Estuve analizando el enunciando planteado, y las recomendaciones de mismo en cuanto a los algoritmos a utilizar, entonces

Realicé las siguiente funciones,

ExactPhrase(*s1*, *s2* *string*) *bool*

: compara dos cadenas de texto y devuelve true si son exactamente iguales

PercentageIncludedWords(*s1*, *s2* *string*) *float64*

Devuelve un valor entre 0 y 1 que representa el porcentaje de palabras incluidas en s2 que también están en s1.

LevenshteinDistance(*s1*, *s2* *string*) *int*

La función calcula la distancia de Levenshtein entre dos cadenas de texto, una medida de la diferencia entre ellas expresada como el número mínimo de operaciones necesarias para transformar una cadena en la otra

CompareAll(*entidad*, *comparador* *string*, *sensPercentageIncludeWords* *float64*, *sensLevenshteinDistance* *int*) *bool*

Esta sería la función principal a utilizar para las comparaciones. CompareAll utiliza todas las funciones anteriores para determinar si la comparación es verdadera. Además, asigna un peso a cada una de las funciones para influir en el resultado. Por ejemplo, si la función ExactPhrase es verdadera, CompareAll siempre devolverá verdadero. Si el parámetro sensPercentageIncludeWords es mayor o igual al resultado de la función PercentageIncludeWords, CompareAll devolverá verdadero. Del mismo modo, si el parámetro sensLevenshteinDistance es menor o igual al resultado de la función LevenshteinDistance, CompareAll devolverá verdadero. En cualquier otro caso, devolverá falso.

Ahora bien, analizando los ejemplos de los resultados obtenidos, por ejemplo cuando:

Entidad: *La persona de Pedro Sanchez*

*el gobierno de Pedro Sanchez es FALSO.*

*Spain PM Pedro Sanchez es VERDADERO.*

Evidentemente, las funciones no van a distinguir el contexto de estas comparaciones.

Por este motivo, creamos la función..

VectorSimilarity( *s1*, *s2* *string*, *pathtofilevectors* *string* )

que utiliza Word Embeddings, Esta función vectoriza las frases y utiliza la comparación de cosenos para replicar los resultados. Cuanto más cercano esté el retorno a 1, mayor será la similitud entre las frases.

Sin embargo, es importante destacar un inconveniente: esta función requiere aprender de una fuente de datos adecuada, como podría ser la Wikipedia en español, que lamentablemente no pudimos obtener. En su lugar, creamos un conjunto de datos propio con palabras en español.

Con los datos adecuados, también reconocemos que no se garantiza un conjunto de resultados ideal, ya que carecerá del contexto necesario para analizar las entidades. Por ejemplo, en el caso de "Pedro Sánchez", sería necesario indicar a la función que el resultado debería ser VERDADERO cuando el objeto de comparación se refiere a la persona de Pedro Sánchez.

Además, esta función consume muchos recursos, por lo que se recomienda utilizarla como un servicio de microservicio en una API Rest.

Como conclusión de nuestros análisis, observamos que la solución óptima sería la implementación de un servicio de inteligencia artificial, que permita especificar todos los contextos y modificadores que podrían influir en los resultados. Para ello, es necesario investigar la existencia de alguna biblioteca que pueda ser consumida desde Go.

Aunque hemos expresado nuestras opiniones, les solicitamos que verifiquen las funciones creadas y nos brinden feedback para determinar si es necesario continuar con la investigación. Hemos subido las funciones implementadas junto con sus pruebas correspondientes a GitHub.

**Como utilizar los test.**

En el main.go hacemos las llamadas a las funciones, especificando la entidad a buscar, y un arreglo de strings donde se buscará. Estas pruebas devuelven por pantalla la entidad con la cual se está comparando y su resultado. En datag.go están los datos a comprar por si quieren modificarlos.

Para el caso de la prueba de VectorSimliarity, como solo cargamos ciertas paralabras, las pruebas deberán acotarse a palabras “comunes” del español. Es importate descomprimir el archivo words\_vector.txt para la realización de estos tests.