Submodular Optimization

Nathaniel Grammel

Submodularity

- Captures the notion of Diminishing Returns

Definition

Suppose U is a set. A set function $f : 2^U \to \mathbb{R}$ is submodular if for any $S, T \subseteq U$:

$$f(S \cap T) + f(S \cup T) \leq f(S) + f(T)$$

Submodularity

- Captures the notion of Diminishing Returns

Definition

Suppose U is a set. A set function $f : 2^U \to \mathbb{R}$ is submodular if for any $S, T \subseteq U$:

$$f(S \cap T) + f(S \cup T) \leq f(S) + f(T)$$

- How does this represent diminishing returns?

Submodularity

- Captures the notion of Diminishing Returns

Definition

Suppose U is a set. A set function $f\colon 2^U \to \mathbb{R}$ is submodular if for any $S, T \subseteq U$:

$$f(S \cap T) + f(S \cup T) \leq f(S) + f(T)$$

- How does this represent diminishing returns?
- Equivalent Definition:

Definition

A function $f\colon 2^U \to \mathbb{R}$ is submodular if for any $S, T \subseteq U$ such that $S \subseteq T$, and any $x \in U \setminus T$:

$$f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$$

Submodularity and Diminishing Returns

- Imagine choosing items from U one by one
- At any point, let S be the set of items chosen so far
- Choosing x as the next item gives an *increase* in utility of $f(S \cup \{x\}) - f(S)$

Submodularity and Diminishing Returns

- Imagine choosing items from U one by one
- At any point, let S be the set of items chosen so far
- Choosing x as the next item gives an *increase* in utility of $f(S \cup \{x\}) - f(S)$
- If we choose some other items first to get a set T (notice: $S \subseteq T$), and *then* choose x, the increase in utility is $f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$

Submodularity and Diminishing Returns

- Imagine choosing items from U one by one
- At any point, let S be the set of items chosen so far
- Choosing x as the next item gives an *increase* in utility of $f(S \cup \{x\}) - f(S)$
- If we choose some other items first to get a set T (notice: $S \subseteq T$), and *then* choose x, the increase in utility is $f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$
- Adding x give *less* utility if we start with *more*

Submodularity and Diminishing Returns

- Imagine choosing items from U one by one
- At any point, let S be the set of items chosen so far
- Choosing x as the next item gives an *increase* in utility of $f(S \cup \{x\}) - f(S)$
- If we choose some other items first to get a set T (notice: $S \subseteq T$), and *then* choose x, the increase in utility is $f(T \cup \{x\}) - f(T) \leq f(S \cup \{x\}) - f(S)$
- Adding x give *less* utility if we start with *more*
    - This is the concept of diminishing returns

Monotonicity: Another Useful Property

Definition

A set function $f : 2^U \to \mathbb{R}$ is *monotone* if for every $S, T \subseteq U$ with $S \subseteq T$:

$$f(S) \leq f(T)$$

Monotonicity: Another Useful Property

Definition

A set function $f : 2^U \to \mathbb{R}$ is *monotone* if for every $S, T \subseteq U$ with $S \subseteq T$:

$$f(S) \leq f(T)$$

We are generally interested in *monotone submodular* functions. Often, these are *utility functions*.

A Concrete Example

- Recall this example utility function from early in the semester:

$$\begin{aligned} f(\{apple, orange\}) &= 5 \\ f(\{apple\}) = f(\{orange\}) &= 3 \\ f(\{\}) = f(\varnothing) &= 0 \end{aligned}$$

- Notice it is monotone submodular!

Why do we care?

- Diminishing Returns

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)
  - Determining representative sentences in a document

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)
  - Determining representative sentences in a document
  - Many applications to image and signal processing.

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)
  - Determining representative sentences in a document
  - Many applications to image and signal processing.
  - Sensor Placement

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)
  - Determining representative sentences in a document
  - Many applications to image and signal processing.
  - Sensor Placement
  - Graph Cuts

Why do we care?

- Diminishing Returns
- Closely Related to Convexity and Concavity
- Optimization problems are *very hard* in general (often, NP-hard), especially with set functions.
- But with submodularity, we can make strong guarantees: efficient algorithms for close approximations.
- Many natural functions have these properties
  - Inferring Influence in a Network (Stay tuned!)
  - Determining representative sentences in a document
  - Many applications to image and signal processing.
  - Sensor Placement
  - Graph Cuts
  - And many more! (Check out submodularity.org)

Submodular Optimization

- Two main types of submodular optimization:

Submodular Optimization

- Two main types of submodular optimization:
- Maximization

Submodular Optimization

- Two main types of submodular optimization:
- Maximization
  - Want to find S to maximize $f(S)$ subject to some constraints
  - Most simply: Want to find S that maximizes $f(S)$ subject to $|S| = k$ for some k (cardinality constraint)
  - More generally: Other types of constraints (e.g. knapsack constraints, matroid constraints)

Submodular Optimization

- Two main types of submodular optimization:
- Maximization
  - Want to find S to maximize $f(S)$ subject to some constraints
  - Most simply: Want to find S that maximizes $f(S)$ subject to $|S| = k$ for some k (cardinality constraint)
  - More generally: Other types of constraints (e.g. knapsack constraints, matroid constraints)
- Cover/Minimization

Submodular Optimization

- Two main types of submodular optimization:
- Maximization
  - Want to find S to maximize $f(S)$ subject to some constraints
  - Most simply: Want to find S that maximizes $f(S)$ subject to $|S| = k$ for some k (cardinality constraint)
  - More generally: Other types of constraints (e.g. knapsack constraints, matroid constraints)
- Cover/Minimization
  - Want to minimize the *cost* of *covering* f
  - Most simply: Find S with minimum size that achieves $f(S) = f(U)$
  - More generally: Find S such that $f(S) = f(U)$ while minimizing $cost(S)$ for some cost function.

Submodular Maximization: The Simplest Case

- Suppose we are given a utility function $f : 2^U \to \mathbb{R}$ and a cardinality constraing k
- Goal: Find

$$S^* = \underset{S \subseteq U : |S| \leq k}{\arg\max} \; f(S)$$

- This is NP-hard in general!
- What if f is submodular? Monotone? Nonnegative?
    - We can find good near-optimal solutions!

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f : 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f: 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.
- Start with $S = \{\}$

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f : 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.
- Start with $S = \{\}$
- At each step, pick the item $x \in U \setminus S$ that *maximizes* $f(S \cup \{x\}) - f(S)$ (the item that maximizes the *gain* in utility) and let $S = S \cup \{x\}$.

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f: 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.
- Start with $S = \{\}$
- At each step, pick the item $x \in U \setminus S$ that *maximizes* $f(S \cup \{x\}) - f(S)$ (the item that maximizes the *gain* in utility) and let $S = S \cup \{x\}$.
- Continue until $|S| = k$

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f: 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.
- Start with $S = \{\}$
- At each step, pick the item $x \in U \setminus S$ that *maximizes* $f(S \cup \{x\}) - f(S)$ (the item that maximizes the *gain* in utility) and let $S = S \cup \{x\}$.
- Continue until $|S| = k$

Theorem (Nemhauser et al. 1978)

Let S be the k-element set constructed as above, and let S^* be the set that maximizes f over all sets of size at most k. Then:

$$f(S) \geq (1 - 1/e) f(S^*)$$

Submodular Maximization: A Simple Greedy Algorithm

- Suppose $f : 2^U \to \mathbb{R}$ is monotone submodular. Assume f is nonnegative.
- Start with $S = \{\}$
- At each step, pick the item $x \in U \setminus S$ that *maximizes* $f(S \cup \{x\}) - f(S)$ (the item that maximizes the *gain* in utility) and let $S = S \cup \{x\}$.
- Continue until $|S| = k$

Theorem (Nemhauser et al. 1978)

Let S be the k-element set constructed as above, and let S^* be the set that maximizes f over all sets of size at most k. Then:

$$f(S) \geq (1 - 1/e) f(S^*)$$

- Thus, the set S provides a $(1 - 1/e)$-approximation, and the construction gives a polynomial-time approximation algorithm.

Greedy Algorithm: Proof

For convenience, let $\Delta_S(x) = f(S \cup \{x\}) - f(S)$. Then submodularity states that for $S \subseteq T$ and $x \in U \setminus T$, we have $\Delta_T(x) \leq \Delta_S(x)$.
Let $S_i \subseteq U$ be the subset of i elements chosen greedily:

$$S_i = S_{i-1} \cup \{\arg\max_{x \in U} \Delta_{S_{i-1}}(x)\}$$

with $S_0 = \{\}$.

Proof that $f(S) \geq (1 - 1/e) f(S^*)$.

Due to monotonicity, $|S^*| = k$. Let $S^* = \{e_1, e_2, \ldots, e_k\}$. Further, also due to monotonicity, for any $i < k$:

$$f(S^*) \leq f(S^* \cup S_i) \tag{1}$$

□

Greedy Algorithm: Proof

Proof that $f(S) \geq (1 - 1/e) f(S^*)$.

We also have the following equality

$$f(S^* \cup S_i) = f(S_i) + \sum_{j=1}^{k} \Delta_{S_i \cup \{e_1, \ldots, e_{j-1}\}}(e_j) \qquad (1)$$

since the terms
$\Delta_{S_i \cup \{e_1, \ldots, e_{j-1}\}}(e_j) = f(S_i \cup \{e_1, \ldots, e_j\}) - f(S_i \cup \{e_1, \ldots, e_{j-1}\})$
are telescoping so the sum is equal to
$f(S_i \cup \{e_1, \ldots, e_j\}) - f(S_i \cup \{\}) = f(S_i \cup S^*) - f(S_i)$. $\square$

Greedy Algorithm: Proof

Proof that $f(S) \geq (1 - 1/e) f(S^*)$.

Due to submodularity, we have

$$f(S_i) + \sum_{j=1}^{k} \Delta_{S_i \cup \{e_1, \ldots, e_{j-1}\}}(e_j) \leq f(S_i) + \sum_{j=1}^{k} \Delta_{S_i}(e_j) \qquad (1)$$

The greedy rule states that $f(S_{i+1}) - f(S_i) \geq \Delta_{S_i}(x)$ for any x. Thus:

$$\begin{aligned} f(S_i) + \sum_{j=1}^{k} \Delta_{S_i}(e_j) &\leq f(S_i) + \sum_{j=1}^{k} (f(S_{i+1}) - f(S_i)) \\ &\leq f(S_i) + k(f(S_{i+1}) - f(S_I)) \qquad (2) \end{aligned}$$

where the second inequality holds since $|S^*| = k$. □

Greedy Algorithm: Proof

Proof that $f(S) \geq (1 - 1/e)f(S^*)$.

Putting it all together:

$$f(S^*) - f(S_i) \leq k(f(S_{i+1}) - f(S_i)) \tag{1}$$

Let $\delta_i = f(S^*) - f(S_i)$. Then, we can rearrange to get $\delta_i \leq k(\delta_i - \delta_{i+1})$, or $\delta_{i+1} \leq \delta_i \left(1 - \frac{1}{k}\right)$ which yields

$$\delta_k \leq \delta_0 \left(1 - \frac{1}{k}\right)^k$$

□

Greedy Algorithm: Proof

Proof that $f(S) \geq (1 - 1/e) f(S^*)$.

Since f is nonnegative, $\delta_0 = f(S^*) - f(\{\}) \leq f(S^*)$. A famous inequality states: $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$. This yields

$$\delta_k \leq \left(1 - \frac{1}{k}\right)^k \delta_0 \leq \left(1 - \frac{1}{k}\right)^k f(S^*) \leq e^{-k/k} f(S^*)$$

Since $\delta_k = f(S^*) - f(S_k)$, we get

$$\delta_k = f(S^*) - f(S_k) \leq e^{-1} f(S^*) \tag{1}$$
$$f(S^*) - e^{-1} f(S^*) \leq f(S_k) \tag{2}$$
$$f(S^*)(1 - 1/e) \leq f(S_k) \tag{3}$$

$\square$

$(1 - 1/e)$ is the best we can get

Theorem (Nemhauser and Wolsey 1978)

Any algorithm that evaluate f on at most a polynomial number of inputs cannot do better than a $(1 - 1/e)$-approximation of the optimal solution.

Speedup with Lazy Evaluations (Minoux 1978)

- Evaluating $\Delta_S(x)$ for every x at each iteration may be costly
- Store for each item x a value $\phi(x)$, representing an upper bound on $\Delta_S(x)$. Store the items sorted in order of ϕ.
- At each step, pick the element x at the front of the list (i.e. with maximum $\phi(x)$)
- Lazy Evaluation: Evaluate Δ_S only for element x, and update $\phi(x) \leftarrow \Delta_S(x)$.
- If after update, $\phi(x) \geq \phi(x')$ for all other x', then x is still the best choice for the greedy algorithm! We've avoided re-evaluating Δ for all the other elements!

Submodular Cover

- Suppose instead we want to find S^* so that $f(S^*) = f(U)$ while *minimizing* $|S^*|$.
- Again, suppose f is monotone and submodular. But: this time let $f\colon 2^U \to \mathbb{N}$.
- Suppose we apply the same greedy rule until our constructed set S has $f(S) = f(U)$. Do we have bounds on $|S|$?
- Yes!

Theorem (Wolsey 1982)

$$|S| \leq (1 + \ln \rho)|S^*|$$

where $\rho = \max_{x \in U} f(\{x\})$ *is the maximum possible increase in utility.*

What about non-uniform costs?

- Up until now we have focused on cardinality: either constrained to $|S| \leq k$ (maximization), or trying to minimize $|S|$ (cover)

What about non-uniform costs?

- Up until now we have focused on cardinality: either constrained to $|S| \leq k$ (maximization), or trying to minimize $|S|$ (cover)
- What if we instead have a cost function $c(x)$ for all $x \in U$ and want to maximize $f(S)$ subject to

$$\sum_{x \in S} c(x) \leq B$$

for some *budget* B. This is called a Knapsack Constraint.

What about non-uniform costs?

- Up until now we have focused on cardinality: either constrained to $|S| \leq k$ (maximization), or trying to minimize $|S|$ (cover)
- What if we instead have a cost function $c(x)$ for all $x \in U$ and want to maximize $f(S)$ subject to

$$\sum_{x \in S} c(x) \leq B$$

for some *budget* B. This is called a Knapsack Constraint.
- Or minimize $\sum_{x \in S} c(x)$ such that S *covers* f (i.e. $f(S) = f(U)$)?

Results for non-uniform costs

- Standard Greedy Algorithm could be arbitrarily bad: Doesn't consider costs at all!

Results for non-uniform costs

- Standard Greedy Algorithm could be arbitrarily bad: Doesn't consider costs at all!
- Modification: At each step, choose x that maximizes $\frac{\Delta_S(x)}{c(x)}$ — best "bang for the buck"

Results for non-uniform costs

- Standard Greedy Algorithm could be arbitrarily bad: Doesn't consider costs at all!
- Modification: At each step, choose x that maximizes $\frac{\Delta_S(x)}{c(x)}$ — best "bang for the buck"
- Can get similar results to uniform-cost case, with some slight modifications

Results for non-uniform costs

- Standard Greedy Algorithm could be arbitrarily bad: Doesn't consider costs at all!
- Modification: At each step, choose x that maximizes $\frac{\Delta_S(x)}{c(x)}$ — best "bang for the buck"
- Can get similar results to uniform-cost case, with some slight modifications
- Cover: Wolsey (1982) generalizes the result of uniform-cost case

Results for non-uniform costs

- Standard Greedy Algorithm could be arbitrarily bad: Doesn't consider costs at all!
- Modification: At each step, choose x that maximizes $\frac{\Delta_S(x)}{c(x)}$ — best "bang for the buck"
- Can get similar results to uniform-cost case, with some slight modifications
- Cover: Wolsey (1982) generalizes the result of uniform-cost case
- For maximization: A bit trickier. This greedy rule doesn't suffice! But some simple modifications can yield similar approximations to uniform-cost case.