Applications of Robust PCA

Li-Chen Cheng

Part 1 : Video Denoising

I. Introduction

Video denoising has been a long-standing problem within the field of computer vision. The objective is to remove noise from each video frame and restore it to a pristine state. Figure 1 provides an illustrative example. This problem is challenging due to the largely unfamiliar characteristics of the noise. Additionally, the video's domain may vary, necessitating tailored denoising algorithms for different scenarios.

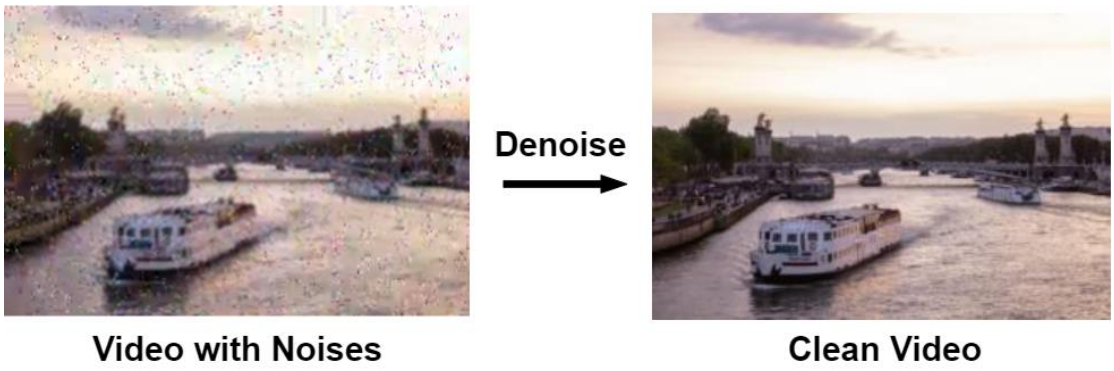

Figure 1: An example of video denoising.

In this report, we deviated from the commonly used deep-learning approaches and explored a denoising method that utilizes robust Principal Component Analysis (PCA). The method rests on the assumption that noise tends to possess relatively sparser characteristics compared to the background. By employing robust PCA, we can effectively separate a given data matrix into two distinct matrices: one representing the smooth component, and the other capturing the sparse component.

II. Methodology

A noisy image, denoted as $D \in R^{h \times w}$, can be divided into two components: a pristine part represented by a low-rank matrix A, and a noisy part captured by a sparse matrix E. Consequently, we can utilize robust PCA to decompose the frame D into its constituent parts, A and E. The optimization problem can be formulated as follows.

$$\min_{A,E} \|A\|_* + \lambda \|E\|_1 \quad , \quad \text{subject to} \quad D = A + E$$

To address the optimization problem, we employed the *Augmented Lagrange Multipliers* (ALM) method as outlined in [1]. For a comprehensive explanation of the algorithm, kindly consult the original article. Figure 2 illustrates the robust PCA via the exact ALM method, while Figure 3 shows its corresponding Python implementation.

Algorithm 4 (RPCA via the Exact ALM Method)

Input: Observation matrix $D \in \mathbb{R}^{m \times n}$, λ.

$Y_0^* = \text{sgn}(D)/J(\text{sgn}(D))$; $\mu_0 > 0$; $\rho > 1$; $k = 0$.
while not converged **do**
  // Lines 4-12 solve $(A_{k+1}^*, E_{k+1}^*) = \arg\min_{A,E} L(A, E, Y_k^*, \mu_k)$.
  $A_{k+1}^0 = A_k^*$, $E_{k+1}^0 = E_k^*$, $j = 0$;
  while not converged **do**
    // Lines 7-8 solve $A_{k+1}^{j+1} = \arg\min_{A} L(A, E_{k+1}^j, Y_k^*, \mu_k)$.
    $(U, S, V) = \text{svd}(D - E_{k+1}^j + \mu_k^{-1} Y_k^*)$;
    $A_{k+1}^{j+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$;
    // Line 10 solves $E_{k+1}^{j+1} = \arg\min_{E} L(A_{k+1}^{j+1}, E, Y_k^*, \mu_k)$.
    $E_{k+1}^{j+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[D - A_{k+1}^{j+1} + \mu_k^{-1} Y_k^*]$;
    $j \quad j + 1$.
  end while
  $Y_{k+1}^* = Y_k^* + \mu_k(D - A_{k+1}^* - E_{k+1}^*)$; $\mu_{k+1} = \rho\mu_k$.
  $k \quad k + 1$.
end while

Output: (A_k^*, E_k^*).

Figure 2: Robust PCA via exact ALM method. From [1].

```python
# Robust PCA via the Exact ALM Method
def RPCA(D, l, mu, rho):
    signD = np.sign(D)
    Y = signD / np.maximum(np.linalg.norm(signD, ord=2, axis=(0,1)), 1/l*np.linalg.norm(signD, ord=np.inf, axis=(0,1)))
    A = np.zeros(D.shape)
    E = np.zeros(D.shape)
    A_prev = np.zeros(D.shape)
    E_prev = np.zeros(D.shape)

    for k in range(500):
        # Solve A and E
        for j in range(500):
            # Solve A
            U, S, Vh = np.linalg.svd(D - E + Y/mu, full_matrices=False)
            A = U @ np.diag(np.where(S > 1/mu, S - 1/mu, np.where(S < -1/mu, S + 1/mu, 0))) @ Vh
            # Solve E
            Q = D - A + Y/mu
            E = np.where(Q > l/mu, Q - l/mu, np.where(Q < -l/mu, Q + l/mu, 0))

            # Convergence condition
            if (np.linalg.norm(A - A_prev, ord='fro') / np.linalg.norm(D, ord='fro') < 1e-5 and
                np.linalg.norm(E - E_prev, ord='fro') / np.linalg.norm(D, ord='fro') < 1e-5):
                break

            A_prev = A
            E_prev = E

        # Update Y and mu
        Y = Y + mu*(D - A - E)
        mu = rho*mu

        # Convergence condition
        if np.linalg.norm(D - A - E, ord='fro') / np.linalg.norm(D, ord='fro') < 1e-7:
            break

    return A, E
```

Figure 3: The Python implementation of robust PCA via the exact ALM method.

We also implemented the inexact version of the ALM method by removing the inner loop present in the exact version. According to [1], it has been observed that A and E can still converge to an optimal solution without precisely solving the sub-problem within the inner loop. Just a single step is sufficient for convergence. The detailed proof can be found in the original article. Figure 4 illustrates the algorithm, and Figure 5 shows its corresponding Python implementation.

Algorithm 5 (RPCA via the Inexact ALM Method)

Input: Observation matrix $D \in \mathbb{R}^{m \times n}$, λ.

$Y_0 = D/J(D)$; $E_0 = 0$; $\mu_0 > 0$; $\rho > 1$; $k = 0$.
while not converged **do**
  // Lines 4-5 solve $A_{k+1} = \arg\min_A L(A, E_k, Y_k, \mu_k)$.
  $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1} Y_k)$;
  $A_{k+1} = U \mathcal{S}_{\mu_k^{-1}}[S] V^T$.
  // Line 7 solves $E_{k+1} = \arg\min_E L(A_{k+1}, E, Y_k, \mu_k)$.
  $E_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}}[D - A_{k+1} + \mu_k^{-1} Y_k]$.
  $Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1})$; $\mu_{k+1} = \rho \mu_k$.
  $k \quad k+1$.
end while

Output: (A_k, E_k).

Figure 4: Robust PCA via the inexact ALM method. From [1].

```python
# Robust PCA via the Inexact ALM Method
def RPCA_inexact(D, l, mu, rho):
    Y = D / np.maximum(np.linalg.norm(D, ord=2, axis=(0,1)), 1/l*np.linalg.norm(D, ord=np.inf, axis=(0,1)))
    A = np.zeros(D.shape)
    E = np.zeros(D.shape)
    A_prev = np.zeros(D.shape)
    E_prev = np.zeros(D.shape)

    for k in range(500):
        # Solve A
        U, S, Vh = np.linalg.svd(D - E + Y/mu, full_matrices=False)
        A = U @ np.diag(np.where(S > 1/mu, S - 1/mu, np.where(S < -1/mu, S + 1/mu, 0))) @ Vh

        # Solve E
        Q = D - A + Y/mu
        E = np.where(Q > l/mu, Q - l/mu, np.where(Q < -l/mu, Q + l/mu, 0))

        # Update Y and mu
        Y = Y + mu*(D - A - E)
        mu = rho*mu

        # Convergence condition
        if np.linalg.norm(D - A - E, ord='fro') / np.linalg.norm(D, ord='fro') < 1e-7:
            break

    return A, E
```

Figure 5: The Python implementation of robust PCA via the inexact ALM method.

Implementation Details

We processed the video frame by frame, where each frame, denoted as F, comprises three channels representing red, green, and blue (F_R, F_G, and F_B, respectively). The aforementioned method was applied to each channel independently.

As for the hyperparameters, we adhered to the suggestion provided in [1]. In the exact version, we set μ_0 to $0.5/\|\mathrm{sgn}(D)\|_2$ and ρ to 6. In the inexact version, we set μ_0 to $1.25/\|D\|_2$ and ρ to 1.5. We conducted experiments with various λ values, and discussions on the parameter's impact are presented in the next section.

III. Evaluation

We employed two benchmark videos for evaluation: one generated by introducing artificial noise to a real image (considered as the ground truth), and another created by adding artificial noise to a real video (also considered as the ground truth). These videos were labeled as "boat" and "flower" respectively, and snapshots of them can be seen in Figure 6.

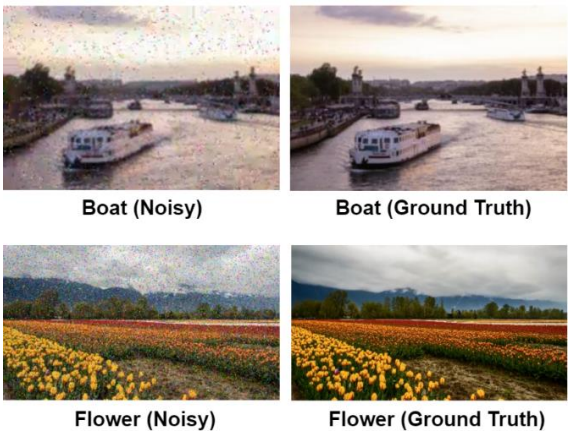

Figure 6: The snapshots of "boat" and "flower" video.

To quantitatively assess the impact of λ, we calculate the root mean square error (RMSE) between the denoised video and the ground truth. We consider λ values ranging from 0.07 to 0.1, specifically $\lambda \in \{0.07, 0.08, 0.081, 0.082, 0.083, 0.084, 0.085, 0.09, 0.1\}$. The obtained results are presented in Figure 7 and Figure 8.

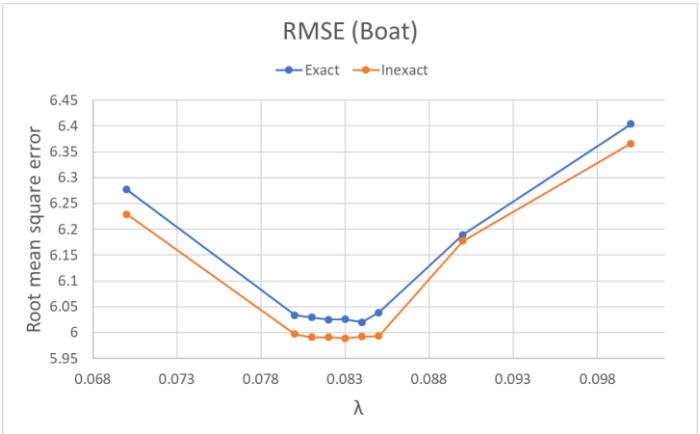

Figure 7: Root mean square errors for the "boat" video across different λ.

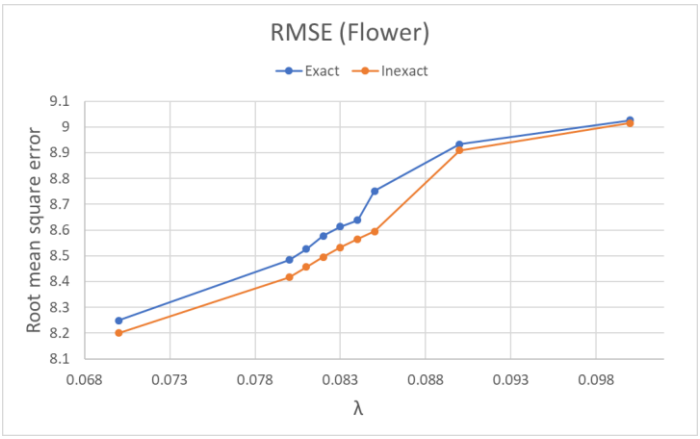

Figure 8: Root mean square errors for the "flower" video across different λ.

There are two observations. Firstly, the most suitable λ differs between videos. The "boat" video achieves the lowest error with λ equal to 0.083, whereas the "flower" video achieves the lowest error with λ equal to 0.07. This observation implies that the input video's domain also plays a crucial role in the denoising problem, as it directly affects the relationship between the noise and the content. Secondly, the inexact version generally outperforms the exact version slightly. We speculate that this is due to the choice of other hyperparameters. If we meticulously tune the hyperparameters, the difference should be minimal.

λ plays a crucial role in determining the sparsity level of E. To further demonstrate the influence of λ, we present the qualitative results in Figure 9, Figure 10, Figure 11, and Figure 12. These figures depict the denoised images and the corresponding noises of the first frame for different λ values using the inexact ALM algorithm.

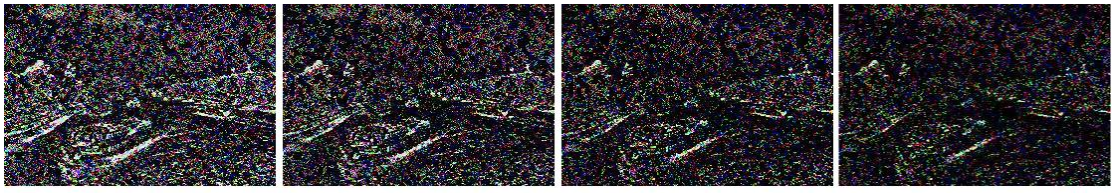

Figure 9: Noises (E) obtained using various λ for the first frame of "boat".

Figure 10: Denoised image (A) obtained using various λ for the first frame of "boat".

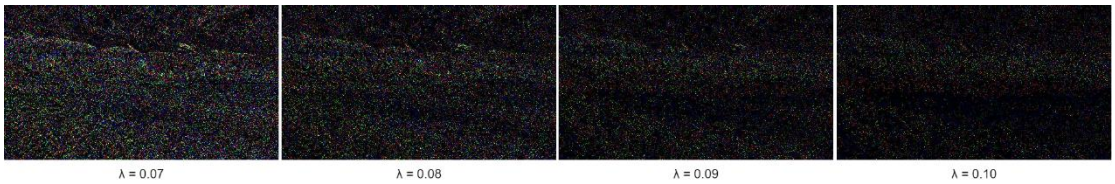

Figure 11: Noises (E) obtained using various λ for the first frame of "flower".

Figure 12: Denoised image (A) obtained using various λ for the first frame of "flower".

As shown in Figure 9 and Figure 11, a smaller λ value leads to a decreased level of sparsity. Nonetheless, the noises detected via robust PCA with a smaller λ may not necessarily represent actual noise. Sparse textures like edges can be inadvertently extracted from the frame. This phenomenon is also evident in Figure 10 and Figure 12, where certain denoised images obtained using a smaller λ exhibit more residual noise compared to those obtained using a larger λ. Consequently, it becomes crucial to carefully fine-tune the λ parameter to achieve satisfactory results.

We also measured the "boat" video's per-frame execution time using various λ values, and the corresponding results (averages of five measurements) are presented in Table 1.

	λ = 0.07	λ = 0.08	λ = 0.09	λ = 0.10
Exact	16.0248	14.8527	13.7330	11.6617
Inexact	2.3615	2.3729	2.2671	1.7245

Table 1: Per-frame execution time with various λ values.

Based on the results in Table 1, it can be observed that the inexact version exhibited a significantly faster convergence compared to the exact version. This notable improvement can be attributed to the removal of the inner loop in the exact version. Additionally, it is noteworthy that higher λ values frequently resulted in accelerated convergence. We speculate that this phenomenon can be attributed to the larger L1 penalty, which induces a greater number of zero elements in matrix E through soft thresholding. Consequently, this leads to expedited matrix computations.

IV. Other Application: Foreground-Background Separation

Robust PCA can also be employed to distinguish between the background and the foreground. Let us assume that the background remains relatively stationary, while the foreground predominantly consists of multiple moving objects. Considering each frame as a data point, the static nature of the background enables its representation through specific vectors. Hence, by applying robust PCA to a series of consecutive frames, we can extract a low-rank matrix corresponding to the background, while the sparse matrix represents the moving objects.

Figure 14 presents an example where the entire video is reshaped as $\mathbb{R}^{n\times(hwc)}$ before serving as the input for robust PCA. The video consists of n frames with a height of h, a width of w, and c channels.

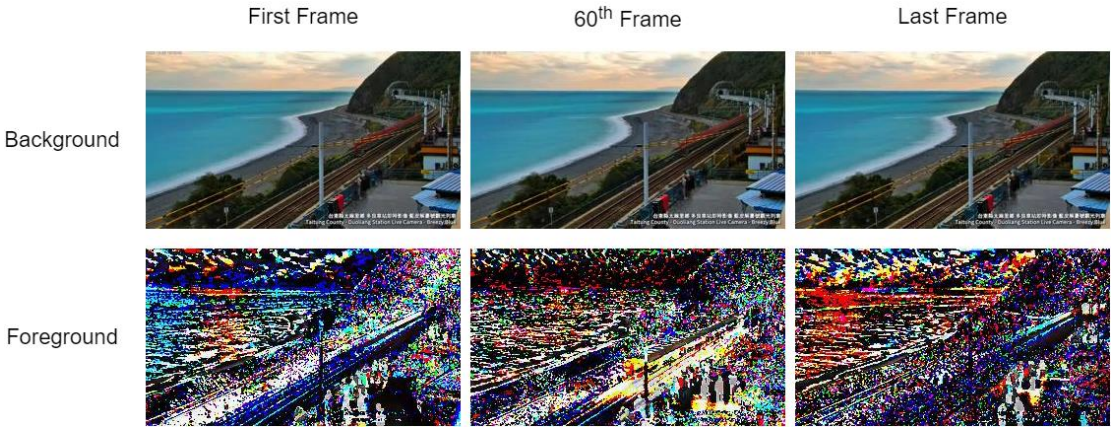

Figure 14: An example of foreground-background separation.

The example illustrates that the background can be effectively isolated from the original video by extracting a low-rank matrix. In this case, the backgrounds in the first, 60th, and last frames exhibit similarities, allowing them to be represented by a set of eigenvectors. Furthermore, moving objects, such as people and waves, are observed in the foreground. Since these objects are not consistently present in all frames, they are effectively filtered out as sparse outliers.

Additionally, suppose we ascertain that a video containing noises does not involve multiple dynamic entities and frequent changes in perspective. In that case, we can employ similar techniques that consider the clear content as a stationary background and treat the noise as moving objects. We implement this approach on both the “boat” and “flower” videos, utilizing the entire video as input for robust PCA. The outcomes are presented in Figure 15 and Figure 16.

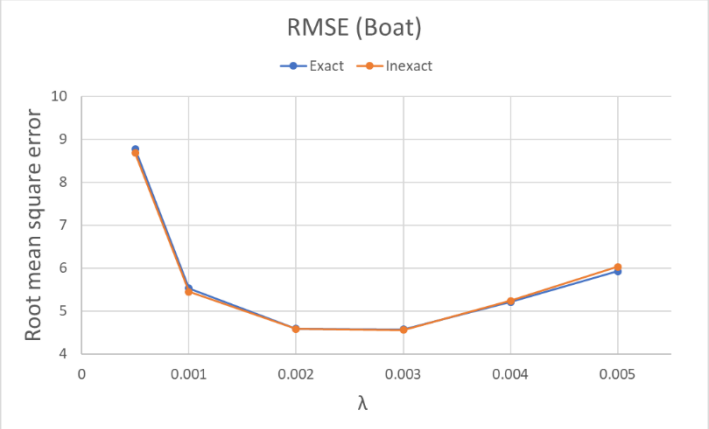

Figure 15: Root mean square errors for the "boat" video across different λ. The optimization problem incorporates all frames jointly.

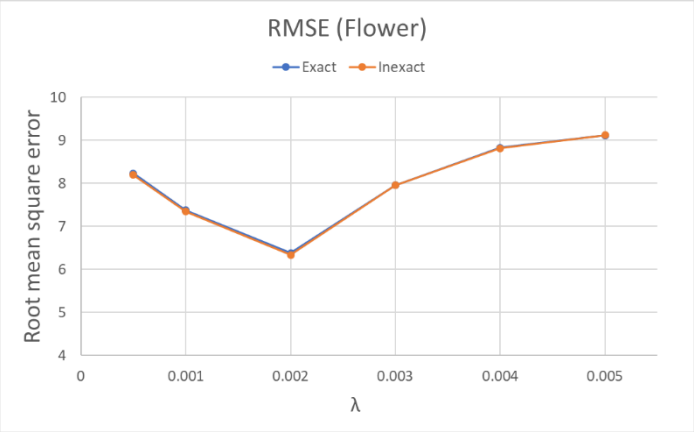

Figure 16: Root mean square errors for the "flower" video across different λ. The optimization problem incorporates all frames jointly.

The disparities between the exact and inexact versions are minimal. Nevertheless, this method attains significantly lower errors compared to the previously mentioned approach of conducting robust PCA on a frame-by-frame basis. Leveraging the information inherent in consecutive frames can potentially result in enhanced performance.

V. Conclusion

In part 1 of our study, we conducted experiments on robust PCA for video denoising. Our evaluation revealed that the sparsity penalty λ should be carefully chosen for each case to achieve satisfactory performance. Additionally, we observed that the inexact ALM version of robust PCA can deliver comparable or even superior results compared to the exact ALM version, while significantly reducing computation time. Lastly, when dealing with stationary videos, applying robust PCA to multiple consecutive frames generally led to overall performance enhancements.

Part 2 : Anomaly Detection

I. Introduction

Anomaly detection is a well-known problem involving identifying and removing anomalous outliers within our dataset. This problem holds significant importance due to its numerous applications, including the prevention of data poisoning attacks in the field of deep learning. Data poisoning attacks aim to surreptitiously insert a backdoor into a deep learning model by training the model with tainted data that contains backdoor triggers. In Figure 17, an illustrative example of poisoned data is presented, featuring a white dot positioned at the lower-right corner, representing the backdoor trigger. In such cases, employing anomaly detection becomes crucial as it allows for the identification and elimination of potentially contaminated data, thereby ensuring the purification of the dataset before training.

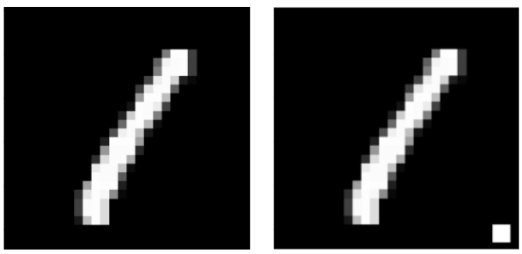

Figure 17: An illustration of poisoned data.

In this report, we further employ robust Principal Component Analysis (PCA) for anomaly detection.

II. Methodology

We assume that the non-poisoned data possess similar characteristics, allowing it to be effectively represented by an eigenvector set. Conversely, backdoor triggers deviate from typical patterns and resemble sparse noise, thus enabling their extraction through robust PCA. We utilized the inexact ALM method, as illustrated in Figure 4 and Figure 5, to address the problem of robust PCA.

Implementation Details

To perform robust PCA on a data matrix $D \in \mathbb{R}^{m \times n}$, consisting of n data points, with each point having a dimension of m, we follow these steps:

(1) Apply robust PCA on D, resulting in the extraction of matrices A and E.
(2) Set all elements in E that have values lower than a threshold τ to zero.
(3) Identify "poisoned" data points by checking if the sum of their corresponding noise in matrix E is non-zero.

The Python implementation for this process is shown in Figure 18. We used the same set of hyperparameters as previously mentioned, except for the parameters λ and τ.

```
# lambda: l
# tau: t

# Robust PCA
A, E = RPCA_inexact(D, l, 1.25 / np.linalg.norm(D, ord=2, axis=(0,1)), 1.5)
# Thresholding
E = np.where(E > t, E, 0)
# 1: poisoned data, 0: non-poisoned data
result = np.where(np.sum(E, axis=0) > 0, 1.0, 0.0)
```

Figure 18: Python implementation of anomaly detection.

III. Evaluation

We utilize the MNIST [2] dataset for evaluation, which consists of 3570 data points. Each data point within the dataset is comprised of 784 dimensions (28 x 28 pixels). Moreover, there are 62 instances of poisoned data.

Grid search is utilized to identify the optimal hyperparameters that yield the highest accuracy. For the training set, λ is set to 0.053, and τ is set to 0.999, resulting in an accuracy of 0.99664.

IV. Conclusion

In Part 2, we demonstrated the utility of robust PCA for anomaly detection. However, it is important to note that the selection of hyperparameter values significantly affects the effectiveness of the detection process. Therefore, it is advisable to explore better strategies for efficiently determining appropriate hyperparameter values.

Reference

[1] Zhouchen Lin, Minming Chen, and Yi Ma, The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices, arXiv preprint, 2010

[2] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE, 1998