HUST

MACHINE LEARNING AND DATA MINING PROJECT REPORT

Movies Recommendation System

Class ID: **131082**

Subject: **IT4242E**

Group: **22**

Members:

Phạm Trung Hiếu	20176755
Cao Tiến Trung	20190098
Nguyễn Trần Minh Tuấn	20194877

TABLE OF CONTENTS

Members: 1

Problem Overview 3

Review Dataset 4

- **Movies_metadata.csv** 4
- **Credits.csv** 4
- **Keywords.csv** 5
- **Ratings_small.csv** 5

Cleaning and Feature Extraction 6

- **Cleaning** 6
  - **Drop all rows which have non-integer value(s) in the metadata dataset.** 6
  - **Drop duplicate ID** 6
  - **Merge datasets** 7
- **Feature extraction** 8

Machine Learning Algorithm 9

- **Problem Formulation** 9
- **A Straight-Forward Regression Model** 9
- **A Classification Model** 11

Experimental Results 13

- **A Straight-Forward Regression Model Result** 13
  - **Learning Phase** 13
  - **Recommendation Phase** 15
- **A Classification Model Result** 16
  - **Learning Phase** 16
  - **Recommendation Phase** 19

Issues and Overcome Solution 20

- **Learning one user's model at a time** 20
- **Problem of Real Value** 21

Future Improvement 22

- **Matrix Factorization** 22
- **Multiclass Classification Model** 22
- **Try Big Data** 22

References 22

Problem Overview

Nowadays, movies have become an important type of entertainment. Users want to see movies but don’t know which one to see. To help that, we try to recommend related movies that are new to the users by building a recommendation system.

The Recommendation System is a tool that assists users in finding content and overcoming information overload. It predicts user interests and makes recommendations based on the user's interest model. The original content-based recommendation system is a continuation and evolution of collaborative filtering, which does not require item evaluation by the user. Instead, the similarity is calculated based on the information of products selected by consumers in the past, and the recommendation is made accordingly.

Take Netflix for example, Netflix estimates that only 20% of its subscriber video choices come from search, with the other 80% coming from recommendations. “80%” is an incredibly high amount. Netflix believes it could lose $1 billion or more every year from subscribers quitting its service if it weren't for its personalized recommendation engine. And that’s why Netflix thinks that its recommendation engine, however much it could be improved in the future, is already worth so much money to the company.

Inspired by the story of Netflix and other product recommendations, our team will conduct A Recommendation System, namely Movie Recommendation System.

This report introduces a content-based movie recommendation system with the aid of an available dataset from Kaggle.

Review Dataset

In this project we use the The Movies Dataset on Kaggle. The dataset has 7 files: **credits.csv, keywords.csv, links.csv, links_small.csv, movies_metadata.csv, ratings.csv, ratings_small.csv.**
However we only use **movies_metadata.csv, credits.csv, keywords.csv, ratings_small.csv**

Movies_metadata.csv

Here is the fields we will extract from **Movies_metadata.csv**

Attribute	**Explanation**	**Example**
budget	Budget to Product The Movies(in **$** unit)	30000000
genres	Genres of the movie	{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'}, {'id': 10751, 'name': 'Family'}
id	Id of the movie	862, 987
original_language	Language use in the movie	en, fr, …
production_companies	Studios participate in production of movie	[{'name': 'TriStar Pictures', 'id': 559}, {'name': 'Teitler Film', 'id': 2550}, {'name': 'Interscope Communications', 'id': 10201}
production_countries	Countries participate in production of movie	[{'iso_3166_1': 'DE', 'name': 'Germany'}, {'iso_3166_1': 'US', 'name': 'United States of America'}]
revenue	Income of Movie	373554033.0
runtime	Movie's Runtime in minutes	81
vote_average	Average vote of movie on IMDB	7.7, 4.0, 0.0, …
vote_count	Number of Votes for movie on IMDB	5415.0, 92.0, 173.0

Credits.csv

Here is the fields we will extract from **credits.csv**

Attribute	**Explanation**	**Example**
cast	Actors of movies	{'cast_id': 2, 'character':

		'Max Goldman', 'credit_id': '52fe466a9251416c75077a8d', 'gender': 2, 'id': 6837, 'name': 'Walter Matthau', 'order': 0, 'profile_path': '/xJVkvprOnzP5Zdh5y63y8HHniDZ.jpg'}
crew	The movie's managers	{'credit_id': '52fe466a9251416c75077a89', 'department': 'Directing', 'gender': 2, 'id': 26502, 'job': 'Director', 'name': 'Howard Deutch', 'profile_path': '/68Vae1HkU1NxQZ6KEmuxIpno7c9.jpg'}
id	Id of Movie	862, 987

Keywords.csv

Here is the fields we will extract from **keywords.csv**

Attribute	**Explanation**	**Example**
keywords	keywords about movie	[{'id': 949, 'name': 'terrorist'}, {'id': 1562, 'name': 'hostage'}, {'id': 1653, 'name': 'explosive'}, {'id': 193533, 'name': 'vice president'}]
id	ID of movie	862, 987

Ratings_small.csv

Here is the fields we will extract from **ratings_small.csv**

Attribute	**Explanation**	**Example**
userId	user's ID	1, 2, …, 671
movieId	ID of movie	862, 987
rating	Rating user rated on movie	0.5, 1.0, …, 4.5, 5.0

Cleaning and Feature Extraction

Cleaning

In the cleaning phase, we use the pandas package to load, manipulate and store the datasets.

Drop all rows which have non-integer value(s) in the *metadata* dataset.

We begin by writing a simple function to check an integer number:

```
def isInteger(number):
    try:
        a = int(number)
    except:
        return False
    return True
```

Then use this function to find all the rows in *metadata* that have non-integer value:

```
metadata['isInteger'] = metadata['id'].apply(isInteger)
metadata[metadata['isInteger'] == False]
```

19730	- Written by Ørnås	0.065736	/ff9qCepilowshEtG2GYWwzt2bs4.jpg
29503	Rune Balot goes to a casino connected to the ...	1.931659	/zV8bHuSL6WXoD6FWogP9j4x80bL.jpg
35587	Avalanche Sharks tells the story of a bikini ...	2.185485	/zaSf5OG7V8X8gqFvly88zDdRm46.jpg

3 rows × 25 columns

After that, we drop those rows by ID and return None:

```
metadata.drop([29503, 35587, 19730], inplace=True)
```

Drop duplicate ID

Since all records should have their own ID, we have to drop those having duplicate ID. First, we take a look at the number of times an ID appears in *metadata*:

```
metadata['id'].value_counts()
```

```
141971    3
5511      2
132641    2
10991     2
168538    2
          ..
55135     1
15877     1
72272     1
1549      1
461257    1
Name: id, Length: 45433, dtype: int64
```

(ID 141971 appears 3 times whereas 5511 appears twice)

Now we know that there are duplicate rows in *metadata*, we proceed to drop those duplicates:

```
metadata.drop_duplicates(subset=['id'], inplace=True)
```

```
credits.drop_duplicates(subset=['id'], inplace=True)
```

```
keywords.drop_duplicates(subset=['id'], inplace=True)
```

Finally, return the *metadata*, *credits* and *keywords* with all duplicate IDs removed.

Merge datasets

In the following step, *metadata*, *credits* and *keywords* are merged into *dataset* with a database-style join on the ID column:

```
dataset = metadata.merge(credits, how = 'inner', on = "id")
dataset = dataset.merge(keywords, how = 'inner', on = "id")
```

Since our project only works with *genres, crew, cast, keywords, popularity, vote_average, language, runtime, vote_count, production_companies, production_countries, overview*, the *final_dataset* uses only those features as its columns:

```
columns = ['genres', 'budget', 'id', 'original_language', 'overview', 'production_companies', \
           'production_countries', 'runtime', 'revenue', 'vote_average', 'vote_count', 'cast', \
           'crew', 'keywords', 'title']
final_dataset = dataset[columns]
```

However, the *final_dataset* still contains *null value* fields:

```
final_dataset.isnull().sum()
```

```
genres                     0
budget                     0
id                         0
original_language         11
overview                 954
production_companies       3
production_countries       3
runtime                  260
revenue                    3
vote_average               3
vote_count                 3
cast                       0
crew                       0
keywords                   0
title                      3
dtype: int64
```

Therefore, we have to fill in those null value indexes before ending the cleaning phase.
As *overview* and *runtime* take up most of the null field, we only handle these 2 columns:

- Fill each *runtime* null value with the mean value of *runtime* in *final_dataset*

```
final_dataset['runtime'] = final_dataset['runtime'].fillna(final_dataset['runtime'].mean())
```

- Fill each *overview* null value with an empty string in place of 'null'

```
final_dataset['overview'] = final_dataset['overview'].fillna("")
```

Feature extraction

Feature Extraction codes are in file **profiles.ipynb** or **profiles_colab.ipynb**
There is a module FeatureExtractor.py
This module use **genres, production_companies, production_countries, cast, crew, keywords** columns to generate features including **CastsRank, NumLeadActors, HasTop50Actors, NumCrews, crewsTeamRank, NumTopCrew, HasTopCrew, NumStudios, StudioRank, NumTopStudios, HasTopStudio, CountryRank, keywordRank, Action, Adventure, Animation, Comedy, Crime, Documentary, Drama, Family, Fantasy, Foreign, History, Horror, Music, Mystery, Romance, Science Fiction, TV Movie, Thriller, War, Western**
The generation is based on the revenue of movies to determine how famous casts, crews, production_companies are. For **genres**, it is one hot encoding.

The FeatureExtractor.py file we reference from FeatureExtractor.py and have made some modification to match our dataset.

Machine Learning Algorithm

In this project, we build a **Content Based Recommendation System**.

Problem Formulation

Given profile of movies **M**, **R** true ratings by users to some movies.

	M1	M2	M3	M4
User1	5.0		4.0	
User2	2.0	2.0	3.0	1.0
User3	4.0		3,5	

	F1	F2
M1	0.1	0.9
M2	0.3	0.8
M3	0.5	0.8
M4	0.2	0.6

M is the profile of movies with shape of (2830, 39) which stands for 2830 movies, 38 features and 1 intercept factor(all ones). This matrix is standardized from the original feature extracted dataset to the mean of 1 and standard deviation of 0 before adding the intercept factor 1.

R is true ratings by users to movies with shape of (671, 2830) which stands for 671 users and 2830 movies.

Will find the user model **U** and **b** which we will call a matrix with shape of (671, 38) which stands for 671 users and 38 features. Each row is the model of each user, which can be called "user's interest". And **b** is a matrix with shape (671, 1) the biased (intercept).

A Straight-Forward Regression Model

First predict the ratings by:

$$\hat{R} = [U, b] * M^T$$

$\hat{R}$ is the predicted ratings with same shape as of 671 x 2830, operator * is matrix multiplication and [U,b] means that they are concatenated axis = 1. (Stack horizontal and in this case the new shape is **(671, 39)**)

After getting the predicted ratings, we use Mean Squared Error (MSE) to define a loss for our model.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Use MSE because it penalizes the output that is far from the true output.

For our case:

$$L = MSE(R, \hat{R})$$

Let **S** is the list of pairs of indexes of movies that user rates for, this will not include the index of the unrated movie.

$$S = [(i,j)|R_{i,j} \neq 0.0]$$

The loss MAE becomes:

$$L = MAE(R, \hat{R}) = \frac{1}{|S|}\sum_{i,j \in S}|R_{i,j} - \hat{R}_{i,j}|$$

Add more regularization term L2,

$$L = MSE(R, \hat{R}) = \frac{1}{|S|}\sum_{i,j \in S}(R_{i,j} - \hat{R}_{i,j})^2 + \lambda \sum_i \sum_j (U_{i,j})^2$$

Here we apply L2 regularization on U only, not on the intercept b.

After we have the loss, compute: $\frac{\delta L}{\delta U}$

With the partial derivatives of U we can update U after each loop to train U.

In the implementation in pytorch, we use the Autograd module of Pytorch to automatically calculate the derivatives and use the built-in Adam optimizer to train for params(**U**). The main MSE loss will use **MSELoss()** and regularization terms will be replaced as **weight_decay** param.

The metrics is also MSE Loss (Mean Squared Error).

A Classification Model

In this model **R_hat** will not be the predicted rating any more. It will be the **probability** that the user will rate more than 4.0 for the movie. In the file Content_Based_Classification.ipynb we will transform the ratings dataset to 1 and 0 for the rating field.

First predict the probs by:

$$\hat{R} = sigmoid([U, b] * M^T)$$

$\widehat{R}$ is the probability matrix with same shape as of 671 x 2830, operator * is matrix multiplication and [U,b] means that they are concatenated axis = 1. (Stack horizontal and in this case the new shape is **(671, 39)**).
Sigmoid is applied to all elements of a matrix after the multiplication of 2 matrices.

After getting the probability matrix, we use Binary Cross Entropy to define a loss for our model.

$$BCE = \frac{1}{n}\sum_{i=0}^{n} y_i \,.\log(\hat{y}_i) + (1 - y_i).\log\,(1 - \hat{y}_i)$$

For our case:

$$L = BCE(R, \hat{R})$$

Let **S** is the list of pairs of indexes of movies that user rates for, this will not include the index of the unrated movie.

$$S = [(i, j)|R_{i,j} \neq 0.0]$$

The loss MAE becomes:

$$L = BCE\big(R, \hat{R}\big) = \frac{1}{|S|}\sum_{i,j \in S} R_{i,j}.\log\big(\hat{R}_{i,j}\big) + \big(1 - R_{i,j}\big).\log\,(1 - \hat{R}_{i,j})$$

Add more regularization term L2,

$$L = \frac{1}{|S|}\sum_{i,j \in S} R_{i,j}.\log\big(\hat{R}_{i,j}\big) + \big(1 - R_{i,j}\big).\log\,(1 - \hat{R}_{i,j}) + \lambda \sum_i \sum_j (U_{i,j})^2$$

Here we apply L2 regularization on U only, not on the intercept b.

After we have the loss, compute: $\frac{\delta L}{\delta U}$

With the partial derivatives of U we can update U after each loop to train U.

In the implementation in pytorch, we use the Autograd module of Pytorch to automatically calculate the derivatives and use the built-in Adam optimizer to train for params(**U**). The main Binary Cross Entropy Loss will use **BCELoss()** and regularization terms will be replaced as **weight_decay** param.

The metrics we use use **Accuracy** which is the number of output guessed right.

Experimental Results

A Straight-Forward Regression Model Result

Learning Phase

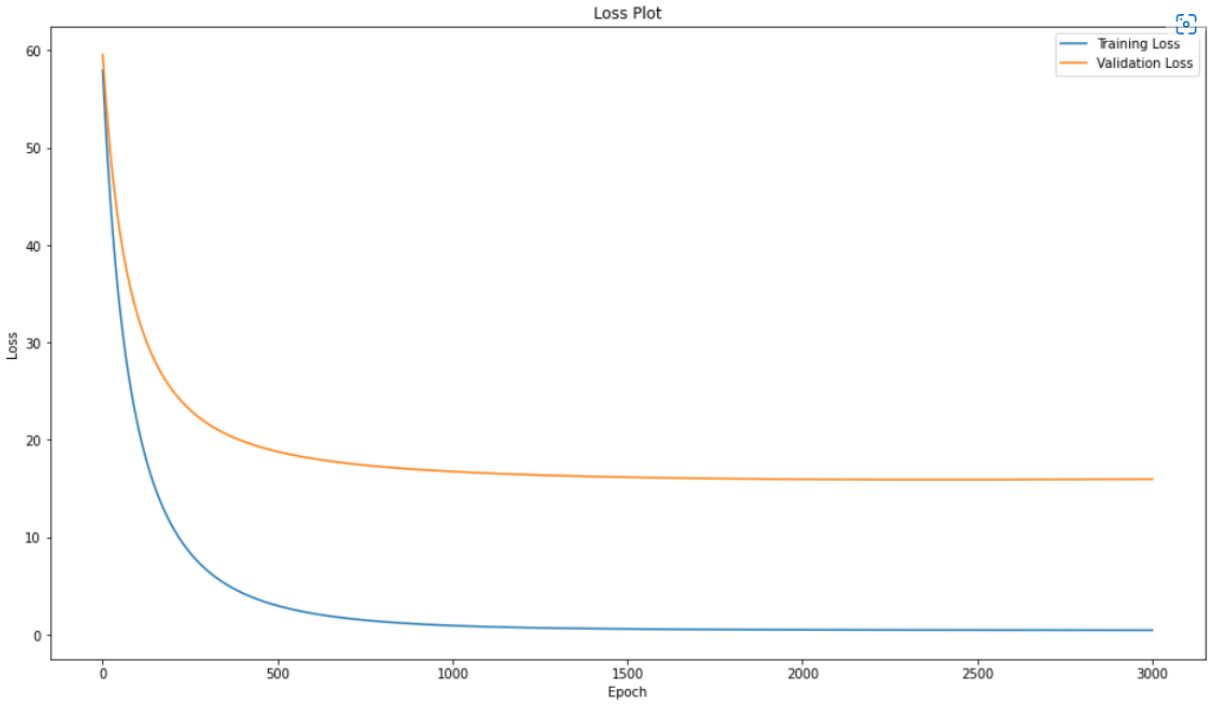
After splitting the rating dataset into two parts **ratings_train.csv** and **ratings_test.csv,** we use them as train dataset and validation dataset.

Using MSELoss and Adam Optimizer.

+ Without regularization

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam([U], lr = 0.005)
optimizer.add_param_group({'params': b, "lr": 0.005, "weight_decay": 0})
```

Take EPOCH of 3001, here is the plot of **MSE** loss.



MSE Loss Min on Train Set: **0.467**

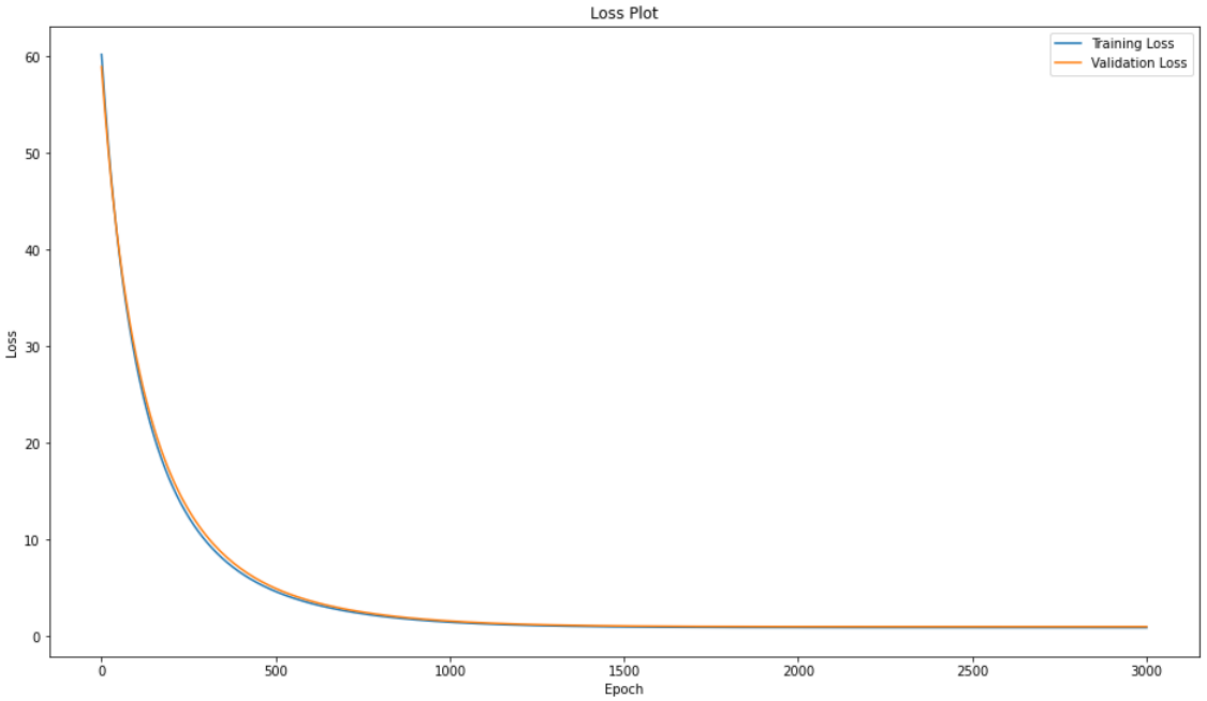
MSE Loss Min on Validation Set: **15.95**

The model overfit the train data.

+ With regularization lamba of **0.05**

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam([U], lr = 0.005, weight_decay = 0.05)
optimizer.add_param_group({'params': b, "lr": 0.005, "weight_decay": 0})
```

Take EPOCH of 3001, here is the plot of MSE Loss while training:



MSE Loss Min on Train Set: **0.85**

MSE Loss Min on Validation Set: **0.95**

Regularization has reduced overfitting.

We save the model with the lowest **MSE loss** on the Validation Set.

```
minLoss
```

```
tensor(0.9489, dtype=torch.float64, grad_fn=<MseLossBackward0>)
```

The parameters to save are U and b

```
bestU.detach().numpy().tofile("ModelStorage/U.txt")
bestb.detach().numpy().tofile("ModelStorage/b.txt")
```

Recommendation Phase

Specify **user = 10**
Predict the ratings of the user for each movie and sort decreasing order.

```
ratings_user_df['rating'].sort_values(ascending = False)
```

```
1016    4.023982
2537    4.017939
2743    4.017563
2583    4.016156
1908    4.015993
          ...
1422    3.923036
1721    3.922210
710     3.918737
2051    3.913459
8       3.909853
Name: rating, Length: 2830, dtype: float64
```

After that remove from that list the movie that the user has seen.

```
ratings_user_df.drop(ratings_user_df[ratings_user_df['movie_Id'].isin(movies_have_seen)].index, inplace = True)
```

Take the first 20 movies and recommend them to the user.

Movie to recommend user

```
[ ]  for x in indexToRecommend:
         print(idToTitle[indexToId[x]])
```

```
The One-Man Band
Idol of the Crowds
The Collector
The Front Page
Hero
42nd Street
Jezebel
Bullitt
Last Year at Marienbad
Aguirre: The Wrath of God
Contraband
Vivre Sa Vie
Bad Education
The Real McCoy
Only the Lonely
Arena
Bedevilled
Kindergarten Cop
Frankenstein
```

A Classification Model Result

Learning Phase

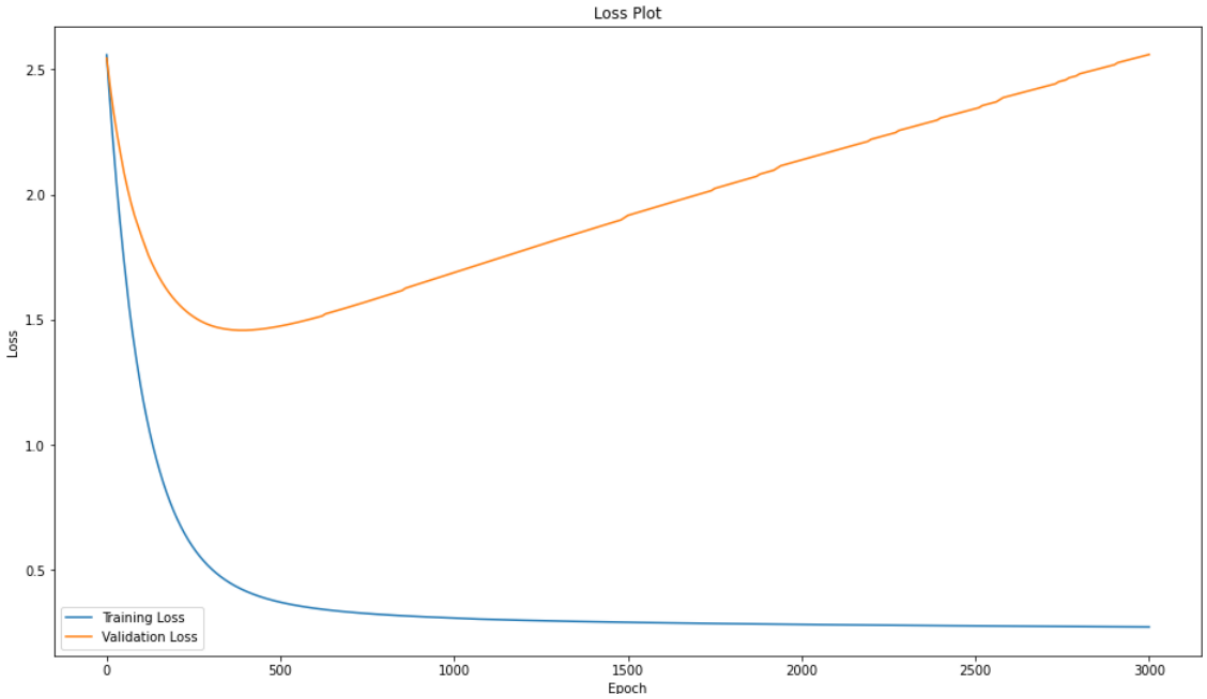
After splitting the rating dataset into two parts **ratings_train.csv** and **ratings_test.csv,** we use them as train dataset and validation dataset.

Using BCELoss and Adam Optimizer.

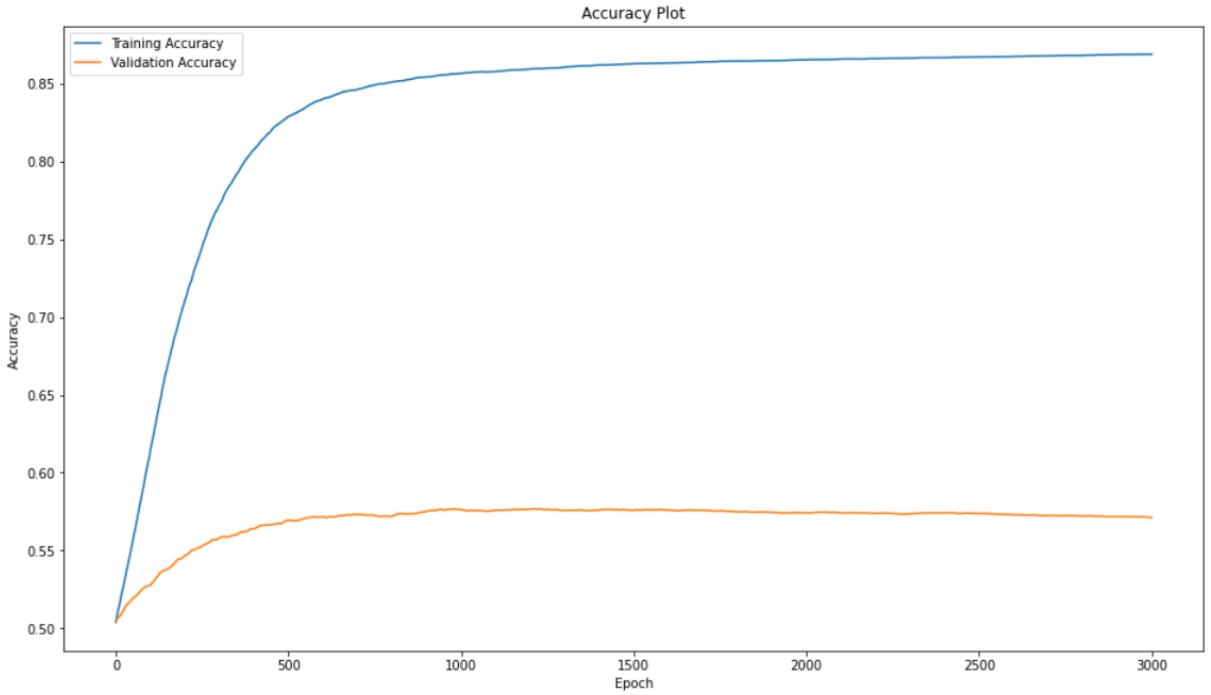
+ Without Regularization

```
criterion = nn.BCELoss()
optimizer = torch.optim.Adam([U], lr = 0.005)
optimizer.add_param_group({'params': b, "lr": 0.005, "weight_decay": 0})
```

Take EPOCH of 3001, here is the loss and accuracy plot while training.



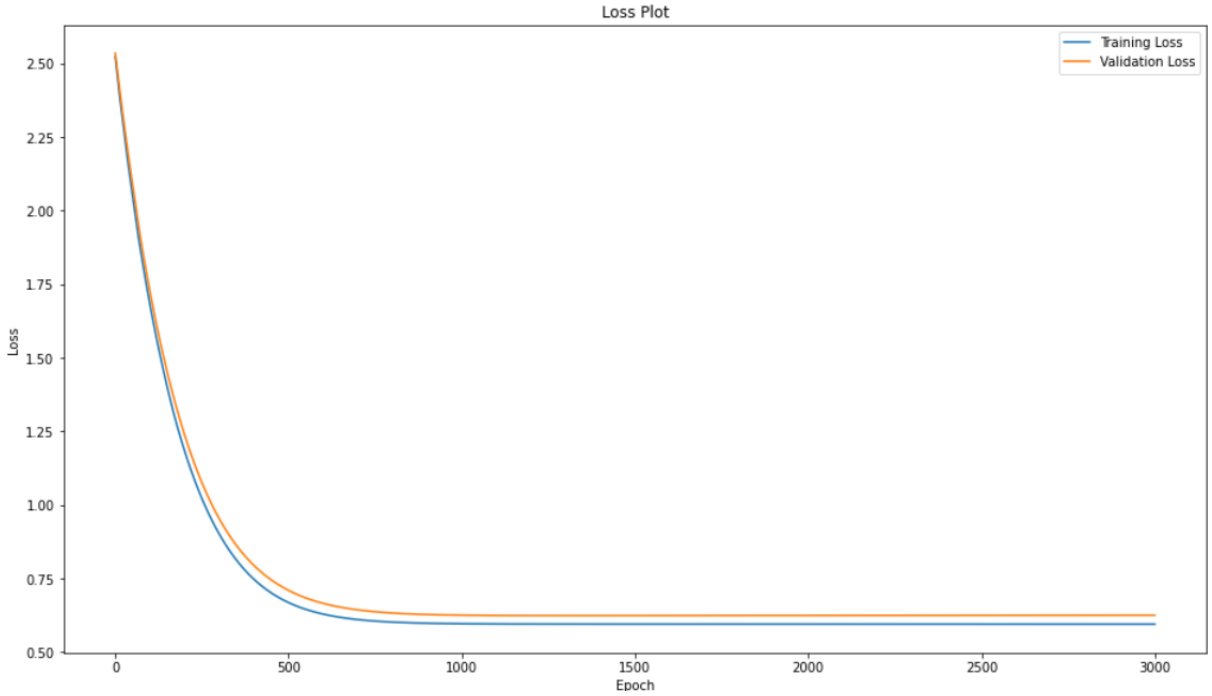
When we keep training after 500 epochs, training Loss decreases but validation loss increases rapidly. Overfitting appears here.

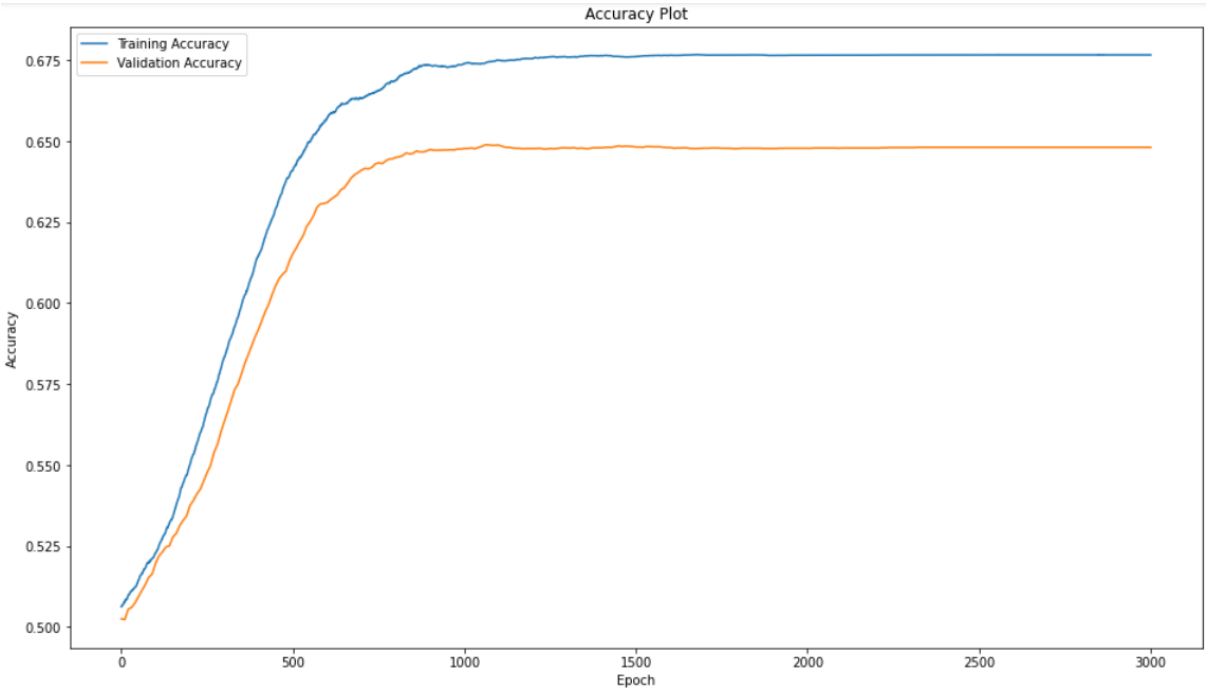


+ With Regularization of lambda 0.05 on U

```
criterion = nn.BCELoss()
optimizer = torch.optim.Adam([U], lr = 0.005, weight_decay = 0.05)
optimizer.add_param_group({'params': b, "lr": 0.005, "weight_decay": 0})
```

Take EPOCH of 3001, here is the loss and accuracy plot while training.





The loss plot looks better compared to the case of not Regularization. And the accuracy in the Validation Test increased by 10%.

We save the model with the highest accuracy on the Validation Set.

```
maxAcc
```

```
tensor(0.6489)
```

The parameters to save are U and b

```
bestU.detach().numpy().tofile("ModelStorage/U.txt")
bestb.detach().numpy().tofile("ModelStorage/b.txt")
```

Recommendation Phase

Specify **user = 10**
Predict the variable shouldRecommend (rating >=4.0) we have the dataframe.

	movie_Id	shouldRecommend
0	949	0
1	710	0
2	1408	0
3	524	1
4	4584	1
...	...	...
2825	80831	1
2826	3104	0
2827	64197	0
2828	98604	1
2829	49280	0

2830 rows × 2 columns

After that remove from that list the movie that the user has seen.

```
ratings_user_df.drop(ratings_user_df[ratings_user_df['movie_Id'].isin(movies_have_seen)].index, inplace = True)
```

Take the first 20 movies and recommend them to the user.

Movie to recommend user

```
[ ]  for x in indexToRecommend:
         print(idToTitle[indexToId[x]])
```

```
The One-Man Band
Idol of the Crowds
The Collector
The Front Page
Hero
42nd Street
Jezebel
Bullitt
Last Year at Marienbad
Aguirre: The Wrath of God
Contraband
Vivre Sa Vie
Bad Education
The Real McCoy
Only the Lonely
Arena
Bedevilled
Kindergarten Cop
Frankenstein
```

Issues and Overcome Solution

In this part, our code is in ***Machine Learning/Old Model,*** it will not be deployed because that is only for experimenting.

Learning one user's model at a time

When trying to use **Linear Regression** on learning one user's model at a time.

```
model = LinearRegression()
model.fit(X_Train, Y_Train)
print("Train Score: ", model.score(X_Train, Y_Train))
print("Test Score: ", model.score(X_Test, Y_Test))
```

✓ 0.1s

```
Train Score:  1.0
Test Score:  -82.77418102157297
```

The model dramatically overfitted. Reason for that is each user only rates for a few movies but the movie profile has 38 features. We considered reducing the features but that is not a good idea.

We also tried to use Ridge Regression which uses L2 regularization term to force the weight under threshold and GridSearchCV to find lambda term of L2.

```
parameters = {'alpha':[0.1, 1, 10, 10e3, 10e6, 10e9, 10e12, 10e15, 10e20]}
scorer = make_scorer(mean_squared_error, greater_is_better = False)
clf = GridSearchCV(Ridge(), param_grid=parameters, scoring=scorer)
clf.fit(X_Train.append(X_Test), np.append(Y_Train, Y_Test))
```

```
clf.best_params_
```

✓ 0.1s

```
{'alpha': 1e+21}
```

The found alpha best param is 1e21.

```
model = Ridge(alpha = 1e21)
model.fit(X_Train, Y_Train)
print("MSE on Train:", mean_squared_error(Y_Train, model.predict(X_Train)))
print("MSE on Test:", mean_squared_error(Y_Test, model.predict(X_Test)))
```

✓ 0.2s

```
MSE on Train: 1.0998840073210203
MSE on Test: 0.8367549698517375
```

The result of MSE seems good but when we use that to predict the ratings. The results are somewhat constant.

```
model.predict(X_Test)
```
✓ 0.1s

```
array([3.35486902, 3.35483419, 3.35480433, 3.35480433, 3.35487091,
       3.35480433, 3.35480433, 3.35492474, 3.35497131, 3.35482349,
       3.35480595, 3.35487669, 3.35489645, 3.35480433, 3.35480433,
       3.35485448, 3.35482369, 3.35480433, 3.3548096 , 3.35489186,
       3.3548108 , 3.35481882, 3.35484738, 3.35480433, 3.35480433,
       3.35480433, 3.35480433, 3.35480433])
```

Problem of Real Value

```
model.predict(X_Test)
```
✓ 0.7s

```
array([2.64362681, 8.25983877])
```

Ratings can only be between 0 to 5 but the predicted one is over **5**. Because we use real value for the output.
We try to use the MLPClassifier. MultiLayer Perceptron Classifier is a Neural Network with a softmax layer at the end when used for multiclass classification.

```
model = MLPClassifier(hidden_layer_sizes=(100, 50, 25, 10), alpha = 100, random_state=1)
model.fit(X_Train, Y_Train)
```

```
model.predict(X_Test)
```

```
array(['3.0', '3.0', '2.0', '3.5', '3.0', '2.0', '3.0', '3.0', '3.0',
       '3.0', '3.0', '3.0', '3.0', '3.5', '3.5', '3.0', '3.0', '2.0',
       '3.0', '3.0', '3.0', '3.0', '3.0', '3.5', '2.0', '2.0', '2.0',
       '2.0'], dtype='<U3')
```

```
print("Train MSE: ", mean_squared_error(Y_Train, model.predict(X_Train)))
print("Test MSE: ", mean_squared_error(Y_Test, model.predict(X_Test)))
```
✓ 0.1s

```
Train MSE:  1.6935483870967742
Test MSE:  1.5
```

That is reasonable. But this model can only train one user at a time.

Future Improvement

Matrix Factorization

Content-Based Recommendation System has advantages and also some disadvantages. One of them is that movie profiles are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features. Later we may use Matrix Factorization instead of Content-Based as currently working.

Multiclass Classification Model

The output of our model is a real number so it is hard to evaluate the predicted ratings. Metrics may be ambiguous or used in the wrong case. Later we will research a classification model.

Try Big Data

Due to memory limitation, we can use only the ratings_small.csv dataset (not the ratings.csv one). Later we may try to use high efficient computing on working with big datasets like that.

References

Here is the reference we refer to when experimenting with models.

- Content-based Recommendation Systems - Machine Learning Cơ Bản
- Recommender Systems | Content Based Recommendations — [Andrew Ng]