

Comp790-166: Computational Biology

Lecture 14

March 23, 2021

Announcements

- I owe some of you proposals, etc. Thanks for your patience

Today

- Predicting cognitive scores in a longitudinal dataset of AD patients (integrating clinical, biological and imaging data)
- Optimization Basics
- ADMM

From Last Time : A Joint Model of Cognitive Scores and Diagnosis

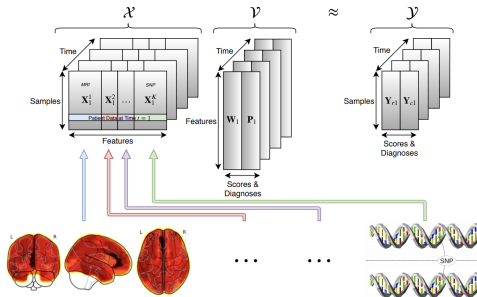


Figure: from Brand, Wang, *et al.* PacSym Biocomputing. 2020.

A Joint Model of Cognitive Scores and Diagnosis, Continued

- The assumption that there is a relationship between the classification and regression tasks
- A patient that performs poorly on cognitive tests is more likely to be diagnosed with AD.
- The overall goal is to find biomarkers (e.g. biological features)

Notation and Problem Formulation

- **Input Features:** $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T\} \in \mathbb{R}^{n \times d \times T}$ This represents patients \times features \times timepoints.
- Note that each \mathbf{X}_t can be broken down across the K modalities as, $\{\mathbf{X}_t\}_{j=1}^K$
- The output diagnoses and cognitive scores are represented by another tensor, $\mathcal{Y} = \{\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T\} \in \mathbb{R}^{n \times c \times T}$
- Each $\mathbf{Y}_t = [\mathbf{Y}_{rt}, \mathbf{Y}_{ct}]$ represents concatenated response variables for regression (r) and classification (c).
- The goal is to learn a tensor, $\mathcal{V} = \{[\mathbf{W}_1, \mathbf{P}_1], [\mathbf{W}_2, \mathbf{P}_2], [\mathbf{W}_T, \mathbf{P}_T]\}$ which represents the coefficients for each feature for regression (\mathbf{W} s) and classification (\mathbf{P} s) across the T timepoints.

A Regularized Joint Learning Model

$$\min_{\mathcal{W}, \mathcal{P}} \mathcal{L}_r(\mathcal{W}) + \mathcal{L}_c(\mathcal{P}) + \mathcal{R}(\mathcal{V})$$

- Here, $\mathcal{L}_r(\mathcal{W})$ and $\mathcal{L}_c(\mathcal{P})$ are the loss functions for the regression and classification tasks, respectively.
- $\mathcal{R}(\mathcal{V})$ is a regularization function applied to the matrix unfolded from the tensor, $\mathcal{V} \rightarrow \mathbf{V}^{d \times cT}$. Here, $\mathbf{V}^{d \times cT}$ is constructed by taking the $(\mathbf{W}_t, \mathbf{P}_t)$ matrix pairs and joining by the columns.

Regularization, $\mathcal{R}(\mathcal{V})$

To associate image and genomic features to cognitive scores and diagnoses over time, apply an $\ell_{2,1}$ norm as,

$$\mathbf{V} : \|\mathbf{V}\|_{2,1} = \sum_{i=1}^d \|\mathbf{v}^i\|_2$$

Next, to capture the impact of each modality (e.g. MRI, SNP, etc), the authors use the group ℓ_1 -norm (G_1 norm) on the rows of \mathbf{V} corresponding to modality j as,

$$\|\mathbf{V}\|_{G_1} = \sum_{j=1}^K \|\mathbf{v}^j\|_2$$

Regularization, $\mathcal{R}(\mathcal{V})$, Continued

Finally, to account for inter-modal relationships (or relatedness of features within a modality to cognitive outcome), they use trace norm regularization of \mathbf{V} as,

$$\mathbf{V} : \|\mathbf{V}\|_* = \sum \sigma_i(\mathbf{V})$$

. Here, $\sigma_i(\mathbf{V})$ are the singular values of \mathbf{V}

Objective

Incorporating the three regularizations, the objective can be specified as follows,

$$\min_{\mathbf{V}} J = \sum_{t=1}^T \left[\|\mathbf{X}_t \mathbf{W}_t - \mathbf{Y}_{rt}\|_F^2 \right] + \sum_{t=1}^T \sum_{i=1}^n \sum_{k=1}^{c_c} \left[\left(1 - (\mathbf{x}^{it} \mathbf{p}_{kt} + b_{kt}) y_{ikt} \right)_+ \right] \\ + \gamma_1 \|\mathbf{V}\|_{2,1} + \gamma_2 \|\mathbf{V}\|_{G_1} + \gamma_3 \|\mathbf{V}\|_*,$$

- The second term is the loss of $c_c \times T$ one-vs-all multi-class SVM
- $y_{ikt} \in \{-1, 1\}$ is the class label associated with the i -th patient at time t
- b_{kt} is the bias associated with the $(k \times t)$ -th SVM
- $(\cdot)_+$ is defined as $(a)_+ = \max(0, a)$

A Coming Attraction for the Optimization: ADMM

- ADMM stands for alternating direction method of multipliers.
- The basic idea is to break up a big problem into sub-problems.
- The complicated objective here will be solved using multi-block ADMM, an extension of regular ADMM as,

We need some theory first....

Quick Intro to Convex Optimization

First regarding **optimization** in general.

- All of us are learning parameters all of the time (even when we do something simple, like regression.....)

The magic here is that if you can formulate your problem as a **convex problem**,

- It can often be solved efficiently (similar to how we can solve least squares problems efficiently)
- Often a challenge is knowing whether or not an objective function is convex
- Sometimes problems can be re-stated as convex problems to use all of the nice theory.

Quick Plug for CVX

- CVX is software for convex optimization.
- It is nice because you can easily specify the objective function and constraints
- <https://www.cvxpy.org/index.html>

An Example for a Classic Least Squares Problem

All of us have seen the following many times before,

$$\min ||Ax - b||_2^2$$

Here A is an $M \times N$ matrix, with a corresponding vector b of the response variables.

Corresponding CVX Code

```
# Import packages.
import cvxpy as cp
import numpy as np

# Generate data.
m = 20
n = 15
np.random.seed(1)
A = np.random.randn(m, n)
b = np.random.randn(m)

# Define and solve the CVXPY problem.
x = cp.Variable(n)
cost = cp.sum_squares(A @ x - b)
prob = cp.Problem(cp.Minimize(cost))
prob.solve()

# Print result.
print("\nThe optimal value is", prob.value)
print("The optimal x is")
print(x.value)
print("The norm of the residual is ", cp.norm(A @ x - b, p=2).value)
```

Who Cares about Least Squares

You can also solve harder problems, like general linear programs.

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & Ax \leq b\end{array}$$

Also easy with CVX

```
# Import packages.
import cvxpy as cp
import numpy as np

# Generate a random non-trivial linear program.
m = 15
n = 10
np.random.seed(1)
s0 = np.random.randn(m)
lamb0 = np.maximum(-s0, 0)
s0 = np.maximum(s0, 0)
x0 = np.random.randn(n)
A = np.random.randn(m, n)
b = A @ x0 + s0
c = -A.T @ lamb0

# Define and solve the CVXPY problem.
x = cp.Variable(n)
prob = cp.Problem(cp.Minimize(c.T @ x),
                  [A @ x <= b])
prob.solve()

# Print result.
print("\nThe optimal value is", prob.value)
print("A solution x is")
print(x.value)
print("A dual solution is")
print(prob.constraints[0].dual_value)
```

Convex Sets

A set C is **convex** if the line segment between any two points in C lies in C . That is, for any $x_1, x_2 \in C$ and for any θ with $0 \leq \theta \leq 1$, we have,

$$\theta x_1 + (1 - \theta)x_2 \in C$$

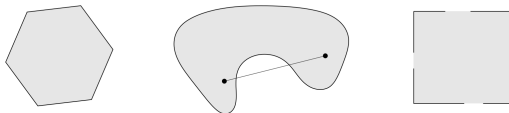


Figure 2.2 Some simple convex and nonconvex sets. *Left.* The hexagon, which includes its boundary (shown darker), is convex. *Middle.* The kidney shaped set is not convex, since the line segment between the two points in the set shown as dots is not contained in the set. *Right.* The square contains some boundary points but not others, and is not convex.

Figure: from the CVX book by Boyd and Vandenberghe

Convex Function

A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex, if the domain of f is a convex set and if for all x and y in the domain of f with $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$



Figure 3.1 Graph of a convex function. The chord (*i.e.*, line segment) between any two points on the graph lies above the graph.

Figure: from the CVX book by Boyd and Vandenberghe. Geometrically, the inequality means that the line segment between $(x, f(x))$ and $(y, f(y))$ lies above f

Examples of Convex Functions

- e^{ax}
- $|x|^p$
- $x \log x$
- x^a
- $ax + b$

Dual Problem

Consider a convex equality constrained optimization problem,

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & Ax = b\end{array}$$

We can write the Lagrangian as,

$$L(x, y) = f(x) + y^T (Ax - b)$$

The idea of Lagrangian duality is to take the constraints into account by augmenting the objective function with a weighted sum of the constraints.

Dual Problem Continued

Given the Lagrangian,

$$L(x, y) = f(x) + y^T (Ax - b)$$

we denote the dual function as,

$$g(y) = \inf_x L(x, y)$$

.

We can also consider the dual problem,

$$\max g(y)$$

Finally recover an optimal value, x^* from a dual optimal, y^* ,

$$x^* = \arg \min_x L(x, y^*)$$

Dual Ascent

Use a gradient method for the dual problem,

$$y^{k+1} = y^k + \alpha^k \nabla g(y^k)$$

In this case, the gradient is,

$$\nabla g(y^k) = A\tilde{x} - b,$$

where

$$\tilde{x} = \operatorname{argmin}_x L(x, y^k)$$

So, the dual ascent method is,

$$x^{k+1} := \operatorname{argmin}_x L(x, y^k) \quad // x\text{-minimization}$$

$$y^{k+1} := y^k + \alpha^k (Ax^{k+1} - b) \quad // \text{dual update}$$

Dual Decomposition

Suppose f is separable :

$$f(x) = f_1(x_1) + \cdots + f_N(x_N), \quad x = (x_1, \dots, x_N)$$

Then L is separable in x :

$$L(x, y) = L_1(x_1, y) + \cdots + L_N(x_N, y) - y^T b$$

with

$$L_i(x_i, y) = f_i(x_i) + y^T A_i x_i$$

Dual Decomposition, Continued

x -minimization in dual ascent splits into N separate minimizations can be carried out in parallel.

$$x_i^{k+1} := \operatorname{argmin}_{x_i} L_i(x_i, y^k)$$

Updates are,

$$\begin{aligned} x_i^{k+1} &:= \operatorname{argmin}_{x_i} L_i(x_i, y^k), \quad i = 1, \dots, N \\ y^{k+1} &:= y^k + \alpha^k \left(\sum_{i=1}^N A_i x_i^{k+1} - b \right) \end{aligned}$$

Augmented Lagrangian

For $\rho > 0$, form the augmented Lagrangian,

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)\|Ax - b\|_2^2$$

- The augmented Lagrangian was developed to in part bring robustness to the dual ascent method, and to encourage convergence without strict assumptions about convexity of f .

Applying dual ascent (also called here method of multipliers),

$$\begin{aligned}x^{k+1} &:= \underset{x}{\operatorname{argmin}} L_\rho(x, y^k) \\ y^{k+1} &:= y^k + \rho(Ax^{k+1} - b)\end{aligned}$$

Alternating Direction Method of Multipliers

The ADMM problem form (where f and g are convex) is,

$$\begin{array}{ll}\text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c\end{array}$$

You can further write down the associated augmented Lagrangian as,

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2$$

ADMM will therefore do updates as follows,

$$x^{k+1} := \operatorname{argmin}_x L_\rho(x, z^k, y^k)$$

$$z^{k+1} := \operatorname{argmin}_z L_\rho(x^{k+1}, z, y^k)$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$