Machine Learning HW_01

0753736_陳懷安

2019-11-06

```
## '3.7.5 (default, Oct 31 2019, 15:18:51) [MSC v.1916 64 bit (AMD64)]'
```

Solve Equation

define M for poly

Here I just make the polynomial $\phi(x)$ up to $M = 2$. But for Gaussian basis function, I set it up to the number of total variables.

define Φ matrix

So the elements of polynomial basis function design matrix $\Phi(x)$ will look like:

$$\phi_j(x_i) = \Sigma\, \Pi_{i=0}^{j} x_i$$

For the Gaussian basis function, the form is shown below:

$$\phi_j(x) = \exp\left\{-\frac{(\mathrm{x}-\mu_\mathrm{j})^2}{2\mathrm{s}^2}\right\}, \text{ where } \mathrm{s} = 0.1$$

Solve w

By linear regression close form, we may solve w with the equation below.

$$w = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T y$$

define Cross Validation

split data

Here I'll define a function to help me with splitting the data into pieces, dealing with cross validation process or even batch learning.

CV

The cross validation will done by the function which will sequentially input train dataset and validation dataset by the dataset we get with the splitting function. And will ultimately leave the best model for us to go further.

define search function for hyperparameter

The hyperparameter here is M, to determine how many **features** are going to modelling Y.

1. Feature Selection

a. In the feature selection stage, please apply polynomials of order M = 1 and M = 2 over the dimension D = 17 of input data.

Please evaluate the corresponding RMS error on the training set and validation set.

```
M_2 = cross_val(data_x, data_t, 5, 2, 0, get_polyphi)
```

```
## by RMSE, we choose fold 4 for the training.
```

```
M_1 = cross_val(data_x, data_t, 5, 1, 0, get_polyphi)
```

```
## by RMSE, we choose fold 1 for the training.
```

The RMSE for each iteration are:

```
M_1[0]
```

```
##    train_RMSE   CV_RMSE
## 0    4.053662  4.366332
## 1    4.171933  3.880432
## 2    4.131678  4.053519
## 3    3.988291  4.687329
## 4    4.116980  4.142000
```

```
M_2[0]
```

```
##    train_RMSE   CV_RMSE
## 0    3.246968  5.234154
## 1    3.276577  4.227515
## 2    3.328116  5.779473
## 3    3.226351  4.620963
## 4    3.331245  4.170861
```

> From the result above, we can easily find out that train_RMSE is lower when M = 2, but CV_RMSE gets higher simultaneously.
> We may say it's the consequence of overfitting becuase the model is too complex.

b. Please analyze the **weights of polynomial models for $\mathrm{M} = 1$** and select the most contributive attribute which has the lowest RMS error on the Training Dataset.

```
M_1[1]
```

```
## array([[-2.44723869e+01],
##        [ 2.60835142e-02],
##        [ 2.60926714e+01],
##        [ 2.17266677e+01],
##        [-2.91426002e+01],
##        [ 4.93014201e-01],
##        [ 9.98870540e-01],
##        [-7.92547548e-01],
##        [ 2.58027849e-02],
##        [ 4.04418498e-01],
##        [-8.94421421e-01],
##        [ 5.16159574e-02],
##        [ 6.42201793e-01],
##        [-1.67082141e+01],
##        [ 5.43975704e-02],
##        [-4.97112073e-02],
##        [ 2.00746656e+00],
##        [-3.64741696e+00]])
```

> The list above is the w of the model which provides the lowest CV_RMSE when $\mathrm{M} = 1$.

2. Maximum Likelihood Approach

a. **Choose some of air quality measurement** in dataset X.csv and design your model.

You can choose any basis functions you like and implemented the feature vector.

> Here I **selected 10 air quality measurements** as independent variables, and conducted a cross-validation process to make sure that the model won't be over-fitting.

```
M_G_10 = cross_val(data_x, data_t, 5, 10, 0, get_gaussphi)
```

```
## by RMSE, we choose fold 2 for the training.
```

```
M_G_10[0]
```

```
##    train_RMSE   CV_RMSE
## 0    8.467973  8.608005
## 1    8.364499  9.124106
## 2    8.602879  8.026633
## 3    8.397402  8.927503
## 4    8.592132  8.259819
```

b. Apply N-fold cross-validation in your training stage to select at least one hyperparameter (order, parameter number, …) for model and do some discussion (underfitting, overfitting).

I go through 18 variables and try how many of them put into the model will provide a best prediction.

```
w_SP = search_hyper(data_x, data_t, 7, 18, 1, get_gaussphi)
```

```
## iter: 0
## by RMSE, we choose fold 0 for the training.
## iter: 1
## by RMSE, we choose fold 4 for the training.
## iter: 2
## by RMSE, we choose fold 4 for the training.
## iter: 3
## by RMSE, we choose fold 6 for the training.
## iter: 4
## by RMSE, we choose fold 1 for the training.
## iter: 5
## by RMSE, we choose fold 5 for the training.
## iter: 6
## by RMSE, we choose fold 5 for the training.
## iter: 7
## by RMSE, we choose fold 5 for the training.
## iter: 8
## by RMSE, we choose fold 5 for the training.
## iter: 9
## by RMSE, we choose fold 5 for the training.
## iter: 10
## by RMSE, we choose fold 3 for the training.
## iter: 11
## by RMSE, we choose fold 2 for the training.
## iter: 12
## by RMSE, we choose fold 6 for the training.
## iter: 13
## by RMSE, we choose fold 1 for the training.
## iter: 14
## by RMSE, we choose fold 4 for the training.
## iter: 15
## by RMSE, we choose fold 4 for the training.
## iter: 16
## by RMSE, we choose fold 2 for the training.
## iter: 17
## by RMSE, we choose fold 5 for the training.
##
## From iteration 11, we got the lowest CV_RMSE: 7.464618560198112, and the weight is save to w_opt
```

> **w_opt**(the w of the best model) is shown in the chunk below.

```
w_SP
```

```
## [            0
## 0   20.924233
## 1    4.879413
## 2    1.304555
## 3   -5.184143
## 4    0.894483
## 5  -10.584976
## 6   -4.150115
## 7    7.439121
## 8    0.510126
## 9  -14.031447
## 10  11.892458
## 11   1.040463, 11]
```

3. Maximum a posteriori approach

a. Use maximum a posteriori approach method and repeat **2.(a)** and **2.(b)**. You could choose Gaussian distribution as a prior.

> I'll choose **Gaussian basis function** here, and try to calculate the posterior distribution.
>
> Add a Gaussian noise to the model:
>
> $$\epsilon \sim N(0,\ \beta)$$
>
> We may renew our parameters by the functions below:
>
> $$p(w|t) = N(w|m_N,\ S_N)\ , \text{where}$$
> $$\mathrm{S_N^{-1}} = \mathrm{S_0^{-1}} + \beta\Phi^\mathrm{T}\Phi$$
> $$\mathrm{m_N} = \mathrm{S_N}(\mathrm{S_0^{-1}}\mathrm{m_0} + \beta\Phi^\mathrm{T}\mathrm{y})$$

```
M = w_SP[1]
m0 = np.zeros(M+1).reshape(-1,1)
s0 = 2*np.eye(M+1)
beta = 0.2
```

```
mn = batch_learn(data_x, data_t, 100, s0, m0, M)[1]
M_G_ML = cross_val(data_x, data_t, 7, 5, 0, get_gaussphi)
```

```
## by RMSE, we choose fold 6 for the training.
```

b. Compare the result between maximum likelihood approach and maximum a posteriori approach.

> The results below, I'll show the RMSE of w from MAP first and then from Maximize Likelihood.

```
((np.ones(data_t.shape[0]) @ ((get_gaussphi(data_x, M) @ mn - data_t.values)**2))/data_t.shape[0])**
```

```
## 0    8.508731
## dtype: float64
```

```
M_G_ML[0]
```

```
##    train_RMSE    CV_RMSE
## 0    9.394594   9.426653
## 1    9.264750  10.165422
## 2    9.221448  10.498745
## 3    9.409140   9.314722
## 4    9.395073   9.409473
## 5    9.448078   9.043666
## 6    9.589472   8.378572
```