Math 6397
Computational and Mathematical Methods in Data Science
Problem Set 2
Due on Monday, April 29, at 10 PM

1 Background

A general convex model fitting problem can be written in the form

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad l(\mathbf{A}\mathbf{x}-\mathbf{y})+r(\mathbf{x})$$

with parameters $\mathbf{x}\in\mathbb{R}^n$, where $\mathbf{A}\in\mathbb{R}^{m,n}$ is the feature matrix, $\mathbf{y}\in\mathbb{R}^m$ is the output vector, $l:\mathbb{R}^m\to\mathbb{R}$ is a convex loss function and $r:\mathbb{R}^n\to\mathbb{R}$ is a convex regularization function.

2 Assignments

1. We consider the ℓ^1-regularized linear regression problem

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{y}\|_2^2+\alpha\|\mathbf{x}\|_1 \tag{1}$$

where $\alpha>0$ is a scalar regularization parameter. We assume that the $m\times n$ matrix $\mathbf{A}=(a_{ij})_{i,j=1}^{m,n}$ is of size $m=1\,500$ (examples) and $n=5\,000$ (features). We choose $a_{ij}\sim\mathcal{N}(0,1)$. We normalize the columns of $\mathbf{A}$ to have unit ℓ_2-norm. A "true" value $\mathbf{x}_{\text{true}}\in\mathbb{R}^n$ is generated with 100 nonzero entries, each sampled from a $\mathcal{N}(0,1)$ distribution. The associated labels $\mathbf{y}\in\mathbb{R}^m$ are computed as $\mathbf{y}=\mathbf{A}\mathbf{x}_{\text{true}}+\boldsymbol{\eta}$, where $\eta\sim\mathcal{N}(\mathbf{0},1\text{e}{-3}\mathbf{I})$. This corresponds to a signal-to-noise ratio $\mathbf{A}\mathbf{x}_{\text{true}}\|_2^2/\|\boldsymbol{\eta}\|_2^2$ of about 60.

The ADMM form of this problem can be written as

$$\begin{aligned}\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f(\mathbf{x})+g(\mathbf{z})\\ \text{subject to} \quad & \mathbf{x}-\mathbf{z}=0\end{aligned}$$

where $f(\mathbf{x})=(1/2)\|\mathbf{A}\mathbf{x}-\mathbf{y}\|_2^2$ and $g(\mathbf{z})=\alpha\|\mathbf{z}\|_1$. We define $\Phi(\mathbf{x},\mathbf{z})=f(\mathbf{x})+g(\mathbf{z})$ The augmented Lagrangian of the problem above is given by

$$\begin{aligned}\ell_\rho(\mathbf{x},\boldsymbol{\nu}) &= f(\mathbf{x})+g(\mathbf{z})+\boldsymbol{\nu}^{\mathsf{T}}(\mathbf{x}-\mathbf{z})+\rho\|\mathbf{x}-\mathbf{z}\|_2^2\\ &= \frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{y}\|_2^2+\alpha\|\mathbf{z}\|_1+\frac{\rho}{2}\|\mathbf{x}-\mathbf{z}+\mathbf{u}\|_2^2\end{aligned} \tag{2}$$

with $\mathbf{u}=\boldsymbol{\nu}/\rho$.

We summarize the ADMM steps in Alg. 1. We note that $\mathbf{A}^{\mathsf{T}}\mathbf{A}+\rho\mathbf{I}\succ\mathbf{0}$ for $\rho>0$. Notice that the $\mathbf{x}$ update is essentially a ridge regression. You can solve the associated linear system using a direct solver. The operator S_κ in Alg. 1 is given by $S_\kappa(w)=(w-\kappa)_+-(-w-\kappa)_+$.

Algorithm 1 ADMM steps for solving (1).

1: $\mathbf{x}^{(k+1)} \leftarrow (\mathbf{A}^\mathsf{T}\mathbf{A} + \rho\mathbf{I})^{-1}(\mathbf{A}^\mathsf{T}\mathbf{y} + \rho(\mathbf{z}^{(k)} - \mathbf{u}^{(k)}))$
2: $\mathbf{z}^{(k+1)} \leftarrow S_{\alpha/\rho}(\mathbf{x}^{(k+1)} + \mathbf{u}^{(k)})$
3: $\mathbf{u}^{(k+1)} \leftarrow \mathbf{u}^{(k)} + \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}$

We terminate the solver if the primal residual $\mathbf{r}^{(k)}$ and the dual residual $\mathbf{s}^{(k)}$ are below some given tolerances $\epsilon_{\text{pri}} > 0$ and $\epsilon_{\text{dual}} > 0$, i.e.,

$$\|\mathbf{r}^{(k)}\|_2^2 \leq \epsilon_{\text{pri}} \quad \text{and} \quad \|\mathbf{s}^{(k)}\|_2^2 \leq \epsilon_{\text{dual}}.$$

The primal residual is given by $\mathbf{r}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}$ and the dual residual is given by $\mathbf{s}^{(k+1)} = -\rho(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)})$, respectively. The tolerances for the primal and dual residual can be computed based on

$$\begin{aligned}\epsilon_{\text{pri}} &= \sqrt{n}\epsilon_{\text{abs}} + \epsilon_{\text{rel}} \max\{\|\mathbf{x}^{(k)}\|_2, \|-\mathbf{z}^{(k)}\|_2\} \\ \epsilon_{\text{dual}} &= \sqrt{n}\epsilon_{\text{abs}} + \epsilon_{\text{rel}}\|\rho\mathbf{u}^{(k)}\|_2\end{aligned}$$

where $\epsilon_{\text{abs}} > 0$ and $\epsilon_{\text{rel}} > 0$ are user-defined tolerances. The $\sqrt{n}$ term is introduced to account for the fact that we use a ℓ^2-norm. We select $\rho = 1$ and use the tolerances $\epsilon_{\text{abs}} = 1\text{e}{-4}$ and $\epsilon_{\text{rel}} = 1\text{e}{-2}$. We initialize the variables $\mathbf{u}^{(0)}$ and $\mathbf{z}^{(0)}$ as $\mathbf{0}$. The regularization parameter α is set to $\alpha = \|\mathbf{A}^\mathsf{T}\mathbf{y}\|_\infty/10$.

a) Derive the steps of the ADMM algorithm in Alg. 1 for the $\mathbf{x}$- and the $\mathbf{z}$-update (pencil and paper).

b) Implement the ADMM algorithm. To invert the matrix $(\mathbf{A}^\mathsf{T}\mathbf{A} + \rho\mathbf{I})$, you can in a first step simply call your preferred python implementation of a direct solver for linear systems. You can use the numpy function `linalg.solve` to solve this linear system.

c) If we apply the Woodbury matrix identity to $(\mathbf{A}^\mathsf{T}\mathbf{A} + \rho\mathbf{I})^{-1}$ we obtain

$$(\rho\mathbf{I}_n + \mathbf{A}^\mathsf{T}\mathbf{A})^{-1} = \mathbf{I}_n - \rho\mathbf{A}^\mathsf{T}(\mathbf{I}_m + \rho\mathbf{A}\mathbf{A}^\mathsf{T})^{-1}\mathbf{A}$$

This allows us to reduce the computational complexity of our problem since $\mathbf{A} \in \mathbb{R}^{m,n}$ is of size $m = 1\,500$ and $n = 5\,000$, i.e., $m < n$. That is, instead of inverting a matrix of size $n \times n$, we invert a matrix of size $m \times m$. Since $\mathbf{M} = \mathbf{I}_m + \rho\mathbf{A}\mathbf{A}^\mathsf{T} \succ \mathbf{0}$ we can apply a Cholesky factorization to obtain the lower triangular matrix $\mathbf{L}$ and $\mathbf{U} = \mathbf{L}^\mathsf{T}$. Instead of directly solving for $\mathbf{x}^{(k+1)}$ based on the system $(\mathbf{A}^\mathsf{T}\mathbf{A} + \rho\mathbf{I})\mathbf{x}^{(k+1)} = (\mathbf{A}^\mathsf{T}\mathbf{y} + \rho(\mathbf{z}^{(k)} - \mathbf{u}^{(k)}))$ use the Cholesky factorization of $\mathbf{M}$ to solve for $\mathbf{x}^{(k+1)}$ instead.

We illustrate the convergence behavior of the solver in Fig. 1. A function that generates a random dataset for $\mathbf{x}_{\text{true}}$ (to evaluate the performance) and $\mathbf{y}$ and $\mathbf{A}$ can be found here.

2. We are considering the problem of denoising a signal. We can formulate this task as the following regularized optimization problem:

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|_2^2 + \alpha\sum_{i=1}^{n-1}|x_{i+1} - x_i|. \tag{3}$$

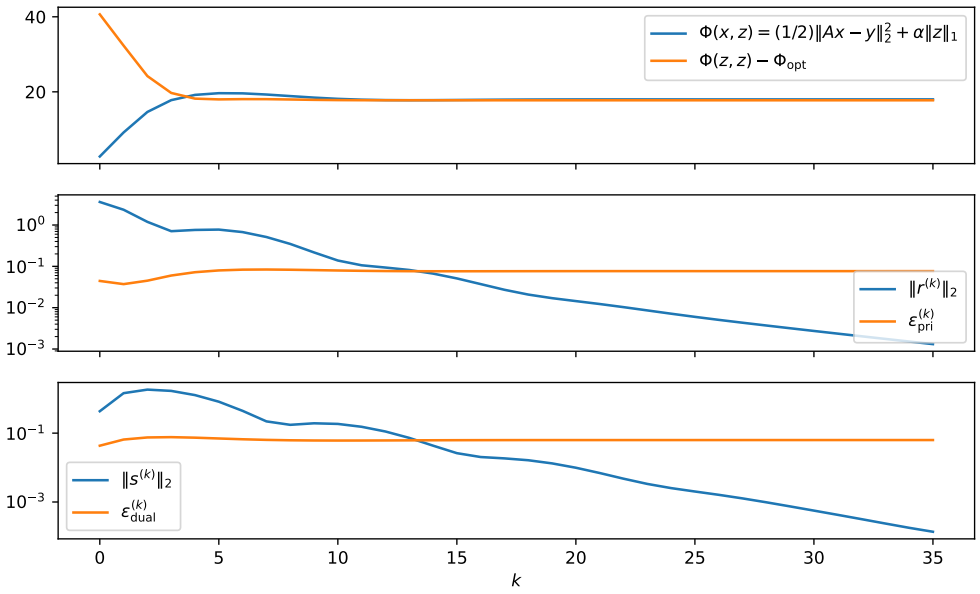

Figure 1: Convergence behavior of the ADMM algorithm. Top: trend of the objective Φ as a function of $(\mathbf{x}^{(k)}, \mathbf{z}^{(k)})$ and the trend of the suboptimality versus the number of iterations k. Here, $\Phi_{\text{opt}} = \Phi(\mathbf{x}_{\text{true}}, \mathbf{x}_{\text{true}})$. Notice the difference in these two plots; for the second plot we evaluate the objective in (1) for $\mathbf{z}^{(k)}$; the first plot corresponds to the objective in the ADMM consensus form in (2). Middle: Trend of the primal residual and the associated stopping tolerance. Bottom: Trend of the dual residual and the associated stopping tolerance. Notice that the stopping conditions are satisfied after 15 iterations. The plots are for 35 iterations to show that the residuals continue to decrease (i.e., we continue to make progress).

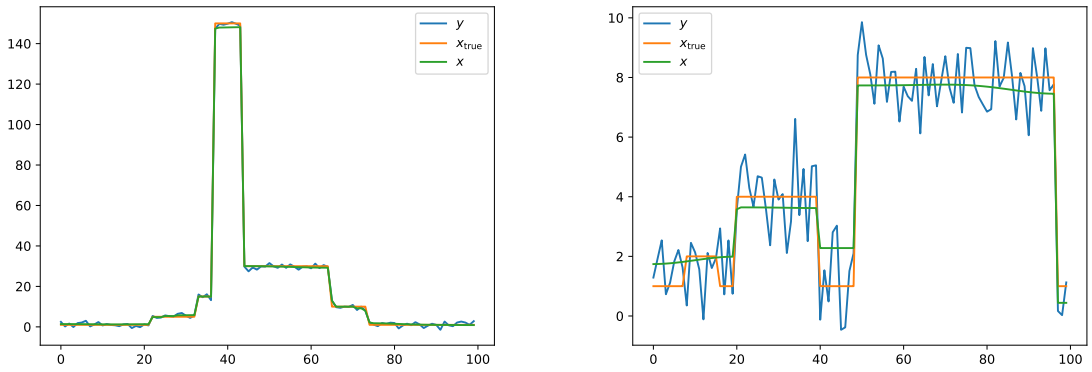

Figure 2: Exemplary results for the solution of the problem in (3).

The regularization model is referred to as total variation. We can express the regularization operator based on a difference matrix $\mathbf{F} \in \mathbb{R}^{n-1,n}$,

$$F_{ij} = \begin{cases} 1 & j = i+1 \\ -1 & j = i \\ 0 & \text{otherwise.} \end{cases}$$

We are going to consider ADMM algorithm for the solution of this problem. The ADMM form of the problem is

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + \alpha\|\mathbf{z}\|_1 \\ \text{subject to} \quad & \mathbf{F}\mathbf{x} - \mathbf{z} = \mathbf{0} \end{aligned}$$

a) Derive the ADMM steps for the algorithm.

b) Implement the ADMM algorithm. You can use the numpy function `linalg.solve` to solve this linear system that appears in the update for $\mathbf{x}$. Note, that this function only works for dense matrices. If your matrix is sparse you can use the `todense` command.

A function that generates a random dataset for $\mathbf{x}_{\text{true}}$ (to evaluate the performance) and $\mathbf{y}$ can be found here. Set the regularization parameter to $\alpha = 5$. We select $\rho = 1$ and use the tolerances $\epsilon_{\text{abs}} = 1\text{e}{-4}$ and $\epsilon_{\text{rel}} = 1\text{e}{-2}$. We initialize the variables $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $\mathbf{u}^{(0)} \in \mathbb{R}^n$, and $\mathbf{z}^{(0)} \in \mathbb{R}^n$ as $\mathbf{0}$. To generate the matrix $\mathbf{F}$ you can use the command

```
F = sp.sparse.spdiags([e,-e], [0,1], n, n )
```

where $\mathbf{e} \in \mathbb{R}^n$ is a vector of ones.

3. We are going to design an algorithm to cluster data. We are going to consider spectral clustering to do so. A `sklearn` implementation is implemented in skl_spectral_clustering.py. We are going to reimplement this method. We use a dataset readily available in `sklearn`. This dataset is visualized in skl_clusterdat_viz.py. Consider the second dataset called `noisy_moons`. Use a noise level of 0.05 and 500 examples. The features are given by $\mathbf{X} \in \mathbb{R}^{500,2}$. The classes are $\mathbf{y} \in \{0,1\}^{500}$, where 250 examples belong to class 0 and 250 examples belong to class 1. To do the clustering you need to do the following steps:

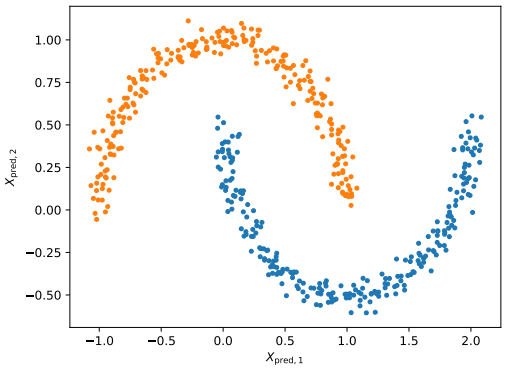

Figure 3: Spectral clustering results..

a) construct a graph for the features $\mathbf{X}$

b) construct the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix (diagonal matrix consisting of the row-sum of $\mathbf{A}$ and $\mathbf{A}$ is the weighted adjacency matrix, i.e., an adjacency matrix for the graph where the entries with 1 are replaced by the value for the distance functional used to construct the graph).

c) compute the eigenvalue decomposition of $\mathbf{L} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{T}}$, with $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_{500})$.

d) identify the second largest eigenvalue λ_i and the associated eigenvector $\mathbf{v}_i$

e) if the j-th entry in $\mathbf{v}_i$ is smaller than 0, the corresponding example belongs to class 1, if it is larger than 0 it belongs to class 0

The main difficulty here is to construct the adjacency matrix $\mathbf{A}$. You can use `sklearn` to do so. An example for constructing the NN graph is given in skl_create_nngraph.py. Notice that in this example the entries of $\mathbf{A}$ are 0 or 1. You want the entries to correspond to the metric you use to measure distances between the nodes in the graph. Look at the help for `radius_neighbors_graph` to accomplish this. The remaining options remain identical to those provided in the example. You can compare your results to skl_spectral_clustering.py. The classification result for the code outlined above is shown in Fig. 3.