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

ÖZET

VERİ HAZIRLIĞI VE ÖN İŞLEME

Veri seti, HAR dosyasından preprocess.py kodu ile gerekli veriler çekilerek hazırlanmıştır. HAR dosyaları, tarayıcıda yapılan tüm web isteklerini JSON arşiv dosyası formatında kaydeder. HAR dosyası ile web sitesinin performansını, güvenliğini ve kullanıcının loglarını kontrol edebiliriz. Bu sebeple, örnek bir veri seti oluşturmak amacıyla siteye istekler gönderilerek kaydedilen HAR dosyası kullanılmıştır. Oluşturulan preprocess.py kodu ile sadece dosya yolu değiştirilerek kolayca HAR dosyasından istenilen veriler çekilebilmektedir. Bu sayede kod, sonraki çalışmalar için kullanılabilir bir hale getirilmiştir.

Yöntem

1) HAR Dosyasını Yükleme

- Chrome tarayışına girmek istediğiniz web sitesini yazın.
- Web sitesinin herhangi bir yerine sağ tık yaparak incele kısmına tıklayarak Chrome Geliştirici Araçlarını açın.
- Sekmelerden ağ sekmesini seçin.
- Sağ tık yapıp “HAR dosyasını kaydet”i seçin.

2) CSV Dosyasını Oluşturma

HAR dosyası, **‘json.load()’** metodu ile yüklendi ve çekilmek istenen veriler **‘labels’** listesinde belirlendi. **‘labels’**, CSV dosyasındaki sütunları temsil etmektedir. Çekilecek veriler; IP adresi, HTTP metodu, URL, tarih, konum, sayfa ID’si, sayfa başlığı, HTTP durumu, durumun açıklama metni, yanıt boyutu, istek süresi ve zamanlama verilerinden oluşmaktadır. Verilerin kaydedileceği dosya ‘**open’** metodu ile açılmıştır. HAR dosyasındaki istek bilgileri **‘entries’** öğesinde, sayfa ile ilgili bilgiler ise **‘pages’** öğesinde tutulmaktadır. Gerekli veriler çekildikten sonra bütün veriler ‘**row**’ adlı bir listeye eklenir ve **‘writerow’** metodu ile CSV dosyasında satırlar oluşturulur.

RAG MODELİNİN KURULUMU

RAG (Retrieval Augmented Generation) modeli kullanarak modeli finetune etmeden veri setinden arama yapılmasına ve dil modeli ile verilerin analiz edilip soru cevap sistemlerinin geliştirilmesine olanak sağlar. Dolayısıyla, soru bu cevap sisteminin geliştirilmesinde RAG modeli kullanılmıştır.

Kullanılan Kütüphaneler

- **Langchain = 0.2.14:** Büyük dil modeli tabanlı soru cevap sistemini oluşturmak için kullanıldı.
- **LangChain Community = 0.2.12:** Topluluk tarafından sağlanan eklentiler.
- **Chroma = 0.5.5:** Vektör veri tabanı olarak ChromaDB kullanıldı.
- **Ollama:** Llama 3.1 modelini kullanmak için gerekli olan araç.
- **os, threading, subprocess:** Ollama kurulumunda kullanıldı.

Kurulumlar

- **Langchain kurulumu:** pip install -q --upgrade langchain
- **Langchain Community kurulumu:** pip install -q --upgrade langchain_community
- **ChromaDB kurulumu:** pip install chromadb
- **Ollama kurulumu:**

  1) Komutlar:

  ```
  sudo apt-get install -y pciutils
  curl https://ollama.ai/install.sh | sh
  ```

  2) **Ollama sunucusunu çalıştırma:**

  - Ollama sunucusunu başlatmak için gerekli ortam değişkenlerini ayarlayın:

    ```
    def ollama():
        os.environ['OLLAMA_HOST'] = '0.0.0.0:11434'
        os.environ['OLLAMA_ORIGINS'] = '*'
        subprocess.Popen(["ollama", "serve"])
    ```

  - Ollama sunucusunu başlatın:

    ```
    ollama_thread = threading.Thread(target=ollama)
    ollama_thread.start()
    ```

  - Ollama modelini çalıştırın:

    ```
    ollama run llama3.1:8b
    ```

  - Tekrardan Ollama sunucusunu başlatın:

    ```
    ollama_thread = threading.Thread(target=ollama)
    ollama_thread.start()
    ```

Not: '-q' komutu kurulum sırasında error dışında çıktı verilmemesi için kullanıldı.

RAG Geliştirimi

1) Veri Yükleme

- **Method:** load_csv()

  Args: file_path (string)

  Veri setinin dosya yolu.

  Çıktı: CSV dosyasındaki veriler

  Açıklama: 'CSVLoader' metodu kullanılarak CSV dosyası yüklendi ve yüklenen veriler çıktı olarak alındı.

2) Prompt Hazırlığı

- **Method:** prompt_func()

  Args: log_data:

  CSV dosyasından alınan log verileri.

  Çıktı: 'ChatPromptTemplate' tipinde bir nesne çıktı olarak alınır. Bu nesne, şablonun **'log_data'** ile doldurulmuş bir halini içerir.

  Açıklama: 'prompt_template_str' ile dil modeline gönderilecek prompt'un şablonu belirlenmiştir. Modele gönderilen prompt sayesinde model, veri setinin de genel bir analizini yapmaktadır. Bu şablon, dil modelinin log verileri ile ilgili bazı anahtar bilgilere odaklanması sağlamıştır:

  - En çok erişilen URL'ler
  - Olağandışı web trafiği ve artışlar
  - Yaygın durum kodları ve hatalar
  - Sık kullanılan IP adresleri
  - Performans darboğazları
  - Zamana dayalı trafik dağılımı

3) Retrieval Aşaması

- **Method:** get_vectorstore()

  Args: data:

  Veri tabanında depolanacak olan veri.

  embedding_model:

  Verileri gömmelere çevirecek olan model.

  Çıktı: Oluşturulan veri tabanı

  Açıklama: Veri setindeki verileri vektör veri tabanında saklanması için önce bir gömme modeli yardımıyla gömmelere çevrilir ve veri tabanında depolanır. Veri tabanı **'Chroma'** kullanılarak oluşturulur ve veriler gömmelere çevrilir.

‘add_documents()’ metodu yardımıyla gömmelere çevrilen veriler veri tabanında depolanır.

- **Method:** retrieve_queries()
  Args: vectorstore:
  Vektör veri tabanı.
  llm:
  Dil modeli.
  prompt:
  prompt_func() metodunu döndürdüğü prompt.
  query:
  Kullanıcının sorgusu.
  Açıklama: ‘as_retriever()’ metodu ile retriever oluşturulur ve veri tabanından gerekli bilgileri çekmek için kullanılır. **‘LLMChain’** ile dil modeli ve prompt kullanılarak bir zincir oluşturulur. Bu zincir, **‘MultiQueryRetriever’’** a gönderilerek çoklu sorgu oluşturma işlemi gerçekleştirilir. **‘invoke’** metodu ile çoklu sorgular oluşturulur ve dil modeline gönderilmek üzere döndürülür.

4) Generation Aşaması

- **Method:** rag_chain()
  Args: multi_query:
  retrieve_queries() metodundan alınan çoklu sorgula.
  prompt:
  prompt_func() metodunu döndürdüğü prompt.
  llm:
  Dil modeli.
  query:
  Kullanıcının sorgusu.
  Açıklama: Bir RAG zinciri oluşturularak yöntemleri zincirler ve kullanıcının sorgusuna cevap verir. **‘RunnableLambda’** sınıfı, lamda kullanılarak çalıştırılabilir bir runnable nesnesi oluşturur ve multi_query’yi döndürür. **‘RunnableParallel’** birden fazla çalıştırılabilir nesneyi paralel olarak çalıştırabilen bir nesne oluşturur. **‘rag_chain’** de bu aşamaları birleştirerek bir zincir oluşturur ve işlem adımlarını sırayla uygular.

TESTLER VE PERFORMANS DEĞERLENDİRMESİ

RAG modelini geliştirilirken projenin son versiyonuna kadar farklı teknikler denendi. İlk önce Multi Query Retriever yerine tek sorgulu retrieval yöntemi denedi fakat model, Multi Query Retriever yöntemi ile daha detaylı yanıtlar üretti. Daha sonra, Few Shot Prompting yöntemi denendi fakat bu yöntem ile model, **‘examples’** olarak gönderilen prompt’a çok bağlı kalarak

veri setini dikkate almadı ve **'prompt_template_str'** ile belirlenen içeriklere ve şablona göre cevap üretmedi. Ayrıca, Self Query Retriever yöntemi de denendi. Self Query Retriever, kullanıcının sorgusundan filtreler çıkarır ve bunları depolanan verilerin meta verilerinde kullanılır. Böylece, modelin sorguya daha uygun bir cevap üretmesi beklenir. Fakat, modelin ürettiği cevap tam tersine çok yüzeysel kaldı ve 'prompt_template_str'te belirtilen içerikleri dikkate almadan yanıt üretti. Ayrıca, **'prompt_template_str'i** JSON formatında çıktı vermesi için yeniden düzenlendi çünkü Llama3.1 Self Query Retriever için gerekli çıktıyı üretemiyordu fakat araştırılına göre bu problem OpenAI kullananlarda yaşanmamıştı. Dolayısıyla, Self Query Retriever yöntemi de kullanılmadı.

Modele gönderilen sorular ve cevaplar incelendiğinde, verilen talimatlara göre veri setini analiz ederek uygun yanıtlar verdiği ve modele gönderilen prompt ile veri setinin genelini analiz edebildiği gözlemlenmiştir. Modele gönderilen sorguya göre model, o sorgu ile alakalı anahtar noktalara daha fazla dikkat etmiştir. Fakat yine de yanlış veya eksik cevaplar da verebilmektedir. Mesela, Soru-Cevap kısmında bulunan 3. Soruda zamana bağlı trafik dağılımını cevaplamamıştır.

KARŞILAŞILAN ZORLUKLAR

1) Veri Setinin Oluşturulması

Veri seti oluşturulurken web trafik loglarını oluşturabileceğim araçlar aradım. Bunun sonucunda HAR dosyasını kullanarak bu loglara ulaşabileceğimi öğrendim ve HAR dosyasından veri ayıklayabilmek için preprocess.py kodunu yazdım.

2) Retriever Aşaması

Retriever aşamasında farklı yöntemler denedim ancak entegrasyonunda sıkıntılar yaşadım ve beklediğim çıktıları alamadım. Sonuç olarak, Multi Query Retriever yöntemini tercih ettim.

3) Prompt Hazırlığı

Dil modelinden beklenen çıktıyı alana kadar farklı prompt tekniklerini ve şablonlarını denedim. Dolayısıyla, doğru prompt yöntemini bulmak zaman alıcıydı.

SORU - CEVAP

1) What are the HTTP status codes returned, and what do they indicate about the success of the requests?

Here are the key points extracted from the log data:

Top Accessed URLs

https://devicebind.ebay.com/signin/sub/tt.html (multiple instances)

https://www.ebay.com/ (multiple instances)

https://sr-client-cfg.amplitude.com/config (1 instance)

Unusual Traffic Patterns or Spikes

A spike in traffic to https://devicebind.ebay.com/signin/sub/tt.html at 15:49:10 GMT

A high response size of 684.0040000388399 for the same URL

Common Status Codes and Errors

200 OK (multiple instances)

No errors or failures reported

Frequent IP Addresses

66.211.166.16 (2 instances)

18.165.183.61 (1 instance)

23.49.235.2 (1 instance)

Performance Bottlenecks

The request time for the POST request to https://devicebind.ebay.com/signin/sub/log is relatively high at 0.8190000000000001 seconds

The response size of 684.0040000388399 for the GET request to https://devicebind.ebay.com/signin/sub/tt.html may indicate a performance bottleneck

Time-Based Traffic Distribution (e.g., by hour, day)

The traffic distribution appears to be evenly spread across different hours and days. However, there is a slight spike at 15:49:10 GMT.

Note that some of these key points may not be directly relevant or actionable without additional context.

2) What is the status code distribution of responses in the HAR file?

Based on the provided log data, here are the key points to extract:

Top Accessed URLs

1. https://www.ebay.com/

2. https://devicebind.ebay.com/signin/sub/tt.html

3. https://ir.ebaystatic.com/rs/c/sub-EF1KPzv6.js

Unusual Traffic Patterns or Spikes

A spike in traffic occurred around 15:48:52 GMT, with a GET request to https://sr-client-cfg.amplitude.com/config resulting in a 200 status code.

Another spike occurred at 15:49:10 GMT, with a POST request to https://devicebind.ebay.com/signin/sub/log also resulting in a 200 status code.

Common Status Codes and Errors

200 OK (most frequent status code)

No errors reported

Frequent IP Addresses

1. 66.211.166.16

2. 18.165.183.61

3. 23.49.235.2

4. 23.49.225.144

Performance Bottlenecks

The POST request to https://devicebind.ebay.com/signin/sub/log at 15:49:11 GMT took approximately 0.819 seconds.

The GET request to https://sr-client-cfg.amplitude.com/config at 15:48:52 GMT took around 3.347 seconds.

Time-Based Traffic Distribution

Most traffic occurred between 15:44:53 and 15:49:15 GMT, with a peak at 15:49:10 GMT.

The frequency of requests decreased after 15:49:15 GMT.

Please note that these key points are based on the provided log data and might not be comprehensive or accurate without further context.

3) How many requests resulted in a timeout or failed to complete, based on their timings?

Based on the provided data, here are the key points to extract:

Top Accesssed URLs

1. `https://www.ebay.com/` (multiple times)

2. `https://devicebind.ebay.com/signin/sub/tt.html` (twice)

3. `https://sr-client-cfg.amplitude.com/config` (once)

Unusual Traffic Patterns or Spikes

A significant spike in traffic at 15:49:10 GMT with a time of 16319, which might indicate a peak usage period.

Multiple GET requests to `https://www.ebay.com/` within a short time frame.

Common Status Codes and Errors

None mentioned in the provided data.

Frequent IP Addresses

1. `66.211.166.16` (twice)

2. `18.165.183.61`

3. `23.49.235.2`

4. `23.49.225.144`

Performance Bottlenecks

The GET request to `https://www.ebay.com/` has a significant wait time of 756.1659999191761 milliseconds, which might indicate a performance bottleneck.

Time-Based Traffic Distribution (e.g., by Hour)**

Unfortunately, the provided data does not include information on time-based traffic distribution.

4) What is the average size of responses for different content types (e.g., text, image)?

Based on the provided data, here are the key points to extract:

Top Accessed URLs

1. https://www.ebay.com/ (multiple requests)

2. https://devicebind.ebay.com/signin/sub/tt.html (multiple requests)

3. https://sr-client-cfg.amplitude.com/config (single request)

Unusual Traffic Patterns or Spikes

A spike in traffic at 15:49:10 GMT, with a significant increase in requests to the devicebind.ebay.com sign-in page.

Common Status Codes and Errors

Most requests return status code 200 (OK), indicating successful responses.

No error codes are reported in the provided data.

Frequent IP Addresses

1. 66.211.166.16 (multiple requests)

2. 18.165.183.61 (single request)

3. 23.49.235.2 (single request)

4. 23.49.225.144 (single request)

Performance Bottlenecks

The longest response time is reported for the devicebind.ebay.com sign-in page (response_time = 0.9830000000000041 seconds).

The largest response size is reported for the ir.ebaystatic.com JavaScript file (response_size = 684.0040000388399 bytes).

Time-Based Traffic Distribution

Traffic distribution by hour:

+ 15:44:53 GMT: 1 request

+ 15:48:52 GMT: 2 requests

+ 15:49:10 GMT: 5 requests (spike)

+ 15:49:11 GMT: 1 request

+ 15:49:15 GMT: 1 request

5) What are the slowest loading resources by response size?

Based on the provided log data, I have extracted the following key points:

Top accessed URLs:

1. `https://www.ebay.com/` (multiple occurrences)

2. `https://devicebind.ebay.com/signin/sub/tt.html` (multiple occurrences)

3. `https://sr-client-cfg.amplitude.com/config` (single occurrence)

Unusual traffic patterns or spikes:

A spike in traffic at 15:49:10 GMT with a response size of 684.004 bytes and a request time of 0.983 seconds

A large wait time of 756.165 seconds for the `https://ir.ebaystatic.com/rs/c/sub-EF1KPzv6.js` URL

Common status codes and errors:**

Status code 200 (OK) for most URLs

No error status codes or notable issues reported

Frequent IP addresses:

1. `66.211.166.16` (multiple occurrences)

2. `23.49.235.2` (single occurrence)

Performance bottlenecks:

A significant wait time of 756.165 seconds for the `https://ir.ebaystatic.com/rs/c/sub-EF1KPzv6.js` URL

Time-based traffic distribution (e.g., by hour):

The majority of requests occur in a short period, specifically between 15:44:53 GMT and 15:49:11 GMT.

İYİLEŞTİRMELER

- Model bazı sorularda istenilen verileri bulamadığını söylemektedir. Dolayısıyla, veri seti büyütülmelidir veya prompt kısmına daha detaylı talimatlar girilebilir.
- Multi Query Retirever yöntemi birden fazla sorgu üreterek modelin cevap üretme süresini uzatmaktadır. Bunun yerine daha verimli bir yöntem tercih edilebilir.