﻿﻿Students: **Fidelis Mnyanyi**, [fidelis.mnyanyi@student.ie.edu](mailto:fidelis.mnyanyi@student.ie.edu)

**Lucas Holstein**, [lucas.holstein@student.ie.edu](mailto:lucas.holstein@student.ie.edu)

**Michael Wagdy**, [michael.wagdy@student.ie.edu](mailto:michael.wagdy@student.ie.edu)

**Ricardo Santos**, [ricardo.santos@student.ie.edu](mailto:ricardo.santos@student.ie.edu)

**Siddhi Dhavale**, [siddhi.dhavale@student.ie.edu](mailto:siddhi.dhavale@student.ie.edu)

**Tomas Susedik**, [tsusedik@student.ie.edu](mailto:tsusedik@student.ie.edu)

Course: Recommendation engines

Program: GMBD-EN-BL-ENE-2020

Assignment: Product recommendations on Amazon

Table of contents:

[Executive summary](#_deoupuzhkn0q) 1

[Final report](#_g68w51j72zt1) 2

[Part 1: Understanding dataset properties](#_v9w09m8s5rlj) 2

[1.1 Data sample](#_tt0sq6jrgyml) 2

[1.2 Classes of data](#_8flkonny8o1u) 2

[Part 2: Exploratory data analysis (EDA)](#_pi5i2vizb3tx) 2

[2.1 Shape](#_yzip641ay83h) 2

[2.2 Transforming Time columns and Renaming columns](#_g9nt9uxyo5fs) 2

[2.3 Missing values](#_m47zw0u1woss) 2

[2.4 Distinct values](#_7g1bnr7bw8im) 2

[2.5 Variable transformation](#_2w7qw7p9ldui) 2

[2.6 Distribution](#_w3p4jbmwm2v0) 3

[2.7 Next steps](#_cjsw16jgn639) 4

[Part 3: Modeling](#_g68748ejx3dt) 5

[3.1 Pre-processing](#_p9hi6h7t5ctd) 5

[3.2 Collaborating Filtering Recommender System](#_7vick5q30ohg) 5

[3.2.1 - Benchmark](#_rfiaxolgcw0w) 5

[3.2.2 - Hyperparamter tuning](#_5opdfjtljiaw) 5

[3.2.3 - Final Model with SVD](#_r37uzmg1y12) 6

[3.3.4 - Analyse Item/User Bias](#_gjwc35a74j1r) 6

[3.3 Content-based recommendation engine](#_9xufxfg2ple6) 7

[3.3.1 TF-IDF vectorizer](#_gvqi2lrnk70q) 7

[3.3.2 Cosine similarity and fit model](#_4x749sl0qnn2) 7

[3.3.3 Check model performance](#_8000vnmt2uki) 8

[Part 4: Closing remarks](#_zduvi1vig27n) 8

# 

# Executive summary

In this assignment, we’re creating a recommendation engine for product recommendations on Amazon.com. The dataset we selected contains information about musical instruments.

After performing the exploratory data analysis and doing the required pre-processing steps, we compare the performance of multiple models. The models can be divided into 2 areas:

1. collaborative filtering recommendation engines
2. content-based recommendation engines

With collaborative filtering, we fit the following models:

1. KNN - achieving RMSE of 0.8754
2. SVD - achieving RMSE of 0.8745
3. SVD++ - achieving RMSE of 0.8742

With content-based models, we fit the following model:

1. TF-IDF - achieving RMSE of 3.6176

Based on the performance, we ended up selecting the SVD model, which ended up getting 0.8578 RMSE on our testing set. The reason we selected it over SVD++, which performed better on the training set, is that SVD took significantly less time to compute (approx. 95% less time than SVD++). While the model performance is certainly important, we also have to consider the time, as once deployed, we would need to make sure that the recommendations are readily available to the users browsing the store.

# Final report

## Part 1: Understanding dataset properties

### 1.1 Data sample

We take a look at the dataset sample and visualize the top 5 rows to understand the values of the dataset.

### 1.2 Classes of data

There are 3 classes of data:

* *int* - present in 1 column (unixReviewTate)
* *object* - present in 7 columns (reviewerID, asin, rewieverName, helpful, reviewText, reviewTime, summary)
* *float64* - present in 1 column (overall)

## Part 2: Exploratory data analysis (EDA)

### 2.1 Shape

10261 rows by 9 columns.

### 2.2 Transforming Time columns and Renaming columns

We remove the *reviewTime* column as we do not require it for the ML pipeline. We convert reviewDate of formatUnix/Epoch time to a readable date format. We rename columns to maintain consistency and relevance:

* asin to *productID*
* Overall to *rating*
* unixReviewTime to *reviewDate*

### 2.3 Missing values

There are null values only in 1 column - reviewerName, with 27 values missing. We drop those rows and the new shape of dataset is (10234, 8).

### 2.4 Distinct values

We check for distinct values of the *reviewID* and *productID*.

### 2.5 Variable transformation

As the variable *helpful* is a tuple consisting of 2 numeric values, e.g. [1,3] where the first value stands for number of helpful reviews and the second for total number of reviews. We split the helpful variable into two to correspond to a value each of the tuple: *num\_helpful* and *num\_review.*

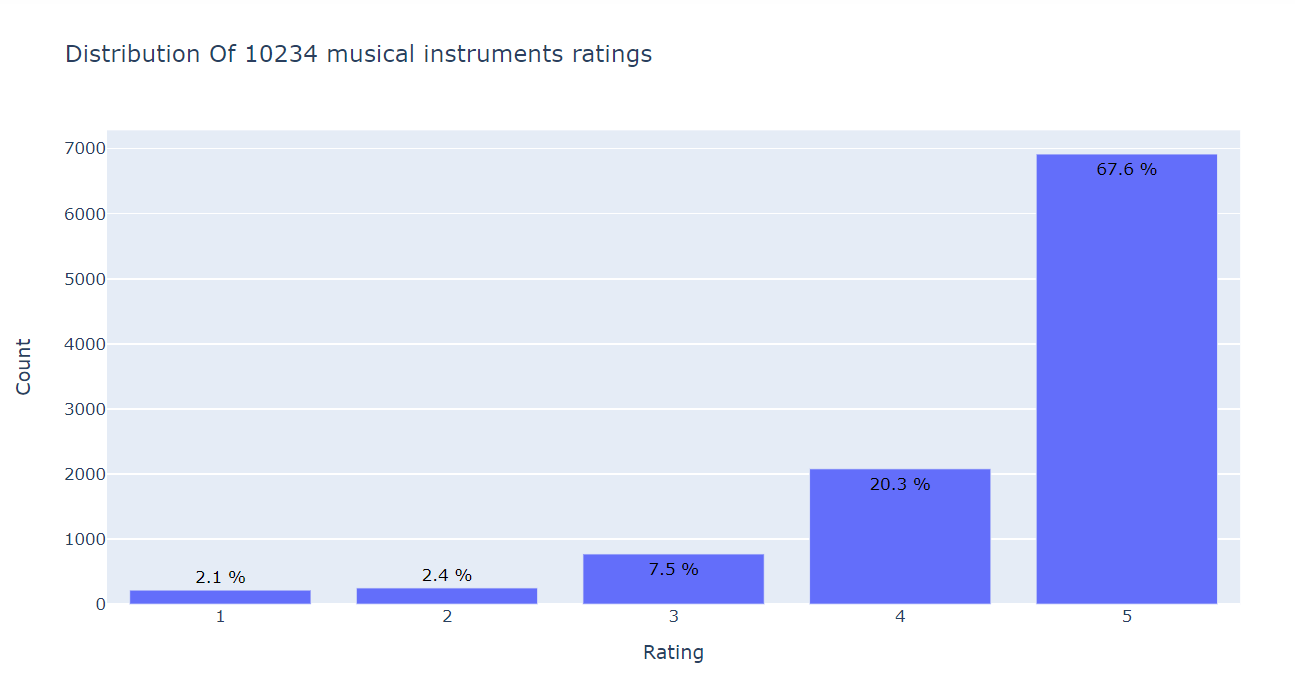
We then transform the variable and create a new variable *helpful\_ratio* to comprise a single calculated variable of number of num-helpful / num-reviews.

We delete the original *helpful* column and add the *helpful\_ratio* and the two newly created variables *num\_helpful* and *num\_review* to the dataframe.

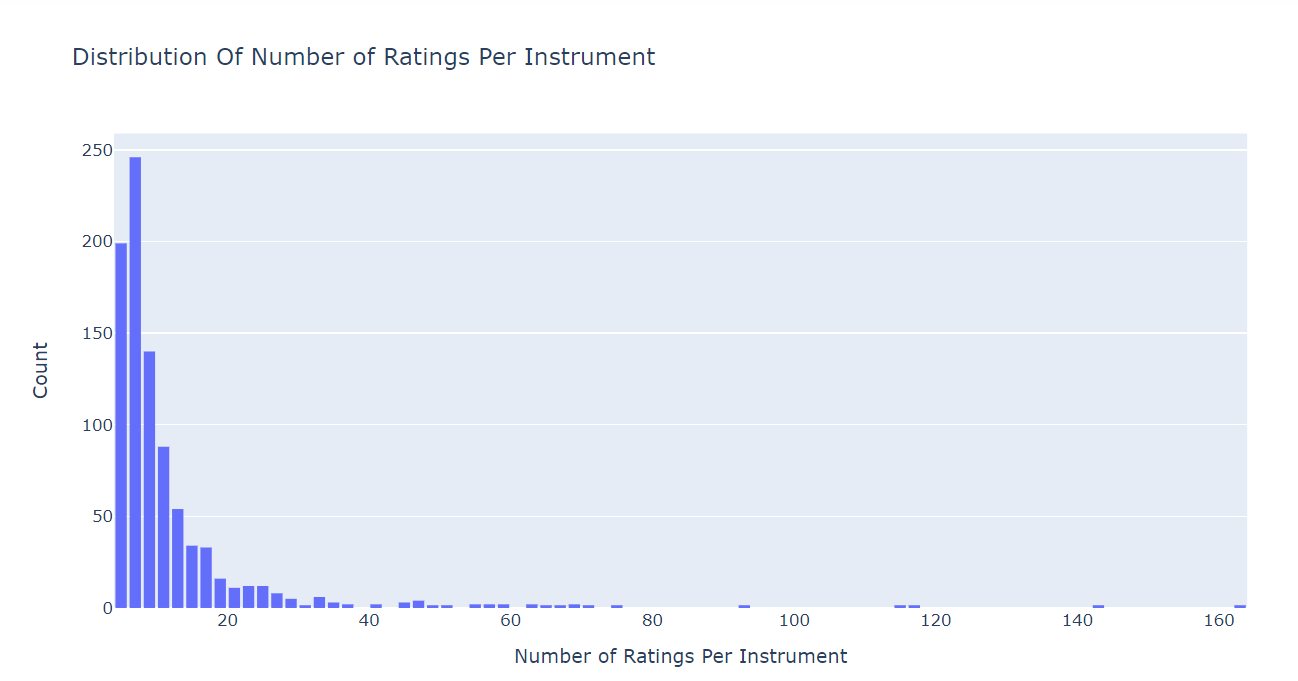
The new shape of our dataset is (10234, 10).

### 2.6 Distribution

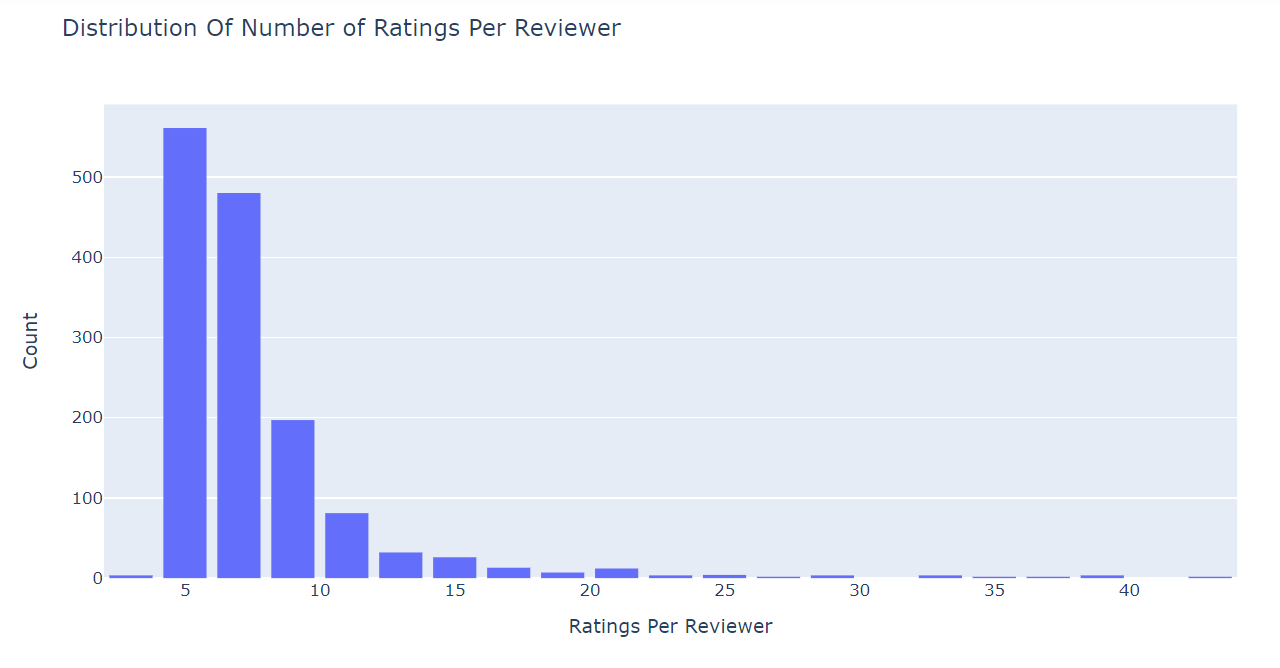
We look at the distribution of ratings for the 10234 instruments and can see that 67.6% of the ratings were skewed to ‘5’, which means that the ratings are highly biased to the positive rating.



We also looked at the distribution of number of ratings per instrument and can see that only a small amount of highly rated instruments and a larger number of unpopular instruments.



We then look at the distribution of the number of ratings per reviewer, and can see that most of the instruments did not receive any ratings from the reviewers.



From the above distributions we can conclude that we have a higher skew towards a small number of ratings per reviewer, who have *highly* reviewed a small number of instruments only.

### 2.7 Next steps

We now have a modeling dataset with dimensions 10234 rows by 10 columns, ready to start modeling. We now proceed to convert our dataset to the format required by the different libraries.

## Part 3: Modeling

### 3.1 Pre-processing

We used train\_test\_split as our method to evaluate the performance of our recommendation system by splitting into training and testing dataset. We chose an 80% split for the train dataset and 20% for the testing dataset to be used in the *Content-based recommender system*.

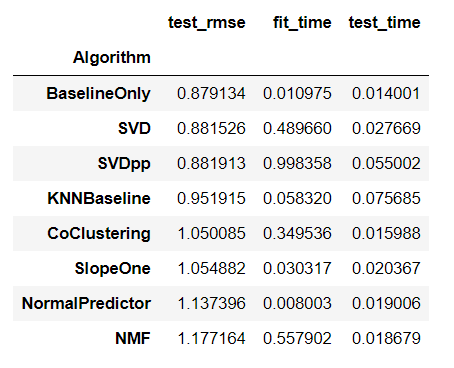
Regarding the *Collaborating Filtering recommender system,* we used Surprise library and converted the dataset to the required format with reader and load from df filtering reviewerID, productID and rating fields using the train and test dataset split we performed before.

### 3.2 Collaborating Filtering Recommender System

Collaborative filtering approach builds a model from a user’s past behaviors (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (in this case ratings for items) that the user may have an interest in.

#### 3.2.1 - Benchmark

Our first approach was to try different algorithms with default parameters to get a benchmark with RMSE as our evaluation metric.



BaselineOnly gives the best model with default parameters. We perform hyperparameter tuning to SVDpp, SVD and KNN Baseline to check if we get better results.

#### 3.2.2 - Hyperparamter tuning

For hyperparameter tuning we used *GridSearchCV* and a *K Fold* = 3 cross validation on the train set.

1. **KNN Algorithm**

KNN is a non-parametric, lazy learning method that relies on item feature similarity and calculates the “distance” between the target item and any other item in the database. For KNN hyperparameter tuning we used as parameters the type of distance (“name), min\_support and user\_based as parameters.

Best RMSE: 0.8754

Best Parameters: {'sim\_options': {'name': 'msd', 'min\_support': 5, 'user\_based': True}}

1. **SVD**

SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from N to K. For SVD hyperparamter tuning we used as parameters: n\_factors, n\_epochs, lr\_all and reg\_all.

Best RMSE: 0.8745

Best Parameters: {'n\_factors': 10, 'n\_epochs': 20, 'lr\_all': 0.005, 'reg\_all': 0.1}

1. **SVD++**

We used the same parameters for SVD++: n\_factors, n\_epochs, lr\_all and reg\_al but with different ranges.

Best RMSE: 0.8742

Best Parameters: {'n\_factors': 25, 'n\_epochs': 10, 'lr\_all': 0.01}

#### 3.2.3 - Final Model with SVD

Even though the 3 models had similar RMSE, we chose SVD because it is also the fastest model.

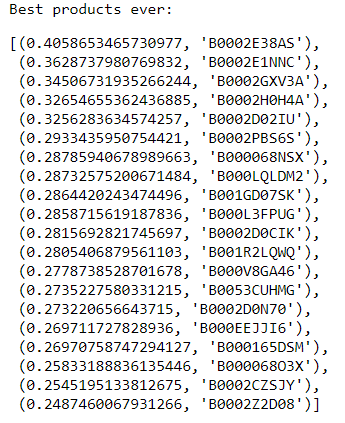
Best Parameters: {'n\_factors': 10, 'n\_epochs': 20, 'lr\_all': 0.005, 'reg\_all': 0.1}

We proceed to run the model in the test set (20% of data) with RMSE as our evaluation metric.

Unbiased accuracy on test set, RMSE: 0.8578

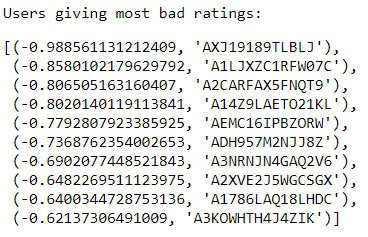
#### 3.3.4 - Analyse Item/User Bias

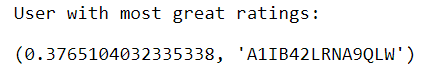
We then proceed to analyse the item bias on the best and worst products:



Some items are biased on the worst products but not on the best products.

Regarding Users bias:





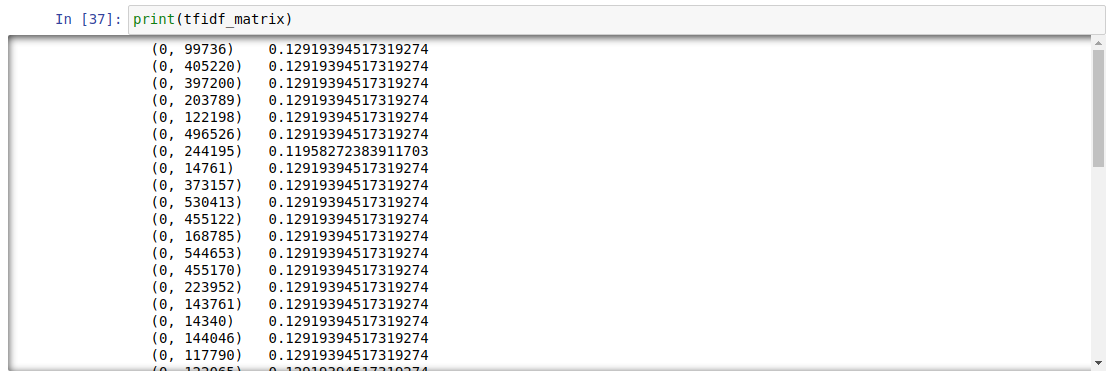
Same like item bias we have more bias on negative rating but not in users when giving good ratings which is expected.

### 3.3 Content-based recommendation engine

In this part, we leverage textual information from the user reviews and the helpfulness of those reviews in order to suggest items to other users.

#### 3.3.1 TF-IDF vectorizer

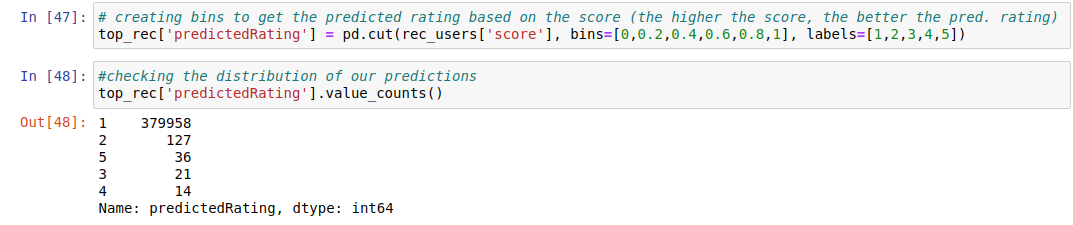
The first step is to create TF-IDF vectorizer, which we use apply to the *reviewText* field in order to create the TF-IDF matrix.



#### 3.3.2 Cosine similarity and fit model

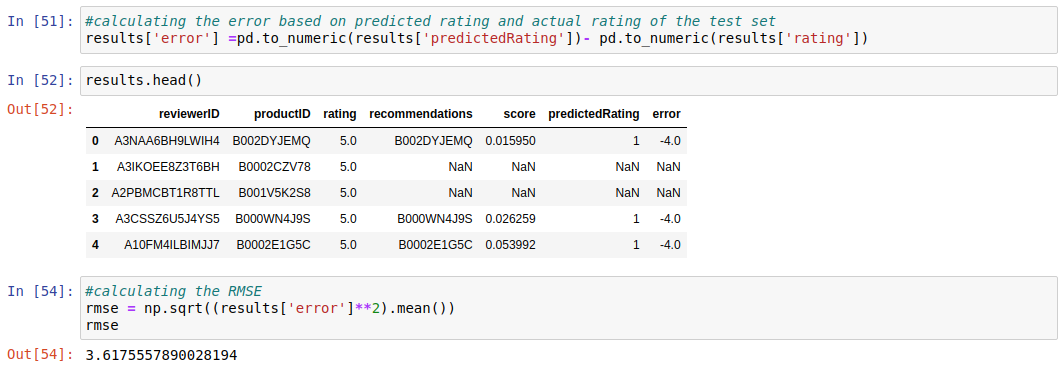
We use linear kernel in order to compute the similarities and find products that are the most similar to one another. At first we define the needed functions, which we then apply to the training set.

We predict the score on a range from 0 to 1, which we then “extrapolate” through the use of bins, i.e. scores from 0 to 0.2 get rating 1, scores from 0.2 to 0.4 get rating 2, etc.



#### 3.3.3 Check model performance

In the final part, we fit our model to the test set, get top recommended items, compute their score and compute the error.



We get the error by comparing the predicted ratings vs. actual ratings on the test set and applying the RMSE formula.

In the end, our RMSE is 3.6176, which is significantly worse than the collaborative filtering models. When analysing what the reason might be, we came to a conclusion that it can be some, or a combination of the following:

1. Size of dataset - in order to leverage textual data properly, we would need to have access to a lot of data to tune the model. In our case, we ran into a limitation with computing power - 16GB of RAM wasn’t sufficient to use one of the larger datasets.
2. Type of products - in our case, we’re working with musical instruments, which is a rather “niche” area, with specific products and buyers. Textual data might work better with more general, daily-use consumer products.
3. Model - in our case, we’re not tuning hyperparameters of the engine, only keeping them default, as our objective was to explore TF-IDF models and we’re not able to do any more sophisticated NLP analysis.

## Part 4: Closing remarks

Based on the performance of models we’ve created, we would prefer to use collaborative filtering models due to better performance. They are also simpler to setup and take less time to compute.

As mentioned in 3.2.3, we would choose SVD model with RMSE 0.8578.