6.883 Learning with Combinatorial Structure
Note for Lecture 15
Authors: David Alvarez-Melis

1 From minimization to maximization

So far we have talked about submodular functions and how to minimize them. We looked at how to create a convex function out of a set function (e.g. the Lovász extension), and then minimize that relaxation using techniques from convex optimization. The fact that we could solve those problems in polynomial time was a direct consequence of being able to get such a convex extension of the original submodular function.

We looked at important concepts related to submodular minimization, such as the submodular base polytope and the Lovász extension. In terms of algorithms, we have talked about the min-norm point algorithm and, for a subclass of problems, graph cuts. We mentioned various applications of these methods: min-cut, sparse estimation, clustering. All these problems have the same flavor: we minimize a cost function that measures some type of *incoherence* in the sets. For example, for the min-cut problem we try to find something that looks like a clique. For group sparsity, we want a set of coordinates that somehow work well together.

But we could turn this around, and instead of having a cost function we could have a *utility* function. This function could be measuring spread or diversity. Today we will look at submodular functions from a different perspective: maximizing a utility function.

There are various applications for this framework. Our initial example of choosing sensor locations can be cast under this formulation; we want to place sensors in such a way that we maximize the information we get from them. There are many other problems with the same flavor: environmental monitoring, active learning. Another common problem that falls in this framework is summarization. In this case we want to get a subset of the data that is "representative" the full document. We will now analyze this maximization framework through some of these examples.

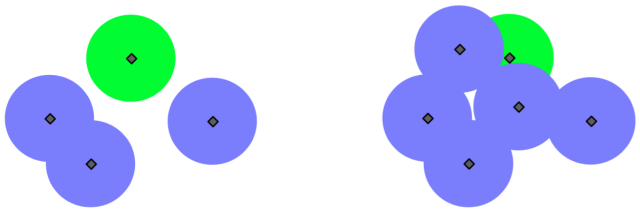
2 Submodular maximization

2.1 Sensing

Recall that in the sensing problem we want to monitor the temperature throughout some space, so we need to pick locations to place sensors. The simplest way to model this problem is to require that the placement maximizes *coverage*. For example, if we let S denote the set of locations for the sensors, we can try to maximize:

$$F(S) = \left| \bigcup_{v \in S} area(v) \right| \tag{1}$$

This function expresses the gain as the joint area covered by the sensors. It is clearly submodular: there larger the area covered by a set of locations, the less gain there is from adding an additional sensor.



$$F(A \cup v) - F(A) \quad \geq \quad F(B \cup v) - F(B)$$

This simplistic approach usually doesn't model the problem very well, so we can try to phrase it in a slightly more sophisticated way by using a graphical model. This model should seek spatial coherence: if two locations are close to each other, their temperatures are very correlated. Let Y_s be the (unknown) temperature at location s, and X_s be the sensor value at that location. The measured temperature will have some noise, so $X_s = Y_s + \text{noise}$.

The question is now where to place the sensors so as to learn as much as possible about the joint distribution $P(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$. We can model this via the mutual information between the X's and the unknown Y's. For this, we define:

$$F(A) = H(\mathbf{Y}) - H(\mathbf{Y} \mid \mathbf{X_A}) = I(\mathbf{Y}; \mathbf{X_A}) \tag{2}$$

In one of the problem sets, we showed that under certain conditions the mutual information yields a submodular function. Specifically, when all the X_i, X_j are conditionally independent given Y, $H(X_A|Y)$ is modular. The entropy is submodular, so the mutual information is submodular too.

Now, how do we write this as an optimization problem? Our objective is to maximize the mutual information function $F(A)$. In other words, we want to maximize the difference in uncertainty about the Y's before and after the sensing. Intuitively, the utility function F will be maximized when the sensors are spread apart in the room, so that every location is not too far away from a sensor.

We can write this optimization problem as

$$\max_{S \subseteq \mathcal{V}} F(S) \tag{3}$$

We could try to proceed as before: find a convex relaxation of F and optimize that. But a convex function is not very useful for maximization: although they are easy to minimize, convex functions are hard to maximize. So the Lovász extension doesn't help us here. We could instead try to find a concave extension, e.g. the upper envelope of $F(S)$. Such an extension exists, but computing it is NP-hard. So even though we could potentially find a way to maximize it, computing this function is too hard for this approach to be useful. In other words, the convex framework is not useful anymore.

Indeed, the problem (3) is NP-hard in general. But this should not come as a surprise, since several well-known NP-hard problems such as Maximum Cover and Max-Cut fall in this framework. We will thus have to settle for an approximate solution. Before we look into this, we will analyze some properties of the functions we are trying to maximize.

Note that the coverage function we defined in (1) is always increasing, so it can be trivially maximized by taking the whole set. At this point, we need to make a distinction in the types of set functions we might encounter, since they will lead to optimization problems with different properties. An important property of set functions is monotonicity.

Definition 2.1. *A set function $F : 2^{|V|} \to \mathbb{R}$ is said to be monotone if for any $S \subseteq T$, $F(S) \leq F(T)$.*

Note that the graph cut function is not monotone: at some point, including additional nodes in the cut set decreases the function. In general, in order to test whether a given a function F is monotone increasing, we need to check that $F(S) \leq F(T)$ for every pair of sets S, T. However, if F is submodular, we can verify this much easier. Let $T = S \cup \{e\}$, we need to check that

$$F(T) - F(S) = F(S \cup e) - F(S) \geq 0$$

but by submodularity,

$$F(S \cup e) - F(S) \geq F(V) - F(V \setminus e)$$

so it suffices to look at $F(V) - F(V \setminus e)$.

Clearly, when F is monotone we will need to add constraints to avoid a trivial solution. For the sensor problem, a natural constraint would be to choose at most k sensors, or to spend at most at certain amount on them. The optimization problem in this case would be:

$$\max_{|S| \leq k} F(S) \tag{4}$$

In summary, for monotone functions we will consider constrained problems of the form (4), while for non-monotone functions we can simply use (3). Both of these versions are NP-hard, so we will seek approximate solutions in either case.

2.2 Summarization

Submodular maximization arises naturally in the problem of document summarization. Suppose we have a large collection of documents or images and we want to extract a small subset that is as representative as possible of the collection as a whole, and not redundant.

A common way to model this is with a function of the form

$$F(S) = R(S) + D(S)$$

where $R(S)$ measures the *relevance* and $D(S)$ measures the *diversity* of the set S. Different options for these two functions have been explored. For example, we can take

$$R(S) = \sum_{a \in V} \max_{b \in S} s_{a,b}$$

where $s_{a,b}$ is a similarity measure of elements a and b. This function scores S by how similar every element in V is to its most representative element in S. Naturally, the best we can do is to take $S = V$. This function is also known as the *facility location function*.

Now, for diversity, we can use something we have already seen:

$$D_1(S) = \sum \sqrt{|S \cap P_j|}$$

Here we group elements into clusters P_j and we have a different scoring function for every cluster. This function has diminishing returns: the more points we choose from a cluster, the lower gain we get. In fact, recall that we proved in one of the problem sets that if g is concave then $g(|S|)$ is submodular, so in this case D is indeed submodular.

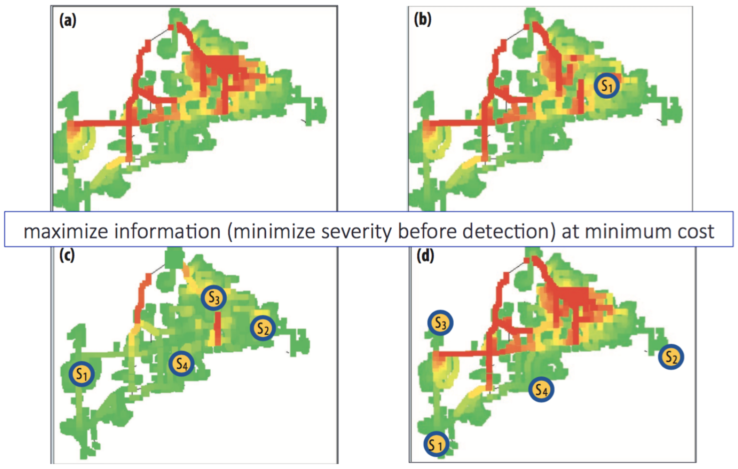
Another option for D is

$$D_2(S) = -\sum_{a,b \in S} s_{a,b}$$

In this case we penalize if we have too many similar points in S. Note that $\sum_{a,b \in S} s_{a,b}$ is supermodular, so D_2 is submodular. A crucial difference between these two diversity functions is that D_2 is decreasing, while D_1 is increasing.

2.3 Networks

Another problem where submodular functions have been successfully applied is in the context of sensing in water networks. The goal is to to detect as quickly as possible a malicious contamination of the water system with a limited number of sensors.


(a)
(b)
(c)
(d)
maximize information (minimize severity before detection) at minimum cost
S1
S2
S3
S4

One way to model this is with

$$F(S) = \sum_{i \in I} P(i) \max_{s \in S} M_{is} \tag{5}$$

where I are infection points, $p(i)$ is the probability of the infection starting at i and M_{is} measures the gain of placing a sensor in s if the infection happens at i. Computing M_{is} might be complicated, so it could require simulation or approximation.

A function similar to (5) also works for the problem of finding relevant topics to include in a blog. Here, each blog is a casacade, people read about it and propagate the information.

Here the idea would be to capture topics early on in the cascade, so that more people will be exposed to it and blog about it.

Yet another problem that can be phrased as submodular optimization and which also uses cascades is network inference. Now we have data from an unknown network where information spreads from a node to its neighboring nodes. We observe a set of cascades $t^c(t_1, \ldots, t_n)$, where t_i is the time at which node i was activated. The ordering of the t_i's gives us information about the way information spreads in the network. From these observations we want to infer the structure of the underlying graph. One possible approach is probabilistic: we can model a distribution of the cascades given the graph a look for a maximum likelihood configuration. The likelihood for a cascade can be expressed as

$$f(t^c|\mathcal{G}) = \sum_{T \in \mathcal{G}} f(t^c|T)P(T|\mathcal{G})$$

where the hidden variables T are possible trees in the graph. The matrix tree theorem makes it possible to sumer over the trees in a graph. The objective is to pick edges to maximize this (log) likelihood, i.e. edges that explain the observed cascades. The intuition is that if multiple trees overlap in a certain edge e, then picking that edge explains many of them and makes it more likely. The diminishing returns arise from the fact that after adding that edge, other edges from those trees don't help that much.

3 Solving the optimization problem

3.1 Cardinality constraints

The general constrained submodular maximization problem that we want to solve has the form

$$\max_S F(S) \text{ s.t. } |S| \leq k \tag{6}$$

where F is monotone and nondecreasing. As mentioned before, this problem is NP-hard, so we will solve it approximately. The most basic algorithm one might think of is a greedy algorithm that in every iteration picks the item that brings the best increase in the objective function F:

```
$S_0 \leftarrow \emptyset$
for $i = 0, \ldots, k-1$ do
    $e^* = \arg\max_{e \in \mathcal{V} \setminus S_i} F(S_i \cup \{e\})$
    $S_{i+1} = S_i \cup \{e^*\}$
end for
```

How good is this greedy heuristic? Note that bad. The following theorem provides a lower bound for the solution.

Theorem 3.1 (Nemhauser, Fisher, Wolsey '78). *Let F be monotone submodular, and S_k the solution of the greedy algorithm. Then*

$$F(S_k) \geq \left(1 - \tfrac{1}{e}\right) F(S^*) \tag{7}$$

Note that S^* is the optimal solution of size k. We will prove a more general version of this result instead. For this, we will make use of the following decomposition

Remark 1. *For a submodular function F and set $S = \{e_1, \dots, e_k\}$ where the e_i are sorted according to their value under F, we can write*

$$\begin{aligned} F(S) &= F(e_1) + F(e_1, e_2) - F(e_1) + F(e_1, e_2, e_3) - F(e_1, e_2) + \dots \\ &= F(e_1) + F(e_2|e_1) + F(e_3|e_1, e_2) + \dots + F(e_k|e_1, \dots, e_{k-1}) \\ &= \sum_{j=1}^{k} F(e_j|\{e_1, \dots, e_{j-1}\}) \end{aligned}$$

Theorem 3.2. *Let S_i be the greedy solution after i steps. Then $\forall k, l$:*

$$F(S_l) \geq \left(1 - e^{-\frac{l}{k}}\right) \max_{|S| \leq k} F(S) \tag{8}$$

Proof. The main idea is to show that the algorithm makes enough progress in each iteration. Let $S^* = \arg\max_{|S| \leq k} F(S)$.

$$F(S^*) \leq F(S^* \cup S_l) = F(S_l) + \sum_{j=1}^{k} F(e_j|\{e_1, \dots, e_{j-1}\} \cup S_l)$$

where we used the above Remark. Note that the terms in the sum are marginal gains, and since F is submodular, we can bound them by $F(e_j|S_l)$. Furthermore, by definition of the greedy algorithm each of these is upper bounded by $F(S_{l+1}) - F(S_l)$. Therefore

$$F(S^*) \leq F(S_l) + \sum_{j=1}^{k} [F(S_{l+1}) - F(S)] = F(S_l) + k[F(S_{l+1}) - F(S)]$$

Re-arranging these terms yields

$$[F(S_{l+1}) - F(S_l) \geq \frac{1}{k}[F(S^*) - F(S_l)] \tag{9}$$

The term $F(S^*) - F(S_l)$ is the gap to optimality, so the inequality above shows that in each iteration we make at least a $\frac{1}{k}$ fractional progress towards optimality.

Now define $\delta_l = F(S^*) - F(S_l)$. Inequality (9) can be written as $\delta_l - \delta_{l+1} \geq \frac{1}{k}\delta_l$, so

$$\delta_{l+1} \leq [1 - \tfrac{1}{k}]\delta_l$$

By recursion, we get

$$\delta_{l+1} \leq (1 - \tfrac{1}{k})^{l+1}\delta_o$$

Now, $F(\emptyset) = 0$, so $\delta_0 = F(S^*) - F(\emptyset) = F(S^*)$. Therefore,

$$F(S^*) - F(S_l) \leq \left(1 - \tfrac{1}{k}\right)^l F(S^*) \leq e^{-\frac{l}{k}} F(S^*)$$

which, after rearranging, yields

$$F(S_l) \geq \left(1 - e^{-\frac{l}{k}}\right) F(S^*)$$

□

We can derive a similar result for an alternative formulation of the problem. Consider now the problem of minimizing the size of the set S subject to a function value above a certain threshold:

$$\min_{S \in \mathcal{V}} |S| \quad \text{s.t.} \quad F(s) \geq q \tag{10}$$

This version of the problem can again be solved by the greedy algorithm, but using now a stopping criterion on the threshold $q, 0 \leq q \leq F(V)$. In this case:

Theorem 3.3 (Wolsey). *If F is monotone, submodular and integer valued:*

$$|S_G| \leq \left(1 + \ln \max_{e \in V} F(e)\right)|S^*| \tag{11}$$

Theorem 3.3 says that using the greedy algorithm will yield a solution that picks at most logarithmically more elements than the optimal one.

3.2 Matroid constraints

Beyond cardinality, there might be other types of constraints that we are interested in. For example, for the problem of choosing locations for security cameras, we not only want to pick k cameras but also pick the direction to which they point at. We can model this using a ground set of cameras and directions $V = \{1_a, 1_b, \ldots, 5_a, 5_b\}$, where $P_i = \{i_a, i_b\}$ are the possible directions for camera i. Since we can only select one direction for each camera, we require $|S \cap P_i| \leq 1$. This is a (partition) matroid constraint.

Another problem with this type of constraints is the submodular welfare problem, where we seek to to assign set S_i to person i to maximize $\sum_{i=1}^{k} F_i(S_i)$, giving each item to at most one person. To enforce this constraint, we can define $\mathcal{V}$ as the set of all possible (object,person) assignments and add a partition matroid constraint of choosing at most one pair for each object.

These two problems have the general form

$$\max_{S \in \mathcal{V}} F(S) \text{ s.t. } S \in \mathcal{I} \tag{12}$$

where $\mathcal{I}$ is the collection of independent sets in the matroid. For this problem we can use an analogous greedy algorithm:

```
$S_0 \leftarrow \emptyset$
**while** There is an element to add **do**
    $e^* = \arg\max_e F(S_i \cup \{e\})$ s.t. $S_i \cup e \in \mathcal{I}$
    $S_{i+1} = S_i \cup \{e^*\}$
**end while**
```

This algorithm gives a weaker bound than before: $F(S_G) \geq \frac{1}{2}F(S^*)$. We can even run this algorithm for an *independence system*, i.e. one that only satisfies Axioms 1 and 2 in the definition of matroids. In this case, the bound will be of the form $\frac{1}{p+1}$, where p depends on the differences between the size of the largest and smallest maximal independent sets.

Other types of constraints can be solved with (some version of) the greedy algorithm, such as budget constraints, but we will not conver those today due to lack of time (see slides for details).

3.3 Empirical performance of the greedy algorithm

As we showed before, in the worst case the greedy algorithm will be $(1 - \frac{1}{e}) \approx 0.63$ away from the optimal solution. However, in practice it tends to perform much better than that, often above 90% of optimality. The reason for this is that the algorithm is actually optimal for linear functions, and decays towards the worst-case bounds for nonlinear ones. In fact, we can quantify how different from the optimal solution the greedy one will be by using a measure of nonlinearity of the function.

We define the curvature C of a submodular function as

$$C = \max_{e \in V, F(e) > 0} 1 - \frac{F(e \mid V \setminus e)}{F(e)} \in [0, 1]$$

The three possible cases for the value of C are

1. For modular functions, $C = 0$ and the greedy algorithm is optimal.
2. If $F(e|V \setminus e) = 0$, then $C = 1$ and we get the worst-case bound $(1 - \frac{1}{e})$.

3. If $C \in (0, 1)$ then the bound is $\frac{1}{C}\left(1 - \frac{1}{e^c}\right) F(S^*)$

More analysis on how curvature affects the performance of greedy algorithm can be found in [4, 1].

We will conclude today's lecture with one final observation. It can be shown that feature selection is not submodular. However, we can still run the greedy algorithm on these problems. In fact, popular methods for feature selection from statistics like orthogonal matching pursuit run some type of greedy algorithm and work relatively well. This, again, can be explained by a notion of linearity similar to the one defined above, the *submodularity ratio* [3, 2]

$$\gamma_k = \min_{S \cap T = \emptyset} \frac{\sum_{e \in S} F(T \cup e) - F(T)}{F(T \cup S) - F(T)} \tag{13}$$

Using this, we can express the bound for the greedy algorithm as $F(S_G) \geq (1 - \frac{1}{e^\gamma})F(S^*)$. Some results on the performance of greedy methods for feature selection can be found in the slides.

References

[1] M. Conforti and G. Cornuéjols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem. *Discrete applied mathematics*, 7(3):251–274, 1984.

[2] A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *ICML*, 2011.

[3] K. D. Johnson, R. A. Stine, and D. P. Foster. Submodularity in statistics: Comparing the success of model selection methods. *arXiv e-prints*, (arXiv:1510.06301), 2015.

[4] D. Sharma, A. Kapoor, and A. Deshpande. On greedy maximization of entropy. In *ICML*, 2015.