Submodular Function Tuning for Data Subset Selection

Megh Bhalerao Jeff Bilmes

University of Washington

December 1, 2023

Table of Contents

Acknowledgements

- Subset selection is performed using Submarine[1][2].
- Similarity Matrices created using Submarine and FAISS [1], and Matrix transformations are done using Submarine.
- Model training done using PyTorch [3].

[1] https://submarine.page/
[2] https://github.com/melodi-lab/submarine

What is a Submodular Function?

Given any set of elements,

$$V = \left\{ \right.$$  $$\left. \right\}$$

What is a Submodular Function?

- It is a set function, more generally, $f : 2^V \to \mathbb{R}$
- Possess the diminishing returns property - if for every $A \subseteq B \subseteq V$ and every $x \in V \setminus B$, the following inequality holds:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \qquad (1)$$

- Naturally suitable for modeling set diversity.

There are many examples of Submodular Functions like ...

- Joint entropy of a set of Random Variables $X = \{X_1, X_2, \dots, X_n\}$: $H(X) = \sum_{x_j \in X} p(x_j) \log p(x_j)$, where $x_j = (x_1^j, \dots, x_n^j)$, and sum runs over the exponential number of combinations of joint probabilities of random variables.
- Log determinant of a positive semidefinite matrix: $\log\det(X + \epsilon I)$ where $\epsilon > 0$
- Facility location: $f(S) = \sum_{j \in \mathcal{C}} \max_{i \in S} u_{ij}$ where $\mathcal{C}$ is the set of clients
- Set cover function: $f(S) = \sum_{u \in \mathcal{U}} \min(1, \sum_{s \in S} w_{su})$ where $\mathcal{U}$ is the universe of elements
- Graph cut: $f(S) = \sum_{i \in S, j \notin S} c_{ij}$ where c_{ij} is the capacity of the edge between i and j
- Rank function of a matroid: $f(S) = \text{rank}(S)$
- Coverage function: $f(S) = |\cup_{s \in S} \mathcal{C}_s|$ where $\mathcal{C}_s$ is the set covered by s

Which Submodular Function do we use in this work?

- We use the Facility Location (FL) Function.
- Definition of FL -

$$f_V(X) = \sum_{j \in V} \max_{i \in X} s(i, j) \tag{2}$$

where:

- V is the ground set.
- X is the set on which we want to evaluate the FL function.
- i, j - index elements in V and X respectively.
- $s(i, j)$ is the similarity function that measures the affinity or similarity from between elements i and j. $s(i, j)$, may not necessarily equal $s(j, i)$. In fact, we sometimes prefer to have asymmetric similarities.

Facility Location (FL) Function

How do we use the FL function?

We try to find a subset $X^* \in V$ which maximizes it.

Why do we want to maximize FL?

- Intuitively (and loosely) speaking, a set X^* which maximizes FL, is a set whose elements are most similar to all elements of the ground set V.
- Hence, this set can act as a good representative of the ground set V, potentially useful for downstream applications.

How do we maximize FL?

- Submodular Function Maximization is NP hard, in general.
- But, greedy algorithm on function gains has a $1 - 1/e$ guarantee, and works well in practice.

Where does our work fit in the literature?

- To the best of our knowledge, our work is the first one to present a detailed *empirical* evaluation of applying *tuned* submodular functions for the task of data subset selection in a supervised learning setting.
- We show that a well *calibrated (or tuned)* (will be explained later) submodular function can beat popular state-of-the-art data subset selection baselines.
- We mainly focus on computer vision datasets (CIFAR-10/100, TinyImagenet, Imagenet) in this work, but our approach can be extended to other modalities, too.

What our work does not show?

- We do not claim to introduce any new kind of submodular function, but rather show that submodular functions can be tuned to get the right set of parameters, and such a function can be a strong data subset selector.

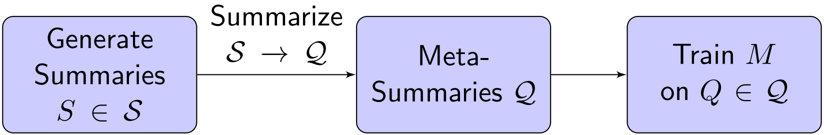
How do we tune the Submodular Function?

To answer this question, we have to first ask what is the objective that we are trying to achieve?

Our Objective

Given a machine learning model M, training algorithm $\mathcal{A}$, a training-dataset V_{tr}, test-dataset V_{te}, we want to find $S \subset V_{tr}$ such that $\mathcal{A}(M, S)$, achieves a highest test-accuracy (a_{te}), i.e. accuracy on V_{te}, where $\mathcal{A}(M, S)$ denotes training a model M on dataset S.

How do we tune the Submodular Function?



Summary Generation Procedure

Input: Dataset $V_{tr} = \{x_i, y_i\}_{i=1}^{i=n}$, Subset size m, Model M, Feature Extractor $\mathcal{F}$, Training Algorithm $\mathcal{A}$, Facility Location Function Parameter Space $\mathcal{X}$. ▷ Calibration Procedure Inputs.

Output: Set of summaries $\mathcal{S}$ ▷ The Output.

procedure GRIDSEARCH
- **for** $y \in \mathcal{X}$ **do** ▷ Loop over set of FL hyperparameters
  - $F \leftarrow \mathcal{F}(V_{tr})$ ▷ Make Design Matrix F
  - $P \leftarrow \textsc{SimMat}(F)$ ▷ Make Similarity Matrix P
  - $f_{fl}(X; V_{tr}, y) = \sum_{v \in V_{tr}} \max_{x \in X} P_y(v, x)$ ▷ Instantiate FL function
  - $I \leftarrow \textsc{GreedyMax}(f_{fl}, m)$
  - $\mathcal{S} \leftarrow \mathcal{S} \cup I$
- **end for**

end procedure

Feature Extractors

- One-epoch trained network features, feature vector tapped from point just before final classification layer.
- Features from CLIP Model [3].
- One-epoch trained network gradient features, i.e. $\nabla_\theta \mathcal{L}(f_\theta(x), y) \in \mathbb{R}^p$, $p =$ number of parameters in the network, similar to CRAIG [2], but gradient w.r.t all parameters instead of only last layer.

General Observation: Gradient Features perform better at smaller subset sizes, whereas a mix of Activation and Gradient work better for larger subset sizes.

[3] https://github.com/openai/CLIP

Similarity Metrics

SIMEUCLID

- Define distance matrix $D(i,j) = ||x_i - x_j||_2^\gamma$, which are euclidean distances raised to power γ. We search over $\gamma \in \{1e-5, 1e-4, 1e-3, 1e-2, 0.1, 0.5, 1, 2, 5, 10\}$.
- Get similarity matrix $P(i,j) = \max_{i,j \in [d]} D - D(i,j)$.

RBF

- Define distance matrix $D(i,j) = ||x_i - x_j||_2^\gamma$, which are euclidean distances raised to power γ. We set $\gamma = 1$ for RBF.
- Get similarity matrix $P(i,j) = e^{\frac{-D(i,j)}{z \times w}}$. z is the divide_by parameter $\in$ {null, Mean, Sum, Min, Max}. w is the kernel width. We search over $w \in \{1e-5, 1e-4, 1e-3, 1e-2, 0.1, 0.5, 1, 2, 5, 10\}$.

Transforms on Similarity Matrices

K-Nearest Neighbor Transform

- Convert similarity P into assymetric sparse KNN graph.
- Given ground set size n, and subset size m, we calculate $h = \frac{n}{m}$, and set $k \in h \times \{0.5, 0.6, 1, 1.5, 2, m\}$

Gravity Transform

- Detailed description of Gravity transform can be found at https://submarine.page/docs/submarine-constructs/ftl#swarpclip
- We use fulrcum $f \in \{1, 50, 75\}$, and gravity value $g \in \{10, 50, 99\}$.

TODO for Hyperparmaters

Add hyperparameter sensitivity analysis.

Meta Summarization

Make Jaccard Matrix

Input: Set of summaries $\mathcal{S}$
Output: Jaccard Matrix J ▷ The Output.

```
procedure MakeJaccardMatrix
    Initialize empty Jaccard Matrix J
    for S_1 ∈ 𝒮 do                                  ▷ Outer Loop.
        for S_2 ∈ 𝒮 do                              ▷ Inner Loop.
            J(i, j) = |S_1 ∩ S_2| / |S_1 ∪ S_2|     ▷ Assign Jaccard Matrix Elements
        end for
    end for
end procedure
```

$$J(i,j) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Meta Summarization

Meta Summarize

Input: Jaccard Matrix J, Set of summaries $\mathcal{S}$, Meta Summary Size mm.

▷ Meta Summarization Inputs.

Output: Summary of summaries (we call them meta-summaries) $\mathcal{Q}$, Meta-summary indices I_{ms} in greedy-max order. ▷ The Output

procedure MetaSummarize

$f_{fl}(X; \mathcal{S}) = \sum_{t \in \mathcal{S}} \max_{x \in X} J(t, x)$ ▷ Index FL via Jaccard Matrix J

$I_{ms} \leftarrow \textsc{GreedyMax}(f_{fl}, mm)$ ▷ Meta-summary indices via greedy max.

$\mathcal{Q} = S(I_{ms})$ ▷ Index actual summaries with I_{ms}

end procedure

Meta-training

Train model on Meta-summaries

Input: Model M, Set of Meta-summaries $\mathcal{Q}$, Training Algorithm $\mathcal{A}$, Test Dataset V_{te} ▷ Meta-training Inputs.

Output: Best performing summary Q^*, and respective best trained model M^*

1: **procedure** MetaTrain

2: $m_{te} = 0$ ▷ Initialize Max Test Accuracy

3: **for** $Q \in \mathcal{Q}$ **do** ▷ Loop over meta-summaries.

4: $a_{te} = \mathcal{A}(M, Q)$ ▷ Train model to get test accuracy.

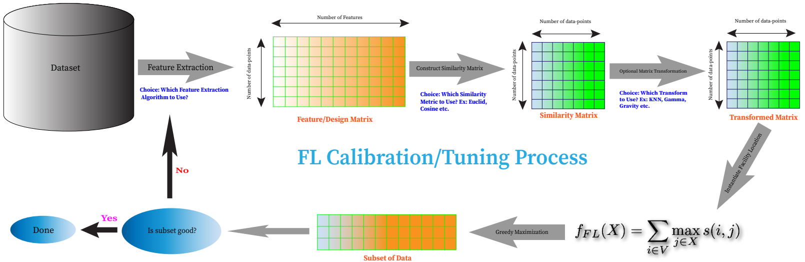
5: $m_{te} = \max(a_{te}, m_{te})$ ▷ Get max test accuracy.

6: **end for**

7: $M^*, Q^* = \arg\max(m_{te})$ ▷ Get best summary and respective trained model.

8: **end procedure**

Flowchat of Tuning Process

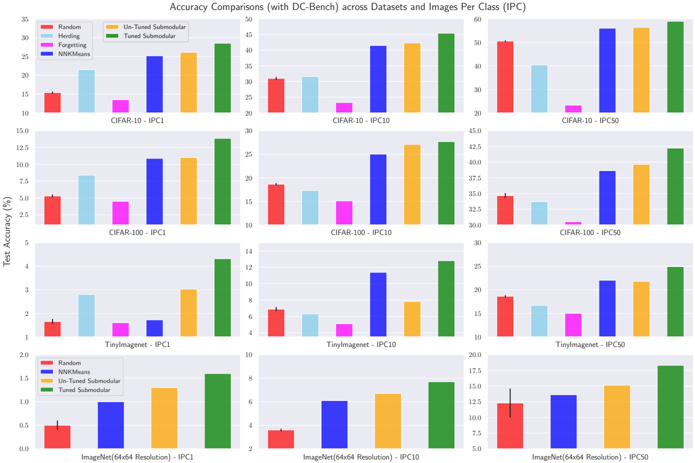


Evaluation Framework

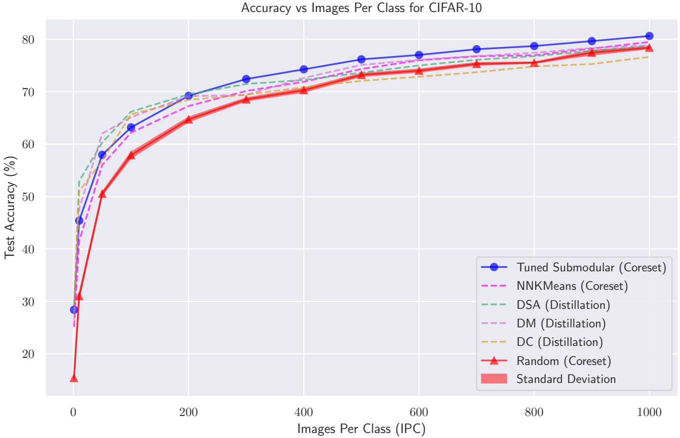
We evaluate our method on the following subset selection benchmarks (mainly on computer vision benchamarks) -

- DC-Bench (https://dc-bench.github.io/)
  - Mainly for Dataset Distillation, but includes some subset selection baselines too.
  - Datasets - CIFAR10, CIFAR100, TinyImagenet, ImageNet (64×64 Resolution)
  - Models M used are ConvNet (Depth $= 3$ & 4)
- DeepCore (https://github.com/PatrickZH/DeepCore)
  - Consists of different subset selection benchmarks into a unified setup.
  - Datasets - CIFAR10, CIFAR100
  - Model used here is ResNet18.
  - In progress - full resolution ImageNet. Have some random results using DaViT model for lowest subset size, but that is not beating resnet18 random, I think that the architecture is way overparameterized for the small subset size.

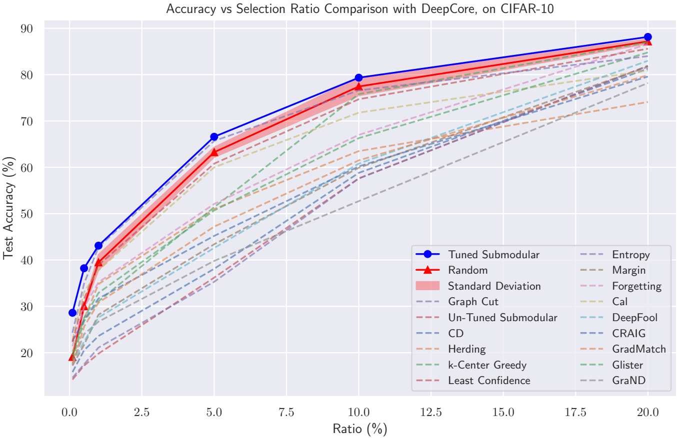
DC-Bench Comparison



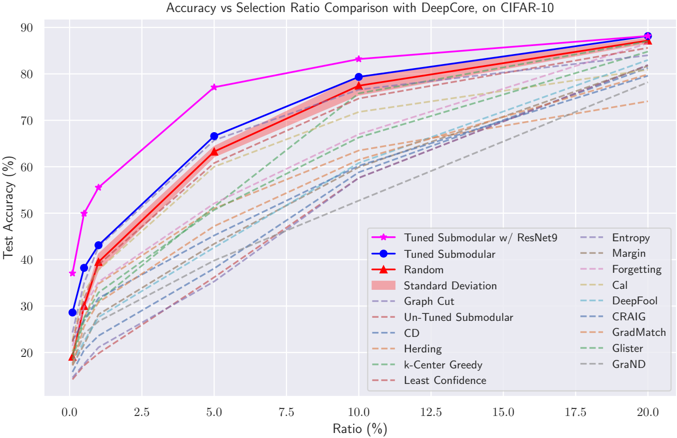
DC-Bench Comparison



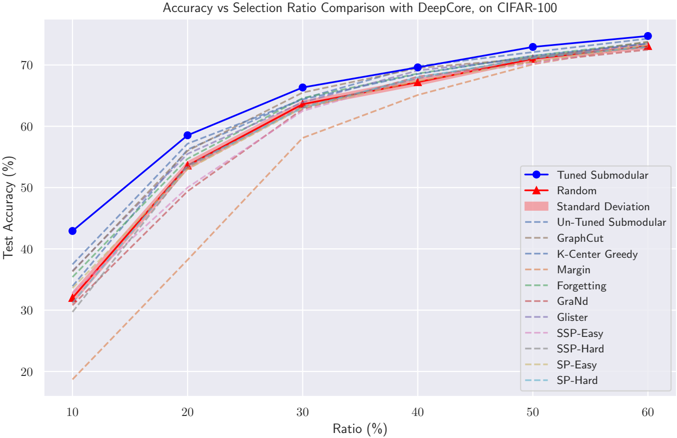
DeepCore Comparison



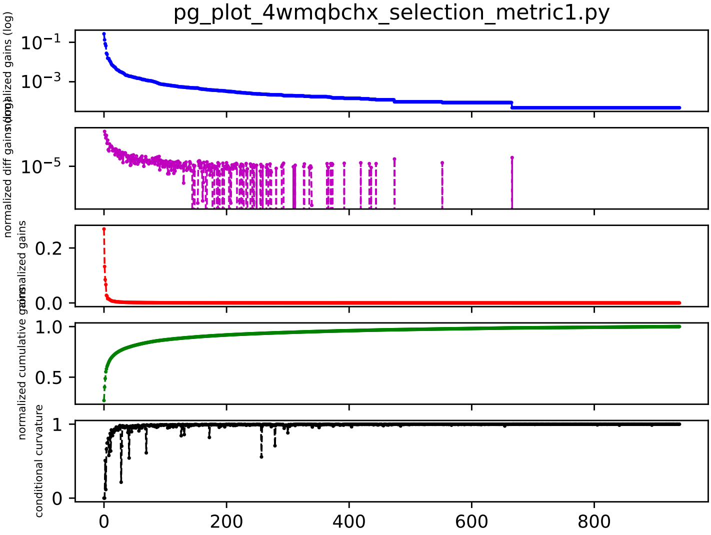
DeepCore Comparison with ResNet9



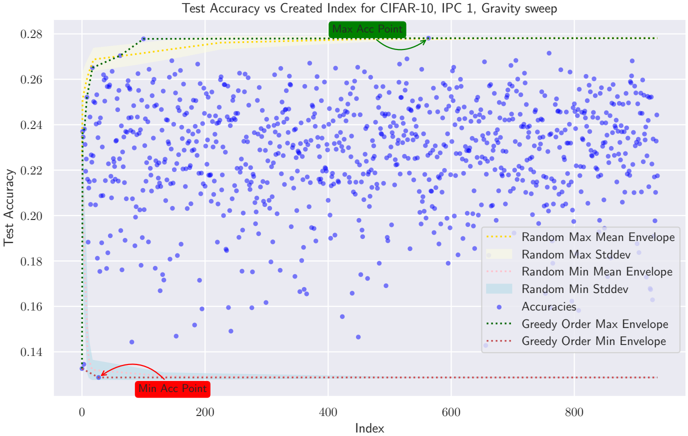
DeepCore Comparison



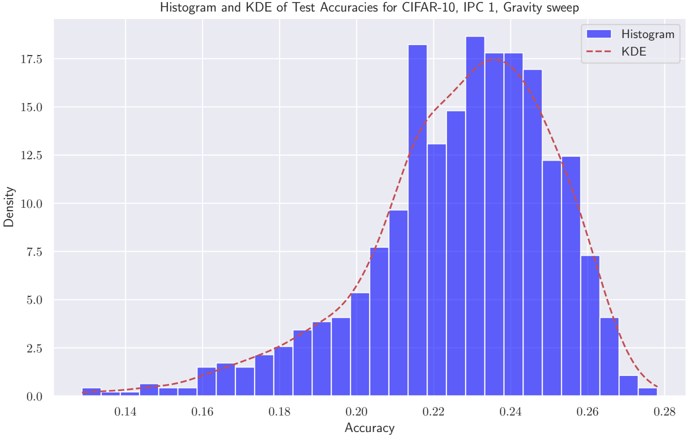
Example Meta Summarization Gains Plot



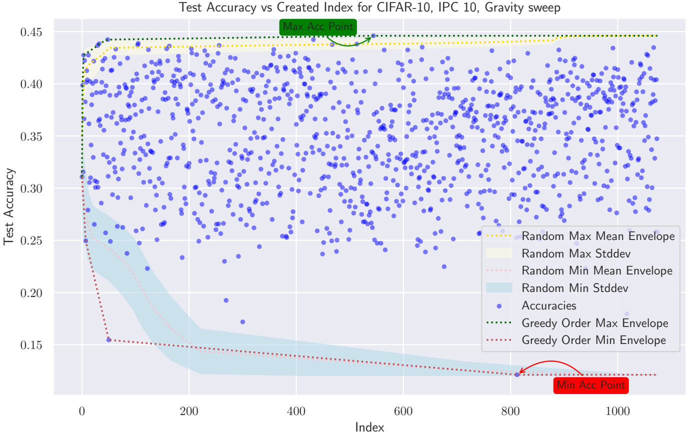
Greedy Training Order Plots & Histograms



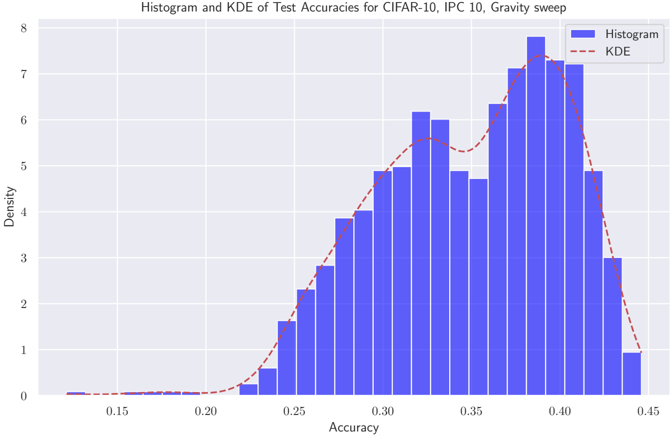
Greedy Training Order Plots & Histograms



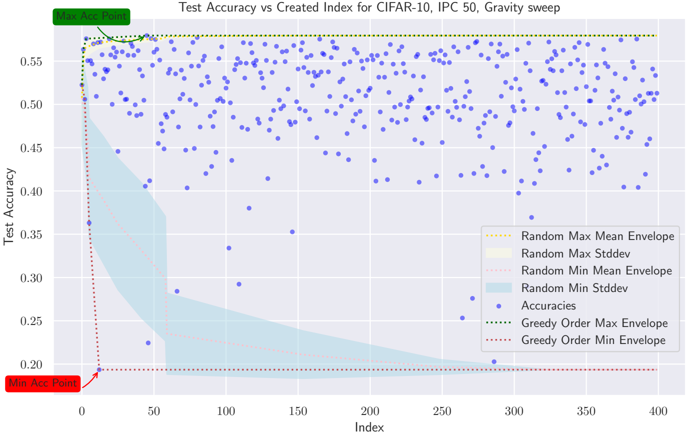
Greedy Training Order Plots & Histograms



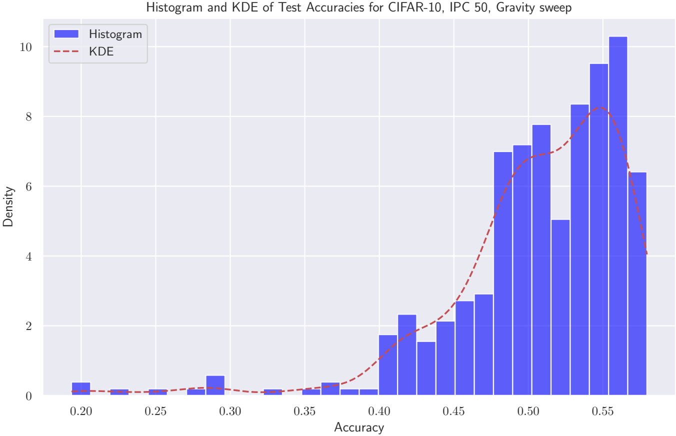
Greedy Training Order Plots & Histograms



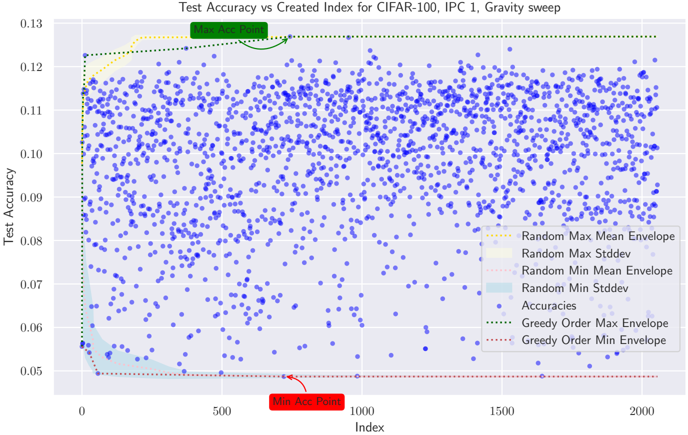
Greedy Training Order Plots & Histograms



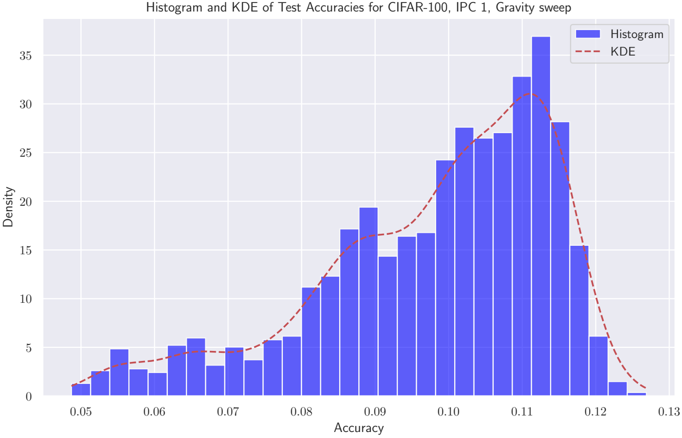
Greedy Training Order Plots & Histograms



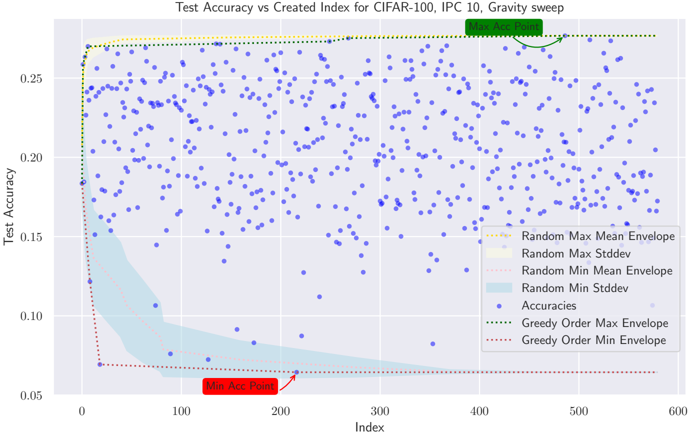
Greedy Training Order Plots & Histograms



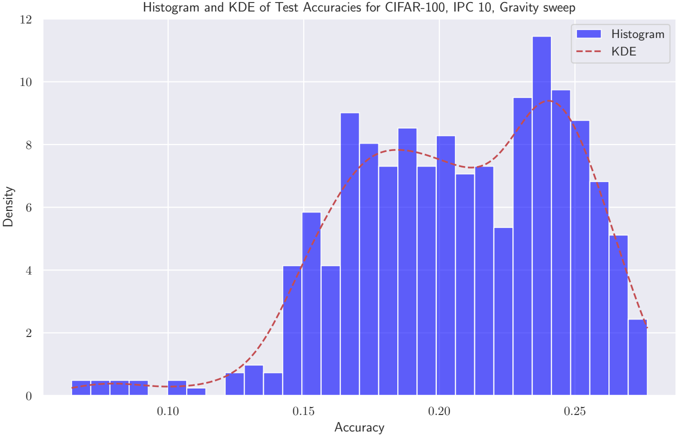
Greedy Training Order Plots & Histograms



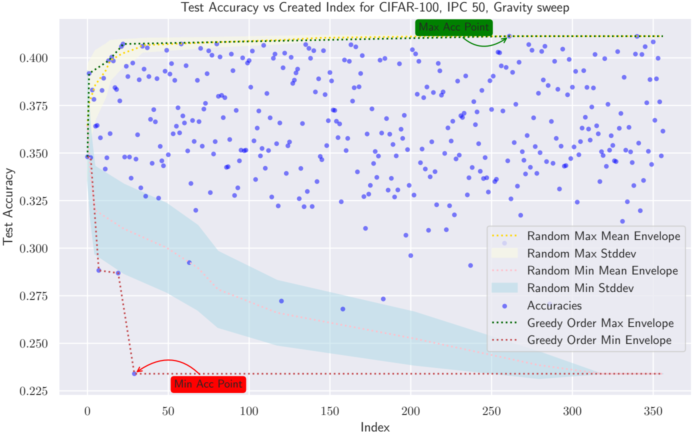
Greedy Training Order Plots & Histograms



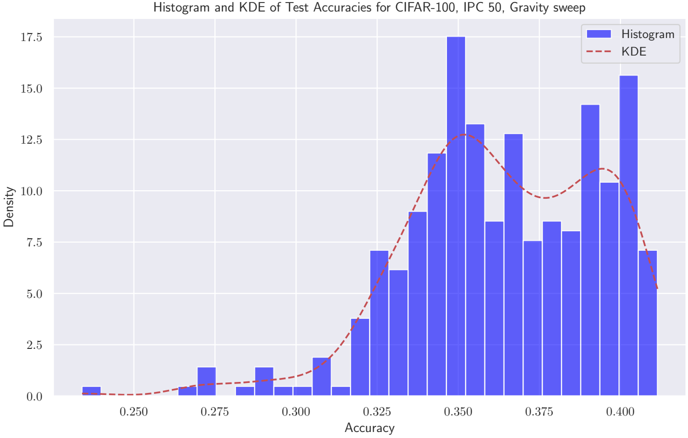
Greedy Training Order Plots & Histograms



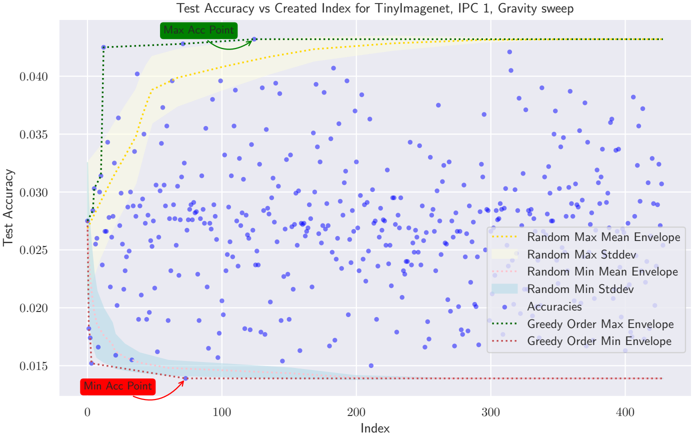
Greedy Training Order Plots & Histograms



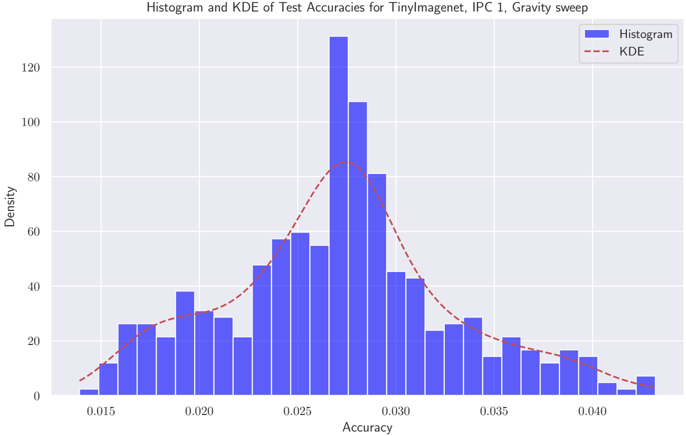
Greedy Training Order Plots & Histograms



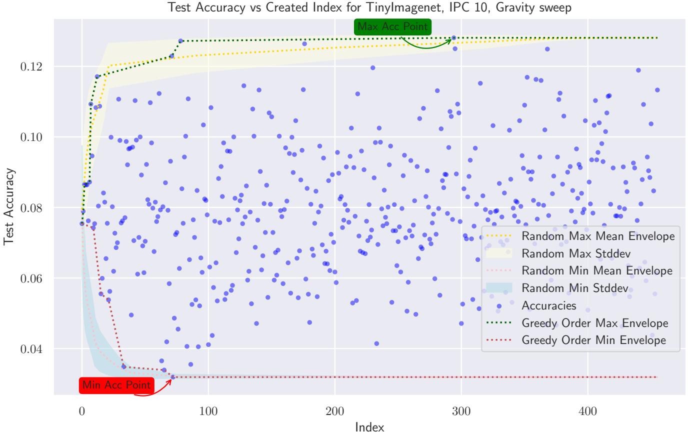
Greedy Training Order Plots & Histograms



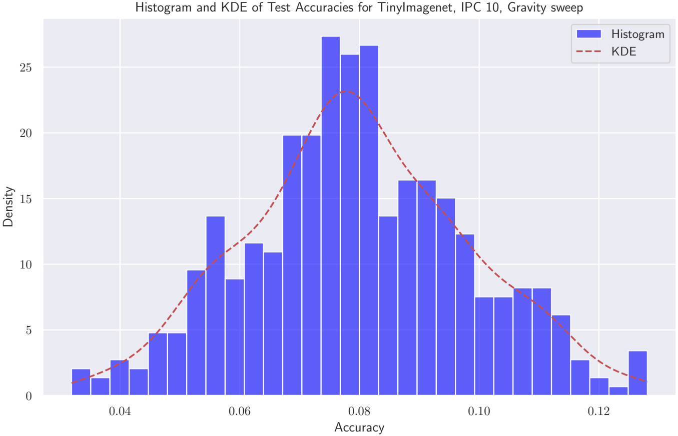
Greedy Training Order Plots & Histograms



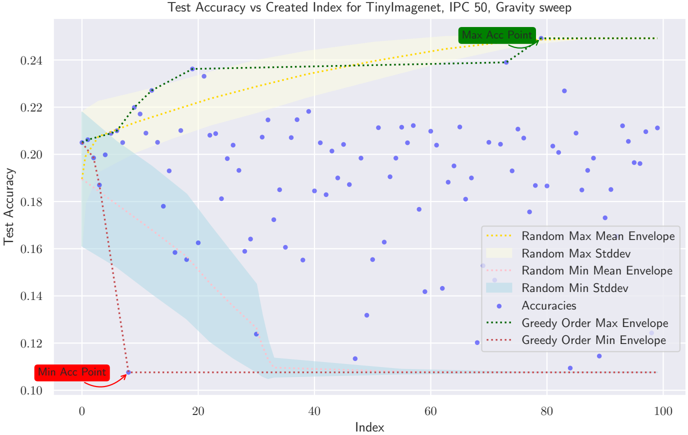
Greedy Training Order Plots & Histograms



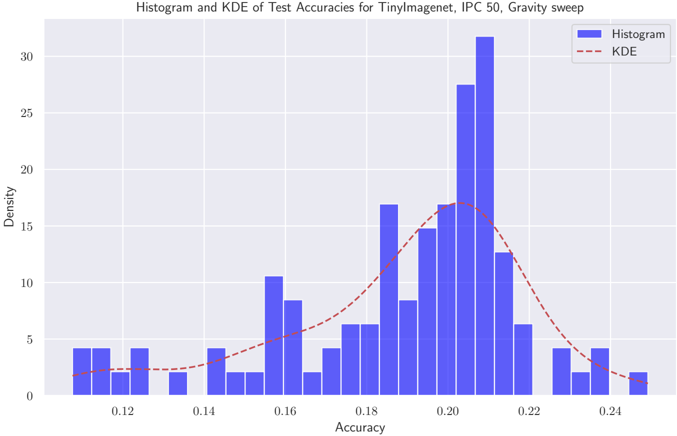
Greedy Training Order Plots & Histograms



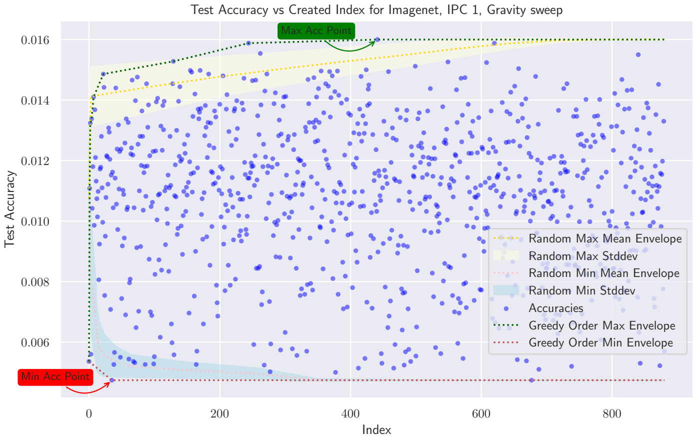
Greedy Training Order Plots & Histograms



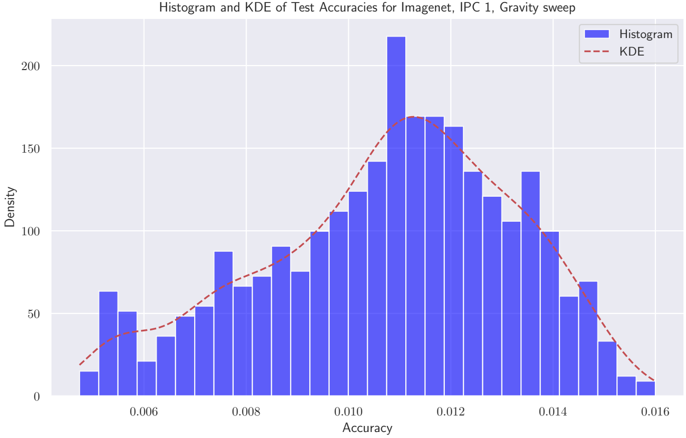
Greedy Training Order Plots & Histograms



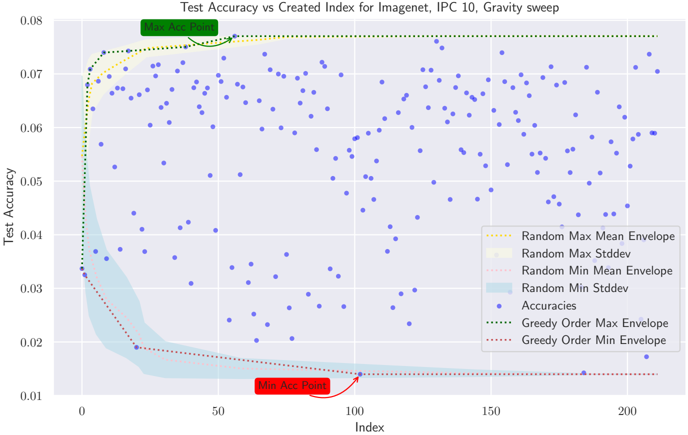
Greedy Training Order Plots & Histograms



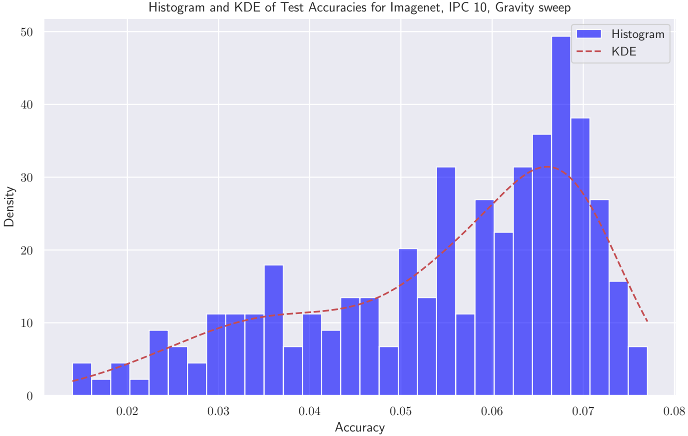
Greedy Training Order Plots & Histograms



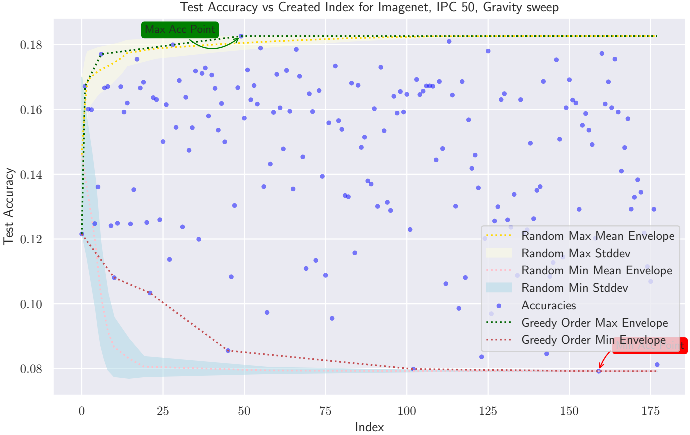
Greedy Training Order Plots & Histograms



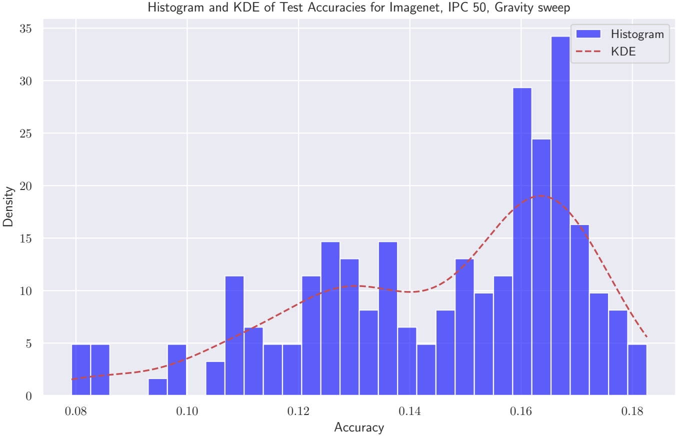
Greedy Training Order Plots & Histograms



Greedy Training Order Plots & Histograms



Greedy Training Order Plots & Histograms



Conclusion

- We show that a well tuned submodular function can act as an effective data subset selector.
- We hope that this can act as a stepping stone for -
  - Using more different types of submodular functions such as GraphCut, DPP etc, for data subset selection.
  - Scaling submodular selection to modern large scale vision and language datasets, and other data modalities.

References I

[1] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[2] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

Thank You!

Questions?