# Introduction to Apache Spark and Implicit Collaborative Filtering in PySpark

Dileep Pasumarthi Follow
Jan 2, 2020 · 4 min read

**Apache Spark:**

Apache Spark is open-source, fast distribution computing framework. It provides APIs for programming clusters of machines with parallelism and fault tolerance [3]. It was originally developed in UC Berkely and was later donated to the Apache foundation.

It is compatible with Hadoop and can run Hadoop clusters using Yarn, Kubernetes, Mesos or a standalone mode. It can process data in many famous distributed data systems like Cassandra, HDFS, HBase etc. It is a very good platform for batch processing (like map-reduce) and for tasks like streaming and machine learning. Its popularity has been climbing rapidly over last few years with many organizations using it for various data processing tasks.

**PySpark:**

Spark provides high-level APIs in Scala, Java, Python and R. Python's wrapper for Spark is called PySpark. PySpark is one of the leading languages for performing data analysis tasks and building machine learning applications. A wide range of adoption for Python language for data analysis makes PySpark intuitive and very easy for anyone to onboard.

Platforms like Azure Databricks, Microsoft's HDInsights etc. makes PySpark very easy to setup and start experimentation very easily using high computing power. In this technology review I focus on what Implicit Collaborative Filtering is and how to implement it in PySpark. I have used Azure Databricks for the experimentation below.

**Implicit Collaborative Filtering:**

Collaborative Filtering (CF) is a state-of-art technique used by recommender systems. The motivation for CF comes from the idea that people often get the best recommendations from someone with tastes like themselves.

Photo by
Mohamed BEN ELLEFI on Profile-based Dataset Recommendation

Implicit CF used for scenarios where there is no explicit data but have access to lots of implicit data like extracting user's interests from search history, predicting if user likes the movie from viewing history (how long he watched the movie etc). This technique is adopted by companies like Amazon and Netflix from recommending products and movies. It is used by search engine companies like Bing and Google for determining user interests.

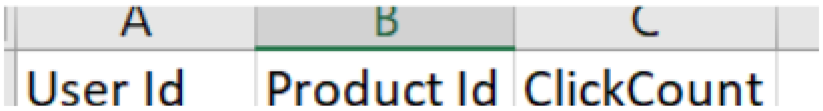
Explicit CF uses low-rank matrix factorization to approximate the entries in the matrix. A very famous application for Explicit CF is in Movies domain. There is a clear difference between low scores (that user doesn't like the item) and missing values (meaning user hasn't rated the item). The blank spaces in the user-item matrix doesn't need to be computed during optimization process, so it is computationally sparse.

On the other hand, Implicit CF is more complicated because low number of interactions between a user and an item could mean that user doesn't like the item or user hasn't heard of that item. It is also computationally intensive because you need to compute values for all entries in the matrix during the optimization.

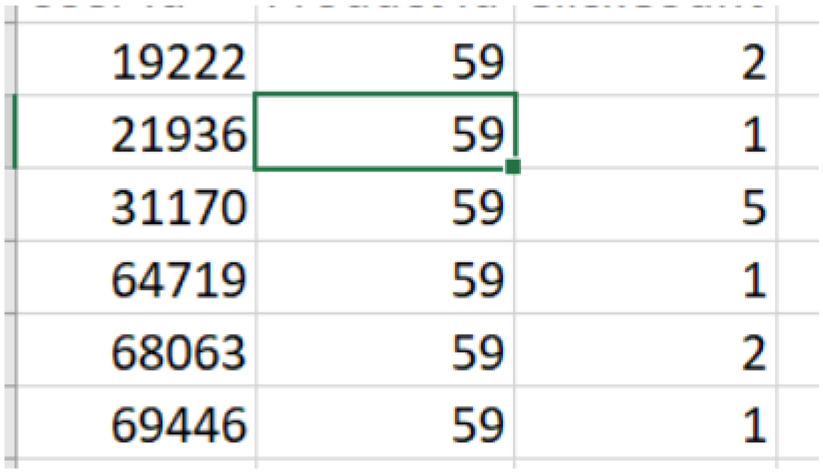
PySpark has inbuilt support for Implicit CF. We will discuss few important APIs here

***Sample Input:***

Here is a snapshot of sample input. This is a dataset is for number of times a given user searched for a product in a search engine.



| A | B | C |
|---|---|---|
| User Id | Product Id | ClickCount |



| | | |
|---|---|---|
| 19222 | 59 | 2 |
| 21936 | 59 | 1 |
| 31170 | 59 | 5 |
| 64719 | 59 | 1 |
| 68063 | 59 | 2 |
| 69446 | 59 | 1 |

***Fitting the model:***

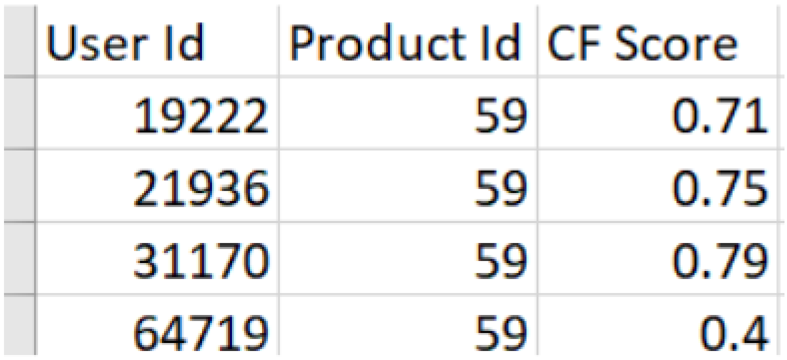
```
als = ALS(maxIter=30, regParam=0.1, implicitPrefs=True, userCol=”userId”,
itemCol=”productId”, ratingCol=”ClickCount”)

savedModel = als.fit(training)
```

Processing time depends on various factors. Using ~50 cores and ~200 gigs of RAM (DDRv2 instances on databricks) it takes about ~1.5 hr to train on 20 million users and 100k products.

***Predicting on a given set:***

```
predictions = savedModel.transform(test)
```

**Sample output**



| User Id | Product Id | CF Score |
|---|---|---|
| 19222 | 59 | 0.71 |
| 21936 | 59 | 0.75 |
| 31170 | 59 | 0.79 |
| 64719 | 59 | 0.4 |

***Recommend Items for a user:***

```
# Generate top 10 product recommendations for each user
```

> *userRecs = savedModel.recommendForAllUsers(10)*

***Recommend Users for an Item:***

> # Generate top 10 user recommendations for each product
>
> *productRecs = savedModel.recommendForAllItems(10)*

***Generating recommendations for a subset:***

> # Generate top 10 product recommendations for a user subset
>
> *userSubsetRecs = savedModel.recommendForUserSubset(userSubset, 10)*
>
> # Generate top 10 user recommendations for a specified set of products
>
> *productSubSetRecs = savedModel.recommendForItemSubset(productsSubset, 10)*

Serving real-time users using the model built is also possible using various services like Azure cognitive services, Databricks ML Flow etc. and is not covered in this article.

**References**:

1. PySpark official documentation: https://spark.apache.org/docs/latest/api/python/index.html
2. Apache Spark: https://spark.apache.org/faq.html
3. Apache Spark: https://en.wikipedia.org/wiki/Apache_Spark
4. Implicit CF: http://yifanhu.net/PUB/cf.pdf
5. PySpark CF: https://spark.apache.org/docs/latest/ml-collaborative-filtering.html

Big Data Spark Recommendation System Recommender Systems Pyspark