Advanced Machine Learning

Exam Training

Rasmus Holm

2017-10-16

1)

```r
library(HMM)

## Hidden variables (true positions)
states <- 1:10

transition_probs <- matrix(c(0.5, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
                             0, 0.5, 0.5, 0, 0, 0, 0, 0, 0, 0,
                             0, 0, 0.5, 0.5, 0, 0, 0, 0, 0, 0,
                             0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0,
                             0, 0, 0, 0, 0.5, 0.5, 0, 0, 0, 0,
                             0, 0, 0, 0, 0, 0.5, 0.5, 0, 0, 0,
                             0, 0, 0, 0, 0, 0, 0.5, 0.5, 0, 0,
                             0, 0, 0, 0, 0, 0, 0, 0.5, 0.5, 0,
                             0, 0, 0, 0, 0, 0, 0, 0, 0.5, 0.5,
                             0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0.5),
                           byrow=TRUE, nrow=length(states), ncol=length(states))

## Emission variables (observed positions)
symbols <- 1:10

emission_probs <- matrix(c(0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2,
                           0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0, 0.2,
                           0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0, 0,
                           0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0,
                           0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0,
                           0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0, 0,
                           0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2, 0,
                           0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2, 0.2,
                           0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2, 0.2,
                           0.2, 0.2, 0, 0, 0, 0, 0, 0.2, 0.2, 0.2),
                         byrow=TRUE, nrow=length(states), ncol=length(states))

start_probs <- rep(1, length(states)) / length(states)

robot_hmm <- initHMM(states, symbols,
                     startProbs=start_probs,
                     transProbs=transition_probs,
                     emissionProbs=emission_probs)
```

2)

```r
set.seed(123)
samples_hmm <- simHMM(robot_hmm, 10)
```

3)

```r
compute_filtered_probs <- function(hmm, observations) {
    log_probs <- HMM::forward(hmm, observations)
    probs <- prop.table(exp(log_probs), 2)
    probs
}

get_most_probable_states_by_filtered <- function(hmm, observations, states) {
    probs <- compute_filtered_probs(hmm, observations)
    most_probable_states <- as.numeric(apply(probs, 2, function(x) {
        states[which.max(x)]
    }))
    most_probable_states
}

compute_smoothed_probs <- function(hmm, observations) {
    probs <- HMM::posterior(hmm, observations)
    probs
}

get_most_probable_states_by_smoothed <- function(hmm, observations, states) {
    probs <- compute_smoothed_probs(hmm, observations)
    most_probable_states <- as.numeric(apply(probs, 2, function(x) {
        states[which.max(x)]
    }))
    most_probable_states
}

get_most_probable_path_by_viterbi <- function(hmm, observations) {
    most_probable_path <- HMM::viterbi(hmm, observations)
    most_probable_path
}

get_accuracy_filtered <- function(hmm, samples, states) {
    predicted_states <- get_most_probable_states_by_filtered(hmm, samples$observation, states)
    sum(predicted_states == samples$states) / length(predicted_states)
}

get_accuracy_smoothed <- function(hmm, samples, states) {
    predicted_states <- get_most_probable_states_by_smoothed(hmm, samples$observation, states)
    sum(predicted_states == samples$states) / length(predicted_states)
}

get_accuracy_viterbi <- function(hmm, samples, states) {
    predicted_states <- get_most_probable_path_by_viterbi(hmm, samples$observation)
    sum(predicted_states == samples$states) / length(predicted_states)
}

sample_states <- samples_hmm$states
sample_obs <- samples_hmm$observation

most_probable_states_filtered <- get_most_probable_states_by_filtered(robot_hmm, sample_obs, states)
most_probable_states_smoothed <- get_most_probable_states_by_smoothed(robot_hmm, sample_obs, states)
most_probable_path <- get_most_probable_path_by_viterbi(robot_hmm, sample_obs)
```

Forward-Backward

```
emission_density <- function(x, z) {
    return(emission_probs[z, x])
}

transition_density <- function(z, previous_z) {
    return(transition_probs[previous_z, z])
}

prior_density <- function(z) {
    return(1 / length(states))
}

myforward_backward <- function(x, states, prior_density, transition_density, emission_density) {
    alphas <- myforward(x, states, prior_density, transition_density, emission_density)
    betas <- mybackward(x, states, prior_density, transition_density, emission_density)

    list(alpha=alphas, beta=betas)
}

myforward <- function(x, states, prior_density, transition_density, emission_density) {
    T <- length(x)
    alphas <- matrix(NA, ncol=length(states), nrow=T)

    for (state in states) {
        alphas[1, state] <- emission_density(x[1], state) * prior_density(state)
    }

    for (t in 2:T) {
        inner_sum <- 0

        for (state in states) {
            inner_sum <- 0

            for (previous_state in states) {
                inner_sum <- inner_sum + alphas[t - 1, previous_state] *
                    transition_density(state, previous_state)
            }

            alphas[t, state] <- emission_density(x[t], state) * inner_sum
        }
    }

    alphas
}

mybackward <- function(x, states, prior_density, transition_density, emission_density) {
    T <- length(x)
    betas <- matrix(NA, ncol=length(states), nrow=T)
    betas[T, ] <- 1

    for (t in (T - 1):1) {
        for (state in states) {
            inner_sum <- 0

            for (next_state in states) {
                inner_sum <- inner_sum + betas[t + 1, next_state] *
                    emission_density(x[t + 1], next_state) * transition_density(next_state, state)
            }

            betas[t, state] <- inner_sum
        }
    }

    betas
}

alphabeta <- myforward_backward(x=sample_obs,
                                states = states,
                                prior_density=prior_density,
                                emission_density = emission_density,
                                transition_density = transition_density)


a <- alphabeta$alpha
b <- alphabeta$beta

filtering <- a / rowSums(a)
filtering
#>       [,1] [,2] [,3]      [,4]      [,5]      [,6]       [,7]      [,8]
#>  [1,]    0    0    0 0.2000000 0.2000000 0.2000000 0.20000000 0.2000000
#>  [2,]    0    0    0 0.1428571 0.2857143 0.2857143 0.28571429 0.0000000
#>  [3,]    0    0    0 0.1250000 0.3750000 0.5000000 0.00000000 0.0000000
#>  [4,]    0    0    0 0.0625000 0.2500000 0.4375000 0.25000000 0.0000000
#>  [5,]    0    0    0 0.0000000 0.0000000 0.4230769 0.42307692 0.1538462
#>  [6,]    0    0    0 0.0000000 0.0000000 0.2291667 0.45833333 0.3125000
#>  [7,]    0    0    0 0.0000000 0.0000000 0.1145833 0.34375000 0.3854167
#>  [8,]    0    0    0 0.0000000 0.0000000 0.0000000 0.24309392 0.3867403
#>  [9,]    0    0    0 0.0000000 0.0000000 0.0000000 0.15714286 0.4071429
#> [10,]    0    0    0 0.0000000 0.0000000 0.0000000 0.07857143 0.2821429
#>            [,9]      [,10]
#>  [1,] 0.0000000 0.00000000
#>  [2,] 0.0000000 0.00000000
#>  [3,] 0.0000000 0.00000000
#>  [4,] 0.0000000 0.00000000
#>  [5,] 0.0000000 0.00000000
#>  [6,] 0.0000000 0.00000000
#>  [7,] 0.1562500 0.00000000
#>  [8,] 0.2872928 0.08287293
#>  [9,] 0.4357143 0.00000000
#> [10,] 0.4214286 0.21785714

t(compute_filtered_probs(robot_hmm, sample_obs))
#>      states
#> index 1 2 3         4         5         6          7         8         9
#>    1  0 0 0 0.2000000 0.2000000 0.2000000 0.20000000 0.2000000 0.0000000
#>    2  0 0 0 0.1428571 0.2857143 0.2857143 0.28571429 0.0000000 0.0000000
#>    3  0 0 0 0.1250000 0.3750000 0.5000000 0.00000000 0.0000000 0.0000000
#>    4  0 0 0 0.0625000 0.2500000 0.4375000 0.25000000 0.0000000 0.0000000
#>    5  0 0 0 0.0000000 0.0000000 0.4230769 0.42307692 0.1538462 0.0000000
#>    6  0 0 0 0.0000000 0.0000000 0.2291667 0.45833333 0.3125000 0.0000000
#>    7  0 0 0 0.0000000 0.0000000 0.1145833 0.34375000 0.3854167 0.1562500
#>    8  0 0 0 0.0000000 0.0000000 0.0000000 0.24309392 0.3867403 0.2872928
#>    9  0 0 0 0.0000000 0.0000000 0.0000000 0.15714286 0.4071429 0.4357143
#>    10 0 0 0 0.0000000 0.0000000 0.0000000 0.07857143 0.2821429 0.4214286
#>      states
#> index         10
#>    1  0.00000000
#>    2  0.00000000
#>    3  0.00000000
#>    4  0.00000000
#>    5  0.00000000
#>    6  0.00000000
#>    7  0.00000000
#>    8  0.08287293
#>    9  0.00000000
#>    10 0.21785714

smoothing <- a * b / rowSums(a * b)
smoothing
#>       [,1] [,2] [,3]       [,4]      [,5]       [,6]       [,7]       [,8]
#>  [1,]    0    0    0 0.45000000 0.4107143 0.13928571 0.00000000 0.00000000
#>  [2,]    0    0    0 0.17857143 0.5428571 0.27857143 0.00000000 0.00000000
#>  [3,]    0    0    0 0.04642857 0.3964286 0.55714286 0.00000000 0.00000000
#>  [4,]    0    0    0 0.00000000 0.1857143 0.60000000 0.21428571 0.00000000
#>  [5,]    0    0    0 0.00000000 0.0000000 0.51071429 0.43214286 0.05714286
#>  [6,]    0    0    0 0.00000000 0.0000000 0.23571429 0.55000000 0.21428571
#>  [7,]    0    0    0 0.00000000 0.0000000 0.07857143 0.47142857 0.39642857
#>  [8,]    0    0    0 0.00000000 0.0000000 0.00000000 0.31428571 0.50000000
#>  [9,]    0    0    0 0.00000000 0.0000000 0.00000000 0.15714286 0.40714286
#> [10,]    0    0    0 0.00000000 0.0000000 0.00000000 0.07857143 0.28214286
#>             [,9]     [,10]
#>  [1,] 0.00000000 0.0000000
#>  [2,] 0.00000000 0.0000000
#>  [3,] 0.00000000 0.0000000
#>  [4,] 0.00000000 0.0000000
#>  [5,] 0.00000000 0.0000000
#>  [6,] 0.00000000 0.0000000
#>  [7,] 0.05357143 0.0000000
#>  [8,] 0.18571429 0.0000000
#>  [9,] 0.43571429 0.0000000
#> [10,] 0.42142857 0.2178571

t(compute_smoothed_probs(robot_hmm, sample_obs))
#>      states
#> index 1 2 3          4         5          6          7          8
#>    1  0 0 0 0.45000000 0.4107143 0.13928571 0.00000000 0.00000000
#>    2  0 0 0 0.17857143 0.5428571 0.27857143 0.00000000 0.00000000
#>     3  0 0 0 0.04642857 0.3964286 0.55714286 0.00000000 0.00000000
#>     4  0 0 0 0.00000000 0.1857143 0.60000000 0.21428571 0.00000000
#>     5  0 0 0 0.00000000 0.0000000 0.51071429 0.43214286 0.05714286
#>     6  0 0 0 0.00000000 0.0000000 0.23571429 0.55000000 0.21428571
#>     7  0 0 0 0.00000000 0.0000000 0.07857143 0.47142857 0.39642857
#>     8  0 0 0 0.00000000 0.0000000 0.00000000 0.31428571 0.50000000
#>     9  0 0 0 0.00000000 0.0000000 0.00000000 0.15714286 0.40714286
#>     10 0 0 0 0.00000000 0.0000000 0.00000000 0.07857143 0.28214286
#>      states
#> index          9        10
#>     1  0.00000000 0.0000000
#>     2  0.00000000 0.0000000
#>     3  0.00000000 0.0000000
#>     4  0.00000000 0.0000000
#>     5  0.00000000 0.0000000
#>     6  0.00000000 0.0000000
#>     7  0.05357143 0.0000000
#>     8  0.18571429 0.0000000
#>     9  0.43571429 0.0000000
#>     10 0.42142857 0.2178571
```

Viterbi

```r
myviterbi <- function(x, states, prior_density, transition_density, emission_density) {
    T <- length(x)
    weights <- matrix(NA, ncol=length(states), nrow=T)
    weights[1, ] <- log(prior_density(states)) + log(emission_density(x[1], states))
    paths <- matrix(NA, ncol=length(states), nrow=T)

    for (t in 1:(T - 1)) {
        for (state in states) {
            weights[t + 1, state] <- log(emission_density(x[t + 1], state)) +
                max(log(transition_density(state, states)) + weights[t, states])
            paths[t + 1, state] <- which.max(log(transition_density(state, states)) +
                                             weights[t, states])
        }
    }

    path <- rep(NA, T)
    path[T] <- which.max(weights[T,])

    for (t in (T - 1):1) {
        path[t] <- paths[t + 1, path[t + 1]]
    }

    path
}

myviterbi(x=sample_obs,
          states=states,
          prior_density=prior_density,
          emission_density = emission_density,
          transition_density = transition_density)
#>  [1] 4 4 4 5 6 6 6 7 7 7

HMM::viterbi(robot_hmm, sample_obs)
#>  [1] 4 4 4 5 6 6 6 7 7 7
```