Milestone 3 Report

CS5150: Software Engineering

Cornell University

Ithaca, NY

Under the guidance of

Prof. William Arms

Department of Computer Science

Clients

Rajiv Kumar, Product Owner

Abhijnan Saha, Architect

Ayush Soni (aas427), Harshit Manchanda (hm545),
Mukul Shukla (ms3528), Rohith MD (rm977), Shubham Agrawal (sa2279),
Shubham Khandelwal (sk3266), Vidush Vishwanath (vv259)

Table of Content

Data, Database, Tables and Constraints

The database was created in mySQL. The reason why we chose mySQL as the first choice in database is due to the following reasons -
It is a relational database
It is free and open source - no additional cost involved to setup the storage
Easy to setup
Mass data import from csv
Easy to modify existing tables

Database tables in mySQL were created in accordance with the proprietary database.
MARA - Table containing the materials which are used in Bills Of Materials
MAST - Table containing Material to BOM link
STKO - Table containing header information
STPO - Table containing items information which are used in BOMs
CDPOS - Table containing historic replacement information
MISC - This table is created as a combination of different tables as used by the clientThis table contains information of items being used in different tables related to MRP , STORAGE LOCATION, MATERIAL GROUP etc.Since all the relative data is distributed among multiple tables which involve approximately joining ~20 tables under different LOB's, the design required to create a combination of all the values which contribute in the determination of best replacement item.
The relations are maintained between the tables.
MARA --- MAST
MAST --- STKO
STKO --- STPO
STPO --- MISC
MARA--- CDPOS
STKO --- STPO

Data: Sample data rows were created together with the client.The rest of the rows for all the tables were created in accordance with the proprietary tables and compliant with the initial rows as provided. The rows were combined to prepare an intermediate table as required by the ML model.

Current Repository Tree

Repo Link - https://github.com/vidushv/BOMRecommendation
Commits - https://github.com/vidushv/BOMRecommendation/commits/master

This is aimed at helping the reader understand what we have achieved and what amount of code is already present in the repository. We talk more about why we implemented each layer in the way we have done in the upcoming sections.

.

```
├── data.csv
├── HELP.md
├── machine_learning
│   ├── cosine_similarity.py
│   ├── data.csv
│   ├── parseData.py
│   └── unit_test.py
├── Milestone 2 Report.pdf
├── mvnw
├── mvnw.cmd
├── parseData.py
├── pom.xml
├── Prototype & Presentation.pdf
├── README.md
├── Software Engineer Feasibility Report.pdf
├── src
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── cornell
│   │   │           └── se
│   │   │               └── bom
│   │   │                   ├── bomRecom
│   │   │                   │   └── BomRecomApplication.java
│   │   │                   ├── controller
│   │   │                   │   └── CDPOSController.java
│   │   │                   ├── model
│   │   │                   │   ├── CDPOS.java
│   │   │                   │   ├── MARA.java
│   │   │                   │   ├── MAST.java
│   │   │                   │   ├── MISCELLANEOUS.java
│   │   │                   │   ├── STKO.java
│   │   │                   │   └── STPO.java
│   │   │                   ├── repository
│   │   │                   │   ├── CDPOSRepository.java
│   │   │                   │   └── MiscRepository.java
│   │   │                   └── service
│   │   │                       ├── CDPOSService.java
│   │   │                       └── impl
```

```
│   │   │                   └── CDPOSServiceImpl.java
│   │   ├── resources
│   │   │   ├── applicationContext.xml
│   │   │   ├── application.properties
│   │   │   ├── static
│   │   │   └── templates
│   │   └── webapp
│   │       └── WEB-INF
│   │           └── jsp
│   │               └── index.jsp
│   └── test
│       └── java
│           └── com
│               └── cornell
│                   └── se
│                       └── bom
│                           └── bomRecom
│                               └── BomRecomApplicationTests.java
└── target
    ├── bomRecom-0.0.1-SNAPSHOT.jar
    ├── bomRecom-0.0.1-SNAPSHOT.jar.original
```

Object Relational Mapping

An ORM library by definition is a completely ordinary library written in your language of choice**(Java in our case)** that encapsulates the code needed to manipulate the data, so you don't use SQL anymore; you interact directly with an object in the same language you're using.

With an **ORM** model we

- Save time. We only write our model once - its easier to update, maintain and reuse.
- Can be more flexible. It fits in the with the language naturally. It abstracts the DB system and is weakly bound to rest of the application. It lets us use OOP directly with the database.

We chose **Hibernate with JPA(Java Persistence API)** - JPA is just a specification, meaning there is no implementation. It lets us annotate our classes with JPA annotations, those annotations are implemented by Hibernate. JPA are the guidelines that must be followed or an interface, while Hibernate's JPA implementation is code that meets the API as defined by the JPA specification and provides the under the hood functionality.

So, what we get is - if we have a table named MAST - we create a class annotate it with @Entity ; create fileds and annotate it with @Column and we're done with the mapping as long as spring component scan is pointed to that class.

Model Layer

As we discussed in the previous section we need model classes for each table. This forms our model layer.

We just create classes for each table(these are our POJOs - plain old Java objects), map the constraints and specify the keys.

Repository Layer

Once we have tables modelled as objects we need functions to save and query those objects. These are provided by the repository layer. Basic functionality is provided by JPA, Hibernate - but we are free to map additional specific queries as suited by our business logic in the same repository.

To get the basic implementation we just implement this interface in spring.

Service Layer

We now have full functionality and even specific queries available for use and consumption into the application.

We write a service layer interface and its implementation which provides functions that interact with the database repositories and give us unit operations.

Controllers

Controllers are basically the API application exposes. Generally a controller is mapped to a model and view. But since we have to talk to the python machine learning model as well - we will have some controllers that don't have a mapping to a model and view.

For now, we estimate that we will have atleast two GET, POST APIs so that the python module can talk to the remaining application. We will also need additional controllers depending upon how the UI is implemented.

Machine Learning

The following section discusses the implementation of Machine learning models as part of the controller system.

Recommendation Models

We discuss the approaches we implemented for making the recommendation models using statistical and machine learning based approaches. The approaches used are explained in the sections below.

Feature Extraction:

A number of features were identified using the MISC table for each BOM:

1. Plant Location
2. MRP Group
3. Purchasing Group
4. Count of Replacement
5. Units of Measure
6. Volume
7. Cost
8. Scheduling
9. Lot Size
10. Storage Location

The features numbers 1-6 are arranged in the decreasing order of importance. The order of importance was conferred with the industry experts having sufficient domain knowledge.

Feature Encoding

The feature objects are encoded using appropriate hash function for creating encodings of each feature of the object passed.

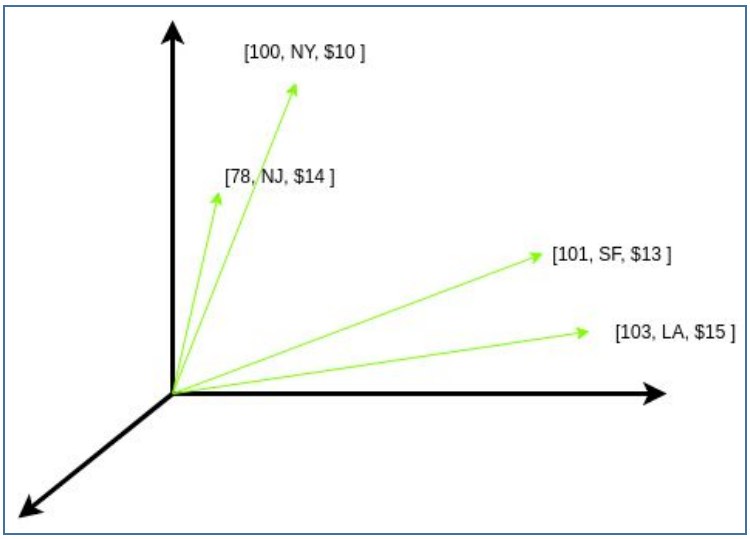
We tried two approaches for hashing:

- Hash based approach with feature count modulus for each feature
- Word2Vec models for word vector formation (word string encoding) and absolute difference for numeric entities.

We tried running different similarity computation techniques using these approaches.

Cosine Similarity

- We use the concept of cosine similarity which computes the similarity scores considering the features represented in the form of vectors..
- The encoded features are used to represent a vector for each BOM object.
- These features are then provided as input to the similarity function for getting the scores.



The figure above shows the vector based representations in a 3-dimensional vector space. The figure annotations are just for explanation, the real implementation involves numeric encoded features mapped onto the vector space.

Output of the models

For the approach, we get the properties of the object to be replaced and all the items (in the historical data, CDPOS table) which have been used to replace the object. We then form the feature vectors of these objects and use the two approaches for feature encoding separately.

```
Shubhams-MBP:machine_learning shubham$ python cosine_
ITEM to be replaced:  PLATINUM AG3 BATTERY
ITEM                                SCORES
AUTOMOTIVE BATTERY                  0.92
EVERSTART LEAD ACID BAT             0.77
DEKA 410 AMP MOWER BAT              0.77
OPTIMA RED TOP BATTERY              0.74
DURALAST                            0.69
BMW M3-75 STARTING BAT              0.68
TOP DEEP CYCLE BATTERY              0.55
NORTHSTAR                           0.54
BMW_FDZ BATTERY                     0.50
```

Similarity Scores: Word2Vec based features

```
Shubhams-MBP:machine_learning shubham$ python cosine_
ITEM to be replaced:  PLATINUM AG3 BATTERY
ITEM                                SCORES
AUTOMOTIVE BATTERY                  0.87
DURALAST                            0.75
EVERSTART LEAD ACID BAT             0.72
BMW M3-75 STARTING BAT              0.69
OPTIMA RED TOP BATTERY              0.67
NORTHSTAR                           0.65
DEKA 410 AMP MOWER BAT              0.64
TOP DEEP CYCLE BATTERY              0.63
BMW_FDZ BATTERY                     0.44
```

Similarity Scores: Hash based features

As mentioned, we used two approaches for feature encoding and evaluated the model for both of them. The results observed are discussed below:

The scores in the above figures show the similarity scores computed when the “ITEM” is chosen as an alternative for “ITEM to be replaced” as shown in the above figure. For instance, the item “AUTOMOTIVE BATTERY” has a similarity computation (based on its features and features of the item to be replaced) of 0.92.

Weighted Similarity Computations

- We also explored an approach for similarity computations on the basis of weights associated with each feature.
- Currently we use a rule-based weight computation technique, with more important features getting more weights.
- Distribution of weights followed importance levels. This helps for use case specific knowledge to be included.
- For incorporating based intuitions for feature importance in our machine learning models, the industry experts can give their feedback for the rules and we can upgrade our rule based model for better efficiencies.

Results

We evaluated our models on 10 different material groups and found the following results:

Average Accuracy Scores

- Cosine Similarity (Hash based approach): 0.68978
- Cosine Similarity (Word2vec approach): 0.76473
- Weighted Similarity Computations: 0.82371

For computing the accuracy we generated the labelled data on the recommendations of the domain expert, and then compared the output of our models with the ones generated by the three models mentioned above. We computed these for all 10 different material groups and took an average for the same.

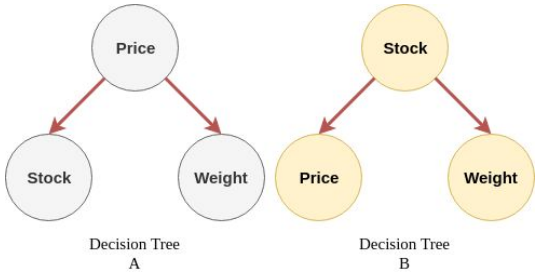
We find that the accuracy scores for weighted similarity computation approach are the highest which could be attributed to the fact that learnt feature weights (rule based) correspond to better recommendation models.

Random Forests

We understand that the client is primarily interested in how the Machine Learning could further improve the results. The next step for the machine learning model is to experiment with Random Forest algorithm.

Motivation:
The Random Forest algorithm would be primarily useful in cases where there are multiple different features, and deciding feature weights is not trivial. In this case, the Random Forest algorithm, once trained on the data would provide feature weights to different features making the feature weighing process much easier when the problem is not trivial.

Brief Description of RF on BOM Data:
Random Forest algorithm is trained by generating multiple decision trees (hyperparameter), where each decision tree is built on a different dataset D_i, where each D_i is formed by sampling data from the original distribution D, with replacement. This helps in reducing model variance, and the technique is called bagging.



For example, in the diagram above, Decision Tree A uses Price as the primary feature to divide the tree, which means that the dataset which the tree A is built on got more information about the target label. While in Decision Tree B, the Stock feature, when used for division gives a higher information gain than the other. Terms like information gain [1] have not been defined in the report due to the fact that these are extremely technical, and more information about these can be found in references.

Desired Result from Random Forest:
Theoretically, RF would help us efficiently generate feature weights for being used in cosine similarity. The hope is that this would be seen in the results even on the simulated data.

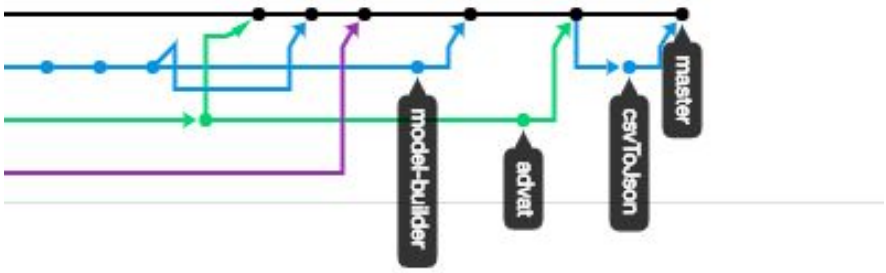
Static verification

The methodology we are using to build the project is by dividing the work into pairs with a person in the team at least with that domain as his/her technical fortay. We make sure that we have a separate person reviewing each pull request a pair has submitted. The work done of the five different parts were done in the following pairs:

- Database: Mukul and Rohith
- Controllers: Harshit and Mukul
- ML: Ayush, Shubham
- Feature Engineering: Shubham, Vidush

- Unit Testing and CI: Mukul, Vidush

Also all pull requests were individually checked before merging to master.
A brief history of the repository is depicted in the picture below:



Brief history of Repository branches

Next steps

- ## Use Random forests as ML Models

We want to build the Random forest machine model on the dataset and try to improve the accuracy over the dataset. We feel Random forest would give out better results.

- ## Evaluate on more features:

We want to include more features in the miscellaneous data table. The model we have right now uses around 7 features but we want to incorporate more features.

- ## Create more data

Right now we built the data keeping in mind the automotive industry in mind and want to cover a broad scope of manufacturing industry to make the results and the model more robust.

- ## Application Integration

We are currently building the integration of the Python Machine Learning component with the Java application. We are building scalable get/put REST APIs to handle the integration.

- ## Unit testing

We want to have more unit testing on the individual components of the application to make sure every function is working properly.

- ## Integration Testing

Once we complete, the integration between the Java application and the Machine Learning component. We want to make sure that everything is smoothly communicating and is robust and scalable.

Build Setup

Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies. Unlike earlier tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. (we use clean install)

Maven dynamically downloads Java libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache. This local cache of downloaded artifacts can also be updated with artifacts created by local projects. Public repositories can also be updated.

Maven also allows us to run tests (integrated with spring boot) directing reading the @Test annotation. We will talk more about tests in the next section.

Tests

We have tests written in Spring via Junit. We use two annotations -

@RunWith(SpringRunner.class) is used to provide a bridge between Spring Boot test features and JUnit. Whenever we are using any Spring Boot testing features in out JUnit tests, this annotation will be required.

@DataJpaTest provides some standard setup needed for testing the persistence layer:

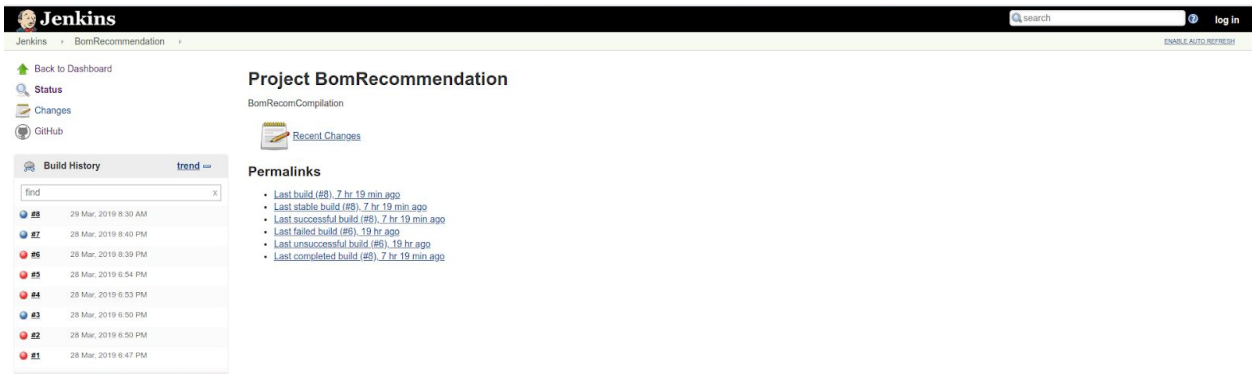
- configuring H2, an in-memory database
- setting Hibernate, Spring Data, and the DataSource
- performing an @EntityScan
- turning on SQL logging

With these we will write tests for each service layer function and controller. So far we have the basic setup and one test.

Apart from Spring, we also have unit tests that test the machine learning portion of the project. These tests are made through the Python unit test framework. We essentially create a test for each major function used in the ML workflow. These tests verify that first, the function works, and secondly the function returns data in the format we expect. As of now, all unit tests pass, and if a future commit breaks the workflow, the team will be notified automatically via Jenkins.

Continuous Integration - Jenkins

We use Jenkins to periodically pull code from our GitHub repository and compile our project along with running both ML and Spring Junit tests. Jenkins does this by accessing the publicly available github URL for the code base and calling into Maven for compiling the code and running other tests.



As can be seen by the screenshot above, current and past builds are immediately visible including whether they were successful or not.

Azure Tradeoffs

In order to run the Jenkins system dynamically, the team was presented with two major options. First, we could run Jenkins locally on a selected team member's personal computer. Another option was to host Jenkins on the cloud and run it through a service such as Azure or Amazon Web Services. The team ultimately decided to host on the cloud with Azure which provide some benefits. For example, confirming the repository health was no longer dependent on a single computer running at all times. Furthermore, all team members can view the Jenkins portal through a simple ssh command. Through this portal, team members can not only view repository health, but also build and run tests on demand through the simple click of a button.

One of the tradeoffs we face when hosting this service on the cloud is that github is no longer able to access it because it does not contain a publicly accessible IP. This restriction is due to security reasons within Azure. A public IP would have allowed github integration with Jenkins and ensured that Jenkins could run the build and test process on each push to master. However, the team was able to avoid this obstacle by creating a daily build schedule and allowing on demand builds by any member of the team.

Final Deliverables:

The deliverables for this project are as follows:

1. A single page web application that can recommend materials to be replaced using a machine learning model.
2. An API specification that can help further integration if required with a third party system.

References

1. https://medium.com/coinmonks/what-is-entropy-and-why-information-gain-is-matter-4e85d46d2f01