CSCI 6971/4971: Homework 5

Assigned Monday March 19 2017. Due at beginning of class Monday March 26 2017.

Submission Instructions: *Your submission should stand alone: I should be able to read your notebook and understand what question each output is answering without referring back to the assignment.*

1. Use Python+NumPy+SciPy+(Sklearn where convenient) and document your work using a Jupyter notebook. This is a common working environment in industry, and will allow you to interleave code, plots, and typeset written information, and submit your entire assignment as one file. Put your name at the top of the notebook and email it to me as `hw1_<FirstnameLastname>.ipynb` at `gittea@rpi.edu`.
2. I should be able to run your code on my system with minimal modifications. To ensure this, isolate any input locations, parameters that you manually set, etc., that I may need to change to a clearly labeled cell at the top of the notebook, and document these settings.
3. Label all plots and tables appropriately and comment your code liberally and meaningfully.

1. [100 points (Approximating Leverage Scores)] In class we saw an approach to leverage score approximation. This assignment recaps that method and applies it to linear regression.

   Let $\tilde{\mathbf{X}} = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$ be a matrix in $\mathbb{R}^{n\times(d+1)}$, where we assume $n \geq (d+1)$. Strictly speaking, when referring to leverage scores, we should refer to the rank with respect to which the leverage scores are defined, but in this assignment, we'll refer to the leverage scores of $\tilde{\mathbf{X}}$ with respect to rank $d+1$ as simply 'the leverage scores' of $\tilde{\mathbf{X}}$.

   The leverage scores of $\tilde{\mathbf{X}}$ show up regularly in the analysis of randomized matrix algorithms for tall and skinny matrices[1]. As one example, when an overdetermined linear system in $\mathbb{R}^d$ is sketched by selecting rows independently at random proportionally to their leverage scores, the number of constraints can be reduced to $\mathrm{O}(\varepsilon^{-2} d \ln(d/\delta)$ while guaranteeing that the solution of the sketched linear system is ε-approximately accurate with probability at least $1-\delta$.

   The drawback of leverage scores is that their computation seems to require the QR decomposition of the matrix, which requires $O(nd^2)$ flops[2]. However, it can be shown that in many applications, including regression, it suffices to use approximations $\{\tilde{\ell}_i\}$ of the leverage scores that satisfy

   $$\ell_i \leq \tilde{\ell}_i \leq \beta \ell_i, \quad \text{for } i = 1, \ldots, n, \tag{1}$$

   for a constant $\beta \geq 1$. In the case of linear regression, one can show that sketching by selecting $c = \mathrm{O}(\beta\varepsilon^{-2} d \ln(d/\delta))$ rows with probability proportional to these approximate leverage scores gives the same guarantees as sketching by selecting $c = \mathrm{O}(\varepsilon^{-2} d \ln(d/\delta))$ rows according to the exact leverage scores.

   Recall that if $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is the reduced SVD, then the leverage scores of $\tilde{\mathbf{X}}$ satisfy

   $$\ell_i = \|\mathrm{e}_i^T \mathbf{U}\|_2^2 = \|\mathrm{e}_i^T \mathbf{U}\mathbf{U}^T\|_2^2,$$

   where the last equality holds because of the orthonormality of the columns of $\mathbf{U}$. Convince yourself, using the reduced SVD, that $\mathbf{U}\mathbf{U}^T = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger$. It follows that

   $$\ell_i = \|\mathrm{e}_i^T \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger\|_2^2.$$

[1] The leverage scores with respect to other ranks show up in the analysis of low-rank approximation algorithms.

[2] Both the QR and SVD decompositions cost $O(nd^2)$, but the QR decomposition has a much lower constant factor, so is preferrable for this application, since *any* orthonormal basis for the column space of $\tilde{\mathbf{X}}$ will suffice for computing the leverage scores.

That is, the ith leverage score is the squared norm of the ith row of $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger$. This is an inconvenient formula, as it costs QR time to construct $\tilde{\mathbf{X}}^\dagger$.

But note that $\tilde{\mathbf{X}}$ has rank at most $d+1$, so the projection matrix $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger = \mathbf{U}\mathbf{U}^T$ is an $n \times n$ matrix of at most rank $d+1$. Thus we can hope to approximate the leverage scores with the squared norms of a convenient low-rank approximation of $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger$. Indeed, we can show[3] that

$$\tilde{\ell}_i = \left\| \mathrm{e}_i^T \tilde{\mathbf{X}} \big(\mathbf{S}^T \tilde{\mathbf{X}}\big)^\dagger \mathbf{\Pi}^T \right\|_2^2 \tag{2}$$

satisfy condition (1) above with $\beta = (1+\varepsilon)$, assuming the matrices $\mathbf{S} \in \mathbb{R}^{n \times c_1}$ and $\mathbf{\Pi} \in \mathbb{R}^{c_2 \times c_1}$ are chosen carefully[4]. In the following discussion, for simplicity, we leave out the dependence on the failure probability δ and the accuracy ε, and use $\tilde{O}$ to denote that we have ignored some multiplicative factors that grow polylogarithmically in the input dimensions.

First, we choose $\mathbf{S}$ so it is an OSE for $\tilde{\mathbf{X}}$. This ensures that $\tilde{\mathbf{X}}\big(\mathbf{S}\tilde{\mathbf{X}}\big)^\dagger$ is close enough to $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\dagger$ that their row norms are within a small multiplicative factor. For speed in computing $\mathbf{S}\tilde{\mathbf{X}}$ we use a Subsampled DCT-II Transform[5]. We could stop at this point and compute the row norms of $\tilde{\mathbf{X}}\big(\mathbf{S}^T\tilde{\mathbf{X}}\big)^\dagger$, but this is an n-by-c_1 matrix, where $c_1 > (d+1)$, so costs $\mathrm{O}(ndc_1) = \mathrm{O}(nd^2)$. This is asymptotically no better than just using the QR decomposition on $\tilde{\mathbf{X}}$!

Thus $\mathbf{\Pi}$ plays an essential role. Note that we are attempting to compute the norms of n points (the rows of $\tilde{\mathbf{X}}\big(\mathbf{S}^T\tilde{\mathbf{X}}\big)^\dagger$); we know that a JLT will allow us to reduce the dimensionality of the points to $\mathrm{O}(\ln(n))$ while preserving their norms to within a small multiplicative factor. Thus the second matrix $\mathbf{\Pi}$ is choosen to be a JLT matrix, with $c_2 = \mathrm{O}(\ln(n))$. Now we exploit the associativity of matrix multiplication to compute the approximations (2) in an order that requires $o(nd^2)$ flops.

Overall, the cost of computing all the $\tilde{\ell}_i$ according to (2) breaks down as:

- $\mathrm{O}(dn\ln(c_1))$ to compute $\mathbf{S}^T\tilde{\mathbf{X}}$, which is a c_1-by-d matrix, using a subsampled randomized Hadmard transform,
- $\mathrm{O}(c_1 d^2)$ to form the pseudoinverse[6] of $\mathbf{S}^T\tilde{\mathbf{X}}$, which is a d-by-c_1 matrix,
- $\mathrm{O}(dc_1c_2)$ to compute $\big(\mathbf{S}\tilde{\mathbf{X}}\big)^\dagger\mathbf{\Pi}^T$, which is a d-by-c_2 matrix,
- $\mathrm{O}(ndc_2)$ to compute $\tilde{\mathbf{X}}\big(\mathbf{S}\tilde{\mathbf{X}}\big)^\dagger\mathbf{\Pi}^T$, which is a n-by-c_2 matrix, and take its row norms.

We take $c_1 = \tilde{\mathrm{O}}(d\ln(nd))$ so that $\mathbf{S}$, which is an SRHT matrix, is an OSE, and $c_2 = \mathrm{O}(\ln(n))$ so that $\mathbf{\Pi}$ is a JLT matrix for n points. Thus, the final cost is

$$\begin{aligned}\mathrm{O}(dn\ln(c_1) + c_1d^2 + dc_1c_2 + ndc_2) &= \tilde{\mathrm{O}}(dn\ln d + d^3\ln(nd) + d^2\ln(nd)\ln(n) + nd\ln(n)) \\ &= \tilde{O}((dn + d^3)\ln n),\end{aligned}$$

which is smaller than the $O(nd^2)$ approach of naively computing the leverage scores when d is sufficiently small compared to n. As an example, consider a regression problem where $n = 100K$ and $d = 90$: for these values, we could expect to see a speed-up of around

$$\frac{T_{\text{exact levscores}}}{T_{\text{approximate levscores}}} \approx \frac{nd^2}{(dn + d^3)\ln n} > 8035$$

[3]This algorithm is taken from "Fast approximation of matrix coherence and statistical leverage." P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, JMLR, 2012.

[4]The algorithm technically returns approxmations satisfying $(1-\varepsilon)\ell_i \leq \tilde{\ell}_i \leq (1+\varepsilon)\ell_i$ with high probability, but by scaling and assuming ε is small enough, this gives a guarantee of the form (1) with $\beta = (1+3\varepsilon)$.

[5]A convenient substitute for the SRHT, since the DCT-II transform exists for all dimensions, while the Hadamard matrix does not.

[6]The pseudoinverse can be computed with negligible additional cost once the SVD is known.

in obtaining useful approximations of the leverage scores. Of course, for various reasons — machine architecture, inefficient coding, cache effects, etc. —, one should be much more conservative in their expectations for the actual speed-ups that will be observed empirically.

We will evaluate the performance of this leverage score approximation algorithm on the year prediction dataset. Obtain the year prediction dataset from the UCI Machine Learning repository (`https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD`). Caveat: this is a >200MB download that unzips to an approximate 500MB text file.

Write a script that does the following:

- Following the dataset description in the repository, load the first $131072 = 2^{17}$ training examples from the text file into the rows of a matrix $\mathbf{X}_{\text{train}}$, and the corresponding years into a vector $\mathbf{y}_{\text{train}}$.
- Following the dataset description in the repository, load the first 20K testing examples from the text file into the rows of a matrix $\mathbf{X}_{\text{test}}$, and the corresponding years into a vector $\mathbf{y}_{\text{test}}$.
- Write a function that takes a matrix $\mathbf{X}$ and computes its leverage scores exactly using the QR decomposition.
- Write a function that takes a matrix $\mathbf{X}$ and sketching dimensions c_1 and c_2 and approximates the leverage scores of $\mathbf{X}$ according to the algorithm described above. Use `scipy.fftpack.dct` to compute the *orthonormal* DCT-II.
- Compute the leverage scores of the rows of $\begin{bmatrix}\mathbf{X}_{\text{train}} & \mathbf{y}_{\text{train}}\end{bmatrix}$.
- For $c_1 \in \{500, 1000, 1500, 2000\}$ and $c_2 \in \{5, 10, 20, 40\}$[7], approximate the leverage scores of the rows of $\begin{bmatrix}\mathbf{X}_{\text{train}} & \mathbf{y}_{\text{train}}\end{bmatrix}$. Compute the minimal value of β for each setting of c_1 and c_2 that satisfies (1). Repeat this experiment 10 times, and average the corresponding values of β. Make a heat plot showing β as a function of c_1 and c_2.
- Write a sketching function which takes a matrix $\mathbf{X}$, a sketching dimension c, and a vector of probabilities as input, and returns $\mathbf{S}^T\mathbf{X}$, where $\mathbf{S}$ corresponds to row selection according to the given probabilities. Do not forget to scale the selected rows appropriately by their probability of selection.
- For each value of c in d, $\lceil d\ln(d)\rceil$, and $10d$, use your sketching routines and the QR algorithm to solve the sketched linear system $\hat{\boldsymbol{\beta}} = \operatorname{argmin}_{\boldsymbol{\beta}} \|\mathbf{S}^T\mathbf{X}_{\text{train}}\boldsymbol{\beta} - \mathbf{S}^T\mathbf{y}_{\text{train}}\|_2^2$ in ten independent trials using the exact leverage scores and the approximate leverage scores with $c_1 = 1500$ and $c_2 = 12$. Record the relative errors $\|\mathbf{X}_{\text{test}}\hat{\boldsymbol{\beta}} - \mathbf{y}_{\text{test}}\|_2/\|\mathbf{X}_{\text{test}}\boldsymbol{\beta}^\star - \mathbf{y}_{\text{test}}\|_2$ and the runtimes of these two algorithms, including the time to calculate or approximate the leverage scores.
- Produce a scatter plot comparing the average runtime to the average relative error for these two algorithms and the three different choices of c. Label or color the different algorithms and choices of c clearly.
- Comment on any noteworthy observations in the scatter and heat plots.

[7]For reference, $n = 2^{17}$ and $d = 90$, so $\lceil d\ln(d)\rceil = 405$, $\lceil d\ln(nd)\rceil = 1466$ and $\lceil \ln(n)\rceil = 12$