Efficient monotone submodular maximization subject to matroid constraints in Submodlib

CS 769: Project

Eeshaan Jain, Ipsit Mantri, Sibasis Nayak

Indian Institute of Technology Bombay

Overview (1)

- Maximization of monotone submodular functions subject to cardinality constraints has been a topic of interest
- For a complicated constraint, greedy algorithm doesn't work well anymore
- For a matroid constraint, the greedy solution provides a $\frac{1}{2}$ approximation of the optimal
- Continuous Greedy provides a $\left(1 - \frac{1}{e}\right)$ approximation of OPT
- Accelerated Continuous Greedy provieds a $\left(1 - \frac{1}{e} - \epsilon\right)$ solution

Overview (2)

- SUBMODLIB - Submodualar optimization library in Python with C++ optimization engine
- Currently only facilitates cardinality constraints
- The goal of the project is to understand the various algorithm based around matroid constraints
- Study the various points of improvement and implement them in the submodlib fashion
- Future Work: discuss with SUBMODLIB team and merge with the master branch

Submodularity

- A set function $f : 2^V \to \mathbb{R}$ is said to be *submodular*, if for all $A, B \subseteq V$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

- A set function f is said to be *monotone* if $f(A) \geq f(B)$ whenever $A \supseteq B$.
- The set function f is said to be *non-negative* if the co-domain of f is $\mathbb{R}_+$
- When considering the optimization problem at hand, we will consider non-negative monotone submodular functions

Smooth submodularity

- A function $F : [0,1]^V \to \mathbb{R}_+$ is *smooth submodular* if

$$F(x) + F(y) \geq F(x \vee y) + F(x \wedge y)$$

  where $(x \vee y) = \max\{x, y\}$ and $(x \wedge y) = \min\{x, y\}$, both in an coordinate-wise fashion
- F is *monotone* if for $x \leq y$ coordinate-wise, $F(x) \leq F(y)$
- $F : [0,1]^V \to \mathbb{R}$ is smooth monotone submodular if
  1. (Smoothness) $F \in \mathcal{C}_2([0,1]^V)$, i.e the second-order partial derivatives exists everywhere
  2. (Monotonicity) For each $j \in V$, $\frac{\partial F}{\partial y_j} \geq 0$ everywhere
  3. (Submodularity) For any $i, j \in V$, $\frac{\partial^2 F}{\partial y_i \partial y_j} \leq 0$ everywhere

Matroids (1)

- A matroid $\mathcal{M} = (E, \mathcal{I})$ is a structure with a finite ground set E, the universe and a collection of subsets of 2^E (power set of E), $\mathcal{I}$ called independent sets, such that
  1. $\emptyset \in \mathcal{I}$
  2. $\forall\, I \in \mathcal{I}$ and $J \subseteq I$, $J \in \mathcal{I}$
  3. (Exchange axiom) $\forall\, I, J \in \mathcal{I}$ and $|I| < |J|$, there exists $x \in J \setminus I$ such that $I \cup \{x\} \in \mathcal{I}$
- A base B of a matroid $\mathcal{M}$ is a maximal independent set, and the exchange axiom guarantees that all bases of $\mathcal{M}$ have the same cardinality
- The rank function of a matroid $\mathcal{M} = (E, \mathcal{I})$ is a set function $r_{\mathcal{M}} : 2^E \rightarrow \mathbb{N}$ such that for any $S \subseteq E$
$$r_{\mathcal{M}}(S) = \max\{|X| : X \subseteq S, X \in \mathcal{I}\}$$
- The matroid-rank theorem states that $r : 2^E \rightarrow \mathbb{N}$ is a rank function for a matroid *if and only if*
  1. $r(\emptyset) = 0$ and $r(A \cup \{e\}) - r(A) \in \{0, 1\}\ \forall\, A \subseteq E, e \in E$
  2. r is submodular, i.e for any $S, T \subseteq E$ we have $r(S) + r(T) \geq r(S \cup T) + r(S \cap T)$

Matroids (2)

- Given a a matroid $\mathcal{M} = (E, \mathcal{I})$, the matroid polytope $P(\mathcal{M})$ is defined as

$$P(\mathcal{M}) = \mathrm{conv}(\{x_S \in \mathbb{R}^{|E|} : S \in \mathcal{I}\})$$

We can also write that

$$\begin{aligned} P(\mathcal{M}) = \{x \in \mathbb{R}^{|E|} : x(S) \leq r(S)\ \forall\ S \subseteq E \\ x_e \geq 0\ \forall\ e \in E\} \end{aligned}$$

where $x(S) = \sum_{e \in S} x_e$.

- Now, the base of a matroid can be defined as the set $S \in \mathcal{I}$ such that $r_{\mathcal{M}}(S) = r_{\mathcal{M}}(E)$ and the base polytope is defined as

$$B(\mathcal{M}) = \left\{ y \in P(\mathcal{M}) : \sum_{i \in E} y_i = r_{\mathcal{M}}(E) \right\}$$

Cardinality Constraints

- The optimization problem at hand

$$\max_S f(S) \text{ over } S \in \mathcal{F}$$

  where $\mathcal{F}$ is a constraint.
- Cardinality constraint has the form $|S| \leq k$ for some $k \in \mathbb{R}_+$
- Commonly seen in tasks such as influence maximization or sensor placement
- Greedy algorithm works well and is fast, giving a $\left(1 - \frac{1}{e}\right)$ approximation in $O(kn)$ time
- Further improvements exist to get the time down to $\widetilde{O}(n)$

Matroid Constraints

- Given $\mathcal{M} = (E, \mathcal{I})$, the problem
$$\max_{S \in \mathcal{I}} f(S)$$
- Classical Greedy only gives a $\frac{1}{2}$ approximation for this problem - not that great
- Continuous Greedy solves this problem - relax the problem to a continuous one!
- For a monotone submodular set function $f : 2^V \to \mathbb{R}_+$, a canonical extension to a smooth monotone submodular function can be obtained
- For $y \in [0,1]^V$, let $\hat{y}$ denote a random vector in $\{0,1\}^V$ where each coordinate is independently rounded to 1 with probability y_j. $\hat{y} \in \{0,1\}^V$ can be identified with $R \subseteq V$ with the indicator given as $\hat{y} = \mathbf{1}_R$. Then the multilinear function F (multilinear $\implies \frac{\partial^2 F}{\partial y_j^2} = 0$) can be defined as

$$F(y) = \mathbb{E}[f(\hat{y})] = \sum_{R \subseteq X} f(R) \prod_{i \in R} y_i \prod_{j \in R} (1 - y_j)$$

Submodular functions with Matroid constraints

- Submodular Welfare Problem
  - Special case of Social Welfare Maximization, when the utility functions are submodular
  - Given a set X of m items, and n players each of which as a utility function $w_i : 2^X \rightarrow \mathbb{R}_+$, the goal is to partition X into disjoint subsets $S_1, \cdots, S_n$ inorder to maximize the social welfare given as $\sum_{i=1}^{n} w_i(S_i)$
  - Main result - with appropriate redefinitions, you can convert the problem into a matroid constraint one!
- Separable Assignment Problem
  - The problem consists of m items and n bins. Each bin has a collection of feasible sets F_j satisfying down-closure (if $A \in F_j$ and $B \subseteq A$, then $B \in F_j$)
  - Each item has a value v_i^j depending on the bin it is placed in. In SAP, we need to choose disjoint feasible sets $S_j \in F_j$ to maximize $\sum_j \sum_i v_i^j$
  - Similar to above, we can reduce the problem to a matroid constraint one
- Generalized Assignment Problem
  - Special case of SAP where the feasible set is a knapsack constraint given as $F_j = \{S : \sum_i s_i^j \leq 1\}$

Continuous Greedy

- Multilinear relaxation transforms the problem to

$$\max\{F(y) : y \in P(\mathcal{M})\}$$

- Main drawback - $\widetilde{O}(n^8)$ running time, with $\widetilde{O}(n^7)$ oracle calls
- Main advantage - $(1 - \frac{1}{e})$ approximation and straightforward
- Two-step algorithm:
  1. Use the continuous greedy process to approximate $F(y)$ within $(1 - \frac{1}{e})$ factor
  2. Use pipage rounding techniques to convert the fractional solution to a discrete one such that

$$f(S) \geq F(y) \geq \left(1 - \frac{1}{e}\right) OPT$$

- Overtime flow, given as

$$\frac{dy}{dt} = v_{\max}(y) = \arg\max_{v \in P}(v \cdot \nabla F(y))$$

Accelerated Continuous Greedy

- interpolates the fast Classical Greedy with the slow Continuous Greedy with a δ parameter
- $\delta = 1$ corresponds to the former and $\delta \in (0, 1)$ corresponds to discretized version of latter
- Acceleration is provided is because of updating partial derivatives after each increment, which gives a cleaner analysis and mimics the discrete greedy algorithm.
- Along with that, due to the discrete greedy nature, we take larger steps and thus need fewer samples per iteration, and fewer iterations
- Moreover, any $\delta > 0$ gives a $\left(1 - \frac{1}{e} - \mathcal{O}(\delta)\right)$ approximation

Continuous Greedy - General

Algorithm **ContinuousGreedy**$(f, \mathcal{M})$:

1. Let $\delta = \frac{1}{9d^2}$, where $d = r_{\mathcal{M}}(X)$ (the rank of the matroid). Let $n = |X|$. Start with $t = 0$ and $y(0) = \mathbf{0}$.
2. Let $R(t)$ contain each j independently with probability $y_j(t)$. For each $j \in X$, let $\omega_j(t)$ be an estimate of $\mathbf{E}[f_{R(t)}(j)]$, obtained by taking the average of $\frac{10}{\delta^2}(1 + \ln n)$ independent samples of $R(t)$.
3. Let $I(t)$ be a maximum-weight independent set in $\mathcal{M}$, according to the weights $\omega_j(t)$. We can find this by the greedy algorithm. Let

$$y(t+\delta) = y(t) + \delta \cdot \mathbf{1}_{I(t)}.$$

4. Increment $t := t + \delta$; if $t < 1$, go back to Step 2. Otherwise, return $y(1)$.

Continuous Greedy - SWP

The Continuous Greedy Algorithm for Submodular Welfare.

1. Let $\delta = \frac{1}{9m^2}$ where m is the number of items. Start with $t = 0$ and $y_{ij}(0) = 0$ for all i, j.
2. Let $R_i(t)$ be a random set containing each item j independently with probability $y_{ij}(t)$. For all i, j, estimate the expected marginal profit of player i from item j,

$$\omega_{ij}(t) = \mathbf{E}[w_i(R_i(t) + j) - w_i(R_i(t))]$$

   by taking the average of $\frac{10}{\delta^2}(1 + \ln(mn))$ independent samples.
3. For each j, let $i_j(t) = \text{argmax}_i\, \omega_{ij}(t)$ be the *preferred player* for item j (breaking possible ties arbitrarily). Set $y_{ij}(t+\delta) = y_{ij}(t)+\delta$ for the preferred player $i = i_j(t)$ and $y_{ij}(t+\delta) = y_{ij}(t)$ otherwise.
4. Increment $t := t + \delta$; if $t < 1$, go back to Step 2.
5. Allocate each item j independently, with probability $y_{ij}(1)$ to player i.

Continuous Greedy - SAP/GAP

The Continuous Greedy Algorithm for SAP/GAP.

1. Let $\delta = \frac{1}{9n^2}$. Start with $t = 0$ and $y_{j,S}(0) = 0$ for all j, S.
2. Let $\mathcal{R}(t)$ be a random collection of pairs (j, S), each pair (j, S) appearing independently with probability $y_{j,S}(t)$. For all i, j, estimate the expected marginal profit of bin j from item i,
$$\omega_{ij}(t) = \mathbf{E}_{\mathcal{R}(t)}[\max\{0, v_{ij} - \max\{v_{ij'} : \exists(j', S) \in \mathcal{R}(t); i \in S\}\}]$$
by taking the average of $(mn)^5$ independent samples.
3. For each j, find an α-approximate solution $S_j^*(t)$ to the problem $\max\{\sum_{i\in S}\omega_{ij}(t) : S \in \mathcal{F}_j\}$. Set $y_{j,S}(t+\delta) = y_{j,S}(t) + \delta$ for the set $S = S_j^*(t)$ and $y_{j,S}(t+\delta) = y_{j,S}(t)$ otherwise.
4. Increment $t := t + \delta$; if $t < 1$, go back to Step 2.
5. For each bin j independently, choose a random set $S_j := S$ with probability $y_{j,S}(1)$. For each item occurring in multiple sets S_j, keep only the occurrence of maximum value. Allocate the items to bins accordingly.

Accelerated Continuous Greedy - Main

Algorithm 4: Accelerated Continuous Greedy

Input : $f : 2^E \to \mathbb{R}_+, \mathcal{I} \subseteq 2^E$
Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$

$\mathbf{x} \leftarrow 0$;
for $t \leftarrow \epsilon; t \leq 1; t \leftarrow t + \epsilon$ **do**
  $B \leftarrow$Decreasing-Threshold($f, \mathbf{x}, \epsilon, \mathcal{I}$);
  $\mathbf{x} \leftarrow \mathbf{x} + \epsilon \cdot \mathbf{1}_B$;
end
$S \leftarrow$Pipage-Rounding($\mathbf{x}, \mathcal{I}$);
return S

Accelerated Continuous Greedy - Decreasing Threshold

Algorithm 3: Decreasing Threshold

function Decreasing-Threshold $(f, \mathbf{x}, \epsilon, \mathcal{I})$;

Input : $f : 2^E \to \mathbb{R}_+, \mathbf{x} \in [0,1]^E, \epsilon \in [0,1], \mathcal{I} \subseteq 2^E$

Output: A set $S \subseteq E$ satisfying $S \in \mathcal{I}$

$B \leftarrow \emptyset$;

$d \leftarrow \max_{j \in E} f(j)$;

for $w = d; w \geq \frac{\epsilon}{r} d; w \leftarrow w(1-\epsilon)$ **do**

    for $e \in E$ **do**

        $w_e(B, \mathbf{x}) \leftarrow$estimate$(\mathbb{E}[f_{R(\mathbf{x}+\epsilon \cdot \mathbf{1}_B)}(e)])$ using $\frac{r \log n}{\epsilon^2}$ samples;

        if $B \cup \{e\} \in \mathcal{I}$ *and* $w_e(B, \mathbf{x}) \geq w$ **then**

            $B \leftarrow B \cup \{e\}$

        end

    end

end

return B

Accelerated Continuous Greedy - Pipage-Rounding (1)

Algorithm 2: Efficient Pipage-Rounding

```
function Pipage-Rounding (f, x, I);
Input  : f : 2^E → R_+, x ∈ [0,1]^E, I ⊆ 2^E
Output: A set S ⊆ E satisfying S ∈ I
T ← [];
for i ← 0; i < length(x); i ← i + 1 do
    if 0 < x[i] < 1 then
        T.push(i);
    end
end
try:
    while length(T) > 0 do
        i, j ← T[0], T[1];
        if x[i] + x[j] < 1 then
            p ← x[j] / (x[i] + x[j]);
            if rand() < p then
                x[j] ← x[i] + x[j];
                x[i] ← 0;
                delete(T, 0);
            end
            else
                x[i] ← x[i] + x[j];
                x[j] ← 0;
                delete(T, 1);
            end
        end
        else
            p ← (1 − x[i]) / (2 − x[i] − x[j]);
            if rand() < p then
                x[i] ← x[i] + x[j] − 1;
                x[j] ← 1;
                delete(T, 1);
            end
            else
                x[j] ← x[i] + x[j] − 1;
                x[i] ← 1;
                delete(T, 0);
            end
        end
    end
catch:
    S ← E[x];
    return S
end
S ← E[x];
return S
```

Accelerated Continuous Greedy - Pipage-Rounding (2)

else

$p \leftarrow \frac{1-\mathbf{x}[i]}{2-\mathbf{x}[i]-\mathbf{x}[j]}$;

if *rand()* $< p$ **then**

$\mathbf{x}[i] \leftarrow \mathbf{x}[i] + \mathbf{x}[j] - 1$;

$\mathbf{x}[j] \leftarrow 1$;

$\mathrm{delete}(T, 1)$;

end

else

$\mathbf{x}[j] \leftarrow \mathbf{x}[i] + \mathbf{x}[j] - 1$;

$\mathbf{x}[i] \leftarrow 1$;

$\mathrm{delete}(T, 0)$;

end