Working With Spark Dataframes

Author : Amit Singh Chowdhery

Agenda

- Need of dataframes.
- Detailing out Dataframes Architecture and DSL
- Methods to convert RDD to Dataframes
- Operating on dataframes in PySpark
- Interacting with Dataframes using PySpark SQL
- Overview of Pyspark MLlib with ML libraries

Problem Statement

IMDb is an online database of information related to films, television series, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews.

At the very initial level , we will restrict ourselves to conduct an **Exploratory Data Analysis** (EDA) on the dataset to get insights, clean the data and will try to answer a few questions by writing code in Spark.

Type of questions that we'll be solving today be like:

a. Which comedy movie has the most ratings and for the same return the title and the number of rankings.

   Answer : American Beauty but how ?

b. Understand viewership by age group for targeted marketing, retention activities or market expansion.

c. Insight to the top 25 movies by viewership rating, indicating the type of movies or sequels that perform better.

d. Find the average rating of each movie for which there are at least 100 ratings. Order the result by average rating in decreasing order.

The **challenge** here is that We need to Answer this question by joining two datasets which contain almost 20M ratings given on 62K movies collected over the last 20 years(1995-2015)?

We already have explored this through RDD in previous class and this was amazing as millions of records are being processed fast and we are able to get desired results.

Solution Through RDD:

To process above data via RDD , we follow below mentioned steps:

```
a. Load the file and create RDD from it.
ratingRDD = sc.textFile("../data-files/movielens/ratings.csv")
movieRDD = sc.textFile("../data-files/movielens/movie.csv")

b. After this, in order to get final output we need to apply below transformations:


                Transformation 1 : group ratings by unique movie
                Transformation 2 : join with the filtered comedies
                Transformation 3 : map to extract title and number of rankings


comedyRDD = ratings.groupBy(lambda x: x[1]).keyBy(lambda x: x[0]) \
.join(movies.filter(lambda x: x[2].count('Comedy')!=0).keyBy(lambda x: x[0])) \
.map(lambda (key, (k,v)): (key, v[1], k[1].__len__())) \
.max(key=lambda x: x[2])

c. Print the result:
comedyRDD.collect()
```

Ok..we got the results

But writing code using RDD is not recommended?

Few reasons are :

a. **Hard To Use** : In our code , we created RDD by hard coding few values ,which is never the preferred way.
b. **Slow Speed :** RDD was not fit, data became extremely large and in multiple experiments it has been proved the same.
c. **Outdated : Yes!!!!** RDD is outdated. As Spark matures, the need for more data warehousing, big data analytics starts increasing and hence RDD starts diminishing and paves the way for SparkSQL

How can I write code in a simpler way, maybe SQL type ?

- In real world scenarios, most of the time processing of structured data is on rise.
- There is a need to have some framework that can do faster processing with SQL-like syntax which more than half of the world knows.
- So Spark committers brought SparkSQL(DATAFRAMES**)** in spark 1.3.0 version.
- The idea behind dataframe is it allows processing of a large amount of structured data.

What is SparkSQL?

- SparkSQL is a module for structured data processing that is built on top of core spark.
- SparkSQL module provides an abstraction called **DataFrame** which simplifies working with structured datasets.
- It provides capability to read and write data in variety of formats:
    - JSON
    - CSV
    - Parquet
- Several programming languages such as Java,R,python and scala are supported.
- Row Level updates is possible

What is a Data Frame ?

As per official spark documentation :

==========

"**A data frame is a data structure that organizes data into a 2 dimensional table of rows and columns , much like a spreadsheet**."

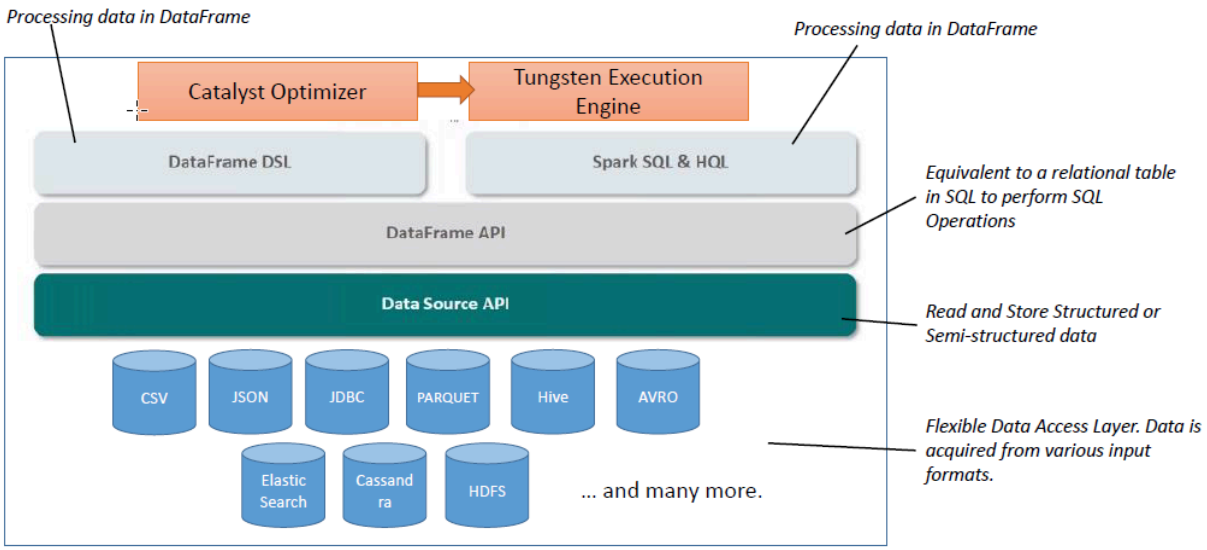
OR

"A data frame is an integrated data structure with an easy to use API for simplifying distributed big data processing. "

==========

It is an extension of the spark RDD API optimized for writing code more efficiently while remaining powerful.

Architecture



1. Catalyst Optimizer

- Responsible for improving the performance of user programs (SQL Query/DataFrame APIs).
- It converts a query plan into optimized query plan.

Offers both rule based and cost based optimization

- **Rule Based** : How to execute the query from a set of defined rules.
- **Cost Based**: Generates multiple execution plans and chooses the lowest cost plan.

2. Tungsten Execution Engine

- Generates code and executes it in the cluster in the distributed fashion.
- The engine builds upon ideas from modern compilers and MPP databases.

3. Data Source API

Used to read and store structured data and semi-structured data in Spark SQL.Data Source API can load files from multiple file formats.

4. DataFrame API

i. Dataframe API converts the data that is read through Data Source API into tabular columns.
ii. Dataframe is a distributed collection of data organized into named columns.
iii. Tabular columns help in performing SQL operations.
iv. **Row** : Represented as a record in DataFrame.

5. DataFrame DSL

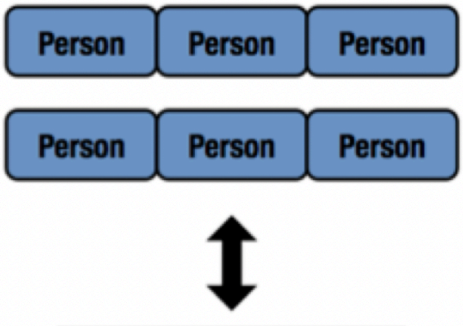
v. Spark SQL introduces dataframes which provide support for structured and semi structured data.
vi. To manipulate the same, by default spark sql provides a domain specific language(DSL) in scala,java, python or R.

Why is Data Frame getting popular?

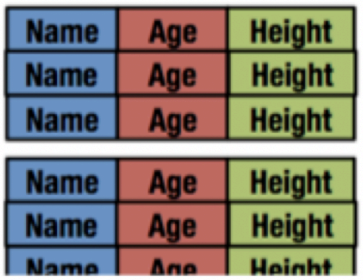
- Dataframe API was introduced in Spark 1.3 version
- Representing a distributed collection of rows organized into named columns like that of the RDBMS table but with richer optimizations under the hood.
- Unlike RDD, dataframe keeps track of its schema.

DataFrame = RDD + Schema (aka SchemaRDD)

RDD: “Immutable partitioned collection of elements”.



SchemaRDD: “An RDD of Row objects that has an associated schema”



Note : schemaRDD has been renamed to dataframes.

Let's revisit our same code with dataframes way :

```
Step 1 : Create sparksession object as :

from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()


Step 2.Load the file and create a dataframe from it.
mydf = spark.read.format("csv").option("header","true").load("/user/file.csv")

Step 3: Write transformations which are needed to get desired result?

from pyspark.sql import functions as F
```

```
wordsDF = wordsDF.filter(F.col("genres") == "Comedy")
wordsDF = wordsDF.select("genres","title")


Step 4 : Now lets see the final output by writing another query.
wordsDF.show()
```

Surprise Surprise!!!!!! same output but this time no hardcoding and writing code was damn fun.

Supported Data Types:

Numeric Types	
1.	IntegerType
2.	FloatType
3.	DoubleType
String Types	
1.	StringType
2.	VarcharType
3.	CharType
Boolean Types	
1.	BooleanType
Date Types	
1.	TimestampType
2.	DateType
Complex Type	
1.	ArrayType(elementType,containsNull)
2.	MapType(keyType , valueType,valueContainsNull)
3.	StructType(fields)

How to create a dataframe?

a. **Csv,Json,Parquet,Avro,Orc** : Any file format having metadata inside can be used to create data frames by writing the simplest line of code.

```
df = spark.read \
   .option("delimiter", "\t") \
   .option("inferSchema", "true") \
   .csv("../data-files/movies.csv")
```

b. **RDBMS** : Spark dataframes can be created from any RDBMS databases like oracle,mysql,postgres etc as shown below :

DBMS

```
df = spark.read\
            .format('jdbc')\
            .option('url', 'jdbc:mysql://localhost:3306/db')\
            .option('driver', 'com.mysql.jdbc.Driver')\
            .option('dbtable','new_table')\
            .option('user','root')\
            .load()
```

```
df.show()
```

```
+---+-------------+
| id|random_number|
+---+-------------+
| 12|         1042|
| 15|         2041|
| 18|           93|
| 24|          276|
| 27|         2252|
```

c. **RDD**: This is a little tricky and important one, yes RDD can be converted into dataframes. We have 2 methods for doing the same. We will cover this upcoming topic

What are the operations can we perform on Dataframes?

Data frames supports 2 types of operations:

a. Transformation
b. Action

1. Transformation: Spark data frames carry the same legacy from RDDs. One can perform transformations on them to generate new data frames.

   Commonly Used Transformations are :

   - Selection : Select,selectExpr,withColumn.drop
   - Filter : filter
   - Sort : sort,orderBy,sortWithinPartitions
   - Join: inner,outer, left outer, right outer,cross
   - Aggregation : avg,mean, count,max,min,sum
   - Window: Ranking,Analytical, aggregate
   - Sample: Sample,SampleBy,

```
filterDF = wordsDF.filter(F.col("genres") == "Comedy")
selectionDF = filterDF.select("genres","title")
```

2. Action: When we call an Action on a Spark dataframe all the Transformations get executed one by one. This happens because of Spark Lazy Evaluation which does not execute the transformations until an Action is called.

Commonly Used Actions are :

- show()
- head()
- first()
- take(n)
- collect()
- saveAsFile()

```
filterDF.select("genres","title").show()
```

Now we will be able to see our final output.

Hey I am not from any Scala,Python, Java background but i too want to play with dataframes.

How can Spark help me?

Ans: Temporary Views/Global Views

So what exactly is this Temp view?

- It is an in-memory table that is scoped to the cluster in which it was created.
- It is just like a view in a database.
- Once you have a view, you can execute SQL on that view.
- Spark offers five data frame methods to create a view.
    - registerTempTable (Spark < = 1.6) : deprecated
    - createOrReplaceTempView (Spark > = 2.0) ⟶
    - createOrReplaceGlobalTempView (Spark > = 2.0)
    - createTempView (Spark > = 2.0)
    - createGlobalTempView(Spark > = 2.0)

The local temporary view is only visible to the current spark session. However, a Global temporary view is visible to the current spark application across the sessions.
Example :

```
moviesDF = spark.read.format("csv").option("header","true").load("/user/file.csv")
ratingsDF =  spark.read.format("csv").option("header","true").load("/user/ratings.csv")

#Register the DataFrame as a global temporary view
moviesDF.createGlobalTempView("movies")
ratingsDF .createGlobalTempView("ratings")

#apply groupBY
finallDF= sql("select movieid,rating from movies mov,ratings rat where mov.movieId = rat.movieId and genre = 'comedy' group by rating ")

# Show the final comedy movie having the most ratings
finalDF.collect()
```

Learning from this Program

- Dataframes have been mapped to temporary tables.
- For processing, we have used normal SQL syntax
- Internally SQL is converted into RDD like operation and is executed across all nodes.

What if I want to have output in UPPERCASE?

Here is the list of functions natively supported by SparkSQL

https://spark.apache.org/docs/latest/api/sql/index.html

But sometimes we need to create our own functions , we call them UDF :

- **User-Defined functions** allow you to register a custom function and use the same within SQL.
- UDFs are a very popular way to add some advanced functionality that is already not present in the in-built functions.
- Spark SQL makes it super easy to define and use UDFs.

To get desired result below are the steps:

1. Create a function to convert a string in Uppercase.

```
def to_uppercase(str):
        resStr=""
        arr = str.split(" ")
        for x in arr:
        resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "
        return resStr
```

2. Register function as UDF

```
spark.udf.register("to_upper", uppercase, StringType())
```

3. It's time to see output by Applying new UDF on our result DF as:

```
finalDF.to_upper().collect()
```

```
Ans :AMERICAN BEAUTY,1999
```

What if I have data in RDD and want to work with DF?

There are 2 methods for converting any RDD to dataframes:

a. Reflection Based
b. Programmatic Based

Reflection based:

- Spark SQL can convert an RDD of Row objects to a DataFrame, inferring the data types
- Rows are constructed by passing a list of key/value pairs as kwargs to the Row class.
- The keys of this list define the column names of the table, and the types are inferred by sampling the whole dataset

Procedure to achieve same:

```
from pyspark.sql import Row

sc = spark.sparkContext

# Load a text file and convert each line to a Row.
ratings= sc.textFile("../data-files/movielens/ratings.csv)
```

```
parts = movies.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(userId=p[0], movieId=int(p[1]),rating=int(p[2]))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = spark.createDataFrame(people)
schemaPeople.createOrReplaceTempView("rating")

# SQL can be run over DataFrames that have been registered as a table.
avgRating= spark.sql("SELECT avg(rating) as finalRating,userid ,movieid FROM rating
GROUP BY movieid order by finalRating limit 10")

# The results of SQL queries are Dataframe objects.
# rdd returns the content as an :class:`pyspark.RDD` of :class:`Row`.
ratingsDF= avgRating.rdd.map(lambda p: "Name: " + p.name).collect()
for ratings in ratingsDF:
   print(ratingsDF)
```

Programmatic based:

- Lets you build schema and apply to an already existing RDD
- Allows you to build Dataframes when you do not know the columns and their types until runtime.

Procedure to achieve same:

1. Create an RDD of Rows from the original RDD;
2. Create the schema represented by a StructType matching the structure of Rows in the RDD created in Step 1.
3. Apply the schema to the RDD of Rows via createDataFrame method provided by SparkSession.

```
# Import data types
from pyspark.sql.types import StringType, StructType, StructField

sc = spark.sparkContext

# Load a text file and convert each line to a Row.
ratings= sc.textFile("../data-files/movielens/ratings.csv")
parts = ratings.map(lambda l: l.split(","))
# Each line is converted to a tuple.
finalRDD= parts.map(lambda p: (p[0], p[1].strip()))

# The schema is encoded in a string.
schemaString = "movieid ratings"

fields = [StructField(field_name, StringType(), True) for field_name in schemaString.split()]
schema = StructType(fields)

# Apply the schema to the RDD.
schemaRatings = spark.createDataFrame(ratings, schema)

# Creates a temporary view using the DataFrame
schemaPeople.createOrReplaceTempView("ratings")

# SQL can be run over DataFrames that have been registered as a table.
results = spark.sql("SELECT avg(rating) as finalRating,userid ,movieid FROM rating GROUP
BY movieid order by finalRating limit 10")

# result
results.show()
```

#####Optional no need to cover this#######

Spark ML Module :

Problem Statement :

Till now, we have done EDA on dataset by using:

a. RDD
b. Dataframe

Now, data has been cleaned and we will take the final step to build a recommendation engine.

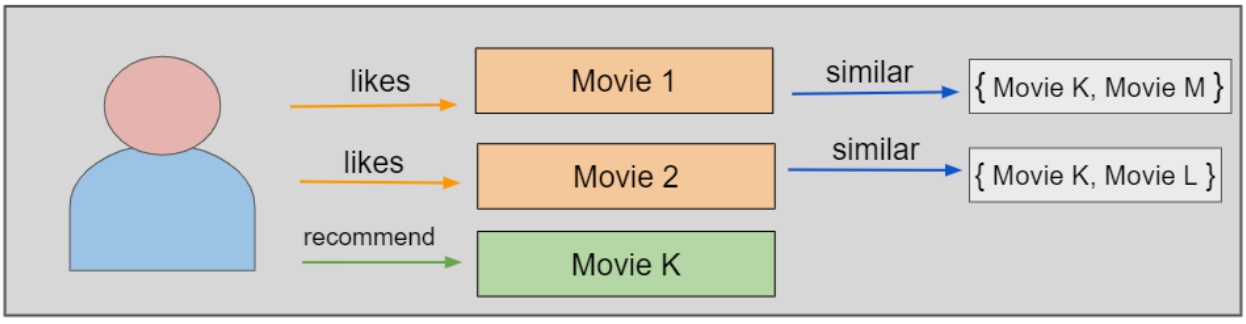
So what exactly is a recommendation engine ?

“Recommender systems are the systems that are designed to recommend things to the user based on many different factors.Companies like Netflix, Amazon, etc. use recommender systems to help their users to identify the correct product or movies for them.”

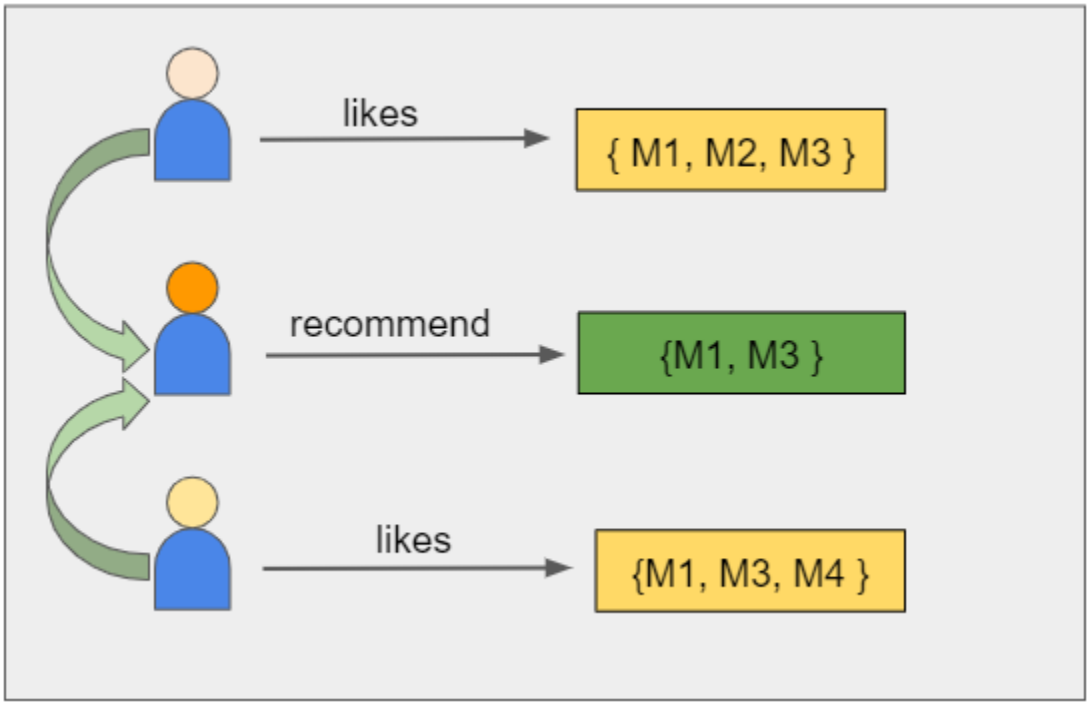
The two widely used approaches for building a recommender system are the

a. Content-based filtering (CBF)
b. Collaborative filtering (CF)

Content-based filtering : uses similarities in products, services, or content features, as well as information accumulated about the user to make recommendations and this is the most widely used.



Collaborative filtering : relies on the preferences of similar users to offer recommendations to a particular user.



So the question is how to build a recommendation algorithm and by using which approach?

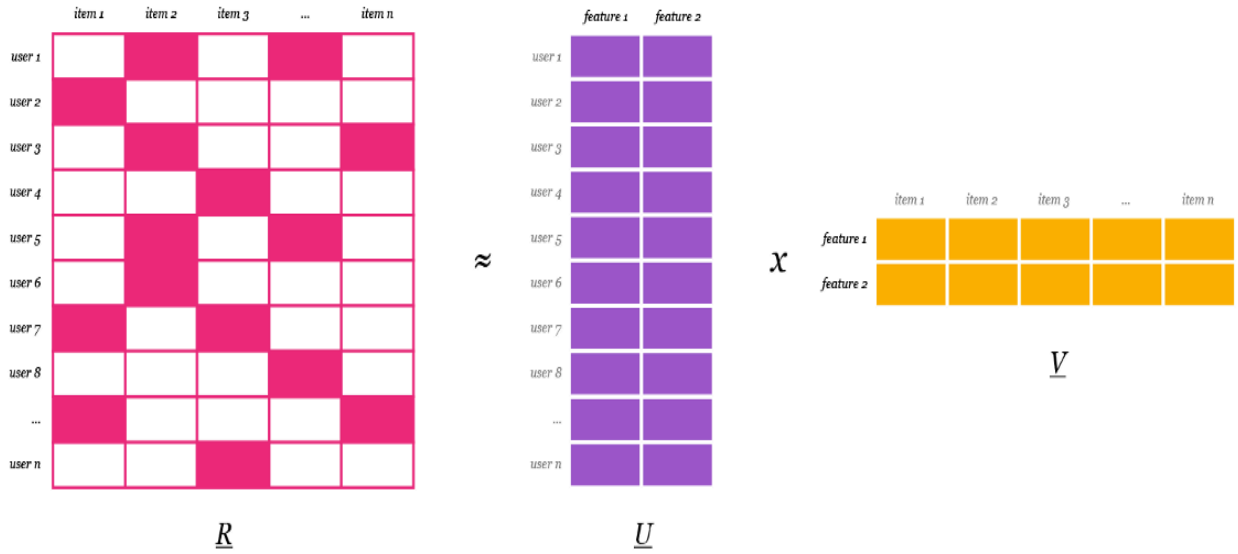
Solution:

In our use case, we will be using **Collaborative filtering - ALS algo (Alternating Least Square) ,** reason for choosing same are :

- It is an optimization technique to solve the matrix factorization problem.
- It achieves good performance and has proven relatively easy to implement.
- Scalability: ALS runs its gradient descent in parallel across multiple partitions of the underlying training data from a cluster of machines
- It belongs to a broad class of latent-factor models which try to explain observed interactions between a large number of users and items/movies through a relatively small number of unobserved, underlying reasons/factors.

What is the Logic behind ALS?

ALS is an iterative optimization process where we for every iteration try to arrive closer and closer to a factorized representation of our original data.



- Let's take an original matrix R of size u x i with our users, items and some type of feedback data.
- We then want to find a way to turn that into one matrix with users and hidden features of size u x f and one with items and hidden features of size f x i. In U and V we have weights for how each user/item relates to each feature.
- What we do is we calculate U and V so that their product approximates Ras closely as possible: R ≈ U x V.
- By randomly assigning the values in U and V and using least squares iteratively we can arrive at what weights yield the best approximation of R.

- The least squares approach in its basic forms means fitting some line to the data, measuring the sum of squared distances from all points to the line and trying to get an optimal fit by minimizing this value.
- With the alternating least squares approach we use the same idea but iteratively alternate between optimizing U and fixing V and vice versa. We do this for each iteration to arrive closer to R = U x V.

What will be the Steps to build our own recommendation engine?

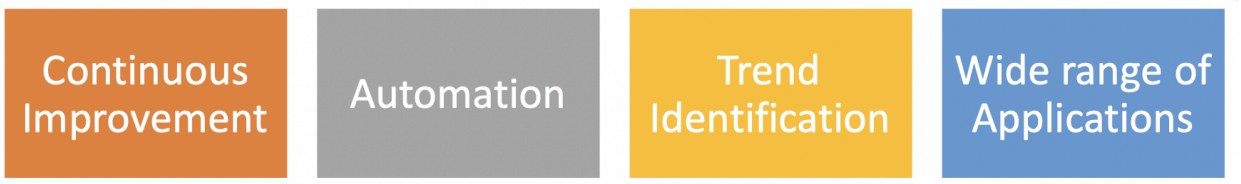
a. Prepare the dataset and load it in a dataframe.
b. Build out an ALS model and train it
c. Test the model (Hyperparameter tuning and cross validation)
d. Check the best model parameters
e. Fit the best model and evaluate predictions
f. Make Recommendations

What is Machine Learning?

- Machine learning is a subset artificial intelligence.
- ML allows software applications to become more accurate at predicting outcomes without being explicitly programmed.
- Machine learning algorithms use historical data as input to predict new output values.

What is the need of ML?

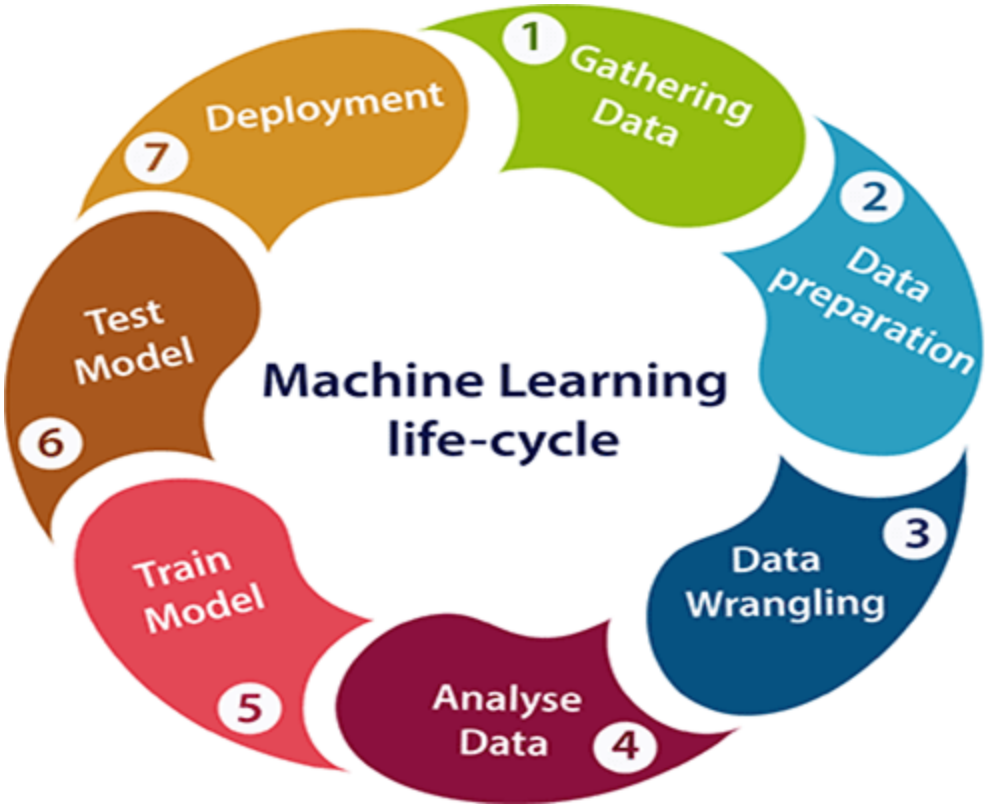
- Machine learning has become a significant competitive differentiator for many companies.
- Leading organizations such as Facebook, Uber make machine learning a central part of their operations.
- Machine learning is important because it gives enterprises a view of trends in customer behavior
- It helps organizations to understand the business operational patterns, as well as supports the development of new products.
- Many organizations use machine learning in manufacturing to reduce costs, enhance quality control and improve supply chain management.



Some machine learning terminologies

Terminologies	Explanation
Dataset	Structured Collection of data like Db tables
Instance	A row in dataset
Feature	Inputs used for prediction, generally a column in dataset
Target	A variable or column algo learns to be predicted
Algorithm	A set of rules to achieve a goal
ML Algorithm	A program used to make a machine learn model from data.
Training	A process in which ML Algorithm is run against Dataset to learn patterns and get model
Model	Output of a machine learning algorithm that run on datasets
Validation	A process to verify the accuracy of a model.
Label	Output that we get by running models on different datasets.
Prediction	A value that is being outputted by model.

What are the Steps involved in Machine Learning?



A typical ML pipeline has below mentioned steps:

- **Gathering Data :** The goal of this step is to identify and obtain all data-related problems.
- **Data preparation :** Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

- **Data Wrangling :** It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step.
- **Analyze Data :** The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome.
- **Train the model :** In this step we train our model to improve its performance for better outcome of the problem.
- **Test the model :** In this step, we check for the accuracy of our model by providing a test dataset to it.Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.
- **Deployment : I**f the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system.

How Spark helps in making ML scalable and easy to use?

Spark MLlib is a module on top of **Spark Core that provides machine learning** primitives as APIs. MLlib in Spark is a scalable Machine learning library that discusses both high-quality algorithms and high speed.

Algos included :

1. Regression
2. Classification
3. Clustering
4. Pattern mining
5. Collaborative filtering.

Lower level machine learning primitives like **generic gradient descent optimization** algorithms are also present in MLlib.

Spark.ml is the primary Machine Learning API for Spark. The library Spark.ml offers a higher-level API built on top of DataFrames for constructing ML pipelines.

Spark MLlib tools are given below:

1. **ML Algorithms** : ML Algorithms form the core of MLlib. These include common learning algorithms such as classification, regression, clustering, and collaborative filtering.
    a. **Transformer** : Transformer implements a method transform(), which converts one DataFrame into another, generally by appending one or more columns.
    b. **Estimator** :An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. Technically, an Estimator implements a method fit(), which accepts a DataFrame and produces a Model, which is a Transformer.
2. **Featurization**: Featurization includes feature extraction, transformation, dimensionality reduction, and selection.
3. **Pipelines** : A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow. It also provides tools for constructing, evaluating and tuning ML Pipelines.
4. **Persistence** : Persistence helps in saving and loading algorithms, models, and Pipelines. This helps in reducing time and efforts as the model is persistence, it can be loaded/ reused any time when needed.
5. **Utilities** :Utilities for linear algebra, statistics, and data handling. Example: mllib.linalg is MLlib utilities for linear algebra

Building a movie recommendation engine using ALS

1. Load and explore the data

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Recommendations').getOrCreate()
movies = spark.read.csv("movies.csv",header=True)
ratings = spark.read.csv("ratings.csv",header=True)
ratings.show()
```

Here we have created movies and ratings dataframes.

2. Split the dataset into train and test

We are splitting our dataset into an 80:20 rule where 80% of data will be used for training the model and remaining 20% for testing the accuracy of the model.

```
# Create test and train set
(train, test) = ratings.randomSplit([0.8, 0.2], seed = 2020)
```

3. Build out an ALS model

```
# Import the required functions
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# Create ALS model
als = ALS(
        userCol="userId",
        itemCol="movieId",
        ratingCol="rating",
        nonnegative = True,
        implicitPrefs = False,
        coldStartStrategy="drop"
)
```

When using simple random splits as in Spark's CrossValidator or TrainValidationSplit, it is actually very common to encounter users and/or items in the evaluation set that are not in the training set.

a. Hyperparameter tuning and cross validation

ParamGridBuilder:

- We will first define the tuning parameter using param_grid function.
- We have only chosen 4 parameters for each grid.
- This will result in 16 models for training.

```
# Import the requisite packages
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator
# Add hyperparameters and their respective values to param_grid
param_grid = ParamGridBuilder() \
        .addGrid(als.rank, [10, 50, 100, 150]) \
        .addGrid(als.regParam, [.01, .05, .1, .15]) \
        .build()
```

RegressionEvaluator: Then define the evaluator, select rmse as metricName in evaluator.

```
# 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))
```

CrossValidator:

- Now feed both param_grid and evaluator into the crossvalidator including the ALS model.
- We have chosen the number of folds as 5. (Feel free to experiment with parameters).

```
# Build cross validation using CrossValidator
cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid,
evaluator=evaluator, numFolds=5)
```

4. Check the best model parameters

Let us check, which parameters out of the 16 parameters fed into the crossvalidator, resulted in the best model.

```
# Print "Rank"
print("  Rank:", best_model._java_obj.parent().getRank())
# Print "MaxIter"
print("  MaxIter:", best_model._java_obj.parent().getMaxIter())
# Print "RegParam"
print("  RegParam:", best_model._java_obj.parent().getRegParam())
```

5. Fit the best model and evaluate predictions

- Now fit the model and make predictions on the test dataset.
- Based on the range of parameters chosen we are testing 16 models, so this might take a while.
- The RMSE for the best model is 0.866 which means that on average the model predicts 0.866 above or below values of the original ratings matrix.

```
#Fit cross validator to the 'train' dataset
model = cv.fit(train)
#Extract best model from the cv model above
best_model = model.bestModel
# View the predictions
test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(RMSE)
```

6. Make Recommendations:

- Lets go ahead and make recommendations based on our best model.
- recommendForAllUsers(n) function in als takes n recommendations.
- Let's go with 5 recommendations for all users.

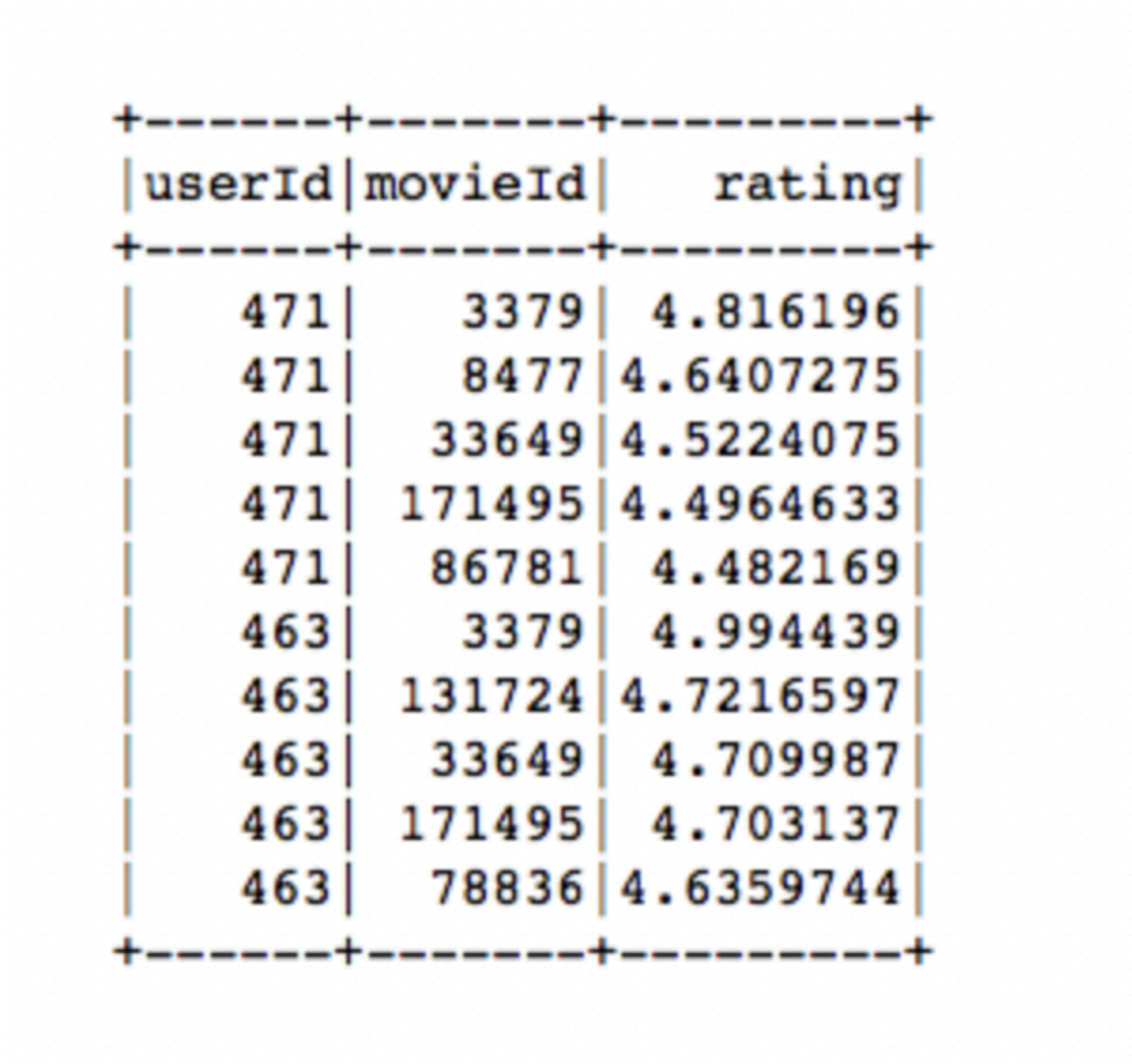
```
# Generate n Recommendations for all users
recommendations = best_model.recommendForAllUsers(5)
recommendations.show()
```

7. Convert recommendations into interpretable format

- The recommendations are generated in a format that easy to use in pyspark
- As you can observe, the recommendations are saved in an array format with movie id and ratings
- To make these recommendations easy to read and compare to check if recommendations make sense, we will want to add more information like movie name and genre
- Then we will explode array to get rows with single recommendations.

```
nrecommendations = nrecommendations\
  .withColumn("rec_exp", explode("recommendations"))\
  .select('userId', col("rec_exp.movieId"), col("rec_exp.rating"))
nrecommendations.limit(10).show()
```

And Result is:



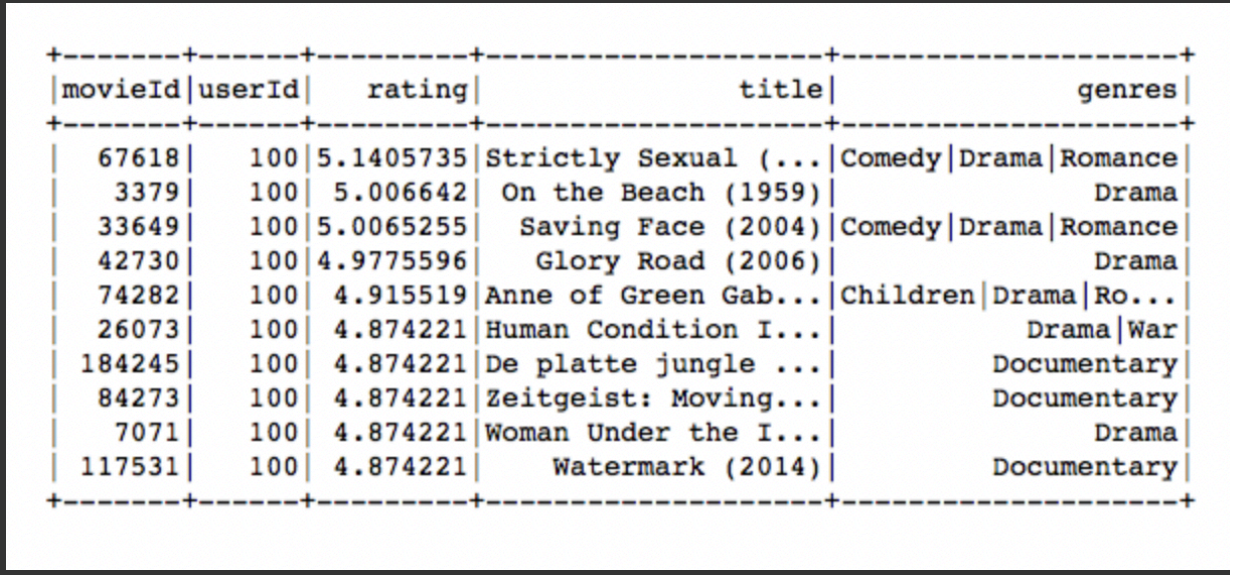
userId	movieId	rating
471	3379	4.816196
471	8477	4.6407275
471	33649	4.5224075
471	171495	4.4964633
471	86781	4.482169
463	3379	4.994439
463	131724	4.7216597
463	33649	4.709987
463	171495	4.703137
463	78836	4.6359744

But Do the recommendations make sense?

To check if the recommendations make sense, join movie name and genre to the above table.Lets randomly pick **100th user** to check if the recommendations make sense.

100th User's ALS Recommendations:

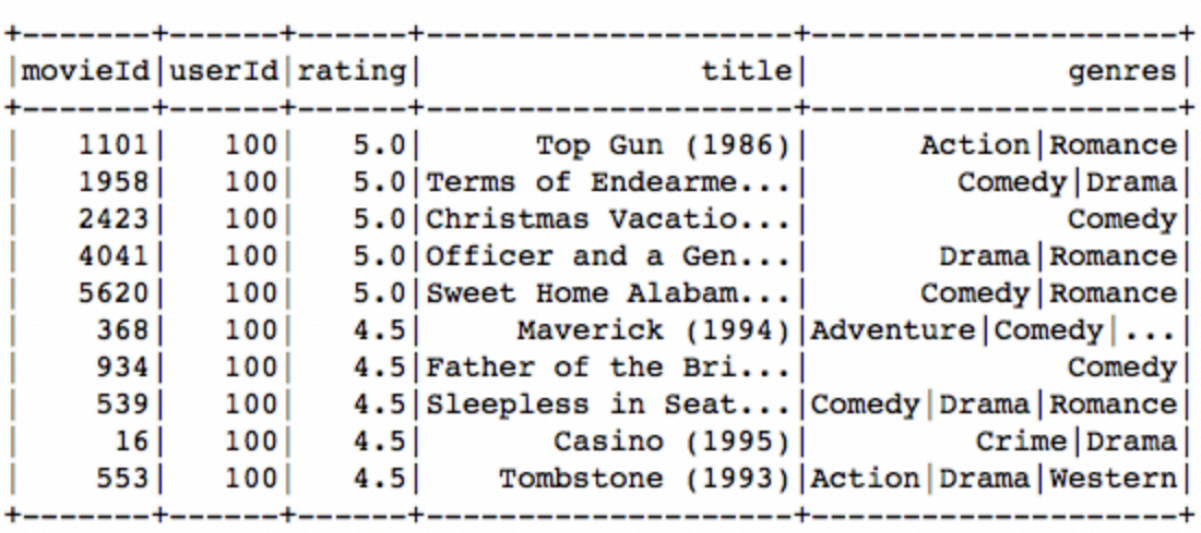
```
nrecommendations.join(movies, on='movieId').filter('userId =
```



```
100').show()
```

100th User's Actual Preference:

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



movieId	userId	rating	title	genres
1101	100	5.0	Top Gun (1986)	Action|Romance
1958	100	5.0	Terms of Endearme...	Comedy|Drama
2423	100	5.0	Christmas Vacatio...	Comedy
4041	100	5.0	Officer and a Gen...	Drama|Romance
5620	100	5.0	Sweet Home Alabam...	Comedy|Romance
368	100	4.5	Maverick (1994)	Adventure|Comedy|...
934	100	4.5	Father of the Bri...	Comedy
539	100	4.5	Sleepless in Seat...	Comedy|Drama|Romance
16	100	4.5	Casino (1995)	Crime|Drama
553	100	4.5	Tombstone (1993)	Action|Drama|Western

The movie recommended to the 100th user primarily belongs to comedy, drama, war and romance genres, and the movies preferred by the user match very closely with these genres.

Code : https://github.com/scaleracademy/dsml-de/tree/main/spark/Recommendation_Engine_MovieLens.ipynb