Extracción de secuencias maximales de una colección de textos

Introduccion

Algoritmo capaz de detectar las secuencias maximales que son frecuentes en una colección de textos dada. para comparar la similitud entre varios

Explicacion Teorica

***\*\*Estado del Arte***

hemos implementado un algoritmo incremental, basado en *pattern-growing* (crecimiento de patrones). Este algoritmo es capaz de encontrar las secuencias idénticas de palabras que se repiten X veces en la colección de documentos, siendo X un umbral de frecuencia determinado por el usuario. Además, le hemos añadido la posibilidad de definir un segundo umbral, de capacidad de salto, mediante el cual el usuario indica cuantas palabras podemos ignorar entre dos palabras que se tienen en cuenta para la secuencia

nuestro algoritmo no solo será capaz de encontrar las secuencias maximales que se repiten con cierta frecuencia, sino que podremos añadirle cierta flexibilidad y permitir que deje huecos entre palabras de la secuencia, detectando no sólo secuencias exactas, sino también “parecidas”.

Mineria de datos

KDD ()(Fayyad, 1996), se refiere al “proceso de descubrir conocimiento útil a partir de datos de bajo nivel que se encuentran en grandes colecciones de datos”

La clave en el proceso de descubrimiento del conocimiento es el paso de data mining (minería de datos), que según (Fayyad, 1996): “consiste en aplicar ciertos algoritmos y análisis a los datos que, con un coste computacional aceptable, produzcan una enumeración particular de patrones sobre los datos

actualmente está cobrando mayor importancia la información contenida en datos no estructurados, como colecciones de textos. Es aquí donde aparece la minería textual

**Mineria de textos**

La minería textual está orientada a la extracción de conocimiento a partir de datos en lenguaje natural no estructurados, almacenados en bases de datos textuales. Se identifica con el descubrimiento de conocimiento en los textos y se le denomina comúnmente Knowledge-Discovery in Text (KDT).

El procesamiento de grandes colecciones de texto no estructurado para extraer conocimiento requiere aplicar técnicas tales como la identificación y extracción de patrones, el análisis de *clustering* (no se conocen a priori las clases o categorías) o clasificación (clases definidas a priori), etc.

Los algoritmos de *Sequential Pattern Mining* afrontan el problema de descubrir las secuencias frecuentes en una colección de textos dada

algoritmos de descubrimiento de patrones en colecciones de textos. Las dos principales aproximaciones son las basadas en métodos a priori y las basadas en crecimiento de patrones (*pattern-growth*).

Los algoritmos más conocidos en éste ámbito son *Generalized Sequential Pattern algorithm* (GSP) y *PrefixSpan*, y representan las dos aproximaciones al problema: a priori y *pattern-growth*, respectivamente.

GSP empieza con una secuencia frecuente de longitud 1, luego las secuencias potencialmente frecuentes k+1 se van formando a partir de las secuencias frecuentes anteriores k. Esto evita que en cada iteracion se este validando cada secuencia formada según el numero de secuencias. La secuencia es frecuente si por lo menos se encuentra repetidamente en al menos BETA documentos de toda la bd, entonces esta secuencia pasa a formar parte del conjunto de secuencias potencialmente frecuentes k+1, BETA es el umbral de frecuencia minimo fijada por el usuario. Optimizaciones : crear un hash tree, el sufijo de una secuencia sea el prefijo de otra secuencia k-1 de esa manera solo se podrá crear un nuevo candidato k, un candidato se elimina si al menos una subsecuencia suya no es frecuente

*PrefixSpan* los patrones se van formando mientras restringen la búsqueda de secuencias en una base de datos proyectada **ALFA** formada por subsecuencias que son sufijos de las frecuencias que tienen **ALFA** como prefijo. Merito: la proyección de la BD original en subconjuntos pequenos de donde se saca los patrones progresivamente. BD->subsecuencias sufijas.

A diferencia de los métodos a priori, los métodos basados en *pattern-growth* no necesitan generar candidatos y luego comprobar si son frecuentes en la base de datos, sino que hacen crecer los patrones a partir de ítems frecuentes, de modo que son más eficientes.

Pattern –growth idea principal es reducir la creacion de candidatos y reducir la búsqueda a una parte de la base de datos.

Las secuencias maximales pueden ser de gran utilidad, puesto que podían representar las partes más importantes de los textos. Dada una colección de textos, el hecho de que algunas secuencias se repitan en varios de ellos, nos señala la importancia de la información que contienen dichas secuencias. También hay que destacar que la aplicabilidad de la extracción de secuencias frecuentes maximales es muy amplia puesto que la técnica es independiente del dominio del problema y del lenguaje en el que estén escritos los textos.

extracción de secuencias maximales frecuentes dada una colección de textos no estructurados. Así pues, buscaremos patrones o sub-secuencias que se repitan en  documentos, siendo  un umbral puesto por el usuario. Estas secuencias serán, por tanto -frecuentes. Además, para que sean maximales, no serán sub-secuencia de ninguna otra secuencia frecuente. Puesto que tratamos con textos largos escritos en lenguaje natural y de temática diversa, la generación de candidatos se haría muy costosa, debido a la explosión combinatoria. Es por eso que utilizaremos un método basado en crecimiento de patrones (*pattern-growth*).

**Secuencias frecuentes maximales**

el hecho de que algunas secuencias se repitan en varios de ellos, nos señala la importancia de la información que contienen dichas secuencias. También hay que destacar que la aplicabilidad de la extracción de secuencias frecuentes maximales es muy amplia puesto que la técnica es independiente del dominio del problema y del lenguaje en el que estén escritos los textos

se pueden utilizar como una descripción de los textos, se pueden utilizar para generar automáticamente resúmenes de los textos a partir de ellas

métodos basados en estilo (autoria), métodos basados en características sintácticas y métodos basados en palabras

***\*\*Similitud entre textos***

Grado de similitud entyre textos.Detectar Plagio.

**2.5 Algoritmo de Busqueda de secuencias maximales**

Dada la estructura anteriormente descrita, y una frecuencia, encontrar todas las secuencias que cumplan, como mínimo, la frecuencia solicitada y que además, sean lo más largas posible.

Este algoritmo está basado en el algoritmo DIMASP, presentado en (García-Hernández et al., 2006). Se trata de un algoritmo incremental basado en *pattern-growing* (crecimiento de patrones). Este algoritmo es capaz de encontrar las secuencias idénticas de palabras que se repiten  veces en la colección de documentos, siendo  la frecuencia indicada por el usuario.

Este algoritmo es capaz de encontrar las secuencias idénticas de palabras que se repiten BETA veces en la colección de documentos, siendo BETA la frecuencia indicada por el usuario

El inconveniente que tiene este modelo para su uso en la detección de similitud entre textos y patrones de escritura es que el algoritmo sólo detectará la secuencia como BETA frecuente si se repite *exactamente igual* en BETA documentos.

Para sortear este obstáculo y que no limite la calidad de nuestros resultados, hemos añadido una nueva característica al algoritmo. El usuario podrá definir un nuevo parámetro, que llamaremos *capacidad de salto, o gap*. La *capacidad de salto* indicará el número máximo de espacios en blanco que se pueden dejar entre dos palabras que pertenecen a una secuencia

La idea básica del algoritmo es, para cada uno de los documentos, ir recorriendo sus palabras y coger aquella que tenga una frecuencia de aparición mayor que el umbral definido por el usuario (). Cuando encontramos una palabra que cumple esta condición, nos la guardamos y miramos qué palabra le sigue. Unimos la palabra siguiente a la actual, obteniendo así un bi-grama, y volvemos a comprobar si sigue superando la frecuencia mínima. Si la supera, agregamos la palabra que le sigue, y volvemos a hacer la comprobación. Seguiremos así hasta que deje de cumplirse, es decir, hasta que la siguiente palabra ya no sea frecuente. De ese modo garantizamos que la secuencia sea maximal, puesto que la hacemos crecer hasta que ya no cumpla la condición.

**2.6 Algoritmo de almacenamiento con comparación**

hay casos en los que las secuencias maximales que encontramos no necesitan ser almacenadas porque, o bien ya están en la colección de secuencias maximales, o están incluidas en alguna de las secuencias maximales de mayor longitud existentes.

El número de comparaciones puede ser muy elevado. Para mejorar la eficiencia del algoritmo habrá que disminuir el número de comparaciones. Nos aprovecharemos de que cada palabra tiene asignado un id (*integer*) que la representa inconfundiblemente. Este id se añadió para almacenarla en la tabla *hash*, y nos será útil ahora. También añadimos un atributo *sumaIndex,* que es la suma (en valor absoluto) de todos los índices que componen la secuencia maximal. El almacenamiento se realiza según la longitud de las secuencias. Denominaremos *k-SM* a una secuencia maximal de longitud *k*.

Se ha elegido una estructura de tabla *hash* de vectores, donde en un mismo vector se almacenan las secuencias de una misma longitud *k*. Dentro de un mismo vector, las palabras se almacenan por orden creciente de su *sumaIndex*. Nótese que para que dos secuencias maximales sean idénticas, su *sumaIndex* deberá ser la misma. Del mismo modo, para que exista la posibilidad de que una secuencia maximal esté incluida en otra, el valor de *sumaIndex* de la contenedora deberá ser estrictamente mayor que el valor de *sumaIndex* de la secuencia maximal contenida, puesto que incluye esta secuencia más algún otro fragmento que hará incrementar el valor de *sumaIndex*

El algoritmo de almacenamiento de las secuencias maximales se divide en tres pasos. En primer lugar debemos comprobar que la secuencia encontrada no se encuentra ya entre las almacenadas con la misma longitud y que no es sub-secuencia de ninguna secuencia maximal de mayor longitud. En caso de que no se diese ninguno de los dos casos anteriores, procedemos a insertarla. Seguidamente, comprobamos que no existe ninguna secuencia ya insertada que esté incluida en la que acabamos de añadir. Para ello se comparan todas las *k-SM* con *k* menor que la longitud de la recién insertada

El atributo de *sumaIndex* se utiliza para mejorar el tiempo de ejecución de la inserción. Si queremos comprobar si la secuencia A es sub-secuencia de la secuencia B, es decir, si A está contenida en B, comprobamos en primer lugar que *sumaIndex(B) > sumaIndex(A)*. Si no se cumple que la *sumaIndex* de B es estrictamente mayor que la *sumaIndex* de A, no es posible que estén contenidas.

**3.2 Similitud entre revisiones del mismo artículo de Wikipedia**

El primero de los experimentos consistirá en comparar las similitudes entre textos co-derivados. Un co-derivado se define como una pareja de documentos que son revisiones o plagios el uno del otro

La comparación se hará con el algoritmo propuesto, es decir, a partir de las secuencias maximales que aparecen en común en los textos. Para ello, iremos comparando cada una de las 9 revisiones con la actual, buscando secuencias maximales que se repitan 2 veces, es decir, que aparezcan en ambos documentos.

Tal y como hemos explicado en el capítulo 1 sección 3, la similitud entre textos vendrá definida por:

podemos decir que el tiempo de ejecución es sensible a la longitud de las secuencias maximales encontradas, más que a la longitud del documento. Se disparan los tiempos de ejecución para el análisis de textos muy similares entre ellos, con similitud entre [0.8, 1].

Hemos ejecutado el algoritmo, cargando los 5 documentos y le hemos pedido que busque las secuencias maximales con una frecuencia mayor o igual a 5, es decir, secuencias maximales que se repitan en al menos 5 documentos distintos

CONCLUSIONES

La extracción de secuencias frecuentes maximales de una colección de textos nos permite conocer en qué grado son similares los distintos textos de una colección. Esto puede ser útil para clasificar. La clasificación automática de textos, consiste en colocar un documento dentro de un grupo de clases previamente definidas

Del mismo modo, se puede hacer una clasificación no-temática, que se haga según los estilos de redacción. De este modo, se pueden utilizar las secuencias maximales para determinar a qué autor pertenece un texto anónimo. las secuencias maximales que mejor lo describirán serán las frases hechas, las maneras de conectar ideas, etc. por eso es interesante efectuar la búsqueda sobre el corpus completo, sin hacer limpieza de *stop words*.

Por otro lado, es muy interesante el uso de este algoritmo para la detección de plagio. Como hemos demostrado en los experimentos realizados, es sencillo establecer la similitud entre dos (o más) textos dados. Dependiendo de las características de los textos a analizar y del dominio del problema, se puede establecer un umbral de confianza, por debajo del cual no se considerará plagio.

A pesar de todo, este algoritmo tiene una limitación. Sólo encuentra una secuencia maximal si al menos en uno de los documentos aparece tal cual. Es decir, si se cambia una palabra por un sinónimo, no sería capaz de detectar la secuencia porque, pese a que sí que es capaz de dejar el gap entre las palabra, no hay un documento que contenga la secuencia tal cual

Explicacion Practica

Esto es, dada la secuencia: A B C 1 2 D, podría encontrar la secuencia ABCD si el umbral para la capacidad de salto es mayor o igual a 2.

Ejemplos:

-se estudió un corpus de revisiones de artículos de la Wikipedia, comparando de un mismo artículo, las diferentes revisiones que ha sufrido a lo largo del tiempo con la versión actual. Esto nos permite determinar qué cantidad de conocimiento incluido en cada una de las revisiones se conserva en el artículo actual

- A partir de un documento, se generaron una serie de documentos derivados del primero, incluyendo en cada uno de ellos diferentes palabras intercaladas

- ADN

-algoritmo de búsqueda de secuencias frecuentes maximales donde tienen en cuenta más información que la palabra en sí. En este caso particular, de cada palabra almacenan un vector de características, que serán lo que luego se comparará, haciendo la búsqueda más precisa. La información que almacenan en el vector de características de la palabra es, la palabra en sí, la raíz y la categoría gramatical *POS*

*-*Las secuencias maximales también se han propuesto (Coyotl-Morales, 2006) como una herramienta para la atribución de autoría cada autor tiene su propio estilo y hábitos de escritura

Ventajas

Detectar frases iguales aún en el caso de que se hubiesen insertado palabras nuevas dentro de la frase para intentar despistar a otros sistemas comparadores menos precisos.

Util para detectar patrones que se repiten en grandes coleeciones de datos

Detectar similitud entre textos, dando beneficios para la auditoria de textos.

destacar que la aplicabilidad de la extracción de secuencias frecuentes maximales es muy amplia puesto que la técnica es independiente del dominio del problema y del lenguaje en el que estén escritos los textos

Desventajas

Palabras:

\* Secuencias Maximales = secuencias máximas identicaas de palabras que se respiten en varios textos. Las secuencias frecuentes maximales consiste en:

-tener una secuencia de palabras (elementos) **p** tal que todos sus elementos aparecen en el mismo orden en otra secuencia llamada **q**, entonces p es subsecuencia de q

-la secuencia p es frecuente en S (colecciones de textos o documentos) si aparece en al menos **beta** documentos de S. beta es el umbral (base, limite minimo) de frecuencia minima

-solo se cuenta una repiticion por documento

-**p** es una **secuencia frecuente máxima** si no existe otra secuencia p’ tal que p es subsecuencia de p’ y p’ es frecuente en S

\* Patttern growing= patrones que se repiten en varios de los documentos.

**Pattern-growth (crecimiento de patrones)**

Los métodos de crecimiento de patrones son una propuesta más actual para los problemas de minería de patrones de secuencias. La idea principal es evitar la generación de candidatos, y restringir la búsqueda en una porción reducida de la base de datos inicial. En general, los métodos de crecimiento de patrones se pueden ver como un algoritmo de búsqueda en profundidad, ya que construyen cada patrón por separado y de un modo recursivo.

hacen crecer los patrones a partir de ítems frecuentes, de modo que son más eficientes. No obstante, es común que los métodos basados en *pattern-growth* proyecten los datos en conjuntos que se solapan, es decir, es muy probable que estos métodos estudien la misma parte de los datos varias veces.