THSP18

Una comparación entre ϵ-greedy y UCB en el problema Bandido multibrazo

Filip Cornell

4 de enero de 2019

Resumen

Con el avance de tecnología y la explosión de datos, las posibilidades de aprender de los datos usando máquinas de aprendizaje se han expandido. Un subcampo dentro de máquinas de aprendizaje es aprendizaje por refuerzo, donde un problema popular de investigar es el bandido multibrazo. Se define por tener K brazos, cada uno con una probabilidad de dar una remuneración. El objetivo es aprender estas probabilidades y al mismo tiempo maximizar la remuneración. En este artículo, se analiza tres algoritmos diferentes usados para resolver el bandido multibrazo; el ϵ-greedy, `UCB` y `SoftMax`. Examinamos su rato de convergencia y eficiencia, ambo empíricamente por hacer experimentos como teóricamente por estudios anteriores. Una sorpresa es que ϵ-greedy es mejor que los otros en casi todas los experimentos, mientras la hipótesis era que `UCB` sería el mejor, que no fue el caso. El algoritmo `SoftMax` resulta ser el peor, siempre teniendo los peores resultados.

Índice

1. Introducción

En los últimos años, el número de investigaciones dentro del campo de máquinas de aprendizaje ha crecido fuertemente. Nuevos campos de aplicación de la ciencia de datos son descubiertos cada día, y en cualquier área en donde se genera datos en alguna manera, se lo puede aplicar para resolver diferentes problemas y llegar a conclusiones nuevas. Suponga que tenemos un millón imágenes de perros y gatos, y necesitamos dividirlas en las de perros y las de gatos. ¿Cómo lo hacemos? O suponga que tenemos una página en el web donde queremos mostrar las noticias más relevantes al visitante, pero solamente podemos mostrar cinco de los mil que hay. ¿Cómo elegimos las más relevantes para el usuario? Estos son problemas típicas en cuales se aplica máquinas de aprendizaje.

El proceso de utilizar máquinas de aprendizaje es bastante simple, aunque los pasos pueden ser complicados. Primero, se enfrenta con un problema de datos. Al problema, se aplica un modelo de resolución, que nos ayudará encontrar estos patrones y la información que se necesita de los datos. Este modelo es, en la mayoría de los casos, un modelo estadístico para poder determinar probabilidades. Por ejemplo, en el caso de los gatos y los perros, se aplicaría un modelo que generaría probabilidades de clasificación. Para decidir modelo, se necesita primero analizar el problema y los datos, para luego poder decidir cuál sería el mejor para la situación. Para poder aplicar el modelo al problema, se necesita implementar un algoritmo que implícitamente o explícitamente implementa el modelo. El algoritmo creado genera un resultado del modelo aplicado a los datos. Este proceso se aplica, más o menos, en todos los subcampos y aplicaciones diferentes de aprendizaje de máquinas.

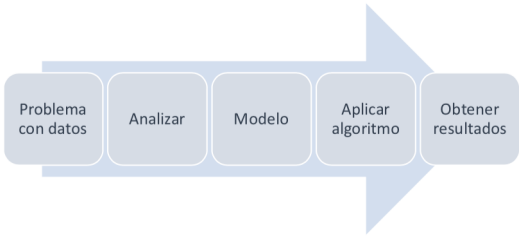


Figura 1: El proceso cuando se utiliza máquinas de aprendizaje para resolver un problema.

1.1. Aprendizaje por refuerzo

Un subcampo de aprendizaje de máquinas es aprendizaje por refuerzo. En muchas situaciones en la vida real, se tiene un problema donde se quiere maximizar los beneficios de una acción específica mientras se va aprendiendo qué sería la opción mejor.[6] Todos los problemas donde se aplica aprendizaje por refuerzo, se puede describir en unos componentes generales; un agente, un ambiente e un interpretador. El agente es el componente que tiene un número de acciones posibles de elegir para afectar al ambiente. Basado de un algoritmo implementado, el agente toma una decisión de acción por analizar el ambiente y acciones anteriores. La acción genera un resultado, que el interpretador interprete. El interpretador le da una remuneración al agente, dependiente de si la acción generó un resultado deseable. El agente registra esta remuneración, y basa sus decisiones siguientes en la información dada por la cantidad de remuneración obtenida.[6]

Es claro que aprendizaje no es lo más deseable; se prefiere poder aprender de los datos antes que se comience predecir los nuevos. No obstante, en muchas aplicaciones en la vida real, esto no es posible, y muchas veces no hay datos antes. Aprendizaje por refuerzo fue creado

debido a esta necesidad, y es utilizado para resolver muchos problemas diferentes.

Un problema donde se lo aplica es el bandido multibrazo, que es un problema muy conocido dentro del área. En el problema, se considera $K \geq 2$ brazos, o palancas, que un agente puede pulsar. Cada brazo tiene una probabilidad p de remunerar el agente con una remuneración, que puede ser fija o distribuida por una distribución estadística. Por fin, se busca maximizar la cantidad de remuneraciones obtenidas, pero esto se modela indirectamente por el arrepentimiento, un término que presentaremos luego en este artículo.

1.2. Campos de aplicación

Uno necesita preguntarse, antes de comenzar estudiar un problema, ¿para qué sirve estudiarlo? En este caso, el problema ha mostrado ser muy útil y aplicable en muchas situaciones reales. Un ejemplo típico son ensayos clínicos, donde se ha encontrado que aprendizaje por refuerzo puede ser beneficioso por su característico adaptable. En estos casos, el problema es que los pacientes tiene una enfermedad, para cual se tiene varias medicinas sin saber cuál es la más eficiente. Se prueba en pacientes diferentes, y por utilizar aprendizaje por refuerzo, se puede determinar más rápido, durante las pruebas, cuáles medicinas son las más eficientes y cuáles se debería dejar de usar. [9]

Otro ejemplo de una aplicación del bandido multibrazo es optimización de anuncios. Hoy en día, muchos servicios en internet tiene su fuente de ingresos por mostrar anuncios relevantes a usuarios que utilizan sus servicios. Unos ejemplos son Google y Yahoo, que recomiendan artículos de noticias que pueden ser interesantes para el usuario. El problema aquí, es cómo se elige los anuncios más relevantes para cada usuario. Si un usuario elige a cliquear un artículo es considerado un beneficio ya que eso indica que el artículo puede ser interesante para el usuario. Ya que esta información relevante no existe antes que los usuarios comienzan actuar con el sistema, el sistema tiene que aprender mientras los usuarios utilizan el servicio. Así, mientras es muy importante que se encuentre los anuncios más relevantes para los usuarios, esto no es posible hacer antes que los usuarios actúan. [5] Por esto, se necesita un algoritmo que aprende mientras se recibe las respuestas, y es aquí que aprendizaje por refuerzo es muy conveniente. En este problema, se representa los anuncios y los usuarios como vectores de información, dependiente del contenido del anuncio y las características del usuario. [5] Dependiente de cuáles usuarios hacen un click en cuáles artículos, el sistema aprende las probabilidades que un usuario con un vector de información específica hará un click a un anuncio, dado la información representada por el vector representando el anuncio. Este problema se considera como un variante del problema bandido multibrazo. [5]

Aunque la formulación del problema Bandido multibrazo es aplicable a los ejemplos mencionados arriba y muchos más, todavía existe poca investigación dentro del área. [3] Hay mucha teoría dentro del área, pero mucha investigación, primeramente investigación empírica, que prueban y comparan los algoritmos, queda para determinar la manera más eficiente de resolver el problema de bandidos multibrazo.

En este artículo se compara tres modelos en el subcampo de aprendizaje de máquinas, `SoftMax`, ϵ-greedy y `UCB` para resolver el problema Bandido Multibrazo. Usamos ϵ-greedy como un punto de referencia para los otros, `UCB` y `SoftMax`, dos otros algoritmos que contituyen la base de otros algoritmos en muchas investigaciones anteriores. [1, 2, 5, 8] El artículo comienza con una introducción a aprendizaje por refuerzo y sus aplicaciones, seguido de una descripción del problema específico y sus campos de aplicación. Luego, se presenta teoría del problema y los algoritmos y el método usado para comparar los algoritmos. Luego, se muestra los resultados,

que son discutidos y analizados. Finalmente, el artículo termina con una conclusión.

1.3. Propósito

El propósito de este artículo es simple; ¿cuál de los métodos `UCB` y ϵ-greedy es el mejor para el problema Bandido multibrazo, con K palancas y una remuneración fija? Para determinar esto, hay unas medidas para decidirlo. En trabajos anteriores, se ha medido ambo el arrepentimiento total y el arrepentimiento como una función del tiempo, para estudiar la convergencia. [7, 8, 5, 1] Las preguntas nos preguntaremos son las siguientes.

- ¿Cuál algoritmo tiene el menos arrepentimiento R_T?
- ¿Cómo cambia el arrepentimiento R_T como una función del tiempo. Es decir, ¿qué conclusiones se puede deducir del grafo R_t en el eje vertical y el tiempo en eje horizontal?
- ¿Bajo de las mismas circunstancias, cuál nos da las ganancias máximas?
- ¿Cómo cambian los factores mencionados arriba bajo de diferentes variantes de Bandido multibrazo?
- ¿Cuántas veces se elige el mejor brazo?

1.4. Hipótesis

La hipótesis es que `UCB` será el mejor algoritmo en términos de arrepentimiento, debido su manera de automáticamente elegir el posiblemente mejor brazo. Los algoritmos ϵ-greedy y `SoftMax` dependerán demasiado de la aleatoriedad y por ende, no serán tan eficientes. Por ende, tendrán arrepentimientos R_T creciendo más rápido que el arrepentimiento del `UCB`, y serán peores.

2. Teoría

En esta sección, presentaremos la teoría del problema y los algoritmos ϵ-greedy, `UCB` y `SoftMax`.

2.1. Modelo matemático

El problema Bandido multibrazo se suele definir en forma matemática. Los variables pueden variar dependiente de a qué se aplica el problema, pero definir el modelo general es trivial debido a su estructura simple. No obstante, es importante notar que hay unos factores que cambian el modelo.

2.2. Variables y conjuntos

Para entender este artículo, necesitamos primeramente explicar unos términos. Se considera los siguientes conjuntos.

- $K = \{1, .., k\}$ son las k palancas.
- $T = \{1, .., t\}$ es el conjunto de las iteraciones. Es decir, el número de iteraciones hecho durante un corrido de un algoritmo. Cuando se estudia este problema teóricamente, se deja que $T \to \infty$ para poder entender qué pasa si se deja el algoritmo correr muchas veces.

Los variables del problema se define como los siguientes.

- $y_{k,t} \begin{cases} 1 & \text{si palanca } k \text{ es elegida en iteración } t \\ 0 & \text{si no} \end{cases} \quad \forall k \in K, \forall t \in T$

- $P_k(\theta_k) \quad \forall k \in K$ se denomina la distribución de remuneración. Esta es normalmente no conocida, pero durante simulaciones, se considera normalmente que $P_k \sim \mathcal{N}(\mu_k, \sigma_k^2)$ por simplicidad. [7, 3]
- Denotamos $T_{k,t}$ como un variable que describe la distribución del número de veces que brazo k es pulsado antes de iteración T. Usaremos esto luego para mostrar unos teoremas.
- $r(t) \sim P_k(\theta_k)$ se denomina la remuneración obtenida en paso t, suponiendo que brazo k fue elegido en paso t.
- $\mu_k \quad \forall k \in K$ se denomina la remuneración esperada de palanca k. Se define como $\mu_k = \int_{-\infty}^{\infty} xP(x;\theta_k)d\nu(x)$, donde θ_k es un parámetro no conocido para la distribución $P_k(x|\theta_k)$, tampoco conocida. En muchas investigaciones, se simplifica esta a ser de una distribución fija, y solamente se considera la remuneración media. [8]
- μ^* se denomina la remuneración media del mejor brazo.
- $n_{k,t} \forall t \in T, k \in K$ se denomina el número de veces que se ha pulsado brazo k antes de iteración t. Se tiene que $n_{k,1} = 0 \ \forall k \in K$.
- $\hat{\mu}_k \quad \forall k \in K$ se denomina la aproximación de la remuneración por cada palanca. Dependiente del algoritmo, se calcula en diferentes maneras, pero un variante de una estimación simple puede ser, por ejemplo, $\hat{\mu}_k = \frac{\sum_{t=1}^{T} y_{k,t} \cdot r(t)}{n_{k,t}}$.

Los parámetros del problema que son ajustables y que busca conocer, son los siguientes.

- K - el número de brazos en el problema. Se ha mostrado que afecta mucho a la complejidad del problema. [2, 3, 6] Lo más brazos que se tiene, lo más complejo sea el problema.
- $\theta_k \quad \forall k \in K$ son los parámetros no conocidos que determinan la distribución no conocida. Normalmente, estos no son interesante conocer, sino solamente el valor esperado del brazo, $\mathbb{E}[P(\theta_k)] = \mu_k$. [4]

También hay otras definiciones importantes para entender este artículo, que usaremos para probar unos teoremas importantes.

- $\delta_k = \mu^* - \mu_k \ \forall k \in K \backslash \{k : \mu_k = \mu^*\}$ es la diferencia en remuneración promedia entre el mejor brazo y brazo k. Dejamos que $\delta_{min} = min_{k \in K} \mu^* - \mu_k$, la diferencia entre el mejor y el segundo mejor brazo en términos de remuneración promedia.
- $\Delta = \sum_{k=1}^{K} \delta_k$. Es decir, se denomina Δ como la suma de las diferencias entre el mejor y los otros brazos.

2.3. ¿Cómo se compara los algoritmos?

Como mencionamos antes, el problema trata de maximizar la función objetiva que se define como

$$max \sum_{t=1}^{T} r_{k,t} \tag{1}$$

donde $r_{k,t}$ es la remuneración en iteración t de brazo k. [6] No obstante, solo mirar en esto para evaluar la eficiencia de un algoritmo puede ser engañoso, ya que solamente se estudia el resultado final. En cada problema que incorpora probabilidades, se tiene un ruido, que se tiene que tener en cuenta. Por ende, solamente mirar a la remuneración total para evaluar el algoritmo no nos dará una vista justa, si no se hace una cantidad de iteraciones inmensa para cada algoritmo

que se evalúa. Por esto, se utiliza otra medida para evaluar el algoritmo, el arrepentimiento. Se describe el arrepentimiento como

$$R_T = \sum_{t=1}^{T} \mu^* - \mu_{k(t)} = \sum_{t=1}^{T} \mu^* - \sum_{t=1}^{T} \mu_{k(t)} \tag{2}$$

Donde $\mu_{k,t}$ es la remuneración esperada del brazo elegido iteración T. Se ha probado que R_T puede ser descrito como

$$R_T = T\mu^* - \mu_{k,t} \sum_{i=1}^{K} \mathbb{E}[T_{k,t}] \tag{3}$$

Aquí, T_k,t depende del algoritmo usado.

2.4. ¿Qué tan eficiente necesita ser el algoritmo para resolver el problema?

Cuando se analiza los algoritmos que intentan resolver el bandido multibrazo, se busca conocer el rato de crecimiento de R_T. Para determinar esto, se quiere estudiar la complejidad mínima del algoritmo. Se analiza esto por utilizar el término $R_T = \Omega(f(T))$, que indica que R_T crece con un rato mínimo de $c \cdot f(T)$, donde c es un constante arbitrario. En 1985, Lai y Robbins mostraron que para todos brazos peores que el mejor, se tiene la garantía que

$$\mathbb{E}[T_{k,t}] \geq \frac{log(T)}{\int P_k log(\frac{P_k}{P^*})} = c \cdot log(T), \text{ donde } c = \frac{1}{\int P_k log(\frac{P_k}{P^*})} \quad \forall k \in \{k : \mu_k \neq \mu^*\} \tag{1}$$

donde $\int P_k log(\frac{P_k}{P^*})$ se puede describir como la diferencia entre la distribución de remuneración para el mejor brazo y el segundo mejor. Debido a la ecuación arriba y a que se puede reducir el denominador a un constante c en términos de T, se puede deducir que la función $f(T)$ es $log(T)$, ya que $\int P_k log(\frac{P_k}{P^*})$ se trata como un constante en términos de T. Por ende, se necesita que el algoritmo que se utiliza llegue logra una eficiencia que hace que $R_T \leq \Omega(log(T))$. [4] Si dividimos por T, obtenemos que $R_T/T \leq \Omega(\frac{log(T)}{T})$, así que un algoritmo que busca resolver el problema tiene que cumplir que, por lo menos, crezca más despacio que $c \cdot \frac{log(T)}{T}$. Ya que $log(T)$, el numerador, crece menos rápido que T, el denominador, tenemos que $R_T/T \leq \Omega(\frac{log(T)}{T}) \rightarrow 0$ cuando $T \rightarrow \infty$. Los algoritmos que logran este requerimiento se llaman *estrategias de zero-regret*, y significa que se logrará, por fin, solamente elegir el mejor brazo todo el tiempo. [7]

2.5. ϵ-greedy

El algoritmo ϵ-greedy es, probablemente, el algoritmo más simple e intuitivo. Se basa en dos componentes; explorar y explotar. Por un lado, se necesita explorar por investigar tantas palancas posibles y determinar sus valores. Por cada iteración que se investiga una palanca, se obtiene más información y se puede determinar en el futuro cuál será el mejor. Por el otro lado, se quiere explotar las palancas más rentables para maximizar las ganancias de las iteraciones.

Suponga que se usamos el algoritmo. Dado los valores iniciales de $\hat{\mu}_{1,(0)}, .., \hat{\mu}_{K,(0)}$, se define la probabilidad $p_{k,t}$ de elegir brazo k en iteración t como

$$p_{k,t+1} = \begin{cases} 1 - \epsilon + \frac{\epsilon}{K} & \text{si } i = argmax_{k \in K} \hat{\mu}_{k,t} \\ \frac{\epsilon}{K} & \text{si no} \end{cases} \quad \forall k \in K, \forall t \in T \tag{1}$$

Hay muchos variantes del ϵ-greedy. El más simple, es cuando se tiene ϵ fijado, así el mismo valor, independiente de la iteración. No obstante, este variante se ha probado que solamente tiene una

garantía lineal y no logarítmico, que hace que no puede tener la propiedad *zero regret*. [1] La prueba es bastante simple. Deje que $\epsilon \in [0, 1]$ es un número fijo. Entonces, se tiene que

$$\begin{aligned}
\mathbb{E}[R_T/T] = \frac{1}{T}\sum_{t=1}^{T}\mu^* - \mu_{k,t} &\geq \frac{1}{T}\sum_{t=1}^{T}(1-\epsilon+\frac{\epsilon}{k})\mu^* - (\epsilon\cdot\frac{k-1}{k})\mu_{k,t} \\
&= \frac{T}{T}\mu^*(1-\epsilon+\frac{\epsilon}{k}) - \frac{\epsilon\cdot\frac{k-1}{k}}{T}\Big(\sum_{t=1}^{t_1}\mu_{k,t} + \sum_{t=t_1+1}^{T}\mu^*\Big) = \\
&= \mu^*\big((1-\epsilon+\frac{\epsilon}{k}) - \frac{1}{T}(T-t_1+1)\big) - \frac{\epsilon\cdot\frac{k-1}{k}}{T}\sum_{t=1}^{t_1}\mu_{k,t} = \qquad (2) \\
&= \mu^*\big(1-\epsilon+\frac{\epsilon}{k}-1+\frac{t_1}{T}-\frac{1}{T}\big) - \frac{\epsilon\cdot\frac{k-1}{k}}{T}\sum_{t=1}^{t_1}\mu_{k,t} \to \\
&\to \mu^*\big(\epsilon+\frac{\epsilon}{k}+0-0\big) - 0\sum_{t=1}^{t_1}\mu_{k,t} = \mu^*(\epsilon+\frac{\epsilon}{k}) \text{ cuando } T\to\infty
\end{aligned}$$

Donde t_1 denomina el tiempo requerido para obtener que μ^* es la mejor según la estimación. Por el resultado obtenido arriba, tenemos el requerimiento que $\epsilon \to 0$ cuando $T \to \infty$ para cumplir que $\mathbb{E}[R_T/T] \to \infty$ cuando $T \to \infty$ y ser una estrategia de *zero regret*. El requerimiento mínimo para que esto pase es que $\epsilon = \Omega(1/log(T))$. Es decir, que ϵ disminuya por lo menos con un rato proporcional a $\frac{1}{log(T)}$. Se ha probado en investigaciones anteriores que para ϵ-greedy donde ϵ disminuye con un rato mayor que log(t), se tiene que $R_T = \mathcal{O}(Klog(T))$, así que $R_T/T = \mathcal{O}(K\frac{log(T)}{T})$ donde K indica el número de brazos. [1] Esto cumple el requisito mencionado anteriormente. Por ende, es mostrado teóricamente que el algoritmo puede resolver el problema dentro del tiempo dicho, suponiendo que $\epsilon \leq \Omega(\frac{1}{log(T)})$.

Abajo, se ve pseudo-código de ϵ-greedy y como se implementará el código en esta investigación.

Algorithm 1 ϵ-greedy

procedure ϵ-GREEDY
  initialize $\mu_{t,k} \quad \forall t \in T, k \in K$
  initialize $\hat{\mu}_{1,0}, .., \hat{\mu}_{K,0} = 0{,}5$
  initialize $n_{k,t} = 0 \quad \forall t \in T, k \in K$
  for t in 1..T **do**
    if variant = 1 **then**
      $\epsilon = 0{,}5$
    else if variant = 2 **then**
      $\epsilon = \frac{1}{\sqrt{t}}$
    else if variant = 3 **then**
      $\epsilon = \frac{1}{log(t)}$
    Set $\hat{\mu}^* = argmax_{k\in K}\hat{\mu}_{k,t}$
    Draw $j(t)$ with $\mathbb{P}(j = k, t^*) = 1 - \epsilon + \frac{\epsilon}{K}$, $\mathbb{P}(j \neq k, t^*) = \frac{\epsilon}{K}$
    Set $n_{k,t+1} = \begin{cases} n_{k,t}+1 & \text{si } k = j(t) \\ n_{k,t} & \text{si no} \end{cases}$
    Draw reward from $r_{j(t)} \sim P_j$
    $\hat{\mu}_{k,t+1} = \begin{cases} \mu_{k,t}\frac{n_{k,t}}{n_{k,t}+1} + \frac{r_{j(t)}}{n_{k,t}+1} & \text{si } k = j(t) \\ \mu_{k,t} & \text{si no} \end{cases} \quad \forall k \in K$

2.6. UCB

El algoritmo `UCB`, que es una abreviación para "Upper Confidence Bound", es un algoritmo donde se utiliza intervalos de confianza para determinar cuál brazo se elegirá en cada iteración. Fue propuesto por Auer, Cesa-Bianchi & Fisher en 2002, y se describe la idea como optimismo bajo inseguridad. [3]

Suponga que se elige el algoritmo para resolver el problema. Dado los valores iniciales de $\hat{\mu}_{1,(0)}, .., \hat{\mu}_{K,(0)}$, en cada iteración t se elige un brazo según la formula

$$y_{k,t} = \begin{cases} 1 & \text{si } i = argmax_{k \in K} \hat{\mu}_{k,t} + \sqrt{\frac{2log(t)}{n_{k,t}}} \\ 0 & \text{si no} \end{cases} \quad \forall t \in T, k \in K \tag{1}$$

El término $\sqrt{\frac{2log(t)}{n_{k,t}}}$ es para obtener el tope del intervalo de confianza, que se describe como $\hat{\mu}_{k,t} \pm \sqrt{\frac{2log(t)}{n_{k,t}}}$. Si un brazo no ha sido elegido muchas veces, $n_{k,t}$ será muy pequeño, haciendo la fracción más larga, mientras $2log(t)$ va creciendo por cada iteración. Esto resultará en que el algoritmo querrá investigar los menos investigados por cada iteración. Además, lo más que crezca $n_{k,t}$, lo menor será la fracción, que indica una seguridad mayor del valor real de μ_k, y lo más que se pulsa cada brazo, se obtiene un intervalo menor. Al mismo tiempo, se elige siempre la más alta, que será el brazo que uno es más inseguro sobre, o el brazo que realmente es el mejor. Es de esto que el algoritmo deriva su denominación "Optimismo bajo inseguridad".

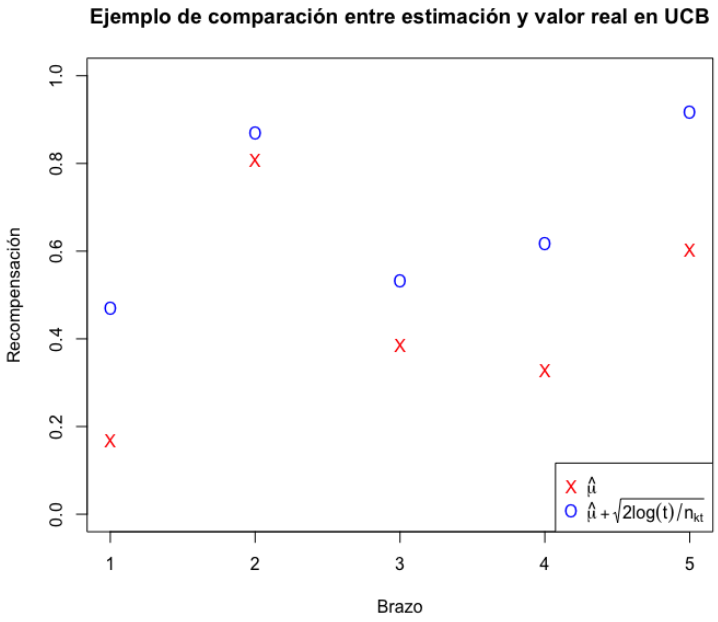


Figura 2: Ejemplo de cómo elegiría el UCB. Aquí, el valor real de la remuneración media es la más alta para brazo 2, pero se elegiría brazo 5 por UCB por la inseguridad

Se ha probado antes que el arrepentimiento esperado del algoritmo `UCB` puede ser limitado por un límite superior. [1] Si denominamos, para todos los brazos peores que el mejor brazo μ^*, $\delta_k = \mu^* - \mu_k \forall k \in \{k : \mu_k < \mu^*\}$ como la diferencia entre la remuneración promedia del mejor brazo y el brazo k, y $\Delta = \sum_{k=1}^{K} \delta_k$ podemos definir un límite. Este límite se define como

$$\mathbb{E}[R_T] \leq 8 \sum_{i \neq k: \mu_k = \mu^*} \Big(\frac{log(T)}{\delta_i} \Big) + \Big(1 + \frac{\pi^2}{3} \Big) \Big(\sum_{j=1}^{K} \mu^* - \mu_j \Big) \leq$$
$$\leq \frac{klog(T)}{\Delta} + \Big(1 + \frac{\pi^2}{3} \Big) \Big(\sum_{j=1}^{K} \mu^* - \mu_j \Big) = \mathcal{O}(\frac{Klog(T)}{\Delta}) \Leftrightarrow$$
$$\Leftrightarrow \mathbb{E}[R_T] = \mathcal{O}(\tfrac{Klog(T)}{\Delta}) = \mathcal{O}(log(T)) \tag{2}$$

El término $\Big(1 + \frac{\pi^2}{3} \Big) \Big(\sum_{j=1}^{K} \mu^* - \mu_j \Big)$ resulta no ser importante, ya que se considera un constante en términos de T. Con esto, podemos probar facilmente que si $\mathbb{E}[R_T] = \mathcal{O}(log(T))$, tenemos que $\mathbb{E}[R_T/T] = \mathcal{O}(\frac{log(T)}{T})$ que cumple que $\mathbb{E}[R_T/T] \to 0$ as $T \to \infty$, cumpliendo el requerimiento de *zero regret*. Por ende, se puede concluir que este algoritmo teóricamente resolverá el problema del bandido multibrazo. La eficiencia real queda ver en los experimentos.

Abajo, se ve pseudocódigo mostrando cómo funciona exactamente el `UCB` que se implementará.

Algorithm 2 UCB

procedure UCB
  initialize $\mu_{t,k} \quad \forall t \in T, k \in K$
  initialize $\hat{\mu}_{1,0}, .., \hat{\mu}_{K,0} = 0{,}5$
  initialize $n_{k,t} = 0 \quad \forall t \in T, k \in K$
  for t in 1..T **do**
    Set $j(t) = argmax_{j \in K} \hat{\mu}_{k,t} + \sqrt{\frac{2log(t)}{n_{k,t}}}$
    $n_{k,t+1} = \begin{cases} n_{k,t} + 1 & \text{si } k = j(t) \\ n_{k,t} & \text{si no} \end{cases}$
    Draw reward from $r_{j(t)} \sim P_j$
    $\hat{\mu}_{k,t+1} = \begin{cases} \mu_{k,t} \frac{n_{k,t}}{n_{k,t}+1} + \frac{r_{j(t)}}{n_{k,t}+1} & \text{si } k = j(t) \\ \mu_{k,t} & \text{si no} \end{cases} \qquad \forall k \in K$

2.7. Exploración de Boltzmann (SoftMax)

Otro algoritmo muy popular dentro aprendizaje por refuerzo para resolver el problema de bandidos multibrazo es la exploración de Boltzmann, también conocido como `SoftMax`. Este algoritmo también es simple, aunque es un poco menos intuitivo que ϵ-greedy. Dado los valores iniciales de $\hat{\mu}_{1,(0)}, .., \hat{\mu}_{K,(0)}$, en cada iteración t se elige un brazo por las distribuciones $p_i(t)$, que son calculadas según la formula

$$p_{i,t} = \frac{e^{\eta_t \hat{\mu}_{i,t}}}{\sum_{i=1}^{K} e^{\eta_t \hat{\mu}_{t,k}}}, \ \forall t \in T, k \in K$$

donde η_t es el rato de aprendizaje. Este rato es extremadamente importante para garantizar que $\mathbb{E}[R_T/T] \to 0$ cuando $T \to \infty$. En 2002, Cesa-Bianchi y Gentile lograron construir un variante de Bolzmann, SoftMix, que cumple el requerimiento de obtener un límite superior del R_T, que dice que

$$R_T \leq \frac{\Delta}{\delta_{min}} \Big(8log \frac{5K}{\delta_{min}^2} + \frac{5}{2} log^2 T \Big) + \Delta log(T) = \Omega(log(T) + log^2 T) \tag{1}$$

No obstante, esto es para una versión mejorada del algoritmo `SoftMax`, y todavía falta prueba teórica de si el algoritmo básico de Bolzmann tiene la propiedad de zero-regret. [1, 2] Cesa-Bianchi y Gentile muestran en 2002 que se puede obtener un límite superior del arrepentimiento

$R_T \leq \frac{16eKlog^2T}{\delta^2_{min}} + \frac{9K}{\delta^2_{min}}$ por poner $\eta_t = \mathbb{1}(t < \tau) + \frac{log(T)\delta^2_{min}}{\delta_{min}}\mathbb{1}(t \geq \tau)$, $\tau = \frac{16eKlog^2T}{\delta^2_{min}}$. [2] Sin embargo, este variante del algoritmo requiere conocimiento completo de todas las remuneraciones promedias, algo que no se tiene bajo circunstancias normales. Por ende, es una falta en la literatura y investigamos empíricamente en este artículo el crecimiento del algoritmo Bolzmann clásico. Artículos anteriores han propuesto un $\eta_t = \frac{c}{t}$, donde c es un constante que se cambia. [7] Investigaremos empíricamente si el arrepentimiento crece en una manera lineal, o si, lo mejor de los casos, crece logarítimicamente.

Abajo, se muestra pseudocódigo del algoritmo que implementaremos.

Algorithm 3 Boltzmann (SoftMax)

procedure UCB
  initialize $\mu_{t,k} \quad \forall t \in T, k \in K$
  initialize $\hat{\mu}_{1,0}, .., \hat{\mu}_{K,0} = 0{,}5$
  Set c
  for t in 1..T **do**
    Set $\eta_t = \frac{c}{t}$
    $p_{i,t} = \frac{e^{\eta_t \hat{\mu}_{i,t}}}{\sum_{i=1}^{K} e^{\eta_t \hat{\mu}_{t,k}}},\ k \in K$
    Draw j(t) from $I \sim Multinomial(1; p_{1,t}, ..., p_{K,t})$
    Draw $r_{j(t)} \sim P_k$
    $n_{k,t+1} = \begin{cases} n_{k,t} + 1 & \text{si } k = j(t) \\ n_{k,t} & \text{si no} \end{cases}$
    Calculate $\hat{\mu}_{k,t+1} = \begin{cases} \mu_{k,t}\frac{n_{k,t}}{n_{k,t}+1} + \frac{r_{j(t)}}{n_{k,t}+1} & \text{si } k = j(t) \\ \mu_{k,t} & \text{si no} \end{cases} \quad \forall k \in K$

3. Método

Para llegar a respuestas a las preguntas definidas en el propósito, correremos un número de instancias de cada algoritmo bajo de diferentes circunstancias para confirmar lo que dice la teoría. Las circunstancias variarán dependiente de los híperparámetros del problema; es decir, el número de brazos K y la varianza de las remuneraciones σ^2. Las remuneraciones medias para los brazos serán elegidos uniformemente al azar pero con una semilla aleatoria para poder reproducir los resultados.
El programa será implementado en R, un lenguaje de programación para diseñado para investigaciones estadísticas. Por cada combinación de valores de K y σ^2, se hará 2000 iteraciones 1000 veces. De estas instancias, calcularemos el promedio del arrepentimiento y el número de veces que se ha pulsado el mejor brazo para obtener 3 gráficos para estudiar.

1. El arrepentimiento promedio en cada iteración, para poder estudiar cómo convergen los algoritmos.

2. El arrepentimiento cumulativo, para poder estudiar cómo desacelera el arrepentimiento total en su crecimiento.

3. El número de veces total que se elige el mejor brazo para estudiar cómo cambia esta.

3.1. Ajustamiento de parámetros

Se ha probado en trabajos anteriores que los únicos parámetros de la formulación del problema que importan son el K y la varianza de las remuneraciones. [3] Por ende, nos limitamos a

ajustar estos parámetros para cambiar las condiciones del problema. Se ha probado antes que el problema es difícil de resolver con más de 50 brazos. [3] Después de esto, la complejidad crece muy rápido. Por ende, hacemos pruebas para valores de $K \in \{2, 5, 10, 25, 50\}$, así bajo de 50. Para la varianza σ^2 tendremos los valores $\sigma^2 \in \{0{,}01^2, 0{,}1^2, 1^2, 10^2\}$. Una varianza mayor indica una dificultad mayor a diferir entre los brazos por las remuneraciones dadas.

Para ϵ-greedy, se probará 3 variaciones por cambiar la manera de elegir ϵ. El primer es el más simple y básico, con el valor de ϵ fijado a 0.5. Este variante servirá como un punto de referencia de un algoritmo primitivo y simple, que debería ser el peor de todos. Si un algoritmo resulta ser peor o similar a esta, indica que el algoritmo prácticamente no cumple el requerimiento de *zero regret*, aunque debería según la teoría. En el segundo y tercer variante, se determinará ϵ según las fórmulas $\epsilon = \frac{1}{\sqrt{t}}$ y $\frac{1}{log(t)}$, para dejarlo converger a solamente explorar por fin. El primer, convergerá más rápido para solamente explotar debido al hecho que $\sqrt{t}$ crece más rápido que log(T). Si esto es mejor o peor queda ver en los experimentos.

Para el algoritmo `SoftMax`, probaremos dos instancias; una con $\eta_t = \frac{10}{t}$ y otra con $\eta_t = 0{,}9$. Con esto, se puede determinar si realmente hace una diferencia entre tener un rato de aprendizaje constante y uno disminuyendo. Si los dos tienen el mismo comportamiento, esto indicaría que la convergencia del algoritmo `SoftMax` no es afectado mucho por cambiar el rato de aprendizaje en este problema específico. Esto resultaría en que hay pocas maneras de afectar la convergencia, que resultaría en que `SoftMax` no sería un algoritmo adecuado en esta situación.

3.2. Crítica del método

Es importante notar que el método usado está lejos de perfecto. Primero, la eficiencia de algoritmos cambia mucho dependiente de la estructura del problema. Tener remuneraciones como las en este variante del problema, $r_k \sim \mathcal{N}(\mu_k, \sigma^2)$ es una suposición fuerte, y varía mucho del problema. Es una suposición simple utilizado muchas veces, ya que muchas distribuciones por fin pueden ser aproximados por distribuciones normales, pero en la vida real, las distribuciones de los algoritmos pueden variar mucho. Tener la misma varianza para todos los brazos también es una suposición muy fuerte, que hace que estas instancias tal vez no serán representativas para versiones reales del problema, por ejemplo para ensayos clínicos optimización de elección de anuncios.

4. Marco empírico

En esta sección se presentará los resultados de los experimentos. Se corrió todo como planificado, sino los del algoritmo de `SoftMax` para $\sigma = 10$, debido a que estas instancias generaron errores que paró el programa. Esto fue causado por el hecho que generaron probabilidades distribuidas tan desproporcionadas, que el muestreo fue imposible con las funciones normales en R. No obstante, ya se vio resultados iguales y interesantes en corridos anteriores, así no había necesidad de correr estas últimas instancias.

4.1. Gráficos

Todos los 48 gráficos se puede encontrar en el apéndice. Abajo, se ve una guía a cuál línea corresponde a cuál algoritmo.

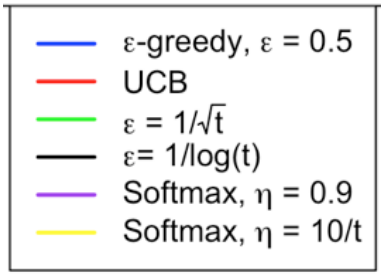


Figura 3: Colores de las líneas de los algoritmos en los grafos.

5. Análisis

En esta sección se analiza los resultados de las instancias de los algoritmos, y respondemos a las preguntas. Se puede ver por los gráficos cuáles algoritmos son los mejores y peores. Aunque los resultados cambian claramente de los valores de K y σ, la comparación entre los algoritmos cambia poco. Se puede sumarlo con decir que la teoría es, en muchas maneras, confirmada por estos experimentos. No obstante, una sorpresa es que `UCB` es peor que los variantes de ϵ-greedy que tiene un ϵ disminuyendo en la mayoría de los casos.

5.1. Arrepentimiento promedio

Estudiando los gráficos del arrepentimiento promedio, se puede ver unas cosas. Mientras ϵ-greedy con ϵ disminuyendo rápidamente converge por poder elegir el mejor brazo rápidamente, UCB parece disminuir muy rápido, y parece llegar, después de un tiempo, al mismo nivel que los de ϵ. En particular, cuando $K = 50$ and $\sigma = 10$, las circunstancias más complejas, UCB parece ser mejor que $\epsilon = \frac{1}{log(T)}$. No obstante, durante circunstancias más simples, es peor, debido a su tendencia de querer explorar los brazos que no se sabe mucho de. Esto indica que UCB puede ser mejor durante las circunstancias más complejas pero peor que los más simples, pero esto queda estudiar más.

El arrepentimiento promedio para SoftMax es constante, indicando que no convergerá a 0, otra vez indicando que no tiene la propiedad de *zero-regret*.

5.2. Arrepentimiento cumulativo

Estudiando los gráficos del arrepentimiento cumulativo, se ve claramente que ϵ-greedy con ϵ disminuyendo tiene un arrepentimiento cumulativo que parece crecer en un rato logarítmico, es decir, $\mathcal{O}(log(T))$. Esto confirma la teoría presentada en sección 2.1, que estos algoritmos por fin elegirá el mejor brazo siempre, teniendo la propiedad de *zero regret*. Los gráficos claramente nos muestran que ϵ-greedy con $\epsilon = \frac{1}{\sqrt{t}}$ tiene el arrepentimiento menor bajo de estas circunstancias, independientemente de los valores de K y σ.

El algoritmo `SoftMax` tiene los peores resultados en todos las instancias, mostrando un crecimiento que parece ser lineal. Aunque no es confirmado, este resultado indica fuertemente que no tiene la propiedad de *zero regret*. Aún queda mostrar esto teóricamente, pero esto nos da una indicación clara que el algoritmo `SoftMax`, bajo de las circunstancias dadas, crece con un rato $\mathcal{O}(T)$. Esto indica que para usar este método para resolver el problema, probablemente se necesite mejorarlo, como hicieron Cesa- Bianchi y Gentile. [2] Los gráficos que muestran que el arrepentimiento promedio no disminuye para `SoftMax` con un η_t constante, sino incrementa y converge dentro de aproximadamente 500 iteraciones en todos los casos. Esto indica claramente que nunca convergerá a 0.

Se ve también, estudiando los gráficos, que `UCB` es en la mayoría de los casos, peor que los variantes de ϵ-greedy. No obstante, el crecimiento de R_T también parece logarítmico, indicando y confirmando la propiedad de *zero regret*. Esto nos muestra que no necesariamente es el algoritmo más complicado y sofisticado que genera los mejores resultados. `UCB` parece ser mejor cuando se necesita mucha exploración, ya que el algoritmo tiene la tendencia de siempre elegir el brazo que se siente menos seguro sobre. Por ende, una teoría que puede ser interesante investigar es, como dicho antes, investigar los algoritmos durante circunstancias más complejas; por ejemplo con un K mayor, para determinar si UCB será el mejor algoritmo.

5.3. Veces elegido el mejor brazo

Estudiando la cantidad de veces que se elige el mejor brazo, se ve que los algoritmos de $\epsilon = \frac{1}{\sqrt{t}}$ y $\epsilon = \frac{1}{log(T)}$ son los mejores en la mayoría de los casos. No obstante, UCB supera $\epsilon = \frac{1}{log(T)}$ cuando $\sigma \geq 1$ muchas veces, indicando que el incertidumbre causado por la varianza pone el UCB en favor, debido a una necesidad de explorar más. Entonces, no basta con algoritmos tan simples como ϵ-greedy. No obstante, un crecimiento de K empeora el resultado de UCB significativamente, indicando que UCB es bueno para K pequeños con una varianza grande.

Otra vez, SoftMax resulta ser el peor de todos, también peor que el variante de ϵ-greedy con $\epsilon = 0{,}5$.

6. Conclusión

En este artículo, hemos estudiado tres algoritmos que se ha usado antes para resolver el problema del bandido multibrazo. Hicimos un estudio teórico, y luego hicimos experimentos para confirmar las teorías. Resumiendo los resultados, se puede concluir que ϵ-greedy con un $\epsilon = \frac{1}{\sqrt{t}}$ es sin duda el mejor en este variante del problema. Esto es, probablemente, debido a la estructura simple del problema, que está a favor de los algoritmos que rápidamente decide a solamente elegir el brazo mejor. No obstante, cabe mencionar que UCB probablemente será mejor durante circunstancias más complejas, cuando el problema requiere más exploración. El ϵ-greedy podría probablemente tener resultados mejores en situaciones más complejas también por ajustar los parámetros, como el c en $\epsilon = \frac{c}{\sqrt{t}}$. Esto queda investigar en obras futuras.

7. Referencias

Referencias

[1] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

[2] Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *CoRR*, abs/1705.10257, 2017.

[3] Precup D. Kuleshov V. Algorithms for the multi-armed bandit problem. *Journal of Machine Learning Research*, 1(1-48), 2000.

[4] T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.*, 6(1):4–22, March 1985.

[5] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 661–670, New York, NY, USA, 2010. ACM.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[7] Joannès Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *Proceedings of the 16th European Conference on Machine Learning*, ECML'05, pages 437–448, Berlin, Heidelberg, 2005. Springer-Verlag.

[8] Yizao Wang, Jean yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1729–1736. Curran Associates, Inc., 2009.

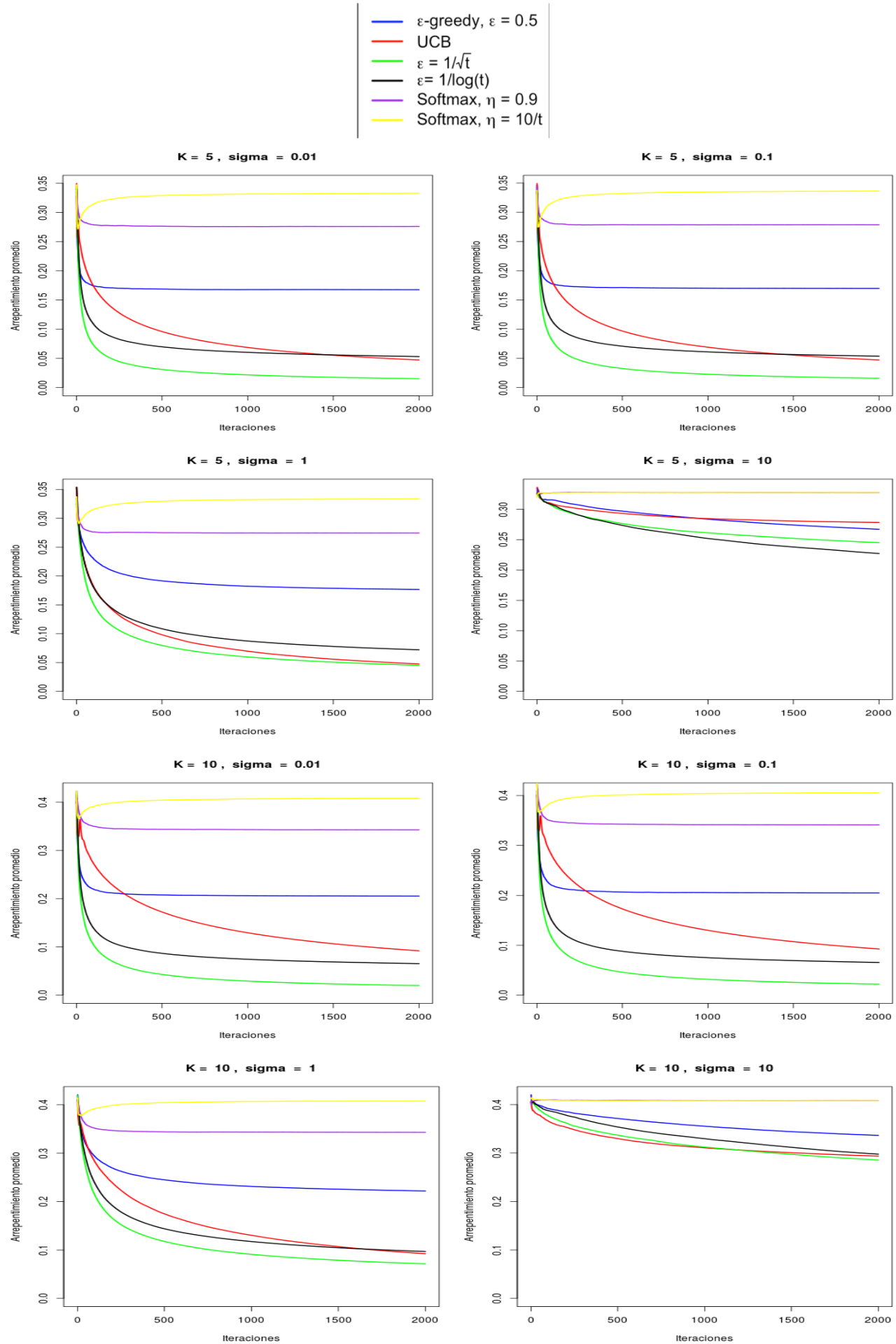
[9] Kosorok M. R. Zhao, Y. and D Zeng. Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28(3294–3315), 2009.

8. Apéndices - gráficos de resultados

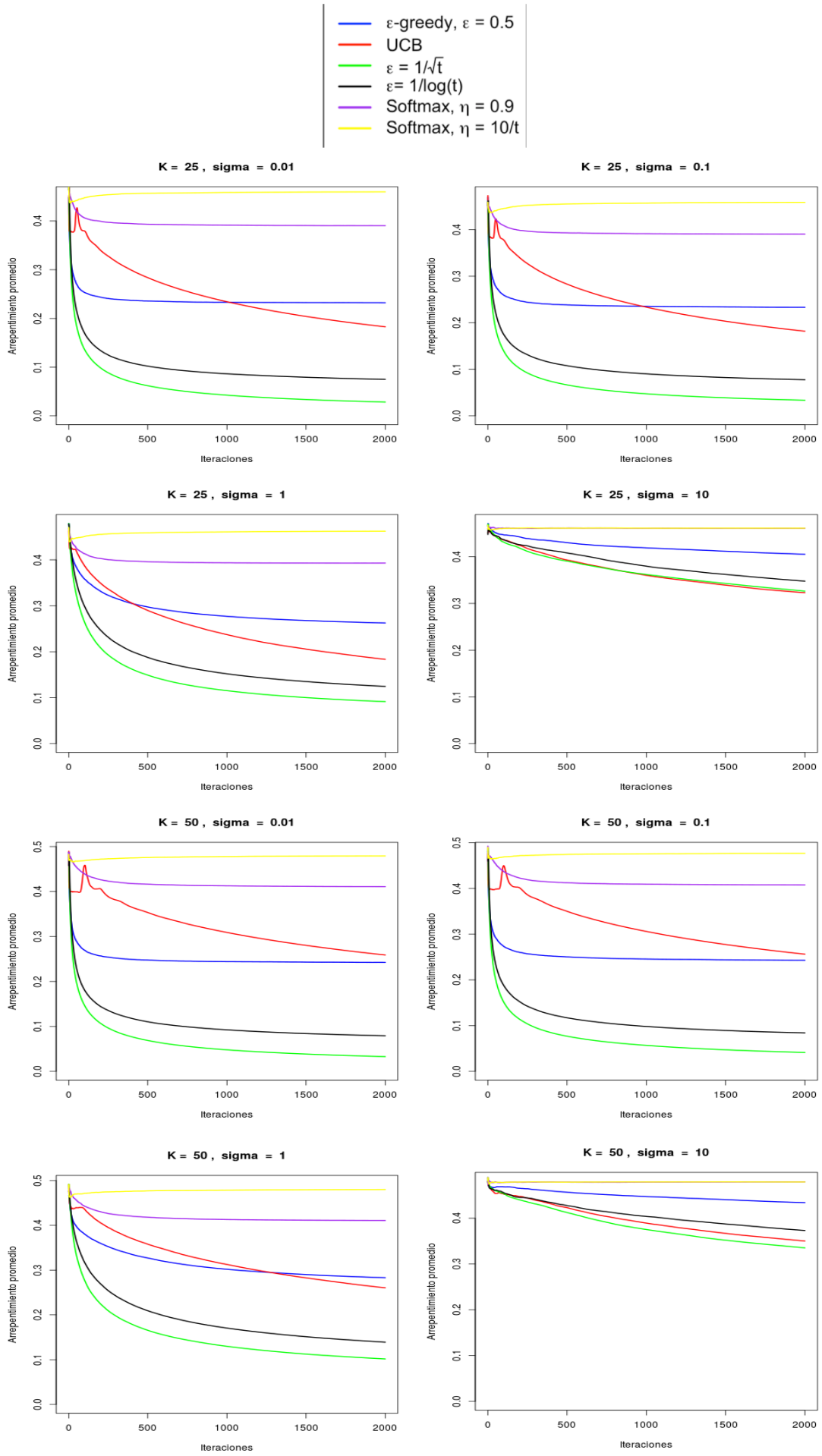
En las próximas páginas, incluimos los gráficos de los resultados. Después se incluye el código hecho en R.

8.1. Apéndice 1 - Gráficos de resultados

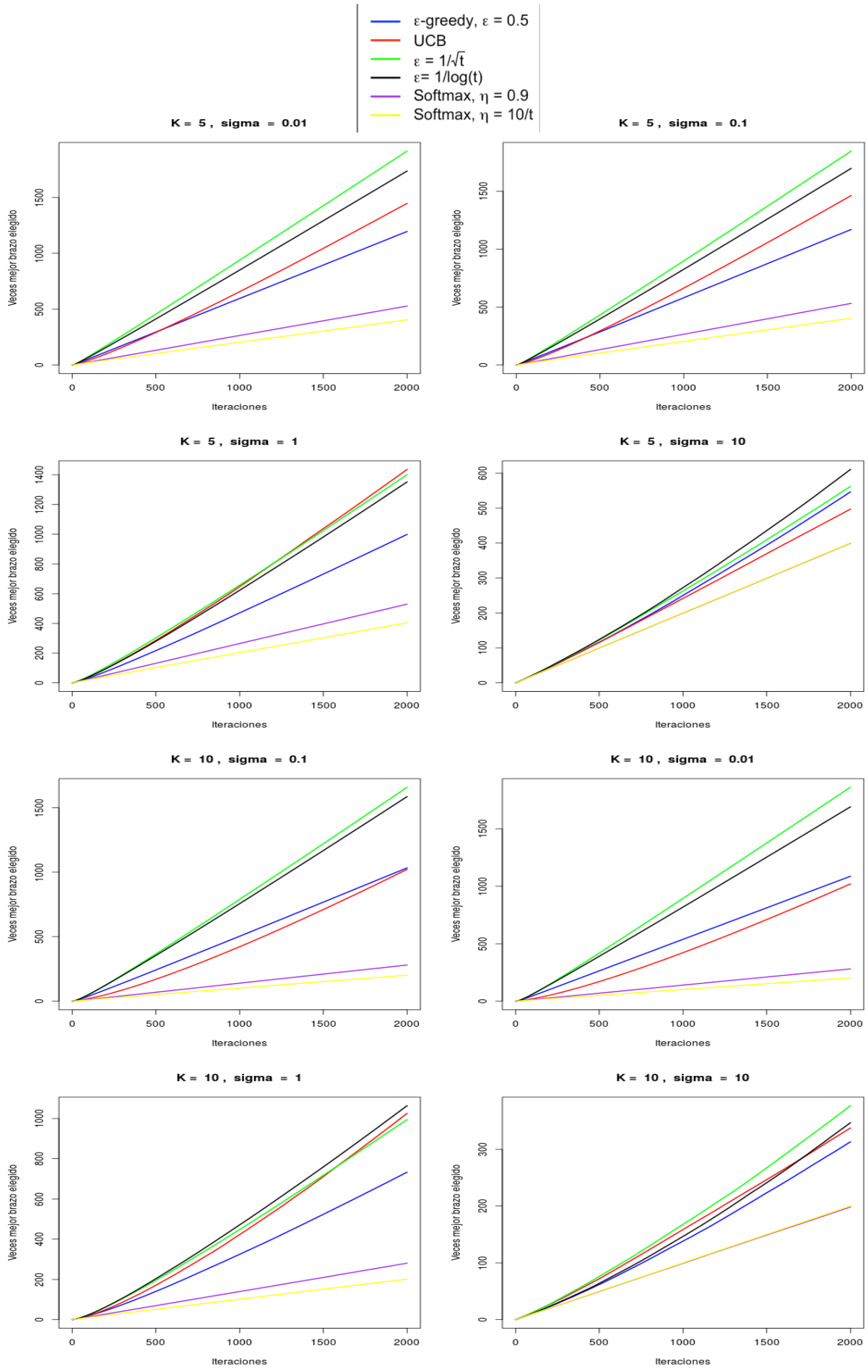
Arrepentiemiento promedio, K = 5 y 10



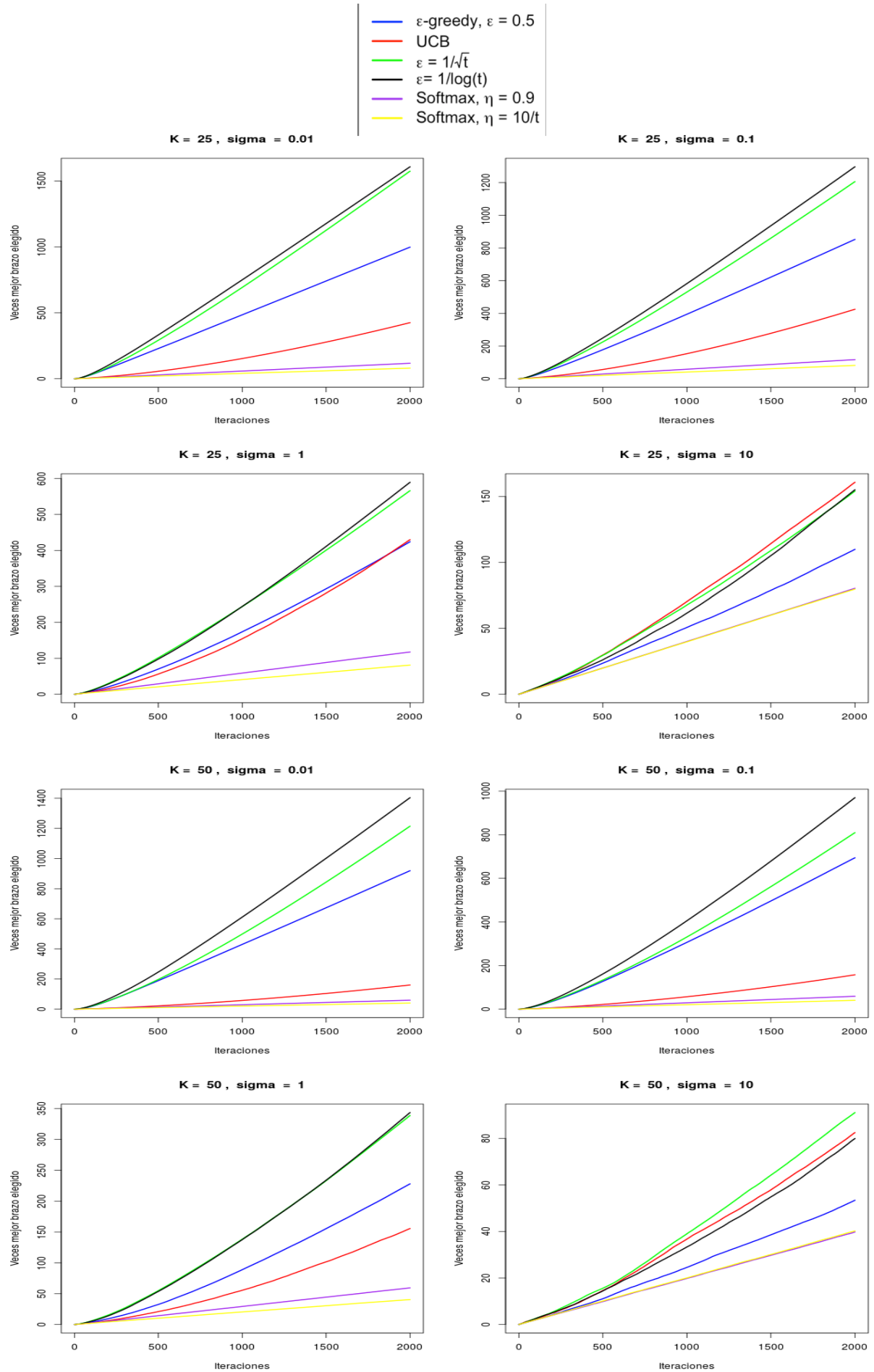
Arrepentiemiento promedio, K = 25 y 50



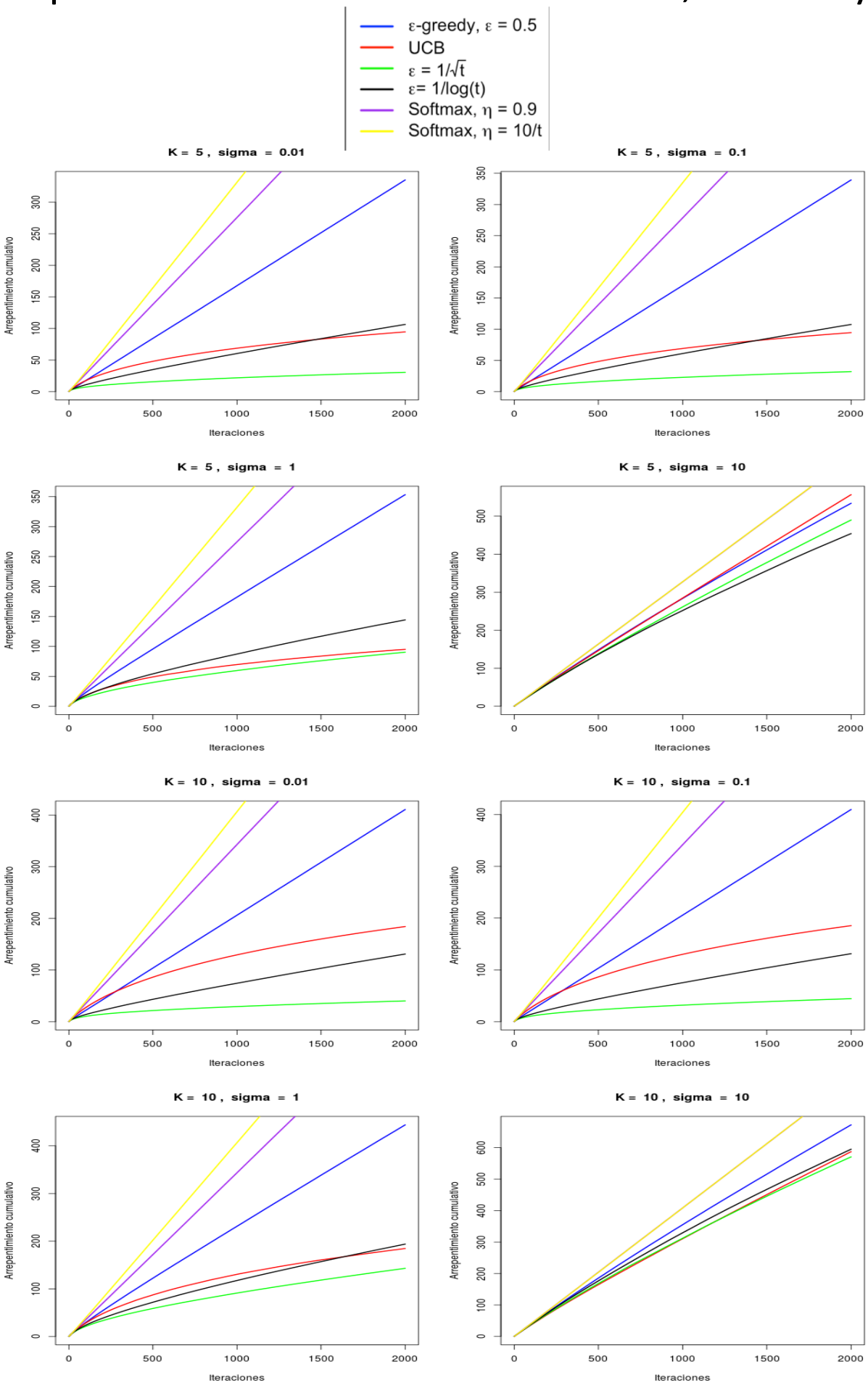
Veces mejor brazo elegido, K = 0.01 y 0.1



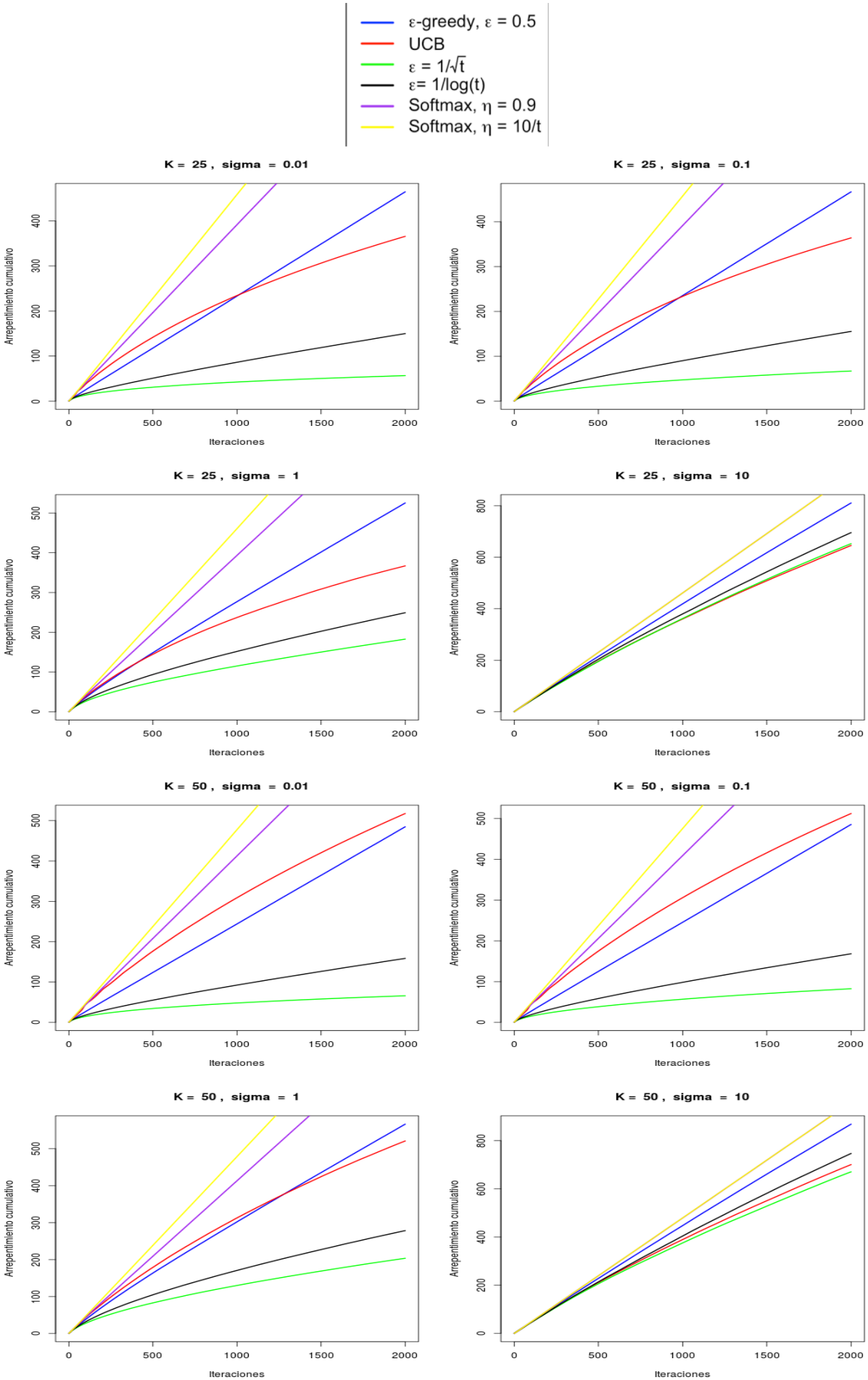
Veces mejor brazo elegido, K = 25 y 50



Arrepentiemiento cumulativo, K = 5 y 10



Arrepentiemiento cumulativo, K = 25 y 50



8.2. Apéndice 2 - Código de los experimentos

```
#####################################
######## Initialize functions ######
#####################################

Get_Convergence = function(mus) {
  sortedMaxRewards = sort(mus, decreasing = TRUE)

  return(sortedMaxRewards[1] - sortedMaxRewards[2])
}
normalize = function(vec) {

  print(sd(vec))
  if (sd(vec) != 0) {
    return((vec - mean(vec)/sd(vec)))
  } else {
    return(vec)
  }

}
Convergence_Means <- function (samples) {
  means <- numeric()
  for (i in 1:length(samples)) {
    means[i] <- mean(samples[1:i])
  }
  return(means)
}

ret_epsilon = function(mode = 1, t = NULL, c = 1) {
  if (mode == 1) {
    return(0.5)
  } else if (mode == 2) {
    return(min(1,c/sqrt(t)))
  } else if (mode == 3) {
    return(min(1,c/log(t)))
  } else if (mode == 4) {
    # Another suggestion?
  }
}

epsilonGreedy = function(k, mus, sigma2) {

  epsilon = ret_epsilon(mode = 1)

  # Set all mu estimates to 0.5 first.
  muEstimates = rep(0.5, k)

  # Set probability to pick each arm to epsilon/k,
  probabilities = rep(epsilon/k, k)

  # Set the best arm's probability to 1 - epsilon + e/k, i.e. add 1 - epsilon
  # In this case, all
  # Initialize vectors to store information
  rewards = as.numeric()
  armChosen = as.numeric()
  t = 1
  while (t <= T) {

    # update epsilon
    epsilon = ret_epsilon(epsilonMode, t = t)


    # Sample an arm
    armChosen[t] = sample(seq(1, k, 1), size = 1, prob = probabilities)
```

```
    rewards[t] = rnorm(n=1,mean = mus[armChosen[t]], sd = sqrt(sigma2))


    # Update muEstimates
    muEstimates[armChosen[t]] = sum(rewards[armChosen == armChosen[t]])/sum(
    armChosen == armChosen[t])

    # Update probabilities
    probabilities = rep(epsilon/k, k)
    # Set the best arm's probability to 1 - epsilon + e/k, i.e. add 1 - epsilon
    probabilities[which.max(muEstimates)] = probabilities[which.max(muEstimates)]
     + 1 - epsilon

    t = t + 1
  }

  data = data.frame(armChosen, rewards)
  colnames(data) = c("ArmChosen", "Rewards")
  return(data)
}

SoftMax = function(k, mus, sigma2, c = NULL) {
  muEstimates = rep(0.5, k)

  # Initialize vectors to store information
  rewards = as.numeric()
  armChosen = as.numeric()
  n_k_t = rep(0,k)

  for (t in 1:T) {
    if (!is.null(c)) {
      eta_t = c/t
    } else {
      eta_t = 0.9
    }


    probs = exp(eta_t*muEstimates)


    if(any(is.na(probs))) {
      print(probs)
      print(muEstimates)
      muEstimates = normalize(muEstimates)
      print(muEstimates)
      probs = exp(eta_t*muEstimates)/sum(exp(eta_t*muEstimates))
      print(probs)
    }
    armChosen[t] = sample(seq(1,k,1), size = 1, prob = probs)
    rewards[t] = rnorm(n=1,mean = mus[armChosen[t]], sd = sqrt(sigma2))
    n_k_t[armChosen[t]] = n_k_t[armChosen[t]] + 1
    muEstimates[armChosen[t]] = sum(rewards[armChosen == armChosen[t]])/(n_k_t[
    armChosen[t]] + max(0, 1 - n_k_t[armChosen[t]]))
  }
  data = data.frame(armChosen, rewards)
  colnames(data) = c("ArmChosen", "Rewards")
  return(data)
}


UCB = function(k, mus, sigma2) {

  muEstimates = rep(0.5, k)

```

```
  # Initialize vectors to store information
  rewards = as.numeric()
  armChosen = as.numeric()
  n_k_t = rep(0,k)

  t = 1
  while (t <= T) {

    # Generate the ucbs. We need the indices for this, hence the for loop.
    ucbs = as.numeric()
    for (i in 1:k) {

      ucbs[i] = muEstimates[i] + sqrt(2*log(t)/(n_k_t[i] + max(0, 1 - n_k_t[i])))
    }

    # Choose arm from the one with highest UCB
    armChosen[t] = which.max(ucbs)
    # update n_k_t
    n_k_t[armChosen[t]] = n_k_t[armChosen[t]] + 1

    # Sample reward
    rewards[t] = rnorm(n=1, mean = mus[armChosen[t]], sd = sqrt(sigma2))


    # Update muEstimates

    muEstimates[armChosen[t]] = sum(rewards[armChosen == armChosen[t]])/(n_k_t[
    armChosen[t]] + max(0, 1 - n_k_t[armChosen[t]]))

    t = t + 1

  }

  data = data.frame(armChosen, rewards)
  colnames(data) = c("ArmChosen", "Rewards")
  return(data)
}

calc_regret = function(muStar, armChosen, mus) {

  return(apply(as.matrix(armChosen), 2, function(row, muStar) {
    return(muStar - mus[armChosen])
  }, muStar))
}

cumRegret = function(regrets) {
  cumRegrets = as.numeric()
  for (i in 1:length(regrets)) {
    cumRegrets[i] = sum(regrets[1:i])
  }
  return(cumRegrets)
}

analyze = function(k, sigma2, regretsEps1, regretsEps2, regretsEps3, regretsUCB,
    regretsSM1, regretsSM10, avgBestArmEps1, avgBestArmEps2, avgBestArmEps3,
    avgBestArmUCB, avgBestArmSM1, avgBestArmSM10) {
  xSeq = seq(1, length(regretsEps1),1)
  plot(x=xSeq, y = Convergence_Means(regretsEps1), type = "l", lwd = 2, xlab = "
    Iteraciones", ylab = "Arrepentimiento promedio", col = "blue", ylim = c(0,max
    (Convergence_Means(regretsEps1), Convergence_Means(regretsUCB))),main = paste
    ("K = ",k, ", ", expression(sigma), " = ", sigma2))
  lines(Convergence_Means(regretsUCB), lwd = 2, col = "red")
  lines(Convergence_Means(regretsEps2), lwd = 2, col = "green")
  lines(Convergence_Means(regretsEps3), lwd = 2, col = "black")
  lines(Convergence_Means(regretsSM1), lwd = 2, col = "purple")
```

```
  lines(Convergence_Means(regretsSM10), lwd = 2, col = "yellow")
  cumulativeRegretEpsilon1 = cumRegret(regretsEps1)
  cumulativeRegretEpsilon2 = cumRegret(regretsEps2)
  cumulativeRegretEpsilon3 = cumRegret(regretsEps3)
  cumulativeRegretUCV = cumRegret(regretsUCB)
  cumulativeRegretSM1 = cumRegret(regretsSM1)
  cumulativeRegretSM10 = cumRegret(regretsSM10)

  legend("top",
         col = c("blue", "red", "green", "black", "purple", "yellow"),
         lwd = c(2,2),
         legend = c(expression(paste(epsilon,"-greedy, ", epsilon, " = 0.5")),
                    "UCB",
                    expression(paste(epsilon," = 1/",sqrt(t))),
                    expression(paste(epsilon, "= 1/log(t)")),
                    expression(paste("Softmax, ",eta," = 0.9")),
                    expression(paste("Softmax, ",eta," = 10/t"))
         ))
  plot(x=xSeq, y = cumulativeRegretEpsilon1, type = "l", lwd = 2, xlab = "
    Iteraciones", ylab = "Arrepentimiento cumulativo", col = "blue", ylim = c(min
    (cumulativeRegretUCV, cumulativeRegretEpsilon1),max(cumulativeRegretUCV,
    cumulativeRegretEpsilon1)),main = paste("K = ",k, ", ", expression(sigma), "
    = ", sigma2))
  lines(cumulativeRegretUCV, lwd = 2, col = "red")
  lines(cumulativeRegretEpsilon2, lwd = 2, col = "green")
  lines(cumulativeRegretEpsilon3, lwd = 2, col = "black")
  lines(cumulativeRegretSM1, lwd = 2, col = "purple")
  lines(cumulativeRegretSM10, lwd = 2, col = "yellow")
  legend("top",
         col = c("blue", "red", "green", "black", "purple", "yellow"),
         lwd = c(2,2),
         legend = c(expression(paste(epsilon,"-greedy, ", epsilon, " = 0.5")),
                    "UCB",
                    expression(paste(epsilon," = 1/",sqrt(t))),
                    expression(paste(epsilon, "= 1/log(t)")),
                    expression(paste("Softmax, ",eta," = 0.9")),
                    expression(paste("Softmax, ",eta," = 10/t"))
                    ))

  ## Plot best arm

  plot(x=xSeq, y = avgBestArmEps1, type = "l", lwd = 2, xlab = "Iteraciones",
    ylab = "Veces mejor brazo elegido", col = "blue", ylim = c(0,max(
    avgBestArmEps1, avgBestArmEps2, avgBestArmEps3, avgBestArmUCB)),main = paste(
    "K = ",k, ", ", expression(sigma), " = ", sigma2))
  lines(avgBestArmUCB, lwd = 2, col = "red")
  lines(avgBestArmEps2, lwd = 2, col = "green")
  lines(avgBestArmEps3, lwd = 2, col = "black")
  lines(avgBestArmSM1, lwd = 2, col = "purple")
  lines(avgBestArmSM10, lwd = 2, col = "yellow")
  legend("top",
         col = c("blue", "red", "green", "black", "purple", "yellow"),
         lwd = c(2,2),
         legend = c(expression(paste(epsilon,"-greedy, ", epsilon, " = 0.5")),
                    "UCB",
                    expression(paste(epsilon," = 1/",sqrt(t))),
                    expression(paste(epsilon, "= 1/log(t)")),
                    expression(paste("Softmax, ",eta," = 0.9")),
                    expression(paste("Softmax, ",eta," = 10/t"))
         ))
}

bestArmPlayed = function(armPlayed, mus) {
  percArmPlayed = as.numeric()
  for (i in 1:length(armPlayed)) {
```

```
    tmp = armPlayed[1:i]
    percArmPlayed[i] = sum(tmp == which.max(mus))
  }
  return(percArmPlayed)
}

##############################################
######## Initialize and run problem #########
##############################################

set.seed(123)
# Set parameters

K = c(5,10,25,50)

sigma2 = c(0.01^2,0.1^2,1^2, 10^2)

## Set algorithm specifications
# Set epsilon mode. Sets which way the epsilon is decided. 1 for fixed, 2 for epsilon = 1/t, 3 for epsilon = 1/sqrt(t), 4 for epsilon = 1/log(t)
epsilonMode = 1
c = 1
#Setting number of max iterations and convergence criteria. Problem will run until convergence or for the max nr iterations specified.
T = 2000
nRuns = 1
convergence = Get_Convergence(mus)
# Generate results
for (k in K) {
  for (sig in sigma2) {
    for (i in 1:nRuns) {
      # Get results for the run in terms of ArmChosen and reward
      mus = runif(k, min = 0, max = 1)
      epsilonMode = 1
      epsilonResults1 = epsilonGreedy(k, mus, sig)
      epsilonMode = 2
      epsilonResults2 = epsilonGreedy(k, mus, sig)
      epsilonMode = 3
      epsilonResults3 = epsilonGreedy(k, mus, sig)
      UCBResults = UCB(k, mus, sig)
      if (sig != 10^2) {
        softMaxC1 = SoftMax(k, mus, sig)
        softMaxC10 = SoftMax(k, mus, sig, c = 10)
      }
      # Now, convert to AVERAGE regret over the 1000 iterations.
      if (i == 1) {
        avgRegretEps1 = calc_regret(mus[which.max(mus)], epsilonResults1$ArmChosen, mus)
        avgRegretEps2 = calc_regret(mus[which.max(mus)], epsilonResults2$ArmChosen, mus)
        avgRegretEps3 = calc_regret(mus[which.max(mus)], epsilonResults3$ArmChosen, mus)
        avgRegretUCB = calc_regret(mus[which.max(mus)], UCBResults$ArmChosen, mus)
        avgRegretSM1 = calc_regret(mus[which.max(mus)], softMaxC1$ArmChosen, mus)
        avgRegretSM10 = calc_regret(mus[which.max(mus)], softMaxC10$ArmChosen, mus)

        avgBestArmEps1 = bestArmPlayed(epsilonResults1$ArmChosen, mus)
        avgBestArmEps2 = bestArmPlayed(epsilonResults2$ArmChosen, mus)
        avgBestArmEps3 = bestArmPlayed(epsilonResults3$ArmChosen, mus)
        avgBestArmUCB = bestArmPlayed(UCBResults$ArmChosen, mus)
        avgBestArmSM1 = bestArmPlayed(softMaxC1$ArmChosen, mus)
        avgBestArmSM10 = bestArmPlayed(softMaxC10$ArmChosen, mus)
      } else {
```

```
301         avgRegretEps1 = avgRegretEps1*(i-1)/i + calc_regret(mus[which.max(mus)],
    epsilonResults1$ArmChosen, mus)/i
            avgRegretEps2 = avgRegretEps2*(i-1)/i + calc_regret(mus[which.max(mus)],
    epsilonResults2$ArmChosen, mus)/i
303         avgRegretEps3 = avgRegretEps3*(i-1)/i + calc_regret(mus[which.max(mus)],
    epsilonResults3$ArmChosen, mus)/i
            avgRegretUCB = avgRegretUCB*(i-1)/i + calc_regret(mus[which.max(mus)],
    UCBResults$ArmChosen, mus)/i
305         avgRegretSM1 = avgRegretSM1*(i-1)/i + calc_regret(mus[which.max(mus)],
    softMaxC1$ArmChosen, mus)/i
            avgRegretSM10 = avgRegretSM10*(i-1)/i + calc_regret(mus[which.max(mus)],
    softMaxC10$ArmChosen, mus)/i
307
            avgBestArmEps1 = avgBestArmEps1*(i-1)/i + bestArmPlayed(epsilonResults1$
    ArmChosen,mus)/i
309         avgBestArmEps2 = avgBestArmEps2*(i-1)/i + bestArmPlayed(epsilonResults2$
    ArmChosen,mus)/i
            avgBestArmEps3 = avgBestArmEps3*(i-1)/i + bestArmPlayed(epsilonResults3$
    ArmChosen,mus)/i
311         avgBestArmUCB = avgBestArmUCB*(i-1)/i + bestArmPlayed(UCBResults$
    ArmChosen,mus)/i
            avgBestArmSM1 = avgBestArmSM1*(i-1)/i + bestArmPlayed(softMaxC1$ArmChosen
    ,mus)/i
313         avgBestArmSM10 = avgBestArmSM10*(i-1)/i + bestArmPlayed(softMaxC10$
    ArmChosen,mus)/i
          }
315
          print(i)
317     }

319     analyze(k, sqrt(sig), avgRegretEps1, avgRegretEps2, avgRegretEps3,
    avgRegretUCB, avgRegretSM1, avgRegretSM10, avgBestArmEps1, avgBestArmEps2,
    avgBestArmEps3, avgBestArmUCB, avgBestArmSM1, avgBestArmSM10)

321   }
}
```