Ejercicio 1.2

a. Gratis, barato, oferta, !, %, hastado, ganancias

b. Fecha limite, enviar, anuncio, hola, buenos días/tardes, recordatorio, gracias

c. El parámetro b (bias) en el perceptron afecta directamente cuantos mensajes son clasificados como spam ya que b es el umbral usado para clasificar emails entre spam y no spam

Ejercicio 1.3

a. Si $x(t)$ se clasifica mal por $w(t)$, es decir $y(t) \neq w^T(t)x(t)$ donde $y(t)$ es la etiqueta correcta de $x(t)$, por lo tanto $y(t)$ y $w^T(t)x(t)$ tienen signos diferentes y de esta manera $y(t)w^T(t)x(t) < 0$

b. $y(t)w^T(t+1)x(t)$ $\boxed{w(t+1) = w(t) + y(t)x(t)}$

Sabemos que la regla para actualizar los pesos del perceptrón es $w(t+1) = w(t) + y(t)x(t)$, así:

$$
\begin{aligned}
y(t)w^T(t+1)x(t) &= y(t)\left(w(t) + y(t)x(t)\right)^T x(t) \\
&= y(t)\left(w^T(t) + y(t)x^T(t)\right)x(t) \\
&= \left(y(t)w^T(t) + y(t)y(t)x^T(t)\right)x(t) && \text{distributividad} \\
&= y(t)w^T(t)x(t) + \underbrace{y(t)y(t)x^T(t)x(t)}_{\geq 0} && \text{distributividad} \\
&> y(t)w^T(t)x(t)
\end{aligned}
$$

c. Acabamos de ver que $y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$, es decir, en cada iteración t $y(t)w^T(t)x(t)$ se está incrementando.

Supongamos que $x(t)$ se clasificó mal mediante $w(t)$, es decir, $x(t)w^T(t) \neq y(t)$, entonces:

$y(t) > w^T(t)x(t)$

1. Si $y(t)$ es positiva entonces $w^T(t)x(t)$ es negativa, la actualización sería $w(t+1) = w(t) + y(t)x(t)$, como $y(t)$ es positivo movemos $w^T(t)x(t)$ hacía el lado positivo incrementandolo en $y(t)x(t)$

2. Si $y(t)$ es negativo entonces $w^T(t)x(t)$ es positivo, la actualización sería $w(t+1) = w^T(t) + y(t)x(t)$, como $y(t)$ es negativo movemos $w^T(t)x(t)$ hacía el lado negativo "incrementandolo" $y(t)x(t)$, en este caso como $y(t) < 0$ incrementar significa que $w^T(t)x(t)$ está decreciendo.

Convergencia algoritmo del perceptron

Asuma que para toda muestra (x, y) del conjunto de entrenamiento $\|x\| \leq R$ siendo R un número finito y también suponga que existe un clasificador lineal que clasifique todas las muestras correctamente, más precisamente, asuma que existe $\gamma > 0$ tal que $y(t)w_*^T x(t) \geq \gamma$, gamma se usa para garantizar que toda muestra es clasificada correctamente, nótese que w_* es el vector de pesos del clasificador supuesto.

1. Mostraremos que $w_*^T w(t)$ se incrementa al menos linealmente en cada iteración:

Suponga que en la t-ésima iteración hubo una mala clasificación en $x(t)$ entonces:

$$
\begin{aligned}
w_*^T w(t) &= w_*^T\left(w(t-1) + y(t)x(t)\right) && \text{Actualización de los pesos} \\
&= w_*^T w(t-1) + y(t)x(t)w_*^T && \text{Distributividad} \\
&\geq w_*^T w(t-1) + \gamma && \text{Hipótesis clasificador lineal perfecto con vector pesos } w_*
\end{aligned}
$$

Por lo tanto después de T iteraciones:

$$W_*^T w(t) \geq t\gamma.$$

Luego $W_*^T w(t)$ en cada iteración se incrementa al menos linealmente (t)

2. La norma cuadrada $\|w(t)\|^2$ incrementa máximo linealmente en el número de iteraciones t; también suponga que hubo una mala clasificación:

$$\|w(t)\|^2 = \|w(t-1) + y(t)x(t)\|^2 \quad \text{Actualización pesos debido al error}$$
$$= \|w(t-1)\|^2 + 2y(t)w(t-1)^T x(t) + \|x(t)\|^2 \quad \text{propiedades norma}$$
$$\leq \|w(t-1)\|^2 + \|x(t)\|^2 \quad 2y(t)w(t-1)^T x(t) \geq 0$$
$$\leq \|w(t-1)\|^2 + R^2 \quad \text{Hipótesis}$$

Ya que $y(t)(w(t-1))^T x(t) < 0$ cada vez que se hace una actualización, pues $y(t)$ y $w(t-1)^T x(t)$ tienen signos distintos, y por hipótesis $\|x(t)\| \leq R$; entonces:

$$\|w(t)\|^2 \leq K R^2$$

Ahora si acotamos el coseno del ángulo entre w^* y $w(t)$ en la t-ésima iteración:

$$\cos(w^*, w(t)) = \frac{w_*^T w(t)}{\|w(t)\| \, \|w^*\|} \geq \frac{K\gamma}{\|w(t)\| \, \|w^*\|} \quad \text{por la 1ra parte}$$

$$\geq \frac{K\gamma}{\sqrt{KR^2}\, \|w^*\|} \quad \text{por la 2da parte}$$

y como el coseno está acotado por 1:

$$1 \geq \frac{K\gamma}{\sqrt{KR^2}\, \|w^*\|}$$

Ejercicio 1.10

```
import numpy as np
import matplotlib.pyplot as plt


def tirar_monedas(n):
  #cara: 1, sello: 0
  resultados = np.zeros(n)
  probs = np.random.uniform(size=n)
  resultados[probs > 0.5] = 1 # si la probabilidad es mayor a 0.5 salió cara
  return resultados


def simulacion(n, m,punto_b=False):
  #n -> numero de monedas
  #m -> numero de veces que se tira cada moneda
  #v1 -> Fraccion de caras de la primera moneda tirada.
  #vrand -> Fraccion de caras de una moneda escogida al azar.
  #vmin -> Fraccion de caras de la moneda con menor frecuencia de caras.

    v1 = []
    vrand = []
    vmin = []

    crand = np.random.choice(n) # Escoge la moneda al azar de las n que se van a tirar para v
    caras = np.zeros(n) #suma de caras por cada moneda

    for tirada in range(m):
      # tira las n monedas m veces
        caras = caras + tirar_monedas(n)

    frecuenciaCaras = caras/m

    v1 = frecuenciaCaras[0]
    vrand = frecuenciaCaras[crand]
    cmin = np.argmin(caras)
    vmin = frecuenciaCaras[cmin]

    if not punto_b:
      print(f'Primera moneda: {v1}')
      print(f'Moneda al azar: {vrand} ')
      print(f'Moneda con menos caras: {vmin}')
    return v1,vrand,vmin
```

Punto (a)

Al ser monedas justas tenemos que $\mu = 0.5$, pues estamos utilizando una distribución uniforme.

```
n = 1000
m = 10
```

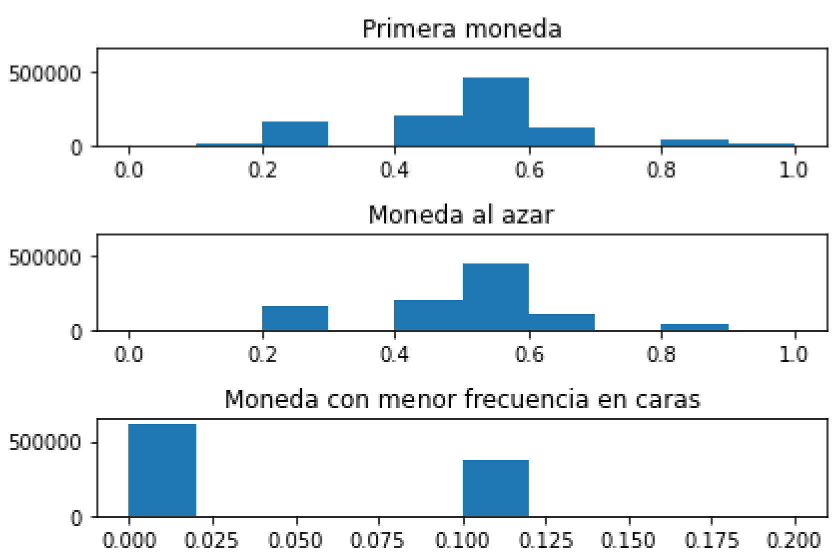
```
simulacion(n, m)
```

```
Primera moneda: 0.3
Moneda al azar: 0.6
Moneda con menos caras: 0.0
(0.3, 0.6, 0.0)
```

```
# Punto b
n = 1000
m = 10
ejecuciones = 1000000
v1s, vrands, vmins = [],[],[] # arreglos para guardar las frecuencias de caras de las monedas
for run in range(ejecuciones):
    v1,vrand,vmin = simulacion(n, m,punto_b=True)
    v1s.append(v1)
    vrands.append(vrand)
    vmins.append(vmin)

fig, axs = plt.subplots(3,1,sharey=True, tight_layout=True)
n_bins = 10
axs[0].hist(v1s,bins=n_bins)
axs[1].hist(vrands,bins=n_bins)
axs[2].hist(vmins,bins=n_bins)

axs[0].set_title('Primera moneda')
axs[1].set_title('Moneda al azar')
axs[2].set_title('Moneda con menor frecuencia en caras')
plt.show()
```



Punto d

Aunque no pude realizar el punto c según la teoría vista en clase la moneda con la menor frecuencia no debería cumplir la cota de Hoeffding pues esta se escogió después de realizar el experimento y no antes como las otras dos monedas, y como ya sabemos esto viola la condición para la desigualdad de Hoeffding que dice que la hipotesis se debe haber fijado antes de que se extraigan las muestras.

Punto e

Al escoger la moneda con menor frecuencia de caras es como de nuestro espacio de 1000 hipotesis o de nuestra bolsa con 1000 hipotesis tomar una hipotesis o bin (moneda con menor frecuencia de caras), estamos tomando el bin después del muestreo de los datos (error), las otras dos monedas si se tomaron antes del muestreo.

Ejercicio 1.11

Tenemos $f : X \to Y$ donde $X = \mathbb{R}$ y $Y = \{-1, 1\}$, para aprender f tenemos el siguiente espacio de hpotesis $H = \{h_1, h_2\}$ donde h_1 es la función constante $+1$ y h_2 la función constante -1.

Punto a

No hay garantía que S pueda producir una mejor hipotesis que tenga mejor desempeño fuera de D, supongamos que f tiene 100 $+1$ en D pero tiene -1 en el resto de puntos en X, es decir, por fuera de D, aqui vemos que S escogerá una mala hipotesis pues nunca le dará a ningún punto, sin enmbargo el algoritmo C tiene mas chances pues tiene un chance del 50% de ajustar los datos, en resumen, en este caso es mejor C que S por fuera de D.

Punto b

En el punto a hay un caso donde el algortimo C escoge una mejor hipotesis que el algoritmo S.

Punto c

Con $p = 0.9$ esta vez el algoritmo C escogerá siempre una peor hipotesis que el algoritmo S, ya que si cada punto en D es +1 según la definición de S este escogerá la hipotesis h_1 y el algoritmo C escogerá la hipotesis h_2, afuera de D, es decir, en $X - D$ la hipotesis h_1 tiene un chance del 0.9 de dar con f, mientras que h_2 sólo un 0.1.

Punto d

Se necesita que C siempre escoja a h_2, esto se lograría según lo visto si $p < 0.5$, de esta manera h_2 se parecerá mucho más a f que la hipotesis h_1.

5 min 40 s completado a las 21:21