Collaborative filtering via matrix decomposition in mlpack

Siddharth Agrawal siddharth.950@gmail.com
Ryan R. Curtin ryan@ratml.org
Sumedh Ghaisas sumedhghaisas@gmail.com
Mudit Raj Gupta mudit.raaj.gupta@gmail.com

Recommendation systems are particularly useful in the real world, notably in the fields of e-commerce and behavior prediction. Because of this, there are several new techniques for recommendation systems every year. Often, these techniques are based on matrix factorization; for instance, a matrix decomposition model won the Netflix prize in 2009 (Koren et al., 2009). The key to this approach is the representation of the known ratings as an incomplete user-item rating matrix $M \in \mathcal{R}^{n \times m}$, where M_{ij} (if known) is the rating of item j by user i. The matrix M is then assumed to be low-rank, and decomposed: $M = WH$.

The wide, ever-increasing variety of matrix decomposition techniques has led to a fractured software landscape, with some techniques being widely available and others only available in one or two places. Further, recent attempts to unify this landscape, such as **PREA** (Lee et al., 2012) and **recommenderlab** (Hahsler, 2011), which both provide APIs for the development of new techniques, tend to be slow in practice. This situation has motivated us to develop an efficient matrix factorization framework for collaborative filtering, `cf`, as part of **mlpack**, a C++ machine learning library (Curtin et al., 2013). This framework primarily targets two types of users: data scientists who want to quickly apply matrix factorization techniques, and machine learning researchers who want to implement their own techniques.

Users who are interested in applying matrix factorization may do so easily through **mlpack**'s command-line interface; an example is below:

```
$ cf -i movielens-1m-train.csv -a RegSVD -rank 20 -o recommendations.csv
```

Researchers who wish to implement their own matrix factorization algorithms on top of the existing framework in **mlpack** have several options open to them:

- If the factorization technique is alternating (i.e. update W, update H, repeat until convergence), only two functions must be supplied: `WUpdate()` and `HUpdate()`.
- If the factorization technique is more general, only an `Apply()` function must be supplied, which takes M as input and returns W and H as output.
- The existing factorization techniques (`NMF`, `RegularizedSVD`, `QUIC_SVD`, `SVDBatchFactorizer`, `SVDIncompleteIncrementalFactorizer`, and `SVDCompleteIncrementalFactorizer`) may be adapted or tuned.

The implementations available in **mlpack** are efficient; they outperform other matrix decomposition implementations. For example, on our test system, the **mlpack** regularized SVD implementation decomposes the MovieLens-1M dataset (1 million ratings) in approximately 16 seconds with an RMSE of 0.8838, whereas with the same parameters, **PREA** takes approximately 335 seconds with an RMSE of 0.8850. A tutorial may be found at `http://www.mlpack.org/doxygen.php?doc=cftutorial.html`; the code may be found at `https://github.com/mlpack/mlpack/` and `https://mloss.org/software/view/152/`.

References

R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, and A.G. Gray. MLPACK: A scalable C++ machine learning library. *The Journal of Machine Learning Research*, 14:801–805, 2013.

M. Hahsler. recommenderlab: A framework for developing and testing recommendation algorithms. 2011.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

J. Lee, M. Sun, and G. Lebanon. PREA: Personalized recommendation algorithms toolkit. *The Journal of Machine Learning Research*, 13(1):2699–2703, 2012.