گزارش تمرین ۴ رایانش ابری

بردیا اردکانیان - ۹۸۳۱۰۷۲
آراد فیروزکوهی - ۹۸۳۱۰۴۷

تمامی پروژه در یک نوت‌بوک در google collab در این لینک انجام شده. برای این کار در نوتبوک گوگل کولب را با کمک دستوراتی یک کلاستر اسپارک راه اندازی می‌کنیم. این کار تنها به جهت افزایش سرعت محاسبات می‌باشد و چون سیستم‌های ما ضعیفه =)

گام محیط پیاده سازی

جاوا و اسپارک دانلود و نصب شده، و دیتا از داخل گوگل درایو خوانده می‌شود. توجه کنید که دیتاست‌ها را در ادرسی در گوگل درایو قرار داده‌ام.

```
[ ] from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

PySpark Setup

Download Java Virtual Machine (JVM)

```
[ ] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
```

Download Hadoop

```
[ ] !wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
```

Install

```
[ ] !tar xf spark-3.0.0-bin-hadoop3.2.tgz
```

```
[ ] !pip install -q findspark
```

Find Spark

```
[ ] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"
```

```
[ ] import findspark
    findspark.init()
```

همانطور که مشاهده می‌کنید باینری‌های جاوا و اسپارک دانلود شده‌اند و در نهایت با کمک pip اسپارک را راه می‌اندازیم.

گام داده‌گان

در این گام دیتافریم‌های مربوطه به فایل games و rating خوانده و کش می‌شوند. توجه کنید اگر کش نکنیم سرعت actionها و transformationها، مدل کردن و یادگیری مدل کمتر می‌شود. در مرحله بعدی می‌بایست ورودی‌های مدل را نرمالایز کنیم تا در بازه ۰ تا ۱ قرار گیرند. امتیاز داده شده از طرف کاربر بین ۱ تا ۵ و امتیاز داده شده به بازی از طرف منتقدین بین ۱ تا ۱۰۰ می‌باشد. پس به ترتیب دو ردیف جدید در دیتافریم آماده می کنیم که حاصل موارد فوق را تقسیم بر ۵ و ۱۰۰ ذخیره می‌کند تا در بازه ۰ و ۱ قرار بگیرند.

```
--------------+
a_score_normal|
--------------+
          0.99|
          0.99|
          0.99|
          0.99|
          0.99|
--------------+
```

آماده سازی داده اموزشی و تست

در مرحله قبلی برای سادگی کار دو دیتافریم را بر اساس ردیف مشترک game_id با یکدیگر ادغام می‌کنیم. برای آموزش مدل تنها به داده شناسه کاربر، شناسه بازی و امتیاز داده شده به بازی توسط کاربر نیاز داریم. دقیقا با همین سه ورودی می‌توانیم شباهت را بیابیم که کدام کاربرها علایق مشترکی دارد یا به عبارتی به بازی‌های مشابه نمرات مشابهی می‌دهند. انتخاب ۳۰ ۷۰ تماما بر اساس تجربه بوده می‌تواند هر عددی باشد.

```
[15] x = join_df.select('user_id', 'game_id', 'rating_normal')
     # split into 70 train 30 test
     train_test_split = x.randomSplit([0.7, 0.3])
     train = train_test_split[0].withColumnRenamed('rating_normal', 'y')
     test = train_test_split[1].withColumnRenamed('rating_normal', 'y*')
```

```
[16] train_size = train.count()
     test_size = test.count()

     print(f''' - [Train/Test size] - {train_size}, {test_size}''')
```

```
 - [Train/Test size] - 686875, 294673
```

آموزش مدل

در نهایت یک مدل ALS آموزش می‌دهیم تا بتوانیم تخمینی بر امتیازات کاربران داشته باشیم. در این مدل، ابتدا ماتریس آیتم‌ها فریز شده و gradient descent بر روی کاربران اجرا شده، و سپس ماتریس کاربران فریز شده و gradient descent را روی ماتریس آیتم‌ها

اعمال میکنیم. این کار باعث امکان موازی‌سازی الگوریتم و استفاده آن در دیتابیس‌های بزرگ و توضیع‌شده را می‌دهد. همچنین این الگوریتم باعث رفع اشکال‌هایی مانند cold start می‌شود. در نهایت خروجی مدل برای داده تست را مشاهده می‌کنیم.

ALS model

```
als = ALS(maxIter=20, regParam=0.01, coldStartStrategy='drop', seed=5)

als.setUserCol('user_id')
als.setItemCol('game_id')
als.setRatingCol('y')
```

```
ALS_92f6e2a5e803
```

```
model = als.fit(train)
print(' - [X] ALS model training is complete.')
```

```
 - [X] ALS model training is complete.
```

```
prediction = model.transform(test)
print(' - [X] ALS model training is complete.!')
```

```
 - [X] ALS model training is complete.!
```

```
prediction.join(game_df, 'game_id').select(
    'user_id', 'name', 'prediction', 'y*').show(n=10, truncate=False)
```

```
+-------+-----------------------------+----------+---+
|user_id|name                         |prediction|y* |
+-------+-----------------------------+----------+---+
|35982  |Uncharted 3: Drake's Deception|0.6844709 |0.6|
|3087   |Uncharted 3: Drake's Deception|0.5412595 |0.6|
|9165   |Uncharted 3: Drake's Deception|0.7475925 |0.6|
|22164  |Uncharted 3: Drake's Deception|0.61334807|0.6|
|10140  |Uncharted 3: Drake's Deception|0.5575196 |0.6|
|7001   |Uncharted 3: Drake's Deception|0.7741144 |0.8|
|26244  |Uncharted 3: Drake's Deception|0.7292792 |0.8|
|13991  |Uncharted 3: Drake's Deception|0.8026867 |0.8|
|10610  |Uncharted 3: Drake's Deception|0.7423294 |0.8|
|18313  |Uncharted 3: Drake's Deception|0.56284404|0.8|
+-------+-----------------------------+----------+---+
only showing top 10 rows
```

ارزیابی مدل

برای ارزیابی مدل از Root-mean square error استفاده شده است. می‌توان از sse یا هر تابع هزینه دیگری استفاده شود صرفا سلیقه ما rmse بوده. قبل از محاسبه rmse متوجه شدم بعضی از ردیف‌ها داده ندارند و باعث می‌شود عملیات به مشکل منطقی بخورد و exception دهد. برای همین تمامی ردیف‌هایی که این مشکل را دارند حذف می‌کنیم تا داده‌ها تمیز شوند.

Evaluation

```
[21] # Prediction count
     pred_count = prediction.count()
     # Drop rows with any missing data
     prediction = prediction.dropna(how="any", subset=["prediction"])
     # Clean prediction count
     clean_pred_count = prediction.count()

     print(f''' - [X] [Nan values] - {pred_count - clean_pred_count}''')
```

```
 - [X] [Nan values] - 0
```

```
[ ] evaluator = RegressionEvaluator(labelCol='y*', predictionCol='prediction', metricName='rmse')
    rmse = evaluator.evaluate(prediction)
    print(f''' - [X] [Root Mean Square Error] - {rmse}''')
```

```
 - [X] [Root Mean Square Error] - 0.013039743453157025
```

در نهایت از این مدل به کمک شباهت کسینوسی مواردی که بیشترین شباهت را دارند را انتخاب می‌کنیم.

```python
def cosine_similarity(vector_1, vector_2):
    v1 = np.asarray(vector_1)
    v2 = np.asarray(vector_2)
    cs = v1.dot(v2) / (LA.norm(v1) * LA.norm(v2))
    return(cs)
```

```python
def get_recommendations(similar_df):
    recom_df = train.join(similar_df, train.game_id == similar_df.item_index)
    recom_df = recom_df.select('game_id', 'similarity_score').distinct()
    recom_df = recom_df.orderBy(col('similarity_score').desc()).limit(5)
    recom_df = recom_df.join(game_df, on = 'game_id')
    recom_df.show()

def compute_cosine_similarity(itemFactors, game_id):
    item = itemFactors.where(col('id') == game_id).select(col('features'))
    item_features = item.rdd.map(lambda x: x.features).first()

    res = []
    for row in itemFactors.rdd.toLocalIterator():
      _id = row.__getattr__('id')
      features = row.__getattr__('features')
      similarity_score = cosine_similarity(features, item_features)
      if _id != game_id:
        res.append([_id, similarity_score])

    R = Row('item_index', 'similarity_score')
    return spark.createDataFrame([R(col[0], float(col[1])) for col in res])

in_game_id = int(input('[INP] Enter item index to generate similar recommendations: '))
print("Showing games similar to:")
game_df.filter(col("game_id")==in_game_id).show()
similar_itmes_df = compute_cosine_similarity(model.itemFactors, in_game_id)
get_recommendations(similar_itmes_df)
```

شباهت کسینوسی

تابع cosine_similarity فرمول شباهت کسینوسی را بین دو بردار پیاده‌سازی می‌کند. تابع compute_cosine_similarity شباهت زوج موارد را بر اساس فیچرهای مدل و تابع قبلی محاسبه می‌کند. در نهایت در تابع get_recommendations بر اساس شباهت موارد را مرتب کرده و چند مورد شبیه‌تر را انتخاب می‌کنیم.

خروجی

همانطور که می‌بینید مدل به سادگی توانسته برای بازی company of heros و شباهت نمره داده شده توسط کاربران ۵ بازی دیگر را پیشنهاد دهد. فقط اون گیمری که هم chuchu rocket بازی می کنه هم hitman آدم جالبیه :)

```
[INP] Enter item index to generate similar recommendations: 89
Showing games similar to:
+-------+----------------+------------+--------------------+----------+----------------+
|game_id|            name|release_date|             summary|meta_score|meta_score_normal|
+-------+----------------+------------+--------------------+----------+----------------+
|     89|Company of Heroes|   13-Sep-06|A real-time strat...|        93|            0.93|
+-------+----------------+------------+--------------------+----------+----------------+

+-------+------------------+--------------------+------------+--------------------+--------------------+----------------+
|game_id|  similarity_score|                name|release_date|             summary|          meta_score|meta_score_normal|
+-------+------------------+--------------------+------------+--------------------+--------------------+----------------+
|   1629|0.9721168249583939|      ChuChu Rocket!|   10-Jun-01|It'll take you a ...|                  84|            0.84|
|   1888|0.9813503370335455|  Hitman: Absolution|   20-Nov-12|Wear the uniform ...|                  83|            0.83|
|   3344|0.9699528560161206|Jumpgate: The Rec...|   23-Sep-01|"Real Time Spacef...| struggle to stay...|            null|
|   6606|0.9649898008408242|Mobile Suit Gunda...|   15-Jan-02|Become your own w...|                  71|            0.71|
|   7305| 0.96381918540109|The King of Fight...|    2-Dec-08|The first ever co...|                  70|             0.7|
+-------+------------------+--------------------+------------+--------------------+--------------------+----------------+
```