Recommender Project

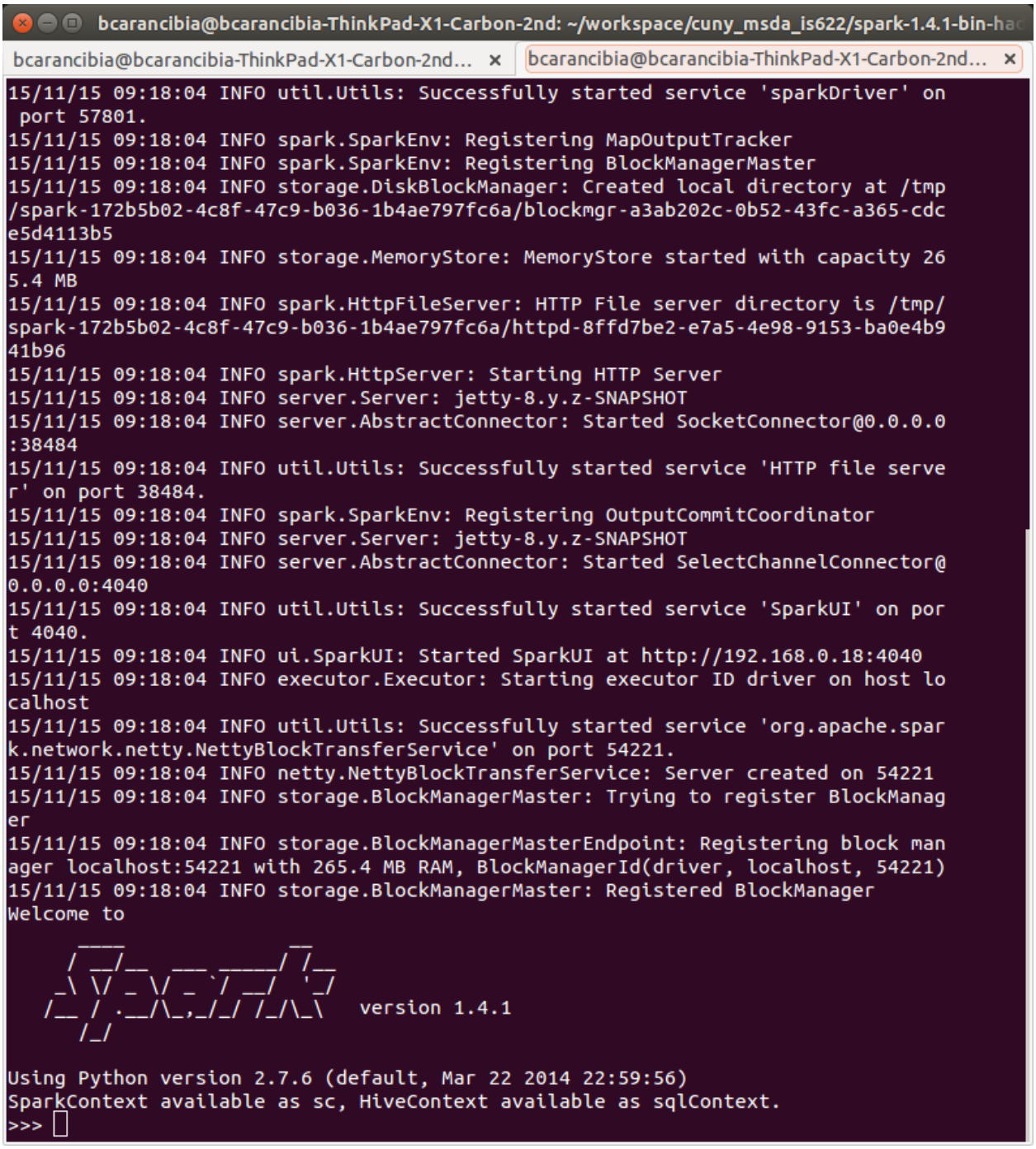
Ben Arancibia

11/15/2015

Introduction

For the week 12 mini project on recommendation systems, a recommendation system for movies was investigated. This was chosen because of the availability of the data. GroupLens Research has collected and made available rating data sets from the MovieLens web site (http://movielens.org). The data sets were collected over various periods of time, depending on the size of the set.

As a note, this work was done in the PySpark Interactive shell because so far I have been unsuccessful in linking Spark to Ipython/Jupyter notebooks. To account for this code was will be shown below



Data

The data was downloaded from movielens.org and there are multiple datasets that can be downloaded. The data downloaded for this was the complete data set (http://files.grouplens.org/datasets/movielens/ml-latest.zip). The data was downloaded and upzipped in a data folder in my home directory.

Recommendation

The next step is to load the data, parse it, and then create the recommendation. Within the ml-latest there are 4 csvs:

ratings.csv - PK is movieID movies.csv - PK is movieID tags.csv - PK is movieID links.csv - PK is movieID

```
import os
from pyspark.mllib.recommendation import ALS
import math

ratings_file = os.path.join('home/bcarancibia/ml-latest/ratings.csv', 'ml-latest', 'ratings.csv')
ratings_raw_data = sc.textFile(ratings_file)
ratings_raw_data_header = ratings_raw_data.take(1)[0]

complete_ratings_data = ratings_raw_data.filter(lambda line: line!=ratings_raw_data_header)\
    .map(lambda line: line.split(",")).map(lambda tokens: (int(tokens[0]),int(tokens[1]),float(tokens[2]
print (complete_ratings_data.count())
```

There are 21622187 recommendations in the complete dataset

Next we have to create a test dataset

```
training_RDD, test_RDD = complete_ratings_data.randomSplit([7, 3], seed=0L)
complete_model = ALS.train(training_RDD, best_rank, seed=seed,
                           iterations=iterations, lambda_=regularization_parameter)
```

Then test the accuracy of predictions

```
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))

predictions = complete_model.predictAll(test_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
rates_and_preds = test_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())

print(error)
```

Error is 0.809

Now it is time to make the recommendation engine. Load in the movies data set

```
complete_movies_file = os.path.join('home/bcarancibia/ml-latest/ratings.csv', 'ml-latest', 'movies.csv'
complete_movies_raw_data = sc.textFile(complete_movies_file)
complete_movies_raw_data_header = complete_movies_raw_data.take(1)[0]

# Parse
complete_movies_data = complete_movies_raw_data.filter(lambda line: line!=complete_movies_raw_data_heade
    .map(lambda line: line.split(",")).map(lambda tokens: (int(tokens[0]),tokens[1],tokens[2])).cache()

complete_movies_titles = complete_movies_data.map(lambda x: (int(x[0]),x[1]))

print(complete_movies_titles.count())
```

There are 30106 movies in the dataset.

Now we need to count number of ratings per movie

```
def get_counts_and_averages(ID_and_ratings_tuple):
    nratings = len(ID_and_ratings_tuple[1])
    return ID_and_ratings_tuple[0], (nratings, float(sum(x for x in ID_and_ratings_tuple[1]))/nratings)

movie_ID_with_ratings_RDD = (complete_ratings_data.map(lambda x: (x[1], x[2])).groupByKey())
movie_ID_with_avg_ratings_RDD = movie_ID_with_ratings_RDD.map(get_counts_and_averages)
movie_rating_counts_RDD = movie_ID_with_avg_ratings_RDD.map(lambda x: (x[0], x[1][0]))
```

To make a recommendation system, it would be good to add user ratings and then based on those ratings create a recommendation of movies for the user.

```
new_user_ID = 0

# The format of each line is (userID, movieID, rating)
new_user_ratings = [
     (0,260,4), # Star Wars
     (0,16,3), # Casino
     (0,25,4), # Leaving Las Vegas
     (0,296,3), # Pulp Fiction
     (0,858,5) , # Godfather, The
     (0,50,4), # Usual Suspects, The
     (0, 2028, 5), #Saving Private Ryan
     (0, 3062, 5), #Longest Day, The
     (0, 1272, 5) #Patton


    ]
new_user_ratings_RDD = sc.parallelize(new_user_ratings)
```

With the new rating we train the model

```
new_ratings_model = ALS.train(complete_data_with_new_ratings_RDD, best_rank,
    seed = seed, iterations = iterations, lambda_ = regularization_parameter)
```

To get the recommendations we will get an RDD with ratings for movies that user has not rated

```
new_user_ratings_ids = map(lambda x: x[1], new_user_ratings) # get just movie IDs

new_user_unrated_movies_RDD = (complete_movies_data.filter(lambda x: x[0] not in new_user_ratings_ids).m

new_user_recommendations_RDD = new_ratings_model.predictAll(new_user_unrated_movies_RDD)
```

Need to join movies and movie ratings

```
new_user_recommendations_rating_RDD = new_user_recommendations_RDD.map(lambda x: (x.product, x.rating))
new_user_recommendations_rating_title_and_count_RDD = \
    new_user_recommendations_rating_RDD.join(complete_movies_titles).join(movie_rating_counts_RDD)
```

```
#flatten
new_user_recommendations_rating_title_and_count_RDD = \
    new_user_recommendations_rating_title_and_count_RDD.map(lambda r: (r[1][0][1], r[1][0][0], r[1][1]))
```

Get top ten movies for user=0, filtering out movies with less than 25 ratings.

```
top_movies = new_user_recommendations_rating_title_and_count_RDD.filter(lambda r: r[2]>=25).takeOrdered

print(top_movies)
```

Results:

(u'"Godfather: Part II', 8.503749129186701, 29198)

(u'"Civil War', 8.386497469089297, 257)

(u'"Shawshank Redemption', 8.258510064442426, 67741)

(u'Cosmos (1980)', 8.252254825768972, 948)

(u'Band of Brothers (2001)', 8.225114960311624, 4450)

(u'Generation Kill (2008)', 8.206487040524653, 52)

(u"Schindler's List (1993)", 8.172761674773625, 53609)

(u'Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)', 8.166229786764168, 23915)

(u'Goodfellas (1990)', 8.12931139039048, 27123)

(u'Apocalypse Now (1979)', 8.005094327199552, 23905)

Results Recommendation engine performed well and the list output makes sense based on the inputs by the user. I do not think that I would use this recommendation system in place of another one, but it might be useful as an outside input for movie selection. For example, it could help determine if you would like a new movie that has come out. You could input it into the dataset and then see what movies in the dataset cluster around the new input movie. The main reason why you would want it is because of the issues with rating movies, see this 538 post: http://fivethirtyeight.com/features/fandango-movies-ratings/. It could also be used as an outside input for streaming movie selection (Netflix, Amazon, HBO) but I doubt that this would be better for movie selection compared to the already existing solutions.

References

https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html

http://www.slideshare.net/CasertaConcepts/analytics-week-recommendations-on-spark

https://www.codementor.io/spark/tutorial/building-a-recommender-with-apache-spark-python-example-app-part1

Coworkers - Booz Allen Hamilton