Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba

Algoritmos para la búsqueda eficiente de instancias similares

Matthias Gallé[1]
Director: Dr. Gabriel Infante-López[2]

7 de mayo 2007

[1]galle@hal.famaf.unc.edu.ar
[2]gabriel@famaf.unc.edu.ar

Clasificación

(ACM Computing Classification System)

H. Information Systems
H.3 Information Storage and Retrieval
H.3.3 Information Search and Retrieval: Information filtering, Retrieval models, Search process

Resumen

En el presente trabajo encaramos el desafío de buscar objetos similares dentro de una colección muy grande de éstos objetos.

Encontramos dos dificultades en éste problema: en primer lugar, definir una medida de similitud entre dos objetos y luego implementar un algoritmo que – basándose en ésta medida – encuentre de una manera eficiente los objetos suficientemente parecidos.

La solución presentada utiliza una medida basada fuertemente en los conceptos de *precision* y *recall*, obteniéndose una medida similar a la de *Jaccard*. La eficiencia del algoritmo radica en la generación de grupos de objetos similares, y solamente después busca éstos objetos en la base de datos.

Usamos éste algoritmo en dos aplicaciones: por un lado a una base de datos de usuarios que evalúan películas a fin de proyectar éstas notas. Por otro lado, la utilizamos para encontrar perfiles genéticos que pueden haber aportado a una evidencia genética.

Palabras claves: information retrieval, data mining, índice de similitud, precision and recall, algoritmos de búsqueda, cotejo de ADN

A mis amigos y compañeros de la ABUA/CIEE y del FaMAF

Por la condición de **trabajo final** de las presentes hojas, le debo agradecimiento a:

- (por lo de **trabajo**):

  Al Ceprocor y a la Agencia Córdoba Ciencia por haber financiado la aplicación del algoritmo a perfiles genéticos.

  A Nicolás Wolovick y Miguel Pagano por haberse tragado los insultos cuando se hartaban de que les robaba fuerza de cómputo de sus máquinas. Ah, y por habérmelas prestado

  A Ale, Santi, Sergio y Germán por su generosidad en forma de mates, criollitos, jugos y pollos asados; además de ideas interesantes y hacerle someter a éste algoritmo a un testeo cruel.

  A los de la lista [acm] que leyeron mi pedido de ideas, y en particular a Diego por su respuesta.

  A mi familia y Cecilia que se tragaron mi mal humor, reclamo de silencio y menudo desasosiego.

  Sobre todas las cosas a Gabriel, no sólo por su honestidad, trabajo, guía y disponibilidad sino también por acompañarme en mis apuros finales.

- (por lo de **final**):

  A*l* FaMAF y a la UNC como institución: por la enseñanza gratuita y de calidad y por haberme permitido viajar y enseñar.

  A los profesores de computación del FaMAF: sí es posible unir calidez humano con excelencia académica.

  A Maria José, Pancho y el resto del equipo de Despacho de Alumnos: su ayuda y disponibilidad son la envidia de mis amigos universitarios.

  A Cecilia, por recordarme las cosas importantes.

  A mi familia: por incentivarme y permitirme estudiar; y por enseñarme cosas que son imposibles de aprender en un aula.

  A mis amigos y compañeros del FaMAF y de ABUA/CIEE: por las charlas nocturnas y diurnas, amistad, ayuda y compañía. Es sobre todo por ustedes que amo éste país y éste continente.

Soli Deo Gloria

Matthias Gallé
Córdoba – Argentina
Mayo 2007

Índice general

1. Introducción **1**
- 1.1. En éste trabajo: . . . 2
  - 1.1.1. Definición de similitud . . . 2
  - 1.1.2. Búsqueda de similares . . . 2
- 1.2. Esquema general . . . 3
- 1.3. Otras ideas . . . 3
  - 1.3.1. kD-Tree . . . 4

2. El problema: Búsqueda de parecidos **6**
- 2.1. Definiendo el problema . . . 6
  - 2.1.1. Definiciones . . . 6
  - 2.1.2. Descripción del problema . . . 7
  - 2.1.3. Una simplificación de notación . . . 7
- 2.2. ¿Qué significa ser parecido? . . . 8
  - 2.2.1. Noción intuitiva . . . 8
  - 2.2.2. Primeros intentos . . . 8
  - 2.2.3. Precision y Recall . . . 10
  - 2.2.4. F-Measure . . . 11
  - 2.2.5. Adaptación . . . 12
  - 2.2.6. Extensión de la medida a Individuos . . . 14
  - 2.2.7. Definiendo el problema: ahora sí . . . 15
- 2.3. Algoritmo . . . 16
  - 2.3.1. Idea general . . . 16
  - 2.3.2. Empezando por una posición . . . 16
  - 2.3.3. Descripciones . . . 17
  - 2.3.4. Grupos . . . 18
  - 2.3.5. Generación de números . . . 19
  - 2.3.6. Algoritmo . . . 21
  - 2.3.7. Complejidad . . . 21
- 2.4. Mejoras de implementación . . . 23
  - 2.4.1. Reducir acceso a la base de datos . . . 23

2.4.2. Un sólo acceso a la Base de Datos 24
2.4.3. Elegir la primera iteración 24
2.5. Cambiando el algoritmo 24
2.5.1. $\wedge$ y/o $\vee$ 24
2.5.2. Tratamiento diferencial de las posiciones 26

3. Aplicación: Netflix **28**
3.1. Netflix 28
3.1.1. Acerca de... 28
3.1.2. Cómo funciona 29
3.1.3. Algunos números de Netflix 29
3.1.4. No todo es color de rosa 30
3.1.5. Evaluación de películas 31
3.2. Neflix Prize 31
3.2.1. Definiendo el problema 31
3.2.2. Información provista 31
3.2.3. Objetivos 32
3.2.4. Premio 32
3.2.5. Estado Actual 32
3.3. Adaptando 33
3.3.1. Conjuntos no válidos 33

4. Aplicación: ADN-Matching **35**
4.1. Preliminares biológicos 35
4.1.1. El plan de la vida 36
4.1.2. Perfil genético 39
4.1.3. Bases genéticas forenses 42
4.2. Matematizando los perfiles 43
4.3. Adaptando 44

5. Conclusiones **46**
5.1. El Algoritmo: propiedades bonitas y otras no tantas 46
5.2. Resultados concretos 47
5.2.1. Netflix 47
5.2.2. ADN 50
5.3. Trabajo Futuro 51
5.3.1. Netflix 51
5.3.2. ADN 51

Capítulo 1

Introducción

Uno de los mayores desafíos que tienen que afrontar las personas que manejan datos, y las ciencias que los auxilian en ésta tarea, es la posibilidad ya no de juntar estos datos sino la de extraer información (e información útil) de manera automática (sin intervención humana) de manera rápida y organizarla en un formato útil para el usuario o próximo programa.

El problema de extraer información se vio incrementado con la aparición de la computación y la generación exponencial de nuevos datos (que no necesariamente contienen información), la posibilidad de guardarlos – extendida por los avances del hardware tanto en capacidad de almacenamiento como de procesamiento (ver [Moo65]) – y disponibilizarlos a una gran cantidad de usuarios.

Pero las ciencias de computación también trajeron mucha ayuda y herramientas para facilitar, acelerar y organizar la montaña de información con la que actualmente se encuentran investigadores de índole diverso[1] y ramas enteras de ésta ciencia se están dedicando a facilitar una mejor comprensión y proveer enfoques y herramientas más adecuadas.

Uno de los problemas mas comunes en el minado de información es el de encontrar objetos similares. En estos problemas se contiene una gran base de dato de objetos (que pueden ser, personas, problemas, descripción de objetos físicos en general, etc), y dado un objeto nuevo, queremos encontrar en la base de datos todos los objetos que se le parezcan de alguna manera. Asociado a ésto se encuentran dos problemas: en primer lugar es necesario definir alguna medida de similitud para poder medir qué significa que un objeto se le parezca a otro. Luego, el tamaño de la base de datos hace necesario generalmente algún algoritmo que – basada en la medida de similitud definida anteriormente – pueda encontrar los elementos parecidos de manera eficiente.

[1] por ejemplo, actualmente más de 50 Gigabytes de secuencias brutas de ADN se encuentran disponibles y se calcula que en los próximos diez año éste cifra se multiplique por 1000

En el siguiente trabajo nos enfocamos en éste problema de encontrar individuos parecida a cierto dado dentro de una base de datos grande de información. Ésta información está estructurado de una manera específica, pero que es lo suficientemente genérico como para aplicarla a casos reales de índole muy diversos: la forma que tomarán los **`individuos`** será un arreglo de conjuntos de elementos.

Para ello, primero tuvimos que desarrollar una métrica para definir qué entendemos bajo similitud o cercanía entre dos individuos. Una vez resuelto ésto, usamos ésta métrica para diseñar un algoritmo que retorna todos los individuos suficientemente cerca, de acuerdo a un *threshold*. Dado que el conjunto donde buscaremos éstos individuos será muy grande, enfrentamos y solucionamos la dificultad computacional a éste cálculo.

Mostraremos dos posibles aplicaciones para los cuales el algoritmo fue implementando y aplicado, y presentaremos los resultados en ambos casos.

1.1. En éste trabajo:

1.1.1. Definición de similitud

En nuestro trabajo los individuos se representan como colecciones de conjuntos. Estos conjuntos recogen características relevantes de los individuos. Por ello, buscamos alguna medida que refleje la cercanía de individuos en función de la cantidad de elementos comunes y no-comunes. La dificultad de encontrar alguna métrica que expresa ésto radica justamente en el hecho de que en nuestra noción intuitiva de similitud influyen tanto los elementos en comunes, como aquellos exclusivos de los dos conjuntos en cuestión

Encontramos ésta medida basándonos en unos índices utilizados en Information-retrieval llamados precision, recall y f–measure y los adaptamos para nuestro uso, convirtiéndolos en un porcentaje de inclusión. Juntos, revelan características parecidas a las descriptas por el índice de Jaccard[2]. Éstos tres índices en conjunto sin embargo, revelan más información y le dan mayor versatilidad al algoritmo: nos resultan útiles a la hora de querer adaptar el algoritmo para aplicaciones particulares.

1.1.2. Búsqueda de similares

La manera trivial de encontrar los elementos más parecidos dado una medida que determine la cercanía claramente es recorrer toda la base de datos y recuperar

[2]Dado dos conjuntos A y B, el índice de Jaccard es un índice que revela la similitud entre éstos conjuntos: $Jac(A, B) \triangleq \dfrac{||A \cap B||}{||A \cup B||}$

aquellos que superen cierto límite. Sin embargo, a partir de cierto tamaño éste enfoque es inviable, pues es necesario tener que recorrer toda la base en busca de un eventual pequeño conjunto resultante de elemenos parecidos.

Nuestro enfoque consiste en obtener primero grupos de individuos teóricos (o sea, no existentes necesariamente en la base) que sean lo cerca suficiente y luego instanciar éstos grupos de acuerdo a una descripción general y la instancia particular del individuo que estamos buscando. La idea de "cerca suficiente" será definida utilizando un parámetro de *threshold*: aquellos individuos cuyo índice de similitud definido anteriormente supera éste *threshold* serán considerados "cerca suficiente".

1.2. Esquema general

En el capítulo 2 presentamos el problema de manera formal, definimos los elementos con los cuales trabajaremos y presentamos tanto la métrica usada como el algoritmo que - a base de ésta métrica - devuelve el conjunto de individuos más próximos.

En los siguientes dos capítulos se mostrarán dos aplicaciones: en el capítulo 3 se trata de encontrar usuarios parecidos en una base de datos de preferencias de películas para pronosticar cuáles películas serán del agrado de un usuario dado. Para el capítulo 4 cambiamos de ambiente y entramos en el campo de la Bioinformática: aplicamos el algoritmo y la métrica para encontrar perfiles de ADN parecidos a aquellos encontrados en una escena de crimen. Primero explicaremos algunos conceptos de biología molecular para luego ver cómo convertir éstos perfiles a individuos a los cuales se puede aplicar no sólo la métrica de similitud, sino también el algoritmo en su totalidad.

En las conclusiones daremos un resumen de las cualidades del algoritmo, explicaremos los resultados obtenidos,y comentaremos trabajos futuros posibles.

1.3. Otras ideas

Hemos realizado diferentes aproximaciones al problema, para finalmente quedarnos con la que es explicada en éste trabajo. Sin embargo, nos pareció pertinente detallar aunque sea uno de éstas implementaciones dejadas de lado, no sólo porque sí guardan algunas ideas interesantes, sino también porque explican un poco el proceso que desembocó en la solución actual.

Una de los primeros intentos fue atacar el problema organizando los individuos de alguna manera inteligente para luego acelerar las búsquedas.

Figura 1.1: Ejemplo de 3D-tree con 10 elementos

1.3.1. kD-Tree

Un árbol kD es una estructura de datos para almacenar un conjunto finito de un espacio *k*-dimensional. Fue examinado en detalle por J. Bentley ([Ben80]), y usado en information-retrieval o redes neuronales (véase por ejemplo [Moo91]).

Un árbol kD-Tree es un árbol de decisión binario en cuyas hojas se encuentran conjuntos de individuos, mientras que en cada uno de sus nodos internos se toma la decisión de elegir una de las k dimensiones (digamos, *i*) y un individuo de la base de datos. El resto de los elementos se dividirá de acuerdo a lo que tengan en su *i*-ésima posición en relación al elemento de la *i*-ésima posición del individuo elegido. Así, si trabajamos con números enteros, se los divide si son menores; o mayores o iguales que el discriminante.

Ejemplo 1.1 *En la figura 1.3.1 se detalló un ejemplo de un 3D-Tree. En el primer nodo se discrimina por la segundo dimensión. Así, todos los individuo del subárbol izquierda tendrán elementos menores que 5 en su segunda dimensión.*

Nótese que en la tercera hoja se encuentran los elementos cuyo segundo elemento está entre 5 y 13. En cambio, no existe ningún individuo cuyo segundo elemento es mayor a 13, por lo que la última hoja queda vacía

No necesariamente el árbol resultante tiene que estar balanceado.

Un primer acercamiento nuestro fue utilizar un versión muy simplificada de

un kD-tree, que redundó en un árbol de decisión binario simple. Construimos un 1D-tree para cada posición i, que tiene tanto niveles como elementos posibles. En cada nodo de profundidad j se decide como partir el conjunto de individuos actual de acuerdo a la pertenencia o no del elemento j-ésimo en el conjunto de la posición i.

Por lo tanto, en las hojas quedan los individuo que contienen idéntico conjunto en ésa posición i.

Este enfoque fue dejado de lado pues era ideal para obtener una coincidencia exacta: lo que sin embargo deseábamos encontrar era una intersección no necesariamente completa pero sí suficientemente grande. Rescatamos a pesar de ello la idea de enumerar las hojas de 0 a $2^n - 1$: he aquí que ahora se tiene un identificador único de los posibles conjuntos, que es el *bitmap* asociado a ése conjunto si se toma al identificador en base 2.

Capítulo 2

El problema: Búsqueda de parecidos

El problema planteado es el de la búsqueda de parecidos. En general, tendremos un individuo como entrada y queremos buscar en una base de datos los individuos más parecidos a éstos. Nos abstraeremos de la base de dato y supondremos funciones particulares de acceso a ella en caso de necesitarla.

Los individuos con los que trabajaremos serán arreglos que en cada posición tendrán un conjunto de elementos.

En primera instancia formalizaremos éste problema y los elementos con los cuales trabajamos. En la sección 2.2 explicaremos detalladamente la solución encontrada al problema de medir la similitud o distancia entre dos individuos, para luego, en la sección 2.3 mostrar los problemas que tuvimos que encarar a la hora de hacer viable el cómputo de la búsqueda de individuos más próximos. Mostraremos una solución en la sección 2.3.2. Dicha solución se basa no en buscar directamente sobre los individuos existentes en la base de datos, sino en encontrar descripciones genéricas de soluciones ideales, generar las instancias que cumplen éstas descripciones y luego verificar si éstas existen en nuestra base de datos.

2.1. Definiendo el problema

Como hemos dicho, trabajaremos con individuos, que serán arreglos de conjunto de elementos. Formalizaremos éstas ideas con las siguientes

2.1.1. Definiciones

Definición 2.1
Dado $N \geq 0$ *los números* $1, 2 \ldots N$ *se llamarán* ***posiciones***.
Posiciones $\triangleq \{1 \ldots N\}$

Definición 2.2
Sean $\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_N$ conjuntos finitos, cuyos miembros llamaremos genéricamente **`elementos`**

$$\textbf{\texttt{Elementos}} \triangleq \bigcup_{i=1}^{N} \mathcal{A}_i$$

Debido a que trabajaremos diferentes aplicaciones - y creemos que se puede generalizar y aplicar a otros casos también - definimos la noción de escenario:

Definición 2.3 *Un* **`Escenario`** *será una tupla compuesta por:*

- *un número Natural* $\boldsymbol{N}$
- *conjuntos* $\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_N$

Definición 2.4 *Dado un Escenario, un* **`individuo`***, será una función*

$$f : Posiciones \rightarrow \mathbb{P}(Elementos)$$

de tal manera que $f(i) \subseteq \mathcal{A}_i$

También se lo representará como un arreglo $[A_1, A_2 \ldots A_N]$, donde $f(i) = A_i$

Así, a la hora de querer aplicar el algoritmo a un caso particular, será necesario definir cuál es el escenario con el que se trabajará y qué representan los individuos y las posiciones para dicho caso.

2.1.2. Descripción del problema

Dado un conjunto de *individuos*, y un *individuo* particular que se llamará **`buscado`**, el problema radica en encontrar los datos "más parecidos" de acuerdo a un *threshold* que definirá a partir de qué momento un individuo de la base se puede considerar cercano al *buscado*.

El primer desafío a resolver es por lo tanto definir alguna métrica, orden, distancia o índice para saber, dado dos individuos cuál es el más cercano al *buscado*.

2.1.3. Una simplificación de notación

Dado que los $\mathcal{A}_i$ son finitos, y haciendo un primer esfuerzo por abstraernos de la naturaleza de los elementos veremos a los conjuntos A_i como un número binario, donde el n-ésimo bit será uno si A_i contiene el i-ésimo elemento de $\mathcal{A}_i$. Ésto también es conocido como tener un *bitmap* de un conjunto, pues mapearemos los elementos a un conjunto en bits.

Claramente, eso implica tener un orden subyacente de los $\mathcal{A}_i$: dado que este orden no tendrá otra utilidad que ésta simplificación, puede ser uno cualquiera y arbitrario.

Ejemplo 2.1 *Sea* $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ *y* $<$ *un orden sobre éste conjunto de tal manera que:* $a_2 < a_5 < a_4 < a_3 < a_6 < a_1$

Entonces, el conjunto $A = \{a_3, a_4, a_5\}$ *se representará como:*

a_2	a_5	a_4	a_3	a_6	a_1
0	*1*	*1*	*1*	*0*	*0*

2.2. ¿Qué significa ser parecido?

2.2.1. Noción intuitiva

Claramente, la distancia entre dos individuos depende de una noción de distancia entre los conjuntos que tienen en cada uno de sus posiciones. Dos conjuntos son lo más parecido posibles cuando tienen exactamente los mismos elementos. El extremo opuesto sin embargo no es aquél caso en el que los dos conjuntos no comparten ningún elemento:

x (*buscado*)	y	z
$\{a_1, a_3\}$	$\{a_2\}$	$\{a_2, a_4, a_5\}$

El conjunto z posee más datos de "sobra" que el conjunto y, por lo que debería encontrarse más lejos.

Así mismo,

x (*buscado*)	y	z
$\{a_1, a_3\}$	$\{a_3, a_6\}$	$\{a_3, a_4, a_5\}$

aquí también el conjunto y debería estar más cerca de x pues posee menos elementos exclusivos.

Exploraremos ésta idea más detalladamente en los primeros acercamientos al problema, pero todos ellos trabajan sobre la idea de que no alcanza solamente en tener en cuenta los elementos en común, sino también aquellos exclusivos

2.2.2. Primeros intentos

Intersección

Al estar trabajando con arreglo de conjuntos, una aproximación naïv podría ser querer maximizar simplemente la intersección de éstos conjuntos. Sin embargo, ejemplificaremos porque ésta aproximación se queda corta:

Ejemplo 2.2

x (buscado)	*y*	*x & y*
0101	*1100*	*0100*
0101	*1110*	*0100*

donde & es el y binario (que se aplica bit a bit) que tiene la propiedad de ser la operación de intersección si se ve - como lo hacemos nosotros - los operandos como bitmaps de conjuntos.

Notamos que aunque la intersección da el mismo resultado en ambos casos, el segundo conjunto tiene más elementos "de sobra" que el primero, por lo que debería encontrarse más lejos.

Al tomar solamente la intersección, estamos perdiendo demasiada información acerca de los conjuntos originales y los elementos que éstos poseen.

Distancia de Hamming

Al trabajar con distancia entre números binarios, el problema de encontrar una métrica entre dos individuos parece reducirse a encontrar distancias entre dos números binarios. Una solución que parece viable es aplicar la distancia de Hamming, introducido por Richard Hamming en 1950 ([Ham50]), que tiene la útil propiedad de efectivamente ser una distancia (cumple las propiedades de no-negatividad, simetría, identidad y desigualdad triangular).

La distancia de Hamming entre dos números binarios x y y se define como la cantidad de 1's del *o exclusivo*:

$$Hamming(x, y) \triangleq ||x \oplus y||$$

donde ||_||, es el **`bitCount`**: aplicado a un número binario devuelve la cantidad de bits prendidos que tiene ese número binario.

Sin embargo, seguimos perdiendo información demasiada importante al aplicar la distancia de Hamming. Veamos el siguiente ejemplo:

Ejemplo 2.3

x (buscado)	*y*	*Hamming(x,y)*
0101	*1101*	*1*
0101	*0001*	*1*

Sin embargo, intuitivamente la distancia de la segunda fila, con respecto al mismo *buscado* debería ser menor. El elemento que tiene de más el primer *y* no debería pesar tanto como el elemento de menos del segundo.

Aunque nos acercamos un poco, seguimos sin captar completamente la noción de elementos originales de un conjunto, y de cuántos de éstos elementos **no** son compartidas con el *buscado*.

2.2.3. Precision y Recall

precision y recall son dos medidas ampliamente usado en el área de Information Retrieval para evaluar el resultado de una búsqueda[1].

Éstas medidas se basan sobre un conjunto de documentos y una búsqueda para la cual la relevancia de los documentos es conocida (suponiendo que un documento o es completamente relevante o no lo es). Expresan la noción de qué tan completa ha sido la búsqueda (cuántos documentos relevantes ha retornado), y qué tan precisa ha sida (cuántos irrelevantes ha retornado).

En éstos términos, se definen como:

$$Recall \triangleq \frac{||\{documentos\ relevantes\} \bigcap \{documentos\ encontrados\}||}{\{documentos\ relevantes\}} \tag{2.1}$$

$$Precision \triangleq \frac{||\{documentos\ irrelevantes\} \bigcap \{documentos\ encontrados\}||}{\{documentos\ irrelevantes\}} \tag{2.2}$$

Ejemplo 2.4 *Supóngase que dado una base de datos se realiza una búsqueda sobre ella, obteniéndose los siguientes resultados*

De ahí se obtiene:

[1]para una descripción más detallada ver `http://de.wikipedia.org/w/index.php?title=Recall_und_Precision&oldid=28993005` (alemán)

- *recall*

  A: cantidad de documentos relevantes recuperados

  B: cantidad de documentos relevantes no recuperados

  $\mathsf{recall} = A/(A + B)$

- *precision*

  A: cantidad de documentos relevantes recuperados

  C: cantidad de documentos irrelevantes recuperados

  $\mathsf{precision} = A/(A + C)$

Claramente, tanto precision como recall siempre están comprendidos entre 0 y 1, indicando el porcentaje de la medida correspondiente (1 corresponde a un 100 % y 0 a un 0 %).

Generalmente, se recomienda usar ambas medidas de manera paralela, pues revelan propiedades distintas. De hecho, cuando se los usa en information-retrieval manifiestan una correlación negativa entre ellos; pues al ampliar los criterios de búsqueda la precisión (precision) disminuye mientras que la completitud (recall) aumenta. En cambio, al ser más exigente (restrictivo) se pone en evidencia un comportamiento contrario[2].

En un caso extremo, tenemos una búsqueda que devuelve todos los documentos: tal búsqueda tiene un recall de 1, pero un precision de $\frac{1}{T}$, donde T es el tamaño de la base de datos.

2.2.4. F-Measure

Por esta correlación negativa, y para cuando se quiera usar solamente una medida que englobe ambas indicadores anteriores, se usa la media armónica entre los dos indicadores:

$$F \triangleq \frac{2}{\frac{1}{\mathsf{precision}} + \frac{1}{\mathsf{recall}}} = 2 * \frac{\mathsf{precision} * \mathsf{recall}}{\mathsf{precision} + \mathsf{recall}} \tag{2.3}$$

Definición 2.5 *La medida armónica entre* precision *(Ecuación 2.2) y* recall *(Ecuación 2.1) se denomina* ***f–measure***

[2]una explicación detallada se puede encontrar en [WF05]

2.2.5. Adaptación

Para nuestro objetivo, los términos precision y recall pueden llevar a confusión ya que ni se trabaja con documentos ni éstos son (ir)relevantes. Redefiniéndolos levemente sin embargo, sirven asombrosamente a nuestro objetivo, pues expresan una relación de los elementos comunes y exclusivos, ignorando la naturaleza propia de éstos elementos.

Para ello, usamos a uno de los conjuntos como *documentos recuperados* y el otro como *documentos irrelevantes* en la definición de precision (Ecuación 2.2), y al primero de éstos como *documentos relevantes* y el segundo como *documentos recuperados* en la definición de recall (Ecuación 2.1).

Aplicando además la abstracción introducida en la sección 2.1.3, y usando las ecuaciones 2.2 y 2.1 obtenemos:

$$P_A(a,b) \triangleq \frac{||a \;\&\; b||}{||a||} \tag{2.4}$$

$$P_B(a,b) \triangleq \frac{||a \;\&\; b||}{||b||} \tag{2.5}$$

dónde ||_|| aplicado a un conjunto, se llama la **`cardinalidad`** de un conjunto. El usar el mismo símbolo para la cardinalidad y el bitCount no es casualidad. Al ver los números binarios como conjuntos, el bitCount (cantidad de bits prendidos) es justamente la cardinalidad (cantidad de elementos) del conjunto asociado.

Podemos interpretar a P_A como "cuánto de A está en B" y a P_B como "cuánto de B está en A".

Veamos ahora la forma de f–measure, combinando la definición de 2.3 con las ecuaciones 2.4 y 2.5, :

Lema 2.1 $F(a,b) = 2 * \frac{||a \;\&\; b||}{||a|| * ||b||}$

Demostracion

$$
\begin{aligned}
& F(a,b) \\
= \quad & 2 * \frac{P_A(a,b) * P_B(a,b)}{P_A(a,b) + P_B(a,b)} \\
= \quad & 2 * \frac{\frac{||a \,\&\, b||}{||a||} * \frac{||a \,\&\, b||}{||b||}}{\frac{||a \,\&\, b||}{||a||} + \frac{||a \,\&\, b||}{||b||}} \\
= \quad & 2 * \frac{\frac{\cancel{||a \,\&\, b||} * ||a \,\&\, b||}{\cancel{||b||} * \cancel{||a||}}}{\frac{\cancel{||a \,\&\, b||} * (||a|| + ||b||)}{\cancel{||a||} * \cancel{||b||}}} \\
= \quad & 2 * \frac{||a \,\&\, b||}{||a|| * ||b||}
\end{aligned}
\tag{2.6}
$$

∎

Además,

Lema 2.2 $F(a,b) = F(b,a)$

Demostracion

$$
\begin{aligned}
& F(a,b) & \\
= \quad & 2 * \frac{||a \,\&\, b||}{||a|| * ||b||} & (\textit{Lema anterior}) \\
= \quad & 2 * \frac{||b \,\&\, a||}{||b|| * ||a||} & (\textit{conmutatividad del } \&, *) \\
= \quad & F(b,a) & (\textit{Lema anterior})
\end{aligned}
$$

∎

Así, volviendo al ejemplo 2.3:

Ejemplo 2.5

x	y	*Hamming(x,y)*	$P_A(x,y)$	$P_B(x,y)$	$F(x,y)$
0101	*1101*	*1*	*1*	*0.66*	*0.8*
0101	*0001*	*1*	*0.5*	*1*	*0.75*

Notamos que ahora sí $F(x,y_1) > F(x,y_2)$, a pesar de que $P_B(x,y_1) < P_B(x,y_2)$, lo que coincide con la noción de que y_2 está "más contenido" en x que y_1.

Definición 2.6 `Medidas` $\triangleq \{F, P_A, P_B\}$*, dónde las funciones de* `Medidas` *son de tipo* $\mathbb{B} \times \mathbb{B} \to \mathbb{R}$*, dónde* $\mathbb{B}$ *es el conjunto de números en base 2.*

2.2.6. Extensión de la medida a Individuos

Hemos resuelto pues el problema conceptual de definir una medida entre dos conjuntos de elementos. Las tres medidas son igualmente aplicables, pero preferiremos la $f - measure$, por balancear armónicamente las otras dos medidas.

Una vez definido cómo mediremos las distancias entre dos conjuntos de elementos extenderlo a una medida entre individuos es sencilla. Existen, sin embargo varias maneras de hacerlo y dependiendo de la aplicación concreta preferiremos una que otra: para ellos sobrecargaremos el nombre de las funciones de índice.

Sea $\chi \in$ `Medidas` , d_1, d_2 dos individuos:

Definición 2.7

$$\chi_0(d_1, d_2) \triangleq \frac{\sum_{\substack{i:=1 \\ i \in dom(d1) \cap dom(d2)}}^{N} \chi(d_1(i), d_2(i))}{N}$$

Definición 2.8

$$\chi_1(d_1, d_2) \triangleq \frac{\sum_{\substack{i:=1 \\ i \in dom(d1) \cap dom(d2)}}^{N} \chi(d_1(i), d_2(i))}{||\boldsymbol{dom}(d_1) \cap \boldsymbol{dom}(d_2)||}$$

Definición 2.9

$$\chi_A(d_1, d_2) \triangleq \frac{\sum_{\substack{i:=1 \\ i \in dom(d1) \cap dom(d2)}}^{N} \chi(d_1(i), d_2)(i)}{||\boldsymbol{dom}(d_1)||}$$

$$\chi_B(d_1, d_2) \triangleq \frac{\sum_{\substack{i:=1 \\ i \in dom(d1) \cap dom(d2)}}^{N} \chi(d_1(i), d_2(i))}{||\boldsymbol{dom}(d_2)||}$$

Notamos que $\chi_0(d_1, d_2) = \chi_0(d_2, d_2)$. Sin embargo, $\chi_X(d_1, d_2) \neq \chi_X(d_2, d_2)$, $(X = A, B)$ pues

$\chi_A(d_1, d_2) < \chi_A(d_2, d_2) \Leftrightarrow ||\mathbf{dom}(d1)|| > ||\mathbf{dom}(d2)||$

Definición 2.10

$$\chi_{max}(d_1, d_2) \triangleq max_{i \in dom(d1) \cap dom(d2)}\, \chi(d_1(i), d_2(i))$$

Definición 2.11

$$\chi_{min}(d_1, d_2) \triangleq min_{i \in dom(d1) \cap dom(d2)}\, \chi(d_1(i), d_2(i))$$

2.2.7. Definiendo el problema: ahora sí

Ahora que finalmente tenemos una medida de similitud entre dos individuos, podemos definir el problema en término de los conceptos vistos.

Pero antes agregamos nuestro último elemento. Un *threshold* será un límite que limitará hasta qué similitud buscaremos los individuos cercanos. Así, si usamos la medida definida en la Definición 2.8, un *threshold* de 0 significa que será devuelta toda la base de datos; en cambio, con un *threshold* de 1 usando la medida F, será devuelto solamente ése mismo conjunto.

Definamos ahora nuestro problema que hasta ahora solamente fue planteada de manera intuitiva.

Definición 2.12 *Dado*

- *un conjunto de individuos* D*,*
- *un individuo* $\mathsf{buscado}$*,*
- *un número* $\mathsf{threshold} \in \mathbb{R}$*,* $threshold > 0$*,*

encontrar los $\mathsf{d} \in \mathsf{D}$ *tal que* $\chi_*(\mathsf{buscado}, \mathsf{d}) \geq \mathsf{threshold}$*, donde* $\chi \in$ `Medidas`*, y el* $*$ *puede ser reemplazado por alguna de las definiciones vistas anteriormente.*

2.3. Algoritmo

2.3.1. Idea general

Una vez definido qué entendemos por distancias entre dos individuos, estamos en condiciones de poder ordenar un conjunto de individuos en función del *buscado*. Sin embargo, es computacionalmente pesado (y a partir de cierto tamaño de conjunto hasta inviable) hacerlo de la manera trivial; o sea, comparar uno por uno por los individuos del conjunto y luego ordenarlos de acuerdo a la métrica obtenida.

Recapitulemos acerca del problema planteado: dado un número binario queremos obtener los individuos de una base de datos que están lo suficiente cerca (de acuerdo al *threshold*) usando un índice (P_A, P_B, F). El enfoque trivial radica en calcular este índice para todos los números binarios de la base y luego devolver aquellos que superen el *threshold*.

Sin embargo, dado que sólo estamos interesados en los *más* próximos (dónde "más próximos" son aquellos que superan el *threshold* de la definición 2.12), cambiamos el enfoque: buscamos primero cuáles serían los hipotéticos más cercanos y luego chequeamos que efectivamente se encuentren en la base. De esta manera, todos los individuos que están demasiados lejos ni siquiera pasarán por nuestras manos.

Para ello, generamos primero descripciones que generalizan a cierto grupo de números binarios, de tal manera que para los integrantes de un mismo grupo, el resultado de la medida χ utilizada es el mismo.

En ésta sección explicaremos el algoritmo en sí, daremos las justificaciones del enfoque elegido y formalizaremos los conceptos en juego. Al final comentaremos brevemente la complejidad del algoritmo y luego en la próxima sección mostraremos brevemente algunas mejoras a la hora de implementarlo.

En la sección final de éste capítulo (2.5) detallamos algunas posibles modificaciones al algoritmo para usarlo en escenarios específicos. Todas ésas modificaciones se pueden combinar y en conjunto le otorgan una versatilidad y utilidad interesante a la solución propuesta.

2.3.2. Empezando por una posición

Suponiendo que los conjuntos de las diferentes posiciones de los individuos son independientes, no hace falta que busquemos en todas los eventuales valores que pueden tomar los individuos ($\prod_{i:=1}^{N} 2^{\|\mathcal{A}_i\|}$)[3], sino alcanza con buscar parecidos

[3] $2^{\|B\|}$ es la cantidad total de subconjuntos del conjunto B

para cada posición y luego unirlos de alguna manera. Concentraremos luego nuestro análisis en un sólo conjunto de elemenos, y al final de ésta sección (sección 2.3.6) mostraremos el algoritmo final que une éstas búsquedas.

2.3.3. Descripciones

En vez de fijarnos qué número binario es el más próximo al que estamos buscando y luego el próximo hasta que el *threshold* no se vea satisfecho agrupamos a varios números binarios en grupos que tienen el mismo índice.

Explicaremos primero qué es una descripción, luego demostraremos que todos los números dentro de un grupo tienen el mismo índice. Ésto justifica la decisión de ordenar los grupos según el índice usado, pues éste orden respeta el orden del índice de cada uno de los integrantes de los grupos.

Una descripción de un número binario de longitud m con k bits prendidos está constituido por dos números:

- l: cantidad de bits en común ($l \leq k$)
- n: cantidad de bits exclusivos ($n \leq m - k$)

Los integrantes de éste grupo son los números binarios que tienen l bits prendidos en algunas de las k posiciones del número de referencia; y otros n bits de entre los bits apagados del número de referencia.

A no ser que lo explicitemos de otra manera, las letras k, l, n y m siempre tendrán el significado aquí explicado.

Ejemplo 2.6 *Sea* $x = 011001$. *Por lo tanto,* $m = 6, k = 3$.
Los integrantes del grupo con descripción $l = 2, m = 2$ *son:*

1. 011001	*2. 011110*	*3. 111010*
4. 010111	*5. 110011*	*6. 110101*
7. 001111	*8. 111011*	*9. 101101*

Es claro, que el *bitCount* de los integrantes de un grupo con descripción $\langle l, n \rangle$ es $l + n$.

Antes de seguir...

... un pequeño adelanto: Como éstos números binarios serán los identificadores de los conjuntos que los individuos tienen en cada posición, la longitud m es fija para cada posición. Omitiremos por lo tanto éste parámetro: en el momento de implementar el algoritmo se pueden guardar éstos números estáticamente.

2.3.4. Grupos

Cada descripción define un grupo de números. Veremos que para dos números del mismo grupo, cualquiera de las medidas vistas (P_A, P_B, F) es la misma.

O sea:

Definición 2.13 *Dado x llamaremos* **`descripción`** *a la terna $\langle ||y||, ||x \,\&\, y|| \rangle$ y lo denotaremos* $\mathtt{desc}_x(y)$

Entonces:

Lema 2.3 *Dado números binarios x, y, z,*
$desc_x(y) = desc_x(z) \Rightarrow \chi(x,y) = \chi(x,z)$, *donde* $\chi \in$ `Medidas`

Demostracion

$$
\begin{aligned}
& desc_x(y) = desc_x(z) \\
\Leftrightarrow \quad & \langle ||y||, ||x \,\&\, y|| \rangle = \langle ||z||, ||x \,\&\, z|| \rangle && (def\ de\ desc) \\
\Leftrightarrow \quad & ||y|| = ||z|| \wedge ||x \,\&\, y|| = ||x \,\&\, z|| && (tuplas) \\
\Rightarrow \quad & \frac{||x \,\&\, y||}{||y||} = \frac{||x \,\&\, z||}{||z||} \\
\Leftrightarrow \quad & P_B(x,y) = P_B(x,z) && (def\ de\ P_B)
\end{aligned}
$$

Las demostraciones para P_A y F son equivalentes. ∎

Por lo tanto, al agrupar los números binario de acuerdo a sus descripciones se preserva el orden dado por la medida usada.

Generación estática

En nuestro intento de encontrar una lista ordenado de números cercanos a uno cierto dado, hemos encontrado hasta ahora la manera de agruparlos de acuerdo a una descripción.

Ahora bien, el orden inter-grupo depende (por la naturaleza de las medidas usadas) solamente de la **cantidad** de bits en común y exclusivos respectivos, no de la posición de éstos bits. Ésta información (la ubicación de los bits) se vuelve relevante recién a la hora de instanciar un grupo en particular.

Por lo tanto, para cada k, la lista de descripciones $\langle l, n \rangle$ es siempre igual. La excepción es dada por aquellas descripciones que son imposibles de cumplir (o sea, cuando $n + l > m$ o $n > m - k$): podemos ver éstos grupos como grupos vacíos.

Dada la universalidad de las listas de descripciones, éstas se pueden generar una sola vez, tomando un m suficientemente grande (o sea, que sea mayor o igual que los $||\mathcal{A}_i||$). Luego, a la hora de buscar los números cercanos al *buscado*, instanciamos éstos grupos, generando todos los números que cumplen la descripción.

Resumiendo

Los pasos a seguir por lo tanto son:

1. generamos todas las descripciones factibles: o sea, aquellas en las que $l \leq k$
2. ordenamos éstas descripciones de acuerdo a χ
3. generamos los números binarios a partir de éstas descripciones, tomando como grupos vacíos aquellos que son imposibles de satisfacer.

Los primeros dos pasos se pueden realizar de manera estática ya que son siempre iguales para un mismo escenario.

2.3.5. Generación de números

Planteamos el problema de generar todos los números binarios y que cumplen una descripción dada las cantidades de bits prendidas de x (número cuyos cercanos queremos encontrar).

El conjunto final tendrá un tamaño de $\binom{k}{l} * \binom{m-k}{n}$. El primer factor indica la cantidad de posibilidades de tener bits prendidos compartidas con x, mientras que el segundo factor son los bits prendidos **no** compartidos con x.

El algoritmo 1 resuelve ésta generación de números:

- Para recorrer todas las permutaciones posibles, suponemos la existencia de funciones que las calculan dado un arreglo que simboliza el subconjunto actual. Entonces, para calcular todos los $\binom{k}{l}$ conjuntos posibles, usaremos un arreglo de longitud l que contendrá números del 1 al k sin repetir indicando los elementos que contendrá el subconjunto en cuestión. En el algoritmo, los arreglos *bits_prendidos* y *otros_bits_prendidos* mantendrán las permutaciones respectivas de los bits de y compartidas con x y de los bits exclusivos.

- $generarEntero_x$ (algoritmo 2: línea 13 en algoritmo 1) supone un arreglo $pos\_de\_bits_x$ cuyos primeros l elementos son las posiciones de los bits prendidos de x, mientras que las restantes son las posiciones de los bits apagados.

Algoritmo 1 *generar_enteros*: Dado x, con $k = ||x||$, l y n genera todos los número que cumplen la descripción $desc_x(l, n)$

Require: $l \leq k \wedge n \leq m - k$

```
1.  bits_prendidos: Array of ℤ(l)
2.  otros_bits_prendidos: Array of ℤ(n)
3.  resultado : 𝒫(𝔹)
4.  agregar :=∅
5.  for i :=to l do
6.          bits_prendidos[i] :=i
7.  end for
8.  for i :=1 to n do
9.          otros_bits_prendidos[i] :=i
10. end for
11. while tiene_más_permutaciones(otros_bits_prendidos, m − k) do
12.         while tiene_más_permutaciones(bits_prendidos, k)) do
13.                 resultado.agregar(generarEntero_x(bits_prendidos,
                    otros_bits_prendidos) )
14.                 próxima_permutación(bits_prendidos, k)
15.         end while
16.         próxima_permutación(otros_bits_prendidos, m − k)
17.         reset(bits_prendidos)
18. end while
19. return resultado
```

Algoritmo 2 $generarEntero_x$: Toma dos arreglos de permutaciones: bits_prendidos y otros_bits_prendidos que hacen referencia a la posición de los bits prendidos y apagados de x y genera un número binario de acuerdo a éstas permutaciones

```
número : 𝔹
número :=0
for all pos ∈ bits_prendidos do
        número ↑ pos_de_bits_x[pos]
end for
for all pos ∈ otros_bits_prendidos do
        número ↑ pos_de_bits_x[pos+k]
end for
return número
```

donde $\_ \uparrow \_ :: \mathbb{B} \times \mathbb{Z} \rightarrow \mathbb{B}$ cumple:
$b \uparrow i = b$ con el i-ésimo bit seteado

2.3.6. Algoritmo

Ahora sí, estamos en condiciones de dar un algoritmo que dado un individuo busca en la base de datos (usando funciones externas) los individuos suficientemete cercanos de acuerdo a un *threshold*: detallamos los pasos a seguir en el algoritmo 3, y explicamos algunas líneas del mismo:

1. línea 1: genera estáticamente todas las posibles descripciones y las ordena. Como ya dijimos, no es necesario hacerlo todas las veces que se llama el algoritmo, sino solamente la primera vez.

2. línea 4: buscamos todos los parecidos de una posición y tomamos luego simplemente la intersección de todos éstos conjuntos.

3. línea 10: suponemos que la base de datos permite realizar una búsqueda de datos por posición y elemento en dicha posición.

4. línea 8: como las descripciones están ordenados por χ, una vez que superamos el threshold todas las restantes descripciones también lo harán. Como vimos, cualquiera de las medidas propuestas puede ser calculado teniendo en cuenta solamente la cantidad de elementos comunes y excluyentes de los dos conjuntos considerados. Sobrecargamos por lo tanto una vez más las medidas χ para que tome éstos tres parámetros.

2.3.7. Complejidad

Intentaremos ahora dar un análisis de la complejidad algorítmica. Tomaremos como operación básico el acceso a la base de datos.

La operación de la línea 1 es de ejecución única para todas las búsquedas (siempre que se tome el m lo suficientemente grande), así que no lo analizaremos. Haremos el análisis sobre la ejecución de una posición, pues evidentemente el algoritmo es lineal en $Posiciones$.

Las asignaciones en la línea 5 y 6 son atómicas.

Podemos suponer que la línea 8 es constante, pues la generación de número depende de las permutaciones y trabajar sobre ellos es trabajar sobre arreglos de tamaño k y $m-k$ (pequeño). Hemos visto ya que la generación de enteros (línea 9) es necesariamente $\binom{k}{l} * \binom{m-k}{n}$. Cómo de ésta cantidad depende las veces que se ejecutará el loop interno, es aquí dónde enfocaremos nuestra atención.

El algoritmo se incremente en el mismo orden que los números binomiales: sin embargo, el secreto en la eficiencia radica en escoger inteligentemente el *threshold*: si éste es 0, entonces se generan todas las posibles combinaciones de

Algoritmo 3 Búsqueda de parecidos. Entrada: *buscado* : *Dato*, *base_datos* : $\mathcal{P}(Dato)$, *threshold* : $\mathbb{R}$

```
1.  preparar_descripciones()
2.  resultado : P(B)
3.  resultados_tmp: P(B)
4.  for i :=1 to N do
5.          resultados_tmp :=∅
6.          k :=||buscado(i)||
7.          ⟨l, n⟩ :=próxima_descripción(k)
8.          while χ(k, l, n) ≥ threshold do
9.                  for all b ∈ generar_enteros(target(i), l, n) do
10.                         resultados_tmp :=resultados_tmp
                            ⋃ base_datos.obtener_por_posicion(i, b)
11.                         k :=||buscado(i)||
12.                 end for
13.                 ⟨l, n⟩ :=próxima_descripción(k)
14.         end while
15.         if i = 1 then
16.                 resultado :=resultados_tmp
17.         else
18.                 resultado :=resultados_tmp ⋂ resultado
19.         end if
20. end for
```

subconjuntos y ésto claramente es peor que una búsqueda trivial en la base de datos. Este del *threshold* elección depende del escenario, de la aplicación concreta y del χ utilizado.

Nótese finalmente que las únicas variables que aparecen en el análisis de complejidad son los cardinales de los conjuntos $\mathcal{A}_i$, y la longitud de los $buscado(i)$ y $||buscado(i)||$, pero nunca se hace referencia al tamaño de la base de datos. Por ende, tomando al acceso a la base de datos como operación atómica, la complejidad del algoritmo 3 es independiente del tamaño o composición de la base de datos.

2.4. Mejoras de implementación

Como casi siempre, el algoritmo teórico suele diferir del que se encuentre en la implementación. A la hora de calcular complejidad y transmitir las ideas detrás del funcionamiento es preferible una versión clara y concisa. Sin embargo, a la hora de la ejecución aplicamos algunos cambios que - sin cambiar la complejidad ni tocar las ideas más importantes - resultan en una ejecución más rápida

2.4.1. Reducir acceso a la base de datos

Tal como mencionamos a la hora de calcular el orden del algoritmo, éste no depende del tamaño de la base de datos siempre que tomemos como operación básica el acceso a ésta. Sin embargo, éste acceso sí suele depender del tamaño de la base. Además, se suele tratar de una base externa al programa por lo que es preferible disminuir al máximo el acceso a la misma, para reducir el traspaso de datos.

Dado que queremos obtener la intersección de los conjuntos *resultados_tmp*, un invariante del algoritmo es que el conjunto *resultado* siempre disminuye a partir de la finalización de la primera iteración. Luego, en las demás iteraciones alcanza con reemplazar la base de datos real por el conjunto *resultado* (que guarda los individuos próximos considerando solamente la primera posición).

Por lo tanto, sea g la función computado por las líneas comprendidas entre 4 y 14 del algoritmo 3 y $g(i)$ el conjunto correspondiente a *resultados_tmp* luego de la i-ésima iteración. O sea, el conjunto de individuos que coinciden de tal manera en la posición i tal que se satisface la cota del *threshold*:

$$g :: Posiciones \to \mathcal{P}(individuo) \tag{2.7}$$

Entonces, el algoritmo 3 corresponde a

$$\bigcap_{i:=1}^{N} g(i)$$

El cambio propuesto radica en calcular $\bigcap_{i:=2}^{N} g(i)$, reemplazando la búsqueda sobre la base de datos en la línea 10, por una sobre el conjunto $g(1)$.

2.4.2. Un sólo acceso a la Base de Datos

Llevamos ahora hasta el extremo el cambio propuesto en la sección anterior al acceder una sola vez a la base de datos. Para ello, necesitamos que ésta permite no sólo realizar búsquedas de individuos por posición, sino también por conjunto de posiciones; devolviendo el mismo resultado que resultaría de una unión de los elementos de éste conjunto.

Así, acumulamos los valores b de la línea 9 en un conjunto y luego realizamos una sola búsqueda grande. Dependiendo de cómo se está almacenando la base ésto podría mostrar una mejor perfomance que realizar varias búsquedas pequeñas. Sin embargo - dependiendo del escenario en cuestión - ésta *query* podría devolver un resultado demasiado grande.

2.4.3. Elegir la primera iteración

¿Qué sucede si todos los $g(i), i < N$ son de tamaños "grandes" y $g(N)$ es de tamaño mucho más reducido que éstos otros? En tal caso, acarrearemos por toda la ejecución del algoritmo un conjunto grande, solamente para reducirlo sustancialmente en la última iteración.

Es por lo tanto deseable que el j que elijamos para efectuar la primera iteración y que reemplazará a la base de datos en las demás iteraciones sea aquella que minimice $\|g(i)\|$. A no ser que contemos con alguna información adicional sobre la composición de los elementos o alguna heurística una buena elección de j es aquella que minimice $\mathcal{A}_i$ o $\|buscado(i)\|$. En el caso extremo de que exista un i tal que $buscado(i) = \emptyset$, entonces claramente la intersección final será vacía.

2.5. Cambiando el algoritmo

2.5.1. $\wedge$ y/o $\vee$

Un individuo solamente será contenido en el conjunto devuelto por el algoritmo 3 si coincide en todas las posiciones con el *threshold* especificado.

Una modificación trivial[4] pero muy útil es aclarar explícitamente cuales son las posiciones que queremos que se contemplen. O sea, en vez de iterar el for sobre

[4] sólo hace falta modificar la línea 4

todos las posiciones de Posicion, lo hacemos solamente sobre un subconjunto de éste:

Sea

$$\tilde{g} :: \mathcal{P}(Posiciones) \rightarrow \mathcal{P}(Individuo)$$

$$\tilde{g}(X) = \bigcap_{x \in X} g(x)$$

dónde $\bigcup_{x \in X} g(x)$ significa no iterar sobre todos los elementos de *Posiciones*, sino solamente sobre el subconjunto X.

Otro cambio deseable es una que modifique el algoritmo de tal manera que no devuelve el conjunto de los individuos que coinciden en **todos** las posiciones, sino que lo haga en **alguna(s)** de las posiciones.

Aunque no modifica el algoritmo de manera substancial (alcanza con cambiar $\bigcap$ por $\bigcup$ en la línea 18), ni lo hace en la complejidad del algoritmo (pues la cantidad de accesos a la base de datos permanece invariable) no permite realizar la mejora expuesta en la sección 2.4.2. Ésta mejora radicaba en el hecho de que el objetivo era calcular la intersección, por lo que el conjunto *resultado* se reducía en cada iteración y por lo tanto se podía partir desde el conjunto devuelto por la primera iteración. Ahora sin embargo, el conjunto *resultado* siempre crece en cada iteración.

Definición 2.14 *Llamaremos $f_\wedge$ a la función computada por el algoritmo 3.*

Definición 2.15 *Llamaremos $f_\vee$ a la función computada por el algoritmo 3, reemplazando $\bigcap$ por $\bigcup$ en la línea 18.*

Por lo tanto, utilizando g usada en la Ecuación 2.7 tenemos:

$$f_\wedge = \bigcap_{i:=1}^{N} g(i)$$

$$f_\vee = \bigcup_{i:=1}^{N} g(i) \tag{2.8}$$

Otra manera de ver la $f_\vee$ es tomarla en función de la $f_\wedge$.

Entonces:

$$f_\vee = \bigcup_{X \in \mathcal{P}(Posiciones)} \tilde{g}(X) \tag{2.9}$$

o sea, tomando todos los subconjuntos posibles.

Computacionalmente, la definición 2.8 es muchísima más eficiente que aquella dada en 2.9, pues aquella es lineal en el tamaño de *Posiciones*, mientras que ésta es exponencial (pues $\|\mathcal{P}(A)\| = 2^{\|A\|}$). A pesar de ello, no solamente son matemáticamente iguales sino que la segunda refleja de mejor manera lo que sucede cuando realizamos una búsqueda en la que no estamos interesados en una coincidencia en todos las posiciones. Primero buscamos sobre el conjunto total, y en el caso de que ésto no devuelva resultado comenzamos a disminuir las posiciones a tener en cuenta siguiendo algún patrón específico del escenario.

La próxima idea es hacer éste patrón específico explícito. O sea, ordenamos todos los subconjuntos de *Posiciones*, y f_V ahora comienza con el subconjunto más importante ejecutando $\tilde{g}$ sobre ello. Si no devuelve resultados (o sea, el resultado $\emptyset$) sigue con el siguiente más importante y así sucesivamente.

Cambiando la métrica: cantidad de coincidencias

Supongamos que estamos calculando la f_V y buscamos aquellos individuos que coincidan en al menos una cierta cantidad de marcadores (y no solamente en más de cero), simplemente hay que guardar en el conjunto *resultados* la cantidad de coincidencias que viene llevando éste individuo. Así, en cada iteración chequeamos si el individuo ya pertenece a *resultados*: caso lo haga incrementamos el contador que lleva la cantidad de coincidencias para ése individuo. Caso contrario, lo inicializamos en uno.

Esto permite además cambiar la métrica para ordenar luego el conjunto resultante: puede que antes del índice de similitud estemos interesados en la cantidad de posiciones en la que hubo coincidencias. Esto abarcaría por ejemplo el caso con varias posiciones donde un individuo coincide en la mayoría con el *buscado*, pero lo hace con un índice muy bajo. En cambio, otro lo hace en unos pocos pero en una medida mucho mayor. Depende de cada escenario analizar a cuál individuo darle más prioridad.

2.5.2. Tratamiento diferencial de las posiciones

El algoritmo 3 se ocupa de cada posición de manera independiente. Solamente al final se une dichas búsquedas individuales y se los combina: ya se con una unión, o con una intersección.

Sin embargo, ésta independencia permite realizar un tratamiento diferencial para las distintas posiciones. Veremos dos posibles modificaciones al algoritmo que pueden ser útiles y que también se pueden combinar:

Varios *thresholds*

Dependiendo del escenario, puede ser que deseamos una mayor similitud en una posición que en otra. Generalizar los *thresholds*, para que éstos estén parametrizados de acuerdo a la posición es directo: en vez de tomar uno sólo como parámetro toma varios.

Ese sí o sí

Vimos en la sección 2.5.1 una manera de no buscar en la intersección del resultado de g en todas las posiciones, sino en la unión. Considerar cada posición de una manera separada puede llegar a ser necesaria en el caso de una búsqueda en la que es obligatorio que haya coincidencia en un(os) posición(es), mientras que existen otros cuya coincidencia son deseables pero no necesarias.

También es sencillo adaptar el algoritmo a este caso: a la hora de unir las búsquedas al final de cada iteración, definimos una estrategia distinta para cada posición.

Los operandores sobre conjuntos dan una libertad mucho mayor, y utilizar la unión y la intersección solamente son algunas de ellos. Si, por ejemplo deseamos **eliminar** los individuos que tengan coincidencia en cierta posición; entonces buscamos al final sobre dicha posición y restamos el conjunto obtenido de *resultados*.

Capítulo 3

Aplicación: Netflix

"*We're quite curious, really. To the tune of one million dollars.*"
Netflix, servicio de alquiler de películas

Presentaremos ahora una posible aplicación del algoritmo descripto en la sección 2.3.6. Se trata de una base de datos de una alquiladora de DVD's que provee a los **`usuarios`** la posibilidad de poner **`notas`** a las **`películas`**.

Dado un conjunto de asignaciones (pares de películas y nota) y una nueva película vista por un usuario el objetivo es pronosticar la nota que el usuario le pondrá.

Presentaremos el problema introduciendo a la empresa que dio origen al problema y a la competencia que lanzó: explicaremos las reglas de éste, antes de mostrar como lo mapearemos a nuestro algoritmo.

3.1. Netflix

3.1.1. Acerca de...

Netflix Inc. (NASDAQ: NFLX) es una empresa que provee un servicio de alquiler de películas. Surgió en el 1998 y gracias a su sistema de entrega de DVD's a domicilio que lanzó en 1999, se convirtió en la mayor empresa en su rubro (alquiler de DVD's online).

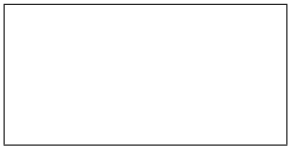

Figura 3.1: Logo oficial

3.1.2. Cómo funciona

Los pasos siguientes detallan el funcionamiento de Netflix desde el punto de vista de un usuario:

1. El usuario crea una lista personalizada de películas
2. Netflix manda cierta cantidad de los DVD's de la lista en un día hábil
3. El usuario se puede quedar con la película cuanto tiempo lo desea
4. Para obtener la próximo película de la lista, el usuario devuelve la película con un sobre pre-pago

Netflix no cuenta con locales físicos tradicionales, sino solamente con centros de envío.

3.1.3. Algunos números de Netflix

- posee 75000 títulos a elegir, de los cuales 35000 por día están en distribución
- posee 42 millones de DVD's, de los cuales 1.6 millones por día están en distribución.
- emplea 1300 personas
- tiene 44 centros de envío en los Estados Unidos
- La empresa afirma que el 90 % de sus usuarios puede ser alcanzado en menos de un día desde su centro de envío local
- En el primer cuatrimestre del 2007 publicó los siguiente datos ([Net07a]):
  - Usuarios para ésa fecha: 6.8 millones
  - Ganancias en ése período: U$S 305.3 millones
  - Usuarios estimados para fines del 2007: 8.4 millones
  - Ganancias estimada para el 2007: U$S 1300 millones (ver figura 3.1.3)
- El 24 de febrero del 2007, Netflix anunció el envío de su 1000-millonésimo DVD[1].

[1] ver `http://www.msnbc.msn.com/id/17331123/`, `www.foxnews.com/story/0,2933,254632,00.html`, `http://biz.yahoo.com/prnews/070225/nysu004.html?.v=78`

Año	Ganancia (millones)
2003	U$S 270.4
2004	U$S 500.6
2005	U$S 682.2
2006	U$S 996.7

Figura 3.2: Ganancias de Netflix a lo largo de los años

Suscripciones

Netflix cuenta con 9 diferentes planes de suscripción. La diferencia general radica en la cantidad de películas que pueden estar al mismo tiempo en la casa del usuario:

Cuota mensual	Títulos al mismo tiempo	Límite mensual
$4.99	1	2
$9.99	1	ilimitado
$14.99	2	ilimitado
$17.99	3	ilimitado
$23.99	4	ilimitado
$29.99	5	ilimitado
$35.99	6	ilimitado
$41.99	7	ilimitado
$47.99	8	ilimitado

3.1.4. No todo es color de rosa

La propaganda de la empresa promete el envío del título pedido dentro de un día hábil desde el centro de distribución local al usuario. Sin embargo, muchas veces los títulos no son enviados del centro local, y usuarios que piden muchas películas han presentado quejas por experimentar demoras, surgiendo críticas de que Netflix favorece éste comportamiento del sistema, pues su mayor gasto es el envío (y el sobre pre-pago).

En setiembre del 2004, se dio inicio al caso *Frank Chavez vs. Netflix Inc*[2] en California. Se acusó a la empresa de "propaganda falsa" al insistir sobre la incompatibilidad entre la promesa de "alquiler ilimitado" y la demora de un día en el envío.

[2]ver `http://www.sftc.org/Scripts/Magic94/mgrqispi94.dll?APPNAME=IJS&PRGNAME=ROA&ARGUMENTS=-ACGC04434884`

3.1.5. Evaluación de películas

Uno de los "caballitos de batalla" de Netflix es su sistema de recomendación de películas. Éste sistema se basa en evaluaciones hechas por los propios usuarios: los usuarios tienen la posibilidad de otorgarle a las películas una nota entre 1 y 5. Netflix afirma ([Net07b]) que el usuario promedio evaluó 200 películas.

Basándose en las preferencias del usuario (las evaluaciones hechas en el pasado), más las preferencias de los otros usuarios, se le recomienda al usuario una serie de películas. Actualmente, el sistema cuenta con 1700 millones de evaluaciones.

3.2. Neflix Prize

El 2 de octubre del 2006, Netflix lanzó el **`Netflix Prize`**, una competencia abierta cuyo fin es mejorar el algoritmo actualmente usado por la empresa para la proyección de evaluaciones. Éste algoritmo - llamado **`Cinematch`** - mejora en un 9,6 % la forma más trivial de resolver el problema (calcular la media de todas las notas sobre ésa película); y la meta puesta por la competencia es mejorar a su vez ésto en otro 10 %.

3.2.1. Definiendo el problema

Netflix provee por un lado un conjunto de películas, usuarios y evaluaciones realizadas. Por otro lado un test (un conjunto de usuarios y películas) que se usará para evaluar qué tan bueno es el algoritmo propuesto.

Para evaluar ésto, la medida elegida es **`rmse`** (root mean square deviation):

Sea $\tilde{\Theta}$ una estimación del parámetro Θ. Entonces:

$$RMSE(\tilde{\Theta}) = \sqrt{E((\Theta - \tilde{\Theta})^2)} \tag{3.1}$$

dónde E es el *valor esperado* de una variable aleatoria.

Por lo tanto: sean $x_1, x_2, \ldots, x_n$ las notas puestas para la evaluación i y $\tilde{x_1}, \tilde{x_2}, \ldots, \tilde{x_n}$ la estimación devuelta por un algoritmo.

Entonces, la Ecuación 3.1 queda:

$$RMSE(\tilde{\Theta}) = \sqrt{\frac{\sum_{i:=1}^{n}(x_i - \tilde{x_i})^2}{n}}$$

3.2.2. Información provista

El conjunto de prueba que provee Netflix consiste de:

- 17.770 películas
- aproximadamente 480.000 usuarios
- aproximadamente 100 millones de evaluaciones

Además, el test consiste de 2.817.131 pares de usuarios y películas. La entrega cosiste en un archivo con las notas estimadas para cada uno de ésta evaluaciones, y Netflix informará la medida RMSE correspondiente (no se tienen acceso a las notas reales de éstos evaluaciones, pero sí a un archivo similar de prueba cuya nota real sí está contenida en el conjunto de prueba).

3.2.3. Objetivos

El RMSE aplicado a la media de las notas de cada película es 1,0540 sobre el conjunto de prueba. El algoritmo de Cinematch sobre el conjunto de prueba da un RMSE de 0,9514 (sin entrenamiento) ó 0,9525 (entrenándolo con el conjunto de prueba). Ésto representa una mejora del 9,6 % con respecto al cálculo trivial.

Objetivo de la competencia es obtener una mejora de otro 10 % sobre el algoritmo Cinematch, o sea, obtener un RMSE de 0,8572 (un 19 % de mejora con respecto a la medida trivial).

3.2.4. Premio

Para hacerse acreedor del premio, el grupo o persona que logró la meta anterior debe entregar a Netflix una descripción y el código del algoritmo. El premio consisten en U$S 1.000.000, y además se entrega un premio de progreso al que logró el mejor puntaje cada año y además superó en 1 % al ganador del año anterior, que consiste de U$S 50.000 (5 % del premio).

3.2.5. Estado Actual

A los seis días de haberse largado la competencia ya había un grupo que superó el resultado obtenido por Cinematch.

En el momento de escribir este trabajo[3], el líder de la competencia es un grupo de la Universidad de Budapest que logró un RMSE de 0,8847 (7,02 % mejor que el resultado de Cinematch), enviado el 5 de abril de 2007. En segundo lugar está un profesor y dos estudiantes de la Universidad de Toronto (0,8853: 6,95 %).

[3] abril 2007

3.3. Adaptando

Para poder aplicar el algoritmo 3, definimos como individuos a los usuarios, y en cada posición i mapearemos películas a la nota puesta por ese usuario a la película i.

El escenario definido es – por lo tanto –:

- N = cantidad de películas (17770 para el conjunto de prueba)
- $\mathcal{A}_i = \{1, 2, 3, 4, 5\}$

Con éste escenario, tenemos que $\forall individuo, \forall i, ||individuo(i)|| \leq 1$, pues tomaremos las notas únicas[4]. Para extender esta limitación (que volvería demasiado exigente al algoritmo y haría perder las ventajas de buscar individuos parecidos), decidimos que el conjunto no sólo contendrá la nota actual, sino también las que están a distancia no mayor a 1 de ésa nota. Así, si el usuario d evalúa la película i con 3, y la j con 5, entonces $d(i) = \{2, 3, 4\}$ mientras que $d(j) = \{4, 5\}$.

Ejemplo 3.1 *A continuación se detallan las películas vistas y evaluadas por el usuario* 1473980*:*

Id Película	*Nombre*	*Nota*
10767	*"The Chronicles of Narnia: The Silver Chair"*	*4*
28	*"Lilo and Stich"*	*2*
2102	*"The Simpsons: Season 6"*	*5*

Entonces, el dominio del individuo 1473980 *será {2102. 28 10767}, con:*

$$\begin{aligned} ind1473980\,(10767) &= \{3, 4, 5\} \\ ind1473980\,(28) &= \{1, 2, 3\} \\ ind1473980\,(2102) &= \{4, 5\} \end{aligned}$$

3.3.1. Conjuntos no válidos

Dado que extendemos la nota puesta por el usuario a un conjunto que también abarca a su predecesor y sucesor, los números binarios que identifican a conjuntos válidos son:

- 00011 (1)

[4]no estamos considerando varias evaluaciones de un mismo usuario a una misma película

- 00111 (2)
- 01110 (3)
- 11100 (4)
- 11000 (5)

Así, existen miembros de grupos que no son válidos, a pesar de cumplir con la descripción. Así, por ejemplo, el conjunto 10011 cumple con la descripción de compartir dos bits y tener uno exclusivo con respecto de 00111, pero no es aquél un identificador válido (como si lo sería 01110). Es necesario por lo tanto filtrar las instancias generadas en la línea 8 del algoritmo 3.

Capítulo 4

Aplicación: ADN-Matching

"*DNA, you know, is Midas' gold. Everyone who touches it goes mad.*"
Maurice Wilkins, físico y biólogo molecular

Mostraremos en éste capítulo otro posible uso de nuestro algoritmo, que fue el problema que incentivó en un principio el diseño del mismo.

Se trata de la búsqueda de perfiles de ADN entre perfiles de individuos y de perfiles de mezcla (obtenidos, por ejemplo de casos de abuso sexual).

Primero explicaremos muy por arriba algunos conceptos biológicos necesarias para comprender a los perfiles[1]. Luego mostraremos como abstraímos éstos conceptos para que podamos definir nuestro escenario, y finalmente cómo aplicamos el algoritmo a él y características y cambios particulares.

4.1. Preliminares biológicos

Antes de entrar de pleno en la descripción de lo qué es un perfil genético - o siquiera de cómo aplicamos nuestro algoritmo a la búsqueda de perfiles parecidos - nos detendremos para hacer una descripción muy incompleta y superficial de algunos conceptos de biología molecular. A medida que avanzamos, rescataremos algunas propiedades importantes del ADN.

Luego analizaremos más en profundidad los perfiles genéticos, algo de su historia, sus usos actuales y diferentes mecanismos de obtención. Finalmente mencionaremos el uso de bases de datos de perfiles y daremos el ejemplo de las dos más conocidas.

[1] Para un estudio superficial pero más detallado, se recomienda la lectura del libro "Introduction to Computational Molecular Biology" de Setubal y Meidanis ([SM97]), y un poquito más detallado los capítulos 14-17 de [Cur00]

Es importante recalcar que - a diferencia de la matemática y por ende de las ciencias de computación - en la biología las excepciones son comunes y los biólogos están acostumbrados a trabajar con ellas como la regla. Una de las mayores desafíos en el trabajo interdisciplinario entre éstas ciencias sea probablemente justo éste cambio de paradigma.

No nos detendremos a explayarnos en todas las excepciones, y a veces ni siquiera las mencionaremos.

4.1.1. El plan de la vida

Vida, proteínas y nucleótidos

Los actores principales en el proceso de mantener el intercambio de un organismo vivo con su entorno (condición necesaria para la vida según [SM97]) son las **`proteínas`** y los **`nucleótidos`**. Las proteínas se encargan de lo que la vida es y hace en un sentido físico, mientras que los nucleótidos codifican la información necesaria para la producción de proteínas y son los responsables de pasar esta información a las siguientes generaciones.

ADN

Los organismos vivos tienen dos diferentes tipos de ácidos nucleicos: el **`ácido ribonucleico`** (**`ARN`**) y el **`ácido desoxiribunucleico`** (**`ADN`**). Además de que existen diferencias estructurales y de forma, la función de ambos difiere: mientras que la de los ARN varía, el ADN realiza prácticamente una sola: codificar información.

Dado que la información necesaria para construir cada proteína y ARN se encuentra en las moléculas de ADN [2], se le suele denominar al ADN el "plano de la vida".

Propiedad 1 *El ADN de una persona permanece inalterable a lo largo de su vida.*

El ADN consta de nucleótidos que difieren una de otro en una molécula llamada base, de la cual existen 4 diferentes: adenina (A), guanina (G), citosina (C) y timina (T). Dada que ésta molécula es la única diferencia entre los nucleótidos, intercambiaremos las palabras *bases* y *nucleótidos* indistintamente.

[2] a partir de aquí haremos referencia solo a proteínas, aunque en la mayoría de los casos lo correcto sería decir proteínas y/o moléculas de ARN

Sin embargo, el ADN no consta de una sola cadena de nucleótidos, sino de dos enlazadas una a la otra en una estructura de hélice. Fue éste el famoso descubrimiento - que les valió el título Nobel de Fisiología o Medicina - de James Watson, Francis Crick y Maurice Wilkins en 1952[3] (ver figura 4.1).

Figura 4.1: Esquema de la estructura de ADN

Por encontrarse las bases de a pares, la unidad de medida del ADN es el **bp** (*Watson-Crick base pair*). En la figura 4.1.1 se detallan la longitud del ADN de algunas especies.

Genes y ADN no-codificante

Existen pedazos continuos de ADN que codifican la información para producir proteínas, pero no todo el ADN lo hace. Además, cada proteína es codificada (generalmente) por un y sólo un pedazo continuo de ADN: este pedazo se denomina **gen**. Sin embargo, no todo el ADN está compuesto por genes: de hecho - en

[3]"En lugar de creer que Watson y Crick hicieron la estructura del ADN, yo más bien pondría el acento en que fue la estructura la que hizo a Watson y Crick. Después de todo, yo era casi totalmente desconocido en esa época y a Watson se lo consideraba en la mayoría de los círculos como demasiado brillante para ser realmente bueno. Pero me parece que lo que se pasa por alto en esos argumentos es la belleza intrínseca de la doble hélice de ADN. Es la molécula la que tiene estilo, tanto como los científicos." Francis Crick en [Cri74]

Especie	Número de cromosomas	Tamaño del genoma (bp)
Bacteriófago δ (virus)	1	$5 * 10^6$
Escherichia coli (bacteria)	1	$5 * 10^6$
Saccharomyces cerevisiae (levadura)	32	$1 * 10^7$
Caenorhapditis elegans (gusano)	12	$1 * 10^8$
Drosophila melanogaster (mosca de la fruta)	8	$2 * 10^8$
Homo sapiens (humano)	64	$3 * 10^9$

Figura 4.2: Algunas especies importantes para la biología molecular e investigaciones genéticas

el humano - cerca del 90 % no lo hace. Éstas regiones intergénicas se llama **ADN no-codificante** (también llamado **ADN basura**), y los biólogos todavía no se pusieron de acuerdo acerca de la utilidad de éstas regiones (o si tienen alguna utilidad) y del porque se mantienen después de millones de años de evolución.

Propiedad 2 *Existen porciones del ADN que no codifican características físicas (fenotipo)*

La gran mayoría de los pares bases de ADN coincide en todos los humanos. Sin embargo, unos 3 millones de pares (cerca del 0.10 % del genoma entero) difiere de persona en persona.

Todo humano, animal o planta reproducida sexualmente tiene un fenotipo o apariencia física característico porque posee una composición hereditaria única. La excepción a ésta regla son los mellizos gemelos, que possen el mismo genotipo pero, como consecuencia de complejos eventos desarrollan fenotipos substancialmente diferentes[4].

Propiedad 3 *Cada persona tiene suficiente ADN único como para determinarlo unívocamente.*

[4]actualmente ya existen métodos para distinguir entre dos mellizos gemelos. Ver [Ben06]

Cromosomas

Cada célula de los organismos tiene algunas moléculas de ADN muy largas, llamadas **cromosoma** (ver figura 4.1.1).

Los procariontes[5] (generalmente) tienen un sólo cromosoma. Por otro lado, en los eucariontes[6] los cromosomas (suelen) aparecer de a pares, y por ello las células que los cargan se llaman **diploides**.

Propiedad 4 *El ADN de una persona es siempre el mismo, no importa el tejido del cual fue extraído.*

Cada miembro de cada par es heredado por uno de los padres y a cada gen de un par le corresponde otro gen del otro par. Algunos genes pueden aparecer en formas diferentes (toman valores distintos), llamados **alelos**.

Propiedad 5 *El ADN de una persona consta de cromosomas formado por un alelo del padre y otro de la madre.*

4.1.2. Perfil genético

Notación

En el inglés se usan tanto los palabras de *profile* y *typing* como *genetic typing*. Usaremos la palabra española *perfil* para denotar a un perfil genético.

"Análisis de identificación de ADN, testeo de identidad, *profiling*, huella genética, *gentic typing* se refiere a la caracterización de una o más relativamente poco frecuentes característica de un genoma individual o composición hereditaria" [Kir90].

Justificación del uso

Los perfiles genéticos se basan en las propiedades enunciadas en la sección anterior:

- se puede usar un perfil genético como clave para determinar unívocamente a las personas (propiedad 3)
- dicha unicidad está garantizada por la inalterabilidad del ADN (propiedad 1)

[5] organismos compuestos por células sin núcleo celular diferenciado, es decir, cuyo ADN no se encuentra confinado dentro de un compartimento limitado por membranas

[6] organismos compuestos por células cuyo ADN se encuentra encerrado dentro de una doble membrana que limita el núcleo

- no importa el tejido del cual se extrajo la muestra de ADN (propiedad 4).
- éste perfil genético se extrae de regiones que no codifican características físicas (propiedad 2)
- cada trecho de ADN que se elija será compuesto por dos valores (propiedad 5)

En el comienzo estaba Jeffreys

El perfil genético fue descripto por primera vez en marzo de 1985 por el equipo de Sir Alec Jeffreys de la Universidad de Leicester (Inglaterra) en su artículo publicado en la revista *Nature* "Hypervariable 'minisatellite' regions in human DNA" [JWT85].

Jeffrey descubrió que ciertas regiones de ADN contienen secuencias de ADN repetidas una y otra vez. Encontró además de que el número de secuencias repetidas en una muestra puede variar de individuo en individuo. Al desarrollar una técnica para examinar la variación de éstas secuencias de ADN repetidas, dio inicio a la posibilidad de realizar test de identidades genéticas.

Algo de laboratorio

Ésta regiones de ADN repetidas se denominan **VNTR**s, (Variable Number of Tandem Repeats). La técnica desarrollada por Jeffreys se conoce como RFLP (restriction fragment length polymorphism, restricción de longitud de fragmentos polimórficos), porque implica el uso de una encima restrictora para cortar el ADN alrededor del VNTRs.

Actualmente existen diferentes métodos para examinar el ADN a fin de crear un perfil genético, pero la mayoría utiliza porciones no-codificantes del ADN:

- RFLP: Restriction Fragment Length Polymorphism
- PCR:Polymerase Chain Reaction
- STR: Short Tandem Repeat

El último procedimiento es el más utilizado.

Excede el objetivo de nuestro trabajo hacer un estudio más intensivo sobre éstos procedimientos, pero animamos a consultar la bibliografía respectiva.

Armando un perfil

Un perfil se forma por cierto número de **marcador** (comúnmente de 8 a 12, según [Ben06]). Cada marcador hace referencia a cierto *loci* sobre el cual se aplica alguna de las técnicas enunciadas para obtener los VNTRs.

La importancia a la hora de elegir la cantidad de *loci* a tener en cuenta depende de cuan grande se quiere hacer la probabilidad de el mismo perfil para más de una persona (nótese que generalmente en una escena de crimen, la cantidad de sospechosos es reducida). El CODIS (sección 4.1.3) usa 13 *locis* de tal manera que la probabilidad de que dos personas tengan el mismo perfil sea $1 : 10^9$. Claramente, ésta es una medida teórica. O sea, no toma en cuenta la introducción de errores por muestras impuras, o simplemente herramientas y ambientes no-perfectas en el laboratorio.

La clava en el uso de ADN como evidencia radica en la comparación del ADN de un perfil de mezcla con aquél de un sospechoso. El proceso engloba tres pasos:

- extraer ADN de una escena de crimen y de los sospechos
- analizar al ADN para crear un perfil de ADN
- comparar los perfiles entre ellos

Es posible extraer ADN de casi cualquier tejido: pelo, uñas, hueso, dientes y fluídos corporales.

Ejemplos

En cada marcador o *locus* de un perfil **individual**, cada individuo tiene dos *alelos*: uno heredado del padre, y uno de la madre. En el caso que los dos valores sean iguales, entonces se los designa como uno solo.

Daremos un ejemplo con dos alelos

Ejemplo 4.1

	Marcador 1	*Marcador 2*
Padre	*3*	*5,7*
Madre	*5,3*	*5,8*
Hijo	*3,5*	*5*

Llamamos un perfil **mezclado** a una mezcla de varios perfiles de DNA: la cota aquí está dado por la cantidad de valores posibles que pueden tomar los alelos en éste marcador.

Ejemplo 4.2

Perfil mezclado	*3,5,2*
Posible Perfiles aportadoras	*3; 5; 2; 3,5; 3,2; 5,2*

Como los posibles perfiles deben tener en cada marcador dos alelos (iguales o distintos) de aquellos que aparecen en ése marcador del perfil mezclado, la cantidad total de posibles aportantes es:

$$\prod_i \left(\binom{n_i}{2} + n_i \right)$$

dónde n_i es la cantidad de alelos en el marcador i del perfil de mezcla.

4.1.3. Bases genéticas forenses

En muchas casos se puede obtener el ADN de la escena de crimen, pero no de sospechosos. Para tales casos, existen las base de datos de perfiles genéticos. En Estados Unidos, por ley todos los estados tienen que aportar a la base de datos nacional a condenados por crímenes sexuales.

Las preguntas típicas que debe responder un sistema de administración de perfiles son las siguientes:

- ¿Qué escenas de crimen tienen agresores comunes?
- ¿En qué escena de crimen ha participado cierto individuo?

Comentaremos brevemente algunas de las bases de datos más famosas:

NDNAD

El *UK National Criminal Intelligence DNA Database* es la mayor base de datos de perfiles actualmente existente. Contiene el perfil de 3.4 millones de perfiles. Utiliza un análisis STR sobre 12 *locis* (desde 1999, 20), aparte del marcador de género (*Amelogenin*). Cuando se agrega un nuevo perfil, se chequea contra todas las escenas de crímenes, y lo mismo se hace al agregar escenas de crímenes (además de chequear en éste caso también contra otras escenas de crímenes). [For05]

CODIS

Combined DNA Index System es el nombre del sistema que administra los perfiles de ADN de la base de datos nacional y estadual del FBI (Oficina de investigaciones federales de los Estados Unidos de América). Incorpora además un algoritmo que busca por perfiles, resultando en un conjunto de eventuales candidatos, los que luego serán analizados. Dicho algoritmo es de código cerrado, aunque una licencia puede ser adquirida por los países que así lo pida.

Surgió para ayudar en la resolución de casos criminales de índole sexual y para uso forense, pero se expandió y actualmente abarca también casos de desaparecidos y no-identificados,

CODIS está implementando como una base da datos distribuida con tres nivel jerárquicos: local, estadual y nacional. NDIS es el mayor nivel en la jerarquía CODIS, y permite a los laboratorios participando en el CODIS programa intercambiar y comparar perfiles de ADN a nivel nacional.

El CODIS usa un análisis STR que examina 13 *locis*. Para febrero del 2007, el CODIS tenía 4398639 de perfiles [7].

4.2. Matematizando los perfiles

Para adaptar a un perfil como entrada a nuestro algoritmo, definimos el siguiente escenario:

- N = 12, 16, ó 20 dependiendo de cuántos marcadores estamos considerando
- $\mathcal{A}_i$ será el conjunto de posibles valores de alelos que puede tomar el marcador i.

Tenemos entonces la restricción de que para todos los individuos que representan a un individuo:

$$||individuo(i)|| \leq 2$$

Ejemplo 4.3 *Supongamos el siguiente perfil* $p1$ *de 3 marcadores:*

Loci	*Valor*
Mpmv6	*4, 11*
Ren1	*3.2, 8*
D1Mit496	*5*

[7] `http://www.fbi.gov/hq/lab/codis/index1.htm`

Entonces, el mapeo a nuestros individuos es bastante directo. En éste caso:

$$\begin{aligned} dom(p1) &= \{Mpmv6, Ren1, D1Mit496\} \\ p1(D1Mit496) &= \{5\} \\ p1(Ren1) &= \{3,2,8\} \\ p1(Mpmv6) &= \{4,11\} \end{aligned}$$

4.3. Adaptando

Nótese que una intersección exacta no es lo que estamos buscando acá: por ejemplo, en el ejemplo 4.2 el perfil que tienen en dicho marcador los alelos $(5,7)$ no es sospechoso, a pesar de sí tener una intersección no vacía (5).

Al usar sin embargo las diferentes medidas presentadas y los *thresholds* de manera inteligente, se pueden salvaguardar éstos casos. Así, el caso anterior se resuelve con restringir el P_B con un *threshold* de 1,0.

Por lo tanto, las preguntas de la sección 4.1.3 evidentemente se pueden generalizar con una búsqueda de parecidos.

En general tenemos pues cuatro casos:

Tipo *buscado*	Tipo Base de Datos
Individuo	Individuo
Individuo	Mezcla
Mezcla	Individuo
Mezcla	Mezcla

Para el primer caso, depende de lo que queremos lograr: si la búsqueda es exacta, entonces - dependiendo del motor de base de datos utilizado - existen formas más rápidas de obtener el resultado. Si lo que deseamos encontrar son personas "parecidas" (genéticamente parecido, o sea, algún tipo de familiar), probablemente convenga usar algún algoritmo de filiación, campo mucho más estudiado que el que nos está ocupando (véase por ejemplo [Bre97]).

Para el segundo caso, basta con pedir $P_A \geq 1$ y en el tercer $P_B \geq 1$.

Finalmente, los *thresholds* para P_B, P_A ó P_B y P_A (para el 2°, 3° ó 4° respectivamente) dependen de los objetivos del usuario.

Por lo tanto:

Tipo *buscado*	Tipo Base de individuos	
Individuo	Individuo	$P_A = P_B = 1$ (no lo consideraremos)
Individuo	Mezcla	$P_A = 1, P_B$ de acuerdo al usuario
Mezcla	Individuo	$P_B = 1, P_A$ de acuerdo al usuario
Mezcla	Mezcla	P_A, P_B de acuerdo al usuario

Queda claro por lo tanto que trabajaremos con dos *thresholds* en vez de uno sólo.

Sin embargo, ésto no complica la situación pues simplemente hay que modificar la guarda en la línea 8 del algoritmo 3. Si ordenamos las descripciones por P_A ó P_B es indistinto pues pedimos que $P_A \geq threshold_A$ **y** $P_B \geq threshold_B$. Cuando la condición de la línea 8 falla habremos recuperados **exactamente** todas las descripciones que satisfacen las dos desigualdades impuestas por los *thresholds*.

Capítulo 5

Conclusiones

Planteamos en éste trabajo el problema general de encontrar individuos parecidos a uno dado dentro de un conjunto grande de individuos. Para ello, en primer lugar desarrollamos una métrica basada fuertemente en los conceptos de precision y recall de Information-retrieval y luego usamos éstas métricas para diseñar un algoritmo que ataca el problema computacional de procesar un gran conjunto de datos por un lado distinto al habitual: tratamos de ignorar los individuos que sabemos de antemano estarán lejos y solamente considerar los cercanos.

5.1. El Algoritmo: propiedades bonitas y otras no tantas

Encontramos una manera de agrupar los individuos en grupos, con una descripción general que los caracteriza. Ésta descripción solamente hace referencia a la cantidad de elemenos excluyentes y comunes entre dos conjuntos; no es influenciada por la naturaleza de los elementos. Sabemos de antemano cuáles descripciones son las que realmente satisfacen nuestro requerimiento (léase, el *threshold* que se le pasa como parámetro al algoritmo) e instanciamos solamente a los grupos que tienen ésta descripción, donde instanciar significa generar todos los números que cumplen dicha descripción.

El enfoque distinto del problema que consiste en primero buscar eventuales candidatos y luego verificar que dichos candidatos realmente se encuentran en la base de datos tiene como consecuencia que el algoritmo presentado es independiente del tamaño de la base de datos.

Además, siendo nuestros individuos arreglos de conjuntos, buscamos independiente sobre cada uno de las posiciones. Eso permite tener una actitud distinta hacia las distintas posiciones y adaptar el algoritmo a diferentes escenarios posibles. Hemos dado una serie de éstas modificaciones (ver Sección 2.5).

Una propiedad negativa del algoritmo es que es extremadamente sensible al tamaño de los $\mathcal{A}_i$: si éstos crecen demasiado, la complejidad se dispara de manera exponencial. Es imprescindible aquí el uso del *threshold* para reducir este impacto.

Resumiendo, hemos desarrollado un algoritmo que es:

- independiente en complejidad de la base de datos
- independiente de la naturaleza de los elementos utilizados
- fácilmente modificable para adaptarlo a escenarios distintos

5.2. Resultados concretos

Analizamos ahora los resultados obtenidos al aplicar el algoritmo a las dos aplicaciones expuestas en el trabajo:

5.2.1. Netflix

Dado un usuario, buscamos los usuarios parecidos, intersectamos éstos con aquellos que vieron la película cuya nota queremos proyectar y aplicamos alguna estrategia para obtener la nota propuesta para el usuario *buscado* a base de la evaluación que hicieron éstos usuarios parecidos de dicha película.

Es fácilmente evaluable el desempeño del algoritmo en ésta aplicación, pues la misma competencia provee una manera de verificar objetivamente si se alcanzaron los resultados esperados.

Tiempos excesivos

No hemos realizado una submisión a la competencia, porque en el caso concreto de testear el algoritmo contra el archivo de prueba, la magnitud de las búsquedas a realizar supera el tiempo disponible (extrapolando, una estimación de 30 días). No creemos que es éste un problema propio del algoritmo, pues en ningún momento nos esforzamos por optimizar el acceso de la base de datos en sí. Como mencionamos varias veces, el algoritmo en sí no se ocupa con éstos problemas.

Es importante por lo tanto destacar que los números que mencionaremos son los resultados de haber procesado entre un 5 % y un 25 % del total necesario para hacer una comparación real contra los resultados de los otros competidores en la competencia.

Estrategias

Una vez resuelto usar nuestro algoritmo, resta la pregunta de:

1. definir el *threshold*
2. qué medida de las propuestas usar
3. cuál estrategia utilizar para obtener una proyección de la nota del usuario del conjunto resultante de usuarios parecidos.
4. elegir correctamente el conjunto de películas de entrada al algoritmo.

threshold Como deseamos filtrar todos los usuarios que hayan puesto no una nota exacta a la misma película que vio el usuario (pues es un enfoque demasiado restrictivo), sino que haya puesto una nota menos o más. Como explicamos en la sección 3.3, las notas del 1 al 5 forman el $\mathcal{A}_i$ de éste escenario.

Por lo tanto, lo que deseamos es una coincidencia de 2 elementos en 3 (o de 1 en 2, si las notas son de los extremos): por lo tanto tomamos nuestro *threshold* como 0.6

Medida Con respecto a la medida, nos quedamos con la f–measure por balancear precision y recall. A nivel de individuos, tomamos la F_1 (ver Definición 2.8, que recordemos es:

$$F_1(d_1, d_2) \triangleq \frac{\sum_{\substack{i:=1 \\ i \in dom(d1) \cap dom(d2)}}^{N} F(d_1(i), d_2(i))}{||\mathbf{dom}(d_1) \cap \mathbf{dom}(d_2)||}$$

Dividimos pues la suma sobre la cantidad de elementos sumados, pues que un usuario haya visto más películas que otro no expresa nada sobre su similitud entre dichos usuarios.

Estrategia para calcular proyección Para ésta decisión, tomamos una elección trivial: ordenamos el conjunto resultante de acuerdo a la medida usada y promediamos las notas puestas por los primeros (más cercanos) 10 usuarios. Es ésta nuestra propuesta de nota para el usuario.

En el caso de que el conjunto devuelto sea de cardinalidad cero luego de filtrar el usuario buscado (pues en los casos de prueba, las evaluaciones están contenidas en la base de datos), elegimos arbitrariamente poner una evaluación de 3, por ser éste el valor medio.

Conjunto de películas Hemos probado diferentes posibilidades para responder la última pregunta, y llegamos a la conclusión de que en la elección de éste conjunto radica el éxito o fracaso de la aplicación del algoritmo

Intersección total Primero aplicamos el algoritmo en su versión más pura, y calculamos el conjunto de todas los usuarios que hayan visto exactamente las mismas películas que el usuario cuya preferencia se quiere proyectar.

Sin embargo, con 17770 películas, parece ser que siempre hay alguna película que *A* vio, pero *B* no. La cardinalidad de prácticamente la totalidad de los conjuntos devueltos es 1 (o sea, el mismo usuario).

Películas menos vistas Reducimos pues el conjunto de posiciones sobre el cual el algoritmo tomaba la intersección. Decidimos darle más importancia a las películas menos vistas: no sólo porque el solo hecho de que dos personas hayan visto una película poco conocida revela cierta simetría; sino que las notas que les darán a éstas películas "alternativas" revela mucho más sobre su gusto que las notas puestas a películas más populares.

Perdemos sin embargo eventuales usuarios muy parecidos que no vieron una de éstas películas.

Aún tomando las 10 películas menos vistas, la cardinalidad del 84 % de los conjuntos resultantes era 1 (el propio usuario). Reducimos el conjunto a 3 películas, y aunque ahora la cantidad de conjuntos "inservibles" se reducía al 27 %, perdíamos mucha información al considerar solamente aquellos pocos usuarios que justo hayan coincidido en éstas películas.

Películas más o menos vistas Vimos que tomar las películas menos vistas era excesivamente restrictivo. Decidimos buscar un compromiso y descartar las primeras películas poco vistas, y tomar las siguientes.

A continuación, detallamos la cardinalidad de los conjuntos devueltos para el caso en que tomamos entre el 10 % y 20 % de películas menos vistas. Probamos este enfoque sobre el conjunto real de 17770, y también sobre un conjunto más reducido de 9000 películas.

cantidad películas	0	1	>1	rmse
9000	9 %	28 %	63 %	1.15502
17770	1 %	53 %	46 %	1.1951

Las búsquedas para los cuales obtuvimos un conjunto vacío son aquellas en las que el 10 % de películas es menos que 1. Para comprender mejor el impacto del alto porcentaje de las columnas 0 y 1, recordemos que para éstos *queries* proyectamos una nota por defecto de 3.

Observamos que al aumentar la cantidad de películas tenemos menos candidatos que hayan visto ése 10 % de películas.

Sin embargo, un 50 % de conjuntos vacíos (luego de filtrar al usuario buscado) sigue siendo mucho, y el *rmse* revela un comportamiento menos acertada que el enfoque trivial (calcular la media).

Unión Cansados de no encontrar suficientes usuarios sobre los cuales poder basarnos para proyectar la nota de la película, decidimos no buscar en los usuarios que hayan visto **exactamente** ésas películas, sino sobre los cuáles vieron **alguna** de ellas.

Detallamos a continuación el resultado obtenido de tomar las 10 películas menos vistas, buscar todos los usuarios que hayan visto alguna de éstas y luego promediar los 10 usuarios más parecidos:

cantidad películas	rmse
9000	0.8479
17770	0.9793

Por primera vez obtuvimos un resultado que es mejor que la media, pero aún así no nos acercamos lo suficiente al resultado obtenido por el sistema de Netflix (0.9525).

Resumiendo

- a diferencia de otros algoritmos (inclusive el propio usado por Netflix), nuestro enfoque no usa un material de entrenamiento
- obtuvimos un resultado mejor que la media

5.2.2. ADN

Investigando sobre cuestiones específicas de los perfiles genéticos, nos sorprendimos ante la falta generalizada de material bibliográfico con respecto de búsquedas de perfiles genéticos con aplicación forense o criminal.

El tema no parece ser excesivamente popular: un tema muy parecido como el del problema de filiación ha generado mucho más bibliografía y análisis. Parece ser más interesante desmascarar impostoras de la realeza rusa que encontrar violadores seriales.

Además, los sistemas ya existentes que implementan algoritmos probablemente parecidos son de código cerrado y al ser parte de bases de datos nacionales solamente obtenidos por pedidos de gobiernos.

El algoritmo aplicado a éste escenario se está usando en un sistema desarrollado por el grupo de Procesamiento de Lenguaje Natural de la FaMAF, coordinado por el Dr. Gabriel Infante-López.

5.3. Trabajo Futuro

5.3.1. Netflix

Por falta de tiempo, no experimentamos los suficientes casos como para hacer conclusiones tajantes, pero sí podemos asegurar no sólo haber encontrado un algoritmo que mejora substancialmente la media; sino también creemos poder mejorar el resultad actual.

Afirmamos esto porque:

- ignoramos las fechas en las que los usuarios evalúan una película. En particular, existe muchos usuarios que evaluaron varias veces (con notas muy disímiles) una misma película: actualmente nuestra implementación se queda con la última nota
- eligiendo bien el conjunto de películas sobre los cuales buscar, es muy posible obtener mejoras sobre la implementación actual. Para ello sin embargo, es menester optimizar el acceso a los datos para acelerar los tiempos que por ahora son excesivos.

5.3.2. ADN

También en el caso de los perfiles de ADN, una mejora substancial puede ser obtenida si hacemos uso de la naturaleza de los alelos y tratamos de buscar en los subconjuntos posibles de perfiles coincidentes (ver sección 2.5.1).

En particular, los alelos tienen asociado una *frecuencia alélica*, que es una medida de porcentaje de aparición en la naturaleza. Usando ésto, podemos detectar marcadores que contiene información más importante que otros: así, si un marcador tiene alelos de baja probabilidad, entonces si otro perfil tiene los mismos alelos en ése marcador debería encontrarse más cerca que en el caso de que los alelos sean de una frecuencia común.

Bibliografía

[Ben80] J. L. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, 23:214–229, 1980.

[Ben06] Mark Benecke. *Genetischer Fingerabdruck / DNA-Typisierung*, pages 449–454. F.A. Brockhaus, Leipzig, Alemania, 2006. version online: `http://www.benecke.com/brockhaus.html`.

[Bre97] C. H. Brenner. Symbolic kinship program. *Genetics Society of America*, 145 (2):535–542, febrero 1997.

[Cri74] Francis Crick. La doble hélice: una visión personal. *Nature*, 248:766–769, 1974.

[Cur00] Helena Curtis. *Biología*. Medica Panamericana, quinta edition, 2000.

[For05] Forensic Science Service, Gran Bretaña. *Annual Report and Accounts*, 2004-2005.

[Ham50] Richard Hamming. Error detection and error correcting codes. *American telephone and Telegraph Company*, XXVI:15, 1950.

[JWT85] Alec J. Jeffreys, Victoria Wilson, and Swee Lay Thein. Hypervariable 'minisatellite' regions in human dna. *Nature*, 314:67–73, 1985.

[Kir90] Lorne T. Kirby. *DNA Fingerprint, an Introduction*. Oxford University Press, Gran Bretaña, 1990.

[Moo65] Gordon E. Moore. Gramming more components onto integrated circuits. *Electronics*, abril 1965.

[Moo91] Andrew Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge, Gran Bretaña, 1991.

[Net07a] Netflix Inc., Los Gatos, California, EEUU. *Netflix Announces Financial Report*, april 2007.

[Net07b] Netflix Inc., Los Gatos, California, EEUU. *Netflix Press Kit*, 2007.

[SM97] João Setubal and João Meidanis. *Introduction to computational molecular biology*. PWS Publishing Company, Campinas, Brasil, 1997.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining, Practical Machine Learning Tools and Techniques*. Elsevier, segunda edition, 2005.