Algorithms for Massive Data Science **Lecture date: April 1, 2019**
Instructor: Ben Moseley **Scribe: Violet Chen**

Sampling is widely used to quickly measure certain quantities of large data sets. A naive sampling may work for some simple problems. For example, if we seek to estimate an approximate median of the input, taking the median of a small sample will be a good approximate measure of the true median with a high probability. However, for many problems, some data points may affect the optimal solution a lot, hence it may not be a good idea to construct a solution only based on a sample impacted by these data points. In this section, we will give a powerful recipe which we term Sample and Prune.

1 Basic setup

We begin with defining Sample and Prune in a general sense. Sample and Prune can be applied to problems which admit a simple greedy algorithm. Given a universe U of elements as input and some objective function, let $\mathcal{G}$ be the greedy algorithm or a close approximation algorithm. We given the following **semi-formal definition of Sample-Prune algorithms**.

1. Initialize the sample $S = \emptyset$.
2. Sample each element in U independently with probability $M/|U|$ and add it to S, where M is the memory size.
3. Construct a tester $\mathcal{T}(\mathcal{S})$ from the output of the algorithm $\mathcal{G}$ on the sample S: typically, the tester rejects element e if and only if running algorithm $\mathcal{G}$ on S and $S \bigcup \{e\}$ would return the same output.
4. Remove from U every element e that fails the tester $\mathcal{T}(\mathcal{S})$.
5. Recurse until U becomes small enough, i.e. $|U| < M$.
6. Solve the problem on the small instance, $S \cup U$; typically we run $\mathcal{G}$ on $S \cup U$.

The above algorithm uses the observation that $S \cup U$ is a good sketch on the input in all iterations: the optimal solution on the reduced input $S \cup U$ is a good approximation to the original input. As mentioned before, in each iteration, we sample a subset of the input which is small enough to fit into a single machine's memory. Then, we broadcast the sample to all machines where each element in the current universe is tested based on the sample. Intuitively, each element is discarded if it is deemed useless for improving the solution when added to the sample S.

A natural way of constructing such a tester, although not unique, is to run the greedy algorithm $\mathcal{G}$ on $S \bigcup \{e\}$. If the greedy algorithm's solution remains the same compared to running on the input S, the element is intuitively useless, thus discarded. The idea is that a large fraction of the elements are discarded in each iteration. Then, we repeat this process until the remaining universe becomes sufficiently small. Once we have a small universe, we put the sample and the small number of elements left in the universe into a single machine, and solve the instance of reduced size.

The analysis of this algorithm consists of two key steps.

- (i) Effectiveness of Pruning: Let R be the final pruned universe. We need to prove that R is small enough to fit in a single machine's memory.
- (ii) Approximation guarantee: we also need to show that solving the reduced instance $S \cup R$ leads to a good solution, either optimal or approximate.

2 Warm up problem: choose top k elements

We now apply Sample and Prune algorithm to solve the problem of choosing largest k elements from a given universe of elements.

Algorithm 1 Choose Top k Elements with Sample and Prune

```
Input: A universe $U$ of $n$ elements (or numbers), greedy algorithm $\mathcal{G}(\mathcal{S})$ which returns the top $k$
    elements from set $S$. Assume each machine has memory size $\sqrt{n}$.
Output: $k$ largest from input elements.
1: Partition $n$ elements into $m$ partitions $U_i$ such that each $U_i$ fits on one machine.
2: Each machine samples elements in $U_i$ independently with uniform probability. Send all samples
    to one machine, denote the final sample as $S$.
3: Let $B = \mathcal{G}(\mathcal{S})$ be the top $k$ numbers from sample $S$.
4: On each machine, discard (or prune) elements smaller than the smallest element in $B$. Let $R$
    be remaining elements after pruning. (This step is constructing a tester: discard element $e$ if
    $G(S \bigcup \{e\}) = G(S) = B$.)
5: Send all elements of $R$ to one machine, run $\mathcal{G}(\mathcal{R})$ and return the output set.
```

Now we show the two key analysis steps. Key step (ii) is straightforward: $\mathcal{G}(\mathcal{S} \bigcup \mathcal{R})$ outputs the top k elements from U. Key step (i) can be shown with the following lemma.

Lemma 1 (Effective pruning) *$|R| = O(k\sqrt{n})$ with high probability.*

Proof: If $|R| \geq 10k\sqrt{n}\log n$, we say this is a bad event. Once B is fixed, R elements are also deterministically fixed. Let $X(B')$ denote a fixed bad event, where B' is computed from our sample. Let $R(B')$ denote the remaining elements if we construct a tester with $B = B'$ in step 4 of algorithm.

We observe that if any element in $R(B')$ is included in the sample S, then B computed in step 3 of the algorithm must satisfy $B \neq B'$, thus the bad event $X(B')$ does not occur. Therefore,

$$\begin{aligned}
\Pr[X(B')] &\leq \Pr[e \notin S, \forall e \in R(B')] \leq (1 - \frac{1}{n^{1/2}})^{|R(B')|} \\
&\leq (1 - \frac{1}{n^{1/2}})^{10kn^{1/2}\log n} \leq \exp(-10k \log n) \leq n^{-10k}
\end{aligned}$$

Since there are at most n^k possible realizations of B, by union bound over all such bad realizations:

$$\Pr[X(B)] \leq \binom{n}{k} n^{-10k} \leq n^{-9k}$$

Since bad event happens with very small probability, we can conclude that $|R| = O(k\sqrt{n})$ *w.h.p.* $\square$

3 k-center clustering

Now we apply Sample and Prune to k-center clustering. We begin with reviewing a sequential greedy approximation algorithm for k-center clustering, then introduce a distributed approximation algorithm.

3.1 Greedy algorithm

Algorithm 2 Greedy k-center Clustering $\mathcal{G}(\mathcal{U})$

```
Input: A universe $U$ of $n$ points in a metric space, $d(u, v)$ as distance between two points $u$ and
    $v$, $d(u, V') := \min_{v \in V'} d(u, v')$
Output: $k$ points/centers $A$ such that $A$ attains $\min_{A' \subseteq U : |A'| \le k} \sum_{u \in U} d(u, A')$
    Suppose optimal objective value $OPT$ is known.
1: Set $A = \emptyset$.
2: Add an arbitrary point in $U$ to $A$, and remove all points from $U$ that are within distance 2OPT
    from $A$.
3: Repeat this until $U = \emptyset$ and output $A$ as the final solution.
```

Theorem 2 *$\mathcal{G}(\mathcal{U})$ is a 2-approximation of optimal k-center clustering.*

Proof: Suppose A is the output from $\mathcal{G}(\mathcal{U})$. If $|A| \le k$, then every point in U is within distance $2OPT$ from A by construction in algorithm.

For sake of contradiction, assume $|A| > k$, then there exist $k+1$ points with distance greater than $2OPT$ from each other. By Pigeon Hole principle, 2 of these points are assigned the same center in optimal clustering (corresponding to OPT), say u, w are two such points assigned to center v_c. Then $d(u, v_c) \le OPT, d(w, v_c) \le OPT$, but $d(u, w) > 2OPT$, which violates triangle inequality. □

3.2 Distributed algorithm

The above greedy algorithm is very sequential in nature and is hard to adapt to the distributed setting. To give a distributed algorithm with sample-prune, the key question to answer is: after taking a sample S, what is the most natural tester one could build from the greedy algorithm?

Let $\mathcal{G}(\mathcal{S})$ denote the output $\mathcal{G}$ gives on the sampled input S. One pruning option is to discard each point u from U if the point is too close from $\mathcal{G}(\mathcal{S})$: within distance $2OPT$ from $\mathcal{G}(\mathcal{S})$.

Algorithm 3 Distributed k-center Clustering with Sample and Prune

```
Input: A universe $U$ of $n$ points in a metric space, distance function $d(u,v)$, $d(u,V') := \min_{v\in V'} d(u,v')$, machines with individual machine memory $\tilde{\Theta}(k\sqrt{n})$
Output: $k$ points/centers $A$ such that $A$ attains $\min_{A'\subseteq U:|A'|\le k} \sum_{u\in U} d(u,A')$
    Suppose optimal objective value $OPT$ is known.
 1: Sample each point of $U$ with probability $1/\sqrt{n}$ independently to obtain sample $S$.
 2: Compute a set $A = \mathcal{G}(\mathcal{S})$.
 3: Partition $n$ points into $m$ partitions $U_i$ such that $U_i \bigcup X$ fits on one machine. Place $U_i \bigcup A$ on machines.
 4: On each machine, discard elements within $2OPT$ of the points in $A$. Let $R$ be remaining elements after pruning.
 5: Send all elements of $R$ to one machine, run $\mathcal{G}(\mathcal{R})$ and return the output set.
```

For analysis of the distributed algorithm, we show the following:
(i) Most of the points in U do not pass the tester and are discarded.
(ii) The algorithm is a 2-approximation.

Lemma 3 *$|R| = O(kn^{1/2}\log n)$ w.h.p.*

Proof: If $|A| = k$, then A found in step 2 of algorithm gives a 2-approximation. For the following, we consider the case $|A| < k$.

If $|R| \ge 10k\sqrt{n}\log n$, we say this is a bad event. Note that sample S determines $A(S) := \mathcal{G}(\mathcal{S})$, which in turn determines R.

For each possible realization A' of A, we let $R(A')$ denote the resulting R, that is, $R(A') := \{u \in U \mid d(u,A') > 2OPT\}$. Observe that the bad event with $A = A'$ can happen only when no points in $R(A')$ are included in the sample S since otherwise at least one of those points would have been added to the set A, contradicting to the condition $A = A'$. Therefore,

$$\begin{aligned}
\Pr[|R| \ge 10kn^{1/2}\log n] &\le \Pr\Big[\bigcup_{|A'|<k} \Big(A = A'\Big) \wedge \Big(|R(A')| \ge 10kn^{1/2}\log n\Big) \wedge \Big(R(A') \cap S = \emptyset\Big)\Big] \\
&\le \sum_{|A'|<k} \Pr\Big[\Big(|R(A')| \ge 10kn^{1/2}\log n\Big) \wedge \Big(R(A') \cap S = \emptyset\Big)\Big] \\
&\le n^k \cdot (1 - \frac{1}{n^{1/2}})^{10kn^{1/2}\log n} \le n^k \cdot \exp(-10k\log n) \le n^{-9k}
\end{aligned}$$

The second inequality is an easy consequence of union bound over all possible realizations of A. The third inequality follows from the fact that each point in $R(A')$ is sampled to be in S independently with probability $1/\sqrt{n}$ and because $|A| < k$. □

We now show (ii).

Lemma 4 *The algorithm returns a 2-approximate solution.*

Proof: The key observation is that the remaining points R are exactly the same as those the greedy algorithm has left over after choosing A as centers. Thus, we get a 2-approximation by running $\mathcal{G}$ on R (or equivalently running $\mathcal{G}$ on $S \bigcup R$). □