Machine Learning 2021 Homework 1

310511048 張奕廷

1. Bayesian Linear Regression

1. A and B are said to be conditionally independent if

$$P(A \mid B, C) = P(A \mid C)$$

Since (x, t) is a new test point and label, $(\mathbf{x}, \mathbf{t})$ are the training data and corresponding label, and $\mathbf{w}$ is only trained on $(\mathbf{x}, \mathbf{t})$, we can conclude that (x, t) and $(\mathbf{x}, \mathbf{t})$ are independent given $\mathbf{w}$. Hence, we can obtain:

$$p(t \mid \mathbf{w}, x, \mathbf{x}, \mathbf{t}) = p(t \mid x, \mathbf{w})$$

Furthermore, $\mathbf{w}$ is only dependent on its training data $(\mathbf{x}, \mathbf{t})$, hence:

$$p(\mathbf{w} \mid x, \mathbf{x}, \mathbf{t}) = p(\mathbf{w} \mid \mathbf{x}, \mathbf{t})$$

2. The equations from textbook page 93 are list below.

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mu, \mathbf{\Lambda}^{-1}) \tag{2.113}$$
$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid \mathbf{A}\mathbf{x} + b, \mathbf{L}^{-1}) \tag{2.114}$$
$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid \mathbf{A}\mu + b, \mathbf{L}^{-1} + \mathbf{A}\mathbf{\Lambda}^{-1}\mathbf{A}^T) \tag{2.115}$$
$$p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \mathbf{\Sigma}(\mathbf{A}^T\mathbf{L}(\mathbf{y} - b) + \mathbf{\Lambda}\mu), \mathbf{\Sigma}) \tag{2.116}$$
$$\mathbf{\Sigma} = (\mathbf{\Lambda} + \mathbf{A}^T\mathbf{L}\mathbf{A})^{-1} \tag{2.117}$$

From 1.1, we know that,

$$p(t \mid x, \mathbf{x}, \mathbf{t}) = \int_{-\infty}^{\infty} p(t \mid x, \mathbf{w}) p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) d\mathbf{w}$$

First Step

To derive $p(t \mid x, \mathbf{x}, \mathbf{t})$, we first derive $p(\mathbf{w} \mid \mathbf{x}, \mathbf{t})$, from hint:

$$p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}) \propto p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) p(\mathbf{w})$$

By equation (2.114),

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) = \mathcal{N}(\mathbf{t} \mid \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}), \beta^{-1}\mathbf{I})$$
$$\Rightarrow \mathbf{A} = \mathbf{\Phi}(\mathbf{x})^T, b = 0, \mathbf{L} = \beta\mathbf{I}$$

and by equation (2.113), we obtain the prior distribution as follows,

$$\begin{aligned} p(\mathbf{w}) &= p(\mathbf{w} \mid \alpha) \\ &= \mathcal{N}(\mathbf{w} \mid 0, \alpha^{-1}\mathbf{I}) \\ &\Rightarrow \mu = 0, \mathbf{\Lambda} = \alpha\mathbf{I} \end{aligned}$$

Then, by equation (2.117) and substitute the results above,

$$\begin{aligned} \mathbf{\Sigma} &= (\mathbf{\Lambda} + \mathbf{A}^T\mathbf{L}\mathbf{A})^{-1} \\ &= (\alpha\mathbf{I} + \mathbf{\Phi}(\mathbf{x})\beta\mathbf{I}\mathbf{\Phi}(\mathbf{x})^T)^{-1} \end{aligned}$$

$$= \left(\alpha \mathbf{I} + \beta \sum_{n=1}^{N} \mathbf{\Phi}(x_n) \mathbf{\Phi}(x_n)^T \right)^{-1}$$

$$= \mathbf{S}$$

Hence, by equation (2.116),

$$p(\,\mathbf{w} \mid \mathbf{x}, \mathbf{t}\,) = \mathcal{N}(\,\mathbf{w} \mid \mathbf{S}(\mathbf{\Phi}(\mathbf{x})\beta\mathbf{I}\mathbf{t}), \mathbf{S}\,)$$

$$= \mathcal{N}(\,\mathbf{w} \mid \beta\mathbf{S}(\mathbf{\Phi}(\mathbf{x})\mathbf{t}), \mathbf{S}\,)$$

<u>Second Step</u>

Now we derive $p(\,t \mid x, \mathbf{w}\,)$. Again, by equation (2.114),

$$p(\,t \mid x, \mathbf{w}\,) = \mathcal{N}(\,t \mid \mathbf{w}^T \mathbf{\Phi}(x), \beta^{-1}\mathbf{I}\,)$$

$$\Rightarrow \boldsymbol{A} = \mathbf{\Phi}(x)^T, b = 0, \mathbf{L} = \beta\mathbf{I}$$

By equation (2.113) and the derivation from first step,

$$p(\,\mathbf{w} \mid \mathbf{x}, \mathbf{t}\,) = \mathcal{N}(\,\mathbf{w} \mid \beta\mathbf{S}(\mathbf{\Phi}(\mathbf{x})\mathbf{t}), \mathbf{S}\,)$$

$$= p(\,\mathbf{w} \mid \mu, \mathbf{\Lambda}^{-1}\,)$$

$$\Rightarrow \mu = \beta\mathbf{S}(\mathbf{\Phi}(\mathbf{x})\mathbf{t}), \mathbf{\Lambda}^{-1} = \mathbf{S}$$

Finally, by equation (2.115) and substitute the results above,

$$p(\,t \mid x, \mathbf{x}, \mathbf{t}\,) = \int_{-\infty}^{\infty} p(\,t \mid x, \mathbf{w}\,) p(\,\mathbf{w} \mid \mathbf{x}, \mathbf{t}\,) d\mathbf{w}$$

$$= \mathcal{N}\left(t \,\middle|\, \beta\mathbf{\Phi}(x)^T \mathbf{S} \textstyle\sum_{n=1}^{N} \mathbf{\Phi}(x_n) t_n \,, \beta^{-1} I + \mathbf{\Phi}(x)^T \mathbf{S} \mathbf{\Phi}(x)\right)$$

$$= \mathcal{N}\left(t \,\middle|\, m(x), s^2(x)\right)$$

2. Linear Regression

1. Feature Selection

(a) **Code Result.**

```
M = 1, train_rms: 0.60200, valid_rms: 0.60505
M = 2, train_rms: 0.54713, valid_rms: 0.55658
```

(b) **Code Result. Explain.** I remove one feature from the dataset at a time and observe the RMS error. As we can see from the code result, both training and validation RMS error are greatest when <u>feature 8</u> (*median income*) is removed. As a result, the feature *median income* is the most contributive one in this dataset.

```
Without feature 1, train_rms: 0.65197, valid_rms: 0.65645
Without feature 2, train_rms: 0.65700, valid_rms: 0.66291
Without feature 3, train_rms: 0.61271, valid_rms: 0.61434
Without feature 4, train_rms: 0.60368, valid_rms: 0.60633
Without feature 5, train_rms: 0.60594, valid_rms: 0.60917
Without feature 6, train_rms: 0.61915, valid_rms: 0.62723
Without feature 7, train_rms: 0.60250, valid_rms: 0.60596
Without feature 8, train_rms: 0.78324, valid_rms: 0.79439
```

2. Maximum Likelihood Approach

(a) **Explain.** In problem 2.1, I already used polynomial as the basis function for my regression model. However, polynomials are *global* basis functions, each affecting the prediction over the whole input space. *Local* basis functions are often more appropriate, so I choose Gaussian distribution as the basis function.

(b) **Code Result. Explain.** Since we're not required to find the best parameters for basis functions in this homework, I choose the Gaussian distribution as follows,

$$\mathcal{N}\left(\mu = \frac{\mathrm{m}}{\mathrm{M}+1}, \sigma^2 = (0.05)^2\right)$$

where M is the order of basis function and m is all positive integers less than M. Below is the code result, we can see that as M increases, the training error slightly decreases yet the validation error increases significantly after $\mathrm{M} = 14$, which means that the model is overfitting. Also, I discover that changing the basis function to Gaussian doesn't make the RMS error better than in problem 2.1. I think this is because I didn't choose the best parameters for Gaussian.

```
M =  1, train_rms:   0.95876, valid_rms:       0.96147
M =  2, train_rms:   0.98351, valid_rms:       0.98325
M =  3, train_rms:   0.92702, valid_rms:       0.93159
M =  4, train_rms:   0.92522, valid_rms:       0.91183
M =  5, train_rms:   0.90463, valid_rms:       0.90941
M =  6, train_rms:   0.88238, valid_rms:       0.87905
M =  7, train_rms:   0.88527, valid_rms:       0.89028
M =  8, train_rms:   0.87205, valid_rms:       0.87643
M =  9, train_rms:   0.87575, valid_rms:       0.87266
M = 10, train_rms:   0.86510, valid_rms:       0.87303
M = 11, train_rms:   0.86259, valid_rms:       0.86671
M = 12, train_rms:   0.86212, valid_rms:       0.86906
M = 13, train_rms:   0.85737, valid_rms:       0.86629
M = 14, train_rms:   0.85473, valid_rms:      11.71325
M = 15, train_rms:   0.85388, valid_rms:      54.10811
M = 16, train_rms:   0.85018, valid_rms:    3760.48644
M = 17, train_rms:   0.84604, valid_rms:   10612.07434
M = 18, train_rms:   0.84314, valid_rms:   23111.32806
M = 19, train_rms:   0.83869, valid_rms:   53358.10631
M = 20, train_rms:   0.83536, valid_rms:   15483.01912
```

(c) **Code Result. Explain.** Code result for N-fold cross validation with N set as 5 are as follows. Obviously, we can see that every fold has similar behavior. When the order is low ($\mathrm{M} = 1 \sim 5$), both training and validation RMS error are still high, demonstrating that the model is underfitting, but when the order grows higher ($\mathrm{M} = 10 \sim 13$), the error seems to converge. However, the validation RMS error starts to increase significantly when $\mathrm{M} = 14$ yet the training RMS error keep on decreasing, indicating the model is overfitting.

```
Fold = 0, M =  1, train_rms:   0.96006, valid_rms:      0.95570
Fold = 0, M =  2, train_rms:   0.98184, valid_rms:      0.98808
Fold = 0, M =  3, train_rms:   0.92887, valid_rms:      0.92692
Fold = 0, M =  4, train_rms:   0.92312, valid_rms:      0.92921
Fold = 0, M =  5, train_rms:   0.90650, valid_rms:      0.90211
Fold = 0, M =  6, train_rms:   0.88368, valid_rms:      0.87692
Fold = 0, M =  7, train_rms:   0.88850, valid_rms:      0.88114
Fold = 0, M =  8, train_rms:   0.87501, valid_rms:      0.86862
Fold = 0, M =  9, train_rms:   0.87499, valid_rms:      0.88111
Fold = 0, M = 10, train_rms:   0.86906, valid_rms:      0.86105
Fold = 0, M = 11, train_rms:   0.86406, valid_rms:      0.86774
Fold = 0, M = 12, train_rms:   0.86449, valid_rms:      0.86494
Fold = 0, M = 13, train_rms:   0.86071, valid_rms:      0.86011
Fold = 0, M = 14, train_rms:   0.85810, valid_rms:      4.71295
Fold = 0, M = 15, train_rms:   0.85563, valid_rms:      4.03407
Fold = 0, M = 16, train_rms:   0.85269, valid_rms:   3363.25024
Fold = 0, M = 17, train_rms:   0.84704, valid_rms:   2819.54999
Fold = 0, M = 18, train_rms:   0.84273, valid_rms:   4439.28902
Fold = 0, M = 19, train_rms:   0.83920, valid_rms:   3906.48486
Fold = 0, M = 20, train_rms:   5.13988, valid_rms: 236094.10735

Fold = 1, M =  1, train_rms:   0.96160, valid_rms:      0.95185
Fold = 1, M =  2, train_rms:   0.98257, valid_rms:      0.98605
Fold = 1, M =  3, train_rms:   0.92944, valid_rms:      0.92406
Fold = 1, M =  4, train_rms:   0.91755, valid_rms:      0.94278
Fold = 1, M =  5, train_rms:   0.90434, valid_rms:      0.90976
Fold = 1, M =  6, train_rms:   0.87767, valid_rms:      0.90293
Fold = 1, M =  7, train_rms:   0.88394, valid_rms:      0.90092
Fold = 1, M =  8, train_rms:   0.87004, valid_rms:      0.89280
Fold = 1, M =  9, train_rms:   0.86980, valid_rms:      0.90094
Fold = 1, M = 10, train_rms:   0.86259, valid_rms:      0.89042
Fold = 1, M = 11, train_rms:   0.85696, valid_rms:      0.89689
Fold = 1, M = 12, train_rms:   0.85757, valid_rms:      0.89500
Fold = 1, M = 13, train_rms:   0.85343, valid_rms:      0.89087
Fold = 1, M = 14, train_rms:   0.85030, valid_rms:     19.14862
Fold = 1, M = 15, train_rms:   0.84758, valid_rms:     80.25730
Fold = 1, M = 16, train_rms:   0.84285, valid_rms:  17628.84597
Fold = 1, M = 17, train_rms:   0.84723, valid_rms:  33279.47381
Fold = 1, M = 18, train_rms:   0.83314, valid_rms: 104836.49409
Fold = 1, M = 19, train_rms:   0.86782, valid_rms:  58096.96479
Fold = 1, M = 20, train_rms:   0.84779, valid_rms:  75532.86477

Fold = 2, M =  1, train_rms:   0.95548, valid_rms:      0.97568
Fold = 2, M =  2, train_rms:   0.98507, valid_rms:      0.98001
Fold = 2, M =  3, train_rms:   0.92471, valid_rms:      0.94254
Fold = 2, M =  4, train_rms:   0.92490, valid_rms:      0.92423
Fold = 2, M =  5, train_rms:   0.90347, valid_rms:      0.92036
Fold = 2, M =  6, train_rms:   0.88064, valid_rms:      0.89554
Fold = 2, M =  7, train_rms:   0.88348, valid_rms:      0.90449
Fold = 2, M =  8, train_rms:   0.86956, valid_rms:      0.89380
Fold = 2, M =  9, train_rms:   0.87551, valid_rms:      0.88252
Fold = 2, M = 10, train_rms:   0.86235, valid_rms:      0.89552
Fold = 2, M = 11, train_rms:   0.86199, valid_rms:      0.88819
Fold = 2, M = 12, train_rms:   0.86057, valid_rms:      0.89092
Fold = 2, M = 13, train_rms:   0.85519, valid_rms:      0.89391
Fold = 2, M = 14, train_rms:   0.85332, valid_rms:     20.33935
Fold = 2, M = 15, train_rms:   0.85302, valid_rms:     42.16960
Fold = 2, M = 16, train_rms:   0.84989, valid_rms:   2412.15083
Fold = 2, M = 17, train_rms:   0.92612, valid_rms:  31108.68737
Fold = 2, M = 18, train_rms:   0.84307, valid_rms:  46743.65033
Fold = 2, M = 19, train_rms:   0.85871, valid_rms:  90007.72399
Fold = 2, M = 20, train_rms:   0.84031, valid_rms: 193380.37135

Fold = 3, M =  1, train_rms:   0.95949, valid_rms:      0.95668
Fold = 3, M =  2, train_rms:   0.98248, valid_rms:      0.98819
Fold = 3, M =  3, train_rms:   0.92699, valid_rms:      0.92810
Fold = 3, M =  4, train_rms:   0.91779, valid_rms:      0.94211
Fold = 3, M =  5, train_rms:   0.90351, valid_rms:      0.91005
Fold = 3, M =  6, train_rms:   0.87758, valid_rms:      0.89905
Fold = 3, M =  7, train_rms:   0.88293, valid_rms:      0.89721
Fold = 3, M =  8, train_rms:   0.86983, valid_rms:      0.88582
Fold = 3, M =  9, train_rms:   0.87017, valid_rms:      0.89499
Fold = 3, M = 10, train_rms:   0.86370, valid_rms:      0.87732
Fold = 3, M = 11, train_rms:   0.85812, valid_rms:      0.88409
Fold = 3, M = 12, train_rms:   0.85853, valid_rms:      0.88358
Fold = 3, M = 13, train_rms:   0.85438, valid_rms:      0.87980
Fold = 3, M = 14, train_rms:   0.85133, valid_rms:     45.20021
Fold = 3, M = 15, train_rms:   0.84978, valid_rms:     29.65273
Fold = 3, M = 16, train_rms:   0.84457, valid_rms:   1314.61251
Fold = 3, M = 17, train_rms:   0.84039, valid_rms:  76179.26295
Fold = 3, M = 18, train_rms:   0.83668, valid_rms: 221838.38775
Fold = 3, M = 19, train_rms:   0.83777, valid_rms:  53186.04450
Fold = 3, M = 20, train_rms:   0.85107, valid_rms: 188812.25618

Fold = 4, M =  1, train_rms:   0.95802, valid_rms:      0.96355
Fold = 4, M =  2, train_rms:   0.98420, valid_rms:      0.98023
Fold = 4, M =  3, train_rms:   0.92657, valid_rms:      0.93376
Fold = 4, M =  4, train_rms:   0.92563, valid_rms:      0.91418
Fold = 4, M =  5, train_rms:   0.90444, valid_rms:      0.90694
Fold = 4, M =  6, train_rms:   0.88422, valid_rms:      0.87720
Fold = 4, M =  7, train_rms:   0.88698, valid_rms:      0.88632
Fold = 4, M =  8, train_rms:   0.87370, valid_rms:      0.87624
Fold = 4, M =  9, train_rms:   0.87747, valid_rms:      0.87108
Fold = 4, M = 10, train_rms:   0.86579, valid_rms:      0.87595
Fold = 4, M = 11, train_rms:   0.86454, valid_rms:      0.86715
Fold = 4, M = 12, train_rms:   0.86358, valid_rms:      0.86977
Fold = 4, M = 13, train_rms:   0.85834, valid_rms:      0.87121
Fold = 4, M = 14, train_rms:   0.85663, valid_rms:      1.81110
Fold = 4, M = 15, train_rms:   0.85503, valid_rms:     17.10808
Fold = 4, M = 16, train_rms:   0.85078, valid_rms:    398.16723
Fold = 4, M = 17, train_rms:   0.84701, valid_rms:  21207.31205
Fold = 4, M = 18, train_rms:   0.84301, valid_rms:  36028.76683
Fold = 4, M = 19, train_rms:   0.84017, valid_rms:  31253.34101
Fold = 4, M = 20, train_rms:   0.94297, valid_rms: 109723.05547
```

3. Maximum A Posterior Approach

(a) **Explain.**

Maximum Likelihood

$$\mathbf{w}_{ML} = \underset{\mathbf{w}}{\operatorname{argmax}}\left(\ln\left(p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \beta)\right)\right)$$

$$\propto \underset{\mathbf{w}}{\operatorname{argmax}}\left(-\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2\right)$$

$$\propto \underset{\mathbf{w}}{\operatorname{argmin}}\left(\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2\right)$$

Maximum a Posterior

$$\mathbf{w}_{MAP} = \underset{\mathbf{w}}{\operatorname{argmax}}\left(\ln\left(p(\mathbf{w} \mid \mathbf{x}, \mathbf{t}, \alpha, \beta)\right)\right)$$

$$\propto \underset{\mathbf{w}}{\operatorname{argmax}}\left(\ln\left(p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}, \beta)\right) + \ln\left(p(\mathbf{w} \mid \alpha)\right)\right)$$

If we choose Gaussian distribution as the prior, then

$$\mathbf{w}_{MAP} \propto \underset{\mathbf{w}}{\operatorname{argmax}}\left(-\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2 - \frac{\alpha}{2}\|w\|^2\right)$$

$$\propto \underset{\mathbf{w}}{\operatorname{argmin}}\left(\frac{\beta}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2 + \frac{\alpha}{2}\|w\|^2\right)$$

The difference between *maximum likelihood* and *maximum a posterior approach* is highlighted in red in the above equations, which is the prior distribution. In *maximum likelihood approach*, there is no this term. Also, the final derivation of *maximum likelihood approach* can be viewed as **L**east **S**quares (**LS**), while in *maximum a posterior approach*, if we choose Gaussian distribution as the prior, the final derivation can be viewed as **R**egularized **L**east **S**quares (**RLS**). Due to the regularized term, we can avoid models from overfitting to training data to some extent with *maximum a posterior approach*.

(b) **Code Result.**

```
M =  1, train_rms:   0.95876, valid_rms:   0.96147
M =  2, train_rms:   0.98351, valid_rms:   0.98325
M =  3, train_rms:   0.92702, valid_rms:   0.93159
M =  4, train_rms:   0.92522, valid_rms:   0.91183
M =  5, train_rms:   0.90463, valid_rms:   0.90941
M =  6, train_rms:   0.88238, valid_rms:   0.87905
M =  7, train_rms:   0.88527, valid_rms:   0.89028
M =  8, train_rms:   0.87205, valid_rms:   0.87643
M =  9, train_rms:   0.87575, valid_rms:   0.87266
M = 10, train_rms:   0.86510, valid_rms:   0.87303
M = 11, train_rms:   0.86259, valid_rms:   0.86671
M = 12, train_rms:   0.86212, valid_rms:   0.86906
M = 13, train_rms:   0.85737, valid_rms:   0.86629
M = 14, train_rms:   0.85475, valid_rms:   0.87559
M = 15, train_rms:   0.85409, valid_rms:   0.96351
M = 16, train_rms:   0.85023, valid_rms:   2.17879
M = 17, train_rms:   0.84595, valid_rms:   2.10646
M = 18, train_rms:   0.84205, valid_rms:   1.33886
M = 19, train_rms:   0.83848, valid_rms:   1.02324
M = 20, train_rms:   0.83549, valid_rms:   0.91889
```

(c) **Explain.** Compared to the results from *maximum likelihood approach*, applying *maximum a posterior approach* can indeed ease the effect of overfitting. As we can see in the code result above, with the regularized term parameter set as $\lambda = 0.0001$, the validation RMS error is much smaller compared to previous results in problem 2.2.