ALS Algorithm

Import Lib and init spark

```
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Group 7 - Recommendation System') \
.config('spark.sql.execution.arrow.pyspark.enabled', True)\
.config('spark.driver.memory','8G')\
.config('spark.ui.showConsoleProgress', True)\
.config('spark.sql.repl.eagerEval.enabled', True)\
.getOrCreate()
```

Read data:

```
# Data is downloaded from https://www.kaggle.com/bandikarthik/movie-recommendation-system
movies = spark.read.csv('../MovieLens/movie.csv', header=True, inferSchema=True)
ratings = spark.read.csv('../MovieLens/rating.csv',  header=True, inferSchema=True)
```

```
movies.limit(5).show()
```

```
+-------+--------------------+--------------------+
|movieId|               title|              genres|
+-------+--------------------+--------------------+
|      1|    Toy Story (1995)|Adventure|Animati...|
|      2|      Jumanji (1995)|Adventure|Childre...|
|      3|Grumpier Old Men ...|      Comedy|Romance|
|      4|Waiting to Exhale...|Comedy|Drama|Romance|
|      5|Father of the Bri...|              Comedy|
+-------+--------------------+--------------------+
```

```
ratings.limit(5).show()
```

```
+------+-------+------+-------------------+
|userId|movieId|rating|          timestamp|
+------+-------+------+-------------------+
|     1|      2|   3.5|2005-04-02 23:53:47|
|     1|     29|   3.5|2005-04-02 23:31:16|
|     1|     32|   3.5|2005-04-02 23:33:39|
|     1|     47|   3.5|2005-04-02 23:32:07|
|     1|     50|   3.5|2005-04-02 23:29:40|
+------+-------+------+-------------------+
```

```
print(ratings.agg({"rating": "max"}).collect()[0])
print(ratings.agg({"rating": "min"}).collect()[0])
```

```
Row(max(rating)=5.0)
[Stage 12:============================>
Row(min(rating)=0.5)
```

Create test, train set and als model

```
# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 1234)

# Create ALS model
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", nonnegative = True, implicitPrefs = False
          , coldStartStrategy="drop")
```

Add hyperparameters and build cross validation

```
# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [100]) \
            .addGrid(als.regParam, [.15]) \
            .build()
            #             .addGrid(als.maxIter, [5, 50, 100, 200]) \

# Define evaluator as RMSE and print length of evaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))
```

```
Num models to be tested:  1
```

```
# Build cross validation using CrossValidator
# numFolds=3 means the CrossValidator will create 3 different models.
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=3)
```

Fit Model and get best model

```
# We fit the cross validator to the 'train' dataset
model = cv.fit(train)

# We Extract best model from the cv model above
best_model = model.bestModel
```

Calculate RMSE

```
predictions = best_model.transform(test)
rmse = evaluator.evaluate(predictions)
print(f"Root mean square error: {rmse}")
print("====BEST MODEL ====")
print(f"BEST RANK: {best_model.rank}")
print(f"maxIter: {best_model._java_obj.parent().getMaxIter()}")
print(f"regParam: {best_model._java_obj.parent().getRegParam()}")
```

```
[Stage 344:==================================================>(199 + 1) / 200]
Root mean square error: 0.8143051599489648
====BEST MODEL ====
BEST RANK: 10
maxIter: 10
regParam: 0.1
```

Recommend movies for all users

```
# Generate n Recommendations for all users
recommendations = best_model.recommendForAllUsers(10)
recommendations.limit(10).show()
```

```
[Stage 399:=========================================>
+------+--------------------+
|userId|     recommendations|
+------+--------------------+
|   148|[{120821, 6.22960...|
|   463|[{3226, 6.3365936...|
|   471|[{3226, 5.771446}...|
|   496|[{121029, 6.44937...|
|   833|[{3226, 6.089091}...|
|  1088|[{3226, 5.434558}...|
|  1238|[{3226, 5.8392224...|
|  1342|[{121029, 6.59056...|
|  1580|[{120821, 5.34024...|
|  1591|[{3226, 6.2007923...|
+------+--------------------+
```

Find 7th User's Actual Preference:

```
ratings.join(movies, on='movieId').filter('userId = 7') \
.sort('rating', ascending=False).limit(10)
```

movieId	userId	rating	timestamp	title	genres
912	7	5.0	2002-01-16 18:09:56	Casablanca (1942)	Drama|Romance
3179	7	5.0	2002-01-16 19:22:51	Angela's Ashes (1...	Drama
1077	7	5.0	2002-01-16 18:48:18	Sleeper (1973)	Comedy|Sci-Fi
750	7	5.0	2002-01-16 18:44:19	Dr. Strangelove o...	Comedy|War
1196	7	5.0	2002-01-16 18:09:32	Star Wars: Episod...	Action|Adventure|...
587	7	5.0	2002-01-16 19:10:20	Ghost (1990)	Comedy|Drama|Fant...
1210	7	5.0	2002-01-16 18:10:54	Star Wars: Episod...	Action|Adventure|...
1721	7	5.0	2002-01-16 19:06:05	Titanic (1997)	Drama|Romance
2942	7	5.0	2002-01-16 18:38:41	Flashdance (1983)	Drama|Romance
2028	7	5.0	2002-01-16 18:24:41	Saving Private Ry...	Action|Drama|War

Recommend film for 7th user

```
recommendations = recommendations.withColumn("rec_exp", explode("recommendations")).select('userId',
col("rec_exp.movieId"), col("rec_exp.rating"))
recommendations.join(movies, on='movieId').filter('userId = 7').show()
```

```
+-------+------+---------+--------------------+--------------------+
|movieId|userId|   rating|               title|              genres|
+-------+------+---------+--------------------+--------------------+
|   3226|     7| 5.637633|Hellhounds on My ...|         Documentary|
| 121029|     7| 5.573067|No Distance Left ...|         Documentary|
| 120821|     7| 5.295107|The War at Home (...|     Documentary|War|
| 129536|     7|5.0036817|Code Name Coq Rou...|  (no genres listed)|
| 114070|     7|4.9300246|Good Job:  Storie...|         Documentary|
| 128366|     7|4.8328657|Patton Oswalt: Tr...|              Comedy|
| 117907|     7| 4.705026|My Brother Tom (2...|               Drama|
| 129451|     7| 4.669075|    Ingenious (2009)|Comedy|Drama|Romance|
| 112473|     7|4.6646147|Stuart: A Life Ba...|               Drama|
| 129243|     7| 4.609404|Afstiros katallil...|              Comedy|
+-------+------+---------+--------------------+--------------------+
```

SVD Algorithm

Import Lib and init spark, dask

```python
import joblib
import numpy as np
from dask.distributed import Client
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Group 7 - Recommendation System')\
.config('spark.sql.execution.arrow.pyspark.enabled', True)\
.config('spark.driver.memory','8G')\
.config('spark.ui.showConsoleProgress', True)\
.config('spark.sql.repl.eagerEval.enabled', True)\
.getOrCreate()

client = Client()
```

Read data and install funk-svd lib

```python
# Data is downloaded from https://www.kaggle.com/bandikarthik/movie-recommendation-system
movies = spark.read.csv('../MovieLens/movie.csv', header=True, inferSchema=True)
ratings = spark.read.csv('../MovieLens/rating.csv',  header=True, inferSchema=True)
```

```
!pip install git+https://github.com/gbolmier/funk-svd
```

```
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15: CryptographyDeprecationWarning: in
rom_bytes instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19: CryptographyDeprecationWarning: int_fr
bytes instead
```

Convert pyspark dataframe to pandas dataframe

```
import pandas as pd
from funk_svd import SVD

with joblib.parallel_backend('dask'):
    movies_df = movies.toPandas()
    rating_df = ratings.toPandas()
```

```
rating_df.columns = ['u_id', 'i_id', 'rating', 'timestamps']
movies_df.columns = ['i_id', 'title', 'genres']
rating_df
```

	u_id	i_id	rating	timestamps
0	1	2	3.5	2005-04-02 23:53:47
1	1	29	3.5	2005-04-02 23:31:16
2	1	32	3.5	2005-04-02 23:33:39
3	1	47	3.5	2005-04-02 23:32:07

Split data to train, test, validate set

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
# movielens18.drop(columns = 'timestamp', inplace = True)

with joblib.parallel_backend('dask'):
  train = rating_df.sample(frac=0.8)
  val = rating_df.drop(train.index.tolist()).sample(frac=0.5, random_state=8)
  test = rating_df.drop(train.index.tolist()).drop(val.index.tolist())
```

Fit Model

```python
lr, reg, factors = (0.01, 0.03, 90)

with joblib.parallel_backend('dask'):
  svd = SVD(lr=lr, reg=reg, n_epochs=20, n_factors=factors,
            min_rating=0.5, max_rating=5)
  svd.fit(X=train, X_val=val)

pred = svd.predict(test)
mae = mean_absolute_error(test["rating"], pred)
rmse = np.sqrt(mean_squared_error(test["rating"], pred))
print("Test MAE:  {:.2f}".format(mae))
print("Test RMSE: {:.2f}".format(rmse))
print('{} factors, {} lr, {} reg'.format(factors, lr, reg))
```

```
IOStream.flush timed out
val_loss: 0.64 - val_rmse: 0.80 - val_mae: 0.61 - took 6.5 sec
Epoch 9/20  | val_loss: 0.63 - val_rmse: 0.79 - val_mae: 0.61 - took 6.3 sec
Epoch 10/20 | val_loss: 0.63 - val_rmse: 0.79 - val_mae: 0.60 - took 6.2 sec
Epoch 11/20 | val_loss: 0.62 - val_rmse: 0.79 - val_mae: 0.60 - took 6.2 sec
Epoch 12/20 | val_loss: 0.62 - val_rmse: 0.79 - val_mae: 0.60 - took 6.0 sec
Epoch 13/20 | val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.60 - took 6.0 sec
Epoch 14/20 |
IOStream.flush timed out
val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.60 - took 6.0 sec
Epoch 15/20 | val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.60 - took 6.0 sec
Epoch 16/20 | val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.59 - took 5.9 sec
Epoch 17/20 | val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.59 - took 6.0 sec
Epoch 18/20 | val_loss: 0.61 - val_rmse: 0.78 - val_mae: 0.59 - took 6.5 sec
Epoch 19/20 | val_loss: 0.60 - val_rmse: 0.78 - val_mae: 0.59 - took 6.5 sec
Epoch 20/20 |
IOStream.flush timed out
val_loss: 0.60 - val_rmse: 0.78 - val_mae: 0.59 - took 6.5 sec

Training took 2 min and 15 sec
Test MAE:  0.59
Test RMSE: 0.78
90 factors, 0.01 lr, 0.03 reg
```

Function to recommend movies:

```
from itertools import product


def funk_svd_predict(userID, data_with_user, movies_df):
    userID = [userID]

    # all_users = data_with_user.u_id.unique()
    all_movies = data_with_user.i_id.unique()
    recommendations = pd.DataFrame(list(product(userID, all_movies)), columns=['u_id', 'i_id'])

    #Getting predictions for the selected userID
    pred_train = svd.predict(recommendations)
    recommendations['prediction'] = pred_train
    recommendations.head(10)

    sorted_user_predictions = recommendations.sort_values(by='prediction', ascending=False)

    user_ratings = data_with_user[data_with_user.u_id == userID[0]]
    user_ratings.columns = ['u_id', 'i_id', 'rating']
    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = movies_df[~movies_df['i_id'].isin(user_ratings['i_id'])].\
        merge(pd.DataFrame(sorted_user_predictions).reset_index(drop=True), how = 'inner', left_on = 'i_id', right_on = 'i_id').\
        sort_values(by='prediction', ascending = False)#.drop(['i_id'],axis=1)

    rated_df = movies_df[movies_df['i_id'].isin(user_ratings['i_id'])].\
        merge(pd.DataFrame(data_with_user).reset_index(drop=True), how = 'inner', left_on = 'i_id', right_on = 'i_id')
    rated_df = rated_df.loc[rated_df.u_id==userID[0]].sort_values(by='rating', ascending = False)

    return recommendations, rated_df
```

Find 100th User's Actual Preference:

```
ratings.join(movies, on='movieId').filter('userId = 100') \
.sort('rating', ascending=False).limit(10)
```

movieId	userId	rating	timestamp	title	genres
50	100	5.0	1996-06-25 16:24:49	Usual Suspects, T...	Crime|Mystery|Thr...
293	100	5.0	1996-06-25 16:28:27	Léon: The Profess...	Action|Crime|Dram...
680	100	5.0	1996-06-25 16:58:31	Alphaville (Alpha...	Drama|Mystery|Rom...
1449	100	5.0	1997-06-09 16:38:17	Waiting for Guffm...	Comedy
235	100	4.0	1996-06-25 16:28:27	Ed Wood (1994)	Comedy|Drama
162	100	4.0	1996-06-25 16:43:19	Crumb (1994)	Documentary
223	100	4.0	1996-06-25 16:31:02	Clerks (1994)	Comedy
260	100	4.0	1997-06-09 16:40:56	Star Wars: Episod...	Action|Adventure|...
265	100	4.0	1996-06-25 16:29:49	Like Water for Ch...	Drama|Fantasy|Rom...

Recommend film for 100th user

```
## Recommend for user 100
recommendations, rated_df = funk_svd_predict(100, rating_df, movies_df)
recommendations.head(10)
```

	i_id	title	genres	u_id	prediction
20420	100556	Act of Killing, The (2012)	Documentary	100	4.680756
5467	5618	Spirited Away (Sen to Chihiro no kamikakushi) ...	Adventure|Animation|Fantasy	100	4.602811
202	214	Before the Rain (Pred dozhdot) (1994)	Drama|War	100	4.589975
18887	94466	Black Mirror (2011)	Drama|Sci-Fi	100	4.542922
13127	64241	Lonely Wife, The (Charulata) (1964)	Drama|Romance	100	4.518235
20419	100553	Frozen Planet (2011)	Documentary	100	4.512393
8799	26453	Smiley's People (1982)	Drama|Mystery	100	4.511136
4131	4278	Triumph of the Will (Triumph des Willens) (1934)	Documentary	100	4.506769
15136	77658	Cosmos (1980)	Documentary	100	4.495588
2793	2931	Time of the Gypsies (Dom za vesanje) (1989)	Comedy|Crime|Drama|Fantasy	100	4.492110

KNN Algorithm

Import Lib and init spark, dask

```
import joblib
from dask.distributed import Client
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import numpy as np
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Group 7 - Recommendation System')\
.config('spark.sql.execution.arrow.pyspark.enabled', True)\
.config('spark.driver.memory','8G')\
.config('spark.ui.showConsoleProgress', True)\
.config('spark.sql.repl.eagerEval.enabled', True)\
.getOrCreate()

client = Client()
```

Read data and convert pyspark dataframe to pandas dataframe:

```
# Data is downloaded from https://www.kaggle.com/bandikarthik/movie-recommendation-system
movies = spark.read.csv('../MovieLens/movie.csv', header=True, inferSchema=True)
ratings = spark.read.csv('../MovieLens/rating.csv',  header=True, inferSchema=True)
```

```
with joblib.parallel_backend('dask'):
    movies_df = movies.toPandas()
    rating_df = ratings.toPandas()
```

Build And Fit Model

```
model_knn= NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20)
movies_users= rating_df.head(1000000).pivot(index='movieId', columns='userId',values='rating').fillna(0)
```

```
with joblib.parallel_backend('dask'):
    mat_movies_users=csr_matrix(movies_users.values)
    model_knn.fit(mat_movies_users)
```

Recommend Movies

```
from fuzzywuzzy import process
def recommender(movie_name, data, model, n_recommendations ):
    df_movies = movies.toPandas()
    model.fit(data)
    idx=process.extractOne(movie_name, df_movies['title'])[2]
    print('Movie Selected: ', df_movies['title'][idx], 'Index: ',idx)
    print('Searching for recommendations.....')
    distances, indices=model.kneighbors(data[idx], n_neighbors=n_recommendations)
    for i in indices:
        print(df_movies['title'][i].where(i!=idx))

movie = "Heavy (1995)"
print("Recommend movies for people watched " + movie)
recommender(film, mat_movies_users, model_knn, 10)
```

```
Recommend movies for people watched Heavy (1995)
Movie Selected:  Heavy (1995) Index:  751
Searching for recommendations.....
751                                              NaN
628                            Family Thing, A (1996)
709           Halfmoon (Paul Bowles - Halbmond) (1995)
470                          In the Line of Fire (1993)
706              Visitors, The (Visiteurs, Les) (1993)
254                                  Just Cause (1995)
0                                     Toy Story (1995)
1155                             Paths of Glory (1957)
1489    Second Jungle Book: Mowgli & Baloo, The (1997)
577                       Dear Diary (Caro Diario) (1994)
Name: title, dtype: object
```

Cosine similarity and Jaccard similarity

Import Lib and init spark

```python
from pyspark.sql.functions import col, explode
from pyspark import SparkContext
# Get distance functions from Sklearn
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics.pairwise import manhattan_distances
from pprint import pprint
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.linalg import Vectors
import numpy as np
from pyspark.sql import SparkSession
sc = SparkContext
# sc.setCheckpointDir('checkpoint')
spark = SparkSession.builder.appName('Group 7 - Recommendation System')\
.config('spark.sql.execution.arrow.pyspark.enabled', True) \
.config('spark.driver.memory','8G') \
.config('spark.ui.showConsoleProgress', True) \
.config('spark.sql.repl.eagerEval.enabled', True) \
.config('spark.sql.pivotMaxValues', 100000000)\
.getOrCreate()
```

Read data:

```python
# Data is downloaded from https://www.kaggle.com/bandikarthik/movie-recommendation-system
movies = spark.read.csv('../MovieLens/movie.csv', header=True, inferSchema=True)
ratings = spark.read.csv('../MovieLens/rating.csv',  header=True, inferSchema=True)
```

```python
movies.limit(5).show()
```

```
+-------+--------------------+--------------------+
|movieId|               title|              genres|
+-------+--------------------+--------------------+
|      1|    Toy Story (1995)|Adventure|Animati...|
|      2|      Jumanji (1995)|Adventure|Childre...|
|      3|Grumpier Old Men ...|      Comedy|Romance|
|      4|Waiting to Exhale...|Comedy|Drama|Romance|
|      5|Father of the Bri...|              Comedy|
+-------+--------------------+--------------------+
```

Create new movies dataframe and change split genre to list from genres

```
movies_df = spark.createDataFrame(movies.rdd.map(lambda x: (x[0], x[2].lower()\
.replace('"',"").replace(' ',"").split('|'))), ['movieId','genre'])
movies_df.show(5)
```

```
+-------+--------------------+
|movieId|               genre|
+-------+--------------------+
|      1|[adventure, anima...|
|      2|[adventure, child...|
|      3|   [comedy, romance]|
|      4|[comedy, drama, r...|
|      5|            [comedy]|
+-------+--------------------+
only showing top 5 rows
```

Find Count of unique genre

```
#Find Count of unique Genre
count  = []
for i in movies_df.collect():
  count.extend(i[1])
print(len(count), len(set(count)))
count_genre = len(set(count))
```

```
66668 20
```

Vectorize the data and fit model

```
#For Vectorize the data
from pyspark.ml.feature import CountVectorizer
from pyspark.ml.linalg import Vectors
#Count Vectorizer Fitting
cv = CountVectorizer(inputCol="genre", outputCol="features", vocabSize=count_genre, minDF=2.0)
cvmodel = cv.fit(movies_df)
```

Transform Data using Count Vectorizer

]:

```
# Transform Data using Count Vectorizer
movies_transformed = cvmodel.transform(movies_df)
movies_transformed.show(5)
```

```
+-------+--------------------+--------------------+
|movieId|               genre|            features|
+-------+--------------------+--------------------+
|      1|[adventure, anima...|(20,[1,8,11,12,13...|
|      2|[adventure, child...|(20,[8,11,12],[1....|
|      3|   [comedy, romance]|(20,[1,3],[1.0,1.0])|
|      4|[comedy, drama, r...|(20,[0,1,3],[1.0,...|
|      5|            [comedy]|      (20,[1],[1.0])|
+-------+--------------------+--------------------+
only showing top 5 rows
```

Convert Sparse Vector to Dense

]:

```
# Convert Sparse Vector to Dense
fnldata = spark.createDataFrame(movies_transformed.select('movieId', 'features')\
                               .rdd.map(lambda x: (x[0], Vectors.dense(x[1]))), ['id', 'DenseVector'])
fnldata.take(2)
fnldata.cache()
```

]:

id	DenseVector
1	[0.0,1.0,0.0,0.0,...
2	[0.0,0.0,0.0,0.0,...
3	[0.0,1.0,0.0,1.0,...
4	[1.0,1.0,0.0,1.0,...
5	[0.0,1.0,0.0,0.0,...
6	[0.0,0.0,1.0,0.0,...
7	[0.0,1.0,0.0,1.0,...
8	[0.0,0.0,0.0,0.0,...

Calculate Cosine similarity:

```
# Test the
test_id = 45
test_vector= fnldata.rdd.lookup(test_id)
```

```
cosine_dist =spark.createDataFrame(fnldata.rdd.map(lambda x: (x[0],
float(cosine_similarity(np.array(x[1]).reshape(1, -1), np.array(test_vector)\
.reshape(1, -1))[0][0]))), ['movieId', 'cosine_sim'])
```

Calculate Jaccard similarity:

```
jaccard_sim =spark.createDataFrame(fnldata.rdd.map(lambda x: (x[0], \
float(jaccard_similarity_score(np.array(test_vector[0]) \
.reshape(1, -1), np.array(x[1]).reshape(1, -1))))), ['movieId', 'jaccard_similarity'])
```

Recommend movies for viewer who watched movie with movieId = 45

Cosine recommend:

```
cosine_recomm=cosine_dist.join(movies_df, movies_df['movieId']==cosine_dist.movieId)\
.sort('cosine_sim',ascending=False).take(10)
cosine_recomm_df = spark.createDataFrame(cosine_recomm)
cosine_recomm_df.join(movies, on="movieId")
```

movieId	cosine_sim	movieId	genre	title	genres
105835	1.0000000000000002	105835	[comedy, drama, t...	Double, The (2013)	Comedy|Drama|Thri...
147845	1.0000000000000002	147845	[comedy, drama, t...	Manson Family Vac...	Comedy|Drama|Thri...
64327	1.0000000000000002	64327	[comedy, drama, t...	Fools' Parade (1971)	Comedy|Drama|Thri...
6193	1.0000000000000002	6193	[comedy, drama, t...	Poolhall Junkies ...	Comedy|Drama|Thri...
5416	1.0000000000000002	5416	[comedy, drama, t...	Cherish (2002)	Comedy|Drama|Thri...
2438	1.0000000000000002	2438	[comedy, drama, t...	Outside Ozona (1998)	Comedy|Drama|Thri...
92906	1.0000000000000002	92906	[comedy, drama, t...	Girls on the Road...	Comedy|Drama|Thri...
82097	1.0000000000000002	82097	[comedy, drama, t...	Karthik Calling K...	Comedy|Drama|Thri...
8330	1.0000000000000002	8330	[comedy, drama, t...	Our Man in Havana...	Comedy|Drama|Thri...
30767	1.0000000000000002	30767	[comedy, drama, t...	Sitcom (1998)	Comedy|Drama|Thri...

Jaccard recommend

```
jaccard_recomm=jaccard_sim.join(movies_df, movies_df.movieId==jaccard_sim.movieId)\
.sort('jaccard_similarity',ascending=False).take(10)
jaccard_recomm_df = spark.createDataFrame(jaccard_recomm)
jaccard_recomm_df.join(movies, on="movieId")
```

movieId	jaccard_similarity	movieId	genre	title	genres
105835	1.0	105835	[comedy, drama, t...	Double, The (2013)	Comedy|Drama|Thri...
147845	1.0	147845	[comedy, drama, t...	Manson Family Vac...	Comedy|Drama|Thri...
64327	1.0	64327	[comedy, drama, t...	Fools' Parade (1971)	Comedy|Drama|Thri...
6193	1.0	6193	[comedy, drama, t...	Poolhall Junkies ...	Comedy|Drama|Thri...
5416	1.0	5416	[comedy, drama, t...	Cherish (2002)	Comedy|Drama|Thri...
2438	1.0	2438	[comedy, drama, t...	Outside Ozona (1998)	Comedy|Drama|Thri...
92906	1.0	92906	[comedy, drama, t...	Girls on the Road...	Comedy|Drama|Thri...
82097	1.0	82097	[comedy, drama, t...	Karthik Calling K...	Comedy|Drama|Thri...
8330	1.0	8330	[comedy, drama, t...	Our Man in Havana...	Comedy|Drama|Thri...
30767	1.0	30767	[comedy, drama, t...	Sitcom (1998)	Comedy|Drama|Thri...