# Recomendador de Problemas

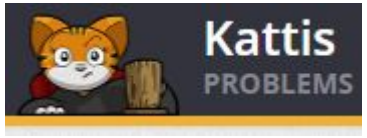
Para ¡Acepta el reto!

David Sarnago Ojuel

# Índice

# Jueces en línea

Un juez online es un sistema de validación de problemas que, dada una serie de entradas, compila, ejecuta y compara la solución original con la obtenida mediante el programa que el usuario ha introducido.






# Obtención de datos I

- La página no cuenta con ninguna API.
- Tampoco cuenta con ninguna herramientas para obtener los datos de forma sencilla.

Lo único que parecemos tener son las estadísticas, que son los últimos envíos al juez:

## Últimos envíos recibidos

| Envío | Usuario | Problema | Resultado | Lenguaje | Tiempo | Memoria | Pos | Fecha |
|---|---|---|---|---|---|---|---|---|
| 526595 | kerqvs | Constante de Kaprekar (100) | WA | Java | 0.71 | 1738 | - | Hace 1 minuto |
| 526594 | Lopixenko | Por el hueco de la escalera... | AC | Java | 0.382 | 1737 | 208 | Hace 13 minutos |
| 526593 | Lopixenko | ¡Feliz década nueva? (539) | AC | Java | 0.796 | 1738 | 153 | Hace 13 minutos |
| 526592 | Lopixenko | Toesto era campo (538) | AC | Java | 0.757 | 1729 | 222 | Hace 14 minutos |
| 526591 | Lopixenko | Reduciendo envases (532) | AC | Java | 0.162 | 1655 | 381 | Hace 15 minutos |
| 526590 | JoseRam | Aprendiendo el código Mo... | RTE | Java | - | - | - | Hace 15 minutos |

# Obtención de datos II

Aunque si vamos expandiendo los últimos envíos llegamos al final, con solo 100 de los +500k envíos que tiene la página en este momento:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 526499 | Diancaceres | Aburrimiento en la autopis... | WA | Java | 1.627 | 3075 | - | Hace 18 horas |
| 526498 | miaumiau | Números polidivisibles (237) | AC | C++ | 1.011 | 1696 | 482 | Hace 18 horas |
| 526497 | getWrapper | Una, dola, tela, catola... (127) | WA | Java | 0.141 | 1723 | - | Hace 18 horas |
| 526496 | miaumiau | Números polidivisibles (237) | WA | C++ | 1.011 | 1696 | - | Hace 18 horas |
| 526495 | getWrapper | Una, dola, tela, catola... (127) | WA | Java | 0.151 | 1723 | - | Hace 18 horas |

# Obtención de datos III

La solución es la siguiente:

1. Realizar un envío a un problema
2. En la URL se ve el ID del envío
3. Utilizar esa URL para hacer peticiones

# Obtención de datos IV

Una vez tenemos a nuestra disposición los datos en una página web, escribimos un web scrapper en Python que obtenga los datos de todos los envíos

1. Obtenemos el HTML de la página correspondiente a un envío con *request* de Python
2. Extraemos información del HTML usando *Beautifulsoup*.
3. Dependiendo del veredicto del juez, algunos campos que obtenemos puede que no existan, por lo que debemos limpiar los datos. En este caso, esta tarea la realizamos antes de insertarlos en base de datos.
4. Una vez obtenidos y limpiados, se insertan en BD.

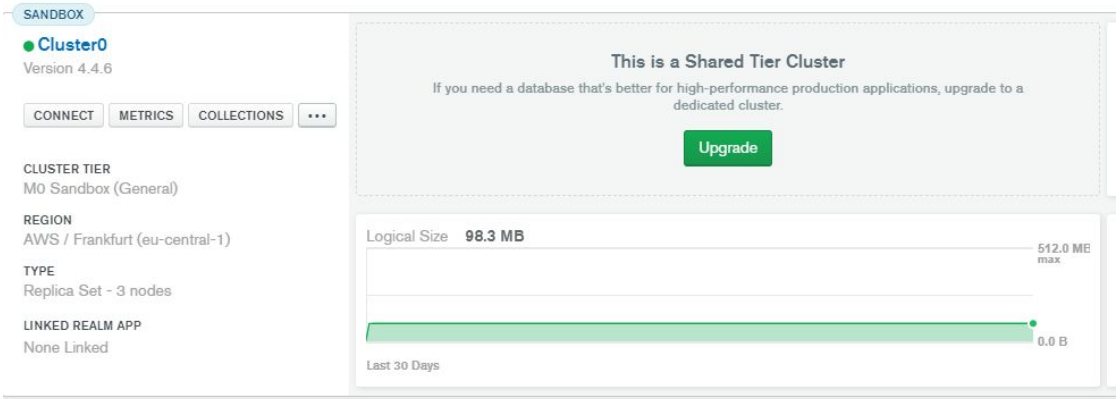
```
page = "https://aceptaelreto.com/user/submission.php?id=" + str(ind)
response = get(page)
soup = BeautifulSoup(response.content, "html.parser")

#usuario = soup.findAll('div', attrs={'class':'col-md-10'})
scrap = soup.findAll('div', attrs={'class':'col-md-10'})[0]
l = scrap.text.split("\n")
l = [value for value in l if value != ""]
```

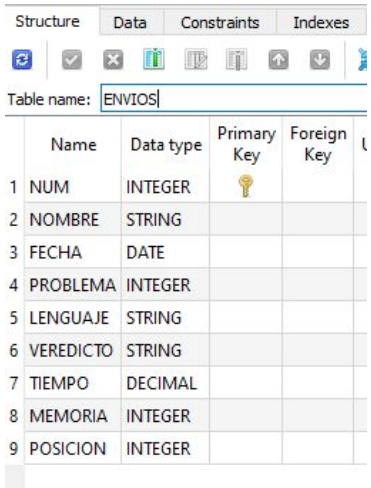
```
def insert_datos(datos):
    command = "INSERT INTO ENVIOS VALUES("
    command += datos['envio']
    command += ", \"" + datos['usuario'] + "\""
    command += ", \"" + datos['fecha'] + "\""
    command += ", " + str(datos['problema'])
    command += ", \"" + datos['lenguaje'] + "\""
    command += ", \"" + datos['veredicto'] + "\""
    command += ", " + str(datos['tiempo'])
    command += ", " + str(datos['memoria'])
    command += ", " + str(datos['posicion'])
    command += ")"
    c.execute(command)
```

# Almacenamiento de los datos I

Se realiza sobre 2 bases de datos con diferentes filosofías:

1. Una base de datos relacional local, en SQLite3: parte de la librería estándar de Python, rápida por accederse directamente desde el disco y es relacional.
2. Una base de datos NoSQL, MongoDB en la nube: en un clúster gratuito de la página MongoDB Atlas. Ocupa 98 MB de los 512 que nos ofrece la página. Usada para aumentar la redundancia, en caso de pérdida de la local poder reconstruirla en minutos y no horas.




| | Name | Data type | Primary Key | Foreign Key |
|---|---|---|---|---|
| 1 | NUM | INTEGER | | |
| 2 | NOMBRE | STRING | | |
| 3 | FECHA | DATE | | |
| 4 | PROBLEMA | INTEGER | | |
| 5 | LENGUAJE | STRING | | |
| 6 | VEREDICTO | STRING | | |
| 7 | TIEMPO | DECIMAL | | |
| 8 | MEMORIA | INTEGER | | |
| 9 | POSICION | INTEGER | | |

# Almacenamiento de datos II

Diferencia entre insertar elementos de uno en uno a insertar en bloque:

- 524640 envíos de 9 campos cada uno, insertados en paquetes de 1000: 53.6 s
- 899 categorías de 3 campos cada una, insertados de 1 en 1: 34.1 s
- 504 problemas de 5 campos cada uno, insertados a la vez: 67.1 milisegundos
- 13982 ratings, de 2 campos cada uno, insertados a la vez: 851 milisegundos

Y con esto aclarado, aquí los tamaños de cada base de datos en MongoDB

| | CATEGORIAS | ENVIOS | PROBLEM_DIFF | USER_RATING |
|---|---|---|---|---|
| Nº Documentos | 898 | 524640 | 504 | 13982 |
| Tamaño colección | 49.99KB | 85.53MB | 55.13KB | 833.44KB |
| Tamaño índices | 36KB | 11.4MB | 32KB | 368KB |

# Recomendadores construidos

Se han desarrollado 2 recomendadores en base a criterios diferentes:

- En base a la dificultad de los problemas y habilidad del usuarios
- En base a la similitud con otros usuarios, *User-Based*

```
Recommended problems for Davian99 (6.95)
 - Nothing to recommend

Recommended problems for KPsychO (5.61)
 - TRIVIAL: 143 (4.40, 0.80)
 - EASY:    486 (5.25, 0.60)
 - MEDIUM:  414 (5.96, 0.40)
 - HARD:    118 (6.77, 0.21)

Recommended problems for gremio (6.81)
 - TRIVIAL: 280 (5.59, 0.80)
 - EASY:    294 (6.34, 0.63)
 - MEDIUM:  240 (6.74, 0.52)
 - HARD:    530 (7.18, 0.40)
```

```
Recommended problems for KPsychO
 - 224 (Solved by 4)
 - 306 (Solved by 4)
 - 247 (Solved by 2)
 - 320 (Solved by 2)
 - 472 (Solved by 2)

Recommended problems for gremio
 - 515 (Solved by 4)
 - 517 (Solved by 4)
 - 518 (Solved by 4)
 - 519 (Solved by 4)
```

# Recomendador según la dificultad I

Para poder recomendar problemas en base a cómo de difíciles les resulten a los usuarios debemos tener 2 métricas entre nuestros datos:

- Los problemas deben tener una dificultad asignada.
- El usuario debe tener cuantificada su habilidad, que llamaremos *rating*.

Cómo ni los problemas están categorizados por dificultad ni los usuarios tienen una habilidad asignada, debemos hacerlo nosotros mismos con no más datos que los envíos a la página.

La dificultad de los problemas depende de los usuarios que lo resuelven y la habilidad de los usuarios depende de los problemas que resuelvan -> Buena paradoja

# Asignar dificultad a problemas I

Solución -> Romper la paradoja

Establecemos la dificultad de los problemas en base a una seria de métricas y con esta dificultad, obtenemos los *ratings* de los usuarios.

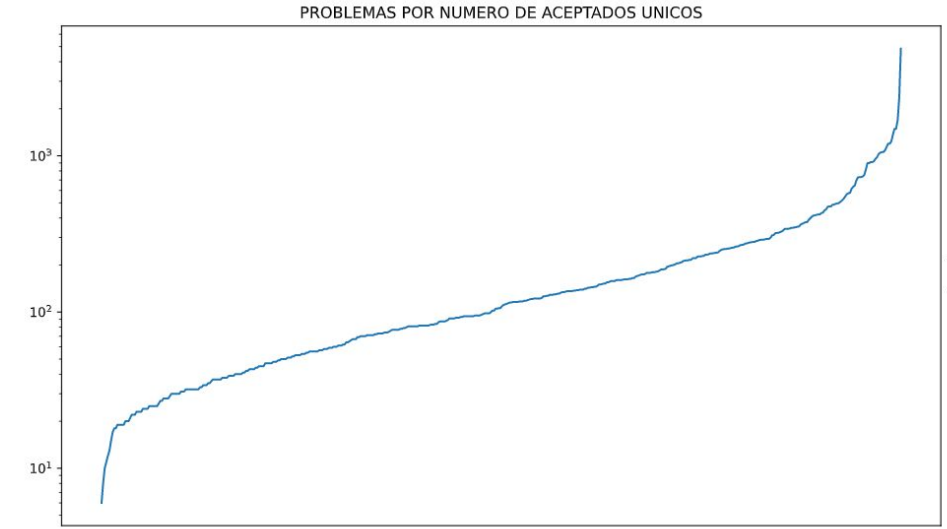
Las métricas que vamos a utilizar son:

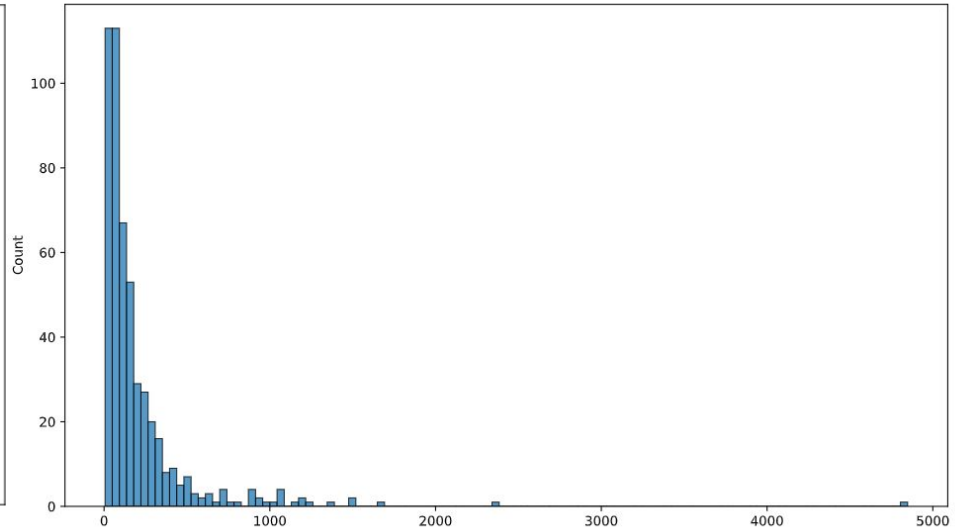
- Número de usuarios únicos que han resuelto el problema.
- Media de envíos hasta el primer aceptado.
- Media de problemas resueltos de los usuarios que lo resuelven.

# Asignar dificultad a problemas II

Número de usuarios únicos que han resuelto el problema

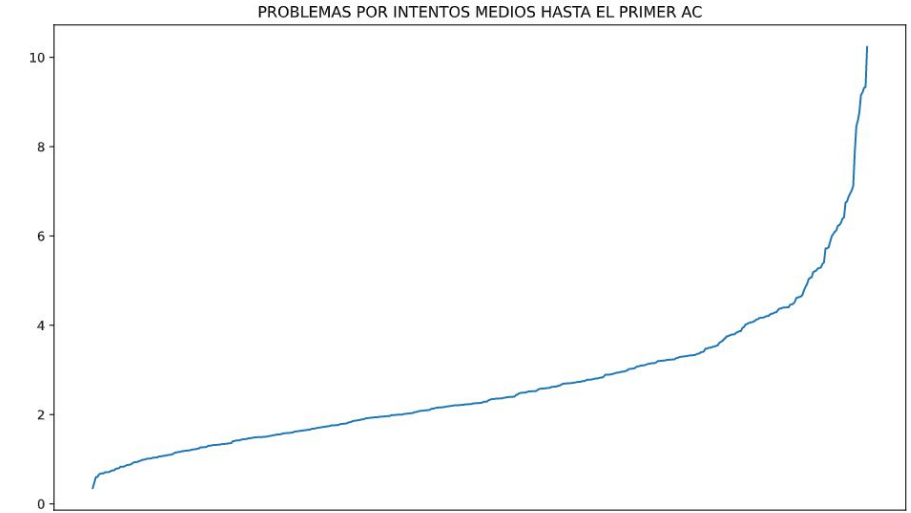
```
SELECT PROBLEMA, COUNT(T) as C FROM (
    SELECT PROBLEMA, COUNT(*) AS T FROM ENVIOS
    WHERE VEREDICTO = 'AC'
    GROUP BY NOMBRE, PROBLEMA
)
GROUP BY PROBLEMA
```

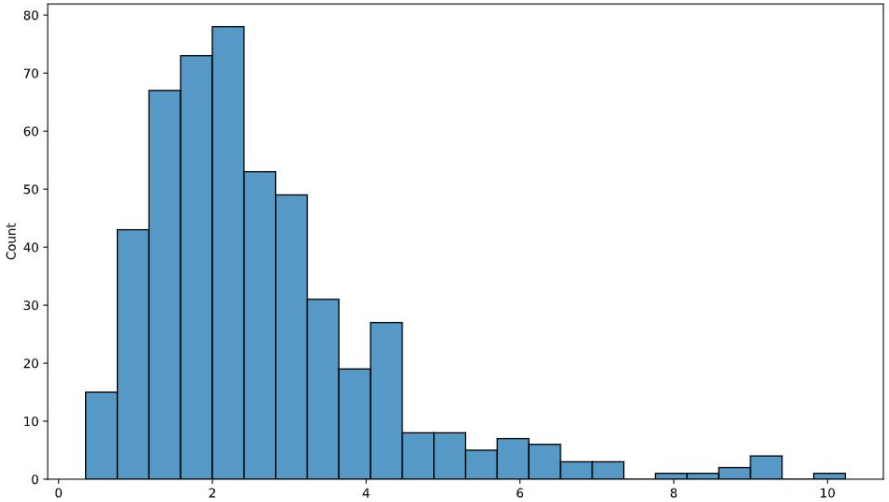


# Asignar dificultad a problemas III

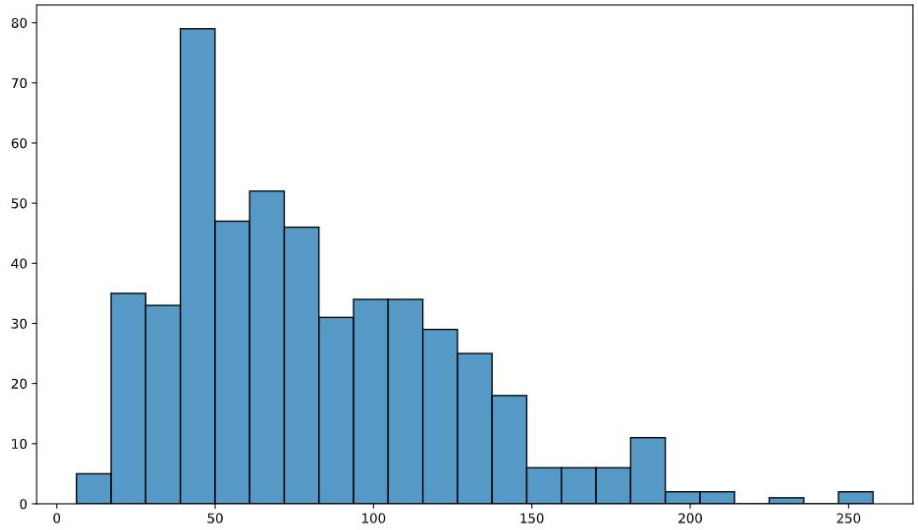
Media de envíos hasta el primer aceptado



# Asignar dificultad a problemas IV

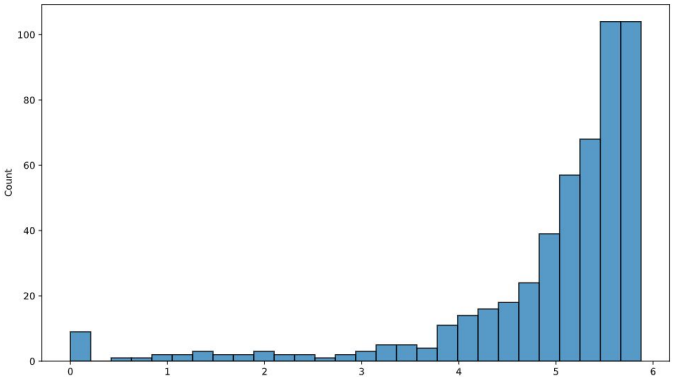
Media de problemas resueltos de los usuarios que lo resuelven

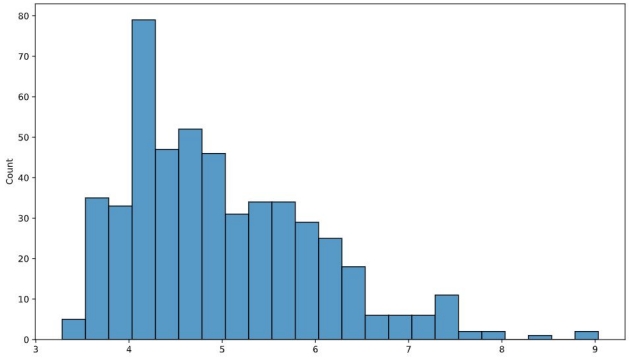
# Asignar dificultad a problemas V

Una vez tenemos las diferentes métricas, vamos a normalizar sus valores:

Todos ellos tienen media 5 y STD 1.

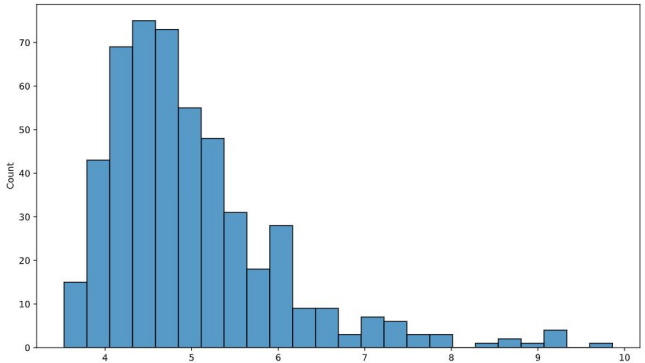



Resueltos únicos




Resueltos de resolutores

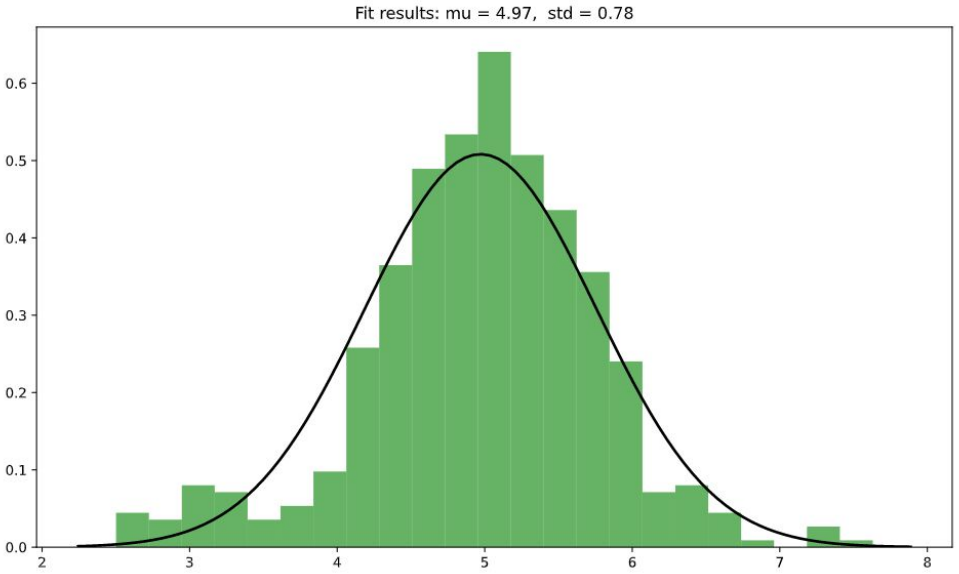
Intentos hasta



# Asignar dificultad a problemas VI

Dificultad final de los problemas

| POS | PROBLEMA | VALOR | UNIQUE | TRIES | AVG_SOL |
|---|---|---|---|---|---|
| 1 | 313 | 2.54 | 1236 | 1.21 | 32.17 |
| 2 | 217 | 2.55 | 1689 | 1.34 | 27.67 |
| 3 | 116 | 2.58 | 4831 | 1.76 | 13.1 |
| 502 | 566 | 7.38 | 10 | 9.21 | 331.1 |
| 503 | 391 | 7.49 | 17 | 8.7 | 389.18 |
| 504 | 567 | 7.65 | 6 | 9.33 | 390.33 |



# Asignar rating a los usuarios I

Una vez hemos asignado dificultad a los problemas, calcular el *rating* de los usuarios es más fácil.

Cuidado con la forma en la que se calcula:

- Recompensar el resolver problemas difíciles.
- No tener en cuenta los problemas intentados, queremos que los usuarios intenten.
- No permitir la inflación del *rating* resolviendo problemas fáciles.

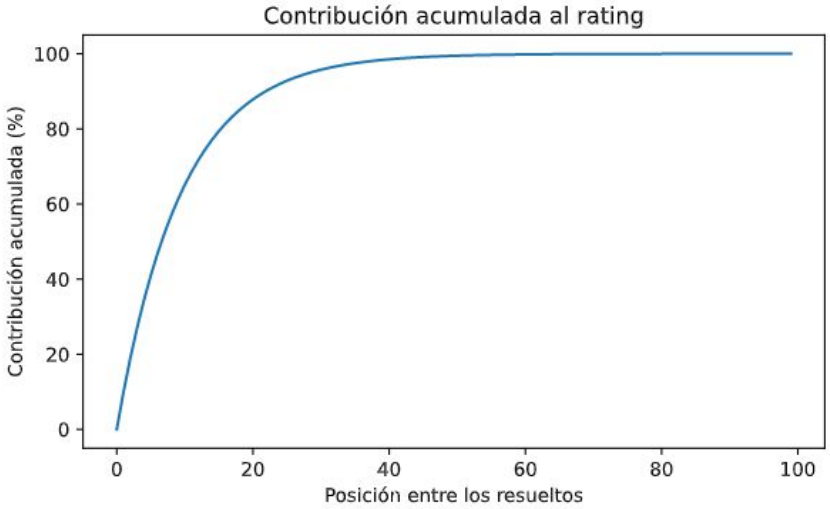
Con todos estos requisitos, se ha utilizado finalmente un sistema ponderado con la siguiente fórmula:

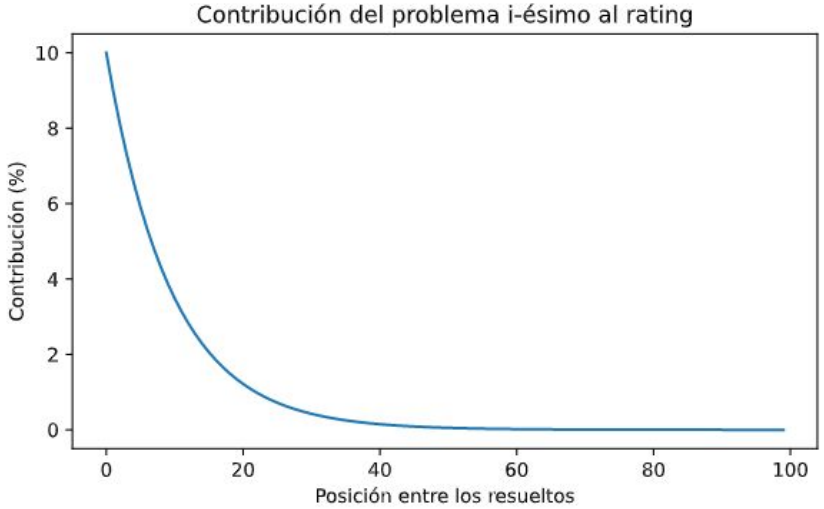
$$\frac{1}{10} * \sum_{n=0}^{nSolved} PD_n * 0.9^n \longrightarrow$$

| | 3 | 5 | 7 |
|---|---|---|---|
| 10 | 1.95 | 3.25 | 4.55 |
| 20 | 2.63 | 4.39 | 6.14 |
| 100 | 2.99 | 4.99 | 6.99 |

# Asignar rating a los usuarios II

Más gráficas para ilustrar el sistema ponderado:

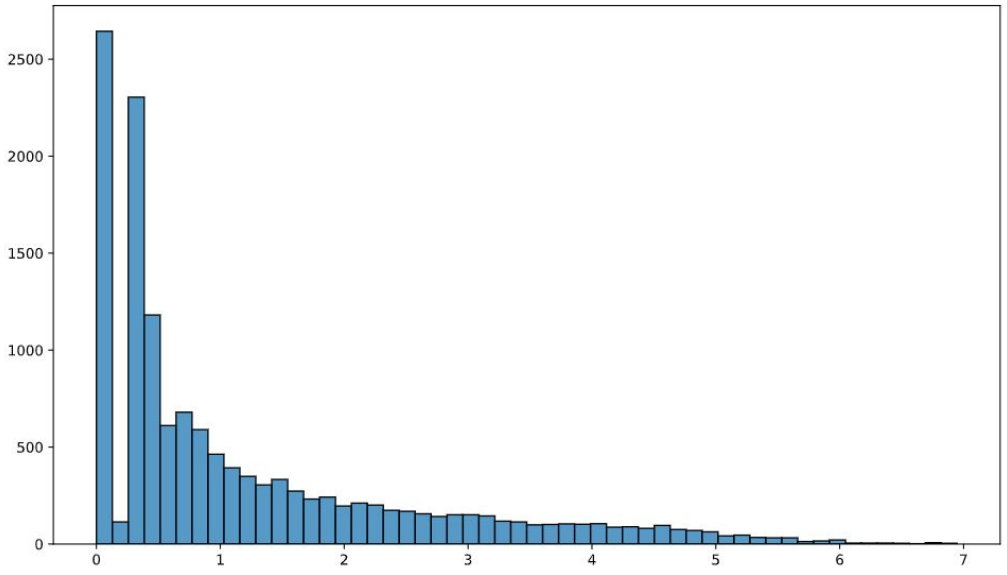


# Asignar rating a los usuarios III

*Ratings* finales de los usuarios:

| Usuario | Rating | Resueltos |
|---|---|---|
| Aletheia | 6.95 | 504 |
| Davian99 | 6.95 | 504 |
| Davvid | 6.95 | 504 |
| tirutu | 6.95 | 504 |
| gremio | 6.81 | 317 |
| CecilioG | 6.79 | 498 |
| cobotejelalana | 6.79 | 499 |
| karel-the-robot | 6.77 | 474 |
| AperezaC | 6.71 | 490 |
| Apereza | 6.71 | 492 |



# Recomendador según la dificultad II

Utilizando la dificultad del problema y el *rating* del usuario, recomienda 4 problemas de diferentes rangos de dificultad.

Las recomendaciones se hacen en base al *expected_score*, o resultado esperado. Esta es una medida que nos proporciona cuál es la probabilidad de que un usuario de *rating ra* resuelva un problema de dificultad *pd*:

$$expected\_score = \frac{1}{1 + 10^{\frac{PD-RA}{2}}}$$

# Recomendador según la dificultad III

Se recomiendan 4 problemas:

- Trivial: un problema que el usuario tiene un 80% de probabilidades de resolver.
- Fácil: tiene un 60%
- Medio: un 40%
- Difícil: un 20%

No se recomiendan problemas que ya ha resuelto, por lo que puede que se repitan problemas o que directamente no pueda recomendar nada.

```
Recommended problems for Davian99 (6.95)
 - Nothing to recommend

Recommended problems for KPsychO (5.61)
 - TRIVIAL: 143 (4.40, 0.80)
 - EASY:    486 (5.25, 0.60)
 - MEDIUM:  414 (5.96, 0.40)
 - HARD:    118 (6.77, 0.21)

Recommended problems for Ignacio1997corrales (4.75)
 - TRIVIAL: 165 (3.53, 0.80)
 - EASY:    109 (4.39, 0.60)
 - MEDIUM:  329 (5.10, 0.40)
 - HARD:    271 (5.95, 0.20)

Recommended problems for gremio (6.81)
 - TRIVIAL: 280 (5.59, 0.80)
 - EASY:    294 (6.34, 0.63)
 - MEDIUM:  240 (6.74, 0.52)
 - HARD:    530 (7.18, 0.40)
```

# Recomendador según la afinidad I

El segundo recomendador resultó ser mucho más fácil de implementar:

- No necesita asignar dificultades ni *ratings*
- No necesita normalizar y combinar métricas para recomendar

Simplemente, utiliza los problemas resueltos de un usuario y obtiene los usuarios más parecidos a él de toda la página.

Este tipo de recomendadores son conocidos cómo *User-based*, ya que utilizan la información de los usuarios para recomendar problemas, sin tener en cuenta estos.

# Recomendador según la afinidad II

Vamos a utilizar la similitud del coseno para encontrar los usuarios más parecidos al dado.

La similitud del coseno encuentra cómo de parecidos son dos vectores, por lo que tenemos que vectorizar los problemas de cada usuario.

Esto es sencillo, construimos un array de Numpy de dos dimensiones:

```
problems_solved_user_np = np.zeros(shape=(len(lista_usuarios), max(lista_problemas) + 1))
```

Y lo rellenamos en base a si el usuario ha resuelto el problema o no.

# Recomendador según la afinidad III

Con las similitudes, obtenemos los usuarios más parecidos a uno dado:

Y para recomendar, simplemente obtenemos los n usuarios más similares y recomendamos los problemas que no tengamos resueltos que más se repiten entre nuestros afines:

```
Recommended problems for KPsychO
 - 224 (Solved by 4)
 - 306 (Solved by 4)
 - 247 (Solved by 2)
 - 320 (Solved by 2)
 - 472 (Solved by 2)

Recommended problems for gremio
 - 515 (Solved by 4)
 - 517 (Solved by 4)
 - 518 (Solved by 4)
 - 519 (Solved by 4)
```

```
Affinity with         Davian99
==================== ========
Aletheia                1.0000
Davvid                  1.0000
tirutu                  1.0000
cobotejelalana          0.9950
CecilioG                0.9940

Affinity with          KPsychO
==================== ========
dsinapellido            0.7619
nickN.A.S               0.6835
JMaria                  0.6529

Affinity with           gremio
==================== ========
Aletheia                0.7931
Davian99                0.7931
Davvid                  0.7931
```

# Conclusiones

A estado guay hacer este proyecto. Los recomendadores obtenidos finales funcionan bastante bien y estoy muy contento con los resultados obtenidos en la asignación de dificultad y *rating*.

Recomendador basado en los problemas, o *Content-Based* no terminó de funcionar por que lo único que une a los problemas entre sí son las categorías.

Estas son muy pocas por problema, por lo que la similitud del coseno o métodos parecidos ofrecen una sugerencias muy pobres, por lo que no pasó a formar parte de la entrega.

Problemas sin categorías asignada cómo positivo.