AOS4 : Exercise involving weighted average, value functions and Python - Solution

Hugo Martin

January 7, 2023

1. (a)

```python
import numpy as np

class WeightedAvgModel:

    def __init__(self, weights=None):
        self.weights = weights

    def predict(self, offers, weights=None):
        if weights is None:
            if self.weights is None:
                raise Exception("No weights provided.")
            weights = self.weights

        weighted_avgs = offers@weights.T
        print(weighted_avgs)
        results = np.argsort(weighted_avgs)
        return results[::-1]
```

(b)

```python
offers = np.array([
    [1.2, 3, 4],
    [2.5, 2, 2],
    [5, 2, 1],
    [7, 3, 1]])

weights = np.array([0.5, 0.25, 0.25])
model = WeightedAvgModel(weights)
print(model.predict(offers))
# [3, 2, 0, 1] (the indexes are shifted by 1 unit)
```

(c) There is nothing to do, it can be easily shown that a weighted average model will always favor O' over O, since by definition O' is element-wise superior to O and for one element strictly superior, thus the weighted average of O' is always higher than that of O.

(d)

```python
offers = np.array([
    [1.2, 3, 4],
    [2.5, 2, 2],
    [5, 2, 1],
    [7, 3, 1]])

weights = np.array([0.5, 0.25, 0.25])
model = WeightedAvgModel(weights)
print(model.predict(offers))
# [4, 3, 2, 0, 1]
```

(e) Let w_i' be the weights such that w_1''s value is maximal under the condition : $\exists i$ s.t. $G(O_i) > G(O_5)$. We define δ to be the difference between w_1 and w_1'. Then : $w_1' = w_1 - \delta$ and $w_2' = w_3' = w_2 + \frac{\delta}{2}$. We just have to compute the minimum value δ for which O_i becomes better than O_5 for $i \in \{1, 2, 3, 4\}$, and pick the lowest value. The minimum is obtained for $i = 1$, which has the minimum value δ :

$$
\begin{aligned}
& G'(O_1) > G'(O_5) \\
\iff & 1.2(0.5 - \delta) + 3(0.25 + \frac{\delta}{2}) + 4(0.25 + \frac{\delta}{2}) > 12(0.5 - \delta) + 1(0.25 + \frac{\delta}{2}) + 1(0.25 + \frac{\delta}{2}) \\
\iff & 2.35 + 2.3\delta > 6.5 - 11\delta \iff \delta > 0.31203007...
\end{aligned}
$$

We thus choose the value $\delta = 0.3121$.

```
w_1 = 0.5 - 0.3121
w_23 = (1 - w_1) / 2
weights = np.array([w_1, w_23, w_23])
model = WeightedAvgModel(weights)
print(model.predict(offers))
# [0, 4, 3, 2, 1]
```

(f)

```
offers = np.array([
    [1.2, 3, 4],
    [2.5, 2, 2],
    [5, 2, 1],
    [7, 3, 1],
    [12,1,1],
    [0,5,5]])

print(model.predict(offers))
# [5, 0, 3, 1, 4, 2]
```

(g) If we decrease w_1, O_5 will obviously stay at the top, since with a 0 salary it is already the best option. If we increase w_1, the best option will necessarily be O_4 again, as shown in a previous question (or O_5 since we only showed it for O_1, O_2, O_3, but it leads to the same problem).

2. (a) This is the law of diminishing marginal returns. 10$ to a poor person holds much more utility than the same 10$ to a rich person.

This function should be concave (also not linear).

(b) $x, a \geq 0$ so $e^{-ax} \in [0, 1]$. From this we see that $\frac{1}{1+e^{-xa}} \in [0.5, 1]$. By adding that extra linear transformation, we have that $v_1(x) \in [0, 5] \forall x \in \mathbf{R}^+, a \in \mathbf{R}^+$.

To check that v_1 concave, we can use the second-order necessary and sufficient condition for concavity; namely that the second derivative of the function has to be strictly inferior to 0 on the domain studied. After some computation, we find :

$$v_1'(x) = 10\frac{ae^{-ax}}{(1 + e^{-ax})^2}$$

The direct computation of v_1'' plus a factorisation of the numerator yields :

$$v_1''(x) = 10\frac{a^2 e^{-ax}(1+e^{-ax})(-1+e^{-ax})}{(1+e^{-ax})^4}$$

Every factor in this expression is positive on $\mathbf{R}^+$, except for $-1+e^{-ax}$, which is non-positive since $e^{-ax} \in (0,1]$ with $x \geq 0$ and $a > 0$. We then know that $v_1''(x) \leq 0$ when $x \geq 0$, thus this function is concave on the domain where $x \geq 0$.

(c)

```
def modified_sigmoid(x, a):
    return 10*( (1 / (1 + np.exp(-a*x))) - 0.5)

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
space = np.arange(0,10,0.001)
v = [[modified_sigmoid(x,a) for x in space] for a in [0.1,0.4,1.4,10]]
plt.plot(space, v[0], color='r')
plt.plot(space, v[1], color='g')
plt.plot(space, v[2], color='b')
plt.plot(space, v[3], color='y')
plt.xlim([0,10])
plt.show()
```

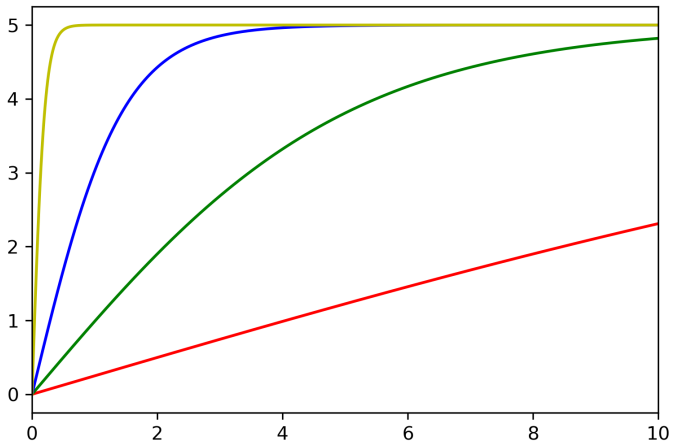

Figure 1: Plot of the modified sigmoid for $a = 0.1$ (red), $a = 0.4$ (green), $a = 1.4$ (blue), $a = 10$ (yellow)

a controls the initial steepness of the function, that is how fast it reaches saturation.

(d) 1.4 seems to be a good value for a in this case.

```
offers = np.array([
    [1.2, 3, 4],
    [2.5, 2, 2],
    [5, 2, 1],
    [7, 3, 1],
    [12,1,1],
    [0,5,5]])

weights = np.array([0.5, 0.25, 0.25])
model = WeightedAvgModel(weights)
```

```
offers_modified = np.copy(offers)
offers_modified[:,0] = modified_sigmoid(offers[:,0], 1.4)
print(model.predict(offers_modified))
# [3, 0, 1, 2, 4, 5]
```