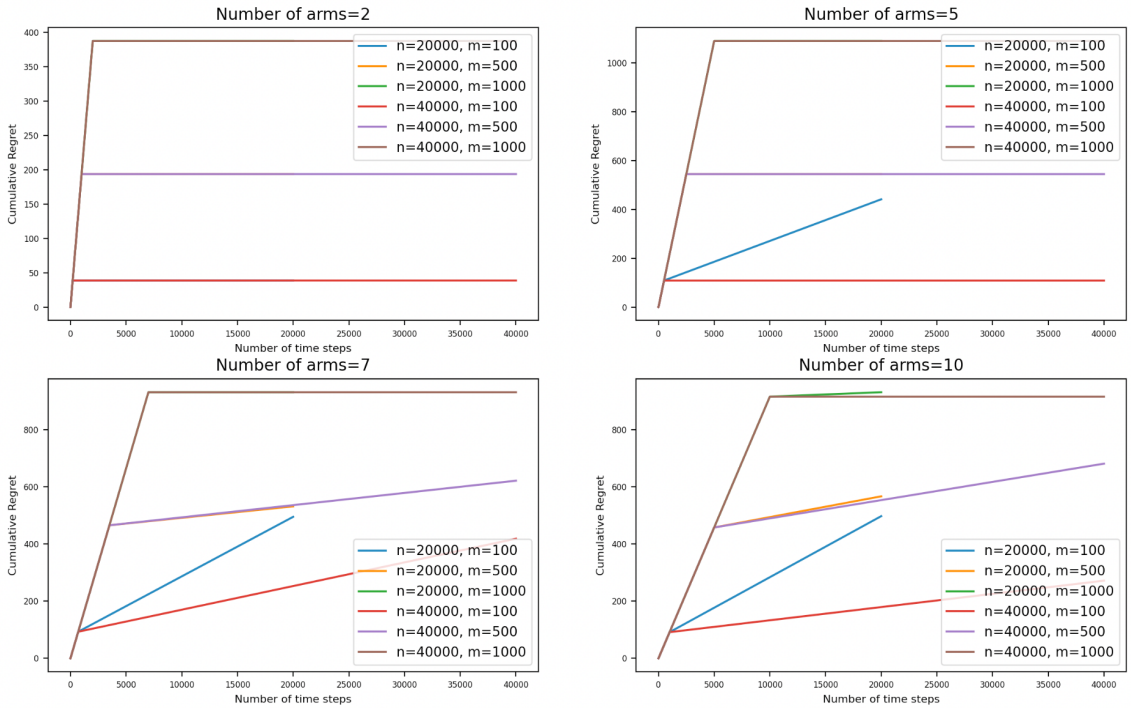
Multi-armed Bandit (K-armed bandit) Algorithms

The entire code of these algorithms can be found at my github link.

Explore-then-Commit (ETC)

```
def ETC(arm_means, num_arms, total_steps, m):
 ### Choosing the optimal arm
 optimal_arm = np.argmax(arm_means)

 num_iterations = 10 # number of times we perform the same experiment

 regret = np.zeros([total_steps,num_iterations])
  for iter in range(num_iterations):
   num_steps = 0
   emp_means = np.zeros(num_arms)
   num_pulls = np.zeros(num_arms)
   # Pure Exploration Phase
   for i in range(m):
       for j in range(num_arms):
         num_pulls[j] += 1
         # generate bernoulli reward from the picked greedy arm
         reward = np.random.binomial(1, arm_means[j])
         emp_means[j] += (reward - emp_means[j])/num_pulls[j]
         regret[num_steps, iter] += arm_means[optimal_arm] - arm_means[j]
         num_steps += 1

   # Pure Exploitation Phase
   selected_best_arm = np.argmax(emp_means)

   exploitation_steps = total_steps - m * num_arms
   for _ in range(exploitation_steps):
     regret[num_steps, iter] += arm_means[optimal_arm] - arm_means[selected_best_arm]
     num_steps += 1
  return regret
```

This algorithm follows a pure exploration phase, and then a pure exploitation phase. The exploration phase consists of a total of mk steps, where k is the number of arms we have, and m is the number of times we pull each arm. We estimate the empirical mean reward for each arm. The regret increases almost linearly with n in the exploration phase. At the end of the exploration phase, we choose the best arm which has the maximum empirical mean. If we choose m wisely, we could select the optimal arm which has the actual best mean reward. In the exploitation phase, we continue to pull the best arm we selected at the end of the first phase. If the arm chosen was the arm with the best actual mean, we would get no further regret. Else the regret will continue to grow linearly with the amount of gap (difference between the actual means of the optimal arm and chosen arm) at each timestep.



The different experiments performed using the ETC algorithm are shown above.

Analysis: The upper bound of regret for ETC with R-subgaussian rewards is:

$$R_n \leq \min\left(m, \lceil \frac{n}{k} \rceil\right) \sum_{i=1}^{k} \Delta i + \max\left(n - mk, 0\right) \sum_{i=1}^{k} \Delta i \exp\left(-\frac{m\Delta i^2}{4R}\right)$$

In the experiment, k = 2, the actual mean of the arms is 0.69 and 0.31. The upper bound at the end of exploration phase we get for m = 100 is 100 * 0.38 = 38, which is depicted in the first plot shown above. Similarly, for m = 500, the exploration regret is bounded by 500 * 0.38 = 190 shown in the figure. For the exploitation phase with k=2, the agent was able to select the optimal arm, and so no regret in the exploitation phase.

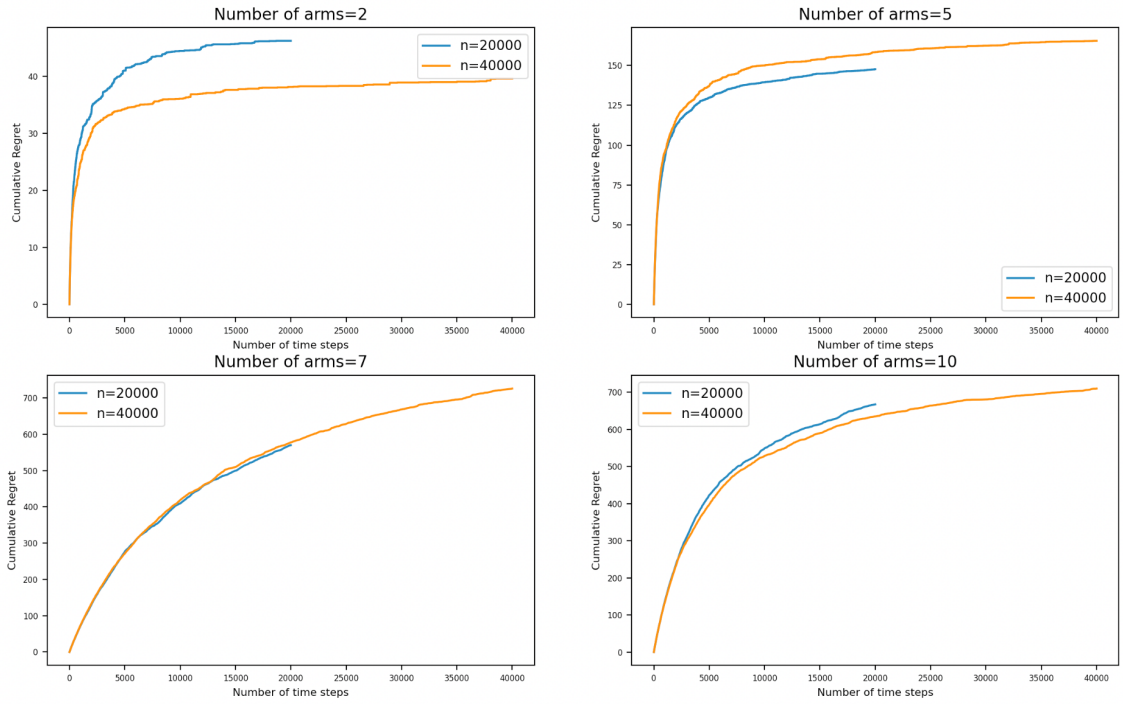
For k = 5 and m = 100, the actual mean of the arms were 0.22, 0.42, 0.33, 0.02 and 0.01. The exploration regret calculated using the above equation would be 100 * 1.1 = 110 which is shown in the plot. For the exploitation phase with n = 20000, the regret should be bounded by 110 + 1700 = 1810. We just get a regret of 400 which is good.

Upper Confidence Bound (UCB)

```
def UCB(arm_means, num_arms, total_steps, delta=1e-4):
 ### Choosing the optimal arm
 optimal_arm = np.argmax(arm_means)
  num_iterations = 10 # number of times we perform the same experiment to
reduce randomness
  regret = np.zeros([total_steps, num_iterations])
  for iter in range(num_iterations):
   ucb = 100 * np.ones(num_arms)
   emp_means = np.zeros(num_arms)
   num_pulls = np.zeros(num_arms)
   for step_count in range(total_steps):
       greedy_arm = np.argmax(ucb)
       # generate bernoulli reward from the picked greedy arm
       reward = np.random.binomial(1, arm_means[greedy_arm])
       num_pulls[greedy_arm] += 1
       regret[step_count, iter] = arm_means[optimal_arm] -
arm_means[greedy_arm]
       emp_means[greedy_arm] += (reward -
emp_means[greedy_arm])/num_pulls[greedy_arm]
       ucb[greedy_arm] = emp_means[greedy_arm] + np.sqrt(2 *
np.log(1/delta) / num_pulls[greedy_arm])
  return regret
```

The algorithm performs better than the ETC algorithm, and gives a better bound on regret. After each time step, our agent selects the arm which has the highest confidence. The confidence of each arm is based on the empirical mean calculated so far, and another term which depends inversely on the number of times the arm has been pulled, and delta which is chosen to be 0.0001 in our case. The more an arm is pulled, we have more confidence towards its calculated empirical mean.

UCB



Analysis:

The regret of UCB is upper bounded as:

$$R_n \leq \sum_{\Delta i > 0} \frac{16 \log(n)}{\Delta i} + 3 \sum_{i=1}^{k} \Delta i$$

For k = 2 with means 0.69 and 0.31, n = 20000, we get an upper bound of 418.
For even a smaller n = 5000, the computed upper bound for regret is 360. The regret we get in the experiment is way below these values, meaning that our agent was able to pick the optimal arm much faster. Note that the above regret bound is problem dependent and depends on the reward gap. If the reward gap is too small, the upper bound becomes very large. We can also show the upper bound of regret in case of UCB as:

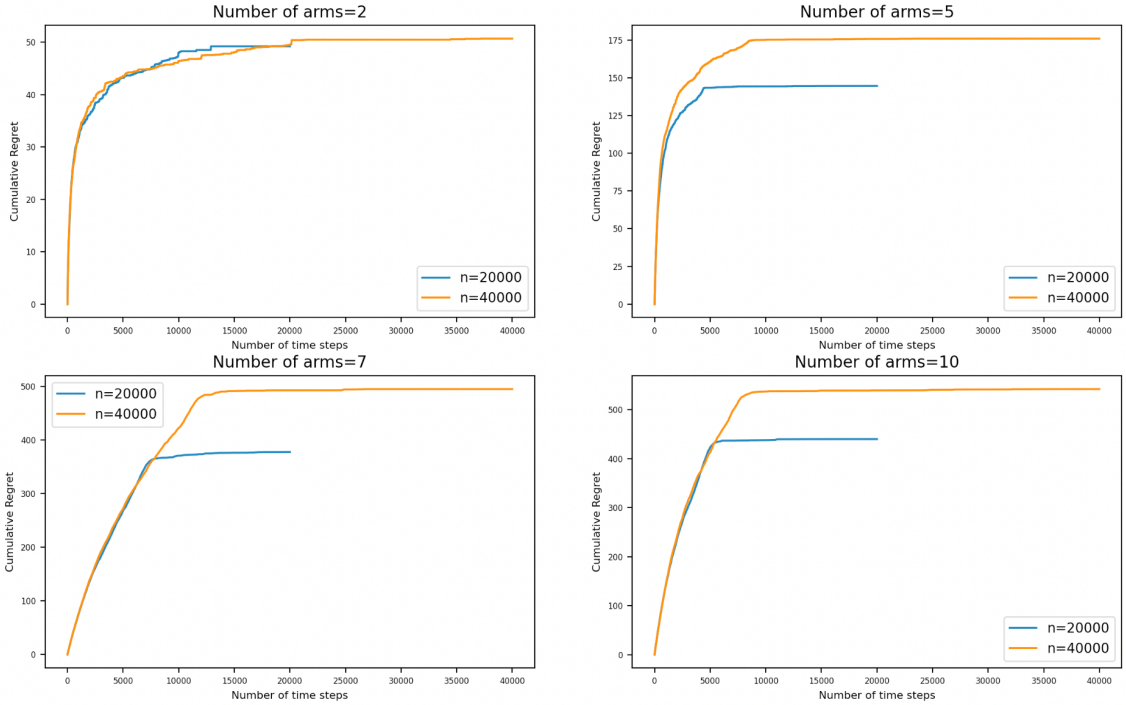
$$R_n \leq 8\sqrt{nk \log(n)} + 3 \sum_{i=1}^{k} \Delta i$$

Minimax-Optimal Strategy in Stochastic Multi Armed Bandits (MOSS)

```
def MOSS(arm_means, num_arms, total_steps):
 ### Choosing the optimal arm
 optimal_arm = np.argmax(arm_means)

 num_iterations = 10 # number of times we perform the same experiment
  regret = np.zeros([total_steps, num_iterations])
  for iter in range(num_iterations):
   ucb = 10 * np.ones(num_arms)
   emp_means = np.zeros(num_arms)
   num_pulls = np.zeros(num_arms)
   for step_count in range(total_steps):
       greedy_arm = np.argmax(ucb)
       # generate bernoulli reward from the picked greedy arm
       reward = np.random.binomial(1, arm_means[greedy_arm])
       num_pulls[greedy_arm] += 1
       regret[step_count, iter] = arm_means[optimal_arm] - arm_means[greedy_arm]
       emp_means[greedy_arm] += (reward - emp_means[greedy_arm])/num_pulls[greedy_arm]
       ucb[greedy_arm] = emp_means[greedy_arm] + np.sqrt((4 / num_pulls[greedy_arm]) *
np.log(max(1, total_steps/(num_arms * num_pulls[greedy_arm]))))

return regret
```

This algorithm achieves a regret which doesn't have a log factor in the regret upper bound unlike the upper confidence bound algorithm. The algorithm converges and gives better regret than UCB algorithm when the number of arms are more in the experiment.

MOSS



Analysis: The regret for MOSS is upper bounded as:

$$R_n \leq 39\sqrt{kn} + \sum_{i=1}^{k} \Delta i$$

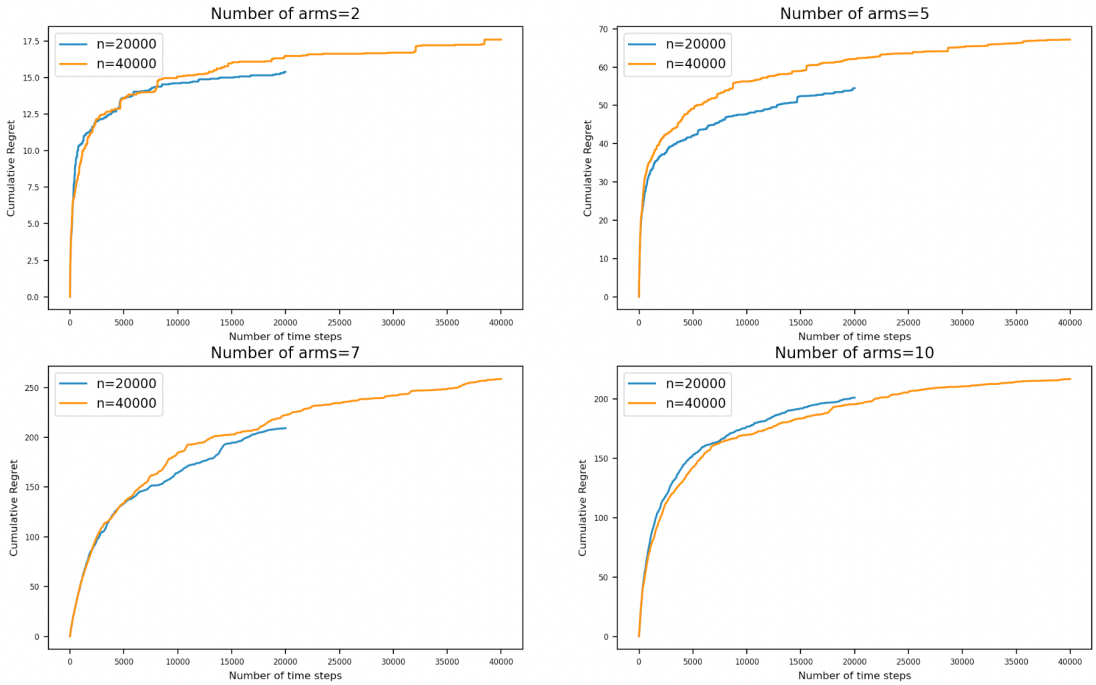
For n = 20000 and k = 2, the value we get is 7800. In the experiment, we get a regret of ~50 which is very small.

Kullback-Leibler upper confidence bound (KLUCB):

```
def KLUCB(arm_means, num_arms, total_steps):
 ### Choosing the optimal arm
 optimal_arm = np.argmax(arm_means)

 num_iterations = 10 # number of times we perform the same experiment
  regret = np.zeros([total_steps, num_iterations])
  for iter in range(num_iterations):
   emp_means = np.zeros(num_arms)
   num_pulls = np.zeros(num_arms)
   t = 0
   for step_count in range(0, total_steps):
       t += 1
       if step_count < num_arms:
           greedy_arm = step_count % num_arms
       else:
           # pick the best arm according to KL-UCB algorithm
           arm_confidence = np.zeros(num_arms)
           for idx in range(num_arms):
               arm_confidence[idx] = kl_confidence(t, emp_means[idx], num_pulls[idx])
           greedy_arm = np.argmax(arm_confidence)
       # generate bernoulli reward from the picked greedy arm
       reward = np.random.binomial(1, arm_means[greedy_arm])
       num_pulls[greedy_arm] += 1
       regret[step_count, iter] += arm_means[optimal_arm] - arm_means[greedy_arm]
       emp_means[greedy_arm] += (reward - emp_means[greedy_arm])/num_pulls[greedy_arm]

 return regret
```

KL-UCB algorithm performs the best of all the above algorithm, and also satisfies a uniformly better regret bound than UCB and MOSS in the special case of Bernoulli rewards. It reaches the lower bound of Lai and Robbins. KL-UCB is remarkably efficient and stable, including for short time horizons. Also to note that, KL-UCB is also the only method that always performs better than the basic UCB policy.

KLUCB



Analysis:

Lai-Robbins Theorem: Asymptotic total regret is at least logarithmic in number of steps.
In other words, regret grows at least logarithmically.

$$\lim_{n\to\infty} R_n \geq \log(n) \sum_{\Delta i>0} \frac{\Delta i}{kl(p_i, p_*)}$$

where, $kl(p_i, p_*)$ is the KL-divergence between the distributions of the optimal arm and other arms with a reward gap more than zero.

In our experiments, KL-UCB performs the best. Take the case of k=2 arms, we achieve a regret of 15.5 for n = 20000. The arms have the same distribution like the other experiments for previous algorithms with mean 0.69 and 0.31 and have Bernoulli distribution. The lower bound as per Lai-Robbins theorem is also calculated to be 15.5. Thus, we are able to achieve the lower bound.