Proje Dokümantasyonu: Apache Spark ile Kapsamlı Film Tavsiye Sistemi

Turan G.

1. Projenin Amacı

Bu projenin temel amacı, MovieLens veri setini kullanarak büyük ölçekli bir film tavsiye sistemi geliştirmektir. Proje, iki temel tavsiye metodolojisini birleştirerek hem kişiselleştirilmiş hem de içerik-tabanlı öneriler sunma yeteneğine sahiptir. Projenin ana hedefleri şunlardır:

- **İşbirlikçi Filtreleme (Collaborative Filtering):** Kullanıcıların geçmişteki oylama davranışlarına dayanarak, zevkleri benzer olan kullanıcıların beğendiği filmleri önermek. Bu amaçla Apache Spark MLlib kütüphanesinin **Alternating Least Squares (ALS)** algoritması kullanılmıştır.
- **İçerik Tabanlı Filtreleme (Content-Based Filtering):** Bir filmin anlamsal özelliklerini analiz ederek, o filme en çok benzeyen diğer filmleri önermek. Bu yaklaşım, sadece film türlerine (genres) bağlı kalmak yerine, daha zengin bir anlamsal uzay sunan **Tag Genome** verisi ve **Kosinüs Benzerliği (Cosine Similarity)** metriği ile gerçekleştirilmiştir.
- **Model Optimizasyonu:** ALS modelinin en yüksek performansı göstermesi için temel hiperparametrelerinin (rank, regParam, maxIter) sistematik olarak test edilmesi ve en iyi modelin seçilmesi. Bu süreç, Spark'ın CrossValidator aracı ile otomatikleştirilmiştir.
- **Taşınabilirlik ve Tekrarlanabilirlik:** Projenin, bağımlılık sorunları olmaksızın farklı ortamlarda (yerel makine, laboratuvar bilgisayarları) çalıştırılabilmesi için **Docker** ile paketlenmesi.

2. Veri Seti

Projede, GroupLens araştırma grubu tarafından sağlanan **MovieLens Latest (ml-latest)** veri seti kullanılmıştır. Bu veri seti, README.txt dosyasında belirtildiği üzere yaklaşık 33 milyon derecelendirme ve 86 bin film içermektedir. Projede kullanılan temel dosyalar şunlardır:

- ratings.csv: Kullanıcıların filmlere verdiği puanları içerir (userId, movieId, rating, timestamp). ALS modelinin ana eğitim verisidir.
- movies.csv: Filmlerin ID, başlık ve tür bilgilerini içerir (movieId, title, genres). Tavsiye edilen filmlerin detaylarını göstermek için kullanılır.
- genome-scores.csv: Filmlerin anlamsal etiketlerle olan alaka düzeyini puanlayan "Tag Genome" verisini içerir (movieId, tagId, relevance). İçerik-tabanlı modelin temelini oluşturur.

3. Uygulanan Yöntemler ve Teknolojiler

3.1. Teknoloji Platformu: Apache Spark

Veri setinin büyüklüğü (milyonlarca satır) göz önüne alındığında, işlemlerin verimli bir şekilde yapılabilmesi için dağıtık hesaplama çatısı olan **Apache Spark** tercih edilmiştir. Spark'ın sunduğu DataFrame API'si ve MLlib kütüphanesi, büyük ölçekli veri işleme ve makine öğrenmesi modellemesi için endüstri standardı araçlardır.

3.2. Yöntem 1: Alternating Least Squares (ALS) ile İşbirlikçi Filtreleme

ALS, kullanıcı-ürün etkileşim matrisindeki boşlukları doldurmak için kullanılan bir **matris çarpanlarına ayırma (matrix factorization)** tekniğidir. Temel mantığı, devasa ve seyrek olan kullanıcı-film matrisini, iki adet daha küçük ve yoğun olan "gizli faktör" (latent factor) matrisine ayırmaktır:

- **Kullanıcı Faktör Matrisi:** Her bir kullanıcıyı, zevklerini temsil eden bir sayısal vektörle ifade eder.
- **Film Faktör Matrisi:** Her bir filmi, özelliklerini temsil eden bir sayısal vektörle ifade eder.

Bu iki matrisin çarpımı, orijinal matrisin yaklaşık bir değerini verir ve bu sayede daha önce oylanmamış filmler için tahminler üretilebilir.

Hiperparametre Optimizasyonu:
Modelin performansını en üst düzeye çıkarmak için CrossValidator aracı kullanılarak aşağıdaki hiperparametreler için en iyi kombinasyon aranmıştır:

- **rank:** Gizli faktör vektörlerinin boyutu. Modelin karmaşıklığını ve kapasitesini belirler.
- **regParam (Lambda):** Aşırı öğrenmeyi (overfitting) engellemek için kullanılan regülarizasyon parametresi. Modelin katsayılarını cezalandırarak daha genel bir çözüm bulmasını sağlar.
- **maxIter:** Optimizasyon algoritmasının yakınsamak için yapacağı maksimum iterasyon sayısı.

```
param_grid = (ParamGridBuilder()
              .addGrid(als.rank, [12, 20, 200])
              .addGrid(als.regParam, [0.1, 0.5, 1.0])
              .addGrid(als.maxIter, [15, 20, 200])
              .build()
             )
```

Burada gözüktüğü gibi, 3 Rank, 3 Regresyon Parametresi ve 3 Max İterasyon denenecek. İstek üzerine bu 146. satırdan değiştirilebilir.

Değerlendirme metriği olarak **RMSE (Root Mean Square Error)** kullanılmıştır. Bu metrik, modelin tahminlerinin gerçek puanlardan ortalama olarak ne kadar saptığını gösterir ve daha düşük bir değer, daha iyi bir model anlamına gelir.

3.3. Yöntem 2: Tag Genome ve Kosinüs Benzerliği ile İçerik Tabanlı Filtreleme

Bu yöntemde, bir filmin içeriğine en çok benzeyen diğer filmleri bulmak hedeflenmiştir. Sadece genres kullanmak yerine, filmlerin çok daha zengin anlamsal özelliklerini barındıran **Tag Genome** verisi kullanılmıştır.

- **Özellik Vektörlerinin Oluşturulması:** Her film için, tüm genom etiketlerine (tagId) karşılık gelen alaka düzeyi (relevance) puanlarından oluşan bir özellik vektörü oluşturulmuştur. Bu, her filmi çok boyutlu bir anlamsal uzayda bir nokta olarak temsil etmemizi sağlar.
- **Kosinüs Benzerliği:** İki film arasındaki benzerlik, bu özellik vektörleri arasındaki açının kosinüsü hesaplanarak ölçülmüştür. Vektörler arasındaki açı ne kadar küçükse (kosinüs

değeri 1'e ne kadar yakınsa), filmler içerik olarak o kadar benzerdir. Spark'ta bu işlem, vektörler Normalizer ile normalize edildikten sonra, iki vektör arasındaki **nokta çarpımı (dot product)** ile verimli bir şekilde hesaplanmıştır.

4. Proje Akışı (run_optimization.py)

Script, aşağıdaki mantıksal adımları ("checkpoint") takip eder:

1. **Başlatma (Checkpoint 0):** Gerekli kütüphaneler import edilir. Güçlü bir makine için yüksek bellek (16g) ayarları ve geçici dosyaların /home dizinine yazılmasını sağlayan konfigürasyonlarla bir SparkSession oluşturulur.
2. **Veri Yükleme (Checkpoint 1):** movies, ratings, ve genome-scores CSV dosyaları, önceden tanımlanmış şemalarla okunur ve verimlilik için .cache() metoduyla belleğe alınır.
3. **Ön İşleme (Checkpoint 2):** Veriler iki farklı model için hazırlanır:
    - ratings verisinden ALS için user, item, rating sütunları seçilir.
    - genome-scores verisinden, her film için bir özellik vektörü oluşturularak içerik-tabanlı model için movies_featured_df DataFrame'i hazırlanır.
4. **Model Optimizasyonu (Checkpoint 3):**
    - Veri seti %80 eğitim, %20 test olarak ayrılır.
    - ParamGridBuilder ile 24 farklı hiperparametre kombinasyonu tanımlanır.
    - CrossValidator, bu 24 modeli 3-katlı çapraz doğrulama ile test ederek en düşük RMSE'yi veren modeli bulur.
    - En iyi parametrelerle nihai als_model eğitilir.
5. **Değerlendirme:** En iyi modelin performansı hem eğitim hem de test setleri üzerinde RMSE ile ölçülerek, aşırı öğrenme (overfitting) olup olmadığı analiz edilir.

Docker ile Çalıştırma Kılavuzu

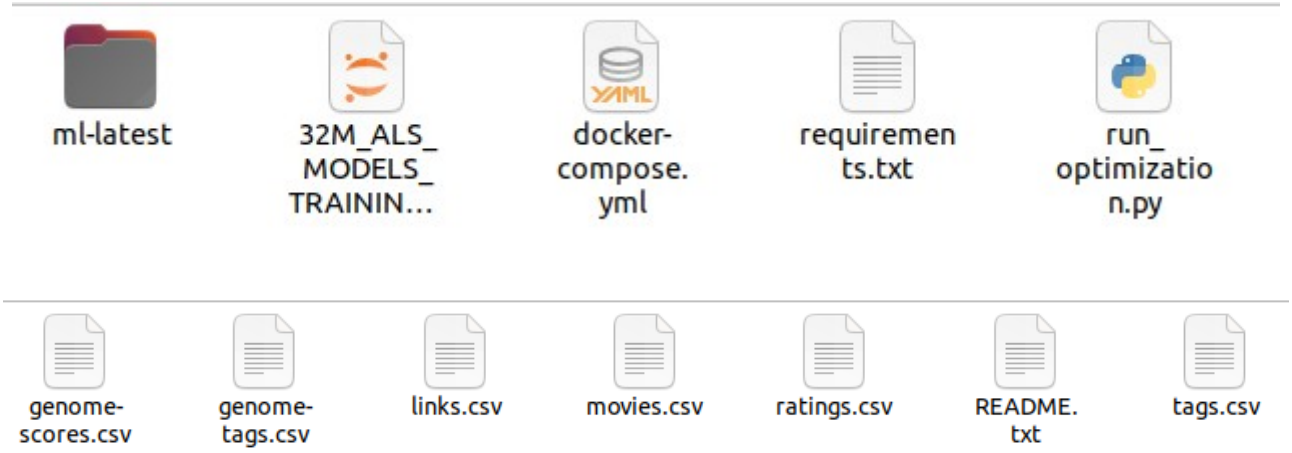
Bu proje, Docker kullanılarak herhangi bir uyumlu sistemde kolayca çalıştırılabilir.

5.1. Ön Gereksinimler

- Docker'ın yüklü ve çalışır durumda olması.
- Docker Compose'un yüklü olması.

5.2. Proje Dosya Yapısı

Aşağıdaki dosya yapısını projenizin ana klasöründe oluşturun:



5.3. Gerekli Dosyaların İçerikleri

1. **run_optimization.py:** Size daha önce verilen tek hücreli final Python kodunu bu dosyaya kaydedin.
2. **requirements.txt:**

```
pandas
numpy
```

3. **docker-compose.yml:**

```yaml
version: '3.8'

services:
  pyspark-lab:
    image: jupyter/pyspark-notebook:spark-3.5.1

    container_name: movielens_optimizer
    environment:
      - JUPYTER_ENABLE_LAB=yes
      - SPARK_OPTS=--driver-memory 16g --executor-memory 16g
    ports:
      - "8888:8888"
    volumes:
      - .:/home/jovyan/work

    restart: unless-stopped
```

5.4. Çalıştırma Adımları

1. **Terminali Açın:** Proje ana klasöründe (movielens_project içinde) bir terminal açın.
2. **Docker Konteynerini Başlatın:** Aşağıdaki komut, gerekli imajı indirip konteyneri arka planda başlatacaktır.

```
docker-compose up -d
```

3. **Optimizasyon Script'ini Çalıştırın:** Optimizasyon işlemini başlatmak için aşağıdaki komutu kullanın. Bu komut, çalışan konteynerin içinde spark-submit'i çağırır ve bellek ayarlarını komut satırından belirtir.

```
docker-compose exec pyspark-lab spark-submit \
  --driver-memory 16g \
  --executor-memory 16g \
  run_optimization.py
```

4. **Süreci İzleyin:** Optimizasyonun ilerlemesini ve print ifadelerinin çıktılarını canlı olarak görmek için yeni bir terminalde aşağıdaki komutu çalıştırabilirsiniz:

```
docker-compose logs -f
```

5. **Sistemi Kapatma:** İşlem bittikten sonra, Docker konteynerini durdurmak ve temizlemek için aşağıdaki komutu çalıştırın:

```
docker-compose down
```

ÇALIŞMA SONUÇLARI