Infinite Possibilities in User Review: A Recommendation System based on Sentimental Analysis and Collaborative Filtering for STEAM Gamers

Group 11

Yingxuan Bian, Yu Lu, Yuqi Wang, Junzhe Tang, Jiawei Zhang, Jingheng Zhang

1. Introduction

Understanding the needs and expectations of online customers is crucial in today's consumer-oriented e-commerce market. Recommendation systems, prevalent on the internet, offer personalized suggestions to users (Ji et al., 2003). A notable technique in this domain is Collaborative Filtering (Lee et al., 2005), grounded in the premise that users with akin tastes exhibit similar preferences for products or items.

In the context of Collaborative Filtering (CF), the algorithm predicts a user's potential preferences by considering the opinions of other users. This method is widely employed by online shopping, news, and social networking sites to recommend movies, books, articles, or other items to their users. The literature distinguishes two primary CF algorithms – user-based and item-based CF (Kumar & Fan, 2015). User-based CF algorithms identify users with akin preference patterns, particularly in item ratings, and suggest items highly rated by these similar users to the target user of concern.

To enhance User-Based Collaborative Filtering, we specifically chose the Steam dataset for analysis. Steam, as the largest online cross-platform game distribution system, boasts 120 million monthly active users, with 62.6 million engaging daily. Its extensive catalog comprises over 50,000 games, rendering it an intriguing subject for market research.

In this report, we aim to tackle the following research questions: First, how to predict the future degree of satisfaction of Steam users on different games based on their comments? Second, how to recommend games to potential customers by sentimental analysis and neural collaborative filtering?

2. Literature Review

Various recommendation systems operate on different foundations.

One approach is a Novel Self-Attention-Based Sequential Model proposed by Kang and McAuley (2018). Diverging from conventional methods like Markov Chains (MCs) and Recurrent Neural Networks (RNNs), SASRec strives to strike a balance between capturing long-term semantics and making predictions grounded in relatively recent actions. Notably, SASRec exhibits a significantly faster performance compared to analogous models based on Convolutional Neural Networks (CNNs) or RNNs. However, a notable limitation lies in the challenge of extending the model to incorporate personal information such as tags, prices, and reviews.

Continuing, Cheuque et al. (2019) devised tailored methods for the Steam platform. In their study, they explored the efficacy of Factorization Machines (FM), deep neural networks (DeepNN), and a hybrid model that combines both (DeepFM). These models were specifically tested for their capability to manage multiple inputs and diverse variable types. The research leveraged a dataset from the Steam platform, amalgamating user purchase details, playtime, social interactions (critics), and game characteristics. The study concludes by suggesting future research directions, encompassing parameter analysis, user studies, and the utilization of updated datasets for more comprehensive text analysis.

To address the research gap, we aim to broaden the scope by incorporating more diverse data and reviews. This expansion is essential because, firstly, when conducting sentiment analysis, it is crucial to account for not only the customer's historical purchase behavior but also factors like their preferences, purchasing power, and specific item preferences. Additionally, relying solely on direct ratings may be unreliable due to susceptibility to manipulations. In contrast, reviews and recommendations carry value as credible indicators, as they provide a more nuanced and authentic reflection of user opinions.

3. Methodology

3.1 Dataset Description

We obtained data on Steam Video Games from UCSD's Personalization Datasets. To be specific, there are 2 datasets. One is users' feedback on video games, which contains reviews, whether to recommend, the percentage of people that think the review helpful, etc. Another is games' information like gernes, tags, titles and so on. Totally, there are 25799 users and 3682 items. See Appendix 1 to find the example of dataset information. For dataset example see Apendix 1.

3.2 Sentiment Analysis

The first task is to extract users' attitudes towards video games they gave commends to. Here sentimental analysis is conducted. In addition, a Transformer model called BERT, which can determine the sentiment behind a piece of text about whether it's positive or negative, is chosen. The specific version is called DistilBERT, a smaller but faster version of BERT.

The input is the users' reviews. The output is a score of sentimental attitudes from -1 to 1. See the result for examples of sentiment analysis.

3.3 Factorization Machine Model

The second task is to predict users' ratings of other items and offer them reliable recommendations. Based on the features in our dataset and results of sentimental analysis, we choose Factorization Machine, one of the common models in Collaborative Filtering, which is helpful for understanding and capturing the interactions between data and usually applied in recommendation systems. The methodology following is inspired by "Factorization Machine" written by Steffen Rendle (2010).

There is significant difference between Factorization Machines and Matrix Factorization. Factorization Machines is a supervised learning algorithm that can capture interactions between any features in large, sparse datasets, and used for tasks like classification and recommendation. They model both linear and non-linear feature interactions. While Matrix Factorization, typically an unsupervised technique, decomposes a matrix into latent factors, mainly for dimensionality reduction and collaborative filtering in recommendation systems. Factorization Machines are more flexible and can handle various data types, while Matrix Factorization focus on uncovering hidden structures within matrix data, such as user-item interactions.

Ordinary linear model equation: $y = w_0 + \sum_{i=1}^{n} w_i x_i$, where w_0 is the initial weight, or it can be understood as a bias term, and w_i is the weight corresponding to each feature x_i.

From the above equation, it is easy to see that the ordinary linear model does not consider the correlation between features at all. To describe the correlation between features, we use a polynomial model.

The hypothesis function for a factorization machine of degree d=2 is defined as:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < v_i, v_j > x_i x_j$$

$\hat{y}(x)$ is the predicted sentimental analysis scores for the user on all the items.
w_0 is the global bias.
w_i is the weight of the i-th variable, which is the parameter we want to obtain.
x is the input vector of one user.
x_i is the value of the i-th variable in the input vector x. $x_i = 1$ indicates that the user has a review on item i, and we have obtained its sentimental analysis result. $x_i = 0$ indicates that the user has no review on item i.
v_i and v_j are the factor vectors of the i-th and j-th variable.
$< v_i, v_j >$ is the dot product of the factor vectors. It can also be regarded as the inner product of the two vectors in dimension k, representing the interaction between variables i and j. $< v_i, v_j > = \sum_{f=1}^{k} v_i, f \; v_j, f$, where v_i, f means the fth element of the implicit vector v_i.

Based on the proof made by Steffen Rendle (2010), below is the gradient of the hypothesis function:

$$\frac{\partial}{\partial\theta}\hat{y}(x) = \begin{cases} 1, & if\ \theta\ is\ w_0 \\ x_i, & if\ \theta\ is\ w_i \\ x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2, & if\ \theta\ is\ v_{i,f} \end{cases}$$

We use the Python library LightFM to develop our Factorization Machine Model. We select 3000 sentimental analysis scores for 1214 users and 894 items. We treat sentimental analysis scores as the user-item interactions, split the training set and testing set with the ratio of 1:4, and preprocess them into 4 sparse matrices.

The first one is X_{train}, a matrix with 1214 rows and 894 columns, where x_{ij} =1 if there is a sentimental analysis score of the user i on the item j according to the training set, and x_{ij} =0 if there is no sentimental analysis score of the user i on the item j. Here is the format of X_{train}:

Sparse Matrix X_{train}				
	Item 1	Item 2	Item 3	
User 1	1	0	0	…
User 2	1	0	0	…
User 3	1	0	0	…
User 4	0	1	0	…
User 5	0	1	0	…
User 6	0	0	1	…
User 7	0	0	1	…

The second one is Y_{train}, a matrix with 1214 rows and 894 columns, where y_{ij} = sentimental analysis score if there is a sentimental analysis score of the user i on the item j according to the training set, and y_{ij}=0 if there is no sentimental analysis score of the user i on the item j. Here is the format of Y_{train}:

Sparse Matrix Y_{train}				
	Item 1	Item 2	Item 3	
User 1	0.5969	0	0	…
User 2	0.4352	0	0	…
User 3	-0.3692	0	0	…
User 4	0	0.7932	0	…
User 5	0	-0.2363	0	…
User 6	0	0	0.6192	…
User 7	0	0	0.1354	…

The third and fourth sparse matrices are built based on the testing set, the preprocessing ideas are the same as previous ones.

We choose the Weighted Approximate-Rank Pairwise Loss, a common loss applied in learning to rank model, to supervise the training, and set the epoch as 30.

3.4 Deep Neural Network

Though Steffen Rendle (2010) used one-hot feature in movie recommendations, we still decided to use float instead of one-hot feature. The reason for not employing one-hot feature is that based on our dataset, the non-numeric feature includes "posted time", "last edited"," item/user id", "whether others consider helpful". Intuitively, encoding these features into one-hot does not logically make sense because factors like "id" or "posted time" may not reflect the preference characteristics of the user. In another word, these features can be perceived to be redundant, which may result in lower accuracy when predicting the rating score.

We want to further improve the accuracy of our recommendation system. So, we employ Deep Neural Network. The Deep Neural Network allows it to model non-linear relationships between features and capture more complex, high-order feature interactions, which can generate more accurate results.

The employed neural network architecture consists of several layers, which can map matrices to scores. The input vector is a matrix of 3681×1 with every entry representing the review of one user to other games. It is initially processed by the first fully connected layer, containing 32 hidden units with the ReLU activation function. There are 3759 matrices like this in total. The output of this layer is then normalized via a Batch Normalization layer.

Subsequently, the normalized output is fed into the second fully connected layer, comprising 16 hidden units with the ReLU activation function. Finally, the output of the second fully connected layer is passed through a third fully connected layer with only 1 hidden unit, without any activation function. The output size is 1, which means predicated score of users' preference.

The model is trained with an initial learning rate of 0.01, utilizing the Root Mean Squared Error (RMSE) as the loss function and the Mean Absolute Error (MAE) as the performance metric.

4. Result & Conclusion

4.1 Numerical Results

The partial results of sentimental analysis are listed below.

Reviews	**Label**	**Score**
So much fun! All the classes, weapons all suit that cl ass and they are also quite well balanced. Much fun to play!	POSITIVE	0.999879
This game is amazing, and it still is only in early acc ess but yes it does get bit dull after a while but hey if we help support the game and give feedback then to gether, we can build this game from a shitty house to a villa in Italy	NEGATIVE	- 0.873343

Our Factorization Machine model's training RMSE is 7.9564 and testing RMSE is 2.6992, which are relatively low. Its training AUC is 0.6058, which is larger than 0.5. And we used it to make recommendation for users, we try to recommend top 10 items for each users according to the predicted ratings. Here are part of our results.

User id: sergarino		
Rank	recommended_item	score
1	437701	1.027002
2	730	0.969984
3	220240	0.959285
4	299360	0.893786
5	292030	0.858746
6	346330	0.8561
7	440	0.824873
8	273350	0.79499
9	221640	0.788892
10	227300	0.782004

User id:sad-commie		
Rank	recommended_item	score
1	730	6.05161
2	212680	1.50385
3	220240	1.218665
4	304930	1.039608
5	212070	0.858863
6	440	0.796803
7	264200	0.764298
8	346330	0.684524
9	4700	0.668468
10	264140	0.547908

Another methodology we used was Deep Neural Network. The general logic of this model is looking for suitable users (i.e. users with highest predicted preference and rating) for each game. Take the game with item_id 440 as example. We train the model with this game because there are significantly large number of users. By ranking the score for the game from high to low, we select top 10% users' score from 22,000 users' rating. Managers can recommend this game to these users with high score because they are more likely to purchase the game. Part of the score are as below.

Rank	**user_id**	**score**
1	1604	0.9927146
2	12269	0.9895033
3	2803	0.98801476
4	9405	0.96504986
5	15454	0.9611386
……	……	……
2196	16554	0.66067237
2197	4719	0.66064554
2198	5774	0.66063327
2199	18344	0.6606153
2200	5981	0.6605062

RMSE for this model is 0.1118, which is much better than RMSE of Factorization Machines Model which is 2.6992.

4.2 Managerial Results

From the numerical results and the modelling process, we also find some managerial results. First of all, when collecting user feedback and establishing the recommendation system, manager should not only check the score and click, but also pay attention to the reviews, because the reviews show the attitude of users in a more natural way.

Secondly, it is suggested to collect not only user feedback, but also feedback from other users (i.e. whether other users find this comment helpful), which can provide information about the quality of the feedback and figure out inauthentic or exaggerated ones.

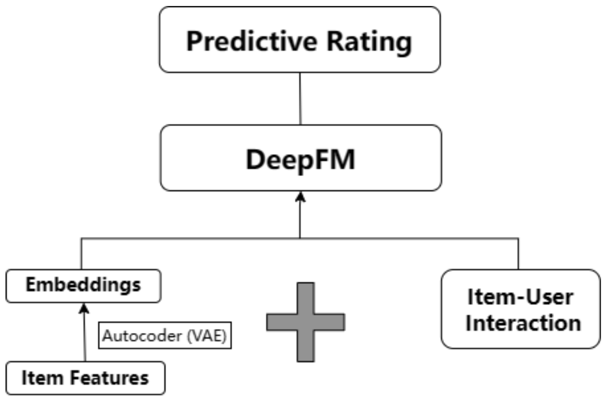
Last but not least, the measurement of sentiment analysis on user reviews can be good choice for marketing managers, which can better capture customer characteristics and design, enhancing the effectiveness of campaigns and marketing.

4.3 Discussion and Future Work

In conclusion, by using Sentimental Analysis, Factorization Machine (FM), and Deep Neural Network (DNN), our project successfully predicted degree of satisfaction of Steam users based on their past comments and recommended preferred games to them. The high accuracy of sentimental analysis and matching between users and games could help Steam increase potential revenue. What's more, these methodologies and framework can be expanded to other entertainment platform or service.

However, it is undeniable that there are some limitations. Firstly, there is a cold start problem in the Recommendation System we built, which means it is only applicable to old users and old items, instead of new users and new items. Secondly, we are unsure about authenticity of reviews.

In future work, we have two aspects of suggestions. Firstly, for the cold start problem, consider hybrid method which combines content-based methods with DeepFM. It is suggested to include items' information like genres, tags and descriptions in our training data. An Autoencoder such as VAE can be used to extract embeddings for items and provide a more efficient representation of games, it can also be useful for both existing and new games. Then by combining item embedding and result of sentimental analysis together as input data for DeepFM, the predicted ratings can be calculated. This model will solve the cold start problem for new items. Also, it is suggested to collect initial preferences when new users open the game and input those data in the model.



Second, regarding authenticity of reviews, we could expand the scope of text analysis, not limiting it to sentiment analysis in reviews.

References

Cheuque, G., Guzmán, J. L., & Parra, D. (2019). Recommender Systems for Online Video Game Platforms: the Case of STEAM. *Companion Proceedings of the 2019 World Wide Web Conference*. https://doi.org/10.1145/3308560.3316457.

Gallagher, S., & Park, S. H. (2002). Innovation and competition in standard-based industries: a historical analysis of the US home video game market. *IEEE Transactions on Engineering Management, 49*(1), 67–82. https://doi.org/10.1109/17.985749.

Ji, J., Sha, Z., Liu, C., & Zhong, N. (2003). Online recommendation based on customer shopping model in e-commerce. *Proceedings IEEE/WIC International Conference on Web Intelligence*, 68–74. https://doi.org/10.1109/WI.2003.1241175.

Kang, W., & McAuley, J. (2018). Self-Attentive Sequential Recommendation. 2018 *IEEE International Conference on Data Mining (ICDM)*, 197–206. https://doi.org/10.1109/icdm.2018.00035.

Kumar, N., & Fan, Z. (2015). Hybrid User-Item based Collaborative Filtering. *Procedia Computer Science, 60*, 1453–1461. https://doi.org/10.1016/j.procs.2015.08.222.

Lee, J., Jun, C., Lee, J., & Kim, S. (2005). Classification-based collaborative filtering using market basket data. *Expert Systems With Applications, 29*(3), 700–704. https://doi.org/10.1016/j.eswa.2005.04.037.

Nguyen, N., Johnson, J., & Tsiros, M. (2023). Unlimited Testing: Let's Test Your Emails with AI. *Marketing Science*. https://doi.org/10.1287/mksc.2021.0126.

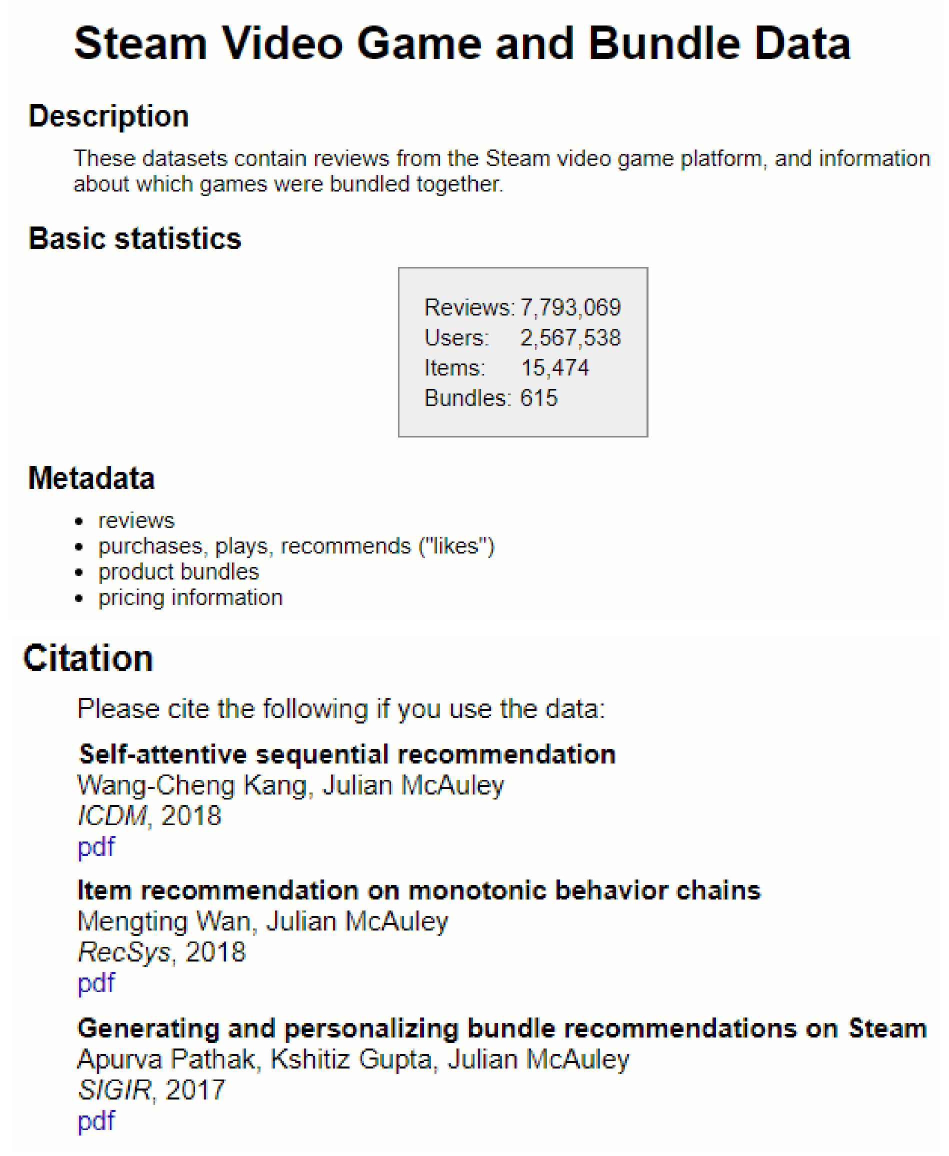
Rendel, Steffen. (2010). Factorization Machines. The Institute of Scientific and Industrial Research. Osaka University, Japan. https://sdcast.ksdaemon.ru/wp-content/uploads/2020/02/Rendle2010FM.pdf

Rosa, R. L., Schwartz, G. M., Ruggiero, W. V., & Rodríguez, D. Z. (2019). A Knowledge-Based recommendation system that includes sentiment analysis and deep learning. *IEEE Transactions on Industrial Informatics, 15*(4), 2124–2135. https://doi.org/10.1109/tii.2018.2867174. https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data

Appendix

Appendix 1. Dataset source

https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data



Steam Video Game and Bundle Data

Description

These datasets contain reviews from the Steam video game platform, and information about which games were bundled together.

Basic statistics

Reviews:	7,793,069
Users:	2,567,538
Items:	15,474
Bundles:	615

Metadata

- reviews
- purchases, plays, recommends ("likes")
- product bundles
- pricing information

Citation

Please cite the following if you use the data:

Self-attentive sequential recommendation
Wang-Cheng Kang, Julian McAuley
ICDM, 2018
pdf

Item recommendation on monotonic behavior chains
Mengting Wan, Julian McAuley
RecSys, 2018
pdf

Generating and personalizing bundle recommendations on Steam
Apurva Pathak, Kshitiz Gupta, Julian McAuley
SIGIR, 2017
pdf

Sample Data:

Review Data

user_id	de_Butl3r
posted	October 13, 2014
helpful	0.38 (3 of 8 people found this review helpful)
recommend	True
Item_id	209870
reviews	It's a very fun game i recommend as its nearly like......

User and Item data

develper	Kotoshiro
genres	['Action', 'Casual', 'Indie', 'Simulation', 'Strategy'],
item_id	761140
title	'Lost Summoner Kitty'
release_data	'2018-01-04'
tags	['Strategy', 'Action', 'Indie', 'Casual', 'Simulation']
price	4.99
discount_price	4.49
early_access	False

Appendix 2: The deducing process of FM

To find ω_{ij}, we introduce the auxiliary vector $V_i=(v_{i1},v_{i2},...,v_{ik})$ for each x_i. Then, $v_iv_j^T$ is used to solve ω_{ij}.

$$\mathbf{V} = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1k} \\ v_{21} & v_{22} & \cdots & v_{2k} \\ \vdots & \vdots & & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nk} \end{pmatrix}_{n\times k} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{pmatrix}$$

Then the matrix composed of ω_{ij} can be expressed as:

$$\hat{\mathbf{W}} = \mathbf{V}\mathbf{V}^T = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T & \mathbf{v}_2^T & \cdots & \mathbf{v}_n^T \end{pmatrix}$$

Due to the factorization of the pairwise interactions, there is no model parameter that directly depends on two variables (e.g. a parameter with an index (i, j)). So the pairwise interactions can be reformulated:

$$\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=i+1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle\, x_i\, x_j \\
=&\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_j\rangle\, x_i\, x_j - \frac{1}{2}\sum_{i=1}^{n}\langle \mathbf{v}_i,\mathbf{v}_i\rangle\, x_i\, x_i \\
=&\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{j,f}\, x_i\, x_j - \sum_{i=1}^{n}\sum_{f=1}^{k} v_{i,f}\, v_{i,f}\, x_i\, x_i\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)\left(\sum_{j=1}^{n} v_{j,f}\, x_j\right) - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right) \\
=&\frac{1}{2}\sum_{f=1}^{k}\left(\left(\sum_{i=1}^{n} v_{i,f}\, x_i\right)^2 - \sum_{i=1}^{n} v_{i,f}^2\, x_i^2\right)
\end{aligned}$$

The original function can be rewritten as:

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}((\sum_{i=1}^{n} v_{i,f}x_i)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2)$$

For FM model, we can use gradient descent algorithm to optimize, since we want to use gradient descent, then we need to write out the partial derivatives of all parameters in the model.

As we have shown, FMs have a closed model equation that can be computed in linear time. Thus, the model parameters (w0, w and V) of FMs can be learned efficiently by gradient descent methods – e.g. stochastic gradient descent (SGD) – for a variety of losses, among them are square, logit or hinge loss. The gradient of the FM model is:

$$\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^{n} v_{j,f}x_j - v_{i,f}x_i^2, & \text{if } \theta \text{ is } v_{i,f} \end{cases}$$

Appendix 3: Codes

Sentimental Analysis

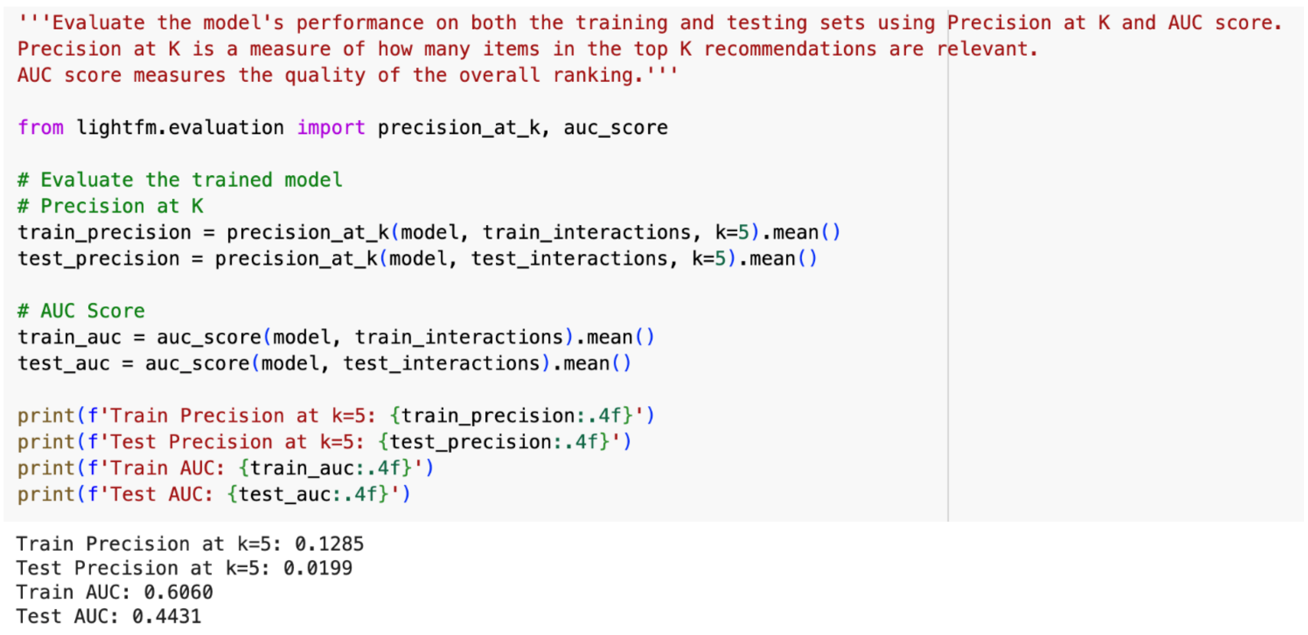
```
from transformers import pipeline, AutoModelForSequenceClassification, AutoTokenizer
import requests

def analyze_sentiment(review):
    model_name = "distilbert-base-uncased-finetuned-sst-2-english"
    model = AutoModelForSequenceClassification.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)


    classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer,truncation=True, padding=True, max_length=512)

    result = classifier(review)

    return result
```

Factorization Machine Model

```
import pandas as pd
from lightfm.data import Dataset
from sklearn.model_selection import train_test_split

# Load your data
user_ids = pd.read_csv('user_ids.txt', header=None, names=['user_id'])
item_ids = pd.read_csv('item_ids.txt', header=None, names=['item_id'])
ratings = pd.read_csv('rating_results.txt', header=None, names=['rating'])

# Combine into a single DataFrame
data = pd.concat([user_ids[:3000], item_ids[:3000], ratings], axis=1)

# Split the data into training and testing sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=42)

# Create a dataset object
dataset = Dataset()
dataset.fit((x for x in data['user_id']),
            (x for x in data['item_id']))

# # Build the interactions matrix
# (interactions, weights) = dataset.build_interactions((x[0], x[1], x[2]) for x in data.values)

# Build the interactions matrix for training data
(train_interactions, train_weights) = dataset.build_interactions((x[0], x[1], x[2]) for x in train_data.values)

# Build the interactions matrix for testing data
(test_interactions, test_weights) = dataset.build_interactions((x[0], x[1], x[2]) for x in test_data.values)
```

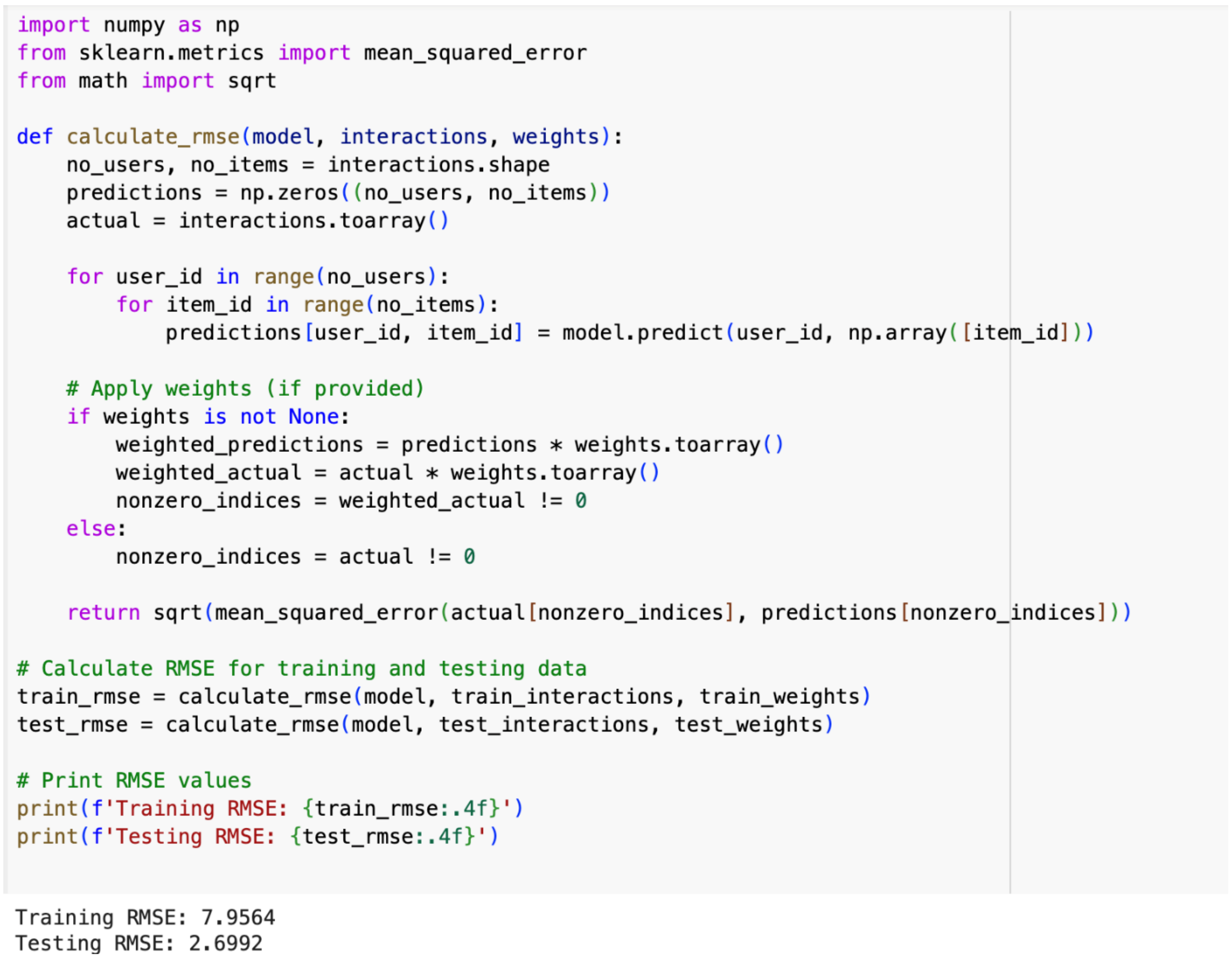
```
from lightfm import LightFM
from lightfm.evaluation import auc_score

# Create model
model = LightFM(loss='warp')  # Weighted Approximate-Rank Pairwise

# Train model
model.fit(train_interactions, sample_weight=train_weights, epochs=30)
```

```
<lightfm.lightfm.LightFM at 0x7e781420a6b0>
```



```python
'''Evaluate the model's performance on both the training and testing sets using Precision at K and AUC score.
Precision at K is a measure of how many items in the top K recommendations are relevant.
AUC score measures the quality of the overall ranking.'''

from lightfm.evaluation import precision_at_k, auc_score

# Evaluate the trained model
# Precision at K
train_precision = precision_at_k(model, train_interactions, k=5).mean()
test_precision = precision_at_k(model, test_interactions, k=5).mean()

# AUC Score
train_auc = auc_score(model, train_interactions).mean()
test_auc = auc_score(model, test_interactions).mean()

print(f'Train Precision at k=5: {train_precision:.4f}')
print(f'Test Precision at k=5: {test_precision:.4f}')
print(f'Train AUC: {train_auc:.4f}')
print(f'Test AUC: {test_auc:.4f}')
```

```
Train Precision at k=5: 0.1285
Test Precision at k=5: 0.0199
Train AUC: 0.6060
Test AUC: 0.4431
```



```python
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

def calculate_rmse(model, interactions, weights):
    no_users, no_items = interactions.shape
    predictions = np.zeros((no_users, no_items))
    actual = interactions.toarray()

    for user_id in range(no_users):
        for item_id in range(no_items):
            predictions[user_id, item_id] = model.predict(user_id, np.array([item_id]))

    # Apply weights (if provided)
    if weights is not None:
        weighted_predictions = predictions * weights.toarray()
        weighted_actual = actual * weights.toarray()
        nonzero_indices = weighted_actual != 0
    else:
        nonzero_indices = actual != 0

    return sqrt(mean_squared_error(actual[nonzero_indices], predictions[nonzero_indices]))

# Calculate RMSE for training and testing data
train_rmse = calculate_rmse(model, train_interactions, train_weights)
test_rmse = calculate_rmse(model, test_interactions, test_weights)

# Print RMSE values
print(f'Training RMSE: {train_rmse:.4f}')
print(f'Testing RMSE: {test_rmse:.4f}')
```

```
Training RMSE: 7.9564
Testing RMSE: 2.6992
```

```
dataset1=Dataset()
# Fit the dataset
dataset1.fit(users=(str(x) for x in user_ids['user_id']),
             items=(str(x) for x in item_ids['item_id']))

# Get the mappings
user_id_map, user_feature_map, item_id_map, item_feature_map = dataset1.mapping()
```

```
import pandas as pd
import numpy as np

def sample_recommendation(model, data, user_ids, user_id_map, item_labels,n):
    n_users, n_items = data.shape
    all_recommendations = []

    for user_id in user_ids:
        # Convert user ID to user index
        user_index = user_id_map[user_id]

        # Predict scores for all items
        scores = model.predict(user_index, np.arange(n_items))

        # Rank items and get top scores
        top_items_indices = np.argsort(-scores)[:n]
        top_scores = scores[top_items_indices]
        top_items = [item_labels[i] for i in top_items_indices]

        # Collect recommendations and their scores
        for item, score in zip(top_items, top_scores):
            all_recommendations.append({
                # 'user_id': user_id,
                'recommended_item': item,
                'score': score
            })
    # Create DataFrame
    recommendations_df = pd.DataFrame(all_recommendations)
    return recommendations_df
```

```
item_labels = item_ids_list[:3000]
user_labels=list(user_id_map.keys())
```

```
sample_user_ids = [user_labels[80]]
recommendations_df = sample_recommendation(model, train_interactions, sample_user_ids, user_id_map, item_labels,10)

print('User id:'+sample_user_ids[0])
print(recommendations_df)
```

```
User id:sergarino
  recommended_item     score
0           437701  1.027002
1              730  0.969984
2           220240  0.959285
3           299360  0.893786
4           292030  0.858746
5           346330  0.856100
6              440  0.824873
7           273350  0.794990
8           221640  0.788892
9           227300  0.782004
```

Deep Neural Network

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=3681))
model.add(BatchNormalization())
model.add(Dense(16, activation='relu'))
#model.add(BatchNormalization())
model.add(Dense(1))


model.compile(optimizer = tf.keras.optimizers.Adam(lr=0.01),loss='mse',metrics=['mae'])

callbacks = [
tf.keras.callbacks.EarlyStopping(

    monitor='val_loss',
    patience=20,
    verbose=1),

tf.keras.callbacks.ReduceLROnPlateau(
    monitor= 'val_loss',
    factor=0.8,
    patience=5,
    min_lr = 0.00001,
    verbose=1
),

tf.keras.callbacks.TensorBoard('log')
]

model.fit(X_train, y_train, epochs=10, batch_size=32)
print ("Train finished")
model.evaluate(X_test, y_test)
print("Evaluate finished")
prediction = model.predict(user_not_involved)
```

```
Epoch 1/10
94/94 [==============================] - 1s 2ms/step - loss: 0.0754 - mae: 0.2075
Epoch 2/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0309 - mae: 0.1332
Epoch 3/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0187 - mae: 0.1057
Epoch 4/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0133 - mae: 0.0902
Epoch 5/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0105 - mae: 0.0816
Epoch 6/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0100 - mae: 0.0796
Epoch 7/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0093 - mae: 0.0773
Epoch 8/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0089 - mae: 0.0753
Epoch 9/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0087 - mae: 0.0741
Epoch 10/10
94/94 [==============================] - 0s 2ms/step - loss: 0.0084 - mae: 0.0732
Train finished
24/24 [==============================] - 0s 1ms/step - loss: 0.0202 - mae: 0.1111
Evaluate finished
689/689 [==============================] - 1s 796us/step
[5556, 6813, 14215, 17833, 14201, 21781, 2761, 3908, 3665, 6243] [1457, 4842, 9801, 3476, 647, 11605, 1865, 4631, 2
1117, 14383]
[1.6400127, 1.570535, 1.4727734, 1.4329534, 1.3955538, 1.3866407, 1.3277098, 1.3133959, 1.3077716, 1.3074853] [0.68
724704, 0.6872158, 0.68714607, 0.68708515, 0.68706554, 0.6870331, 0.6870078, 0.6870013, 0.6869906, 0.6869869]
```