Recuperación de Información:

Entregable de ElasticSearch

Samuel Rodríguez Ares (UO271612)

Moisés Sanjurjo Sánchez (UO270824)

Ejercicio 1

En este ejercicio, se propone encontrar el total de documentos relacionados con una temática dada, partiendo de la colección **mentalhealth-subreddits** proporcionada por los profesores.

Se creará un nuevo índice a través del script llamado `bulk-indexer-Ejercicio1.py` para eliminar palabras vacías mediante la configuración de un nuevo analyzer.

```
argumentos={
    "settings": {
        "analysis": {
            "analyzer": {
                "palabras_vacias_ingles_porter": {
                    "type": "stop",
                    "stopwords": ["a","about","above","after","again",
                                  "against","all","am","an","and","any","are",
                                  "aren't","as","at","be","because","been",
                                  "before","being","below","between","both",
                                  "but","by","can't","cannot","could",
                                  "couldn't","did","didn't","do","does",
                                  "doesn't","doing","don't","down","during",
                                  "each","few","for","from","further","had",
                                  "hadn't","has","hasn't","have","haven't",
                                  "having","he","he'd","he'll","he's","her",
                                  "here","here's","hers","herself","him",
                                  "himself","his","how","how's","i","i'd",
                                  "i'll","i'm","i've","if","in","into","is",
                                  "isn't","it","it's","its","itself","let's",
                                  "me","more","most","mustn't","my","myself",
                                  "no","nor","not","of","off","on","once",
                                  "only","or","other","ought","our","ours",
                                  "ourselves","out","over","own","same",
                                  "shan't","she","she'd","she'll","she's",
                                  "should","shouldn't","so","some","such",
                                  "than","that","that's","the","their",
                                  "theirs","them","themselves","then","there",
                                  "there's","these","they","they'd","they'll",
```

En los mappings se establece el analyzer creado para los campos selftext y title, de modo que la eliminación de palabras vacía se aplique en ambos.

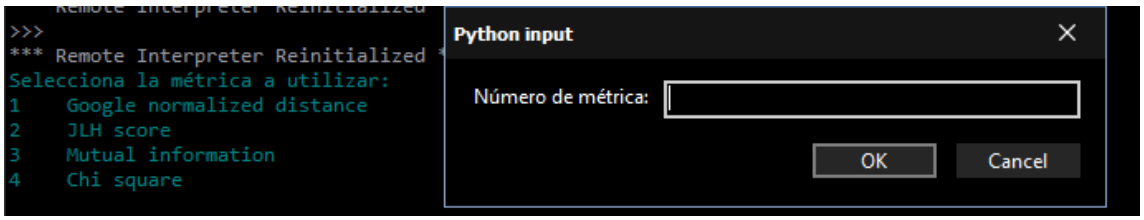
Además, se establece el campo `fielddata` a `true` para poder realizar agregaciones sobre estos campos.

```
        },
        "mappings": {
            "properties": {
                "edited": {
                    "type":"text"
                },
                "crosspost_parent_list.edited": {
                    "type":"text"
                },
                "title": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                },
                "selftext": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                }
            }
        }
    }
```

Mediante la ejecución de este script, se crea un nuevo índice denominado **`ejercicio1`** que implementa estas modificaciones y será utilizado en diferentes ejercicios.

El resto del ejercicio es completado con el script `ejercicio1.py`. en el que se ha escogido como **temática** el **estrés**. Se han llevado a cabo los siguientes pasos:

1. Se pide por pantalla la métrica de similitud a utilizar. El script soporta NGD, JLH, información mutua (sin inclusión de negativos) y Chi cuadrado.



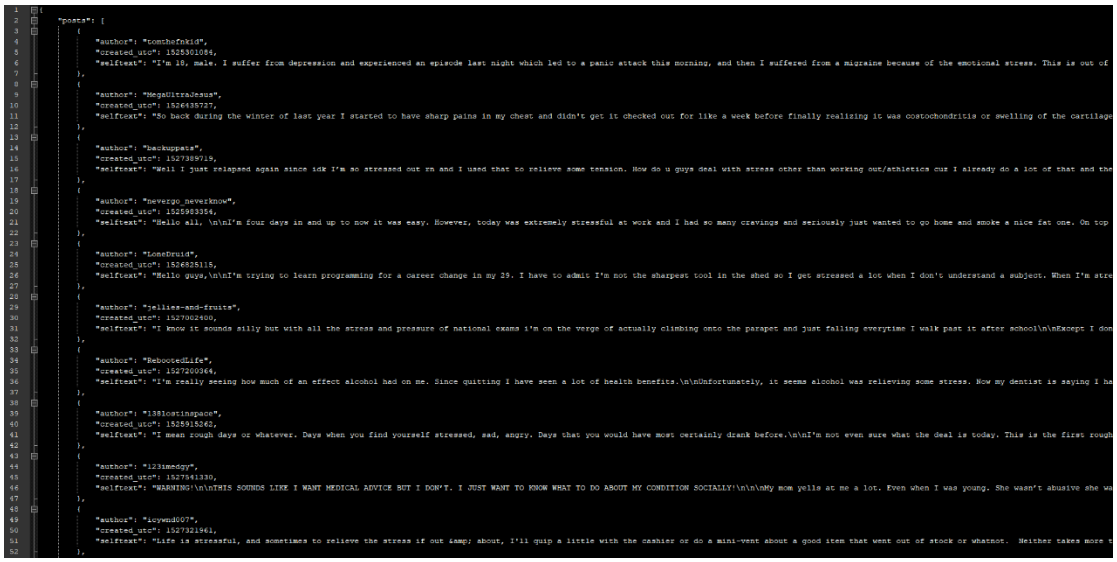
2. En función de la métrica seleccionada, **se buscarán los 10 términos más significativos** a partir de la temática "stress" utilizando dicha métrica.

```
if (metrica == 1):
    #Obtenemos autmáticamente los términos más significativos según la métrica usada
    results = es.search(
        index="ejercicio1",
        body = {
            "query": {
                "match": {
                    "selftext": {
                        "query": "stress"
                    }
                }
            },
            "aggs": {
                "Terminos más significativos": {
                    "significant_terms": {
                        "field": "selftext",
                        "size": 10,
                        "gnd": {}
                    }
                }
            }
        }
    )
```

3. Se imprimen los términos encontrados y se introducen en una cadena de caracteres que será empleada para buscar los documentos de forma que se recuperen los más relevantes relacionados con cualquiera de esos términos.

```
Número de métrica: 1
 stress stressed stressful caused handle due pressure causing panic relieve
>>>
```

4. Por último, se extrae el autor, fecha de creación y texto, para posteriormente volcarse a un fichero denominado **`ejercicio1.json`**.



La idea consiste en obtener las diferentes **consecuencias, a nivel fisiológico y mental, de aquellas personas que sufren estrés** y lo han reflejado en alguno de los posts de la colección.

Cabe mencionar que para la obtención de estos documentos relevantes se ha empleado la métrica de similitud **Distancia Normalizada de Google**, ya que permitió obtener, con diferencia, los términos significativos más cercanos al tema de investigación.

Métrica de similitud	**Términos más significativos**
gnd	stress stressed stressful caused handle due pressure causing panic relieve
jlh	stress stressed anxiety time work now just life feel like
mutual_information	stress just like time now feel get life work anxiety
chi_square	stress stressed anxiety time work just now like life feel

A continuación, seleccionaremos los **15 documentos más relevantes** y **juzgaremos su relevancia** de acuerdo con nuestra búsqueda.

#número	**¿Relevante?**
#1	NO
#2	NO
#3	SÍ
#4	NO
#5	SÍ
#6	SÍ
#7	SÍ
#8	NO
#9	SÍ
#10	NO
#11	SÍ
#12	SÍ
#13	NO
#14	SÍ
#15	SÍ

$$Precisión = \frac{A}{A+C} \cdot 100 = \frac{9}{15} \cdot 100 = \mathbf{60}\%$$

Nota: Se ha limitado la salida de documentos a 15 con el objetivo de adecuar el tamaño de la entrega de los diferentes ejercicios al estipulado en el Campus Virtual (40 MB).

Ejercicio 2

En este ejercicio, se pide emular la expansión de términos llevada a cabo mediante la métrica de similitud NGD (Distancia Normalizada de Google) a través de la consulta **More Like This**.

Este tipo de consulta de ElasticSearch funciona de la siguiente manera:

1. Se busca las palabras con mayor ***tf-idf*** pertenecientes a un conjunto de documentos sobre uno o varios campos determinados en los de un índice específico.
2. Se lleva a cabo una expansión de términos sobre los documentos resultantes de la búsqueda. El número máximo de términos a expandir se determina en la consulta mediante el parámetro `max_query_terms`.
3. Realiza una nueva búsqueda de documentos a partir de todos los términos encontrados y los devuelve.

Para afrontar este ejercicio, se realizó un **script en Python** denominado `ejercicio2.py` que funciona, bajo la misma temática del Ejercicio 1 (estrés) mediante una serie de pasos:

Buscamos los 15 documentos más relevantes relacionados con la temática en cuestión:

```
# Obtenemos automáticamente los documentos relacionados
results = es.search(
    index="ejercicio1",
    body = {
        "query": {
            "match": {
                "selftext": {
                    "query": "stress"
                }
            }
        },
        "size": 15
    }
)
```

Se obtendrá una lista con 15 documentos bajo un índice común. De estos documentos se extraerá su identificador para poder ser utilizados en la consulta ***More Like This***:

```
ids = []

# Iteramos sobre los resultados, no es preciso preocuparse de las
# conexiones consecutivas que hay que hacer con el servidor ES
for i in range(0, len(results["hits"]["hits"])):
    ids.append({
        "_id": results["hits"]["hits"][i]["_id"]
    })
```

Esta es la **consulta More Like This** que utilizará posteriormente esos identificadores:

```
# Inicializamos consulta MLT a la que pasamos posteriormente los documentos
consulta = {
    "query": {
        "more_like_this": {
            "fields": [
                "title",
                "selftext"
            ],
            "like": [],
            "min_term_freq": 1,
            "max_query_terms": 10
        }
    },
    "size": 15
}
```

El campo `like` de una consulta More Like This puede ser completado de dos maneras: especificando palabras clave, o especificando documentos pertenecientes a algún índice.

En este caso se optará por la segunda opción; la consulta necesitará recibir los documentos en el siguiente formato:

{

“_index”: “ejercicio1”,

“_id”: “8jin57”

}

Por tanto, convertiremos cada identificador extraído de la primera consulta para formar un objeto JSON que especifique su índice e identificador en dicho formato y que será introducido en el campo `like` de la consulta *More Like This*.

```
# Recorremos los identificadores para insertarlos en la consulta
for id in ids:
    consulta["query"]["more_like_this"]["like"].append({
        "_index": "ejercicio1",
        "_id": id["_id"]
    })
```

De esta manera, ya tendremos el cuerpo de la consulta listo para ser ejecutado y obtener los documentos que necesitamos, emulando lo realizado en el Ejercicio 1:

```
# Realizamos la búsqueda
results = es.search(
    index = "ejercicio1",
    body = consulta
)
```

Los documentos obtenidos de la última consulta serán reducidos a un JSON en el que se especifique únicamente **el autor, la fecha y el texto**, al igual que en el Ejercicio 1.

```
# Obtenemos los resultados, que volcaremos a un JSON
posts = results["hits"]["hits"]

datos = {}
datos["posts"] = []

# Iteramos sobre los resultados, no es preciso preocuparse de las
# conexiones consecutivas que hay que hacer con el servidor ES
for hit in posts:
    datos["posts"].append({
        "author": hit["_source"]["author"],
        "created_utc": hit["_source"]["created_utc"],
        "selftext": hit["_source"]["selftext"]
    })
```

La salida obtenida será almacenada en un archivo **`ejercicio2.json`**.

```
# Guardamos los documentos más relevantes en un fichero JSON
with open("ejercicio2.json","w",encoding="utf8") as fichero:
    json.dump(datos, fichero, indent=4, ensure_ascii=False)
```

Para finalizar la realización del ejercicio, **juzgaremos la relevancia de los 15 documentos encontrados**, de acuerdo con nuestro objetivo de búsqueda.

#número	¿Relevante?
#1	SÍ
#2	NO
#3	NO
#4	NO
#5	SÍ
#6	NO
#7	NO
#8	NO
#9	NO
#10	SÍ
#11	SÍ
#12	NO
#13	SÍ
#14	NO
#15	NO

$$Precisión = \frac{A}{A+C} \cdot 100 = \frac{5}{15} \cdot 100 = \mathbf{33.33}\%$$

Ejercicio 3

Se pide obtener una lista exhaustiva de medicamentos a partir de los documentos albergados en un índice que no contemple palabras vacías (en nuestro caso, el índice denominado **`ejercicio1`**).

Además, se pide, si es posible, automatizar la validación de los medicamentos obtenidos de forma que los resultados garanticen ser medicamentos.

Para afrontar la realización de este ejercicio, el primer paso ha consistido en realizar una **búsqueda de términos significativos relacionados con medicamentos**, que además es expandida con un tamaño de **hasta 3000 términos relacionados** haciendo uso de la métrica **NGD** (*Distancia Normalizada de Google*).

```
# Realizamos la búsqueda de términos relacionados
results = es.search(
        index="ejercicio1",
        body = {
            "query": {
                "match": {
                    "selftext": {
                        "query": "medicine medication medicament drug",
                        "operator": "or"
                    }
                }
            },
            "aggs": {
                "Terminos más significativos": {
                    "significant_terms": {
                        "field": "selftext",
                        "size": 3000,
                        "gnd": {}
                    }
                }
            }
        }
)
```

Una vez obtenidos los términos relacionados con medicamentos, se procede a validar cada uno de estos términos para comprobar si realmente son medicamentos o no.

Para ello, se dispone de la **API de Wikidata**, que nos permite obtener, a partir de la búsqueda de un término, los identificadores de diferentes entradas en la página.

La URL utilizada para consumir el servicio Web de búsqueda en Wikidata sigue el formato:

https://www.wikidata.org/w/api.php?action=wbsearchentities&search=xanax&language=en&format=json

Esta devuelve un archivo JSON con las entidades encontradas en Wikidata (si existen):

```
{
   "searchinfo":{
      "search":"xanax"
   },
   "search":[
      {
         "id":"Q319877",
         "title":"Q319877",
         "pageid":307289,
         "repository":"wikidata",
         "url":"//www.wikidata.org/wiki/Q319877",
         "concepturi":"http://www.wikidata.org/entity/Q319877",
         "label":"alprazolam",
         "description":"chemical compound: potent, short-acting anxiolytic of the benzodiazepine class\u2014a minor tranquilizer",
         "match":{
            "type":"alias",
            "language":"en",
            "text":"Xanax\u00ae"
         },
         "aliases":[
            "Xanax\u00ae"
         ]
      },
      {
         "id":"Q47521014",
         "title":"Q47521014",
         "pageid":48564158,
         "repository":"wikidata",
         "url":"//www.wikidata.org/wiki/Q47521014",
         "concepturi":"http://www.wikidata.org/entity/Q47521014",
         "label":"Xanax",
         "description":"pharmaceutical product",
         "match":{
            "type":"label",
            "language":"en",
            "text":"Xanax"
         }
      },
      {
         "id":"Q11306385",
         "title":"Q11306385",
         "pageid":12445460,
         "repository":"wikidata",
         "url":"//www.wikidata.org/wiki/Q11306385",
         "concepturi":"http://www.wikidata.org/entity/Q11306385",
         "label":"Xanax",
         "match":{
            "type":"label",
            "language":"en",
            "text":"Xanax"
         }
      },
```


Wikidata, a través de su API, permite realizar otro tipo de consultas como la siguiente:

https://www.wikidata.org/w/api.php?action=wbgetentities&ids=Q12140|Q1916282&languages=en&format=json

Este tipo de consulta recibe uno (o más, separados por el carácter **|**) identificadores, devolviendo información útil sobre las propiedades de cada entidad.

Por defecto, el API admite hasta un total de 50 identificadores por consulta de forma gratuita, aunque es posible extender el límite hasta 500 identificadores.

Lo interesante de esta consulta, es que nos permite conocer si una entidad, dado un identificador, es instancia de un medicamento. Tras investigar en la propia página de Wikidata, se ha deducido que el statement "Instance of" corresponde a la **propiedad P31**; y que, para ser un medicamento, **el identificador de esta propiedad debe ser Q12140**.

Se adjunta un ejemplo de JSON devuelto por la consulta mencionada:

```
{
  "entities":{
    "Q12140":{
      "pageid":13710,
      "ns":0,
      "title":"Q12140",
      "lastrevid":1314983347,
      "modified":"2020-11-29T06:18:21Z",
      "type":"item",
      "id":"Q12140",
      "labels":{
        "en":{
          "language":"en",
          "value":"medication"
        }
      },
      "descriptions":{
        "en":{
          "language":"en",
          "value":"substance used to diagnose, cure, treat, or prevent disease"
        }
      },
      "aliases":{
        "en":[
          {
            "language":"en",
            "value":"medicine"
          },
          {
            "language":"en",
            "value":"pharmaceutical drug"
          },
          {
            "language":"en",
            "value":"drug"
          },
          {
            "language":"en",
            "value":"pharmaceutical"
          },
          {
            "language":"en",
            "value":"curetive substance"
          },
          {
            "language":"en",
            "value":"supprative substance"
          },
          {
            "language":"en",
            "value":"proactive substance"
          }
```



```
          "value":"proactive substance"
        }
      ]
    },
    "claims":{
      "P373":[
        {
          "mainsnak":{
            "snaktype":"value",
            "property":"P373",
            "hash":"39c11f482b4106e10ec1953a5971c73c32b7d2a6",
            "datavalue":{
              "value":"Pharmaceutical drugs",
              "type":"string"
            },
            "datatype":"string"
          },
          "type":"statement",
          "id":"q12140$90591A19-FB27-4F47-A18E-354AEF68D86C",
          "rank":"normal"
        }
      ],
      "P508":[
        {
          "mainsnak":{
            "snaktype":"value",
            "property":"P508",
            "hash":"d735277ef8e565617b047c4cec18c6b519b26a60",
            "datavalue":{
              "value":"846",
              "type":"string"
            },
            "datatype":"external-id"
          },
          "type":"statement",
          "id":"q12140$B8E9DFA6-3B06-483D-8A9A-A661D337285B",
          "rank":"normal",
          "references":[
            {
              "hash":"a235ba3d92daf43853b23688674b7ab464b4eea8",
              "snaks":{
                "P248":[
                  {
                    "snaktype":"value",
                    "property":"P248",
                    "hash":"afdc73e2b2508208bcfb2c9e8b8b010bb19acef4",
                    "datavalue":{
                      "value":{
                        "entity-type":"item",
                        "numeric-id":460907,
                        "id":"Q460907"
                      },
```


El objetivo consiste en acceder a los "claims" de cada entidad y buscar si existe la "P31", de forma que, en caso afirmativo, se acceda al "mainsnak" y se obtenga el campo "id" para comprobar si coincide con el valor "Q12140".

Si se cumplen estas condiciones, entonces el término será añadido a la lista final de medicamentos.

Se ha implementado el siguiente **código para realizar las peticiones y comprobaciones**:

```
medicamentos = set([])

# Para cada palabra, se obtienen sus ID de búsqueda en Wikidata
# con el objetivo de validar que realmente sean medicamentos
for term in terminos:
    jsonIds = freeTextSearch(term)
    ids = []

    for entity in jsonIds:
        ids.append(entity["id"])

    ids = "|".join(ids)

    urlProps = baseURL + "wbgetentities&ids="+ urllib.parse.quote_plus(ids) + "&languages=en&format=json"
    data = urllib.request.urlopen(urlProps).read()
    data = json.loads(data)

    for id in ids.split("|"):
        try:
            if "P31" in data["entities"][id]["claims"]:
                if data["entities"][id]["claims"]["P31"][0]["mainsnak"]["datavalue"]["value"]["id"] == "Q12140":
                    print(term)
                    medicamentos.add(term.lower())
            else:
                break
        except:
            break

with open("medicamentos.txt", "wb") as f:
    for med in medicamentos:
        f.write(med.encode("UTF-8") + "\n".encode("UTF-8"))
```

Además, se ha reutilizado la función `freeTextSearch()` proporcionada por el profesor **Daniel Gayo-Avello**, para realizar las peticiones de consumo Web a la API de Wikidata:

```
90  def freeTextSearch(search):
        searchURL=baseURL+"wbsearchentities&search="+urllib.parse.quote_plus(search)+"&language=en&format=json"
        data=urllib.request.urlopen(searchURL).read()
        data=json.loads(data)
        return data["search"]
```

Una vez finalizada la ejecución del script, para un máximo de 3000 términos (cuyo tiempo de ejecución ronda entre los 40 y 50 minutos), **se obtiene una lista de 16 medicamentos**:

```
1  vyvanse
2  epinephrine
3  nystatin
4  adderall
5  lisdexamfetamine
6  invega
7  seroquel
8  dopamine
9  clindamycin
10 sedative
11 meth
12 tylenol
13 adderal
14 quetiapine
15 paliperidone
16 amlodipine
17
```

Ejercicio 4

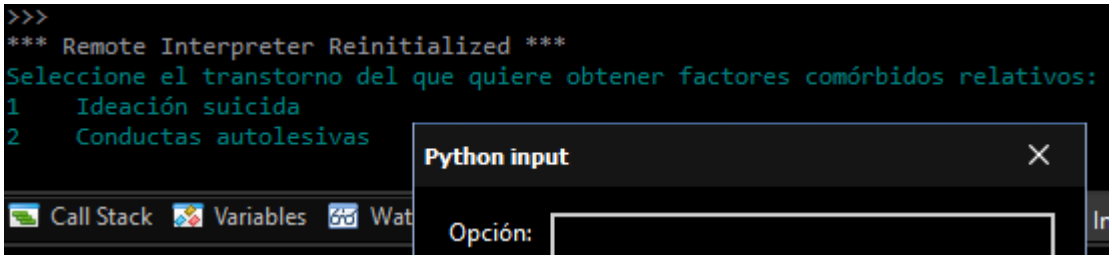
En el cuarto y último ejercicio, se pide realizar una búsqueda **de factores comórbidos relativos a la ideación suicida y a las conductas autolesivas**; es decir, la búsqueda de términos asociados a posibles trastornos y consecuencias a raíz de un trastorno o enfermedad mental padecida por una persona.

Para afrontar este ejercicio, fue necesario crear un nuevo índice denominado `ejercicio4` con el objetivo de permitir trabajar sobre cuatro campos: `selftext`, `title`, `author` y `subreddit`.

Este índice puede ser generado a partir de la ejecución del script `bulk-indexer-Ejercicio4.py` que se encuentra adjunto al entregable.

```
        "mappings": {
            "properties": {
                "edited": {
                    "type":"text"
                },
                "crosspost_parent_list.edited": {
                    "type":"text"
                },
                "title": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                },
                "selftext": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                },
                "author": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                },
                "subreddit": {
                    "type":"text",
                    "analyzer":"palabras_vacias_ingles_porter",
                    "fielddata": "true"
                }
            }
        }
    }
```

El ejercicio en sí se encuentra desarrollado en el script `ejercicio4.py`, que pide una entrada por consola para obtener los factores comórbidos relativos a una temática específica.



Para obtener la lista completa de candidatos a factores comórbidos relativos, se ejecutará una consulta inicial que buscará en función de una cadena de texto.

```
# Búsqueda del subconjunto de posibles factores comórbidos
busqueda = es.search(
    index = "ejercicio4",
    body = {
        "query": {
            "query_string": {
                "fields": [
                    "selftext",
                    "title",
                    "author",
                    "subreddit"
                ],
                "query": consulta
            }
        },
        "aggs": {
            "terms_suicida_title": {
                "significant_terms": {
                    "field": "title",
                    "size": 500,
                    "gnd": {}
                }
            },
            "terms_suicida_author": {
                "significant_terms": {
                    "field": "author",
                    "size": 500,
                    "gnd": {}
                }
            },
            "terms_suicida_selftext": {
                "significant_terms": {
                    "field": "selftext",
                    "size": 500,
                    "gnd": {}
                }
            },
            "terms_suicida_subreddit": {
                "significant_terms": {
                    "field": "subreddit",
                    "size": 500,
                    "gnd": {}
                }
            }
        }
    }
)
```

La cadena de texto a utilizar (almacenada en la variable `consulta`) variará en función de la opción seleccionada. Cada opción utiliza la siguiente lista de cadenas de caracteres:

```
    # Definimos la consulta a realizar en función de la opción escogida
38  palabras = []
    if (opcion == 1):
40      palabras = ["suicide", "suicidal", "kill myself", "killing myself", "end my life"]
    elif (opcion == 2):
        palabras = ["self harm"]
    else:
        print("El valor introducido no es válido. Debes seleccionar un valor entre 1 y 2 inclusive.")
        return
```

Dicha lista será posteriormente procesada para adaptarse al formato de una `query_string` en Elasticsearch, de la siguiente manera:

```
# Formación de parámetro query a partir del array de palabras
consulta = ""
for i in range(0, len(palabras)):
    consulta += "(" + palabras[i] + ")"
    if (len(palabras) != 1 and i < len(palabras) - 1):
        consulta += " OR "
```

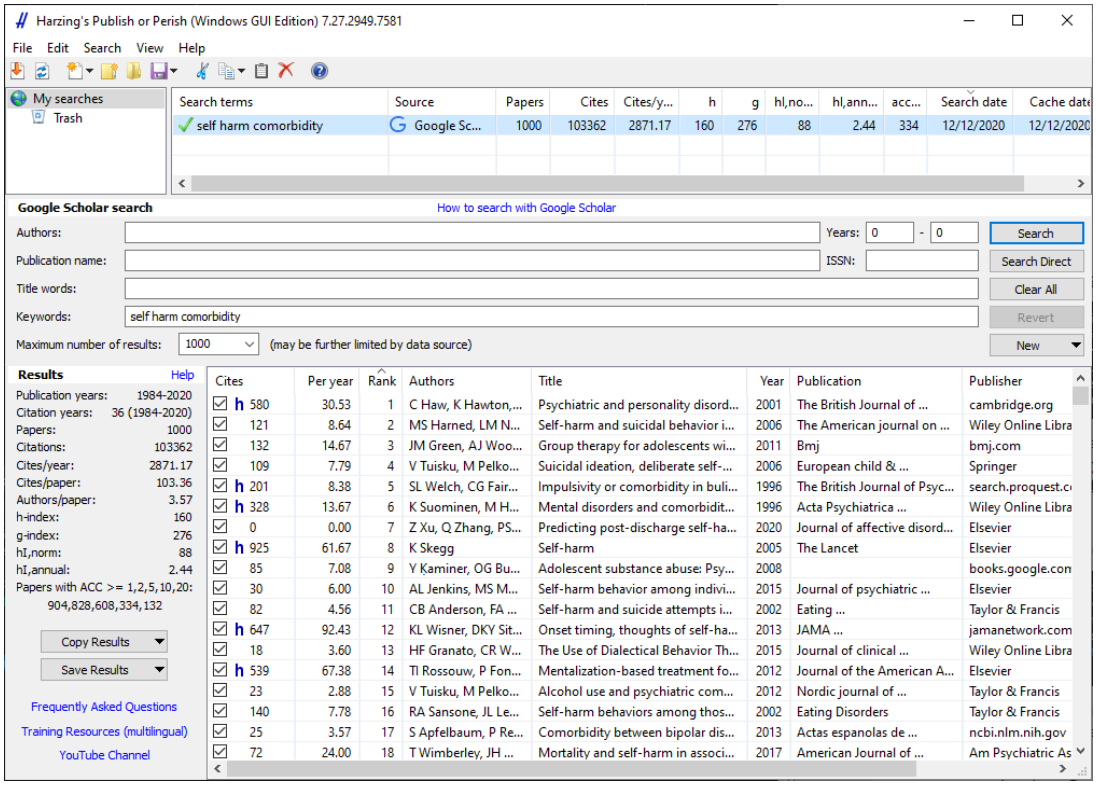
De esta manera, la lista `["suicide", "suicidal", "kill myself", "killing myself", "end my life"]` se convertiría a la cadena `(suicide) OR (suicidal) OR (kill myself) OR (killing myself) OR (end my life)`.

Una vez realizada la búsqueda, extraemos los términos utilizados en una lista.

```
# Recogemos los términos significativos obtenidos
terminos = []
for agg in busqueda["aggregations"]:
    for term in busqueda["aggregations"][agg]["buckets"]:
        if term not in palabras:
            terminos.append(term["key"])
```

No obstante, dichos términos necesitan ser validados de alguna manera, al igual que se realizó en el ejercicio anterior a través de la API de Wikidata.

Para realizar la validación, utilizaremos la herramienta ***Publish or Perish***, recomendada por el profesor Daniel Gayo-Avello durante la sesión de prácticas de laboratorio de la asignatura.



A través de esta herramienta, es posible obtener una lista de publicaciones (utilizando como fuente de información, en este caso, **Google Scholar**), la cual puede ser exportada a diferentes extensiones de archivos; entre ellas, JSON. De esta manera, podremos abrir el fichero obtenido con los resultados para validar los términos obtenidos contra estos.

Para realizar las búsquedas, en el caso de la ideación suicida, se han utilizado las keywords **`suicide comorbidity`**, mientras que para las conductas autolesivas se han utilizado las keywords **`self harm comorbidity`**.

Una vez exportados los resultados, se procede a cargar los títulos de las obras almacenadas en el JSON correspondiente a la opción seleccionada por el usuario al principio del script.

```
titulos = []
fichero = ""
if (opcion == 1):
    fichero = "SuicideComorbidity.json"
elif (opcion == 2):
    fichero = "SelfHarmComorbidity.json"

# Cargamos la lista de títulos del JSON correspondiente
with open(fichero, encoding = "utf-8-sig") as f:
    datos = json.load(f)
    for d in datos:
        titulos.append(d["title"])
```

Para realizar la validación de los términos obtenidos mediante la consulta de Elasticsearch, basta con **comprobar si cada uno de los términos está incluido en la lista de títulos** obtenidos a partir del JSON de *Google Scholar*:

```
# Nos quedamos con aquellos términos que estén en la lista correspondiente
salida = []
for term in terminos:
    for titulo in titulos:
        if term in titulo.split() and term not in salida:
            print(term)
            salida.append(term)
```

Los términos finales serán almacenados en la lista `salida`, tal y como se muestra en la imagen anterior. El contenido de esta lista será volcado a un fichero TXT denominado **`salida.txt`**, donde se mostrará en cada línea un factor comórbido relativo:

```
# Guardamos los resultados en un fichero TXT
with open("salida.txt", "wb") as f:
    for term in salida:
        f.write(term.encode("UTF-8") + "\n".encode("UTF-8"))
```