c1-tp-mab

September 22, 2024

1 Introduction à l'apprentissage par renforcement

2 TP 1 - les manchots multi-bras

1/4 de la note finale est liée à la mise en forme :

- pensez à nettoyer les outputs inutiles (installation, messages de débuggage, …)
- soignez vos figures : les axes sont-ils faciles à comprendre ? L'échelle est adaptée ?
- commentez vos résultats : vous attendiez-vous à les avoir ? Est-ce étonnant ? Faites le lien avec la théorie.

Ce TP reprend l'exemple d'un médecin et de ses vaccins. Vous allez comparer plusieurs stratégies et trouver celle optimale. Un TP se fait en groupe de 2 à 4. Aucun groupe de plus de 4 personnes.

Vous allez rendre le TP dans une archive ZIP. L'archive ZIP contient ce notebook au format `ipynb`, mais aussi exporté en PDF & HTML. L'archive ZIP doit aussi contenir un fichier txt appelé `groupe.txt` sous le format:

```
Nom1, Prenom1, Email1, NumEtudiant1
Nom2, Prenom2, Email2, NumEtudiant2
Nom3, Prenom3, Email3, NumEtudiant3
Nom4, Prenom4, Email4, NumEtudiant4
```

Un script vient extraire vos réponses : ne changez pas l'ordre des cellules et soyez sûrs que les graphes sont bien présents dans la version notebook soumise.

[50]:
```
! pip install matplotlib tqdm numpy ipympl opencv-python torch tqdm
!jupyter labextension install @jupyter-widgets/jupyterlab-manager
!jupyter labextension install jupyter-matplotlib
```

```
Requirement already satisfied: matplotlib in ./.venv/lib/python3.10/site-
packages (3.9.2)
Requirement already satisfied: tqdm in ./.venv/lib/python3.10/site-packages
(4.66.5)
Requirement already satisfied: numpy in ./.venv/lib/python3.10/site-packages
(2.1.1)
Requirement already satisfied: ipympl in ./.venv/lib/python3.10/site-packages
(0.9.4)
Requirement already satisfied: opencv-python in ./.venv/lib/python3.10/site-
packages (4.10.0.84)
Requirement already satisfied: torch in ./.venv/lib/python3.10/site-packages
```

```
(2.4.1)
Requirement already satisfied: kiwisolver>=1.3.1 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (1.4.7)
Requirement already satisfied: python-dateutil>=2.7 in
./.venv/lib/python3.10/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: packaging>=20.0 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (3.1.4)
Requirement already satisfied: fonttools>=4.22.0 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (4.53.1)
Requirement already satisfied: contourpy>=1.0.1 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (1.3.0)
Requirement already satisfied: pillow>=8 in ./.venv/lib/python3.10/site-packages
(from matplotlib) (10.4.0)
Requirement already satisfied: cycler>=0.10 in ./.venv/lib/python3.10/site-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: ipython<9 in ./.venv/lib/python3.10/site-packages
(from ipympl) (8.27.0)
Requirement already satisfied: traitlets<6 in ./.venv/lib/python3.10/site-
packages (from ipympl) (5.14.3)
Requirement already satisfied: ipywidgets<9,>=7.6.0 in
./.venv/lib/python3.10/site-packages (from ipympl) (8.1.5)
Requirement already satisfied: ipython-genutils in ./.venv/lib/python3.10/site-
packages (from ipympl) (0.2.0)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
./.venv/lib/python3.10/site-packages (from torch) (12.1.0.106)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
./.venv/lib/python3.10/site-packages (from torch) (12.1.105)
Requirement already satisfied: filelock in ./.venv/lib/python3.10/site-packages
(from torch) (3.16.0)
Requirement already satisfied: sympy in ./.venv/lib/python3.10/site-packages
(from torch) (1.13.2)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
./.venv/lib/python3.10/site-packages (from torch) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
./.venv/lib/python3.10/site-packages (from torch) (11.0.2.54)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
./.venv/lib/python3.10/site-packages (from torch) (11.4.5.107)
Requirement already satisfied: nvidia-nccl-cu12==2.20.5 in
./.venv/lib/python3.10/site-packages (from torch) (2.20.5)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in
./.venv/lib/python3.10/site-packages (from torch) (9.1.0.70)
Requirement already satisfied: fsspec in ./.venv/lib/python3.10/site-packages
(from torch) (2024.9.0)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
./.venv/lib/python3.10/site-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
```

```
./.venv/lib/python3.10/site-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
./.venv/lib/python3.10/site-packages (from torch) (12.1.105)
Requirement already satisfied: triton==3.0.0 in ./.venv/lib/python3.10/site-
packages (from torch) (3.0.0)
Requirement already satisfied: jinja2 in ./.venv/lib/python3.10/site-packages
(from torch) (3.1.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
./.venv/lib/python3.10/site-packages (from torch) (10.3.2.106)
Requirement already satisfied: networkx in ./.venv/lib/python3.10/site-packages
(from torch) (3.3)
Requirement already satisfied: typing-extensions>=4.8.0 in
./.venv/lib/python3.10/site-packages (from torch) (4.12.2)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
./.venv/lib/python3.10/site-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch) (12.6.68)
Requirement already satisfied: exceptiongroup in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (1.2.2)
Requirement already satisfied: stack-data in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (0.6.3)
Requirement already satisfied: pygments>=2.4.0 in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (2.18.0)
Requirement already satisfied: decorator in ./.venv/lib/python3.10/site-packages
(from ipython<9->ipympl) (5.1.1)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in
./.venv/lib/python3.10/site-packages (from ipython<9->ipympl) (3.0.47)
Requirement already satisfied: jedi>=0.16 in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (0.19.1)
Requirement already satisfied: pexpect>4.3 in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (4.9.0)
Requirement already satisfied: matplotlib-inline in ./.venv/lib/python3.10/site-
packages (from ipython<9->ipympl) (0.1.7)
Requirement already satisfied: widgetsnbextension~=4.0.12 in
./.venv/lib/python3.10/site-packages (from ipywidgets<9,>=7.6.0->ipympl)
(4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in
./.venv/lib/python3.10/site-packages (from ipywidgets<9,>=7.6.0->ipympl)
(3.0.13)
Requirement already satisfied: comm>=0.1.3 in ./.venv/lib/python3.10/site-
packages (from ipywidgets<9,>=7.6.0->ipympl) (0.2.2)
Requirement already satisfied: six>=1.5 in ./.venv/lib/python3.10/site-packages
(from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in ./.venv/lib/python3.10/site-
packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
./.venv/lib/python3.10/site-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
./.venv/lib/python3.10/site-packages (from jedi>=0.16->ipython<9->ipympl)
```

```
(0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in ./.venv/lib/python3.10/site-
packages (from pexpect>4.3->ipython<9->ipympl) (0.7.0)
Requirement already satisfied: wcwidth in ./.venv/lib/python3.10/site-packages
(from prompt-toolkit<3.1.0,>=3.0.41->ipython<9->ipympl) (0.2.13)
Requirement already satisfied: executing>=1.2.0 in ./.venv/lib/python3.10/site-
packages (from stack-data->ipython<9->ipympl) (2.1.0)
Requirement already satisfied: asttokens>=2.1.0 in ./.venv/lib/python3.10/site-
packages (from stack-data->ipython<9->ipympl) (2.4.1)
Requirement already satisfied: pure-eval in ./.venv/lib/python3.10/site-packages
(from stack-data->ipython<9->ipympl) (0.2.3)

[notice] A new release of pip is
available: 23.0.1 -> 24.2
[notice] To update, run:
pip install --upgrade pip
usage: jupyter [-h] [--version] [--config-dir] [--data-dir] [--runtime-dir]
               [--paths] [--json] [--debug]
               [subcommand]

Jupyter: Interactive Computing

positional arguments:
  subcommand     the subcommand to launch

options:
  -h, --help     show this help message and exit
  --version      show the versions of core jupyter packages and exit
  --config-dir   show Jupyter config dir
  --data-dir     show Jupyter data dir
  --runtime-dir  show Jupyter runtime dir
  --paths        show all Jupyter paths. Add --json for machine-readable
                 format.
  --json         output paths as machine-readable json
  --debug        output debug information about paths

Available subcommands: kernel kernelspec migrate run troubleshoot

Jupyter command `jupyter-labextension` not found.
usage: jupyter [-h] [--version] [--config-dir] [--data-dir] [--runtime-dir]
               [--paths] [--json] [--debug]
               [subcommand]

Jupyter: Interactive Computing

positional arguments:
  subcommand     the subcommand to launch
```

```
options:
  -h, --help     show this help message and exit
  --version      show the versions of core jupyter packages and exit
  --config-dir   show Jupyter config dir
  --data-dir     show Jupyter data dir
  --runtime-dir  show Jupyter runtime dir
  --paths        show all Jupyter paths. Add --json for machine-readable
                 format.
  --json         output paths as machine-readable json
  --debug        output debug information about paths

Available subcommands: kernel kernelspec migrate run troubleshoot

Jupyter command `jupyter-labextension` not found.
```

[51]:
```ipython
%load_ext autoreload
%autoreload 2
%matplotlib inline

from tqdm import tqdm
from typing import List, Tuple
import typing as t
import math
import torch
import numpy as np
from tqdm.auto import trange, tqdm
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import matplotlib.animation as animation
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
import cv2
from IPython.display import display, clear_output
import random


torch.random.manual_seed(0)

K = 5 # num arms
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

3 Présentation du problème

[3]:
```
import torch.testing


class ArmBernoulli:
    def __init__(self, p: float):
        """
        Vaccine treatment following a Bernoulli law (mean is p and variance is␣
 ↪p(1-p)
        Args:
             p (float): mean parameter

        >>> torch.random.manual_seed(random_state)
        >>> arm = ArmBernoulli(0.5)
        >>> arm.sample(5)
        tensor([ True, False,  True,  True,  True])
        """
        self.immunity_rate = p

    def sample(self, n: int = 1) -> torch.Tensor:
        return torch.rand(n) < self.immunity_rate

    def __repr__(self):
        return f'<ArmBernoulli p={self.immunity_rate}'

def generate_arms(num_arms: int):
    means = torch.rand(num_arms)
    # print("means =", means)
    MAB = [ArmBernoulli(m) for m in means]
    assert MAB[0].immunity_rate == means[0]
    assert (MAB[0].sample(10) <= 1).all() and (MAB[0].sample(10) >= 0).all()
    return MAB

MAB = generate_arms(K)
```

Ce TP reprend l'exemple du médecin présenté en cours.

Q1. Créez une fonction pour trouver μ^* à partir d'un `MAB`. Comment est définie la récompense R_k ? Que représente concrètement le regret dans cet exemple ?

[4]:
```
def find_mean_reward_from_best_vaccin(MAB : List[ArmBernoulli],␣
 ↪hide_immunity_rate = False) -> float:

    N = 1000
    means : List[float]

    if hide_immunity_rate:
```

```
        means = [torch.mean(arm.sample(N), dtype=float) for arm in MAB]
    else:
        means = [arm.immunity_rate for arm in MAB]

    print("means =", means)
    mu_star : float = max(means).item()
    print("mu_star =", mu_star)
    return mu_star

def apply_vaccin(arm : ArmBernoulli) -> int:
    """Rk

    Returns 1 if the patient is immunised thanks to the vaccin and 0 if not"""
    return 1 if arm.sample(1).item() else 0

def compute_regret(MAB : List[ArmBernoulli], N : int) -> float:
    mu_star = find_mean_reward_from_best_vaccin(MAB, False)
    # rn = N * mu_star -
```

[5]:
```
find_mean_reward_from_best_vaccin(MAB)
```

```
means = [tensor(0.4963), tensor(0.7682), tensor(0.0885), tensor(0.1320),
tensor(0.3074)]
mu_star = 0.7682217955589294
```

[5]:
```
0.7682217955589294
```

[Ajoutez votre commentaire ici]

Que représente concrètement le regret dans cet exemple ?

[Réponse] Le regret représente le nombre de patient qui aurait pu être immunisé si le meilleur avait été utilisé à chaque fois.

Note importante : pour la suite, les tests seront faits avec 10 MAB différents ou plus pour réduire le bruit de simulation. Concrètement, on exécutera au moins 10x `generate_arms`.

4 I. Cas classique des bandits manchots

4.1 I.a. Solution Gloutonne

Le médecin fonctionne sur deux phases :

1. **Exploration :** N patients reçoivent une dose d'un vaccin choisi aléatoirement. Le médecin calcule le taux d'immunisation empirique :

$$\bar{R}_i = \frac{1}{T_i} \sum_{k=0}^{N-1} \chi_{v_k,i} R_k,$$

avec $T_i = \sum_{k=0}^{N-1} \chi_{v_k,i}$.

2. **Exploitation :** Le vaccin $v_i = \arg\max_j \bar{R}_j$ est utilisé pour les M patients suivants. C'est la phase de test.

Q2. Implémentez la solution gloutonne avec N = 50 et M = 500 et testez la avec 100 MAB différents (tous ont 5 vaccins). On s'intéresse à la variable aléatoire "la phase d'exploration a trouvé le bon vaccin". Quelle est l'espérance empirique de cette variable ? Et son écart-type ? Calculez de même l'espérance et l'écart-type du regret sur vos 100 simulations.

Pour rappel, le regret est défini par :

$$r_n = n\mu^* - \sum_{k=0}^{n-1} R_k$$

Attention : n est le nombre total de patients, donc ici $N + M$.

[6]:
```
MABs = [generate_arms(5) for _ in range(100)]
```

[52]:
```
N = 50
M = 500
log = print if False else (lambda *_, **_2 : None)

def compute_empirical_immuniation_rate(vaccins_used : torch.Tensor, Rk : torch.
 ↪Tensor, nb_vaccin : int) -> torch.Tensor:

    n_patients : int = vaccins_used.shape[0]

    vaccins_used = vaccins_used.to(torch.int)
    log("vaccins_used =", vaccins_used)

    log("Rk =", Rk)
    log("Rk shape =", Rk.shape)

    n_patient = vaccins_used.shape[0]
    assert(Rk.shape[0] == n_patient)

    # compute the number of times each vaccin has been used
    T = torch.bincount(vaccins_used)
    log("T =", T)

    # Compute the indicator function
    X = torch.zeros((n_patients, nb_vaccin))
    X[torch.arange(n_patients), vaccins_used] = 1
    log("X shape =", X.shape)
    # log("X :", X)
```

```
    Ris = torch.Tensor([1 / T[i] * (X[:,i] @ Rk) for i in range(nb_vaccin)]).
 ↪flatten()
    log(Ris)

    return Ris


def exploration(MAB : List[ArmBernoulli], n_patients : int) -> tuple[torch.
 ↪tensor, torch.tensor]:

    nb_vaccin = len(MAB)

    log("Arms immunity rate :", [arm.immunity_rate for arm in MAB])

    # Vaccin used for all patients
    vaccins_used = torch.randint(0, nb_vaccin, (n_patients,))

    # apply vaccins
    Rk = torch.cat([MAB[vaccins_used[k]].sample(1) for k in range(n_patients)]).
 ↪flatten().to(torch.float)

    return vaccins_used, Rk

def compute_regret(n_success_exploration_vaccin : int, MAB :␣
 ↪List[ArmBernoulli], real_best_vaccin : int, n_patients : int):

    max_vaccin = (n_patients * MAB[real_best_vaccin].immunity_rate).item()
    log("max_vaccin :", max_vaccin)

    return max_vaccin - n_success_exploration_vaccin

def exploitation_no_update(MAB : List[ArmBernoulli], exploration_best_vaccin :␣
 ↪int, n_patients : int) -> int:
    """
    Returns:
        float: Number of patients that have been immune by a vaccin
    """

    # apply vaccin in exploitation
    n_success_exploration_vaccin = MAB[exploration_best_vaccin].
 ↪sample(n_patients).sum().item()
    log("exploration_vaccin :", n_success_exploration_vaccin)

    return n_success_exploration_vaccin
```

```python
def simulation(MAB : List[ArmBernoulli], n_patients_exploration : int,␣
 ↪n_patients_exploitation : int) -> Tuple[bool, float]:
    """
    Returns:
        Tuple[bool, float]: Is vaccin found by exploration then TRUE | Regret␣
 ↪of the exploitation
    """

    vaccins_used, Rk = exploration(MAB, n_patients_exploration)

    Ris = compute_empirical_immuniation_rate(vaccins_used=vaccins_used, Rk=Rk,␣
 ↪nb_vaccin=len(MAB))
    exploration_best_vaccin = Ris.argmax()
    log("exploration best vaccin :", exploration_best_vaccin)

    real_best_vaccin = np.array([arm.immunity_rate for arm in MAB]).argmax()
    log("real best vaccin :", real_best_vaccin)

    n_success_exploration_vaccin = exploitation_no_update(MAB,␣
 ↪exploration_best_vaccin, n_patients=n_patients_exploitation)

    regret = compute_regret(n_success_exploration_vaccin, MAB,␣
 ↪real_best_vaccin, n_patients_exploitation)
    log("Regret :", regret)

    return real_best_vaccin == exploration_best_vaccin, regret

def print_results(samples_exploration_found_good_vaccin : torch.Tensor,␣
 ↪exploitation_regrets : torch.Tensor) -> None:

    if samples_exploration_found_good_vaccin is not None:
        print("E_exploration_found_good_vaccin :",␣
 ↪samples_exploration_found_good_vaccin.mean())
        print("std_exploration_found_good_vaccin :",␣
 ↪samples_exploration_found_good_vaccin.std())

    print("E_exploitation_regrets :", exploitation_regrets.mean())
    std_exploitation_regrets = exploitation_regrets.std()
    print("std_exploitation_regrets :", std_exploitation_regrets)
    print("std_exploitation_regrets rate :", std_exploitation_regrets / M)

if False: # test with one simulation
    i = torch.randint(0, 100, (1, )).item()
    print("index :", i)
    simulation(MABs[i], N, M)
```

```
if True: # test with all the simulations

    l = torch.Tensor([simulation(MAB, N, M) for MAB in tqdm(MABs)])

    samples_exploration_found_good_vaccin = l[:, 0]
    exploitation_regrets = l[:, 1]

    print_results(samples_exploration_found_good_vaccin, exploitation_regrets)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
E_exploration_found_good_vaccin : tensor(0.7000)
std_exploration_found_good_vaccin : tensor(0.4606)
E_exploitation_regrets : tensor(12.1021)
std_exploitation_regrets : tensor(31.6172)
std_exploitation_regrets rate : tensor(0.0632)
```

Q3. On propose d'améliorer l'algorithme précédant en mettant à jour les taux d'immunisation empiriques $\bar{R}_i$ pendant l'exploitation. Notez vous une amélioration du regret ? Proposez un exemple dans lequel cette mise à jour ne changera rien.

[53]:
```
def exploitation_update_immune_rate(MAB : List[ArmBernoulli], vaccins_used :␣
↪torch.Tensor, Rk : torch.Tensor, n_patients : int) -> float:
    """
    Returns:
        float: Number of patients that have been immune by a vaccin
    """

    sum = 0
    for _ in range(n_patients):

        # Choose the vaccin considered as the best one
        Ris = compute_empirical_immuniation_rate(vaccins_used=vaccins_used,␣
↪Rk=Rk, nb_vaccin=len(MAB))
        best_vaccin = Ris.argmax().to(torch.int)
        log("exploration best vaccin :", best_vaccin)

        # apply vaccin
        success = MAB[best_vaccin].sample(1)
        sum += success.item()

        # update vaccins_used and Rk
        vaccins_used = torch.cat((vaccins_used, torch.Tensor([best_vaccin])))
        Rk = torch.cat((Rk, success))

    return sum
```

```python
def simulation(MAB : List[ArmBernoulli], n_patients_exploration : int,␣
↪n_patients_exploitation : int) -> float:
    """
    Returns:
        float: Regret of the exploitation
    """

    # exploration
    vaccins_used, Rk = exploration(MAB, n_patients_exploration)

    # exploitation
    n_success_exploration_vaccin = exploitation_update_immune_rate(MAB,␣
↪vaccins_used, Rk, n_patients_exploitation)

    # compute regret
    real_best_vaccin = np.array([arm.immunity_rate for arm in MAB]).argmax()
    log("real best vaccin :", real_best_vaccin)

    regret = compute_regret(n_success_exploration_vaccin, MAB,␣
↪real_best_vaccin, n_patients_exploitation)
    log("Regret :", regret)

    return regret

if True: # test with all the simulations

    exploitation_regrets = torch.Tensor([simulation(MAB, N, M) for MAB in␣
↪tqdm(MABs)])

    print_results(None, exploitation_regrets)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
E_exploitation_regrets : tensor(3.3921)
std_exploitation_regrets : tensor(14.1552)
std_exploitation_regrets rate : tensor(0.0283)
```

[Ajoutez votre commentaire ici]

Cette mise à jour ne sert à rien lorsque le meilleur vaccin a été trouvé lors de l'exploration et que mettre à jour les taux d'immunsation des vaccins ne fait pas changer le vaccin qui est considéré comme le plus performant.

Q4. Créez une figure contenant deux sous-figures : à gauche, le taux d'immunisation empirique $\bar{R}_i$ pour les 5 vaccins ; à droite, le regret r_n. La figure sera animée avec les patients : chaque frame k de l'animation représente le vaccin que l'on donne au k-ième patient.

[ ]:

[Ajoutez votre commentaire ici]

Q5. On étudie maintenant l'influence de la taille du training set N. On considère que N+M est une constante, puis on fait varier N entre K et M. Calculez le regret pour ces différentes tailles du training set différents MAB et representez le regret moyen, le regret min et max (vous devriez trouver une courbe en U ou en V pour le regret moyen). Quelle est la taille optimale du training set ?

[ ]:

[Ajoutez votre commentaire ici]

4.2 I.b. Borne inférieure de Lai & Robbins [Lai et Robbins, 1985]

Pour un modèle de manchot de Bernoulli (équivalent au problème étudié), la borne inférieure de Lai et Robbins [Lai et Robbins, 1985] stipule que :

$$\liminf_{n\to\infty} \frac{\sum_{k=0}^{n-1} R_k}{\log n} \geq \sum_{i \text{ tel que } \mu_i<\mu^*} \frac{\mu^* - \mu_i}{\mathrm{KL}(\mu_i, \mu^*)} := C(\mu)$$

avec $\mathrm{KL}(x, y) = x\log(x/y) + (1-x)\log((1-x)/(1-y))$ (distance de Kullback-Leibler) et $\sum_{k=0}^{n-1} R_k$ la récompense obtenue sur n patients (avec un algorithme optimal).

Q6. Justifiez pourquoi cette borne signifie que la machine optimale est jouée exponentiellement plus souvent que les autres machines.

[Ajoutez votre commentaire ici]

Q7. Tracez le regret issu de la borne de Lai & Robbins et comparez le au regret obtenu avec l'algorithme glouton.

[ ]:

[Ajoutez votre commentaire ici]

4.3 I.c. Upper Confidence Bounds

Cet algorithme améliore la version précédente en ajoutant un biais lié à la fréquentation de chaque vaccin :

$$\hat{R}_i = \bar{R}_i + \sqrt{\frac{C\log n}{T_i}}$$

,

avec $C = 2$.

Q8. Implémentez la modification de cette algorithme. Conservez les deux phases exploration/exploitation décrites ci-dessus. En prenant les valeurs de N et M trouvées à la question Q5, quel regret obtenez-vous ? Faites l'expérience avec au moins 10 MAB différents (tous ayant 5 vaccins) afin de calculer la moyenne et l'écart-type du regret.

[ ]:

[Ajoutez votre commentaire ici]

Q9. Reprenez la questions Q4 avec cette algorithme. Dans la figure de gauche, vous representerez $\bar{R}_i$ et $\hat{R}_i$.

[ ]:

[Ajoutez votre commentaire ici]

Q10. Reprenez la question Q5 avec cette algorithme. Concluez sur l'utilité (ou l'inutilité) de la phase d'exploration. Comparez les performances d'UCB avec celles de l'algorithme glouton.

[ ]:

[Ajoutez votre commentaire ici]

Q11. Testez différentes valeurs pour C et trouvez sa valeur optimale expérimentalement.

[ ]:

[Ajoutez votre commentaire ici]

5 Echantillonnage de Thomson

Cet algorithme propose de modéliser la variable aléatoire de chaque vaccin avec une loi β dont les paramètres a et b correspondent au nombre de patients que le vaccin a immunisés (resp. non immunisés).

Pour chaque patient, on tire un valeur aléatoire pour la loi β décrivant chaque vaccin, puis on choisit le vaccin avec la plus grande valeur tirée.

Q12. Implémentez cet algorithme. Conservez les deux phases exploration/exploitation décrites ci-dessus. En prenant les valeurs de N et M trouvées à la question Q5, quel regret obtenez-vous ? Faites l'expérience avec au moins 10 MAB différents (tous ayant 5 vaccins) afin de calculer la moyenne et l'écart-type du regret.

[ ]:

[Ajoutez votre commentaire ici]

Q13. Reprenez la question Q4, mais cette fois-ci, vous representerez le taux d'immunisation empirique avec un graphique en violon qui représente la loi beta associée à chaque vaccin.

[ ]:

[Ajoutez votre commentaire ici]

Q14. Représentez son regret pour différentes tailles du training set (comme dans la Q5). Comparez le regret avec les autres algorithmes.

[ ]:

[Ajoutez votre commentaire ici]

6 Conclusion

Q15. Calculez le regret des algorithmes glouton, UCB & Thomson lorsqu'il y a un grand nombre de vaccins disponibles (K=100) (on prendra N=100). Faites le lien avec la malédiction de la dimension.

[ ]:

[Ajoutez votre commentaire ici]