Ejercicio 2

Descripción general

Para el cálculo de la similitud, se desarrolló la clase `Similarity`, la cual toma como atributos el dataset de nombres limpiado y transformado, un nombre y un umbral de porcentaje de similitud ingresados por el usuario y devuelve un diccionario, el cual, por cada `"ID"` (llave) de usuario que cumpla con este umbral, contiene un diccionario subsiguiente, que contiene el nombre del dataset con el cual el nombre de entrada fue comparado (`"name"`) y el porcentaje de similitud (`"similarity"`) entre ambos nombres.

Usando la librería `flask` se implementó una API, la cual, a través de un formulario HTML, solicita al usuario final el nombre y el umbral de porcentaje de solicitud mencionados y entrega los resultados calculados por el algoritmo incluido en la clase, en formato JSON.

Contenido

En el directorio solución del ejercicio (**Ejercicio_2**), se encuentran los siguientes directorios y archivos:

- **templates**. Directorio que contiene el template *.html* utilizado para el formulario de la API implementada (**form.html**).
- app.py. Archivo principal. Contiene la configuración y la ejecución de la API implementada.
- similarity.py. Código fuente de la clase construida para el ejercicio.
- **names_dataset.csv**. Dataset que contiene los nombres con los que se comparará el nombre-entrada del usuario.

Librerías utilizadas

Se utilizaron las siguientes librerías:

- **`flask`** : Para la creación de la API requerida.
- **`pandas`** : Para la carga y limpieza del dataset de nombres.
- **`unidecode`** : Para la eliminación de acentos del dataset de nombres.
- **`Levenshtein`** : Para el cálculo de la similitud entre las cadenas de texto.

Dataset

El dataset fue importado por medio del método `read_csv()` de la librería `pandas`. Para los cálculos de similaridad, se creó una columna nueva de nombres llamada `"full_name_transformed"`, **la cual sólo tiene en cuenta los caracteres alfanuméricos en minúscula, sin espacios ni acentos, de la columna de nombres (`"Full Name"`) para facilitar las comparaciones de acuerdo al algoritmo escogido (basado en la distancia de Levenshtein)**. De manera similar, se creó una nueva columna de nombres llamada `"full_name_cleansed"`, **la cual limpia la columna `"Full Name"` de caracteres que no sean alfanuméricos (a excepción de espacios), para mostrar un output limpio al usuario final**. También se eliminaron del dataset, tanto para el output como para los cálculos, los títulos antecedidos a los nombres de las personas ("Sr.", "Dr.", "Lic.", etc.).

Para tales propósitos, se utilizó el método `.map()` de `pandas` y una función lambda que, para cada cadena de texto presente en la columna `"Full Name"`, realiza las tareas de eliminación de acentos (usando la función `unidecode` de la librería homónima), conversión a minúsculas (usando el método nativo `.lower()`), limpieza de caracteres no alfanuméricos (usando el método nativo `.isalnum()`) y limpieza de títulos que anteceden los nombres (usando el método nativo `.replace()`).

El dataset transformado es cargado como atributo de la clase `Similarity`.

Cálculo del porcentaje de similitud

Para el cálculo de la similitud del nombre ingresado por el usuario y los nombres contenidos en el dataset, se utilizó la distancia de Levenshtein, normalizándola y convirtiéndola a porcentaje con la siguiente fórmula:

$$Porcentaje\ de\ similitud\,[\%] = \left(1 - \frac{D_L}{máx(L_1, L_2)}\right) \times 100$$

Donde D_L es la distancia de Levenshtein entre el nombre ingresado por el usuario, cuya longitud es (L_1) y cada nombre (de longitud L_2) presente en el dataset de comparación entregado, sobre el cual se itera para hacer la comparación uno a uno. Para cuando la máxima longitud entre ambos nombres ($máx(L_1, L_2)$) es cero, se hace la excepción particular de que el porcentaje de similitud entre ambos nombres es 100%, pues, efectivamente, esto sólo puede suceder cuando ambas cadenas de texto son nulas, y así se evita el error de la división entre cero en la fórmula mostrada arriba.

Output

Se crearon dos diccionarios vacíos: uno para almacenar cada uno de los ID (`"ID"`) de las personas del dataset, y otro para almacenar el nombre (`"name"`) y su porcentaje de similitud (`"similarity"`). Este último diccionario está anidado en el anterior, y ambos son llenados en cada iteración sobre el dataset de nombres, **siempre y cuando se cumpla con el umbral de similitud ingresado por el usuario**. Los datos son ordenados por el porcentaje de similitud usando la función `.sorted()` y una función lambda que entra hasta el diccionario anidado y selecciona este parámetro para el ordenamiento del diccionario principal.

Una vez ordenado el diccionario, se procedió a crear la API. Se creó un nuevo ejecutable (**app.py**), se importó la librería `flask`, se instanció la clase `Flask` para la app y se definió un template para el Frontend del servicio, el cual, en este caso, es un sencillo formato HTML (**form.html**) para el ingreso por parte del usuario del nombre y del umbral que necesite. Luego de esto, se creó una función (`process`) para procesar las solicitudes recibidas por el usuario a través del formulario (nombre y umbral de similitud), instanciar la clase `Similarity` para poder generar el diccionario con los usuarios que cumplieron el umbral de similitud y convertir este diccionario en un JSON, utilizando el método `json.dumps()` (permitiendo en éste poder visualizar caracteres, como los que tienen acentos, y **no** permitiendo su ordenamiento automático por llaves, como sucede automáticamente cuando se visualiza en algunos navegadores). Para efectos de este ejercicio, se creó el código para su visualización automática en el navegador una vez el usuario envía los datos ingresado, pero puede también generarse un archivo *.json* de descarga automática, modificando las dos últimas líneas de código antes del último `return` de la función `process` de la siguiente manera:

```
response=Response(result_json, content_type='/process; charset=utf-8')
return  response
```

Esta versión alternativa puede verse en el directorio **Ejercicio_2_Alt_1**.

También se puede utilizar la función `jsonify()` de `flask`, en vez de `json.dumps()` para crear un JSON con el resultado, lo cual permite visualizarlo con un mejor formato desde el navegador. Sin embargo, en la mayoría de los navegadores, se ordena automáticamente el JSON por llaves y no por uno de sus atributos, como es el caso necesario de este ejercicio. De igual manera, se incluye esta versión alternativa en el repositorio, la cual puede verse de en el directorio **Ejercicio_2_Alt_2**.