Unconstrained Submodular Maximization & Applications

In the previous lecture, we introduced submodular functions and looked at algorithms to maximize non-negative, monotone submodular functions where inputs were subject to some constraints. In this lecture, we look at unconstrained submodular maximization in which the function is not necessarily monotone. Afterwards, we look at two applications of submodular functions, and conclude by considering scenarios in which we do not know the submodular function and wish to learn it.

1 Unconstrained Submodular Maximization

Let \( f : \{0,1\}^n \to \mathbb{R}_+ \) be a nonnegative submodular function, where inputs \( S \) come from some ground set \( \mathcal{N} \). Then, the objective (with no constraints) is the following:

\[
\max_{S \subseteq \mathcal{N}} f(S)
\]

In the constrained version of the problem, we presented a greedy algorithm, where at each step we added the item that maximized the marginal utility. Since \( f \) is no longer monotone, we need to modify this approach. Instead of only adding items, we should also be able to decide that we do not want a certain item. This motivates Algorithm 1, the double greedy algorithm.

**Algorithm 1: Double Greedy**

**Result:** Write here the result

Initialize \( X^{(0)} = \emptyset, Y^{(0)} \to \mathcal{N} \);

for \( i = 1, \ldots, n \) do

\[
\begin{align*}
\alpha_i &\leftarrow f(X^{(i-1)} + u_i) - f(X^{(i-1)}); \\
\beta_i &\leftarrow f(Y^{(i-1)} - u_i) - f(Y^{(i-1)});
\end{align*}
\]

if \( \alpha_i \geq \beta_i \) then

\[
\begin{align*}
X^{(i)} &\leftarrow X^{(i-1)} + u_i; \\
Y^{(i)} &\leftarrow Y^{(i-1)};
\end{align*}
\]

else

\[
\begin{align*}
X^{(i)} &\leftarrow X^{(i-1)}; \\
Y^{(i)} &\leftarrow Y^{(i-1)} - u_i;
\end{align*}
\]

end

end

return \( X^{(n)} = Y^{(n)} \)

We will keep track of two collections, \( X \) and \( Y \), with \( X \) being initialized to the empty set and \( Y \) being initialized to \( \mathcal{N} \) (contains all items). At each step \( i \), we consider the next item, and decide
to either add it to set $X^{(i-1)}$ or remove it from set $Y^{(i-1)}$. Therefore, at the beginning of each step $i$, $X^{(i-1)}$ and $Y^{(i-1)}$ agree on the first $i - 1$ elements (think of the two sets as a bit vector), and disagree on all elements $j > i - 1$. After all items have been either added to $X$ or removed from $Y$, the two sets will be the same. We now analyze the performance of Algorithm 2.

**Theorem 1** Double greedy achieves an approximation ratio of $\frac{1}{3}$.

**Proof:** We first establish some notation. Let $\text{OPT} \subseteq \mathcal{N}$ be the vector representing the set that maximizes $f$, and let $\text{OPT}_S \subseteq \mathcal{N}$ be $\text{OPT}$ restricted to the elements of $S$ (with zeros for all elements not in $S$). Define $O^{(i)} = (\text{OPT}_{\{i+1, \ldots, n\}} \cup X^{(i)}) \cap Y^{(i)}$, i.e. the set that agrees with $X^{(i)}$ on the first $i$ elements, and agrees with $\text{OPT}$ on elements $i + 1$ through $n$.

To prove the theorem, we first prove the following two lemmas. Lemma 1 will be used to prove Lemma 2. Lemma 2 gives an upper bound on how much value we lose (compared to $\text{OPT}$) at each step $i$ by our choice to either include or not include $u_i$, and will do the heavy lifting for the proof of the theorem.

**Lemma 1** $\forall i \in \{1, \ldots, n\}, a_i + b_i \geq 0$

**Proof of Lemma 1:** By definition,

$$a_i + b_i = f(X^{(i-1)} + u_i) - f(X^{(i-1)}) + f(Y^{(i-1)} - u_i) - f(Y^{(i-1)})$$

Observe the following:

$$X^{(i-1)} = (X^{(i-1)} + u_i) \cap (Y^{(i-1)} - u_i)$$

$$Y^{(i-1)} = (X^{(i-1)} + u_i) \cup (Y^{(i-1)} - u_i)$$

Substituting this back in,

$$a_i + b_i = f(X^{(i-1)} + u_i) + f(Y^{(i-1)} - u_i)$$

$$- \left[f((X^{(i-1)} + u_i) \cap (Y^{(i-1)} - u_i)) + f((X^{(i-1)} + u_i) \cup (Y^{(i-1)} - u_i))\right]$$

By definition of submodular functions, $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$, so we can conclude that $a_i + b_i \geq 0$. We now use Lemma 1 to prove the following lemma:

**Lemma 2** $f(O^{(i-1)}) - f(O^{(i)}) \leq \left[f(X^{(i-1)}) - f(X^{(i-1)})\right] + \left[f(Y^{(i)}) - f(Y^{(i-1)})\right]$

**Proof of Lemma 2:** Assume that $a_i \geq b_i$ (the proof for the opposite case is analogous). Then, based on the updates made by the algorithm, the lemma statement simplifies to

$$f(O^{(i-1)}) - f(O^{(i)}) \leq \left[f(X^{(i-1)} + u_i) - f(X^{(i-1)})\right] + \left[f(Y^{(i-1)} - f(Y^{(i-1)})\right] = a$$

There are two possible cases: either $u_i \in \text{OPT}$ or $u_i \notin \text{OPT}$. If $u_i \in \text{OPT}$, then $u_i \in O^{(i-1)}$, so the lemma simplifies to the inequality $a_i \geq 0$. Since by assumption, $a_i \geq b_i$, the inequality follows from Lemma 1. If instead $u_i \notin \text{OPT}$, then $u_i \notin O^{(i-1)}$, i.e.

$$f(O^{(i-1)}) - f(O^{(i)}) = f(O^{(i-1)}) - f(O^{(i-1)} + u_i)$$
Next, observe the following:

\[ O^{(i-1)} = \left( \text{OPT}_{\{i+1, \ldots, n\}} \cup X^{(i)} \right) \cap Y^{(i)} \subseteq Y^{(i-1)} - u_i \]

Therefore,

\[ f(O^{(i-1)}) = f(O^{(i)}) \cap f(Y^{(i-1)} - u_i) \]
\[ f(O^{(i-1)} + u_i) = f(O^{(i)}) \cup f(Y^{(i-1)} - u_i) \]

By submodularity,

\[
\begin{align*}
& f(O^{(i-1)}) - f(O^{(i)}) = f(O^{(i-1)}) - f(O^{(i-1)} + u_i) \\
& \quad \leq f(Y^{(i-1)} - u_i) - f(Y^{(i)}) \\
& \quad = b_i \\
& \quad \leq a_i \quad \text{(by assumption)}
\end{align*}
\]

This concludes the proof of Lemma 2. We now return to the proof of Theorem 1. Recall that the output of the algorithm is \( X^{(n)} = Y^{(n)} \). We can rewrite \( f(X^{(n)} = Y^{(n)}) \) as the sum of the value we lost in each round compared to OPT, and by Lemma 2

\[
\sum_{i=1}^{n} \left[ f(O^{(i-1)}) - f(O^{(i)}) \right] \leq \sum_{i=1}^{n} \left[ f(X^{(i)}) - f(X^{(i-1)}) \right] + \sum_{i=1}^{n} \left[ f(Y^{(i)}) - f(Y^{(i-1)}) \right]
\]

The summation on the left is a telescoping sum, so we can simplify it to the following;

\[
f(O^{(0)}) - f(O^{(n)}) \leq f(X^{(n)}) - f(X^{(0)}) + f(Y^{(n)}) - f(Y^{(0)})
\]

Observe that \( O^{(0)} \) is OPT and \( O^{(n)} \) is \( X^{(n)} \). Then,

\[ f(O^{(0)}) \leq 3f(X^{(n)}) \quad \text{ (since } f \text{ is nonnegative)}
\]

\[
\square
\]

2 Application - Document Summarization

One useful application is the task of summarizing a document (e.g. summarizing a news article). Given a set of sentences \( N \), we wish to find the find the following:

\[
\operatorname{arg} \max_{S \subseteq N} f(S) \quad \text{ subject to } \sum_{i \in S} c_i \leq b
\]

where \( c_i \) is the cost of including sentence \( i \), and \( b \) is some predetermined budget. It is natural to assume that \( f \) is a submodular function; the more sentences you add to a summary, the less valuable each additional sentence is.
We might also want to add additional metrics to our objective. In particular, we might care about the **coverage** of a summary (whether the summary captures the main idea) as well as the **diversity** of a summary (avoiding sentence redundancy). To that end, we can define our function \( f(S) \) to be the sum of two separate submodular functions \( R(S) + D(S) \), where \( R(S) \) measures the coverage and \( D(S) \) measures the diversity. For example, we can define the coverage of a set in terms of its total similarity with the entire document.

\[
R(S) = \sum_{i \in \mathcal{N}} \sum_{j \in S} s_{ij}
\]

where \( s_{ij} \) is a measure of similarity between sentence \( i \) and sentence \( j \). For diversity, the following example function measures the amount of content captured across different sections of the document.

\[
D(S) = \sum_{l=1}^{m} \sqrt{|S \cap P_l|}
\]

where \( P_1, \ldots, P_m \) is a partition of \( \mathcal{N} \), i.e. their union is \( \mathcal{N} \) and they are pairwise disjoint. Both \( R(S) \) and \( D(S) \) as defined are submodular functions, and we can use variants of the greedy algorithm we have discussed in the past few lectures to solve this problem.

### 3 Application - Determinantal Point Processes

A **point process** \( P \) on \( \mathcal{N} \) is a probability distribution on \( 2^\mathcal{N} \). The probability distribution in a **determinantal point process**, or DPP, is the determinant of some function. DPPs were first studied in statistical physics, and are effective at modeling probability distributions over a collection of items where diverse subsets are preferred. For example, we might care about diversity in product recommendations, pose estimation, placement of sensor locations, and as mentioned in the previous application, document summarization.

A restricted class of DPPs known as **L-ensembles** are defined by a real, symmetric similarity matrix \( L \in \mathbb{R}^{\mathcal{N} \times \mathcal{N}} \) where the following holds:

\[
\Pr_{Y \sim L}(Y) \propto \det(L_Y) \quad (\forall Y \subseteq \mathcal{N})
\]

Since \( L \) is a symmetric matrix, we can decompose it into the product of a matrix times its transpose, i.e. \( L = B^T B \). Therefore,

\[
\det(L_Y) = \det(B_Y^T B) \\
= \det^2(B_Y) \\
= \text{volume}^2\{B_i\}_{i \in Y} \\
\Pr_{Y \sim L}(Y) \propto \text{volume}^2\{B_i\}_{i \in Y}
\]

This implies that two vectors \( B_i \) and \( B_j \) have a higher probability of being included if they are less similar, i.e. the angle between them is larger, because the volume of the corresponding parallelepiped will be larger.
Note that $\Pr_{Y \sim L}(Y)$ is log-submodular. We can see geometrically that for $X \subseteq Y \subseteq \mathcal{N} - \{i\}$

$$\frac{\Pr_{X \sim L}(X \cup \{i\})}{\Pr_{X \sim L}(X)} \geq \frac{\Pr_{Y \sim L}(Y \cup \{i\})}{\Pr_{Y \sim L}(Y)} \quad \log \Pr_{X \sim L}(X \cup \{i\}) - \log \Pr_{X \sim L}(X) \geq \log \Pr_{Y \sim L}(Y \cup \{i\}) - \log \Pr_{Y \sim L}(Y)$$

4 Learning Submodular Functions

Under some scenarios, we might not know the function $f$, and want to learn the function from a small number of samples. In particular, if we can sample the value of $f$ at $m$ different sets, we would like to find an approximate function $\hat{f}$ that achieves an $\alpha$-approximation of $f$ over all possible sets. Examples of such settings include graph inference, in which we want to learn a graph from the value of a few of its cuts, and combinatorial auctions, in which we want to learn a buyer’s utility function from only a few bids.

Unfortunately, [1] shows that even under the assumption that $f$ is monotone, we need to sample exponentially many values to achieve better than an $\alpha = \sqrt{n}/\log n$-approximation.

References


