Exercise 1

Bowen Hua

September 11, 2017

1 Linear Regression

1.1 (A)

Matrix form:

$$\hat{\beta} = \arg\min_{\beta \in \mathcal{R}^P} \frac{1}{2}(y - X\beta)^T W (y - X\beta).$$

Since W is symmetric and positive semidefinite, this is a convex optimization problem. Its first-order optimality condition is necessary and sufficient:

$$X^T W(y - X\beta) = \mathbf{0}.$$

1.2 (B)

1.2.1 Method 1: direct inversion

Described in the problem statement:

$$\beta = (X^T W X)^{-1} X^T W y.$$

To exploit sparsity of W, we use broadcasting in Python instead of matrix multiplication to compute operations w.r.t. W matrix. This method has a complexity of $O(np^2)$.

1.2.2 Method 2: pseudoinverse

Since the weight matrix W is diagonal, we can define $\sqrt{W}$ to be a diagonal matrix where the i-th diagonal element equals $\sqrt{w_i}$. Now we can re-write the optimality condition:

$$\beta = [(\sqrt{W}X)^T(\sqrt{W}X)]^{-1}(\sqrt{W}X)^T y,$$

which we re-write as

$$\beta = (\sqrt{W}X)^{\dagger}\sqrt{W}y,$$

where $(\sqrt{W}X)^{\dagger}$ is the pseudoinverse of $\sqrt{W}X$.

We first "preprocess" the feature matrix X and y by multiplying $\sqrt{W}$ on the left.

We then compute the pseudoinverse through computing SVD of $\sqrt{W}X$. This method is numerically more stable than the direct inverse method.

Pseudocode for pseudoinverse of matrix A:

$(U, \Sigma, V) = \texttt{svd}(A)$
for Σ_{ii} in Σ: `#traverse through the diagonal elements`
  if $\Sigma_{ii} \neq 0$:
    $\Sigma_{ii} = 1/\Sigma_{ii}$
return $V^T \Sigma U^T$

In the Python code, we call `numpy.linalg.pinv` to perform the pseudoinverse. This method also has a complexity of $O(np^2)$.

In addition, this is the implementation of `scikit-learn`.

1.2.3 Method 3: Cholesky decomposition

We can use Cholesky decomposition on the matrix $X^T W X$. Now we have $LL^T \beta = D$. Then we can obtain β by solving two linear systems.

This method also has a complexity of $O(np^2)$.

Pseudocode for Cholesky-decomposition-based method:

Let $C = X^T W X$, $d = X^T W y$.
Compute Cholesky decomposition $C = LL^T$.
Solve for α in $L\alpha = d$.
Solve for β in $L^T \beta = \alpha$.

1.3 (C)

I coded the three methods in Python, using the `numpy` package. The codes can be find in my GitHub.

The results are summarized here:

Table 1: CPU Times (s) for Three Methods of Weighted Least Squares

(n, p)	Method 1	Method 2	Method 3
$(2000, 50)$	0.001	0.007	0.001
$(1000, 1000)$	0.122	0.346	0.064
$(20000, 50)$	0.012	0.034	0.005
$(50000, 50)$	0.028	0.118	0.013
$(5000, 5000)$	6.402	37.00	4.462

We can see that:

- Method 3 (Choleskey) consistently performs better than Method 1 (Direct Inverse), which is faster than Method 2 (pseudoinverse).
- When X is close to a square matrix, the performance of Method 3 is way worse than the other two methods.

1.4 (D)

We use `scipy.sparse` in Python as our tool for sparse matrix operations. In particular, we use `scipy.sparse.linalg.lsqr` to solve the sparse least square problem:

$$\beta = (\sqrt{W}X)^{\dagger}\sqrt{W}y.$$

We generate random X matrices of size $(200000, 50)$, with different density. We name the sparse method as Method 4. The results are summarized as follows:

Table 2: CPU Times (s) of Four Methods for the Sparse Matrix

Density	Method 1	Method 2	Method 3	Method 4
0.01	0.142	0.417	0.056	0.015
0.1	0.116	0.383	0.061	0.035
0.25	0.111	0.409	0.061	0.053
0.5	0.099	0.372	0.053	0.212
1	0.098	0.441	0.051	0.613

We can see that:

- Method 1–3 do not exploit sparsity. Their CPU Times do not change with density.
- When density is less than or equal to 0.25, the sparse method has an advantage over the other three method. When density is greater than 0.25, the sparse method is slower, possibly due to overhead of the sparse data structure.