Ukrainian Catholic University

Faculty of Applied Sciences

Computer Science & Business Analytics
Programmes

Handwritten Digit Recognition Using Principal Component Analysis For Matrix Detection On Images

Linear Algebra final project report

Authors:
Mykola Biliaiev
Andrii Koval
Adrian Hryn

May 14, 2019

Abstract

Nowadays, in such modern world, where people are surrounded by different technologies the field of linear algebra is more than just important. Algorithms of LA are used almost at all spheres that touch the technology world. It is actually used wherever you can see, from face recognition on your phone to creating very complicated algorithms for launching spaceships. But, what are the general things, that LA would not exist without them. Matrices? Vectors? Of course, matrix and vectors are the base notions that are used at all the algorithms of LA. However, number and particularly digits are the things without which visualizing neither vectors nor matrices would be possible. When you compose some great algorithms, you do plenty of operations with matrices and vectors in your notebooks. Sometimes it can be extremely time-consuming, so it would be great if you could digitalize your handwritten vectors and matrices, so that computer will help you with all that staff. Here comes handwritten digits recognition problem and our team found a solution. We developed an algorithm that recognizes handwritten digits with a high accuracy.

1 Introduction

In the recent years, lots of developers change their qualifications and become AI/ML engineers, so that every day are being produced not only new different implementations of solution for standard problems but also are being found brand new problems or improved algorithms to solve them. Such rapid growth of developers who work with linear algebra algorithms almost every day shows that the demand of solving data problems is extremely high even though some may thing that their algorithms are perfect.

Digit recognition is quite popular thing among the linear algebra and data science problems, so lots of implementation can be found on the Internet. Our purpose was not to create the best algorithm of all already existing, but to implement recognizing of handwritten digits without using neural networks and complicated data science algorithms. We have researched and decided that linear algorithm called Principal Component Analysis is the thing that we actually need. PCA is a pure linear algebra algorithm, which means that we use only manipulations and operations on vectors and matrices to reach the desired result. In comparison to different solution of digit recognition, PCA might lose in accuracy, but it definitely wins with its flexibility and efficiency.

2 Principal Component Analysis

2.1 How it works?

It is important to decrease the dimensionality of an image to save memory (it has many pixels that bring very few information).
PCA works for it.

So the goal for dimension reduction is to represent instances with fewer variables. For it, we take a subset of original dimensions and construct a new set of dimensions. We should preserve as much structure as possible!

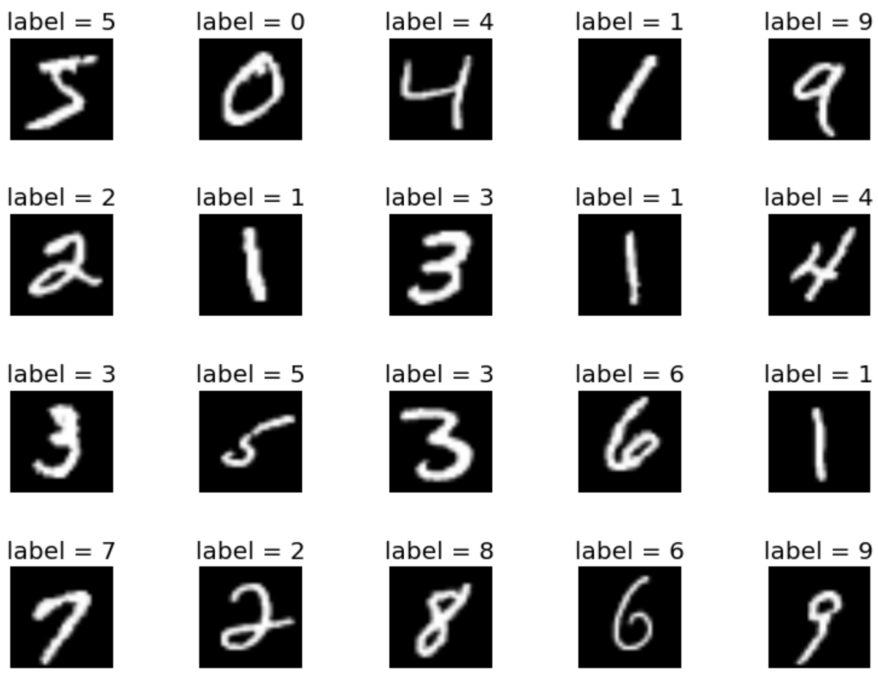


Figure 1: Dataset example

2.2 How to define principal components

```
Data: $d_t$ is a true dimensionality.
Result: Principal components for analysis in $D$.
$count = 0$;
$D = []$;
while $count$ is not equal to $d_t$ do
    $d_1$ = direction of the greatest variability in the data;
    $d_2$ = perpendicular to $d_1$, take the greatest variability of what‘s left;
    $D$.append($d_2$);
    $count$++;
end
```

Algorithm 1: Algorithm of finding principal components

Remark

Why the greatest spread? One thing that high variance preserve is relative distances. So if two data points will be pretty far from each other in the original space if we project them on one of the principal components (call it e_1), they still are pretty far from each other.

The key takeaway is that we preserve a structure by taking a high variance (to preserve distances from the original dimension to reduced dimension)

First $m < d$ principal components determine a new dimension of the data. Then we need to change coordinates of every data point to these dimensions. Notice, that we chose each principal component orthogonal to previous one so we will have easier calculation when finding coordinates of each data point to these dimensions.

2.3 What principal components are?

1. Center the data by subtracting the mean from each attribute:

$$x_{i,a} = x_{i,a} - \mu \tag{1}$$

2. Now it easy to compute covariance matrix (mean is zero). It will answer if dimension x_1 and x_2 tend to increase together, or not. The covariance matrix will consist of covariances on all positions, except the main diagonal. Each element on this diagonal will be equal to the variance.

$$Cov(b,a) = \frac{1}{n}\sum_{1}^{n} x_{ib} * x_{ia} \tag{2}$$

$$Var(a) = \frac{1}{n}\sum_{1}^{n} (x_{ia})^2 \tag{3}$$

3. Let's take an arbitrary vector. And right multiply it by the covariance matrix. We can multiply it again and again. We can see that multiplying a vector by a covariance matrix turn it into a certain direction. And if we do it several times, we will have a bigger vector, of course, but if we look at the slopes - they are converging to the dimension what we visually would pick up as a dimension of the greatest variance – dimension where the data points seem to be spread out the most.

Example

Let's take covariance matrix equal to

$$\begin{pmatrix} 2 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}$$

and multiply it by a vector

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

After next multiplications we get:

$$\begin{pmatrix} -2.5 \\ -1 \end{pmatrix} (slope: 0,400) \quad \begin{pmatrix} -6 \\ -2.7 \end{pmatrix} (slope: 0,450)$$

$$\begin{pmatrix} -14.1 \\ -6.4 \end{pmatrix} (slope: 0,454) \quad \begin{pmatrix} -33.3 \\ -15.1 \end{pmatrix} (slope: 0,454)$$

Remark to plots: blue and red represent different x_i;x_j combinations. Our arbitrary vector has coordinates (-1;1). Other vectors in counterclockwise direction represent our vector after multiplication by the covariance matrix.

So one way to pick principal axes is to look on the converging of the slopes after the multiplication by the covariance matrix.

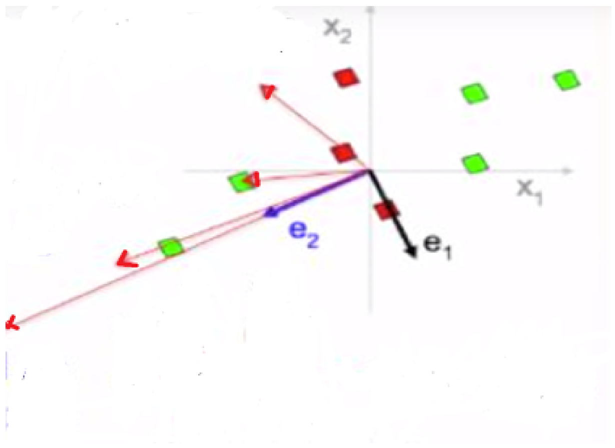


Figure 2: Corresponding plots

Or another way to look on vectors that do not get a turn after multiplication by a covariance matrix. So our $e2$ (look on the attached picture), won't turn! It will be only longer, but keep pointing in the same direction. The names for such **vectors are eigenvectors, and they become our principal components.** We will take the eigenvectors which correspond to the biggest λs and then show that they point out on dimensions in which the variance is the greatest.

2.4 What is mean to project our data points to the new dimensions?

- We assume that we have a smaller number of eigenvectors, say $e_1, e_2, \ldots, e_m$. They will represent a new dimension on which we will project our data points.

  Remark: we should take m bigger eigenvectors

- What is mean to project points? We have our vector $x = x_1, x_2, \ldots, x_d$ in original dimension d. We want to have new coordinates $x' = x_1{}', x_2{}', \ldots, x_m{}'$ in dimension m.
  Numerically, project to each dimension means to take a dot product with centred x and e_j, where j goes from $1\, to\, m$. So we started with a high dimensional$x$ and end up with a low dimensional x'.

$$(\vec{x}-\vec{\mu}) = \begin{bmatrix}(x_1-\mu_1) & (x_2-\mu_2) & \cdots & (x_d-\mu_d)\end{bmatrix}$$

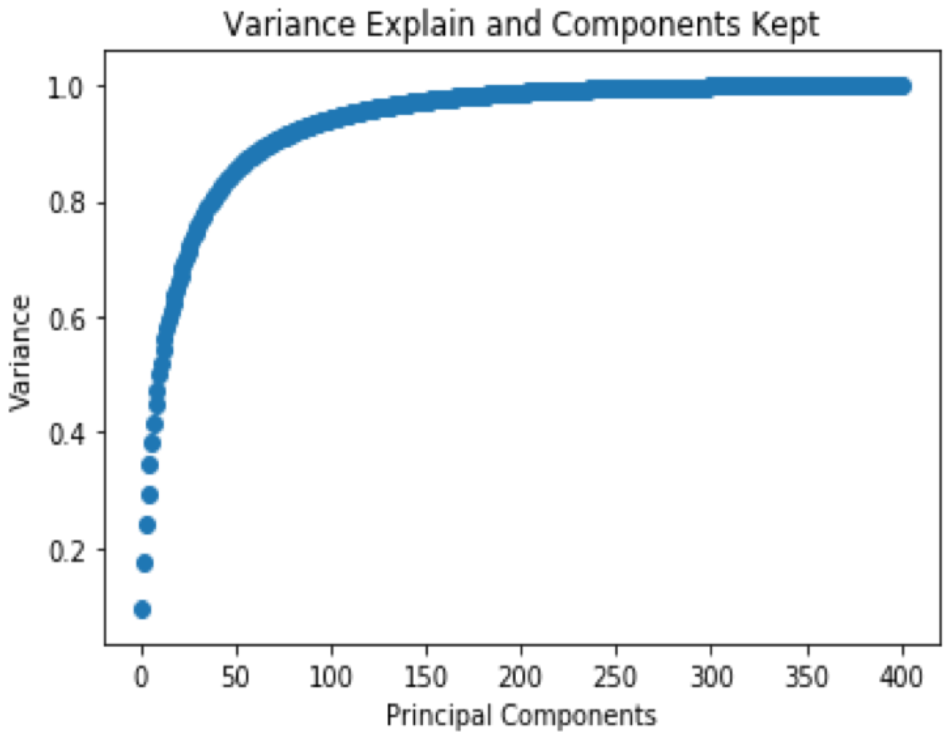
$$\begin{bmatrix} x_1' \\ x_2' \\ \vdots \\ x_m' \end{bmatrix} = \begin{bmatrix} (\vec{x}-\vec{\mu})^T\vec{e}_1 \\ (\vec{x}-\vec{\mu})^T\vec{e}_2 \\ \vdots \\ (\vec{x}-\vec{\mu})^T\vec{e}_m \end{bmatrix} = \begin{bmatrix} (x_1-\mu_1)e_{1,1} + (x_2-\mu_2)e_{1,2} + \cdots + (x_d-\mu_d)e_{1,d} \\ (x_1-\mu_1)e_{2,1} + (x_2-\mu_2)e_{2,2} + \cdots + (x_d-\mu_d)e_{2,d} \\ \vdots \\ (x_1-\mu_1)e_{m,1} + (x_2-\mu_2)e_{m,2} + \cdots + (x_d-\mu_d)e_{m,d} \end{bmatrix}$$

Remark: for a convenient understanding of an algorithm, we should understand how eigenvector represents a greater variance and why the corresponding eigenvalue point out on a variance along the eigenvector.

2.5 How many dimensions do we need?

At first, we will have d eigenvectors, but for reducing dimensionality, we want to pick m of them. $(m < d)$
λ_i is a variance along e_i. We should pick up e_i that explains the largest variance. We sort lambdas, $\lambda_1 >= \lambda_2 >= \cdots >= \lambda_d$ and pick up first m eigenvectors which explain 95% of the total variance (can be different threshold). We plot a graph (number of principal components vs variance), and we can see then first m principal components give the most weight to the variance. So we take this m as a desirable dimension.
We will calculate a share of m vectors by a formula (as lambdas represent variance):

$$\frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{d} \lambda_i} <= 1 \tag{4}$$



2.6 PCA general steps

1. Take some h_i-dimensional data
2. Center the points
3. Compute a covariance matrix
4. Find eigenvectors and eigenvalues
5. Pick $m < d$ eigenvectors with highest eigenvalues
6. Project data points to those eigenvectors
7. Obtain your low-dimensional data

3 Conclusions

So, in this report we described how the PCA algorithms works and what where the main steps and explanations while implementing and testing it in real life. We used mnist dataset to test our algorithm and have written our code using Python and numpy library. After fully developing the PCA algorithm we achieved 85% accuracy, and it is pretty good result for this algorithm. In the future it can be modified and improved to increase accuracy and decrease training time. The main purpose of the project was accomplished and our prototype can successfully recognize digits.
With this possibility we are open to extend and improve our algorithm to reach higher accuracy and apply it to our main project, web application as a main UI part and matrix detection as a main server-side part. We didn't have enough time to completely implement and finish our idea, but hopefully, in the near future we will use already known linear algebra algorithms in developing this application.

References

[1] Gilbert Strang, *Introduction to Linear Algebra*, 5th Edition, Wellesley–Cambridge Press, 2016.

[2] `https://github.com/nickolanick/PCA_digit_recognition`

[3] `https://courses.cs.ut.ee/MTAT.03.227/2016_spring/uploads/Main/lecture-notes-9.pdf`

[4] `https://www.cs.cmu.edu/ elaw/papers/pca.pdf`