Journal of Machine Learning Research 21 (2020) 1-6 Submitted 6/19; Revised 2/20; Published 8/20

apricot: Submodular selection for data summarization in Python

Jacob Schreiber JMSCHR@CS.WASHINGTON.EDU
Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle, WA 98195-4322, USA

Jeffrey Bilmes BILMES@UW.EDU
Department of Electrical & Computer Engineering, University of Washington, Seattle, WA 98195-4322, USA

William Stafford Noble WILLIAM-NOBLE@UW.EDU
Department of Genome Science, University of Washington, Seattle, WA 98195-4322, USA

Editor: Balazs Kegl

Abstract

We present apricot, an open source Python package for selecting representative subsets from large data sets using submodular optimization. The package implements several efficient greedy selection algorithms that offer strong theoretical guarantees on the quality of the selected set. Additionally, several submodular set functions are implemented, including facility location, which is broadly applicable but requires memory quadratic in the number of examples in the data set, and a feature-based function that is less broadly applicable but can scale to millions of examples. Apricot is extremely efficient, using both algorithmic speedups such as the lazy greedy algorithm and memoization as well as code optimization using numba. We demonstrate the use of subset selection by training machine learning models to comparable accuracy using either the full data set or a representative subset thereof. This paper presents an explanation of submodular selection, an overview of the features in apricot, and applications to two data sets. The code and tutorial Jupyter notebooks are available at `https://github.com/jmschrei/apricot`

Keywords: submodular selection, submodularity, big data, machine learning, subset selection.

1. Introduction

Recent years have seen a surge in the number of massive publicly available data sets across a variety of fields. Relative to smaller data sets, larger data sets offer higher coverage of important modalities that exist within the data, as well as examples of rare events that are nonetheless important. However, as data sets become larger, they risk containing redundant data. Indeed, Recht et al. (2018) found almost 800 nearly identical images in the popular CIFAR-10 (Krizhevsky, 2009) data set of images.

Several existing Python packages focus on selecting subsets of features (Pedregosa et al., 2011; Urbanowicz et al., 2018); however, we are not aware of packages that identify subsets of examples from large data sets. Submodular optimization has emerged as a potential solution to this problem (Lin and Bilmes, 2009), by providing a framework for selecting examples that minimize redundancy with each other (Lin and Bilmes, 2010) (for brevity, we refer to this process as "submodular selection"). Specifically, submodular functions are those that, for any two sets X, Y satisfying $X \subseteq Y$ and any example $v \notin Y$, have the "diminishing returns" property $f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y)$. A function is also monotone if $f(X) \leq f(Y)$. In the setting of selecting examples from a large data set (called the "ground set"), a monotone submodular function operates on a subset of the ground set and, due to the diminishing returns property, returns a value that is inversely related to the redundancy. Finding the subset of examples that maximizes this value, subject to a cardinality constraint, is NP-hard. Fortunately, a greedy algorithm can find a subset whose objective value is guaranteed to be within a constant factor $(1 - e^{-1})$ of the optimal subset (Nemhauser and Wolsey, 1978). For a more thorough introduction to submodularity, we recommend Fujishige (2005); Krause and Golovin (2014); Lovász (1983).

Here, we describe apricot, a Python package that implements submodular optimization for the purpose of summarizing large data sets. The implementation of these optimization approaches is extremely efficient due to algorithmic tricks, such as memoization, and code optimization using numba (Lam et al., 2015). These functions are implemented using the API of scikit-learn transformers, allowing them to be easily dropped into existing machine learning workflows. Lastly, because far more submodular functions exist than those currently included, apricot can be easily extended to user-defined submodular functions simply by defining what the gain would be of adding an example to the growing subset. Apricot can be easily installed using `pip install apricot-select`.

2. Organization

Internally, apricot is organized in a similar manner to deep learning packages such as keras (Chollet et al., 2015). Accordingly, the code is comprised of two main components: (1) the selector objects, which are analogous to keras's model objects, that specify the submodular function and cache important statistics to accelerate the selection process (memoization) and (2) the optimizer objects that implement algorithms for selecting the subset. These components are independent of each other in that each optimizer can be used with each function, and custom selectors and optimizers can easily work with existing ones.

2.1. Functions and Selectors

The selectors in apricot follow the form of scikit-learn transformers. As with all scikit-learn models, parameters of the selection process and hyperparameters of submodular function are passed into the constructor upon initialization. The API consists of the three signature functions from scikit-learn: (1) `fit`, which performs the submodular selection step and stores the resulting ranking of examples, (2) `transform`, which uses the stored ranking to select a subset of examples from the data set, and (3) `fit_transform`, which successively calls the `fit` and then the `transform` functions on a data set. The parameters of the selector depend on the function used but include the number of examples to select, the similarity metric to use for graph-based functions, and the optimizer to use. An example of selecting a subset of size 100 from a data set `X` using a facility location function with (negative) Euclidean distance as the similarity measure is

```
from apricot import FacilityLocationSelection
selector = FacilityLocationSelection(100, ‘euclidean’, optimizer=’lazy’, verbose=True)
X_subset = selector.fit_transform(X)
```

Selectors can be written for custom submodular functions by inheriting from the `BaseSelector` (or `BaseGraphSelector`) objects and implementing three private methods. The first method, `_initialize`, initializes the important attributes of the model and optionally caches statistics about an initial subset of data that the process is building upon. The second method, `_calculate_gains`, returns a vector specifying the gain in set function value associated with each unselected element. This method usually takes the majority of time so the built-in selectors rely on numba-accelerated functions that are compiled to efficient machine code using LLVM (Lattner and Adve, 2004). The third method, `_select_next`, takes the next example selected by the optimizer (see below) and updates the cached statistics and attributes.

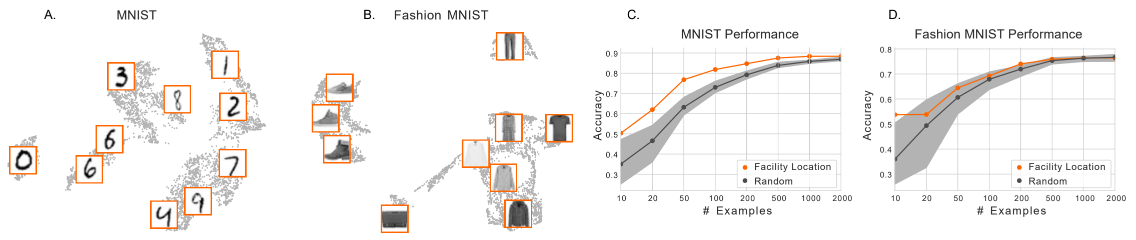

Figure 1: **Example usage of apricot.** (A) A UMAP projection of 10% of the MNIST digits data set (downsampled for visualization purposes), with the first 10 examples selected using the facility location function (orange boxes). (B) The same as A, but for the Fashion MNIST data set. (C) The accuracy (y-axis) of a logistic regression model trained on subsets of varying sizes (x-axis) from the MNIST training data chosen using either facility location (orange) or random selection (grey, band shows min and max accuracy). (D) The same as C but for the Fashion MNIST data set.

2.2. Optimizers

Optimizers encapsulate an algorithm for selecting a subset of data given a submodular function. Most of these algorithms are greedy procedures that select examples one at a time. The optimizer objects in aprricot have a single method, `select`, that takes a data set, a budget, and optionally the cost of including each element in the subset, and chooses the examples that should be in the subset subject to the budget. When costs are passed in the optimization is automatically performed subject to knapsack constraints.

The implementation of the optimization process involves iterative communication between the optimizer and the selector. Although the exact details differ across algorithms, the general idea is that the optimizer retrieves the gain of unselected elements from the ground set through calls to the `_calculate_gains` method of the selector object and, after making a choice based on these gains, informs the selector of this choice using the `_select_next` method. Thus, optimizers are agnostic to the mathematical details of the submodular function that they are optimizing, and selectors are agnostic to the algorithm used to optimize them.

The apricot software implements several built-in optimizers. The simplest optimizer implements the naive greedy algorithm, which simply iterates through each unselected example in the ground set in parallel and then selects the example with the largest gain, repeating this procedure until the budget is exhausted. However, discarding all knowledge of the ranking at each iteration is suboptimal in terms of speed. Accordingly, an extension of this selection procedure, called the accelerated greedy algorithm Minoux (1978), can dramatically improve speed by ordering not-yet-chosen examples using a priority queue. Other optimizers include the stochastic greedy (Mirzasoleiman et al., 2015), sample greedy (Mirzasoleiman et al., 2015), approximate lazy greedy (Wei et al., 2014), and bidirectional greedy algorithms (Feige et al., 2011), as well as GreeDi (Mirzasoleiman et al., 2016), an approximation based on the modular upper-bound, and a two-stage optimizer that switches from using one algorithm initially to a second algorithm after some number of iterations.

3. Applications to Machine Learning

A common application of submodular selection is to select a subset of data for faster training of machine learning models (Lin and Bilmes, 2009; Kirchhoff and Bilmes, 2014; Wei et al., 2015). The reasoning is that the selected set will preserve the diversity of the data, and so yield accurate estimators, but will reduce the redundancy in the data, and thus allow estimators to be trained significantly faster. To illustrate this approach in apricot, we consider two data sets: classifying digits from images in the MNIST data set (LeCun et al., 1998) and classifying articles of clothing from images in the Fashion MNIST data set (Xiao et al., 2017).

First, we visualize the examples selected using a facility location function. Facility location is a canonical submodular function that is parameterized by similarities between pairs of examples, such as correlations or cosine similarities. Specifically, the facility location function takes the form

$$f(X) = \sum_{y \in Y} \max_{x \in X} \phi(x, y) \tag{1}$$

where Y is a data set, X is the selected subset where $X \subseteq Y$, x and y are examples in that data set, and $\phi(x, y)$ is the similarity between the examples. When facility location functions are maximized, the chosen examples tend to represent the space of the data well. This property can be seen when visualizing the MNIST (Figure 1A) and Fashion MNIST (Figure 1B) data sets and highlighting the first 10 examples that are chosen in each data set.

To demonstrate the practical utility of the selected examples, we evaluated logistic regression models trained on subsets of examples from the two data sets. The subsets were chosen solely from the training sets (of 60,000 examples each) using either a facility location function or 20 iterations of random selection. The model is evaluated on the full test set each time. For both MNIST (Figure 1C) and Fashion MNIST (Figure 1D) the models trained using examples selected by submodular selection are more accurate, substantially so on MNIST, than those selected randomly.

4. Discussion

Given the growing popularity of Python in data science, we anticipate that apricot will be a valuable and widely used contribution. However, apricot is not the only implementation of submodular optimization algorithms. One other well known alternative is the Matlab SFO toolbox (Krause, 2010). This toolbox implements several algorithms for both maximizing and minimizing submodular functions. An important distinction between the two is that SFO is a stand-alone toolkit that requires a Matlab license to use, whereas apricot is freely available in Python and integrates with other data science tools such as scikit-learn.

A key challenge in any submodular selection is choosing the right input representation and the right submodular function. Therefore, careful consideration should be given to choosing a similarity measure for graph-based selections, or feature representation for feature-based selections. In many cases, features that are not well suited for particular functions can be transformed using clever tricks. For instance, while the pixel values for images may not themselves be a good input for some submodular functions, the internal representations of these images from a pre-trained neural network might yield higher quality results. Alternatively, data can be clustered using mixture models and feature-based selection run on the posterior probabilities of each example, choosing examples that are close to each cluster center. While apricot does not directly implement these transformations, they would undoubtedly be valuable when used in conjunction with apricot.

Acknowledgments

This work was supported by NSF IGERT award DGE-1258485 and by NIH award U01 HG009395.

References

François Chollet et al. Keras. https://keras.io, 2015.

U. Feige, S. V. Mirrokni, and J. Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, July 2011. ISSN 0097-5397. doi: 10.1137/090779346. URL `https://doi.org/10.1137/090779346`.

S. Fujishige. *Submodular Functions and Optimization.* Number 58 in Annals of Discrete Mathematics. Elsevier Science, 2nd edition, 2005.

K. Kirchhoff and J. A. Bilmes. Submodularity for data selection in machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, October 2014.

A. Krause. SFO: A Toolbox for Submodular Function Optimization. *Journal of Machine Learning Research*, 11:1141–1144, March 2010.

A. Krause and D. Golovin. Submodular function maximization., 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. In *Technical Report, University of Toronto*, 2009.

S.K. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 7:1–7:6, 2015.

C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, CGO '04, page 75, USA, 2004. IEEE Computer Society. ISBN 0769521029.

Y. LeCun, Bottou L., Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

H. Lin and J. A. Bilmes. How to select a good training-data subset for transcription: Submodular active selection for sequences. In *Proc. Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Brighton, UK, September 2009.

H. Lin and J. A. Bilmes. An application of the submodular principal partition to training data subset selection. In *Neural Information Processing Society (NIPS) Workshop*, 2010.

L. Lovász. Submodular functions and convexity. In M. Grotchel A. Bachem and B. Korte, editors, *Mathematical Programming – The State of the Art*, pages 235–257. Springer-Verlag, 1983.

M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques*, 7:234–243, 01 1978.

B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrak, and A. Krause. Lazier than lazy greedy. In *Proc. Conference on Artificial Intelligence (AAAI)*, 2015.

B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization. *Journal of Machine Learning Research*, 17(235):1–44, 2016. URL `http://jmlr.org/papers/v17/mirzasoleiman16a.html`.

G.L. Nemhauser and L.A. Wolsey. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, pages 265–294, 1978.

F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, pages 2825–2830, 2011.

B. Recht et al. Do CIFAR-10 classifiers generalize to CIFAR-10? *CoRR*, abs/1806.00451, 2018.

R. J. Urbanowicz et al. Benchmarking relief-based feature selection methods for bioinformatics data mining. *Journal of Biomedical Informatics*, pages 168–188, 2018.

K. Wei, R. Iyer, and J. Bilmes. Fast multi-stage submodular maximization. In *International Conference on Machine Learning (ICML)*, 2014.

K. Wei, R. Iyer, and J. A. Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning (ICML)*, Lille, France, 2015.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, 2017. cs.LG/1708.07747.