

# Collaborative filtering with FastAI

Building a book recommendation system using the FastAI deep learning library



Gilbert Tanner

Follow

Apr 7 · 6 min read ★



Figure 1: Photo by Aaron Burden on Unsplash

A collaborative filtering model/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.

In one of my [first articles](#), I created a book recommendation system using [Keras](#) a high-level API build on [Tensorflow](#). In this article, I will show you **how to build the same recommendation system using the FastAI library as well as how to build a neural network based model to get even better results.**

For our data, we will use the [goodbooks-10k dataset](#) which contains ten thousand different books and about one million ratings. It has three features the book\_id, user\_id and rating. If you want you can get the all the data files as well as the complete code covered in this article from my Github repository.

If you prefer a visual tutorial you can check out my FastAi videos.

. . .

## Getting data

After downloading the data-set from Kaggle we need to load in FastAIs collab filtering module, specify the path to the data-set and load in the csv containing the ratings as well as the csv containing the book information.

```
1  from fastai.collab import *
2
3  # specify path
4  path = Path('<path to the data-set>')
5  print(path.ls())
6
7  # load in ratings data
8  ratings = pd.read_csv(path/'ratings.csv')
9  print(ratings.head())
```

	user_id	book_id	rating
0	1	258	5
1	2	4081	4
2	2	260	5
3	2	9296	5
4	2	2318	3

Figure 2: Recommendation dataframe

[4]:

	book_id	goodreads_book_id	best_book_id	work_id	books_count	isbn	isbn13	authors	original_publication_year	original_title	...
0	1	2767052	2767052	2792775	272	439023483	9.780439e+12	Suzanne Collins	2008.0	The Hunger Games	...
1	2	3	3	4640799	491	439554934	9.780440e+12	J.K. Rowling, Mary GrandPré	1997.0	Harry Potter and the Philosopher's Stone	...
2	3	41865	41865	3212258	226	316015849	9.780316e+12	Stephenie Meyer	2005.0	Twilight	...
3	4	2657	2657	3275794	487	61120081	9.780061e+12	Harper Lee	1960.0	To Kill a Mockingbird	...
4	5	4671	4671	245494	1356	743273567	9.780743e+12	F. Scott Fitzgerald	1925.0	The Great Gatsby	...

5 rows x 23 columns

Figure 3: Book information dataframe

With our data loaded in we can create a `CollabDataBunch`, which is a databunch specifically created for collaborative filtering problems. We will pass it our rating data, a random seed as well as the size of our validation set which is defined by the `valid_pct` argument.

```
1 data = CollabDataBunch.from_df(ratings, seed=42, valid_pct=0.2)
```

As always we can show a batch of our data using the `show_batch` method.

```
1 data.show_batch()
```

book_id	user_id	target
1470	36544	2.0
661	5461	4.0
596	9285	4.0
3952	17522	3.0
6628	10394	3.0

Figure 4: Random data batch

The last thing we need to do before creating and training our model is getting the max and min values of our ratings. We will then pass these values to our model so it can then squeeze the final outputs between these two values.

```
1 ratings.rating.min(), ratings.rating.max()
```

Output: (1, 5)

. . .

## EmbeddingDotBias Model

FastAI provides two different types of collab models. A simple model called `EmbeddingDotBias` which used for almost all recommendation systems a few years ago. It creates embeddings for both users and books and then takes the dot product of them. The second is a neural network based model which uses embeddings and fully-connected layers.

An embedding is a mapping from discrete objects, such as words or ids of books and users in our case, to a vector of continuous values. This can be used to find similarities between the discrete objects, that wouldn't be apparent to the model if it didn't use embedding layers.

These embedding vectors are low-dimensional and get updated whilst training the network.

Both models can be created using the `collab_learner` class. Standardly the `use_nn` argument is set to false and therefore we are creating an `EmbeddingDotBias` model.

As further arguments, we can pass the collab learner the `n_factors` argument which represents the size of the embedding vectors as well as the `yrange` argument which specifies the range of the rating values we found earlier.

```
1 learn = collab_learner(data, n_factors=40, y_range=(1,
```

Now we can find the learning rate, train our model using the `fit_one_cycle` method and save the model. If you aren't familiar with this process yet I would recommend you to check out [my first article](#) about Image Classification using the FastAI library.

```
1 learn.lr_find() # find learning rate
2 learn.recorder.plot() # plot learning rate graph
```

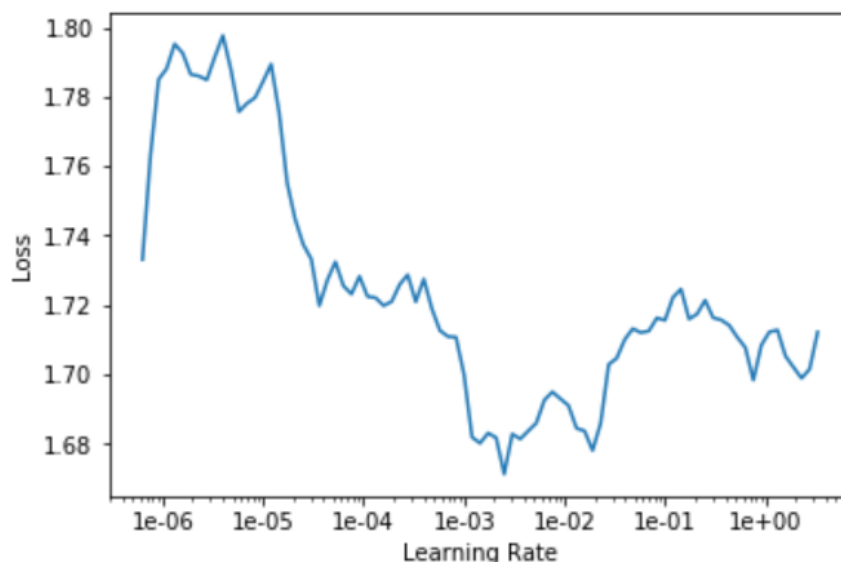


Figure 4: Learning rate plot

```
1 learn.fit_one_cycle(5, 3e-4)
```

Total time: 11:48

epoch	train_loss	valid_loss	time
1	1.574284	1.588789	02:21
2	1.315979	1.313377	02:21
3	1.133663	1.126915	02:22
4	1.059580	1.062018	02:22
5	1.045705	1.052467	02:21

Figure 5: Training results

```
1 learn.save('goodbooks-dot-1')
```

## EmbeddingNN Model

The second type of collaborative filtering model provided by FastAI is called *EmbeddingNN*. It provides us with the ability to create embeddings with different sizes and feed them into a neural network.

FastAI also provides us with the ability to tweak the number of layers and their units.

```
1 learn = collab_learner(data, use_nn=True, emb_szs={'use
```

As always the next steps are to find the learning rate and train the model.

```
1 learn.lr_find() # find learning rate
2 learn.recorder.plot() # plot learning rate graph
```

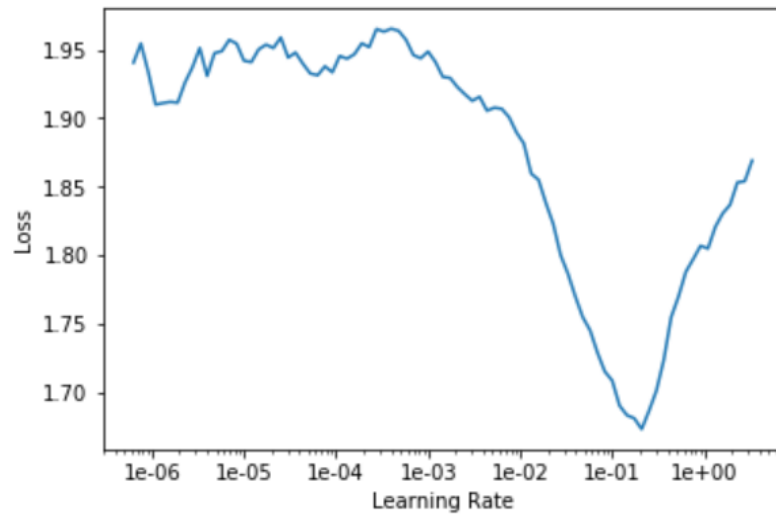


Figure 6: Learning rate plot

```
1 learn.fit_one_cycle(5, 1e-2)
```

Total time: 13:47

epoch	train_loss	valid_loss	time
1	0.763226	0.761729	02:45
2	0.747357	0.743163	02:46
3	0.702217	0.725356	02:45
4	0.675362	0.704841	02:45
5	0.571993	0.717858	02:45

Figure 7: Training results

We can see that the neural network performed a lot better than our dot-product model.

. . .

## Interpretation

Because the embeddings learned should represent the style and kind of books and users as best as possible they might contain interesting features which we can extract and visualize in order of getting insides.

For this purpose FastAI allows you to easily access both the user and book embeddings as well as their biases.

In this article, we will extract the book biases and weights in order to get insights of what books should be ranked lower or higher, using the embedding bias, as well as how similar some of the most popular books are using the embedding weights.

To start off we will load in our EmbeddingDotBias model and get the 1000 most popular books by how much reviews they have.

```
1  # load in EmbeddingDotBias model
2  learn = collab_learner(data, n_factors=40, y_range=(1,
3  learn.load('goodbooks-dot-1');
4
5  # get top books
6  g = ratings.groupby('book_id')['rating'].count()
7  top_books = g.sort_values(ascending=False).index.values
8  top_books = top_books.astype(str)
9
10 # create array containing the names of the top books
```

Now we can extract the biases for the top books as well as the mean rating of the top books and print them out. With this, we can get information about the books that are generally rated low or high no matter what user is rating them.



```

1  # get biases for top books
2  book_bias = learn.bias(top_books, is_item=True)
3
4  # get mean ratings
5  mean_ratings = ratings.groupby('book_id')['rating'].mean()
6  book_ratings = [(b, top_books_with_name[i], mean_ratings[i]) for i in range(len(top_books_with_name))]
7
8  # print book bias information
9  item0 = lambda o:o[0]
10 print(sorted(book_ratings, key=item0)[:15])
11 print(sorted(book_ratings, key=item0, reverse=True)[:15])
12
13 # get weights
14 book_w = learn.weight(top_books, is_item=True)
15
16 # transform weights to 3 dimensions
17 book_pca = book_w.pca(3)
18
19 # get principal components
20 fac0, fac1, fac2 = book_pca.t()

```

Output:

Top idx:

```
array(['5000', '3315', '3313', '3312', '3311', '3309',
      '3308', '3307', '3306', '3304'], dtype='<U21')
```

Top names:

```
array(['Passion Unleashed (Demonica #3)', 'My Story', 'The
      Gargoyle', 'Pretty Baby', ...,
      'Top Secret Twenty-One (Stephanie Plum, #21)', 'The
      Warrior Heir (The Heir Chronicles, #1)', 'Stone Soup',
      'The Sixth Man (Sean King & Michelle Maxwell, #5)'],
      dtype='<U144')
```

Most negative bias:

```
[(tensor(-0.1021), 'The Almost Moon', 2.49),
 (tensor(-0.0341), 'Skinny Bitch', 2.9),
 (tensor(-0.0325), 'Bergdorf Blondes', 3.0),
 (tensor(-0.0316), 'The Particular Sadness of Lemon Cake',
 2.93),
 (tensor(-0.0148), 'The Weird Sisters', 3.08)]
...
```

We can also visualize the two principal components using a graphing library like Matplotlib.

```
1 idxs = np.random.choice(len(top_books_with_name), 50, r
2 idxs = list(range(50))
3 X = fac0[idxs]
4 Y = fac2[idxs]
5 plt.figure(figsize=(15,15))
6 plt.scatter(X, Y)
7 for i, x, y in zip(top_books_with_name[idxs], X, Y):
```

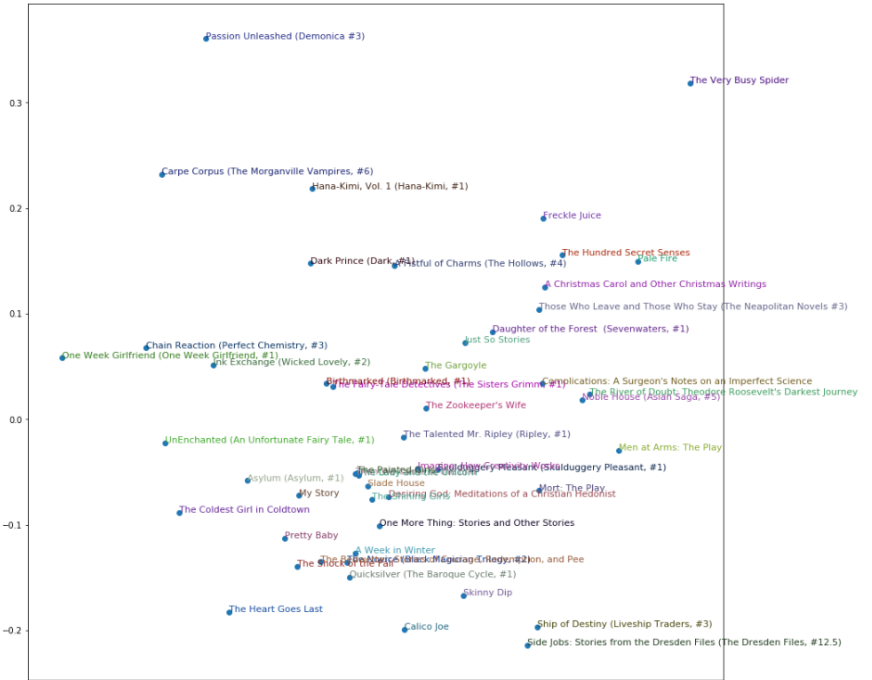


Figure 8: Weight Embedding visualization

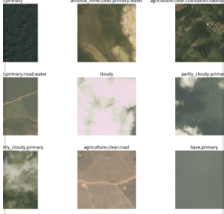
. . .

## Recommended Reading

FastAI Multi-label image classification

Learn how to work with multi-label data

[towardsdatascience.com](https://towardsdatascience.com)



. . .

## Conclusion

A recommendation system seeks to predict the rating or preference a user would give to an item given his old item ratings or preferences.

The FastAI deep learning library provides us with functionality to easily load in our data and build our collaborative filtering/recommendation system model.

If you liked this article consider subscribing to my [Youtube Channel](#) and following me on social media.

The code covered in this article is available as a [Github Repository](#).

If you have any questions, recommendations or critiques, I can be reached via [Twitter](#) or the comment section.

