Resumen CBRKit

CBRkit es un toolkit modular en Python para el Razonamiento Basado en Casos. Actualmente se encuentra en desarrollo y en su versión actual permite cargar casos, definir medidas de similitud y realizar la recuperación sobre bases de casos en memoria.

Más detalles:

- Codigo y descarga (descarga de la versión 0.14.2))
- Documentación
- Paper y video de presentación

Los siguientes módulos son parte de la versión actual de CBRkit:

- **cbrkit.loaders**: Funciones para cargar casos y consultas.
- **cbrkit.sim**: Funciones de similaridad para tipos de datos comunes: cadenas, números, colecciones, etc.
- **cbrkit.retrieval**: Funciones para definir y aplicar *pipelines* de recuperación.
- **cbrkit.typing**: Definiciones de tipos genéricos para definir funciones personalizadas.

1. Carga de casos (módulo cbrkit.loaders)

La librería CBRkit incluye un módulo de cargadores que permite leer datos de diferentes formatos de archivo y convertirlos en una "base de casos". Los mismos métodos pueden emplearse para cargar "consultas" (casos a resolver).

- Dependiendo del formato de carga, la "base de casos" (*Casebase* en terminología de CBRkit) será:
  - un "diccionario de diccionarios" o un "diccionario de objetos" para las funciones de carga desde:
    - JSON (`dict[typing.Any, typing.Any]`)
    - YAML (`dict[typing.Any, typing.Any]`)
    - CSV (`dict[int, dict[str, str]]`)
    - TOML (`dict[str, typing.Any]`)
    - XML (`dict[str, typing.Any]`)
  - un "mapeo" de *Series* para la función `dataframe()` que carga *DataFrames* de Pandas (`Mapping[typing.Any, pandas.core.series.Series]`)

    - un "mapeo" de un *tipo clave* a un *tipo valor* (el tipo del caso) para las funciones de carga desde archivos y directorios (`Mapping[typing.Any, typing.Any]`)

  Nota: En las versiones más recientes de CBRkit (v0.18.0) se está unificando el tipado de los elementos de la librería y se migra hacia un tipo de dato *Casebase* que unificará esta variedad de formatos internos para las bases de casos.

- Adicionalmente, es posible forzar la validación de tipo de los casos cargados utilizando definiciones de la libreria Pydantic

Funciones principales:

- `cbrkit.loaders.csv(path)`: Lee un archivo CSV y lo convierte en un diccionario.
- `cbrkit.loaders.json(path)`: Lee un archivo JSON y lo convierte en un diccionario.
- `cbrkit.loaders.yaml(path)`: Lee un archivo YAML y lo convierte en un diccionario.
- `cbrkit.loaders.xml(path)`: Lee un archivo XML y lo convierte en un diccionario.
- `cbrkit.loaders.file(path)`: Delegando en los métodos anteriores, convierte un archivo (CSV, JSON, XML, YAML, etc) en una "base de casos".
- `cbrkit.loaders.folder(path, pattern)`: Convierte todos los archivos en una carpeta que coincidan con un patrón específico en un *Casebase*.
- `cbrkit.loaders.validate(data, validation_model)`: Valida los datos (diccionario de casos) contra un modelo de Pydantic, arrojando errores si los datos no coinciden con el modelo esperado.

Las mismas funciones que se utilizan para cargar las "bases de casos" se puden utilizar para cargar el caso o casos "a resolver" por el sistema CBR.

- La única restricción es que el tipos de los "casos a resolver" (casos *query*) que utilicen una "base de casos" debe de ser compatible con el tipo de los casos almacenados en la misma.

Nota. En el caso de la carga desde archivos JSON (con la función `json()`) que se emplea en el ejemplo de la tasación, se espera que el fichero de entrada contenga una lista/array de objetos JSON o un diccionario (tabla hash) de objetos JSON.

- En ambos caso la "base de casos" cargada será un diccionario Python con los casos.
- En el caso de que el fichero de entrada contuviera lista de objetos se usará como clave e identificador del caso su índice en la lista/array.
- Si el fichero JSON de entrada ya era un diccionario/tabla hash se mantienen las claves utilizadas en el mismo.

2. Fase "Recuperar" (módulo cbrkit.retrieval)

El módulo **cbrkit.retrieval** ofrece clases y funciones de utilidad para dar soporte a la recuperación de casos basada en métricas de similaridad y un conjunto de tipos de datos para encapsular los resultados de una búsqueda por similaridad en la base de casos

2.1 Construcción de recuperadores (`cbrkit.retrieval.build()`)

- La clase de utilidad `build(similarity_func, limit, min_similarity, max_similarity)` permite **crear funciones de recuperación** personalizadas basadas en una función de similitud.
    - Se pueden establecer límites en el número de casos devueltos y filtrar por similitud mínima y máxima.
    - El valor de retorno es una "función de recuperación" (`cbrkit.typing.RetrieverFunc`) que puede ser utilizada por las funciones `apply()` o `mapply()` para consultar una "base de casos"

2.2 Recuperación de casos similares (`cbrkit.retrieval.apply()` y `cbrkit.retrieval.mapply()`)

- La función de utilidad `apply(casebase, query, retriever/s, processes)` lanza un *"caso consulta"* (caso a resolver) `query` contra una "base de casos" `casebase` aplicando las métricas de similaridad de uno o varios `retriever` proporcionados por la función `build()` para **recuperar** los **casos más similares**.
  - Puede paralelizarse el cálculo de similaridad sobre la base de casos indicando el nº de procesos a ejcutar en paralelo
- La función de utilidad `mapply(casebase, queries, retriever/s, processes, parallel)` es una generalziación de la anterior que permite lanzar múltiples *"casos consulta"* (*queries*) a una base de casos, soportando paralelización (`parallel`) por consultas o por base de casos.

2.3 Resultados de la recuperación

- La clase `Result` encapsula los resultados de la recuperación. En concreto ofrece:
  - atributo `ranking`: lista ordenada con los `ids` (claves) de los *n* casos más similares
  - atributo `similarities`: lista ordenada de *n* valores de tipo `float` con los valores de similaridad de los *n* casos más similares
  - atributo `casebase`: base casos (= diccionario) con los *n* casos más similares
  - atributo `steps`: lista ordenada de objetos `ResultStep` con la información de los resultados intermedios en el caso de utilizar multiples *Retrievers*
- La clase `ResultStep` representa la información de cada paso en el proceso de recuperación, incluyendo similitudes y rankings.

3. Métricas de similaridad (módulo cbrkit.sim)

El módulo **cbrkit.sim** (y sus submódulos) incluye un conjunto de métricas de similaridad para distintos tipos de datos, como números, cadenas de texto, listas y datos genéricos.

- Proporciona una implementación de diversos tipos de métricas específicas, que junto con funciones de similaridad definidas por el programador, pueden ser utilizadas para configurar los `Retrievers` creados con la función `cbrkit.retrieval.build()`
- Ofrece la clase/función de utilidad `attribute_value()` que permite definir una métrica de similaridad compleja que combine un conjunto de funciones de similaridad específicas para cada atributo de los casos procesados.
- Define la clase/función de utilidad `aggregator()` que permite especificar el tipo de combinación de métricas (media, máximo, etc) y su ponderación

3.1 Especificación de métricas de atributos (`cbrkit.sim.attribute_value()`)

La función/clase de utilidad `attribute_value(attributes, aggregator, value_getter)` permite definir una **métrica de similaridad compuesta** que calcule la similaridad entre dos casos a partir de la combinación de valores de similaridad entre pares de atributos.

- En el parámetro `attributes` se vincula a cada *nombre de atributo* del caso con la *función de similaridad* a utilizar al comparar sus valores
  - Las funciones de similaridad pueden ser las propocionadas en los submódulos `cbrkit.sim.*` o funciones de similaridad específicas definidas por el programador

- En el parámetro `aggregator` se especifica el método de combinación de métricas a utilizar (`'mean', 'fmean', 'geometric_mean', 'harmonic_mean', 'median', 'median_low', 'median_high', 'mode', 'min', 'max', 'sum'`) y, opcionalmente, los pesos con los que se combinan las métricas de atributos

3.2 Funciones de similaridad para atributos numéricos (`cbrkit.sim.numbers`)

El módulo **cbrkit.sim.numbers** proporciona varias funciones de similitud para valores numéricos

- `linear_interval(min, max)`: Calcula la similitud lineal entre dos valores dentro de un intervalo definido por un mínimo y un máximo. La similitud se basa en la distancia relativa de los valores dentro de este rango.
- `linear(max[, min])`: Similar a `linear_interval`, pero no se limita a un rango específico. Define un mínimo y un máximo, donde la similitud es 1.0 en el mínimo y 0.0 en el máximo.
- `threshold(umbral)`: Devuelve una similitud de 1.0 si la diferencia absoluta entre dos valores es menor o igual a un umbral definido; de lo contrario, devuelve 0.0.
- `exponential(alpha)`: Utiliza una función exponencial para calcular la similitud, controlada por un parámetro *alpha*. Un valor de alpha más alto provoca una disminución más rápida de la similitud.
- `sigmoid(alpha, theta)`: Implementa una función sigmoide para calcular la similitud, donde *alpha* controla la pendiente de la curva y *theta* determina el punto en el que la similitud es 0.5.

3.3 Funciones de similaridad para colecciones (`cbrkit.sim.collections`)

El módulo **cbrkit.sim.collections** proporciona varias funciones para calcular la similitud entre atributos que almacenan colecciones y secuencias

- `isolated_mapping(element_similarity)`: Compara cada elemento de una secuencia con todos los elementos de otra, utilizando la función de similaridad proporcionada (`element_similarity`) y tomando el máximo de similitud para cada elemento
- `mapping(similarity_function, max_queue_size)`: Implementa un algoritmo A* para encontrar la mejor coincidencia entre elementos basándose en la función de similaridad proporcionada
- `sequence_mapping(element_similarity, exact)`: Calcula la similitud entre dos secuencias utilizando la función de similaridad proporcionada (`element_similarity`) para comparar posición a posición sus elementos
- `sequence_correctness(worst_case_sim)`: Evalúa la similaridad de dos secuencias comparando sus elementos, otorgando el valor `worst_case_sim` cuando todos los pares son discordantes y valores proporcionales cuando hay algunas correspondencias
- `jaccard()`: Calcula la similitud de Jaccard entre dos colecciones, midiendo la razón entre la intersección sobre la unión.
- `smith_waterman(match_score, mismatch_penalty, gap_penalty)`: Realiza un alineamiento de Smith-Waterman, permitiendo ajustar los parámetros de puntuación
- `dtw()` (*Dynamic Time Warping*): Calcula la similitud entre secuencias que pueden variar en longitud.

3.4 Funciones de similaridad para atributos de tipo String (`cbrkit.sim.strings`)

El módulo **cbrkit.sim.strings** ofrece varias funciones para calcular la similaridad entre cadenas de texto.

(a) Métricas de **comparación de cadenas**:

- `levenshtein(score_cutoff, case_sensitive)`: Calcula la similaridad normalizada entre dos cadenas basándose en la distancia de Levenshtein o distancia de edición (número de inserciones, eliminaciones y sustituciones). Puede especificarse un umbral y la diferenciación entre mayúsculas y minúsculas.
- `jaro(score_cutoff)`: Calcula la similaridad entre dos cadenas utilizando el algoritmo de Jaro.
- `jaro_winkler(score_cutoff, prefix_weight)`: Variante del algoritmo de Jaro que tiene en cuenta los prefijos comunes entre las cadenas

(b) Métricas de comparación de **representaciones vectoriales** (usan modelos del lenguaje para convertir las cadenas en "vectores semánticos" [*embeddings*]):

- `spacy(model_name)`: Utiliza un modelo de la librería NLP spaCy para calcular la similaridad semántica entre pares de textos mediante vectores de palabras
- `sentence_transformers(model_name)`: Usa un modelo preentrenado de la librería Sentence-Transformers y calcula la similaridad semántica entre textos usando vectores de palabras y la métrica del coseno
- `openai(model_name)`*: Utiliza los modelos de *embeddings* disponibles en el API de OpenAI (requiere una *API key*) para calcular la similitud semántica entre pares de textos mediante la métrica del coseno

3.5 Funciones de similaridad para atributos organizados en Taxonomias (`cbrkit.sim.strings.taxonomy`)

El módulo **cbrkit.sim.strings.taxonomy** permite trabajar con **taxonomías**, que son estructuras jerárquicas de categorías.

- Cada categoría se representa como un nodo con atributos como nombre, peso y posibles hijos
- Las clases `Taxonomy` y `TaxonomyNode` del módulo se usan para representar la organización jerárquica de las categorías de la Taxonomía
- Se dispone de la función/clase de utilidad `load()` para cargar los elementos de una Taxonomía desde ficheros en formato JSON, YAML o TOML y de métricas de similaridad que explotan las relaciones jerárquicas de los nodos

Carga de Taxonomías y función de similaridad (`cbrkit.sim.strings.taxonomy.load()`)

La función/clase de utilidad `load(path, measure)` carga una taxonomía desde un archivo (en formato JSON, YAML o TOML) y devuelve una función para medir la similaridad entre categorias.

Las **métricas de similaridad** sobre taxonomias disponibles son:

- `wu_palmer()`: Calcula la similaridad entre dos nodos de la taxonomía utilizando el método de Wu & Palmer basado en la profundidad de los nodos y la de su ancestro común más cercano (LCA)
- `user_weights(strategy)`: Calcula la similaridad entre nodos basándose en pesos definidos por el usuario en el archivo de la taxonomía
- `auto_weights(strategy)`: Calcula automáticamente los pesos de los nodos basándose en la profundidad de los mismos
- `node_levels(strategy)`: Mide la similaridad entre nodos según su nivel en la jerarquía

- `path_steps(weightUp, weightDown)`: Mide la similaridad basándose en los pasos hacia arriba y hacia abajo desde el ancestro común más cercano (LCA)

3.6 Funciones de similaridad genéricas (`cbrkit.sim.generic`)

El módulo **cbrkit.sim.generic** ofrece funciones de similitud que no están limitadas a tipos de datos específicos

- `table(entries, default, symmetric)`: Permite asignar valores de similitud desde una tabla definida por una lista de entradas (`entries`) que relacionan pares de valores con su similitud.
    - Se puede definir si la tabla es simétrica y establecer un valor de similitud predeterminado para pares que no estén en la tabla.
      Nota: En versiones recientes de CBRkit se diferencia entre `dynamic_table()` y `static_table()`, siendo esta última la opción equivalente a `table()`
- `equality()`: Devuelve una similitud de 1.0 sólo si los dos valores son iguales y 0.0 si son diferentes.