
Disco

:fire: Recommendations for Ruby and Rails using collaborative filtering

- Supports user-based and item-based recommendations
- Works with explicit and implicit feedback
- Uses high-performance matrix factorization



Installation

Add this line to your application's Gemfile:

```
1 gem "disco"
```

Getting Started

Create a recommender

```
1 recommender = Disco::Recommender.new
```

If users rate items directly, this is known as explicit feedback. Fit the recommender with:

```
1 recommender.fit([
2   {user_id: 1, item_id: 1, rating: 5},
3   {user_id: 2, item_id: 1, rating: 3}
4 ])
```

IDs can be integers, strings, or any other data type

If users don't rate items directly (for instance, they're purchasing items or reading posts), this is known as implicit feedback. Leave out the rating.

```
1 recommender.fit([
2   {user_id: 1, item_id: 1},
3   {user_id: 2, item_id: 1}
4 ])
```

Each `user_id/item_id` combination should only appear once

Get user-based recommendations - "users like you also liked"

```
1 recommender.user_recs(user_id)
```

Get item-based recommendations - “users who liked this item also liked”

```
1 recommender.item_recs(item_id)
```

Use the `count` option to specify the number of recommendations (default is 5)

```
1 recommender.user_recs(user_id, count: 3)
```

Get predicted ratings for specific users and items

```
1 recommender.predict([{:user_id: 1, :item_id: 2}, {:user_id: 2, :item_id: 4}])
```

Get similar users

```
1 recommender.similar_users(user_id)
```

Examples

MovieLens

Load the data

```
1 data = Disco.load_movielens
```

Create a recommender and get similar movies

```
1 recommender = Disco::Recommender.new(factors: 20)
2 recommender.fit(data)
3 recommender.item_recs("Star Wars (1977)")
```

Ahoy

Ahoy is a great source for implicit feedback

```
1 views = Ahoy::Event.where(name: "Viewed post").group(:user_id).
  group_prop(:post_id).count
2
3 data =
4   views.map do |(user_id, post_id), _|
5     {
6       user_id: user_id,
7       item_id: post_id
```

```
8   }  
9   end
```

Create a recommender and get recommended posts for a user

```
1 recommender = Disco::Recommender.new  
2 recommender.fit(data)  
3 recommender.user_recs(current_user.id)
```

Storing Recommendations

Disco makes it easy to store recommendations in Rails.

```
1 rails generate disco:recommendation  
2 rails db:migrate
```

For user-based recommendations, use:

```
1 class User < ApplicationRecord  
2   has_recommended :products  
3 end
```

Change `:products` to match the model you're recommending

Save recommendations

```
1 User.find_each do |user|  
2   recs = recommender.user_recs(user.id)  
3   user.update_recommended_products(recs)  
4 end
```

Get recommendations

```
1 user.recommended_products
```

For item-based recommendations, use:

```
1 class Product < ApplicationRecord  
2   has_recommended :products  
3 end
```

Specify multiple types of recommendations for a model with:

```
1 class User < ApplicationRecord  
2   has_recommended :products  
3   has_recommended :products_v2, class_name: "Product"  
4 end
```

And use the appropriate methods:

```
1 user.update_recommended_products_v2(recs)
2 user.recommended_products_v2
```

Storing Recommenders

If you'd prefer to perform recommendations on-the-fly, store the recommender

```
1 json = recommender.to_json
2 File.write("recommender.json", json)
```

The serialized recommender includes user activity from the training data (to avoid recommending previously rated items), so be sure to protect it. You can save it to a file, database, or any other storage system, or use a tool like Trove. Also, user and item IDs should be integers or strings for this.

Load a recommender

```
1 json = File.read("recommender.json")
2 recommender = Disco::Recommender.load_json(json)
```

Alternatively, you can store only the factors and use a library like Neighbor. See the examples.

Algorithms

Disco uses high-performance matrix factorization.

- For explicit feedback, it uses stochastic gradient descent
- For implicit feedback, it uses coordinate descent

Specify the number of factors and epochs

```
1 Disco::Recommender.new(factors: 8, epochs: 20)
```

If recommendations look off, trying changing `factors`. The default is 8, but 3 could be good for some applications and 300 good for others.

Validation

Pass a validation set with:

```
1 recommender.fit(data, validation_set: validation_set)
```

Cold Start

Collaborative filtering suffers from the cold start problem. It's unable to make good recommendations without data on a user or item, which is problematic for new users and items.

```
1 recommender.user_recs(new_user_id) # returns empty array
```

There are a number of ways to deal with this, but here are some common ones:

- For user-based recommendations, show new users the most popular items
- For item-based recommendations, make content-based recommendations with a gem like tf-idf-similarity

Get top items with:

```
1 recommender = Disco::Recommender.new(top_items: true)
2 recommender.fit(data)
3 recommender.top_items
```

This uses Wilson score for explicit feedback and item frequency for implicit feedback.

Data

Data can be an array of hashes

```
1 [{user_id: 1, item_id: 1, rating: 5}, {user_id: 2, item_id: 1, rating: 3}]
```

Or a Rover data frame

```
1 Rover.read_csv("ratings.csv")
```

Or a Daru data frame

```
1 Daru::DataFrame.from_csv("ratings.csv")
```

Performance

If you have a large number of users or items, you can use an approximate nearest neighbors library like Faiss to improve the performance of certain methods.

Add this line to your application's Gemfile:

```
1 gem "faiss"
```

Speed up the `user_recs` method with:

```
1 recommender.optimize_user_recs
```

Speed up the `item_recs` method with:

```
1 recommender.optimize_item_recs
```

Speed up the `similar_users` method with:

```
1 recommender.optimize_similar_users
```

This should be called after fitting or loading the recommender.

Reference

Get ids

```
1 recommender.user_ids
2 recommender.item_ids
```

Get the global mean

```
1 recommender.global_mean
```

Get factors

```
1 recommender.user_factors
2 recommender.item_factors
```

Get factors for specific users and items

```
1 recommender.user_factors(user_id)
2 recommender.item_factors(item_id)
```

Credits

Thanks to:

- LIBMF for providing high performance matrix factorization
- Implicit for serving as an initial reference for user and item similarity
- @dasch for the gem name

History

[View the changelog](#)

Contributing

Everyone is encouraged to help improve this project. Here are a few ways you can help:

- Report bugs
- Fix bugs and submit pull requests
- Write, clarify, or fix documentation
- Suggest or add new features

To get started with development:

```
1 git clone https://github.com/ankane/disco.git
2 cd disco
3 bundle install
4 bundle exec rake test
```