Notes on PA3

TAs for CSE 151

May 5, 2016

In this notes, we will give an overview of PA3, discuss back solving, QR and Householder method for QR. We will keep the language as simple as possible and give examples and steps to guide you through PA3. In the following discussions, let's fix:

- n: The number of training samples
- d: Each sample is with d dimension features
- X: $n \times d$ feature data matrix
- y: $n \times 1$ label matrix
- β: $d \times 1$ linear regression coefficients

1 PA3 Overview

In PA3, we are going to solve a linear regression problem, i.e. for n training samples, each sample with d dimension features. The goal of this assignment is to solve β such that $X\beta = y$,

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ x_{(n-1)1} & x_{(n-1)2} & \dots & x_{(n-1)d} \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \dots \\ \dots \\ \beta_d \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ \dots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

Notes that we are not using bias term here, i.e. we are NOT using

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ x_{(n-1)1} & x_{(n-1)2} & \dots & x_{(n-1)d} \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \dots \\ \dots \\ \beta_d \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \dots \\ \dots \\ \epsilon_{n-1} \\ \epsilon_n \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ \dots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

After the β is solved, we can generate prediction $\hat{y}$. Let $\hat{y}$

$$\begin{bmatrix} \hat{y_1} \\ \hat{y_2} \\ \dots \\ \dots \\ \hat{y_{n-1}} \\ \hat{y_n} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ x_{(n-1)1} & x_{(n-1)2} & \dots & x_{(n-1)d} \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \dots \\ \dots \\ \beta_d \end{bmatrix}$$

then the RMSE can be calculated as follows

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}.$$

This turns out to be a "one line solution" with addition to some data loading code. To put it in python:

```
import csv
import numpy as np

# load data
with open('regression-0.05.csv', 'rb') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    data = np.array([[float(r) for r in row] for row in reader])

# create random sample train (80%) and test set (20%)
np.random.seed(0)
np.random.shuffle(data)
train_num = int(data.shape[0] * 0.8)
X_train = data[:train_num,:-1]
Y_train = data[:train_num,-1]
X_test  = data[train_num:,:-1]
Y_test  = data[train_num:,-1]

# linear least square Y = X beta
beta, _, _, _ = np.linalg.lstsq(X_train, Y_train)

# root mean square error
print np.sqrt(np.mean((np.dot(X_test, beta) - Y_test) ** 2))

# output >> 0.0512797312056
# which matches the regression-0.05 filename
```

And you are right, the above code runs out of the box, and (almost) solves this PA directly, except that, we are not allowed to use something like `np.linalg.lstsq`, so our code will look more like this:

```
import csv
import numpy as np

# load data
with open('regression-0.05.csv', 'rb') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    data = np.array([[float(r) for r in row] for row in reader])

# create random sample train (80%) and test set (20%)
np.random.seed(0)
np.random.shuffle(data)
train_num = int(data.shape[0] * 0.8)
X_train = data[:train_num,:-1]
Y_train = data[:train_num,-1]
X_test  = data[train_num:,:-1]
Y_test  = data[train_num:,-1]

# linear least square Y = X beta
Q, R = qr_decompose(X_train)
beta = back_solve(R, np.dot(Q.T, Y_train))

# root mean square error
print np.sqrt(np.mean((np.dot(X_test, beta) - Y_test) ** 2))
```

where you should implement the `qr_decompose` and `back_solve` function on your own.

P.S. As you may have discovered already, calling library functions such as `np.random.shuffle(data)` to do the random sampling of training and test set is much simpler, and in most cases faster and less prone to error than implementing it by ourselves. In this assignment, since the core concept is QR decomposition, you are allowed to use any other libraries as long as it doesn't make your QR decomposition trivial.

2 QR Decomposition

We want to decomposition of matrix X (only consider real matrix here) into the product of Q and R, i.e.

$$X = QR,$$

where Q is a orthogonal matrix and R is an upper triangular matrix. Thus we have,

$$I = Q^T Q \text{ and } Q^T = Q^{-1}.$$

- If X is a $n \times n$ square matrix, then both Q and R are $n \times n$ square matrix.
- In our case, X is an $n \times d$ matrix where $n \geq d$.
  - Method 1: Q as an $n \times n$ matrix and R as an $n \times d$ matrix with the lower $n - d$ lines be all zero. For example, if X is a 3×2 matrix,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} -0.169 & 0.897 & 0.408 \\ -0.507 & 0.276 & -0.816 \\ -0.845 & -0.345 & 0.408 \end{bmatrix} \begin{bmatrix} -5.916 & -7.437 \\ 0 & 0.828 \\ 0 & 0 \end{bmatrix}.$$

  - Method 2: Notes that in Method 1, the last column of Q is not used. The other way to is to let Q as a $n \times d$ matrix and R as a $d \times d$ matrix. For example,

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} -0.169 & 0.897 \\ -0.507 & 0.276 \\ -0.845 & -0.345 \end{bmatrix} \begin{bmatrix} -5.916 & -7.437 \\ 0 & 0.828 \end{bmatrix}.$$

In this PA, the simplest way is to implement Method 1. The performance of Method 1 will be an issue, since as we will see later, we need to perform a large matrix multiplication during each round of Householder. You are welcomed to come up with more efficient way of QR decomposition, which might make use of the representation of Method 2.

3 Back Solving

First, let's relate QR decomposition with the requirement of this PA. We need to solve β for

$$X\beta \approx y,$$

and just for convenience, let's call

$$X\beta = y.$$

Assume we are given QR decomposition algorithm, let's perform QR decomposition on X, so that

$$X = QR.$$

Therefore, we can simplify

$$\begin{aligned} X\beta &= y \\ QR\beta &= y \\ Q^T QR\beta &= Q^T y \\ R\beta &= Q^T y, \end{aligned}$$

where R is an upper triangular matrix and $Q^T y$ is vector. In this case we can solve for β trivially using the so-called back solving method.

And example of back solving is given as follows. Consider

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Back solve (from bottom)

- Step 1: $0\beta_1 + 0\beta_2 + 1\beta_3 = 3$, thus $\beta_3 = 3$.
- Step 2: $0\beta_1 + 1\beta_2 + 2\beta_3 = 2$, thus $\beta_2 = 2 - 2\beta_3 = -4$.
- Step 3: $1\beta_1 + 2\beta_2 + 1\beta_3 = 1$, thus $\beta_1 = 1 - 2\beta_2 - \beta_3 = 1 + 8 - 3 = 6$.

4 Householder Method for QR Decomposition

Now the only missing part is the `qr_decompose` function. We are going to use Householder's method to solve that. Thanks Kareem for finding this video https://www.youtube.com/watch?v=d-yPM-bxREs that explains the process in great detail. Please do watch that video if you feel confused after reading the notes.

4.1 Algorithm Overview

We are going to mostly use the same notation as in the video. Notes that we will be adding a step of determining the sign for $||z||_2 e_1$ when calculating v.

Householder method for QR is an iterative process that takes d steps for matrix X of size $n \times d$, notes that we restrict $n \geq d$ for this discussion. Consider a 5×3 matrix X,

$$X = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix},$$

we construct Q_1 such that it makes the first column all zero except the upper triangular element

$$Q_1 X = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{bmatrix},$$

we construct Q_2 such that it makes the second column all zero except the upper triangular element

$$Q_2(Q_1 X) = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \\ 0 & 0 & * \end{bmatrix},$$

we then construct Q_3 such that it makes the third column all zero except the upper triangular element

$$Q_3(Q_2 Q_1 X) = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Finally we let

$$\begin{aligned} R &= \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ Q &= (Q_3 Q_2 Q_1)^T \end{aligned}$$

Now let's implement the actual algorithm. First we init $R = X$. R here will be used as a temp veritable to store X, $Q_1 X$, $Q_2 Q_1 X$ and etc. When the algorithm terminates, R will get its final result as an upper triangular matrix. Furthermore, we keep track Q_1, $Q_2 Q_1$, $Q_3 Q_2 Q_1$ in variable $Q_{\text{accumulated}}$. For iteration $i = 1 \sim d$, to calculate Q_i, we take the following steps.

1. First, obtain the target column z_i,

$$z_i = \begin{bmatrix} R_{i,i} \\ R_{i,i+1} \\ \dots \\ \dots \\ R_{n,i} \end{bmatrix} \in \mathbb{R}^{n-i+1}.$$

2. Then we find v_i as

$$v_i = \begin{cases} -||z_i||_2 e_{i1} - z_i & \text{if the first element of } z_i \text{ is positive} \\ ||z_i||_2 e_{i1} - z_i & \text{otherwise} \end{cases}$$

where

$$e_{i1} = \begin{bmatrix} 1 \\ 0 \\ \dots \\ \dots \\ 0 \end{bmatrix} \in \mathbb{R}^{n-i+1}.$$

Notes that same as e_{i1}, v_i is of $n - i + 1$ dimensions. Refer to Piazza @232 post on why we need to determine the sign like this.

3. Then we find Householder matrix P_i as follows

$$P_i = I - \frac{2v_i v_i^T}{v_i^T v_i}$$

where P_i is a $(n - i + 1) \times (n - i + 1)$ matrix.

4. Let

$$Q_i = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \dots & & \\ & & & 1 & \\ & & & & P_i \end{bmatrix}$$

where Q_i is a $n \times n$ matrix.

5. Update R as

$$R \leftarrow Q_i R$$

After d iterations of the above process, we got R, and let

$$\begin{aligned} Q &= (Q_d Q_{d-1} \dots Q_1)^T \\ &= Q^T_{\text{accumulated}}, \end{aligned}$$

and now we have

$$X = QR.$$

4.2 An Example Run

Householder method for QR decomposition is quite straight forward if we look at an example. Let X be

$$X = \begin{bmatrix} 1 & -1 & -1 \\ 1 & 2 & 3 \\ 2 & 1 & 1 \\ 2 & -2 & 1 \\ 3 & 2 & 1 \end{bmatrix}$$

Here's the algorithm dump for this X

```
### round #1 ###
z=
array([1, 1, 2, 2, 3])
z_norm=
4.358898943540674
v=
array([-5.36, -1.  , -2.  , -2.  , -3.  ])
P=
array([[-0.23, -0.23, -0.46, -0.46, -0.69],
       [-0.23,  0.96, -0.09, -0.09, -0.13],
       [-0.46, -0.09,  0.83, -0.17, -0.26],
       [-0.46, -0.09, -0.17,  0.83, -0.26],
       [-0.69, -0.13, -0.26, -0.26,  0.61]])
Q=
array([[-0.23, -0.23, -0.46, -0.46, -0.69],
       [-0.23,  0.96, -0.09, -0.09, -0.13],
       [-0.46, -0.09,  0.83, -0.17, -0.26],
       [-0.46, -0.09, -0.17,  0.83, -0.26],
       [-0.69, -0.13, -0.26, -0.26,  0.61]])
R=
array([[-4.36, -1.15, -2.06],
       [ 0.  ,  1.97,  2.8 ],
       [ 0.  ,  0.95,  0.6 ],
       [ 0.  , -2.05,  0.6 ],
       [ 0.  ,  1.92,  0.4 ]])
Q_accumulated=
array([[-0.23, -0.23, -0.46, -0.46, -0.69],
       [-0.23,  0.96, -0.09, -0.09, -0.13],
       [-0.46, -0.09,  0.83, -0.17, -0.26],
       [-0.46, -0.09, -0.17,  0.83, -0.26],
       [-0.69, -0.13, -0.26, -0.26,  0.61]])

### round #2 ###
z=
array([ 1.97,  0.95, -2.05,  1.92])
z_norm=
3.5614899306772996
v=
array([-5.53, -0.95,  2.05, -1.92])
P=
array([[-0.55, -0.27,  0.58, -0.54],
       [-0.27,  0.95,  0.1 , -0.09],
       [ 0.58,  0.1 ,  0.79,  0.2 ],
       [-0.54, -0.09,  0.2 ,  0.81]])
Q=
array([[ 1.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  , -0.55, -0.27,  0.58, -0.54],
       [ 0.  , -0.27,  0.95,  0.1 , -0.09],
       [ 0.  ,  0.58,  0.1 ,  0.79,  0.2 ],
       [ 0.  , -0.54, -0.09,  0.2 ,  0.81]])
R=
array([[-4.36, -1.15, -2.06],
       [ 0.  , -3.56, -1.58],
       [ 0.  ,  0.  , -0.15],
       [ 0.  ,  0.  ,  2.23],
       [ 0.  ,  0.  , -1.11]])
Q_accumulated=
array([[-0.23, -0.23, -0.46, -0.46, -0.69],
       [ 0.35, -0.49, -0.13,  0.71, -0.34],
```

```
        [-0.36, -0.33,  0.82, -0.04, -0.29],
        [-0.68,  0.45, -0.15,  0.53, -0.18],
        [-0.49, -0.63, -0.27,  0.02,  0.54]])

### round #3 ###
z=
array([-0.15,  2.23, -1.11])
z_norm=
2.4973014481278737
v=
array([ 2.64, -2.23,  1.11])
P=
array([[-0.06,  0.89, -0.45],
       [ 0.89,  0.25,  0.38],
       [-0.45,  0.38,  0.81]])
Q=
array([[ 1.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  1.  ,  0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  , -0.06,  0.89, -0.45],
       [ 0.  ,  0.  ,  0.89,  0.25,  0.38],
       [ 0.  ,  0.  , -0.45,  0.38,  0.81]])
R=
array([[-4.36, -1.15, -2.06],
       [ 0.  , -3.56, -1.58],
       [ 0.  ,  0.  ,  2.5 ],
       [ 0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ]])
Q_accumulated=
array([[-0.23, -0.23, -0.46, -0.46, -0.69],
       [ 0.35, -0.49, -0.13,  0.71, -0.34],
       [-0.37,  0.7 , -0.06,  0.47, -0.38],
       [-0.67, -0.42,  0.59,  0.11, -0.1 ],
       [-0.49, -0.19, -0.65,  0.23,  0.5 ]])
```

Finally, we output $Q = Q^T_{\text{accumulated}}$ and R.

```
Q=
array([[-0.23,  0.35, -0.37, -0.67, -0.49],
       [-0.23, -0.49,  0.7 , -0.42, -0.19],
       [-0.46, -0.13, -0.06,  0.59, -0.65],
       [-0.46,  0.71,  0.47,  0.11,  0.23],
       [-0.69, -0.34, -0.38, -0.1 ,  0.5 ]])

R=
array([[-4.36, -1.15, -2.06],
       [ 0.  , -3.56, -1.58],
       [ 0.  ,  0.  ,  2.5 ],
       [ 0.  ,  0.  ,  0.  ],
       [ 0.  ,  0.  ,  0.  ]])
```

We can verify that $X = QR$.

Let's look at the calculation of v_1 in round 1 in more details to get a understanding on how to determine the sign when applying $v_i = \pm||z_i||_2 e_{i1} - z_i$.

$$R = X = \begin{bmatrix} 1 & -1 & -1 \\ 1 & 2 & 3 \\ 2 & 1 & 1 \\ 2 & -2 & 1 \\ 3 & 2 & 1 \end{bmatrix}$$

$$z_1 = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \end{bmatrix}$$

$$||z_1|| = 4.359$$

$$
\begin{aligned}
v_1 &= -||z_1||_2 e_{11} - z_1 \\
&= -4.36 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \end{bmatrix} \\
&= \begin{bmatrix} -5.36 \\ 1 \\ 2 \\ 2 \\ 3 \end{bmatrix}.
\end{aligned}
$$

Now, we have seen the description of the algorithm and an example run. For the proof of the algorithm, please refer to the video or the discussion sections.

5 Notes on Performance

Our matrix Q are in the order of $10,000 \times 10,000$ in this assignment.

Let's do an estimation of what this implies. To store such matrix Q in 64 bit floating points, each entry is 64bits which is 8 bytes. And we have 100,000,000 of such entries, which translates to 800MB for just storing one Q. Something worse than that is that we need to perform matrix multiplications of two $10,000 \times 10,000$ matrices for d times, which could take a long time. In a typical python implementation without other optimizations, an iteration could takes around 10 seconds on a desktop for a 60%-40% split of training and test set. So be patient and print out partial results during your algorithm is running.

Speeding this algorithm up and reducing the space usage are interesting problems. Here are some potential methods we thought about (though haven't really implemented):

1. Cheat on the training / testing split. In fact, doing 60%-40% split vs 90%-10% split, can results in $(0.6 \times 0.6)/(0.9 \times 0.9) = 44\%$ of memory space used, and $(0.9 \times 0.9 \times 0.9)/(0.6 \times 0.6 \times 0.6) = 3.375$ time speed up.

2. Start with smaller size train / test samples for debugging. Then move on to the complete dataset (suggested by the professor).

3. Use 32 bit floating points to do all the computations instead of 64.

4. Observe that Q_i matrix is identity matrix with P_i in the bottom right, the upper identity matrix have no effect when Q_i is multiplied with another matrix. Try to avoid that computation.

5. Try to see if we can use the Method 2 representation of QR decomposition. If that is possible, it will significantly reduce space and time consumption.

Please let us know your methods of improvements. We'll be glad to share it with the whole class.