DATA 643 Project 3

Justin Hink

June 30, 2016

Introduction

For the third project in this course I'm sticking with a movie recommendation engine, again with a toy dataset that I've created. The new feature being added this week is a methods (and corresponding API) that returns a user's #1 recommended movie by leveraging an ALS selection algorithm.

Again, I've wrapped the engine has been wrapped in RESTful Flask web service to prove out how such a system might be stood up as a microservice in a larger environment.

Code

For a full listing, please visit the project's source code repo at:

https://github.com/jhink7/data643Proj3

Data

As mentioned, the dataset used for this exercise was completely fabricated. The values within do not represent anything real.

However, the structure of the data is similar to what you might see in a real world application. We have 3 tables and they are defined as follows:

Users

var	description
`id`	A simple integer id for a user
`name`	Name of the user

Movies

var	description
`movie_id`	A simple integer id for a movie
`title`	Name of the movie

Ratings

var	description
`id`	A simple integer id for a movie
`user_id`	An integer id of the user making the rating
`movie_id`	An integer id of the move being rated
`rating`	The rating for the movie (integers from 1 to 5 inclusive)

The data is stored in CSV files and loaded via pandas. Since the data is already in a fairly decent relational form, porting this dataset to SQLite or Postgres (or any RDBMS for that matter) would be fairly trivial.

Top Recommendation Engine Algorithm

The algorithm I've implemented uses alternating leas squares (ALS).

In this approach, we create factor matrices X and Y. X is a factor matrix where each movie is a column vector and Y is a factor matrix where each row represents a user. We have 2 unknowns and wil use an iterative approach to estimate both from each other.

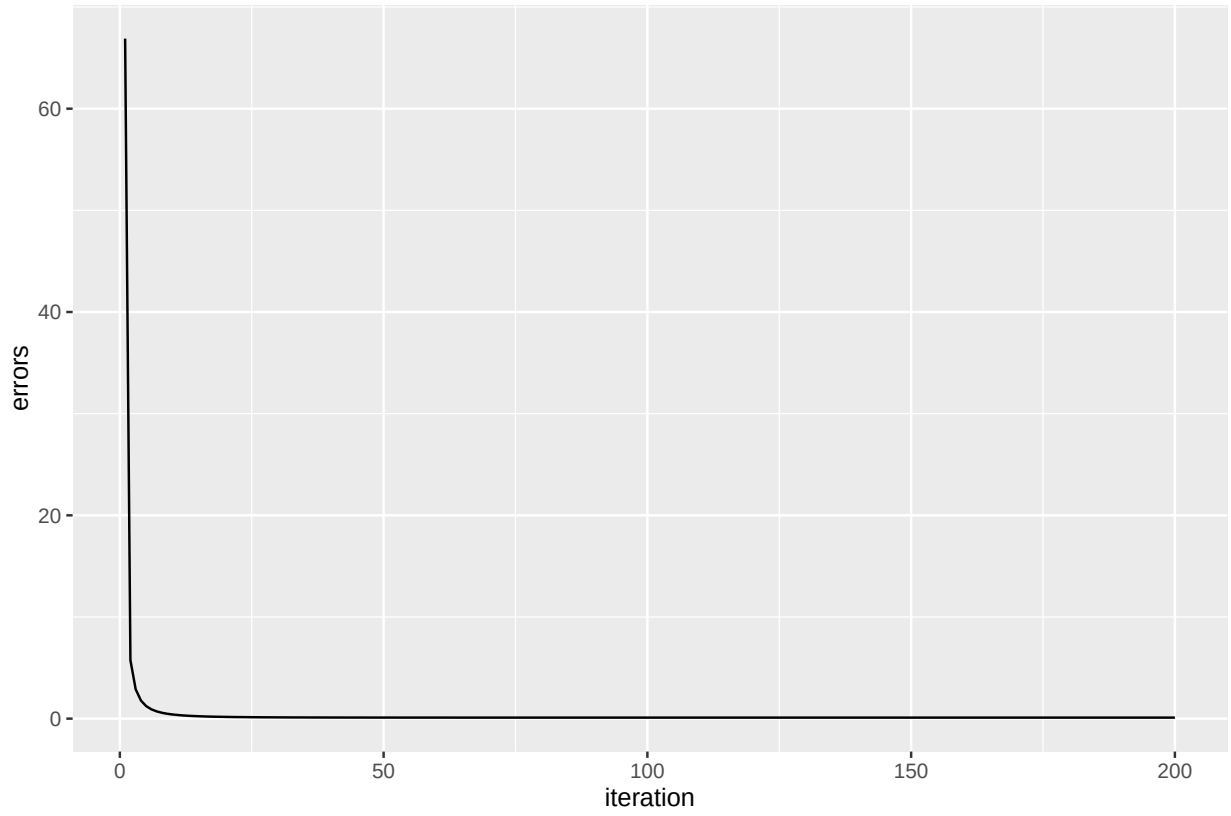
The heart of the algorithm can be seen in the following code snippet (which is inside the def generate_top_recommendation(self, target_user_id): function)

```
    # rebuild X and Y, decreasing our error (hopefully) with each iteration
    for j in range(self.n_iterations):
        # Update Y
        for i, Wi in enumerate(wt.T):
            Y[:, i] = np.linalg.solve(np.dot(X.T, np.dot(np.diag(Wi), X))
                                      + self.lambda_ * np.eye(self.num_factors),
                                        np.dot(X.T, np.dot(np.diag(Wi), Q[:, i])))
        # Update X
        for k, Wu in enumerate(wt):
            X[k] = np.linalg.solve(np.dot(Y, np.dot(np.diag(Wu), Y.T))
                                   + self.lambda_ * np.eye(self.num_factors),
                                    np.dot(Y, np.dot(np.diag(Wu), Q[k].T))).T

       error = self.get_error(Q, X, Y, wt)
       errors.append(error)
```

As the inline comments indicate, we're looking for the errors converge to a near 0 value. The number of iterations run is a tweakable value but the algo does seem to converge very quickly, with around 10 iterations usually doing a fairly solid job of providing answers similar to higher iteration runs.

In a 200 iteration run, the errors converge to 0 as follows (in other words, very quickly indeed):



Running the application

As with any Flask app, it's easiest to setup a virtual environment with a project local python and dependencies.

Steps to get up and running.

1) Create local env
2) pip install dependenies (flask, numpy, pandas)
3) run the run.py file (you may have to chmod a+x on run_flask.py)

If unfamiliar with flask, the following intro tutorial explains the above steps in more detail:

http://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

The application will start up on localhost on port 5000. Please see flask's documentation if the default port creates a conflict on your system.

Running Just The Recommendation Engine

If setting up the full flask app presents a hassle, the recommendation engine is flask agnostic and can be invoked from any other Python script (as long as you've included it correctly of course).

For example:

```
user_id=7
engine = RecommendationEngine()
user_exists, rec_movie = engine.generate_top_recommendation(user_id)
```

A Sample Call

Once flask is up and running, make calls as follows (note the /2/ represents the id of the user you're requesting the top movie rec for):

http://localhost:5000/rec-engine/api/v1.0/users/2/top-rec

The endpoint is an unauthenticated GET so typing the above URL into a browser should return data that looks like:

```
{
  "top_pick": "title15"
}
```

Possible improvements

This is quite obviously not a production ready system. The following list of things would need to be added before anyone would consider using this at scale:

1) Use a real database (not CSV)
2) Distribute iterations among nodes
3) Don't load data in-line from data store with http requests
4) Implement some sort of app to app authentication (ex mutual auth via SSL client and server certificates)
5) Optimize the tweakable parameters in the algorithm

Referernces

The cited work below was key in understanding the implementation details of ALS. The code there was borrowed, tweaked and semi-operationalized. (v, 2015)

v. Alternating Least Squares Method for Collaborative Filtering. 2015. URL: http://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/.