MBOM AUTOMATION

Automate with ease

Contents

Pre-requisite :

H2 –Database (2.1.214)

Python (3.9)

ActiveMQ (5.16.5)

Gateway

TC2X

AWC (5.2)

Teamcenter 12

Apache Tomcat (9.0.71)

Other libraries required for tasks related to machine learning, there is one requirements.txt file, which consist all of them, and they can easily be installed with the help of pip command instantly. e.g pip3 install –r requirements.txt

```
>>pip3 install –r requirements.txt
```

ENVIRONMENT SETUP

H2 Database

Database should be active

TC2XProcessor

Start TC2X Processor either directly from executing TC2XStart.bat or from TC2XProcessor.jar e.g.

```
java –jar TC2XConsumer.jar
```

Gateway

Run Gateway Service using Start.bat file

ActiveMQ

Should be running

TOMCAT

Should be running

Teamcenter

Should be running

These above mentioned services must be active while working with this tool.

AWC CONFIGURATION

For consuming ML APIs' , we need to make some modification in UI, we have added a button which will open a panel , inside this panel we will show our corresponding recommendations for selected TOP.

One can directly dump those changes in src inside stage folder and run a couple of npm commands e.g. < npm run refresh > and <npm run publish >

```
>> npm run refresh
```

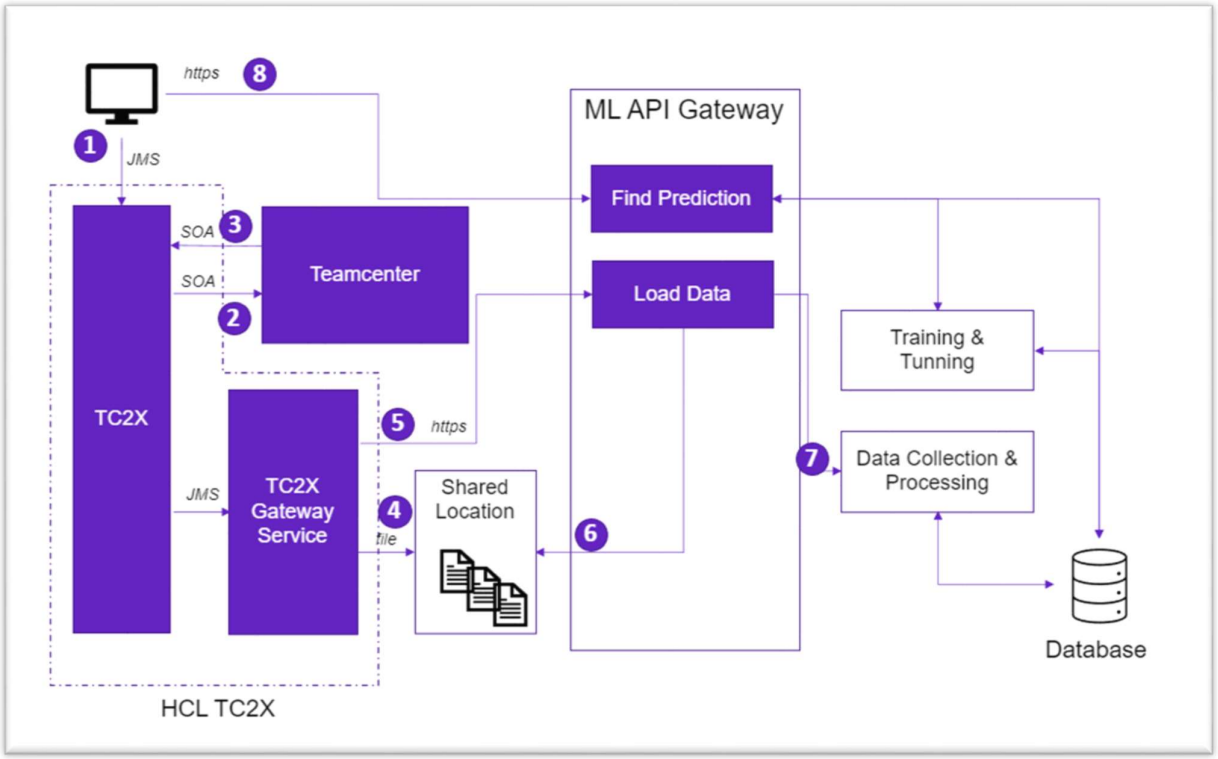
```
>> npm run publish
```

ML API

Start Flask API

```
>> python api.py
```

HIGH LEVEL OVERVIEW



1. MBOM Data Export Request sent to TC2X
2. Request Processed by TC2X and sent export request to TC
3. Data Exported from TC will be returned to TC2X
4. TC2X create a file and put at shared Location
5. TC2X call ML LOAD_DATA API
6. API will get the file from that shared Location
7. API will read the file and store required data to H2 Database during Model Training.
8. Client Request for prediction for selected MBOM Node

H2 Database

- For H2 Database default port is 8080, need to change this port to another free available port.

- We need to create one database e.g. HAMF_MBOMDB , our script will automatically generate required tables with columns in that given database also take care of other operations related to it.

- Note: Our MBOMA tool API has nothing to do with port on which H2 database is working.

TC2XProcessor

- This Processor will extract required MBOM data from Teamcenter on getting request from workflow and dump it to a particular location.

- We need to configure what number of attributes we want to extract, in which sequence to, in what location we want to store that data.

- One dataset attached in template Item we can configure all these things in that file.

.

Gateway

- This is a node based service acting as a bridge between TC2X Processor and our machine learning APIs'.
- Objective Function of this service is to transfer extracted MBOM data from TC2X Processor to our LOAD_DATA API.
- Communication between TC2X and Gateway takes place with the help of JMS Provider (Active MQ in our case).
- Particular Queues' are designated for request and response for data extraction.
- This gateway continuously looking a particular response queue, when so ever we got response from TC2X in that queue , our gateway consumed this response and sends it to our ML API.

ML API

- Currently we have two active APIs' one is LOAD_DATA and other one RECOMMEND.
- LOAD_DATA: This API get MBOM data in a pre-defined format mentioned in dataset file attached in template item in Teamcenter.
- RECOMMEND: This API is directly consumed at AWC end, when we want to predict recommendations.

Solution Overview

This tool will help users to automate MBOM Creation, it will provide suggestions based on earlier created MBOMs .It can be extended to different Plants and Product-Lines as well.

This tool will learn as the number of MBOM created with it increased. It will learn those patterns and behavior from data and will predict parts and assemblies on that basis .As the data fed to model will increased , efficiency or accuracy of relevant suggestion also expected to be better.

We have used combination of Collaborative filtering and Singular Valued Decomposition with SGD by taking **Yehuda Koren** Paper on **Factor in the Neighbors: Scalable and Accurate Collaborative Filtering** as reference with small changes. We have also introduced one target function which is responsible for calculating similarity value for each node with respect to all other existing nodes , it is also capable of generating new nodes and edges between them and calculate their value of importance which can be updated in synchronization with upcoming data

Problem Statement

To suggest possible assemblies, sub-assemblies and parts ,based on their past behavior/hierarchy to automate or to make MBOM generation easy.

Constraints

1. Here one node can not be dealt as a individual entity because it can have multiple children.
2. These children can also have children upto n levels .
3. If we are recommending any assembly , it means we are recommending their children as well , to add them in that MBOM.
4. All these children and top of that assembly altogether acts as an entity.

Our solution has four layer , every layer contribute to output in different manner .

These layers are as follows ;

1. Foundational Layer
2. Collection / Group Approach
3. Contextual Layer
4. Categorical Suggestions

Foundational Layer

We have adopted Graphical approach for extracting features from existing relations among different assemblies and their children from data.

As the number of MBOM's increased, we updated these importance value with each iteration on that basis between those existing relations in this manner our solution tries to learn those patterns and behavior from data, and recommend suggestions with respect to it.

We have formulated one Target Function to calculate importance value, in such a way that when we give extracted feature for a pair(source node ,destination node) as input it return importance value for that pair , conclusively , this target function is responsible for calculating importance value between a pair with the help of extracted feature.

- Each part /assembly is represented as Node .
- Relation among nodes are represented with the help of edges.
- Importance of relation is quantized with weight of edge.
- Number of incoming edges to any node is indegree and number of outgoing edges from that node in outdegree of that node.
- Sum of indegree and outdegree is 'Degree' of that Node

- Node can be of two type source node or destination Node

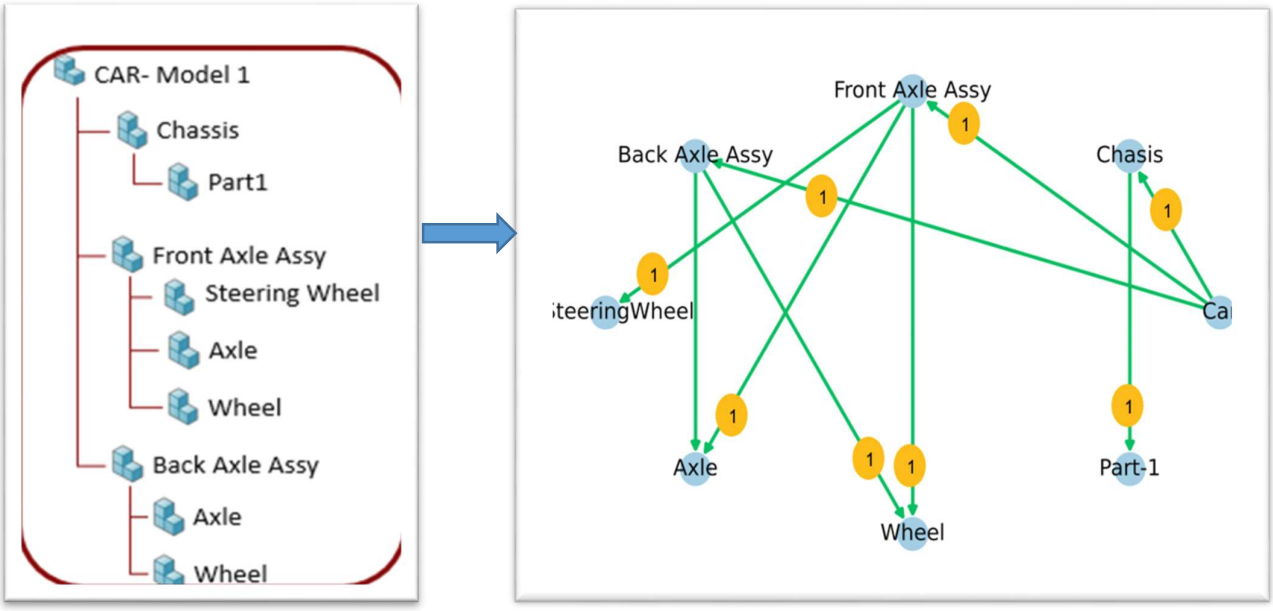
For now this very first iteration we are using simple graphical features ,for each pair of assemblies/parts we have used three attribute for now ,

n_edges (int) : number of edges between source and destination node
weight (int) : weight of the corresponding edge between nodes
degree (int) : number of incoming and outgoing edges from destination node

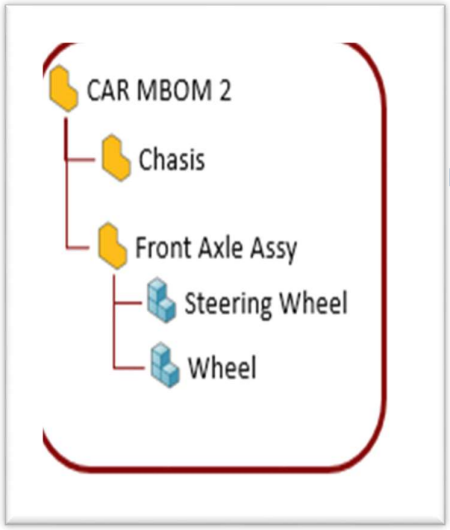
Target Function

This is a custom function specifically formulated in a manner that, it will give more weightage to the **n_edges** in compare to **weight** attribute and if their value is less then importance given to them will be more ,as e^-x is a continues decreasing function and log is a continuously increasing function so combination of these functions is very helpful in getting good target value. At last we are adding **degree** attribute to this value so that pairs with greater degree will have higher priority.

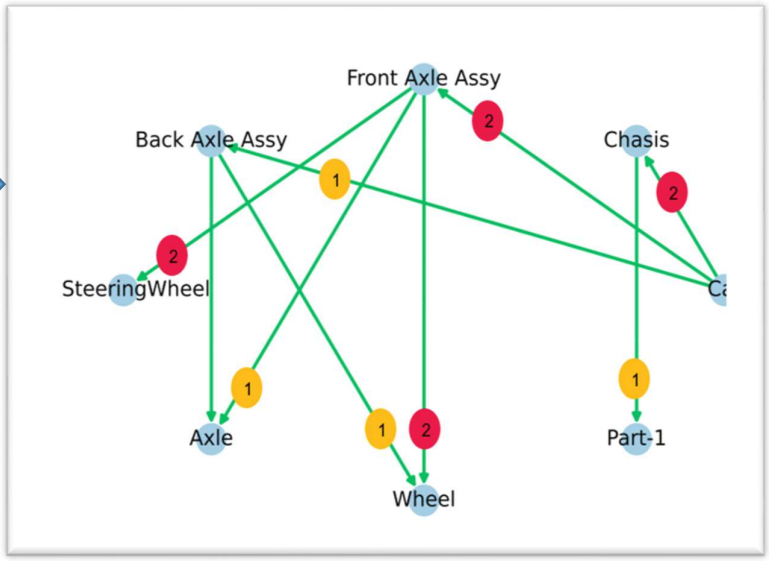
MBOM **GRAPHICAL REPRESENTATION**



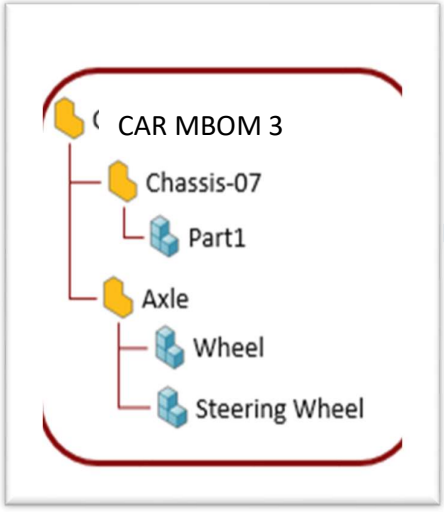
2ND MBOM



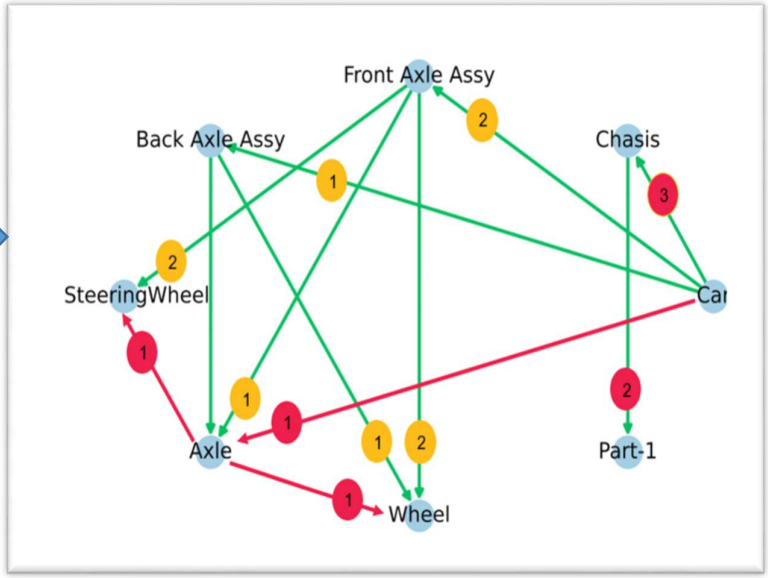
Updated Graphical Representation



3rd MBOM



Updated Graphical Representation



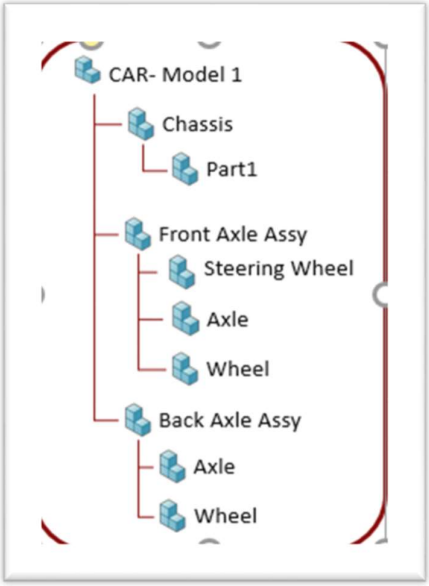
- Default weight for every edge is 1.
- Red weights are updated ones , and yellow weight are those with no changes.
- Green edges are already generated edges .
- Red edges are newly generated edges .

- In 2nd MBOM we have merely updated some weights.
- In 3rd MBOM we have added some new edges and updated some weights as well.

This is just a glimpse , how our base approach will work.

Collection / Group Approach

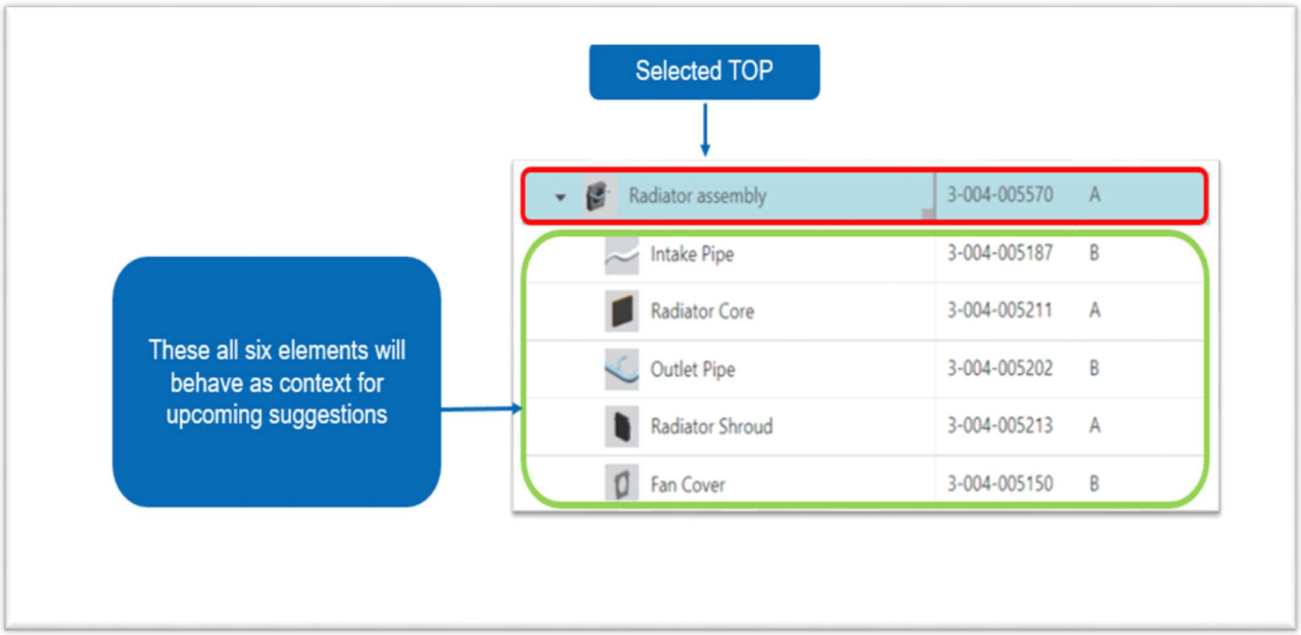
- Two assemblies with more than X% of similar child will be considered belongs to a group or collection .
- In this case target value for unknown children of same group/collection will be shared with other assemblies as well.
- Our approach is like , if there are some assemblies with mostly similar children then there different connection or occurrence can also be utilized for finding possible relationship with other assemblies as well.



In this assembly we can see that Back axle Assm have two children and these two same children are also available in Front Axle Assm as well so they can belong to one same group , in other words our thought process behind this is that these two different top surely share some common behavior or common ground so further connections of those parts will be shared with each other.

Contextual Layer

- Context : With our frame of reference it mean what parts we have used under that selected TOP behaves as neighbors or surrounding or context to the upcoming suggestion.
- Here these already used parts altogether contributes in deciding about upcoming suggestiosn according to current usage of parts and assemblies under that selected top.
- If we have added any new assembly which is not available in our EBOM and earlier MBOMs as well in our existing MBOM ,after adding merely one element to it user will be able to contextual suggestion after then.



- We have got a vector representation of these available parts.
- Altogether combination of these contextual vector will be responsible for upcoming suggestions.

Categorical Suggestions

Internally we are dealing this problem statement as regression , so giving any kind of accuracy in number for suggestions is not feasible , so for that we have divided our suggestions into multiple categories , and these categories consist of suggestions based on their relevance of usage according to requirement.

There are three main categories

a) Green
b) Yellow
c) Red

Green

Those suggestions which can directly be added under selected TOP and all their children also exist in corresponding EBOM.

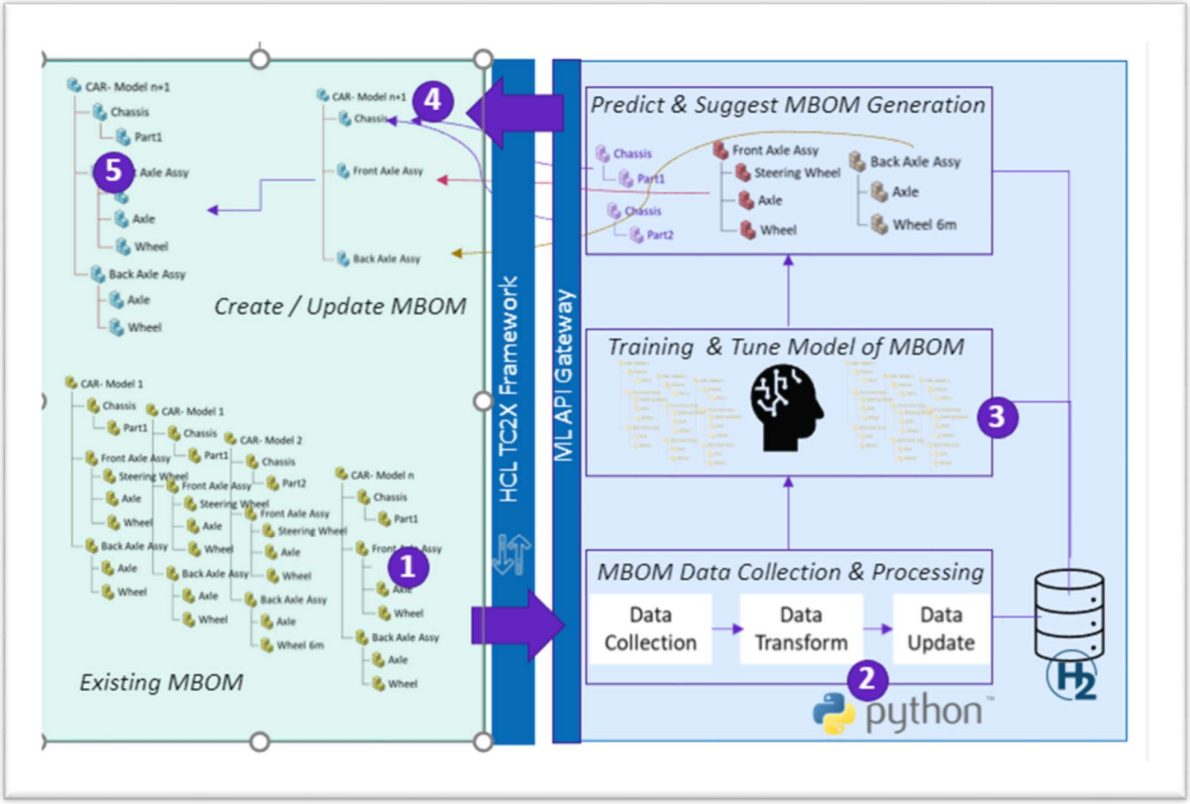
Yellow

Those suggestions which can directly be added under selected TOP but all their children doesn't exist in corresponding EBOM and its TOP Object type belongs to Delivery Unit /Phantom Unit or Make Assembly Type

Red

Suggestions which can directly be added under selected TOP neither all their children exist in corresponding EBOM nor Object Type of its TOP belongs to Delivery Unit /Phantom Unit or Make Assembly Type

SOLUTION ARCHITECTURE



- Export and send existing BOM to ML Framework
- Transform data as per schema defined and load into DB.
- Train model with exported BOM
- Show BOM Suggestion while creating new BOM in TC
- Create/ Update BOM Based on suggestion

Monitoring

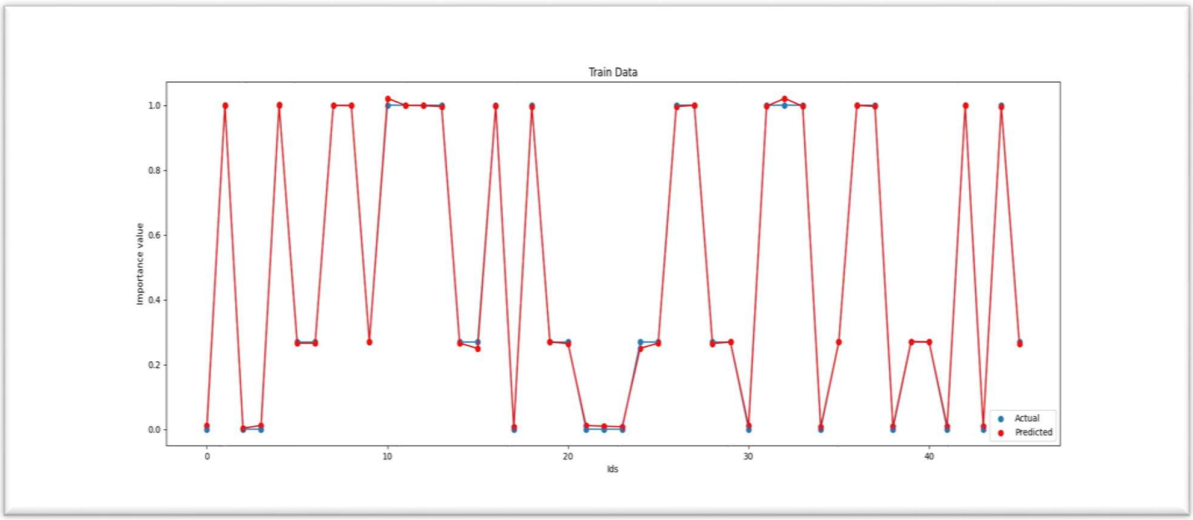
- We are generating some results during training to get insights of model performance.
- These all results created and saved in given directory as different images and their details are as follows :

(a) TrainDataResult (b) TestDataResult (c) TrainDataBehavior

(d) TestDataBehavior (e) Training Curve

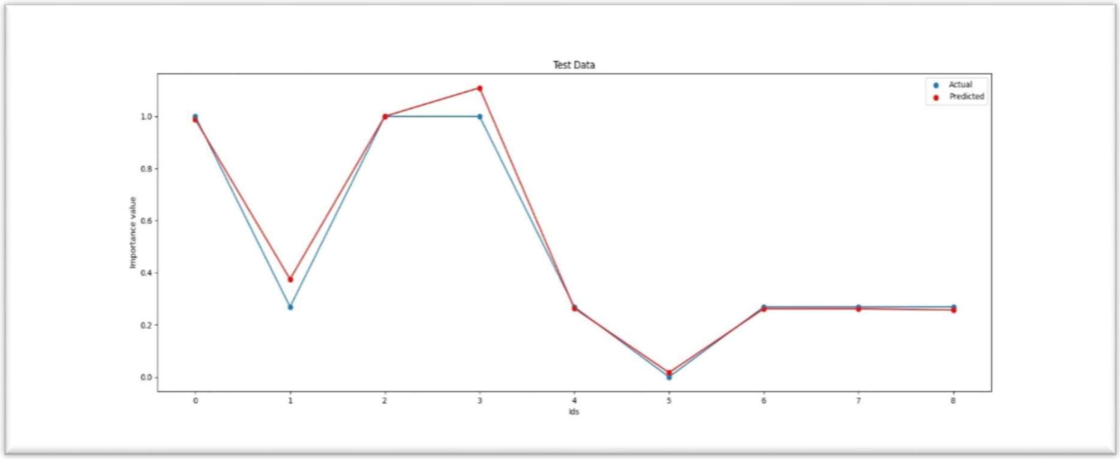
Train Data Result

This will give us insight of our model performance on Train Data with help of Train Data Points



Test Data Result

This will give us insight of our model performance on Test Data with help of Test Data Points.

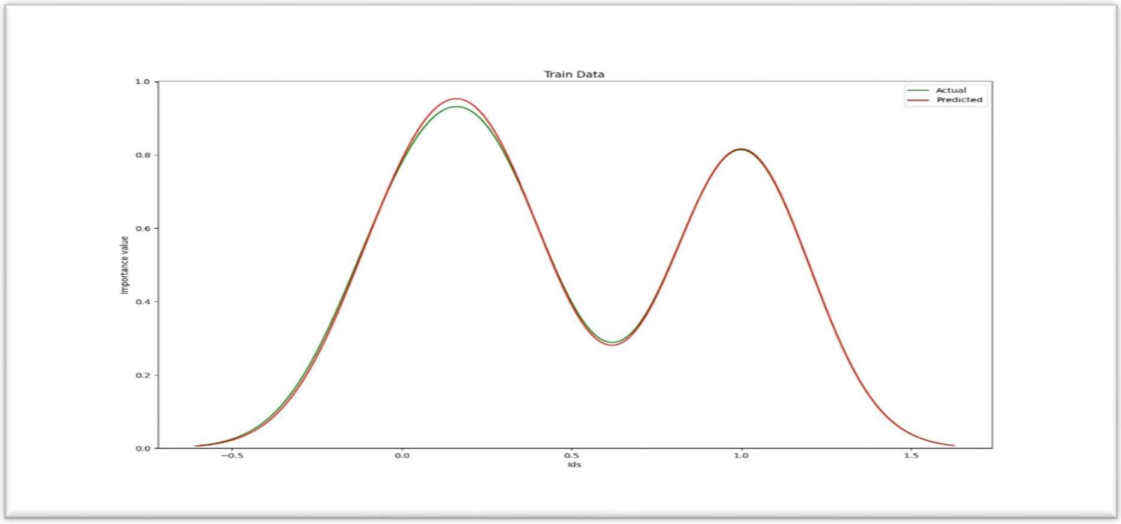


Predicted

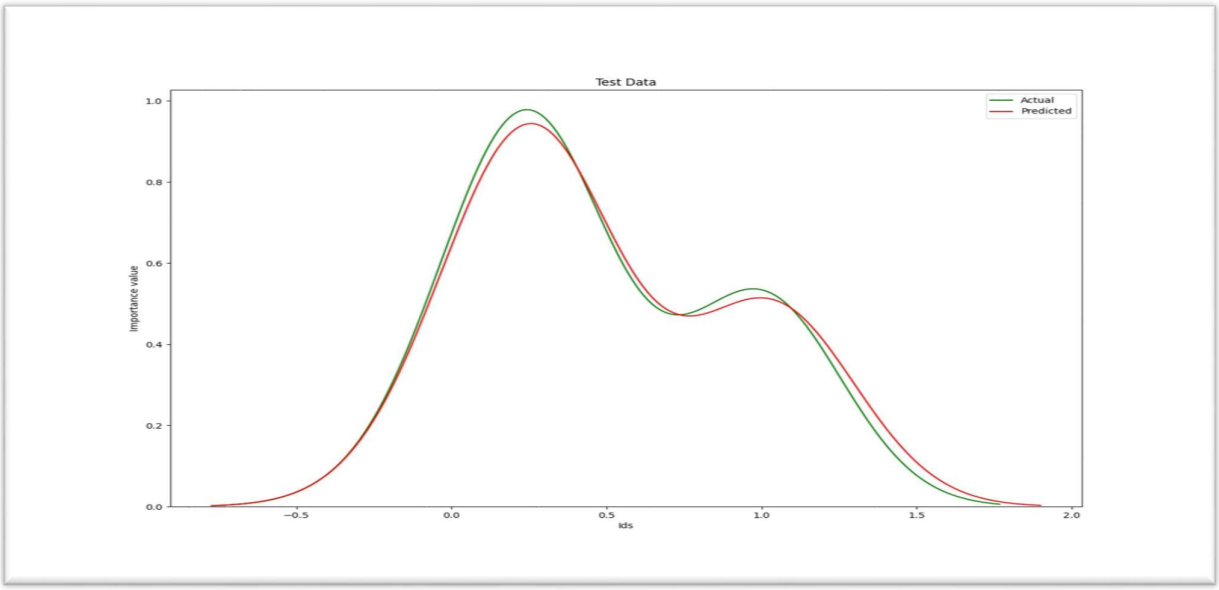
Actual

Train Data Behaviour

With the help of available data points and their results we have created a distribution from them and referring to it as behavior. When this is done train /test data result it is train /test data behavior.

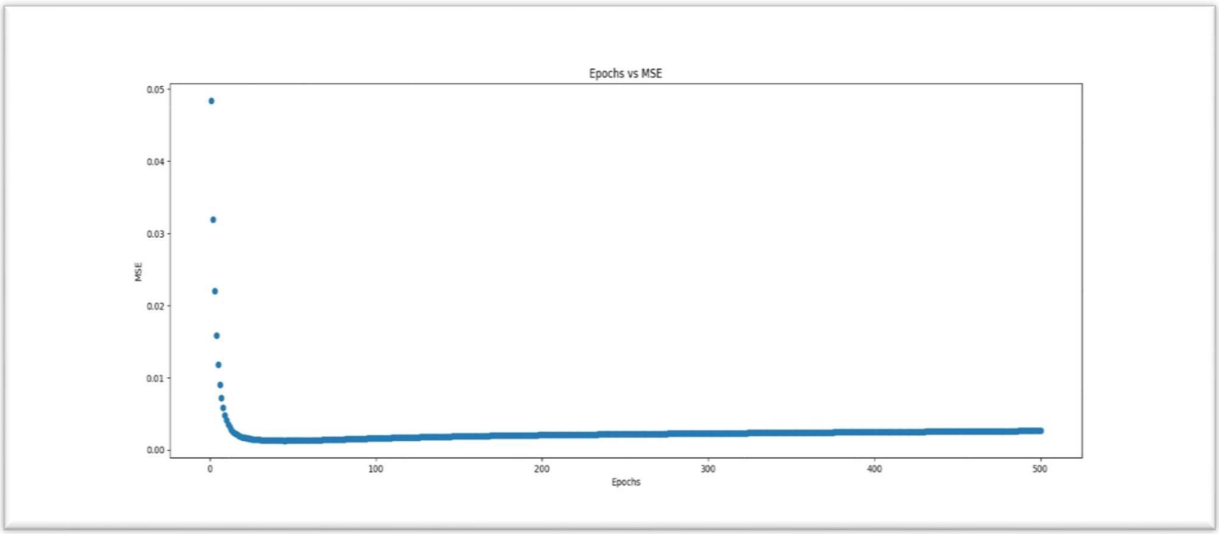


Test Data Result Behavior



Epoch vs MSE

It tracks error minimization with respect to each epoch/iteration.

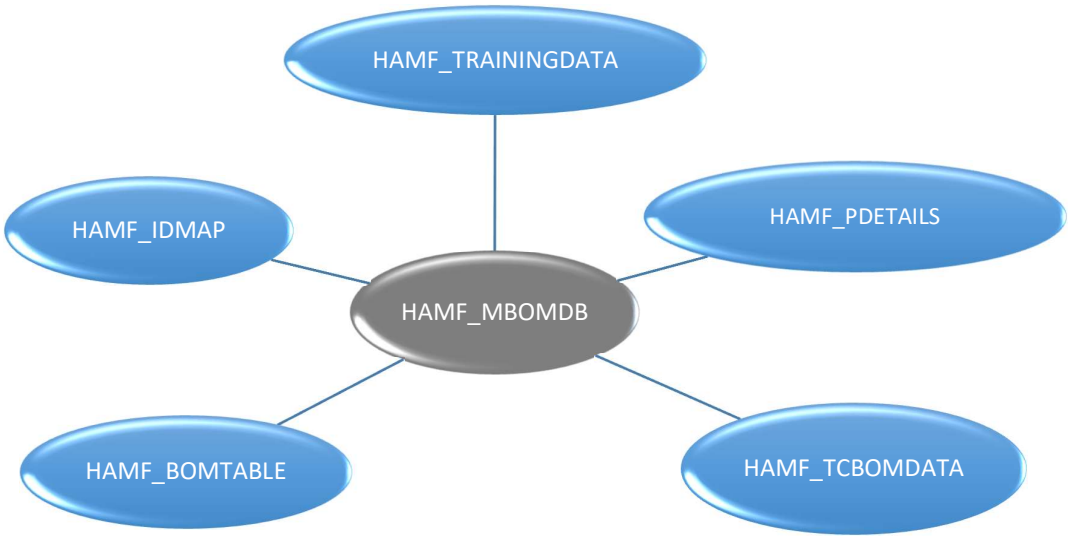


Database

The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 2.5 MB jar file size

For our usecase , we have build one schema to fulfill our requirements, Lets say if HAMF_MBOMDB is our Database then , we have five tables inside this database



- HAMF_BOMTABLE
  - _id
  - PARENTID (Int 256)
  - CHILDID (Int 256)

- HAMF_TCBOMDATA
  - _id
  - MBOMLINE
  - LEVEL
  - CHILDCUID
  - QUANTITY
  - IDINCONTEXT
  - ElementUID
  - FINDNO
  - PRODUCTLINE
  - XFORM
  - OBJECTTYPE

- HAMF_IDMAP
  - _id
  - UNIQUEID (Int 256)
  - Name (Char 256)

- HAMF_TCBOMDATA
  - _id
  - MBOMLINE
  - LEVEL
  - PARENTCUID
  - QUANTITY
  - IDINCONTEXT
  - ElEMENTUID
  - FINDNO
  - PRODUCTLINE
  - XFORM
  - OBJECTTYPE

- HAMF_TRAININGDATA
  - _id
  - PARENTID (Int 256)
  - CHILDID (Int 256)
  - N_EDGES
  - WIEGHT
  - DEGREE

UNIQUEID (UID) : This attribute is combination of three properties of that particular node which are IDIC , Name and ProductLine .

This tool can work for multiple plants and productlines as well.

HAMF_IDMAP

This table stores map of UNIQUEID (uid) to Name of node (Assembly /part name).

- **UNIQUEID** : this attribute is combination of three attributes of a node which are IDIC ,Name and ProductLine.
- **Name** : name of node

HAMF_TRAININGDATA

It stores basic features between pair of nodes which will be used for training further.

- **PARENTID** : source node
- **CHILDID** : destination node
- N_**EDGES** : number of edges between source and destination node
- **WEIGHT** : weight of edge
- **DEGREE** : sum of indegree and outdegree of source node

HAMF_TCBOMDATA

It stores all attributes of a particular node.

-**MBOMLINE** : name of parent
-**LEVEL** : level of child node with respect to MBOM TOP
- **CHILDCUID** : cuid of child
(cuid is combination of three properties of node ,IDIC, name and PLINE)

- **QUANTITY** : quantity of node
- **IDINCONTEXT** : idic of node
- **ELEMENTUID** : UID of node
- **PRODUCTLINE** : productline of node
- **XFORM** : TRANAFORMATION MATRIX
- **EBOMTOPUID** : uid of MBOM TOP
- **FINDNO** : sequence no of node
- **OBJECTTYPE** :object type of node

HAMF_PDETAILS

It store details of multiple productlines .

-**MBOMPARENTBOMLINE** : name of parent
- **LEVEL** : level of child node with respect to MBOM TOP
- **PARENTCUID** : cuid of TOP Node

(cuid is combination of three properties of node ,IDIC, name and PLINE)

- **QUANTITY** : quantity of node
- **IDINCONTEXT** : idic of node
- **ELEMENTUID** : UID of node
- **PRODUCTLINE** : productline of node
- **XFORM** : TRANAFORMATION MATRIX
- **EBOMTOPUID** : uid of MBOM TOP
- **FINDNO** : sequence no of node
- **OBJECTTYPE** :object type of node

Note : there is enough scope to optimize this schema , willing to achieve it in upcoming iterations.

After first feed of MBOM data , it starts updating these table from next iteration of MBOM data feed and retrain model and after then , recommednations will be given on the basis of updated/retrained model

KEY FEATURES

- We have build specific python module to handle all database related operations according to our requirement.
- Handle Database operations for multiple Product-Line without any manual Interference
- This tool is capable of handling different MBOMs belongs to differnet Plants and ProductLine simultaneously.
- It can create multiple models for multiple EBOM belongs to different ProductLine.
- Proper Monitoring of Model Training
- We can have control mostly at every operations and every single manipulation involved at any stage.
- Model is interpretable , in other words we can give reasons that why we are giving these suggestions.
- Trained Model can directly be exported and used in different to achieve same results.
- Logging File Generation to keep of operations of ML API's.
- Already consumed suggestion will not be recommended again inside that particular selected top.

FRAMEWORK VARIABLES

Before starting those ML API's we need to set a couple of variables to save a couple of results , which are as follows :

MODELPATH : ML Model will be stored in this location after training .

RESULTPATH : Results generated for monitoring of model training will be saved in this location.

H2JARFILE : Location of h2 database jar file .Mostly path for this file is C:\Program Files (x86)\H2\bin\h2-2.1.214.jar

LOGFILE : LogFile will be generated at this location

We are also treating name of database table as constants so ,all these constants are stored in on Object and can be accessed altogether in any module .In case of further extending the functionality of this tool it will be helpful to make changes in this manner.

API REFRENCES

Currently we have two active APIs' one is LOAD_DATA and other one RECOMMEND.

- LOAD_DATA: This API get MBOM data in a pre-defined format mentioned in dataset file attached in template item in Teamcenter. Attached dataset File in template Item in Teamcenter

```
<RootObjectType name ="ItemRevision">
  <searchdetails>
    <!--Enter the query name which is created on ItemRevison as search class-->
    <query name="Item Revision...">
      <filter>

        <attribute name="Item ID" value= "___ITEMID___" />
        <attribute name="Type" value= "___ITEMTYPE___" />
        <attribute name="Revision" value= "___ITEMREVISION___" />
      </filter>
      </query>
    </searchdetails>

  <PropertySet>
      <ObjectType type="BOMLine">
      <Property type ="string" source="BOMLine" header_name="BOMLine_ID">bl_item_object_name</Property>
      <Property type ="string" source="BOMLine" header_name="ProductLine">bl_rev_object_desc</Property>
      <Property type ="relation" source="BOMLine" header_name="EBOMTOPUID">TYPE(bl_line_object,structure_revisions).REL(bom_view,IMAN_METarge
      <Property type ="string" source="BOMLine" header_name="MBOMBOMLine">bl_item_object_name</Property>
      <Property type ="typedref" source="BOMLine" header_name="ELEMENTUID">bl_line_object.UID</Property>
      <Property type ="string" source="BOMLine" header_name="MBOMParentBOMLine">bl_formatted_parent_name</Property>
      <Property type ="string" source="BOMLine" header_name="Level">bl_level_starting_0</Property>
      <Property type ="string" source="BOMLine" header_name="FindNo">bl_sequence_no</Property>
      <Property type ="string" source="BOMLine" header_name="Quantity">bl_quantity</Property>
      <Property type ="string" source="BOMLine" header_name="IDInContext ">bl_abs_occ_id</Property>
      <Property type ="string" source="BOMLine" header_name="ObjectType ">bl_rev_object_type</Property>
      <Property type ="string" source="BOMLine" header_name="XForm">bl_plmxml_abs_xform</Property>

    </ObjectType>
```

We can configure what many attributes we want to extract , in our MBOM data , we can even determine their names and sequences as well.

INPUT

{"path":"C:/Users/prashant.goswami/DS/EBOM2MultipleMBOM/MBOMSolwithDB/data/LtData/MBOM_Structure.csv"}

OUTPUT

{ "Success" : "Data Upload and Model Training is successfully completed"}

- RECOMMEND: This API is directly consumed at AWC end, when we want to predict recommendations.

Input

{'ProductLine': 'Truck', 'Name': 'Radiator assembly', 'IDInContext': ['gPJE$gkzMAA$XD', 'rxtSh3r9MAgAKB', 'ktx36vLuMAgQ0B', 'VgTPkrwOMAwr$A', 'PwT_W4P6MAAtfA'], 'MBOMIDICList': ['VgTPkrwOMAwr$A', 'PwT_W4P6MAAtfA'], 'Level':1}

ProductLine : name of productline

Name : name of selected top

IDInContext : list of idics' of all elements of corresponding EBOM available in left side

MBOMIDICList : list of idic of already added parts/ assemnlies under selected top

Level : level of selected top

Output

'Suggestions': [{'CUID': 'TruckVgTPkrwOMAwr$ARadiator Core', 'Level': '2', 'Name': 'Radiator Core', 'IDInContext': 'VgTPkrwOMAwr$A', 'Quantity': '', 'FindNo': '20', 'XForm': '1000010000100001', 'matchType': 'Green', 'ELEMENTUID': 'xhEAAAy2JeLUPB', 'id': 0}, {'CUID': 'Truckktx36vLuMAgQ0BOutlet Pipe', 'Level': '2', 'Name': 'Outlet Pipe', 'IDInContext': 'ktx36vLuMAgQ0B', 'Quantity': '', 'FindNo': '30', 'XForm': '1000010000100001', 'matchType': 'Green', 'ELEMENTUID': 'XFOAAAy2JeLUPB', 'id': 1}, {'CUID': 'TruckrxtSh3r9MAgAKBRadiator Shroud', 'Level': '2', 'Name': 'Radiator Shroud', 'IDInContext': 'rxtSh3r9MAgAKB', 'Quantity': '', 'FindNo': '40', 'XForm': '1.33226762955019e-1512.76101316827354e-300-11.33226762955019e-15-1.97215226305254e-310-1.97215226305258e-31-2.76101316827354e-3010-0.848500000000001-0.763500000000002-0.09000000000000031', 'matchType': 'Purple', 'ELEMENTUID': 'XpNAAAy2JeLUPB', 'id': 2}, {'CUID': 'TruckgPJE$gkzMAA$XDFan Cover', 'Level': '2', 'Name': 'Fan Cover', 'IDInContext': 'gPJE$gkzMAA$XD', 'Quantity': '', 'FindNo': '50', 'XForm': '1000010000100001', 'matchType': 'Purple', 'ELEMENTUID': 'BqCAAAy2JeLUPB', 'id': 3}]}

Its output consist of 5 top suggestions with their corresponding attributes, so this ouput json consist array of 5 elements where each element is also a json object which consist of properties and value of that corresponding node.

USAGE INSTRUCTIONS

- H2 DATABASE should be working on port other than 8080.
- “HAMF_MBOMDB” should be created after H2 database installation.
- Dataset should be attached to template Item in Teamcenter.
- Workflow should be available to extract data from teamcenter with the help of Data-Extractor SOA.
- AWC UI Configuration should be done to get recommendations.
- IDIC of every component of EBOM and MBOM should be available if not available else run MPMIDIC Stamping workflow on their TOP .
- Check all required service are working.
- Open EBOM and its corresponding MBOM side by side and execute “Send data to AI/ML “ workflow.
- Data will be send and response will be received to gateway on successful completion .
- For recommendations of next MBOM , we need to open corresponding EBOM and new MBOM side by side need to select TOP of MBOM and Click on AI Suggest button.
- After click Suggestion Panel will open and suggestions will be visible in panel .
- You can select which suggestion you want to add under selected top and click update it will be added directly.
- This same process will be applicable for this MBOM as well ,after completion of this MBOM .

Constraints

- Currently we are treating Description attribute of TC Items as ProductLine .
- In our current usecase we had used hardcoded ProductLine .
- Config file for Framework environment variable has not integrated .

Demo

Scope For Extensions

- Database schema can be optimized .
- Integrate Deep Learning and Embeddings to improve results and Model variance.
- It also be helpful in reducing latency prediction.
- Scalability can be a issue need to work on improve approach.
- More API Endpoints needed to performance monitoring , to provide shallow ,deep , and different type of training and model.
- Features Extraction must be improved for better performance .
- Product-Line attribute should not be hard coded.

Glossary /References

NODE : Alias for Individual Assembly or any part

CUID : (Custom Unique Identification) It is combination of three attributes of any node IDIC , Name and its ProductLine

SRC NODE: Edge originating from that node (Parent Node)

DESTINATION NODE : Edge going to that node

N_EDGES : number of edges between source node and destination node

MONOTONIC : continuous increasing

TARGET FUNCTION : $F(x,y,z) = C\,(e^{-x}) + \log(y) + \frac{1}{2}(1/\,1+e^{(-z)})$