***1. Use Case Name* – Job Recommendation Engine using Implicit Data**

***2. Use Case URL*** (GitHub or web site URL) - https://towardsdatascience.com/large-scale-jobs-recommendation-engine-using-implicit-data-in-pyspark-ccf8df5d910e

***3. Introduction*** – Offering relevant recommendations in almost every consumer facing business is very much essential to convert prospects into customers. On a greater scale we take recommendations in our daily lives from system without even thinking about it, for example: LinkedIn and Glassdoor giving recommendations about which job we should apply to based on our previous job search behavior. In this use case I have built job recommendation engine to predict what jobs users will apply based on their number of clicks to a Job ID.

***4. Dataset used*** – The data file used is an open source data file name "RE_data" which consists of two data set: "job_clicks.csv" (100014 rows) and "jobs.csv" (9125 rows) which can be downloaded from the following source address: https://github.com/Akxay/recommendation_engine/tree/master/RE_data
"jobs.csv" consists of information about the job and "job_clicks.csv" consists of users' information regarding how many times they clicked on particular job. What's unique about the "job_clicks.csv" dataset is that it is implicit data, that is, the information is not provided intentionally but gathered from available data streams, either directly or through analysis of explicit data. For example: I purchased an item, just because I bought an item doesn't mean I liked it. It might be possible that I didn't bought it for someone else. How are we going to make recommendations in that case? For that we are going to see how to build a recommendation engine below using Spark tool.

***5. Technical Details***: Following are the important parameters considered: U: Users, P: Preference, R: Recordings denoted as number of clicks here. All the code is written in Python. In python there is this **mllib** recommendation module which is machine learning library and within that there is this **ALS** class and that meets all of our ALS needs. Within ALS class there is train implicit function and it takes in a few arguments. It takes in the dataset in the form of RDD. ALS parameters are rank, lambda, alpha and iterations: **rank** is the width of the matrices U and P, **lambda** is a parameter which prevents overfitting of the model, **alpha** is the tuning parameter. This parameter turns up in the equation for confidence and it relates to the scale of the recordings. It can be used to enforce a strong relationship, a strong increase in confidence between no recording and one recording. **Iterations** is how many times we oscillate between the process that generates the matrices U and P. Iterations unlike rank and lambda is not quite so straightforward to tune for. The RMSE is monotonically non increasing in iterations, so, as we increase the number of iterations, our model is never going to get worse but it's unrealistic to set iteration to high value like 1000 because that's going to take a lot of time to run. For prediction, there is this **predictAll** function that takes in the RDD of user item pairs and returns the prediction under the model that I trained. **Other aspects of Spark are:**
- **pyspark.sql.functions:** to apply sql built in functions to RDD dataset like row_number(), get current date and time stamp while displaying top recommendations and desc to display recommendation in descending order of job preferences.
- **SparkConf:** to run a Spark application on the local/cluster, we need to set a few configurations and parameters, this is what SparkConf helps with
- **SparkContext:** When we run any Spark application, a driver program starts, which has the main function and SparkContext gets initiated. The driver program then runs the operations inside the executors on worker nodes.
- **SQLContext:** used for initializing the functionalities of Spark SQL.
- **SparkSession:** SparkSession helps to interact with Spark functionality and allows programming Spark with DataFrame and Dataset APIs.

***6. Results***: First Model(left) is having training set of 60%, validation set of 20% and test set of 20%: **Rank** 5, **Lamb da** 10, **Iterations** 20, **Alpha** 40 and **RMSE** is 56.82. Second model(right) is having training set of 20%, validation s et of 30% and test set of 50% **Rank** 5, **Lambda** 10, **Iterations** 20, **Alpha** 20, **RMSE** is 56.99.

| user_id | job_recommendations | job_category | preference_confidence |
|---|---|---|---|
| 26 | 27611 | Business Intellig... | 1.12 |
| 26 | 106489 | Retial | 1.06 |
| 26 | 5971 | Banking | 1.05 |
| 26 | 7022 | HR | 1.05 |
| 26 | 7502 | HR | 1.05 |

**First Model**

| user_id | job_recommendations | job_category | preference_confidence |
|---|---|---|---|
| 26 | 4369 | Finance | 1.16 |
| 26 | 46578 | IT | 1.13 |
| 26 | 35836 | IT | 1.12 |
| 26 | 8970 | Business Intellig... | 1.11 |
| 26 | 1957 | IT | 1.11 |

**Second Model**

***7. Insight***: Business implications/impacts of Job recommendation engine are:

- Job Recommendation System can help to increase correct matching of recruiters with job seekers, which will encourage more job seekers and recruiters to use the site, enabling better algorithm and increase in performance.
- It also increases retention, since continuously catering to users' preferences makes them more likely to remain loyal subscribers of the service.
- This in turn help job sites to increase revenue by 10-50% caused by accurate "People also viewed" or "Jobs you may like" recommendations.
- This can be used to provide recommendations and job prediction and based on location of the users. Not only this, we can also provide recommendation to join groups, like posts and add connections.
- Users can save a lot of time when served tailored suggestions for jobs necessary for further research.
- We can evaluate this further on how business metrics are impacted, for example: there can be few job IDs that are rated peculiarly because that might be clicked only by one user a lot of times, so, if someone's factor vector (likes/ dislikes in layman's terms) is similar to that person's factor vector for whom we are providing recommendation, the algorithm is going to suggest them really weird job IDs which user clicked, but we would rather return something in which we have more confidence and for that we can filter out all the Job IDs which are clicked by fewer than say 20 people and that could improve the results a lot.

**8.** ***Debugging Details****:* Challenges:

- If I do not create instance of SparkContext in the code, it throws error stating that RDD has no attribute "toDF". Hence, I created spark context which was not created earlier.

I made following changes to the code:

- I created another model with different training, validation and test set.

Any achievements you have made?

- Filtered only those jobIDs which have been clicked by 20 or more users so that we can have significant difference between preference confidence and preference confidence are not impacted by those users who clicked only one JobID. In a way this can avoid false positives.

What are key features of coding practices in this project?

- Created SparkContext which represents a connection to the spark computing cluster.
- Created RDD, that can be run in parallel.
- Did not return all the elements of a large RDD back to the driver. Avoid using collect and count on large RDDS, instead displayed only 5 or less elements and displayed only 15 top recommendations.
- Created functions for code reusability.
- Used proper naming conventions and provided comments and headings wherever necessary.

**9.** ***Conceptual Framework:***

Lecture 07: Applied MLLib framework and ALS from Spark which we learned in lecture 07
Lecture 03: I did my research in microservices for future work. Thinking about LinkedIn, we can't build a system in a notebook to observe 470 million customers, so we will have to build a system which runs in the cloud and every component runs in the container. Microservices architecture are popular way to build applications by structuring them as a collection of cooperating light weight services.

***10.5 V's****:*

| **Volume** | **Velocity** | **Variety** | **Veracity** | **Value** |
|---|---|---|---|---|
| I have used only 100000 data. But this engine can be used to train and test large volume of data. | I focused on getting knowledge from the data already available. However, this engine can also be used to get recommendations from data as they arrive, that is, focus on real time big data. | The data here is structured. The engine has the capability to be used on unstructured data combined with various external sources of data. | To increase confidence in the accuracy of data collected, it can be more real by collecting it from number of sources which are authentic and trust worthy. It reflects the emphasis of the need for quality data in a Big Data system. Hence this enables the engine to be capable of more accurate. | For my recommendation engine, sorted data is not required which is fulfilled here. Data should be random and free from biasness. It should represent the true population. Real world data is very sparse. Since, the data I have used here is not sparse, the engine is capable of using more real data because of the use of matrix factorization. |

***11. References (at least 5 references including the main project source, lecture, presentation, etc.)***


Main Project Source
https://towardsdatascience.com/large-scale-jobs-recommendation-engine-using-implicit-data-in-pyspark-ccf8df5d910e

Data
https://github.com/Akxay/recommendation_engine/tree/master/RE_data

Lectures
https://seattleu.instructure.com/courses/1584669/files/folder/Lecture?preview=64671670
https://seattleu.instructure.com/courses/1584669/files/folder/Lecture?preview=64725619
https://seattleu.instructure.com/courses/1584669/files/folder/Assignment/hw7?preview=64726281
https://seattleu.instructure.com/courses/1584669/files/folder/Assignment/hw8?preview=64739743

Other Sources
https://databricks.com/session/building-an-implicit-recommendation-engine-with-spark

http://yifanhu.net/PUB/cf.pdf

https://towardsdatascience.com/collaborative-filtering-and-embeddings-part-1-63b00b9739ce

https://github.com/GoogleCloudPlatform/spark-recommendation-engine/blob/master/pyspark/find_model_collaborative.py

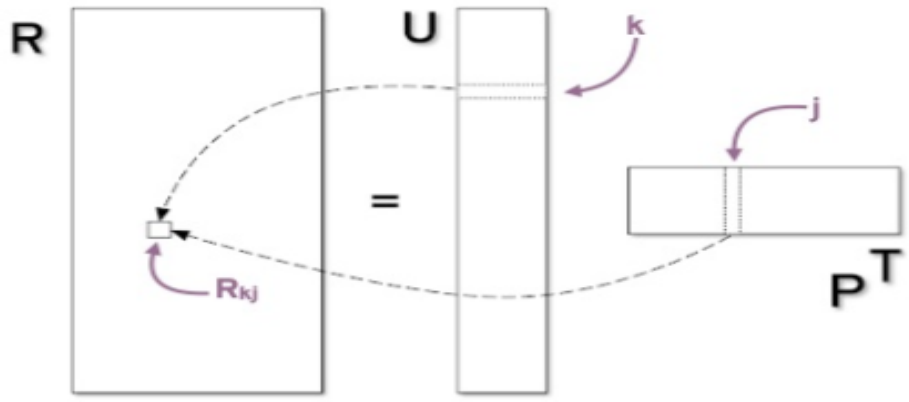
https://www.jobspikr.com/blog/how-do-job-recommendation-engines-work/


***12. A list with short descriptions of all fields in dataset used***

Fields in jobs.csv:
jobID – numeric data type and unique, primary key
job_category – variable character and non-unique

Fields in job_clicks.csv:
userId – numeric data type and non-unique, since one user can click on multiple jobIds.
jobId – numeric data type and non-unique, since one jobId can be clicked by multiple users.
Clicks – numeric data type and non-unique. It represents number of clicks by user on respective jobId.

***13. Appendix – any other contents that can support your project***




**Matrix Factorization used in Collaborative Filtering**