Recommendation System

Applied Data Analysis - Final Project

Cnet ID: krishnaveni
UCID: 12374512

Problem Space: The Information Age

- Internet brings the Age of Information Explosion
- Increasing brand rivalry
- Increasing need to build customer loyalty
- Information Overload
- Over Choice
- Need for marketing strategies to increase sales and revenue

"Customers seek brands which align with their personal values and beliefs"

Solution and its Advancements

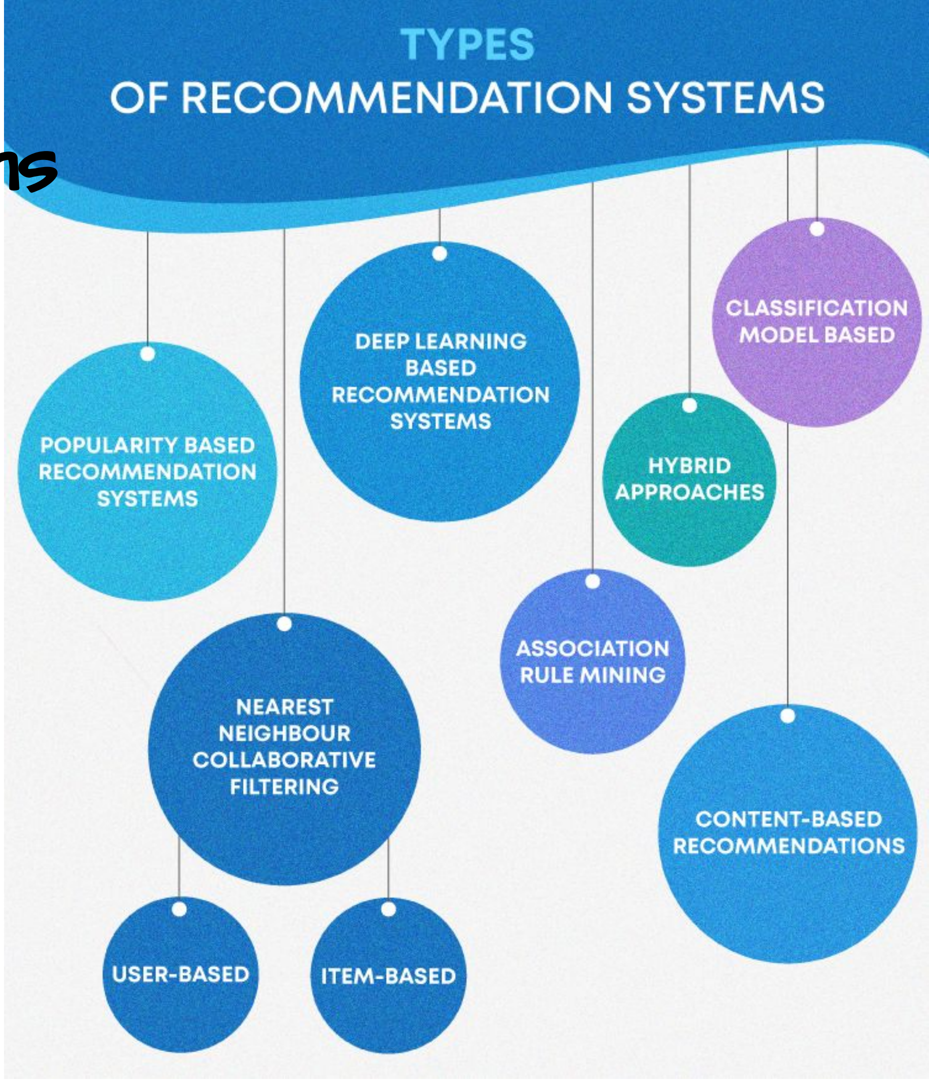
- Customized products
- Targeted Marketing
- Acknowledge customer behaviours and personalities
- Build solutions which are at the crux of Personality Psychology and Economics

"Better access to more product options has made brand loyalty less secure"

Recommendation Systems

Evolution of Models:

- Item Hierarchy
- Attribute Based
- Statistical Models
  - Matrix Factorization
  - Formulae Based
    - Model implemented:
      - Content Based System
- Model Based Recommendation Systems
  - Model implemented:
    - Collaborative Filtering

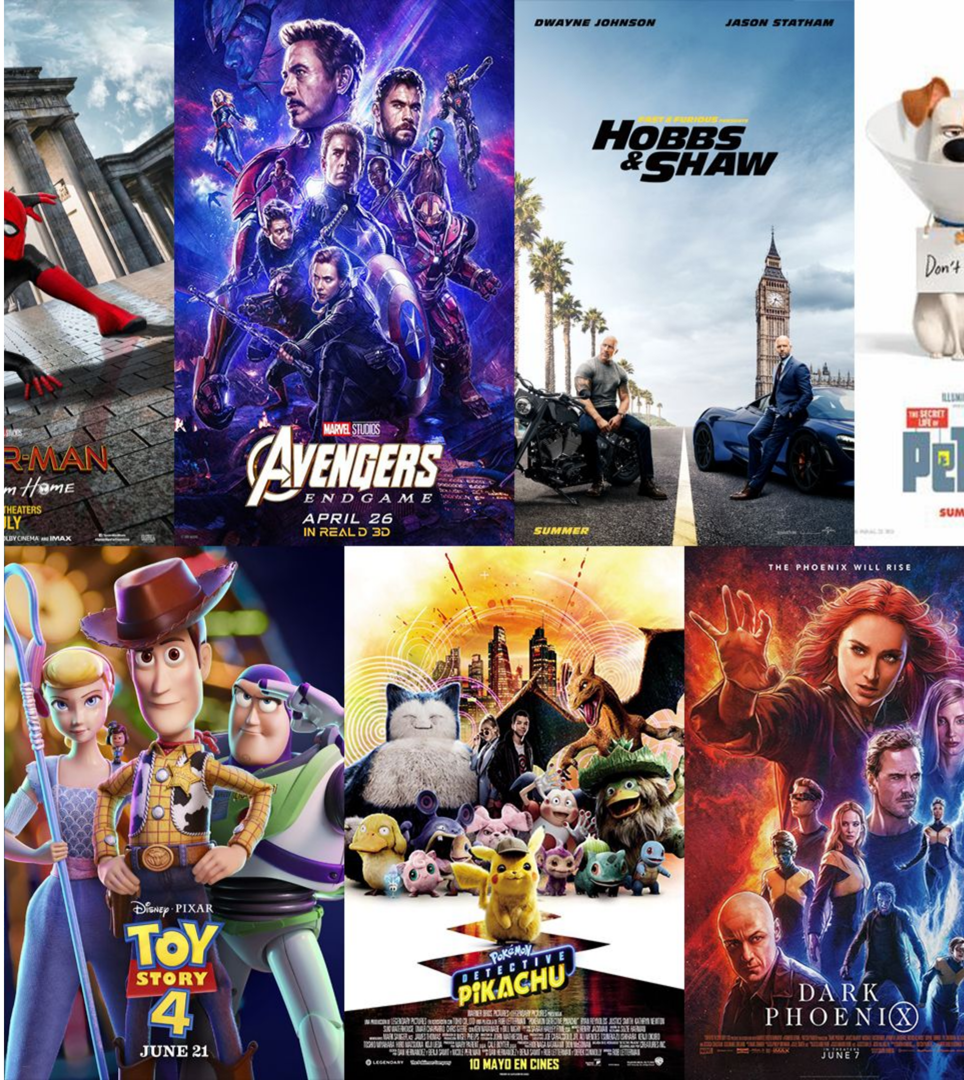


Implementation Details.....

- Dataset
  - Curse of Dimensionality
- Models
  - Content-Based Filtering
    - Model description
    - Performance
    - Challenges
  - Collaborative Filtering
    - Model description
    - Data Augmentation
    - Performance
    - Challenges
- Platform Support
- Resources & References

Dataset Details

- Movie Lens 20M dataset
- Data:
  - Movies
  - Users
  - Ratings
  - Timestamp
  - Genres
  - Year of Release
- Data Augmentation: The Why?
- Used: 7M data points
- Tested on: 14k users



```python
movie_ids = ratings['movieId'].unique()
user_test_movie_list = zip(test_ratings['userId'], test_ratings['movieId'])

# The set is used only for faster lookup. There are no duplicates here
user_test_movie_set = set(user_test_movie_list)

# Create 99 negative interaction points for each user to create the 100 count sample
for (u, i) in tqdm(user_test_movie_set):

    for _ in range(99):
        negative_item = np.random.choice(movie_ids)
        while ((u, negative_item) in user_test_movie_set) or ((u, negative_item) in user_movie_set):
            negative_item = np.random.choice(movie_ids)
        users.append(u)
        items.append(negative_item)
```

```
100%|██████████| 14315/14315 [00:23<00:00, 600.14it/s]
```

Content Based Recommendation System

- Item-Item similarity
  - Thematically linked items
  - Meta-data of the items
- Cosine similarity
- Use of rating to measure relevance
- Statistical Model
- Results:

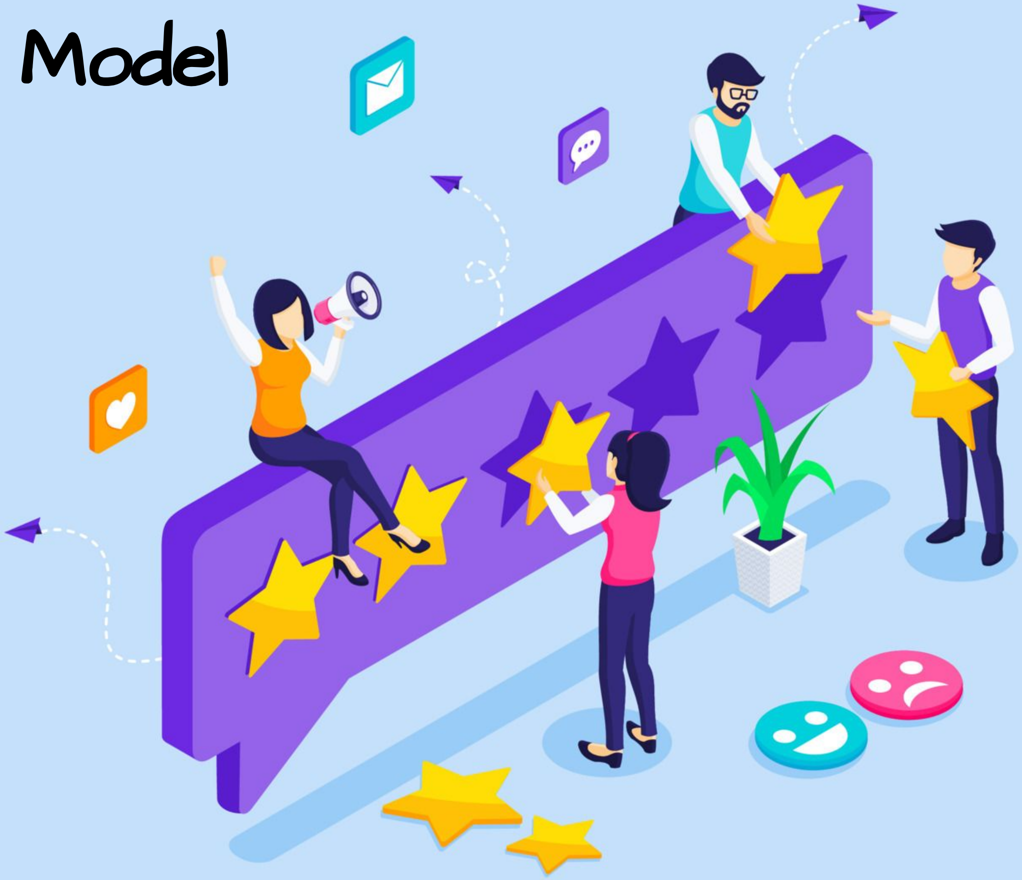
```python
# I have built Content-based filtering as a purely mathematical model using similarity of its genres
def get_recommendations(movie_title, n=20):
  movie_id = content[content['title'] == movie_title].index[0]
  movie_of_interest = movie_genres.loc[movie_id]

  result = movie_genres.dot(movie_of_interest)

  recommendations_index = result.sort_values(ascending=False)[:n].index
  recommendations = content.loc[recommendations_index]
  return recommendations
```

```python
# Getting results/recommendations for `Toy Story`
result = get_recommendations('Toy Story')
print(result['title'])
```

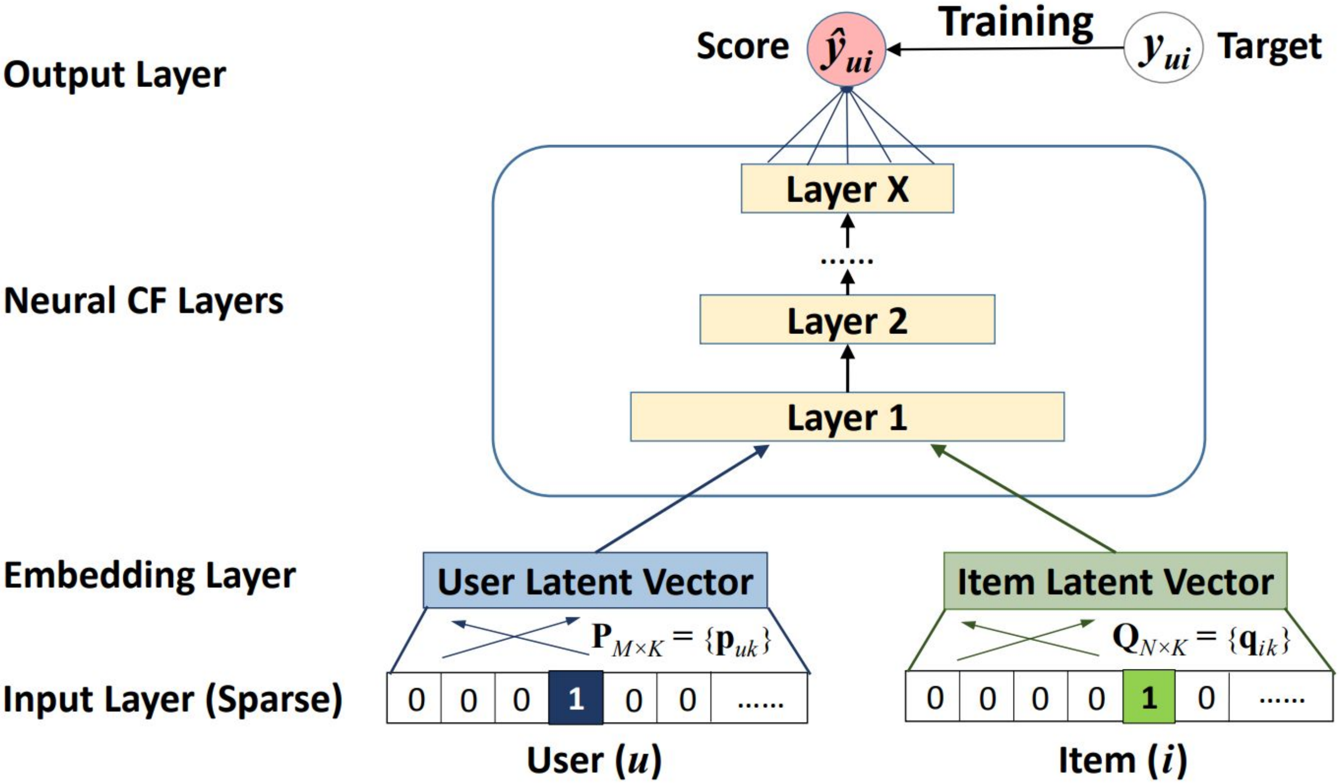
```
80158                        Cartoon All-Stars to the Rescue
131248                                        Brother Bear 2
78499                                            Toy Story 3
1                                                  Toy Story
26340                           Twelve Tasks of Asterix, The
4886                                          Monsters, Inc.
3114                                             Toy Story 2
108932                                        The Lego Movie
4306                                                   Shrek
4016                              Emperor's New Groove, The
2987                              Who Framed Roger Rabbit?
56152                                              Enchanted
114552                                        Boxtrolls, The
114240                                               Aladdin
33463     DuckTales: The Movie - Treasure of the Lost Lamp
115875                   Toy Story Toons: Hawaiian Vacation
```

Collaborative Filtering Model

- Based on user-item interactions
  - Ratings
- Neural Collaborative Filtering model
  - Implicit Feedback
  - Tower structure
  - Activation: Relu
  - Optimizer: Adam
  - Loss function: Log Loss
  - Improvements: Diversity regularizer
  - Metric: Hit Ratio@10
    - Result:



Neural Collaborative Filtering



```python
# PyTorch Lightning is an open-source Python library that provides a high-level interface for PyTorch
class CollaborativeFiltering(pl.LightningModule):
  def __init__(self, train_ratings, dataloader):
    super().__init__()
    self.train_ratings = train_ratings
    self.dataloader = dataloader

    # Tried with len() first, it fails when the IDs are not in order or exceed length
    # This is because embedding is just a lookup table we are building for n items
    self.number_of_users = train_ratings['userId'].max() + 1
    self.number_of_items = train_ratings['movieId'].max() + 1

    # Longer embedding vectors don't add more valuable information and smaller ones don't represent the semantics well enough
    # The rule of thumb for determining the embedding size is the cardinality size divided by 2, but no bigger than 50
    # I have chosen 16 here, as the cardinality is too huge
    self.user_embedding = nn.Embedding(num_embeddings=self.number_of_users, embedding_dim=16)
    self.item_embedding = nn.Embedding(num_embeddings=self.number_of_items, embedding_dim=16)

    # Tower pattern is implemented, where the bottom layer is the widest and each successive layer has a smaller number of neurons
    # The reference paper halves the neurons by half each time, but I have tried a more generalized model
    self.layer1 = nn.Linear(in_features=32, out_features=64)
    self.layer2 = nn.Linear(in_features=64, out_features=32)
    self.layer3 = nn.Linear(in_features=32, out_features=16)
    self.layer4 = nn.Linear(in_features=16, out_features=8)

    # Reference: https://stats.stackexchange.com/questions/207049/neural-network-for-binary-classification-use-1-or-2-output-neurons
    self.output_layer = nn.Linear(in_features=8, out_features=1)
```

```python
def forward(self, user_input, item_input):
  dense_user = self.user_embedding(user_input)
  dense_item = self.item_embedding(item_input)
  vector = torch.cat([dense_user, dense_item], dim=-1)

  # Results from various posts and research papers
  # The sigmoid function restricts each neuron to be in (0,1), which may limit the model's performance; and it is known to suffer from saturation, where neurons stop le
  # Even though tanh is a better choice and has been widely adopted it only alleviates the issues of sigmoid to a certain extent, since it can be seen as a rescaled ver
  # ReLU, which is more plausible and proven to be non-saturated, it encourages sparse activations, making the model less likely to be overfitting.
  vector = nn.ReLU()(self.layer1(vector))
  vector = nn.ReLU()(self.layer2(vector))
  vector = nn.ReLU()(self.layer3(vector))

  # sigmoid is the same as softmax. The better choice for the binary classification is to use one output unit with sigmoid instead of softmax with two output units, bec
  pred = nn.Sigmoid()(self.output_layer(vector))

  return pred
```

```python
trainer.fit(model)
trainer.save_checkpoint('/content/drive/MyDrive/small_dataset/checkpoint_3layer_regularizer.ckpt')
```

```
INFO:pytorch_lightning.utilities.rank_zero:You are using a CUDA device ('NVIDIA A100-SXM4-40GB') that has Tensor Cores. To properly utilize them, you should set `torch.set_fl
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name           | Type      | Params
---------------------------------------------
0 | user_embedding | Embedding | 1.1 M
1 | item_embedding | Embedding | 1.1 M
2 | layer1         | Linear    | 544
3 | layer2         | Linear    | 528
4 | layer3         | Linear    | 136
5 | output_layer   | Linear    | 9
---------------------------------------------
2.2 M     Trainable params
0         Non-trainable params
2.2 M     Total params
8.636     Total estimated model params size (MB)
/usr/local/lib/python3.8/dist-packages/pytorch_lightning/trainer/connectors/data_connector.py:224: PossibleUserWarning: The dataloader, train_dataloader, does not have many wo
  rank_zero_warn(
Epoch 19: 100% 3496/3496 [02:01<00:00, 28.85it/s, loss=0.0092, v_num=16]
```

```python
def diversity_loss(self, y_true, y_pred, movie_ids):
  # Adding the diversity loss as a regularizer to the log loss function
  # This has been added to enhance diversity of the model predictions
  alpha = 10**-3
  movie_ids_list = np.squeeze(movie_ids).tolist()
  indexes = np.argsort(np.squeeze(y_pred).tolist())[::-1][:10]
  positives = [movie_ids_list[index] for index in indexes]
  batch_grid = self.genre_grid.loc[positives]
  similarity = batch_grid.corr()
  diversity_regularizer = (similarity.sum()).sum()

  # Alpha here is multiplied to soften impact of the size of loss
  return alpha * diversity_regularizer

def training_step(self, batch, batch_idx):
  user_input, item_input, labels = batch
  predicted_labels = self(user_input, item_input)

  # Binary Cross-Entropy/Log Loss
  bce_loss_obj = nn.BCELoss()
  loss = bce_loss_obj(predicted_labels, labels.view(-1, 1).float())
  # Adding similarity as diversity regularizer
  diversity_regularizer = self.diversity_loss(labels.view(-1, 1).float(), predicted_labels, item_input)
  return loss + diversity_regularizer
```

```
test_user_item_set = set(zip(test_ratings['userId'], test_ratings['movieId']))
test_dataset = pd.read_csv('/content/drive/MyDrive/small_dataset/augmented_test_dataset.csv')

hits = []
user_ids = test_dataset['userId'].unique()
for user_id in tqdm(user_ids):
    test_item = test_ratings[test_ratings['userId']==user_id]['movieId'].iloc[0]
    user_df = test_dataset[test_dataset['userId'] == user_id].reset_index()
    data_loader = DataLoader(TestingData(user_df), batch_size=100, num_workers=4, shuffle=False)

    # Returns a list of dictionaries, one for each provided dataloader containing their respective predictions
    predictions = model(torch.tensor(user_df['userId']), torch.tensor(user_df['movieId']))
    # To convert to numpy array and solve issue: Can't call numpy() on Tensor that requires grad. Use tensor.detach().numpy() instead
    predictions = predictions.detach().numpy()

    # To solve : Buffer has wrong number of dimensions (expected 1, got 2) because dimensions of predictions are (100, 1)
    # Reference: https://deeplizard.com/learn/video/fCVuiW9AFzY
    predictions = np.squeeze(predictions)

    # Since we need the movieId,
    top_10 = set(user_df.iloc[np.argsort(predictions)[::-1][:10]]['movieId'])

    hits.append(1) if test_item in top_10 else hits.append(0)

print(f'Hit Ratio @ 10 is {np.average(hits)}')
```

```
100%|██████████| 14315/14315 [01:10<00:00, 202.85it/s]Hit Ratio @ 10 is 0.593223891023402
```

Results…

- Without Regularization
  - 3 hidden layers: 0.524
  - 4 hidden layers: 0.593
  - Benchmark from Research Paper: 0.67
- With Regularization
  - 3 hidden layers with regularization: 0.57
- Improvements:
  - More Data
  - Resources
    - Compute
    - Memory
    - Time Taken for each epoch
  - Optimization of Custom loss

Challenges and Future Scope

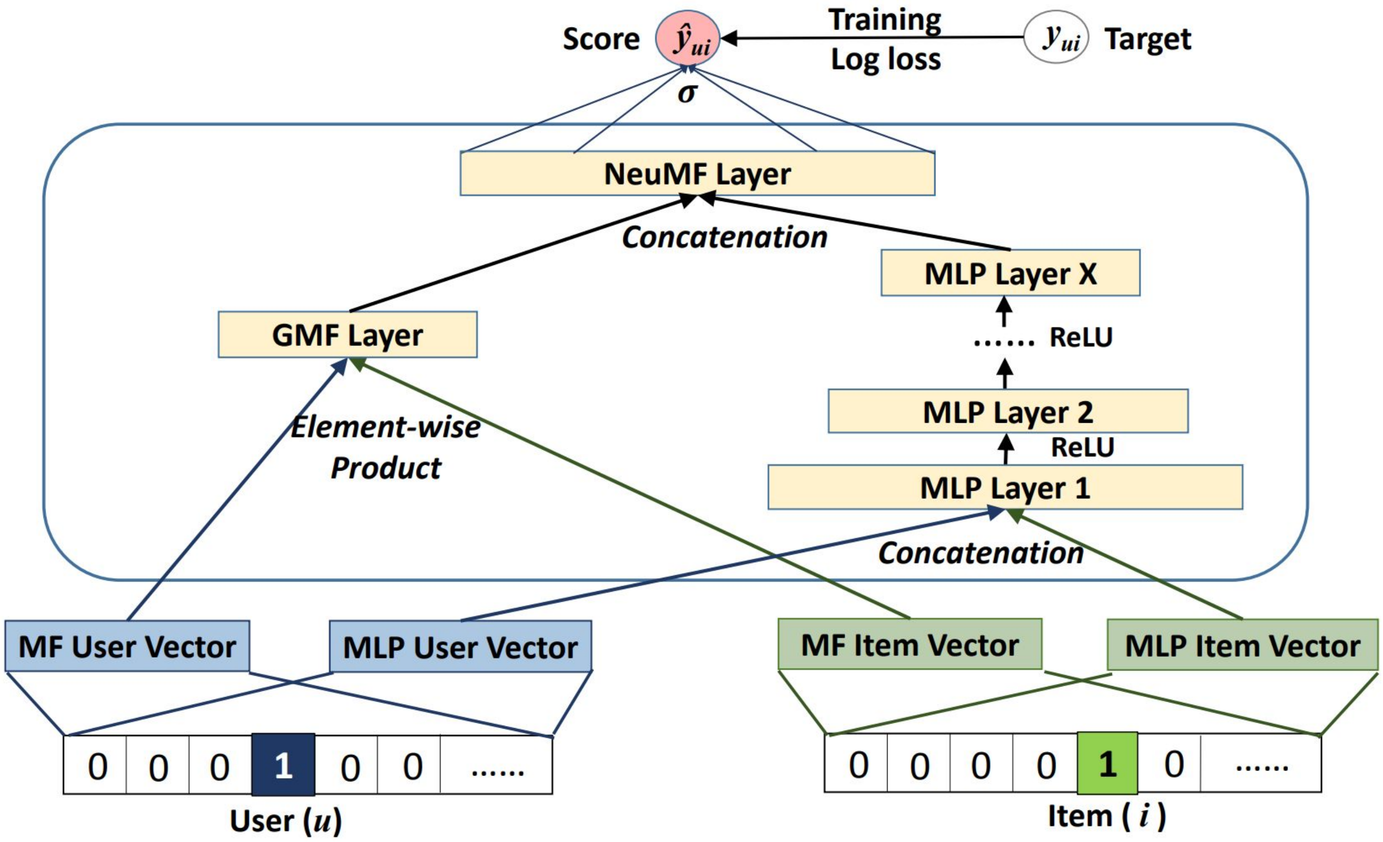
Challenges

- Changing user preferences
- Choosing an architecture
- Curse of Dimensionality
- Platform availability
- Tuning of Hyperparameters
- Space and Time Constraints

Future Scope

- Deep Learning model for Content-Based Filtering
- Tune hyper parameters
- Improve diversity regularization
- Implement the NeuMF model
- Session-based recommendations

Neural Matrix Factorization



Learning Journey...

- Understanding of Recommendation Systems: The secret sauce
- Know-how of various advancements
- Deep learning fundamentals
- Data Augmentation
- Model development
- Pytorch and Pytorch-lightning
- Trade-offs between activation functions and optimizers
- Troubleshooting
- Experimentation

THANK YOU!