STA 623 HW8

Lingyun Shao

2018/11/28

Homework Assignment 8

Suppose we are interested in predicting the time series of a discrete variable with d states of nature, $\{y_t\}, y_t = 1, ..., d, t = 1, ...$

Given data $y^{(T)} = \{y_1, ..., y_T\}, y_T = l$, we want to derive a probabilistic forecast for y_{T+1}, i.e.

$$Pr(y_{T+1} = j | y_T = l, y^{(T)}) := \tilde{\lambda}_{jl}$$

.

We can get $n_{jl}^{(T)} = \#$ of $l \to j$ transitions within $y_1, ..., y_T$ and $n_{+l}^{(T)} = \#$ of the points we start at l within transitions within $y_1, ..., y_{T-1}$ from data $y^{(T)}$. Also we have prior belief that the transition probabilities from state l, λ_{+l} follow a $Dir(\alpha_{1l}, ..., \alpha_{dl})$.

(1) Derive an analytic expression for $\tilde{\lambda}_{jl}$[the predictive].

From lecture 18, we know

$$n_{jl}^{(T)} = \sum_{t=2}^{T} \mathbf{1}(y_t = j, y_{t-1} = l)$$
$$n_{+l}^{(T)} = \sum_{t=2}^{T} \mathbf{1}(y_{t-1} = l)$$

For conjugacy, we can easily update the parameters in our prior and get posterior distribution of λ_{+l}

$$(\lambda_{+l} | y_1, ..., y_T) = Dir(\alpha_{1l} + n_{1l}^{(T)}, ..., \alpha_{dl} + n_{dl}^{(T)})$$
$$(\Lambda | y_1, ..., y_T) = \prod_{l=1}^{d} Dir(\lambda_{+l}; \alpha_{1l} + n_{1l}^{(T)}, ..., \alpha_{dl} + n_{dl}^{(T)})$$

where $\Lambda = [\lambda_{+1}, ..., \lambda_{+d}]$ is the transition matrix and $\lambda_{+1} = \begin{pmatrix} \lambda_{1l} \\ \vdots \\ \lambda_{dl} \end{pmatrix}$.

Given all the information above, we can derive our probabilistic forecast (predictive distribution):

$$
\begin{aligned}
\tilde{\lambda}_{jl} &= Pr(y_{T+1}=j|y_T=l,y^{(T)}) \\
&= \int_\Delta Pr(y_{T+1}=j|\lambda_{+l})p(\lambda_{+l}|y_T=l,y^{(T)})d\lambda_{+l} \\
&= \int_\Delta \lambda_{jl}p(\lambda_{+l}|y_T=l,y^{(T)})d\lambda_{+l} \\
&= \iint \lambda_{jl} \times \frac{1}{B(\boldsymbol{\alpha_{+l}}+\boldsymbol{n^{(T)}_{+l}})}\prod_{k=1}^{d}\lambda_{kl}^{\alpha_{kl}+n_{kl}^{(T)}-1}d\lambda_{1l}...d\lambda_{dl} \\
&= \iint \frac{1}{B(\boldsymbol{\alpha_{+l}}+\boldsymbol{n^{(T)}_{+l}})}\lambda_{jl}^{\alpha_{jl}+n_{jl}^{(T)}}\prod_{j\neq k=1}^{d}\lambda_{kl}^{\alpha_{kl}+n_{kl}^{(T)}-1}d\lambda_{1l}...d\lambda_{dl}
\end{aligned}
$$

where Δ is the support of Dirichlet distribution, $B(\boldsymbol{\alpha_{+l}}+\boldsymbol{n^{(T)}_{+l}}) = \frac{\prod_{k=1}^{d}\Gamma(\alpha_{kl}+n_{kl}^{(T)})}{\Gamma(\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)})}$, is a multivariate beta function.

We can easily find that $\lambda_{jl}^{\alpha_{jl}+n_{jl}^{(T)}}\prod_{j\neq k=1}^{d}\lambda_{kl}^{\alpha_{kl}+n_{kl}^{(T)}-1}$ is the kernel of $Dir(\alpha_{1l}+n_{1l}^{(T)},...,\alpha_{jl}+n_{jl}^{(T)}+1,...,\alpha_{dl}+n_{dl}^{(T)})$, its integral is inverse of the normalizing constant for this Dirichlet distribution. Therefore, we can derive that

$$
\begin{aligned}
\tilde{\lambda}_{jl} &= \frac{\frac{\Gamma(\alpha_{jl}+n_{jl}^{(T)}+1)\prod_{j\neq k=1}^{d}\Gamma(\alpha_{kl}+n_{kl}^{(T)})}{\Gamma(\alpha_{jl}+n_{jl}^{(T)}+1+\sum_{j\neq k=1}^{d}\alpha_{kl}+n_{kl}^{(T)})}}{B(\boldsymbol{\alpha_{+l}}+\boldsymbol{n^{(T)}_{+l}})} \\
&= \frac{\frac{\Gamma(\alpha_{jl}+n_{jl}^{(T)}+1)\prod_{j\neq k=1}^{d}\Gamma(\alpha_{kl}+n_{kl}^{(T)})}{\Gamma(1+\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)})}}{\frac{\prod_{k=1}^{d}\Gamma(\alpha_{kl}+n_{kl}^{(T)})}{\Gamma(\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)})}} \\
&= \frac{\Gamma(\alpha_{jl}+n_{jl}^{(T)}+1)}{\Gamma(\alpha_{jl}+n_{jl}^{(T)})}\frac{\Gamma(\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)})}{\Gamma(\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)}+1)} \\
&= \frac{\alpha_{jl}+n_{jl}^{(T)}}{\sum_{k=1}^{d}\alpha_{kl}+n_{kl}^{(T)}}
\end{aligned}
$$

(2) Define an algorithm for automatically producing a probabilistic forecast from $y^{(T)}$.

To derive a probabilistic forecast from $y^{(T)}$, I defined a function `f` to automatically do the prediction, which takes two argument.

- `y` is the time series $y^{(T)}$ we have observed, our data.
- `alpha` is a vector of α parameters for our Dirichlet prior for λ_{+l}

```
# assume state being from 1 to d
f = function(y, alpha) {
    d = length(unique(y))
    n = length(y)
    yt = y[n]  # the state at time t
    N = rep(0, d)
    for (t in 2:n) {
        to = y[t]
        from = y[t - 1]
        if (from == yt) {
            N[to] = N[to] + 1
        }
    }
    Nalpha = N + alpha
    pred = Nalpha/sum(Nalpha)  # predicted probabilities
    names(pred) = paste("y[t+1]=", 1:d, seq = "")
    # barplot(pred, col = rainbow(d))
    require(ggplot2)
    dt = data.frame(prob = pred, pre = factor(1:d))
    p = ggplot(data = dt, aes(x = pre, prob, fill = pre)) + geom_bar(position = "dodge",
        stat = "identity")
    print(p)
    return(pred)
}
```

Based on the forecast function I defined, I can do a test of my forecast function based on some simulated data `y` and prior `alpha`, which shows how my prediction works.

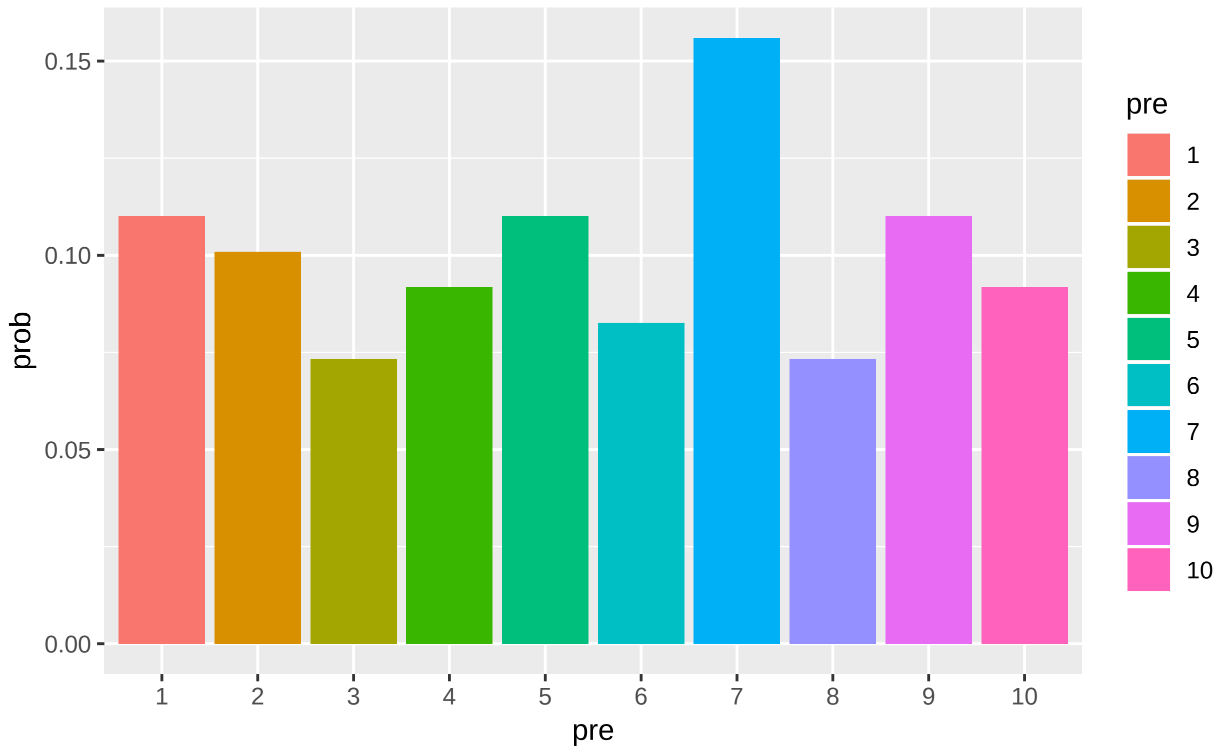
Forecast test 1

Suppose we have 10 states of nature, and we generate `y` by randomly sample from 10 states with equal probability. Given $y_T = 10$, $y^{(T)}$ and parameters all being 1, we can get the prediction for y_{T+1} as below.

```
d = 10  # number of states
set.seed(10)
y_10 = sample(d, size = 1000, replace = TRUE)  # simulated time series
y_10[1000]
```

```
## [1] 10
```

```
alpha = rep(1, d)  # prior
(p10 = f(y_10, alpha))
```



```
##  y[t+1]= 1   y[t+1]= 2   y[t+1]= 3   y[t+1]= 4   y[t+1]= 5   y[t+1]= 6
##  0.11009174  0.10091743  0.07339450  0.09174312  0.11009174  0.08256881
##  y[t+1]= 7   y[t+1]= 8   y[t+1]= 9  y[t+1]= 10
##  0.15596330  0.07339450  0.11009174  0.09174312
```

Forecast test 2

Actually the previous simulation is not a very good example since we just randomly sampled y values and they have no dependence on their previous values. Now I will use a transition-matrix-based data generating process and generate some training data to get a probabilistic forecast and test data for later calibration check. Suppose we have 4 states of nature, and we generate a sequence of `y` based on a randomly generated transition matrix and $y_1 = 1$. Given $y_T = 3$, $y^{(T)}, T = 1000$ and prior parameters all being 1, we can get the prediction for y_{T+1} as below.

```
d = 4  # number of states
set.seed(4)
trans_m = matrix(runif(16), nrow = 4)
trans_m = trans_m/t(colSums(trans_m) %*% t(rep(1, d)))
# transition matrix Lambda, lambda_ij = prob from j to i
trans_m
```

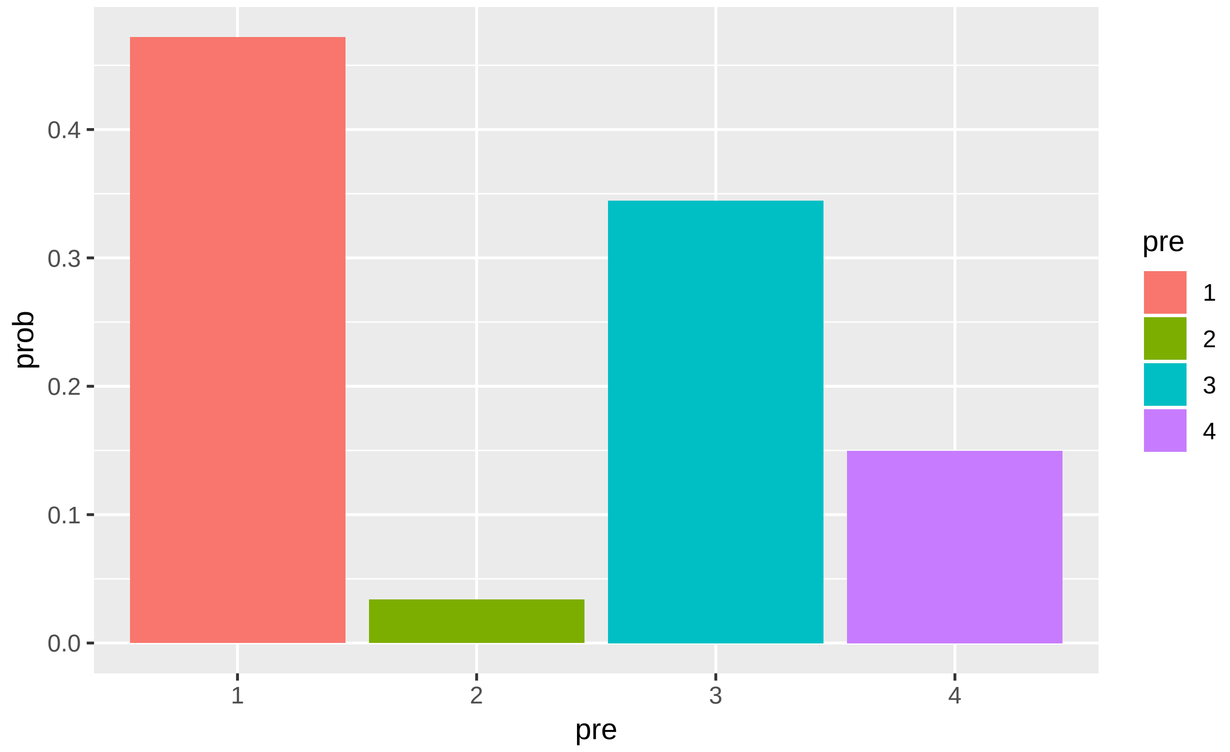
```
##             [,1]       [,2]       [,3]      [,4]
## [1,] 0.502461675 0.30082241 0.46006034 0.0519804
## [2,] 0.007673126 0.09629424 0.03545779 0.4956634
## [3,] 0.251950871 0.26785206 0.36583913 0.2159187
## [4,] 0.237914328 0.33503130 0.13864275 0.2364375
```

```
y1 = 1
l = 1500
y = rep(NA, l)
```

```
y[1] = y1
for (t in 2:l) {
    y[t] = sample(d, size = 1, prob = trans_m[, y[t - 1]])
}
# split to train and test
train = y[1:1000]
test = y[1001:1500]
# y_t = 1
train[1000]
```

```
## [1] 3
```

```
alpha = rep(1, d)  # prior
(p4 = f(train, alpha))
```



```
## y[t+1]= 1  y[t+1]= 2  y[t+1]= 3  y[t+1]= 4
## 0.47191011 0.03370787 0.34456929 0.14981273
```

(3) Define a proper scoring rule for evaluating your probabilistic forecast.

We can define a scoring rule as the following:

$$S(\mathbf{p}, j) = \sum_{j=1}^{d} (\delta_{ij} - p_j)^2$$

where $\mathbf{p} = (p_1, ..., p_d)^T, p_j$ is the probability assigned to class j, d is the number of classes and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

Actually, the scoring rule we defined for my predictive distribution is called Brier score and it's a proper scoring rule.

After defining the scoring rule, we can get a expected score for our predictive distribution.

$$\bar{S}(\mathbf{P}) = \sum_{i=1}^{d} p_i S(\mathbf{P}, i)$$

For example, for forecast test 2, we can calculate the expected scores for our two forecast tests.

```
# expected score for test 1 using Brier
sum(p10 * (1 - p10)^2)
```

```
## [1] 0.8011342
```

```
# expected score for test 1 using Brier
sum(p4 * (1 - p4)^2)
```

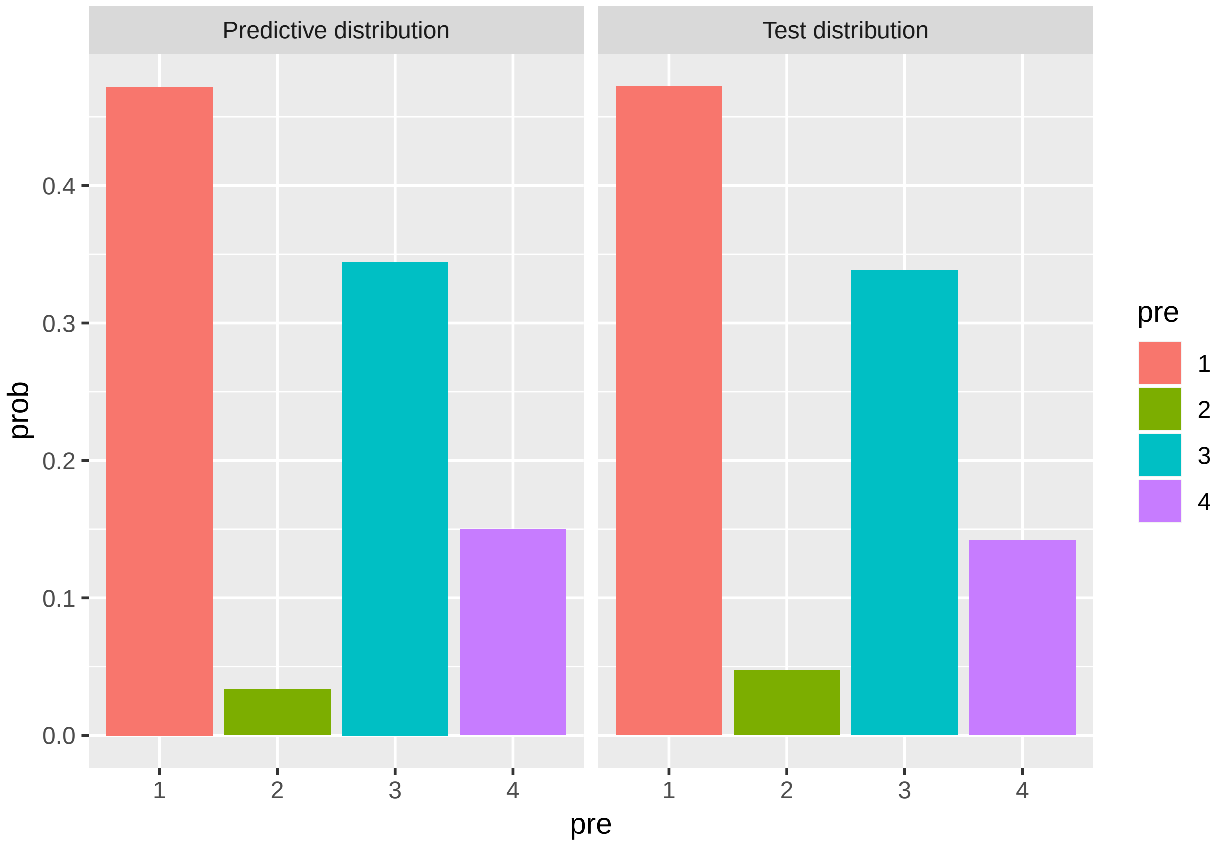
```
## [1] 0.4111187
```

(4) How could we check if it's well calibrated?

In (3), we generated some test data for our forecast test 2, and we can use the test data to check whether our prediction is well calibrated.

More specifically, we can derive the probability from state $= y_T = 3$ to each state and get a probability distribution based on test data. Then we can compare the distribution shape in our test data and that from the forecast. If their shapes match well, then we may conclude our probabilistic forecast is well calibrated.

```
d = 4
yt = 3
Nfrom1 = rep(0, 4)
for(t in 2:500){
  to = y[t]
  from = y[t-1]
  if(from == yt) {
    Nfrom1[to] = Nfrom1[to] + 1
  }
}
p_test = Nfrom1/sum(Nfrom1)
dt2 = data.frame(
  prob = c(p4, p_test), # probability
  pre = factor(rep(1:d, 2)),
  fr = rep(c('Predictive distribution', 'Test distribution'), each = 4)
)
ggplot(data = dt2, aes(x = pre, prob, fill = pre)) +
  geom_bar(position="dodge", stat="identity") +
  facet_wrap(~fr)
```



From the plot above, we can find that our predictive distibution $Pr(y_{T+1}|y_T = 3, y^{(T)}, \boldsymbol{\alpha})$ and the distribution derived from the test data are almost the same. Since the distibution of the test data does match my predictive distibution, then we may conclude that the forecast is well calibrated.