Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Predicción de especificidad de unión de receptores de células T mediante kmeros

Tesis de Licenciatura en Ciencias de la Computación

Maximiliano Rey

Director: Esteban Lanzarotti
Departamento de Computación - UBA, 2021

Índice general

1. ABSTRACT

El presente proyecto propone un modelo algorítmico para identificar el antígeno reconocido por el receptor de una célula T, llamado TCR, usando solamente la secuencia de aminoácidos de este último. El sistema propuesto utiliza una base de datos con información de TCRs conocidos y los respectivos antígenos identificados por ellos. A partir de las secuencias de estos TCRs, el modelo genera todas las subsecuencias de un determinado largo y luego aplica la estadística buscando aquellas subsecuencias relevantes para que el reconocimiento suceda.

El sistema recibe un TCR del cual se quiere saber qué antígeno reconoce. Al igual que en el paso anterior, se calculan todas las subsecuencias posibles de largo fijo a partir de su secuencia. Finalmente, se utilizan las estadísticas calculadas anteriormente para determinar a qué antígeno es más afín el TCR.

En esta tesis se intenta buscar cuál es el largo óptimo para las subsecuencias analizadas, en dos cadenas distintas que integran el TCR. Se probó el sistema en TCRs de ratones y de humanos y el largo óptimo resultó ser el mismo en ambas especies.

Las predicciones van acompañadas con un índice de especificidad, el mismo muestra qué tan probable es que el TCR de entrada sea afín al antígeno predicho y no lo sea hacia otros antígenos.

En este proyecto no sólo se analizó la cadena beta de los TCRs, como en otros estudios de bioinformática, sino que también se incluyó la cadena alfa. Si bien esto limitó los datos que pudimos obtener para entrenar y testear el algoritmo, ya que existen menos TCRs de los cuales se conocen ambas cadenas, los resultados fueron cercanos al 70 %.

Además, la especificidad logró ser una buena medida para descartar predicciones erróneas. En ambas especies existe un valor de especificidad relativamente bajo en el cual el coeficiente de Matthews es alto al evaluar si una predicción es correcta o no, utilizando dicho valor como punto de corte.

El método analizado en la tesis es aplicado a la unión TCR-antígeno pero el problema que resuelve es más amplio. Este sistema podría adaptarse a otros casos de interacción entre proteínas.

2. AGRADECIMIENTOS

A Esteban, mi director de tesis, por su paciencia y por el tiempo que ha dedicado para guiarme en mi proyecto. A los docentes de la facultad, por brindarme los conocimientos que necesité para llegar a esta instancia. A mi familia, que siempre fomentó en mí el interés por lo académico.

3. INTRODUCCIÓN

En 1953 Francis Crick y James Watson propusieron por primera vez la estructura de la doble hélice del ADN [29]. Hoy, a poco más de 60 años tenemos un conocimiento mucho más profundo sobre el funcionamiento de estas secuencias de nucleótidos dentro de las células. Los avances sobre esta área han permitido alterar bacterias y hongos para transformarlos en fábricas biológicas de productos necesarios en la medicina, como la insulina y otras hormonas [10]. Dada la naturaleza del ADN dentro de la maquinaria celular como forma de codificar información, no es de extrañar que se empiece a utilizar la informática para intentar descubrir patrones en el mismo. En los segmentos del ADN que codifican proteínas, cada conjunto de tres nucleótidos codifica un aminoácido en particular, formando cadenas de aminoácidos que luego se utilizan para ensamblar proteínas.

La humanidad lleva siglos estudiando cómo combatir enfermedades y, sin conocerlo a fondo, logró influenciar el sistema inmune para fortalecerlo contra determinadas enfermedades. La palabra vacuna deriva de la viruela de las vacas, llamada también viruela vacuna; esta cepa de la enfermedad era relativamente benigna, y aquellos que pasaban por esta enfermedad tenían más probabilidades de no ser infectados por cualquier cepa de viruela en el futuro [23]. En ese momento no lo sabían, pero ambas cepas de viruela tenían una determinada semejanza, que provocaba una respuesta del sistema inmune. Hoy en día, las vacunas siguen la misma idea, inoculan un organismo 'semejante' al patógeno (o parte del mismo), pero que no posee la capacidad de dañar al anfitrión. Al ser un cuerpo extraño, el sistema inmune desarrolla defensas contra él, las cuales identificarán también al patógeno real. Cualquier cuerpo que desata una respuesta del sistema inmune se denomina 'antígeno'. El antígeno posee grupos de proteínas que son reconocidos como extraños por el sistema inmune y se convierten en objetivos a eliminar por el mismo. Tanto en el antígeno como en el anticuerpo que lo detecta existen regiones de interacción que son vitales para el reconocimiento entre ambos complejos.

La hipótesis en el presente proyecto es que es posible identificar cuál es el antígeno reconocido por un determinado receptor de célula T, basándose solamente en la secuencia de este último, con un determinado margen de error. El método desarrollado se basa en calcular estadísticas sobre fragmentos de la secuencia de un sector del receptor TCR (T cell receptor) llamado CDR3, utilizando una base de datos conocida, que incluye TCRs que reconocen múltiples antígenos. Luego se utilizan dichas estadísticas para chequear si un TCR desconocido sigue el patrón de los TCRs que reconocen a un antígeno específico. Para poder lograr esto definiremos un 'puntaje' de 'especificidad' que permita distinguir a aquellos TCRs que unan específicamente a un antígeno de aquellos que interactúan inespecifícamente con varios antígenos, o bien no interactúan con ninguno.

3.1. Sistema inmune

Todos los organismos vivos pueden sufrir ataques de patógenos. Incluso las formas de vida simples como bacterias pueden ser el blanco de ataques de virus.[3] Los organismos multicelulares son más complejos. Las células de un mismo organismo interactúan entre sí formando un espacio interno en el cual mantienen un ambiente controlado. Por su complejidad existen más procesos producidos por agentes externos que pueden poner en peligro

la integridad de todo el organismo. Por eso es que en estas formas de vida existen diversos mecanismos que identifican y eliminan agentes externos, que en su conjunto integran el sistema inmune [1].

Los mecanismos del sistema inmune se pueden agrupar principalmente en dos grupos: inmunidad innata e inmunidad adaptativa [1].En esta tesis nos vamos a centrar sólo en esta última. Este sistema provee una respuesta más lenta pero a la vez más efectiva contra el patógeno. Las células producidas por esta línea de defensa, llamadas "linfocitos", tienen como objetivo atacar a un antígeno específico. Además, como explicaremos más adelante, existen subtipos de esas células que se mantienen vivas por décadas y pueden hacer frente al patógeno en forma más rápida si aparece nuevamente en un futuro. Al reconocer al patógeno, proliferan células que también lo reconocen. El objetivo de las vacunas es producir la generación de este tipo de células. Cuando se diseñan vacunas, se utilizan antígenos que no supongan una amenaza real contra el cuerpo, pero que formen células de memoria que reconozcan también al patógeno contra el cual se quiere provocar inmunidad. La medicina está en busca de vacunas más eficientes, antígenos más universales cuya probabilidad de aparecer en varias cepas sea alta, y por otro lado más simples, que sólo tengan en común con el patógeno la región del antígeno que es reconocida por las células T que se formarán, para que no representen una amenaza para el huésped. Secuenciar todos los antígenos posibles es una tarea difícil, por eso se intenta estimar probabilidades sobre dichos antígenos, para buscar los mejores candidatos.

3.1.1. MHC

Los MHC, abreviatura de Major Histocompatibility Complex, son proteínas que se encuentran en la membrana celular de algunas células nucleadas y su función es ligarse a determinados antígenos y presentarlos en la membrana celular para que las células T puedan reconocerlos [1].Los MHC más estudiados son los de clase I y los de clase II. Todas las células nucleadas del organismo humano expresan MHC de clase I, cuya función radica en exponer fragmentos de proteínas generadas en el interior de la célula al exterior donde el sistema inmune puede detectar si la célula está produciendo secuencias que son foráneas al organismo. En una célula sana los MHC de clase I sólo aparecerán unidos a secuencias que se sintetizan en las células normalmente. Sin embargo, en una célula cancerígena, la cual sufre una mutación muy grave, es irremediable que se generen proteínas anómalas. Otro caso es una célula que ha sido infectada por un virus, la cual está produciendo las proteínas necesarias para sintetizar más virus. En estos casos, algunos segmentos de estas proteínas extrañas quedarán ligadas al complejo MHC y serán expuestas en la membrana celular, donde podrán ser detectadas por el sistema inmune.

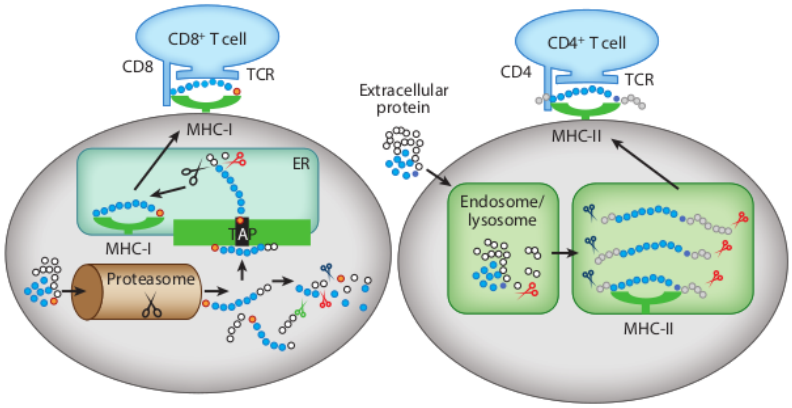

Fig. 3.1: Representación del camino metabólico de presentación y reconocimiento de péptido. En la figura de la izquierda, el MHC es de clase I, el cual se liga a parte de una proteína que fue expresada por la célula. Este complejo es reconocido por el receptor de una célula T citotóxica. En la figura de la derecha, se trata de un MHC de clase II, el cual se liga a parte de una proteína procedente del exterior de la célula. Dicho complejo es reconocido por el receptor de una célula T auxiliar.

Ahora bien ¿Qué ocurre si debido a una mutación, natural o producida por un virus, el MHC de clase I no se expresara? Dentro del sistema inmune innato existen células llamadas Natural Killer, abreviado NK, que destruyen toda célula que no posea el MHC de clase I específico del organismo [30].

La mayoría de las células que expresan el MHC de clase II se denominan presentadoras de antígenos. Las mismas se encargan de fagocitar antígenos, digerirlos en su interior y exponer fragmentos de sus proteínas en su membrana celular unidos a sus MHC de clase II. Existe un tipo de células llamada 'célula dendrítica' que cumple el rol de 'presentadora de antígenos profesional'. Su principal función radica en colaborar con aquellas células T que reconocen al antígeno para indicarles que deben comenzar a dividirse. Para eso, se dedican a fagocitar cuerpos extraños, cuyos péptidos se ligan a los MHC de clase II en el interior de la célula para ser expuestos luego en la membrana celular [1].

Existe un tercer tipo de MHC, pero su función está menos estudiada, y a diferencia de las otras dos no unen a péptidos.

3.1.2. Células T

Las células T son un tipo de linfocitos, que se desarrollan en el timo (de ahí su nombre). Poseen unidos a su membrana celular receptores denominados TCR cuya función es unirse a un MHC-péptido específico. Se diferencia en tres subtipos: naturales (o vírgenes), efectoras y de memoria. Las células T naturales son las predecesoras de las otras dos. Antes de madurar, estas células presentan enzimas que alteran al azar la región de la secuencia de ADN que codifica los TCR, con lo cual se forman millones de células T, cada una con receptores diferentes. Luego migran al timo, donde atraviesan una etapa de selección en la que se descartan aquellas que reconocen antígenos del propio cuerpo, para evitar que las mismas ataquen células del propio cuerpo identificándolas como patógenos, generando

una enfermedad autoinmune [1].Las células T naturales maduras se alojan en los ganglios en espera de la aparición de un antígeno que sea reconocido por su TCR. La gran mayoría de ellas no harán nada, pero cuando alguna, por azar, reconoce a un antígeno, se divide generando células T efectoras y de memoria con el mismo receptor, por lo cual reconocerán al mismo antígeno. Las efectoras son las células que realmente van a combatir la infección. Las de memoria son células que pueden permanecer en el organismo por varias décadas y, al reconocer al antígeno, se dividen formando nuevas células de memoria y también efectoras. Las células T de memoria cumplen el papel de proteger al organismo de futuras invasiones del mismo patógeno: ante una segunda aparición del patógeno existen miles de células que procrearán, generando una inundación de linfocitos específicos contra un tipo de antígeno [1].

Dentro de las células T existen varios subtipos, los cuales poseen sus propias versiones de células efectoras y de memoria, pero principalmente se destacan dos subtipos: las citotóxicas y las auxiliares.

Las células T citotóxicas son linfocitos encargados de la eliminación de células propias que han sufrido alteraciones, ya sea por la infección de un virus o por una mutación grave. Éstas actúan identificando péptidos foráneos presentados por los MHC de clase I de las células infectadas. Cuando una célula citotóxica efectora reconoce el complejo MHC-Péptido libera proteínas que perforan la membrana del objetivo, destruyen sus organelas y desensamblan su ADN [1].

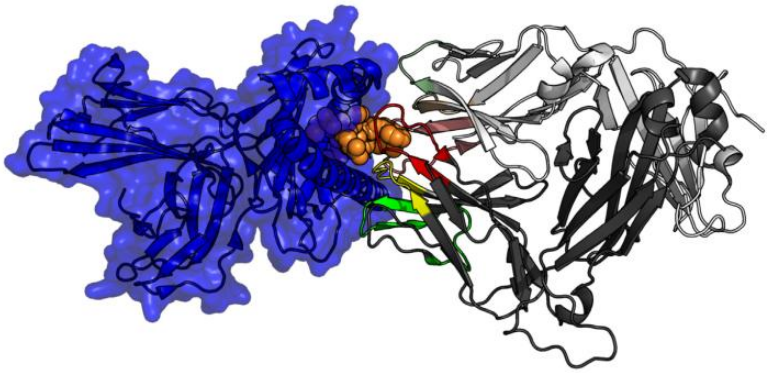

Fig. 3.2: Representación del TCR, interactuando con el complejo MHC(azul) - péptido (naranja). En el mismo se encuentran diferenciadas las regiones CDR1 (amarillo), CDR2 (verde) y CDR3 (rojo)

Las células T auxiliares interactúan con el complejo MHC de clase II ligado al antígeno. Cuando una célula T auxiliar virgen reconoce un complejo MHC-péptido de una célula dendrítica, desencadena un intercambio de señales químicas. Si la señal no es respondida por la célula dendrítica, la célula asume que es una respuesta autoinmune y ya no reconocerá a dicho antígeno, caso contrario comienza a dividirse y a diferenciarse.

Cuando una célula auxiliar T efectora reconoce el complejo MHCII-péptido en una célula presentadora de antígenos libera señales químicas que provocan que otras células responsables del sistema inmune, como los macrófagos, se vuelvan más activas.

Además, las células T auxiliares son las responsables de activar a las células B. A diferencia de las T, cuando una célula B virgen interactúa con su antígeno correspondiente no comienza a procrear, sino que fagocita al antígeno, lo digiere y lo expone en su

complejo MHC de clase II. Cuando una célula T auxiliar efectora reconoce el complejo MHCII-péptido, envía la señal química que provoca que la célula B comience a dividirse y diferenciarse en células efectoras y de memoria.

El TCR está conformado por dos secuencias: alfa y beta, las cuales componen el anticuerpo. Si bien la última es la que más interactúa en la unión con el antígeno, ambas secuencias aportan información. Puntualmente, existe una región compuesta por los primeros aminoácidos de ambas cadenas, con una longitud variable entre 100 y 120 aproximadamente. En la misma se encuentran tres regiones llamadas CDR (abreviatura de *Complementarity-determining region*) las cuales conforman el sitio activo que reconoce al antígeno. Como existen tres regiones en cada cadena, en total existen seis CDR: $CDR1_A$, $CDR1_B$, $CDR2_A$, $CDR2_B$, $CDR3_A$, $CDR3_B$. Por su importancia en la interacción con el antígeno, otros proyectos de bioinformática se han concentrado en trabajar específicamente con los CDR [17] [7]. Ambos CDR3, principalmente el que se encuentra en la cadena beta, interactúan directamente con el péptido presentado por el MHC que el TCR reconoce. Además, en el $CDR3_B$ es donde ocurre la recombinación de genes en las células T naturales. El hecho de que la recombinación se dé en la cadena más importante para la interacción con el péptido del antígeno reconocido no es casualidad. La capacidad del sistema inmune de desarrollar anticuerpos contra nuevos antígenos radica en que existan una gran variedad de células que expresan anticuerpos que reconocen a péptidos distintos.

Ahora bien, ¿se puede determinar a qué péptido foráneo une un TCR analizando sólo la secuencia del $CDR3_A$ y $CDR3_B$?. Si la secuencia analizada cambia ligeramente, ¿conservará su capacidad de reconocer al mismo antígeno? El problema que surge es: ¿cómo determinamos que dos secuencias son parecidas? ¿cómo definimos una noción de 'distancia' entre secuencias?

3.2. Comparación de secuencias biológicas

El descubrimiento de que secuencias biológicas similares suelen tener funciones similares, dio lugar a una variedad de metodologías computacionales que analizan la similaridad entre dos secuencias biológicas dadas, de las cuales la más conocida es la familia de algoritmos BLAST [5]. Estas herramientas permiten comparar y deducir si su función biológica es similar, o si no tienen ningún parecido. Con el paso del tiempo, también se propusieron metodologías que permitieron calcular el grado de similitud entre una secuencia dada y un conjunto de secuencias parecidas entre sí (por ejemplo: PSI-BLAST y HMMER[6]). Esto hizo posible, más adelante, la aplicación de estos métodos en *pipelines* que caracterizan funcionalidades biológicas a gran escala, con las que fue posible llevar adelante proyectos, como por ejemplo, el del genoma humano.

Establecer una 'similaridad' entre dos secuecias no es algo trivial, y existen múltiples heurísticas para implementarla. La heurística más utilizada es la llamada alineamiento, la cual cuenta la cantidad de 'cambios' necesarios para pasar de una secuencia a otra, los cuales pueden ser reemplazar aminoácidos o insertar secuencias [4]. Pero si se quieren resultados óptimos sus parámetros deben ajustarse a cada caso, el cambio de un aminoácido por otro no tiene el mismo peso que una inserción, y una inserción de una secuencia consecutiva no es equivalente a insertar varios aminoácidos en posiciones separadas, y esos 'pesos' cambian de acuerdo al problema. Además comparar dos secuencias tiene una complejidad temporal de $O(|s1||s2|)$, lo cual es alto si se quieren comparar muchos pares de secuencias.

3.3. Kmeros

Como alternativa al alineamiento, se utiliza la comparación por kmeros, que consiste en separar la cadena en subsecuencias de largo fijo que se superponen entre sí, las cuales reciben el nombre de 'kmeros' [21].

De esta forma, si dos secuencias son similares, esto implicará que poseen kmeros en común. Entonces, para medir la 'similaridad' entre las secuencias se cuentan los kmeros presentes en ambas.

El largo que poseen estos kmeros es algo que debe ajustarse a cada problema. Por un lado si los kmeros son demasiado cortos, la similaridad entre dos secuencias al azar será medida como muy alta, ya que la cantidad de potenciales subsecuencias es menor, con lo que la probabilidad de que un determinado kmero aparezca en ambas aumentará. Si los kmeros son demasiado largos, la probabilidad de que un kmero sea compartido entre dos secuencias será más baja, aunque las secuencias en sí sean similares, con lo cual la similaridad entre cualquier par de secuencias será medida como demasiado baja. En ambos extremos la noción de similaridad entre secuencias se desdibuja.

La comparación por kmeros, puede, en realidad, aplicarse a cualquier cadena de caracteres y establecer una similaridad entre cualquier par de palabras. Por ejemplo, dada la palabra antígeno y una longitud 5, sus kmeros son: ANTIG NTIGE TIGEN IGENO

Dadas las palabras HABLADO, HALLADO, CANTADO y un k = 2, tenemos que sus respectivos kmeros son:

HABLADO: HA AB BL LA AD DO
HALLADO: HA AL LL LA AD DO
CANTADO: CA AN NT TA AD DO

Podemos establecer una noción de similaridad entre estas palabras utilizando esta comparación por kmeros. HABLADO y HALLADO serán 'mas similares' que HABLADO y CANTADO ya que las primeras comparten cuatro kmeros en común mientras el segundo par comparten dos.

3.4. Base de datos de TCRs: VDJdb

Para el desarrollo de la tesis necesitamos obtener datos conocidos de uniones de TCRs con antígenos. Afortunadamente, múltiples laboratorios biológicos han identificado algunos antígenos reconocidos por determinados TCRs [27] [12]. VDJdb [28] es un proyecto que recopila y estandariza información sobre células T a partir de varias bases de datos. Puntualmente, presentan las secuencias del TCR, señalando sus respectivas subsecuencias en CDR1, CDR2 y CDR3, junto al complejo reconocido por el mismo. El mismo consiste del MHC, identificado por un nombre, y de la secuencia del péptido foráneo.

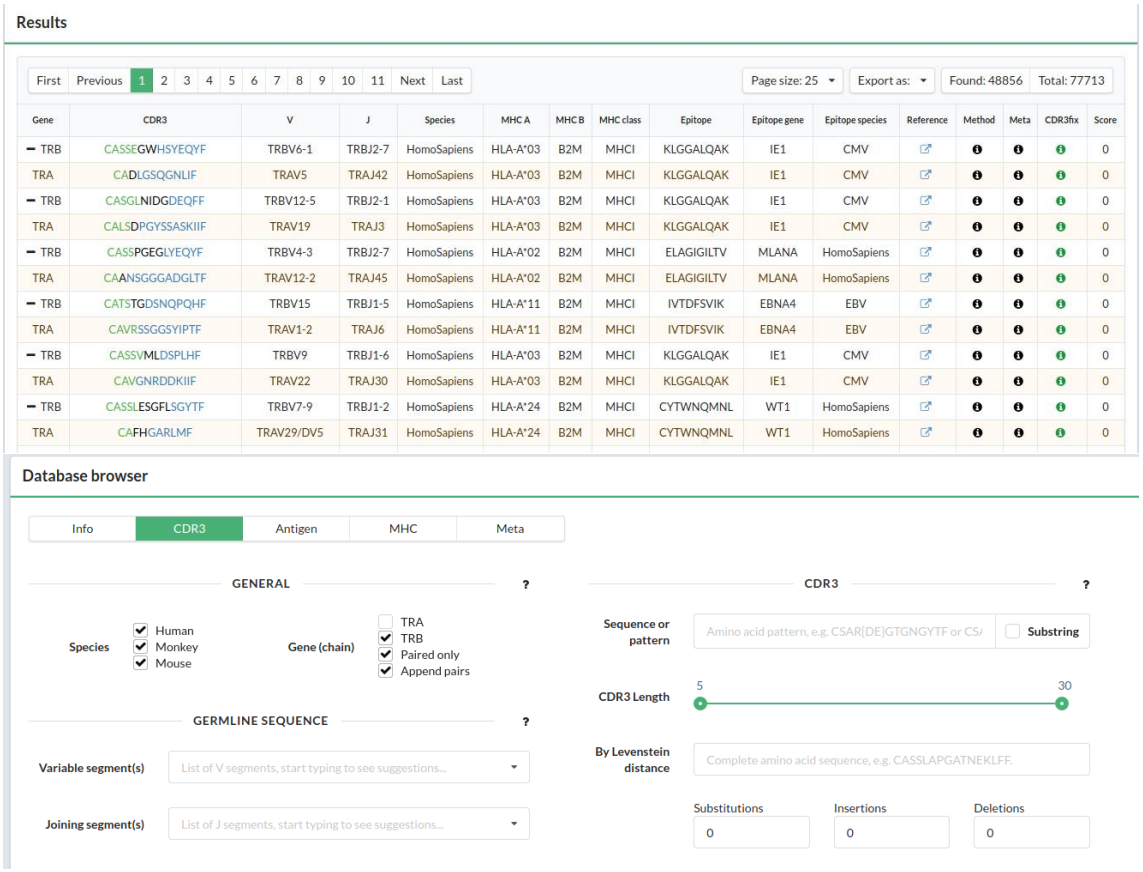

Fig. 3.3: Arriba, captura de la tabla de búsqueda de VDJdb. Abajo, filtros aplicables en la búsqueda. En este caso, buscamos sólo aquellos TCRs de los que se conoce la cadena alfa y beta

Las especies de las cuales la base de datos posee información son humano, ratón y mono. Este último no se utilizó para nuestro proyecto ya que no existen datos disponibles para esta especie con la cadena alfa y beta.

Como se ve en la figura 3.3, se puede filtrar TCRs que reconocen antígenos de determinada enfermedad, e incluso puede introducirse parte de una secuencia para listar todos los TCRs cuyo antígeno incluye dicha secuencia. Por otro lado se puede seleccionar a qué subtipo de MHC reconocen los TCRs mostrados y si se filtran sólo los que reconocen MHC de clase I, de clase II o ambos.

Por último se encuentran filtros que seleccionan TCRs de acuerdo al CDR3, los cuales fueron utilizados en nuestro trabajo. En este caso se puede elegir la cadena alfa, la beta, o filtrar aquellos de los cuales se conocen ambas cadenas. Esta opción es importante en nuestro trabajo ya que requerimos ambas cadenas para nuestro algoritmo. Adicionalmente, existen filtros para el largo del CDR3 (existen registros desde 5 hasta 30 aminoácidos), se puede buscar por secuencia, e incluso se pueden filtrar aquellos cuya distancia de Levenshtein (este concepto se explicará más adelante) respecto a una secuencia introducida es menor a cierto valor.

3.5. Inferencia de probabilidad condicional

En el reconocimiento del antígeno por parte del TCR existen más factores que los que están registrados en la base de datos. Por lo tanto, no podremos demostrar que un TCR se una a un antígeno con seguridad, pero sí inferir una probabilidad de que eso suceda teniendo en cuenta ciertas características en su cadena. Puntualmente, el patrón que buscamos reconocer es la presencia de determinado kmero en la cadena, ya que, como dijimos anteriormente, en la secuencia de un TCR existen kmeros que son importantes para la unión con el antígeno. Este enfoque fue utilizado también en otros proyectos que platean objetivos similares [19] [26].

Se define como probabilidad condicional P(A|B) como la probabilidad de que un evento A suceda sabiendo que un evento B sucede. Esto no pone ninguna restricción de causalidad o temporalidad entre los eventos A y B. Además, $P(A|B) = \frac{P(A \cap B)}{P(B)}$.Por otro lado, definimos un experimento en donde el resultado puede ser positivo o negativo y sea P(X) la probabilidad de que el resultado de dicho experimento sea positivo, se dice que la variable aleatoria X sigue la distribución de Bernoulli. En este caso, dada la probabilidad $P(X) = p$, entonces ocurre que la esperanza es E(X) = p y la varianza es $V(X) = p(1-p)$.

Cuando no se conoce dicha probabilidad se puede realizar una estimación a partir de datos conocidos. La ley de los grandes números es la herramienta matemática que nos permite demostrar que la inferencia funciona. La misma dicta que, sean $X_1 ... X_n$ variables aleatorias idénticamente distribuidas, tal que $\forall_i P(X_i) = p$, $\forall_i E(X_i) = u$ y $\forall_i V(X_i) = o^2$, entonces $\sum_{i=0}^{n} X_i$, $E(\overline{X}) = u$ y $V(\overline{X}) = o^2/n$, siendo $\overline{X} = 1/n$. Por lo tanto, si n es suficientemente grande entonces $V(\overline{X})$ tiende a 0, con lo cual $\overline{X}$ tiende a $E(\overline{X}) = p$. De esa forma, podemos estimar p con $\overline{X}$ si el número de variables es lo suficientemente grande. Ahora bien, sean A y B dos variables aleatorias que siguen la distribución de Bernoulli, $A \cap B$ es la probabilidad de que tanto A como B se cumplan. Por definición, esto es también un experimento que puede ser verdadero o falso, con lo cual $C = A \cap B$ es una variable aleatoria que sigue la distribución de Bernoulli.

Dado un TCR, podemos considerar entonces R como una variable aleatoria que indica si el TCR une o no a un determinado antígeno y K una variable aleatoria que indica que el TCR posee un determinado kmero en su cadena. Ambas son variables aleatorias que siguen la distribución de Bernoulli, ya que un TCR puede unir o no a un antígeno y puede contener o no el kmero. Nuestro objetivo en este proyecto es calcular, P(R | K), es decir que detectando un determinado kmero en la cadena, podamos estimar una probabilidad de que reconozca a un antígeno determinado.

En este caso, podemos definir variables aleatorias a partir de cada TCR de la base de datos utilizada para entrenar. Definimos K_T como una variable que es verdadera si el TCR T contiene al kmero. Definimos además R_T, que es verdadera si el mismo T reconoce al antígeno. Además, la variable RK_T indica si un TCR reconoce al antígeno y a la vez contiene al kmero. Entonces:

$$\overline{K} = (K_1 + ... + K_n)/n$$
$$\overline{K_T \cap R_T} = (RK_1 + ... + RK_n)/n.$$

Utilizando la ley de los grandes números, podemos estimar:

$$P(R|K) = \frac{P(K \cap R)}{P(K)}$$

Utilizando la definición de la inferencia, podemos estimar $P(K \cap R)$ como $\overline{RK}$ y P(K) con $\overline{K}$. Entonces reescribimos la fórmula como:

$$P(R|K) = \frac{\overline{RK}}{\overline{K}}$$

Por la definición mencionada, equivale a:

$$P(R|K) = \frac{\sum_{i=0}^{n} RK_i/n}{\sum_{i=0}^{n} K_i/n}$$

Simplificando cada término:

$$P(R|K) = \frac{\sum_{i=0}^{n} RK_i}{\sum_{i=0}^{n} K_i}$$

En resumen, la probabilidad de que un TCR reconozca un antígeno sabiendo que posee un kmero se estima como la cantidad de TCRs que poseen el kmero y reconocen al antígeno dividido por la cantidad de TCRs que contienen el kmero. Este proceso se repite para cada par kmero-antígeno.

3.5.1. Ejemplo

Daremos a modo explicativo un seguimiento del algoritmo. Tomamos, como grupo de entrenamiento, las siguientes secuencias que corresponden a $CDR3_B$ de un TCR, de los cuales sabemos (por los datos obtenidos de VDJdb) a qué péptido foráneo reconoce. Para este caso sólo trabajaremos con dos antígenos.

CAGPQGRETQYF que reconoce a **YVLDHLIVV**
CASSQGFGGGETQYF que reconoce a **YVLDHLIVV**
CASSELNTGELFF que reconoce a **YLQPRTFLL**
CASTDLNTGELFF que reconoce a **YLQPRTFLL**

Sus kmeros con k = 3 son:

CAGPQGRETQYF:
CAG AGP GPQ PQG QGR GRE RET ETQ TQY QYF
CASSQGFGGGETQYF:
CAS ASS SSQ SQG QGF GFG FGG GGG GGE GET ETQ TQY QYF
CASSELNTGELFF:
CAS ASS SSE SEL ELN LNT NTG TGE GEL ELF LFF
CASTDLNTGELFF:
CAS AST STD TDL DLN LNT NTG TGE GEL ELF LFF

Cabe aclarar que no importa la posición del kmero dentro de la secuencia original, sólo contamos la presencia o ausencia del kmero. Si un kmero se repite múltiples veces en la secuencia, se cuenta una sola vez. Esto lo hacemos porque asumimos que solamente existe una región en donde el $CDR3_B$ interactúa con el péptido, entonces consideramos que repetir una subsecuencia no aumenta la probabilidad de que reconozca el antígeno.

A continuación, calculamos la probabilidad condicional de que un TCR reconozca un antígeno sabiendo que posee un kmero: P(R|K). Por simplicidad, llamaremos 1 al antígeno YVLDHLIVV y 2 al antígeno YLQPRTFLL.

P(1 | CAG) = 1	P(1 | GGG) = 1	P(2 | AGP) = 0	P(2 | GGE) = 0
P(1 | AGP) = 1	P(1 | GGE) = 1	P(2 | GPQ) = 0	P(2 | GET) = 0
P(1 | GPQ) = 1	P(1 | GET) = 1	P(2 | PQG) = 0	P(2 | SSE) = 1
P(1 | PQG) = 1	P(1 | SSE) = 0	P(2 | QGR) = 0	P(2 | SEL) = 1
P(1 | QGR) = 1	P(1 | SEL) = 0	P(2 | GRE) = 0	P(2 | ELN) = 1
P(1 | GRE) = 1	P(1 | ELN) = 0	P(2 | RET) = 0	P(2 | LNT) = 1
P(1 | RET) = 1	P(1 | LNT) = 0	P(2 | ETQ) = 0	P(2 | NTG) = 1
P(1 | ETQ) = 1	P(1 | NTG) = 0	P(2 | TQY) = 0	P(2 | TGE) = 1
P(1 | TQY) = 1	P(1 | TGE) = 0	P(2 | QYF) = 0	P(2 | GEL) = 1
P(1 | QYF) = 1	P(1 | GEL) = 0	P(2 | CAS) = 2/3	P(2 | ELF) = 1
P(1 | CAS) = 1/3	P(1 | ELF) = 0	P(2 | ASS) = 1/2	P(2 | LFF) = 1
P(1 | ASS) = 1/2	P(1 | LFF) = 0	P(2 | SSQ) = 0	P(2 | AST) = 1
P(1 | SSQ) = 1	P(1 | AST) = 0	P(2 | SQG) = 0	P(2 | STD) = 1
P(1 | SQG) = 1	P(1 | STD) = 0	P(2 | QGF) = 0	P(2 | TDL) = 1
P(1 | QGF) = 1	P(1 | TDL) = 0	P(2 | GFG) = 0	P(2 | DLN) = 1
P(1 | GFG) = 1	P(1 | DLN) = 0	P(2 | FGG) = 0	
P(1 | FGG) = 1	P(2 | CAG) = 0	P(2 | GGG) = 0	

De esta manera concluye la etapa de 'entrenamiento' del algoritmo, hasta ahora hemos recopilado estadísticas sobre los kmeros en el conjunto de *train*. Luego pasamos a la etapa de predicción, donde utilizamos dichas estadísticas de los kmeros del TCR a predecir.

En este ejemplo, intentaremos predecir a qué antígeno une el CDR3 CASTVQGRETQYF cuyos kmeros son:
CAS AST STV TVQ VQG QGR GRE RET ETQ TQY QYF

Para esto, seleccionamos los P(R|K) obtenidos en la etapa de entrenamiento, que corresponden a los kmeros de CASTVQGRETQYF, primero asumiendo que reconoce al antígeno 1 y luego asumiendo que reconoce al antígeno 2.

$P(1\|CAS) = 1/3$	$P(2\|CAS) = 2/3$
$P(1\|AST) = 0$	$P(2\|AST) = 1$
$P(1\|QGR) = 1$	$P(2\|QGR) = 0$
$P(1\|GRE) = 1$	$P(2\|GRE) = 0$
$P(1\|RET) = 1$	$P(2\|RET) = 0$
$P(1\|ETQ) = 1$	$P(2\|ETQ) = 0$
$P(1\|TQY) = 1$	$P(2\|TQY) = 0$
$P(1\|QYF) = 1$	$P(2\|QYF) = 0$

Los datos obtenidos muestran que si se analiza un sólo kmero, los resultados en la predicción son bastante inexactos. Por ejemplo, para ambos antígenos existen kmeros en donde P(R|K) = 1. Entonces, necesitamos una forma de utilizar la información de todos los kmeros del TCR en la predicción.

En la tesis exploraremos formas de calcular un 'puntaje' a partir de estas estimaciones para así obtener un resultado más exacto al intentar predecir. Ahora bien ¿qué pasa si dos antígenos tienen un mismo puntaje para un TCR? ¿Podemos decir que el TCR une específicamente a uno de los dos antígenos? ¿Y qué ocurre si el puntaje no es igual pero es muy parecido? Ese será el objetivo de esta tesis, construir un puntaje de especificidad de unión para identificar los TCRs que unen específicamente a un antígeno y a su vez no unen a otros.

3.6. Detección de TCRs específicos

Desde la base de datos, filtramos aquellos TCRs de los cuales se conoce las cadenas alfa y beta y cuál es el antígeno que reconocen. Interpretamos esa información como una relación uno a muchos, en donde un antígeno está relacionado con muchos TCRs (con sus cadenas alfa y beta). Asumimos en este modelo que un TCR no reconoce a más de un antígeno. Esto puede no ser cierto en la realidad, pero estos casos suelen ser muy raros. Desarrollaremos un algoritmo capaz de estimar la especificidad de un determinado anticuerpo desconocido hacia un complejo MHC - péptido, utilizando como entrada un conjunto de relaciones entre antígenos y TCRs ya conocidos.

Y entonces ¿cuál sería la ventaja práctica? Reconocer la especificidad de unión del TCR, es decir, a qué antígeno reacciona un TCR, puede utilizarse para encontrar y sintetizar antígenos que pueden servir para diseñar vacunas contra una enfermedad determinada, es decir, que exista una mayor probabilidad de que el antígeno sea reconocido por el sistema inmune. Sabemos que en las interacciones entre proteínas, existen regiones cortas, que cuentan con pocos aminoácidos, que son clave para el reconocimiento. La interacción entre el CDR3 y el péptido del antígeno no es una excepción, en este caso se da en secuencias de menos de diez aminoácidos. Por eso, la utilización de kmeros para el reconocimiento de la unión CDR3 - péptido es una opción natural. El proceso que el algoritmo utiliza para captar patrones específicos se asemeja al proceso que permite al CDR3 interactuar con el péptido del antígeno de forma específica.

Dado que existen múltiples variantes de TCRs que reconocen a un mismo antígeno el problema es más complejo que aplicar una noción de similitud conocida. Varios grupos de investigadores han encarado el problema con distintos enfoques, ya sea analizando secuencias o intentando modelar los TCRs y sus respectivos complejos MHC-péptido [16] [20] [18] [11], pero aún no se desarrollaron herramientas que se consideren óptimas en cualquier circunstancia. Nuestro algoritmo utiliza como entrada los CDR3 del TCR, según estudios anteriores, el CDR3 es la principal región del TCR que interactúa con el péptido que une al MHC [25]. El CDR1 tiene una menor participación y el CDR2 reconoce solamente el MHC. Asumimos en esta tesis que podemos identificar al péptido foráneo reconocido utilizando únicamente los CDR3. Analizaremos en este trabajo si un algoritmo basado en estas ideas realmente puede captar los patrones específicos para cada conjunto de TCRs que reconoce a un antígeno en particular. Con lo cual, si se dispone de un conjunto de datos razonablemente grande, puede transformarse en una herramienta útil.

4. DESARROLLO

4.1. Definición del problema

El problema planteado consiste en predecir la unión entre dos clases de proteínas. Dados dos conjuntos de proteínas A y B, asumimos que para cada proteína del conjunto A pueden existir múltiples proteínas del conjunto B que interactúan con la misma. Todas las proteínas registradas del conjunto A poseen al menos una proteína del conjunto B que interactúa ella, pero cada proteína del conjunto B interactúa sólo con una proteína del conjunto A. En el caso de la unión TCR-péptido la cardinalidad del conjunto B (los TCR) es mucho más alta. Esto es así porque existen varios TCRs que reconocen a un mismo antígeno, y a su vez, en la mayoría de los casos, un TCR reconoce a un único antígeno (esto último puede no ser cierto, pero para fines del presente trabajo asumimos que siempre se cumple). Nos concentramos en analizar las secuencias del conjunto B y las proteínas del conjunto A son identificadas con una etiqueta ($LABEL$), la cual no tiene significado extra para el problema.

Formalmente, la entrada (*input*) consiste en dos parámetros. El primer parámetro representa la proteína de la cual se intenta predecir a qué otra proteína puede unir, esta proteína debe ser del conjunto B. El segundo parámetro es una relación entre un conjunto de secuencias de proteínas del conjunto B y otro conjunto de proteínas del conjunto A representadas por etiquetas, por ejemplo:

$$\begin{aligned}
&(\text{Proteína}_1, \quad S_1 = RKEVEQDPGPFNVPEGATV...)\\
&(\text{Proteína}_2, \quad S_2 = GAGVSQSPRYKVTKRGQDV...)\\
&(\text{Proteína}_2, \quad S_3 = DAKTTQPNSMESNEEEPVH...)\\
&(\text{Proteína}_1, \quad S_4 = QKEVEQNSGPLSVPEGAIA...)
\end{aligned}$$

Donde Proteína_1 y Proteína_2 representan proteínas distintas del conjunto A.

Para resolver el problema se calcula un puntaje de especificidad de la secuencia introducida para cada proteína del conjunto A. La salida (*output*) del problema es la lista de proteínas del conjunto A que aparecen en el conjunto de entrenamiento, con el puntaje correspondiente a la proteína a predecir.

$$\begin{aligned}
P(Proteina_1|DGGITQSPKYLFRKEGQNVTLS...) &= 10^{-16}\\
P(Proteina_2|DGGITQSPKYLFRKEGQNVTLS...) &= 10^{-120}\\
&\dots
\end{aligned}$$

Como utilizamos herramientas de estadística para resolver este problema, precisamos de una base de datos que describa un conjunto de interacciones para poder estimar las probabilidades para nuevas secuencias. Se debe tener en cuenta que no se utilizan propiedades conocidas acerca de las secuencias, sino que se pretende detectarlas de la base de datos, con lo cual la precisión del algoritmo puede variar con el tamaño y la diversidad de la base de datos. En nuestro caso, descartamos aquellos antígenos que son reconocidos por menos de 70 TCRs, ya que consideramos que de no filtrarlos, al generarse los conjuntos de *test-train* (ver sección 4.3) los TCRs que queden en los conjuntos de *train* para dichos antígenos serán demasiado pocos como para que el algoritmo pueda aprender patrones de los mismos. Además, si bien mencionamos que en general los TCRs reconocen un único

antígeno, en la práctica existen algunas pocas excepciones a la regla, y se les da el nombre de TCRs con reactividad cruzada. El problema que surge es que estos TCRs no son apropiados para nuestra definición en el paso de entrenamiento, dado que usamos una distribución de Bernoulli. Dichos TCRs fueron descartados ya que no pueden ser catalogados correctamente. Los elementos descartados no fueron considerados en el momento de entrenar y probar el algoritmo. Se descartaron 53 TCRs porque reconocían múltiples antígenos y 60 antígenos con 415 TCRs en total, porque eran reconocidos por menos de 70 TCRs cada uno. Adicionalmente, se descartaron 68 TCRs para que las proporciones en todos los casos de test (ver sección 4.3.1) sean equivalentes.

La base de datos con la cual contamos luego de descartar los casos anteriores posee la siguiente distribución:

Antígenpp'	Cantidad de TCRs	Especie
'GILGFVFTL'	540	HUMANO
'LLWNGPMAV'	230	HUMANO
'NLVPMVATV'	160	HUMANO
'GLCTLVAML'	140	HUMANO
'CINGVCWTV'	70	HUMANO
'SSYRRPVGI'	510	RATÓN
'SSLENFRAYV'	380	RATÓN
'HGIRNASFI'	260	RATÓN
'ASNENMETM'	250	RATÓN
'LSLRNPILV'	140	RATÓN
'SSPPMFRV'	130	RATÓN
'TVYGFCLL'	80	RATÓN

Como se puede ver, de momento los TCRs de los cuales se conocen ambas cadenas son bastante acotados. Esto ocurre porque codificar la cadena alfa y la beta resulta más costoso y la mayoría de los estudios se centran en analizar la cadena beta. Si bien dichos datos no son aún suficientes como para poder usar el algoritmo en el plano industrial, veremos en esta tesis que el mismo es capaz de aprender patrones con estos datos, con lo cual puede funcionar cuando exista más disponibilidad de datos.

4.2. Conjunto de datos utilizados

Los datos utilizados pertenecen a dos especies distintas, los humanos y los ratones. La importancia de poseer dos conjuntos de datos distintos, en este caso los conjuntos de datos de especies distintas, radica en que el algoritmo, en su etapa de entrenamiento, funciona captando patrones. Se debe demostrar que los patrones que captó el algoritmo son válidos para más de un conjunto de datos y no son un patrón fortuito proveniente de la distribución de un conjunto de datos concreto. Por eso, probamos el algoritmo con ambas especies y demostramos que el largo óptimo para los kmeros es el mismo.

4.3. Creación de conjuntos de prueba(*test*) - entrenamiento(*train*)

4.3.1. Método de K-Fold

Algunos algoritmos predictivos requieren pasar por una etapa de 'entrenamiento' para capturar patrones a partir de un conjunto de datos. Existen varias estrategias para ge-

nerar verificaciones de este tipo de algoritmos. La idea es, dado un conjunto de datos, se utiliza parte del mismo para entrenar el algoritmo y otro parte para la verificación. Naturalmente, este proceso se puede repetir de forma tal que los conjuntos de *train* y *test* no contengan siempre los mismos elementos. Un ejemplo es el método Leave-One-Out [24], en el cual se substrae un elemento del conjunto de datos, se utiliza el resto del conjunto para el entrenamiento, excepto los que son muy similares al substraido, y el valor extraído se utiliza para la verificación. Luego se repite el proceso con otro elemento distinto, hasta que cada elemento se haya utilizado una vez para la verificación. No utilizamos este mecanismo porque la base de datos de entrenamiento podría resultar muy desbalanceada cuando el TCR substraido reconoce a un antígeno que posee relativamente pocos TCRs que lo reconocen. El método de *K-Fold* (ilustrado en 4.1) es otra estrategia para verificar esa clase de algoritmos. El mismo consiste en, dado un conjunto de datos, crear múltiples particiones en donde el conjunto de datos es dividido en dos subconjuntos a los que llamamos *train* y *test*, el primero se utiliza en la etapa de entrenamiento del algoritmo y el segundo para determinar qué porcentaje de las predicciones del algoritmo entrenado son correctas. Además, todo elemento dentro del conjunto de datos debe aparecer en un subconjunto de *test* exactamente una vez. En nuestro caso decidimos aplicar el método de *K-Fold*, con la propiedad de que cada conjunto de *test* tenga 1/10 del conjunto de datos, es decir que existirán diez conjuntos de *test* - *train*. Dividimos el conjunto de datos en 10 subconjuntos, que serán los conjuntos de *test* de cada partición. Sin embargo, estos no pueden generarse simplemente seleccionando un 1/10 de TCRs al azar, ya que esto alteraría las proporciones de TCR por antígeno y el conjuto de *test* creado resultaría muy desbalanceado. Además, podrían aparecer conjuntos de test que contengan todos los TCRs para determinado antígeno, con lo cual su respectivo conjunto de train no tendría ningún TCR para el mismo, y si un determinado antígeno no aparece en el conjunto de *train* pero si en el de *test*, será imposible de predecir ya que los antígenos se interpretan como *labels*, y no existe forma de asumir una etiqueta que no se encuentra en el conjunto de entrenamiento. Es decir si un antígeno está en un conjunto de *test*, debe aparecer en su respectivo conjunto de *train*. En otras palabras, el algoritmo tiene que haber sido entrenado con TCRs que reconozcan ese antígeno. Para solucionar esto, planteamos un modelo en donde para cada antígeno en la base de datos, se selecciona 1/10 de los TCRs que lo reconocen para cada conjunto de *test*. De esta forma, las proporciones de TCR por antígeno en cada partición son las mismas que en el conjunto de datos original. Además todos los antígenos que no fueron filtrados del conjunto original aparecen en todos los conjuntos de train, con lo cual no existe un antígeno en un conjunto de test que no aparezca también en su respectivo conjunto de train.

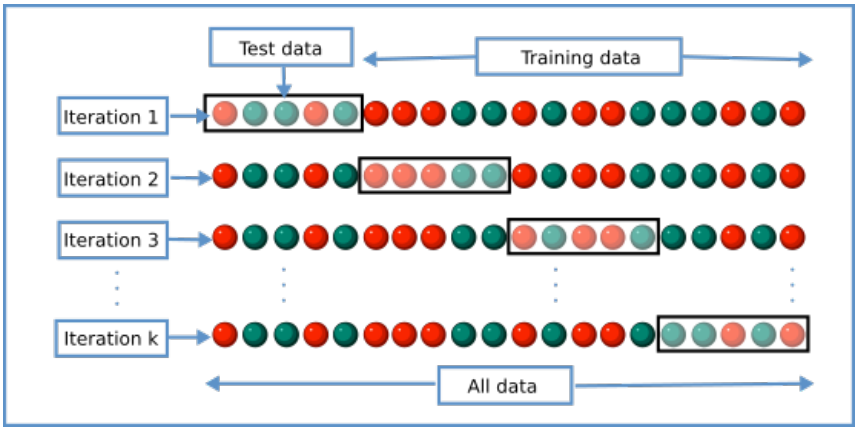

Fig. 4.1: Ejemplo de aplicación del modelo de KFold.

Sin embargo, ¿cómo construimos los conjuntos de *train* correspondientes a cada conjunto de *test*?. Podríamos simplemente incluir todos los TCRs que no están en el *test*, pero haciendo esto se corre el riesgo de que las mediciones realizadas en los *test* muestren mejores valores que la realidad. Esto es porque pueden existir TCRs muy similares, con lo cual algoritmo capta patrones demasiado básicos. Por eso, en los conjuntos de *train* debemos purgar todos aquellos TCRs que sean 'similares' a alguno ya existente en el conjunto de *test*.

4.3.2. Distancia de Levenshtein

Ahora bien, ¿cómo definimos una noción de distancia entre dos TCRs? En este caso optamos por una versión simple, introducida por Vladimir Levenshtein [31] en 1966: la distancia de Levenshtein. Esto es, en realidad, una noción de cercanía para dos textos distintos y se basa en la cantidad de 'cambios' necesarios para convertir un texto en otro. Para esto, se recorren ambas secuencias de principio a fin y en cada paso se pueden realizar las siguientes acciones:
- Comparar una letra y avanzar en ambos. Si las letras son distintas se incrementa el contador de cambios.
- Avanzar a la siguiente posición en uno de los textos. El contador de cambios se incrementa.
Formalmente, dados dos string s1 y s2 definimos la función $F_l(s1, s2, i, j)$, que representa la diferencia entre $s1_{0\_i}$ y $s2_{0\_j}$, de la siguiente forma:
$F_l(s1, s2, 0, 0) = 0$
$F_l(s1, s2, i, 0) = F_l(s1, s2, i-1, 0) + 1$
$F_l(s1, s2, 0, j) = F_l(s1, s2, 0, j-1) + 1$
$F_l(s1, s2, i, j) = min(match(i, j) + F_l(s1, s2, i-1, j-1), F_l(s1, s2, i-1, j) + 1, F_l(s1, s2, i, j-1) + 1)$
$match(i, j) = 0 \Leftrightarrow s1_i = s2_j$ y 1 en caso contrario

Dada esta definición, la distancia de Levenshtein se define como $F_{lev}(s1, s2) = F_l(s1, s2, |s1|, |s2|)$ De esta forma, la función se ejecuta en forma recursiva desde el último caracter de ambas secuencias hasta el primero.

Calcular esta función en forma recursiva no es eficiente, ya que en varios casos se está calculando la misma función con los mismos parámetros múltiples veces. Entonces, para implementar la función en una forma más eficiente, se utiliza el método de Programación Dinámica, que consiste en calcular primero los casos base (donde no hay recursión) y luego los posibles valores que requieren los datos calculados previamente. Como todos los resultados intermedios se mantienen en memoria (en este caso en una matriz), de esta forma nos ahorramos repetir cálculos ya que en cada iteración utilizamos únicamente los valores calculados anteriormente.
Supongamos que tenemos dos secuencias: s1 y s2. Sabemos que $F_l(s1, s2, 0, 0)$. Luego, podemos calcular fácilmente $F_l(s1, s2, i, 0)$ para todo i, ya que en cada iteración sumamos 1 al anterior, con lo cual $F_l(s1, s2, i, 0) = i$. En forma análoga $F_l(s1, s2, 0, j) = j$. Ahora bien, supongamos que tenemos calculado $F_l(s1, s2, i, j)$ para todo i menor que I. $F_l(s1, s2, I, 1) = min(match(i, j) + F_l(s1, s2, I-1, 0), F_l(s1, s2, I-1, 1)+1, F_l(s1, s2, I, 0)+1)$. Conocemos los tres valores con los cuales se calcula F_l, los dos primeros porque I-1 < I y el último porque equivale a I ya que i = I y j = 0 en este caso. Con esto podemos calcular $F_l(s1, s2, I, 2) = min(match(i, j)+F_l(s1, s2, I-1, 1), F_l(s1, s2, I-1, 2)+1, F_l(s1, s2, I, 1)+1)$, los primeros porque I-1 < I y el último es el que calculamos en la iteración anterior. De esta forma se incrementa el valor de j en 1 en cada iteración hasta que j = $|s2|$, y calculando así todos los valores para $F_l(s1, s2, I, j)$. Luego se repite este paso con I+1. Al final, se habrá calculado el valor $F_l(s1, s2, |s1|, |s2|)$

```
int Levenshtein(s1,s2)
    Matrix matrix;
    matrix[0,0] = 0 //Caso base F_l(s1,s2,0,0)
    for(j< - [1,2,...|s2|])
        matrix[0,j] = j // Caso base F_l(s1,s2,0,j)
    for(i <- [1,2,...|s1|])
        matrix[i,0] = i // Caso base F_l(s1,s2,i,0)
        for(j< - [1,2,...|s2|])
            int diagonal
            if(s1[i]==s2[j])
                diagonal = matrix[i-1,j-1]
            else
                diagonal = matrix[i-1,j-1] + 1
            int insertion = matrix[i,j-1] + 1
            int delection = matrix[i-1,j] + 1
            if(diagonal < insertion && diagonal < delection)
                //Caso recursivo F_l(s1,s2,i,j) si F_l(s1,s2,i-1,j-i) + match(i,j)
                //es el m\'inimo
                matrix[i,j] = diagonal
            else if(insertion < delection)
                //Caso recursivo F_l(s1,s2,i,j) si F_l(s1,s2,i,j-i) + 1 es el m\'inimo
                matrix[i,j] = insertion
            else
                //Caso recursivo F_l(s1,s2,i,j) si F_l(s1,s2,i-1,j) + 1 es el m\'inimo
                matrix[i,j] = delection
    return matrix[|s1| - 1,|s2| - 1]
```

Fig. 4.2: Algoritmo que calcula la distancia de Levenshtein entre dos secuencias utilizando el método de programación dinámica.

Ahora bien, el algoritmo para calcular la distancia de Levenshtein entre dos secuencias (fig. 4.3.2) tiene una complejidad temporal de $O(|s1||s2|)$. La comparación de todo par de secuencias de un conjunto de datos D tiene una complejidad de $O(\sum_{i,j\in[0..|D|),i<j} |D_i||D_j|)$. Como esto tardaba demasiado en el *script* de Python, se precalcularon todos los valores implementando la distancia de Levenshtein en C++.

Utilizando la distancia de Levenshtein, filtramos aquellos TCR 'semejantes' de los conjuntos de *train* con el objetivo de evitar que el algoritmo de predicción de especificidad de TCRs se encuentre en un contexto en el cual le sea demasiado sencillo realizar las predicciones. Sin embargo, como no es claro cuál es el límite menor al cual dos TCR deberían considerarse demasiado similares, para cada partición creamos cinco conjuntos de *train* en lugar de uno, de la siguiente forma: primero un conjunto de *train* con todos los TCR que no están en el *test* y luego se eliminan aquellos que poseen una distancia de Levenshtein inferior a uno para al menos una secuencia del conjunto de *test*, creando el primer conjunto de *train*. A continuación, se eliminan de ese los que tienen una distancia inferior a dos, creando el segundo y en forma análoga se crean los otros tres conjuntos, en cada uno aumentando en 1 la distancia de Levenshtein máxima en los cuales los TCR son descartados.

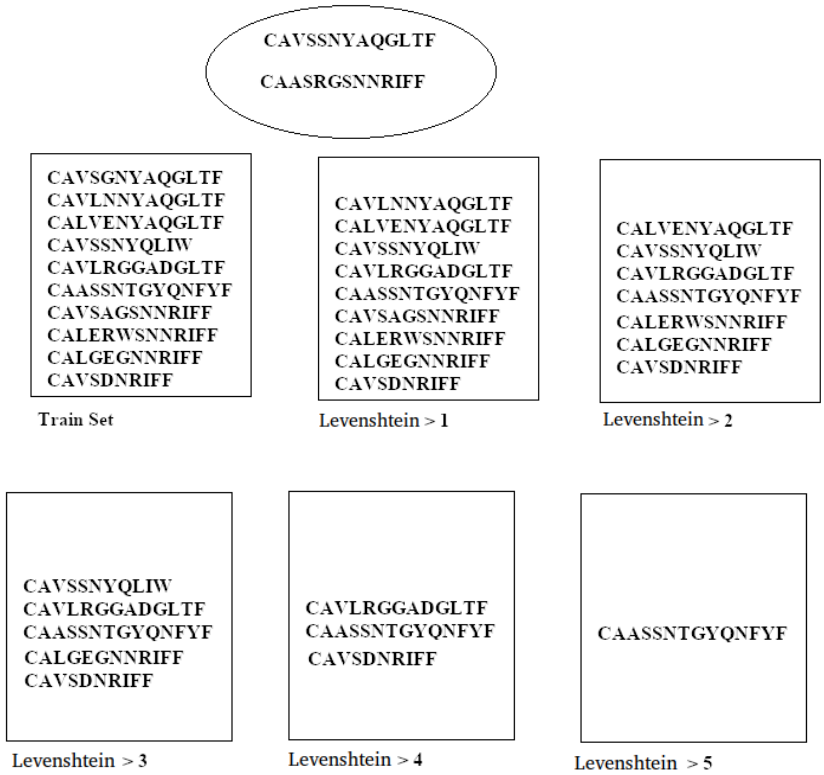

Fig. 4.3: Ejemplo que ilustra cómo se forman los conjuntos de *train* respecto al conjunto *test-train* original. CAVSGNYAQGLTF tiene una distancia de Levenshtein 1 respecto a CAVSSNYAQGLTFcon lo cual es eliminado de todos los conjuntos *train*. CAVLNNYAQGLTF y CAVSAGSNNRIFF, tienen una distancia de Levenshtein 2 con CAVSSNYAQGLTF y CAASRGSNNRIFF respectivamente, con lo cual son descartados en todos los conjuntos excepto el primero. Y así se siguen descartando TCRs de los conjuntos.

4.4. Análisis de kmeros

La mayoría de los algoritmos utilizados para analizar secuencias utilizan una estrategia denominada alineamiento [2], que es similar a la distancia de Levenshtein, con la diferencia que cuando se escoge la opción de avanzar en una de las secuencias y no en la otra, el marcador de diferencias no necesariamente aumenta en la misma proporción que en una comparación fallida. Además, considera la existencia de regiones que son comunes en ambas secuencias, es decir, si existen muchas comparaciones exitosas, considera que son más similares. Por último en algunas refinaciones del algoritmo no toda comparación fallida aumenta de la misma forma el contador de diferencias, existen aminoácidos que considera más 'semejantes' que otros. Estos valores deben ser ajustados según el problema para lograr resultados óptimos.

Sin embargo, si bien esta estrategia suele ser útil, no deja de ser una heurística; además, está diseñada para calcular la 'distancia evolutiva' [22] entre ambas secuencias, es decir, el camino más probable que una secuencia, con el paso de las generaciones, haya mutado en la otra. Esto suele coincidir con la similitud estructural, dado que la mayoría de las

mutaciones no alteran drásticamente las funciones de las proteínas. Esto es porque cuando un ser vivo se reproduce, sobre todo uno complejo, un cambio grande es más propenso a causar que el organismo nuevo no pueda subsistir. En el caso de las células T los rearreglos de los genes que codifican el TCR para cada célula T no siguen las mismas reglas que la evolución de las especies [8]. La única función de este tipo de células es identificar posibles patógenos. Si no encuentra el patógeno al cual identifica simplemente no hace nada; movilizarse dentro de un espacio controlado le permite subsistir en esas condiciones. Entonces, si bien al formarse las variaciones de TCRs pueden aparecer secuencias con cambios que afectan bastante a la estructura del TCR, la célula T portadora podrá subsistir en el ambiente en el que se encuentra. Es por eso que en este caso la "distancia evolutiva"no coincide tan exactamente con el parecido estructural del TCR. Por ende, las distancias *standar* entre aminoácidos (como BLOSUM o PAM) pueden no representar adecuadamente las semejanzas estructurales entre distintos CDR3.

El análisis de kmeros no es una estrategia nueva en bioinformática, y se presenta como una alternativa popular al alineamiento [21]. Se basa en comparación de subsecuencias en forma booleana, es decir, si son idénticas o no. Para cada secuencia se generan todas las subsecuecias posibles, incluso las que se solapan entre sí, de un largo determinado. Se aplicó esta metodología porque en toda unión de proteínas siempre existen determinadas subsecuencias específicas relevantes para la interacción. En particular, dentro de la secuencia de un TCR existen subsecuencias del mismo que son importantes para efectuar la interacción con el antígeno [12] [7].

Hasta ahora hablamos de metodologías para comparar dos secuencias; pero lo que necesitamos para resolver el problema planteado es algo distinto: una noción de pertenencia de un TCR a un determinado grupo de TCRs, es decir, aquellos que unen a un determinado antígeno. Para hacerlo adaptamos estas metodologías utilizando la estadística como se explicará a continuación.

4.5. Modelo estadístico

Utilizando la estadística intentamos determinar la probabilidad de que una proteína interactúe con otra proteína, sabiendo que la primera contiene un determinado kmero en su secuencia. A la primera proteína la llamaremos SEQ, ya que se representa con la secuencia de aminoácidos correspondiente a la misma y a la segunda la denominaremos LABEL, ya que se simboliza con una etiqueta.

Con una base de datos conocida, podemos inferir la probabilidad de que la proteína representada por SEQ interactúe con la proteína representada por LABEL, asumiendo que posee un determinado KMERO en su secuencia:

$$P(LABEL|KMERO) = N_{KMERO\_LABEL}/N_{KMERO}$$

Siendo N_{KMERO} la cantidad de proteínas SEQ que contiene KMERO y N_{KMERO_LABEL} la cantidad de proteínas que contienen KMERO y a la vez interactúan con LABEL (cumplen ambas condiciones).

En los kmeros que sean importantes para que una interacción LABEL-SEQ pueda efectuarse, si la base de entrenamiento es lo suficientemente grande y balanceada, podremos observar una mayor frecuencia de los mismos en las secuencias de proteínas que unen a LABEL.

Siguiendo el modelo, puntuamos la probabilidad de que una proteína SEQ interactúe con un $LABEL$ basados en la probabilidad $P(LABEL|KMERO)$ para cada uno de sus kmeros. Utilizando lo que tenemos hasta ahora, queremos combinar la información obtenida para cada uno de los kmeros para calcular un puntaje que mida la probabilidad de unión de la secuencia completa hacia el antígeno representado por LABEL. Definimos el puntaje de que un CDR3 cuya secuencia es SEQ interactúe con el antígeno LABEL, $S(LABEL|SEQ)$ como:

$$S(LABEL|SEQ) = COMBINAR(P(LABEL|KMERO_1), ..., P(LABEL|KMERO_n))$$

Para la función COMBINAR pueden utilizarse distintas estrategias, como elegir el máximo, el mínimo, etc. En nuestro caso, optamos por calcular el promedio. Entonces el puntaje se define como:

$$S(LABEL|SEQ) = \sum_{Kmero \in Kmeros(Seq)} P(Label|Kmero)/cantidadKmeros$$

Siendo $Kmeros(SEQ) = \{KMERO_1, ..., KMERO_n\}$

Se obtienen mejores resultados si la base de datos está balanceada, ya que en caso de que existan antígenos asociado a muchos TCRs y otros asociados a relativamente pocos, hay mayor probabilidad de que kmeros que no aportan a la predicción se encuentren sólo en los grupos de TCRs más grandes, con lo cual el algoritmo los identificará como relevantes para el reconocimiento. Sin embargo, veremos en los resultados que nuestro algoritmo resultó parcialmente resistente al desbalanceo.

4.6. Implementación del sistema predictivo

Como mencionamos anteriormente, el sistema predictivo consta de dos partes, entrenamiento y predicción. Siguiendo ese esquema dividimos también la implementación en dos partes. La primera, que corresponde al entrenamiento, debería ejecutarse sólo cuando la base de datos se actualiza. En el mismo se calculan los puntajes anteriormente mencionados para que luego sean usados en la parte de predicción.

Primero, el algoritmo crea en memoria tres contadores que registran apariciones de TCRs que reconocen un antígeno, kmeros y kmeros de TCRs que reconocen a un antígeno. Itera sobre las relaciones TCR-antígeno presentes en el conjunto de entrenamiento y aumenta el contador de TCRs para el antígeno, luego divide el TCR en kmeros y para cada uno aumenta el contador de kmeros para el kmero y el contador de kmeros de TCRs que reconocen al antígeno para la relación kmero-antígeno. Luego, itera todos los pares kmero - antígeno y cálcula P(LABEL | KMERO) para cada par LABEL-KMERO utilizando los valores recopilados en los contadores. Como dijimos anteriormente, la probabilidad de que un TCR reconozca a un antígeno dado que posee un kmero puede calcularse como cantidad de TCRs que poseen el kmero y reconocen al antígeno sobre la cantidad de TCRs que poseen el kmero.

```
train(relations,k)
  Counter kmerCounter, antigenPerKmer;
  Set kmers, antigens;
  for(relation <- relations)
    antigen.add(relation.antigen)
    for(kmer <- getKmers(relation.tcr,k))
      kmers.add(kmer)
      kmerCounter.incremet(kmer)
      antigenPerKmer.increment(kmer,relation.antigen)
  Map probs;
  for(kmer <- kmers)
     for(antigen <- antigens)
      double p = antigenPerKmer.get(kmer,antigen)/kmerCounter.get(kmer)
      probs[(antigen,kmer)] = p
  return probs;

class Relation
  String tcr;
  String antigen;
```

Fig. 4.4: Pseudocódigo que describe el cálculo de la probabilidad P(LABEL | KMERO) para todo par LABEL, KMERO. *'relations'* representa la base de datos de entrenamiento, como un objeto con los atributos antigen, que devuelve el LABEL que representa al antígeno, y TCR, que retorna la secuencia del TCR. k es el largo de los kmeros

En el código **??** Counter' es una estructura de datos similar a un diccionario, con valores enteros. Posee una función llamada increment, que al recibir una clave aumenta el entero asociado a esa clave. En caso de que la clave no exista anteriormente, la agrega con un valor de 1.

Con esto obtenemos una estructura de datos intermedia que contiene la información estadística sobre el conjunto de entrenamiento, que es necesaria para la predicción de un TCR. Como hasta el momento se utilizó sólo la base de datos y no el TCR del *input*, la misma puede guardarse en disco para ser utilizada en múltiples ocasiones para realizar predicciones sobre distintos TCRs sin repetir el entrenamiento del algoritmo.

Existen 20^k combinaciones de kmeros posibles, ya que el kmero tiene largo k y existen 20 aminoácidos posibles en cada posición del kmero. Como máximo, puede ocurrir que para cada antígeno existente en la base de datos se registren cada una de dichas combinaciones, por lo cual como máximo la complejidad espacial de esta estructura intermedia, siendo N la cantidad de antígenos en la base de datos, es de $O(N * 20^k)$. Como los CDR3 no suelen tener un largo mayor a 20, podemos asumir que k es acotado. Sin embargo, incluso con un k igual a 20, la capacidad consumida por esta estructura es muy alta (del orden de 20^{20}). Dado que el consumo de memoria es muy elevada necesitamos saber qué tan grande es el k óptimo para el algoritmo. Afortunadamente, como veremos más adelante, el mismo es considerablemente menor a veinte.

La segunda parte (fig. 4.6) utiliza esta estructura intermedia, el TCR recibido como *input* y el conjunto de antígenos presentes en la base de datos de entrenamiento y se ejecuta cada vez que se necesita predecir un TCR. En esta etapa, se itera entre los antígenos conocidos y se calcula para cada uno el puntaje para el TCR.

```
predict(TCR,probs,antigens,k)
    OrderSet<(String,Double)> res; //Ordena en forma decreciente por segundo elemento
    for(antigen <- antigens)
        double prob = 0;
        for(kmer <- getKmers(TCR,k))
            //En caso de que el par (antigen,kmer) no exista  en probs devuelve 0
            prob += probs[(antigen,kmer)];
        res.add((antigen,prob))
    return res
```

Fig. 4.5: Pseudocódigo de la parte de predicción del algoritmo. Se recorre el conjunto de antígenos, para cada antígeno se suman las probabilidades de que ese antígeno sea reconocido por la secuencia dado que posee un kmero, para todos los kmeros de la secuencia.

Para validar el funcionamiento del sistema de predicción de antígenos lo entrenamos con cada conjunto de *train* y luego iteramos sobre su respectivo conjunto de *test* corriendo el programa de predicción para cada TCR dentro del mismo, validando si la salida coincide con el antígeno real, considerando el antígeno predicho como el que obtuvo un mayor puntaje en el algoritmo de ***predict***. Luego agrupamos los resultados según la distancia de Levenshtein y calculamos el porcentaje de aciertos para cada grupo.

```
hitRate(relationsTrain,relationsTest,k)
    int matchs = 0;
    Map probs = train(relationTrain)
    Set antigens = getAntigens(relationsTrain)
    for(relation <- relationsTest)
     prediction = predict(relation.tcr,probs,antigens,k)[0]
     [String antigenPredicted,double maxProb] = prediction
     if(antigenPredicted == relation.antigen)
        matchs++
    return matchs/relationsTest.size()
```

Fig. 4.6: Pseudocódigo que describe el cálculo del *hit rate* para un par de conjuntos *train* y *test*. relationsTrain representa la base de datos de entrenamiento, como un objeto con los atributos antigen, que devuelve el LABEL que representa al antígeno y TCR, que devuelve la secuencia del TCR. **relationsTest** tiene la misma estructura y representa la base de datos de test. k es el largo de los kmeros

4.7. Validación del algoritmo en conjuntos de datos independientes

Corrimos nuestro algoritmo sobre la cadena beta con distintos kmeros de diferentes tamaños para determinar cuál era el óptimo. Luego, calculamos el *hit-rate* (fig. 4.6) de los resultados de todos los conjuntos de *train-test* para un mismo k y en base a los mismos construimos las gráficas.

Ahora bien, para garantizar que los patrones captados no se encuentran fortuitamente en el conjunto de datos analizado, repetimos la misma prueba, *K-Fold*, *train* y *test* para dos subconjuntos diferentes, los TCRs de ratón y los de humano.

La validación de los resultados consiste en comparar los resultados de ambos y verificar que sean similares. Como ambos conjuntos de datos proceden de especies distintas, se espera que los patrones en común aparezcan también TCRs de otras especies.

Además, como mencionamos en la sección de *K-Fold*, para cada conjunto de datos construimos múltiples grupos de conjuntos de *test-train*, en donde la máxima distancia de Levenshtein entre ambos es distinta. Para cada uno de estos grupos de conjuntos, el algoritmo fue ejecutado en forma independiente desde el comienzo.

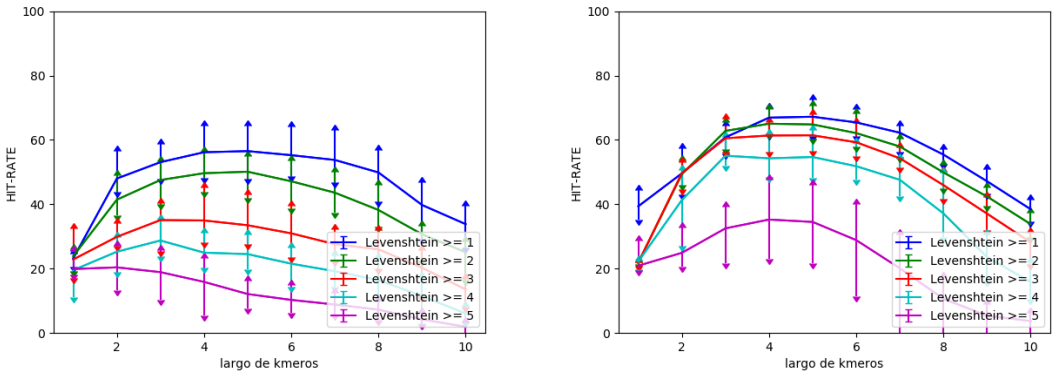

Fig. 4.7: Porcentaje de aciertos de TCR respecto a sus antígenos en función del largo de los kmeros seleccionado (k) en la cadena alfa en ratones (izquierda) y en la cadena beta ratones (derecha).

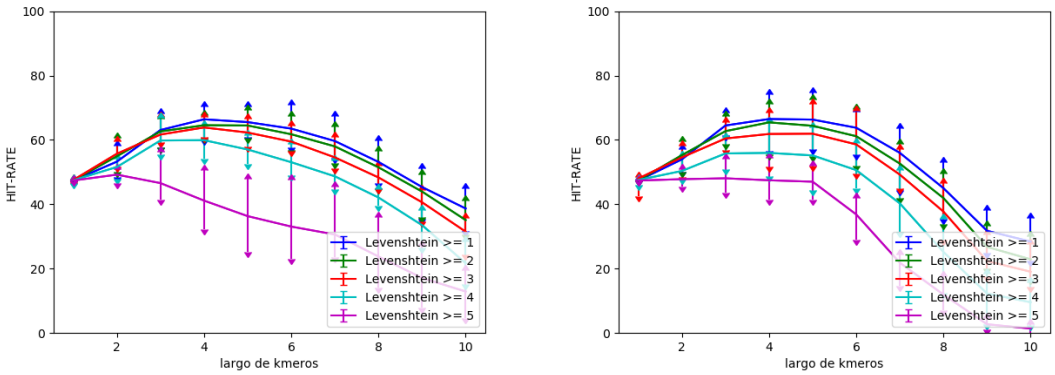

Fig. 4.8: Porcentaje de aciertos de TCR a sus respectivos antígenos en función del largo de los kmeros seleccionado (k) en la cadena alfa en humanos (izquierda) y en la cadena beta humanos (derecha).

Como podemos observar en las figuras 4.7 y 4.8,si el k = 1, la predicción debería ser cercana al azar, ya que un sólo aminoácido aporta poca información. En el conjunto de datos de humanos hay una mejoría respecto al de ratones en k = 1, pero esto probablemente sea porque el mismo se encuentra más desbalanceado. Sin embargo, se puede observar que

en ambos gráficos el máximo es alcanzado en un k entre 4 y 5, y aun más, dicho máximo posee un valor similar.

En humanos, en la cadena alfa el porcentaje de aciertos aumentó desde 47.54 % en k=1 hasta 63.86 % en k = 4, mientras que en la cadena beta aumentó desde 47.98 % en k = 1 hasta 61.93 % en k = 5.

4.8. Modelo con múltiples secuencias

El TCR está compuesto por dos cadenas denominadas alfa y beta, en cada una de ellas hay una región denominada CDR3. Como dijimos anteriormente, la parte del *input* del problema que representa los TCR consiste en dichos segmentos, ya que es la región que compone el principal sitio activo en la interacción con el péptido foráneo.
Si lo queremos analizar completo, necesitamos modificar dicho modelo para trabajar con secuencias. Sin embargo, concatenar ambos CDR3 no tendría sentido, ya que el hecho de que un kmero aparezca en la cadena beta en el *train* no debería afectar a las estadísticas si el kmero aparece en la cadena alfa en la secuencia a predecir. Además, aparecerían kmeros que contienen elementos de las dos secuencias. Evaluamos distintas formas de combinar la información proveniente de ambas secuencias. En nuestra tesis analizamos el promedio y la multiplicación. Definimos el score (S) para dos secuencias como:

$$S(LABEL|SEQ1, SEQ2) = COMBINAR(S(LABEL|SEQ1), S(LABEL|SEQ2))$$

Repetimos el experimento usando los mismos conjuntos de *test-train*, esta vez utilizando el modelo con múltiples secuencias, en donde la función COMBINAR puede ser el promedio o la multiplicación. Esta noción se puede generalizar: si consideramos que existen n secuencias, podemos definir una función COMBINAR que reciba n parámetros:

$$S(LABEL|SEQ1, ..., SEQn) = \\ COMBINAR(S(LABEL|SEQ1), ..., S(LABEL|SEQn))$$

Este caso es útil para analizar la interacción de proteínas en donde existen tres o más regiones críticas para la unión. De hecho este es el caso de los TCR, que interactúan con el antígeno en seis regiones críticas, sin embargo, el volumen de datos actualmente no es suficiente, y para fines de la presente tesis sólo analizamos los CDR3 de alfa y de beta.

Realizamos la comparación en el punto en donde se detectaron mejores resultados: largo 4 en alfa y largo 5 en beta utilizando los algoritmos descriptos en 4.8, 4.8 y 4.8.

```
Class TCR
    String alpha;
    String beta;


class RelationTCRComplete
    TCR tcr;
    String antigen;
```

```
trainMultiple(relations,k1,k2)
    Map probsAlpha = train(relations.map(\x => (x.tcr.alpha,x.antigen,k1) ))
    Map probsBeta = train(relations.map(\x => (x.tcr.beta,x.antigen,k2) ))
    return [probsAlpha,probsBeta]
```

Fig. 4.9: Algoritmo de train extendido a dos secuencias. Utiliza el algotimo de train para una sola secuencia en forma independiente para las cadenas alfa y beta. relation es un objeto similar a la entrada del algoritmo de train para una cadena, pero el parámetro 'tcr' tiene dos atributos: 'alfa' y 'beta'

```
predictMultiple(TCR,probsAplha,probsBeta,antigens,k1,k2)
    //Ordena por el segundo atributo de la tupla, en orden decreciente
    OrderSet<Tupla<String,Int>> probs;
    for(antigen <- antigens)
        float probAlpha = 0;
        for(kmer <- getKmers(TCR.alpha,k1))
            probAlpha += probsAlpha[kmer| antigen];
        float probBeta = 0;
        for(kmer <- getKmers(TCR.alpha,k2))
            probBeta += probsBeta[kmer| antigen];
        float prob = combine(probAlpha,probBeta)
        probs.add((antigen,prob))
    return probs
```

Fig. 4.10: Algoritmo de predicción extendido a dos secuencias, con una función combine genérica. Dentro del loop, replica el código del predict original para ambas cadenas en forma independiente y luego las combina.

```
hitRateMultiple(relationsTrain,relationsTest,k)
  [Map probsAlpha,Map probsBeta] = trainMultiple(relationsTrain,k)
  Set antigens = getAntigens(relationsTrain)
  for(relation <- relationsTest)
    prediction = predictMultiple(relation.tcr,probsAlpha,probsBeta,antigens,k)
    [String antigenPredicted,double maxProb] = prediction
    if(antigenPredicted == relation.antigen)
      matchs++
  return matchs/antigen.size()
```

Fig. 4.11: Algoritmo que calcula el hit-rate cuando se analizan ambas cadenas. relationTrain y relationTest son objetos de la misma clase que relation.[en] el algoritmo de train múltiple.

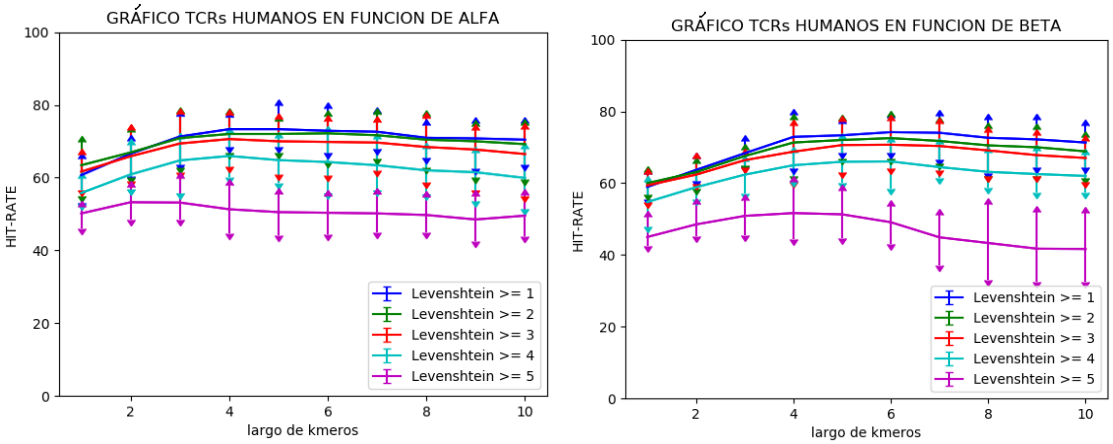

Fig. 4.12: Se analizan los resultados cuando se utilizan ambas secuencias de los TCRs en humanos. A la izquierda, el largo de los kmeros en alfa varía y el largo de los kmeros en beta es 5. A la derecha, el largo de los kmeros en beta varía y el tamaño del k en alfa es 4.

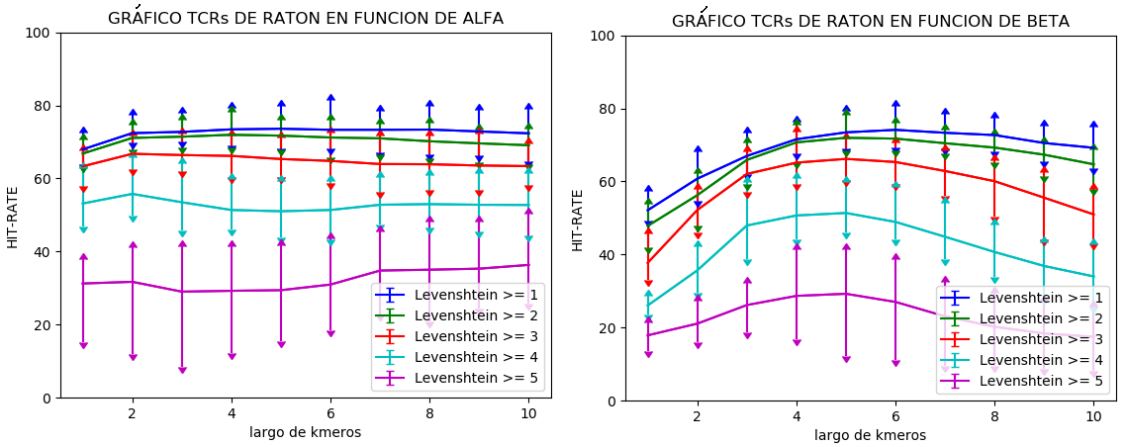

Fig. 4.13: Se analizan los resultados cuando se utilizan ambas secuencias de los TCRs en ratones. A la izquierda, el largo de los kmeros en alfa varía y el largo de los kmeros en beta es 5. A la derecha, el largo de los kmeros en beta varía y el tamaño del k en alfa es 4

En las figura 4.12 y 4.13 se muestran los *hit-rate* combinando la información de la alfa y la beta. Podemos ver que los resultados alcanzan un mayor *hit-rate* si analizamos ambas cadenas que cuando analizamos la alfa y la beta por separado. El máximo en la cadena beta resultó ser 5 mientras que en alfa es 4 (sin embargo la diferencia en beta entre k = 6 y k = 5 es mínima)

Analizamos los resultados donde la distancia de Levenshtein es 3. Elegimos este valor porque consideramos que la cantidad de TCRs que no fueron eliminados en los conjuntos de *train* con una distancia de Levenshtein > 4 son demasiado acotados para poder entrenar el algoritmo, en otras palabras, se eliminan demasiados TCRs del conjunto de *train*. Con la distancia Levenshtein de corte en 3 el algoritmo tiene suficiente información para poder entrenar pero se filtran más elementos que podrían ser demasiado similares, que no son eliminados si la distancia de Levenshtein es 1 o 2. Con esto último se reduce la posibilidad de que el algoritmo pueda acertar predicciones por el hecho de aprender

patrones demasiado específicos.

En humanos, la cadena beta mejoró desde 57.19 % en k = 1 a 70,79 % en k = 5 y la cadena alfa de 59.74 % en k = 1 a 70,79 % en k = 4

En ratones, la cadena beta mejoró desde 35.83 % en k = 1 a 65,43 % en k = 4 y k = 5

En la cadena beta, si bien el *hit-rate* es mejor en humanos en general, se puede apreciar que en el caso de los humanos el aumento fue cercano al 23 % y en el caso de los ratones cerca de un 83 %. Esto puede deberse a que existen menos antígenos registrados para humanos. Además existe un mayor desbalance, es decir que existe una mayor cantidad de TCRs que reconocen a un antígeno respecto a los demás antígenos. Como el desbalance es igual en los conjuntos de *train* y *test*, ya que se tomó 1/10 de los TCRs de cada antígeno para el conjunto de *test* y el 9/10 restante para el conjunto de *train* (excepto los que fueron filtrados), el algoritmo tiende a elegir al antígeno con mayor cantidad de TCRs. Esto provoca que en las condiciones donde el algoritmo perdió considerablemente su capacidad predictiva, existan probabilidades de que acierte al azar. En los ratones el conjunto de datos es mayor y más variado, con lo cual la predicción al azar posee un menor *hit-rate*, pero por esa misma razón se puede apreciar mejor el rendimiento cuando k se acerca al punto óptimo.

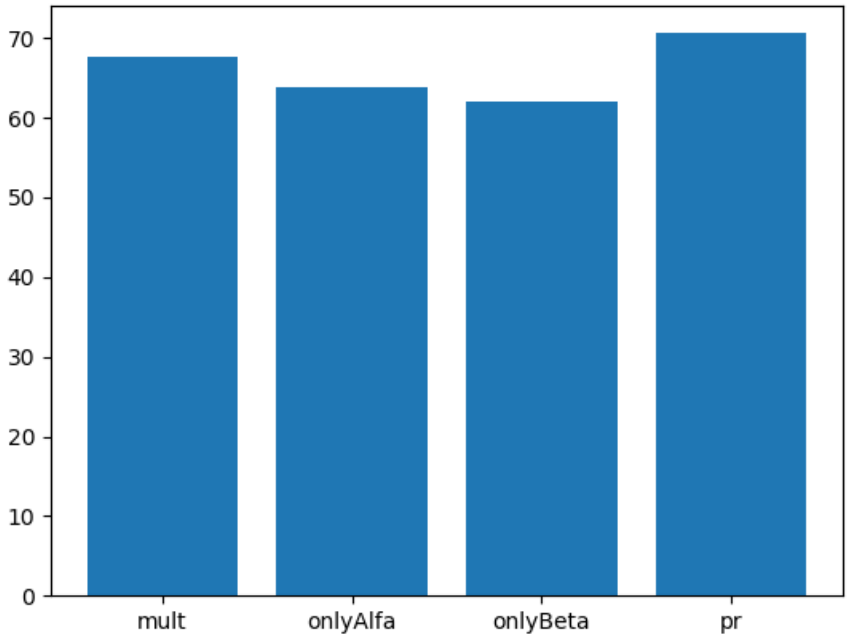

Fig. 4.14: Comparación del total de aciertos con kmeros de largo 4 en alfa y largo 5 en beta, entre los modelos en donde las probabilidades de ambas secuencias se multiplican(mult), donde ambas se promedian(pr), donde sólo se considera la cadena alfa(onlyAlfa) y donde sólo se considera la cadena beta(onlyBeta). Se ha optado por el promedio ya que generó mejores resultados.

Para determinar cuál es la mejor forma de combinar la información de ambas cadenas, si el promedio o la multiplicación, se utiliza el algoritmo de *hit-rate* múltiple, donde en un caso la función 'combine' utilizada es la multiplicación del puntaje de ambas cadenas, y en otro caso la función 'combine' se define como el promedio. Además se comparan dichos resultados con el *hit-rate* utilizando sólo la cadena alfa o sólo la cadena beta y se puede observar que en estos casos el *hit-rate* resulta inferior que si se utilizan ambas secuencias. Para la comparación se ha utilizado el conjunto de datos de TCRs de humanos ya que el de ratones no ha revelado un buen rendimiento en la cadena alfa. El largo de los kmeros para cada cadena fue seteado en el óptimo encontrado en las pruebas anteriores (4 para la cadena alfa y 5 para la beta) y se han utilizado los conjuntos *train* con una distancia de Levenshtein mayor a 3.

La comparación graficada en 4.14 muestra que el promedio da mejores resultados que la multiplicación, por lo cual se ha escogido este mecanismo. Sin embargo, la diferencia entre ambos es pequeña, con lo cual esto puede deberse a la distorsión causada por poseer un conjunto de datos reducido.

4.9. Tolerancia al desbalanceo

Hay que tener en cuenta que si el conjunto de datos está desbalanceado, es decir que existen muchos registros de TCRs que reconocen a un antígeno determinado respecto al total de los TCRs, el algoritmo tiende a optar en las predicciones por quién posea más registros. Esto es porque, al existir más registros de ese antígeno, se detectará una mayor aparición de kmeros en los TCR que corresponden al mismo. Entonces, creamos histogramas evaluando el *hit-rate* de cada antígeno individualmente y descubrimos que el antígeno con más registros concentra la mayoría de los aciertos.

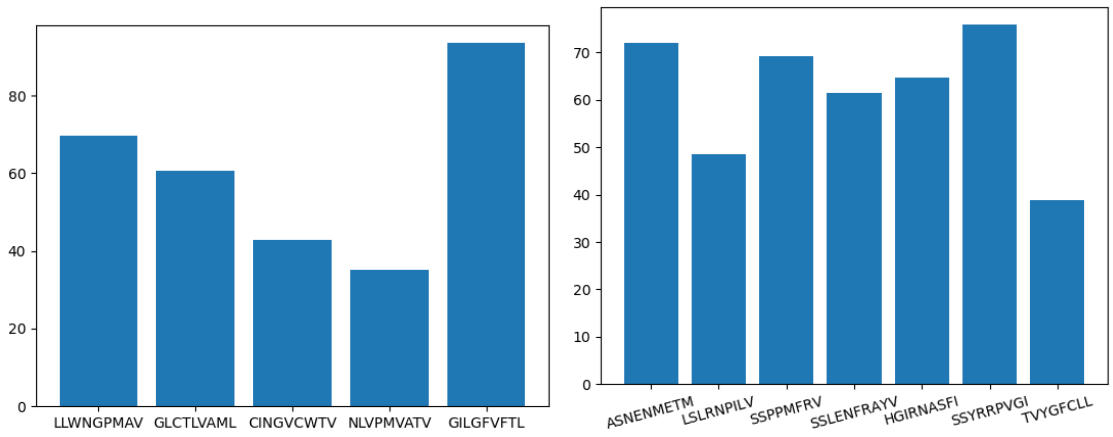

Fig. 4.15: Porcentaje de aciertos en la predicción para cada antígeno, a la izquierda, sobre la base de datos de antígenos humanos y a la derecha, sobre la base de datos de ratones. En la primera gráfica, se puede observar que el péptido GILGFVFTL posee una cantidad mucho mayor de aciertos que el resto de los péptidos, de lo cual se deduce que el algoritmo entrenado con el conjunto de datos propuesto tiende a elegir a dicho antígeno.

Si se diera el caso en donde el algoritmo está seleccionando la mayoría de las veces al antígeno con más TCRs y no está prediciendo realmente, entonces veríamos que el resto de los antígenos tendrían un *hit-rate* muy bajo, ya que la predicción correcta sólo se da en el antígeno con mayor cantidad de TCR. Sin embargo, como se ve en la figura 4.15, el número de predicciones correctas por antígeno supera el 50 % en su mayoría, con lo cual comprobamos que conserva poder predictivo para la mayoría de los antígenos.

4.10. Especificidad de unión

Hasta el momento la predicción se basaba en cuál antígeno poseía un puntaje más alto sin determinar cuánto era ese puntaje. Sin embargo, pueden existir casos en donde la diferencia de puntaje entre el antígeno marcado como más probable respecto al obtenido por otro antígeno sea muy pequeña. En estos casos, es probable que el TCR una tanto al antígeno predicho como al segundo con mayor puntaje. Definimos la especificidad del TCR como:

$$SP(Antigeno_1|TCR_\alpha, TCR_\beta) = S(Antigeno_1|TCR_\alpha, TCR_\beta)/S(Antigeno_2|TCR_\alpha, TCR_\beta)$$

Donde $Antigeno_1$ se refiere al antígeno cuyo puntaje para el TCR representado por TCR_α, TCR_β es mayor, mientras que $Antigeno_2$ es el siguiente en puntaje.

Si este valor es bajo, significa que la probabilidad de que el TCR una al segundo antígeno no es mucho menor que la probabilidad de que una al antígeno predicho. En estos casos puede ser que el TCR realmente sea promiscuo, pero también es probable que la base de datos sea demasiado acotada para realizar una predicción fiable, o que en realidad el TCR reconozca a un antígeno desconocido. Intentaremos filtrar estos casos eliminando las predicciones donde la especificidad es baja.

Dado un experimento en el que se quiere evaluar una condición que puede ser positiva o negativa, para cada resultado existen cuatro posibilidades, que la condición testeada sea positiva y el resultado marque positivo, que la condición sea negativa y el resultado indique negativo, que la condición sea positiva pero el resultado infiera que es negativo o que la condición sea negativa pero el resultado indique positivo. A las primeras dos posibilidades se las denomina verdadero positivo y verdadero negativo, mientras que a las dos últimas se las denomina falso negativo y falso positivo.

De nuevo en nuestro caso, planteamos un experimento en donde, dado una predicción, la consideramos incorrecta si la especificidad de unión es inferior a un determinado umbral y correcta si es superior o igual. Entonces, un verdadero positivo es una predicción correcta por encima del umbral, un verdadero negativo es una predicción incorrecta por debajo del umbral, un falso positivo una predicción incorrecta por encima del umbral y un falso negativo una predicción correcta por debajo del umbral.

4.10.1. Curva ROC

La curva ROC, acrónimo de *Receiver Operating Characteristic*, es un gráfico que considera los falsos positivos y los verdaderos positivos, mostrando cuántos verdaderos positivos se pueden lograr al precio de cometer una determinada cantidad de falsos positivos. Para realizar dichos gráficos se calcularon para cada especificidad registrada en la salida del algoritmo la cantidad de falsos positivos y verdaderos positivos utilizando dicho valor como umbral para colocar un punto en la curva.

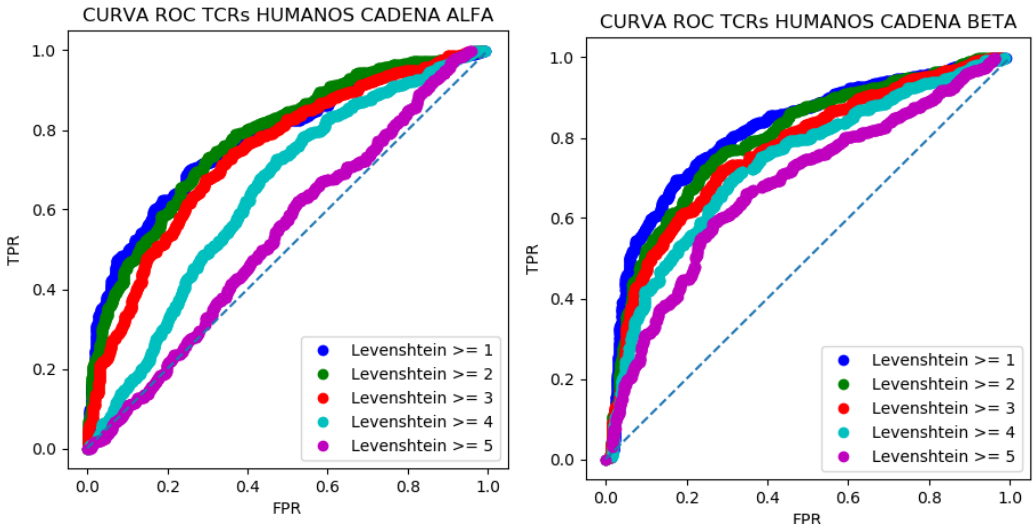

Fig. 4.16: Verdaderos positivos en función de falsos positivos en la cadena alfa en humanos (izquierda), la cadena beta en humanos (derecha). El largo de kmeros para alfa es 4 y para beta es 5.

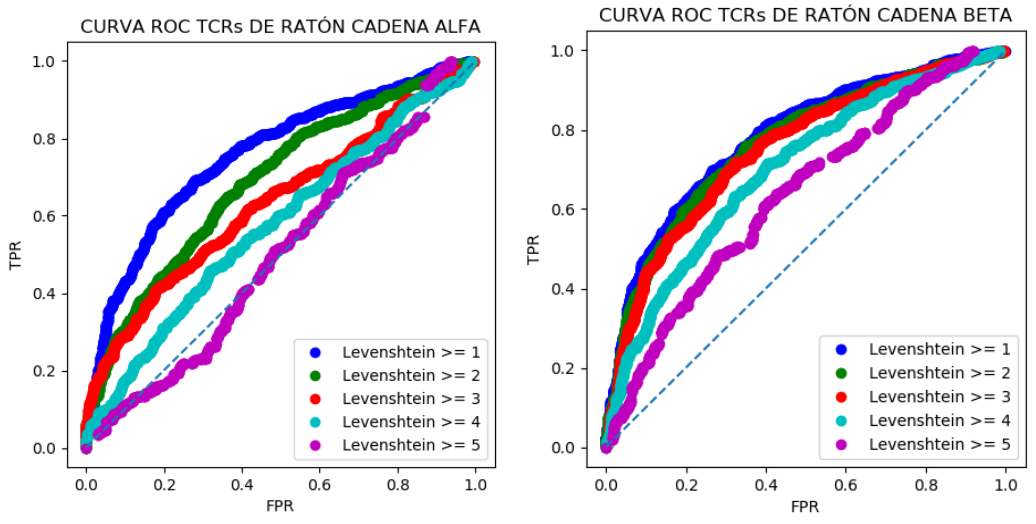

Fig. 4.17: Verdaderos positivos en función de falsos positivos en la cadena alfa en ratones (izquierda) y cadena beta en ratones (derecha). El largo de kmeros para alfa es 4 y para beta es 5.

En los gráficos 4.16 y 4.17 se puede apreciar cómo a medida que aumenta la distancia de Levenshtein, la curva roc se asemeja más a la recta y = x. Esto puede deberse a que cuanto

mayor es la distancia de Levenshtein mayor es la diferencia entre los datos del conjunto de *train* y los del conjunto de *test*, o también puede ser consecuencia de que el conjunto de *train* tenga menor cantidad de datos. Sin embargo, hasta Levenshtein 3 la curvatura se mantiene significativamente pronunciada (excepto en la cadena alfa en ratones). Podemos ver que las curvas que corresponden a la cadena beta son más pronunciadas, lo cual implica que el algoritmo es capaz de conseguir más verdaderos positivos con menos falsos positivos. Además, como esperábamos, los resultados de la cadena alfa para ratones se aproximan más al azar, ya que el gráfico de *hit-rate* (fig: 4.7) para el mismo parecía no evidenciar predicción. La excepción parece ser el conjunto de *train* con una distancia de Levenshtein mayor que 1, pero como mencionamos anteriormente, puede que en este caso existan elementos en el conjunto de *train* similares a los de su conjunto de *test*. Hemos visto qué ocurre con la especificidad en las cadenas alfa y beta, pero queremos saber qué ocurre cuando medimos la especificidad y se utilizan ambas cadenas (fig 4.10.1).

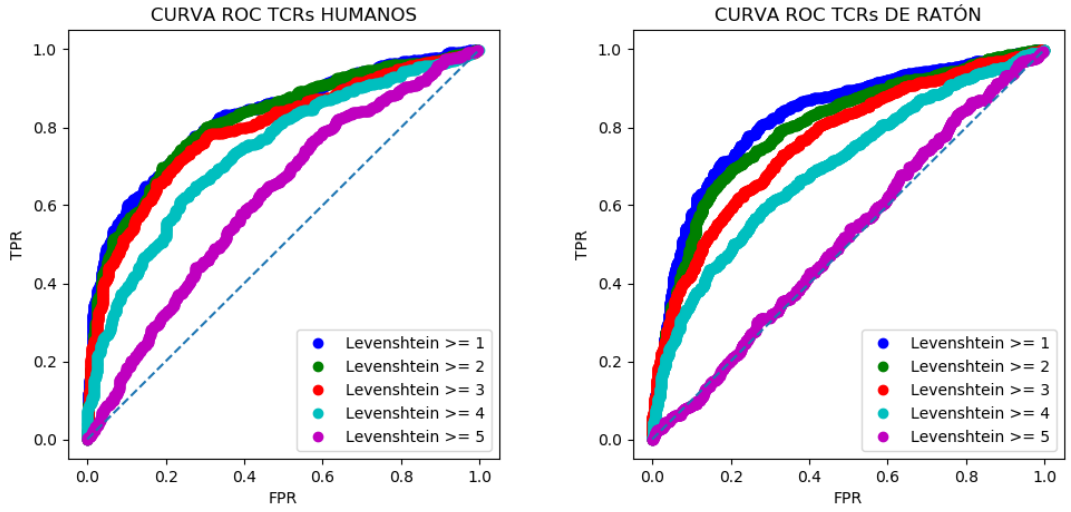

Fig. 4.18: Verdaderos positivos en función de falsos positivos utilizando ambas cadenas en humanos (izquierda) y ratones (derecha). El largo de kmeros para alfa es 4 y para beta es 5.

4.10.2. *Recall* y Precisión

Recall y Precisión son medidas contrapuestas. La primera cuenta la proporción de verdaderos positivos respecto al total de verdaderos positivos y falsos negativos, en otras palabras, la proporción de positivos que fueron considerados por el algoritmo como tales respecto al total de positivos. Mientras más alta sea esta medida, significa que la probabilidad de considerar a un positivo real como tal es mayor. Por otro lado, "precisión"muestra la proporción de verdaderos positivos respecto a la cantidad total de datos considerados positivos por el algoritmo. Cuando esta medida es alta, significa que la probabilidad de considerar un negativo como positivo es baja. Si el umbral utilizado para determinar si una predicción es correcta de acuerdo a su especificidad de unión es demasiado bajo la *recall* subirá, ya que estamos considerando más positivos. Por lo tanto, es más probable

incluir verdaderos positivos, pero aumentará la probabilidad de incluir también negativos, con lo cual la precisión disminuirá. Y a la inversa, si el umbral es demasiado alto aumenta la probabilidad de excluir negativos, aumentando la precisión, pero excluyendo también positivos, disminuyendo el *recall*. A continuación graficamos la precisión en función de la *recall* calculando ambas medidas con un umbral equivalente a cada especificidad registrada por el algoritmo, poniendo un punto en la recta por cada medición.

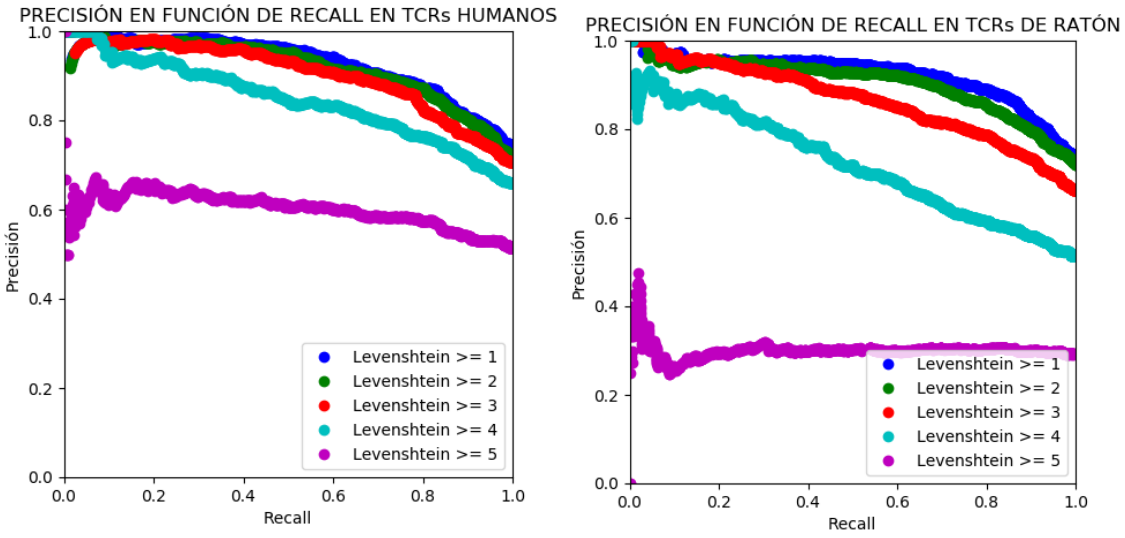

Fig. 4.19: precisión (verdaderos positivos sobre verdaderos positivos + falsos positivos) en humanos (izquierda) y ratones (derecha) en función de *Recall* (verdaderos positivos sobre verdaderos positivos + falsos negativos). Se utilizó el set de datos de humanos. El largo de kmeros para alfa es 4 y para beta es 5.

Cuanto mayor sea el área bajo la curva, significa que mejor funciona el algoritmo. Es decir que hasta cierto punto se puede aumentar la especificidad excluyendo negativos sin excluir positivos. En la figura 4.19 se puede ver que con un recall de aproximadamente 0.8, la precisión también es aproximadamente 0.8 en ambos casos. Es decir, tanto para humanos como para ratones existe un umbral para la especificidad en donde el 80 % de los casos marcados como positivos son verdaderos, y el 80 % de los valores verdaderos fueron marcados como positivos.

4.10.3. Coeficiente de correlación de Matthews

Ahora bien, queremos saber a partir de qué valor de especificidad es confiable una predicción usando este algoritmo. Para eso, calculamos el coeficiente de Matthews [15] para cada valor de especificidad registrado por el algoritmo.

Sea P_v cantidad de verdaderos positivos, P_f cantidad de falsos positivos, N_v cantidad de verdaderos negativos y N_f cantidad de falsos negativos, definimos el coeficiente de Matthews como: $MCC = \frac{P_v * N_v - P_f * N_f}{\sqrt{(P_v+N_v)*(P_f+N_f)*(P_v+P_f)*(N_v+N_f)}}$

Mientras más alto sea este valor, significa que los resultados del experimento son más confiables.

Calculamos entonces el coeficiente de Matthews para cada especificidad registrada. En la cadena alfa, el máximo coeficiente registrado con Levenshtein 3 es 0.373, con la especificidad en 1.821, mientras que en la cadena beta, el máximo que se registró fue 0.42386, alcanzando dicho valor en 2.1795

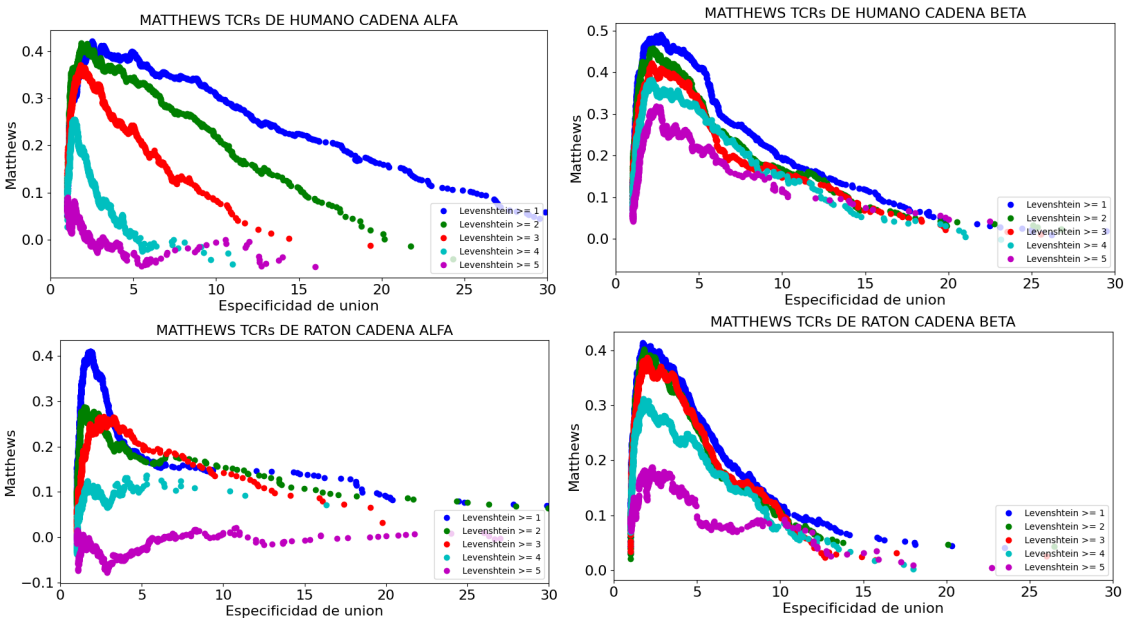

Fig. 4.20: Coeficiente de correlación de Matthews en función a la especificidad de la predicción en la cadena alfa en humanos (arriba izquierda), beta en humanos (arriba derecha), alfa en ratones (abajo izquierda) y beta en ratones (abajo derecha). El largo de kmeros elegido para la cadena alfa es 4 y para la beta 5. Se grafícan los conjuntos con distintas distancias de Levenshtein

Se puede apreciar en las gráficas 4.20 que el máximo coeficiente de Matthews al analizar la cadena beta suele ser mayor que cuando se utiliza la cadena alfa y además dicho máximo se alcanza con una especificidad más alta. Esto se ajusta a la teoría que marca que la cadena beta participa más activamente en la unión con el antígeno, con lo cual las pequeñas variaciones en la misma serán más críticas por lo que la cadena beta aportará más información que la alfa. Esta diferencia no se pudo apreciar en el *hit-rate*, probablemente porque la base de datos utilizada para generar los conjuntos de entrenamiento es acotada, pero se puede ver ahora que utilizando la cadena beta se obtienen mejores resultados.

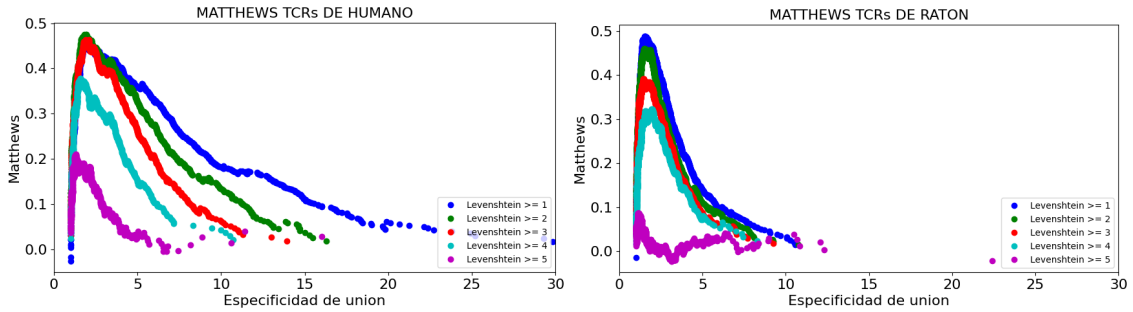

Fig. 4.21: Coeficiente de correlación de Matthews en función a la especificidad de la predicción en humanos (izquierda) y ratones (derecha). El largo de kmeros elegido para la cadena alfa es 4 y para la beta 5. Se grafican los conjuntos con distintas distancias de Levenshtein

Finalmente aplicamos el coeficiente de Matthews a predicciones con ambas cadenas(fig 4.21) (utilizando la suma para combinar la información). Con la distancia Levenshtein en 3, se puede observar que el máximo coeficiente es mayor en este caso que cuando se testea sólo la cadena beta o la cadena alfa.

Levenshtein	Esp. de unión	Matthews	# aceptados	% aceptados
L $\geq$ 1	1.720	0.470	767	69,47 %
L $\geq$ 2	1.749	0.474	705	66,32 %
L $\geq$ 3	1.672	0.449	669	64,39 %
L $\geq$ 4	1.887	0.356	687	48,25 %
L $\geq$ 5	1.236	0.206	791	73,33 %

Como se puede ver en la tabla, la especificidad óptima se mantiene entre 1.6 y 1.75 para la distancia de Levenshtein. El coeficiente de Matthews es bastante bueno, relativamente cercano a 0.5, hasta que se alcanza la distancia de Levenshtein 4, en donde se observa un decremento importante en el coeficiente. El porcentaje de aceptados tampoco varía demasiado en las distancias de Levenshtein de 1 a 3 y decae considerablemente en Levenshtein 4, ya que la capacidad predictiva disminuye pero no desaparece totalmente, entonces, descartando la mayoría de las predicciones, se puede obtener un bajo porcentaje de predicciones fiables. Cuando Levenshtein llega a 5, se pierde absolutamente la capacidad predictiva. El porcentaje de aceptados aumenta drásticamente, ya que la especificidad calculada en las predicciones es mínima, con lo cual si la especificidad tope se mantiene baja, existen posibilidades de que por azar la predicción sea correcta, pero si se aumenta el tope de especificidad, la misma es descartada, con lo cual el coeficiente de Matthews disminuye.

4.11. Complejidad

Si se intentara comparar un TCR con todos los TCRs en el conjunto de *train* utilizando la distancia de Levenshtein, la complejidad sería O(|TCR| * |TCRTrain| * N), siendo N la cantidad de kmeros en el conjunto de *train* y TCRTrain el TCR más largo de este conjunto. Una de las ventajas de nuestro algoritmo respecto a los que usan alineamiento o Levenshtein es su baja complejidad temporal. Nuestro algoritmo en la etapa de predicción debe recorrer el TCR y crear los kmeros. La complejidad de esto es O((|TCR| - K)*K), ya que el TCR tiene (|TCR| - K) kmeros y cada uno tiene un largo de K. Luego, cada uno

de ellos debe ser encontrado en un diccionario en memoria. Comparar dos kmeros tiene una complejidad de O(K), y la búsqueda en un diccionario, por ejemplo si asumimos que es un AVL, es O(log n).

En resumen, la complejidad de la etapa de predicción es de $O((|TCR| - K) * (logN) * K^2)$. Hay que tener en cuenta que el valor de K es bajo, como dijimos anteriormente, ronda entre 3 y 6 para que el algoritmo sea óptimo (y esto se ha comprobado en dos especies distintas). La complejidad sin considerar K sería O(|TCR| * (log N))

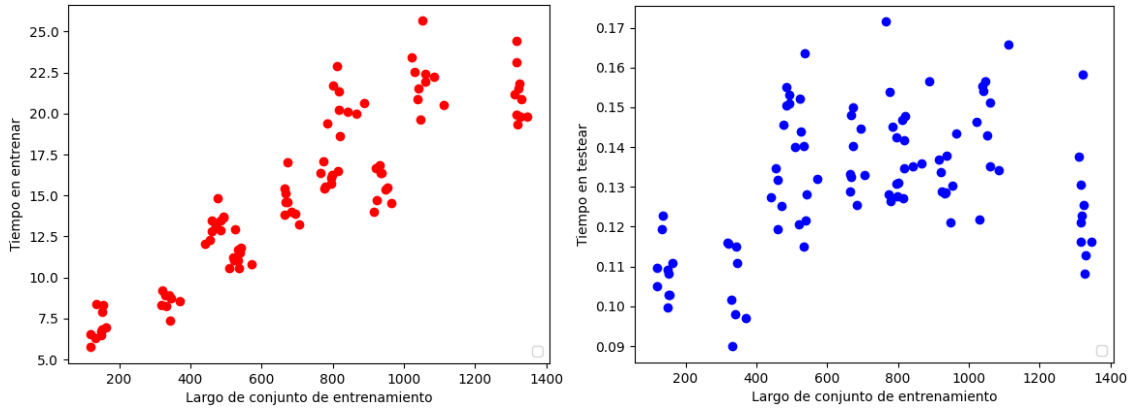

Fig. 4.22: Gráficos del tiempo de entrenamiento (izquierda) y de predicción (derecha) en microsegundos en función del largo de la base de datos de entrenamiento.

Como se muestra en fig 4.22 se puede apreciar un aumento en el tiempo de entrenamiento cuando la base de datos de *train* es más grande, pero no se apreciaron cambios significativos en la etapa de predicción. Tal vez esto se deba a que la variación aleatoria en el tiempo de ejecución sea alta respecto a su duración promedio o bien que el aumento en la base de datos no aporta nuevos kmeros.

5. CONCLUSIONES

Utilizando una variante de la similitud por kmeros, conseguimos producir un algoritmo con la capacidad de aprender patrones del conjunto de datos, simple y de baja complejidad respecto a estrategias como el alineamiento. La técnica empleada es puntualmente útil cuando no se pueden conocer todas las variables de un problema, por lo cual se debe trabajar con probabilidades. El uso de esta métrica fue probado para la unión de TCRs pero puede no limitarse al mismo, ya que podría aplicarse para medir la afinidad entre otros grupos de proteínas.

El trabajo desarrollado por VDJdb [28] nos brindó información recopilada por distintos grupos de investigación en un formato unificado que pudimos adaptar fácilmente como entrada del problema. Además, el hecho de que probamos el algoritmo con dos especies de mamíferos diferentes refuerza la teoría de que los resultados son universales en mamíferos, pero hay que tener en cuenta que sólo se ha podido probar en dichas especies, ya que no se disponía de más datos. Si bien estos conjuntos de datos no eran tan grandes como deseábamos, parecería tener tamaño suficiente para que el sistema pueda aprender patrones de los mismos.

Mediante el análisis de los datos vimos que el conjunto de TCRs de humanos está desbalanceado en VDJdb. Es decir que existe una cantidad significativamente mayor de TCRs que reconocen a un determinado antígeno. Las pruebas con este conjunto demostraron que en estos casos el sistema tiene una ligera preferencia a eligir al antígeno con más TCRs en la predicción. Esto sucede porque como existen más registros para dicho antígeno, kmeros que no son vitales para la interacción pueden aparecer para el antígeno en mayor cantidad. Sin embargo, la cadena beta de los ratones fue lo suficientemente variada y evidenció que el algoritmo genera resultados superiores al 50 % para la mayoría de los antígenos, si bien el conjunto de entrenamiento posee numerosos antígenos con distintas cantidades de TCRs. Finalmente, aunque el índice de *hit-rate* no superó el 70 %, los resultados demostraron que sí existe un patrón entre los kmeros que detectan un mismo antígeno, el cual puede ser identificado por un algoritmo.

Por las pruebas realizadas, se determinó que el largo óptimo del kmero en la secuencia alfa está entre 3 y 4, mientras que en el beta ronda entre 4 y 6. En el caso de la cadena beta, esto se refuerza por el hecho de que aplicamos el procedimiento en dos conjuntos de datos independientes (de TCRs de ratones y de humanos) y obtuvimos resultados similares. Si se quisiera adaptar el algoritmo para utilizarlo con otro problema de unión de proteínas, el k óptimo debería ser recalculado.

En el proyecto analizamos el algoritmo para las cadenas alfa y beta. Si bien la cadena beta obtuvo un buen *hit-rate* y, en el caso de humanos, la alfa también, cuando se utilizaron ambas cadenas, combinándolas tanto con la multiplicación como con el promedio, los resultados fueron aún mejores. Mientras otros estudios analizan la cadena beta solamente, en este caso pudimos determinar que la cadena alfa contribuye al reconocimiento del antígeno. Aún más importante, aporta información relevante para que el algoritmo pueda entrenar y predecir. Una hipótesis que no analizamos en este proyecto es si la cadena alfa y la beta tienen que tener un determinado patrón combinado para reconocer al antígeno. En un futuro se puede ampliar el problema para analizar determinados patrones en la cadena alfa considerando los encontrados en beta y viceversa.

Como dijimos anteriormente, en el algoritmo no contamos apariciones múltiples de un kmero en un TCR. Esto lo hicimos porque consideramos que el TCR sólo une en una región, pero podría ocurrir que con la repetición de un kmero aumenten las probabilidades de que ocurra la unión, con lo cual una posible variante a considerar en el futuro sería considerar dichas apariciones múltiples.

Para el proyecto consideramos que la posición del kmero era independiente en las predicciones, sin embargo esto podría no ser cierto. Una variante a probar puede ser registrar en la etapa de entrenamiento la posición donde se encuentra cada kmero y en la etapa de *test* ponderar la probabilidad calculada con la diferencia entre la posición donde el kmero fue encontrado en la etapa de *test* respecto a donde fue encontrado en el TCR a predecir.

Otra medida importante que acompaña a la predicción es la 'especificidad', para determinar qué tan afín al antígeno predicho es el TCR analizado, respecto del resto de los antígenos. Comparar el puntaje obtenido para el antígeno predicho con aquel que poseía el segundo mejor puntaje resultó una buena métrica. Utilizando el coeficiente de Matthews pudimos encontrar para cada conjunto de entrenamiento el valor que actúa como límite para determinar si una predicción es fallida, ya sea porque el TCR es realmente promiscuo o porque no reconoce a ningún antígeno registrado (o bien porque no tenemos información suficiente en la base de datos de entrenamiento).

Si bien en este proyecto se analizaron kmeros de largo fijo, en la realidad la secuencia que interactúa con el antígeno no siempre tiene el mismo largo. Es cierto que dicha longitud no varía demasiado, y además al analizar kmeros consecutivos con un k cercano a la longitud real de la subsecuencia que interactúa con el antígeno se consigue un buen resultado. Sin embargo, el *hit-rate* tal vez mejore si se considera analizar subsecuencias de largo variable. Cabe aclarar que la complejidad temporal y espacial en este caso será más grande.

En esta tesis hemos analizado la interacción de dos de los CDRs del TCR (los CDR3) con el péptido foráneo, porque dichas subsecuencias son las más importantes en la interacción con el péptido. El antígeno reconocido consta en realidad de dicho péptido y el MHC unido al mismo, y el TCR completo posee seisCDRs para reconocer la totalidad del antígeno. Modificando el problema ligeramente, se puede usar este modelo para analizar todos los CDRs y el antígeno completo. En este caso, la complejidad temporal y espacial de una predicción será lineal respecto a la suma del largo de todos los CDRs.

Durante el desarrollo del algoritmo que implementa el problema probamos utilizar la matriz de comparación BLOSUM [13], pero las pruebas realizadas no obtuvieron buenos resultados. Sin embargo, es cierto que a nivel estructural la diferencia entre dos aminoácidos no siempre es la misma, con lo cual puede existir alguna métrica que compare aminoácidos similar a BLOSUM, que pueda ser aplicada para hacer una comparación no booleana entre kmeros. De esta forma kmeros más 'similares' aportarían menos diferencia en una predicción. De momento no hemos encontrado una métrica similar que mejore los resultados, o bien no encontramos una buena forma de comparar kmeros aplicándola.

En las mediciones tomadas notamos que a medida que aumenta la distancia de Levenshtein que colocamos como umbral para descartar elementos del conjunto de *test*, el poder predictivo decrementaba. Sin embargo, esto puede deberse a que los datos entre el conjunto de *train* y el de *test* son más distintos y/o simplemente a que hay una menor cantidad de datos en los conjuntos de *train*. A futuro, pueden realizarse mediciones en donde se generen conjuntos de *train* descartando elementos al azar, para comparar cuánto

cae el poder predictivo en ese caso.

En el desarrollo del trabajo, desde VDJdb [28] sólo pudimos obtener TCRs de humano y de ratón que posean información de ambas cadenas. Esto se debe a que para conseguir información de ambas cadenas se precisan experimentos más costosos que los utilizados para secuenciar sólo la cadena beta. Esperamos que a futuro se incorpore información de otras especies, en una cantidad suficiente para poder repetir las pruebas con otros conjuntos de datos adicionales.

A diferencia de otros métodos, el algoritmo propuesto no depende del azar o del orden en el cual recibe los datos y por lo tanto es totalmente determinístico. Sin embargo, la principal ventaja es la complejidad de ejecución. Existen muchas estrategias para intentar abordar el problema de predecir la unión de antígeno - TCR [14] [20]. Optamos por la opción presentada ya que es un algoritmo sencillo y a la vez tiene correlación con la realidad del problema.

El modelo matemático planteado tiene una relación directa con el problema real que representa predecir uniones antígeno - TCR, ya que en la biología el TCR identifica a su antígeno utilizando fracciones codificadas por subsecuencias en su cadena de aminoácidos. Nuestro modelo se centra en aprender cuáles subsecuencias en la secuencia de aminoácidos del CDR son importantes. Si bien no deja de ser una heurística, logramos conseguir resultados consistentes con un dataset reducido.

Bibliografía

[1] Abul K. Abbas, Andrew H. H. Lichtman, Shiv Pillai. 2017. ***Celular and Molecular Immunology-Elsevier*** (9º Edition). Elsevier.

[2] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. **Basic local alignment search tool.** J Mol Biol, 1990. 215(3):403-10.

[3] De Robertis Eduardo, Hib José. 2004. ***Fundamentos de Biología celular y molecular*** (4º Edition). El Ateneo.

[4] Carrillo, H., & Lipman, D. (1988). **The multiple sequence alignment problem in biology.** SIAM journal on applied mathematics, 48(5), 1073-1082.

[5] Altschul SF, Madden TL, Schäffer AA, et al. **Gapped BLAST and PSI-BLAST: a new generation of protein database search programs.** Nucleic Acids Res, 1997. 25(17):3389-3402.

[6] Eddy SR. **Accelerated Profile HMM Searches.** PLoS Comput Biol, 2011. 7(10):e1002195.

[7] Dash P, Fiore-Gartland AJ, Hertz T, et al. **Quantifiable predictive features define epitope-specific T cell receptor repertoires.** Nature, 2017. 547(7661):89-93.

[8] Krangel, M. S. (2009). **Mechanics of T cell receptor gene rearrangement.** Current opinion in immunology, 21(2), 133-139.

[9] eters B, Nielsen M, Sette A. **T Cell Epitope Predictions.** Annu Rev Immunol, 2020. 38:123-145.

[10] Ghasemi F, et al. **Using L-arabinose for production of human growth hormone in Escherichia coli, studying the processing of gIII: hGH precursor.** Iran J Biotechnol. 2004;2:250–60.

[11] Gielis S, Moris P, Bittremieux W, et al. **Detection of Enriched T Cell Epitope Specificity in Full T Cell Receptor Sequence Repertoires.** Front Immunol, 2019. 10:2820.

[12] Glanville, J., Huang, H., Nau, A. et al. **Identifying specificity groups in the T cell receptor repertoire.** Nature, (2017). 547, 94–98.

[13] Henikoff, S., & Henikoff, J. G. (1992). **Amino acid substitution matrices from protein blocks.** Proceedings of the National Academy of Sciences of the United States of America, 89(22), 10915–10919. https://doi.org/10.1073/pnas.89.22.10915

[14] Huang, J., Zarnitsyna, V. I., Liu, B., Edwards, L. J., Jiang, N., Evavold, B. D., & Zhu, C. (2010). **The kinetics of two-dimensional TCR and pMHC interactions determine T-cell responsiveness.** Nature, 464(7290), 932-936.

[15] Matthews, B. W. (1975). **Comparison of the predicted and observed secondary structure of T4 phage lysozyme.** Biochimica et Biophysica Acta (BBA)-Protein Structure, 405(2), 442-451.

[16] Meysman, P., De Neuter, N., Gielis, S., Thi, D. B., Ogunjimi, B., & Laukens, K. (2018). **The workings and failings of clustering T-cell receptor beta-chain sequences without a known epitope preference.** bioRxiv, 318360.

[17] Lanzarotti E, Marcatili P, Nielsen M. **T-Cell Receptor Cognate Target Prediction Based on Paired α and β Chain Sequence and Structural CDR Loop Similarities.** Front Immunol, 2019. 10:2080.

[18] Lanzarotti E, Marcatili P, Nielsen M. **Identification of the cognate peptide-MHC target of T cell receptors using molecular modeling and force field scoring.** Mol Immunol, 2018. 94:91-97.

[19] Liu JS, Lawrence CE. **Bayesian inference on biopolymer models.** Bioinformatics. 1999 Jan;15(1):38-52. doi: 10.1093/bioinformatics/15.1.38. PMID: 10068691.

[20] Liu, I. H., Lo, Y. S., & Yang, J. M. (2011). **PAComplex: a web server to infer peptide antigen families and binding models from TCR–pMHC complexes.** Nucleic acids research, 39(suppl_2), W254-W260.

[21] Luczak BB, James BT, Girgis HZ. **A survey and evaluations of histogram-based statistics in alignment-free sequence comparison.** Brief Bioinform, 2019. 20(4):1222-1237.

[22] Petroni, F., Serva, M., & Volchenkov, D. (2015). **Levenshtein's Distance for Measuring Lexical Evolution Rates.** In Nonlinear Dynamics New Directions (pp. 215-240). Springer, Cham.

[23] Riedel, Stefan. **Edward Jenner and the history of smallpox and vaccination.** Baylor University Medical Center Proceedings. Vol. 18. No. 1. Taylor & Francis, 2005.

[24] Trevor Hastie, Robert Tibshirani, Jerome Friedman. 2009. ***The Elements of Statistical Learning: Data Mining, Inference, and Prediction.*** (2º Edition). Springer

[25] Tsuchiya, Y., Namiuchi, Y., Wako, H., & Tsurui, H. (2018). **A study of CDR 3 loop dynamics reveals distinct mechanisms of peptide recognition by T-cell receptors exhibiting different levels of cross-reactivity.** Immunology, 153(4), 466-478.

[26] Schmidler SC, Liu JS, Brutlag DL. **Bayesian segmentation of protein secondary structure.** J Comput Biol, 2000. 7(1-2):233-48.

[27] Joglekar AV, Li G. **T cell antigen discovery.** Nat Methods, 2020.

[28] Shugay M, Bagaev DV, Zvyagin IV, Vroomans RM, Crawford JC, Dolton G, Komech EA, Sycheva AL, Koneva AE, Egorov ES, Eliseev AV, Van Dyk E, Dash P, Attaf M, Rius C, Ladell K, McLaren JE, Matthews KK, Clemens EB, Douek DC, Luciani F, van Baarle D, Kedzierska K, Kesmir C, Thomas PG, Price DA, Sewell AK, Chudakov

DM. **VDJdb: a curated database of T-cell receptor sequences with known antigen specificity.** Nucleic Acids Res, 2018. 46(D1):D419-D427.

[29] Watson JD, Crick FH. **A structure for deoxyribose nucleic acid.** Nature 1953;171:737–738

[30] Vinay Kumar, Megan E. McNerney. **A new self: MHC-class-I-independent Natural-killer-cell self-tolerance**. Nat Rev Immunol 5, 363–374 (2005).

[31] Zhao, C., & Sahni, S. (2019). **String correction using the Damerau-Levenshtein distance.** BMC bioinformatics, 20(11), 1-28.

[32] Zvyagin IV, Tsvetkov VO, Chudakov DM, Shugay M. **An overview of immunoinformatics approaches and databases linking T cell receptor repertoires to their antigen specificity.** Immunogenetics, 2020. 72(1-2):77-84.