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

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]
```

Exercise 5

See Referencesappendix for code.

```
Probabilities:

Round: 1 Probabilities: [0.33333333 0.33333333 0.33333333]
Round: 2 Probabilities: [0.45955989 0.27022006 0.27022006]
Round: 3 Probabilities: [0.28157333 0.51113437 0.20729231]
Round: 4 Probabilities: [0.25170754 0.42160258 0.32668988]

Weight vectors:

Round: 1 Weights: [1. 1. 1.]
Round: 2 Weights: [2.82842712 1.         1.        ]
Round: 3 Weights: [2.82842712 8.47907147 1.        ]
Round: 4 Weights: [2.82842712 8.47907147 5.32231117]
```

Exercise 6

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

```python
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}')

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