# Book Recommendation System

[Run in Google Colab](#) [View source on GitHub](#)

A recommendation system seeks to predict the rating or preference a user would give to an item given his old item ratings or preferences. Recommendation systems are used by pretty much every major company in order to enhance the quality of their services.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import warnings

from tensorflow.keras.layers import Input, Embedding, Flatten, Dot, Dense, Concatenate
from tensorflow.keras.models import Model

warnings.filterwarnings('ignore')
%matplotlib inline
```

# Loading in data

```
# Install Kaggle API
!pip install -q kaggle
!pip install -q kaggle-cli
```

```
     |████████████████████████████████| 81kB 9.4MB/s
     |████████████████████████████████| 4.2MB 39.3MB/s
     |████████████████████████████████| 51kB 7.6MB/s
     |████████████████████████████████| 143kB 49.3MB/s
     |████████████████████████████████| 112kB 56.9MB/s
  Building wheel for kaggle-cli (setup.py) ... done
  Building wheel for lxml (setup.py) ... error
  ERROR: Failed building wheel for lxml
  Building wheel for PrettyTable (setup.py) ... done
  Building wheel for pyperclip (setup.py) ... done
    Running setup.py install for lxml ... error
ERROR: Command errored out with exit status 1: /usr/bin/python3 -u -c 'import sy
```

```
# only for google colab
import os
os.environ['KAGGLE_USERNAME'] = "<username>"
os.environ['KAGGLE_KEY'] = "<key>"
```

```
!kaggle datasets download -d zygmunt/goodbooks-10k --unzip
```

```
Downloading goodbooks-10k.zip to /content
 86% 10.0M/11.6M [00:00<00:00, 103MB/s]
100% 11.6M/11.6M [00:00<00:00, 74.3MB/s]
```

```
dataset = pd.read_csv('ratings.csv')
```

```
dataset.head()
```

| | book_id | user_id | rating |
|---|---|---|---|
| **0** | 1 | 314 | 5 |
| **1** | 1 | 439 | 3 |
| **2** | 1 | 588 | 5 |
| **3** | 1 | 1169 | 4 |
| **4** | 1 | 1185 | 4 |

```
dataset.shape
```

```
(981756, 3)
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(dataset, test_size=0.2, random_state=42)
```

```
train.head()
```

| | book_id | user_id | rating |
|---|---|---|---|
| **341848** | 3423 | 4608 | 2 |
| **964349** | 9811 | 36373 | 5 |
| **645459** | 6485 | 2957 | 4 |
| **74960** | 750 | 42400 | 3 |
| **358670** | 3591 | 36886 | 5 |

```
test.head()
```

| | book_id | user_id | rating |
|---|---|---|---|
| **646451** | 6495 | 19643 | 5 |
| **614851** | 6175 | 8563 | 4 |
| **974393** | 9920 | 52110 | 3 |
| **21471** | 215 | 33864 | 5 |
| **272540** | 2728 | 16587 | 3 |

```
n_users = len(dataset.user_id.unique())
n_users
```

```
53424
```

```
n_books = len(dataset.book_id.unique())
n_books
```

```
10000
```

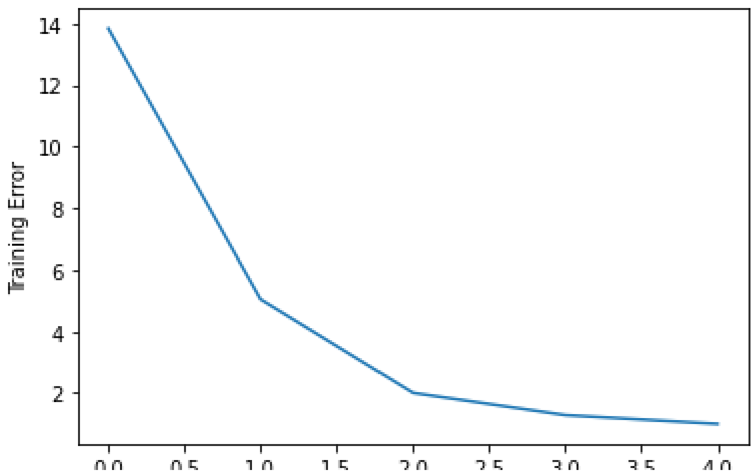
## Creating dot product model

Most recommendation systems are build using a simple dot product as shown below but newer ones are now implementing a neural network instead of the simple dot product.

```
# creating book embedding path
book_input = Input(shape=[1], name="Book-Input")
book_embedding = Embedding(n_books+1, 5, name="Book-Embedding")(book_input)
book_vec = Flatten(name="Flatten-Books")(book_embedding)

# creating user embedding path
user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users+1, 5, name="User-Embedding")(user_input)
user_vec = Flatten(name="Flatten-Users")(user_embedding)

# performing dot product and creating model
prod = Dot(name="Dot-Product", axes=1)([book_vec, user_vec])
model = Model([user_input, book_input], prod)
model.compile('adam', 'mean_squared_error')
```

```
from keras.models import load_model

if os.path.exists('regression_model.h5'):
    model = load_model('regression_model.h5')
else:
    history = model.fit([train.user_id, train.book_id], train.rating, epochs=5, verbose=1)
    model.save('regression_model.h5')
    plt.plot(history.history['loss'])
    plt.xlabel("Epochs")
    plt.ylabel("Training Error")
```

```
Epoch 1/5
24544/24544 [==============================] - 112s 4ms/step - loss: 15.3271
Epoch 2/5
24544/24544 [==============================] - 107s 4ms/step - loss: 6.6203
Epoch 3/5
24544/24544 [==============================] - 107s 4ms/step - loss: 2.2780
Epoch 4/5
24544/24544 [==============================] - 108s 4ms/step - loss: 1.3414
Epoch 5/5
24544/24544 [==============================] - 107s 4ms/step - loss: 1.0093
```




```
model.evaluate([test.user_id, test.book_id], test.rating)
```

```
6136/6136 [==============================] - 8s 1ms/step - loss: 1.2534
1.253379225730896
```

```
predictions = model.predict([test.user_id.head(10), test.book_id.head(10)])

[print(predictions[i], test.rating.iloc[i]) for i in range(0,10)]
```

```
[4.9011655] 5
[3.909601] 4
[3.5981612] 3
[4.3802824] 5
[3.7667456] 3
[3.7533305] 3
[3.0974283] 3
[4.841637] 4
[3.8465247] 3
```

```
[4.0708756] 5
[None, None, None, None, None, None, None, None, None, None]
```

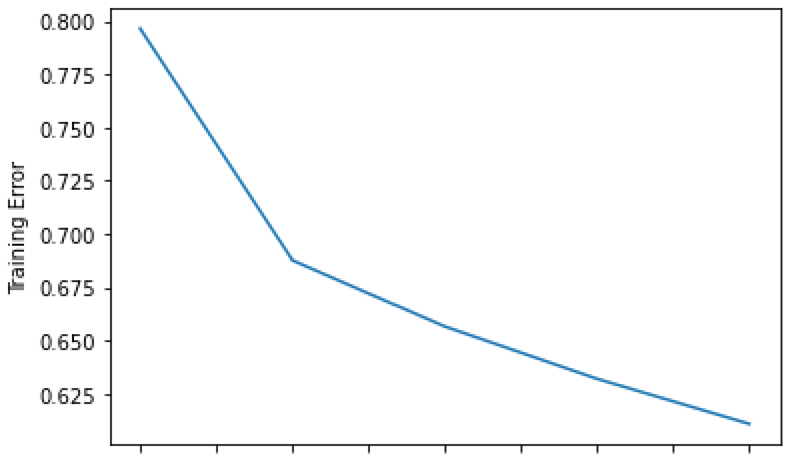
## Creating Neural Network

Neural Networks proved there effectivness for almost every machine learning problem as of now and they also perform exceptionally well for recommendation systems.

```
# creating book embedding path
book_input = Input(shape=[1], name="Book-Input")
book_embedding = Embedding(n_books+1, 5, name="Book-Embedding")(book_input)
book_vec = Flatten(name="Flatten-Books")(book_embedding)

# creating user embedding path
user_input = Input(shape=[1], name="User-Input")
user_embedding = Embedding(n_users+1, 5, name="User-Embedding")(user_input)
user_vec = Flatten(name="Flatten-Users")(user_embedding)

# concatenate features
conc = Concatenate()([book_vec, user_vec])

# add fully-connected-layers
fc1 = Dense(128, activation='relu')(conc)
fc2 = Dense(32, activation='relu')(fc1)
out = Dense(1)(fc2)

# Create model and compile it
model2 = Model([user_input, book_input], out)
model2.compile('adam', 'mean_squared_error')
```

```
from keras.models import load_model

if os.path.exists('regression_model2.h5'):
    model2 = load_model('regression_model2.h5')
else:
    history = model2.fit([train.user_id, train.book_id], train.rating, epochs=5, verbose=1
    model2.save('regression_model2.h5')
    plt.plot(history.history['loss'])
    plt.xlabel("Epochs")
    plt.ylabel("Training Error")
```

```
Epoch 1/5
24544/24544 [==============================] - 115s 5ms/step - loss: 0.9909
Epoch 2/5
24544/24544 [==============================] - 114s 5ms/step - loss: 0.6816
Epoch 3/5
24544/24544 [==============================] - 112s 5ms/step - loss: 0.6462
Epoch 4/5
24544/24544 [==============================] - 112s 5ms/step - loss: 0.6210
Epoch 5/5
24544/24544 [==============================] - 112s 5ms/step - loss: 0.5978
```




```
model2.evaluate([test.user_id, test.book_id], test.rating)
```

```
6136/6136 [==============================] - 9s 1ms/step - loss: 0.7132
0.7132264971733093
```

```
predictions = model2.predict([test.user_id.head(10), test.book_id.head(10)])

[print(predictions[i], test.rating.iloc[i]) for i in range(0,10)]
```

```
[5.107734] 5
[3.902279] 4
[3.2454526] 3
[3.968115] 5
[3.1826286] 3
[4.056015] 3
[3.8675883] 3
[4.7206354] 4
[4.269019] 3
[4.1037846] 5
[None, None, None, None, None, None, None, None, None, None]
```

## Visualizing Embeddings

Embeddings are weights that are learned to represent some specific variable like books and user in our case and therefore we can not only use them to get good results on our problem but also to extract inside about our data.

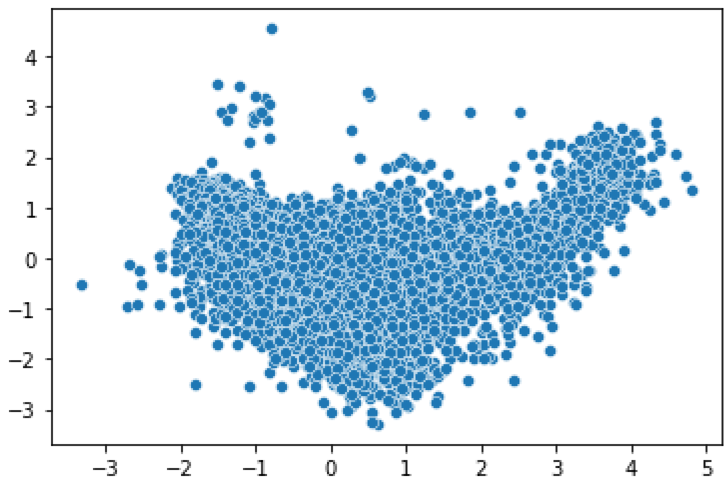
```
# Extract embeddings
book_em = model.get_layer('Book-Embedding')
book_em_weights = book_em.get_weights()[0]
```

```
book_em_weights[:5]
```

```
array([[-0.0047317 ,  0.02763932, -0.04600314, -0.0412247 ,  0.00620728],
       [ 1.2028387 ,  1.2857639 , -1.5278164 , -1.1005933 ,  0.8269744 ],
       [ 0.97660714,  1.328868  , -1.3593423 , -1.3042482 ,  1.0568628 ],
       [ 0.949213  ,  0.67221147, -1.2034689 , -0.46824247,  0.86552835],
       [ 0.8755239 ,  1.3966527 , -1.370951  , -1.350987  ,  1.3144002 ]],
      dtype=float32)
```

```
from sklearn.decomposition import PCA
import seaborn as sns

pca = PCA(n_components=2)
pca_result = pca.fit_transform(book_em_weights)
sns.scatterplot(x=pca_result[:,0], y=pca_result[:,1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff8c0478210>
```

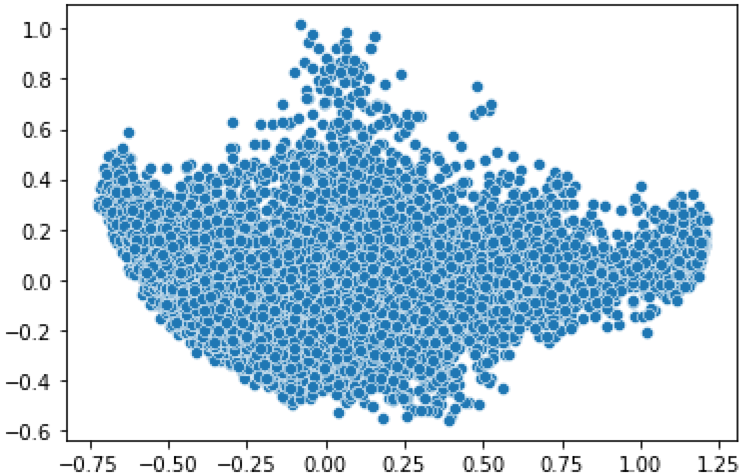



```
book_em_weights = book_em_weights / np.linalg.norm(book_em_weights, axis = 1).reshape((-1,
book_em_weights[0][:10]
np.sum(np.square(book_em_weights[0]))
```

```
0.99999994
```

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(book_em_weights)
sns.scatterplot(x=pca_result[:,0], y=pca_result[:,1])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff8bf991450>

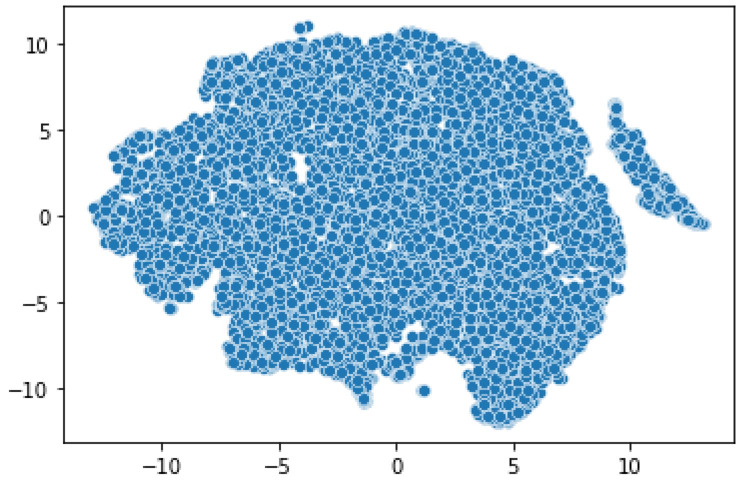


```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tnse_results = tsne.fit_transform(book_em_weights)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 10001 samples in 0.007s...
[t-SNE] Computed neighbors for 10001 samples in 0.443s...
[t-SNE] Computed conditional probabilities for sample 1000 / 10001
[t-SNE] Computed conditional probabilities for sample 2000 / 10001
[t-SNE] Computed conditional probabilities for sample 3000 / 10001
[t-SNE] Computed conditional probabilities for sample 4000 / 10001
[t-SNE] Computed conditional probabilities for sample 5000 / 10001
[t-SNE] Computed conditional probabilities for sample 6000 / 10001
[t-SNE] Computed conditional probabilities for sample 7000 / 10001
[t-SNE] Computed conditional probabilities for sample 8000 / 10001
[t-SNE] Computed conditional probabilities for sample 9000 / 10001
[t-SNE] Computed conditional probabilities for sample 10000 / 10001
[t-SNE] Computed conditional probabilities for sample 10001 / 10001
[t-SNE] Mean sigma: 0.048991
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.993668
[t-SNE] KL divergence after 300 iterations: 2.458674
```

```
sns.scatterplot(x=tnse_results[:,0], y=tnse_results[:,1])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff930149290>
```



## Making Recommendations

```
# Creating dataset for making recommendations for the first user
book_data = np.array(list(set(dataset.book_id)))
book_data[:5]
```

```
array([1, 2, 3, 4, 5])
```

```
user = np.array([1 for i in range(len(book_data))])
user[:5]
```

```
array([1, 1, 1, 1, 1])
```

```
predictions = model.predict([user, book_data])

predictions = np.array([a[0] for a in predictions])

recommended_book_ids = (-predictions).argsort()[:5]

recommended_book_ids
```

```
array([5858, 7638, 6246, 9208, 8232])
```

```
# print predicted scores
predictions[recommended_book_ids]
```

```
array([3.8500102, 3.5082545, 3.3832717, 3.3525784, 3.1709607],
      dtype=float32)
```

```
books = pd.read_csv('books.csv')
books.head()
```

| | id | book_id | best_book_id | work_id | books_count | isbn | isbn13 | author |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2767052 | 2767052 | 2792775 | 272 | 439023483 | 9.780439e+12 | Suzann Collir |
| **1** | 2 | 3 | 3 | 4640799 | 491 | 439554934 | 9.780440e+12 | J. Rowlin Ma GrandP |
| **2** | 3 | 41865 | 41865 | 3212258 | 226 | 316015849 | 9.780316e+12 | Stephen Mey |
| **3** | 4 | 2657 | 2657 | 3275794 | 487 | 61120081 | 9.780061e+12 | Harp Le |
| **4** | 5 | 4671 | 4671 | 245494 | 1356 | 743273567 | 9.780743e+12 | F. Scc Fitzgera |

```
books[books['id'].isin(recommended_book_ids)]
```

| | id | book_id | best_book_id | work_id | books_count | isbn | isbn13 |
|---|---|---|---|---|---|---|---|