

Factor Analysis (FA)

Non-negative Matrix Factorization (NMF)

CSE 5290 - Artificial Intelligence Grad Project

Dr. Debasis Mitra

Group 6

Taher Patanwala

Zubin Kadva

## Factor Analysis (FA)

## 1. Introduction

Factor analysis is used to determine the variability among the data. It is a technique that can be used to reduce the dimensionality of the data. The idea behind factor analysis is to identify the variance in the observed data, and determine the unobserved data that is smaller than the actual data, but represents the same thing.

It becomes easier to analyze the data, after it has been reduced, allowing us to focus more on key distinguishing factors, rather than wasting time on too many variables.

In order to perform factor analysis, we have to operate under the assumption that there exists a linear relationship between the variables of the data set. We identify factors by determining the correlations between the variables in the data set.

Factor loading is the measure of how much correlation does the variable have with the factor. Thus, a higher factor loading means that the variables are closely related to the identified factor.

## 2. Exploratory Factor Analysis

There are two types of Factor Analysis. Exploratory Factor Analysis(EFA) and Confirmatory Factor Analysis(CFA). CFA is a more complex approach. EFA can be termed as an advanced version of Principal Component Analysis(PCA). EFA splits the dataset in to different categories making it easier to analyze. EFA usually works better with larger sample sizes, but if the factor loading among the variables is high then smaller sample sizes can also be used. The correlation needs to be higher than 0.30, otherwise it would mean that the relationship is very weak between the variable and the factor.

Problems arise when a variable falls into more than one factor. Such a situation is called split loadings. The variable has low correlation with multiple factors, and it becomes difficult to put such a variable into a specific factor group.

## 3. Theoretical Background

The theory behind the working of EFA can be explained using the mathematical and geometrical approaches.

### 3.1. Mathematical Approach

In this approach  $p$  denotes the number of variables in the dataset( $X_1, X_2, \dots, X_p$ ) and  $m$  denotes the number of factors( $F_1, F_2, \dots, F_m$ ). Each variable in the dataset is represented mathematically as below:

$$X_j = a_{j1}F_1 + a_{j2}F_2 + \cdots + a_{jm}F_m + e_j$$

where  $j = 1, 2, 3, \dots, p$ .

In the above equation  $a_{j1}, a_{j2}, \dots, a_{jm}$  are the factor loadings. This specifies how much effect the variable has on a particular factor. The specific or unique factor is denoted by  $e_j$ . It can be said that the Factor Analysis is similar to weights. A higher factor loading represents that there is a high correlation between the factor and the variable. So, factor loading determines the strength of correlation between the variable and the factor.

We need to calculate the correlation coefficient, and in order to do that, we need to identify the common features in the variables and based on that either create a correlation matrix or a covariance matrix. The correlation coefficient is used to determine the relationship between two variables.

Let us say that we have  $p$  variables and  $m$  factors. For every two pairs of variables we have to try to extract factors such that there are no intercorrelations left between those variables, as the factor itself will behave as the intercorrelations. Factor analysis can be represented by the below equation:

$$R = P C P' + U^2$$

where  $R$  = correlation coefficients matrix of the observed variables

$P$  = Factor loading matrix

$C$  = correlation matrix among the factors

$U$  = diagonal matrix of unique variances of each variable

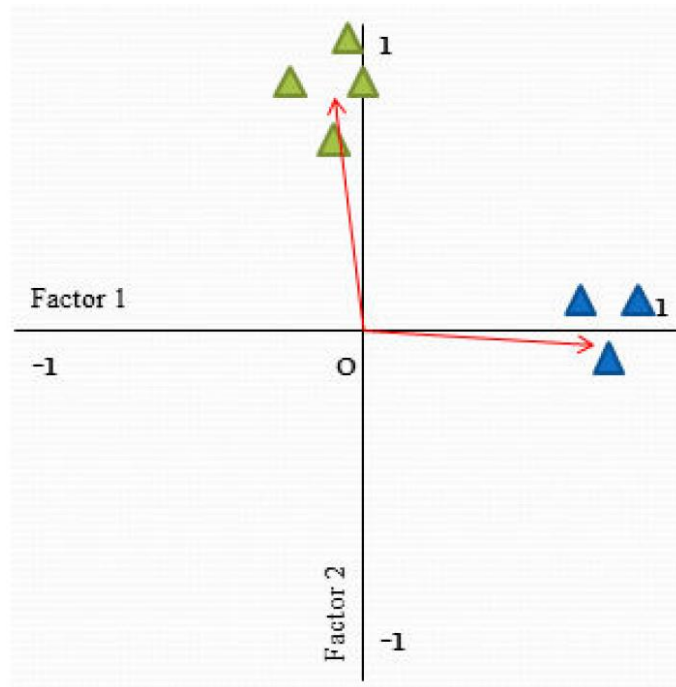
Communality can be produced in factor analysis by using variances. It is the square of the summation of factor loadings for a particular variable.

The formula is  $h_j^2 = a_{j1}^2 + a_{j2}^2 + \cdots + a_{jm}^2$ .

### **3.2 Geometrical Approach**

We can represent factor analysis using geometry to for better understanding. In this representation each axis represents a factor and the vectors or lines on the graph represents the variable. So if a variable is highly correlated to a factor than, it will be very close to that axis.

The axis will range from -1 to 1 which represents the factor loading.



The above figure is an example of factor analysis in geometrical representation. In this figure the two axis are the two factors. Factor 1 and Factor 2 and the triangles are the variables. The Blue triangles have higher correlation with Factor 1 and the Green triangles have a higher correlation with Factor 2.

#### 4. Factor Extraction

There are several techniques available that can be used to determine the factors from the variables. We can select a technique based on the requirements and the research that we are trying to perform. Maximum Likelihood, Principal Axis Factor, are some of these techniques. Principal Component Analysis(PCA) can also be used, as it is used for data reduction. An issue has been raised that whether PCA is actually a factor analysis technique on its own.

After factors are identified, rotation is performed on the factors. The objective of performing rotation on the factors is to reduce ambiguity. This allows us to fit more variables into less number of factors.

Next, we need to determine the strength of the factors by examining the factor loadings. We need to examine the factors with high factor loading, and factors with low factor loading, and ensure that they are consistent with the data. Like there should not be cases where a factor that should have low correlation with a variable has high factor loading. Also there should be very low split loadings, meaning variables that load into more than one factor should be less.

The next step is to determine the number of factors to retain. This is very important, because if we keep too many factors then it may result in a high error variance, on the other end keeping

very few factors may result in loss of important data. We can use eigenvalues and scree test to determine the number of factors to retain. It is recommended to use both the techniques together, as using only eigenvalues may result in overestimation.

## 5. Program

First we load the data into a variable say X. Then we call the in-build factoran() method of Matlab to do factor analysis[4]. The in-built factor analysis method of Matlab makes use of Maximum Likelihood technique to extract factors from the variables. The syntax is as below.

```
factoran(X,<no_of_factors>,'rotate','<type>');
```

Parameters:

- Variable containing the data
- The number of factors to extract
- Here we pass the keyword rotate, to say that we are rotating
- The type of rotation to perform

Output:

- Lambda – This returns the Factor Loading matrix. Factor Loading matrix shows how much the variable contributes to which factor. If a variable has high factor loading to a factor, it means that it is highly correlated to that factor.
- Psi – Diagonal matrix of unique variances
- T – Component Transformation matrix. This is the matrix that was used to perform rotation on the factor loadings. We can get back the un-rotated factor loadings by multiplying the factor loading matrix with the inverse of Component Transformation Matrix
- stats – Here we can get the Maximum likelihood value that was used and the error degree
- F – Factor Scores. This gives the factor scores of the data set. In other words, this gives the reduced dataset. So, if initially we had 7 variables then, it will be reduced to less variables say 2, and then this can be used for analysis rather than using the entire dataset.

In Factor Analysis, we have to determine how many factors to extract. This requires some research, and it also depends on the type of data and what we are trying to achieve. So, after we make a decision on the number of factors to extract, and after running factor analysis on the data, we need to determine how good are the factors that we have obtained. This is done by computing the correlation matrix and analyzing the factor loading matrix that we get as output from the method. The correlation matrix can be computed using the below formula that we took from the paper.

$$R = \text{Lambda} * C * \text{Lambda}' + \text{diag}(\text{Psi});$$

In the above equation, C is the correlation matrix of the factor loadings. C is computed using the below formula[4].

$$C = \text{inv}(T' * T);$$

We need to ensure that factors are consistent with the data. We analyze the correlation matrix and if a variable has high correlation with another variable, then both of them should have high factor loading towards the same factor. If variables that have high correlation in the correlation matrix, fall into different factors, then it means that the factors we have obtained are not good. So, we need to try with a different number of factors, or try some other rotation technique.

Another thing to check in the factor loading matrix is that there should be less split loadings. Split loadings is a situation when a variable has high correlation with multiple factors, or if a variable does not fall into any factor.

```
X = xlsread('data.xls'); % Read the data from excel
X = X(all(~isnan(X),2),:); % To get rid of empty rows.
[Lambda,Psi,T,stats,F] = factoran(X,2,'rotate','varimax'); % Do Factor
Analysis
C = inv(T'*T); % Correlation matrix of Factor Loadings
(Reference: https://www.mathworks.com/help/stats/factoran.html)
R = Lambda*C*Lambda' + diag(Psi); % Calculate correlation matrix of
variables
'Factor Loading'
Lambda
'Correlation Matrix'
R
'Component Transformation Matrix'
T
```

## 6. Output

Correlation Matrix:

	BPETBLAP	BPETECOP	BPETHISP	BPETLEPP	BPETRSKP	BPETSPEP	BPETWHIP
BPETBLAP	1	0.0614	0.0615	-0.0874	0.0165	0.157	-0.0806
BPETECOP	0.0614	1	0.9131	0.7902	0.9142	-0.2684	-0.9193
BPETHISP	0.0615	0.9131	1	0.7765	0.9001	-0.2617	-0.9061
BPETLEPP	-0.0874	0.7902	0.7765	1.0001	0.8809	-0.6308	-0.7387
BPETRSKP	0.0165	0.9142	0.9001	0.8809	1	-0.3916	-0.8926
BPETSPEP	0.157	-0.2684	-0.2617	-0.6308	-0.3916	1	0.2097
BPETWHIP	-0.0806	-0.9193	-0.9061	-0.7387	-0.8926	0.2097	1

### Factor Loading Matrix:

	<b>1</b>	<b>2</b>
<b>BPETBLAP</b>	0.0673	0.2456
<b>BPETECOP</b>	0.9625	-0.0138
<b>BPETHISP</b>	0.9485	-0.0096
<b>BPETLEPP</b>	0.8127	-0.5784
<b>BPETRSKP</b>	0.947	-0.1924
<b>BPETSPEP</b>	-0.2687	0.713
<b>BPETWHIP</b>	-0.956	-0.0662

After running factor analysis on our dataset, we find that 2 factors are enough to represent our original dataset of 7 variables. So, this reduced dataset is easier to analyze as we can focus on key distinguishing factors. Also this reduced dataset could be passed to a classifier algorithm for faster and easier computation.

## **7. Data Set Description**

We will use the data from AEIS(Academic Excellence Indicator System) which is provided by Texas Education Agency. This dataset has records of thousands of schools in Texas. We will use factor analysis to reduce the dimensionality of the data, making it easier to analyze this huge dataset.

The dataset used in the paper was from “Questions about Canadians’ perception of potential food risk taken from the National Public Survey on Risk Perceptions and Risk Acceptability of Prion Disease and Food Safety”. So we were looking for similar kind of dataset, that had similar data. That’s why we used the data from AEIS. We extracted data about the percentage of African American students, Hispanic Students, White students and so on for each campus in the state of Texas. It has data of 8530 campuses in Texas.

The below is a sample of the data that we used.

<b>1</b>	<b>BPETBLAP</b>	<b>BPETECOP</b>	<b>BPETHISP</b>	<b>BPETLEPP</b>	<b>BPETRSKP</b>	<b>BPETSPEP</b>	<b>BPETWHIP</b>
<b>2</b>	4.7	25.4	9.9	0.5	26.4	9.9	81.1
<b>3</b>	1.9	39.6	6.9	0.5	30.5	10.4	88
<b>4</b>	2	49.7	8.4	1.5	30	9	86.2
<b>5</b>	1.8	34.9	7	0.4	36.1	11.4	87.9
<b>6</b>	2.1	40.7	9.6	0.8	28.7	10.1	84.8



Legend:

- BPETBLAP - African American Students Percent
- BPETECOP – Economically Disadvantaged Students Percent
- BPETHISP – Hispanic Students Percent
- BPETLEPP – Limited English Proficient Students Percent
- BPETRSKP – At Risk Students Percent
- BPETSPEP – Special Education Students Percent
- BPETWHIP – White Students Percent

## 8. Difference between FA and NMF

FA	NMF
Can work with negative data	Cannot work with negative data
It reduces data dimensionality by identifying factors and can be used to represent the variables	It reduces the data, by splitting the data into smaller subsets
It identifies the factors between the variables with high correlation, and then the we can use to factors to analyze our data set	It splits the data, such that the distance(Euclidean or Frobenius) between the original matrix and the subset matrices is minimum

## 9. References

- [1] An Gie Yong and Sean Pearce. “A Beginner’s Guide to Factor Analysis: Focusing on Exploratory Factor Analysis” published in “Tutorials in Quantitative Methods for Psychology”, Vol. 9(2), p. 79-94, 2013.
- [2] Academic Excellence Indicator System. (n.d.). Retrieved October 11, 2016, from “<https://rptsvr1.tea.texas.gov/perfreport/aeis/>”
- [3] Matlab documentation of factoran method. (n.d.). Retrieved October 30, 2016, from “<https://www.mathworks.com/help/stats/factoran.html>”

Non-negative matrix  
factorization (NMF)

## 1. Introduction

NMF is a group of algorithms where a matrix  $V$  can be decomposed into two matrices  $W$  and  $H$ , each of which are easier to work with and when multiplied together, yield the original matrix.

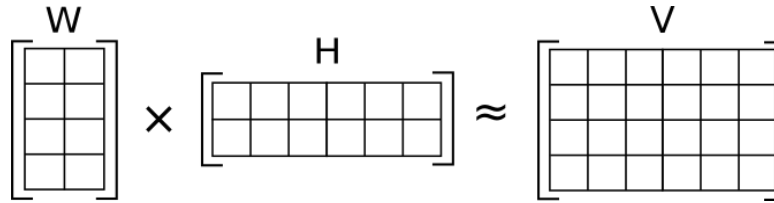


Fig. 1.1  $V$  (4 X 6) is approximated as  $W$  (4 X 2) multiplied by  $H$  (2 X 6)

(Source: [https://en.wikipedia.org/wiki/Non-negative\\_matrix\\_factorization](https://en.wikipedia.org/wiki/Non-negative_matrix_factorization))

Given, a matrix  $V$  of dimension  $m \times n$  and  $v_{ij} \geq 0$ , NMF decomposes it into 2 matrices  $W$  and  $H$  of dimension  $m \times r$  and  $r \times n$  where

$$W_{ij} \geq 0$$

$$H_{ij} \geq 0$$

$$r < \min(m, n)$$

Thus,  $V$  is decomposed into a tall, skinny matrix  $W$  and a short, wide matrix  $H$ . The user can specify  $r$  as the inner dimension of  $W$  and  $H$  as long as  $r < \min(m, n)$ .

Each column of  $V$ ,  $v_i$  can be calculated as:

$$v_i = W * h_i$$

Thus, each column of  $W$  is weighted by its corresponding row in  $h_i$ , which are then added together to form columns of  $V$ .

## 2. Purpose

Suppose  $V$  is a large dataset where each column is an observation and each row is a feature. For example, in a database of images, a column might represent some image and a row can represent a pixel. In machine learning, it is necessary to reduce the feature space for easy computation. In the above example, it is difficult to consider each pixel value every time an image is handled, so it is worthwhile to break it down into fewer components. Thus, NMF is used as a new way of reducing the dimensionality of data.

Since NMF has a non-negative constraint, it is used to represent data with positive features. This advantage can be used in image processing since each image has a positive pixel value.

NMF is similar to PCA where each base is assigned a weight. But in NMF, the weights are constrained to be positive.

## 3. Applications

### 3.1. Computer vision

NMF is beginning to be used in many fields. It is used in computer vision to reduce the feature space in images. This can be useful in identifying and classifying images.

### 3.2. Text mining

NMF is also used in text mining. For example, you might organize a series of documents into a matrix where each column may represent the frequency a particular word and a row might represent the document. Then you would extract semantic features about the data.

### 3.3. Speech denoising

NMF is used to break audio recordings of speech into speech parts and noise parts so that the speech parts alone can be isolated.

## 4. The problem

A fundamental model in NMF utilizes the least squares cost function to measure the closeness of matrices, resulting in the following standard NMF problem:

$$\text{Minimize } \|V - WH\|^2 \text{ subject to } W, H \geq 0 \quad (1)$$

where  $\|\cdot\|$  is Frobenius norm, and the inequalities are component-wise [1].

The conventional approach to find  $W$  and  $H$  is by minimizing the difference between  $V$  and  $WH$ :

$$f(W, H) \equiv \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^m (v_{ij} - (WH)_{ij})^2 \quad (2)$$

$$\text{where } W_{ia} \geq 0, H_{bj} \geq 0, \quad \forall i, a, b, j$$

Moreover, the problem is considered non-convex which makes the problem difficult and hence is of type NP(hard). Non-convexity is a concept from optimization where a problem is considered to be non-convex if the objective or any of the constraints are non-convex or unbounded. Eg. A sine wave

## 5. Framework

Many algorithms have been proposed for this. But all of them use a concept from optimization theory called block coordinate descent. This framework is used mainly because of its nice convergence property. This is based on alternating updates between  $W$  and  $H$  until a tolerance is met. So, when it converges, it will converge at a local minimum and because the problem is non-convex, this is what most algorithms should achieve.

```

while (tolerance met) {
    fix H
    update W

    fix W
    update H
}

```

*Fig. 5.1 The BCD framework*

## 6. Algorithms

We need certain properties of the NMF problem (2). The gradient of the function  $f(W, H)$  consists of 2 parts:

$$\nabla_W f(W, H) = (WH - V)H^T \quad (3)$$

$$\nabla_H f(W, H) = (WH - V)W^T \quad (4)$$

From the Karush-Kuhn-Tucker (KKT) optimality condition (Bertsekas, 1999),  $(W, H)$  is a stationary point of (2) if and only if

$$W_{ia} \geq 0$$

$$H_{bj} \geq 0$$

$$\nabla_W f(W, H)_{ia} \geq 0 \quad (5)$$

$$\nabla_H f(W, H)_{bj} \geq 0 \quad (6)$$

$$W_{ia} \cdot \nabla_W f(W, H)_{ia} = 0$$

$$H_{bj} \cdot \nabla_H f(W, H)_{bj} = 0 \quad \forall a, i, b, j$$

Thus, problem (2) is considered to be non-convex and may have several local minima. Most non-convex optimization methods guarantee only the stationarity of the limit points. Such a property is still useful, as any local minimum must be a stationary point.

### 6.1. Multiplicative update methods

The most widely used approach to solving (2) is a multiplicative update method proposed by Lee and Seung (2001).

**Algorithm 1: Multiplicative Update**

1. Initialize  $W_{ia}^1 \geq 0, H_{bj}^1 \geq 0, \forall i, a, b, j$ .

2. For  $k = 1, 2, \dots$

$$W_{ia}^{k+1} = W_{ia}^k \frac{(V(H^k)^T)_{ia}}{(W^k H^k (H^k)^T)_{ia}}, \quad \forall i, a$$

$$H_{bj}^{k+1} = H_{bj}^k \frac{((W^{k+1})^T V)_{bj}}{((W^{k+1})^T W^{k+1} H^k)_{bj}}, \quad \forall b, j$$

*Fig. 6.1 The multiplicative update algorithm*

This is a fixed-point type method. If  $(W^k H^k (H^k)^T)_{ia} \neq 0$  and  $W_{ia}^{k+1} = W_{ia}^k > 0$ , then

$$(V(H^k)^T)_{ia} = (W^k H^k (H^k)^T)_{ia} \rightarrow \nabla_W f(W^k, H^k)_{ia} = 0$$

which is part of the KKT condition (5) and (6). Lee and Seung (2001) have shown that the function value is non-increasing after every update [3]:

$$f(W^{k+1}, H^k) \leq f(W^k, H^k) \text{ and} \quad (7)$$

$$f(W^{k+1}, H^{k+1}) \leq f(W^{k+1}, H^k) \quad (8)$$

They claim that the limit point satisfies the KKT condition. However, this claim is wrong as having (7) and (8) does not mean convergence. Therefore, this method lacks optimization properties.

The overall cost of algorithm 1 is:

$$\# \text{ of trials} * O(nmr)$$

## 6.2. Alternating non-negative least squares (ANLS)

The method used for solving the above problem seems to be the alternating least squares (ALS) algorithm utilized by Paatero and Tapper in 1994 [2]. It minimizes the least squares cost function with respect to either  $W$  or  $H$ , one at a time, while fixing the other and disregarding non-negativity, and then sets any negative entries to zero after each least squares step.

From the non-increasing properties (7) and (8), the multiplicative update algorithm is a special case of a general framework, which alternatively fixes one matrix and improves the other:

Find  $W^{k+1}$  such that  $f(W^{k+1}, H^k) \leq f(W^k, H^k)$  and  
Find  $H^{k+1}$  such that  $f(W^{k+1}, H^{k+1}) \leq f(W^{k+1}, H^k)$

**Algorithm 2: Alternating non-negative least squares**

1. Initialize  $W_{ia}^1 \geq 0, H_{bj}^1 \geq 0, \quad \forall i, a, b, j.$
2. For  $k = 1, 2, \dots$

$$W^{k+1} = \arg \min_{W \geq 0} f(W, H^k) \quad (9)$$

$$H^{k+1} = \arg \min_{H \geq 0} f(W^{k+1}, H) \quad (10)$$

*Fig. 6.2 The ANLS algorithm*

This approach is the “block coordinate descent” method in bound-constrained optimization (Bertsekas, 1999) [5], where sequentially one block of variables is minimized under corresponding constraints and the remaining blocks are fixed. For NMF, we have the simplest case of only two block variables  $W$  and  $H$ .

Suppose (9) and (10) are considered sub-problems in algorithm 2. If one variable (eg.  $W$ ), then the sub problem is a collection of non-negative least squares. Thus, from (10)

$$H^{k+1'} s^{j^{th} column} = \min_{H \geq 0} \|v - W^{k+1}h\|^2 \quad (11)$$

where  $v$  is the  $j^{th}$  column of  $V$  and  $h$  is a vector variable. But solving problems (9) and (10) for every iteration would be slower than the multiplicative update algorithm 1. Efficient methods to solve them are required. Therefore, we use the projected gradient method described next.

## 7. ANLS using projected gradient

Algorithm 2 requires solving sub-problems (9) and (10). Thus, use projected gradient methods. Sub-problem (10) consists of several non-negative least squares (11), so one can solve them separately but only in parallel environments. In serial environments, we solve them together.

To obtain  $H^{k+1}$ , rewrite (10) as:

$$\min_H \bar{f}(H) \equiv \frac{1}{2} \|V - WH\|^2$$

$$\text{subject to } H_{bj} \geq 0 \quad \forall b, j$$

where both  $V$  and  $H$  are constant matrices. Similarly, we can obtain  $W^{k+1}$  by rewriting (10) as:

$$\bar{f}(W) \equiv \frac{1}{2} \|V^T - W^T H^T\|^2$$

$$\text{subject to } W_{ia} \geq 0 \quad \forall a, i$$

where both  $V^T$  and  $H^T$  are constant matrices.

The total computation cost is:

$$\# \text{ of trials} * O(\text{tmnr})$$

## 8. Data

We consider an image problem:

CBCL face image database [6]

<http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html>

This data set consists of 472 faces, 23,573 non-faces as 19 x 19 images. The idea is to reduce the dimensionality of the above data so that it can be used for further analysis and computation.

## 9. MATLAB Code [7]

The MATLAB code used for solving the NMF problem consists of a function call described as:

```
function [W,H] = nmf(V, Winit, Hinit, tol, timelimit, maxiter)
```

where

$W, H$	=	output matrices;	$V$	=	input matrix
$Winit$	=	<code>abs(randn(size(V, 1), r));</code>	$Hinit$	=	<code>abs(randn(r, size(V, 2)));</code>
$tol$	=	tolerance for stopping condition	$timelimit$	=	stopping condition
$maxiter$	=	maximum iterations; will break if tolerance or time limit reached			



The initial values for  $W$  and  $H$  plays an important role in the final result. Larger the value of  $r$ , the more accurate will be the result. But this defeats the main purpose of dimension reducing algorithms.

### 9.1. Stopping conditions

In the above MATLAB code, `tolerance` and `timeilimt` are used as stopping conditions. An initial value of the gradient is calculated by computing forbenius norm between `gradW` and `gradH`. During iteration, another value of project gradient is computed as `norm([gradW(gradW<0 | W>0); gradH(gradH<0 | H>0)])`. If this value is less than multiplication of `tolerance` and inital gradient, it terminates.

`timelimit` is another stopping condition. An initial value of `cputime` is computed. Then during iteration, the current value of `cputime` is subtracted from the earlier value. If it is greater than `timelimit`, it terminates.

`maxiter` is the number of iterations, but again this depends on values of `tolerance` and `timelimit`.

## 10. Output





W (19 x 9)	H (9 x 19)	W*H (19 x 19)	V (19 x 19)
			

Fig 10.1  $W$ ,  $H$ ,  $W*H$  and the original image

### 10.1. Verification

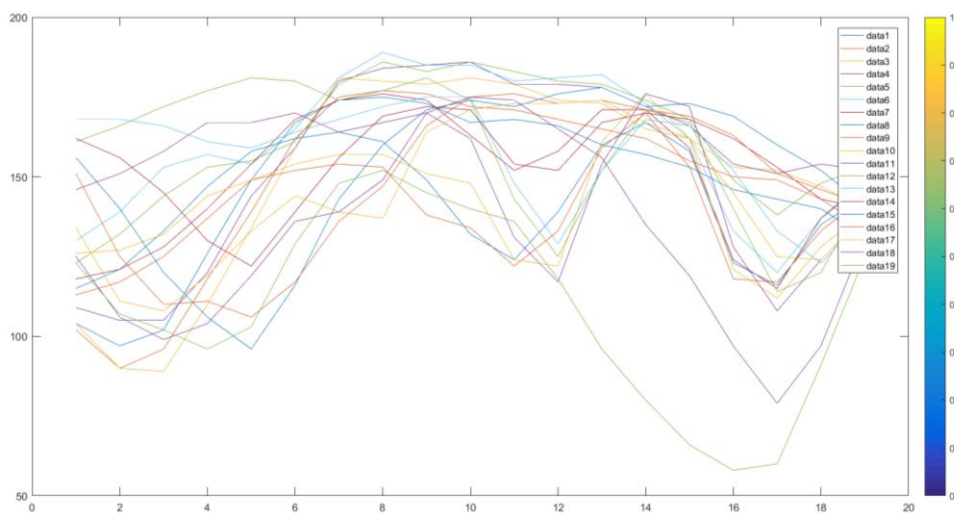


Fig 10.2 A plot of  $W*H$

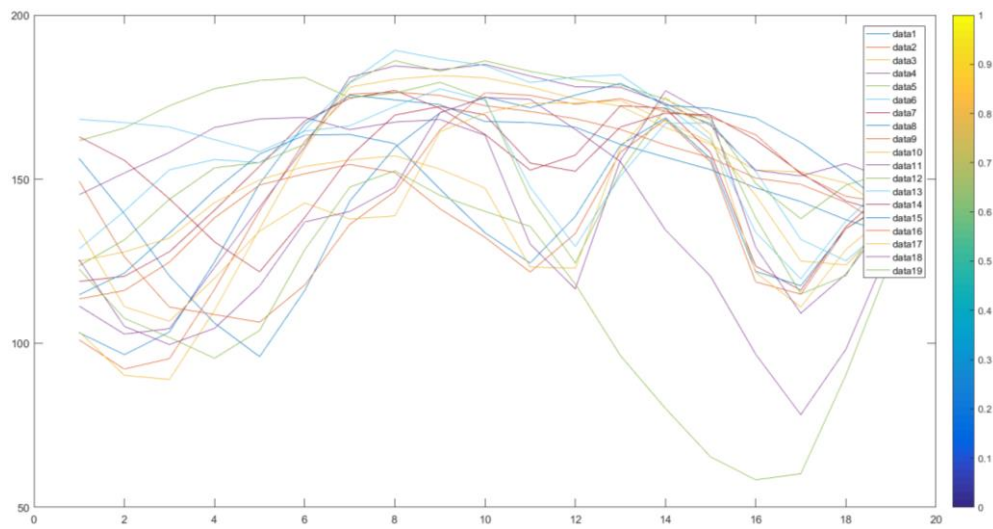


Fig 10.3 A plot of  $V$

## 10.2. Varying $r$

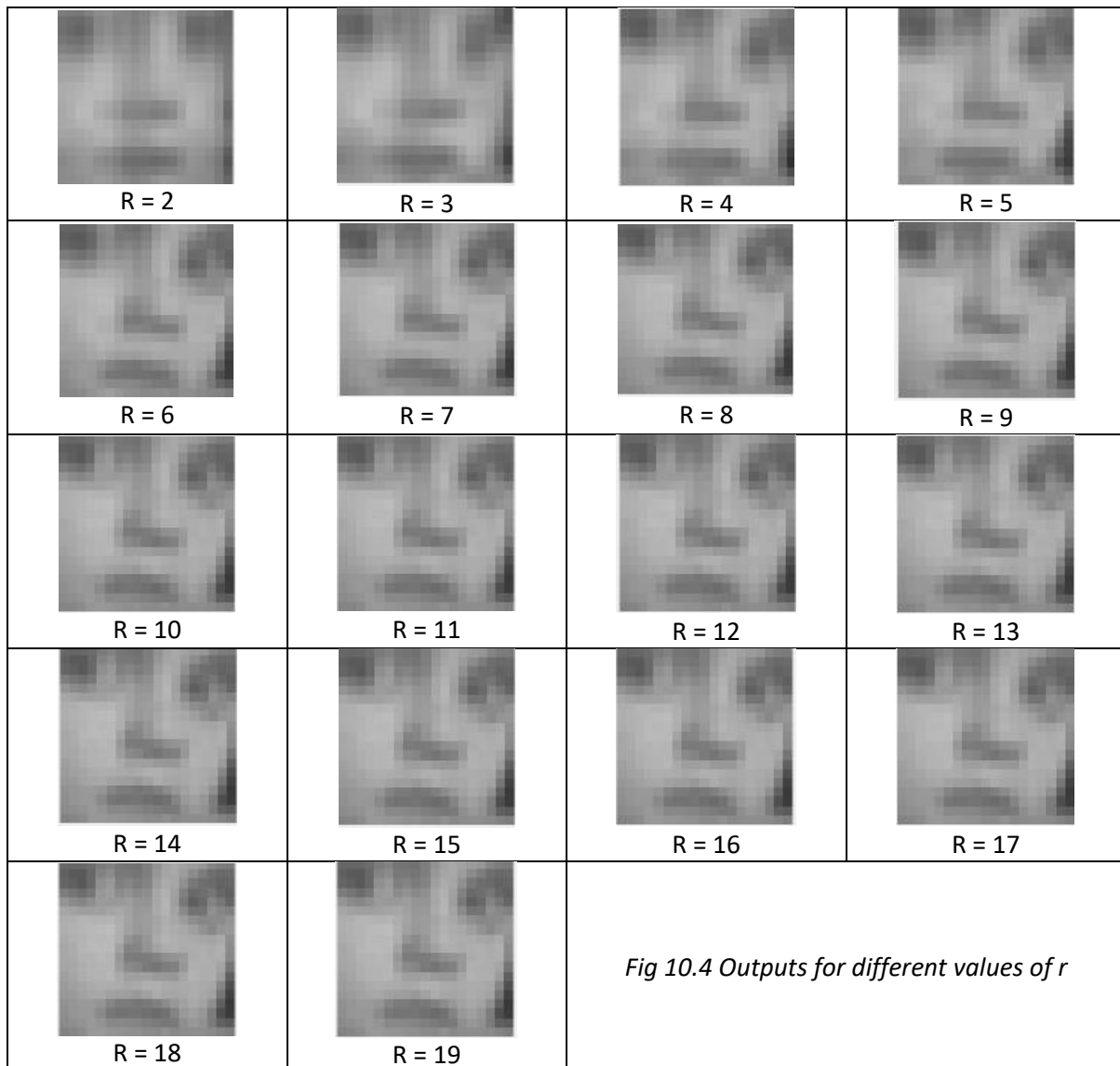


Fig 10.4 Outputs for different values of  $r$

## 11. Conclusion

Non negative matrix factorization reduces the dimensionality of an object ( $V$ ) by decomposing it into 2 matrices ( $W$  and  $H$ ), each of which are easier to work with. When these decomposed matrices are multiplied together, it yields the original matrix.

The decomposed matrices ( $W$  or  $H$ ) can be used as an input to a classifier algorithm viz. CNN. Furthermore, since these matrices are smaller than the original matrix, the classifier algorithm learns faster and more efficiently. Thus, the algorithm learns from a set of NMF basis images. In turn, this improves the learning rate and the accuracy whilst testing.

## 12. References

- [1] Yin Zhang, "An Alternating Direction Algorithm for Nonnegative Matrix Factorization", Rice University, January 2010
- [2] Paatero, P. and Tapper, U., "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values", Environmetrics, 1994
- [3] Lee, D. and Seung, H., "*Algorithms for Non-Negative Matrix Factorization*". Advances in Neural Information Processing Systems, 2001
- [4] Chih-Jen Lin, "Projected Gradient Methods for Non-Negative Matrix Factorization", Neural Computation, MIT Press, 2007
- [5] Dimitri P. Bertsekas. "*Nonlinear Programming*", Athena Scientific, 1999
- [6] CBCL Face Database #1, MIT Center for Biological and Computation Learning, <http://www.ai.mit.edu/projects/cbcl>
- [7] Chih-Jen Lin, National Taiwan University, <https://www.csie.ntu.edu.tw/~cjlin/nmf/>