# Matrix Decompositions in Data Analysis

# Project 2 Report on Non Negative Matrix Decomposition

**Student ID:322079 Date: 13th February 2021**

**Abhishek N. Singh**

**absingh@student.uef.fi**

## Introduction

The project consisted of 3 tasks, all of which helped me understand non-negative matrix factorization and its applications in data science. Scripts were already provided for each of the tasks to a great extent, and I needed to essentially fill in the missing components.

## Task 1

In task 1, three versions of NMF algorithm were implemented, the first on alternating least squares, second based on multiplicative updates as proposed by Lee and Seung, and third by Oblique Projected Landweber (OPL) updates. The alternating least squares method was implemented in the method:

```
def nmf_als2(A, W, H)
```

was implemented by using the pseudo-inverse formula of matrix multiplication as has been discussed in the lecture and using python operator '@' for matrix multiplication:

```
H = np.linalg.pinv(W) @ A
```

Then the negative values in the H matrix is replaced by 0,

```
H[H < 0] = 0
```

before using this H to obtain W in a similar way.

```
W = A @ np.linalg.pinv(H)
```

Then W is also set to 0 for entries which are less than 0. The initial values of W and H which are the factors of matrix A are chosen at random. This process is repeated for 300 iterations such that for each iteration we look at the error which is the frobenius norm of A - WH . The errors are saved for subsequently plotting the error drop vs iterations.

There is one other method def nmf_als(A, W, H) which has been implemented to see the least square solution and works almost the same way as nmf_als2 method, although this has not been evaluated any further. Yet another method implemented is the accelerated-Hierarchical Alternating Least Squares method for NMF:

W, H, n_iter = non_negative_factorization(A, n_components=2, init='random',solver='cd', random_state=0, verbose='int'),

which is an accelerated version of the alternating least squares method , but this has not been evaluated any further in the exercise, and has been provided additionally in exploratory perspective.

The Boilerplate code was modified for the implementation of OPL updates as in the OPL:
def nmf4OPD(A, k, optFunc=nmf_opl, maxiter=300, repetitions=1)

The update is done both to the W as well as the H matrix for which the step function neta and the corresponding gradient function is calculated by the same way calling def nmf_opl(A, W, H). In this function notice how I use python matrix multiplication ‘@’ operator for the purpose:

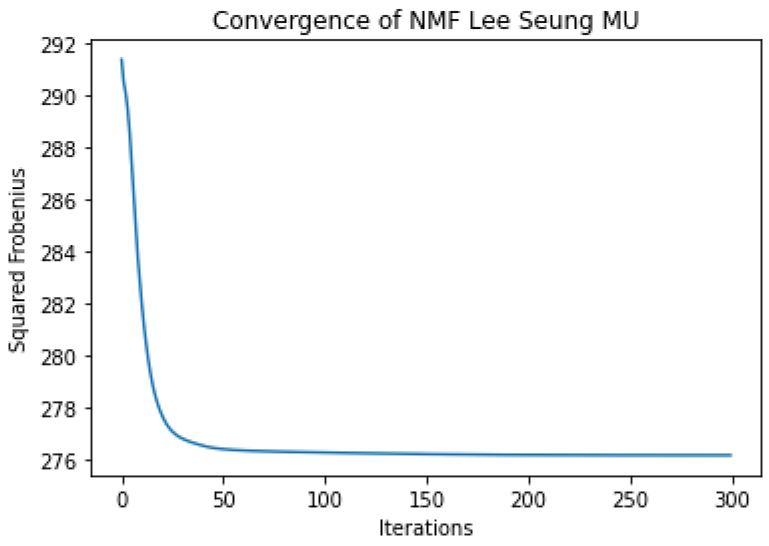
```
 NetaH = np.diag(1 / np.sum( (W.T @ W), axis = 1)) #Doing Row Sum and getting
  #the reciprocal and then creating a diagonal matrix
  G = ( W.T @ W @ H ) - ( W.T @ A ) #This is the gradient matrix
  H = H - ( NetaH @ G ) #Updating H by NetaH steps times G
  # Set negative elements of H to 0 as we have been given permission for this
  H[H < 0] = 0
```

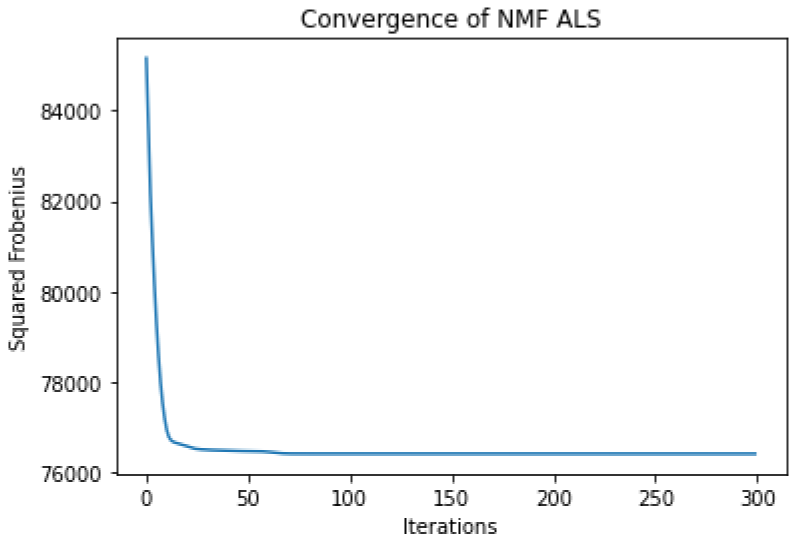
Notice that the same function can be used for updating W as well as H, by simply passing different values which calling through nmf4OPD function:

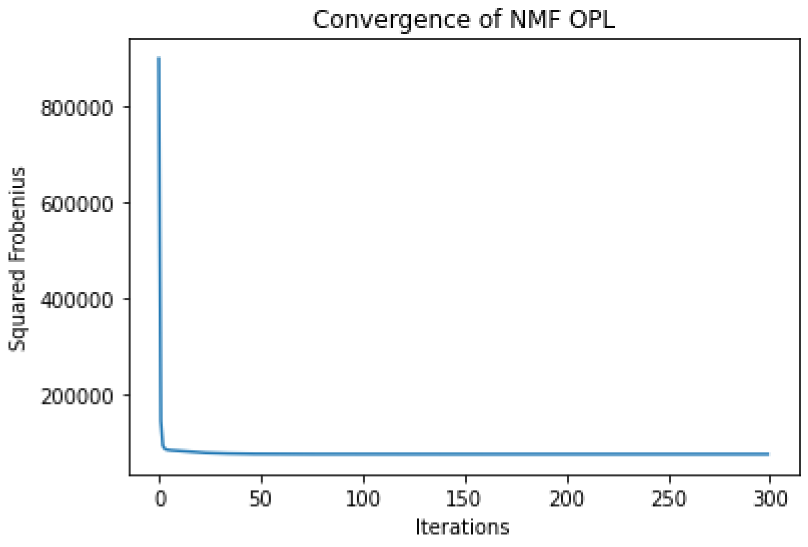
```
H = optFunc(A, W, H)
W = (optFunc(A.T, H.T, W.T)).T
```

In the Lee & Seung multiplicative update, a non-zero small delta value of 0.0001 is used in order to prevent a possible division by zero. The algorithm is implemented exactly the same way as the formula taught in the lectures, where numerator and denominator are updated and then each element of H as well as W are updated in a nested for-loop.

All of the three forms of NMF factorization methods are called using k=20. The default iteration of 300 was changed to 3000 for instance in some cases, but there was not much of a difference. The method of Multiplicative Update by Lee and Seung gives the least error
in the Frobenius norm between the matrix A and its factors W and H, and so it can be preferred. However, the steepest descent for convergence was seen in OPL. Below in the 3 plots are the drop in the Frobenius norm in terms of error with respect to increase in iteration, using the 3 NMF factorization methods. All of the methods seem to converge very quickly, and that increasing the iterations does not seem to further reduce the errors much significantly.


Convergence of NMF Lee Seung MU
Squared Frobenius
Iterations
292
290
288
286
284
282
280
278
276
0
50
100
150
200
250
300


Convergence of NMF ALS
Squared Frobenius
Iterations
84000
82000
80000
78000
76000
0
50
100
150
200
250
300


Convergence of NMF OPL
Squared Frobenius
Iterations
800000
600000
400000
200000
0
50
100
150
200
250
300

The convergence of the Lee and Seung Multiplicative update seems to be leading to lower frobenius norm error. 76243.32315479756 for OPL method,
276.46586723679246 for Lee and Seung method,
76472.67982627494 for alternating least squares method, all after 300 iterations.

Thus, we can conclude that the Lee and Seung method of multiplicative update seems to be performing better for 300 iterations for the data we have. Thus, given that the frobenius norm has the least error for Lee and Seung Multiplicative update method, the reconstruction error using W and H to get back A would also be least using this method.

***However, the best method to judge if the matrix is decomposed into its relevant factors is by first multiplying two matrices W and H to get A, and then using these algorithms on A matrix to see how much of W and H are we able to recover. Thus we will be then able to assign a relevant statistical match score for recovery. This will also help the algorithms to determine the best 'k' value that serves the best for factoring a dataset matrix.***

**Task 2**:

In task 2 what was needed is to do a normalization of the dataset matrix A for which the function was already provided:

```
B = A/sum(sum(A))
```

Then Lee & Seung's Multiplicative Update method was applied with k=20. I particularly liked the OPL method of gradient descent and so applied it to B and obtained the first top 10 terms of H leading to following possible topic:

k=20
Topic: The topic over here seems to correspond to 'Study, Research, Tests, Production certainty of Food Science in which Steve was involved'.

K = 5
The topic could be "Watch the 3rd sign on final night in a ride before getting hit by fan which can cause flame"

k=14
The topic could be "Bike ride by while and black members"

K = 32

The topic could be "Research starting with letter W at a Unit has satellite signal on radio at green field"

K = 40
The topic could be "Canadian and American capital jet cost at Winnipeg during fall term and their insurance"

With decrease in k value, the frequency of each word increases. The best k rank to be chosen is very subjective. Typically, what can be done is that we can remove some elements from the A matrix and then calculate its factors W and H. Then, we can see which of the W and H obtained via various values of k, help us get back the missing value in A. This would be computationally very expensive. However, if the ground truth has been provided, we can look for that k value for which the ground truth can be obtained using the k that suits the best. **Alternatively, as stated earlier, another way could be to have already W and H matrices using which we can obtain A by the product of W and H, and then apply various decomposition algorithms to matrix A to see which one yields us W and H and for what value of k. Clearly, since we know the value of k for W and H in that case, we should expect to get the best result of decomposition using the k that W and H had at its columns and rows respectively.**

The method was repeated for various values of k using generalized K-L divergence optimization for NMF. The results change again and they do not match with what was obtained using OPL. We can’t even comment whether the results are better or worse, as long as we do not compare the results with the ground truth. The same holds true even if we use euclidean distance i.e., we cannot comment if increasing k or changing methods leads to a better result without comparing the result with ground truth. Apparently, comparing the result with ground truth is covered in task 3.

**Task 3.**

To get pLSA, here first we applied Kullbak Leibler divergence to get the W matrix
model = NMF(n_components= 20, solver='mu', init='random', beta_loss='kullback-leibler', random_state=0)
```
W = model.fit_transform(B)
```

Then the W matrix was normalized:
```
Wnorm = W/sum(sum(W))
```

zscore(A) was used to normalize the A data to z-score, and then SVD was calculated.

```
U, S, V = svd(Z, full_matrices=False)
```

Then the top 20 factors were chosen.

```
V = V[0:20,:] #Taking the 20 topics
```

And then PCA was done:

```
KL = np.matmul(Z, V.transpose())
```

The later steps involved doing clustering using Kmeans and first fitting to the original data A matrix, with 20 cluster centres.

Then normalized mutual information NMI was calculated using the ground truth data and the script that was provided. The NMI for the original matrix was calculated to be 0.9473 .

The same was repeated to get NMI for the data using k=20 PCs (principal components) for which NMI was 0.9256

A lower NMI indicates better performance and thus PCA did improve the results. The NMI score dropped even further when pLSA was used and the score was 0.8184 .

Different values of k were used to factorize A matrix using pLSA, and the NMI scores were obtained. The best NMI score obtained was for k=10 with a score of 0.7066 .

Clearly the clustering by pLSA performed much well compared to PCA or raw data A matrix clustering. The reason being that pLSA does a probability distribution based topic prediction.

## Conclusion

We went through non-negative matrix factorization and saw different ways this can be done. We also examined as to how to predict the best k that suits matrix factorization, and noted that judging a method just by the drop in the frobenius norm in the error might not be the best way to conclude a method to be best. Methods such as NMI for various k values of the deployed algorithm when tested with ground reality can help us better understand the topic and thus make a reliable model for matrix factorization for the given dataset.