Project 5

July 9, 2019

1 DATA 612 Project 5: Implementing a Recommender System on Spark

The purpose of this project is to gain hands on experience of building a recommender using Apache Spark. Apache Spark is an open-source distributed cluster computing framework with that facilitates in-memory computation with less reliance on the disk storage. Making it one of the most widely used framework for big data processing in both the academic and industry setting.

I am using the Jester data set 1 from [http://eigentaste.berkeley.edu/dataset/] (that I have used on previous projects) for this project. The data set contains anonymous ratings from 24,983 users who have rated 36 or more jokes. The Ratings are real values ranging from -10.00 to +10.00. The selected subset of the data contains 2,498,300 ratings with missing ratings denoted with the number 99, making it in an ideal data set to utilize on the Spark framework.

1.0.1 Load Libraries

```
In [1]: import findspark
        findspark.init()

        from pyspark.sql import  SparkSession
        from pyspark.ml.feature import IDF, Tokenizer, CountVectorizer
        from sklearn.metrics.pairwise import cosine_similarity
        import pandas as pd
        import numpy as np
```

1.0.2 Load Data into Spark

The data is separated into two data sets, a text/csv file that contains the ratings for each joke the users and a directory of html files where each file contains the content for a given joke. We will now connet to a local Spark instance and load these data into Spark dataframes.

```
In [4]: # initialize lists to store texts for each joke and their IDs
        jokes = []
        joke_ids = list(range(1,101))

        # load each html file and extract the text
        for i in range(0,100):
            joke_text = open("data/jokes/init"+str((i+1))+".html", "r")
            a = joke_text.read()
```

```
            joke=a[a.find("-->")+3:a.find("<!--end")]
            jokes.append(joke)
            joke_text.close()

        # use intermediate pandas dataframe
        jokes_df = pd.DataFrame()
        jokes_df['joke_id'] = joke_ids
        jokes_df['text'] = jokes


        # Create a spark session
        spark = SparkSession.builder.getOrCreate()

        # load jokes into spark
        spark_jokes_df = spark.createDataFrame(jokes_df)
        spark_jokes_df.createOrReplaceTempView('spark_jokes_df')
        spark_jokes_df.head(2)
```

```
Out[4]: [Row(joke_id=1, text='\nA man visits the doctor. The doctor says "I have bad news for y
         Row(joke_id=2, text='\nThis couple had an excellent relationship going until one day h
```

The output above shows the first two jokes in row format that was loaded from the HTML files. Now let's load the ratings into Spark.

```
In [5]: # load user interactions and ratings
        spark_interactions = spark.read.csv("data/jester-data-1.csv")
        spark_interactions.createOrReplaceTempView('spark_interactions')

        # cache dataframe for later use
        spark_interactions.cache()

        # preview data
        spark_interactions.select('_c0','_c1','_c2','_c3','_c4','_c5').show(10)
```

```
+---+-----+-----+-----+-----+-----+
|_c0|  _c1|  _c2|  _c3|  _c4|  _c5|
+---+-----+-----+-----+-----+-----+
| 74|-7.82| 8.79|-9.66|-8.16|-7.52|
|100| 4.08|-0.29| 6.36| 4.37|-2.38|
| 49|   99|   99|   99|   99| 9.03|
| 48|   99| 8.35|   99|   99|  1.8|
| 91|  8.5| 4.61|-4.17|-5.39| 1.36|
|100|-6.17|-3.54| 0.44| -8.5|-7.09|
| 47|   99|   99|   99|   99| 8.59|
|100| 6.84| 3.16| 9.17|-6.21|-8.16|
|100|-3.79|-3.54|-9.42|-6.89|-8.74|
| 72| 3.01| 5.15| 5.15| 3.01| 6.41|
+---+-----+-----+-----+-----+-----+
only showing top 10 rows
```

The output above shows the first 10 rows and 6 columns of the ratings dataframe in Spark. Each row represents a user, the first column is number of jokes that the user rated and the each subsequent column represents the ratings for a specific joke. The output below shows that both data sets have been loaded into Spark and thus building the recommender may proceed.

```
In [6]: spark.catalog.listTables()
```

```
Out[6]: [Table(name='spark_interactions', database=None, description=None, tableType='TEMPORAR
         Table(name='spark_jokes_df', database=None, description=None, tableType='TEMPORARY', i
```

1.0.3 Content-Based Filtering Model

I have previously used SVD and other approaches on these data but will be taking a different approach for this project. I have yet to build a Content-Based recommender on this data set so I will take this opportunity to do so. As a reminder, content-based recommenders uses descriptions of items the user has interacted with and recommends them similar items with which they have not interacted. I will be building the item profiles using the text for each joke by applying TF-IDF transformation to get the most important terms in each joke. I will then build user profiles my taking an average of the item profiles the user has interacted with. To make a recommendation a user's profile will be compared to the profiles of items he/she has not yet rated, and the jokes that are most similar to the user's profile will be recommended. Similarity will determined based on the Cosine similarity.

1.0.4 Create Item Profiles

```
In [13]: # transform each joke content into an array of words
         tokenizer = Tokenizer(inputCol="text", outputCol="words")
         wordsData = tokenizer.transform(spark_jokes_df)

         # calculate the frequency of each work in each document
         countVectorizer = CountVectorizer(inputCol="words", outputCol="rawFeatures").fit(words
         featurizedData = countVectorizer.transform(wordsData)

         # now calculate the TF-IDF
         idf = IDF(inputCol="rawFeatures", outputCol="features")
         idfModel = idf.fit(featurizedData)
         rescaledData = idfModel.transform(featurizedData)

         # cache matrix
         rescaledData.cache()

         # create preview
         df = pd.DataFrame( {'feature' : countVectorizer.vocabulary, 'importance' : rescaledDat
         df.head(7)
```

```
Out[13]:   feature  importance
         0            0.000000
         1     the    0.552911
         2       a    0.233094
         3     and    0.338454
         4      to    0.000000
         5     <p>    0.456237
         6      of    0.000000
```

The output above shows a part of the item profile for the first joke. It shows the a sample of the terms and their importance. Now the user profiles will be created by averaging the item profiles that the user has rated before. We will need to load the interactions for a given user.

```
In [15]: # function to load items the user has rated
         def getUserInteractions(user_id):

             # extract items the user has rated
             user_interactions = [ key for (key,value) in spark_interactions.collect()[user_id]
             user_interactions = [ int(i.replace("_c", "")) for i in user_interactions ]

             # ignore and remove references to the first column, it is not needed here
             if user_interactions[0] == 0:
                 user_interactions.pop(0)
             return user_interactions

         # average the item profiles for a given user
         def createUserProfile(user_interactions):
             user_profile = np.zeros(len(countVectorizer.vocabulary))
             for i in user_interactions:
                 user_profile = user_profile + rescaledData.collect()[i-1]['features'].toArray

             user_profile = user_profile/len(countVectorizer.vocabulary)
             return user_profile

         # select a user at random to demonstrate
         user_id = 567
         user_interactions = getUserInteractions(user_id)
         user_profile = createUserProfile(user_interactions)

         # preview the profile
         user_profile_df = pd.DataFrame()
         user_profile_df['words'] = countVectorizer.vocabulary
         user_profile_df['importance'] = user_profile
         user_profile_df.head(10)
```

```
Out[15]:   words  importance
         0          0.000000
         1   the    0.022241
```

```
2     a    0.017553
3   and    0.019630
4    to    0.022089
5   <p>    0.014297
6    of    0.023257
7   you    0.021652
8    in    0.023165
9    he    0.032403
```

The output below shows a portion of the user 567's profile. Now let's provide this user with a recommendation.

1.0.5 Make Recommendataions

We will use the Cosine similarity to recommend jokes the user haven't rated before.

```
In [16]: # function to provide recommendations for a given user
         def makeRecommendations(user_profile):
             # inilize lists for recommendation
             recommendations = []
             joke = []

             # calculate similarity of items profiles not yet rated to the user profile
             for i in range(1,100):
                 if i not in user_interactions:
                     similarity = cosine_similarity(user_profile.reshape(1, -1),
                                                    rescaledData.collect()[i-1]['features'].toArra
                     recommendations.append(similarity[0][0])
                     joke.append(i)

             # for viewing purposes return a data frame
             recommendations_df = pd.DataFrame()
             recommendations_df['joke_id'] = joke
             recommendations_df['similarity'] = recommendations
             recommendations_df.sort_values(by='similarity', inplace=True, ascending=False)

             return recommendations_df
         #.sort_values(by=['similarity'], inplace=True, ascending=False)
         recommendations = makeRecommendations(user_profile)
         recommendations.head(5)
```

```
Out[16]:     joke_id  similarity
         5        76    0.201211
         19       93    0.173483
         17       91    0.162182
         18       92    0.141980
         23       97    0.138308
```

The above shows that based on the user's content, user 567 would probably like jokes 76, 93, 91, 92, and 97.

1.0.6 Conclusion

Working with text data is generally more computionally expensive than working with numbers because they usually require more memory and disk storage and cannot be indexed as simple as numeric values. Hence, this project provided a good use-case for utilizing Spark in memory processing. The benefits of Spark will shine as more users rate jokes and more jokes are added to the system. In an online environment, the main issue of this system will be to dynamically update user profiles and item profiles without sacrificing user experience. This makes Spark one of the best solution for this problem. I chose PySpark over R because of the ability to build a complete web experience for this solution. I will be presenting my web application for this recommender for my Data Science in Context Presentation.