Music Recommendation Service

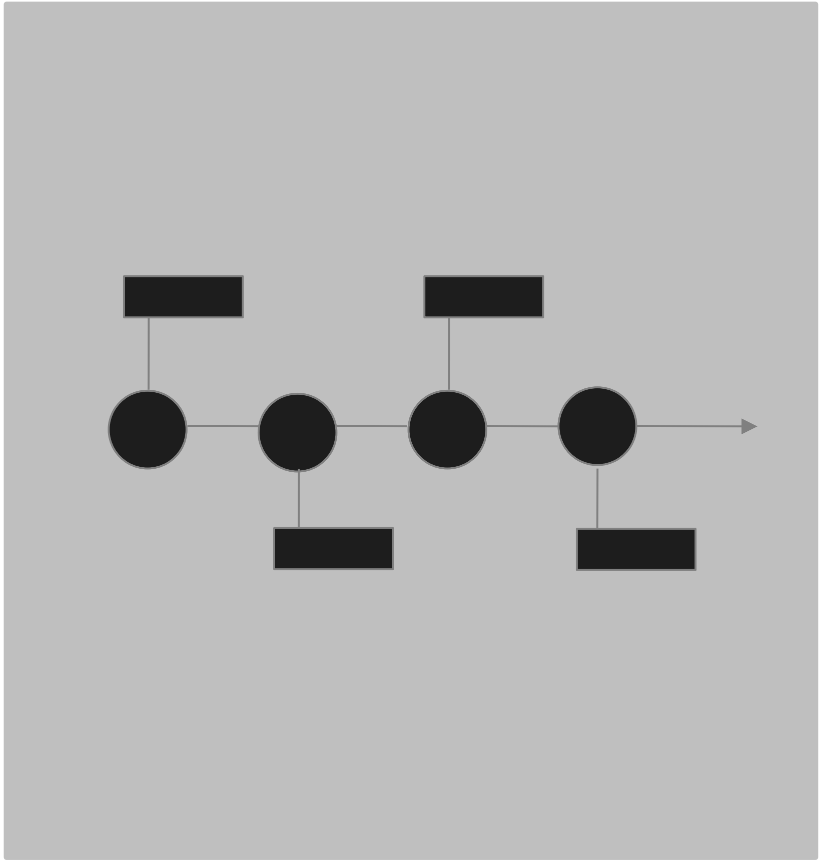
Group 2: Jason Ingram, Isaiah Martinez

Table of contents

1) **Dataset Overview**
   a) **Data Visualization**
   b) **Data Preparation**
2) **Models & Algorithms**
3) **Future Work**
4) **Conclusion**

01

Data Overview



Data Overview

User and Music Data contained in several 'Parquet' files

- Required an external library - Arrow
- Turned into CSV

MusicBrainz

APACHE ARROW

Dataset Visualization

	user_id	artist_name	release_name	recording_name	date	time
1	16493	Greg MacPherson Band	Good Times Coming Back Again	Numbers	2006-11-29	13:19:10
2	8793	Wolfgang Amadeus Mozart	The World of Sacred Music	Ave Verum Corpus	2006-11-29	13:52:16
3	6263	Japan	Tin Drum	Ghosts	2006-11-29	13:59:42
4	5838	Enigma	The Cross of Changes	Age of Loneliness (Carly's Song)	2006-11-29	13:55:42
5	1061	Paul Simon	Graceland	All Around the World or the Myth of Fingerprints (early versi...	2006-11-29	14:04:29
6	5838	Enigma	The Cross of Changes	Age of Loneliness (Carly's Song)	2006-11-29	14:39:42
7	8115	Pixies	Surfer Rosa	I'm Amazed	2006-11-29	14:44:38
8	1061	Paul Simon	Graceland	All Around the World or the Myth of Fingerprints (early versi...	2006-11-29	15:07:16
9	6419	Partizani	Unknown Album	Pociva jezero v tihoti	2006-11-29	15:14:45
10	4685	Red Hot Chili Peppers	Stadium Arcadium	Dani California	2006-11-29	15:26:37

First 10 entries from file

Data Preparation

Encoded each categorical column:

	user_id	artist_name	release_name
0	16493	Greg MacPherson Band	Good Times Coming Back Again
1	8793	Wolfgang Amadeus Mozart	The World of Sacred Music
2	6263	Japan	Tin Drum
3	5838	Enigma	The Cross of Changes
4	1061	Paul Simon	Graceland

```python
#encode the categorical variables:
#artist_name, release_name, recording_name

df['artist_name'] = df['artist_name'].astype('category').cat.codes
df['release_name'] = df['release_name'].astype('category').cat.codes
df['recording_name'] = df['recording_name'].astype('category').cat.codes

#print new head and shape of df
print(df.head(), "\n", df.shape)

#print the translation from code to artists, so we can see it worked

print(dict(zip(df['artist_name'].astype('category').cat.categories, df['artist_na
```

Python

	user_id	artist_name	release_name	recording_name	date	time
0	16493	3665	4240	9463	2006-11-29	13:19:10
1	8793	9899	10764	1240	2006-11-29	13:52:16
2	6263	4286	10945	5040	2006-11-29	13:59:42
3	5838	2913	9918	676	2006-11-29	13:55:42
4	1061	6645	4260	759	2006-11-29	14:04:29

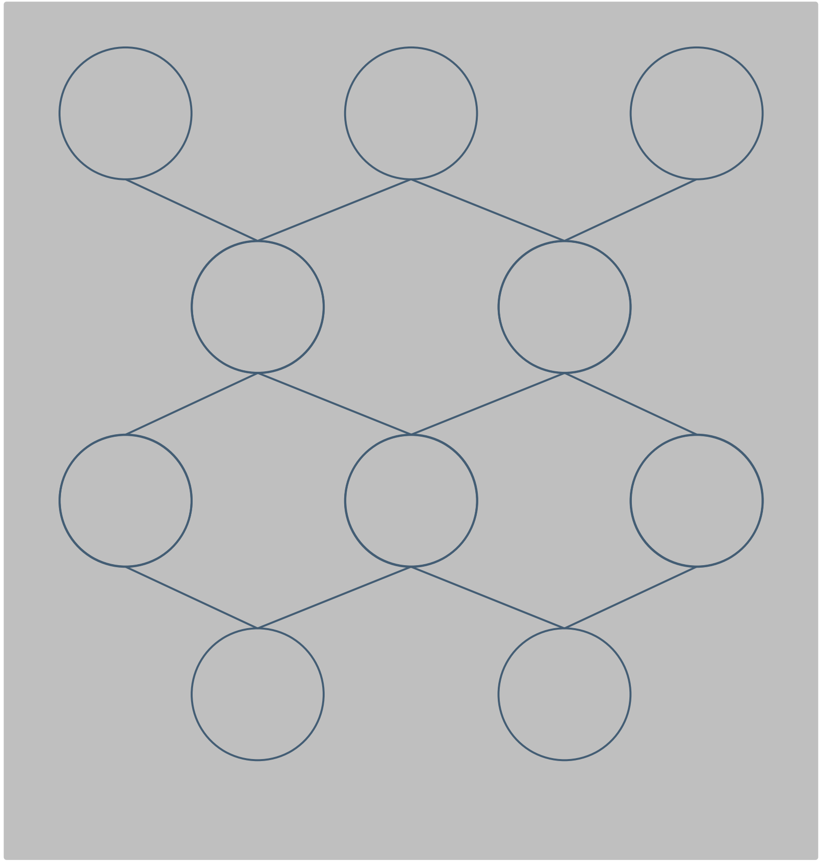
```
(66936, 6)
{0: 3665, 1: 9899, 2: 4286, 3: 2913, 4: 6645, 5: 2913, 6: 6797, 7: 6645, 8: 6593, 9: 712
```

User Interaction Matrix

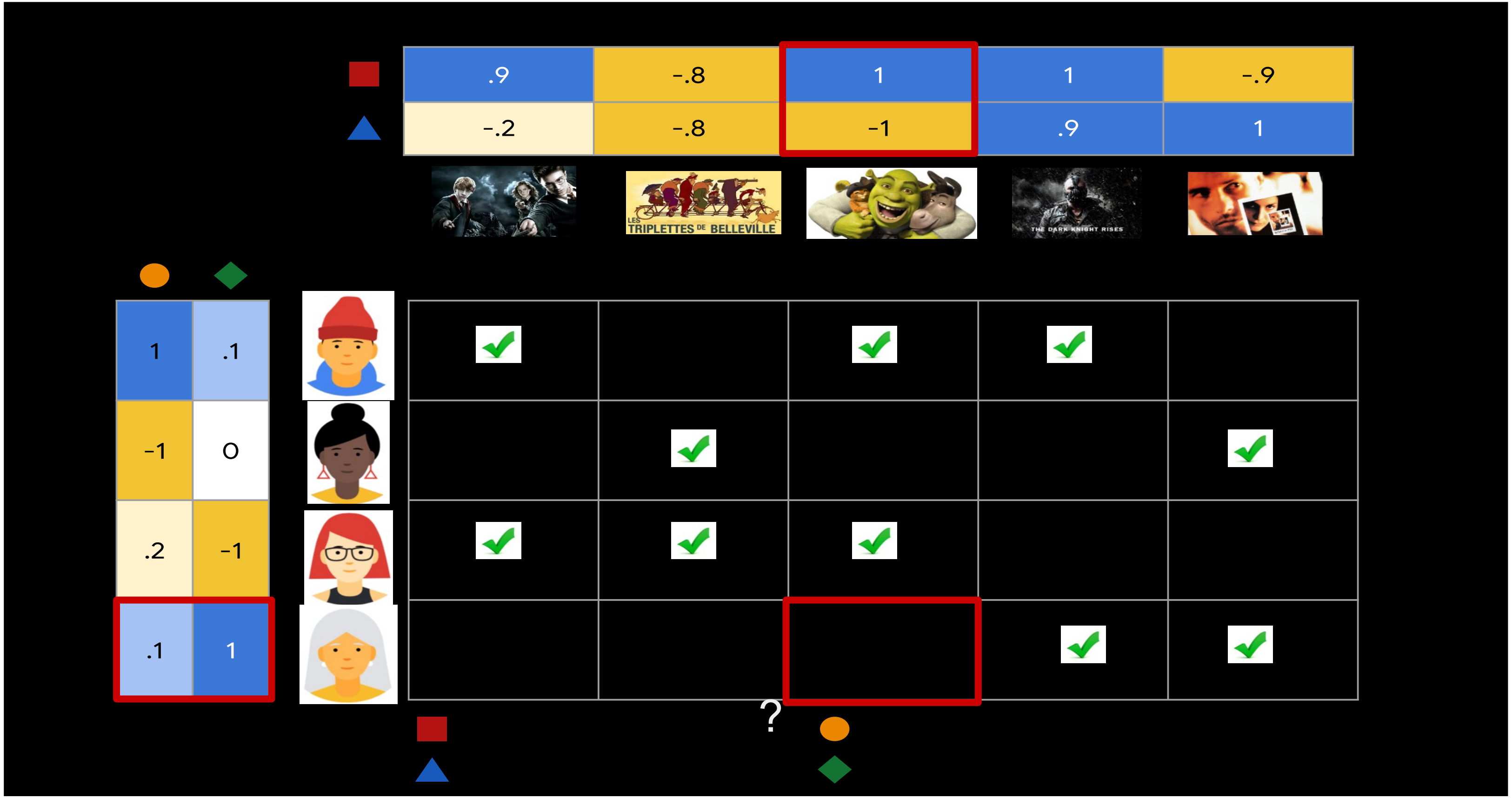
USERS \ ARTISTS	Artist 1	Artist 2	Artist 3
User 1	1	0	1
User 2	0	1	0
User 3	0	1	1

02

Models & Algorithms



Collaborative Filtering


.9
-.8
1
1
-.9
-.2
-.8
-1
.9
1
LES TRIPLETTES DE BELLEVILLE
THE DARK KNIGHT RISES
1
.1
-1
0
.2
-1
.1
1
?

CF (ALS) - Overview

Step 1

Analyze smaller k-sized matrices

Step 2

Optimize User & Item matrices independently

Step 3

Iterate until convergence

CF (ALS) – Code Explanation

Training

(User_ID, Artist)

```
(5, 3617)       1
(5, 4580)       1
(5, 8272)       1
(5, 10094)      1
(6, 344)        1
(6, 1396)       1
(6, 2571)       1
(6, 2685)       1
(6, 3431)       1
(6, 4560)       1
(6, 5746)       1
(6, 6133)       1
(6, 6346)       1
```

Feeds into

```
import implicit

#model fitting
model = implicit.als.AlternatingLeastSquares(factors = 150, regularization = 0.001, iterations = 100)
model.fit(user_artist_train_matrix)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
100%|██████████| 100/100 [00:44<00:00,  2.26it/s]
```

Hyper-parameters:

- Factors (k)
- Regularization (λ)
- Iterations
- Alpha (not seen in the above image)

CF (ALS) – Code Explanation

Testing

```
#model predictions
import implicit.evaluation

implicit.evaluation.AUC_at_k(model, train_user_items = user_artist_train_matrix, test_user_items = user_artist_test_matrix, K = 10,

#AUC score of 0.5210884671362229 ~ random === guessing
```

```
100%|██████████| 1330/1330 [00:00<00:00, 10472.59it/s]

0.5202834576915786
```

Using AUC for performance

How to improve performance?

<u>Grid Search</u>

CF (ALS) – Code Explanation

Grid Search

```
param_grid = {
    'factors' : [8, 10, 15],
    'regularization' : [0.15, 0.18, 0.2, 0.23, 0.27, 0.3],
    'iterations' : [60, 65, 75, 80, 85],
    'alphas' : [10]
}
```

Adjust all parameters to improve performance

```
#perform Grid Search
for factor in param_grid['factors']:
    for reg in param_grid['regularization']:
        for iter in param_grid['iterations']:
            for alp in param_grid['alphas']:
                #define model with given params
                #utilizing 4 cores on my CPU to speed up process
                model = implicit.als.AlternatingLeastSquares(factors = factor, regularization = reg, iterations = iter, alpha = alp, num_threads = 4)

                #model training
                model.fit(user_artist_train_matrix)

                #obtain AUC (eval)
                #k = 10 => model bases performance on recommendation for top 10 artists
                auc = implicit.evaluation.AUC_at_k(model, train_user_items = user_artist_train_matrix, test_user_items = user_artist_test_matrix, K = 10

                #compare AUC score and if auc is better, we update current stored bests
                if auc > best_AUC:
                    best_AUC = auc
                    best_params = {'factor' : factor, 'reg' : reg, 'iter' : iter, 'alpha' : alp}

print(f"Best hyperparams via grid search: {best_params}\nBest AUC: {best_AUC}")
#order for best_params is: factor, reg, iter, alpha
```

CF (ALS) – Performance Evaluation

```
Best hyperparams via grid search: {'factor': 8, 'reg': 0.2, 'iter': 75, 'alpha': 10}
Best AUC: 0.5472981589956554
```

~0.55 AUC score… Not good

Slightly better than randomly guessing

Take Aways from CF (ALS)

Fast

Avg run time on smaller dataset was < 5 mins per train and test

Easy Tuning

Grid Search allowed for highly varied values to be tried in sequence

Poor Prediction

Such a low AUC means it is not learning well. This would require many more iterations of Grid Search to find optimal solutions

Neural Collaborative Filtering

NCF – Overview

Step 1

User Matrix and Item Matrix are fed. Then performs Dot product to create input matrix

Step 2

Run through Hidden Layers

Step 3

Output Binary Classifier (0;1) for interactions

NCF – Code Explanation

Setup

```python
import tensorflow as tf

if tf.config.list_physical_devices('GPU'):
    print("GPU is available")
else:
    print("GPU is not available")

# List the available GPUs and their memory information
gpus = tf.config.get_visible_devices('GPU')
for gpu in gpus:
    memory_info = tf.config.experimental.get_memory_info('GPU:0')
    print(f"GPU: {gpu.name}")
    print(f"Current memory usage: {memory_info['current']} bytes")
    print(f"Peak memory usage: {memory_info['peak']} bytes")

#ensure we're able to use the GPU for processing the stuff
#I'm utilizing my home PC which has an NVIDIA RTX 3070
```

```
GPU is available
GPU: /physical_device:GPU:0
Current memory usage: 0 bytes
Peak memory usage: 0 bytes
```

Use GPU for faster processing

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Dot, Dense, Flatten, Dropout

#create NCF model structure

#input layers for user and item
user_input = Input(shape=(1,), name='user_input')
item_input = Input(shape=(1,), name='item_input')

#embedding layers to flatten the information from user-item interaction matrix
user_embedding = Embedding(input_dim=len(user_ids_flat), output_dim=5, name='user_embedding')(user_input)
item_embedding = Embedding(input_dim=len(item_ids_flat), output_dim=5, name='item_embedding')(item_input)

#flatten embedding layers for dot product
user_vec = Flatten()(user_embedding)
item_vec = Flatten()(item_embedding)

#dot product between user and item vectors (matrices)
dot_product = Dot(axes=1)([user_vec, item_vec])
```

Input → Embedding → Dot product

NCF – Code Explanation

Model Structure

```
#hidden layer with 2048 neurons
h1 = Dense(2048, activation = 'relu') (dot_product)

#add 20% dropout to reduce overfitting
h1 = Dropout(0.2)(h1)


#hidden layer with 1024 neurons
h2 = Dense(1024, activation = 'relu')(h1)

#add 20% dropout to reduce overfitting
h2 = Dropout(0.2)(h2)
```

```
#hidden layer with 512 neurons
h3 = Dense(512, activation = 'relu')(h2)

#add 20% dropout to reduce overfitting
h3 = Dropout(0.2)(h3)

#output with sigmoid activation function for binary classification (user likes artist)
output = Dense(1, activation='sigmoid')(h3)

#defining the model
ncf_model = Model(inputs=[user_input, item_input], outputs=output)
```

Dot Product → Hidden layers 1 → 2 → 3 → output

NCF – Code Explanation

Compile then Train model

```python
from tensorflow.keras import metrics

#model compilation
ncf_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', metrics.Precision(), metrics.Recall()])

#print summary of model too
ncf_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape         Param #     Connected to
==================================================================================================
 user_input (InputLayer)     [(None, 1)]          0           []

 item_input (InputLayer)     [(None, 1)]          0           []

 user_embedding (Embedding)  (None, 1, 5)         77220560    ['user_input[0][0]']

 item_embedding (Embedding)  (None, 1, 5)         77220560    ['item_input[0][0]']

 flatten (Flatten)           (None, 5)            0           ['user_embedding[0][0]']

 flatten_1 (Flatten)         (None, 5)            0           ['item_embedding[0][0]']

 dot (Dot)                   (None, 1)            0           ['flatten[0][0]',
                                                               'flatten_1[0][0]']

 dense (Dense)               (None, 2048)         4096        ['dot[0][0]']

 dropout (Dropout)           (None, 2048)         0           ['dense[0][0]']

 dense_1 (Dense)             (None, 1024)         2098176     ['dropout[0][0]']

 dropout_1 (Dropout)         (None, 1024)         0           ['dense_1[0][0]']

 dense_2 (Dense)             (None, 512)          524800      ['dropout_1[0][0]']

 dropout_2 (Dropout)         (None, 512)          0           ['dense_2[0][0]']

 dense_3 (Dense)             (None, 1)            513         ['dropout_2[0][0]']

==================================================================================================
```

Model summary

```python
#insert data as a single table into the model
user_item_data = [user_ids_flat, item_ids_flat]

#train the model on the combined data
ncf_model.fit(user_item_data, labels, epochs=5, batch_size=32768) # 65536 32768
```

```
Epoch 1/5
472/472 [==============================] - 59s 121ms/step - loss: 0.0228 - accuracy: 0.9972 - precision: 0.0016 - recall: 0.0010
Epoch 2/5
472/472 [==============================] - 58s 124ms/step - loss: 0.0122 - accuracy: 0.9983 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 3/5
472/472 [==============================] - 58s 123ms/step - loss: 0.0105 - accuracy: 0.9983 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 4/5
472/472 [==============================] - 58s 124ms/step - loss: 0.0098 - accuracy: 0.9983 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 5/5
472/472 [==============================] - 58s 123ms/step - loss: 0.0095 - accuracy: 0.9983 - precision: 0.0000e+00 - recall: 0.0000e+00
```

Model Training

Note: I printed Precision and Recall to get a better idea of performance

NCF – Performance Evaluation

```
#reshape the prediction so we can see the results
pred = pred.reshape(len(user_artist_test.index.to_numpy()),len(user_artist_test.columns.to_numpy()))
print(pred.shape)
print(pred)

#save to csv for viewing on machine
np.savetxt("test.csv", pred, delimiter=',')
```

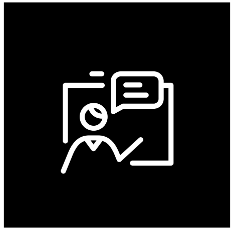
```
(1422, 4595)
[[6.41770894e-05 1.50387571e-03 1.14942246e-04 ... 3.81117308e-04
  6.41770894e-05 6.85204053e-03]
 [3.93285554e-05 4.01007431e-03 6.11971179e-03 ... 3.90591379e-03
  2.04896159e-03 2.10656915e-02]
 [3.27590038e-03 1.05575717e-03 2.87834046e-05 ... 7.01974437e-04
  1.40420525e-04 1.94723054e-03]
 ...
 [1.51610207e-02 3.65556101e-03 9.83655264e-05 ... 3.49230453e-04
  5.85206086e-04 4.67651896e-03]
 [4.98800888e-04 1.44600589e-02 9.85409133e-04 ... 1.17245328e-03
  2.76998297e-04 9.75499395e-03]
 [1.29936016e-04 8.16146086e-04 8.52018013e-04 ... 2.89623014e-04
  2.62918533e-04 8.34032334e-03]]
```

Very low probability for interactions (< 0.2)

Take Aways from NCF

Slower than CF

Despite me running many different models using both my CPU and GPU, NCF would run slower on average for training and testing



High PC Req’s

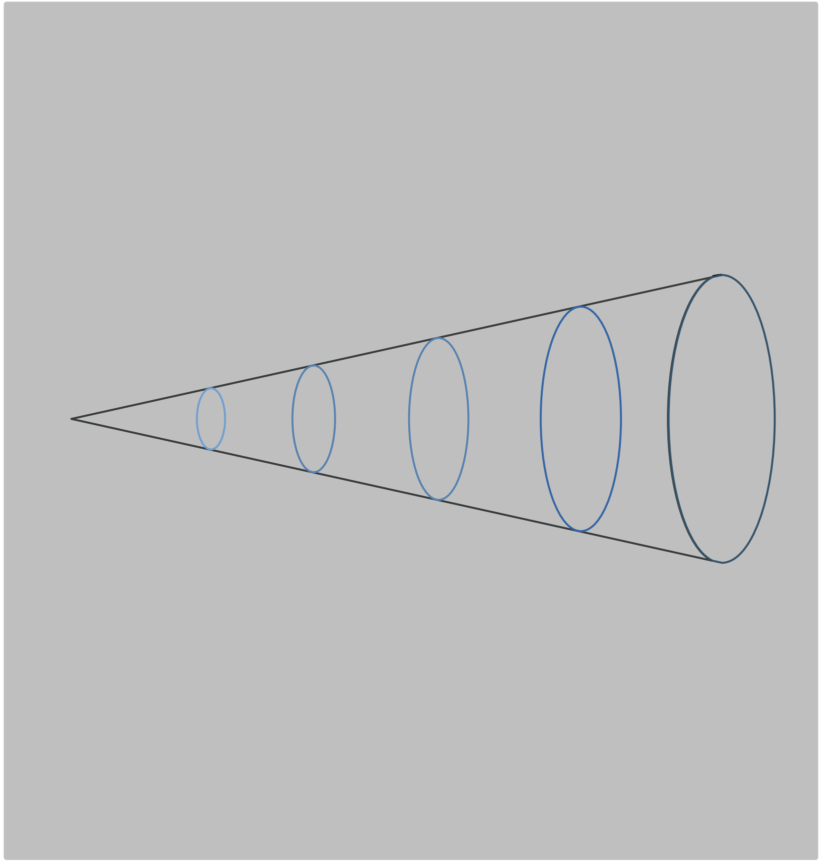
I frequently would run out of Memory (VRAM) when using the GPU. Using the CPU it would take many hours (>6 hours) per training session

Poor Learning

High Accuracy but Extremely low Precision and Recall indicates model is overfitting

03

Future Work

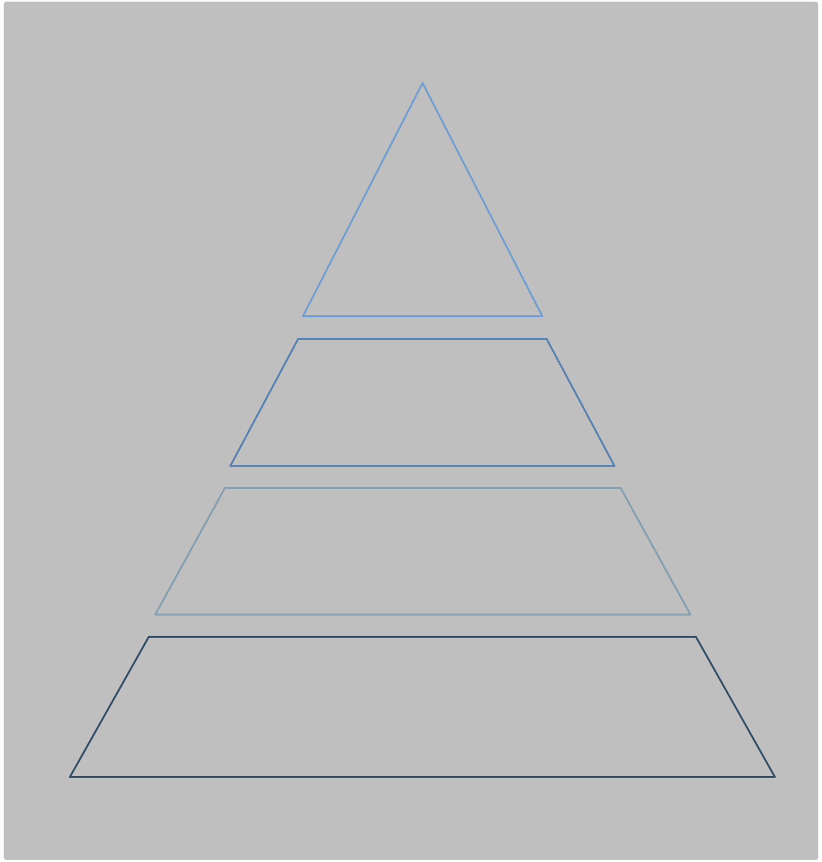


Future Work

- Utilize Cloud Services for DL models
- Obtain dataset with user ratings
- Pick alternative algorithms
- Utilize Hybrid Methods

04

Conclusion



Conclusion

This was a fun yet challenging project. My models' performance wasn't great, but this demonstrates the difficulty with building an implicit recommender system.

References

- Neural Collaborative Filtering (2017) He, Liao, Zhang, et. al.
- https://developers.google.com/machine-learning/recommendation/collaborative/basics

  https://towardsdatascience.com/alternating-least-square-for-implicit-dataset-with-code-8e7999277f4b

- https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

Questions?

Thank you for your time and attention

CREDITS: This presentation template was created by , including icons by , infographics & images by

Please keep this slide for attribution