Probabilistic Machine Learning

Exercise Sheet #2

due on Tuesday, 4 May 2021, 10am sharp

1. **EXAMple Question** Assume that N binary observations $X := [x_1, \ldots, x_N]$, with $x_i \in \{0; 1\}$ have been drawn independently from the Bernoulli distribution

$$p(x_i \mid f) = f^{x_i} \cdot (1 - f)^{1 - x_i} \qquad \text{for } i = 1, \ldots, N.$$

That is, $p(x = 1) = f$ and $p(x = 0) = 1 - f$ with an unknown probability $f \in [0, 1]$. As a prior for f, consider the Beta distribution with parameters $a, b \in \mathbb{R}_+$ and a normalization constant $B(a, b)$ (the Beta function),

$$p(f \mid a, b) = \mathcal{B}(f; a, b) := \frac{1}{B(a, b)} f^{a-1} \cdot (1 - f)^{b-1} \qquad \text{with } a, b > 0, \quad f \in [0, 1].$$

What is the posterior distribution $p(f \mid X)$?

2. **Theory Question: Random Variables** The normalization constant $B(a, b)$ of the Beta distribution $\mathcal{B}(f; a, b)$(see Ex. 1 above) can also be written with the *Gamma function* as

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)}.$$

(The Gamma function $\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t)\, dt$ is a continuation of the factorial function, and satisfies $\Gamma(x + 1) = x\Gamma(x)$).

   (a) The standard way to draw one Beta distributed random number is to use an existing method to draw *two* random variables X, Y with *Gamma distributions*,

$$p(X, Y) = \mathcal{G}(X; a, 1) \cdot \mathcal{G}(Y; b, 1) \quad \text{where} \quad \mathcal{G}(x; \alpha, \beta) = \frac{\beta^\alpha \cdot x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}.$$

Show that the random variable

$$Z = \frac{X}{X + Y}$$

has the pdf $p(Z = f) = \mathcal{B}(f; a, b)$.

   (b) Show that the mean of the Beta distribution is given by $\mathbb{E}_{\mathcal{B}(f;a,b)}[f] = \frac{a}{a+b}$.

3. **Practical Question** This week marks the start of a two-week project in which you get to build an autonomous agent that can play the pen-and-paper game *Battleships*. For more, refer to `Exercise_02.ipynb`

1/ Bayes' Theorem:

$$p(f \mid X) = \frac{p(X \mid f)\, p(f)}{p(X)}$$

Likelihood:

$$p(X \mid f) \overset{\text{ind}}{=} \prod_{i=1}^{N} f^{x_i} (1-f)^{1-x_i} = f^k (1-f)^{N-k} \quad , \text{ where } k = \sum_{i=1}^{N} x_i$$

$$\Rightarrow p(X \mid f)\, p(f) = \frac{1}{B(a,b)} f^{\underbrace{k+a-1}_{a'}} (1-f)^{\underbrace{N-k+b-1}_{b'}}$$

The form is clearly again like a Beta distribution (see a', b') (conjugate prior). Thus the new normalization constant is $B(a', b') = B(k+a, N-k+b)$.

$$\Rightarrow p(f \mid X) = \frac{1}{B(k+a, N-k+b)} f^{k+a-1} (1-f)^{N-k+b-1}$$

2/ This can be shown using the general transformation law. For this, we need to construct an injective function $g: \mathbb{R}^2 \to \mathbb{R}^2$ that takes (x, y) as input and (z, w) as output. To make g injective, we choose $w = x$. (Then $y = \frac{w}{z} - w$ → input can be reconstructed → injective.) The corresponding inverse function is thus:

$$g^{-1}\begin{pmatrix} z \\ w \end{pmatrix} = \begin{pmatrix} w \\ \frac{w}{z} - w \end{pmatrix}$$

$$\rightarrow \left| J_{g^{-1}}\begin{pmatrix} z \\ w \end{pmatrix} \right| = \begin{vmatrix} 0 & 1 \\ \frac{1}{z} - 1 & * \end{vmatrix} = 1 - \frac{1}{z}$$

$$\rightarrow p(z,w) = p_{x,y}\left(w, \frac{w}{z} - w\right)\left(1 - \frac{1}{z}\right)$$

($g^{-1}(z,w)$ is defined for all $z, w \in \mathbb{R}$).

$$\rightarrow p(z,w) = g(w; a, 1)\, g\left(\frac{w}{z} - w; b, 1\right)\left(1 - \frac{1}{z}\right)$$

$$= \frac{w^{a-1} e^{-w}}{\Gamma(a)} \frac{\left(\frac{w}{z} - w\right)^{b-1} e^{-\left(\frac{w}{z} - w\right)}}{\Gamma(b)} \cdot \left(1 - \frac{1}{z}\right)$$

must be positive (probability distribution), $z \in [0,1]$

$$= \frac{\overbrace{(-1)^{b-1}\left(1 - \frac{1}{z}\right)^{b}}}{\Gamma(a)\Gamma(b)} e^{-\frac{w}{z}} w^{b-1} w^{a-1}$$

$$\left(\frac{w}{z} - w\right)^{b-1} = \left(-w\left(-\frac{1}{z} + 1\right)\right)^{b-1}$$

$$= (-1)^{b-1} w^{b-1} \left(-\frac{1}{z} + 1\right)^{b-1}$$

$$e^{-w} e^{-\left(\frac{w}{z} - w\right)} = e^{-w - \frac{w}{z} + w} = e^{-\frac{w}{z}}$$

$$= \frac{\left(\frac{1}{z} - 1\right)^{b}}{\Gamma(a)\Gamma(b)} e^{-\frac{w}{z}} w^{b+a-2}$$

$$p(z) = \int_0^\infty p(z,w)\, dw = \frac{\left(\frac{1}{z} - 1\right)^{b}}{\Gamma(a)\Gamma(b)} \cdot \int_0^\infty e^{-\frac{w}{z}} w^{b+a-2}\, dw$$

↳ w is gamma-distributed.

$$\rightsquigarrow \tilde{w} := \frac{w}{z} \rightarrow \frac{d\tilde{w}}{dw} = \frac{1}{z}$$

$$= \frac{\left(\frac{1}{z} - 1\right)^{b}}{\Gamma(a)\Gamma(b)} z^{b+a-2} \cdot z \underbrace{\int_0^\infty e^{-\tilde{w}} \tilde{w}^{a+b-2}\, d\tilde{w}}$$

$$= \Gamma(a+b-1) = \frac{\Gamma(a+b)}{a+b-1}$$

$$= \underbrace{\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}}_{= \frac{1}{B(a,b)}} \cdot \underbrace{\frac{\left(\frac{1}{z} - 1\right)^{b} z^{b+a-1}}{a+b-1}}$$

$$= \frac{z^{a-1} z^{b} \left(\frac{1}{z} - 1\right)^{b}}{a+b-1} = \frac{z^{a-1}(1-z)^{b}}{a+b-1}$$

b)

$$\mathbb{E}_{B(f;a,b)}[f] = \frac{1}{B(a,b)} \int_0^1 f^{a-1}(1-f)^{b-1} \cdot f \, df$$

$$= \frac{1}{B(a,b)} \underbrace{\int_0^1 f^{a}(1-f)^{b-1} \, df}_{= B(a+1,\,b)}$$

(because it is the unnormalized beta-distribution $B(f;a+1,b)$, thus the integral over the domain must be equal to the normalization constant.)

$$= \frac{\Gamma(a+1)\,\Gamma(b)}{\Gamma(a+b+1)} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\,\Gamma(b)}$$

$$= \frac{a\,\Gamma(a) \cdot \Gamma(a+b)}{(a+b)\,\Gamma(a+b+1) \cdot \Gamma(a)}$$

$$= \frac{a}{a+b} \;//$$

Exercise_02

May 1, 2021

1 Exercise Sheet #2: Battleships (1/2)

1.0.1 Probabilistic Machine Learning

- **Lecturer**: Prof. Philipp Hennig
- **Term**: SoSe 2020
- **Due Date**: Monday, 04 May 2020, 10am

Over the course of two weeks, we will implement an agent that can play the pen-and-paper game *Battleships*. The goal of this exercise sheet is to find exact prior probabilities of getting a hit by enumeration, and to update to a posterior given observations of hits and misses. This week we will understand why we can't construct the agent with this approach.

Next week we will use Monte Carlo techniques to build an agent that we can play against :). Stay tuned!

[1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

1.0.2 2.1 A priori probability for a hit with one boat

Tasks 1. Write a function that takes the length of a boat and that returns the prior probability to observe a hit. 2. Plot the prior for a carrier (length 5).

Hint: You can find this probability by enumerating all possible positions of the boat.

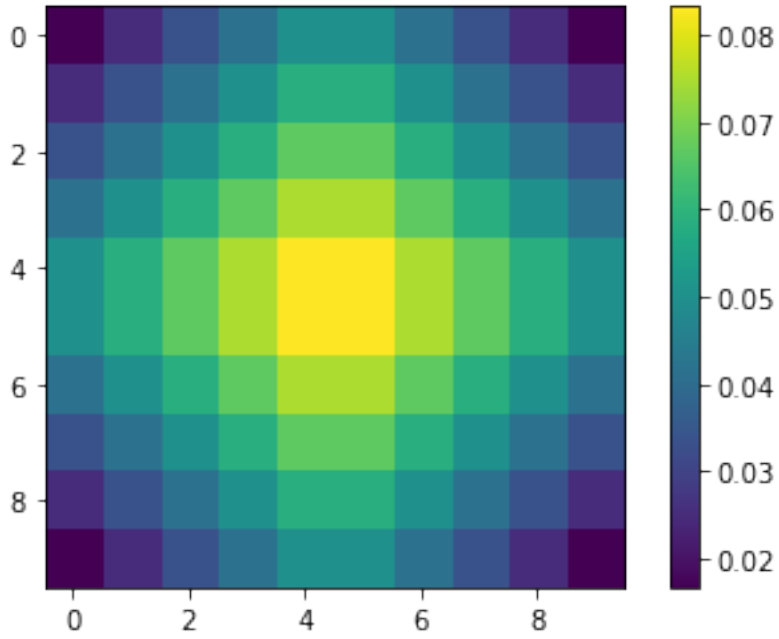
[2]:
```
field_size = 10
```

Note: Let `i` denote the *row* index and `j` the *column* index

[3]:
```
def boat_prior(boat_length, field_size):
    """
    Computes the prior probability to get a hit given the boat length and the␣
 ↪size of the board.
    For a single-boat setup only.
    :param boat_length: Length of the boat, type: int
    :param field_size: size of the board, type: int
```

```
    :returns: np.ndarray of size (field_size, field_size) containing the␣
↪probability of a hit for every field.
    """

    map_ = np.zeros((field_size, field_size))

    # Loop over all possible positions and orientations of the boat
    for x in range(field_size):
        for y in range(field_size):
            orientations = []
            if x <= field_size - boat_length:
                orientations.append('horizontal')
            if y <= field_size - boat_length:
                orientations.append('vertical')
            for o in orientations:
                d = np.hstack((np.zeros((boat_length, 1)), np.
↪arange(boat_length)[:, None])
                               [::(-1 if o == 'horizontal' else 1)])
                boat = tuple((np.array([[x, y]]) + d).astype(int).T)
                map_[boat] += 1

    return map_ / map_.sum() * boat_length

L_carrier = 5
carrier_prior = boat_prior(L_carrier, field_size)
plt.imshow(carrier_prior)
plt.colorbar()
```

[3]: <matplotlib.colorbar.Colorbar at 0x7fdbd83ab280>



[4]:
```
# Sanity check: carrier_prior should sum to the length of the boat
carrier_prior.sum()
```

[4]:
```
5.000000000000001
```

1.0.3 2.2 Updating to a posterior given hit/miss observations

Let us define an `observation_board` that contains the observations. We use the following notation:
`0` unseen field
`-1` miss (water)
`1` hit

Task 1: Write a function that takes as input the `observation_board` as well as the length of one boat, and that returns an array that contains the probability to get a hit at every coordinate of the field.

Hint: Again, this can be achieved by enumeration. Take care of all the constraints imposed by the `observation_board`!

[5]:
```
def posterior_one_ship(observation_board, boat_length):
    """
    Computes the posterior probability to get a hit given an array of observed␣
↪hits and misses as well as the boat length.
    For a single-boat setup only.
    :param observation_board: the board containing unobserved locations (0),␣
↪misses (-1), and hits (1), type: np.ndarray
    :param boat_length: Length of the boat, type: int
```

```
    :returns: np.ndarray of the same size as the observation board containing␣
↪the probability of a hit
              for every field given the observations
    """

    map_ = np.zeros((field_size, field_size))

    # Loop over all possible positions and orientations of the boat
    for x in range(field_size):
        for y in range(field_size):
            orientations = []
            if x <= field_size - boat_length:
                orientations.append('horizontal')
            if y <= field_size - boat_length:
                orientations.append('vertical')
            for o in orientations:
                d = np.hstack((np.zeros((boat_length, 1)), np.
↪arange(boat_length)[:, None])
                               [::(-1 if o == 'horizontal' else 1)])
                boat = tuple((np.array([[x, y]]) + d).astype(int).T)

                # - Check if boat position satisfies constraints -
                # Check if boat contains misses
                if -1 in observation_board[boat]:
                    continue

                # Check if board without boat contains hits
                observation_board_ = observation_board.copy()
                observation_board_[boat] = 2
                if 1 in observation_board_:
                    continue

                map_[boat] += 1

    if map_.sum() == 0:
        raise ValueError("What kind of boat is that?")
    return map_ / map_.sum() * boat_length
```

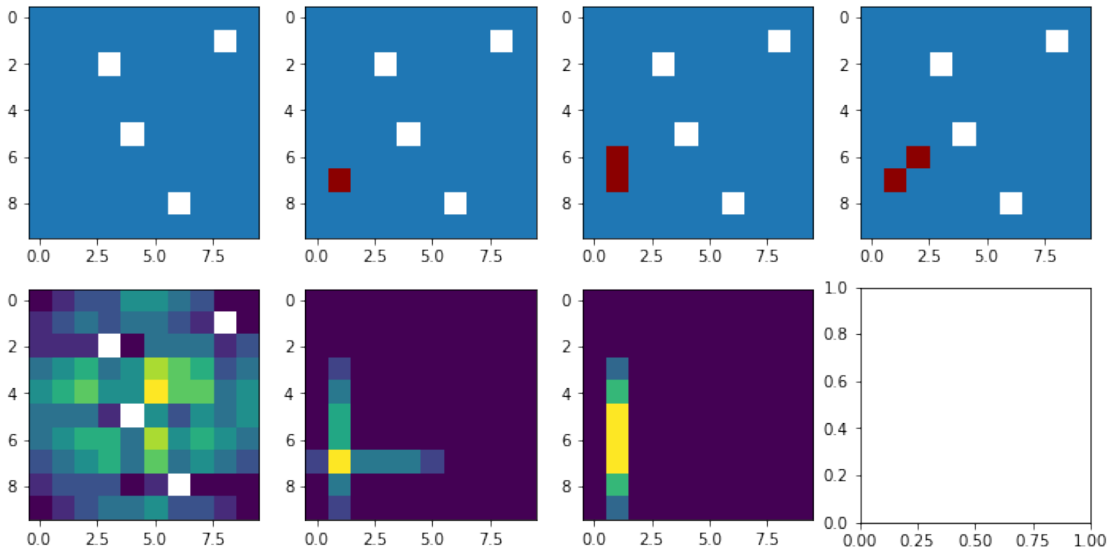
Task 2: Test your function on the following `observation_boards` (just run the cells)

[6]:
```
# board 1: Only misses
obs_board_1 = np.zeros((field_size, field_size)); obs_board_1[[1, 2, 5, 8], [8,␣
↪3, 4, 6]] = -1

# board 2: a few misses, one hit
obs_board_2 = np.copy(obs_board_1); obs_board_2[7, 1] = 1
```

```
# board 3: a few misses, two hits
obs_board_3 = np.copy(obs_board_2); obs_board_3[6, 1] = 1

# board 4: two for one boat impossible hits: This should cause an error
obs_board_4 = np.copy(obs_board_2); obs_board_4[6, 2] = 1
```

[7]:
```
### VISUALIZATION
from matplotlib import colors

# making a custom discrete colormap for hits and misses
cmap_discrete = colors.ListedColormap(['white', 'C0','darkred'])
boundaries = [-1.5, -0.5, 0.5, 1.5]
norm = colors.BoundaryNorm(boundaries, cmap_discrete.N, clip=True)

# Plot the boards and the corresponding posterior
f, axs = plt.subplots(2, 4, figsize=(12,6))

cmap = matplotlib.cm.viridis  # Can be any colormap that you want after the cm
cmap.set_bad(color='w')

# board 1
post_1 = posterior_one_ship(obs_board_1, 5)
axs[0, 0].imshow(obs_board_1, cmap=cmap_discrete, norm=norm)
axs[1, 0].imshow(np.ma.masked_where(post_1 == 0, post_1), cmap=cmap)

# board 2
axs[0, 1].imshow(obs_board_2, cmap=cmap_discrete, norm=norm)
axs[1, 1].imshow(posterior_one_ship(obs_board_2, 5))

# board 3
axs[0, 2].imshow(obs_board_3, cmap=cmap_discrete, norm=norm)
axs[1, 2].imshow(posterior_one_ship(obs_board_3, 5))

# board 4
axs[0, 3].imshow(obs_board_4, cmap=cmap_discrete, norm=norm)
try:
    axs[1, 3].imshow(posterior_one_ship(obs_board_4, 5))
except:
    pass
```

```
<ipython-input-7-945bb33682b5>:13: MatplotlibDeprecationWarning: You are
modifying the state of a globally registered colormap. In future versions, you
will not be able to modify a registered colormap in-place. To remove this
warning, you can make a copy of the colormap first. cmap =
copy.copy(mpl.cm.get_cmap("viridis"))
  cmap.set_bad(color='w')
```



[11]:
```
# board 4 should raise an error
posterior_one_ship(obs_board_4, 5)
```

```
␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-11-ea8741cfb342> in <module>
         1 # board 4 should raise an error
    ----> 2 posterior_one_ship(obs_board_4, 5)


       <ipython-input-5-1057fec84a33> in posterior_one_ship(observation_board,␣
↪boat_length)
        39
        40     if map_.sum() == 0:
    ---> 41         raise ValueError("What kind of boat is that?")
        42     return map_ / map_.sum() * boat_length


       ValueError: What kind of boat is that?
```

1.0.4 2.3 Towards battleship with more than one boat

Note: No coding required!

Next week's assignment will deal with the full game of Battleships, which, according to the above rules, contains seven ships.

1. Think about how you would need to modify your above routines to compute
   (a) the prior over ship locations
   (b) the posterior over ship locations given hit/miss observations
   by enumerating all states. Describe the changes you would need to make to your code.
2. Why will it be hard to compute the posterior with multiple boats?

1. (a) It is not enough to just sum up the individual priors, because this would ignore that two ships cannot overlap. This needs to be taken into account. The code could look something like:

```
for x1, y1, o1 in board:
    for x2, y2, o2 in board:
        if not valid:
            continue
        for x3, y3, o3 in board:
            if not valid:
                continue
            for ...
```

1. (b) Here we need to check for each possible boats configuration whether there is a conflict, i.e. a miss inside a boat or a hit inside the water. The boat configurations could be generated like above.

2. For 7 boats we have $7 \cdot 3 = 21$ nested for-loops, or 7 times looping over the board twice (horizontal, vertical). This means instead of just enumerating `2 * board_length^2` possibilities (for `board_lenght=10` this is 200), we need to loop over `(2 * board_length^2) ^ 7` possibilities (in this case $= 200^7 = 1.28 \times 10^{16}$). This is not counting the checking for validity described above, which of course also needs to loop over the whole board each time.