732A96: Labs
Advanced Machine Learning

Carles Sans Fuentes

October 16, 2017

```

trans_probs <- diag(1/2, 10) +
  diag(1/2, 10)[, c(10, 1:9)]
emission_probs <-
  diag(1/5, 10)[, c(3:10, 1:2)] +
  diag(1/5, 10)[, c(2:10, 1)] +
  diag(1/5, 10) +
  diag(1/5, 10)[, c(10, 1:9)] +
  diag(1/5, 10)[, c(9:10, 1:8)]

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

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

transition_density2 <- function(z,previous_z){
  if( z == zt){

    return(0.5)

  } else if( (z + 1) == zt){
    return(0.5)
  } else return(0)

}

get_alpha_scalar <- function(zt, xt, previous_alpha, previous_z) {
  # Args:
  #   zt Scalar, hidden state at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  summation_term <- 0
  for (i in 1:length(previous_z)) {
    summation_term <- summation_term +
      previous_alpha[i] * transition_density(zt, previous_z[i])
  }

  alpha <- emission_density(xt, zt) * sum(summation_term)
  return(alpha)
}

get_alpha <- function(Zt, xt, previous_alpha, previous_z) {
  # Args:
  #   Zt Vector, hidden states at which to compute alpha.
  #   xt Scalar, observed state.
  #   previous_alpha Vector, alpha for all z_{t-1}.
  #   previous_z     Vector, all z_{t-1}.

  alpha <- sapply(Zt, function(zt) {
    get_alpha_scalar(zt, xt, previous_alpha, previous_z)
  })

  return(alpha)
}

get_beta_scalar <- function(zt, next_x, next_beta, next_z) {
  # Args:
  #   zt        Scalar, hidden state at which to compute alpha.
  #   next_x    Scalar, observed next state.
  #   next_beta Vector, alpha for all z_{t+1}.
  #   next_z    Vector, all z_{t+1}.
```

```r

  summation_term <- 0
  for (i in 1:length(next_z)) {
    summation_term <- summation_term +
      next_beta[i] * emission_density(next_x, next_z[i]) * transition_density(next_z[i], zt)
  }

  # P(z_(t+1) | z_t) =
  # 0.5 if z_t = z_(t+1)
  # 0.5 if z_t = z_t + 1
  # 0 otherwise


  return(summation_term)
}

get_beta <- function(Zt, next_x, next_beta, next_z) {
  # Args:
  #   Zt         Vector, hidden states at which to compute alpha.
  #   next_x     Scalar, observed next state.
  #   next_beta  Vector, alpha for all z_{t+1}.
  #   next_z     Vector, all z_{t+1}.

  beta <- sapply(Zt, function(zt) {
    get_beta_scalar(zt, next_x, next_beta, next_z)
  })

  return(beta)
}

fb_algorithm <- function(
  observations,
  emission_density,
  transition_density,
  possible_states,
  initial_density) {

  t_total <- length(observations)
  cardinality <- length(possible_states)

  # Alpha
  alpha <- matrix(NA, ncol=cardinality, nrow=t_total)

  for (i in 1:cardinality) {
    alpha[1, i] <-
      emission_density(observations[1], possible_states[i]) * initial_density[i]
  }

  for (t in 2:t_total) {
    alpha[t, ] <- get_alpha(possible_states, observations[t], alpha[t - 1, ], possible_states)
  }


  # Beta
  beta <- matrix(NA, ncol=cardinality, nrow=t_total)

  beta[t_total, ] <- 1

  for (t in (t_total - 1):1) {
    beta[t, ] <- get_beta(possible_states, observations[t + 1], beta[t + 1, ], possible_states)
  }

  return(list(alpha = alpha, beta = beta))
}

filtering <- function(alpha) {
  alpha / rowSums(alpha)
}

smoothing <- function(alpha, beta) {
  alpha * beta / rowSums(alpha * beta)
}







robotHmm <- HMM::initHMM(
  States = 1:10,
  Symbols = 1:10,
  transProbs = trans_probs,
  emissionProbs = emission_probs
)

# Create a wrapper for simHMM to assign class to the output
```

```r
simHMM <- function(hmm, length) {
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

# Simulate
nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

#debugonce(fb_algorithm)

alphabeta <- fb_algorithm(observations = robotSimultation$observation,
                          emission_density = emission_density,
                          transition_density = transition_density,
                          possible_states = 1:10,
                          initial_density = rep(0.1, 10))

filtering(alphabeta$alpha)
smoothing(alphabeta$alpha, alphabeta$beta)


plot(apply(filtering(alphabeta$alpha), 1, which.max), type = "l")
plot(apply(smoothing(alphabeta$alpha, alphabeta$beta), 1, which.max), type = "l")
lines(x = 1:100,robotSimultation$states, type = "l", col ="green")




# # # Test
# zt <- 5
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha_scalar(zt, xt, previous_alpha, previous_z)


# # Test
# zt <- 1:10
# xt <- 6
# previous_alpha <- rep(0.1, 10)
# previous_z <- 1:10
# transition_density(zt, previous_z[5])
# get_alpha(zt, xt, previous_alpha, previous_z)
#


# # Test
# Zt <- 1:10
# next_x <- 6
# next_beta <- rep(0.1, 10)
# next_z <- 1:10
# transition_density(zt, previous_z[5])
# get_beta_scalar(5, next_x, next_beta, next_z)
# get_beta(zt, next_x, next_beta, next_z)


# Define the transition, emission and initialization probabilities --------

emission_probs <- matrix(c(.2, .2, .2, 0, 0, 0, 0, 0, .2, .2,
                           .2, .2, .2, .2, 0, 0, 0, 0, 0, .2,
                           .2, .2, .2, .2, .2, 0, 0, 0, 0, 0,
                           0, .2, .2, .2, .2, .2, 0, 0, 0, 0,
                           0, 0, .2, .2, .2, .2, .2, 0, 0, 0,
                           0, 0, 0, .2, .2, .2, .2, .2, 0, 0,
                           0, 0, 0, 0, .2, .2, .2, .2, .2, 0,
                           0, 0, 0, 0, 0, .2, .2, .2, .2, .2,
                           .2, 0, 0, 0, 0, 0, .2, .2, .2, .2,
                           .2, .2, 0, 0, 0, 0, 0, .2, .2, .2), byrow=TRUE, nrow=10)

transition_probs <- matrix(c(.5, .5, 0, 0, 0, 0, 0, 0, 0, 0,
                             0, .5, .5, 0, 0, 0, 0, 0, 0, 0,
                             0, 0, .5, .5, 0, 0, 0, 0, 0, 0,
                             0, 0, 0, .5, .5, 0, 0, 0, 0, 0,
                             0, 0, 0, 0, .5, .5, 0, 0, 0, 0,
                             0, 0, 0, 0, 0, .5, .5, 0, 0, 0,
                             0, 0, 0, 0, 0, 0, .5, .5, 0, 0,
                             0, 0, 0, 0, 0, 0, 0, .5, .5, 0,
                             0, 0, 0, 0, 0, 0, 0, 0, .5, .5,
                             .5, 0, 0, 0, 0, 0, 0, 0, 0, .5), byrow=TRUE, nrow=10)

tProbDensity <- function(zt, zt_1) {
  return(transition_probs[zt_1, zt])
}

eProbDensity <- function(xt, zt) {
  return(emission_probs[zt, xt])
```

```r
}

initProbDensity <- function(z0) {
  return(dunif(z0, min=1, max=10))
}


# Simulate data ----------------------------------------------------------

library(HMM)

robotHmm <- HMM::initHMM(
  States = 1:10,
  Symbols = 1:10,
  transProbs = transition_probs,
  emissionProbs = emission_probs
)

simHMM <- function(hmm, length) {
  simulation <- HMM::simHMM(hmm, length)
  return(structure(simulation, class="HmmSimulation"))
}

nSim <- 100
robotSimultation <- simHMM(hmm=robotHmm, length=nSim)

X <- robotSimultation$observation
Z <- robotSimultation$states

# Implement Viterbi ------------------------------------------------------

possibleStates <- 1:10
get_omega <- function(Z, Omega, Z_next, x_next) {
  sapply(Z_next, function(z_next) {
    term1 <- log(eProbDensity(x_next, z_next))

    term2 <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega

    return(term1+ max(term2))
  })
}

get_phi <- function(Z, Z_next, Omega) {
  sapply(Z_next, function(z_next) {
    term <- sapply(Z, function(z) {
      log(tProbDensity(z_next, z))
    }) + Omega
    return(Z[which.max(term)])
  })
}

viterbi <- function(observations, possibleStates) {
  cardinality <- length(possibleStates)
  t_total <- length(observations)

  omega_0 <- vector("numeric", length = cardinality)
  for (i in 1:cardinality) {
    omega_0[i] <- log(initProbDensity(possibleStates[i])) + log(eProbDensity(observations[1],
        possibleStates[i]))
  }


  omega <- matrix(NA, nrow=t_total, ncol=cardinality)
  phi <- matrix(NA, nrow=t_total, ncol=cardinality)
  omega[1, ] <- omega_0

  for (i in 1:(t_total-1)) {
    omega[i+1, ] <- get_omega(possibleStates, omega[i, ], possibleStates, observations[i+1])
    phi[i+1, ] <- get_phi(possibleStates, possibleStates, omega[i, ])
  }

  mpp <- rep(NA, t_total)
  mpp[t_total] <- possibleStates[which.max(omega[t_total, ])]
  for (t in (t_total - 1):1) {
    mpp[t] <- phi[t + 1, possibleStates[mpp[t + 1]] == possibleStates]
  }

  return(list(path = mpp, omega = omega, phi = phi))

}

results <- viterbi(X, possibleStates)
results$path

results_HMM <- HMM::viterbi(robotHmm, X)
```

```
cbind(results$path, results_HMM)
```