Using Coresets And Sketches To Maintain Accuracy And Improve Run-Time of Least-Mean-Squares Solvers

Shanshan Chen, Zhongxuan Liu, Joseph Gonzalez

6/4/2021

Abstract

In the early 1900s, Constantin Carathéodory and Ernst Steinitz developed and proved Carathéodory's theorem. This theorem states that we can write each point contained in a convex hull of n points in R^d as a convex combination of at most d+1 points. These details suggest that we can maintain a $d^2 + 1$ scaled set of points that we can use to calculate the covariance matrix for a design matrix X and, with some other calculations, solve the LMS equation. While this method avoids updating the linear combinations of all n points and reduces numerical errors, it is not popular for its run time of $O(n^2d^2)$ or $O(nd^3)$.

In this paper, we implement a novel Least-Mean-Squares Solvers from scratch that combines sketches and coresets. This method modifies Carathéodory's theorem to improve the runtime and maintain accuracy. For the application, we show that this solver can boost the performance of existing solvers such as those in the scikit-learn library.

Introduction

In *Fast and Accurate Least-Mean-Squares Solvers*(Maalouf et al.), the researchers claim that they developed a novel method that uses coresets and sketches to improve run-time and numerical issues that LMS solvers, like ordinary least squares, lasso/ridge regression, SVD, and QR decomposition, encounter. The researchers also claim that this method can enhance the performance of these LMS solvers in python's sklearn package up to one hundred times faster. These claims sparked our interest in novel LMS solvers with a mixture of data summarization processes(coresets/sketches method). As a result, we attempt to develop algorithm(s) 5(see appendix A) from scratch and compare the performance outcomes on the coreset data and the non-coreset data.

Methodologies

The Optimization Problem

For any linear solver, the goal is to minimize the sum squared error promptly and, as a result, fit the data accurately. We use a design matrix A(n × d) and a response vector b to optimize the problem. The generic formula for this problem is

$$\min_{x \in X}(||Ax - b||_2^2 + g(x))$$

Where $g(x)$ is a regularization term. Using the covariance matrix A^TA, We can compute the LMS solvers(e.g. linear regression solution is $(A^TA)^{-1}A^Tb$). We use the following LMS solvers throughout this paper.

Ordinary Least Squares

The ordinary least squares regression fits a linear model with coefficients $x = (x_1, ..., x_d)$ to minimize the sum squared error between the observed values and the linearly approximated values. This solver has no regularization term. Linear regression has the following optimization formula:

$$\min_{x\in X}||Ax-b||_2^2$$

In sklearn's linear_model module, the function we use to fit the Ordinary Least Squares regression is LinearRegression().

Ridge Regression

The ridge regression fits a linear model with coefficients $x = (x_1, ..., x_d)$ to minimize the least squares error and the regularization term. This solver's regularization term is the l2-norm and this penalty alleviates unstable coefficients(pulls them towards 0). Ridge regression has the following optimization formula:

$$\min_{x\in X}(||Ax-b||_2^2 + \alpha||x||_2^2)$$

where α is the tuning parameter. Larger alpha values mean more penalty on the model(pulling the coefficients closer to zero). A zero penalty indicates an OLS fit. Ridge always has a solution because $(A^TA + \lambda I)$ exists(biased solution). The Ridge function(RidgeCV) is also in sklearn's linear_model module. RidgeCV fits on the data and tunes α through cross-validation. The function selects α that generates the smallest prediction error.

Lasso Regression

The lasso regression fits a linear model with coefficients $x = (x_1, ..., x_d)$ to minimize the least squares error and the regularization term. This solver's regularization term is the l1-norm and this also alleviates unstable coefficients(if the coefficient is zero, removes variables from the model). Lasso regression has the following optimization formula:

$$\min_{x\in X}(\frac{1}{2n}||Ax-b||_2^2 + \alpha||x||_1)$$

where α is the tuning parameter. Larger alpha values indicate more penalty on the model(pulling the coefficients closer to zero). A zero penalty indicates an OLS fit. The lasso function(LassoCV) is in sklearn's linear_model module and it uses the Least Angle Regression algorithm. LassoCV fits on the data and tunes α through cross-validation.

Elastic Net Regression

The elastic net regression fits a linear model with coefficients $x = (x_1, ..., x_d)$ to minimize the least squares error and two regularization terms. This solver's regularization terms are the l1-norm and l2-norm(ridge and lasso mix). Elastic net regression also overcomes Lasso's shortcomings, which is it chooses only one of two highly correlated variables(elastic net will most likely pick both). Elastic Net Regression has the following optimization formula:

$$\min_{x\in X}(\frac{1}{2n}||Ax-b||_2^2 + \rho\alpha||x||_2^2 + \frac{(1-\rho)}{2}\alpha||x||_1)$$

Where α and ρ are the hyper-parameters. ElasticNetCV fits on the data and tunes both α and ρ through cross-validation.

Issues

Another motivation for the researchers to pursue the coreset method comes from the numerical and run issues LMS solvers face. This originates from the two ways we can compute the covariance matrix:

1. $A^TA = \sum_{i=1}^n a_i a_i^T$
2. Using SVD or QR decomposition, factorize A. SVD: $A^TA = VD^2V^T$, QR: $A^TA = R^TR$

Numerical issues: With additional observations over time(streaming data), the numerical error increases in method 1(increases significantly with 32-bit float type). A problem the second method faces is that the SVD is not a subset of A and the result may not be invertible(due to the numerical errors). Lastly, the small numerical errors may prevent us from using the Cholesky decomposition to compute A^TA(needs a positive definite matrix).

Run Time Issues: The SVD and QR methods are time-consuming and almost impossible to compute for streaming data.

Caratheodory's Theorem

The researchers' method stems from Caratheodory's Theorem. This theorem states that we can write each point in a convex hull of n points($\mathbb{R}^d$) as a convex combination of a subset with at most d+1 points(*Caratheodory set*). This further insinuates that we can maintain a weighted/scaled set of $d^2 + 1$ points(observations), which has a covariance matrix the same as the input matrix A(The mean of n matrices is $\frac{1}{n}\sum_i a_i a_i^T$ and is in the convex hull of corresponding points in $\mathbb{R}^{d^2}$). The small subset of points significantly reduces numerical errors. However, this theorem is not practical for LMS solvers because it takes $O(n^2d^2)$ or $O(nd^3)$ time with $O(n)$ calls to an LMS solver. The researchers wanted to adjust this theorem to be more applicable for LMS solvers.

Coresets and Sketches

An approach to solve the LMS equation with a data summarization algorithm is to compute a small matrix S, whose covariance matrix S^TS approximates the covariance matrix A^TA.

Definition 1: Coreset

A coreset is a matrix C with weighted/scaled subset of rows from the n rows of an input matrix A.

Definition 2: Sketch

A sketch is a matrix S where each row is a linear combination of a few rows or all rows in A(S = WA, W $\in \mathbb{R}^{s\times n}$).

Other Coresets and Sketches(not used in this project) frequently yield a $(1 + \epsilon)$ multiplicative approximations for $|Ax||_2^2$ by $||Sx||_2^2$ where the matrix S is $(d/\epsilon)^{O(1)}$ rows and x can be any vector of S or A. A (1 + ϵ)-approximation to $||Ax||_2^2$ by $||Sx||_2^2$ also does not guarantee an approximation to the actual entries or eigenvectors of A by S.

LMS Coresets Improvements

In *Fast and Accurate Least-Mean-Squares Solvers*, the authors list many advantages for using their coreset method. These advantages include:

- Their algorithms produce coresets that are accurate($\epsilon = 0$) and can compute A^TA with a scaled subset of rows from the matrix A.
- They also state that the coreset supports streams of input rows(sub-linear in their size) and dynamic data.
- Their coreset has "merge-and-reduce" properties that support handling big data.
- The weighted subsets preserve the desired statistics accurately(compression does not increase error).
- Their algorithm produces a Caratheodory set of n input points in $O(nd + d^4 logn)$ run time.
- The coreset preserves the sparsity of the input rows(reduces theoretical run time) and improves the numerical stability of existing algorithms.
- We can use the same coreset parameter tuning over a large set of candidates(reduces the running time).

LMS - Coreset Process Steps

The reader may find the full algorithm in appendix 1 in the original paper(pages 12, 8, and 7). The following is the researchers' meta-algorithm for the LMS coreset algorithm:

Input: A set P of n items, an integer k $\in \{1, \cdots ,n\}$(n is highest numerical accuracy and longest run time), and a pair of coreset and sketch blueprints.

Step I: Generate k balanced clusters $P_1, \cdots, P_k$ of the input set P. In a best-case scenario, we want to generate partitions that minimize the sum of loss.

Step II: For each cluster in $\{P_1, \cdots, P_k\}$, we compute sketches $\{S_1, \cdots, S_k\}$ using the input sketch scheme. The researchers warn that "this step does not return a subset of P as desired, and is usually numerically less stable."

Step III: Generate the coreset B from S = $S_1 \cup \cdots \cup S_k$ using the input coreset scheme(B is not a subset or coreset of P).

Step IV: For the corresponding selected sketches in step III, we compute the union C of clusters in $P_1, \cdots, P_k$(C = $\cup_{S_i \in B} P_i$). C is the coreset.

Step V: Until we reach an adequately small coreset, we repeatedly compute a coreset for C(reduces run time without using a k that is too small.

Output: A coreset with a construction time is quicker than the construction time of the given coreset scheme.

Data Sets

We applied the LMS-Coreset algorithm on data sets 1, 2, and 3 below. Then, we fit the LMS solvers on the coreset and the regular data set, and compared their performance. The data set descriptions are below:

1. **Car Purchasing Data:** The car purchasing data contains customer information. There are 500 observations and 5 predictors(d =5). We use customer gender(boolean), age(Integer), annual salary(float), credit card debt(float), and net worth(float) to predict car purchase amount(float).

2. **Electric Motor Temperature Data:** This data contains electric car information. There are 1,330,816 observations and 11 predictor variables. We use voltage q-component measurement, stator winding temperature, voltage d-component measurement, stator tooth temperature, motor speed, current d-component measurement, current q-component measurement, and stator yoke temperature to predict permanent magnet temperature(all float data types).

3. **Austin Housing Data:** This data contains information on Austin house listings. The data set contains 15,171 observations and 35 predictors(d = 35). Some predictors are property tax rate(float), average school distance(float), number of bedrooms(int), number of bathrooms(int), and cooling(boolean). We use the predictors to predict house prices(float).

Results

Conclusion

Appendix 1: LMS Coreset Algorithm

```
Algorithm 5 LMS-CORESET$(A, b, m, k)$
Input:  A matrix $A \in \mathbb{R}^{n \times d}$, a vector $b \in \mathbb{R}^n$, a number (integer) $m$ of cross-validation folds,
        and an integer $k \in \{1, \cdots, n\}$ that denotes accuracy/speed trade-off.
Output: A matrix $C \in \mathbb{R}^{O(md^2) \times d}$ whose rows are scaled rows from $A$, and a vector $y \in \mathbb{R}^d$.
1 $A' := (A \mid b)$      // A matrix $A' \in \mathbb{R}^{n \times (d+1)}$
2 $\{A'_1, \cdots, A'_m\}$ := a partition of the rows of $A'$ into $m$ matrices, each of size $(\frac{n}{m}) \times (d+1)$
3 for every $i \in \{1, \cdots, m\}$ do
4 |   $S_i :=$ CARATHEODORY-MATRIX$(A'_i, k)$    // see Algorithm 2
5 $S := (S_1^T | \cdots | S_m^T)^T$ // concatenation of the $m$ matrices into a single matrix of
      $m(d+1)^2 + m$ rows and $d+1$ columns
6 $C :=$ the first $d$ columns of $S$
7 $y :=$ the last column of $S$
8 return $(C, y)$
```

Figure 1: Alt text

```
Algorithm 2 CARATHEODORY-MATRIX$(A, k)$; see Theorem 4
Input  : A matrix $A = (a_1 \mid \cdots \mid a_n)^T \in \mathbb{R}^{n \times d}$, and an integer $k \in \{1, \cdots, n\}$ for numerical
         accuracy/speed trade-off.
Output: A matrix $S \in \mathbb{R}^{(d^2+1) \times d}$ whose rows are scaled rows from $A$, and $A^T A = S^T S$.
1 for every $i \in \{1 \cdots, n\}$ do
2 |   Set $p_i \in \mathbb{R}^{(d^2)}$ as the concatenation of the $d^2$ entries of $a_i a_i^T \in \mathbb{R}^{d \times d}$.
  |       // The order of entries may be arbitrary but the same for all points.
3 |   $u(p_i) := 1/n$
4 $P := \{p_i \mid i \in \{1, \cdots, n\}\}$        // $P$ is a set of $n$ vectors in $\mathbb{R}^{(d^2)}$.
5 $(C, w) :=$ FAST-CARATHEODORY-SET$(P, u, k)$ // $C \subseteq P$ and $|C| = d^2 + 1$ by Theorem 3
6 $S :=$ a $(d^2 + 1) \times d$ matrix whose $i$th row is $\sqrt{n \cdot w(p_i)} \cdot a_i^T$ for every $p_i \in C$.
7 return $S$
```

Figure 2: Alt text

Appendix :

Appendix :

Algorithm 1 FAST-CARATHEODORY-SET(P, u, k); see Theorem 3

```
Input : A set $P$ of $n$ points in $\mathbb{R}^d$, a (weight) function $u : P \to [0, \infty)$ such that
        $\sum_{p \in P} u(p) = 1$, and an integer (number of clusters) $k \in \{1, \cdots, n\}$ for the nu-
        merical accuracy/speed trade-off.
Output: A Caratheodory set of $(P, u)$; see Definition 1.
$P := P \setminus \{p \in P \mid u(p) = 0\}$.      // Remove all points with zero weight.
if $|P| \leq d + 1$ then
    return $(P, u)$      // $|P|$ is already small
$\{P_1, \cdots, P_k\}$ := a partition of $P$ into $k$ disjoint subsets (clusters), each contains at most
    $\lceil n/k \rceil$ points.
for every $i \in \{1, \cdots, k\}$ do
    $\mu_i := \frac{1}{\sum_{q \in P_i} u(q)} \cdot \sum_{p \in P_i} u(p) \cdot p$      // the weighted mean of $P_i$
    $u'(\mu_i) := \sum_{p \in P_i} u(p)$      // The weight of the $i$th cluster.
$(\tilde{\mu}, \tilde{w}) :=$ CARATHEODORY$(\{\mu_1, \cdots, \mu_k\}, u')$
    // see Algorithm 16 in the Appendix.
$C := \bigcup_{\mu_i \in \tilde{\mu}} P_i$
    // $C$ is the union over all clusters $P_i \subseteq P$ whose representative $\mu_i$ was
    chosen for $\tilde{\mu}$.
for every $\mu_i \in \tilde{\mu}$ and $p \in P_i$ do
    $w(p) := \frac{\tilde{w}(\mu_i) u(p)}{\sum_{q \in P_i} u(q)}$      // assign weight for each point in $C$
$(C, w) :=$ FAST-CARATHEODORY-SET$(C, w, k)$      // recursive call
return $(C, w)$
```

Figure 3: Alt text