Greedy and convex methods for sparse problems

INSA Rennes - 5MA - Parcimonie

In sparse problems, we consider a target $\mathbf{y} \in \mathbf{R}^m$ constructed from a dictionary $\mathbf{A} \in \mathbf{R}^{m \times n}$ as

$$\mathbf{y} = \mathbf{A}\mathbf{x}^\star + \boldsymbol{\epsilon}$$

where $\mathbf{x}^\star \in \mathbf{R}^n$ is usually dubbed the *ground truth* and where $\boldsymbol{\epsilon} \in \mathbf{R}^m$ is a potential noise. The goal of this work is to approach the solutions of different problems involving the ℓ_0-norm that aim to reconstruct $\mathbf{x}^\star$ from $\mathbf{y}$ and $\mathbf{A}$. More precisely, we consider the following problems

$$\min \tfrac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \text{ s.t. } \|\mathbf{x}\|_0 \leq k \qquad (\mathcal{P}_k)$$

$$\min \|\mathbf{x}\|_0 \text{ s.t. } \tfrac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \leq \epsilon \qquad (\mathcal{P}_\epsilon)$$

$$\min \tfrac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_0 \qquad (\mathcal{P}_\lambda)$$

We remind that these problems are NP-hard and solving them exactly is usually out of computational reach. Thus, one can rather leverage thresholding and greedy methods that provide approximate solutions of the above problems. Also, one can relax the ℓ_0-norm into an ℓ_1-norm so as to use convex modelling languages such as `CVX` to solve the relaxed problems in polynomial time. All these strategies aim to find some suitable $\hat{\mathbf{x}}$ approximating $\mathbf{x}^\dagger$. We will assess the quality of the solutions obtained using the following performance criterions :

- The **signal error** : $R = \frac{1}{n}\|\hat{\mathbf{x}} - \mathbf{x}^\star\|_2^2$
- The **target error** : $A = \frac{1}{m}\|\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}\|_2^2$
- The **F1 score** : $F = 2PR/(P+R)$ where

  $$P = \frac{|\text{supp}(\mathbf{x}^\star) \cap \text{supp}(\hat{\mathbf{x}})|}{|\text{supp}(\hat{\mathbf{x}})|} \quad \text{and} \quad R = \frac{|\text{supp}(\mathbf{x}^\star) \cap \text{supp}(\hat{\mathbf{x}})|}{|\text{supp}(\mathbf{x}^\star)|}$$

  are the recovery *precision* and the recovery *recall*. One has $F = 1$ when a perfect recovery occurs and $F = 0$ when no elements of the support of $\mathbf{x}^\star$ are identified.

1 Data generation and performance criteria

The first objective is to create routines to generate synthetic data and to evaluate performance criteria.

1. Write a function to generate synthetic data. This function must

(a) Generate a k-sparse vector with evenly-distributed non-zero entries whose amplitude is set to $x_i = s + \text{sign}(s)$ with $s \sim \mathcal{N}(0, 1)$.

(b) Generate a matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$ with entries $a_{ij} \sim \mathcal{N}(0, 1)$.

(c) Normalize the matrix columns to a unit ℓ_2-norm.

(d) Generate the observation $\mathbf{y} = \mathbf{A}\mathbf{x}^\star$.

(e) Add a noise $\boldsymbol{\epsilon}$ with a SNR[1] of value σ.

(f) Return $\mathbf{x}^\star$, $\mathbf{A}$ and $\mathbf{y}$.

2. Explain why it makes more sense to add the sign of s to set the non-zero amplitudes of $\mathbf{x}^\star$.

3. Write routines to compute the three performance criteria.

2 Thresholding methods

In this first part, we investigate how thresholding methods can produce approximate solutions of the problem $(\mathcal{P}_k)$. We use the IHT algorithm previously seen in the lecture notes.

Algorithm 1: Iterative Hard Thresholding

Input: $\mathbf{A}$, $\mathbf{y}$, α, k

Initialize $\mathbf{x} \leftarrow \mathbf{0}$, $\mathbf{z} \leftarrow \mathbf{0}$
while *the stopping criterion is not met* **do**
  $\mathbf{z} \leftarrow \mathbf{x} + \alpha \mathbf{A}^{\mathrm{T}}(\mathbf{y} - \mathbf{A}\mathbf{x})$ // Gradient step
  $\mathbf{x} \leftarrow \texttt{Threshold}_k(\mathbf{z})$ // Best k-term approx.
end

1. Propose at least two stopping criteria for the algorithm.

2. Write a function that outputs the k-term best approximation of a vector.

3. Implement the IHT algorithm.

4. Test the algorithm on synthetic data and compute performance criteria.

5. Set $k = 5$, $m = 50$ and $n = 100$ and plot the number of iterations as a function of the step-size α for $\alpha \in [0.1, 1]$. You may perform several trial to average the results. What do you observe ?

3 Greedy procedures

Now, we investigate how greedy procedures can produce approximate solutions to the problems $(\mathcal{P}_k)$-$(\mathcal{P}_\epsilon)$-$(\mathcal{P}_\lambda)$. In particular, we consider the OMP algorithm previously seen in the lecture notes.

[1] The "Signal-to-Noise Ratio" is defined as $\text{SNR}(\mathbf{A}\mathbf{x}, \boldsymbol{\epsilon}) = \|\mathbf{A}\mathbf{x}\|_2^2 / \|\boldsymbol{\epsilon}\|_2^2$. It is often expressed in dB as $\text{SNR}_{\text{dB}}(\mathbf{A}\mathbf{x}, \boldsymbol{\epsilon}) = 10 \log_{10}(\text{SNR}(\mathbf{A}\mathbf{x}, \boldsymbol{\epsilon}))$.

Algorithm 2: Orthogonal Matching Pursuit

Input: $\mathbf{A}$, $\mathbf{y}$

Initialize $\mathbf{r} \leftarrow \mathbf{y}$, $\mathbf{x} \leftarrow \mathbf{0}$, $\mathcal{S} \leftarrow \emptyset$
while *the stopping criterion is not met* **do**
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\text{argmax}_{i \notin \mathcal{S}} |\mathbf{a}_i^{\mathrm{T}} \mathbf{r}|\}$ `// Selection step`
  $\mathbf{x}_{\mathcal{S}} \leftarrow \mathbf{A}_{\mathcal{S}}^{\dagger} \mathbf{y}$ `// Minimization step`
  $\mathbf{r} \leftarrow \mathbf{y} - \mathbf{A}\mathbf{x}$ `// Residual update`
end

1. Recall the definition of the pseudo-inverse $\mathbf{A}^{\dagger}$ of a matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$. *Hint : In the minimization step, we wish to optimize* $\frac{1}{2}\|\mathbf{y} - \mathbf{A}_{\mathcal{S}}\mathbf{x}_{\mathcal{S}}\|_2^2$.

2. Propose a termination criterion depending on the problem ($\mathcal{P}_k$), ($\mathcal{P}_\epsilon$) or ($\mathcal{P}_\lambda$) solved.

3. Implement the OMP algorithm.

4. Test the algorithm on synthetic data and compute reconstruction criteria.

5. Discuss the choice of the parameters k, ϵ and λ for the OMP algorithm. What if the sparsity of $\mathbf{x}^\star$ is not known *a priori* ?

6. Implement a test to ensure the orthogonality property. *Hint : This property states that* $\mathbf{A}_{\mathcal{S}}^{\mathrm{T}}\mathbf{r} = \mathbf{0}$ *at each iteration of the OMP algorithm.*

7. **Bonus.** Implement the MP algorithm, which is a weaker version of OMP. Compare their performances in term of number of iterations, objective improvement, *etc...*

4 Convex programming

In this third part, we relax the ℓ_0-norm into an ℓ_1-norm in order to consider fully convex problems. This allows to use modelling languages such as `CVX` and to solve the problem using a broad class of off-the-shelf solvers.

1. Write the three convex problems obtained when replacing the ℓ_0-norm by an ℓ_1-norm.

2. Using the `CVX` modelling language, write functions that define and solve the three ℓ_1-norm problems. *Hint : The documentation of the package is available at* *https://www.cvxpy.org*.

3. Solve the problems with synthetic data and compute performance criteria.

4. A well known result is that the solution of the ℓ_1 version of ($\mathcal{P}_\lambda$) is $\mathbf{x}^\star = \mathbf{0}$ when $\lambda \geq \lambda_{\max} = \|\mathbf{A}^{\mathrm{T}}\mathbf{y}\|_\infty$. Based on this information, select a range of λ decreasing from $\lambda_{\max}$ to some $\lambda_{\min}$ and plot different statistics of the solutions obtained when solving the problem.

5. Comment you graphics. How the two terms of the objective evolve with λ ? How the solution support size evolves with λ ? How the reconstruction statistics evolve with λ ?

5 Applications

In this final part, we will try to reconstruct an image $\mathbf{X} \in \mathbf{R}^{p \times p}$ that is vectorized into some vector $\mathbf{x} \in \mathbf{R}^n$ with $n = p^2$. We assume that $\mathbf{x}$ admits a sparse representation in some basis $\mathbf{V} \in \mathbf{R}^{n \times r}$, that is $\mathbf{x} = \mathbf{V}\boldsymbol{\alpha}$ with $\boldsymbol{\alpha} \in \mathbf{R}^r$ sparse. Moreover, we use a sensing matrix $\mathbf{M} \in \mathbf{R}^{m \times n}$ allowing to access some observation $\mathbf{y} = \mathbf{M}\mathbf{x} \in \mathbf{R}^m$ of the original image. Our goal is therefore to find some sparse $\hat{\boldsymbol{\alpha}}$ verifying $\mathbf{y} \simeq \mathbf{M}\mathbf{V}\hat{\boldsymbol{\alpha}} = \mathbf{A}\hat{\boldsymbol{\alpha}}$.

1. Load the image `imagette.png` as a matrix $\mathbf{X} \in \mathbf{R}^{64 \times 64}$ and display it.
2. With the different algorithms used in this TP, compute approximations $\hat{\boldsymbol{\alpha}}$ of $\boldsymbol{\alpha}$ and display the reconstructed images $\hat{\mathbf{x}} = \mathbf{V}\hat{\boldsymbol{\alpha}}$.
3. Try to vary the number of observations m with respect to the dimension n. What do you observe ?

The following portion of code allows you to generate the matrices $\mathbf{M}$, $\mathbf{V}$ and the observation $\mathbf{y}$ obtained from $\mathbf{X}$ with the function

```
generate_image_data(img_file, mn_ratio, c)
```

The parameter `img_file` must point to the image file path. The parameter `mn_ratio` corresponds to the ratio m/n between the dimensions of the matrix $\mathbf{M}$. Finally, the parameter `c` is the square root of the dimension r of the subspace considered.

```
import numpy as np
from scipy.fftpack import dct, idct
from scipy.fft import dctn, idctn

def image_to_vector(img):
  p, p = img.shape
  return np.reshape(img, (p*p,))

def vector_to_image(vect):
  n = vect.shape[0]
  p = int(np.sqrt(n))
  return np.reshape(vect, (p, p))

def get_subspace_image(c, p):
    if c > p:
        raise ValueError("Parameter c must be lower than p")
    n = p**2
    V = np.zeros((n, c**2))
    counter = 0
```

```
    for i in range(c):
        for j in range(c):
            e = np.zeros((p, p))
            e[i, j] = 1
            V[:, counter] = image_to_vector(idctn(e, type=2))
            counter += 1
    return V

def generate_image_data(img_file, mn_ratio, c):
  X = mpimg.imread(img_file)
  x = image_to_vector(X)
  n = x.size
  p = int(np.sqrt(n))
  m = int(round(mn_ratio * n))
  V = get_subspace_image(c, p)
  M = np.random.randn(m, n)
  y = M @ x
  return V, M, y
```