Daimler technical coding test

# Proposed solution

1. Environment and testing

Proposed solution has been designed using Python POO paradigm and tested in Python 3.7.3 version using Spyder IDE. Aditionally the code is uploaded into a github private repository.

Solution is composed of the file “Daimler\_technical\_test.py”, the dataset “Daimler-test-data.json”, this document “Overview daimler technical coding test” and the instructions for the test “Daimler-DE-test.zip”

1. Testing

For testing the solution Daimler\_technical\_test.py and Daimler-test-data.json files must be in the same directory. Process can be started executing .py file.

1. About the code

Code is written following POO paradigm, distributed in classes and methods. Some logs have been added for testing and debugging purposes (log format like “[Info] Starting main process at 05:35:37”)

* Main

Firstly, the main function is invocated, in main function all functionality required for the test is executed. Main code is inside a try except block, in case an exception is raised an error is printed ([Error] description…)

* Dataset read process

Once main is running “inputFileObject” is created from class “InputFile”. In the init method “inputfile” attribute is declared with relative path value “Daimler-test-data.json”.

“read\_file” method is invocated for reading the dataset where literally json is read and dataframe is transposed for having sku codes as row index and att-a…j as columns.

* Dataset validation

An extra validation functionality has been added. Firstly null values are checked, all values must have “att-a-“… “att-j-“ substring and a number as a suffix. Is there is an error in the dataset an exception is raised. Some lambda functions have been used for avoiding making for loops.

* Request user sku code

requestedSkuObject is created and inserted\_sku code request in init method.

Sku code must have “sku-“ prefix and exists in the dataset. Exception is raised If any condition is not filled in validate\_sku method.

* Get recommendation

Once recommendationObject is created scoring process starts. In this process a linear search is made for each column. We find values in each column that are equal to the value in user’s selected sku code, a counter is increased in a new score column. Linear search has been designed using a lambda function.

Once we have the score of each row dataframe is sorted in descending order and truncated to 11. This way we have a dataframe with the sku selected and the 10 more similar in the dataset. Is possible we have sku with same score so a new logic must be implemented in future to support this case.

At the end the recommendation is printed row by row in json format as the example.

# ¿What algorithm could be the best case for finding recommendations?

If we try to approach a scalable solution, we can check these points:

- If we focus in algorithms, in the technical test example we can appreciate a behavior similar to collaborative filtering. User needs to choose one sku code giving explicit feedback and a linear search is being made into the dataset to find similarities. This solution works for the test but is not scalable if the dataset grows.

One better algorithm can be binary search that reduce to half the complexity of the search from O(n) to O(log n).

If dataset is uniformly distributed complexity can be reduced using **interpolation search.** Complexity would be log(log(n)) being n the number of elements to be searched. This algorithm can scale better due the reduction of search time.

- A scalable solution can be to migrate the dataset into a distributed nosql database. This way we can use the DB indexing and sharding power in the search and scale better.

- As an ideal and scalable solution, we could migrate the whole process into a Hadoop cluster and use spark for parallelizing the process. In this case if we use a search algorithm through a distributed dataframe we can share the search complexity between the nodes of the cluster. Dataset can be distributed into a storage like s3 or hdfs or even migrated into a distributed database nosql database like the previous point. The code needs to be migrated to scala, java or pyspark.

- Supposing sku are products user has bought or search, we can storage user information to provide collaborative-filtering recommendation, comparing products between other users. The problem in this case would be the cold start.

- ML recommendation systems can be contemplated, If we have user information like age, range salary, number of children, marital status,… a ML model can be trained using the decisions of all users in the platform to provide recommendations depending of those parameters.