МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №4**

по дисциплине«Методы машинного обучения»

Тема: «Алгоритм Policy Iteration»  
Cliff Walking

ИСПОЛНИТЕЛЬ: \_\_\_Очеретная С.В.\_\_

ФИО

группа ИУ5-25М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_Гапанюк Ю.Е.\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

Москва - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Цель лабораторной работы:** ознакомление с базовыми методами обучения с подкреплением.

# Задание

На основе рассмотренного на лекции примера реализуйте алгоритм Policy Iteration для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

# Ход работы

## 1. Подготовка

Создали виртуальное окружение:

python -m venv env

Перешли в него:

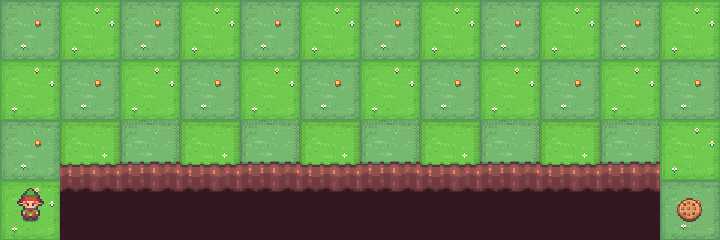
env/Scripts/activate.

Установили зависимости:

pip install -r .\examples\requirements.txt

## 2. Описание среды

Будем работать со средой Cliff Walking:



Поле представляет собой матрицу 4x12. Агент начинает проходить карту с ячейки [3, 0] (левый нижний угол). Ему необходимо достичь ячейки [3, 11], т.е. цель размещена в правом нижнем углу. Также агенту нельзя наступать на обрыв – это ячейки [3, 1...10] (внизу, по центру). Если агент наступит на обрыв, он вернется к началу. Эпизод заканчивается, когда агент достигает цели.

Агент может совершить 4 действия:

0: переместиться вверх;

1: передвинуться вправо;

2: передвинуться вниз;

3: передвинуться влево.

За каждый шаг полагается -1 награда, а за шаг в обрыв — -100.

Выведем информацию о наборе с помощью следующего кода:

state, action = 0, 0

env = gym.make("CliffWalking-v0")

print('Пространство состояний:')

pprint(env.observation\_space)

print()

print('Пространство действий:')

pprint(env.action\_space)

print()

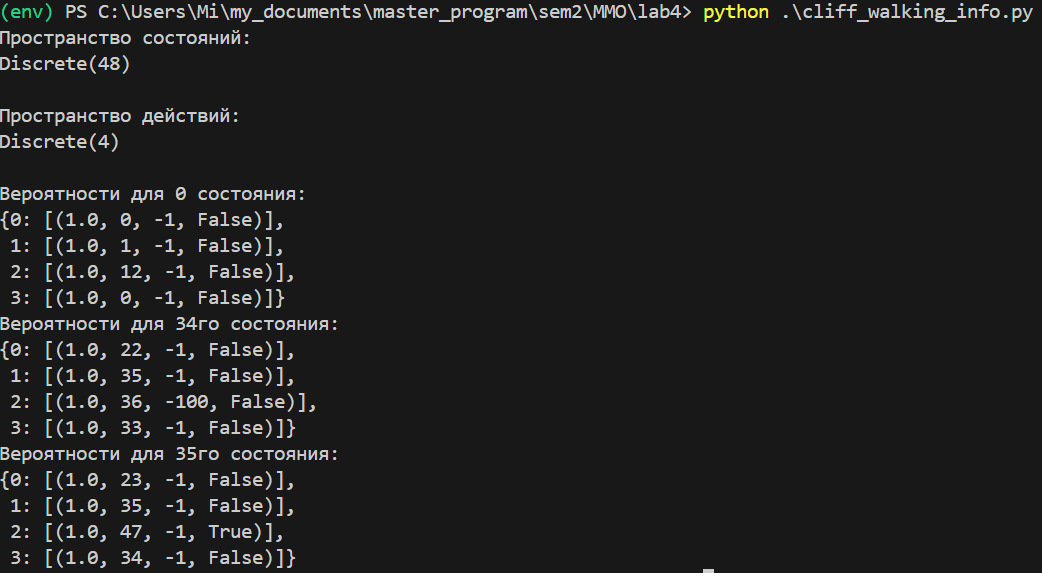
print('Вероятности для 0 состояния:')

pprint(env.P[state])

print(‘Вероятности для 46го состояния:')

pprint(env.P[46])

Вывод:



В итоге получаем: размерность пространства состояний 48 (это размерность поля 4х12), пространства действий 4 (вверх, право, вниз, влево).

Также в качестве примера просмотрели нулевое, 34ое и 35ое состояния из матрицы состояний env.P. Матрица env.P состоит из 48 строк (состояния от 0го до 47). Каждая строка состоит из объекта с 4мя действиями. Для каждого действия указан массив возможных состояний, в которое можно перейти из текущего состояния. Значение флага в конце массива равное True означает достижение цели. Формат строки:

{action: [(probability, nextstate, reward, done)]}

Таким образом, находясь в нулевом состоянии:

* можем остаться в нем при действиях 0 (вверх) и 3 (влево), т.к. врежемся в стенку карты;
* можем переместиться в состояние 1 при действии 1 (вправо);
* можем переместиться в состояние 12 при действии 2 (вниз);

Находясь в 34ом состоянии:

* переместимся в другое состояние при действиях 0, 1, 3;
* попадем в обрыв (-100 к награде) при действии 2 (вниз).

Находясь в 35ом состоянии:

* останемся в 35ом при действии 1 (вправо);
* переместимся в другое состояние при действиях 0, 3;
* достигнем цели (состояния 37) – флаг done=True

## 3. Реализация алгоритма

Произведем обучение с подкреплением для нашей модели. Для этого реализуем класс PolicyIterationAgent, эмулирующий работу агента.

Инициализируем класс, задав пространство состояний (observation\_dim = 48), действия (0-3), политику попыток (25%, что выполнится одно из 4ех действий), начальные значения состояний (state\_values), максимальное число итераций (10000) и начальные значения параметров theta, gamma:

def \_\_init\_\_(self, env):

self.env = env

self.observation\_dim = 48

self.actions\_variants = np.array([0,1,2,3])

self.policy\_probs = np.full((self.observation\_dim, len(self.actions\_variants)), 0.25)

self.state\_values = np.zeros(shape=(self.observation\_dim))

self.maxNumberOfIterations = 10000

self.theta=1e-6

self.gamma=0.99

Также добавим в класс метод вывода политики:

def print\_policy(self):

print('Стратегия:')

pprint(self.policy\_probs)

Метод оценивания стратегии:

def policy\_evaluation(self):

    valueFunctionVector = self.state\_values

    for iterations in range(self.maxNumberOfIterations):

        valueFunctionVectorNextIteration=np.zeros(shape=(self.observation\_dim))

        for state in range(self.observation\_dim):

            action\_probabilities = self.policy\_probs[state]

            outerSum=0

            for action, prob in enumerate(action\_probabilities):

                innerSum=0

                for probability, next\_state, reward, isTerminalState in self.env.P[state][action]:

                    innerSum=innerSum+probability\*(reward+self.gamma\*self.state\_values[next\_state])

                outerSum=outerSum+self.policy\_probs[state][action]\*innerSum

            valueFunctionVectorNextIteration[state]=outerSum

        if(np.max(np.abs(valueFunctionVectorNextIteration-valueFunctionVector))<self.theta):

            valueFunctionVector=valueFunctionVectorNextIteration

            break

        valueFunctionVector=valueFunctionVectorNextIteration

    return valueFunctionVector

Метод для улучшения стратегии:

def policy\_improvement(self):

    qvaluesMatrix=np.zeros((self.observation\_dim, len(self.actions\_variants)))

    improvedPolicy=np.zeros((self.observation\_dim, len(self.actions\_variants)))

    for state in range(self.observation\_dim):

        for action in range(len(self.actions\_variants)):

            for probability, next\_state, reward, isTerminalState in self.env.P[state][action]:

                qvaluesMatrix[state,action]=qvaluesMatrix[state,action]+probability\*(reward+self.gamma\*self.state\_values[next\_state])

        bestActionIndex=np.where(qvaluesMatrix[state,:]==np.max(qvaluesMatrix[state,:]))

        improvedPolicy[state,bestActionIndex]=1/np.size(bestActionIndex)

    return improvedPolicy

Реализация алгоритма:

def policy\_iteration(self, cnt):

    policy\_stable = False

    for i in range(1, cnt+1):

        self.state\_values = self.policy\_evaluation()

        self.policy\_probs = self.policy\_improvement()

    print(f'Алгоритм выполнился за {i} шагов.')

Проигрывание сцены:

def play\_agent(agent):

env2 = gym.make('CliffWalking-v0', render\_mode='human')

state = env2.reset()[0]

done = False

while not done:

    p = agent.policy\_probs[state]

    if isinstance(p, np.ndarray):

        action = np.random.choice(len(agent.actions\_variants), p=p)

    else:

        action = p

    next\_state, reward, terminated, truncated, \_ = env2.step(action)

    env2.render()

    state = next\_state

    if terminated or truncated:

        done = True

Теперь создадим основную функцию для выполнения и запустим скрипт:

def main():

    # Создание среды

    env = gym.make('FrozenLake-v1')

    env.reset()

    # Обучение агента

    agent = PolicyIterationAgent(env)

    agent.print\_policy()

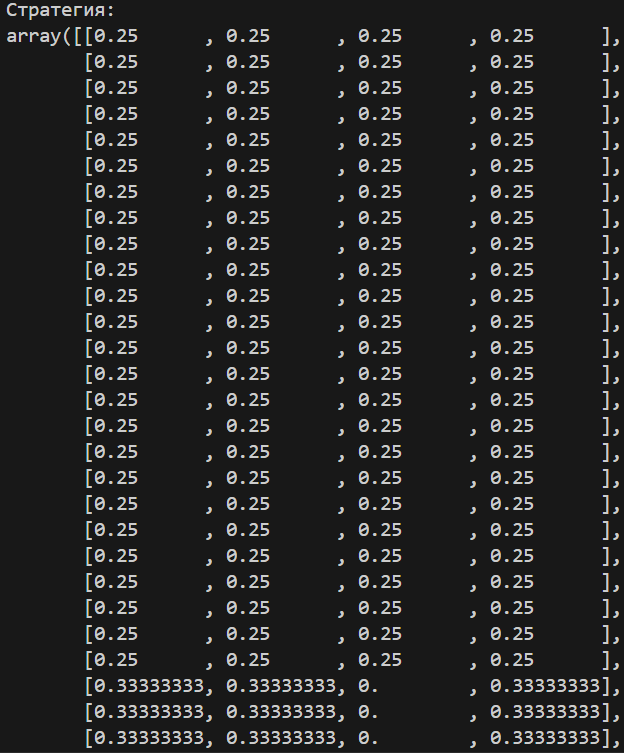
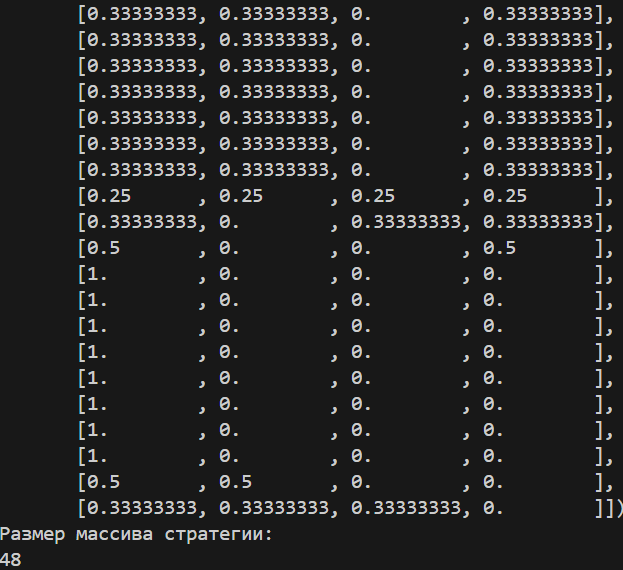
    agent.policy\_iteration(10000)

    agent.print\_policy()

    # Проигрывание сцены для обученного агента

    play\_agent(agent)

Наш агент долго блуждал по карте, но в итоге достиг цели. Получили следующий массив стратегии

Данный массив представляет собой матрицу состояний обученного агента. Каждая строка – массив вероятностей 1го действия (чем больше вероятность, тем лучше выполнение конкретного действия по мнению обученного агента). Как видим по матрице – для состояний 0-24,35 массивы с одинаковыми значениями 0,25, поскольку при перемещениях из верхних ячеек (и 24ой, 35ой) ничего не происходит. Для состояний 25-34 агент понял, что не выгодно двигаться вниз (для действия 0 – вероятность 0, а для остальных 0,33). При 36ом состоянии невыгодно двигаться вправо (второе значения массива – 0, остальные 0,33). Находясь в обрыве (хотя это невозможно) в состоянии 37 – нужно двигаться влево или вверх, в состояниях 38-45 – только вверх, в состоянии 46 – вправо или вверх. Находясь в состоянии 47, нужно не двигаться влево (в обрыв).