Factorization Techniques in Non-negative Matrix Factorization

## Zack Tillotson / December 2010

# Abstract

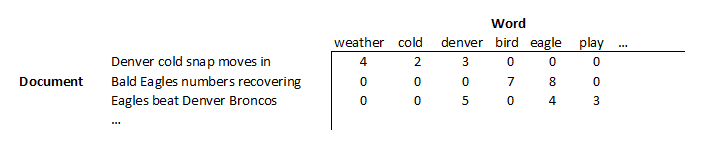
Non-negative Matrix Factorization is a powerful machine learning technique used to model and describe complex data structures. Creating the model involves factoring an input matrix V into two approximation matrices, W and H. Several algorithms to find these approximation matrices are discussed. Each is also implemented and the results on several synthetic and real world data sets are presented.

# Background

Non-negative matrix factorization (NMF) is a technique which has gained prominence in the last decade as a powerful machine learning tool. NMF works by extracting the prominent features of a dataset and describing each data point as a weighted combination of each of the features. This approach is not only powerful in its representational ability, but also because the underlying structure of the feature based model can be intuitively understood and applied. This contrasts with other classification techniques such as Support Vector Machines, Neural Networks, and Principle Component Analysis which are largely black boxes in which the model works, but no deeper understanding of the data is gained by analyzing the model itself.

NMF has been used in a wide variety of tasks, such as classifying images [1], grouping text documents for text mining [1][3], spectral analysis [3], and collaborative filtering in reviewing movies [4]. This wide variety of applications, with many different environments and constraints, shows that NMF is a powerful tool.

For an example of how NMF works consider a text mining application in which many text documents have been broken down into a list of the non-trivial words in that document and the number of times each word appears. This can be converted into a matrix where each row is a document and each column is one of the non-trivial words. The value of an element in the matrix is the number of times that word appears in that document. For example:

Figure : Document Word Counts

In NMF an input matrix V of this sort is factorized into two new matrices - the weights matrix W and the features matrix H where each value in W and H must be non-negative. When these two new matrices are multiplied back together they approximate the original input matrix. The factorization takes the form:

Figure : Non-negative Matrix Factorization Equation

The features matrix describes the features of the data set, in our example our features might be document themes such as "weather" or "sports", but the features could be any other descriptions of the documents as well, such as "long" or "biographical". The values in this matrix describe how much that word is important to describing a feature, so the elements in a "sports" feature would have high values for the words "play", "ball", and "teebow" and low scores for the words "weather" and "madonna".

The weights matrix describes each document by scoring each it with regard to each of these features. A lengthy article about politics should then get a high score for the "political" and "long" features and a small score for "Sports". These scores are all constrained to be non-negative, this additive representation enforces that each element is described as the sum of a set of parts since no part can be cancelled out by another part.

In order to create the perfect factorization it would be easiest to assign each document one feature which describes it exactly, but by having a small number of features that each document must be assigned to, similar documents are categorized together and themes can be discovered. This understanding via generalization is a consistent theme in machine learning.

In an actual NMF implementation the features are not given, instead of describing a feature and then scoring each item on it, the input matrix is blindly factorized into the two sub matrices such that the error when they are unfactored is minimized. The features and weights matrices are then interpreted as describing important features of the data set, but obviously don't have explicit descriptions. One common way humans interpret a feature is to look at the highest weighted objects relating to it. In the text mining example, the highest scoring words in the features matrix would show which words compose that feature and the highest scoring articles in the weights matrix would show which articles are best described by the feature.

Simply being able to understand the deconstructed model is not the only way in which NMF is a powerful tool, though. NMF can be used to approximate missing members of an incomplete input matrix. For example, consider a movie suggestion engine in which users rate movies they have seen and then wish to be provided with probable scores for unseen movies. The form of this input would be a sparse matrix with a row for each user and a column for each movie. Given a large number of movies in a typical movie database, most users will not have seen even a small fraction of the movies, so very few of the elements will have scores.

If NMF is used on this input, the input matrix can be factored into the features and weights matrix using just the few scores that do exist. Doing the unfactoring will then will provide a full matrix with values in every element. These elements can be viewed as guesses for how the person will score that movie.

# Algorithms

Constructing the model can be viewed as an attempt to find the weights and features matrices such that V ≈ WH. Most algorithms work by minimizing the difference between V and MH. The "difference" is often defined as the squared Euclidean distance or Mean Squared Error (MSE), both being monotonically equal. We will consider the MSE for our purposes in this paper.

The ideal way to factor the input matrix is into the sub matrices such that the difference is minimized is still an open question in many regards. Finding the global optimal factorization is considered impractical, and so these algorithms are used to find a local optimal factorization [2]. Several algorithms are popular, including Multiplicative Update (MU) [2][3], Stochastic Gradient Descent (SGD) [2][3][9], and Alternating Least Squares (ALS) [3][5]. Each of the algorithms will be described and analyzed in the context of NMF.

## Multiplicative Update

Multiplicative Update was first described by Lee and Seung in 2001 and is popular because of its ease of implementation and the quality of results in many situations. Starting from a random starting factorization, MU works by repeatedly updating the weights and features matrices by a factor based on the quality of their approximation of the original matrix. This update rule as given by Lee and Seung has been shown to converge on locally optimal solutions [1][3]:

The time complexity of the MU method = <Number of iterations until converged> \* <Cost per iteration>. Each update consists of 6 matrix multiplications which cost O(n3), and so the cost per iteration O(n3)[3]. The number of iterations I is not a hard value and depends on the initial state and problem instance, but has been shown to converge less quickly than other solutions [3][5][10]. The final cost C of the MU method then is:

In very large matrices the relatively complex matrix operations used in this algorithm can make implementation prohibitively inefficient. In addition the MU method does not work well with sparse matrices. In sparse matrices, the relatively small amount of real information is overridden by the large amount of missing information and the quality of the solution is poor.

## Stochastic Gradient Descent

Another useful algorithm is the application of a simple piecewise gradient descent strategy. In this method, the approximation matrices are created piecemeal, one feature at a time. For any feature, the weights and features matrices initializes a random vector and at each iteration the AU method moves some distance in the direction of the derivative of the difference function, eventually moving to the local optimum. At this point that feature is considered done and a new feature begins being learned, each time improving on the factorization approximation. The additive update rule can be shown as [3][9]:

*For each feature f*

*For each non empty element Vij of V*

One item to note is the learning rate r. If r is too small the algorithm will be slow to converge, repeatedly making many small movements in approximately the same direction. If r is too large the algorithm can "overshoot" the desired local minimum. If this happens the local minimum can either be lost leading to a situation where convergence is not guaranteed, or the new state can repeatedly jump back and forth over the local minimum, each time getting closer but at a slow rate. Because of this, finding a workable learning rate can be prohibitively costly in and of itself.

This method does have several positive attributes, including ease of implementation and scalability. The piecemeal construction of the models allows for piecemeal handling of the input data which allows the method to scale very well. SDG has been implemented on very large datasets, including the Netflix dataset which contains over 100 million movie ratings on about 18 thousand movies by 480 thousand users [4][9].

The time complexity of this SGD algorithm = <Number of features> \* <Number of iterations> \* <Cost per iteration>. Each iteration consists of iterating through each value of in the input matrix and updating the current feature's weights and features vector in its direction. The cost per iteration therefore is nominally = nm, but in sparse matrices the number of known elements is much less than the total number of possible elements. The total cost then is:

## Alternating Least Squares

Our final method of solving the NMF problem is Alternating Least Squares. Motivated by the observation that in the equation V=WH, given that V is constant, if either W or H is also held constant the equation becomes convex and so can be optimized, in ALS, each of these approximation matrices is optimized in turn. First one of the approximation matrices is set as constant and the other is optimized using the least squares method, and then the two matrices are switched, with that matrix set as constant and the other optimized. This two action iteration continues until a convergence point is found. The update rule for this algorithm is [3][11]:

The time complexity of ALS is = <Number of iterations> \* <2 least squares per iteration> \* <Cost per least square>. The cost per least square is 3 matrix multiplications for a total cost of O(n3), so the total cost C is:

This method by passes several limiting aspects of the previously discussed methods. It not only has a smaller cost than MU, it has been shown to tend to converge more quickly than MU. Also, it does not have the stochastic learning rate limitations of SGD, bypassing the need to discover a proper learning rate.

# Experiments

In order to show experimentally the differences between each of these methods of creating the NMF model, each has been implemented in Java and tested on a variety of data.

The MU and ALS implementations use the JAMA matrix operations library developed by NIST and the University of Maryland [12] to efficiently compute matrix operations such as matrix transposition, multiplication, and inversion. For the SGD learning rate I use a value of .001 for all the situations, this was chosen experimentally and seems to work best.

The datasets chosen to test the algorithms vary widely. First a set of synthetic datasets were constructed with sizes 250x150 and 2500x1500, with the values a positive Gaussian distribution centered at a corner of the matrix. Next, blog documents from the DailyKos.com document word dataset at the University of California at Irvine dataset repository [8] was used, this dataset consists of about 7000 words and 3500 documents which were reduced to the number of times each word occurs in each document. This is a sparse real world dataset, about 2% of the elements have non zero values. Finally, also included is a portion of the very large Netflix Prize dataset, consisting of about 21 million movie reviews done by 100 thousand users on 18 thousand movies.

Figure : Small Synthetic Data, Time To Build Model in Seconds vs Number Of Features

Figure : Large Synthetic Data, Time To Build Model in Seconds vs Number Of Features

Figure 5: KOS Blog Document Word Data, Time To Build Model in Seconds vs Number Of Features

Figure : Netflix Movie Rating Data, Time To Build Model in Seconds vs Number Of Features

As shown in these figures, MU tends to converge more slowly than either SGD or ALS. SGD scales the best, with the best times on the large, real world data sets (MU and ALS would not finish on datasets of this size without more optimization).

# Conclusion

The algorithm used to create the approximation matrices in NMF have a large impact on its useability. Each algorithm discussed in this paper worked well on small datasets, but the matrix multiplication based approaches MU and ALS tend to not scale as well as SGD. SGD on the other hand has the non trivial problem of requiring an intelligent learning rate to converge well.

In future work with these algorithms I will implement improved versions of SGD and ALS. Starting SGD with a large learning rate and lowering it over time in the same manner as Simulated Annealing would lead to a more adaptable application, and implementing a piecemeal or distributed ALS algorithm could lead to an improvement in its scalability issues.

# References

1. **Learning the parts of objects by non-negative matrix factorization**, Daniel D. Lee & H. Sebastian Seung, 1999 - <http://www.seas.upenn.edu/~ddlee/Papers/nmf.pdf>
2. **Algorithms for Non-negative Matrix Factorization**, Daniel D. Lee & H. Sebastian Seung, 2001 - <http://hebb.mit.edu/people/seung/papers/nmfconverge.pdf>
3. **Algorithms and Applications for Approximate Nonnegative Matrix Factorization**, Michael W. Berry, et al., 2006 - <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.1963&rep=rep1&type=pdf>
4. **The BellKor Solution to the Netflix Grand Prize**,Y. Koren, 2009 - <http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf>
5. **Non-Negative Matrix Factorization Based on Alternating Non-Negativity Constrained Least Squares and Active Set Method**, Hyunsoo [Kim](http://smartech.gatech.edu/browse?value=Kim,%20Hyunsoo&type=author) & Haesun [Park](http://smartech.gatech.edu/browse?value=Park,%20Haesun&type=author) , 2007 - <http://www.cc.gatech.edu/~hpark/papers/simax-nmf.pdf>
6. **Programming Collective Intelligence**, Toby Segaran 2007
7. **On the Convergence of Multiplicative Update Algorithms for Non-negative Matrix Factorization**, Chih-Jen Lin, 2007 - <http://www.csie.ntu.edu.tw/~cjlin/papers/multconv.pdf>
8. **Bag of Words Dataset**, UC Irvine machine learning dataset repository - <http://archive.ics.uci.edu/ml/datasets/Bag+of+Words>
9. **Netflix Update: Try This at Home**, Simon Funk, 2006 - <http://sifter.org/~simon/journal/20061211.html>
10. **Projected Gradient Methods for Non-negative Matrix Factorization**, Chih-Jen Lin, 2007 - <http://www.csie.ntu.edu.tw/~cjlin/papers/pgradnmf.pdf>
11. [**Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation**](http://www.amazon.com/Nonnegative-Matrix-Tensor-Factorizations-Applications/dp/0470746661/ref=sr_1_1?s=books&ie=UTF8&qid=1291577927&sr=1-1)**,** Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, & Shun-ichi Amari, 2009
12. ***JAMA* : A Java Matrix Package**, <http://math.nist.gov/javanumerics/jama/>