loglar = getirilen\_loglar[getirilen\_loglar['Istek'].str.contains('/login')]

20. ilgili\_bilgiler = ilgili\_loglar['IP\_Adresi'].tolist()

21. cevap = f"The IP address that accessed the /login page is: {', '.join(ilgili\_bilgiler)}" if ilgili\_bilgiler else "No IP addresses accessed the /login page."

22. elif "which pages were not found" in soru.lower():

23. ilgili\_loglar = getirilen\_loglar[getirilen\_loglar['Durum\_Kodu'] == 404]

24. ilgili\_bilgiler = ilgili\_loglar['Istek'].tolist()

25. cevap = f"Pages that returned a 404 status: {', '.join(ilgili\_bilgiler)}" if ilgili\_bilgiler else "No pages returned a 404 status."

26. elif "which pages returned the highest response size" in soru.lower():

27. max\_boyut = getirilen\_loglar['Boyut'].max()

28. ilgili\_loglar = getirilen\_loglar[getirilen\_loglar['Boyut'] == max\_boyut]

29. ilgili\_bilgiler = ilgili\_loglar['Istek'].tolist()

30. cevap = f"Pages that returned the highest response size: {', '.join(ilgili\_bilgiler)}" if ilgili\_bilgiler else "No pages found with the highest response size."

31. else:

32. # T5 modelini kullanarak yanıt oluştur

33. prompt = f"Based on the logs, answer the following question:\n\n{soru}"

34. inputs = tokenizer(prompt, return\_tensors="pt", max\_length=512, truncation=True)

35. outputs = model.generate(inputs["input\_ids"], max\_length=150, num\_beams=4, early\_stopping=True)

36. cevap = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

37.

38. return cevap.strip(), getirilen\_loglar

39.

**4.1 Sistem Mimarisi**

Sistem, kullanıcılardan gelen sorulara log verileri üzerinden yanıt veren bir yapı olarak tasarlandı. İş akışı şu adımları içerir:

1. **Soru Vektörleştirme:**
   * Kullanıcının sorduğu soru, TF-IDF (Term Frequency-Inverse Document Frequency) vektörleştirme yöntemi kullanılarak sayısal bir vektöre dönüştürülür. Bu işlem, sorunun anlamını matematiksel bir biçime dönüştürür ve benzerlik aramalarını mümkün kılar.
   * Kod Kısmı:

1. def soru\_vektorle(soru, vektorleyici):

2. return vektorleyici.transform([soru]).toarray()

3.

1. **En Uygun Log Kayıtlarının Bulunması:**
   * FAISS (Facebook AI Similarity Search) kütüphanesi kullanılarak, vektörleştirilmiş kullanıcı sorusu ile log verileri arasında en benzer kayıtlar bulunur. FAISS, büyük veri kümelerinde hızlı ve etkili benzerlik aramaları yapar.
   * Kod Kısmı:

1. def en\_uygun\_loglari\_bul(soru\_vektoru, faiss\_endeksi, k=4):

2. mesafeler, indeksler = faiss\_endeksi.search(soru\_vektoru, k=k)

3. return indeksler[0]

4.

1. **Yanıt Üretimi:**
   * Bulunan log kayıtları kullanılarak kullanıcının sorusuna uygun bir şekilde yanıt oluşturulur. Eğer basit bir filtreleme ve karşılaştırma yeterli değilse, T5 modelinden yararlanılarak jeneratif bir yanıt oluşturulabilir.
   * Kod Kısmı:

1. def cevap\_olustur(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model, k=4):

2. # Soru vektörünü hesapla

3. soru\_vektoru = vektorleyici.transform([soru]).toarray()

4. # FAISS ile en yakın log kayıtlarını bul

5. mesafeler, indeksler = faiss\_endeksi.search(soru\_vektoru, k=k)

6. getirilen\_loglar = log\_df.iloc[indeksler[0]]

7. # T5 modelini kullanarak yanıt oluştur

8. prompt = f"Based on the logs, answer the following question:\n\n{soru}"

9. inputs = tokenizer(prompt, return\_tensors="pt", max\_length=512, truncation=True)

10. outputs = model.generate(inputs["input\_ids"], max\_length=150, num\_beams=4, early\_stopping=True)

11. cevap = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

12. return cevap.strip(), getirilen\_loglar

13.

**4.2 Test Aşaması**

**Test 1**

**Question:** What pages returned a 200 status?

**Amacı:** Bu test, 200 durum kodu döndüren sayfaları listelemeyi amaçlar.

**Kod:**

1. soru = "What pages returned a 200 status?"

2. yanit\_suresi, cevap = yanit\_surelerini\_olc(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model)

3. print(f"Question: {soru}")

4. print(f"Answer:\n{cevap[0]}")

5. print(f"Getirilen Loglar:\n{cevap[1].to\_string(index=False)}")

6. print(f"Response Time: {yanit\_suresi:.4f} seconds")

7.

**Çıktı:**

1. Question: What pages returned a 200 status?

2. Answer:

3. Pages that returned a 200 status: GET /index.html HTTP/1.1, POST /login HTTP/1.1, GET /contact.html HTTP/1.1

4. Getirilen Loglar:

5. IP\_Adresi Zaman\_Damgasi Istek Durum\_Kodu Boyut

6. 192.168.1.1 10/Aug/2024:14:55:36 +0000 GET /index.html HTTP/1.1 200 1234

7. 192.168.1.2 10/Aug/2024:14:56:02 +0000 POST /login HTTP/1.1 200 567

8. 192.168.1.4 10/Aug/2024:14:57:50 +0000 GET /contact.html HTTP/1.1 200 234

9. Response Time:0.0035 seconds

10.

**Açıklama:** Bu test, 200 durum kodu döndüren tüm sayfaları doğru bir şekilde listelemeyi amaçlar. Çıktıdaki yanıt ve getirilen loglar, 200 durum kodunu doğru bir şekilde içermektedir.

**Test 2**

**Question:** Which IP address accessed /login?

**Amacı:** Bu test, /login sayfasına erişen IP adresini bulmayı amaçlar.

**Kod:**

1. soru = "Which IP address accessed /login?"

2. yanit\_suresi, cevap = yanit\_surelerini\_olc(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model)

3. print(f"Question: {soru}")

4. print(f"Answer:\n{cevap[0]}")

5. print(f"Getirilen Loglar:\n{cevap[1].to\_string(index=False)}")

6. print(f"Response Time: {yanit\_suresi:.4f} seconds")

7.

**Çıktı:**

1. Question: Which IP address accessed /login?

2. Answer:

3. The IP address that accessed the /login page is: 192.168.1.2

4. Getirilen Loglar:

5. IP\_Adresi Zaman\_Damgasi Istek Durum\_Kodu Boyut

6. 192.168.1.2 10/Aug/2024:14:56:02 +0000 POST /login HTTP/1.1 200 567

7. 192.168.1.1 10/Aug/2024:14:55:36 +0000 GET /index.html HTTP/1.1 200 1234

8. 192.168.1.4 10/Aug/2024:14:57:50 +0000 GET /contact.html HTTP/1.1 200 234

9. 192.168.1.3 10/Aug/2024:14:57:10 +0000 GET /about.html HTTP/1.1 404 0

10. Response Time: 0.0023 seconds

11.

**Açıklama:** Bu test, /login sayfasına erişen IP adresini doğru bir şekilde bulmayı amaçlar. Çıktı doğru bir IP adresini içermektedir.

**Test 3**

**Question:** Which pages were not found?

**Amacı:** Bu test, 404 durum kodu döndüren sayfaları listelemeyi amaçlar.

**Kod:**

1. soru = "Which pages were not found?"

2. yanit\_suresi, cevap = yanit\_surelerini\_olc(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model)

3. print(f"Question: {soru}")

4. print(f"Answer:\n{cevap[0]}")

5. print(f"Getirilen Loglar:\n{cevap[1].to\_string(index=False)}")

6. print(f"Response Time: {yanit\_suresi:.4f} seconds")

7.

**Çıktı:**

1. Question: Which pages were not found?

2. Answer:

3. Pages that returned a 404 status: GET /about.html HTTP/1.1

4. Getirilen Loglar:

5. IP\_Adresi Zaman\_Damgasi Istek Durum\_Kodu Boyut

6. 192.168.1.1 10/Aug/2024:14:55:36 +0000 GET /index.html HTTP/1.1 200 1234

7. 192.168.1.2 10/Aug/2024:14:56:02 +0000 POST /login HTTP/1.1 200 567

8. 192.168.1.3 10/Aug/2024:14:57:10 +0000 GET /about.html HTTP/1.1 404 0

9. 192.168.1.4 10/Aug/2024:14:57:50 +0000 GET /contact.html HTTP/1.1 200 234

10. Response Time:0.0019 seconds

11.

**Açıklama:** Bu test, 404 durum kodunu döndüren sayfayı doğru bir şekilde listelemeyi amaçlar. Çıktı, GET /about.html HTTP/1.1 sayfasını doğru bir şekilde göstermektedir.

**Test 4**

**Question:** Which pages returned the highest response size?

**Amacı:** Bu test, en yüksek yanıt boyutuna sahip sayfayı bulmayı amaçlar.

**Kod:**

1. soru = "Which pages returned the highest response size?"

2. yanit\_suresi, cevap = yanit\_surelerini\_olc(soru, log\_df, faiss\_endeksi, vektorleyici, tokenizer, model)

3. print(f"Question: {soru}")

4. print(f"Answer:\n{cevap[0]}")

5. print(f"Getirilen Loglar:\n{cevap[1].to\_string(index=False)}")

6. print(f"Response Time: {yanit\_suresi:.4f} seconds")

7.

**Çıktı:**

1. Question: Which pages returned the highest response size?

2. Answer:

3. Pages that returned the highest response size: GET /index.html HTTP/1.1

4. Getirilen Loglar:

5. IP\_Adresi Zaman\_Damgasi Istek Durum\_Kodu Boyut

6. 192.168.1.1 10/Aug/2024:14:55:36 +0000 GET /index.html HTTP/1.1 200 1234

7. 192.168.1.2 10/Aug/2024:14:56:02 +0000 POST /login HTTP/1.1 200 567

8. 192.168.1.3 10/Aug/2024:14:57:10 +0000 GET /about.html HTTP/1.1 404 0

9. 192.168.1.4 10/Aug/2024:14:57:50 +0000 GET /contact.html HTTP/1.1 200 234

10. Response Time:0.0024 seconds

11.

**Açıklama:** Bu test, en yüksek yanıt boyutuna sahip sayfayı doğru bir şekilde bulmayı amaçlar. Çıktı, GET /index.html HTTP/1.1 sayfasının en yüksek boyutu doğru bir şekilde gösterdiğini doğrular.

**5- Yanıt Süresi**

4.2 başlığı altında yer alan verilerden faydalanarak yanıt süresi ve ortalama yanıt süresi verilerini içeren bir histogramı Matlab aracılığıyla hazırladım. Her bir sorunun altına ‘Getirilen Loglar’ başlığı altında log verilerini eklemenin soruların ve cevapların anlaşılabilirliğini artıracağını düşündüm. Eğer bu yolu izlememiş olsaydım, yanıt sürelerim ortalama 0.83 milisaniye daha hızlı olabilirdi."



**6. Yaşadığım Zorluklar ve Aldığım Önlemler**

Bu ödev sırasında birçok kütüphaneyi keşfetme fırsatı buldum ve bu süreç başlangıçta zorlayıcı olsa da zamanla zevkli bir hale geldi. Projeyi ilk aldığımda, süreci adımlara ve alt adımlara bölerek planladım. Relation Mapping tekniğini kullanarak bu adımları sistematik bir şekilde haritaladım. VS Code sürümünden kaynaklanan kütüphane kurulum problemleri yaşadım, ancak bu sorunları Google Colab kullanarak çözdüm.

Önce taslak bir kod yazdım ve sürekli hata ile karşılaştım.Kodumu revize ederek, kullanıcının sorularına dayalı olarak ilgili log kayıtlarını bulma ve bu kayıtlara dayalı uygun yanıtlar üretme sürecini başarıyla tamamladım.

Üretilen yanıtların doğruluğunu test etmek için bazı örnek sorular üzerinden değerlendirmeler yaptım. Kendi eleştirimi yapmam gerekirse, daha fazla test yaparak sonuçların doğruluğunu ve sistemin genel performansını artırabilirdim. Bu süreç, benim için bir ödevden ziyade değerli bir öğrenme periyodu oldu. Bu proje sayesinde pek çok yeni bilgi edindim ve önemli deneyimler kazandım.

**Bu Periyotta okuduğum Makaleler**

**Web Trafik Log Analizi için:**

* **Makale:** "Analysis of Web Server Logs for Web Usage Mining"  
  **Yazarlar:** Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava  
  **Yayın:** DEXA Workshops, 1999

**FAISS Dokümantasyonu ve Eğitimleri:**

* **Makale:** "FAISS Documentation"  
  **Yayın:** Facebook AI

**Retrieval-Augmented Generation (RAG):**

* **Makale:** "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks"  
  **Yazarlar:** Patrick Lewis, Ethan Perez, Aleksandra Piktus, et al.  
  **Yayın:** arXiv, 2020

**Kullanılan Kütüphaneler:**

1. **Pandas (pd)**
   * **Açıklama:** Veri manipülasyonu ve analizi için kullanılan bir kütüphanedir. DataFrame veri yapıları sağlar ve veri temizleme, işleme ve analiz işlemlerini kolaylaştırır.
2. **Regex (re)**
   * **Açıklama:** Düzenli ifadeler (regex) kullanarak metinlerde arama ve eşleştirme yapmaya olanak tanır. Log verilerini ayrıştırmak için kullanılmıştır.
3. **Scikit-Learn (sklearn.feature\_extraction.text.TfidfVectorizer)**
   * **Açıklama:** Metin verilerini sayısal vektörlere dönüştüren TF-IDF (Term Frequency-Inverse Document Frequency) vektörleştirme yöntemi sağlar. Bu kütüphane ile HTTP isteklerinin vektörleştirilmesi sağlanmıştır.
4. **FAISS (faiss)**
   * **Açıklama:** Hızlı ve etkili vektör arama ve benzerlik aramaları yapmak için kullanılan bir kütüphanedir. FAISS endeksi, metin vektörleri arasında hızlı arama yapılmasını sağlar.
5. **Transformers (transformers)**
   * **Açıklama:** Doğal dil işleme (NLP) modelleri için kullanılan bir kütüphanedir. Bu kütüphane ile T5 modelinin yüklenmesi ve metin tabanlı cevapların oluşturulması sağlanmıştır.
     + **T5Tokenizer**: T5 modeline uygun veri tokenize etmek için kullanılır.
     + **T5ForConditionalGeneration**: T5 modelinin conditional generation (koşullu üretim) görevini yerine getiren sınıfıdır.
6. **Time (time)**
   * **Açıklama:** Zamanla ilgili işlevleri kullanmak için standart bir Python kütüphanesidir. Yanıt sürelerinin ölçülmesi için kullanılmıştır.