aaw / **IncrementalSVD.jl**

Simon Funk's approach to collaborative filtering using the singular value decomposition, implemented in Julia.

9 commits | **3** branches | **0** releases | **3** contributors | View license

Branch: **master** | New pull request | Create new file | Upload files | Find File | Clone or download

aaw Merge pull request #10 from joeltay/patch-1 ··· Latest commit 22f8e54 on Dec 7, 2016

src	Fixed link	3 years ago
LICENSE.md	IncrementalSVD.jl generated files.	5 years ago
README.md	IncrementalSVD.jl generated files.	5 years ago
REQUIRE	IncrementalSVD.jl generated files.	5 years ago

README.md

IncrementalSVD.jl

What is this?

Simon Funk's singular value decomposition implemented in Julia. Collaborative filtering that can analyze millions of ratings on a laptop in a few minutes.

What does that mean?

If you have a set of user ratings, saves, follows, etc., you can use this package to turn those ratings into:

- Predictions of how users might rate other items.
- Clusters of similar items and similar users.
- Identification of important features in your data.

What is a singular value decomposition?

The singular value decomposition (SVD) factors any m by n matrix M into an m by m matrix U, an m by n diagonal matrix S, and an n by n matrix V such that $M = USV^T$. One of the many interesting properties of this decomposition is that you can choose any $r < n$, create a new matrix T from S by zeroing out anything that isn't on the first r rows and columns in S, and then UTV^T is the closest matrix of rank r to M in terms of the Frobenius norm (in other words, over all rank r matrices N, $N = UTV^T$ minimizes the sum of the squared differences between N and M.)

How do you use an SVD to build a recommendation engine?

Let's say that you have a large set of movie ratings. Consider some cluster of similar movies, say, the Friday the 13th horror franchise. One user's ratings of the Friday the 13th movies are probably pretty consistent: either they like the series and rate all of the Friday the 13th films pretty high or they don't like any of them and rate them all pretty low. In addition, maybe the first couple of films in the series were a lot better than the later ones, so even users who didn't really like the series would rate Friday the 13th part I and II a little higher.

If you wanted to build a really simple model of these Friday the 13th ratings, you could just set up a vector of base ratings (say, 1.0 for the first two films, then 0.8 for all of the rest) and fit each user's predictions by finding the multiple of that vector that best fits all of the ratings you have from that user. Given a user who rated only Friday the 13th part II and gave it a 5/5, we'd just multiply the base ratings vector by 5 to get that user's predicted ratings for the entire series: 5/5 for part I and II and 4/5 for the rest. The resulting vector follows our imagined model for ratings of the series and it also agrees on the one actual rating we know from the user.

Now imagine putting all of the movie ratings you have into one big user-by-item matrix. Somewhere in there, you'd have 12 columns for Friday the 13th movies, one for each movie in the series. Since we can't force our users to watch and rate all of the movies in the series, what we see in those columns is some sparse sample of ratings. Taken as vectors projected on just those 12 columns, it's pretty likely that we'd need a basis of size 12 to represent them all (this would be true, for example, if there are 12 users who have each only rated a single different Friday the 13th film.) But that's a basis for the sparse set of known ratings, not the dense set of predicted ratings. What we'd actually like is a basis of smaller rank the like single-vector basis we used in the previous paragraph that generalizes the sparse, observed ratings.

That's where the SVD comes in: we can choose a small rank and extract a matrix of exactly that rank from the SVD. The resulting matrix will still approximate the original matrix, so decreasing the rank will just smooth out the ratings by forcing them to be linear combinations of only a few basis vectors while matching our sparsely observed ratings as closely as possible. And the best part is that you don't even need to have specific knowledge of the domain that you're trying to extract information about - in this case, the Friday the 13th movies aren't annotated in any way, but as long as we have enough ratings that follow the linear ratings model that we described earlier, the SVD will cluster those movies together simply because it's the easiest thing to do under the sum of squared differences penalty.

Great, but Julia already has an `svd` function. I'll just use that.

Computing an SVD of a user-item matrix is expensive. If you have reasonable numbers of users or items, you may not even be able to fit the matrix in memory to begin with. More importantly, a model based on a reduced-rank SVD minimizes the error on both the sparse set of ratings you know and on the dense set of unknown ratings you're trying to predict. You can initialize the unknown entries based on some prior instead of setting them all to zeros, but when you compute an SVD over the entire matrix, you're penalizing both the differences from your known ratings and the differences from your predictions of unknown ratings equally, and there are typically many more unknown ratings than known in the ratings matrix. So you end up essentially fitting the model to your prior when you'd like to fit the model only to your observed ratings instead.

So what does this package do that's different than an SVD?

Simon Funk's algorithm creates a decomposition that attempts to minimize the sum of squared errors against only the known ratings. It finds this decomposition via gradient descent, which along with the fact that you're only considering a (relatively) small number of ratings instead of the entire user-item matrix means that it's fast and you can run it on any machine that can hold all of your ratings in memory.

Where can I read more about Simon Funk's algorithm?

The gradient descent approach to SVD originated in a 2005 paper by Funk and Gorrell. Simon Funk then popularized this approach by using it to do exceptionally well in the Netflix Prize competition in 2006 and writing up a series of posts about his implementation:

- Introduction
- More detail on the algorithm
- Derivation of the gradient descent update rule

There's also a good discussion in the Netflix prize forum about implementing and tuning Funk's algorithm on the Netflix Prize dataset.

How do I install this package?

```
Pkg.clone("git://github.com/aaw/IncrementalSVD.jl.git")
```

How do I use this package build a recommendation engine?

IncrementalSVD comes with functions to load two benchmark datasets for collaborative filtering: the MovieLens dataset and the Book-Crossing dataset. Let's load the small MovieLens dataset:

```
julia> import IncrementalSVD

julia> rating_set = IncrementalSVD.load_small_movielens_dataset();
```

The small MovieLens dataset has 1 million ratings from 6000 users over 4000 movies. Train a model with 25 features on these ratings:

```
julia> model = IncrementalSVD.train(rating_set, 25);
```

More later on how to choose a good number of features, for now just consider 25 a lucky guess. The truncated SVD that we generated with the above command gave us, among other things, a matrix U, which has dimension [number of users] x 25, and a matrix V, which has dimension [number of movies] x 25. You can use either of these matrices as feature vectors for your users or movies. You can compare these vectors using, say, cosine similarity to get a list of similar movies to use for clustering or recommendations. For example,

```
julia> IncrementalSVD.similar_items(model, "Friday the 13th (1980)")
10-element Array{String,1}:
 "Friday the 13th (1980)"
 "Amityville Horror, The (1979)"
 "Jaws 2 (1978)"
 "Pet Sematary (1989)"
 "Omen, The (1976)"
 "Porky's (1981)"
 "Cujo (1983)"
 "Stepford Wives, The (1975)"
 "Halloween II (1981)"
 "Battle for the Planet of the Apes (1973)"

julia> IncrementalSVD.similar_items(model, "Friday the 13th Part 2 (1981)")
10-element Array{String,1}:
 "Friday the 13th Part 2 (1981)"
 "Halloween II (1981)"
 "Friday the 13th Part V: A New Beginning (1985)"
 "Porky's II: The Next Day (1983)"
 "Friday the 13th: The Final Chapter (1984)"
 "Cujo (1983)"
 "Damien: Omen II (1978)"
 "Towering Inferno, The (1974)"
 "Missing in Action (1984)"
 "Nightmare on Elm Street 5: The Dream Child, A (1989)"
```

Looks like the first Friday the 13th is clustered together with some classic scary movies, while part 2 and later get clustered together with some lower-tier scary movies.

The MovieLens dataset identifies movies by their title and year of release, so there's a helper function that lets you search for a movie by a case-insensitive substring of the title:

```
julia> IncrementalSVD.item_search(model, "godfather")
3-element Array{String,1}:
 "Godfather: Part III, The (1990)"
 "Godfather, The (1972)"
 "Godfather: Part II, The (1974)"
```

```
julia> IncrementalSVD.similar_items(model, "Godfather, The (1972)")
10-element Array{String,1}:
 "Godfather, The (1972)"
 "Godfather: Part II, The (1974)"
 "GoodFellas (1990)"
 "One Flew Over the Cuckoo's Nest (1975)"
 "Butch Cassidy and the Sundance Kid (1969)"
 "Silence of the Lambs, The (1991)"
 "Taxi Driver (1976)"
 "Apocalypse Now (1979)"
 "Saving Private Ryan (1998)"
 "L.A. Confidential (1997)"
```

So what are the features in these 25-dimensional feature vectors we're using? The first feature is a line through our ratings data in the direction it varies the most:

```
julia> IncrementalSVD.show_items_by_feature(model, 1)
3706-element Array{String,1}:
 "Nueba Yol (1995)"
 "Silence of the Palace, The (Saimt el Qusur) (1994)"
 "Windows (1980)"
 "Mutters Courage (1995)"
 "Loves of Carmen, The (1948)"
 "Diebinnen (1995)"
 "Aiqing wansui (1994)"
 "Chain of Fools (2000)"
 "White Boys (1999)"
 "With Byrd at the South Pole (1930)"
 "Roula (1995)"
 "Cheetah (1989)"
 "Uninvited Guest, An (2000)"
 "Elstree Calling (1930)"
 "Waltzes from Vienna (1933)"
 "Legal Deceit (1997)"
 "Kestrel's Eye (Falkens \U1667e9 (1998)"
 ⋮
 "Braveheart (1995)"
 "North by Northwest (1959)"
 "Godfather: Part II, The (1974)"
 "Silence of the Lambs, The (1991)"
 "Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)"
 "Saving Private Ryan (1998)"
 "Wrong Trousers, The (1993)"
 "Usual Suspects, The (1995)"
 "Sixth Sense, The (1999)"
 "One Flew Over the Cuckoo's Nest (1975)"
 "Casablanca (1942)"
 "Raiders of the Lost Ark (1981)"
 "Schindler's List (1993)"
 "Star Wars: Episode IV - A New Hope (1977)"
 "American Beauty (1999)"
 "Shawshank Redemption, The (1994)"
 "Godfather, The (1972)"
```

You might call this first feature "well known versus obscure movies". Unfortunately, you have to come up with interpretations of these features - they're just whatever your ratings data suggests:

```
julia> IncrementalSVD.show_item_feature(model, 2)
3706-element Array{String,1}:
 "Batman (1989)"
 "Superman (1978)"
 "Die Hard (1988)"
```

```
 "Indiana Jones and the Last Crusade (1989)"
 "Wayne's World (1992)"
 "Rocky (1976)"
 "Blues Brothers, The (1980)"
 "Ferris Bueller's Day Off (1986)"
 "Star Trek: The Wrath of Khan (1982)"
 "Untouchables, The (1987)"
 "Sneakers (1992)"
 "Raising Arizona (1987)"
 "Predator (1987)"
 "Good Morning, Vietnam (1987)"
 "Robocop (1987)"
 "Indiana Jones and the Temple of Doom (1984)"
 "Big (1988)"
 ⋮
 "Lady Vanishes, The (1938)"
 "Run Silent, Run Deep (1958)"
 "Shadow of a Doubt (1943)"
 "8 1/2 (1963)"
 "Affair to Remember, An (1957)"
 "Shall We Dance? (1937)"
 "Pawnbroker, The (1965)"
 "Laura (1944)"
 "Central Station (Central do Brasil) (1998)"
 "Bicycle Thief, The (Ladri di biciclette) (1948)"
 "General, The (1927)"
 "Dancer in the Dark (2000)"
 "Anatomy of a Murder (1959)"
 "Grand Illusion (Grande illusion, La) (1937)"
 "City Lights (1931)"
 "Yojimbo (1961)"
 "400 Blows, The (Les Quatre cents coups) (1959)"
```

The second feature is more obviously something like "art house versus mainstream blockbuster". The model also lets investigate similar users and user features just like we did with items, it's just not that interesting to look at the anonymized user ids in these public data sets. If you wanted to, say, create a dating site and suggest partners based on movie/book ratings, those user-centric functions might be more interesting to explore.

Finally, you can use the model to get predicted numeric ratings for specific users and movies. Let's look a little closer at one particular user:

```
julia> IncrementalSVD.user_ratings(rating_set, "3000")
100-element Array{(ASCIIString,Float64),1}:
 ("Gattaca (1997)",5.0)
 ("Groundhog Day (1993)",5.0)
 ("When Harry Met Sally... (1989)",5.0)
 ("American Beauty (1999)",5.0)
 ("Brazil (1985)",5.0)
 ("Twelve Monkeys (1995)",5.0)
 ("Time Bandits (1981)",5.0)
 ("One Flew Over the Cuckoo's Nest (1975)",5.0)
 ("Brothers McMullen, The (1995)",5.0)
 ("Dances with Wolves (1990)",5.0)
 ("Defending Your Life (1991)",5.0)
 ("Babe (1995)",5.0)
 ("Princess Bride, The (1987)",5.0)
 ("Caddyshack (1980)",5.0)
 ("Rock, The (1996)",5.0)
 ("Star Wars: Episode V - The Empire Strikes Back (1980)",4.0)
 ("Batman Returns (1992)",4.0)
 ⋮
 ("Total Recall (1990)",2.0)
 ("Mad Max (1979)",2.0)
 ("True Lies (1994)",2.0)
```

```
("Empire Records (1995)",2.0)
("Dark Crystal, The (1982)",2.0)
("Simon Birch (1998)",2.0)
("Romancing the Stone (1984)",2.0)
("Dirty Dancing (1987)",1.0)
("Star Trek VI: The Undiscovered Country (1991)",1.0)
("Mission: Impossible (1996)",1.0)
("Sudden Death (1995)",1.0)
("Jewel of the Nile, The (1985)",1.0)
("Flashdance (1983)",1.0)
("Blind Date (1987)",1.0)
("Blue Lagoon, The (1980)",1.0)
("Sphere (1998)",1.0)
("Grease 2 (1982)",1.0)
```

Among other things, this user seems to love movies about dystopian futures (Gattaca, 12 Monkeys, Brazil) and dislike movies that involve dancing (Dirty Dancing, Flashdance, Grease 2). So what might that user think of another dystopian future movie like Blade Runner?

```
julia> IncrementalSVD.predicted_rating(model, "3000", "Blade Runner (1982)")
4.257683865384008
```

And what would the same user think of Footloose?

```
julia> IncrementalSVD.predicted_rating(model, "3000", "Footloose (1984)")
2.810785878396833
```

How do I know that the results are any good?

Root-mean-square error (RMSE) is commonly used to measure predictions like these. It measures the average difference of a predicted rating to the actual rating over all actual ratings in some test set. All of the datasets loaded by this package hold out around 10% of the ratings for testing RMSE. After you've trained a model on a rating set, calling `IncrementalSVD.rmse(rating_set, model)` will calculate the RMSE on the test ratings, none of which were seen by the model during training.

You can also pass a specific rank (something less than the total rank of the model) to the `rmse` function to see what the RMSE would have been on the same model had you stopped adding features a little earlier. If you called `model = IncrementalSVD.train(rating_set, 50)`, you could then call `IncrementalSVD.rmse(rating_set, model, rank=i)` for any `1 <= i <= 50` to find the best rank in that range. If you find a rank with better RMSE, you can truncate your existing model by calling `IncrementalSVD.truncate_model!(model, rank)` to convert it into a model with smaller rank and better RMSE.

We don't round predicted ratings or ceiling/floor them if they go out of range in `IncrementalSVD.rmse`. This makes the RMSE returned a little worse than it could be. RMSE is a very informative statistic but optimizing it isn't the goal of this package.

What other datasets does this package load?

This package has built-in support for loading and cleaning three datasets:

- `load_small_movielens_dataset`: 1 million ratings from 6,000 users over 4,000 movies. Ratings are in the range 1 (lowest) to 5 (highest). You should be able to train 25 features in about 5 minutes on this dataset with an RMSE of around 0.91.
- `load_large_movielens_dataset`: 10 million ratings from 72,000 users over 10,000 movies. Ratings are in the range 1 (lowest) to 5 (highest). You should be able to train 15 features in about an hour on this dataset with a resulting RMSE of around 0.8. RMSE tends to keep going down as you add more features, so keep training if you have the time.

- `load_book_crossing_dataset` : 1.1 million ratings from about 278,000 users over about 271,000 books. Ratings are in the range 0 (lowest) to 10 (highest), but zeros mean that the user read the book but didn't rate it. This data comes from a crawl of the bookcrossing.com site and needs a lot of cleaning before you can do anything useful with it. During loading, we throw away any rating by a user who hasn't rated at least 30 books and given at least 6 distinct numeric ratings and we also throw away any rating of a book that hasn't been rated at least 30 times. In additions, we remap the ratings onto the range 1-6: everything from 0-5 becomes 1 and the rest of the ratings are scaled down by 4. This leaves about 190,000 ratings in the range 1-5 by 3900 users over 3900 books. You should be train a model with 30 features in about a minute with an RMSE a little over 1.5.