Weighted Alternating Least Squares (WALS) for Movie Recommendations)

Drew Hodun
SCPD

Abstract

There are two common main approaches to ML recommender systems, feedback-based systems and content-based systems. Feedback based systems look at user activity, examining explicit feedback such as Movie or product reviews and/or implicit feedback such as time spent viewing an item or show, total number of pageviews, etc. Content based systems look at item metadata, tagging, and sometimes the content itself, sometimes with a complex understanding of relevance of each of these items, to create mappings or clusters around similar items to recommend. Here I look at an extension of the most common feedback-based system, collaborative filtering. One of the most common ways to solve a collaborative filtering problem is using low-rank matrix factorization and the Alternating Least Squares method. I implement and examine the weighted version of this algorithm (WALS), particularly in the context of the Movielens movie review dataset. While WALS is commonly used for implicit datasets, I look at how it has meaning for explicit datasets.

1. Introduction

Collaborative Filtering is a common and powerful way for building feedback-based recommender systems. As opposed to content based systems looking at what we know about items and clustering them accordingly to recommend similar products to a user's activity history, we look primarily for a weigh to correlate user's activity with other users history to identify similar users *and* suggest items not in common between the two users that we have reason to believe will be good recommendations. This has important applications in e-commerce, media recommender systems, and more and more commercial applications. Collaborative Filtering does suffer from a 'cold start' problem, that is, until we see feedback and activity for either a new user or item, we have no way to profile it and recommend it to other users. There are various methods of solving this, such as using a combination of content based methods or making estimations of the latent factors that would eventually be solved once the item had more activity.

There are two many areas of feedback we can examine, *implicit* and *explicit* feedback. Explicit would include user reviews, very common in e-commerce and media applications such as Netflix.[1] Implicit would be time spent on various pages of a website. High quality explicit feedback tends to yield great results and can be implemented more easily in an algorithm. Implicit feedback is trickier, as we have to interpret when a user's activity implies preference vs non-preference or anti-preference. We have to build an idea of preference, as well as confidence in our assumptions of preference. Traditionally Collaborative Filtering models have tackled explicit feedback problems, because of the high quality data and the ease of achieving

great results. There are variety of methods of tackling implicit datasets, which will be discussed below.

Low-rank matrix factorization is a common way of solving this problem. There are various methods of computing the low-rank representations, including 'Alternating Least Squares' (ALS) and the weighted version (WALS). ALS is more a classic approach, while Weight Alternating Least Squares offers the data scientist a great deal of flexibility in modeling the data while still retaining the ease of use and low-compute cost inherent in Collaborative Filtering.

In this paper, I will implement, explore and compare the WALS approach to collaborative filtering, in particular for explicit-preference datasets as opposed to implicit-preference datasets where this is more common. I'll use the popular MovieLens Movie Recommendation Dataset.

2. Theory

2. 1 Low-Rank Matrix Factorization

Collaborative filtering through low-rank matrix factorization is a way of taking a sparse matrix of users and ratings, assuming a certain number of latent factors **(k)**, and factoring out a lower-rank representation of all the users and items. Can be roughly interpreted as 'genres' in Movielens dataset. This has a few advantages. It is computationally efficient for large datasets, and easily scales into the Millions of users and items. It is particularly easy to compute the suggested rating for a given item. Also, it has proven great at modeling complicated relationships between users and items with the ability to decompose each user and item to k-factors.

2.2 Alternating Least Squares

One of the most common methods of solving the Low-Rank Matrix Factorization problem is using Alternating Least Squares. First let's example the row factor and column factor matrices that we're looking to compute:

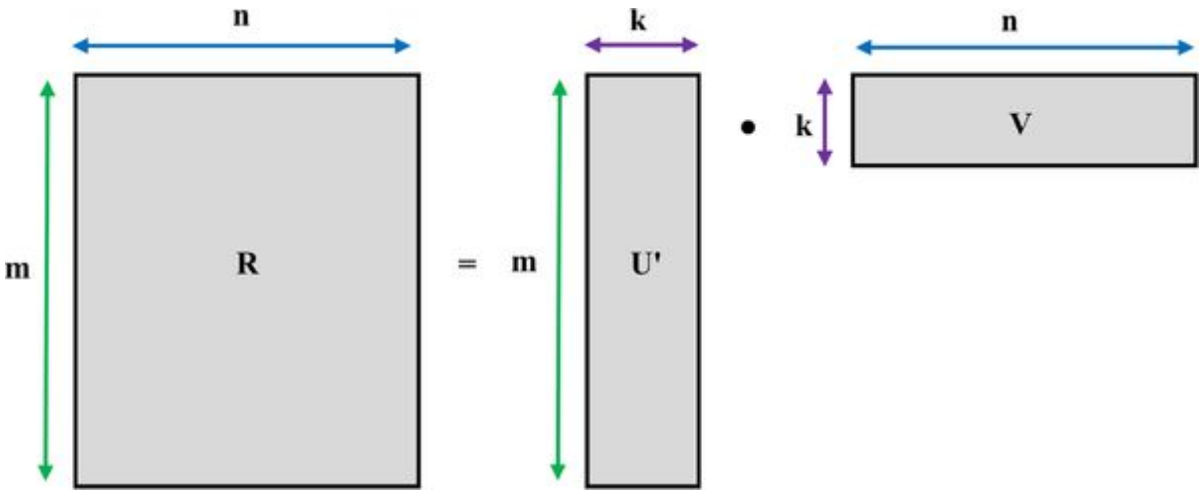

Figure 1. The goal of ALS is to factor out U' and V from sparse ratings matrix R

We have a sparse matrix R, sparse meaning 99% or more of the entries are 0, when we consider all known ratings of items **m** by users **n**. We can use the following loss function to set U' and V such that we lose the least amount of information possible:

$$L = \sum_{m,n}(R_{mn} - U_m^T \cdot V_n)^2 + \lambda \sum_m ||U_m||^2 + \lambda \sum_n ||V_n||^2$$

There are regularization terms on both the size of U and V to prevent overfitting. This can be easily solved when we recognize that setting the user factors constant results in a quadractic loss function that can be easily optimized. The process then becomes:

1. When either user-factors or item-factors is held constant, Loss becomes quadratic and we can then optimize, alternating rows and columns
2. Set row constant
3. Set derivative to 0 and solve
4. Repeat for constant column

$$\frac{\partial L}{\partial U_m} = -2\sum_n (R_{mn} - U_m^T \cdot V_n)V_n^T + 2\lambda U_m^T$$
$$0 = -(R_m - U_m^T V^T)Y + \lambda U_m^T$$
$$U_m^T(V^T V + \lambda I) = R_m Y$$
$$U_m^T = R_m Y(Y^T Y + \lambda I)^{-1}$$

This process usually converges in as small as 8-10 iterations, even for large sparse matrices. Now to approximate a rating, a fairly simple low-rank calculation:

$R_{mn} = U_m^T \cdot V_n$

Calculating all ratings for a user that can then be easily sorted for a recommenation amounts to multiplying:

$R_m = U_M^T \cdot V$

2.3 Weighted Alternating Least Squares

Weighted Alternating Least Squares (WALS) then becomes the weighted case of ALS with a Weight matrix W like below:

$$L^w = W \circ \sum_{m,n}(R_{mn} - U_m^T \cdot V_n)^2$$

W will take different meanings depending on the type of dataset and desired effect on the data. In particular for explicit datasets, using a linear scaling is common.

$w_{mn} = \omega_0 + f(c_m)$ unobserved weight + function of observed weight

$c_m = \sum_{m,n} if R_{mn} > 0$ sum of number of non-zero entries for each column (reviews per movie)

$f(c_m) = \frac{\omega_k}{c_m}$ linearly (explicit) scaling - scale down reviews of often-reviewed movies

This is the setup we will use with the MovieLens dataset to essentially normalize the signal for each movie. In essence, we are looking to decrease the signal for movies that are rated more often than movies that aren't, in the hopes that we're more likely to recommend a less seen movie. In general, this type of weighting allows us to more flexibly model preferences and gives us better results.[2]

For implicit datasets, more common with WALS.

$f = (\frac{1}{c_m})^e$ exponential(implicit) scaling

This is more common with web traffic analysis, or any dataset where we believe we have an exponential distribution of our signal to the perceived preference.

3. Previous Work

Yifan Hu, Yehuda Koren, and Chris Volinsky of AT&T Labs Research examined a variety of methods of modeling implicit feedback datasets with WALS. In particular they observed on set-top-TV player systems that it was helpful to have a minimum confidence in a given rating, represented by ω_o, as well as to calculate two separate magnitudes for preference and confidence levels in that preference. The also noted it was essential to take all user actions into account, including actions which indicated 0 preference.

TensorFlow conveniently includes a WALS matrix factorization class, which can also be simplified to ALS by not weighting either the rows or columns (tf.contrib.factorization.WALSModel).

4. Our Model and Data

In this section I describe the model that I implemented on the MovieLens dataset and, in particular, the hyper-parameter tuning required to achieve better results with WALS. This is of particular interest, as WALS converges relatively quickly (8-10 epochs), though running for longer often does achieve slight improvements in test accuracy (usually 0.1% or worse after 10 epochs).

4.1 Data Acquisition

There are several iterations of the MovieLens dataset, ranging in sizes, age, and cleanliness. I primarilyi worked with the more recent experimental 100k rating dataset that consists of 1,000 users, 1,700 movies, and 100k ratings. This dataset happens to be particularly clean and relatively normalized. Each user has at least 20 reviews. I loaded the data into Google's BigQuery cloud tool to run some basic observations about the distributions in the dataset:

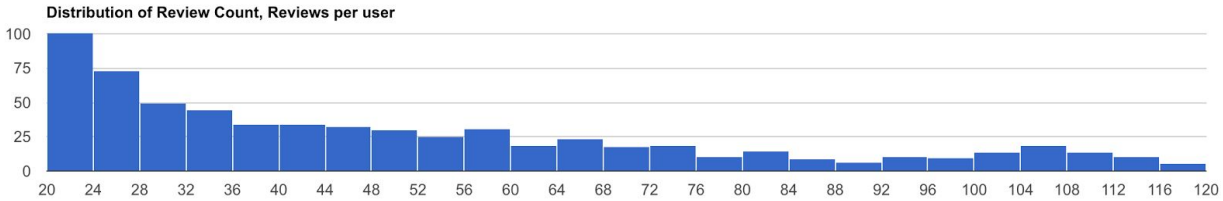

Figure 2. Reviews per user in MovieLens dataset

min	percentile1	percentile25	median	percentile75	percentile90	percentile99	max
20	20	33	65	148	245	448	737

Figure 3. Percentiles for Reviews per User Distribution

cluster	mean	sttdev
0-25	3.581	1.153
25-50	3.655	1.104
50-75	3.6	1.112
75-90	3.536	1.095
90-99	3.277	1.198

Figure 4. Mean and distribution of reviews based on clustered reviewers

min	percentile1	percentile25	median	percentile75	percentile90	percentile99	max
0.344	0.583	0.876	1.012	1.141	1.292	1.538	1.75

Figure 5. Distribution of individual Standard Deviations

In general what we find is a very clean, consistent datasets. There appear to be few if any anomaly or noisy users with wide spreads in reviews, abnormal means, etc. Similar analysis on the number of reviews per movie indicate there was a minimum number of reviews for each, in general, though there was a spread.

4.2 Pre Processing

Aside from analysis, the most notable thing done in pre-processing was removing any user and/or movie from the 10% test set that did not exist in the training set, as this would constitue a cold-start item and thus our model would have no real predictive power.

4.3 Implementation

The model was implemented using the WALSModel module in TensorFlow. In particular, the WALSModel requires sparse tensors as inputs for the ratings matrix, an 'unobserved weight' corresponding to ω_0 and either row and column weight scalars or entire vectors. In this use-case I chose not to normalize or alter the user weights and focus on movie weights instead, in particular using a linear scaling instead of an exponential scaling. The weights for a given column (movie) were scaled down as number of reviews per movie grew. In essence, a movie with 10 reviews would have a weight 10x the size of a movie with 100 reviews as such:

$$w_{mn} = \omega_0 + f(c_m)$$
$$c_m = \sum_{m,n} if R_{mn} > 0$$
$$f(c_m) = \frac{\omega_k}{c_m}$$

4.4 Hyper-Parameter Tuning

Since the WALS and ALS algorithms converge relatively quickly, a large part of the focus was tuning the various hyper-parameters for optimal performance, including regularization weight, unobserved weight ω_0 and column-factor weight ω_k. To achieve this, I setup the algorithm to run in Google Cloud Platforms' Machine Learning Engine on GPUs with 'hyper-parameter tuning' enabled. This is based on the Google Vizier service which enables me to specify a variety of hyper parameters to tune, their possible ranges and scale type (linear, logarithmic, reverse logarithmic), as well as the output metric (test RMSE) to optimize (either maximize or minimize), and then a number of total runs and maximum parallel runs to train and test the algorithm.

Vizier is a google-internal as well as now external through GCP black-box optimization algorithm, particularly helpful for tuning hyper-parameters. For studies under a thousand trials, Vizier defaults to using Batched Gaussian Process Bandits to manage explore / exploit trade-offs, an appropriate fundamentally Bayesian in nature.[3]

```yaml
trainingInput:
  scaleTier: CUSTOM
  masterType: standard_gpu
  hyperparameters:
    goal: MINIMIZE
    maxTrials: 300
    maxParallelTrials: 15
    params:
      - parameterName: num_factors
        type: INTEGER
        minValue: 1
        maxValue: 100
        scaleType: UNIT_LINEAR_SCALE
      - parameterName: regularization
        type: DOUBLE
        minValue: 0.0001
        maxValue: 10000
        scaleType: UNIT_REVERSE_LOG_SCALE
      - parameterName: unobserved_weight
        type: DOUBLE
        minValue: 1e-15
        maxValue: 5
        scaleType: UNIT_LOG_SCALE
      - parameterName: col_weight_factor
        type: DOUBLE
        minValue: 1
        maxValue: 20000
        scaleType: UNIT_LINEAR_SCALE
      - parameterName: epochs
        type: INTEGER
        minValue: 20
        maxValue: 90
        scaleType: UNIT_LINEAR_SCALE
```

Figure 6. Hyperparameter search space spec

4.5 Model Training Results

After identifying best hyper-parameters for WALS as well as best for ALS, the results were plotted and WALS showed a 3.1% improvement over ALS from an RMSE of 0.97062 to 0.94064.

Figure 7. Train / Test error for WALS and ALS.

This improvement of 3.1% is good, and gets us closer to the well known 0.83 value of unweighted collaborative filtering on the 20 Million ALS number. I did not run the experiment on the 20 Million rating dataset primarily because of time and cost related to hyper-tuning. The dataset is too large to fit into memory for K80 GPUs and takes 1.0-1.5 minutes per epoch on CPU and requires a large amount of memory.

I observed an interesting condition where either the column factor weight or the regularization value would always reach the edge of the hyperparameter space. Growing the search space would always result in one or the other growing further while the unobserved weight was trending towards 0. Ultimately the RMSE was stable the resulting predicted ratings were sensical, so an equilibrium was reached, there was simply an interesting scaling of the solution space.

The final hyper-parameters for the WALS use case are below. Some of the factors are fairly large, due to this scaling condition I witnessed.

Training output


```
{
 "completedTrialCount": "300",
 "trials": [
  {
   "trialId": "190",
   "hyperparameters": {
    "epochs": "77",
    "col_weight_factor": "19688.3375213518",
    "num_factors": "94",
    "unobserved_weight": "8.3349819188828016e-08",
    "regularization": "1914.2944448849639"
   },
   "finalMetric": {
    "trainingStep": "1",
    "objectiveValue": 0.940646231174
   }
```

Figure 8. Final Hyper-parameter selection values

5 Results Analysis

The results showed a 3.1% improvement, though greater improvement could be hoped for in a larger, more disparate datasets such as the 20 Million dataset or something more noisy. We did show that WALS, an algorithm often used in implicit-feedback collaborative filtering scenarios, still showed an improved in this explicit-feedback scenario.

The previous analysis of the data showed a fairly normalized dataset. Comparing the 20 million rating dataset to the 100k datasets, its clear that the is a larger spread of movies, i.e. there are more popular movies that take a majority of the reviews and a much longer long-tail of rarely reviewed movies. In particular, looking at the variance to mean ratio (VMR), also called dispersion index, we see there is more likelihood for improvement with the 20 Million dataset with a 12.7k VMR compared to 0.1k VMR.

stddev	mean	min	percentile1	percentile25	median	percentile75	percentile90	percentile99	max
80.384	59.453	1	1	6	27	80	169	378	583
3,085.818	747.841	1	1	3	18	205	1,306	14,396	67,310

Figure 9. Dispersion Ratio of 100k dataset and then 20M dataset, 0.1k to 12.7k

6 Discussion

We've used and proven out WALS as a potentially more powerful alternative to straight ALS based collaborative filtering, even for explicit-feedback datasets. The linear scaling still normalizes the signal and improves RMSE on the test set. This could potentially improve the results more than the 3.1% seen with a more 'real world' or noisy, skewed dataset. Next steps would be to try on the MovieLens 20 Million rating set. WALS provides a lot of flexibility in modeling user and item preference. For example, we could experiment with weighting users based on individual mean and variance, also weighting down users that we see as anomalous or unreliable, or perhaps a given userid that is a mix of multiple users. We can still enjoy the benefits of collaborative filtering, including the quick convergence and the ease of predicting future ratings with a simple dot product, while imparting additional opinions onto the data.

Next steps would also be to implement WALS on an implicit dataset and compare results, for example examining click stream data, since WALS is more common in these use cases.

References

1. J. Bennet and S. Lanning, "The Netflix Prize", *KDD Cup and Workshop*, 2007. www.netflixprize.com.

2. Hu, Yifan, Koren, Yehuda, and Volinsky, Chris. "Collaborative Filtering for Implicit Feedback Datasets - Yifan Hu." http://yifanhu.net/PUB/cf.pdf. Accessed 15 Dec. 2017.
3. Daniel Golovin, Benjamin Solnik, Subhodeep moitra, Greg Kochanski, John Karro, and D. Sculley. "Google Vizier: A Service for Black-Box ... - Research at Google." Accessed December 15, 2017. https://research.google.com/pubs/archive/46180.pdf.