Adaptive Systeme - Hausaufgabe 4

Henry Fock

2023-03-26

Epsilon-Greedy

```
eps_greedy <- function(means, epsilon, timesteps) {
    # Erhaltene Rewards
    rewards <- vector("numeric", timesteps)

    # Herkunft des erhaltenen Rewards
    by <- vector("integer", timesteps)

    # Zufällige Wahl der ersten Aktion
    first_action <- sample(seq_along(means), 1)
    rewards[1] <- rnorm(1, means[first_action])
    by[1] <- first_action

    for (i in seq_len(timesteps - 1)) {
        # Mittlerer Reward pro Herkunft.
        # Wurde eine Aktion noch nicht gewählt, wird ein
        #   mittlerer Reward von 0 angenommen
        current_mean_rewards <-
            tapply(rewards[1:i],
                   factor(by[1:i], levels = seq_along(means)),
                   mean,
                   default = 0)

        # Wählen der Aktion At
        # Finden der aktuellen reward-maximierenden Aktion
        max_action <- as.integer(names(which.max(current_mean_rewards)))
        # In epsilon % der Fälle wird eine zufällige Aktion gewählt
        action <-
            if (epsilon < runif(1))
                max_action
            else
                sample(seq_along(means), 1)

        # Speichern des neuen Rewards der gewählten Aktion
        # Rt mit Mittelwert q*(At)
        rewards[i + 1] <- rnorm(1, means[action])
        by[i + 1] <- action
    }
```

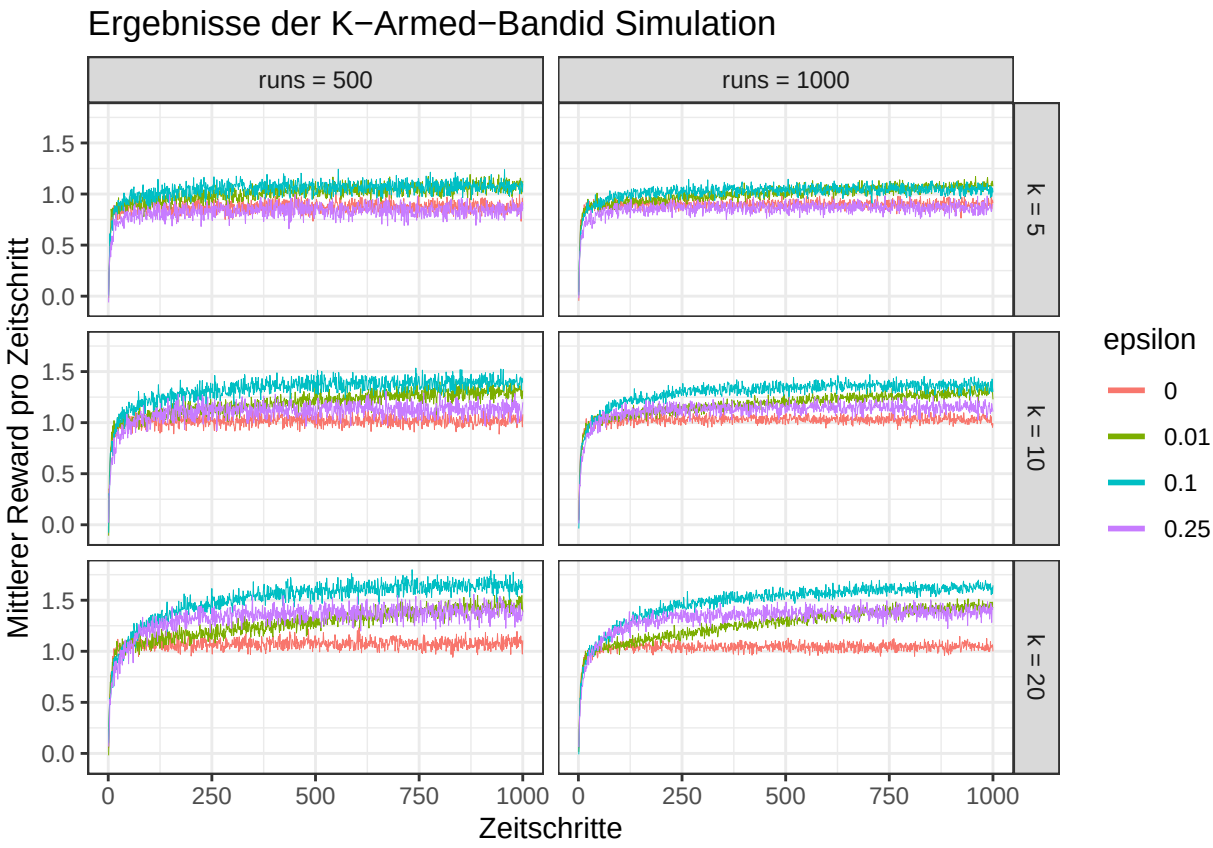
```
    # Ausgabe als Data Frame
    tibble(timestep = seq_len(timesteps),
           reward = rewards,
           by = by)
}
```

K-Armed-Bandid

```
k_armed_bandid <- function(k, epsilon, timesteps, runs) {
    # Funktion zum vordefinieren von Parametern.
    # means wird über das Mapping in die Funktion gegeben
    map_builder <- function(epsilon, timesteps) {
        function (means) {
            eps_greedy(means, epsilon, timesteps)
        }
    }

    # Erzeugen von #runs q*(a) mit jeweils k normalverteilten Zufallszahlen
    repetitions <- tibble(run = seq_len(runs),
                          means = replicate(runs, rnorm(k), simplify = F))

    # Ausführen der einzelnen Runs und anschließendem
    #   Mitteln der Rewards pro Timestep
    repetitions <- repetitions |>
        mutate(rewards = map(
            means,
            map_builder(epsilon, timesteps),
            .progress = list(clear = F)
        )) |>
        select(rewards) |>
        unnest(rewards) |>
        group_by(timestep) |>
        summarise(mean_reward = mean(reward))

    repetitions
}

# Definieren der Parameterkombinationen
# expand_grid erzeugt jede mögliche Parameterkombination
params <- expand_grid(
    k = c(5, 10. 20),
    epsilon = c(0, 0.01, 0.1, 0.25),
    timesteps = c(1000),
    runs = c(500, 1000)
)

# Ausführen des K-Armed-Bandid mit allen Parameterkombinationen
all_mean_rewards <- params %>%
    mutate(mean_rewards = pmap(
        .,
        k_armed_bandid,
```

```
        .progress = list(name = "Parameterkombination",
                         clear = F)
    ))
```

Auswertung



Runs: Die Anzahl der Runs, gezeigt in den Spalten des Plots, haben Einfluss auf die Stichprobenverteilung der Stichprobenmittelwerte. Mit zunehmender Anzahl an Runs, wird die Streuung der mittleren Rewards pro Timestep kleiner. Dies entspricht in etwa der Aussage des zentralen Grenzwertsatzes, dass mit zunehmender Stichprobengröße, sich die Verteilung der Mittelwerte an die Normalverteilung anpassen, die hier, bei der Erzeugung der Rewards, zugrundeliegt. Für 2000 Runs wären also die Linien noch dünner. Schwankungen werden immer auftreten

Epsilon: Epsilon gibt an, mit welcher Wahrscheinlichkeit eine zufällige Aktion gewählt wird, und nicht die mit dem aktuellen höchsten mittleren Reward. Für Epsilon $= 0$, wird immer nur die Aktion gewählt, die aktuell den höchsten Reward erziehlt hat. Entsprechend steigt die Rewardkurve zu beginn schnell (schneller als bei allen anderen) an, fängt sich aber nach kurzer Zeit bei einem mittleren Reward von 1. Es ist Zufall, ob sich die Simulation auf die beste Aktion einschießt oder eine weniger bessere. Steigt die Wahrscheinlichkeit auch andere aktionen auszurobieren, obwohl diese zur Zeit nicht das Maximum an Rewards liefern, erhöht sich auch die Wahrscheinlichkeit häufiger eine noch bessere Aktion zu wählen, wodurch diese ihren mittleren Reward schnell steigern kann und damit diese Aktion häufiger ausgewählt wird. Ein Epsilon von 0.1 scheint eine gute Wahl zu sein, da hier schnell ein höher Reward erziehlt wird und dieser auch lange genug ausgenutzt werden kann, um den Reward zu maximieren. Zum Vergleich, $\epsilon = 0.01$ hat eine sehr langsame Wachstumsrate und überschreitet bei $k = 20$ $\epsilon = 0.25$ erst nach > 500 Zeitschritten. Auf eine Höhe mit $k \geq 10$ gelangt es erst bei > 1000 Schritten.

Ks: Mit zunehmender Anzahl an möglicher Aktionen, steigt die diskrepanz zwischen den verschiedenen Epsilons. Das lässt sich damit erklären, dass die Wahrscheinlichkeit, ein (vergleichsweise) hohes q_* zu generieren, mit zunehmendem K steigt. Da die alle Läufe mit $\epsilon \geq 0$ diese höheren Rewards häufiger erreichen, während $\epsilon = 0$ an suboptimalen Akrionen hängen bleibt, vergrößert sich der zu erwartende Gewinn bei ersteren und bleibt im Mittel gleich bei letzterem. Dem entsprechend ist bei $k = 5$ die Wahrscheinlichkeit geringer häufig mindestens einen relativ hohen Reward zu erhalten.

Zeitschritte: Die Simulation berücksichtigt lediglich Geschehnisse aus der Vergangenheit, die Zukunft spielt keine Rolle. Daher wurden keine Ergebnisse für Zeitschritte < 1000 ausgerechnet. Möchte man ein Ergebnis für Timesteps $= 500$ sehen, kann man die hintere Hälfte der Graphen zuhalten und sich die ersten 500 Schritte angucken ;). Es wurde eine Simulation mit 2000 Zeitschritten durchgeführt (s.u.), in dem zu erkennen ist, dass $\epsilon = 0.01$ ab ca. 1250 Zeitschritten mit $\epsilon = 0.1$ gleich zieht.