Universidad del Bío-Bío, Chile

Facultad de Ciencias Empresariales

Departamento de Sistemas de Información

# Identificación de nuevos factores discriminantes basados en distancia entre aminoácidos en zona de interacción entre proteínas

Tesis presentada por Juan Francisco Saavedra Molina
para obtener el grado de Ingeniero Civil Informático
Dirigida por Tatiana Gutiérrez-Bunster

2017

# Resumen

Las proteínas son herramientas biológicas, responsables de la gran mayoría de las funciones biológicas de los seres vivos, además de servir como material estructural de éstos. Las proteínas raramente trabajan solas, estas interactúan con ADN, ARN, lípidos y otras proteínas. Las interacciones proteína-proteína en particular, son de vital importancia ya que actúan en la mayoría de los procesos biológicos, tales como la biosíntesis y traducción de señales. Por este motivo áreas como la bioinformática y bioquímica, han intentado comprender las interacciones proteína-proteína. Métodos como la cristalografía de rayos X y de resonancia magnética nuclear han generado gran cantidad de información con respecto a estas interacciones, como la estructura tridimensional de estos complejos. En esta investigación se utilizó los datos de estructuras tridimensionales previamente clasificados (clasificados en complejos transitorios y permanentes) para determinar si la distancia entre aminoácidos pertenecientes a la zona de interacción de los complejos proteicos, influyen en el tipo de interacción que se produce. Los resultados que se obtuvieron indican que existe dicha relación, pero estas características, en la forma en que se generaron y aplicaron, no resultaron tan determinantes como otros métodos como por ejemplo la utilización de las energías producidas por la interacción de aminoácidos pertenecientes a la zona de interacción. Se obtuvo un 75 % de complejos proteicos clasificados correctamente utilizando las características de distancia generadas en esta investigación.

***Palabras Clave*** — distancias, enfoque de coordenadas, interacciones proteína-proteína, minería de datos.

# Índice general

# Lista de figuras

# Índice de tablas

# Capítulo 1

# Introducción

Una de las moléculas que desempeña un mayor número de funciones en las células de todos los seres vivos, son las proteínas. Estas, además de formar parte de la estructura básica de los tejidos y desempeñar funciones metabólicas y reguladoras, son además los elementos que definen la identidad de cada ser vivo por ser la base de la estructura del código genético (ADN) y de los sistemas de reconocimiento de organismos extraños en el sistema inmunitario.

A raíz de sus variadas funciones y posibles empleos en el ámbito de creación de medicamentos, los estudios de la proteína son variados y llamativos. El ámbito que nos convoca en particular con respecto a la proteína, es el estudio de interacciones proteína-proteína.

Las interacciones proteína-proteína hacen referencia a su asociación, y el estudio de esas asociaciones puede observarse desde la perspectiva de la bioquímica, transducción de señales[1] y redes de interacción de proteínas.

Estas interacciones proteína-proteína desempeñan un papel fundamental en distintos aspectos de la organización estructural y funcional de la célula, para muchos procesos celulares cuya extensión va desde la formación de estructuras macromoleculares y complejos enzimáticos, hasta la regulación y transducción de señales. Las interacciones proteína-proteína pueden clasificarse en interacciones transitorias (cortas en el tiempo) y interacciones permanentes (duraderas en el tiempo)[7].

La comprensión de estas interacciones, otorga información útil acerca del mecanismo molecular de funcionamiento de la célula, ayudando así en el diseño de fármacos más específicos, como también en la ingeniería de procesos celulares.

Existen varios métodos bioquímicos para determinar el tipo de interacción proteína-proteína

[1] Traducción de señales: Se refiere al proceso celular en el cual una señal se mueve desde el exterior de una célula a su interior. La generación y la transmisión intracelular de estas señales se realiza por medio de interacciones proteína-proteína.

que existe en un complejo proteico, pero estos métodos son costosos y consumidores de tiempo. Sin embargo, durando muchos años estos métodos han generado un gran volumen de datos con respecto a la estructura tridimensional de los complejos, datos que se han utilizado en la área bioinformática arduamente.

Existen diversos estudios enfocados en la predicción de interacciones proteína-proteína, como por ejemplo estudios que hablan sobre la asociación entre la energía contribuida por las interacciones de los aminoácidos que conforman las interacciones proteína-proteína y el tipo de interacción que se conforma [8].

Este estudio, siguiendo la hipótesis sobre la importancia que representan la zona de interacción de un complejo proteico de [8], intenta dilucidar si la distancia entre los aminoácidos presentes en la zona de interacción de un complejo proteico, influye el el tipo de interacción que este tiene (transitoria o permanente).

## 1.1. Objetivos

El objetivo general de esta investigación es analizar las distancias entre aminoácidos presentes en la zona de interacción de proteínas, con el fin de averiguar si el tipo de interacción (permanente o transitoria) en un complejo proteico depende de la distancia entre los aminoácidos participantes, considerando además las energías producidas en dicha interacción.

Para lograr el objetivo general se deben completar los siguientes objetivos específicos:

1. Estudiar algoritmos y métodos de medición de distancias masivas utilizando la información tridimensional de complejos proteicos (transitorios y permanentes) y desarrollar un programa en lenguaje Python que permita la generación de nuevas características para su posterior evaluación.

2. Generar matrices de características con los resultados del objetivo anterior, para luego ser evaluadas utilizando el software de aprendizaje automático WEKA, y así obtener resultados de eficacia con respecto a la predicción de interacciones proteína proteína de un conjunto de datos de complejos proteicos ya clasificados.

3. Analizar los resultados entregados por el software WEKA mediante el empleo de métodos estadísticos.

## 1.2. Organización

- Capítulo 2: Estado del arte

En este capítulo se hablará sobre las proteínas, su importancia, sobre las interacciones entre ellas. Además introduciremos a la bioinformática y su búsqueda por predecir este tipo de interacciones. Veremos la minería de datos y su importancia en la bioinformática.

- Capítulo 3: Estudio del problema

Veremos la metodología que se utilizará en el estudio. El problema de la predicción de interacciones proteína-proteína. Los enfoques de coordenadas que se utilizarán para calcular distancias entre proteínas. Y por último los métodos de cálculo de distancia que se utilizarán.

- Capítulo 4: Implementación de algoritmos

Veremos la implementación de algoritmos de enfoque de coordenadas y de distancias entre proteínas, y la generación de matrices de características.

- Capítulo 5: Resultados

Se utilizará el programa de aprendizaje automático WEKA para obtener una evaluación de las matrices con características de distancias. Luego utilizaremos los datos estadísticos para evaluar si existe una asociación entre las distancias entre proteínas y el tipo de interacción que se produce.

- Capítulo 6: Conclusiones y trabajos futuros

Se expondrán las conclusiones finales de este estudio, además de ofrecer indicaciones para trabajos futuros.

# Capítulo 2

# Estado del Arte

## 2.1. Introducción a la Bioinformática

Como consecuencia de la gran cantidad de datos biológicos generados desde principios de los años 90, principalmente del proyecto de genoma humano [9] que logró exitosamente determinar la secuencia de genoma humano además de los genes que contenía; fue que nació la necesidad de analizar estos datos de una manera más eficiente, razón que volvió indispensable la utilización de herramientas computacionales para el análisis de estos.

Existe una amplia variedad de definiciones de bioinformática en la literatura, entre estas N. M. Luscombe, D. Greenbaum y M. Gerstein la definen como "la conceptualización de la biología en términos de macromoléculas para luego aplicar técnicas informáticas (provenientes de matemáticas aplicadas, ciencias computacionales y estadísticas) para comprender y analizar la información asociadas a estas moléculas biológicas, a gran escala" [10].

Otra definición entregada por *The Nacional Center For Biotechnologic Information* (centro nacional para información biotecnológica) propone que "la bioinformática es el campo de la ciencia en donde la biología, ciencias de la computación y tecnología de la información se unen en una sola disciplina" [11].

Podemos entonces sub-dividir la bioinformática en tres disciplinas: primero la creación de algoritmos con el propósito encontrar relaciones en grandes repositorios de datos, segundo el análisis e interpretación de datos como nucleótidos y aminoácidos, complejos proteicos, estructuras proteicas, entre otros, y por último el desarrollo de herramientas que permitan el fácil acceso y administración de estos datos.

Las ciencias de la computación y la biología molecular pueden verse como dos elementos muy distintos, pero la verdad es que la vida en sí es una tecnología de información, la fisiología

de cualquier organismo está determinada por sus genes, los cuales en su expresión más básica se pueden expresar como información digital. Lo mismo ocurre con las proteínas, su estructura tridimensional, como también su secuencia de aminoácidos, entre otras características, se pueden representar de manera digital [10].

### 2.1.1. Proteínas

Las proteínas son herramientas biológicas, responsables de la gran mayoría de las funciones biológicas de los seres vivos, además de servir como material estructural de éstos. Están formadas por la unión de aminoácidos por medio de enlaces covalentes llamados enlaces péptidos, por lo general llamamos proteína a la unión de más de 50 aminoácidos; la unión de menos de 10 aminoácidos se denomina oligopéptido, y de 10 a 50 aminoácidos son llamados polipéptidos.

Al estar formadas por la unión de aminoácidos, su composición química deriva de estos, por lo cual las proteínas están formadas por Carbono, Hidrógeno, Oxígeno, Nitrógeno y en no todos los casos, Azufre.

Los seres vivos producen un gran número proteínas distintas, esto ya que existen 20 aminoácidos diferentes, y en teoría se pueden crear proteínas de cualquier tamaño y de cualquier organización distinta de secuencias, por esta razón la cantidad total de posibles proteínas es astronómica.

Cabe señalar que no todas las secuencias dan como resultado una proteína útil, de los billones de posibles combinaciones no más de dos millones son realmente producidas por los seres vivos. Esta discrepancia existente entre las posibles combinaciones y el número real de proteínas funcionales es prueba del complejo conjunto de propiedades estructurales y funcionales de las proteínas de origen natural que han evolucionado a lo largo de miles de millones de años.

### 2.1.2. Aminoácidos

Los aminoácidos son las unidades básicas de los péptidos y las proteínas. Son moléculas orgánicas formadas por un grupo amino (NH2) en un extremo de la molécula y un grupo ácido carboxílico (COOH) en el otro [12]. Existen más de 300 aminoácidos diferentes en la naturaleza pero sólo 20 forman las proteínas, estos 20 aminoácidos son denominados L-alfa-aminoácidos [13].

Todos los L-alfa-aminoácidos poseen un grupo amino[1] y un grupo carboxilo[2] unidos a un átomo de carbono, y este a su vez se une a un átomo de hidrógeno y a una cadena lateral o radical R. La cadena lateral distingue un aminoácido de otro, ya que el resto de la estructura es igual para todos los L-alfa-aminoácidos. El carbono central es denominado carbono alfa, debido a que une al grupo amino, grupo carboxilo y radical R.

Es importante señalar que en nuestra investigación haremos una distinción en la estructura de los aminoácidos, entre la cadena lateral y el resto de la estructura, por lo tanto denominaremos como "Cadena Base" a la estructura compuesta por el carbono alfa, el átomo de hidrógeno, grupo carboxilo y el grupo amino.

En la Figura 2.1 podemos observar la representación general de todos los L-alfa-aminoácidos, siendo el carbono central el carbono alfa. Además podemos ver cuál es la Cadena Base (Estructura incluida en la sección segmentada de la figura) y cuál es la Cadena Lateral (Estructura excluida de la sección segmentada).

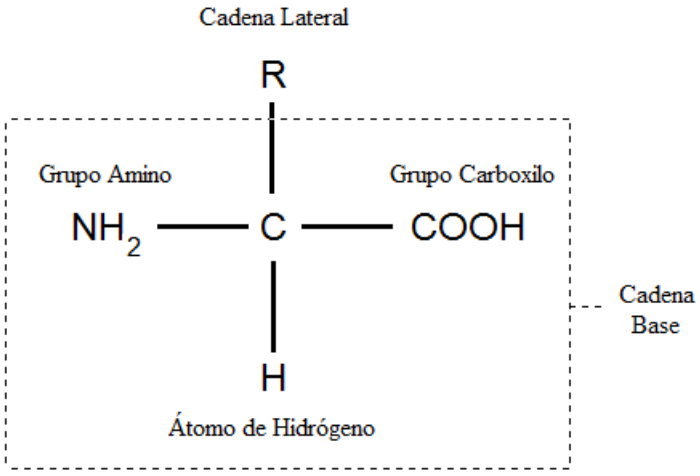


Figura 2.1: Fórmula estructural general para todos los aminoácidos.

En la Figura 2.2 podemos apreciar dos aminoácidos, uno estructuralmente más simple (Contiene menos elementos químicos), que vendría a ser la alanina, y otro más complejo, la glutamina. En los cuales podemos observar que la diferencia radica sólo en la cadena lateral. En el aminoácido alanina, la cadena lateral es un grupo metilo, y en la glutamina, la cadena lateral es un grupo amida.

[1] Grupo amino: grupo funcional derivado del amoniaco (NH3).

[2] Grupo carboxilo: grupo funcional orgánico que consiste en un átomo de carbono unido de forma doble a un átomo de oxígeno y unido por unión simple a un grupo hidroxilo.

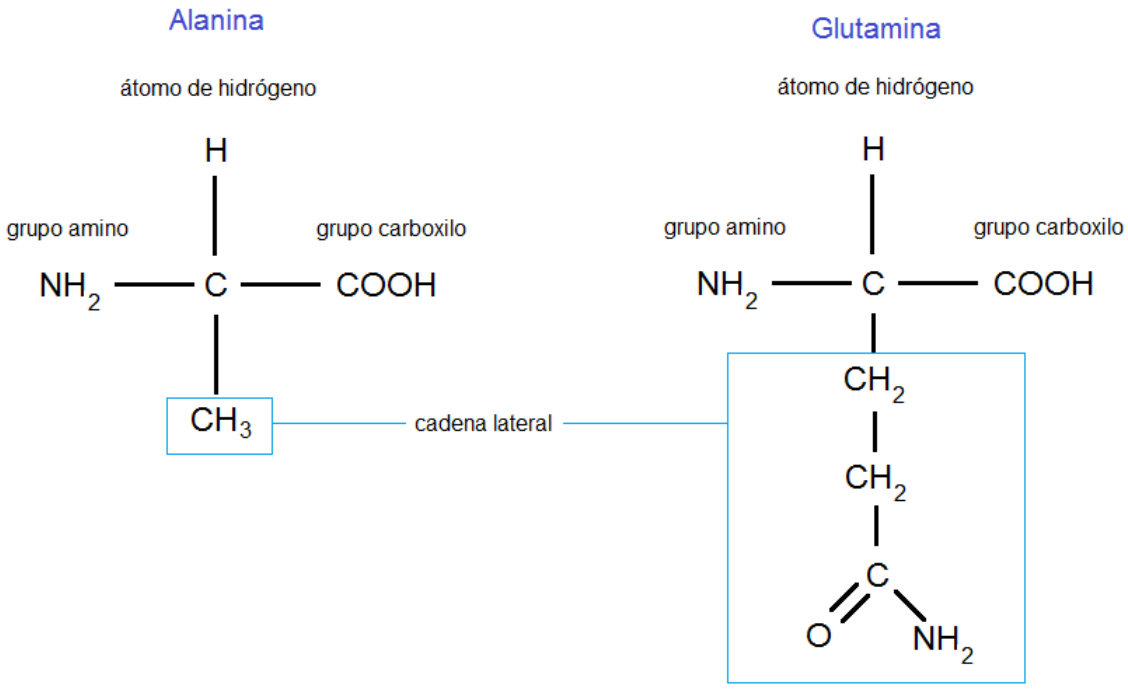


Figura 2.2: Fórmula estructural de los aminoácidos alanina y glutamina.

En la Tabla 2.1 podemos observar los 20 L-alfa-aminoácidos con su respectiva abreviatura.

| **Aminoácido** | **Abreviatura** | **Aminoácido** | **Abreviatura** |
|---|---|---|---|
| Alanina | Ala | Leucina | Leu |
| Arginina | Arg | Lisina | Lys |
| Asparagina | Asn | Metionina | Met |
| Ácido aspártico | Asp | Fenilalanina | Phe |
| Cisteína | Cys | Prolina | Pro |
| Glutamina | Gln | Serina | Ser |
| Ácido glutámico | Glu | Treonina | Thr |
| Glicina | Gly | Triptófano | Trp |
| Histidina | His | Tirosina | Tyr |
| Isoleucina | Lle | Valina | Val |

Tabla 2.1: L-alfa-Aminoácidos.

Cada proteína posee una estructura tridimensional única y la funcionalidad biológica de éstas, es a su vez única ya que depende de esta estructura. Por esta razón la bioinformática busca a

través del análisis de datos de proteínas conocidas, predecir qué tipo de estructuras forma una proteína. En el Apéndice C podemos ver las funciones que cumplen las proteínas en los procesos celulares.

### 2.1.3. Estructura de las proteínas

Hablar sobre la estructura de las proteínas es más complejo que referirse a otras moléculas más pequeñas, ya que las proteínas son macromoléculas muy complejas, hasta la más pequeña cadena polipéptida es casi imposible de comprender. Por esto los bioquímicos diferencian cuatro niveles estructurales en las proteínas, divididos acorde su complejidad. Estos niveles estructurales son: la estructura primaria, secundaria, terciaria y cuaternaria [14].

**Estructura primaria**: Es el nivel estructural más básico de una proteína. La estructura primaria corresponde a la secuencia de aminoácidos que conforman la proteína y es determinante en la función que cumple la proteína [15].

**Estructura secundaria**: Este nivel estructural es formado por varios patrones repetitivos consistentes en plegamientos que se generan en la proteína por la formación de puentes de hidrógeno entre los átomos que forman el enlace péptido. En este nivel estructural podemos ver a la proteína adoptándose espacialmente. Las dos estructuras más frecuentes y estables en este nivel son la hélice alfa y lámina beta [14].

**Estructura terciaria**: La estructura tridimensional completa de una proteína es llamada estructura terciaria. En este nivel estructural podemos observar en la proteína los dobleces y torcimientos de los elementos de la estructura secundaria para conseguir la máxima estabilidad o el estado de energía más bajo y se especifica la posición de cada átomo en la proteína, en los cuales se incluyen los átomos de las cadenas laterales. La estructura terciaria es única para cada proteína y es la responsable directa de las propiedades biológicas de ésta, ya que la propiedad que determina la interacción con diferentes ligandos es la disposición espacial de los grupos funcionales (grupo amino, carboxilo, etc) [14].

En la Figura 2.3 podemos observar una representación de cada nivel estructural.

Para obtener la información de la estructura tridimensional de las proteínas se utilizan los métodos de cristalografía de rayos X y de resonancia magnética nuclear de los cuales se hablará en la sección 2.1.7. La información de las coordenadas atómicas de la mayoría de las proteínas estudiadas por estos métodos se encuentran depositadas en la base de datos *Protein Data Bank* (Banco de datos de proteínas), de la cuál se hablará en la sección 2.1.4.

**Estructura cuaternaria**: Decimos que una proteína tiene una estructura cuaternaria, cuando tratamos con una proteína oligomérica, una proteína formada por múltiples polipéptidos. Un ejemplo de esto es la Hemoglobina, una proteína formada por cuatro cadenas polipéptidicas llamadas globinas [16].

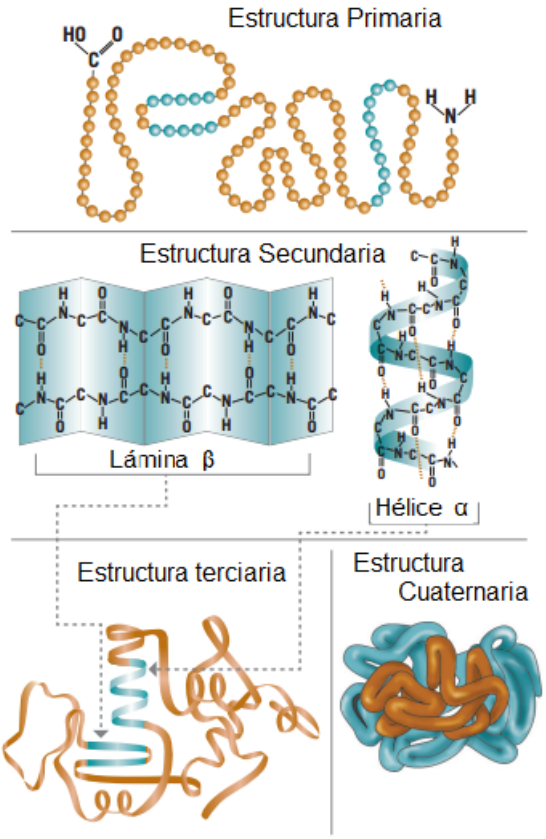


Figura 2.3: Niveles estructurales de las proteínas [1].

### 2.1.4. Protein Data Bank

*Protein Data Bank*(Banco de datos de proteínas) o comúnmente llamado PDB es el único repositorio mundial que alberga información de estructuras tridimensionales de macromoléculas como proteínas y ácidos nucleicos. Hoy en día la base de datos contiene más de cien mil entradas de datos de proteínas disponibles de manera gratuita para todo el público. La gran mayoría de los datos son conseguidos mediante a los métodos de difracción de rayos X y de resonancia magnética nuclear [17].

El PDB se estableció en el año 1971 por el doctor Walter Hamilton en el *Brookhaven Nacional Laboratory* (Laboratorio Nacional Brookhaven) por la sugerencia de la *American Crystallographic Association* (Asociación americana de cristalografía) y desde el año 1998 hasta la actualidad es gestionado por el *Research Collaboratory for Structural Bionformatics* (RCSB). En sus inicios el PDB contaba sólo con 7 estructuras, pero con el esfuerzo de biólogos y bioquímicos de todo el mundo, este número ha crecido anualmente de manera casi exponencial hasta llegar a lo que conocemos hoy en día [17].

Como las entradas de datos en el PDB se tratan de un esfuerzo en conjunto de científicos de todo el mundo, se creó un formato estándar para así facilitar la lectura de estos independiente de su creador, además de definir la forma para describir las estructuras tridimensionales y facilitar la creación de algoritmos para el análisis masivo de los datos. Este formato se creo en el año 1976 por miembros del *Brookhaven Nacional Laboratory* y se ha ido actualizando en base a nuevos requerimientos y necesidades.

El nombre del archivo que contiene la información estructural de una macromolécula cuenta con cuatro caracteres alfanuméricos seguido por la extensión ".pdb", este nombre es único para cada entrada de datos e identifica a la macromolécula ingresada a la base de datos. Cabe señalar que el nombre del archivo identifica la entrada de datos, no a la molécula en sí. Esto quiere decir que en la base de datos podemos encontrar varias moléculas hemoglobinas, pero con diferente nombre de archivos [18].

Los archivos del PDB cuentan con las siguientes secciones:

**Sección de título:** La primera sección dentro de un archivo PDB es la sección del título. En esta sección se encuentran datos como el nombre de la molécula, tipo de método utilizado para su análisis, los autores involucrados en el estudio de la molécula, etc.

**Sección de estructura primaria:** Esta sección contiene la información de los residuos de cada cadena de la macromolécula. En estos registros se incluyen identificadores de cadenas y números de secuencia que permiten a los otros registros vincularse a las secuencias.

**Sección heterógeno:** En esta sección encontraremos la descripción completa de los residuos no estándares de la proteína ingresada.

**Sección de estructura secundaria:** En esta sección se describen las Hélice alfa y lamina beta que se encuentran dentro de la estructura de la proteína.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ATOM | 1 | N | ALA | E | 1 | 23.208 | 25.479 | 38.434 | 1.00 | 69.40 | N |
| ATOM | 2 | CA | ALA | E | 1 | 21.874 | 25.448 | 37.769 | 1.00 | 67.24 | C |
| ATOM | 3 | C | ALA | E | 1 | 20.763 | 25.477 | 38.815 | 1.00 | 65.79 | C |
| ATOM | 4 | O | ALA | E | 1 | 20.973 | 25.901 | 39.956 | 1.00 | 69.75 | O |
| ATOM | 5 | CB | ALA | E | 1 | 21.747 | 24.196 | 36.899 | 1.00 | 48.49 | C |
| ATOM | 6 | N | MET | E | 2 | 19.582 | 25.025 | 38.413 | 1.00 | 60.09 | N |
| ATOM | 7 | CA | MET | E | 2 | 18.423 | 24.980 | 39.293 | 1.00 | 55.22 | C |
| ATOM | 8 | C | MET | E | 2 | 17.880 | 26.343 | 39.721 | 1.00 | 46.03 | C |
| ATOM | 9 | O | MET | E | 2 | 16.735 | 26.637 | 39.421 | 1.00 | 43.58 | O |
| ATOM | 10 | CB | MET | E | 2 | 18.721 | 24.131 | 40.519 | 1.00 | 54.54 | C |
| ATOM | 11 | CG | MET | E | 2 | 17.879 | 22.881 | 40.583 | 1.00 | 58.06 | C |
| ATOM | 12 | SD | MET | E | 2 | 16.182 | 23.244 | 41.023 | 1.00 | 80.13 | S |
| ATOM | 13 | CE | MET | E | 2 | 15.297 | 22.720 | 39.523 | 1.00 | 72.12 | C |
| ATOM | 14 | N | THR | E | 3 | 18.663 | 27.168 | 40.414 | 1.00 | 40.24 | N |
| ATOM | 15 | CA | THR | E | 3 | 18.152 | 28.483 | 40.811 | 1.00 | 41.75 | C |
| ATOM | 16 | C | THR | E | 3 | 19.150 | 29.619 | 40.670 | 1.00 | 38.78 | C |
| ATOM | 17 | O | THR | E | 3 | 20.353 | 29.395 | 40.566 | 1.00 | 41.40 | O |
| ATOM | 18 | CB | THR | E | 3 | 17.698 | 28.522 | 42.290 | 1.00 | 31.30 | C |
| ATOM | 19 | OG1 | THR | E | 3 | 18.835 | 28.317 | 43.127 | 1.00 | 38.11 | O |
| ATOM | 20 | CG2 | THR | E | 3 | 16.632 | 27.458 | 42.583 | 1.00 | 32.78 | C |
| ATOM | 21 | N | TYR | E | 4 | 18.622 | 30.844 | 40.656 | 1.00 | 35.40 | N |
| ATOM | 22 | CA | TYR | E | 4 | 19.440 | 32.060 | 40.615 | 1.00 | 31.36 | C |
| ATOM | 23 | C | TYR | E | 4 | 18.736 | 33.094 | 41.502 | 1.00 | 33.38 | C |
| ATOM | 24 | O | TYR | E | 4 | 17.539 | 32.960 | 41.783 | 1.00 | 29.15 | O |
| ATOM | 25 | CB | TYR | E | 4 | 19.619 | 32.610 | 39.177 | 1.00 | 31.43 | C |
| ATOM | 26 | CG | TYR | E | 4 | 18.365 | 32.962 | 38.397 | 1.00 | 34.04 | C |
| ATOM | 27 | CD1 | TYR | E | 4 | 17.813 | 32.065 | 37.473 | 1.00 | 35.28 | C |
| ATOM | 28 | CD2 | TYR | E | 4 | 17.748 | 34.200 | 38.554 | 1.00 | 32.96 | C |
| ATOM | 29 | CE1 | TYR | E | 4 | 16.674 | 32.400 | 36.728 | 1.00 | 37.16 | C |
| ATOM | 30 | CE2 | TYR | E | 4 | 16.604 | 34.548 | 37.802 | 1.00 | 31.87 | C |
| ATOM | 31 | CZ | TYR | E | 4 | 16.076 | 33.647 | 36.897 | 1.00 | 34.89 | C |
| ATOM | 32 | OH | TYR | E | 4 | 14.960 | 33.995 | 36.148 | 1.00 | 33.52 | O |

Figura 2.4: Sección de coordenadas de un archivo con formato “PDB”.

**Sección de anotaciones de conectividad:** Permite a los depositadores especificar la existencia y ubicación de enlaces de disulfuros[3] u otros.

**Sección de características misceláneas:** En esta sección se describen propiedades como el ambiente que rodea a los residuos no estándares o el ensamblado de un sitio activo.

**Sección de cristalografía y transformación de coordenadas:** En esta sección se describe la geometría de los experimentos cristalográficos y el sistema de transformación de coordenadas.

**Sección de coordenadas:** Esta es la sección más importante para nuestro trabajo de investigación, ya que en ella se incluyen las coordenadas de cada molécula de aminoácido a nivel atómico, esto quiere decir que cada elemento químico del residuo del aminoácido posee su registro de coordenada en el espacio tridimensional.

[3]Disulfuro: unión covalente de dos átomos de azufre

En la Figura 2.4 podemos ver la sección de coordenadas de un archivo PDB. La primera linea de la figura tiene como propósito obtener una mejor visualización del archivo, pero no forma parte de los archivos PDB [18].

Columna 1: Describe el nombre del registro, el cual puede ser MODEL (modelo), ENDMDL, ATOM (átomo), ANISOU, HETATM y TER. Los registros con los que se trabajará son ATOM, ya que es el registro que posee el nombre y coordenadas de los aminoácidos y sus respectivos elementos químicos que lo conforman, y TER, ya que señala cuando finaliza una cadena de registros ATOM.
Como utilizaremos el tipo de registro ATOM, podemos continuar explicando el significado de las columnas siguientes:

Columna 2: Consiste en el número serial del átomo.

Columna 3: Muestra el nombre del átomo. Por lo general en el nombre del átomo podemos observar que la primera letra es el símbolo químico de este, seguido por su identificación en cuanto a la cercanía del átomo al grupo funcional. Por ejemplo CA es una abreviación para carbono alfa, carbono central que enlaza el grupo carboxilo y el grupo amino en un aminoácido, como se señaló en la Sección 2.1.2.

Columna 4: representa el nombre del residuo, tres caracteres que representan la abreviatura de los residuos de los aminoácidos en el registro. Las abreviaturas se pueden observar en la figura (aminoácidos).

Columna 5: Representa el identificador de la cadena, ya que la estructura puede estar formada por varias cadenas polipéptidas, en estos casos cada cadena posee un identificador distinto.

Columna 6: Representa el número de secuencia del residuo, que vendría a ser el identificador de cada registro de aminoácidos.

Columna 7-8-9: Representan las coordenadas X, Y y Z de todos los átomos de la proteína o complejo proteico registrado. Estas coordenadas describen la posición de cada átomo en un sistema de coordenadas ortogonal. La unidad de medida que utilizan las coordenadas es el Angstrom, que equivale a 1e-8 centímetros de longitud. Estas coordenadas hacen posible representar gráficamente la estructura terciaria de las proteínas.

Columna 10: Representa la ocupancia de los átomos. En algunos casos la cadena Lateral e incluso la cadena principal de un aminoácido puede tener dos o más conformaciones, esto debido a la flexibilidad local de esta, por este motivo se utiliza la columna de ocupancia, la cual representa la probabilidad de que el átomo en el registro ocupe la posición determinada. Por lo general ya ocupancia es 1.00, significando que el átomo efectivamente ocupa esa posición y no otro alternativa.

Columna 11: Representa el factor temperatura del átomo.

Columna 12: Indica el tipo de átomo del registro, representado por su símbolo químico.

### 2.1.5. Interacciones Proteína-Proteína (IPP)

Las proteínas son moléculas que están constantemente interactuando con otras proteínas, ADN y ARN, moléculas más pequeñas, entre otras. Raramente actúan solas, por eso es de gran importancia el estudio de sus interacciones.

Las interacciones que se producen entre dos o más proteínas se denominan complejos proteicos o complejos multiproteicos, estos son contactos físicos que se producen por un acoplamiento molecular entre proteínas que se lleva a cabo en una célula o en un organismo vivo *in vivo*[4], son esenciales para cualquier proceso celular, juegan un papel clave para la regulación celular, biosíntesis y degradación de vías, traducción de señales, iniciación de replicación de ADN, control de expresión de genes, inhibición de enzimas, reconocimiento de anticuerpo-antígeno [19].

Las interacciones proteína-proteína se pueden clasificar considerando diversos factores, en diferentes tipos. Algunos de los que se pueden considerar son su composición, según la cual podemos distinguir complejos homo y hetero-oligomeros; otro factor también es su afinidad de unión, aquí encontraremos las IPP obligadas y no-obligadas; y luego tenemos el factor de tiempo de vida, en el cual se dividen los complejos transitorios y permanentes:

**Complejos homo-oligomeros y hetero-oligomeros**:

Las interacciones proteína-proteína pueden ocurrir por la interacción de cadenas idénticas (homo-oligomeros) así como también con cadenas diferentes (hetero-oligomeros)

**Complejos obligados y no-obligados**:

En una interacción proteína-proteína obligada, los protómeros no se pueden encontrar como estructuras estables por sí solas, tienen que estar emparejadas *in vivo*; en una interacción proteína proteína no-obligada, los protómeros pueden existir de manera independiente.

**Complejos transitorios y permanentes**

Las interacciones proteína proteína también se pueden clasificar según el tiempo de vida que posea la interacción entre ellas, aquí podemos encontrar a los complejos permanentes y a los complejos transitorios. Los primeros, que son muy estables, tienen una vida media larga. Diferentes son las IPP transitorias, las cuales se asocian y disocian continuamente *in vivo*.

Las IPP transitorias también se subdividen en otra clasificación, transitorias débiles que se caracterizan por tener una baja afinidad de unión y un tiempo de vida muy bajo, y transitorias fuertes, que pueden cambiar su estructura al ser gatilladas.

Según [7] las interacciones obligadas son usualmente permanentes, en cambio las no-obligadas pueden ser transitorias o permanentes. Eso se puede apreciar en la Figura 2.5, que presenta los

[4]In vivo: experimento realizado dentro de un tejido u organismo vivo

tipos de interacción proteína-proteína y como se corresponden entre si.

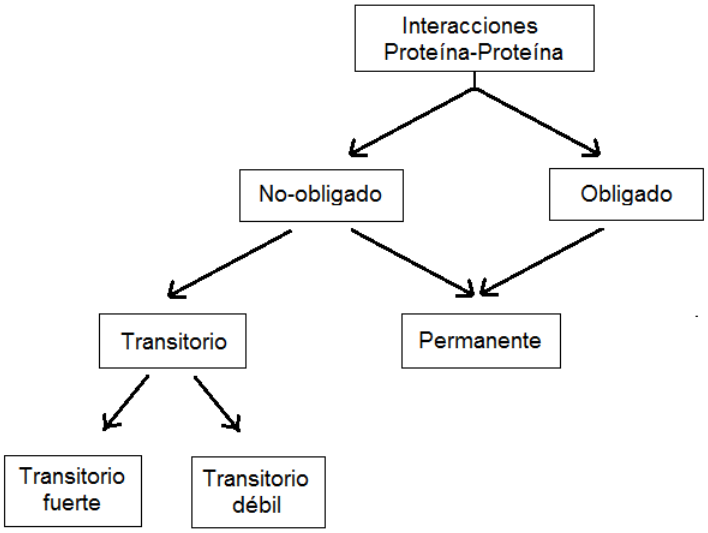


Figura 2.5: Tipos de interacciones proteína-proteína [2].

### 2.1.6. Zona de interacción

Las proteínas involucradas en una interacción proteína-proteína no utilizan toda su estructura para formar la interacción entre cadenas, sino que lo hacen mediante una pequeña zona en la cual se produce el enlace entre proteínas. Esta zona se denomina zona de interacción (*interface*), la que consiste en el enlace de residuos de aminoácidos mediante puentes de hidrógeno, que pertenecen a dos cadenas diferentes, por lo que podemos decir que esta zona está formada por fragmentos de ambas cadenas polipéptidas [20].

La zona de interacción posee propiedades diferentes al resto de la estructura del complejo proteico, esto permite que la proteína interactúe específicamente con una o más proteínas.

En la Figura 2.6 podemos observar la representación gráfica de un complejo proteico, en color calipso y azul están representadas las proteínas que interactúan y en color verde y rojo tenemos la zona de interacción.

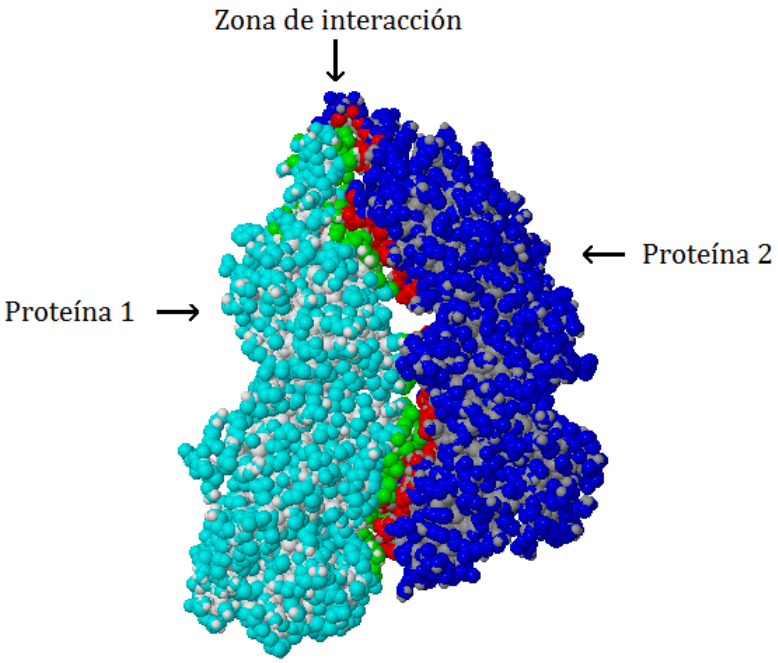


Figura 2.6: Zona de interacción de un complejo proteico [3].

La imagen se obtuvo gracias al visualizador web PDBePISA y representa un complejo proteico permanente cuya ID es 1a6d.

### 2.1.7. Métodos de detección de interacciones proteína proteína

Existen varios métodos para detectar los diferentes tipos de complejos proteicos, estos métodos se pueden clasificar en métodos *in vivo*, los cuales se basan en el uso de organismos vivos que mantienen un ambiente celular semejante al de tiene la proteína nativa, y métodos *in vitro*, los cuales se basan en el uso de extractos celulares más o menos complejos.

**Métodos in vitro o bioquímicos-biofísicos**:

- Coinmunoprecipitación: Este método utiliza la inmunoprecipitación[5] por afinidad de una proteína, lo que permite una coprecipitación selectiva de las proteínas que actúan con ella [19].

[5]Inmunoprecipitación: Técnica en la cuál un antígeno proteico es precipitado de una solución utilizando un anticuerpo específico de esa proteína.

- Espectrometría de masas: Esta técnica si bien no tiene como principal función el determinar el tipo de interacción entre proteínas, si cuenta con una eficaz capacidad para la identificación de proteínas. Esto lo realiza por medio de la estimación precisa de la masa atómica de la proteína completa, esto también es utilizado con los otros métodos de interacción proteica que proporcionan proteínas que interaccionan, pero que necesitan ser identificadas [19].

- Cristalografía de rayos X: Este método permite la determinación de los complejos proteicos a nivel atómico y estructural. La cristalografía de rayos X es uno de los métodos más poderosos para estudiar estructuras macromoleculares. Este método utiliza difracción de rayos X en las macromoléculas previamente cristalizadas para así obtener información de su estructura tridimensional. El inconveniente que presenta este método consiste en que los complejos proteicos deben ser previamente identificados y purificados a nivel de homogeneidad para lograr la cristalización de los complejos proteicos para su posterior análisis [19].

- Espectrometría de RMN (resonancia magnética nuclear): Este método permite extraer las distancias entre los átomos midiendo transmisiones en un campo magnético los diferentes estados de spin nuclear. Luego las distancias son usadas como restricciones para así construir la estructura tridimensional [19].

**Métodos in vivo**:

- Sistema de dos híbridos: El sistema de dos híbridos requiere la coexpresión de dos proteínas de fusión, una definida como cebo y la otra como presa, las cuales al interactuar, son capaz de reconstruir una proteína que por sí misma, o por medio de la activación de genes reporteros, confieren al organismo una propiedad que puede ser detectada [19].

- FRET (Transferencia de energía de resonancia de Förster): Con este método, se hacen interactuar dos proteínas, una etiquetada como cromóforo donador y la otra cromóforo receptor, cuando interactúan se produce una transferencia energética desde el cromóforo donador al receptor y la energía producida genera un cambio en la fluorescencia del cromóforo receptor que puede ser detectada por medio de microscopía confocal de fluorescencia [19].

### 2.1.8. Predicción de interacciones proteína proteína

Gracias a los grandes avances en tecnología de alto rendimiento, una gran cantidad de datos de interacciones proteína-proteína se encuentran disponibles en numerosas bases de datos, varias de estas bases de datos se encuentran en línea y disponibles para la comunidad científica para su análisis. Entre ellas se destacan *BioGRID* (General Repository of Interaction Database), una base de datos que contiene todas las interacciones genéticas o entre proteínas conocidas, *DIP* (Database of Interacting Proteins), una base de datos que combina datos provenientes de varias fuentes para formar un conjunto consistente de interacciones proteína-proteína y el *Protein Data Bank*, del cual se habló en la Sección 2.1.4. Estos datos son utilizados entre otras cosas, en el campo de la bioinformática sobre la predicción de interacciones de proteína-proteína que busca por medio de herramientas computacionales, identificar y categorizar el tipo de interacción que se produce en un complejo proteico dado.

Existen diversos métodos utilizados en el área de la predicción de interacciones, los cuales se pueden clasificar en 4 categorías, métodos basados en el contenido genómico y función estructural, métodos que utilizan redes topológicas, métodos que utilizan minería de textos y métodos que utilizan aprendizaje automático para la predicción de interacciones.

## 2.2. Métodos de Aprendizaje

### 2.2.1. Minería de datos y descubrimiento de conocimiento

Para obtener información de valor de las bases de datos masivas como son los bancos de datos de proteínas, no se pueden utilizar métodos manuales para el análisis de datos por su clara ineficiencia y su alta probabilidad de generar error humano. Por este motivo es indispensable la utilización del proceso de minería de datos.

La minería de datos es un área de las ciencias de la computación, cuya finalidad es descubrir patrones en repositorios de datos masivos, [21] define minería de datos como “el proceso no trivial de identificación de patrones en los datos, que sean validos, novedosos, potencialmente útiles y por ultimo comprensibles”[6]. Aunque no se incluye en su definición, se asume que este proceso debe estar al menos parcialmente automatizado, por lo que requiere fuertemente la utilización de algoritmos computacionales especializados de áreas como inteligencia artificial, aprendizaje automatizado y estadística [21]. Existe cierta ambigüedad al hablar de minería de datos ya que algunos consideran a esta como uno de las etapas del descubrimiento de conocimiento en bases

[6]Data mining: Is the nontrivial process of identifying valid, novel, potentially useful, and ultimate ly understandable patterns in data

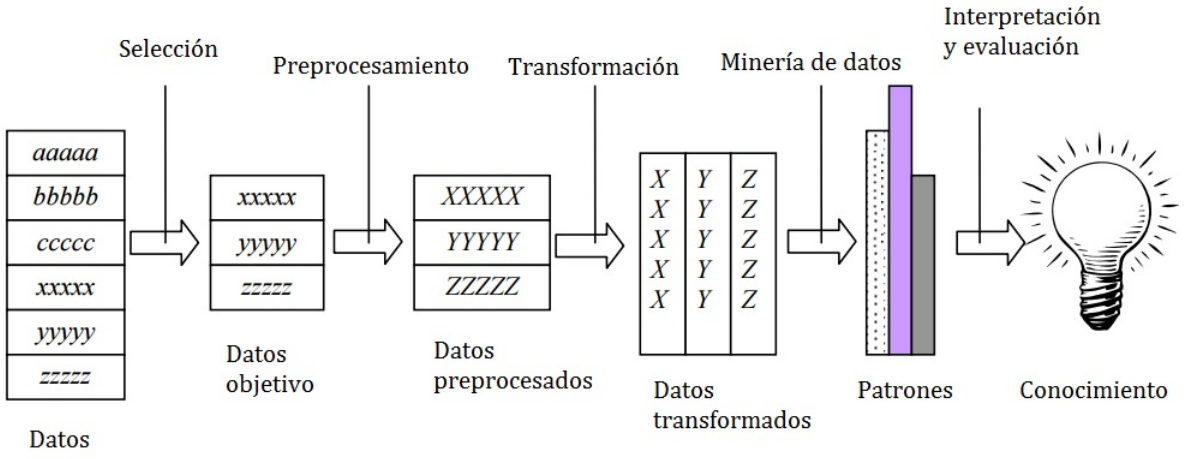


Figura 2.7: Etapas del descubrimiento del conocimiento en las bases de datos [4].

de datos o KDD (Knowledge discovery in databases) y otros consideran la minería de datos como el proceso completo. Para evitar esta ambigüedad se considerará la minería de datos como uno de los pasos algorítmicos dentro del KDD [22].

El descubrimiento de conocimiento en bases de datos consta de cinco etapas: Selección de datos, preprocesamiento, transformación, minería de datos y por ultimo el paso de interpretación y evaluación.

1. **Selección de datos**: No todos los datos de un repositorio sirven para obtener patrones de estos, por otra parte se puede optar por realizar una búsqueda de patrones con un modelo más acotado, el paso de la selección de datos consiste en analizar el problema que estamos desarrollando y luego seleccionar el conjunto concorde a nuestras exigencias.

2. **Preprocesamiento**: Luego de seleccionados los datos, se deben transformar a un formato apropiado para el análisis de estos, esto ya que los datos pueden venir en un formato que no cumpla con las exigencias para trabajar con ellos, además en el paso de preprocesamiento se deben limpiar los datos, eliminando zonas que estén incompletas además de eliminar posibles datos que contengan ruido que pueda afectar el análisis.

3. **Transformación**: Este paso consiste en la transformación de los datos para obtener mejores resultados en su análisis, por ejemplo si tenemos en nuestros datos atributos de texto y otros de números, se deben normalizar estos datos ya que la mayoría de los algoritmos de minería de datos están creados para utilizar datos similares.

4. **Minería de datos**: Como pudimos observar, antes de aplicar el proceso de minería de datos como tal, se deben adaptar los datos a utilizar de manera de evitar posibles resultados inconsistentes. Luego de finalizado el proceso de refinación de datos, se pueden aplicar los algoritmos computacionales de minería de datos para la identificación de patrones relevantes.

5. **Interpretación y evaluación**: El último paso del KDD es la interpretación y evaluación de los patrones obtenidos por la minería de datos. El descubrimiento de conocimiento en bases de datos es un proceso iterativo, por este motivo en este paso se obtiene una idea sobre el como se debe proceder, si es que existe una disconformidad por los resultados obtenidos se puede volver al paso número uno para repetir el proceso con algunos ajustes.

### 2.2.2. Aprendizaje automático

El aprendizaje automático (machine learning) es un campo de la inteligencia artificial que se define como un conjunto de métodos cuyo objetivo es detectar automáticamente patrones en datos con la finalidad de predecir datos futuros. Básicamente se busca que los computadores tengan la capacidad de aprender mediante el desarrollo de técnicas de aprendizaje. Es posible conseguir aprendizaje ya que este no necesariamente tiene que ver con la conciencia (Conocimiento que un ser tiene consigo mismo y su entorno), también se puede conseguir encontrando regularidades estadísticas u otros patrones en los datos, con los cuales se generan un conjunto de reglas en el sistema. Por este motivo los algoritmos de aprendizajes no se asemejan tanto al aprendizaje realizado por un humano. En las últimas décadas ha sido primordial el uso del aprendizaje automático a la hora de extraer información relevante en grandes bases de datos y la predicción de comportamiento en base a ellas [23].

Estamos rodeados de tecnologías que utilizan el aprendizaje automático, como por ejemplo motores de búsqueda como Google [24], que predicen qué queremos buscar y cuáles son los resultados más específicos con respecto a nuestra búsqueda, programas de protección de transacciones de tarjetas de crédito, que aprenden a identificar y evitar fraudes, sistemas de reconocimientos facial, también de reconocimiento de voz y reconocimiento de escritura manual, además se utiliza para generar recomendaciones personalizadas en variadas plataformas web. El aprendizaje automático además es muy utilizado en las áreas científicas como la astronomía, medicina y bioinformática, por ejemplo en la bioinformática se utiliza para la clasificación de secuencias de ADN [25].

Para lograr que una máquina aprenda mediante al aprendizaje automático, se deben cargar en el sistema un conjunto de datos de entrenamiento, estos datos son analizados por un algoritmo de aprendizaje (Algoritmos encargados del procesamiento de los conjuntos de datos de entrenamiento) y en base a ese conjunto de datos se genera un árbol o reglas de decisión (Reglas que generalizan el comportamiento de los datos de entrenamiento, con la finalidad de predecir siguientes entradas), que vendría a ser el conocimiento. Luego de esto podemos utilizar ese conocimiento en conjuntos de datos futuros. En la Figura 2.8 se presenta un esquema básico del problema de aprendizaje automático.

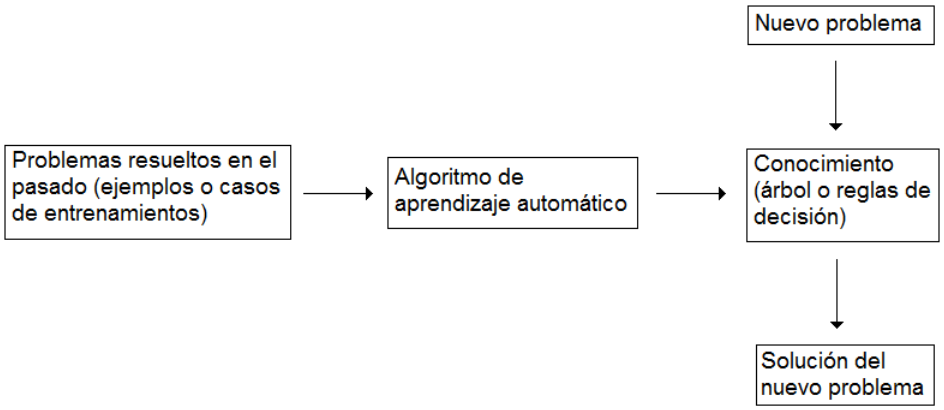


Figura 2.8: Esquema general del problema de aprendizaje [5].

Existen tres tipos de problemas que intenta resolver el aprendizaje automático: problemas que clasificación, regresión y de agrupamiento.

- **Problema de clasificación:** involucra predecir si un conjunto de datos pertenece a una cierta categoría predefinida. Cuando tratamos con dos posibles categorías o clases, hablamos de clasificación binario, si están definidas más de dos clases, hablamos de clasificación multiclase.

- **Problema de regresión:** es similar al problema de clasificación ya que ambos intentan relacionar las entradas y salidas de los conjuntos de datos de entrenamiento para predecir conjuntos de datos futuros, pero a diferencia de la clasificación, en el problema de regresión se intenta predecir valores numéricos reales.

- **Problema de agrupamiento:** En el problema de agrupamiento pretende formar agrupaciones de objetos que son similares, asegurándose que todas las agrupaciones formadas sean distintas entre sí. A diferencia del problema de clasificación, el problema de agrupamiento no necesita tener una categorización de clases.

Debido a lo amplio y complejo que es el campo del conocimiento, existe una gran cantidad de algoritmos de aprendizaje que lidian con diferentes tareas de aprendizajes. Estos algoritmos se clasifican por estilo de aprendizaje.

- **Aprendizaje supervisado:** Los algoritmos de aprendizaje supervisado utilizan un conjunto de datos de entrenamiento clasificados previamente, generando una función basada en la correlación entre las entradas y salidas del conjunto de datos ingresados. Estos algoritmos son los más comunes en aprendizaje automático y son convenientes cuando obtener la clasificación de un problema es útil y la clasificación en si es fácil de determinar [26].

- **Aprendizaje no supervisado:** El aprendizaje no supervisado es más complicado que su contraparte, ya que el conjunto de datos ingresados no tienen una clasificación previa. Los algoritmos que pertenecen a la categoría de aprendizaje no supervisado, deben lograr que el sistema sea capaz de reconocer patrones para poder clasificar conjuntos nuevos de datos.

- **Aprendizaje semi supervisado:** Los algoritmos correspondientes al aprendizaje semi supervisado son una combinación entre el aprendizaje supervisado y no supervisado, la entrada de conjuntos de datos de entrenamiento son una mezcla entre datos clasificados y no clasificados previamente.

- **Aprendizaje por esfuerzo:** Este tipo de algoritmo adquiere conocimiento observando el mundo que lo rodea, generando reacciones del entorno por medio de las acciones del sistema, las cuales se transforman en una retroalimentación para el sistema, reforzando las acciones con respuesta positiva en el entorno.

- **Transducción:** Es semejante al aprendizaje supervisado, pero la diferencia radica en que en los algoritmos de traducción no generan una función explicita sobre la correlación de las entradas y salidas de un conjunto de entrenamiento, en cambio intenta predecir la clasificación de nuevas salidas (categorías) por medio de datos de entradas de entrenamiento, datos de salida de entrenamiento y nuevas entradas de datos [27].

- **Aprendizaje multitarea:** Este tipo de algoritmo resuelve varias tareas al mismo tiempo y a la vez intenta conseguir generalidades y diferencias entre las diferentes tareas, con el fin de utilizar esta nueva información en el mejoramiento de la predicción de las tareas involucradas, cuando lo comparamos con la predicción de cada tarea unitariamente.

Como veremos en la Sección 3.2, el conjunto de datos de entrenamiento que utilizaremos, están

clasificados previamente, por lo que se optará por utilizar algoritmos de aprendizaje supervisado

Con la introducción a la bioinformática, además de entender cómo están formadas las proteínas y sus unidades básicas, los aminoácidos, y teniendo un entendimiento del aprendizaje automático, tenemos la información necesaria para abarcar el problema de principal, el demostrar que la distancia entre aminoácidos en la zona de interacción influye en la clase de interacción de los complejos. Pero para esto necesitamos obtener la información acerca de las distancias y energías involucradas en dicha interacción, lo que veremos en el siguiente capítulo.

# Capítulo 3

# Estudio del problema

Las interacciones proteína-proteína como pudimos ver en el capítulo anterior, son de vital importancia ya que actúan en la mayoría de los procesos biológicos, tales como la biosíntesis, traducción de señales, entre otros. Las interacciones proteína-proteína pueden formar un complejo estable en el tiempo (complejo permanente) o un complejo que se asocia y disocia continuamente (complejo transitorio); conocer el tipo de interacción que forma un complejo proteico puede ayudar a tener un mejor entendimiento de la naturaleza y función de una interacción.

Los métodos para detectar el tipo de interacción de forma eficaz son los métodos *in vitro*, los más utilizados son la cristalografía de rayos X y la resonancia nuclear, el problema con estos métodos es que resultan costosos, consumen mucho tiempo, por lo cual son eficaces pero no eficientes.

En las últimas décadas se ha buscado un enfoque informático que resuelva este problema, debido a que se ha generado una gran cantidad de datos de secuencias y estructura tridimensional de las proteínas y complejos proteicos que manualmente no eran posibles de abarcar. Cabe señalar que aún no existe un método computacional 100 % eficaz para la predicción de interacciones proteína-proteína.

Enfoques como la investigación desarrollada por [8], en el cual propone el estudio de las características energéticas más significativas, pertenecientes a la zona de interacción en un conjunto de complejos proteicos, por medio de la búsqueda de patrones mediante a aprendizaje automático. Los mejores resultados de su estudio estuvieron cercanos al 81 % de predicción de interacciones.

El enfoque a utilizar en esta investigación es determinar si la distancia entre aminoácidos pertenecientes a la zona de interacción en un complejo proteico afecta al tipo de interacción, enfoque que no ha sido muy explorado en la literatura. Para esto se utilizará la investigación de [8] como base.

## 3.1. Metodología

Para lograr el objetivo propuesto, en la Figura 3.1 se plantea la metodología a seguir. El primer paso es la obtención de los complejos proteicos que serán utilizados como conjunto de datos de entrenamiento para los algoritmos de aprendizaje. Obtenidos estos datos, luego se analizará el como se utilizarán los elementos químicos de cada aminoácido para obtener un punto representativo para cada aminoácido. Luego se estudiarán y seleccionarán métodos de medida de distancia para su utilización en los aminoácidos. Luego se desarrollarán algoritmos que realicen los pasos anteriores, algoritmos de manejo de coordenadas y de cálculo de distancia. Con los resultados obtenidos mediante la aplicación de estos algoritmos se generarán matrices con las características energéticas y de distancias, para luego utilizarlas en un programa de aprendizaje automático. Por último se analizarán los resultados obtenidos y se entregarán conclusiones con respecto a si la distancia es un factor que influye en el tipo de interacción de un complejo proteico.

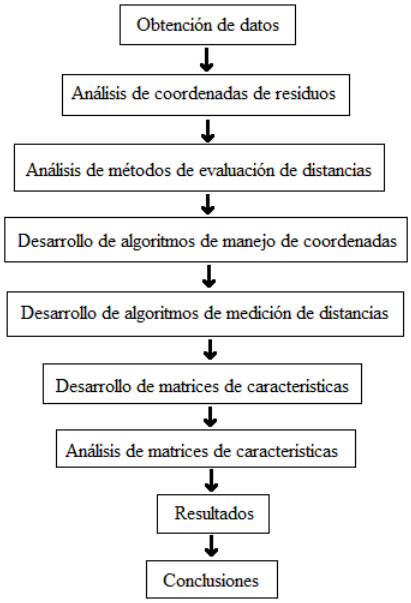


Figura 3.1: Metodología de trabajo.

## 3.2. Obtención de datos

Para resolver un problema de clasificación mediante la utilización de aprendizaje automático, se requiere una considerable cantidad de datos previamente clasificados correctamente para ser utilizados como conjunto de datos de entrenamiento.

Mintseris y Weng en 2005 realizaron un estudio sobre la estructura, función y evolución de la secuencia de interacciones proteína-proteína de complexos permanentes y transitorios para obtener una mejor clasificación de estos. En su estudio, compilaron un conjunto de datos no redundante de complejos proteicos pertenecientes al *Protein Data Bank* (hablado en la Sección 2.1.4) y manualmente separados en complejos proteicos permanentes y transitorios [28].

Este conjunto de datos, corresponde a 326 complejos, de los cuales 115 corresponden a interacciones proteína proteína permanentes y 211 a interacciones transitorias. Este listado se puede encontrar en [28].

El listado corresponde a las ID's de los 326 complejos proteicos, cuya identificación es accesible en el repositorio "Protein Data Bank". En la investigación realizada por Gutiérrez-Bunster [8] se obtuvieron todos los archivos con extensión "pdb" de los complejos.

### 3.2.1. Propiedades energéticas e interacciones entre aminoácidos

En este trabajo de investigación se utilizarán las propiedades energéticas y las distancias entre los aminoácidos pertenecientes a la zona de interacción, para obtener esta información, se utilizará la aplicación llamada FastContact [29].

### 3.2.2. Software FastContact

FastContact es un programa desarrollado por Carlos Camacho y Chao Zhang, en conjunto con su Departamento de Biología Computacional en la Universidad de Pittsburgh [29]. Esta aplicación permite estimar las energías libres de interacción entre dos proteínas. Para lograr esto FastContact utiliza una función de puntuación de energía libre, la cual asegura que las energías que aportan los residuos pertenecen a la zona de interacción.

Para poder utilizar el programa, el archivo de extensión "pdb" de cada complejo se deben separar en dos archivos "pdb" que contienen la información de cada proteína por separado. Gutiérrez-Bunster realizó la separación de estos complejos para su posterior utilización en el programa.

FastContact entrega tres archivos por cada complejo proteico ingresado. Dos de ellos son archivos de texto similares a los archivos ".pdb" ingresados, los cuales contienen sólo la sección de coordenadas del archivo "pdb" original (visto en la Sección 2.1.4), excluyendo en esta sección la ocupancia, factor temperatura y elemento químico.

El tercer archivo de salida de la aplicación contiene las características energéticas. Estas características están organizadas en diferentes secciones. Cada sección contiene los 20 residuos que más energía contribuyen en la interacción y los 20 residuos que menos energía contribuyen. En la Tabla 3.1 podemos ver los datos que contiene el tercer archivo con respecto a las secciones que se enumeran a continuación:

1. Residuos que contribuyen a la energía libre de unión.
2. Residuos Ligando que contribuyen a la energía libre de desolvatación.
3. Residuos Ligando que contribuyen a la energía electrostática.
4. Residuos Receptor que contribuyen a la energía libre de desolvatación.
5. Residuos Receptor que contribuyen a la energía electrostática.

Secciones de pares de residuos que interactúan:

6. Residuos Receptor-Ligando contacto electrostático.
7. Residuos Receptor-Ligando contacto energía libre.

| Sección de FastContact | Columnas de datos | Número total de datos |
|---|---|---|
| Residuos que contribuyen a la energía libre de unión | E R - A 20 max y 20 mín | 120 |
| Residuos ligandos que contribuyen a la energía de desolvatación | E RL - A 20 max y 20 mín | 120 |
| Residuos ligandos que contribuyen a la energía electroestática | E RL - A 20 max y 20 mín | 120 |
| Residuos receptores que contribuyen a la energía de desolvatación | E RR - A 20 max y 20 mín | 120 |
| Residuos receptores que contribuyen a a energía electroestática | E RR - A 20 max y 20 mín | 120 |
| Residuos receptor-ligando de contactos electroestáticos | E RR - A RL - A 20 max y 20 min | 200 |
| Residuos receptor-ligando de contactos de energía libre | E RR - A RL - A 20 max y 20 min | 200 |

Tabla 3.1: Datos de salida de FastContact. E: Energía contribuida, R: Residuo que participa (ligando o receptor), RR: Residuo receptor que participa, RL: Residuo ligando que participa, AA: Indica el tipo de aminoácido correspondiente al residuo que participa.

En las cinco primeras secciones, cada residuo tiene como atributo la energía que contribuye, un identificador numérico (identificador único dentro de la cadena polipéptida, pero no del complejo completo) y el código de tres letras del residuo (Tabla 2.1 de la Sección 2.1.2). Las siguientes dos secciones la energía de la interacción de residuos, es seguida por el identificador del residuo Receptor, luego su código de tres letras, y seguido por el identificador del residuo Ligando, seguido por su código de tres letras.

Las dos últimas secciones son de mayor importancia para este estudio, por el hecho de establecer una relación entre cada par residuos energéticamente más significativos de la zona de interacción. Con el identificador numérico de cada pareja de residuo, podemos buscar las coordenadas de estos y así obtener la distancia que existe entre ellos. El tipo de distancia que se utilizará se verá en la siguiente sección.

## 3.3. Análisis de coordenadas de los residuos

El residuo de un aminoácido (Cadena lateral, Sección 2.1.2) está formado por varios elementos químicos de los cuales cada uno posee su propia coordenada en los archivos "pdb", como podemos ver en la Figura 3.2. Como se vio en la Sección 2.1.2, los aminoácidos están formados por los elementos químicos: Carbono, Hidrógeno, Oxígeno, Nitrógeno y en no todos los casos, Azufre.

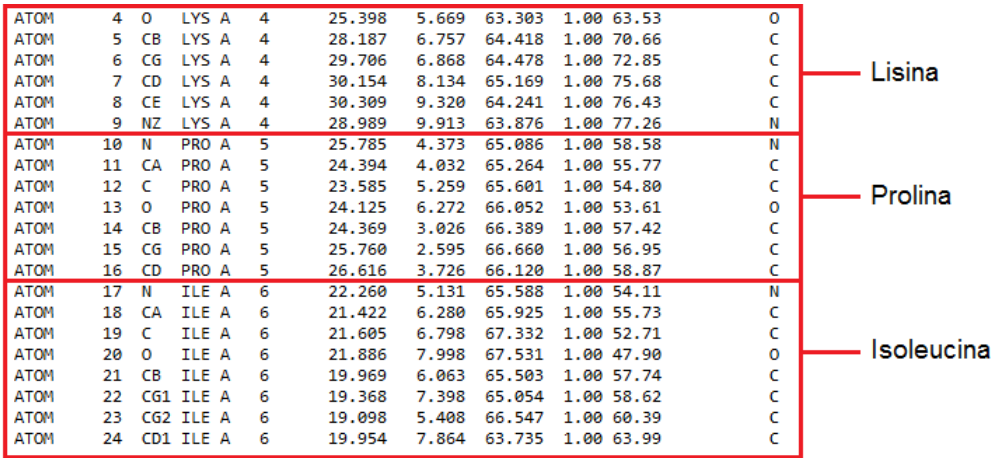


Figura 3.2: Ejemplo de elementos químicos que conforman el residuo de un aminoácido y sus coordenadas. Segmento extraído del archivo PDB del complejo con ID 1a6d.

Para conseguir la distancia entre dos residuos que interactúan, debemos establecer un punto representativo para cada uno de ellos, por lo cual es importante definir una forma de tratar las coordenadas de todos los elementos químicos de cada residuo.

Considerando a los elementos químicos de los residuos como un sistema de partículas, se utilizaron los siguientes enfoques:

### 3.3.1. Centro geométrico

Como primer enfoque se utilizó el centro geométrico de los residuos, el cual consiste en el punto en el espacio promedio de todas las coordenadas que forman un objeto, en este caso, de los residuos. El centro geométrico no considera la masa de los elementos químicos, sólo considera la forma del residuo para generar el punto en el espacio que representa a este.

Para calcular el Centro Geométrico (CG) de los residuos utilizamos la Ecuación 3.1

$$CG = \frac{1}{n}\sum_{i=1}^{n}\vec{x}_i \tag{3.1}$$

Siendo $n$ el número de elementos químicos presentes en el residuo y $\vec{x}_i$ la coordenada (x, y, z) del elemento $i$. Por lo cual el centro geométrico es la sumatoria de todos los vectores coordenada de un residuo, divido por el número total de elementos.

### 3.3.2. Centro de masa total

El centro de masa es un punto geométrico en el cual se concentra la masa de un cuerpo o un conjunto de partículas. Es el punto en el cual, si se le es aplicada una fuerza, sólo se producirá un movimiento de traslación del objeto y no de rotación. Este enfoque considera tanto la forma de los residuos, como también la masa de sus elementos para encontrar un punto representativo de estos.

El Centro de Masa (CM) de un sistema de partículas discretas se calcula utilizando la Ecuación 3.2

$$CM = \frac{1}{M}\sum_{i=1}^{n}\vec{x}_i \cdot m_i \tag{3.2}$$

En donde $n$ es el número de elementos químicos presentes en el residuo, $\vec{x}_i$ la coordenada (x, y, z) del elemento $i$, $m$ la masa atómica del elemento $i$ y "M" la suma de todas las masas de los elementos del residuo.

Como las coordenadas corresponden a los elementos químicos de cada residuo de aminoácido, la masa de cada elemento corresponde a la masa atómica de dicho elemento químico.

### 3.3.3. Centro de masa de la cadena base

En la Sección 2.1.2 se explicó que la estructura compuesta por: el carbono alfa, el grupo amino y el grupo carboxilo se denomina cadena base, la cual es compartida por todos los aminoácidos. Por esto el centro de masa de esta estructura será considerada como una posible característica de interés.

### 3.3.4. Centro de masa de la cadena lateral

La cadena lateral, la cual se habló en la Sección 2.1.2 es la estructura que diferencia a un tipo de aminoácido de otro, por tanto se considerará su centro de masa como posible característica de interés.

### 3.3.5. Coordenada del carbono alfa y carbono beta

El carbono alfa es uno de los elementos centrales en un aminoácido, en el se unen los diferentes sustituyentes de cada aminoácido diferente. los grupos enlazados al carbono alfa son los que le dan a los aminoácidos su diversidad. Cuando describimos una proteína, generalmente se utiliza el carbono alfa de cada aminoácido como localización de dicho aminoácido. El segundo elemento central es el carbono beta. Por este motivo se utilizaron estos dos átomos como enfoques propios.

## 3.4. Análisis de métodos de medición de distancias

En varias disciplinas científicas como el aprendizaje automático y el análisis estadístico, se utilizan diversos métodos para el calculo de distancias con el fin de medir el grado de similitud o disimilitud de un conjunto de objetos, por ser la distancia un factor muy importante a la hora de medir este factor.

Para obtener la distancia para cada par de aminoácidos pertenecientes a la zona de interacción, como eje central de nuestra investigación utilizaremos la distancia euclidiana o distancia en linea recta, pero también expondremos otras medidas de distancia para obtener un mayor espectro en nuestros resultados como la *Manhatthan Distance* (Distancia Manhatthan) o *Taxicab Geometry* (Geometría del Taxista), entre otras, además se propondrá otro tipo de medición que utiliza la distancia euclidiana y el centro geométrico de la zona de interacción.

### 3.4.1. Distancia Euclidiana

La distancia euclidiana entre dos puntos corresponde a lo que normalmente consideramos como medida de distancia, esto es la longitud de una linea recta descrita entre estos puntos. En un espacio tridimensional la distancia euclidiana esta dada por la Ecuación 3.3. Lo conveniente de este sistema de medición es que no depende de la orientación de los puntos, sólo la separación entre ellos [30].

Al ser el eje central de nuestra investigación, esta distancia será utilizada en todos los enfoques de manejo de coordenadas expuestos en la sección anterior.

Punto A: $(a_1, a_2, a_3)$ Punto B: $(b_1, b_2, b_3)$

$$Distanciaeuclidiana_{ab} = \sqrt{(a_1 + b_1)^2 + (a_2 + b_2)^2 + (a_3 + b_3)^2} \tag{3.3}$$

### 3.4.2. Distancia Manhatthan

En la distancia Manhatthan o geometría del taxista, las coordenadas se pueden considerar como cuadras de una ciudad, y la imposibilidad de seguir un camino recto de un punto a otro. La distancia Manhatthan entre dos puntos está definida por la suma de la diferencia de los valores absolutos de sus coordenadas cartesianas como se muestra en la Ecuación 3.4 [31].

Esta forma de medición de distancia depende de la ubicación del objeto con respecto al eje en el cual se encuentra, pero a la hora de generar las coordenadas tridimensionales de una proteína o de un complejo proteico, el eje resulta irrelevante de un punto de vista biológico puesto que no existe un eje estándar para todas las estructuras proteicas lo que hace inutilizable este sistema de medición a este ámbito en específico.

Punto A: $(a_1, a_2, a_3)$ Punto B: $(b_1, b_2, b_3)$

$$DistanciaManhatthan_{ab} =\mid a_1 - b_1 \mid + \mid a_2 - b_2 \mid + \mid a_3 - b_3 \mid \tag{3.4}$$

### 3.4.3. Distancia Chebyshev

La distancia de Chebyshev entre dos puntos en un espacio euclidiano, al igual que la distancia Manhatthan, utiliza la diferencia de los valores absolutos de las coordenadas de estos puntos como podemos apreciar en la Ecuación 3.5, pero utiliza el máximo valor entre todos los ejes de coordenadas, en vez de la suma de estos como sucede en la distancia Manhatthan. El problema con esta medida de distancia, al igual que la distancia Manhatthan, es que depende de la ubicación de los puntos en el espacio euclidiano y no sólo de los puntos en cuestión.

Punto A: $(a_1, a_2, a_3)$ Punto B: $(b_1, b_2, b_3)$

$$DistanciaChebyshev_{ab} = max\left(\mid a_1 - b_1 \mid, \mid a_2 - b_2 \mid, \mid a_3 - b_3 \mid\right) \tag{3.5}$$

Como se mencionó anteriormente, los tipos de distancias que utilicen una orientación bien definida fueron descartados. Por este motivo el método que se utilizará será la distancia euclidiana, ya que esta distancia es independiente de la orientación que tengan los objetos e independiente a como están descritos en el plano con respecto al origen (coordenada cero).

### 3.4.4. Medida de distancia receptor-centro geométrico-ligando (RCL)

Como no se podrán ocupar los métodos que dependan de la orientación de los puntos en el espacio cartesiano, se propone otra perspectiva en la utilización de la distancia euclidiana, para así tener mayor variedad a la hora de analizar los datos obtenidos. La distancia propuesta consiste en utilizar el centro geométrico de la zona de interacción como punto intermedio entre los aminoácidos interactuantes, siendo la distancia propuesta la distancia euclidiana del residuo receptor al centro geométrico de la zona de interacción, sumado la distancia del centro geométrico al residuo ligando, como se muestra en la Ecuación 3.6. Esta distancia propuesta se utilizará con el enfoque de centro geométrico visto en la sección anterior.

La finalidad de utilizar éste método es brindarle mayor importante al centro geométrico de la interacción proteína proteína, con lo que obtendremos distancias pequeñas cuando los aminoácidos interactuantes se encuentran cerca de éste centro, y más grandes cuando se evalúen aminoácidos más lejanos del centro geométrico de la interacción.

Punto A: $(a_1, a_2, a_3)$ Punto B: $(b_1, b_2, b_3)$ Centro geométrico: $(c_1, c_2, c_3)$

$$Distancia_{ab} = \sqrt{(a_1 + c_1)^2 + (a_2 + c_2)^2 + (a_3 + c_3)^2} + \sqrt{(c_1 + b_1)^2 + (c_2 + b_2)^2 + (c_3 + b_3)^2} \tag{3.6}$$

En la Figura 3.3 y la Figura 3.4 podemos visualizar la diferencia entre los métodos de distancias. En ambas, el rectángulo rojo representa la proteína A, y el azul la proteína B, los círculos rojos y azules que se encuentran en la zona de interacción (sección naranja con celeste), representan a los aminoácidos que interactúan en esta zona. La Figura 3.3 corresponde al método de la distancia euclidiana, la distancia entre los aminoácidos interactuantes se encuentra representada por las lineas negras que unen a los aminoácidos. La Figura 3.4 corresponde a la distancia RCL, el círculo verde representa el centro geométrico de la zona de interacción y las lineas purpuras, roja y negra, representan la distancia entre los pares de aminoácidos interactuantes, pasando por el centro geométrico de la zona de interacción.

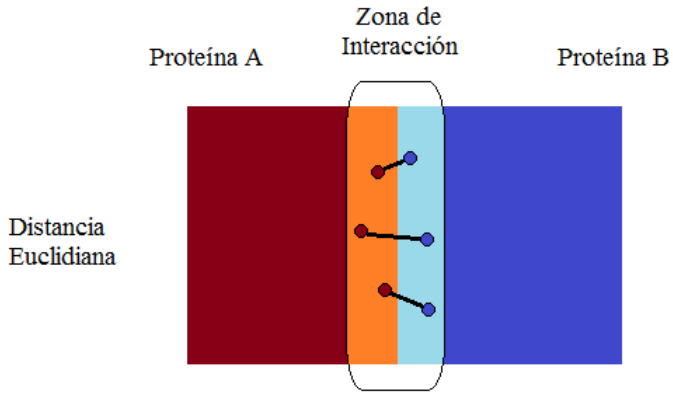


Figura 3.3: Representación de la distancia euclidiana aplicada en las interacciones proteína-proteína.

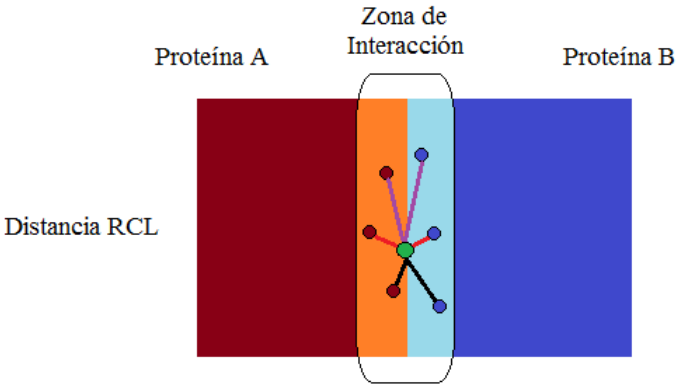


Figura 3.4: Representación de la distancia RCL aplicada en las interacciones proteína-proteína.

Ya definidos los enfoques de manejo de distancias a utilizar, así como también los métodos para el cálculo de estas distancias, debemos desarrollar los programas que nos permitan obtener la aplicación de dicho conocimiento. En el siguiente capítulo veremos los algoritmos que se desarrollaron para lograr este objetivo.

# Capítulo 4

# Implementación de algoritmos

Ya definidos los enfoques a utilizar y las distancias que se aplicarán a estos, se deben crear los programas correspondientes para cada uno de ellos. Para esto se utilizó el lenguaje de programación Python. Se escogió Python porque es uno de los lenguajes más utilizados en bioinformática, por ser un lenguaje interpretado, muy documentado, multiplataforma y por poseer una sintaxis limpia que simplifica la comprensión de los códigos aunque lo haya escrito otra persona.

En esta sección se mostrarán los pseudo-códigos de los algoritmos desarrollados, en el apéndice B se exhiben los códigos completos.

Como deseamos crear coordenadas que representen a cada residuo para luego ser utilizadas en el cálculo de distancias entre los residuos que interactúan, por esto se creó un nuevo archivo para cada proteína, el cual contiene la coordenada que representa el enfoque utilizado en cada residuo, acompañado al identificador numérico de este.

Para comprender mejor los programas desarrollados, en la Figura 4.1 se muestra como están organizados los archivos producto de la ejecución de la aplicación FastContact. En la figura podemos observar que los archivos se encuentran separados en dos carpetas según su clasificación (permanente o transitorio), y dentro encontramos las carpetas que contienen los datos de cada complejo proteico. El nombre de esta carpeta corresponde al código identificador del complejo proteico en el repositorio Protein Data Bank. Por último, en la tercera columna de la figura podemos observar los tres archivos que entrega FastContact por complejo. El archivo que contiene los datos de los residuos que más contribuyen energéticamente en el complejo, al igual que la carpeta que lo contiene, posee su código identificador como nombre, seguido por la extensión de

texto ".txt". los archivos "fort.19" y "fort.20" la información estructural de las dos proteínas que conforman el complejo. Cabe señalar que los archivos "CadenaA.19" y "CadenaB.20" no varían en su nombre en los diferentes complejos.

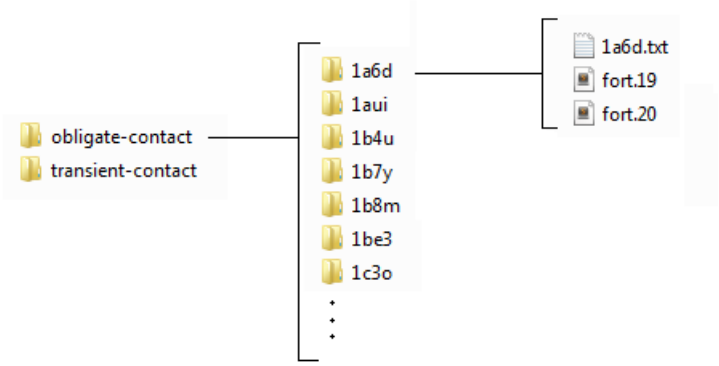


Figura 4.1: Contenido de las carpetas de complejos proteicos.

## 4.1. Desarrollo de programas de manejo de coordenadas

Todos los programas desarrollados para el manejo de coordenadas comparten una sección de código la cual se encarga de leer los archivos fort.19 y fort.20, también de crear una lista que agrupa los elementos químicos y sus respectivas coordenadas por residuo y luego, llamar a la función encargada del manejo de dicha lista. Esta función da la especificidad al programa, por lo cual es la que se expondrá por cada uno de los enfoques vistos en la sección 3.3. Para evitar redundancia a la hora de explicar los códigos desarrollados, la sección del código compartida por todos los programas se mostrará sólo una vez.

### 4.1.1. Segmento compartido por todos los programas

Este segmento es el que todos los programas para el manejo de coordenadas utilizan, su función es recorrer cada carpeta correspondiente a un complejo proteico, abrir los dos archivos que contienen la información de la estructura tridimensional de cada proteína (fort.19 y fort.20), aplicar el enfoque de manejo de coordenadas que corresponde al programa, y guardar los resultados en dos nuevos archivos. Lo único que cambia de un programa con otro, es la "Función Enfoque", la cual representa el enfoque de manejo de coordenadas correspondiente.

**Algoritmo 1:** Segmento compartido por todos los programas

```
1  Función enfoque [función que varía según el enfoque utilizado en el programa]
2  Entrada: Lista de elementos químicos con sus coordenadas
3  Salida: Coordenada con el enfoque aplicado
4  codigoAuxiliar=1 [variable que ayuda a recorrer los elementos del residuo]
5  ElementosResiduo=[] [lista que contendrá los elementos químicos del residuo en revisión,
     con sus respectivas coordenadas]
6  para cada carpeta que contenga los archivos de un complejo proteico hacer
7      abrir archivo fort.19 y fort.20 y crear dos archivos, en los cuales se guardarán los
         resultados de los enfoques aplicados a la cadena A y cadena B
8      para cada linea del archivo fort.19 hacer
9          si la primera columna del archivo corresponde a "ATOM" entonces
10             si El número identificador del residuo es igual al codigoAuxiliar entonces
11                 guardar coordenadas junto a nombres de los elementos químicos en
                     ElementosResiduo
12             en otro caso
13                 Aplicar función ´´enfoque" a la lista ElementosResiduo
14                 guardar resultado junto al nombre del residuo en el archivo creado para la
                     cadena A
15                 codigoAuxiliar aumenta una unidad
16             fin
17         fin
18     fin
19     codigoAuxiliar=0
20     para cada linea del archivo fort.20 hacer
21         si la primera columna del archivo "ATOM" entonces
22             si El número identificador del residuo es igual al codigoAuxiliar entonces
23                 guardar coordenadas junto a nombres de los elementos químicos en
                     ElementosResiduo
24             en otro caso
25                 Aplicar función ´´enfoque" a la lista ElementosResiduo
26                 guardar resultado junto al nombre del residuo en el archivo creado para la
                     cadena A
27                 codigoAuxiliar aumenta una unidad, indicando que se sigue con el siguiente
                     residuo y sus respectivos elementos químicos que lo conforman
28             fin
29         fin
30     fin
31 fin
```

### 4.1.2. Función coordenada de centro geométrico

Esta función toma las coordenadas de los elementos de un residuo, y retorna el promedio de estas coordenadas, lo que corresponde al centro geométrico del residuo en cuestión.

**Algoritmo 2:** Función coordenada de centro geométrico

**Resultado:** El programa completo entrega un archivo de texto con una coordenada por cada residuo en vez de una coordenada por cada elemento químico

```
1 Funcion enfoque centro geométrico
2 Entrada: Lista de coordenadas
3 sumaDeCoodenadas=0
4 para cada coordenada que pertenece a la lista de coordenadas hacer
5 |  sumaDeCoordenadas=sumaDeCoordenadas + coordenada
6 fin
7 devolver suma de coordenadas dividido el numero coordenadas que contiene la lista
```

### 4.1.3. Función coordenada de centro de masa total

Para la función centro de masa, necesitamos la masa atómica de los elementos químicos que forman los aminoácidos, para esto se utilizó la tabla de masas atómicas que entrega el Nacional Institute of Standards and Technology (Instituto Nacional de Estándares y Tecnología) [6].

En la sección 2.1.2 hablamos de los elementos químicos que conforman a los aminoácidos, los cuales son 5: Carbono, hidrógeno, oxígeno, nitrógeno y en no todos los casos, azufre. Por este motivo no es necesario incluir las masas atómicas de todos los elementos, sólo de los elementos ya mencionados. En la Tabla 4.1 podemos ver las masas atómicas de dichos elementos.

| **Elemento** | **Masa atómica** |
|---|---|
| Carbono | 12.011 |
| Hidrógeno | 1.008 |
| Oxígeno | 15.999 |
| Nitrógeno | 14.007 |
| Azufre | 32.06 |

Tabla 4.1: Masa atómica de los elementos químicos que componen a los aminoácidos[6].

**Algoritmo 3:** Función coordenada de centro de masa total

```
Resultado: Archivo de texto que contiene el centro de masa total de todos los residuos
codigoAuxiliar=1 [variable que ayuda a recorrer los elementos del residuo]
ElementosResiduo=[] [lista que contendrá los elementos químicos del residuo en revisión,
  con sus respectivas coordenadas]
Función enfoque centro de masa total
Entrada: Lista de elementos químicos con sus coordenadas
sumaElementos=0
masaTotal=0
para cada par de elementos (elemento químico/coordenadas) que pertenecen a la lista
  hacer
    sumaElementos = sumaElementos + coordenada * masa atómica del elemento químico
    masaTotal = masaTotal + masa atómica del elemento químico
fin
devolver SumaElementos/masaTotal
```

### 4.1.4. Función coordenada de centro de masa de la cadena lateral

La función de coordenada de centro de masa de la cadena lateral presenta sólo una restricción con respecto a la función de coordenada de centro de masa total, esta restricción es que los elementos que serán utilizados para calcular el centro de masa del residuo corresponden a todos los elementos distintos a “CA”, “C”, “CB”, “O” y “N”.

**Algoritmo 4:** Extracto de programa de coordenada de centro de masa de la cadena lateral

```
Función enfoque centro de masa cadena lateral
Entrada: Lista de elementos químicos con sus coordenadas
sumaElementos = 0
masaTotal = 0
para cada par de elementos (elemento químico y su coordenada) que pertenecen a la lista
  hacer
    si el elemento químico es distinto a CA, C, CB, O o N entonces
      sumaElementos = sumaElementos + coordenada * masa atómica del elemento
        químico
      masaTotal = masaTotal + masa atómica del elemento químico
    fin
fin
devolver SumaElementos dividido por masaTotal
```

### 4.1.5. Función coordenada de centro de masa de la cadena base

La función de coordenada de centro de masa de la cadena base presenta sólo una restricción con respecto a la función de coordenada de centro de masa total, esta restricción es que los elementos que serán utilizados para calcular el centro de masa del residuo corresponden a los elementos "CA", "C", "CB", "O" y "N".

**Algoritmo 5:** Extracto de programa de coordenada de centro de masa de la cadena base

```
1  Función Enfoque centro de masa cadena base
2  Entrada: Lista de elementos químicos con sus coordenadas
3  sumaElementos = 0
4  masaTotal = 0
5  para cada par de elementos (elemento químico y su coordenada) que pertenecen a la lista
     hacer
6  |  si el elemento químico es CA, C, CB, O o N entonces
7  |  |  sumaElementos = sumaElementos + coordenada * masa atómica del elemento
   |  |   químico
8  |  |  masaTotal = masaTotal + masa atómica del elemento químico
9  |  fin
10 fin
11 devolver SumaElementos dividido por masaTotal
```

### 4.1.6. Función coordenada del carbono alfa (CA)

Esta función retorna la coordenada del elemento "CA" por cada conjunto de datos de un residuo.

**Algoritmo 6:** Extracto de programa de coordenada CA

```
1  Función Enfoque CA
2  Entrada: Lista de elementos químicos con sus coordenadas
3  sumaElementos = 0
4  masaTotal = 0
5  para cada elemento químico que pertenece a la lista hacer
6  |  si el elemento químico es CA entonces
7  |  |  devolver coordenada CA
8  |  fin
9  fin
```

### 4.1.7. Función coordenada del carbono beta (CB)

Esta función retorna la coordenada del elemento "CB" por cada conjunto de datos de un residuo. Algunos residuos no poseen carbono beta, si esto ocurre ocuparemos su carbono alfa como alternativa.

**Algoritmo 7:** Extracto de programa de coordenada CB

```
1  Función Enfoque CB
2  Entrada: Lista de elementos químicos con sus coordenadas
3  coordenadaSecundaria [variable que guarda la coordenada CA y en caso de que no exista
     coordenada CB en el residuo, la función devuelve esta coordenada en cambio]
4  para cada elemento químico que pertenece a la lista hacer
5  |  si el elemento químico es CB entonces
6  |  |  devolver coordenada CB
7  |  fin
8  |  si elemento químico es CA entonces
9  |  |  coordenadaSecundaria = coordenada CA
10 |  fin
11 fin
12 devolver coordenada CA
```

Ejecutando todos los programas obtenemos 10 nuevos archivos en cada carpeta en donde se encuentran ubicados los complejos, los cuales corresponden a los cinco enfoques aplicados a cada par de proteínas.

| Enfoque | fort.19 | fort.20 |
|---|---|---|
| centro geométrico | cg.19 | cg.20 |
| centro de masa total | cmAll.19 | cmAll.20 |
| centro de masa cadena lateral | cmLat.19 | cmLat.20 |
| centro de masa cadena base | cmBase.19 | cmBase.20 |
| elemento CA | CA.19 | CA.20 |
| elemento CB | CB.19 | CB.20 |

Tabla 4.2: Archivos generados por la ejecución de los programas.

En la Figura 4.2 podemos observar el resultado que se produce al ejecutar el programa "coordenada de centro geométrico" en el complejo "1a6d". Cada residuo pasa a ser representado por una coordenada, en contraste a las cuantiosas coordenadas del archivo "fort".

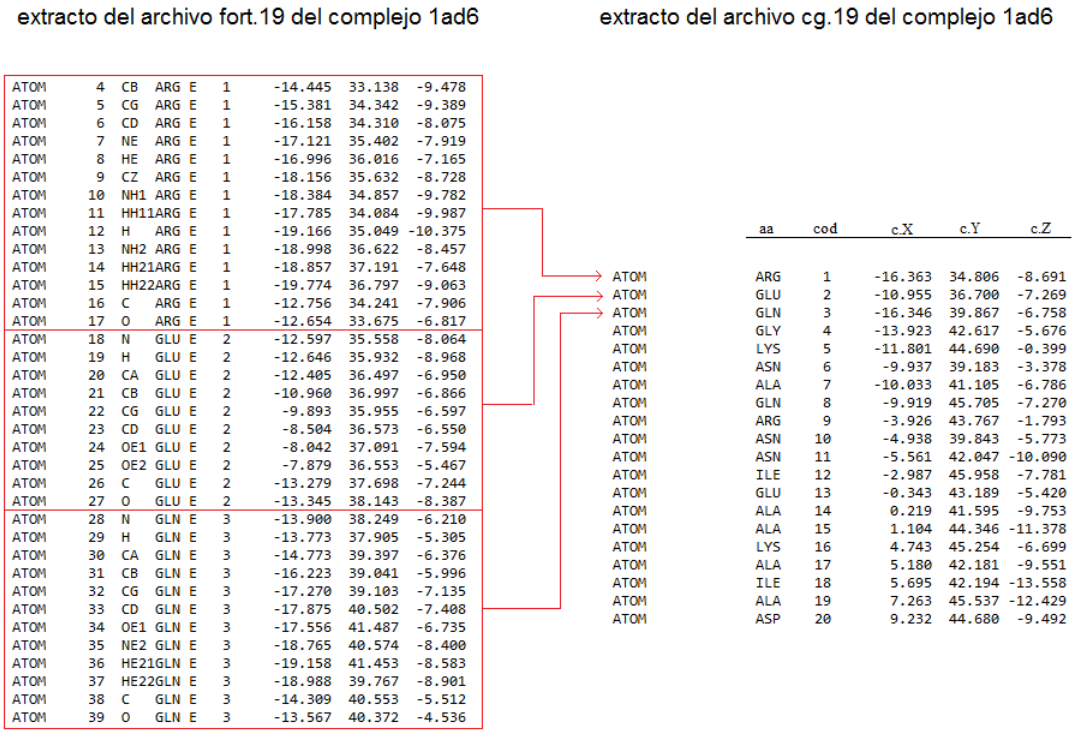


Figura 4.2: Relación entre el archivo "fort.19" y el archivo "cg.19". Significado de las columnas: aa: Nombre del aminoácido, cod: Código identificador de cada aminoácido, c.X: Coordenada en el eje X del aminoácido, c.Y: Coordenada en el eje Y del aminoácido, c.Z: Coordenada en el eje Z del aminoácido.

## 4.2. Implementación de programas para la medición de distancias

En esta sección se expondrá la implementación en pseudo-código tanto de los métodos de medición de distancia euclidiana y la propuesta del método de medición vistos en la Sección 3.4 para los archivos generados por los programas de la Sección 4.1. Los códigos completos se encuentran en el apéndice B.

La función principal de los programas que se expondrán a continuación es añadir nuevas características al archivo de salida de FastContact (archivo que contiene las energías más significativas de la zona de interacción) de cada complejo proteico, específicamente la sección de interacción entre aminoácidos receptor-ligando.

**Algoritmo 8:** Programa de distancia sobre el centro geométrico

```
1  para cada carpeta que contenga los archivos de un complejo proteico hacer
2      abrir archivos cg.19, cg.20 y IDcomplejo.txt [archivo FastContact, ej: 1a6d.txt]
3      crear archivo fcJoin_IDcomplejo.txt que contendrá las nuevas características
4      para cada linea del archivo IDcomplejo.txt que contenga una interacción
         receptor-ligando hacer
5          receptor = [columna que corresponde a la ID del receptor]
6          ligando = [columna que corresponde a la ID del ligando]
7          para cada linea en archivo cg.19 hacer
8              si la linea corresponde a "receptor" entonces
9                  guardar coordenadas en "cReceptor"
10             fin
11         fin
12         para cada linea en archivo cg.20 hacer
13             si la linea corresponde a "receptor" entonces
14                 guardar coordenadas en "cLigando"
15             fin
16         fin
17         aplicar función de distancia euclidiana entre la coordenada cReceptor y cLigando y
             guardarla en "distanciaRL"
18         copiar linea del archivo IDcomplejo.txt y concatenar "distanciaRL" y guardar en
             fcJoin_IDcomplejo.txt
19     fin
20 fin
```

Los programas para el cálculo de distancias sobre los enfoques restantes (centro de masa total, centro de masa cadena lateral, centro de masa cadena base, carbono alfa y carbono beta) comparten gran parte del código del programa de distancia sobre el centro geométrico, cambiando en sólo:

- Linea 2: los archivos cg.19 y cg.20 por el enfoque correspondiente (ej: cmAll.19 y cmAll.20)
- Linea 3: Se debe abrir el archivo fcJoin_IDcomplejo.txt en vez de crearlo
- Linea 7: el archivo cg.19 por el enfoque correspondiente
- Linea 12: el archivo cg.20 por el enfoque correspondiente
- Linea 18: sólo se debe modificar el archivo fcJoin_IDcomplejo.txt concatenando "distanciaRL"

El Algoritmo 9 y su continuación (Algoritmo 10) de distancia RCL (receptor-centro geométrico-ligando) consta de dos secciones principales, la primera calcula el centro geométrico de la zona de interacción con los residuos receptores y ligandos, y la segunda calcula la distancia de cada par de residuos receptor-ligando con el centro geométrico.

**Algoritmo 9:** programa de distancia RCL

```
1  para cada carpeta que contenga los archivos de un complejo proteico hacer
2      abrir archivos cg.19, cg.20 y IDcomplejo.txt [archivo FastContact, ej: 1a6d.txt]
3      archivo archivo fcJoin_IDcomplejo.txt
4      listaReceptor=[] listaLigando=[] sumaCoordenadas=[0,0,0] contadorCoordenadas=0
5      para cada linea del archivo IDcomplejo.txt que pertenece a la sección de residuos
         ligando hacer
6          ligando=[columna que corresponde a la ID del ligando]
7          si ligando no pertenece a la lista listaLigando entonces
8              guardar ligando en listaLigando
9              para cada linea en archivo cg.20 hacer
10                 si la linea corresponde a "receptor" entonces
11                     guardar coordenadas en "cLigando"
12                 fin
13             fin
14             sumar cLingando a sumaCoordenadas
15             aumentar contadorCoordenadas en una unidad
16         fin
17     fin
18     para cada linea del archivo IDcomplejo.txt que pertenece a la sección de residuos
         receptor hacer
19         receptor=[columna que corresponde a la ID del receptor]
20         si receptor no pertenece a la lista listaReceptor entonces
21             guardar receptor en listaReceptor
22             para cada linea en archivo cg.19 hacer
23                 si la linea corresponde a "receptor" entonces
24                     guardar coordenadas en "cReceptor"
25                 fin
26             fin
27             sumar cReceptor a sumaCoordenadas
28             aumentar contadorCoordenadas en una unidad
29         fin
30     fin
31
```

**Algoritmo 10:** Continuación de programa de distancia RCL

```
1  para cada linea del archivo IDcomplejo.txt que contenga una interacción receptor-ligando fin
   hacer
2      receptor = [columna que corresponde a la ID del receptor]
3      ligando = [columna que corresponde a la ID del ligando]
4      si receptor no pertenece a la lista listaReceptor entonces
5          guardar receptor en listaReceptor
6          para cada linea en archivo cg.20 hacer
7              si la linea corresponde a "ligando" entonces
8                  guardar coordenadas en "cLigando"
9          sumar cReceptor a sumaCoordenadas
10         aumentar contadorCoordenadas en una unidad
11     si ligando no pertenece a la lista listaLigando entonces
12         guardar ligando en listaLingando
13         para cada linea en archivo cg.19 hacer
14             si la linea corresponde a "ligando" entonces
15                 guardar coordenadas en "cLingando"
16         sumar cLigando a sumaCoordenadas
17         aumentar contadorCoordenadas en una unidad
18 coordenadaCentroGeometrico = sumaCoordenadas dividido contadorCoordenadas
19 para cada linea del archivo IDcomplejo.txt que contenga una interacción receptor-ligando
   hacer
20     receptor = [columna que corresponde a la ID del receptor]
21     ligando = [columna que corresponde a la ID del ligando]
22     para cada linea en archivo cg.19 hacer
23         si la linea corresponde a "receptor" entonces
24             guardar coordenadas en "cReceptor"
25     para cada linea en archivo cg.20 hacer
26         si la linea corresponde a "ligando" entonces
27             guardar coordenadas en "cLigando"
28     aplicar función de distancia euclidiana entre la coordenada cReceptor con
       coordenadaCentroGeometrico y coordenadaCentroGeometrico con cLingado y
       guardarla en "distanciaRCL"
29     copiar linea del archivo IDcomplejo.txt y concatenar "distanciaRCL" y guardar en
       fcJoin_IDcomplejo.txt
```

Luego de ejecutar todos los programas, contaremos con un archivo de texto (fcJoin_IDcomplejo.txt) por cada complejo proteico, el cual contiene los siete enfoques de distancias, como podemos ver en la Figura 4.3.

| Energy | AA-A | AA-B | GC | CA | CB | CmAll | CmBase | CmLat | RCL | Complex: 1a6d |
|---|---|---|---|---|---|---|---|---|---|---|
| Top 20 Min & Max receptor-ligand residue electrostatic contacts | | | | | | | | | | |
| -9.711 | 32 ASP | 496 ARG | 4.321 | 5.636 | 4.515 | 4.292 | 4.812 | 3.737 | 58.693 | |
| -9.588 | 253 GLU | 228 LYS | 7.608 | 11.168 | 8.696 | 8.035 | 11.117 | 4.793 | 56.018 | |
| -8.074 | 33 LYS | 498 ASP | 4.833 | 6.106 | 7.474 | 5.428 | 6.721 | 4.049 | 63.061 | |
| -7.925 | 231 LYS | 233 ASP | 6.739 | 9.339 | 8.380 | 7.605 | 10.000 | 4.875 | 57.963 | |
| -7.130 | 288 LYS | 317 GLU | 7.604 | 10.477 | 8.075 | 8.179 | 10.701 | 5.449 | 57.617 | |
| -6.868 | 210 LYS | 322 ASP | 8.479 | 11.393 | 10.107 | 9.218 | 12.139 | 5.570 | 53.028 | |
| -4.988 | 41 ASP | 59 LYS | 7.800 | 10.103 | 7.413 | 8.000 | 10.172 | 5.237 | 40.796 | |
| -3.558 | 33 LYS | 499 ASP | 4.492 | 5.010 | 4.942 | 4.833 | 4.964 | 5.314 | 61.550 | |
| -3.446 | 237 LYS | 233 ASP | 4.440 | 4.748 | 4.520 | 4.620 | 4.757 | 5.471 | 66.078 | |
| -3.112 | 258 LYS | 317 GLU | 9.485 | 12.513 | 10.926 | 10.015 | 13.371 | 6.262 | 59.681 | |
| -2.228 | 29 LYS | 105 GLU | 9.398 | 11.404 | 10.859 | 9.901 | 12.521 | 7.026 | 75.506 | |
| -2.103 | 145 GLY | 496 ARG | 7.291 | 9.930 | 8.965 | 7.367 | 9.525 | 8.141 | 61.596 | |
| -1.692 | 30 GLY | 496 ARG | 6.038 | 8.259 | 7.979 | 6.045 | 7.207 | 6.629 | 63.732 | |
| -1.315 | 231 LYS | 231 GLU | 7.209 | 8.191 | 7.128 | 7.469 | 8.448 | 7.872 | 52.422 | |
| -1.305 | 367 GLU | 496 ARG | 9.687 | 12.857 | 9.928 | 9.967 | 12.819 | 7.879 | 52.009 | |
| -1.225 | 358 ARG | 69 ASP | 10.315 | 12.243 | 11.192 | 10.479 | 12.985 | 8.362 | 39.250 | |
| -1.189 | 253 GLU | 230 PRO | 7.383 | 8.036 | 7.050 | 7.348 | 9.124 | 6.181 | 57.946 | |
| -1.092 | 279 ASP | 228 LYS | 12.202 | 14.767 | 13.012 | 12.458 | 15.452 | 8.514 | 46.623 | |
| -1.049 | 23 ARG | 498 ASP | 9.645 | 11.220 | 10.246 | 9.560 | 10.602 | 8.712 | 71.986 | |
| -0.807 | 37 ASP | 59 LYS | 10.583 | 13.248 | 12.524 | 11.373 | 13.035 | 10.047 | 43.165 | |
| --- | | | | | | | | | | |
| 0.746 | 463 ASP | 105 GLU | 12.693 | 14.282 | 12.905 | 12.485 | 15.071 | 9.382 | 82.974 | |
| 0.755 | 231 LYS | 232 PHE | 4.765 | 6.662 | 6.221 | 5.013 | 8.215 | 4.608 | 56.467 | |
| 0.792 | 338 ASP | 69 ASP | 9.597 | 9.781 | 10.063 | 9.486 | 10.389 | 9.200 | 34.148 | |
| 0.801 | 247 GLN | 257 GLU | 9.526 | 11.798 | 9.525 | 9.442 | 12.360 | 6.421 | 73.378 | |
| 0.844 | 210 LYS | 310 SER | 7.086 | 9.105 | 7.422 | 7.907 | 10.135 | 4.840 | 49.403 | |
| 0.877 | 236 ALA | 233 ASP | 5.388 | 6.739 | 6.592 | 5.350 | 5.624 | 5.380 | 64.012 | |
| 0.914 | 24 THR | 498 ASP | 6.524 | 8.411 | 5.772 | 6.424 | 8.006 | 3.992 | 68.839 | |
| 0.961 | 29 LYS | 411 ARG | 11.225 | 15.315 | 13.128 | 12.244 | 15.522 | 9.651 | 84.060 | |
| 0.975 | 237 LYS | 229 LYS | 11.367 | 13.169 | 13.581 | 11.980 | 13.585 | 10.070 | 64.964 | |
| 1.040 | 32 ASP | 498 ASP | 6.036 | 6.176 | 7.185 | 6.452 | 5.612 | 8.104 | 61.910 | |
| 1.096 | 361 THR | 63 GLU | 6.791 | 7.486 | 7.257 | 6.241 | 6.905 | 6.586 | 38.677 | |
| 1.194 | 146 LYS | 496 ARG | 7.902 | 9.328 | 9.265 | 7.998 | 9.437 | 8.135 | 57.740 | |
| 1.205 | 237 LYS | 235 ASN | 5.729 | 7.477 | 6.384 | 6.067 | 7.402 | 4.355 | 71.805 | |
| 1.581 | 239 GLN | 237 ARG | 4.526 | 6.259 | 5.042 | 5.109 | 6.640 | 4.620 | 83.640 | |
| 1.715 | 280 ASP | 222 ASP | 9.221 | 10.731 | 8.533 | 8.909 | 10.253 | 7.052 | 38.487 | |
| 2.134 | 279 ASP | 231 GLU | 10.395 | 12.091 | 9.610 | 10.105 | 12.996 | 6.473 | 44.419 | |

Figura 4.3: Extracto de "fcJoin_1a6d.txt", archivo que contiene los 7 enfoques de distancias aplicados al complejo 1a6d. De izquierda a derecha: Energy: Contribución energética producto de la interacción, AA-A: Aminoácido A, AA-B: Aminoácido B, GC: Centro Geométrico, CA: Carbono Alfa, CB: Carbono Beta, CmAll: Centro de Masa Total, CmBase: Centro de Masa Cadena Base, CmLat: Centro de Masa Cadena Lateral, RCL: Receptor-Centro Geométrico-Ligando y Complex: Código identificador de del complejo protéico.

El último paso para finalizar el proceso obtención de datos, es unir todos los archivos creados en dos archivos de texto, uno que contenga todos los datos de los complejos transitorios y otro que contenga los complejos permanentes. El algoritmo 11 se encarga de esta función.

**Algoritmo 11:** programa de unión de datos de complejos transitorios y permanentes

```
1 crear los archivos "fcAllComplexTransient.txt" y "fcAllComplexObligate.txt", archivos de
    texto que guardarán los datos de lo enfoques de distancia de todos los complejos
    proteicos, separados en complejos
2 para cada carpeta perteneciente a transient-contacts (carpeta de complejos transitorios)
    hacer
3 |  abrir archivo "fcJoin_IDcomplejo.txt"
4 |  escribir en el archivo "fcAllComplexTransient.txt" los datos de
  |   "fcJoin_IDcomplejo.txt"
5 fin
6 para cada carpeta perteneciente a obligate-contacts (carpeta de complejos permanentes)
    hacer
7 |  abrir archivo "fcJoin_IDcomplejo.txt"
8 |  escribir en el archivo "fcAllComplexObligate.txt" los datos de
  |   "fcJoin_IDcomplejo.txt"
9 fin
```

## 4.3. Matrices de características energéticas y de distancias

Con los datos obtenidos mediante la ejecución de FastContact, sumado los datos generados por medio de los enfoques de distancias aplicados a estos, tenemos nuestro conjunto de datos a analizar. El siguiente paso es utilizar estos datos para la creación de conjuntos de características, para obtener un medio de comparación entre las diferentes características de distancia, así como también las características energéticas.

FastContact genera 1.000 características por complejo proteico, de las cuales 280 son características energéticas, 360 características corresponden al número identificador del aminoácido en la cadena proteica y 360 características que corresponden al nombre del aminoácido. De estas 1000 características se utilizarán las 280 correspondientes a las energías.

Por cada enfoque de distancia se generaron 80 nuevas características, como se puede observar en la Tabla 4.3, por lo tanto se generaron 560 nuevas características que, en conjunto con las energías hace un total 840 características. A estas características debemos también incorporarle la clase de interacción del complejo proteico, por lo tanto el total de características es de 841. La clase de interacción es la única característica no numérica de nuestras matrices.

| Enfoque de distancia | Receptor-ligando contactos electroestáticos | | Receptor-ligando contactos energía libre | | Total |
|---|---|---|---|---|---|
| | top 20 max | top 20 min | top 20 max | top 20 min | |
| Centro geométrico | 20 | 20 | 20 | 20 | 80 |
| Centro de masa total | 20 | 20 | 20 | 20 | 80 |
| Centro de masa cadena lateral | 20 | 20 | 20 | 20 | 80 |
| Centro de masa cadena base | 20 | 20 | 20 | 20 | 80 |
| Carbono alfa | 20 | 20 | 20 | 20 | 80 |
| Carbono beta | 20 | 20 | 20 | 20 | 80 |
| Receptor-centro geométrico-ligando | 20 | 20 | 20 | 20 | 80 |
| | | | | Total | 560 |

Tabla 4.3: Características de distancia entre aminoácidos.

Los conjuntos de datos se guardarán como matrices, en donde cada fila corresponde a un complejo proteico, y cada columna corresponde a un tipo de característica. Por este motivo cada matriz tendrá 296 filas y el número de columnas variará dependiendo de las características que se desean en cada una de las matrices.

### 4.3.1. Matriz sólo con características energéticas

Esta matriz contiene sólo las energías de nuestro conjunto de datos, por lo tanto tiene una dimensión de 296x281 y será la matriz que nos servirá como base para comparar nuestros enfoques de distancias.

| |
|---|
| energías libres de unión |
| energías de desolvatación (ligandos) |
| energías electroestáticas (ligandos) |
| energías de desolvatación (receptores) |
| energías electroestáticas (receptores) |
| energías de desolvatación (receptor-ligando) |
| energías electroestáticas (receptor-ligando) |
| clase de interacción |

Tabla 4.4: Características en la matriz de sólo energías.

### 4.3.2. Matriz con todas las características

La siguiente matriz contiene todas las características, 841 por cada complejo proteico, por lo tanto su dimensión es de 296x841. En la Tabla 4.5 podemos ver como están organizadas las características. Las filas de la tabla representan las columnas de la matriz de características. las primeras 200 columnas corresponden a las energías que no poseen contactos receptor-ligando. Las columnas siguientes corresponden a estos contactos, anotando primero la energía del contacto y seguido por los siete enfoques de distancias asociados a él, este formato se repite con los 79 contactos receptor-ligando siguientes y terminando con la característica que representa la clase de interacción del complejo.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Energías libres de unión | | | | | | | |
| Energías de desolvatación (ligandos) | | | | | | | |
| Energías electroestáticas (ligandos) | | | | | | | |
| Energías de desolvatación (receptores) | | | | | | | |
| Energías electroestáticas (receptores) | | | | | | | |
| energía desolvatación (receptor-ligando) | CG | CM all | CM lateral | CM base | CA | CB | RCL |
| energía electroestática (receptor-ligando) | CG | CM all | CM lateral | CM base | CA | CB | RCL |
| clase de interacción | | | | | | | |

Tabla 4.5: Organización de la matriz con todas las características. CG: Distancia Centro Geométrico, CM: Distancia Centro de Masa, CA: Distancia Carbono Alfa, CB: Distancia Carbono Beta, RCL: Distancia Receptor-Centro Geométrico-Ligando.

### 4.3.3. Matriz con todos los enfoques de distancia y su energía asociada

En esta matriz no se consideran las energías que no pertenezcan a las secciones de interacción receptor-ligando. Su dimensión es de 296x641.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| energía desolvatación (receptor-ligando) | CG | CM all | CM lateral | CM base | CA | CB | RCL |
| energía electroestática (receptor-ligando) | CG | CM all | CM lateral | CM base | CA | CB | RCL |
| clase de interacción | | | | | | | |

Tabla 4.6: Características presentes en la matriz de distancias más energía asociada. CG: Distancia Centro Geométrico, CM: Distancia Centro de Masa, CA: Distancia Carbono Alfa, CB: Distancia Carbono Beta, RCL: Distancia Receptor-Centro Geométrico-Ligando.

### 4.3.4. Matrices de sólo con distancias

Se crearon siete matrices que representan los siete enfoques de distancia desarrollados, cada matriz tiene una dimensión de 296x81, ya que sólo estamos considerando el enfoque de distancia (80 distancias por enfoque) y la clase de interacción en los 296 complejos proteicos, como podemos ver en la Tabla 4.7. Estas matrices nos ayudarán a discriminar entre los mejores y peores enfoques de distancia.

| **Matriz de Distancia** | **Filas** | **Columnas** |
|---|---|---|
| Centro Geométrico | 296 complejos | 80 distancias del enfoque centro geométrico más la clase de interacción |
| Centro de Masa Total | 296 complejos | 80 distancias del enfoque centro de masa total más la clase de interacción |
| Centro de Masa Cadena Base | 296 complejos | 80 distancias del enfoque centro de masa cadena base más la clase de interacción |
| Centro de Masa Cadena Lateral | 296 complejos | 80 distancias del enfoque centro de masa cadena lateral más la clase de interacción |
| Carbono Alfa | 296 complejos | 80 distancias del enfoque carbono alfa más la clase de interacción |
| Carbono Beta | 296 complejos | 80 distancias del enfoque carbono beta más la clase de interacción |
| RCL | 296 complejos | 80 distancias del enfoque RCL más la clase de interacción |

Tabla 4.7: Características presentes en las matrices con sólo distancias.

## 4.4. Software de aprendizaje automático e implementación de matrices

Para estudiar las matrices de características, se utilizó el software WEKA (Waikato Environment for Knowledge Analysis), que significa entorno para análisis del conocimiento de la Universidad de Waikato, y es un una herramienta desarrollada en Java que contiene una colección de algoritmos de aprendizaje automático para el estudio de tareas de minería de datos [32].

Los conjuntos de datos de entrenamiento que se utilizarán en WEKA deben estar en formato arff (Attribute-Relation File Format), el cual está especialmente diseñado para su utilización en WEKA. Este formato cuenta con 3 secciones, la primera corresponde al encabezado, en donde incluye el nombre de la relación, la segunda sección corresponde al nombre y tipo de datos que tienen las características y la última sección corresponde a los datos propiamente tal.

### 4.4.1. Matriz en formato arff

El formato arff permite tres comandos para escribir las secciones, "relation" para escribir el nombre de la relación, "attribute" para escribir las características que recibirá y, "data" para escribir los datos propiamente tal.

Para crear el archivo arff con el formato correspondiente, se creó un programa en Python, cuyo pseudocódigo corresponde al Algoritmo 12 y su continuación (Algoritmo 13), que utiliza las energías provenientes de los archivos "energiasComplejosT.txt" y "energiasComplejosO.txt" (estos archivos contienen todas las energías de cada archivo de salida de FastContact, el código del programa que permite conseguir estas características se puede ver en el Apéndice B bajo el nombre .Algoritmo que extrae las características energéticas de cada archivo fastContact"), como también las energías y enfoques de distancias que se encuentran en los archivos "fcAllComplexObligate.txt" y "fcAllComplexTransient.txt". El código completo se puede encontrar en el Apéndice B.

**Algoritmo 12:** programa que genera la matriz de características en formato arff

```
1  abrir archivos "fcAllComplexTransient.txt", "fcAllComplexObligate.txt",
     "energiasComplejosT.txt" y "energiasComplejosO.txt"
2  crear archivo "matrizCompleta.arff", el cual guardará la matriz de características en
     formato arff.
3  contador=0
4  escribir en matrizCompleta "relation Complejos"
5  para i en rango 1-201 hacer
6  |  escribir en matrizCompleta "@attribute-E-i numeric"
7  fin
8  para i en rango -81 hacer
9  |  escribir en matrizCompleta "attribute RL-EE-i numeric"
10 |  escribir "attribute RL-E-i numeric"
11 |  escribir "atribute GC-E-i numeric"
12 |  escribir "atribute CA-E-i numeric"
13 |  escribir "atribute CB-E-i numeric"
14 |  escribir "atribute CMall-E-i numeric"
15 |  escribir "atribute CMbase-E-i numeric"
16 |  escribir "atribute CMlat-E-i numeric"
17 |  escribir "atribute RCL-E-i numeric"
18 fin
19 Escribir en matrizCompleta "attribute class t,o" Escribir data"
20 para i menor al número de filas de "fcAllcomplexTransient.txt" hacer
21 |  si la linea "i" en el archivo fcAllcomplexTransient corresponde al comienzo de un
   |    complejo entonces
22 |  |  escribir en matrizCompleta la linea "contador" del archivo
   |  |    "energiasComplejoT.txt"
23 |  |  contador=contador+1
24 |  |  mientras la linea "i´´ del archivo fcAllcomplexTransient no sea el comienzo de
   |  |    otro complejo hacer
25 |  |  |  escribir en matrizCompleta la energía y los enfoques que pertenecen a la linea
   |  |  |    "i" del archivo fcAllcomplexTransient
26 |  |  fin
27 |  |  escribir ",t" en el archivo matrizCompleta [para indicar que la clase de interacción
   |  |    es transitoria]
28 |  fin
29 fin
30 contador=0
```

**Algoritmo 13:** Continuación de programa que genera la matriz de características en formato arff

```
1 para i menor al número de filas de "fcAllcomplexObligate.txt" hacer
2     si la linea "i" en el archivo fcAllcomplexObligate corresponde al comienzo de un
        complejo entonces
3         escribir en matrizCompleta la linea "contador" del archivo
            "energiasComplejoO.txt"
4         contador=contador+1
5         mientras la linea "i´´ del archivo fcAllcomplexObligate no sea el comienzo de otro
            complejo hacer
6             escribir en matrizCompleta la energía y los enfoques que pertenecen a la linea
                "i" del archivo fcAllcomplexObligate
7         escribir ",o" en el archivo matrizCompleta [para indicar que la clase de interacción
            es permanente]
```

El archivo "matrizCompleta.txt" corresponde a la matriz con todas las características, tanto energéticas como de distancias, de dimensión 296x841. Para crear las otras matrices se optó por utilizar una opción de WEKA que permite eliminar atributos y posteriormente guardar la matriz resultando de esa eliminación.

Ya obtenidas todas las matrices de características, tanto energéticas como de distancias, debemos utilizar el software de aprendizaje automático Weka para obtener los resultados que posteriormente serán analizados. En la siguiente sección se verá el proceso de evaluación y obtención de resultados mediante la aplicación de los algoritmos de aprendizaje automático que contiene WEKA.

# Capítulo 5

# Resultados

En este capítulo se definirán los clasificadores a utilizar para la evaluación de las matrices de características, además de como se manejarán los conjuntos de datos con respecto a la separación de conjuntos de datos para el entrenamiento del clasificador y los conjuntos de datos para probar este. Luego se definirán los datos de salida que se utilizarán producto de la aplicación de los clasificadores. Y por último se evaluarán estos resultados haciendo un análisis estadístico.

## 5.1. Clasificadores y datos de salida a utilizar

Los clasificadores (implementación de algoritmos de aprendizaje) pertenecientes a WEKA que se utilizaron para el análisis de las matrices de características son:

### 5.1.1. Clasificadores

- *NaiveBayes* (clasificador bayesiano ingenuo): Está basado en el Teorema de Bayes[1] y básicamente asume la independencia de las características, lo que quiere decir que si una

[1] Teorema de Bayes: Es un teorema que en pocas palabras, vincula la probabilidad de que un evento 'A'suceda dado un evento 'B'con la probabilidad de que un evento 'B'suceda dado 'A'

característica está presente o ausente, no influye en las demás características.

- *SimpleLogistic* (clasificador logística simple): Clasificador para construir modelos de regresión logística lineal.

- *Secuencial Minimal Optimization* (optimización mínima secuencial): Es un clasificador utilizado para entrenar máquinas de soporte de vectores, las cuales son modelos de aprendizaje supervisado utilizado para analizar datos de problemas de clasificación y regresión.

- *RandomCommittee* (comité aleatorio): Clase para construir un conjunto de clasificadores base aleatorizados. Cada clasificador base se construye utilizando un número semilla aleatorio diferente (pero basado en los mismos datos). La predicción final es un promedio de las predicciones generadas por los clasificadores base individuales.

- *DecisionTable* (tabla de decisiones): Clase para la construcción y uso de un clasificador de mayoría de la tabla de decisión simple. Las cuales consisten en una manera precisa y compacta de modelar conjuntos de condiciones (reglas) complejas y sus acciones correspondientes.

- *J48*: Clase para generar un árbol de decisiones C4.5 (algoritmo utilizado para crear un árbol de decisión) podado o sin podar.

- *LMT* (árbol de modelo logístico): Clasificador para la construcción de 'árboles de modelos logísticos', que son árboles de clasificación con funciones de regresión logística en las hojas.

- *RandomForest* (bosque aleatorio): Clase para construir una combinación de árboles predictores, tal que cada árbol depende de los valores de un vector aleatorio probado independientemente y con la misma distribución para cada uno de estos.

En la comunidad científica se tiene a usar el nombre en inglés de los clasificadores, por lo tanto para evitar posibles confusiones, se operará de la misma manera en este estudio.

### 5.1.2. Modelos de división de conjuntos de datos

Para evaluar los clasificadores, WEKA cuenta con cuatro modelos para la división de los conjuntos de datos (conjunto de datos de entrenamiento y conjunto de datos para probar la clasificación): Use training set, Supplied test set, Cross-validation folds y Percentage split.

#### Use Training Set

*Use training set* (usar un conjunto de entrenamiento), evalúa el clasificador con los mismos datos con los cuales se realizó la clasificación.

#### Supplied Test Set

*Supplied test set* (conjunto de prueba suministrado), evalúa el clasificador con un conjunto de datos independiente al que se utilizó para la clasificación.

#### Cross-validation Folds

*Cross-validation folds* (Pliegues de validación cruzada), modelo que divide los datos en "n" pliegues (número ingresado por el usuario), lo que permite que el clasificador sea aplicado utilizando un pliegue como conjunto de complejos a clasificar, mientras que el resto es utilizado como datos de entrenamiento. Este proceso se ejecuta "n" veces, cambiando cada vez el pliegue a utilizar. Luego el método devuelve el promedio de la clasificación de todos los pliegues.

#### Percentage Split

*Percentage split* (división porcentual) divide el conjunto de datos en dos partes con un porcentaje ingresado por el usuario, en donde una de las partes se utiliza como datos de entrenamiento, y la otra se utiliza para probar el clasificador.

Para nuestras matrices de características, cuya cantidad es considerablemente menor a la cantidad de datos que usualmente se manejan en minería de datos, se utilizó *cross-validation* con diez pliegues para la evaluación de los clasificadores, al utilizar este todas las secciones como conjunto de entrenamiento y de prueba, obteniendo una evaluación más más confiable.

### 5.1.3. Datos de salida utilizados

Al aplicar un clasificador a una matriz de características, WEKA devuelve estadísticas que ayudan a determinar si el clasificador obtuvo resultados positivos o si sus resultados son similares al azar. De todos los elementos estadísticos que se generan por cada clasificación, se utilizaron los siguientes: Complejos clasificados correctamente, Curva ROC y Coeficiente Phi.

#### Complejos clasificados correctamente ( %)

Este indicador es uno de los más utilizados para evaluar una clasificación, indica cual es el porcentaje del total de complejos que fue correctamente clasificados. El problema con este indicador es que no toma en consideración si las clases de los complejos está balanceada.

Los complejos se consideran como balanceados si cada clase contiene una cantidad similar de complejos. En nuestro conjunto de complejos la clase de complejos transitorios contiene 204 complejos, mientras que la de permanentes contiene 92. Como podemos apreciar nuestros complejos no están balanceados (la razón de complejos transitorios y permanentes es 1:2 aproximadamente). Esto puede afectar a la hora de discriminar entre una clasificación de otras. Un ejemplo de esto seria el caso en que un clasificador de como resultado que los 296 complejos son transitorios, lo que da un 69 % de complejos correctamente clasificados, lo cual erróneamente podría considerarse como una clasificación aceptable. Por este motivo es necesario utilizar indicadores que funcionen independientemente al balance de los complejos.

#### Curva ROC

La curva ROC (*Receiver Operating Characteristic*, que en español se traduce como característica operativa del receptor) es una representación gráfica de la razón de los complejos correctamente clasificados de una clase, y la razón de los correctamente clasificados de la otra. Es una curva que funciona independiente del balance entre los tipos de complejos y sólo funciona para un sistema de clasificación de dos tipos de complejos [33].

El área bajo la curva ROC, denominada AUC (*Area Under the Curve* que en español significa área bajo la curva), es uno de los parámetros que nos entrega WEKA al aplicar un clasificador, y puede representarse como la probabilidad de que, al clasificar un par de complejos, uno de cada clase, estos sean clasificados correctamente. El mínimo valor del coeficiente AUC es 0.5, que indica que la clasificación realizada es comparada con el azar, en caso contrario, la clasificación es perfecta cuando su coeficiente AUC es 1. Por lo general se consideran los siguientes intervalos:

- 0.90-1.00 = excelente
- 0.80-0.90 = bueno
- 0.70-0.80 = regular
- 0.60-0.70 = pobre
- 0.50-0.60 = malo

En nuestro estudio se considerará como una clasificación aceptable la que cuenta con un AUC mayor a 0.7, ya que bajo 0.7 podría traer desconfianza en nuestros resultados.

El parámetro AUC se utilizará como nuestro principal identificador para la validación de las matrices de características y de los clasificadores utilizados.

### Matthews Correlation Coefficient (MCC)

Matthews Correlation Coefficient (MCC) cuya traducción es coeficiente de correlación de Matthews y también llamado Coeficiente phi, es otro medidor de clasificaciones que no depende del balance de los tipos de complejos. Los valores que entrega este coeficiente varían entre 1 y -1, en donde 1 representa una clasificación perfecta, 0 representa una clasificación similar al azar, y -1 representa un desacuerdo total entre la predicción y las observaciones [34].

Generalmente los rangos del coeficiente MCC son los siguientes:

- -1.0 a -0.7: Asociación negativa fuerte.
- -0.7 a -0.3: Asociación negativa débil.
- -0.3 a +0.3: Asociación pequeña o sin asociación.
- +0.3 a +0.7: Asociación positiva débil.
- +0.7 a +1.0: Asociación positiva fuerte.

## 5.2. Análisis de datos de salida

Luego de definidos los clasificadores que se utilizarán, el modelo de división de los conjuntos de datos seleccionado y los datos de salida que se analizarán, se debe ejecutar el software WEKA utilizando los parámetros definidos.

En esta sección se analizarán los resultados obtenidos por medio de la ejecución de los clasificadores en nuestras matrices de características.

En la Tabla 5.1 podemos ver el resumen de las matrices que utilizaremos como conjuntos de datos de entrenamiento para los clasificadores. Además se se definen las abreviaciones que se utilizarán en las tablas posteriores.

| **Matriz** | **Abreviación** | **Dimensión** |
|---|---|---|
| Todas las energías | Energías | 296x281 |
| Todas las energías y distancias | Total | 296x841 |
| Todas las distancias más su energía asociada | ED | 296x641 |
| Distancias del enfoque centro geométrico | CG | 296x81 |
| Distancias del enfoque centro de masa total | Cmall | 296x81 |
| Distancias del enfoque centro de masa cadena base | Cmbase | 296x81 |
| Distancias del enfoque centro de masa cadena lateral | Cmlat | 296x81 |
| Distancias del enfoque carbono alfa | CA | 296x81 |
| Distancias del enfoque carbono beta | CB | 296x81 |
| Distancias del enfoque receptor-centro geométrico-ligando | RCL | 296x81 |

Tabla 5.1: Matrices de características.

Para cada clasificador, se creó una tabla en donde se muestran los resultados de su ejecución en cada matriz de características. Las columnas se explican a continuación:

- Columna Matriz: Abreviaciones de las matrices que se utilizaron.
- Columna Correcto ( %): Corresponde al porcentaje de complejos correctamente clasificados.
- Columna TP rate (t): Corresponde al factor de complejos transitorios correctamente clasificados.
- Columna FP rate (t): Corresponde al factor de complejos transitorios incorrectamente clasificados.
- Columna TP rate (o): Corresponde al factor de complejos permanentes correctamente clasificados.
- Columna FP rate (o): Corresponde al factor de complejos permanentes incorrectamente clasificados.
- Columna MCC: Corresponde al Coeficiente de Correlación de Matthews o coeficiente phi.
- Columna AUC (ROC): Corresponde al factor área bajo la curva ROC.

Las columnas “Correcto”, “MCC” y “AUC” contienen colores que ayudan a identificar que atributos y matrices obtienen un valor más alto y más bajo. Verde indica los resultados de cifras más altas, mientras que el color rojo indica lo contrario, es decir, las cifras mas bajas. Los colores son independientes entre una columna y otra.

### 5.2.1. Aplicación del clasificador "Naive Bayes"

En la Tabla 5.2, que corresponde a la aplicación del clasificador 'Naive Bayes', podemos apreciar que los mejores resultados fueron obtenidos por la matriz que contiene sólo energías (matriz Energías) y la matriz de distancias RCL (matriz RCL)[2], con un 73 % y 72 % de complejos correctamente clasificados correctamente, además cuentan con un buen coeficiente AUC[3], superior a 0.7, y un coeficiente MCC[4] superior a 0.3.

Utilizando el clasificador 'Naive Bayesén la matriz de distancia RCL, el 81.4 % de los complejos transitorios fueron clasificados correctamente y el 50.5 % de los complejos permanentes fueron clasificados correctamente.

Debemos destacar que todas las otras matrices con características de distancias (CG, Cmall, Cmbase, Cmlat, CA y CB) obtuvieron en su mayoría resultados menores a 50 % de complejos correctamente clasificados. Estos resultados otorgan una mayor confianza a la matriz con características de distancia RCL con la utilización del clasificador 'Naive Bayes'.

Analizando el coeficiente MCC también podemos observar que los peores resultados fueron obtenidos por todas las matrices de características de distancia exceptuando a la distancia RCL. Estas matrices obtuvieron un coeficiente MCC menor a 0.3, lo que significa una asociación pequeña o sin asociación.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 73 | 0,799 | 0,409 | 0,591 | 0,201 | 0,354 | 0,72 |
| Total | 66,33 | 0,613 | 0,226 | 0,774 | 0,387 | 0,359 | 0,749 |
| ED | 55 | 0,392 | 0,118 | 0,882 | 0,608 | 0,276 | 0,752 |
| CG | 49 | 0,289 | 0,065 | 0,935 | 0,711 | 0,252 | 0,721 |
| Cmall | 49 | 0,294 | 0,065 | 0,935 | 0,706 | 0,256 | 0,72 |
| Cmbase | 49 | 0,304 | 0,086 | 0,914 | 0,696 | 0,238 | 0,71 |
| Cmlat | 58 | 0,456 | 0,161 | 0,839 | 0,544 | 0,284 | 0,735 |
| CA | 49 | 0,304 | 0,097 | 0,903 | 0,696 | 0,225 | 0,711 |
| CB | 51 | 0,314 | 0,075 | 0,925 | 0,686 | 0,259 | 0,715 |
| RCL | 72 | 0,814 | 0,495 | 0,505 | 0,186 | 0,327 | 0,745 |

Tabla 5.2: Aplicación del clasificador 'Naive Bayes'.

[2]Distancia RCL: Distancia Receptor-Centro Geométrico-Ligando, vista en la sección 3.4

[3]Coeficiente AUC: Parámetro entregado por WEKA que representa la probabilidad de que, al clasificar un par de complejos de diferente clase, ambos sean clasificados correctamente. Un coeficiente sobre 0.7 es considerado confiable.

[4]Coeficiente MCC (Matthews Correlation Coefficient): Parámetro entregado por WEKA, que entrega resultados entre -1 y 1, en donde una asociación positiva consiste en valores superiores a 0.3

### 5.2.2. Aplicación del clasificador "Simple Logistic"

En la Tabla 5.3, que corresponde a la aplicación del clasificador 'Simple Logistic', las dos mejores clasificaciones ocurren con las matrices de sólo energías y de todas las características (matriz Energías y Total) con 79 % y 78 % respectivamente. Como podemos observar, las matrices con sólo características energéticas obtuvieron resultados desde un 4 % mejores que las matrices de características de distancia, y al utilizar la matriz con todas las características, no se aumenta la precisión de la clasificación.

Abstrayéndonos de los resultados obtenidos por las matrices de sólo características energéticas, las únicas matrices de distancias con resultados aceptables bajo los criterios definidos en la sección previa (MCC>0.3 y AUC>0.7) fueron la matriz de distancias de centro de masa total y de centro geométrico.

Los peores resultados fueron obtenidos por la matriz de características de distancia de centro de masa de la cadena lateral y de carbono alfa, con un coeficiente MCC de 0.173 y 0.168 respectivamente, lo que significa una asociación débil o sin asociación y un AUC de 0.666 y 0.660 respectivamente, lo que nos dice que existe una asociación pobre.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 79 | 0,922 | 0,505 | 0,495 | 0,078 | 0,475 | 0,785 |
| Total | 78 | 0,926 | 0,527 | 0,473 | 0,074 | 0,464 | 0,781 |
| ED | 75 | 0,897 | 0,57 | 0,43 | 0,103 | 0,376 | 0,771 |
| CG | 74 | 0,882 | 0,57 | 0,43 | 0,118 | 0,352 | 0,713 |
| Cmall | 73 | 0,882 | 0,591 | 0,409 | 0,118 | 0,332 | 0,72 |
| Cmbase | 72 | 0,961 | 0,796 | 0,204 | 0,039 | 0,266 | 0,7 |
| Cmlat | 68 | 0,848 | 0,699 | 0,301 | 0,152 | 0,173 | 0,666 |
| CA | 69 | 0,922 | 0,806 | 0,194 | 0,078 | 0,168 | 0,66 |
| CB | 72 | 0,892 | 0,645 | 0,355 | 0,108 | 0,295 | 0,701 |
| RCL | 70 | 0,868 | 0,656 | 0,344 | 0,132 | 0,246 | 0,71 |

Tabla 5.3: Aplicación del clasificador 'Simple Logistic'

### 5.2.3. Aplicación del clasificador "Sequential Minimal Optimization"

En la Tabla 5.4 se presenta el clasificador 'Sequential Minimal Optimization', el cual aplica en su algoritmo el modelo de soporte de máquina de vectores. Los mejores resultados fueron obtenidos por la matriz de sólo características energéticas, con un 79 % de complejos clasificados correctamente (93.1 % de los complejos transitorios y 51.6 % de los complejos permanentes). Obteniendo resultados desde un 6 % a un 12 % mejores de complejos correctamente clasificados en comparación a las matrices de distancias.

Los peores resultados fueron obtenidos por la matriz de distancias de centro de masa de la cadena base y de carbono alfa, con un 67 % de precisión cada una, un coeficiente MCC de 1.63 y 1.65 respectivamente y un AUC de 0.573 cada una.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 79 | 0,931 | 0,516 | 0,484 | 0,069 | 0,483 | 0,708 |
| Total | 70 | 0,77 | 0,462 | 0,538 | 0,23 | 0,304 | 0,654 |
| ED | 70 | 0,76 | 0,441 | 0,559 | 0,24 | 0,312 | 0,659 |
| CG | 71 | 0,868 | 0,645 | 0,355 | 0,132 | 0,257 | 0,611 |
| Cmall | 71 | 0,873 | 0,645 | 0,355 | 0,127 | 0,264 | 0,614 |
| Cmbase | 67 | 0,824 | 0,677 | 0,323 | 0,176 | 0,163 | 0,573 |
| Cmlat | 71 | 0,843 | 0,57 | 0,43 | 0,157 | 0,296 | 0,637 |
| CA | 67 | 0,833 | 0,688 | 0,312 | 0,167 | 0,165 | 0,573 |
| CB | 69 | 0,838 | 0,645 | 0,355 | 0,162 | 0,215 | 0,597 |
| RCL | 73 | 0,897 | 0,624 | 0,376 | 0,103 | 0,324 | 0,637 |

Tabla 5.4: Aplicación del clasificador 'Sequential Minimal Optimization'.

### 5.2.4. Aplicación del clasificador "Random Commitee"

La Tabla 5.5 muestra la aplicación del clasificador 'Random Commitee', y en ella vemos que los mejores resultados fueron obtenidos por la matriz con sólo características energéticas, la matriz con todas las características.

De las matrices con características de distancias, ninguna matriz cumple con los parámetros establecidos (ROC>0.7 y MCC>0.3). Los peores resultados fueron obtenidos por la matriz con enfoque Carbono Alfa y Carbono Beta, además de la distancia RCL.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 77 | 0,892 | 0,495 | 0,505 | 0,108 | 0,437 | 0,75 |
| Total | 78 | 0,936 | 0,548 | 0,452 | 0,064 | 0,463 | 0,767 |
| ED | 74 | 0,897 | 0,613 | 0,387 | 0,103 | 0,335 | 0,736 |
| CG | 71 | 0,897 | 0,71 | 0,29 | 0,103 | 0,236 | 0,691 |
| Cmall | 73 | 0,917 | 0,677 | 0,323 | 0,083 | 0,304 | 0,677 |
| Cmbase | 71 | 0,917 | 0,753 | 0,247 | 0,083 | 0,223 | 0,643 |
| Cmlat | 70 | 0,912 | 0,753 | 0,247 | 0,088 | 0,214 | 0,675 |
| CA | 67 | 0,873 | 0,785 | 0,215 | 0,127 | 0,112 | 0,61 |
| CB | 70 | 0,882 | 0,71 | 0,29 | 0,118 | 0,212 | 0,67 |
| RCL | 69 | 0,882 | 0,742 | 0,258 | 0,118 | 0,177 | 0,671 |

Tabla 5.5: Aplicación del clasificador 'Random Commitee'.

### 5.2.5. Aplicación del clasificador "Decision Table"

En la Tabla 5.6 se aplicó el clasificador 'Decision Table', en el cual sólo la matriz de energías y la matriz con todas las características cumplen con obtener un buen coeficiente AUC.

Las matrices con características de distancias cuentan con un pobre desempeño en el coeficiente MCC utilizando el clasificador 'Decision Table', con resultados inferiores a 0.250, y coeficiente AUC inferior a 0.650.

El peor resultado fue obtenido por la distancia con enfoque de centro geométrico, con un 67 % de precisión, un coeficiente MCC de 1.76 y un coeficiente AUC de 0.593. Los complejos transitorios clasificados correctamente utilizando este enfoque con 'Decision Table'fue de 83.3 %, y de complejos permanentes fue de 32.3 %, un número muy inferior para una clasificación.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC(ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 74 | 0,892 | 0,581 | 0,419 | 0,108 | 0,358 | 0,758 |
| Total | 75 | 0,892 | 0,548 | 0,452 | 0,108 | 0,388 | 0,77 |
| ED | 73 | 0,887 | 0,624 | 0,376 | 0,113 | 0,308 | 0,688 |
| CG | 67 | 0,833 | 0,677 | 0,323 | 0,167 | 0,176 | 0,593 |
| Cmall | 70 | 0,858 | 0,645 | 0,355 | 0,142 | 0,243 | 0,636 |
| Cmbase | 68 | 0,902 | 0,817 | 0,183 | 0,098 | 0,119 | 0,547 |
| Cmlat | 68 | 0,882 | 0,763 | 0,237 | 0,118 | 0,152 | 0,635 |
| CA | 70 | 0,941 | 0,839 | 0,161 | 0,059 | 0,165 | 0,573 |
| CB | 71 | 0,902 | 0,71 | 0,29 | 0,098 | 0,244 | 0,653 |
| RCL | 68 | 0,887 | 0,763 | 0,237 | 0,113 | 0,16 | 0,647 |

Tabla 5.6: Aplicación del clasificador 'Decision Table'.

### 5.2.6. Aplicación del clasificador "J48"

Aplicando el clasificado J48 como podemos ver en la Tabla 5.7, sólo obtenemos resultados sobre 0.7 del coeficiente AUC y 0.3 del coeficiente phi en la matriz con sólo características energéticas.

Los resultados obtenidos con las matrices de características de distancias fueron pobres en comparación a otros clasificadores ya evaluados. Todos los enfoques de distancias obtuvieron un MCC inferior a 0.200 y un coeficiente AUC inferior a 0.600. Los peores resultados fueron obtenidos por la matriz de distancia con enfoque de centro de masa total, con un coeficiente MCC de 0.082 % y un coeficiente AUC de 0.549 y la matriz de centro de masa de la cadena lateral, con un coeficiente MCC de 0.034 y un coeficiente AUC de 0.514.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 76 | 0,838 | 0,409 | 0,591 | 0,162 | 0,436 | 0,708 |
| Total | 72 | 0,789 | 0,441 | 0,559 | 0,211 | 0,346 | 0,672 |
| ED | 64 | 0,74 | 0,57 | 0,43 | 0,26 | 0,17 | 0,583 |
| CG | 62 | 0,716 | 0,602 | 0,398 | 0,284 | 0,113 | 0,566 |
| Cmall | 60 | 0,706 | 0,624 | 0,376 | 0,294 | 0,082 | 0,549 |
| Cmbase | 62 | 0,74 | 0,645 | 0,355 | 0,26 | 0,097 | 0,537 |
| Cmlat | 62 | 0,804 | 0,774 | 0,226 | 0,196 | 0,034 | 0,514 |
| CA | 64 | 0,716 | 0,538 | 0,462 | 0,284 | 0,174 | 0,591 |
| CB | 62 | 0,725 | 0,602 | 0,398 | 0,275 | 0,123 | 0,554 |
| RCL | 66,33 | 0,789 | 0,613 | 0,387 | 0,211 | 0,185 | 0,597 |

Tabla 5.7: Aplicación del clasificador 'J48'.

### 5.2.7. Aplicación del clasificador "Logistic Model Tree"

En la Tabla 5.8, la cual contiene los resultados de la aplicación del clasificador 'Logistic Model Tree', los mejores resultados fueron obtenidos por las matrices de sólo características energéticas, con un 78 % de complejos correctamente clasificados, un coeficiente MCC de 0.467 y un coeficiente AUC de 0.790.

La matriz con todas las características, al igual que la matriz de sólo energías, obtuvo similar precisión y coeficientes MCC y AUC, lo que nos dice que las características de distancias no influyeron ni positiva ni negativamente en la clasificación.

De las matrices de distancias, los mejores resultados fueron obtenidos por la matriz con enfoque de centro geométrico y de centro de masa total, con una precisión del 74 % y 73 % respectivamente, y ambas con coeficiente MCC superior a 0.3 y coeficiente AUC superior a 0.7.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 78 | 0,917 | 0,505 | 0,495 | 0,083 | 0,467 | 0,79 |
| Total | 78 | 0,917 | 0,516 | 0,484 | 0,083 | 0,457 | 0,765 |
| ED | 75 | 0,897 | 0,57 | 0,43 | 0,103 | 0,376 | 0,771 |
| CG | 74 | 0,882 | 0,57 | 0,43 | 0,118 | 0,352 | 0,713 |
| Cmall | 73 | 0,877 | 0,602 | 0,398 | 0,123 | 0,314 | 0,711 |
| Cmbase | 69 | 0,941 | 0,849 | 0,151 | 0,059 | 0,15 | 0,695 |
| Cmlat | 69 | 0,838 | 0,645 | 0,355 | 0,162 | 0,215 | 0,666 |
| CA | 69 | 0,912 | 0,806 | 0,194 | 0,088 | 0,15 | 0,656 |
| CB | 72 | 0,877 | 0,634 | 0,366 | 0,123 | 0,283 | 0,697 |
| RCL | 70 | 0,868 | 0,656 | 0,344 | 0,132 | 0,246 | 0,71 |

Tabla 5.8: Aplicación del clasificador 'Logistic Model Tree'.

### 5.2.8. Aplicación del clasificador "Random Forest"

En la Tabla 5.9, la cual contiene los resultados del clasificador 'Random Forest', se puede observar que los mejores resultados fueron obtenidos por la matriz de energías, con un porcentaje de complejos correctamente clasificados de 79 % y la matriz con todas las características, con similares resultados. Pero esta última no genera una mejora en la clasificación que se genera con la matriz de sólo características energéticas.

Considerando sólo las matrices de distancias, los mejores resultados fueron obtenidos por la matriz con enfoque de centro e masa total y la matriz con enfoque de centro geométrico, con un 75 % y 74 % de complejos correctamente clasificados respectivamente, y en ambas un coeficiente AUC sobre 0.7 y un coeficiente MCC superior a 0.3.

| Matriz | Correcto (%) | TP rate (t) | FP rate (t) | TP rate (o) | FP rate (o) | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|
| Energias | 79 | 0,922 | 0,495 | 0,505 | 0,078 | 0,484 | 0,792 |
| Total | 79 | 0,931 | 0,516 | 0,484 | 0,069 | 0,483 | 0,795 |
| ED | 76 | 0,912 | 0,57 | 0,43 | 0,088 | 0,4 | 0,754 |
| CG | 74 | 0,936 | 0,677 | 0,323 | 0,064 | 0,341 | 0,717 |
| Cmall | 75 | 0,931 | 0,645 | 0,355 | 0,069 | 0,364 | 0,731 |
| Cmbase | 71 | 0,931 | 0,774 | 0,226 | 0,069 | 0,226 | 0,679 |
| Cmlat | 71 | 0,931 | 0,785 | 0,215 | 0,069 | 0,213 | 0,727 |
| CA | 73 | 0,956 | 0,774 | 0,226 | 0,044 | 0,28 | 0,679 |
| CB | 74 | 0,941 | 0,688 | 0,312 | 0,059 | 0,34 | 0,694 |
| RCL | 73 | 0,892 | 0,624 | 0,376 | 0,108 | 0,219 | 0,608 |

Tabla 5.9: Aplicación del clasificador 'Random Forest'.

### 5.2.9. Análisis de los mejores resultados obtenidos

Luego se analizó cada matriz de características por separado, escogiendo al mejor clasificador de los anteriormente aplicados. En la Tabla 5.10 y la Tabla 5.11 podemos ver el rendimiento de cada clasificador para cada matriz de características. El mejor rendimiento está marcado de color celeste en el clasificador seleccionado.

*Centro Geométrico*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 49 | 0,252 | 0,721 |
| SL | 74 | 0,352 | 0,713 |
| SMO | 71 | 0,257 | 0,611 |
| RC | 71 | 0,236 | 0,691 |
| DT | 67 | 0,176 | 0,593 |
| J48 | 62 | 0,113 | 0,566 |
| LMT | 74 | 0,352 | 0,713 |
| RF | 74 | 0,341 | 0,717 |

*Centro de masa, cadena lateral*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 58 | 0,284 | 0,735 |
| SL | 68 | 0,173 | 0,666 |
| SMO | 71 | 0,296 | 0,637 |
| RC | 70 | 0,214 | 0,675 |
| DT | 68 | 0,152 | 0,635 |
| J48 | 62 | 0,034 | 0,514 |
| LMT | 69 | 0,215 | 0,666 |
| RF | 71 | 0,213 | 0,727 |

*Centro de Masa Total*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 49 | 0,256 | 0,72 |
| SL | 73 | 0,332 | *0,72* |
| SMO | 71 | 0,264 | 0,614 |
| RC | 73 | 0,304 | 0,677 |
| DT | 70 | 0,243 | 0,636 |
| J48 | 60 | 0,082 | 0,549 |
| LMT | 73 | 0,314 | 0,711 |
| RF | 75 | 0,364 | 0,731 |

*Carbono alfa*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 49 | 0,225 | 0,711 |
| SL | 69 | 0,168 | 0,66 |
| SMO | 67 | 0,165 | 0,573 |
| RC | 67 | 0,112 | 0,61 |
| DT | 70 | 0,165 | 0,573 |
| J48 | 64 | 0,174 | 0,591 |
| LMT | 69 | 0,15 | 0,656 |
| RF | 73 | 0,28 | 0,679 |

Tabla 5.10: Rendimiento de clasificadores en la matriz “centro geométrico”, “centro de masa cadena lateral”, “centro de masa total” y “carbono alfa”. NB: ‘Naive Bayes’, SL: ‘Simple Logistic’, SMO: ‘Sequencial Minimal Optimization’, RC: ‘Random Commitee’, DT: ‘Decision Table’, J48: ‘J48’, LMT: ‘Logical Model Tree’, RF: ‘Random Forest’.

*Centro de masa, cadena base*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 49 | 0,238 | 0,71 |
| SL | 72 | 0,266 | 0,7 |
| SMO | 67 | 0,163 | 0,573 |
| RC | 71 | 0,223 | 0,643 |
| DT | 68 | 0,119 | 0,547 |
| J48 | 62 | 0,097 | 0,537 |
| LMT | 69 | 0,15 | 0,695 |
| RF | 71 | 0,226 | 0,679 |

*Carbono beta*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 51 | 0,259 | 0,715 |
| SL | 72 | 0,295 | 0,701 |
| SMO | 69 | 0,215 | 0,597 |
| RC | 70 | 0,212 | 0,67 |
| DT | 71 | 0,244 | 0,653 |
| J48 | 62 | 0,123 | 0,554 |
| LMT | 72 | 0,283 | 0,697 |
| RF | 74 | 0,34 | 0,694 |

*Receptor-centro geometrico-ligando*

| classificador | Correcto (%) | MCC | AUC(ROC) |
|---|---|---|---|
| NB | 72 | 0,327 | 0,745 |
| SL | 70 | 0,246 | 0,71 |
| SMO | 73 | 0,324 | 0,637 |
| RC | 69 | 0,177 | 0,671 |
| DT | 68 | 0,16 | 0,647 |
| J48 | 66,33 | 0,185 | 0,597 |
| LMT | 70 | 0,246 | 0,71 |
| RF | 73 | 0,219 | 0,608 |

Tabla 5.11: Rendimiento de clasificadores en la matriz “centro de masa cadena lateral”, “carbono beta” y “receptor-centro geométrico-ligando”. NB: ‘Naive Bayes’, SL: ‘Simple Logistic’, SMO: ‘Sequencial Minimal Optimization’, RC: ‘Random Commitee’, DT: ‘Decision Table’, J48: ‘J48’, LMT: ‘Logical Model Tree’, RF: ‘Random Forest’.

La Tabla 5.12 contiene el clasificador que mejor rendimiento mostró para cada matriz con características de distancias. Como podemos apreciar, los mejores resultados los obtuvieron la distancia desde el centro geométrico utilizando el clasificador Simple Logistic, y la distancia desde el centro de masa total utilizando el clasificador Random Forest.

Viendo el porcentaje de los complejos correctamente clasificados en cada clase, podemos observar que utilizando el enfoque de centro geométrico obtuvimos un 88.2 % de complejos transitorios correctamente clasificados y un 43 % para los complejos permanentes. En cambio, utilizando el centro de masa total, obtuvimos un 93.1 % de complejos transitorios correctamente clasificados y un 35.5 % de complejos permanentes correctamente clasificados.

Por otra parte, la matriz que mostró un desempeño más equitativo con respecto a los complejos correctamente clasificados, fue la matriz con distancia Receptor-Centro Geométrico-Ligando con el clasificador ‘Naive Bayes’con un 81.4 % de complejos transitorios correctamente clasificados y un 50.5 % de complejos permanentes correctamente clasificados. Además, utilizando esta matriz, se obtuvo el mejor resultado del coeficiente AUC con un 0.745, lo que significa que si tenemos un complejo permanente y un complejo transitorio, existe un 74.5 % de probabilidad de que ambos sean clasificados correctamente.

| Matriz | Clasificador | Correcto (%) | TP (t) | TP (o) | MCC | AUC(ROC) |
|---|---|---|---|---|---|---|
| CG | SL | 74 | 0,882 | 0,43 | 0,352 | 0,713 |
| CmTotal | RF | 75 | 0,931 | 0,355 | 0,364 | 0,731 |
| CmBase | SL | 72 | 0,961 | 0,204 | 0,266 | 0,7 |
| CmLat | SMO | 71 | 0,843 | 0,43 | 0,296 | 0,637 |
| CA | RF | 73 | 0,956 | 0,226 | 0,28 | 0,679 |
| CB | RF | 74 | 0,941 | 0,312 | 0,34 | 0,694 |
| RCL | NB | 72 | 0,814 | 0,505 | 0,327 | 0,745 |

Tabla 5.12: Clasificador más preciso en cada matriz con características de distancia.

En el Gráfico 5.1 podemos visualizar lo que antes se mencionó. En donde los peores resultados fueron obtenidos por el centro de masa de la cadena base y la cadena lateral, junto con el enfoque de carbono alfa.

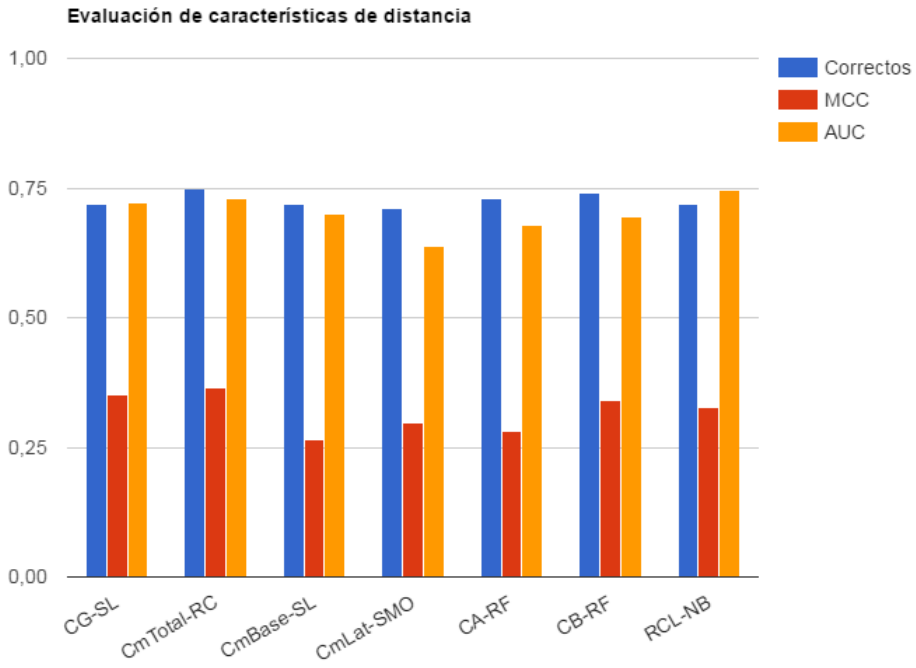


Figura 5.1: Gráfico con los mejores resultados de la evaluación de características de distancias.

Como pudimos ver en este capitulo, el software de aprendizaje automático WEKA, cuenta con variadas herramientas para el manejo y análisis de conjuntos de datos, una basta cantidad de clasificadores y parámetros resultantes de la aplicación de estos, de los cuales se utilizaron los que se consideraban pertinentes para nuestro problema. Luego con estos parámetros se realizó un análisis estadístico para determinar que matrices de enfoques de distancia fueron las que obtuvieron mejores resultados y con que clasificadores se obtuvieron estos.

# Capítulo 6

# Conclusiones

Las interacciones proteína-proteína como vimos, son de vital importancia en una gran cantidad de procesos biológicos, como la biosíntesis y traducción de señales. Estas interacciones pueden ser duraderas en el tiempo (IPP permanentes) o de corta duración (IPP transitorias). El conocer el tipo de interacción ha sido uno de los grandes desafíos de la bioinformática, por las diferencias funcionales que tienen los distintos tipos de interacción. Vimos diferentes métodos para analizar el tipo de interacción que se produce entre un par de proteínas, tanto la cristalografía de rayos X y la espectrometría de resonancia magnética nuclear, son métodos muy costosos y consumidores de tiempo. Estos métodos han generado una gran cantidad de datos sobre la estructura tridimensional de las interacciones proteína-proteína que la bioinformática ha utilizado con herramientas de minado de datos para predecir estas interacciones, pero aún así, no existe un método 100 % efectivo que realice esta tarea.

Por este motivo estudiamos un enfoque distinto que podría ayudar a generar nuevas características que permitan crear métodos más efectivos. Este enfoque consiste en determinar si la distancia entre los aminoácidos que pertenecen a la zona de interacción de proteínas, influyen en el tipo de interacción que se produce. Para esto se obtuvieron los datos datos tridimensionales de complejos transitorios y permanentes ya clasificados correctamente. Cada aminoácido está conformado por diferentes elementos químicos, y cada elemento químico posee una coordenada en el espacio.

Por este motivo se utilizaron distintos tipos de enfoque para determinar un punto en el espacio para cada aminoácido perteneciente a la zona de interacción. Estos enfoques fueron el utilizar el centro geométrico de los aminoácidos. El centro de masa, considerando la masa atómica y

coordenadas de los elementos químicos en cada aminoácido. El centro de masa de la cadena lateral de los aminoácidos. El centro de masa de la cadena base de los aminoácidos. Sólo utilizar la coordenada del carbono alfa y por último, sólo utilizar el carbono beta de un aminoácido.

Luego de determinar que punto en el espacio utilizaríamos en cada aminoácido, se analizaron diferentes métodos de medición de distancias, de los cuales se utilizaron la distancia euclidiana entre dos puntos, y además se utilizó una variación de esta, la distancia euclidiana entre dos puntos pero pasando por el centro geométrico del complejo proteico. Luego se crearon diez matrices de características realizando una combinación enfoques y distancias para luego ser evaluados en el programa de aprendizaje automático WEKA. Los resultados obtenidos en WEKA fueron estudiados utilizando distintos indicadores, entre ellos el porcentaje de complejos correctamente clasificados, el coeficiente MCC y el coeficiente AUC ROC (área bajo la curva de la característica operativa del receptor).

Los enfoques de distancia que obtuvieron mejores resultados fueron el enfoque de centro geométrico aplicando el clasificador 'Simple Logistic', con el cual se consiguió un 74 % de complejos correctamente clasificados, un coeficiente MCC de 0.352 y un coeficiente AUC de 0,713. También el enfoque de centro de masa aplicando el clasificador 'Random Forest', con el cual se consiguió un 75 % de complejos correctamente clasificados, un coeficiente MCC de 0.355 y un coeficiente AUC de 0.731. El enfoque que consiguió un porcentaje un poco menor de complejos correctamente clasificados fue el enfoque Receptor-centro geométrico-ligando utilizando el clasificador utilizando el clasificador 'Naive Bayes'con un 72 %, pero a diferencia de los demás, obtuvo un mejor mejor clasificación de complejos permanentes con un 50.5 %, además obtuvo coeficiente MCC de 0.327 y un coeficiente AUC de 0.745.

Tomando en cuenta que si el clasificador aplicado a una matriz de características consigue un coeficiente MCC entre 0.3 y 0.5, indica que existe una asociación positiva pero débil, y que si se consigue un coeficiente AUC entre 0.7 y 0.9, nos indica que la asociación es positiva pero moderada, podemos determinar que si existe evidencia de que la distancia entre aminoácidos interactuantes en la zona de interacción de dos proteínas influye en el tipo de interacción que se produce entre ambas, pero no es un factor 100 % determinante.

Además pudimos observar que las energías contribuidas por las interacciones entre aminoácidos generan una mejor clasificación de complejos proteicos en comparación a utilizar las distancias entre aminoácidos. Pero al utilizar estas energías en conjunto con los enfoques de distancias, el porcentaje de clasificación disminuyó, por tanto entendemos que estos no funcionaron bien juntos en una misma medición.

Cómo mejorar o qué incorporar como trabajos futuros, se podría considerar otros tipos de mediciones de distancias, como por ejemplo la distancia Manhatthan, la cual no se utilizó en este estudio por requerir un sistema de coordenadas con una orientación bien definida, la cual no es generada por *Protein Data Bank*. También, esta la opción de profundizar en la distancia residuo-centro geométrico-ligando, y utilizar el centro geométrico del complejo proteico completo,

o como otra alternativa, está el cambiar el centro geométrico por el centro de masa, tanto de la zona de interacción como del complejo proteico completo.

# Referencias


[1] Particule-Sciences, "Technical brief 2009 volume 8: Protein structure." `http://www.particlesciences.com/docs/technical_briefs/TB_8.pdf`, 2009. [Acceso: 2016-09-30].

[2] F. Goebels, *Classification of protein protein interactions*. PhD thesis, Universitätsbibliothek der TU München, 2014.

[3] P. D. B. in Europe, "Pdbepisa." `http://www.ebi.ac.uk/msd-srv/prot_int/cgi-bin/piserver`. [Acceso: 2016-10-30].

[4] B. D. Davison, "Gary m. weiss, ph. d., department of computer and information science, fordham university brian d. davison, ph. d., department of computer science and engineering, lehigh university,"

[5] P. P. Moreno, D. de la Fuente García, F. J. P. García, and R. P. Díez, "Utilización del aprendizaje inductivo en la toma de decisiones. aplicación en un problema de secuenciación," *Investigaciones europeas de dirección y economía de la empresa*, vol. 10, no. 3, pp. 17–36, 2004.

[6] C. Suplee, "Atomic weights and isotopic compositions with relative atomic masses," 2009.

[7] I. M. Nooren and J. M. Thornton, "Diversity of protein–protein interactions," *The EMBO journal*, vol. 22, no. 14, pp. 3486–3492, 2003.

[8] T. Gutiérrez-Bunster, "Estudio de características energéticas en zonas de interacción proteína-proteína, para identificación de interacciones transitorias y permanentes," 2008.

[9] M. V. Olson, "The human genome project.," *Proceedings of the National Academy of Sciences*, vol. 90, no. 10, pp. 4338–4344, 1993.

[10] N. M. Luscombe, D. Greenbaum, M. Gerstein, *et al.*, "What is bioinformatics? an introduction and overview," *Yearbook of Medical Informatics*, vol. 1, no. 83-100, p. 2, 2001.

[11] E. W. Sayers, T. Barrett, D. A. Benson, E. Bolton, S. H. Bryant, K. Canese, V. Chetvernin, D. M. Church, M. DiCuccio, S. Federhen, *et al.*, "Database resources of the national center for biotechnology information," *Nucleic acids research*, vol. 39, no. suppl 1, pp. D38–D51, 2011.

[12] R. Bischoff and H. Schlüter, "Amino acids: chemistry, functionality and selected non-enzymatic post-translational modifications," *Journal of proteomics*, vol. 75, no. 8, pp. 2275–2296, 2012.

[13] S. Damodaran, *Amino acids, peptides, and proteins*, vol. 4. CRC Press: Boca Raton, FL, 2008.

[14] N. H. Andersen, "Protein structure, stability, and folding. methods in molecular biology. volume 168 edited by kenneth p. murphy (university of iowa college of medicine)," 2001.

[15] M. V. L. Guillén, *Estructura y propiedades de las proteínas*. 2009.

[16] D. Voet, J. G. Voet, and C. W. Pratt, *Fundamentals of biochemistry*. Wiley New York, 1999.

[17] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic Acids Research*, vol. 28, no. 1, p. 235, 2000.

[18] wwPDB, "Atomic coordinate entry format description." `http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html`. [Acceso: 2016-09-30].

[19] E. M. Phizicky and S. Fields, "Protein-protein interactions: methods for detection and analysis.," *Microbiological reviews*, vol. 59, no. 1, pp. 94–123, 1995.

[20] O. Keskin, A. Gursoy, B. Ma, and R. Nussinov, "Principles of protein- protein interactions: What are the preferred ways for proteins to interact?," *Chemical reviews*, vol. 108, no. 4, pp. 1225–1244, 2008.

[21] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, p. 37, 1996.

[22] H. Bidgoli, *The Handbook of Technology Management, Supply Chain Management, Marketing and Advertising, and Global Management*, vol. 2. John Wiley & Sons, 2010.

[23] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[24] H. Yoganarasimhan, "Search personalization using machine learning," *Browser Download This Paper*, 2016.

[25] F. S. Gharehchopogh and Y. Lotfi, "Machine learning based question classification methods in the question answering systems," *Int J Innovat Appl Stud*, vol. 4, no. 2, 2013.

[26] .AI-Horizon", ""machine learning, part i: Supervised and unsupervised learning"." `http://www.aihorizon.com/essays/generalai/supervised_unsupervised_machine_learning.htm`. [Acceso: 2016-11-30].

[27] T. O. Ayodele, *Types of machine learning algorithms*. INTECH Open Access Publisher, 2010.

[28] M. J and W. Z., "Structure, function, and evolution of transient and obligate protein–protein interactions." `https://zlab.bu.edu/julianm/MintserisWengPNAS05.html`, 2005.

[29] C. CJ and Z. C., "Fastcontact: rapid estimate of contact and binding free energies," 2005.

[30] N. Bourbaki, "Topological vector spaces, chap. 1 to 5, elements of mathematics," 1987.

[31] E. F. Krause, *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 2012.

[32] S. R. Garner *et al.*, "Weka: The waikato environment for knowledge analysis," in *Proceedings of the New Zealand computer science research students conference*, pp. 57–64, Citeseer, 1995.

[33] T. G. Tape, "The area under an roc curve." `http://gim.unmc.edu/dxtests/roc3.htm`. [Acceso: 2017-01-05].

[34] S. Simon, "What is a phi coefficient? retrieved november 2, 2010," 2005.

[35] E. De Hoffmann and V. Stroobant, *Mass spectrometry: principles and applications*. John Wiley & Sons, 2007.

[36] B. Alberts, D. Bray, K. Hopkin, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, *Essential cell biology*. Garland Science, 2013.

[37] R. Gillaspy, "Primary functions of protein in the body," 2010.

[38] Trudy and J. R. McKee, *Biochemistry: the molecular basis of life*. Oxford University Press Oxford, 2012.

# Apéndice A

# Glosario

**In vivo:** En ciencia se utiliza el término in vivo para llamar a un experimento realizado dentro de un organismo.

**In vitro:** A diferencia de los experimentos in vivo, los experimentos in vitro se realizan en un tubo de ensayo.

**Espectrometría de masas :** es una técnica de análisis que identificar y cuantificar las moléculas de una sustancia en función de su masa y su carga[35].

**Péptido:** Los péptidos son un tipo de moléculas formadas por la unión de varios aminoácidos mediante enlaces peptídicos.

**Cromoforo:** Un cromóforo es la parte o conjunto de átomos de una molécula responsable de su color.

**Biosíntesis:** La biosíntesis de proteínas o síntesis de proteínas es el proceso anabólico mediante el cual se forman las proteínas.

**Energía de Desolvatación :** La solvatación es el proceso de asociación de moléculas de un disolvente con moléculas o iones de un soluto. Al disolverse los iones en un soluto, se dispersan y son rodeados por moléculas de solvente.

**Oligómeros:** Los oligómeros son proteínas que están compuestas de mas de una cadena polipeptídica, es decir, poseen estructura cuaternaria.

**Enlace covalente:** Es un enlace químico que une dos átomos por medio de compartición de electrones con el fin de buscar el octeto ideal [36].

**Enlace péptido:** Es un tipo de enlace covalente, pero entre dos aminoácidos.

# Apéndice B

# Algoritmos

## B.1. Algoritmo para el cálculo de las coordenadas de centro geométrico

```
1 from os import walk
2
3 #Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada
      que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la
      coordenada X, Y o Z del centro geométrico del aminoácido, en otras palabras, el promedio
      de las coordenadas.
4 def Promedio(lista,elem):
5     result=0
6     for x in lista:
7         result=result+x[elem]
8     return round(result/len(lista),3)
9
10 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
11 for (path, ficheros, archivos) in walk("./PDB"):
12     if len(archivos)>1:
```

```
#Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo de coordenadas.
        archivoA = open(path+"/fort.19", "r")
        archivoA2= open(path+"/cg.19", "w+")
        archivoB = open(path+"/fort.20", "r")
        archivoB2= open(path+"/cg.20", "w+")
        archivopdbA=archivoA.readlines()
        archivopdbB=archivoB.readlines()
#La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente continuo).
        residuo=1
#X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas del aminoácido que está siendo evaluado.
        X=[]
#Bucle que recorre las lineas del archivo de la proteína A.
        for linesA in archivopdbA:
            if(linesA[0:4]=="ATOM"):
#La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se ignora.
                if(linesA[30]!='*'):
                    residuoPDB=int(linesA[22:26])
#Condición que permite saber que los elementos químicos que guardaremos en la lista X son del mismo aminoácido.
                    if(residuo==residuoPDB):
                        nombreRes=linesA[17:20]
#Guarda la información del elemento químico evaluado en la variable X.
                        X.append([float(linesA[30:38]), float(linesA[38:46]), float(linesA[46:54]), linesA[12:16]])
#Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior, evaluar la lista X en la función enfCA.
                    else:
                        archivoA2.write("ATOM                "+nombreRes+"   {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(residuo, Promedio(X,0), Promedio(X,1), Promedio(X,2)))
                        del X[:]
#Cambiar el aminoácido a evaluar.
                        residuo+=1
```

```
41 #Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
42                             X.append([float(linesA[30:38]), float(↩
                                   linesA[38:46]), float(linesA[46:54]), ↩
                                   linesA[12:16]])
43 #escribe la información del último aminoácido.
44                 else:
45                     archivoA2.write("ATOM              "+nombreRes+"  ↩
                           {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(↩
                           residuo, Promedio(X,0), Promedio(X,1), Promedio(↩
                           X,2)))
46 #Se resetean las variables para la proteína B.
47             residuo=1
48             del X[:]
49 #Se repite el mismo proceso pero esta vez para la proteína B.
50             for linesB in archivopdbB:
51                 if(linesB[0:4]=="ATOM"):
52                     if(linesB[30]!='*'):
53
54                         residuoPDB=int(linesB[22:26])
55                         if(residuo==residuoPDB):
56                             nombreResB=linesB[17:20]
57                             X.append([float(linesB[30:38]), float(↩
                                   linesB[38:46]), float(linesB[46:54]), ↩
                                   linesB[12:16]])
58                         else:
59                             archivoB2.write("ATOM              "+↩
                                   nombreResB+"  {0:4d}     {1:8.3f}{2:8.3f↩
                                   }{3:8.3f}\n".format(residuo, Promedio(X↩
                                   ,0), Promedio(X,1), Promedio(X,2)))
60                             del X[:]
61                             residuo+=1
62                             X.append([float(linesB[30:38]), float(↩
                                   linesB[38:46]), float(linesB[46:54]), ↩
                                   linesB[12:16]])
63                 else:
64                     archivoB2.write("ATOM              "+nombreResB+"  ↩
                           {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(↩
                           residuo, Promedio(X,0), Promedio(X,1), Promedio(↩
                           X,2)))
65
66             archivoA.close()
67             archivoB.close()
```

```
68          archivoA2.close()
69          archivoB2.close()
```

## B.2. Algoritmo para el cálculo de las coordenadas de carbono alfa

```
1 from os import walk
2
3 #Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada
      que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la
      coordenada X, Y o Z del elemento ÇA".
4 def enfCA(lista,elem):
5     for x in lista:
6         if x[3][1:3]=='CA':
7             return x[elem]
8     return False
9
10 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
11 for (path, ficheros, archivos) in walk("./PDB"):
12     if len(archivos)>1:
13 #Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en
      la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo
      de coordenadas.
14          archivoA = open(path+"/fort.19", "r")
15          archivoA2= open(path+"/CA.19", "w+")
16          archivoB = open(path+"/fort.20", "r")
17          archivoB2= open(path+"/CA.20", "w+")
18          archivopdbA=archivoA.readlines()
19          archivopdbB=archivoB.readlines()
20 #La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que
      este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente
      continuo).
21          residuo=1
22 #X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas
      del aminoácido que está siendo evaluado.
23          X=[]
24 #Bucle que recorre las lineas del archivo de la proteína A.
25          for linesA in archivopdbA:
```

```
26                 if(linesA[0:4]=="ATOM"):
27 #La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un
       asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se
       ignora.
28                     if(linesA[30]!='*'):
29                         residuoPDB=int(linesA[22:26])
30 #Condición que permite saber que los elementos químicos que guardaremos en la lista X son
       del mismo aminoácido
31                         if(residuo==residuoPDB):
32                             nombreRes=linesA[17:20]
33 #Guarda la información del elemento químico evaluado en la variable X.
34                             X.append([float(linesA[30:38]), float(↩
                                   linesA[38:46]), float(linesA[46:54]), ↩
                                   linesA[12:16]])
35 #Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior,
       evaluar la lista X en la función enfCA.
36                         else:
37                             promX=enfCA(X,0)
38                             promY=enfCA(X,1)
39                             promZ=enfCA(X,2)
40 #Escribir en el nuevo archivo de la proteína A los resultados obtenidos por la función enfCA.
41                             if promX==False or promY==False or promZ==↩
                                   False:
42                                 archivoA2.write("****\n")
43                             else:
44                                 archivoA2.write("ATOM                "+↩
                                       nombreRes+"  {0:4d}     {1:8.3f↩
                                       }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                        promX, promY, promZ))
45                             del X[:]
46 #Cambiar el aminoácido a evaluar.
47                             residuo+=1
48 #Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
49                             nombreRes=linesA[17:20]
50                             X.append([float(linesA[30:38]), float(↩
                                   linesA[38:46]), float(linesA[46:54]), ↩
                                   linesA[12:16]])
51
52 #Evalua el último aminoácido.
53                 else:
54                     promX=enfCA(X,0)
55                     promY=enfCA(X,1)
```

```
56                  promZ=enfCA(X,2)
57
58                  if promX==False or promY==False or promZ==False:
59                      archivoA2.write("****\n")
60                  else:
61                      archivoA2.write("ATOM              "+nombreRes+"↩
                            {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                           format(residuo, promX, promY, promZ))
62
63 #Se resetean las variables para la proteína B.
64          residuo=1
65          del X[:]
66
67 #Se repite el mismo proceso pero esta vez para la proteína B.
68          for linesB in archivopdbB:
69              if(linesB[0:4]=="ATOM"):
70                  if(linesB[30]!='*'):
71
72                      residuoPDB=int(linesB[22:26])
73                      if(residuo==residuoPDB):
74                          nombreResB=linesB[17:20]
75                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
76                      else:
77                          promX=enfCA(X,0)
78                          promY=enfCA(X,1)
79                          promZ=enfCA(X,2)
80
81                          if promX==False or promY==False or promZ==↩
                                False:
82                              archivoB2.write("****\n")
83                          else:
84                              archivoB2.write("ATOM              "+↩
                                    nombreResB+"  {0:4d}     {1:8.3f↩
                                    }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                     promX, promY, promZ))
85                          del X[:]
86                          residuo+=1
87                          nombreResB=linesB[17:20]
88                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
```

```
                                linesB[12:16]])
89              else:
90                  promX=enfCA(X,0)
91                  promY=enfCA(X,1)
92                  promZ=enfCA(X,2)
93
94                  if promX==False or promY==False or promZ==False:
95                      archivoB2.write("****\n")
96                  else:
97                      archivoB2.write("ATOM              "+nombreResB+↩
                            "  {0:4d}    {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                            format(residuo, promX, promY, promZ))
98          archivoA.close()
99          archivoB.close()
100         archivoA2.close()
101         archivoB2.close()
```

## B.3. Algoritmo para el cálculo de las coordenadas de carbono beta

```
1 from os import walk
2
3 #Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada
      que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la
      coordenada X, Y o Z del elemento ÇB". Si no existe coordenada o elemento ÇBün cierto
      aminoácido, se utiliza el elemento CA.
4 def enfCB(lista,elem):
5     result=-1000
6     for x in lista:
7         if x[3][1:3]=='CB':
8             return x[elem]
9         if x[3][1:3]=='CA':
10            result=x[elem]
11
12    if result==-1000:
13        return False
14    else:
15        return result
```

```
16
17 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
18 for (path, ficheros, archivos) in walk("./PDB"):
19     if len(archivos)>1:
20 #Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en
       la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo
       de coordenadas.
21         archivoA = open(path+"/fort.19", "r")
22         archivoA2= open(path+"/CB.19", "w+")
23         archivoB = open(path+"/fort.20", "r")
24         archivoB2= open(path+"/CB.20", "w+")
25         archivopdbA=archivoA.readlines()
26         archivopdbB=archivoB.readlines()
27 #La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que
       este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente
       continuo).
28         residuo=1
29 #X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas
       del aminoácido que está siendo evaluado.
30         X=[]
31 #Bucle que recorre las lineas del archivo de la proteína A.
32         for linesA in archivopdbA:
33             if(linesA[0:4]=="ATOM"):
34 #La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un
       asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se
       ignora.
35                 if(linesA[30]!='*'):
36                     residuoPDB=int(linesA[22:26])
37 #Condición que permite saber que los elementos químicos que guardaremos en la lista X son
       del mismo aminoácido
38                     if(residuo==residuoPDB):
39                         nombreRes=linesA[17:20]
40 #Guarda la información del elemento químico evaluado en la variable X.
41                         X.append([float(linesA[30:38]), float(↩
                               linesA[38:46]), float(linesA[46:54]), ↩
                               linesA[12:16]])
42 #Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior,
       evaluar la lista X en la función enfCA.
43                     else:
44                         promX=enfCB(X,0)
45                         promY=enfCB(X,1)
46                         promZ=enfCB(X,2)
```

```
#Escribir en el nuevo archivo de la proteína A los resultados obtenidos por la función enfCA.
                            if promX==False or promY==False or promZ==False:
                                archivoA2.write("****\n")
                            else:
                                archivoA2.write("ATOM                "+nombreRes+"  {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(residuo, promX, promY, promZ))
                            del X[:]
#Cambiar el aminoácido a evaluar.
                            residuo+=1
#Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
                            nombreRes=linesA[17:20]
                            X.append([float(linesA[30:38]), float(linesA[38:46]), float(linesA[46:54]), linesA[12:16]])
#Evalua el último aminoácido.
                else:
                    promX=enfCB(X,0)
                    promY=enfCB(X,1)
                    promZ=enfCB(X,2)

                    if promX==False or promY==False or promZ==False:
                        archivoA2.write("****\n")
                    else:
                        archivoA2.write("ATOM                "+nombreRes+"  {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(residuo, promX, promY, promZ))
#Se resetean las variables para la proteína B.
            residuo=1
            del X[:]

#Se repite el mismo proceso pero esta vez para la proteína B.
            for linesB in archivopdbB:
                if(linesB[0:4]=="ATOM"):
                    if(linesB[30]!='*'):

                        residuoPDB=int(linesB[22:26])
                        if(residuo==residuoPDB):
                            nombreResB=linesB[17:20]
                            X.append([float(linesB[30:38]), float(
```

```
                        linesB[38:46]), float(linesB[46:54]), ↩
                        linesB[12:16]])
81                  else:
82                      promX=enfCB(X,0)
83                      promY=enfCB(X,1)
84                      promZ=enfCB(X,2)
85
86                      if promX==False or promY==False or promZ==↩
                        False:
87                          archivoB2.write("****\n")
88                      else:
89                          archivoB2.write("ATOM              "+↩
                            nombreResB+"  {0:4d}     {1:8.3f↩
                            }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                             promX, promY, promZ))
90                      del X[:]
91                      residuo+=1
92                      nombreResB=linesB[17:20]
93                      X.append([float(linesB[30:38]), float(↩
                        linesB[38:46]), float(linesB[46:54]), ↩
                        linesB[12:16]])
94              else:
95                  promX=enfCB(X,0)
96                  promY=enfCB(X,1)
97                  promZ=enfCB(X,2)
98
99                  if promX==False or promY==False or promZ==False:
100                     archivoB2.write("****\n")
101                 else:
102                     archivoB2.write("ATOM              "+nombreResB+↩
                        "  {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                        format(residuo, promX, promY, promZ))
103         archivoA.close()
104         archivoB.close()
105         archivoA2.close()
106         archivoB2.close()
```

## B.4. Algoritmo para el cálculo de las coordenadas de centro de masa total

```
from os import walk

#Diccionario Python en donde la letra de los elementos químicos representan la llave para sus valores de su masa atómica.
CenterMass={"N":14.007, "C":12.011, "H":1.008, "O":15.999, "S":32.06}

#Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la coordenada X, Y o Z del centro de masa del aminoácido.
def enfCmAll(lista,elem):
    masa=0
    mxv=0
    for x in lista:
        mxv=mxv+x[elem]*CenterMass[x[3][1]]
        masa=masa+CenterMass[x[3][1]]
    return round(mxv/masa,3)

#Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
for (path, ficheros, archivos) in walk("./PDB"):
    if len(archivos)>1:
#Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo de coordenadas.
        archivoA = open(path+"/fort.19", "r")
        archivoA2= open(path+"/cmAll.19", "w+")
        archivoB = open(path+"/fort.20", "r")
        archivoB2= open(path+"/cmAll.20", "w+")
        archivopdbA=archivoA.readlines()
        archivopdbB=archivoB.readlines()
#La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente continuo).
        residuo=1
#X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas del aminoácido que está siendo evaluado.
```

```
28          X=[]
29 #Bucle que recorre las lineas del archivo de la proteína A.
30          for linesA in archivopdbA:
31              if(linesA[0:4]=="ATOM"):
32 #La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un
       asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se
       ignora.
33                  if(linesA[30]!='*'):
34                      residuoPDB=int(linesA[22:26])
35 #Condición que permite saber que los elementos químicos que guardaremos en la lista X son
       del mismo aminoácido.
36                      if(residuo==residuoPDB):
37                          nombreRes=linesA[17:20]
38 #Guarda la información del elemento químico evaluado en la variable X.
39                          X.append([float(linesA[30:38]), float(↩
                               linesA[38:46]), float(linesA[46:54]), ↩
                               linesA[12:16]])
40 #Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior,
       evaluar la lista X en la función enfCA.
41                      else:
42                          archivoA2.write("ATOM                "+↩
                               nombreRes+"  {0:4d}     {1:8.3f}{2:8.3f↩
                               }{3:8.3f}\n".format(residuo, enfCmAll(X↩
                               ,0), enfCmAll(X,1), enfCmAll(X,2)))
43                          del X[:]
44 #Cambiar el aminoácido a evaluar.
45                          residuo+=1
46 #Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
47                          X.append([float(linesA[30:38]), float(↩
                               linesA[38:46]), float(linesA[46:54]), ↩
                               linesA[12:16]])
48 #escribe la información del último aminoácido.
49              else:
50                  archivoA2.write("ATOM                "+nombreRes+"  ↩
                       {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(↩
                       residuo, enfCmAll(X,0), enfCmAll(X,1), enfCmAll(↩
                       X,2)))
51 #Se resetean las variables para la proteína B.
52          residuo=1
53          del X[:]
54 #Se repite el mismo proceso pero esta vez para la proteína B.
55          for linesB in archivopdbB:
```

```
56              if(linesB[0:4]=="ATOM"):
57                  if(linesB[30]!='*'):
58
59                      residuoPDB=int(linesB[22:26])
60                      if(residuo==residuoPDB):
61                          nombreResB=linesB[17:20]
62                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
63                      else:
64                          archivoB2.write("ATOM              "+↩
                                nombreResB+"  {0:4d}     {1:8.3f}{2:8.3f↩
                                }{3:8.3f}\n".format(residuo, enfCmAll(X↩
                                ,0), enfCmAll(X,1), enfCmAll(X,2)))
65                          del X[:]
66                          residuo+=1
67                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
68              else:
69                  archivoB2.write("ATOM              "+nombreResB+"  ↩
                        {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".format(↩
                        residuo, enfCmAll(X,0), enfCmAll(X,1), enfCmAll(↩
                        X,2)))
70
71          archivoA.close()
72          archivoB.close()
73          archivoA2.close()
74          archivoB2.close()
```

## B.5. Algoritmo para el cálculo de las coordenadas de centro de masa de la cadena base

```
1 from os import walk
2
3 #Diccionario Python en donde la letra de los elementos químicos representan la llave para sus
      valores de su masa atómica.
4 CenterMass={"N":14.007, "C":12.011, "H":1.008, "O":15.999, "S"↩
```

```
    :32.06}
5
6 #Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada
      que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la
      coordenada X, Y o Z del centro de masa de la cadena base del aminoácido (solo los
      elementos 'C', 'CA', Ó', Ñ'y 'CB').
7 def enfCmBase(lista,elem):
8     masa=0
9     mxv=0
10    for x in lista:
11        if x[3][1:3]=='C ' or x[3][1:3]=='CA' or x[3][1:3]=='O ' or↩
              x[3][1:3]=='N ' or x[3][1:3]=='CB':
12            mxv=mxv+x[elem]*CenterMass[x[3][1]]
13            masa=masa+CenterMass[x[3][1]]
14    if masa==0:
15        return False
16    else:
17        return round(mxv/masa,3)
18
19 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
20 for (path, ficheros, archivos) in walk("./PDB"):
21    if len(archivos)>1:
22 #Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en
      la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo
      de coordenadas.
23        archivoA = open(path+"/fort.19", "r")
24        archivoA2= open(path+"/cmBase.19", "w+")
25        archivoB = open(path+"/fort.20", "r")
26        archivoB2= open(path+"/cmBase.20", "w+")
27        archivopdbA=archivoA.readlines()
28        archivopdbB=archivoB.readlines()
29 #La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que
      este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente
      continuo).
30        residuo=1
31 #X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas
      del aminoácido que está siendo evaluado.
32        X=[]
33 #Bucle que recorre las lineas del archivo de la proteína A.
34        for linesA in archivopdbA:
35            if(linesA[0:4]=="ATOM"):
36 #La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un
```

asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se ignora.

```
37                      if(linesA[30]!='*'):
38                          residuoPDB=int(linesA[22:26])
39 #Condición que permite saber que los elementos químicos que guardaremos en la lista X son
       del mismo aminoácido.
40                          if(residuo==residuoPDB):
41                              nombreRes=linesA[17:20]
42 #Guarda la información del elemento químico evaluado en la variable X.
43                              X.append([float(linesA[30:38]), float(↩
                                    linesA[38:46]), float(linesA[46:54]), ↩
                                    linesA[12:16]])
44 #Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior,
       evaluar la lista X en la función enfCA.
45                              else:
46                                  promX=enfCmBase(X,0)
47                                  promY=enfCmBase(X,1)
48                                  promZ=enfCmBase(X,2)
49 #Escribir en el nuevo archivo de la proteína A los resultados obtenidos por la función enfCA.
50                                  if promX==False or promY==False or promZ==↩
                                        False:
51                                      archivoA2.write("****\n")
52                                  else:
53                                      archivoA2.write("ATOM                "+↩
                                            nombreRes+"  {0:4d}     {1:8.3f↩
                                            }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                             promX, promY, promZ))
54                                  del X[:]
55 #Cambiar el aminoácido a evaluar.
56                                  residuo+=1
57 #Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
58                                  X.append([float(linesA[30:38]), float(↩
                                        linesA[38:46]), float(linesA[46:54]), ↩
                                        linesA[12:16]])
59 #Evalua el último aminoácido.
60                  else:
61                      promX=enfCmBase(X,0)
62                      promY=enfCmBase(X,1)
63                      promZ=enfCmBase(X,2)
64
65                      if promX==False or promY==False or promZ==False:
66                          archivoA2.write("****\n")
```

```
67                  else:
68                      archivoA2.write("ATOM                "+nombreRes+"↩
                            {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                           format(residuo, promX, promY, promZ))
69 #Se resetean las variables para la proteína B.
70          residuo=1
71          del X[:]
72 #Se repite el mismo proceso pero esta vez para la proteína B.
73          for linesB in archivopdbB:
74              if(linesB[0:4]=="ATOM"):
75                  if(linesB[30]!='*'):
76
77                      residuoPDB=int(linesB[22:26])
78                      if(residuo==residuoPDB):
79                          nombreResB=linesB[17:20]
80                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
81                      else:
82                          promX=enfCmBase(X,0)
83                          promY=enfCmBase(X,1)
84                          promZ=enfCmBase(X,2)
85
86                          if promX==False or promY==False or promZ==↩
                                False:
87                              archivoB2.write("****\n")
88                          else:
89                              archivoB2.write("ATOM                "+↩
                                    nombreResB+"  {0:4d}     {1:8.3f↩
                                    }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                     promX, promY, promZ))
90                          del X[:]
91                          residuo+=1
92                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
93              else:
94                  promX=enfCmBase(X,0)
95                  promY=enfCmBase(X,1)
96                  promZ=enfCmBase(X,2)
97
98                  if promX==False or promY==False or promZ==False:
```

```
99                      archivoB2.write("****\n")
100                 else:
101                     archivoB2.write("ATOM              "+nombreResB+↩
                            "  {0:4d}    {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                            format(residuo, promX, promY, promZ))
102         archivoA.close()
103         archivoB.close()
104         archivoA2.close()
105         archivoB2.close()
```

## B.6. Algoritmo para el cálculo de las coordenadas de centro de masa de la cadena lateral

```
1 from os import walk
2
3 #Diccionario Python en donde la letra de los elementos químicos representan la llave para sus
      valores de su masa atómica.
4 CenterMass={"N":14.007, "C":12.011, "H":1.008, "O":15.999, "S"↩
      :32.06}
5
6 #Función que recibe una lista de los elementos químicos con sus coordenadas y la coordenada
      que se esta evaluando (0: Coordenada X, 1: Coordenada Y, 2: Coordenada Z) y retorna la
      coordenada X, Y o Z del centro de masa de la cadena lateral del aminoácido (elementos
      diferentes a 'C', 'CA', Ó', Ñ'y 'CB').
7 def enfCmLat(lista,elem):
8     masa=0
9     mxv=0
10    for x in lista:
11        if x[3][1:3]!='C ' or x[3][1:3]!='CA' or x[3][1:3]!='O ' or↩
              x[3][1:3]!='N ' or x[3][1:3]!='CB':
12            mxv=mxv+x[elem]*CenterMass[x[3][1]]
13            masa=masa+CenterMass[x[3][1]]
14    if masa==0:
15        return False
16    else:
17        return round(mxv/masa,3)
18
19 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
```

```
20 for (path, ficheros, archivos) in walk("./PDB"):
21     if len(archivos)>1:
22 #Dentro de la carpeta de un complejo, abre los archivos fort.19 y fort.20, y crea dos archivos en
       la misma carpeta, los que contendrán el resultado de la aplicación del enfoque de manejo
       de coordenadas.
23         archivoA = open(path+"/fort.19", "r")
24         archivoA2= open(path+"/cmRes.19", "w+")
25         archivoB = open(path+"/fort.20", "r")
26         archivoB2= open(path+"/cmLat.20", "w+")
27         archivopdbA=archivoA.readlines()
28         archivopdbB=archivoB.readlines()
29 #La variable residuo tiene como objetivo conocer el identificador de cada aminoácido, ya que
       este identificador va de 1 hasta el identificador del último aminoácido (en orden ascendente
       continuo).
30         residuo=1
31 #X es una lista que contendrá los nombres de los elementos químicos junto con sus coordenadas
       del aminoácido que está siendo evaluado.
32         X=[]
33 #Bucle que recorre las lineas del archivo de la proteína A.
34         for linesA in archivopdbA:
35             if(linesA[0:4]=="ATOM"):
36 #La linea 30 corresponde al comienzo de las coordenadas del elemento químico, si contiene un
       asterisco significa que no se tiene información sobre dicha coordenada, por lo tanto se
       ignora.
37                 if(linesA[30]!='*'):
38                     residuoPDB=int(linesA[22:26])
39 #Condición que permite saber que los elementos químicos que guardaremos en la lista X son
       del mismo aminoácido.
40                     if(residuo==residuoPDB):
41                         nombreRes=linesA[17:20]
42 #Guarda la información del elemento químico evaluado en la variable X.
43                         X.append([float(linesA[30:38]), float(↩
                               linesA[38:46]), float(linesA[46:54]), ↩
                               linesA[12:16]])
44 #Si la linea que se está leyendo no corresponde a un elemento químico del aminoácido anterior,
       evaluar la lista X en la función enfCA.
45                     else:
46                         promX=enfCmLat(X,0)
47                         promY=enfCmLat(X,1)
48                         promZ=enfCmLat(X,2)
49 #Escribir en el nuevo archivo de la proteína A los resultados obtenidos por la función enfCA.
50                         if promX==False or promY==False or promZ==↩
```

```
                                False:
51                                  archivoA2.write("****\n")
52                              else:
53                                  archivoA2.write("ATOM                "+↩
                                       nombreRes+"  {0:4d}     {1:8.3f↩
                                       }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                        promX, promY, promZ))
54                              del X[:]
55 #Cambiar el aminoácido a evaluar.
56                              residuo+=1
57 #Guardar los datos del primer elemento químico del aminoácido siguiente al ya evaluado.
58                              X.append([float(linesA[30:38]), float(↩
                                   linesA[38:46]), float(linesA[46:54]), ↩
                                   linesA[12:16]])
59 #Evalua el último aminoácido.
60              else:
61                  promX=enfCmLat(X,0)
62                  promY=enfCmLat(X,1)
63                  promZ=enfCmLat(X,2)
64
65                  if promX==False or promY==False or promZ==False:
66                      archivoA2.write("****\n")
67                  else:
68                      archivoA2.write("ATOM              "+nombreRes+"↩
                            {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                           format(residuo, promX, promY, promZ))
69 #Se resetean las variables para la proteína B.
70          residuo=1
71          del X[:]
72 #Se repite el mismo proceso pero esta vez para la proteína B.
73          for linesB in archivopdbB:
74              if(linesB[0:4]=="ATOM"):
75                  if(linesB[30]!='*'):
76
77                      residuoPDB=int(linesB[22:26])
78                      if(residuo==residuoPDB):
79                          nombreResB=linesB[17:20]
80                          X.append([float(linesB[30:38]), float(↩
                                linesB[38:46]), float(linesB[46:54]), ↩
                                linesB[12:16]])
81                      else:
82                          promX=enfCmLat(X,0)
```

```
83                          promY=enfCmLat(X,1)
84                          promZ=enfCmLat(X,2)
85
86                          if promX==False or promY==False or promZ==↩
                               False:
87                              archivoB2.write("****\n")
88                          else:
89                              archivoB2.write("ATOM              "+↩
                                   nombreResB+"  {0:4d}     {1:8.3f↩
                                   }{2:8.3f}{3:8.3f}\n".format(residuo,↩
                                    promX, promY, promZ))
90                          del X[:]
91                          residuo+=1
92                          X.append([float(linesB[30:38]), float(↩
                               linesB[38:46]), float(linesB[46:54]), ↩
                               linesB[12:16]])
93              else:
94                  promX=enfCmLat(X,0)
95                  promY=enfCmLat(X,1)
96                  promZ=enfCmLat(X,2)
97
98                  if promX==False or promY==False or promZ==False:
99                      archivoB2.write("****\n")
100                 else:
101                     archivoB2.write("ATOM              "+nombreResB+↩
                            "  {0:4d}     {1:8.3f}{2:8.3f}{3:8.3f}\n".↩
                            format(residuo, promX, promY, promZ))
102         archivoA.close()
103         archivoB.close()
104         archivoA2.close()
105         archivoB2.close()
```

## B.7. Algoritmo para el cálculo de distancias con coordenada de centro geométrico

```
1 # -*- coding: utf-8 -*-
2 import numpy
3 from os import walk
```

```
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos cg.19 y cg.20, los que representan el
      enfoque de manejo de coordenadas del centro geométrico en cada proteína, y crea dos
      archivos en la misma carpeta, los que contendrán toda la información del archivo
      FastContact para cada proteína, pero agregado la distancia entre los residuos desde la
      coordenada de centro geométrico.
10             archivoFC= open(path+"/"+archivo, "r+")
11             archivoFCM=open(path+"/fcGeoCenter_"+archivo, "w+")
12             archivoA = open(path+"/cg.19", "r")
13             archivoB = open(path+"/cg.20", "r")
14             archfastcont=archivoFC.readlines()
15             largoFC=len(archfastcont)-2
16             archivoA.seek(0,0)
17             archivoB.seek(0,0)
18 #Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
19             for i in range(largoFC):
20                 if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
21                     indEnergia=i+1
22                     break
23
24 #Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final
      del archivo.
25             for i in range(indEnergia,largoFC):
26 #Condicióque ignora los textos que no contienen datos de la interaccion
27                 if archfastcont[i]!=" ---------------------------------\n"↩
                       and archfastcont[i]!=" Top 20 Min & Max ↩
                      receptor-ligand residue free energy contacts\n":
28 #Guarda el identificador de cada aminoácido en dos variables.
29                     residuoA=int(archfastcont[i][9:13])
30                     residuoB=int(archfastcont[i][19:22])
31 #variable booleana, si estado continua siendo True a la hora de calcular la distancia, significa
      que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no
      calculamos la distancia.
32                     estado=True
33
34 #Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína
      A hasta encontrar la ubicación del aminoácido requerido.
```

```
35                          for j, lineA in enumerate(archivoA):
36                              if j==residuoA-1:
37 #FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos
       encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería
       False.
38                                  if lineA[0]!="*":
39 #Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de
       manejo de operaciones numéricas "numPY".
40                                      coorA=numpy.array((float(lineA↩
                                            [30:38]),float(lineA[38:46]),↩
                                            float(lineA[46:54])))
41                                  else:
42                                      estado=False
43                                  archivoA.seek(0,0)
44                                  break
45
46 #Se repite el proceso con la proteína B.
47                          for k, lineB in enumerate(archivoB):
48                              if k==residuoB-1:
49                                  if lineB[0]!="*":
50                                      coorB=numpy.array((float(lineB↩
                                            [30:38]),float(lineB[38:46]),↩
                                            float(lineB[46:54])))
51                                  else:
52                                      estado=False
53                                  archivoB.seek(0,0)
54                                  break
55 #Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
       utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
56                          if estado==True:
57                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    {0:8.3f}".format(float(numpy.linalg.norm↩
                                    (coorA-coorB)))+"\n"
58                          else:
59                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    ********\n"
60                          archfastcont[i]=distancia
61
62                  archivoB.seek(0,0)
63                  archivoA.seek(0,0)
64                  archivoFC.close()
65                  archivoFCM.write("  Energy      AA-A      AA-B        GC ↩
```

```
                Complex: "+path[-4:]+"\n")
66              archivoFCM.writelines(archfastcont[indEnergia-1:-1])
67              archivoFCM.close()
68              archivoA.close()
69              archivoB.close()
```

## B.8. Algoritmo para el cálculo de distancias con coordenada de carbono alfa

```
1 # -*- coding: utf-8 -*-
2 import numpy
3 from os import walk
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos CA.19 y CA.20, los que representan el
      enfoque de manejo de coordenadas del Carbono Alfa en cada proteína, y crea dos archivos
      en la misma carpeta, los que contendrán toda la información del archivo FastContact para
      cada proteína, pero agregado la distancia entre los residuos desde la coordenada de
      Carbono Alfa.
10              archivoFC= open(path+"/"+archivo, "r+")
11              archivoFCM=open(path+"/fcCA_"+archivo, "w+")
12              archivoA = open(path+"/CA.19", "r")
13              archivoB = open(path+"/CA.20", "r")
14              archfastcont=archivoFC.readlines()
15              largoFC=len(archfastcont)-2
16              archivoA.seek(0,0)
17              archivoB.seek(0,0)
18
19 #Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
20              for i in range(largoFC):
21                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
22                      indEnergia=i+1
23                      break
24
```

```
#Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final del archivo.
            for i in range(indEnergia,largoFC):
#Condicióque ignora los textos que no contienen datos de la interaccion
                if archfastcont[i]!=" ----------\n" and archfastcont[i]!=" Top 20 Min & Max receptor-ligand residue free energy contacts\n":
#Guarda el identificador de cada aminoácido en dos variables.
                    residuoA=int(archfastcont[i][9:13])
                    residuoB=int(archfastcont[i][19:22])
#variable booleana, si .estadoçontinua siendo "True.a la hora de calcular la distancia, significa que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no calculamos la distancia.
                    estado=True

#Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína A hasta encontrar la ubicación del aminoácido requerido.
                    for j, lineA in enumerate(archivoA):
                        if j==residuoA-1:
#FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería False.
                            if lineA[0]!="*":
#Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de manejo de operaciones numéricas numPY.
                                coorA=numpy.array((float(lineA[30:38]),float(lineA[38:46]),float(lineA[46:54])))
                            else:
                                estado=False
                            archivoA.seek(0,0)
                            break

#Se repite el proceso con la proteína B.
                    for k, lineB in enumerate(archivoB):
                        if k==residuoB-1:
                            if lineB[0]!="*":
                                coorB=numpy.array((float(lineB[30:38]),float(lineB[38:46]),float(lineB[46:54])))
                            else:
                                estado=False
```

```
54                                  archivoB.seek(0,0)
55                                  break
56 #Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
       utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
57                          if estado==True:
58                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    {0:8.3f}".format(float(numpy.linalg.norm↩
                                    (coorA-coorB)))+"\n"
59                          else:
60                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    ********\n"
61                          archfastcont[i]=distancia
62                  archivoB.seek(0,0)
63                  archivoA.seek(0,0)
64                  archivoFC.close()
65                  archivoFCM.write("  Energy      AA-A      AA-B      CA ↩
                        Complex: "+path[-4:]+"\n")
66                  archivoFCM.writelines(archfastcont[indEnergia-1:-1])
67                  archivoFCM.close()
68                  archivoA.close()
69                  archivoB.close()
```

## B.9. Algoritmo para el cálculo de distancias con coordenada de carbono beta

```
1 # -*- coding: utf-8 -*-
2 import numpy
3 from os import walk
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos CB.19 y CB.20, los que representan el
      enfoque de manejo de coordenadas del Carbono Beta en cada proteína, y crea dos archivos
      en la misma carpeta, los que contendrán toda la información del archivo FastContact para
      cada proteína, pero agregado la distancia entre los residuos desde la coordenada de
      Carbono Beta.
```

```
10              archivoFC= open(path+"/"+archivo, "r+")
11              archivoFCM=open(path+"/fcCB_"+archivo, "w+")
12              archivoA = open(path+"/CB.19", "r")
13              archivoB = open(path+"/CB.20", "r")
14              archfastcont=archivoFC.readlines()
15              largoFC=len(archfastcont)-2
16              archivoA.seek(0,0)
17              archivoB.seek(0,0)
18
19 #Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
20              for i in range(largoFC):
21                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
22                      indEnergia=i+1
23                      break
24
25 #Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final
       del archivo.
26              for i in range(indEnergia,largoFC):
27 #Condicióque ignora los textos que no contienen datos de la interaccion
28                  if archfastcont[i]!=" ----------\n"↩
                        and archfastcont[i]!=" Top 20 Min & Max ↩
                       receptor-ligand residue free energy contacts\n":
29 #Guarda el identificador de cada aminoácido en dos variables.
30                      residuoA=int(archfastcont[i][9:13])
31                      residuoB=int(archfastcont[i][19:22])
32 #variable booleana, si .estadoçontinua siendo "True.a la hora de calcular la distancia, significa
       que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no
       calculamos la distancia.
33                      estado=True
34
35 #Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína
       A hasta encontrar la ubicación del aminoácido requerido.
36                      for j, lineA in enumerate(archivoA):
37                          if j==residuoA-1:
38 #FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos
       encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería
       False.
39                              if lineA[0]!="*":
40 #Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de
       manejo de operaciones numéricas numPY.
41                                  coorA=numpy.array((float(lineA↩
```

```
                                    [30:38]),float(lineA[38:46]),↩
                                    float(lineA[46:54])))
42                                else:
43                                    estado=False
44                                archivoA.seek(0,0)
45                                break
46
47 #Se repite el proceso con la proteína B.
48                            for k, lineB in enumerate(archivoB):
49                                if k==residuoB-1:
50                                    if lineB[0]!="*":
51                                        coorB=numpy.array((float(lineB↩
                                            [30:38]),float(lineB[38:46]),↩
                                            float(lineB[46:54])))
52                                    else:
53                                        estado=False
54                                    archivoB.seek(0,0)
55                                    break
56 #Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
       utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
57                            if estado==True:
58                                distancia=archfastcont[i].rstrip('\n')+"↩
                                    {0:8.3f}".format(float(numpy.linalg.norm↩
                                    (coorA-coorB)))+"\n"
59                            else:
60                                distancia=archfastcont[i].rstrip('\n')+"↩
                                    ********\n"
61                            archfastcont[i]=distancia
62                    archivoA.seek(0,0)
63                    archivoB.seek(0,0)
64                    archivoFC.close()
65                    archivoFCM.write("  Energy      AA-A      AA-B      CB ↩
                          Complex: "+path[-4:]+"\n")
66                    archivoFCM.writelines(archfastcont[indEnergia-1:-1])
67                    archivoFCM.close()
68                    archivoA.close()
69                    archivoB.close()
```

## B.10. Algoritmo para el cálculo de distancias con coordenada de centro de masa total

```
# -*- coding: utf-8 -*-
import numpy
from os import walk

#Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
for (path, ficheros, archivos) in walk("./PDB"):
    for archivo in archivos:
        if archivo==(path[-4:]+".txt"):
#Dentro de la carpeta de un complejo, abre los archivos cmAll.19 y cmAll.20, los que representan el enfoque de manejo de coordenadas del centro de masa en cada proteína, y crea dos archivos en la misma carpeta, los que contendrán toda la información del archivo FastContact para cada proteína, pero agregado la distancia entre los residuos desde la coordenada de centro de masa.
            archivoFC= open(path+"/"+archivo, "r+")
            archivoFCM=open(path+"/fcCmAll_"+archivo, "w+")
            archivoA = open(path+"/cmAll.19", "r")
            archivoB = open(path+"/cmAll.20", "r")
            archfastcont=archivoFC.readlines()
            largoFC=len(archfastcont)-2
            archivoA.seek(0,0)
            archivoB.seek(0,0)

#Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
            for i in range(largoFC):
                if archfastcont[i]==" Top 20 Min & Max receptor-↩
                   ligand residue electrostatic contacts\n":
                    indEnergia=i+1
                    break

#Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final del archivo.
            for i in range(indEnergia,largoFC):
#Condicióque ignora los textos que no contienen datos de la interaccion
                if archfastcont[i]!=" ------------------------------\n"↩
                    and archfastcont[i]!=" Top 20 Min & Max ↩
                   receptor-ligand residue free energy contacts\n":
```

```
#Guarda el identificador de cada aminoácido en dos variables.
                    residuoA=int(archfastcont[i][9:13])
                    residuoB=int(archfastcont[i][19:22])
#variable booleana, si estado continua siendo True a la hora de calcular la distancia, significa
    que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no
    calculamos la distancia.
                    estado=True

#Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína
    A hasta encontrar la ubicación del aminoácido requerido.
                    for j, lineA in enumerate(archivoA):
                        if j==residuoA-1:
#FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos
    encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería
    False.
                            if lineA[0]!="*":
#Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de
    manejo de operaciones numéricas numPY.
                                coorA=numpy.array((float(lineA[30:38]),float(lineA[38:46]),float(lineA[46:54])))
                            else:
                                estado=False
                            archivoA.seek(0,0)
                            break

#Se repite el proceso con la proteína B.
                    for k, lineB in enumerate(archivoB):
                        if k==residuoB-1:
                            if lineB[0]!="*":
                                coorB=numpy.array((float(lineB[30:38]),float(lineB[38:46]),float(lineB[46:54])))
                            else:
                                estado=False
                            archivoB.seek(0,0)
                            break
#Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
    utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
                    if estado==True:
                        distancia=archfastcont[i].rstrip('\n')+"{0:8.3f}".format(float(numpy.linalg.norm
```

```
                        (coorA-coorB)))+"\n"
59                  else:
60                      distancia=archfastcont[i].rstrip('\n')+"↩
                            ********\n"
61                  archfastcont[i]=distancia
62
63          archivoB.seek(0,0)
64          archivoA.seek(0,0)
65          archivoFC.close()
66          archivoFCM.write("  Energy      AA-A      AA-B      CmAll ↩
                Complex: "+path[-4:]+"\n")
67          archivoFCM.writelines(archfastcont[indEnergia-1:-1])
68          archivoFCM.close()
69          archivoA.close()
70          archivoB.close()
```

## B.11. Algoritmo para el cálculo de distancias con coordenada de centro de masa de la cadena base

```
1 # -*- coding: utf-8 -*-
2 import numpy
3 from os import walk
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos cmBase.19 y cmBase.20, los que
      representan el enfoque de manejo de coordenadas del centro de masa de la cadena base en
      cada proteína, y crea dos archivos en la misma carpeta, los que contendrán toda la
      información del archivo FastContact para cada proteína, pero agregado la distancia entre
      los residuos desde la coordenada de centro de masa de la cadena base.
10          archivoFC= open(path+"/"+archivo, "r+")
11          archivoFCM=open(path+"/fcCmBase_"+archivo, "w+")
12          archivoA = open(path+"/cmBase.19", "r")
13          archivoB = open(path+"/cmBase.20", "r")
14          archfastcont=archivoFC.readlines()
15          largoFC=len(archfastcont)-2
```

```
16              archivoA.seek(0,0)
17              archivoB.seek(0,0)
18
19 #Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
20              for i in range(largoFC):
21                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
22                      indEnergia=i+1
23                      break
24
25 #Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final
       del archivo.
26              for i in range(indEnergia,largoFC):
27 #Condicióque ignora los textos que no contienen datos de la interaccion
28                  if archfastcont[i]!=" ---------------------------------\n"↩
                        and archfastcont[i]!=" Top 20 Min & Max ↩
                       receptor-ligand residue free energy contacts\n":
29 #Guarda el identificador de cada aminoácido en dos variables.
30                      residuoA=int(archfastcont[i][9:13])
31                      residuoB=int(archfastcont[i][19:22])
32 #variable booleana, si .estadoçontinua siendo "True.a la hora de calcular la distancia, significa
       que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no
       calculamos la distancia.
33                      estado=True
34
35 #Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína
       A hasta encontrar la ubicación del aminoácido requerido.
36                      for j, lineA in enumerate(archivoA):
37                          if j==residuoA-1:
38 #FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos
       encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería
       False.
39                              if lineA[0]!="*":
40 #Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de
       manejo de operaciones numéricas numPY.
41                                  coorA=numpy.array((float(lineA↩
                                       [30:38]),float(lineA[38:46]),↩
                                       float(lineA[46:54])))
42                              else:
43                                  estado=False
44                              archivoA.seek(0,0)
45                              break
```

```
46
47 #Se repite el proceso con la proteína B.
48                         for k, lineB in enumerate(archivoB):
49                             if k==residuoB-1:
50                                 if lineB[0]!="*":
51                                     coorB=numpy.array((float(lineB↩
                                           [30:38]),float(lineB[38:46]),↩
                                           float(lineB[46:54])))
52                                 else:
53                                     estado=False
54                                 archivoB.seek(0,0)
55                                 break
56 #Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
       utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
57                         if estado==True:
58                             distancia=archfastcont[i].rstrip('\n')+"↩
                                   {0:8.3f}".format(float(numpy.linalg.norm↩
                                   (coorA-coorB)))+"\n"
59                         else:
60                             distancia=archfastcont[i].rstrip('\n')+"↩
                                   ********\n"
61                         archfastcont[i]=distancia
62
63                 archivoB.seek(0,0)
64                 archivoA.seek(0,0)
65                 archivoFC.close()
66                 archivoFCM.write("  Energy      AA-A      AA-B    CmBase ↩
                       Complex: "+path[-4:]+"\n")
67                 archivoFCM.writelines(archfastcont[indEnergia-1:-1])
68                 archivoFCM.close()
69                 archivoA.close()
70                 archivoB.close()
```

## B.12. Algoritmo para el cálculo de distancias con coordenada de centro de masa de la cadena lateral

```
1 # -*- coding: utf-8 -*-
2 import numpy
```

```
3 from os import walk
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos cmLat.19 y cmLat.20, los que
      representan el enfoque de manejo de coordenadas del centro de masa de la cadena lateral
      en cada proteína, y crea dos archivos en la misma carpeta, los que contendrán toda la
      información del archivo FastContact para cada proteína, pero agregado la distancia entre
      los residuos desde la coordenada de centro de masa de la cadena lateral.
10              archivoFC= open(path+"/"+archivo, "r+")
11              archivoFCM=open(path+"/fcCmLat_"+archivo, "w+")
12              archivoA = open(path+"/cmLat.19", "r")
13              archivoB = open(path+"/cmLat.20", "r")
14              archfastcont=archivoFC.readlines()
15              largoFC=len(archfastcont)-2
16              archivoA.seek(0,0)
17              archivoB.seek(0,0)
18
19 #Busca el comienzo de los datos de los pares de aminoácidos que interactuan.
20              for i in range(largoFC):
21                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
22                      indEnergia=i+1
23                      break
24
25 #Recorre el archivo FastContact desde el indice encontrado en el bucle anterior, hasta el final
      del archivo.
26              for i in range(indEnergia,largoFC):
27 #Condicióque ignora los textos que no contienen datos de la interaccion
28                  if archfastcont[i]!=" ------------------------------\n"↩
                        and archfastcont[i]!=" Top 20 Min & Max ↩
                       receptor-ligand residue free energy contacts\n":
29 #Guarda el identificador de cada aminoácido en dos variables.
30                      residuoA=int(archfastcont[i][9:13])
31                      residuoB=int(archfastcont[i][19:22])
32 #variable booleana, si estado continua siendo True a la hora de calcular la distancia, significa
      que tenemos las coordenadas de los dos aminoácidos para logralo, en caso contrario no
      calculamos la distancia.
33                      estado=True
34
```

```
35 #Recorre las lineas del archivo que contiene las coordenadas de cada aminoácido de la proteína
       A hasta encontrar la ubicación del aminoácido requerido.
36                          for j, lineA in enumerate(archivoA):
37                              if j==residuoA-1:
38 #FastContact maneja la falta de coordenadas con los caracteres ***, por lo tanto si nos
       encontramos con este caracter, no podremos calcular la distancia, por lo tanto estado sería
       False.
39                                  if lineA[0]!="*":
40 #Las coordenadas del aminoácido de la proteína A se guardan en un array de la biblioteca de
       manejo de operaciones numéricas numPY.
41                                      coorA=numpy.array((float(lineA↩
                                            [30:38]),float(lineA[38:46]),↩
                                            float(lineA[46:54])))
42                                  else:
43                                      estado=False
44                                  archivoA.seek(0,0)
45                                  break
46
47 #Se repite el proceso con la proteína B.
48                          for k, lineB in enumerate(archivoB):
49                              if k==residuoB-1:
50                                  if lineB[0]!="*":
51                                      coorB=numpy.array((float(lineB↩
                                            [30:38]),float(lineB[38:46]),↩
                                            float(lineB[46:54])))
52                                  else:
53                                      estado=False
54                                  archivoB.seek(0,0)
55                                  break
56 #Si estado es igual a True, aplicamos la distancia euclidiana entre los dos aminoácidos
       utilizando la funcion para calcular la distancia que trae la biblioteca numPY.
57                          if estado==True:
58                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    {0:8.3f}".format(float(numpy.linalg.norm↩
                                    (coorA-coorB)))+"\n"
59                          else:
60                              distancia=archfastcont[i].rstrip('\n')+"↩
                                    ********\n"
61                          archfastcont[i]=distancia
62
63                  archivoB.seek(0,0)
64                  archivoA.seek(0,0)
```

```
65              archivoFC.close()
66              archivoFCM.write("  Energy      AA–A      AA–B      CmLat ↩
                    Complex: "+path[-4:]+"\n")
67              archivoFCM.writelines(archfastcont[indEnergia-1:-1])
68              archivoFCM.close()
69              archivoA.close()
70              archivoB.close()
```

## B.13. Algoritmo para el cálculo de distancia RCL con coordenada de centro geométrico

```
1 # -*- coding: utf-8 -*-
2 import numpy
3 from os import walk
4
5 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
6 for (path, ficheros, archivos) in walk("./PDB"):
7     for archivo in archivos:
8         if archivo==(path[-4:]+".txt"):
9 #Dentro de la carpeta de un complejo, abre los archivos cg.19 y cg.20, los que representan el
      enfoque de manejo de coordenadas del centro geométrico en cada proteína, y crea dos
      archivos en la misma carpeta, los que contendrán toda la información del archivo
      FastContact para cada proteína, pero agregado la distancia entre los residuos desde la
      coordenada de centro geométrico pero utilizando la coordenada de centro geométrico de la
      interacción como punto obligatorio entre cada par de aminoácidos.
10              archivoFC= open(path+"/"+archivo, "r+")
11              archivoFCM=open(path+"/fcRCL_"+archivo, "w+")
12              archivoA = open(path+"/cg.19", "r")
13              archivoB = open(path+"/cg.20", "r")
14              archfastcont=archivoFC.readlines()
15              largoFC=len(archfastcont)-2
16              largoA=len(archivoA.readlines())
17              largoB=len(archivoB.readlines())
18              sumaCoordX=0
19              sumaCoordY=0
20              sumaCoordZ=0
21              countSuma=0
22              residuosL=[]
```

```
23              residuosR=[]
24              archivoA.seek(0,0)
25              archivoB.seek(0,0)
26
27 #Como necesitamos el centro geométrico de la interacción, itilizaremos todos los aminoácidos
       no repetidos presentes en el FastContact de la interacción
28
29 #Bucles que nos permiten encontrar el comienzo de las categorias en donde se encuentran los
       aminoácidos
30              for i in range(largoFC):
31                  if archfastcont[i]==" Top 20 Min & Max ligand ↩
                       residues contributing to the desolvation free ↩
                       energy\n":
32                      indL=i+1
33                      break
34
35              for i in range(largoFC):
36                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
37                      indRL=i+1
38                      break
39
40              for i in range(largoFC):
41                  if archfastcont[i]==" Top 20 Min & Max receptor ↩
                       residues contributing to the desolvation free ↩
                       energy\n":
42                      indR=i+1
43                      break
44
45 #Bucle que guarda las coordenadas de todos los aminoácidos no repetidos de la proteína A.
       sumaCoordX guarda las coordenadas del eje X, sumaCoordY las coordenadas del eje Y, y
       sumaCoordZ las coordenadas del eje Z. countSuma aumenta a medida que se ingresan
       datos a las variables que guardan las coordenadas.
46              for i in range(indL,indR-1):
47                  if archfastcont[i]!=" ------------------\n" and ↩
                       archfastcont[i]!=" Top 20 Min & Max ligand ↩
                       residues contributing to the electrostatics ↩
                       energy\n":
48                      residuoL=int(archfastcont[i][9:13])
49
50                      if residuoL not in residuosL:
51                          residuosL.append(residuoL)
```

```
52                          for j, lineA in enumerate(archivoB):
53                              if j==residuoL-1:
54                                  sumaCoordX=sumaCoordX+float(lineA↩
                                        [30:38])
55                                  sumaCoordY=sumaCoordY+float(lineA↩
                                        [38:46])
56                                  sumaCoordZ=sumaCoordZ+float(lineA↩
                                        [46:54])
57                                  countSuma=countSuma+1
58                                  archivoB.seek(0,0)
59                                  break
60
61
62 #Bucle que guarda las coordenadas de todos los aminoácidos no repetidos de la proteína B.
63              for i in range(indR,indRL-1):
64                  if archfastcont[i]!=" ----------\n" and ↩
                       archfastcont[i]!=" Top 20 Min & Max receptor ↩
                       residues contributing to the electrostatics ↩
                       energy\n":
65                      residuoR=int(archfastcont[i][9:13])
66
67                      if residuoR not in residuosR:
68                          residuosR.append(residuoR)
69
70                          for j, lineA in enumerate(archivoA):
71                              if j==residuoR-1:
72                                  sumaCoordX=sumaCoordX+float(lineA↩
                                        [30:38])
73                                  sumaCoordY=sumaCoordY+float(lineA↩
                                        [38:46])
74                                  sumaCoordZ=sumaCoordZ+float(lineA↩
                                        [46:54])
75                                  countSuma=countSuma+1
76                                  archivoA.seek(0,0)
77                                  break
78
79 #Bucle que guarda las coordenadas de todos los aminoácidos no repetidos de la interacción de
       las proteínas A y B.
80              for i in range(indRL,largoFC):
81                  if archfastcont[i]!=" ----------\n"↩
                        and archfastcont[i]!=" Top 20 Min & Max ↩
                       receptor-ligand residue free energy contacts\n":
```

```
82                      residuoR=int(archfastcont[i][9:13])
83                      residuoL=int(archfastcont[i][19:22])
84
85                      if residuoR not in residuosR:
86                          residuosR.append(residuoR)
87                          for j, lineA in enumerate(archivoA):
88                              if j==residuoR-1:
89                                  sumaCoordX=sumaCoordX+float(lineA↩
                                        [30:38])
90                                  sumaCoordY=sumaCoordY+float(lineA↩
                                        [38:46])
91                                  sumaCoordZ=sumaCoordZ+float(lineA↩
                                        [46:54])
92                                  countSuma=countSuma+1
93                                  archivoA.seek(0,0)
94                                  break
95
96                      if residuoL not in residuosL:
97                          residuosL.append(residuoL)
98                          for k, lineB in enumerate(archivoB):
99                              if k==residuoL-1:
100                                 sumaCoordX=sumaCoordX+float(lineA↩
                                        [30:38])
101                                 sumaCoordY=sumaCoordY+float(lineA↩
                                        [38:46])
102                                 sumaCoordZ=sumaCoordZ+float(lineA↩
                                        [46:54])
103                                 countSuma=countSuma+1
104                                 archivoB.seek(0,0)
105                                 break
106
107
108 #Se obtiene el centro geométrico de la interacción.
109             sumaCoordX=float(sumaCoordX/countSuma)
110             sumaCoordY=float(sumaCoordY/countSuma)
111             sumaCoordZ=float(sumaCoordZ/countSuma)
112             coorRCL=numpy.array((float(sumaCoordX),float(sumaCoordY↩
                    ),float(sumaCoordZ)))
113             archivoA.seek(0,0)
114             archivoB.seek(0,0)
115
116
```

```
#Se obtiene la distancia entre el aminoácido de la proteína A hasta el centro geométrico de la interacción, y se le suma la distancia entre el aminoácido de la proteína B hasta el centro geométrico de la interacción.
            for i in range(indRL,largoFC):
                if archfastcont[i]!=" ----------\n"↩
                    and archfastcont[i]!=" Top 20 Min & Max ↩
                   receptor-ligand residue free energy contacts\n":
                    residuoA=int(archfastcont[i][9:13])
                    residuoB=int(archfastcont[i][19:22])

                    for j, lineA in enumerate(archivoA):
                        if j==residuoA-1:
                            coorA=numpy.array((float(lineA[30:38]),↩
                                float(lineA[38:46]),float(lineA↩
                                [46:54])))
                            archivoA.seek(0,0)
                            break
                    for k, lineB in enumerate(archivoB):
                        if k==residuoB-1:
                            coorB=numpy.array((float(lineB[30:38]),↩
                                float(lineB[38:46]),float(lineB↩
                                [46:54])))
                            archivoB.seek(0,0)
                            break

                    distancia=archfastcont[i].rstrip('\n')+"{0:8.3f↩
                       }".format(float(numpy.linalg.norm(coorA-↩
                       coorRCL))+float(numpy.linalg.norm(coorRCL-↩
                       coorB)))+"\n"
                    archfastcont[i]=distancia
            archivoA.seek(0,0)
            archivoB.seek(0,0)
            archivoFC.close()
            archivoFCM.write("  Energy      AA-A      AA-B      RCL  ↩
               Complex: "+path[-4:]+"\n")
            archivoFCM.writelines(archfastcont[indRL-1:-1])
            archivoFCM.close()
            archivoA.close()
            archivoB.close()
```

## B.14. Algoritmo que une todos los enfoques de distancia en cada archivo de fastContact

```
1  # -*- coding: utf-8 -*-
2
3  from os import walk
4  #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta PDB.
5  for (path, ficheros, archivos) in walk("./PDB"):
6      if len(archivos)>1:
7          nombreArch=path[-4:]+".txt"
8  #Abrimos todos los archivos que contienen las caracteristicas de distancia
9          archivoA= open(path+"/fcGeoCenter_"+nombreArch, "r")
10         archivoB= open(path+"/fcCA_"+nombreArch, "r")
11         archivoC= open(path+"/fcCB_"+nombreArch, "r")
12         archivoD= open(path+"/fcCmAll_"+nombreArch, "r")
13         archivoE= open(path+"/fcCmBase_"+nombreArch, "r")
14         archivoF= open(path+"/fcCmRes_"+nombreArch, "r")
15         archivoG= open(path+"/fcRCL_"+nombreArch, "r")
16         archivoFinal=open(path+"/fcJoin_"+nombreArch, "w+")
17
18         lineasA=archivoA.readlines()
19         lineasB=archivoB.readlines()
20         lineasC=archivoC.readlines()
21         lineasD=archivoD.readlines()
22         lineasE=archivoE.readlines()
23         lineasF=archivoF.readlines()
24         lineasG=archivoG.readlines()
25         largoFC=len(lineasA)-1
26
27 #Se obtiene el indice en donde comienza la información de interacción receptor-ligando.
28         for i in range(largoFC):
29             if lineasA[i]==" Top 20 Min & Max receptor-ligand ↩
                   residue electrostatic contacts\n":
30                 indEnergia=i+1
31                 break
32
33 #Se concatenan todas las distancias.
34         for i in range(indEnergia,largoFC):
35             if(lineasA[i]!=" ------------------------------\n" and ↩
```

```
                lineasA[i]!=" Top 20 Min & Max receptor-ligand ↩
                residue free energy contacts\n"):
36               lineasA[i]=lineasA[i].rstrip('\n')+"{0:8s} {1:8s} ↩
                    {2:8s} {3:8s} {4:8s} {5:8s}\n".format((lineasB[i↩
                    ][27:35]), (lineasC[i][27:35]), (lineasD[i↩
                    ][27:35]), (lineasE[i][27:35]), (lineasF[i↩
                    ][27:35]), (lineasG[i][27:35]))
37
38          lineasA[0]="  Energy      AA-A      AA-B        GC       CA ↩
                  CB     CmAll    CmBase     CmLat     CmRCL   Complex: ↩
               "+path[-4:]+"\n"
39          archivoFinal.writelines(lineasA)
40          archivoFinal.close()
41          archivoA.close()
42          archivoB.close()
43          archivoC.close()
44          archivoD.close()
45          archivoE.close()
46          archivoF.close()
47          archivoG.close()
```

## B.15. Algoritmo que une a todos los archivos fastContact por tipo de interacción

```
1 from os import walk
2
3 #Crea dos archivos, uno para guardar todos los archivos FastContact que contienen los
      enfoques de distancia de los complejos transitorios, y otro con los complejos permanentes.
4 fcTrans=open("fcAllComplexTransient.txt","w+")
5 fcOblig=open("fcAllComplexObligate.txt","w+")
6
7
8
9 fcTrans.write("  Energy      AA-A      AA-B        GC       CA       CB↩
          CmAll    CmBase     CmLat       RCL\n\n")
10 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta
      atransient-contact.
11 for (path, ficheros, archivos) in walk("./PDB/atransient-contact"):
```

```
12      if len(archivos)>2:
13          nombreArch=path[-4:]+".txt"
14          archivoA = open(path+"/fcJoin_"+nombreArch, "r")
15          lineasA=archivoA.readlines()
16 #Concatena los archivos
17          fcTrans.write(" complex: "+path[-4:]+"\n")
18          fcTrans.writelines(lineasA[1:])
19          archivoA.close()
20
21
22
23 fcOblig.write(" Energy AA-A AA-B GC CA CB↩
        CmAll CmBase CmLat RCL\n\n")
24 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta
       aobligate-contact.
25 for (path, ficheros, archivos) in walk("./PDB/aobligate-contact"):
26      if len(archivos)>2:
27          nombreArch=path[-4:]+".txt"
28          archivoA = open(path+"/fcJoin_"+nombreArch, "r")
29          lineasA=archivoA.readlines()
30 #Concatena los archivos
31          fcOblig.write(" complex: "+path[-4:]+"\n")
32          fcOblig.writelines(lineasA[1:])
33          archivoA.close()
34
35 fcTrans.close()
36 fcOblig.close()
```

## B.16. Algoritmo que extrae las características energéticas de cada archivo fastContact

```
1 # -*- coding: utf-8 -*-
2 from os import walk
3
4 #Se crean dos archivos que guardarán todas las características energéticas presentes en cada en
       todos los archivos FastContact de los complejos proteicos.
5 archivoEnergiasT=open("EnergiasSolasTrans.txt", "w")
6 archivoEnergiasO=open("EnergiasSolasOblig.txt", "w")
```

```
7
8 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta
      atransient-contact.
9 for (path, ficheros, archivos) in walk("./PDB/atransient-contact"):
10     for archivo in archivos:
11         if archivo==(path[-4:]+".txt"):
12
13             archivoFC= open(path+"/"+archivo, "r")
14             archfastcont=archivoFC.readlines()
15             largoFC=len(archfastcont)-2
16
17             for i in range(largoFC):
18                 if archfastcont[i]==" Top 20 Min & Max residues ↩
                       contributing to the binding free energy\n":
19                     indEnergia=i+1
20
21                 if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
22                     finEnergia=i
23
24             for i in range(indEnergia,finEnergia):
25                 if archfastcont[i]!=" ------------------\n" and not ↩
                       archfastcont[i].startswith(" Top"):
26                     if archfastcont[i][1]=="*":
27                         ener="?"
28                     else:
29                         ener=float(archfastcont[i][1:9])
30
31                     archivoEnergiasT.write(str(ener)+",")
32
33             archivoEnergiasT.write("\n")
34             archivoFC.close()
35 archivoEnergiasT.close()
36
37
38 #Recorre los archivos de todas las carpetas que estan contenidas en la carpeta
      aobligate-contact.
39 for (path, ficheros, archivos) in walk("./PDB/aobligate-contact"):
40     for archivo in archivos:
41         if archivo==(path[-4:]+".txt"):
42
43             archivoFC= open(path+"/"+archivo, "r")
```

```
44              archfastcont=archivoFC.readlines()
45              largoFC=len(archfastcont)-2
46
47              for i in range(largoFC):
48                  if archfastcont[i]==" Top 20 Min & Max residues ↩
                       contributing to the binding free energy\n":
49                      indEnergia=i+1
50
51                  if archfastcont[i]==" Top 20 Min & Max receptor-↩
                       ligand residue electrostatic contacts\n":
52                      finEnergia=i
53
54              for i in range(indEnergia,finEnergia):
55                  if archfastcont[i]!=" ----------\n" and not ↩
                       archfastcont[i].startswith(" Top"):
56                      if archfastcont[i][1]=="*":
57                          ener=archfastcont[i][1:9]
58                      else:
59                          ener=float(archfastcont[i][1:9])
60
61                      archivoEnergiasO.write(str(ener)+",")
62
63              archivoEnergiasO.write("\n")
64              archivoFC.close()
65  archivoEnergiasO.close()
```

## B.17. Algoritmo que genera la matriz completa con todas las características energéticas y de distancia

```
1
2  #Se abren los dos archivos que contienen todos los enfoques de distancia y los dos archivos que
       contienen todas las caracteristicas energéticas. Además se crea el archivo ARFF que
       guardará todas las características.
3  unionTransient=open("fcAllComplexTransient.txt", "r")
4  unionObligate=open("fcAllComplexObligate.txt", "r")
5  matrizCompleta=open("matrizTodos.arff", "w+")
6  otrasEnergiasT=open("EnergiasSolasTrans.txt", "r")
7  otrasEnergiasO=open("EnergiasSolasOblig.txt", "r")
```

```
8
9  otrasEnerT=otrasEnergiasT.readlines()
10 otrasEner0=otrasEnergias0.readlines()
11 lineasUnionT=unionTransient.readlines()
12 lineasUnion0=unionObligate.readlines()
13 num=0
14
15 #Se sigue el formato que deben tener los archivos ARFF, por lo que de debe escribir el nombre
       de y tipo de todos los atributos, y luego los datos.
16 matrizCompleta.write("@relation Complexes\n\n")
17
18 for i in range(1,41):
19     matrizCompleta.write("@attribute EF-"+str(i)+" numeric\n")
20 for i in range(1,41):
21     matrizCompleta.write("@attribute L-ED"+str(i)+" numeric\n")
22 for i in range(1,41):
23     matrizCompleta.write("@attribute L-EE"+str(i)+" numeric\n")
24 for i in range(1,41):
25     matrizCompleta.write("@attribute R-ED"+str(i)+" numeric\n")
26 for i in range(1,41):
27     matrizCompleta.write("@attribute R-EE"+str(i)+" numeric\n")
28
29
30 for i in range(1,41):
31     matrizCompleta.write("@attribute RL-ED-"+str(i)+" numeric\n")
32     matrizCompleta.write("@attribute GC-ED-"+str(i)+" numeric\n")
33     matrizCompleta.write("@attribute CA-ED-"+str(i)+" numeric\n")
34     matrizCompleta.write("@attribute CB-ED-"+str(i)+" numeric\n")
35     matrizCompleta.write("@attribute CMALL-ED-"+str(i)+" numeric\n"↩
           )
36     matrizCompleta.write("@attribute CMBASE-ED-"+str(i)+" numeric\n↩
           ")
37     matrizCompleta.write("@attribute CMLAT-ED-"+str(i)+" numeric\n"↩
           )
38     matrizCompleta.write("@attribute RCL-ED-"+str(i)+" numeric\n")
39
40 for i in range(1,41):
41     matrizCompleta.write("@attribute RL-EE-"+str(i)+" numeric\n")
42     matrizCompleta.write("@attribute GC-EE-"+str(i)+" numeric\n")
43     matrizCompleta.write("@attribute CA-EE-"+str(i)+" numeric\n")
44     matrizCompleta.write("@attribute CB-EE-"+str(i)+" numeric\n")
45     matrizCompleta.write("@attribute CMALL-EE-"+str(i)+" numeric\n"↩
```

```
        )
46      matrizCompleta.write("@attribute CMBASE-EE-"+str(i)+" numeric\n")
47      matrizCompleta.write("@attribute CMLAT-EE-"+str(i)+" numeric\n")
48      matrizCompleta.write("@attribute RCL-EE-"+str(i)+" numeric\n")
49
50  matrizCompleta.write("@attribute class {t,o}\n")
51  matrizCompleta.write("\n@data\n\n")
52
53
54  #Union de las características energéticas y de distancias de los complejos transitorios. Cada linea corresponde a un complejo proteico.
55  for j in range(len(lineasUnionT)-2):
56      if(lineasUnionT[j].startswith(" complex")):
57          j=j+1
58
59          matrizCompleta.write(otrasEnerT[num][:-1])
60          num=num+1
61
62          while (j<len(lineasUnionT) and not lineasUnionT[j].startswith(" complex")):
63              if(not lineasUnionT[j].startswith(" ----") and not lineasUnionT[j].startswith(" Top 20") and not lineasUnionT[j].startswith(" complex")):
64                  matrizCompleta.write(lineasUnionT[j][1:9].replace(",","")+","+' '.join(lineasUnionT[j][28:89].split()).replace("********", "?").replace(" ",","))
65                  if not lineasUnionT[j+1].startswith(" ----------------------------------------"):
66                      matrizCompleta.write(",")
67              j=j+1
68
69  #En el final de cada linea se escribe la característica de clase de interacción, para los complejos transitorios es la letra t.
70          matrizCompleta.write(",t")
71          matrizCompleta.write("\n")
72
73  num=0
74  #Union de las características energéticas y de distancias de los complejos permanentes. Cada linea corresponde a un complejo proteico.
75  for j in range(len(lineasUnion0)-2):
```

```
76      if(lineasUnion0[j].startswith(" complex")):
77          j=j+1
78
79          matrizCompleta.write(otrasEner0[num][:-1])
80          num=num+1
81
82          while (j<len(lineasUnion0) and not lineasUnion0[j].↩
               startswith(" complex")):
83              if(not lineasUnion0[j].startswith(" ----") and not ↩
                   lineasUnion0[j].startswith(" Top 20") and not ↩
                   lineasUnion0[j].startswith(" complex")):
84                  matrizCompleta.write(lineasUnion0[j][1:9].replace("↩
                        ","")+","+' '.join(lineasUnion0[j][28:89].split↩
                       ()).replace("********", "?").replace(" ",","))
85                  if not lineasUnion0[j+1].startswith(" ↩
                       ----------------------------------------"):
86                      matrizCompleta.write(",")
87              j=j+1
88  #En el final de cada linea se escribe la característica de clase de interacción, para los complejos
        transitorios es la letra o.
89          matrizCompleta.write(",o")
90          matrizCompleta.write("\n")
91
92  union0bligate.close()
93  otrasEnergias0.close()
94  unionTransient.close()
95  matrizCompleta.close()
96  otrasEnergiasT.close()
```

# Apéndice C

# Funciones de las proteínas

Las proteínas son las responsables de la mayoría de las funciones biológicas de los seres vivos, estas funciones incluyen:

**Catálisis**: La catálisis es una función realizada por un tipo de proteína denominada enzima, cuya función es acelerar reacciones bioquímicas que sean energéticamente posibles.

**Estructura**: Esta es considera la función más importante de una proteína, este tipo de proteína genera soporte, forman parte íntegra de los organismos, por ejemplo la queratina, la cual encontraremos en el pelo, uñas y piel, y cuya función es ayudar a dar resistencia a estas estructuras.

**Movimiento**: Las proteínas están involucradas en todos los movimientos celulares, un ejemplo de ello son la Actina y Tubulina, que junto a otras proteínas forman el citoesqueleto.

**Defensa**: Una gran variedad de proteínas pertenecen a la categoría de defensa, aquí podemos encontrar los anticuerpos, que son proteínas encargadas que prevenir infecciones, enfermedades y dolencias.

**Regulación**: Las proteínas también regulan los procesos del organismo, como la Insulina, hormona que regula la azúcar de la sangre enviando un mensaje a las células del cuerpo sobre la cantidad de azúcar presente en la sangre [37].

**Transporte**: Otra de las funciones de las proteínas es el transporte de moléculas o iones a través de membranas o células, un ejemplo de proteína transportadora es la hemoglobina, cuya

función es transportar O2 desde los pulmones hacia el resto del cuerpo a través de la sangre.

**Almacenamiento**: Este tipo de proteína funciona como reserva de nutrientes, por ejemplo la ferritina, que es una proteína que se combina con hierro para ser almacenado en el hígado [38].