Computation in Data Science (Third Part) Homework 1

December 15, 2020

- Arthor: HO Ching Ru (R09946006, NTU Data Science Degree Program)
- Course: Computation in Data Science (MATH 5080), NTU 2020 Fall Semester
- Instructor: Tso-Jung Yen (Institute of Statistical Science, Academia Sinica)

1 Computing the least squares estimate via SVD

Statement **b** is Correct: $\mathbf{VDV}^T$ is the eigenvalue decomposition of $(\mathbf{X}^T\mathbf{X})^{-1}$, where $D = (\Lambda^T\Lambda)^{-1}$ is a $p \times p$ diagonal matrix.

Since $\mathbf{X}$=$\mathbf{U}\Lambda\mathbf{V}^T$, we have

$$
\begin{aligned}
&\mathbf{X}^T\mathbf{X} \\
=&(\mathbf{U}\Lambda\mathbf{V}^T)^T(\mathbf{U}\Lambda\mathbf{V}^T) \\
=&\mathbf{V}\Lambda(\mathbf{U}^T\mathbf{U})\Lambda\mathbf{V}^T \\
=&\mathbf{V}\Lambda^T\Lambda\mathbf{V}^T
\end{aligned}
$$

and

$$
\begin{aligned}
&(\mathbf{X}^T\mathbf{X})^{-1} \\
=&(\mathbf{V}\Lambda^T\Lambda\mathbf{V}^T)^{-1} \\
=&(\mathbf{V}^T)^{-1}(\Lambda^T\Lambda)^{-1}\mathbf{V}^{-1} \\
=&\mathbf{V}(\Lambda^T\Lambda)^{-1}\mathbf{V}^T \\
=&\mathbf{VDV}^T
\end{aligned}
$$

Noticed that $\mathbf{VV}^T = \mathbf{VV}^{-1} = \mathbf{I}$

2 Programming work

Now consider the following estimate:

$$
\hat{\beta}^{ridge} = \arg\min\left(\frac{1}{2}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \frac{\lambda}{2}||\mathbf{W}\beta||_2^2\right)
$$

2.1 Define

Let $\left(\beta^{true}\right)_j = 1$ for $j = 1, 2, \cdots, p$. Generate data by first drawing $(\mathbf{X})_{ij} = x_{ij}$ from Normal(0, 1) for $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, p$ and then computing response vector $\mathbf{y}$ by

$$\mathbf{y} = \mathbf{X}\beta^{true} + \epsilon$$

,

where $(\epsilon)_i$ Normal(0, 1) for $i = 1, 2 \cdots, n$.

[1]:
```
import numpy as np

def SetXy(n, p):
    np.random.seed(87)
    X = np.random.normal(size = (n, p))
    beta_true = np.ones(shape = (p, 1))
    err = np.random.normal(size = (n, 1))
    y = np.dot(X, beta_true) + err
    return X, y
```

2.2 Run the following three algorithms for computing $\hat{\beta}^{ridge}$

In here, I use `sklearn.linear_model.Ridge`. By the offical document (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html), this module is going to minimize the objective function:

`||y - Xw||^2_2 + alpha * ||w||^2_2`

Compare to the funciton define at first, $\alpha = \frac{\lambda}{2}$.

[2]:
```
from sklearn import linear_model
import time

clf = linear_model.Ridge(alpha = 1.0, solver='cholesky')
clf = linear_model.Ridge(alpha = 1.0, solver='svd')

def QR_reg(X, y):
    Q, R = np.linalg.qr(X)
    beta = np.linalg.inv(R).dot(Q.T).dot(y)
    return beta
```

2.3 Record the runtime of the above three algorithms

Report the runtime of the three algorithms in “seconds”.

1. $(n, p, \mathbf{W}, \lambda) = (1100, 10, \mathbf{0}_{p\times p}, 0.1)$, equilvalent to $\alpha = 0$.

[3]:
```
X, y = SetXy(1100, 10)

start_time = time.time()
```

```python
clf = linear_model.Ridge(alpha = 0, solver = 'cholesky')
clf.fit(X, y)
print("Solve Ridge with cholesky (second): ", time.time() - start_time)

start_time = time.time()
QR_reg(X, y)
print("Solve Ridge with QR (second): ", time.time() - start_time)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0, solver = 'svd')
clf.fit(X, y)
print("Solve Ridge with SVD (second): ", time.time() - start_time)
```

```
Solve Ridge with cholesky (second):  0.012967586517333984
Solve Ridge with QR (second):  0.012962102890014648
Solve Ridge with SVD (second):  0.011053085327148438
```

2. $(n, p, \mathbf{W}, \lambda) = (1100, 1000, \mathbf{0}_{p\times p}, 0.1)$, equilvalent to $\alpha = 0$.

[4]:
```python
X, y = SetXy(1100, 1000)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0, solver = 'cholesky')
clf.fit(X, y)
print("Solve Ridge with cholesky (second): ", time.time() - start_time)

start_time = time.time()
QR_reg(X, y)
print("Solve Ridge with QR (second): ", time.time() - start_time)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0, solver = 'svd')
clf.fit(X, y)
print("Solve Ridge with SVD (second): ", time.time() - start_time)
```

```
Solve Ridge with cholesky (second):  0.11369490623474121
Solve Ridge with QR (second):  0.33992648124694824
Solve Ridge with SVD (second):  0.9609847068786621
```

3. $(n, p, \mathbf{W}, \lambda) = (1100, 10, \mathbf{I}_{p\times p}, 0.1)$, equilvalent to $\alpha = 0.05$.

[5]:
```python
X, y = SetXy(1100, 10)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0.05, solver = 'cholesky')
clf.fit(X, y)
print("Solve Ridge with cholesky (second): ", time.time() - start_time)

start_time = time.time()
```

```
QR_reg(X, y)
print("Solve Ridge with QR (second): ", time.time() - start_time)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0.05, solver = 'svd')
clf.fit(X, y)
print("Solve Ridge with SVD (second): ", time.time() - start_time)
```

```
Solve Ridge with cholesky (second):  0.001992940902709961
Solve Ridge with QR (second):  0.0009958744049072266
Solve Ridge with SVD (second):  0.002994537353515625
```

4. $(n, p, \mathbf{W}, \lambda) = (1100, 1000, \mathbf{I}_{p\times p}, 0.1)$, equilvalent to $\alpha = 0.05$.

[6]:
```
X, y = SetXy(1100, 1000)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0.05, solver = 'cholesky')
clf.fit(X, y)
print("Solve Ridge with cholesky (second): ", time.time() - start_time)

start_time = time.time()
QR_reg(X, y)
print("Solve Ridge with QR (second): ", time.time() - start_time)

start_time = time.time()
clf = linear_model.Ridge(alpha = 0.05, solver = 'svd')
clf.fit(X, y)
print("Solve Ridge with SVD (second): ", time.time() - start_time)
```

```
Solve Ridge with cholesky (second):  0.0549015998840332
Solve Ridge with QR (second):  0.4036750793457031
Solve Ridge with SVD (second):  1.0393402576446533
```

2.4 Conclusion

- Table 1: Table format
  - (a): $(1100, 10, \mathbf{0}_{p\times p}, 0.1)$
  - (b): $(1100, 1000, \mathbf{0}_{p\times p}, 0.1)$
  - (c): $(1100, 10, \mathbf{I}_{p\times p}, 0.1)$
  - (d): $(1100, 1000, \mathbf{0}_{p\times p}, 0.1)$

Parameter	Cholesky-FBS	QR-based	SVD-based
(a)	0.012967586517333984	0.012962102890014648	0.011053085327148438
(b)	0.11369490623474121	0.33992648124694824	0.9609847068786621
(c)	0.001992940902709961	0.0009958744049072266	0.002994537353515625
(d)	0.0549015998840332	0.4036750793457031	1.0393402576446533