Coresets

Introduction

Training machine learning models can often be very data intensive. In some cases, expensive computations over the dataset, eg computing the gradients, loss functions, can significantly slow down our progress. Even not in the context of ML, sometimes querying a given dataset for some information can be computationally expensive. Such queries are frequently optimised by using better databases, quicker algorithms, sometimes compression algorithms. The scale to which these improvisations can help us is limited by the sheer magnitude of data.

A solution is to emulate this larger data set using a much smaller set such that solving a problem on this smaller set as its input, provably yields the same result as solving the same problem on the original (full) set, for a given family of problems (models, classifiers, loss functions). This smaller set emulating the behaviour of the full dataset (only for certain class of mathematical queries, of course, this isn't some kind of magical compressing tool to fit in all the information of the larger set into a much smaller one) is called a coreset.

The goal of the coreset is to answer queries about the original dataset as accurately as possible. The simplest coreset is a (possibly weighted) subset of the input data. The advantages
of such subset coresets are:

- preserved sparsity of the input
- interpretability
- coreset may be used (heuristically) for other problems
- less numerical issues that occur when non-exact linear combination of points are used

Coreset constructions are usually tailored to a specific problem at hand. Accurate coresets, are a simple type of coresets that do not introduce any approximation error while compressing the original data, and give accurate solutions.

Definition

A weighted set is a pair $P' = (P, w)$ where P is a set of items called points, and $w : P \to \mathbb{R}$ is a function that maps every $p \in P$ to $w(p) \in R$, called the weight of p.

Def: Query space

Let X be a (possibly infinite) set called query set, $P' = (P, w)$
be a weighted set called the input set, $f : P \times X \to [0, \infty)$ be called a cost function, and loss be a function that assigns a non-negative real number for every real vector.

> The tuple $(P, w, X, f, \text{loss})$ is called a query space.

For every weighted set $C' = (C, u)$ such that $C = \{c_1, \cdots, c_m\}$, and every $x \in X$ we define the overall fitting error of C' to x by

$$f_{\text{loss}}(C', x) := \text{loss}(\{w(c)f(c, x) : c \in C\}) = \text{loss}(w(c_1)f(c_1, x), \cdots, w(c_m)f(c_m, x))$$

- So loss takes as input a vector, C here,(can be understood as a set of scalars) and outputs a real valued loss.
- The query set is the set of possible data points we can be querying about.
- f is a function that outputs a cost based on the input set, and the query we give it. All these make up the query space.

Def: Accurate Coreset

Let $P' = (P, w)$ be a weighted set and $(P, w, X, f, \text{loss})$ be a query space. The weighted set C' is called an accurate coreset for $(P, w, X, f, \text{loss})$ if for every $x \in X$ we have

$$f_{\text{loss}}(P', x) = f_{\text{loss}}(C', x)$$

Thus the loss evaluated at the input set and the coreset are the same by definition. For different settings, different queries etc. different coresets can be constructed.

Accurate Coresets: Examples

1-Center

Suppose that we want to open a shop on our street that will be close to all the residents in the street, and suppose that the street is represented by a linear segment, say the x-axis, while the residents are represented by points on this segment. Since we want to be close to all the potential $n \geq 1$ residents in the street, if we decide to open the shop at some location, our loss will be measured as the distance to the farthest resident. Suppose that tomorrow we will be given few locations to choose from to position our store.

Normally, to compute the distance of the farthest resident from the store, it would take $O(n)$ time, as we go through every resident one by one. Can we do better, how about we

report in $O(1)$ time what this farthest distance is?

Let's frame this problem in the terms of formal variables used earlier.
Our input dataset is $P = \{p_1, \cdots, p_n\}$, the points on the x axis, the coordinates of homes. The cost is between one home and the query store location (say x) is $f(p, x) = |p - x|$, the distance between them. The loss function for the set of inputs and the query location is $\max(|p - x|) : p \in P$, which is simply the ∞ norm of the cost vector.

$$f_{\text{loss}}((P, 1), x) = ||(|p_1 - x|, \cdots, |p_n - x|)||_\infty = \max_{p \in P} |p - x|$$

We need to compute this loss for our coreset C in $O(1)$ time. This can be easily done by observing that for every $x \in X$, the farthest point from x is either the smallest point $p_{\min}$ or the largest point $p_{\max}$ in P.
That is, simply choosing $C = \{p_{\min}, p_{\max}\} \subseteq P$ yields

$$f_{\text{loss}}((P, 1), x) = \max\{|p_{\min} - x|, |p_{\max} - x|\} = f_{\text{loss}}((C, 1), x)$$

Thus comparing the cost for these 2 points only would be sufficient. Of course, there is preprocessing of $O(n)$ to find the leftmost and rightmost points, if the set isn't already sorted, but the subsequent queries are executed in $O(1)$.

Even for the case that $X = \mathbb{R}^d$ and where P is contained on a line ℓ in $\mathbb{R}^d$, we would still have $f_{\text{loss}}((P, 1), x) = f_{\text{loss}}((C, 1), x)$ where x (the query point) can be anywhere in $\mathbb{R}^d$, and C contains the two edge points on ℓ. How? Figure it out, uses the same idea :)

1-Mean Queries

Let us look at a more advanced problem now. Let's say this time, instead of minimising the distance to the farthest point, we are interested in common welfare, and are instead minimising the average distance between points in input space and query location.

To this end, let $P \subseteq \mathbb{R}$ be a set of $|P| = n$ numbers, $X = \mathbb{R}$, $f(p, x) = (p - x)^2$, and $\text{loss}(v) = \frac{1}{n} \sum_i v_i$ for every $v = (v_1, \cdots, v_n) \in \mathbb{R}^n$.
The solution to this is pretty straightforward. The mean of the points minimises this loss. All we need is to store the (coreset) $C = \left\{ \frac{1}{n} \sum_{p \in P} p^2, \frac{1}{n} \sum_{p \in P} p \right\}$ which consists of two numbers. Using a preprocessing of $O(n)$, we can compute the loss for any query location, by defining a new function $h : \mathcal{P}airs(\mathbb{R}) \times \mathbb{R} \to \mathbb{R}$ where

$$h(\{a, b\}, x) = a + x^2 - 2xb$$

Hence, $h(C, x) = f_{\text{loss}}(P, x)$ for every $x \in \mathbb{R}$.
This can be extended to a weighted set $P \subseteq \mathbb{R}^d$ and $w : P \to \mathbb{R}$. In this case too, a pre-

computed coreset consisting of second (the variance), first (the mean) and zeroth moments of p with a new loss function over triplets in $\mathbb{R}^d$.

This coreset, however, is not a subset of the input data set, moreover we had to use a new cost function. Sometimes, for the reasons mentioned in the first paragraph, we need a coreset which is a subset of the input set.

Subset Coreset

For every $p \in P$, let $\hat{p} = (p^T \quad | \quad ||p||^2 \quad |1)^T$ be a corresponding vector in $\mathbb{R}^d + 2$ and $\hat{P} = \{\hat{p} \mid p \in P\}$ be the union of these vectors. Since the weighted mean $\sum_{\hat{p}} w(p) \cdot \hat{p}$ is spanned by $\hat{P}$ (i.e., linear combination of its subset), there is a subset $\hat{C} \subseteq \hat{P}$ of at most $|\hat{C}| = d + 2$ points with a corresponding weight function $\hat{u} : \hat{C} \to \mathbb{R}$ such that

$$\sum_{\hat{p} \in \hat{C}} \hat{u}(\hat{p}) \cdot \hat{p} = \sum_{\hat{p} \in \hat{P}} w(p) \cdot \hat{p}$$

This subset, $\hat{C}$ is obtained by the Gram Schmidt Orthonormalisation procedure on the set of vectors $\hat{P}$. This technique essentially gives a new orthogonal basis to the space spanned by the set.

Let $C = \{p \in P \mid \hat{p} \in \hat{C}\}$, and let $u : C \to \mathbb{R}$ such that $u(p) = \hat{u}(\hat{p})$ for every $p \in C$. We now have that

$$\begin{pmatrix} \sum_{p \in C} u(p)p \\ \sum_{p \in C} u(p)\|p\|^2 \\ \sum_{p \in C} u(p) \end{pmatrix} = \sum_{\hat{p} \in \mathcal{C}} \hat{u}(\hat{p})\hat{p} = \sum_{\hat{p} \in \hat{P}} w(p)\hat{p} = \begin{pmatrix} \sum_{p \in P} w(p)p \\ \sum_{p \in P} w(p)\|p\|^2 \\ \sum_{p \in P} w(p) \end{pmatrix}$$

where the first equality is by the definitions of C and u.
The last equality is by the definition of $\hat{P}$. Therefore, for every $x \in \mathbb{R}^d$, we have that

$$\begin{aligned} f_{\text{loss}}((C, u), x) = \sum_{p \in C} u(p)\|p - x\|^2 &= \sum_{p \in C} u(p)\|p\|^2 + \|x\|^2 \cdot \sum_{p \in C} u(p) - 2x^T \sum_{p \in C} u(p)p \\ &= \sum_{p \in P} w(p)\|p\|^2 + \|x\|^2 \cdot \sum_{p \in P} w(p) - 2x^T \sum_{p \in P} w(p) \\ &= f_{\text{loss}}((P, w), x) \end{aligned}$$

Unlike in the previous case, here the coreset is simply a scaled (weighted) subset of P and the cost function f_{loss} is the same as for the input.