MDS5210: Machine Learning Final project—Robust Matrix Factorization

Peng Deng School of Data Science 220041042

Shihan Wang School of Data Science 220041083

Han Yang School of Data Science 220041046

Jing Sun School of Data Science 116010199

Sicheng Liu School of Data Science 220041011

***Abstract*—This paper[*] is about the application of subgradient method and A-IRLS algorithms to solve the optimization problem of matrix factorization/PCA and apply them into different situations. We first evaluate our algorithms on artificially generated dataset to validate their effectiveness. Then we move to the real dataset and apply the model on the problem of face shadow removal and video frames static background extraction. And we found that both algorithms provide satisfactory performance while A-IRLS algorithm reaches smaller error and converges in much faster speed while subgradient method has much shorter running time.[†]**

***Index Terms*—Machine Learning, PCA, Sungradient, A-IRLS**

I. Introduction

PCA/matrix factorization is a very useful and popular method in the field of computer vision. In our project, we will adopt this method and build a machine learning system to solve the problem of removing the shadow of face images and extracting backgrounds form video frames.

PCA/matrix factorization aims to factorize the data matrix $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ into $\boldsymbol{U}\boldsymbol{V}^\top$. The underlying idea for applying low-rank matrix factorization is that the data matrix $\boldsymbol{X}$ possesses intrinsically low-rank structures. Under the sparsity assumption of the residual error, one possible learning problem formulation is as formula (1).

$$\underset{\boldsymbol{U}\in\mathbb{R}^{d\times r},\boldsymbol{V}\in\mathbb{R}^{n\times r}}{\text{minimize}} \left\|\boldsymbol{X}-\boldsymbol{U}\boldsymbol{V}^\top\right\|_0 \tag{1}$$

Assume that $\boldsymbol{X}=\boldsymbol{L}^\star+\boldsymbol{S}^\star$, $\boldsymbol{L}^\star$ is a low-rank matrix with rank $r \ll \min(d,n)$ and $\boldsymbol{S}^\star \in \mathbb{R}^{d\times n}$ is a sparse matrix (the residual error). $\boldsymbol{L}^\star$ can be factorized as $\boldsymbol{L}^\star=\boldsymbol{U}^\star\boldsymbol{V}^{\star\top}$, where $\boldsymbol{U}^\star \in \mathbb{R}^{d\times r}$, $\boldsymbol{V}^\star \in \mathbb{R}^{n\times r}$. Thus, our goal is equivalent to find the underlying $\boldsymbol{U}^\star$ and $\boldsymbol{V}^\star$ given $\boldsymbol{X}$. Plugging $\boldsymbol{U}^\star$ and $\boldsymbol{V}^\star$ into (1), we get $\|\boldsymbol{S}^\star\|_0$. The fact that $\boldsymbol{S}^\star$ is sparse, together with the sparsity promoting nature of l_0-quasinorm, suggests that $(\boldsymbol{U}^\star,\boldsymbol{V}^\star)$ is likely to be one optimal solution of (1). Thus, solving (1) to the globally optimal solution is likely to provide us the underlying $(\boldsymbol{U}^\star,\boldsymbol{V}^\star)$. However, the l_0-quasinorm makes the learning problem (1) intractable, so that we use the l_1-norm to replace the l_0-quasinorm and the final tractable learning problem is as formula (2).

$$\underset{\boldsymbol{U}\in\mathbb{R}^{d\times r},\boldsymbol{V}\in\mathbb{R}^{n\times r}}{\text{minimize}} \left\|\boldsymbol{X}-\boldsymbol{U}\boldsymbol{V}^\top\right\|_1 \tag{2}$$

In our project, we will implement **subgradient method** and **alternating iteratively reweighed least squares (A-IRLS)** algorithms to solve the learning problem.

II. Algorithms Description

A. Subgradient Method

The loss function $\mathcal{L}=\|\boldsymbol{X}-\boldsymbol{U}\boldsymbol{V}^\top\|_1$ is not differentiable and we cannot employ gradient descent method directly. Thus, we apply subgradient method to solve the problem. We first calculate the subgradient at $\boldsymbol{U}$ and $\boldsymbol{V}$:

$$\begin{aligned}\partial\mathcal{L}(\boldsymbol{U}) &= -\operatorname{sign}(\boldsymbol{X}-\boldsymbol{U}\boldsymbol{V}^\top)\boldsymbol{V}\\ \partial\mathcal{L}(\boldsymbol{V}) &= -\operatorname{sign}(\boldsymbol{X}-\boldsymbol{U}\boldsymbol{V}^\top)^\top\boldsymbol{U}\end{aligned} \tag{3}$$

Then set an initial α and use diminishing stepsize, so in each iteration, we have:

$$\begin{aligned}\boldsymbol{U}_{k+1} &= \boldsymbol{U}_k-\frac{\alpha}{\sqrt{n}}\cdot\partial\mathcal{L}(\boldsymbol{U}_k)\\ \boldsymbol{V}_{k+1} &= \boldsymbol{V}_k-\frac{\alpha}{\sqrt{n}}\cdot\partial\mathcal{L}(\boldsymbol{V}_k)\end{aligned} \tag{4}$$

After certain iteration, we can obtain the final estimation of $(\boldsymbol{U}^\star,\boldsymbol{V}^\star)$ as $\left(\widehat{\boldsymbol{U}},\widehat{\boldsymbol{V}}\right)$.

The subgradient method to solve this learning problem can be summarized as Algorithm (1).

Algorithm 1: The Subradient Method

Initialization: Choose $\boldsymbol{U}_0\in\mathbb{R}^{d\times r}, \boldsymbol{V}_0\in\mathbb{R}^{n\times r}$ with $\boldsymbol{U}_0^{(i,j)}\sim N(0,1)$, $\boldsymbol{V}_0^{(i,j)}\sim N(0,1)$. Choose α.

for $k=0,1,2,\cdots,N$ **do**

  Update $\boldsymbol{U}$ and $\boldsymbol{V}$:

$$\begin{aligned}\boldsymbol{U}_{k+1} &= \boldsymbol{U}_k-\frac{\alpha}{\sqrt{n}}\cdot\partial\mathcal{L}(\boldsymbol{U}_k)\\ \boldsymbol{V}_{k+1} &= \boldsymbol{V}_k-\frac{\alpha}{\sqrt{n}}\cdot\partial\mathcal{L}(\boldsymbol{U}_k)\end{aligned}$$

return $(\boldsymbol{U}_N,\boldsymbol{V}_N)$

[*]Group Member and Responsibilities:
Peng Deng: A-IRLS Algorithm, Data Processing, Result Generation
Shihan Wang: Subgradient method, Data Generation
Han Yang: A-IRLS Algorithm, Report Writing
Jing Sun: Subgradient method, Slides Making
Sicheng Liu: A-IRLS Algorithm, Report Writing

[†]https://github.com/Dpaul891/Machine_Learning_Project

B. A-IRLS Algorithm

The method of Alternating iteratively reweighed least squares (A-IRLS) is used to solve certain optimization problems in the form of p-norm iteratively in which every iteration involves solving a subproblem of weighted least squares [1].

Specifically, in our project $p = 1$ and in each step, we should fix $\boldsymbol{U}$ and $\boldsymbol{V}$ respectively and solve the weighted least squares problem of another variable. Thus, in each iteration:

$$\begin{aligned}\boldsymbol{V}_{k+1}(i,:) &= (\boldsymbol{U}_k^\top \boldsymbol{W}_k^{(v,i)} \boldsymbol{U}_k)^{-1} \boldsymbol{U}_k^\top \boldsymbol{W}_k^{(v,i)} \boldsymbol{X}(:,i) \\ \boldsymbol{U}_{k+1}(i,:) &= (\boldsymbol{V}_{k+1}^\top \boldsymbol{W}_k^{(u,i)} \boldsymbol{V}_{k+1})^{-1} \boldsymbol{V}_{k+1}^\top \boldsymbol{W}_k^{(u,i)} \boldsymbol{X}^\top(:,i)\end{aligned} \tag{5}$$

where $\boldsymbol{W}_k^{(u,i)}$ and $\boldsymbol{W}_k^{(v,i)}$ are the diagonal matrix of weights at the k^{th} iteration for $\boldsymbol{U}(i,:)$ and $\boldsymbol{V}(i,:)$ respectively. They are initialized with $\boldsymbol{W}_0^{(u,i)} = \boldsymbol{I}_n; \boldsymbol{W}_0^{(v,i)} = \boldsymbol{I}_d$. Let δ be some small value like 0.0001 then the update formula is:

$$\begin{aligned}\boldsymbol{W}_k^{(u,i)} &= \operatorname{diag}\left(\frac{1}{\max\{\delta, |\boldsymbol{X}^\top(:,i) - \boldsymbol{V}_k \boldsymbol{U}_k(i,:)|\}}\right) \\ \boldsymbol{W}_k^{(v,i)} &= \operatorname{diag}\left(\frac{1}{\max\{\delta, |\boldsymbol{X}(:,i) - \boldsymbol{U}_k \boldsymbol{V}_k(i,:)|\}}\right)\end{aligned} \tag{6}$$

The A-IRLS Algorithm to solve this learning problem can be summarized as Algorithm (2).

Algorithm 2: The A-IRLS Algorithm

Initialization: Choose $\boldsymbol{U}_0 \in \mathbb{R}^{d\times r}, \boldsymbol{V}_0 \in \mathbb{R}^{n\times r}$ with $\boldsymbol{U}_0^{(i,j)} \sim N(0,1), \boldsymbol{V}_0^{(i,j)} \sim N(0,1)$. Choose $\boldsymbol{W}_0^{(u,i)} = \boldsymbol{I}_n; \boldsymbol{W}_0^{(v,i)} = \boldsymbol{I}_d$. Choose $\delta = 0.0001$.

for $k = 0, 1, 2, \cdots, N$ **do**

  Fix $\boldsymbol{U}_k$ and update $\boldsymbol{V}_{k+1}$:

  for $i = 0, 1, 2, \cdots, n$ **do**

    $\boldsymbol{V}_{k+1}(i,:) = (\boldsymbol{U}_k^\top \boldsymbol{W}_k^{(v,i)} \boldsymbol{U}_k)^{-1} \boldsymbol{U}_k^\top \boldsymbol{W}_k^{(v,i)} \boldsymbol{X}(:,i)$

    $\boldsymbol{W}_k^{(v,i)} = \operatorname{diag}\left(\frac{1}{\max\{\delta, |\boldsymbol{X}(:,i) - \boldsymbol{U}_k \boldsymbol{V}_k(i,:)|\}}\right)$

  Fix $\boldsymbol{V}_{k+1}$ and update $\boldsymbol{U}_{k+1}$:

  for $i = 0, 1, 2, \cdots, d$ **do**

    $\boldsymbol{U}_{k+1}(i,:) = (\boldsymbol{V}_{k+1}^\top \boldsymbol{W}_k^{(u,i)} \boldsymbol{V}_{k+1})^{-1} \boldsymbol{V}_{k+1}^\top \boldsymbol{W}_k^{(u,i)} \boldsymbol{X}^\top(:,i)$

    $\boldsymbol{W}_k^{(u,i)} = \operatorname{diag}\left(\frac{1}{\max\{\delta, |\boldsymbol{X}^\top(:,i) - \boldsymbol{V}_k \boldsymbol{U}_k(i,:)|\}}\right)$

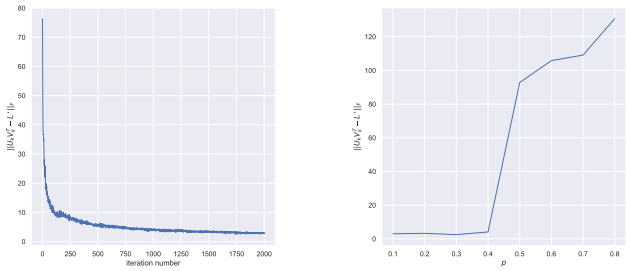
return $(\boldsymbol{U}_N, \boldsymbol{V}_N)$

III. Evaluation on Artificial Dataset

In this part, we evaluate our algorithms on artificially generated dataset. We first generate the underlying low-rank matrix $\boldsymbol{L}^\star = \boldsymbol{U}^\star \boldsymbol{V}^{\star\top}$ by generating $\boldsymbol{U}^\star \in \mathbb{R}^{d\times r}$, $\boldsymbol{V}^\star \in \mathbb{R}^{n\times r}$ with $i.i.d.$ standard Gaussian entries. To generate $\boldsymbol{S}^\star \in \mathbb{R}^{d\times n}$ we first randomly select pdn locations. Then, we fill each of the selected locations with an $i.i.d.$ mean 0 and variance 100 Gaussian entry, while the remaining locations are set to 0. Here p is the ratio of the nonzero elements in $\boldsymbol{S}^\star$ that represent the sparsity of the residual error.

We set $d = n = 50$, $r = 5$ and first set $p = 0.3$ to run the two algorithms on data $\boldsymbol{X}$ to plot the error $\|\boldsymbol{U}_k \boldsymbol{V}_k^\top - \boldsymbol{L}^\star\|_F$ in the process of iteration. After that, we vary $p \in \{0.1, 0.2, 0.3, \cdots, 0.8\}$ and run algorithms on $\boldsymbol{X}$ with different p and plot the final error $\|\widehat{\boldsymbol{U}}\widehat{\boldsymbol{V}}^\top - \boldsymbol{L}^\star\|_F$ for each algorithm to compare the performance under different sparsity of residual errors $\boldsymbol{S}^\star$. The results are shown below:

A. Subgradient Method



(a) Iterated errors of Subgradient with $p = 0.3$

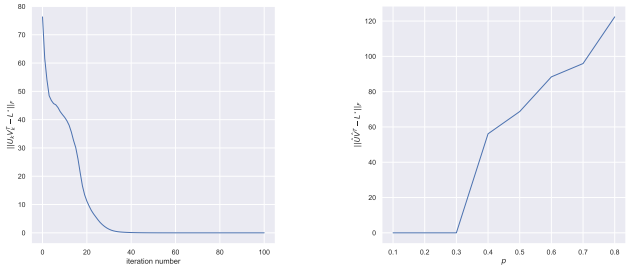
(b) Final errors of Subgradient with different p

Fig. 1. The subgradient method

From Figure 1(a), we found that the error of sub-gradient method will decrease rapidly during first 250 iterations and the rate of descent will be slower in the following iterations. Finally, the error will fluctuate around 3.

Based on Figure 1(b), the final error will generally increase when p increases. When p ranges from 0.1 to 0.4, the achieved error is near 0 which is satisfactory. When p is larger than 0.4, the final error will increase a lot. This is because when p is larger, the matrix $\boldsymbol{S}^\star$ is not sparse, so the algorithm is not suitable at this moment.

B. A-IRLS Algorithm



(a) Iterated errors of A-IRLS with $p = 0.3$

(b) Final errors of A-IRLS with different p

Fig. 2. The A-IRLS method

As shown in Figure 2(a), the error of A-IRLS algorithm reaches 0 in the 60^{th} iteration which is much faster than sub-gradient method.

From Figure 2(b), we found that when p ranges from 0.1 to 0.3, the final error will achieve 0 in the first 100 iterations and the error will increase a lot when p is larger than 0.3, which is very similar to the sub-gradient method.

According to the iterated errors plot, we can see that both algorithms have successfully made the errors decrease while the descent speed of A-IRLS is much better. And from the final errors plot, we can conclude that both algorithms perform well when p ranges from 0.1 to 0.3.

IV. Shadow and Illumination Removing

Next, we will apply our two algorithms on the Yale-B02 face image dataset to remove the shadow and illumination. The main idea is that the clean face images of the same person are similar to each other such that they can form a low-rank matrix (regard each face image as one column), while the illumination and shadow is sparse. The results are shown below:

A. Subgradient Method

We set the iteration number equals to 2000 to get $\boldsymbol{L}^\star = \boldsymbol{U}^\star \boldsymbol{V}^{\star\top}$ and we can see that the loss decreases quickly in the first 250 iterations and then fluctuates around a value in later iterations as in Figure 3.

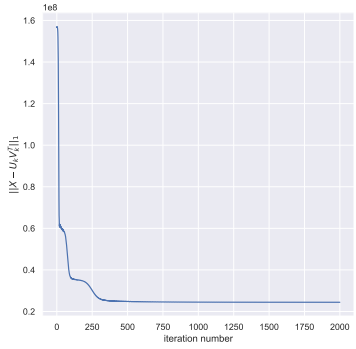


Fig. 3. Iteration Process of Subgradient in Face Image Data

The results of face shadow removal are shown as Figure 4. $\boldsymbol{X}$ is the original images, $\widehat{\boldsymbol{L}}$ is the processed images and $\widehat{\boldsymbol{S}}$ is the removed shadow.

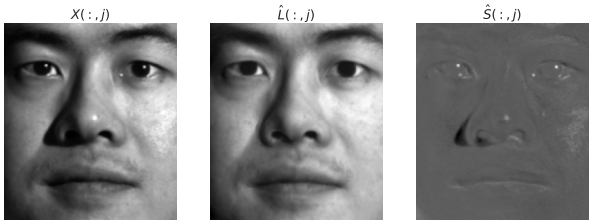


Fig. 4. Face Shadow Removal of Subgradient

B. A-IRLS Algorithm

We set the iteration number equals to 100 to get $\boldsymbol{L}^\star = \boldsymbol{U}^\star \boldsymbol{V}^{\star\top}$ and we can see the loss appears to be decreasing with iteration as Figrue 5.

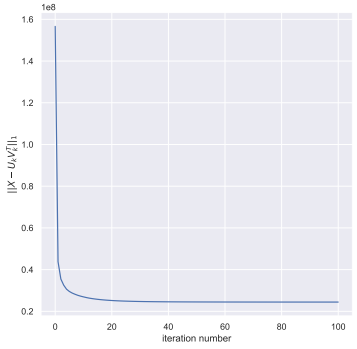


Fig. 5. Iteration Process of A-IRLS in Face Image Data

The results of face shadow removal are shown as Figure 6. $\boldsymbol{X}$ is the original images, $\widehat{\boldsymbol{L}}$ is the processed images and $\widehat{\boldsymbol{S}}$ is the removed shadow. We can see the shadow are removed from the original image.

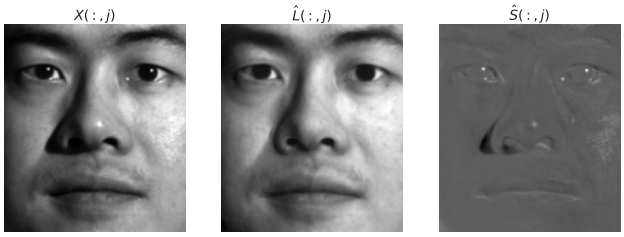


Fig. 6. Face Shadow Removal of A-IRLS

According to the figures shown above, the performance of both algorithms for face shadow removal are well while sub-gradient method needs more iterations to achieve similar result.

V. Static Background Extraction

In this part, we apply our algorithms to the escalator video dataset of a moving escalator and some pedestrians. The csv-format video is composed of 200 frames each with 130×160 pixels. Our goal is to split the video into non-static foreground and static background. The background has similar components as the original image, thus can be expressed by low rank matrices. Thus, the process can also be transformed into the minimization problem we have clarified above.

A. Subgradient Method

We set the iteration number equals to 2000 and the iteration process is similar as previous display of subgradient method. The loss still decreases rapidly in first 250 iterations and then fluctuates around a value in future iterations as in Figure 7.

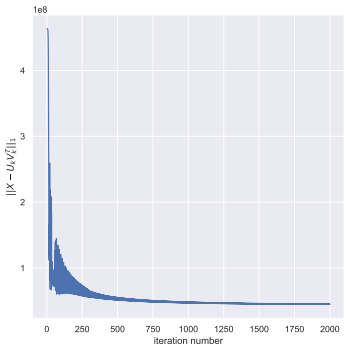


Fig. 7. Iteration Process of Subgradient in Escalator Data

After implementation of algorithm, we can obtain $\widehat{\boldsymbol{L}}$ and $\widehat{\boldsymbol{S}}$. Each column of $\boldsymbol{X}$, $\widehat{\boldsymbol{L}}$ and $\widehat{\boldsymbol{S}}$ represents an image of original picture, estimated background and foreground respectively. Here is an example as Figure 8.

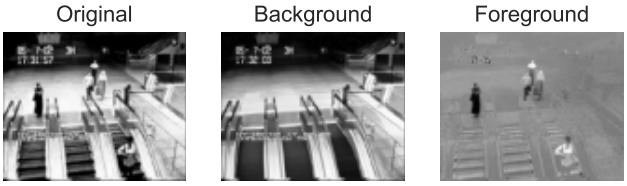


Fig. 8. Example of Static Background Extraction by Subgradient

And by combining the 200 images together, we can obtain the whole video of background and foreground which is attached in the submitted folder.

B. A-IRLS Algorithm

We set the iteration number equals to 50 and the result comes as expected. The loss appears to be decreasing with iteration and we can see the process has the rise fluctuation in about 6^{th} iteration. The results are shown as Figure 9.

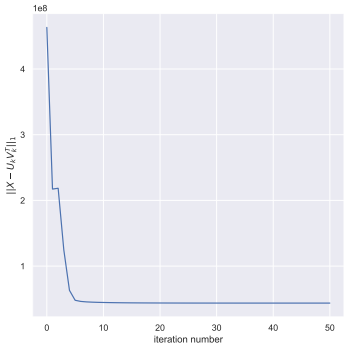


Fig. 9. Iteration Process of A-IRLS in Escalator Data

After implementation of algorithm, we can obtain $\widehat{\boldsymbol{L}}$ and $\widehat{\boldsymbol{S}}$. Each column of $\boldsymbol{X}$, $\widehat{\boldsymbol{L}}$ and $\widehat{\boldsymbol{S}}$ represents an image of original picture, estimated background and foreground respectively. Here is an example as Figure 10.

Fig. 10. Example of Static Background Extraction by A-IRLS

And by combining the 200 images together, we can obtain the whole video of background and foreground which is attached in the submitted folder.

VI. Conclusion

In this report, we apply subgradient method and A-IRLS algorithm on three matrix factorization/PCA problems: artificial datasets, face shadow removal and Static Background Extraction. In the artificial dataset evaluation, we find out good sparsity is useful for our model; In the process of face shadow removal, we have successfully removed a proportion of shadow of original image. However, due to limited iteration number the results are not so perfect; In the problem of escalator video, we have divided background and foreground of each image of the video and then combined them as a new video.

Comparatively, both algorithms provide satisfactory performance while A-IRLS algorithm reaches a litter bit smaller error and converges in a much faster speed.

In the future model tuning and development, we may try different algorithms and increase the iteration numbers in the process of algorithms to get better performance and results.

References

[1] Wikipedia, "Iteratively reweighted least squares," https://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares.

Appendix

All the codes and supporting materials can be found in our github repository.