Journal of Machine Learning Research 11 (2010) 1141-1144 Submitted 6/09; Published 3/10

SFO: A Toolbox for Submodular Function Optimization

Andreas Krause KRAUSEA@CALTECH.EDU
Computer Science
California Institute of Technology
Pasadena, CA 91125 USA

Editor: Soeren Sonnenburg

Abstract

In recent years, a fundamental problem structure has emerged as very useful in a variety of machine learning applications: Submodularity is an intuitive diminishing returns property, stating that adding an element to a smaller set helps more than adding it to a larger set. Similarly to convexity, submodularity allows one to efficiently find provably (near-) optimal solutions for large problems. We present SFO, a toolbox for use in MATLAB or Octave that implements algorithms for minimization and maximization of submodular functions. A tutorial script illustrates the application of submodularity to machine learning and AI problems such as feature selection, clustering, inference and optimized information gathering.

1. Introduction

Convex optimization has become a powerful tool in machine learning: Surprisingly, many problems that intuitively require the optimization of highly multi-modal objectives, such as clustering and non-linear classification, can be reduced to convex programs, allowing efficient and optimal solution. More formally, they require finding a solution $\mathbf{x}^* \in \mathbb{R}^d$:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} g(\mathbf{x}) \text{ s.t. } \mathbf{x} \in \mathfrak{F},$$

where g is a convex function, and $\mathfrak{F} \subseteq \mathbb{R}^d$ is a (convex) set of feasible solutions.

However, many optimization problems in machine learning, such as feature selection, structure learning and inference in discrete graphical models, require finding solutions to *combinatorial* optimization problems: They can be reduced to the problem

$$\mathcal{A}^* = \operatorname*{argmin}_{\mathcal{A} \subseteq \mathcal{V}} F(\mathcal{A}) \text{ s.t. } \mathcal{A} \in \mathfrak{F},$$

where F is a set function $F : 2^{\mathcal{V}} \to \mathbb{R}$ defined over a finite set $\mathcal{V}$, and $\mathfrak{F} \subseteq 2^{\mathcal{V}}$ is a collection of feasible subsets of $\mathcal{V}$, for example, all sets of size at most k, $\mathfrak{F} = \{\mathcal{A} \subseteq \mathcal{V} : |\mathcal{A}| \leq k\}$.

In many machine learning problems, the function F satisfies *submodularity*, an intuitive diminishing returns property, stating that adding an element to a smaller set helps more than adding it to a larger set. Formally, for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $s \in \mathcal{V} \setminus \mathcal{B}$ it must hold that $F(\mathcal{A} \cup \{s\}) - F(\mathcal{A}) \geq F(\mathcal{B} \cup \{s\}) - F(B)$. Similarly to convexity, submodularity allows one to efficiently find provably (near-) optimal solutions for large problems. Interestingly, for submodular functions, guarantees can be obtained both for minimization and for maximization problems. This is important, since applications require both minimization (e.g., in clustering, inference and structure learning) and maximization (e.g., in feature selection and optimized information gathering). We present SFO, a toolbox[1] for use in MATLAB or Octave that implements various algorithms for *minimization and maximization* of submodular functions. Examples illustrate the application of submodularity to machine learning and AI problems such as clustering (Narasimhan et al., 2005), inference in graphical models (Kolmogorov and Zabih, 2004) and optimized information gathering (Krause et al., 2006).

2. Implementation of Submodular Functions

The SFO toolbox includes several examples of submodular functions. It is also easily extendable with additional functions. The ground set $\mathcal{V}$ is implemented as a MATLAB array. Submodular functions are implemented as MATLAB objects, inheriting from sfo_fn. The following shows example code defining the submodular function

$$F(\mathcal{A}) = I(X_{\mathcal{A}}; X_{\mathcal{V}\setminus\mathcal{A}}) = H(X_{\mathcal{V}\setminus\mathcal{A}}) - H(X_{\mathcal{V}\setminus\mathcal{A}} \mid X_{\mathcal{A}}),$$

that is, the mutual information between a set of random variables $X_{\mathcal{A}}$ and its complement $X_{\mathcal{V}\setminus\mathcal{A}}$, based on a joint multivariate normal distribution $P(X_{\mathcal{V}} = \mathbf{x}_{\mathcal{V}}) = \mathcal{N}(\mathbf{x}_{\mathcal{V}}; 0, \Sigma)$ with covariance matrix $\Sigma \in \mathbb{R}^{100\times 100}$:

```
V = 1:100;
F = sfo_fn_mi(Sigma,V);
F(1:3) % evaluate F on set A=[1,2,3]
```

This objective function has been used for experimental design in Gaussian processes (Krause et al., 2008), structure learning (Narasimhan and Bilmes, 2004) and clustering (Narasimhan et al., 2005). Often, algorithms require computing marginal increments

$$\delta_s^+(\mathcal{A}) = F(\mathcal{A} \cup \{s\}) - F(\mathcal{A}) \text{ and } \delta_s^-(\mathcal{A}) = F(\mathcal{A} \setminus \{s\}) - F(\mathcal{A}),$$

that is, computing the change in submodular value by adding (removing) an element s from a set $\mathcal{A}$. Often, computing $F(\mathcal{A} \cup \{s\})$ (or $F(\mathcal{A} \setminus \{s\})$) is more efficient when $F(\mathcal{A})$ has already been computed. E.g., for mutual information, incrementally computing $F(\mathcal{A} \cup \{s\})$ requires up-/downdating of the Cholesky decomposition of covariance matrix $\Sigma_{\mathcal{A}\mathcal{A}}$. To speed up computation, the submodular function objects in SFO support methods inc and dec:

```
F = init(F,1:5); % cache computation of F(1:5)
inc(F,1:5,9) % efficient evaluation of F([1:5 9])
dec(F,1:5,3) % efficient evaluation of F([1:2 4:5])
```

The SFO toolbox implements several other examples of submodular functions, including

sfo_fn_entropy	Entropy of multivariate Gaussians
sfo_fn_infogain	Information gain for multivariate Gaussians
sfo_fn_mi	Mutual information in multivariate Gaussians
sfo_fn_varred	Variance reduction in multivariate Gaussians
sfo_fn_detect	Improvement in detection performance
sfo_fn_cutfun	Cut function in graphs
sfo_fn_ising	Energy in ising models with attractive potentials

1. The toolbox is available at `http://www.submodularity.org`.

Creating submodular functions from other submodular functions is also possible, using sfo_fn_lincomb for nonnegative linear combinations, and sfo_fn_trunc for truncation. Custom submodular functions can be used either by inheriting from sfo_fn, or by using the sfo_fn_wrapper function, which wraps a pointer to an anonymous function in a submodular function object. The following example wraps an anonymous function fn which computes, for any set of integers $\mathcal{A}$, the number of distinct remainders modulo 5:

```
fn = @(A) length(unique(mod(A,5)));
F = sfo_fn_wrapper(fn);
F([1 6]) % returns 1
F([1:10]) % returns 5
```

3. Implemented Algorithms for Submodular Function Optimization

SFO implements various algorithms for (constrained) maximization and minimization of submodular functions. Their use is demonstrated in sfo_tutorial and sfo_tutorial_octave.
Minimization of Submodular Functions

- sfo_min_norm_point: The minimum norm point algorithm of Fujishige (2005) for solving $\mathcal{A}^* = \operatorname{argmin}_{\mathcal{A}\subseteq\mathcal{V}} F(\mathcal{A})$ for general submodular functions.
- sfo_queyranne: Algorithm of Queyranne (1995) solving $\mathcal{A}^* = \operatorname{argmin}_{\mathcal{A}\subseteq\mathcal{V}:0<|\mathcal{A}|<|\mathcal{V}|} F(\mathcal{A})$ for symmetric submodular functions (i.e., $F(\mathcal{A}) = F(\mathcal{V}\setminus\mathcal{A})$ for all sets $\mathcal{A}$).
- sfo_ssp: The submodular-supermodular procedure of Narasimhan and Bilmes (2006) for (heuristically) minimizing the difference between two submodular functions $\mathcal{A}^* = \operatorname{argmin}_{\mathcal{A}\subseteq\mathcal{V}} F_1(\mathcal{A}) - F_2(\mathcal{A})$.
- sfo_s_t_min_cut: Solves $\mathcal{A}^* = \operatorname{argmin}_{\mathcal{A}\subseteq\mathcal{V}} F(\mathcal{A})$ s.t. $s \in \mathcal{A}, t \notin \mathcal{A}$.
- sfo_greedy_splitting: The algorithm of Zhao et al. (2005) for submodular clustering

Maximization of Submodular Functions

- sfo_greedy_lazy: The greedy algorithm of Nemhauser et al. (1978) for constrained maximization / coverage, using the lazy evaluation technique of Minoux (1978).
- sfo_cover: Greedy coverage algorithm using lazy evaluations.
- sfo_celf: The CELF algorithm for approximately solving $\mathcal{A}^* = \operatorname{argmax}_{\mathcal{A}} F(\mathcal{A})$ s.t. $C(\mathcal{A}) \leq B$, for linear cost function C (Leskovec et al., 2007).
- sfo_ls_lazy: The (deterministic) local search algorithm of Feige et al. (2007) for unconstrained maximization of nonnegative submodular functions, using lazy evaluations.
- sfo_pspiel: The pSPIEL algorithm of Krause et al. (2006). pSPIEL approximately solves $\mathcal{A}^* = \operatorname{argmax}_{\mathcal{A}} F(\mathcal{A})$ s.t. $C(\mathcal{A}) \leq B$, where $C(\mathcal{A})$ is the cost of a cheapest path connecting the nodes $\mathcal{A}$ in a graph.
- sfo_saturate: The SATURATE algorithm of Krause et al. (2008) for approximately solving the robust optimization problem $\mathcal{A}^* = \operatorname{argmax}_{|\mathcal{A}|\leq k} \min_i F_i(\mathcal{A})$.
- sfo_balance: The eSPASS algorithm for approximately solving the optimization problem $\max_{|\mathcal{A}_1\cup\cdots\cup\mathcal{A}_k|\leq m} \min_i F(\mathcal{A}_i)$ (Krause et al., 2009).
- sfo_max_dca_lazy: The Data Correcting algorithm for maximizing general (not necessarily nondecreasing) submodular functions (Goldengorin et al., 1999).

Acknowledgments

This research was supported by ONR grant N00014-09-1-1044, NSF CNS-0932392, a gift from Microsoft Corporation and an Okawa Foundation Research Grant.

References

U. Feige, V. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *FOCS*, 2007.

S. Fujishige. *Submodular Functions and Optimization*. Elsevier, 2nd edition, 2005.

B. Goldengorin, G. Sierksma, G. A. Tijssen, and M. Tso. The data-correcting algorithm for the minimization of supermodular functions. *Mgmt Science*, 45(11):1539–1551, 1999.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans Patt An Mach Int (PAMI)*, 26(2):147–159, February 2004.

A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *IPSN*, 2006.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. In *JMLR*, volume 9, 2008.

A. Krause, R. Rajagopal, A. Gupta, and C. Guestrin. Simultaneous placement and scheduling of sensors. In *Information Processing in Sensor Networks*, 2009.

J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, 2007.

M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques, LNCS*, pages 234–243, 1978.

M. Narasimhan and J. Bilmes. Pac-learning bounded tree-width graphical models. In *Uncertainty in Artificial Intelligence*, 2004.

M. Narasimhan and J. Bilmes. A submodular-supermodular procedure with applications to discriminative structure learning. In *NIPS 19*, 2006.

M. Narasimhan, N. Jojic, and J. Bilmes. Q-clustering. In *NIPS*, 2005.

G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

M. Queyranne. A combinatorial algorithm for minimizing symmetric submodular functions. In *SODA*, 1995.

L. Zhao, H. Nagamochi, and T. Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1):167–183, 2005.