1 Recommender Systems-Project Work (Kaggle Competition DSG17)

Getting the perfect music recommendation is a challenging task. Who has never dreamed of laying back and listening to some music while having no buttons to press at all and still getting the perfect tune? However, what defines this perfect tune?" (https://www.kaggle.com/c/dsg17-online-phase/data)

The Kaggle competition addresses this issue in the case of Deezer. Deezer is a world-wide music streaming service that provides user access to more than 43 million tracks. The app suggests what tracks the user might like to listen to next on its music recommendation radio Flow. The concept of Flow is simple: it uses collaborative filtering to provide a user with the music he wants to listen to at the time he wants. Moreover, if he does not want to listen to specific tracks and skips songs by pressing the 'Next song' button, the algorithm should quickly detect it. In this context, getting the first song recommendation right is essential. (https://www.kaggle.com/c/dsg17-online-phase/data)

This paper addresses the following questions:

- Q1: What would the authors do to win this competition?
- Q2: What would the authors do to solve the problem that Deezer is interested in?
- Q3: Why do the two solutions not overlap, or why do they?

The next section explains the data and provides descriptive analysis and the authors' models. Afterwards, we present the results. Finally, we discusses the research questions and concludes.

1.1 Data and Methods

Kaggle provides users' listening history for November 2016 in training and testing data. The training data contains 15 columns and more than 7.5 Million rows. The test data contains 15 columns and about 20'000 rows (see table 1). Each row represents one track listening, including the timestamp and user data. Whereas the train data contains the user's history for a whole month, the test data only includes each user's next song recommendation. (https://www.kaggle.com/c/dsg17-online-phase/data)

VARIABLE	DESCRIPTION	DATATYPE	IN TRAIN	IN TEST
MEDIA_ID	identifier of the song listened by the user	category	x	x
ALBUM_ID	identifier of the album of the song	category	x	x
MEDIA_DURATION	duration of the song	int64	x	x
USER_GENDER	Gender of the user	category	x	x
USER_ID	Anonymized id of the user	category	x	x
CONTEXT_TYPE	type of content where the song was listened: playlist, album …	category	x	x
RELEASE_DATE	release date of the song with the format YYYYMMDD	Datetime64[ns]	x	x
TS_LISTEN	timestamp of the listening in UNIX time	Datetime64[ns]	x	x
PLATFORM_NAME	Type of operating system	category	x	x
PLATFORM_FAMILY	Type of device	category	x	x
USER_AGE	Age of the user	int64	x	x
LISTEN_TYPE	if the songs were listened in a flow or not	category	x	x
ARTIST_ID	identifier of the artist of the song	category	x	x
GENRE_ID	identifier of the genre of the song	category	x	x
IS_LISTENED	Target variable: 1 if the track was listened, 0 otherwise	category	**x**	**0**
SAMPLE_ID	Id to link the true value of the target variable in the test set	category	**0**	**x**

Table 1: Description of the train and test data

The train data includes the target variable IS_LISTENED, but it is missing in the test data. The variable describes if a track is listened to or not. Deezer considers a track to be "listened" if the track is running for more than 30 seconds (IS_LISTENED=1); otherwise, as "not listened" (IS_LISTENED=0). The test data's target variable is only known to the jury, who will evaluate the model by comparing predicted to the actual values.

We remove categories in the training data that are not available in the test data. We would avoid that deletion to prevent overfitting for a more generalist purpose. Further, we remove all samples before November 2016.

The data pre-processing step reduces the number of samples from 7'558'585 to 1'815'679 in the training data.

1.1.1 Descriptive Analysis

See Jupyter Notebook for code (Data_Preprocessing.ipynb)

USER_AGE

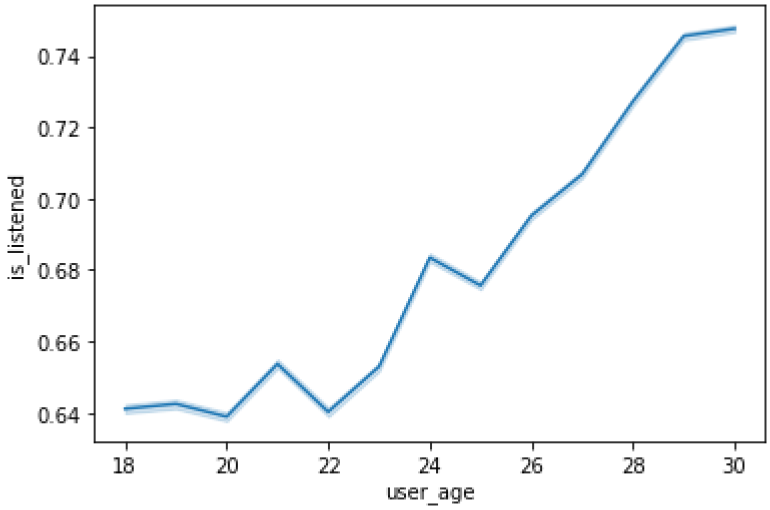
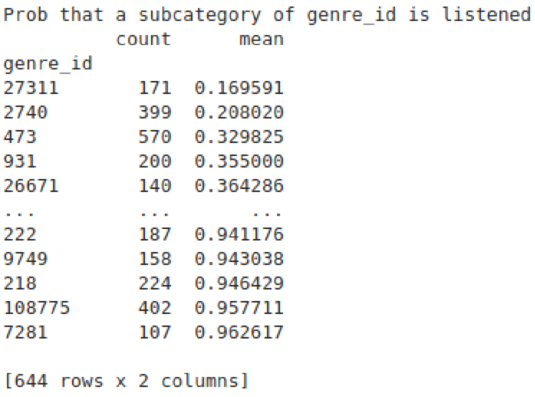


Image 1: How the age of the users affects listening and skip behaviour

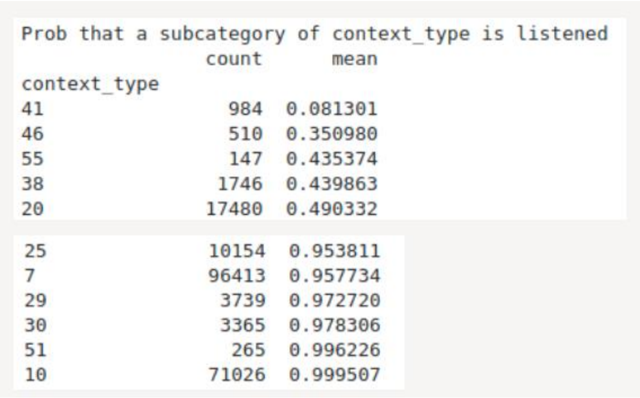
Younger users tend to skip more extensively than older users. While users between 18 and 23 like to skip frequently, users above the age of 23 seem to skip much less (see image 1).

GENRE_ID and CONTEXT_TYPE

The train data contains more than 600 genres and 50 context types. Some subcategories only contain very few values. Since these subcategories are not labelled, we cannot aggregate them to broader subcategories.



```
Prob that a subcategory of genre_id is listened
          count      mean
genre_id
27311       171  0.169591
2740        399  0.208020
473         570  0.329825
931         200  0.355000
26671       140  0.364286
...         ...       ...
222         187  0.941176
9749        158  0.943038
218         224  0.946429
108775      402  0.957711
7281        107  0.962617

[644 rows x 2 columns]
```

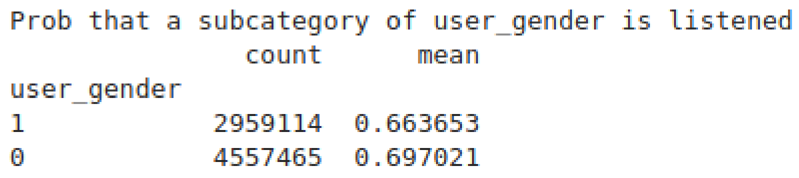


```
Prob that a subcategory of context_type is listened
                  count      mean
context_type
41                  984  0.081301
46                  510  0.350980
55                  147  0.435374
38                 1746  0.439863
20                17480  0.490332

25                10154  0.953811
7                 96413  0.957734
29                 3739  0.972720
30                 3365  0.978306
51                  265  0.996226
10                71026  0.999507
```

Image 2/3: Probability that a specific GENRE_ID or CONTEXT_TYPE keeps listened

Users rarely skip some genres (7281, 108775, 218) but skip others very frequently (27311, 2740, 473).

GENDER_ID



```
Prob that a subcategory of user_gender is listened
                 count      mean
user_gender
1              2959114  0.663653
0              4557465  0.697021
```

Image 4: Probability that a female or male keep listening

Interestingly, females and males do slightly differ in their listening behaviour.

Other variables

The skipping behaviour further deviates strongly across artists and users. The different listening types, platforms, and platform families also influence users' listening behaviour.

To summarize, a user's probability of skipping a song seems to depend on its user profile and the track itself.

1.1.1.1 *Distribution of users listening to songs*

The histograms below show that the user's observations are different from the test data to the train data. The test data contains only one observation per user, whereas the training dataset might contain several hundred times per user.

With this information, we must weight users based on their occurrence. We took a simple approach by calculating a sample weight by using the following formula:
weight = 1 / number of occurrences per user_id

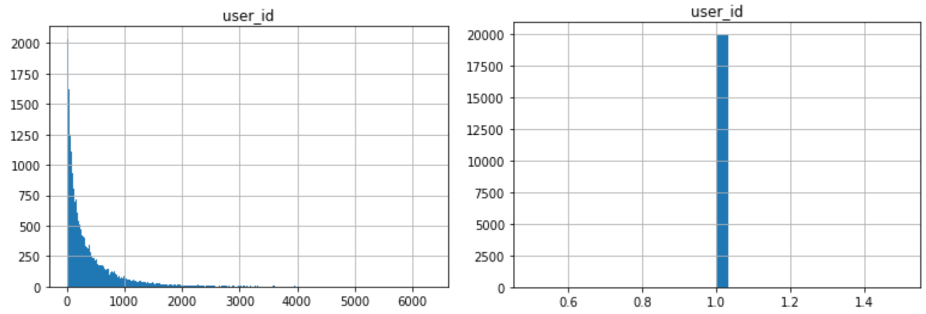


Image 5/6: Distribution per USER_ID in the train vs test dataset

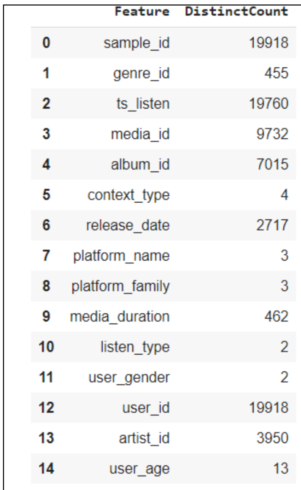
1.1.1.2 *Feature comparison Test vs Train Dataset*

We had to consider that we only can use features to train the model, which are also available in the training dataset. The model is then applied to the test data set. If the model is trained to rely heavily on feature "F1", but the feature does not occur in the test dataset, we cannot use this feature for training.

The column IS_LISTENED is not available in the test dataset as this is the target variable. The column SAMPLE_ID seems to be just an index variable and is not available in the training dataset. Therefore, the model does not depend on this variable.

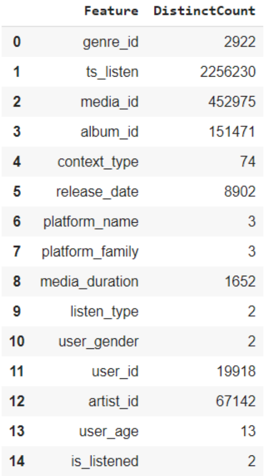
Next, we analyse whether the categories of features available in the test data are the same as in the training dataset. We figured out that this is not the case for two features: CONTEXT_TYPE and LISTEN_TYPE.

We encountered this by listing the distinct count for all the feature, and for each feature, we had a look into the distinct values:



	Feature	DistinctCount
0	sample_id	19918
1	genre_id	455
2	ts_listen	19760
3	media_id	9732
4	album_id	7015
5	context_type	4
6	release_date	2717
7	platform_name	3
8	platform_family	3
9	media_duration	462
10	listen_type	2
11	user_gender	2
12	user_id	19918
13	artist_id	3950
14	user_age	13

Distinct feature count for the **test** dataset



	Feature	DistinctCount
0	genre_id	2922
1	ts_listen	2256230
2	media_id	452975
3	album_id	151471
4	context_type	74
5	release_date	8902
6	platform_name	3
7	platform_family	3
8	media_duration	1652
9	listen_type	2
10	user_gender	2
11	user_id	19918
12	artist_id	67142
13	user_age	13
14	is_listened	2

Distinct feature count for the **train** dataset

Printing out the unique categories of the type CONTEXT_TYPE only showed the following categories in the test dataset: 1, 5, 20, 23

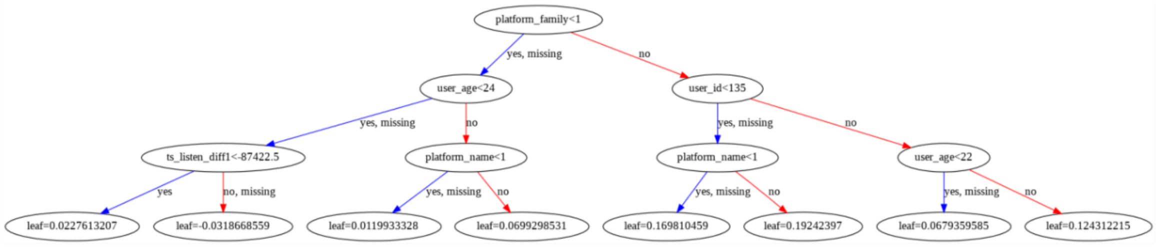
In the data preparation process we have cleaned the CONTEXT_TYP. The LISTEN_TYPE contains two categories in both datasets, but in the test data, only 1 sample exists for the value 0. Thus, we removed the value 0 from the training dataset.

1.1.2 Boosting with XGBoost

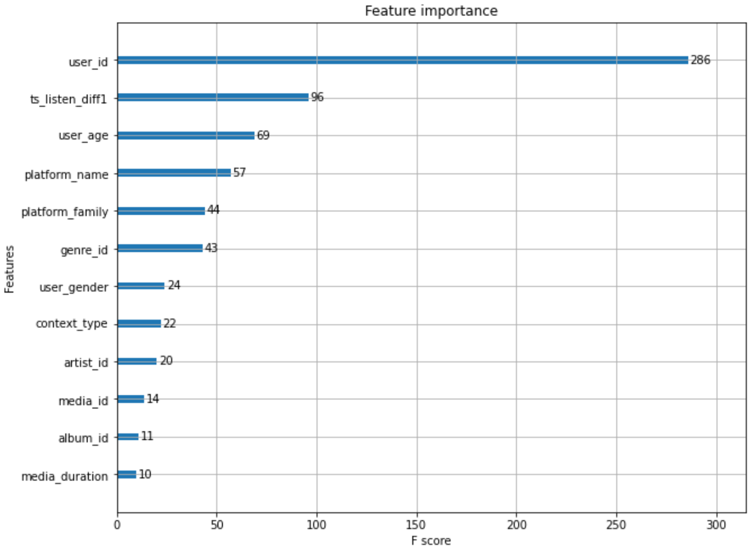
See Jupyter Notebook for code (XGBoost.ipynb)

We used the XGBClassifier from the scikit python package. We tried different hyperparameter settings. Therefore, we used grid-search, for example, to get some of the parameters. Others were chosen by looking at the graphs and evaluation metrics below. Some properties which we used to train the model and hyperparameter are listed here:

- Split: 80/20 Train - Test Split
- Scale_pos_weight: 1 (we have imbalanced classes 60% = 1, 30% = 0, we could scale with this parameter. But as this could not improve the model, we left it default = 1).
- Sample_weight: *weight_per_user = 1 / number of occurance per user_id*
- Learning_rate = 0.1
- Max_depth = 3
- N_estimators=100



Final tree built by XGBoost



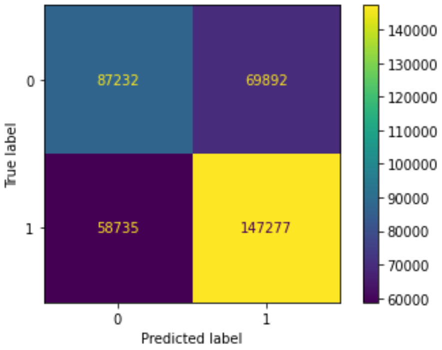
Feature importance

Model Evaluation

Result: With this model and setup, we reached a ROC AUC Value of **64.57 %**

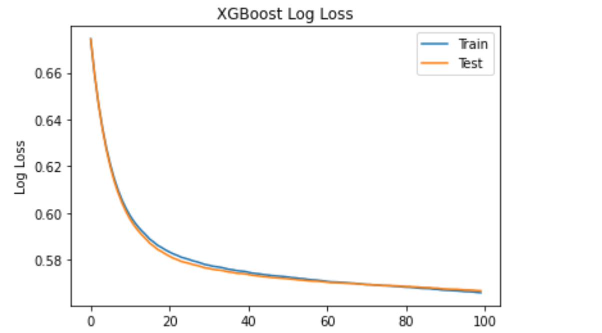
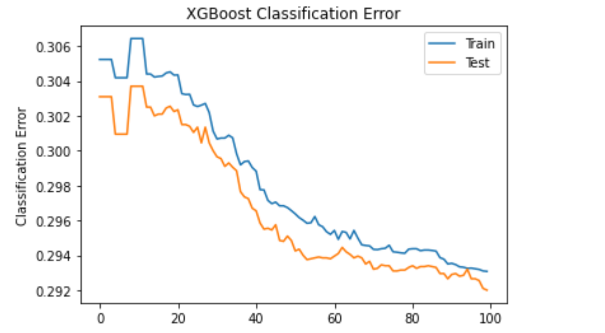
```
              precision    recall  f1-score   support

           0       0.60      0.56      0.58    157124
           1       0.68      0.71      0.70    206012

    accuracy                           0.65    363136
   macro avg       0.64      0.64      0.64    363136
weighted avg       0.64      0.65      0.64    363136
```

The metrics above show the performance of the model. The confusion matrix shows the correct classified as well as the misclassified samples:



Confusion Matrix

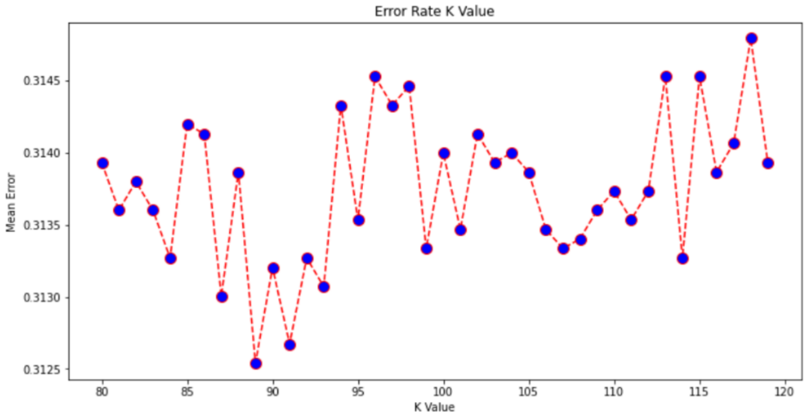
The loss function gave us some insights into the number of estimators to choose. Consequently, the number of trees is built and optimized during the gradient boosting algorithm during the training.

	
Loss Function (x-axis: number of tree)	***Classification Error***

1.1.3 K-Nearest Neighbor

See Jupyter Notebook for code (K-Nearest_Neighbor.ipynb)

We performed a K-Nearest Neighbor classification with the Scikit learn-package. Because our technical infrastructure could not handle the data size when calculating an SVM algorithm, we decided to use a sample of 1%. We did a split of 80/20. To find the optimal K-value, we first defined a random K-value and ran the algorithm using that value. The following plot shows the X-axis's K-values and the corresponding error rate on the Y-Axis. Here we see that the K-value with the lowest error rate is 89.



Model Evaluation

This model achieved a ROC AUC value of **54.8%**. The confusion matrix shows the correct and misclassified values.

```
[[ 731 4152]
 [ 642 9593]]
              precision    recall  f1-score   support

           0       0.53      0.15      0.23      4883
           1       0.70      0.94      0.80     10235

    accuracy                           0.68     15118
   macro avg       0.62      0.54      0.52     15118
weighted avg       0.64      0.68      0.62     15118
```

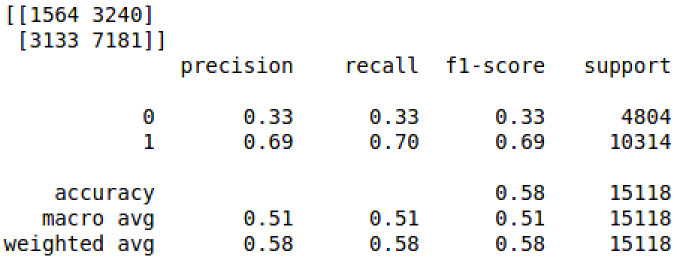
1.1.4 Support Vector Machine

See Jupyter Notebook for code (SVM_Sigmoid.ipynb and SVM_Linear.ipynb)

We also used the Scikit Learn package to perform a Support Vector Machine classification. We applied two different kernels to achieve the best possible result: the linear and sigmoid kernel. Here we also had the problem that our infrastructure could not handle the massive amount of data. Therefore, we decided to apply a sample of 1%.

Model Evaluation Sigmoid Kernel

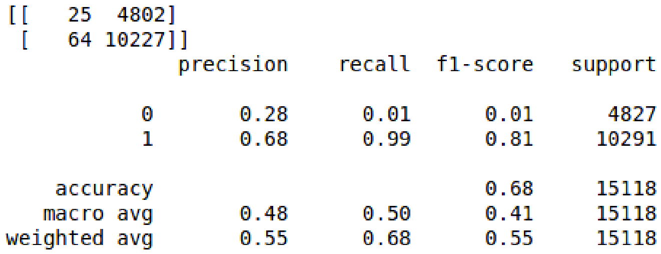
For the Support Vector Machine with the sigmoid kernel we reached a ROC AUC Value of **51%**.



```
[[1564 3240]
 [3133 7181]]
              precision    recall  f1-score   support

           0       0.33      0.33      0.33      4804
           1       0.69      0.70      0.69     10314

    accuracy                           0.58     15118
   macro avg       0.51      0.51      0.51     15118
weighted avg       0.58      0.58      0.58     15118
```

Model Evaluation Linear Kernel

We reached a ROC AUC Value of **49.9%**.



```
[[   25  4802]
 [   64 10227]]
              precision    recall  f1-score   support

           0       0.28      0.01      0.01      4827
           1       0.68      0.99      0.81     10291

    accuracy                           0.68     15118
   macro avg       0.48      0.50      0.41     15118
weighted avg       0.55      0.68      0.55     15118
```

1.2 Results

Model	Size of Dataset	Score ROC AUC
XGBoost	100%	64.5%
KNN	1%	54.%
SVM linear Kernel	1%	49.9%
SVM Sigmoid Kernel	1%	51%

We have seen that the XGBoost Model has the best ROC AUC Score. Fortunately, we could run that model with the whole data. So, we would take the XGBoost model to compete in that challenge.

1.3 Discussion and Conclusion

Q1: What would you do to win this competition?

S1: Our approach to competing in the Kaggle competition was to create different models that accurately predict whether a user would listen to a song. Further, we removed the values of categories present in the train data but missing in the test data. The built models can then be used to propose a given song to the user or not. To decide which model works best, each of us created different models. These models were then evaluated and compared against each other.

Q2: What would you do to solve the problem they are interested in?

S2: Deezer wants to find out how can they keep their user using their platform. Therefore, they want to increase user satisfaction with better track recommendations. Users are not only interested in songs of their past and their bubble. Sometimes, they also like to hear new stuff (Novelty / Diversity). It is like Youtube stated: “There is more art than science in the development of Recommendation = Systems”. The best thing to do is to test lots of algorithms in online A-B testing and evaluate them based on user satisfaction and listening time.

Q3: Why the two solutions might not overlap or why they do?

S3: We fitted the train data in S1 perfectly fitted to the test data by removing non-present subcategories. Consequently, the chosen algorithm best predicts exactly this test data. However, this approach is overfitting the data and will result in dull listening sessions without discovering new music. In practice, the model needs to be much more generalised and flexible.

1.4 Contribution of each team member

At the first meeting, we did not make a clear division of tasks. We looked at the problem, and everyone brought their ideas into the discussion. Then, each member started developing a model to outperform the other members as if it was a competition. At the second meeting, when we looked at the first progress, each member's strengths and weaknesses became apparent, and everyone focused on their strengths. So we complemented each other very well. We can say with a clear conscience that each member put an equal amount of energy into the development of this project work.

1.5 Reflections

Reflection Efrain Rüttimann

I signed up for the course for two reasons. On the one hand because Recommender Systems will be a part of my master thesis and on the other hand out of pure interest. This course gave me an overview of the topic, which I would otherwise have had to work out myself for my master's thesis. Relevant for me were the topics of collaborative filtering, content based filtering, matrix factorization and the hybrid model. I took a closer look at these topics in a Udemy course recommended by the lecturer.

While reading a paper that I and a group member had to present at the end of the course, I came to the following conclusion. I concluded that recommender systems can be used not only for services that serve an end customer, for example movie recommendations on Netflix or product recommendations at Zalando, but also for internal company applications. In our example, we were dealing with analytics software, which recommends data commands based on goal information provided by the user.

I felt enriched by the fact that I was able to work with students from higher semesters on the final report. Since they had advanced programming skills, I had the opportunity to learn from them. For me, the learning effect is greatest when I could apply my acquired knowledge in a practical project. However, I found it a bit unfortunate that the dataset was not suitable to apply a typical Recommender System algorithm, such as Collaborative or User Based Filtering. I also did not have the necessary technical infrastructure to run such a huge dataset. Regarding my master thesis, I really need to be able to ensure an infrastructure that can handle a huge amount of data.

In conclusion, I can say that it was interesting for me to learn what it looks like behind the scenes at Youtube, Netflix, Amazon, etc. I will continue to develop the skills I have learned with regard to my master's thesis. And who knows, maybe these skills will offer me professional opportunities in the future.

Reflection Fabian Trottmann

Before taking the course, I had little understanding of Recommender Systems. From other lectures of this Masters' Studies, I knew models to solve classification and regression problems. I didn't know that there exists such a great varity of totally different approaches to recommend things. Altough I already had all the credits required for the Master Studies (besides the Master Thesis), I felt like I have to take this course, because of the importance of the topic.

During the lecture, a lot of information were provided. I could not deeply understand each topic, so I tried to catch up by reading through the references during the week, provided by the lecturer. The extra material filled the gaps in knowledge. The lessons and the study of the additional information led to quite an intense week. After the course, I repeated the material by working through the Udemy course, which was recommended by the lecturer. This finally led to a solid know how in the methodologies used in Recommender Systems.

After the Udemy course, we started our group project. I first started to use methodologies learned in the course like Collaborative Filtering or Matrix Factorization, as well as Neural Netoworks such as AutoEncoders. But I have realised, that I focused on the wrong methodologies. It is a classic prediction problem with binary output. So I decided, together with the project team, to not focus on methodologies used during the lectures, but using traditional Machine Learning algorithms, stated in the paper.

In conclusion, I learned a lot and the new methodologies will be very useful in the future. Also the project was great fun to work on and I could gain further knowledge by doing the practical work together with the team.

Reflection Andy Gubser

Recommender Systems are somehow different from usual Machine Learning issues. In this course, I wanted to learn what these differences are and how recommender systems work precisely. Since I am quite hooked on Spotify and Netflix, I wanted to know how they recommend new stuff. And last but not least, a colleague has suggested taking the module.

The course in plenum with Guang was very informative. Within a week, he taught many different recommender system algorithms and concepts. I especially liked the coding samples and a long list of extra materials. The industry talk provided some compelling use cases of recommender systems I was not aware of. This considerable information load was quite challenging to process and kept the overview within only one week. Fortunately, the recommended Udemy course repeated the crucial concepts and provided again with some coding examples. This course helped to keep the overview of a broad topic.

Working on the group project was probably the best way to consolidate all the learning. While my colleagues started with modelling, I focused on a deep understanding of the data and easy linear regression models. These insights were finally quite essential to improve our models. This collaboration was quite efficient, and we all improved our knowledge and skills.

To conclude, the module Recommender System was very instructive and provided a good overview of the topic. Most helpful, I found the extra material. So when I can work with recommender systems in the future, I know where to take the first look.