Fabian Grob
Simon Michau
Til Mohr

Exercise 1

(a) Using Theorem 3.6: With $|\mathcal{H}| = 3^3 = 27$

$$
\begin{aligned}
&\Pr_{T\ \mathcal{D}^m}(\forall h \in \mathcal{H} : |err_T(h) - err_D(h)| \leq \epsilon) > 1 - \delta \\
&\Pr_{T\ \mathcal{D}^m}(\forall h \in \mathcal{H} : |err_T(h) - err_D(h)| \leq \epsilon) > 0.9 \\
&\Rightarrow \delta = 0.1
\end{aligned}
$$

$$
\begin{aligned}
m &\geq \frac{1}{2\epsilon^2} \log\left(\frac{2|\mathcal{H}|}{\delta}\right) \\
143 &\geq \frac{1}{2\epsilon^2} \log\left(\frac{2 \cdot 3^3}{0.1}\right) \\
143 &\geq \frac{1}{2\epsilon^2}(\log(54) - \log(0.1)) \\
143 &\geq \frac{1}{2\epsilon^2}(\log(54) - \log(0.1)) \\
\epsilon^2 &\geq \frac{(\log(54) - \log(0.1))}{143\dot{2}} \\
|\epsilon| &\geq \sqrt{\frac{(\log(54) - \log(0.1))}{286}} \\
\Rightarrow \epsilon &\geq \sqrt{\frac{(\log(54) - \log(0.1))}{286}}
\end{aligned}
$$

$$
\Pr_{T\ \mathcal{D}^m}(\forall h \in \mathcal{H} : |err_T(h) - err_D(h)| \leq \epsilon) > 0.9
$$

$$
\Pr_{T\ \mathcal{D}^m}\left(\forall h \in \mathcal{H} : |0.03 - err_D(h)| \leq \sqrt{\frac{(\log(54) - \log(0.1))}{286}}\right) > 0.9
$$

$$
\Rightarrow err_D(h) \leq 0.03 + \sqrt{\frac{(\log(54) - \log(0.1))}{286}} \simeq 0.208149 \simeq 0.21
$$

(b) Using Theorem 3.4:

$$
\begin{aligned}
&\Pr_{T\ \mathcal{D}^m}(\forall h \in \mathcal{H} : \text{if } h \text{ is consistent with } T, \text{ then } err_D(h) \leq \epsilon)\, 1 - \delta \\
&\Pr_{T\ \mathcal{D}^m}(\forall h \in \mathcal{H} : \text{if } h \text{ is consistent with } T, \text{ then } err_D(h) \leq 0.01)\, 0.9 \\
&\Rightarrow \epsilon = 0.01, \delta = 0.1
\end{aligned}
$$

$$
\begin{aligned}
m &\geq \frac{1}{\epsilon} \ln\left(\frac{|\mathcal{H}|}{\delta}\right) \\
m &\geq \frac{1}{0.01} \ln\left(\frac{3^3}{0.1}\right) \\
m &\geq 100(\ln(27) - \ln(0.1)) \simeq= 559.84 \\
&\Rightarrow m \geq 560
\end{aligned}
$$

Exercise 2

Exercise 3

Exercise 4

(a) See Referencesappendix for code.

```
Final probabilities: [0.4 0.4 0.2]

Tracked weight vectors:

Round:  1 Weights:  [1. 1. 1.]
Round:  2 Weights:  [0.5 0.5 1. ]
Round:  3 Weights:  [0.5  0.25 0.5 ]
Round:  4 Weights:  [0.25 0.25 0.25]
Round:  5 Weights:  [0.25  0.125 0.125]
Round:  6 Weights:  [0.125  0.125  0.0625]
Round:  7 Weights:  [0.125      0.0625     0.04419417]
```

(b) No, the weights are always the same. This is because the loss matrix does not change throughout the algorithm, so the loss is always the same. If all the events still happen in the sequence, only in a different order, this won't have an effect as the loss will still be included in the updated weight.

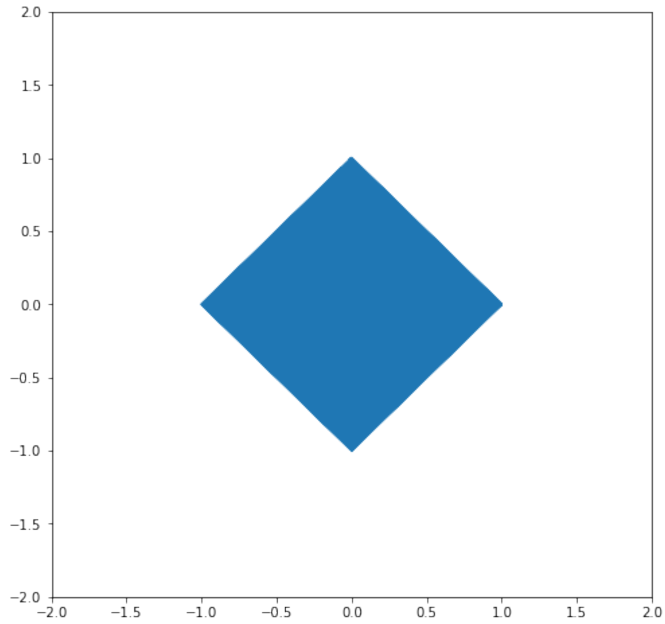
Exercise 5

(a) See Referencesappendix for code.

```
Probabilities:

Round:  1 Probabilities:  [0.33 0.33 0.33]
Round:  2 Probabilities:  [0.46 0.27 0.27]
Round:  3 Probabilities:  [0.28 0.51 0.21]
Round:  4 Probabilities:  [0.25 0.42 0.33]

Weight vectors:

Round:  1 Weights:  [1. 1. 1.]
Round:  2 Weights:  [2.83 1.   1.  ]
Round:  3 Weights:  [2.83 8.48 1.  ]
Round:  4 Weights:  [2.83 8.48 5.32]
```

(b) $w_1^{(4)}$ and $w_3^{(4)}$ are different because with $\gamma = 0.5$ we put a certain weight on exploration. Therefore, even with the same reward, different actions can have different weights as γ is part of the weight updating calculation.

Exercise 6



(a) (i)

(ii) The shape of $B_1^3 \subset \mathcal{R}^3$ would look like a octahedron with the same edge length ($\sqrt{2}$).

(b) $|diag(B_1^2)| = 2 \xrightarrow{\text{Pythagoras}}$ edge length of $B_1^2 = \sqrt{2} \Rightarrow vol(B_1^2) = \sqrt{2}^2 = 2$

$vol(B_1^3) = \frac{1}{3}\sqrt{2}a^3$ with a being the edge length of the octahedron $\Rightarrow vol(B_1^3) = \frac{1}{3}\sqrt{2}^4 = \frac{1}{3}4 = \frac{4}{3} \simeq 1.33$

A more general formula for the volume $vol(B_1^l)$ is $vol(B_1^l) = \frac{2^l}{l!}$ (cmp. this paper).
Using $l = 1, 2$ yields the same results.

(c)

$$\lim_{l \to \inf} \text{vol}(B_1^l) = \frac{2^l}{l!}$$
$$\Rightarrow \lim_{l \to \inf} \text{vol}(B_1^l) = \frac{2}{1} \cdot \frac{2}{2} \cdot \frac{2}{3} \cdot \dots \cdot \frac{2}{l-1} \cdot \frac{2}{l}$$
$$\leq 2 \cdot \frac{2}{l} = \frac{4}{l}$$
$$\Rightarrow \lim_{n \to \inf} \frac{4}{l} = 0$$

Appendix

Code for Exercise 4

```python
import numpy as np


def mwu_algorithm(loss_matrix, events, rounds, alpha):
    # initial weight vector of 1s
    weights = np.ones((loss_matrix.shape[0]))
    weights_tracking = {}
```

```
    weights_tracking[0] = weights
    # more convenient to loop through rounds and events
    rounds_arr = [i for i in range(rounds)]
    for round, event in zip(rounds_arr, events):
        # getting the current probabilities, not really needed here
        p = probabilities(weights)
        # need to use event-1 as events start at 1 but indexing at 0
        weights = np.power((1 - alpha), loss_matrix[:, event-1]) * weights
        # loss isn't really needed
        loss = calculate_loss(loss_matrix, p, event-1)
        weights_tracking[round+1] = weights

    return p, weights_tracking

def probabilities(weights):
    return weights / np.sum(weights)

def calculate_loss(loss_matrix, probabilities, event):
    return np.sum(probabilities * loss_matrix[:, event])


loss_matrix = np.array([[0,1,1,0],
                        [1,0,1,1],
                        [1,1,0,0.5]])

observed_events = [3,1,2,1,2,4]

p_6, weights_tracking = mwu_algorithm(loss_matrix, observed_events, 6, alpha
    =0.5)

print(f'Final probabilities: {p_6}\n')
print(f'Tracked weight vectors: \n')
for key, val in weights_tracking.items():
    print(f'Round:\t{key + 1}\tWeights:\t{val}')
```

Code for Exercise 5

```
import numpy as np
from copy import deepcopy

def exp3(gamma, rounds, actions, rewards):
    weights = np.ones((len(actions)))
    rounds_arr = [i for i in range(rounds)]
    n = len(actions)
    # for tracking weights and probabilities
    weights_tracking = {}
    probabilities_tracking = {}
    weights_tracking[0] = np.ones(len(actions))
    probabilities_tracking[0] = probability_dist(weights, gamma)
    for round, action in zip(rounds_arr, actions):
        probabilities = probability_dist(weights, gamma)
        probabilities_tracking[round] = probabilities
        reward = rewards[action]
        weights = update_weights(weights, reward, probabilities, action,
    gamma)
        weights_tracking[round + 1] = deepcopy(weights)

    probabilities_tracking[rounds] = probability_dist(weights, gamma)
    return weights_tracking, probabilities_tracking

def probability_dist(weights, gamma):
```

```
    return (1 - gamma) * (weights / np.sum(weights)) + gamma / len(weights)

def update_weights(weights, reward, probabilities, action, gamma):
    n = len(weights)
    # only update chosen action
    weights[action] = weights[action] * np.exp((gamma * reward) / (n * probabilities[action]))
    return weights


action_seq = np.array([ 1, 2, 3 ])
rewards = np.array([ 3, 5, 3 ]) * np.log(2)

weights, probs = exp3(gamma=0.5, rounds=3, actions=action_seq - 1, rewards=rewards)

print(f'Probabilities: \n')
for key, val in probs.items():
    print(f'Round:\t{key + 1}\tProbabilities:\t{val.round(2)}')

print(f'\nWeight vectors: \n')
for key, val in weights.items():
    print(f'Round:\t{key + 1}\tWeights:\t{val.round(2)}')
```