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

Shanshan Chen, Zhongxuan Liu, Joseph Gonzalez

6/9/2021

Abstract

In the early 1900s, Constantin Carathodory and Ernst Steinitz developed and proved Carathodory'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 Carathodory'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.

Contents

1 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 scikit-learn 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 5 (see appendix A) from scratch and compare the performance outcomes on the coreset data and the non-coreset data.

2 Methodologies

2.1 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 $\times$ 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 project.

2.1.1 Ordinary Least Squares

The ordinary least squares regression fits a linear model with coefficients $x = (x_1, \ldots, x_d)$ 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 scikit-learn's linear_model module, the function we use to fit the Ordinary Least Squares regression is LinearRegression().

2.1.2 Ridge Regression

The ridge regression fits a linear model with coefficients $x = (x_1, \ldots, x_d)$ to minimize the least squares error and the regularization term. This solver's regularization term is the l_2-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 the inverse of $(A^TA + \lambda I)$ exists (a biased solution). The Ridge function(RidgeCV()) is also in scikit-learn'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.

2.1.3 Lasso Regression

The lasso regression fits a linear model with coefficients $x = (x_1, \ldots, x_d)$ 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:

$$\min_{x \in X} \frac{1}{2n} \|Axb\|_2^2 + \alpha \|x\|_1$$

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 scikit-learn's linear_model module and it uses the Least Angle Regression algorithm. LassoCV() fits on the data and tunes α through cross-validation.

2.1.4 Elastic Net Regression

The elastic net regression fits a linear model with coefficients $x = (x_1, \ldots, x_d)$ 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() in scikit-learn's linear_model module fits on the data and tunes both α and ρ through cross-validation.

2.1.5 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.

2.2 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.

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

Definition 2: 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 $(1+\epsilon)$-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.

2.4 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 $O(nd + d^4\log(n))$ 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).

2.5 LMS - Coreset Process Steps

The reader may find the full algorithm in appendix 1 and in the original paper(pages 12, 8, 7, and 25). The following is the researchers' meta-algorithm for the LMS coreset algorithm:

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

Step I: Generate k balanced clusters $P_1, , P_k$ 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 $\{P_1, ..., P_k\}$, we compute sketches $\{S_1, ..., S_k\}$ 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 = $S_1 \cup ... \cup S_k$ 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 $P_1, ..., P_k$(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.

3 Experimental Results

3.1 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). Since many predictors are categorical variables, we removed variables whose number of unique value is less than 10. The final number of predictors is $d = 15$.

3.2 Results

We applied the LMS-Coreset algorithm on the LMS solvers and compared the running time with the corresponding LMS solvers from the standard scikit-learn library. To see the accuracy, we also compared the difference of R squared values and the sum of the absolute difference of the coefficients. The experiments were conducted on a fifth generation of Surface Pro with an Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz and 16GB RAM. The results are shown in Table 1, 2, 3.

	sklearn Run-Time	LMS-Coreset Run-Time	Diff of R2	Diff of coef
OLS	0.000995s	0.048865s	0.0	4.274e-05
Lasso	0.073799s	0.111675s	0.0	6.245e-17
Ridge	1.051699s	0.897190s	8.697e-07	2.387
Elastic Net	0.079784s	0.146608s	0.0	6.939e-18

Table 1: Results of fitting Car Purchasing Data

For the Car Purchasing Data, since the n and d are relatively small in the Car Purchasing Data, the run time of sklearn and LMS-Coreset algorithms are very close. A notable result is the LMS-Coreset based on Ridge Regression is faster than its competitor. The R Squared and the coefficients are very close too, indicating the result is accurate.

	sklearn Run-Time	LMS-Coreset Run-Time	Diff of R2	Diff of coef
OLS	0.396933s	6.733950s	9.153e-13	2.236e-05
Lasso	17.659321s	14.954995s	6.725e-07	3.138e-06
Ridge	120.666517s	66.743091s	9.373e-13	2.236e-05
Elastic Net	12.226942s	12.412012s	6.984e-07	7.292e-07

Table 2: Results of fitting Motor Data

For the Motor Data, it has a million of observations($n = 1,330,816$) and an intermediate number of features($d = 11$). The effect for Lasso and Ridge Regression on the LMS-Coreset are surprisingly distinguishable. In fact, it boosted the speed of Ridge Regression by almost 1 time. The run times for Elastic Net are close and the run time of scikit-learn for OLS is much faster than the LMS-Coreset. This suggests that the LMS-Coreset is not able to boost scikit-learn's OLS. The result is very accurate since the differences of the R Squared values and the coefficients are both small.

	sklearn Run-Time	LMS-Coreset Run-Time	Diff of R2	Diff of coef
OLS	0.006980s	26.381473s	5.920e-13	570.664
Lasso	0.220410s	25.963129s	3.579e-05	60.5205
Ridge	3.297203s	27.589348s	9.373e-13	560.811
Elastic Net	0.214425s	26.444494s	2.111e-07	0.744

Table 3: Results of fitting Housing Data

For the Housing Data, there are many predictor variables($d = 35$) and many of them are categorical. We removed the categorical variables to run our algorithm(final number of predictors is $d = 15$) and the data is also standardized. The result on the Housing data does not meet our expectations. All LMS-Coreset algorithms need more than twenty seconds to finish and all the sklearn functions are less than four seconds. The results are not accurate too. While the differences of R squared values are small, the differences of coefficients are very large(except for Elastic Net).

4 Conclusion and Discussion

We implemented a novel algorithm that uses Coresets And Sketches to solve linear regression problems. There are claims that the algorithm can boost the LMS solvers in the scikit-learn library, which means that it maintains accuracy and is faster than using the LMS solvers directly. The LMS-Coreset algorithm that we implemented shows a pretty good effect on the Car Purchasing data and the Motor data, while it failed to boost the performance on the Housing data. The reason might be the algorithm still needs to be optimized for a large number of features. Also, the functions in the scikit-learn library are compiled in C/C++ at the bottom layer, which is a hundred times faster than Python itself. In the future, the Sparse Caratheodory algorithm in the original paper should be considered to reduce the running time's polynomial dependency on d. We may also use the module JAX, which is Autograd and XLA. These modules are combined for high-performance machine learning research and, therefore, could increase the speed of our original algorithm.

Appendix: LMS Coreset Algorithms

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 $y \in \mathbb{R}^d$.

$A' := (A \mid b)$ `// A matrix` $A' \in \mathbb{R}^{n \times (d+1)}$

$\{A'_1, \cdots, A'_m\} :=$ a partition of the rows of A' into m matrices, each of size $(\frac{n}{m}) \times (d+1)$

for *every* $i \in \{1, \cdots, m\}$ **do**

  $S_i :=$ Caratheodory-Matrix(A'_i, k) `// see Algorithm 2`

$S := (S_1^T | \cdots | S_m^T)^T$ `// concatenation of the` m `matrices into a single matrix of` $m(d+1)^2 + m$ `rows and` $d+1$ `columns`

$C :=$ the first d columns of S

$y :=$ the last column of S

return (C, y)

Figure 1: LMS Coreset Algorithm

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 $A^T A = S^T S$.

for *every* $i \in \{1 \cdots, n\}$ **do**

  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.`

  $u(p_i) := 1/n$

$P := \{p_i \mid i \in \{1, \cdots, n\}\}$ `//` P `is a set of` n `vectors in` $\mathbb{R}^{(d^2)}$.

$(C, w) :=$ Fast-Caratheodory-Set(P, u, k) `//` $C \subseteq P$ `and` $|C| = d^2 + 1$ `by Theorem 3`

$S :=$ a $(d^2+1) \times d$ matrix whose ith row is $\sqrt{n \cdot w(p_i)} \cdot a_i^T$ for every $p_i \in C$.

return S

Figure 2: Caratheodory-Matrix Algorithm

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 numerical 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` μ_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: Fast-Caratheodory-Set Algorithm

Algorithm 16 CARATHEODORY(P, u)

Input : A weighted set (P, u) of n points in $\mathbb{R}^d$.

Output: A Caratheodory set (S, w) for (P, u) in $O(n^2 d^2)$ time.

if $n \leq d + 1$ **then**

  return (P, u)

Identify $P = \{p_1, \cdots, p_n\}$

for *every* $i \in \{2, \cdots, n\}$ **do**

  $a_i := p_i - p_1$

$A := (a_2 \mid \cdots \mid a_n)$ // $A \in \mathbb{R}^{d \times (n-1)}$

Compute $v = (v_2, \cdots, v_n)^T \neq 0$ such that $Av = 0$.

$v_1 := -\sum_{i=2}^{n} v_i$

$\alpha := \min \left\{ \frac{u_i}{v_i} \mid i \in \{1, \cdots, n\} \text{ and } v_i > 0 \right\}$

$w_i := u_i - \alpha v_i$ for every $i \in \{1, \cdots, n\}$.

$S := \{p_i \mid w_i > 0 \text{ and } i \in \{1, \cdots, n\}\}$

if $|S| > d + 1$ **then**

  $(S, w) :=$ CARATHEODORY(S, w)

return (S, w)

Figure 4: Caratheodory Algorithm

Reference

Main Article:

Alaa Maalouf, Ibrahim Jubran, Dan Feldman. Fast and Accurate Least-Mean-Squares Solvers. *arXiv:1906.04705v2*, 2019.

Data Sets:

1. Sharma, Dev (2021, May). Car Purchasing Data, Version 1(United States). Retrieved May 10, 2021 from https://www.kaggle.com/dev0914sharma/car-purchasing-model.

2. Pierce, Eric (2021, March). Austin Housing Data, Version 4(Austin, Texas). Retrieved May 10, 2021 from https://www.kaggle.com/ericpierce/austinhousingprices.

3. Kirchgssner, Wilhelm (2019). Electric Motor Temperature, Version 4(Paderborn University). Retrieved May 10, 2021 from https://www.kaggle.com/wkirgsn/electric-motor-temperature.

Code:

Our code has been uploaded to GitHub: https://github.com/Joseph-Gonzalez70/Investigating-Novel-LMS-Solver-