به نام خدا

گزارش پروژه

درس : بهینه سازی محدب ۱

استاد : روح الله امیری

تهیه شده توسط: مهیار افشین‌مهر – ۴۰۱۰۹۷۳۵

ایلیا محروق – ۴۰۱۱۰۰۱۱

مهشاد مرادی - ۴۰۱۰۹۳۷۳

دانشگاه صنعتی شریف

دانشکده‌ی مهندسی برق

❖ مقدمه و شرح مسئله

مسائل مدرن پردازش داده عموما با داده‌های ساختارمند سر و کار دارند، به طوری که عموما مقادیر سیگنال‌ها روی رئوس گرافی بدون جهت و وزن دار تعریف شده‌اند. چنین داده‌هایی را graph signal می‌نامیم. در این نوع گراف‌ها رئوس entity های داده هستند و وزن یال‌ها ارتباط میان دو راس را با هم نشان می‌دهد. برای هر راس مقادیر سیگنال‌های ناشی از اندازه‌گیری‌های فیزیکی یا ... گزارش شده‌اند. از کاربردهای این نوع گراف‌ها می‌توان به مواردی مانند دمای مناطق جغرافیایی و ... اشاره کرد.

فرض کنید داده‌ای از سیگنال‌های (گسسته) مشاهده شده از رئوس دارید، هدف یادگیری گرافی با وزن‌های مشخص است که بهترین بازنمایی از داده ما را داشته باشد. برای اینکه تعریف درستی از بازنمایی خوب برای سیگنال‌ها داشته باشیم، نیاز به معیاری مشخص داریم تا گراف‌ها را از هم متمایز کند. بنابراین به دنبال گرافی هستیم که سیگنال مربوطه روی این گراف smooth باشد. یک graph signal را smooth گوییم اگر مقادیر سیگنال‌های مربوط به دو سر یال‌های با وزن نسبتا زیاد، مشابه هم باشند.

برای یادگیری وزن‌های گراف، می‌توان ماتریس Laplacian مربوط به گراف را تخمین زد. گراف وزن دار و بدون جهت $G = (V, E, w)$ متشکل از n راس را در نظر بگیرید که در آن V مجموعه رئوس و E مجموعه یال‌ها است. همچنین w یک مقدار نامنفی به عنوان وزن به هر یال نسبت می‌دهد. یک graph signal با تابعی مانند $f: V \to R^n$ نمایش داده می‌شود که یک اسکالر به هر راس نسبت می‌دهد. ماتریس L را ماتریس Laplacian (unnormalized graph Laplacian matrix) می‌نامیم و داریم:

$$L = D - W$$

که در آن W ماتریس مجاورت وزن دار و D ماتریسی قطری است که درجه‌ی رئوس روی قطر آن قرار می‌گیرد. ماتریس L ماتریسی مثبت نیمه معین است که به صورت یکتا گراف را مشخص می‌کند. به عنوان مثال اگر گراف از c مولفه همبندی تشکیل شده باشد، تکرر ویژه مقدار برابر صفر برا این ماتریس c است و $rank(L) = n - c$ خواهد بود. میزان *smoothness* یک سیگنال روی یک مدار با عبارت زیر اندازه‌گیری می‌شود:

$$f^T L f = \sum_{i,j} w_{ij} (f(i) - f(j))^2$$

که در آن w_{ij} وزن یال میان دو راس i و j است و $f(i)$ و $f(j)$ مقادیر سیگنال مشاهده شده در این دو راس هستند. هدف پیدا کردن ماتریس L است به طوری که عبارت بالا کمینه باشد. بنابراین رئوسی که با یال با وزن بزرگ به هم متصل شده‌اند، عملا مقادیر سیگنال مشابه داشته‌اند. دقت کنید که سیگنال‌های مشاهده شده می‌توانند دارای نویز نیز باشند.

❖ الگوریتم یادگیری

مسئله یادگیری گراف را می‌توان به صورت مسئله بهینه‌سازی زیر نشان داد:

$$\min_{L \in R^{n \times n}, Y \in R^{n \times p}} \|X - Y\|_F^2 + \alpha\, tr(Y^T L Y) + \beta \|L\|_F^2$$

$$\begin{aligned} s.t. \quad & tr(L) = n \\ & L_{ij} = L_{ji} \le 0, i \neq j \\ & L \,.\, 1 = 0 \end{aligned}$$

ماتریس $X \in R^{n \times p}$ دارای p سیگنال نویزی به ازای هر راس گراف به عنوان داده است. مقادیر α و β نیز ضرایب regularization در این معادله هستند. ماتریس‌های Y و L به ترتیب ورژن بدون نویز سیگنال‌ها و ماتریس Laplacian گراف وزن‌دار مربوطه هستند که یاد گرفته می‌شوند. در واقع حل این مسئله به ما مقادیر بدون نویز از سیگنال‌ها و همچنین ساختار گراف را خروجی می‌دهد. دقت کنید که عبارت $\|X - Y\|_F^2$ مشابه بودن سیگنال‌های بدون نویز به سیگنال‌های نویز دار را مدل می‌کند. همچنین α ضریبی برای کنترل smoothness سیگنال‌ها در گراف حاصل اند (تعریف smoothness برای یک سیگنال در بخش قبل ارائه شده است). شرط اول (شرط دارای trace) به عنوان یک جمله normalization عمل کرده و باعث می‌شود تا کمینه‌های بدیهی مسئله feasible

نباشند. همچنین دو شرط دیگر روی ماتریس L باعث می‌شوند تا L یک ماتریس Laplacian باشد. همچنین جمله پنالتی $\beta\|L\|_F^2$ به تابع objective اضافه شده تا توزیع درایه‌های غیر قطری L را کنترل کند و عملا β مقداری برای تعیین sparsity در L است.

این مسئله نسبت به دو متغیر Y و L به صورت همزمان محدب نیست. به همین دلیل برای حل آن از تکنیک متفاوتی استفاده می‌کنیم. در هر گام یکی از متغیرها را ثابت گرفته و مسئله را برای متغیر دیگر حل می‌کنیم. این کار باعث تولید نقاط بهینه‌ی محلی می‌شود. به تکرار این عملیات به global minimum همگرا خواهیم شد. برای اینکار در هر گام ابتدا متغیر Y را ثابت فرض کرده و مسئله زیر را حل می‌کنیم:

$$\begin{aligned}\min_{L}\ \ & \alpha\, tr(Y^TLY)+\beta\|L\|_F^2\\ s.t.\ \ & tr(L)=n\\ & L_{ij}=L_{ji}\le 0, i\ne j\\ & L\,.1=0\end{aligned}$$

سپس L را ثابت گرفته و مسئله زیر را حل می‌کنیم:

$$\min_{Y}\ \ \|X-Y\|_F^2+\alpha\, tr(Y^TLY)$$

هر دو این مسائل محدب بوده و دارای یک کمینه‌ی یکتا هستند. ابتدا به مسئله اول می‌پردازیم. برای تبدیل مسئله به فرم محدب، از فرم vectorize شده‌ی ماتریس L استفاده می‌کنیم که با $vec(L)$ نمایش می‌دهیم. ماتریس L متقارن است و می‌توانیم از فرم half-vectorized آن ($vech(L)\in R^{\frac{n(n+1)}{2}}$) استفاده کنیم. این دو فرم با تساوی زیر و با ماتریس M_{dup} به هم قابل تبدیل اند:

$$M_{dup}vech(L)=vec(L)$$

همچنین داریم:

$$tr(Y^TLY)=vec(YY^T)^Tvec(L)\,,\quad \|L\|_F^2=vec(L)^Tvec(L)$$

با استفاده از این دو رابطه مسئله اول به فرم *QP* (نسبت به متغیر $vech(L)$) زیر تبدیل می‌شود و با روش‌های *interior point* به خوبی قابل حل است: (ماتریس‌های A و B برای برقراری شروط تساوی و نامساوی مسئله اصلی تعریف می‌شوند)

$$\begin{aligned}\min_{vech(L)}\ \ & \alpha\, vec(YY^T)^TM_{dup}vech(L)+\beta vech(L)^TM_{dup}^TM_{dup}vech(L)\\ s.t.\ \ & A\, vech(L)=0\\ & B\, vech(L)\le 0\end{aligned}$$

همانطور که مشاهده می‌کنید، پیچیدگی محاسباتی حل مسئله به صورت *quadratic* با تعداد رئوس گراف افزایش می‌یابد. به همین دلیل در مسائل با ابعاد بزرگ، می‌توانیم از روش‌های *operator splitting* مانند *ADMM* برای پیدا کردن پاسخ استفاده کنیم.

مسئله دوم، مسئله‌ای بدون قید است. بنابراین به سادگی با قرار دادن گرادیان تابع هدف برابر صفر، می‌توانیم فرم بسته پاسخ آن را بدست آوریم. فرم بسته برای ماتریس Y بهینه در این مسئله به شکل زیر خواهد بود:

$$Y=(I_n+\alpha L)^{-1}X$$

برای محاسبه وارون ماتریس $I_n+\alpha L$ نیز می‌توان از تجزیه *Cholesky* استفاده کرد. شبه کد زیر الگوریتم حل مسئله بهینه سازی را نمایش می‌دهد: (معادله‌های ۱۷ و ۱۸ در شبه کد به ترتیب همان دو مسئله بهینه سازی هستند که در بالا معرفی کردیم.)

Algorithm 1: Graph Learning for Smooth Signal Representation (**GL-SigRep**).

1: **Input:** Input signal X, number of iterations iter, α, β
2: **Output:** Output signal Y, graph Laplacian L
3: **Initialization:** $Y = X$
4: **for** $t = 1, 2, ..., \text{iter}$ **do:**
5: **Step to update Graph Laplacian** L**:**
6: Solve the optimization problem of Eq. (17) to update L.
7: **Step to update** Y**:**
8: Solve the optimization problem of Eq. (18) to update Y.
9: **end for**
10: $L = L^{\text{iter}}, Y = Y^{\text{iter}}$.

این الگوریتم در بخش *Model* در فایل *Convex_Optimization_Project.ipynb* پیاده‌سازی و ضمیمه شده است. دقت کنید که در این پیاده‌سازی، مسئله اول به همان فرم اولیه نوشته شده و توسط *CVXPY* حل شده است. اما نقطه بهینه مسئله دوم مستقیما از روی فرم بسته ارائه شده محاسبه شده است. در انتها نیز تا حداکثر ۲۰ تکرار و یا همگرایی تابع هدف با دقت 10^{-7} این فرایند تکرار شده است و ماتریس‌های Y و L نهایی خروجی داده شده‌اند.

❖ تست و بررسی خروجی

برای بررسی عملکرد الگوریتم پیاده‌سازی شده، داده‌هایی را به صورت تصادفی تولید کرده و عملکرد کد را روی آن‌ها بررسی می‌کنیم. تمام کدهای مربوط به این بخش در فایل Convex_Optimization_Project.ipynb پیوست شده است. ۳۰ راس برای گراف نمونه در نظر می‌گیریم و در episode ها مختلف (به تعداد ۲۰۰ تا) مقدار سیگنال مشاهده شده برای هر راس را در داده‌ها قرار می‌دهیم. برای تولید تصادفی سیگنال‌ها، ابتدا رئوس را به ۵ دسته به صورت تصاقی (تعداد اعضای دسته‌ها لزوما برابر نیست) تقسیم می‌کنیم. برای سیگنال مشاهده شده هر گروه یک میانگین در نظر گرفته و در هر episode به هر راس یک گروه داده‌ای تصاقی با همان میانگین و مقداری نویز نسبت می‌دهیم. تکه کد زیر این اعمال را نمایش می‌دهد:

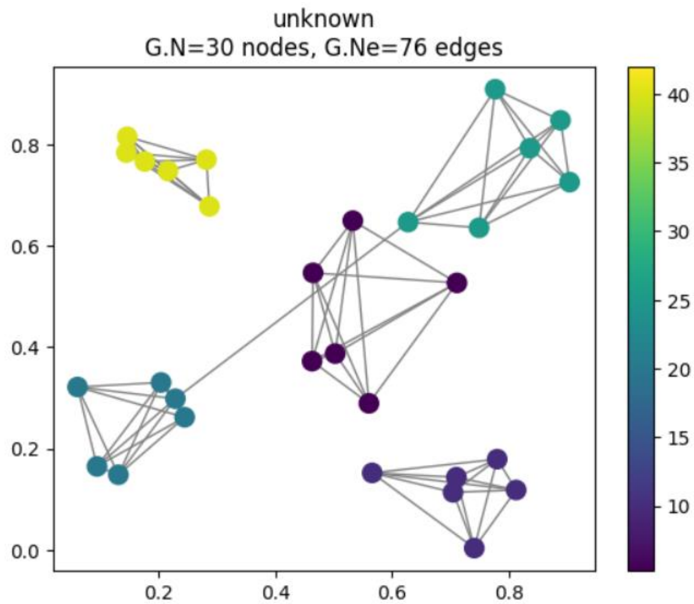
```python
episodes = 200
total_nodes = 30
nodes = [i for i in range(total_nodes)]

def split_list(lst, k):
    random.shuffle(lst)
    split_size = len(lst) // k
    splits = [lst[i * split_size:(i + 1) * split_size] for i in range(k)]
    for i in range(len(lst) % k):
        splits[i].append(lst[-(i + 1)])
    return splits

splits = split_list(nodes, 5)
means = [20, 40, 10, 5, 25]
X = []
for i in range(episodes):
  row = [0] * total_nodes
  for group, num in zip(splits, range(len(splits))):
    for j in group:
      row[j] = means[num] + float(np.random.normal(0, 1, 1))
  X.append(row)
X = np.array(X).T

node_means = [0] * total_nodes
for group, num in zip(splits, range(len(splits))):
    for j in group:
      node_means[j] = means[num]
```

این نمونه‌ها می‌توانند نمایش دهنده دمای نقاط جغرافیای مختلف باشند. به طوری که نقاطی که در یک گروه قرار دارند، نقاط نزدیک به هم بوده و دمای مشابهی دارند (میانگین یکسان دارند و تنها تفاوت آن‌ها نویز است). حال مدل پیاده سازی شده را با $\alpha = 10^{-4}$ و $\beta = 10^{-2}$ روی داده‌ها آموزش می‌دهیم. برای مشاهده خروجی، از pygsp جهت visualization استفاده کرده‌ایم. همانطور که گفتیم داده‌ها می‌توانند نمایش دهنده دمای نقاط جغرافیایی مختلف باشند. به همین دلیل برای رسم گراف نقاط هر گروه را به صورت رندوم در فواصل نزدیک به هم قرار می‌دهیم. همچنین یال‌هایی را در گراف رسم می‌کنیم که وزن معناداری داشته باشند (بالای 10^{-7}). تصویر زیر خروجی الگوریتم را نشان می‌دهد:



همانطور که مشاهده می‌کنید، بین نقاط نزدیک به هم که دمای نزدیکی داشتند، یال رسم شده است. به عبارتی الگوریتم داده شده، با دقت بسیار خوبی گراف مورد نظر را یاد می‌گیرد.

برای *visualization* از کتابخانه *network* نیز استفاده شده که در فایل پیوست آمده است.

❖ منابع

1. https://web.media.mit.edu/~xdong/paper/tsp16.pdf