movieratings

December 16, 2017

1 Readme

In order to create a stand alone cluster please follow the link https://www.davidadrian.cc/posts/2017/08/how-to-spark-cluster/

```
In [ ]: cd
```

```
In [ ]: import findspark
        findspark.init()
        import pyspark
        pyspark.SparkContext.setSystemProperty('spark.executor.memory', '8g')
        sc = pyspark.SparkContext(master='spark://10.0.0.167:7077', appName='akash-app')
```

```
In [ ]: %%time
        import pandas as pd
        import seaborn as sns
        from pyspark.sql import *
        from pyspark.sql import SparkSession
        from pyspark import SparkContext
        from pyspark.sql import SQLContext
        from pyspark.sql.functions import desc
        from pyspark.sql.functions import col
        from pyspark.sql import functions as F
        from pyspark import SparkContext
        from pyspark.ml.tuning import ParamGridBuilder
        from pyspark.ml import Pipeline
        from pyspark.ml.linalg import Vectors
        from pyspark.sql import Row
        from pyspark.sql import functions
        import os, sys, requests
```

```
In [ ]: %%time
        from pyspark.sql.functions import *
        sql = pyspark.SQLContext(sc)
        ratings = (sql.read
                 .format("com.databricks.spark.csv")
                 .option("header", "True")
```

```
                .option("inferSchema", "true")
                .load("ratings.csv"))

        movies = (sql.read
                .format("com.databricks.spark.csv")
                .option("header", "True")
                 .option("inferSchema", "true")
                .load("movies.csv"))

        ratings.printSchema()
        movies.printSchema()
```

In [ ]:
```
%%time
from pyspark.sql import Row
from pyspark.sql import functions
from pyspark.sql.types import *
topMovieIDs = ratings.groupBy("movieId").count().orderBy("count", ascending=False).cach
topMovieIDs.show()
top10 = topMovieIDs.take(10)
```

In [ ]:
```
%%time
#counts of ratings for each movie
movies_counts = ratings.groupBy(col("movieId")).agg(F.count(col("rating")).alias("count
movies_counts.show()
```

In [ ]:
```
%%time
ratings.describe().show()
```

In [ ]:
```
%%time
mat_users = ratings.select('userId').distinct().count()
mat_movies = ratings.select('movieId').distinct().count()
mat_ratings = ratings.count()
print ("Number of different users: " + str(mat_users))
print ("Number of different movies: " + str(mat_movies))
print ("Number of total ratings: " + str(mat_ratings))
ratings_count = ratings.groupBy('rating').count()
ratings_countsort = ratings_count.sort(desc('rating'))
ratings_countsort.show()
matrix_size = mat_users * mat_movies
matrix_size
perct_matrix = (100*mat_ratings/matrix_size)
perct_matrix
```

In [ ]:
```
!pip install seaborn
```

In [ ]:
```
%%time
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
        ratings_pan = ratings.toPandas()
        sns.violinplot([ratings_pan.rating])
```

```
In [ ]: %%time
        temp_mat = pd.DataFrame([[mat_users,mat_movies,mat_ratings, matrix_size, perct_matrix]]
        temp_mat
```

The above matrix shows that 1.64% of the matrix is filled.

```
In [ ]: from pyspark.ml.evaluation import RegressionEvaluator
        from pyspark.ml.recommendation import ALS
        from pyspark.ml import Pipeline
        from pyspark.sql import Row
        import numpy as np
        import math
```

2 Training Model

```
In [ ]: %%time
        model = ALS(userCol="userId", itemCol="movieId", ratingCol="rating").fit(ratings)
        predictionmodel = model.transform(ratings)
        predictionmodel.show()
```

3 Testing the training Model

```
In [ ]: %%time
        test_model = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="
        print("Root Mean Square Error:" + str(test_model.evaluate(predictionmodel)))
```

```
In [ ]: %%time
        avg_rat = ratings.select('rating').groupBy().avg().first()[0]
        print ("The average rating in the dataset is: " + str(avg_rat))

        test_model1 = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="
        print("Root Mean Square Error: " + str(test_model1.evaluate(predictionmodel.na.fill(avg
```

4 Split the dataset into training, validation and testing

```
In [ ]: %%time
        training, test = ratings.randomSplit([70.0, 30.0])
```

```
In [ ]: training.show()
```

```
In [ ]: %%time
        modelals = ALS(userCol="userId", itemCol="movieId", ratingCol="rating").fit(training)
        predictionmodeltrain = model.transform(training)
        predictionmodeltrain.show()
```

```
In [ ]: %%time
        test_model_train = RegressionEvaluator(metricName="rmse", labelCol="rating", predictio
        print("Root Mean Square Error:" + str(test_model_train.evaluate(predictionmodeltrain))

In [ ]: %%time
        avg_rat = ratings.select('rating').groupBy().avg().first()[0]
        print ("The average rating in the dataset is: " + str(avg_rat))

        test_model_train1 = RegressionEvaluator(metricName="rmse", labelCol="rating", predictio
        print("Root Mean Square Error: " + str(test_model_train1.evaluate(predictionmodel.na.fi

In [ ]: test_model_train1 = RegressionEvaluator(metricName="rmse", labelCol="rating", predictio
        print("Root Mean Square Error: " + str(test_model_train1.evaluate(predictionmodel.na.dr

In [ ]: def multiALS(data, k=3, userCol="userId", itemCol="movieId", ratingCol="rating", metric
            modeleval = []
            for i in range(0, k):
                (training_df, test_df) = data.randomSplit([k-1.0, 1.0])
                als = ALS(userCol=userCol, itemCol=itemCol, ratingCol=ratingCol)
                model = als.fit(training_df)
                predictions = model.transform(test_df)
                test_model_train1 = RegressionEvaluator(metricName=metricName, labelCol="rating
                model_evaluation = test_model_train1.evaluate(predictions.na.drop())
                print ("Loop " + str(i+1) + ": " + metricName + " = " + str(model_evaluation))
                modeleval.append(model_evaluation)
                print(modeleval)
            return print(modeleval)

In [ ]: %%time
        modeleval = print(multiALS(ratings, k=4))

In [ ]: alsfinal = ALS(maxIter=10, regParam=0.1, rank=6, userCol="userId", itemCol="movieId", r
        modelfinal= alsfinal.fit(training)
        finalprediction = modelfinal.transform(test)
        finalprediction = finalprediction.filter(col('prediction') != np.nan)
        rmse = test_model_train.evaluate(finalprediction)
        print ("the rmse for optimal grid parameters with cross validation is: {}".format(rmse)

In [ ]: %%time
        np.random.seed(12345)
        user_id = np.random.choice(mat_users)
        newuser_ratings = ratings.filter(ratings.userId == user_id)
        newuser_ratings.sort('rating', ascending = True).take(10)
        newuser_ratings.toPandas()['rating'].hist()
```

List of all unrated movieIds with more than 10 ratings.

```
In [ ]: %%time
        #Collecting distinct Movie ID's
```

```
        newuserrated_movieIds = [i.movieId for i in newuser_ratings.select('movieId').distinct
        #taking counts of ratings above 10
        newusermovieIds = [i.movieId for i in movies_counts.filter(movies_counts.counts>20).sel
        newuserunratedmovieIds = list(set(newusermovieIds) - set(newuserrated_movieIds))
```

In [ ]:
```
%%time
import time
num_ratings = len(newuserunratedmovieIds)
cols = ('userId', 'movieId', 'timestamp')
timestamps = [int(time.time())] * num_ratings
userIds = [user_id] * num_ratings
newuserpreds = sql.createDataFrame(zip(userIds, newuserunratedmovieIds, timestamps), co
```

In [ ]:
```
newuserpreds = modelfinal.transform(newuserpreds)
```

In [ ]:
```
newuserpreds.describe().show()
```

In [ ]:
```
newuserpreds.sort('prediction', ascending=False).take(10)
```