Universidad del BíoBío, Chile
Facultad de Ciencias Empresariales
Departamento de Sistemas de Información

Análisis de Algoritmos de Selección para mejorar tiempos de ejecución en obtención de Características de Interacción de Proteína-Proteína

Tesis presentada por
Sebastián Alberto Tare Bustos
para obtener el título de
Ingeniero de Ejecución en Computación e Informática
dirigida por Tatiana Gutiérrez Bunster

2017

Resumen

Hoy en día existe una gran motivación por entender los procesos biológicos de los seres vivos. Las proteínas son unas bio-moléculas muy importantes para la vida y prácticamente todos los procesos biológicos dependen de la presencia o la actividad de las proteínas.

Este proyecto de título comprende el estudio algoritmos de selección de características para procesar datos de interacciones entre proteínas. La motivación para esto es porque las proteínas actúan en conjunto con otras proteínas, y dependiendo del tipo de interacción se puede ayudar a definir las funciones que poseen.

Se estudiaron 19 algoritmos de selección de características y se eligieron 5 de los más prometedores para probar su desempeño. Estos algoritmos fueron ejecutados con 5 matrices de 295 complejos proteicos(filas) y entre un rango de 600-1800 características (columnas) por matriz. Los subconjuntos generados fueron luego utilizados para construir un clasificador, el cual es un algoritmo que trata de aprender sobre los datos e inferir una regla general para evaluar nuevos datos y poder clasificar si son de clase A o clase B. Se evaluó el desempeño de 4 clasificadores, Naïve Bayes, SVM lineal, Random Tree y Random Forest. Se encontró que al utilizar el algoritmo T score con la matriz de datos sin residuo, se alcanzó un 93.3 % de exactitud en la clasificación utilizando división por porcentaje de 90 %.

Índice general

Índice de figuras

Índice de tablas

Abreviaciones

IPP	**I**nteracción **P**roteína-**P**roteína
IA	**I**nteligencia **A**rtificial
SVM	**S**upport **V**ector **M**achine
MAG	Matriz de datos del trabajo de **MAG**ister de Gutiérrez [11]
DOC	Matriz de datos del trabajo de **DOC**torado de Gutiérrez [14]
B&B	**B**ranch **and** **B**ound
BF	**B**est **F**irst
MDLM	**M**inimum **D**escription **L**ength **M**ethod
CFS	**C**orrelation based **F**eature **S**election
AB&B	**A**utomatic **B**ranch **and** **B**ound
BS	**B**eam **S**earch
AMB&B	**A**pproximate **M**onotonic **B**ranch **and** **B**ound
CIFE	**C**onditional **I**nfomax **F**eature **E**xtraction

Capítulo 1

Introducción

En la Inteligencia Artificial existe una rama llamada Machine Learning o Aprendizaje Automático en la cual se desarrollan algoritmos para que las computadoras puedan automáticamente mejorar su desempeño en tareas específicas basándose en experiencias pasadas, lo cual se conoce como aprendizaje [17].

La clasificación es una de las aplicaciones del Aprendizaje Automático en la cual se desarrollan algoritmos que tratan de aprender patrones de los datos de entrenamiento para generar una regla de clasificación que permita asignar una clase a los datos futuros, este proceso se llama aprendizaje supervisado, ya que estos datos están previamente clasificados. El objetivo de utilizar los datos de entrenamiento es inferir una regla general que permita relacionarlos con una clase. Esto significa que el sistema de aprendizaje debe ajustar a los datos un modelo (minimizar la diferencia entre la regla inducida y los datos de entrenamiento), que es la idea que utilizará el algoritmo para discriminar los datos según su clase [18].

Por ejemplo, los correos spam son mensajes basura o no deseados que llegan a los correos electrónicos de las personas. Una forma de poder evitar que lleguen a la bandeja principal es crear un clasificador que procese los correos y los mueva a la bandeja de correos no deseados si es que el correo entrante es catalogado como spam. Para hacer esto se puede crear un conjunto con correos spam y no spam elegidos por un usuario, y luego se elige algún modelo que permita discriminar sobre si son spam o no. Se puede utilizar un modelo probabilístico el cual genere una distribución en base a la cantidad de palabras más comunes en los correos spam, para calcular qué tan probable es que el mensaje entrante sea spam y entonces filtrarlo a la carpeta spam si la probabilidad es alta, evitando que queden en la bandeja principal.

La Bioinformática es un campo el cual engloba distintas áreas de otras disciplinas como la Estadística, Inteligencia Artificial, Química, Bioquímica, entre otras. Se utiliza para la distribución y análisis de datos biológicos, como el alineamiento de secuencias, predicción de genes, predicción de estructura de proteínas y análisis de Interacciones de Proteína-Proteína (IPP). Entre sus distintas aplicaciones se puede mencionar la detección de enfermedades a partir de los datos genéticos de los pacientes y el descubrimiento de funciones de las proteínas.

Este proyecto de título se enfoca en el análisis de algoritmos de selección de características y los datos de IPP previamente clasificados. El objetivo del análisis es conocer qué algoritmos seleccionan las mejores características de los conjuntos de datos y cuales conjuntos poseen mayor poder predictivo. Para realizar esto se probarán distintos clasificadores con distintos subconjuntos de datos obtenidos a partir de matrices generadas por la ejecución de algoritmos de selección de características. La clasificación es de tipo binaria lo cual significa que se discrimina entre 2 clases.

1.1. Definición del problema

Cuando se quiere entrenar un clasificador, el conjunto de entrenamiento debe tener una cantidad suficiente de ejemplos y una cantidad óptima de características para poder generalizar sobre los datos del conjunto y poder asegurar que los datos que se presenten a futuro tengan los patrones observados, ya que si los ejemplos son pocos pueden quedar fuera patrones importantes los cuales no serán considerados por el clasificador.

Por ejemplo, si se quiere construir un clasificador que diferencie entre perros y gatos, se podría previamente sacar características irrelevantes como el nombre o fecha de nacimiento para dejar las características relevantes como peso, color de pelo, altura, entre otras. Estas características deben representar eficientemente el conjunto y las que posean poco poder para describir la clase deben ser extraídas, como el color, ya que es probable que compartan el mismo color en varias instancias y debido a esa característica un gato sea clasificado incorrectamente como perro.

Cuando se trata de un problema tan complejo como las IPP se busca obtener de ellas la mayor cantidad de información, pero sucede que la intuición falla en grandes cantidades de datos y por lo tanto no es fácil elegir cuales características usar para el entrenamiento, por lo tanto es preferible considerar una gran cantidad de características.

Aunque existe un inconveniente, el cual es decidir cuándo es suficiente, ya que si se calcula que de 1800 características sólo 100 son relevantes para la clasificación, se cae en el problema de sobredimensionar el conjunto al considerar tantas variables como

entrada, lo cual causaría que se construya el clasificador con información redundante, lo que generaría un bajo porcentaje de exactitud de la predicción.

Hay algoritmos que sirven para pre-procesar o filtrar los datos del conjunto, lo cual ayuda para reducir el espacio de búsqueda a uno más óptimo, también hay otros algoritmos que evalúan las características dependiendo de qué tan útiles son para un clasificador en específico, pero son muy costosos en tiempo, sobre todo si la cantidad de características es muy elevada porque iteran sobre ellas. También ocurre que mezclan ambos tipos de algoritmos para que los de tipo filtro (detallado en la Sección 5.1) ayuden a los que demoran más, o simplemente utilizan los algoritmos de filtro para reducir el ruido de los datos, o las características demasiado redundantes y de esta forma aprovechar la capacidad predictiva innata de los datos.

También ocurre que no se tiene suficiente cantidad de ejemplos para poder probar la efectividad real de los algoritmos de selección de características y el clasificador construido, ya que las pruebas son realizadas sobre el mismo conjunto, siendo que lo óptimo sería probar el clasificador con nuevos ejemplos. Se puede dividir el conjunto original pero como se hace mención antes, es mucho mejor tener más ejemplos.

Entonces el problema que se trata en este proyecto de título es encontrar qué opciones se pueden aplicar para realizar selección de características con los conjuntos de datos de las IPP para una gran cantidad de características con un número reducido de ejemplos y en un tiempo de ejecución reducido.

1.2. Hipótesis

La hipótesis de proyecto de título es: existe por lo menos un algoritmo de selección de características que se puede implementar y/o adaptar para mejorar los resultados de algoritmos presentados en los trabajos previos sobre los datos de interacciones proteína-proteína, considerando tiempos de ejecución reducidos.

1.3. Objetivos

Los objetivos de este proyecto de título se han formulado de la siguiente forma.

1.3.1. Objetivo General

El objetivo principal de éste trabajo es obtener ya sea desarrollando, mejorando, implementando o re-enfocando un algoritmo de selección de características que sea capaz de trabajar con matrices de datos de interacción de proteínas de una dimensión aproximada de 1800 características (columnas) y 300 ejemplos (filas), en tiempos de ejecución bajos y alcanzando un porcentaje de exactitud igual o mayor al obtenido en los trabajos previos.

1.3.2. Objetivos Específicos

- Examinar los algoritmos de selección de características en la literatura [11, 14, 16].
- Preparar los conjuntos de datos de IPP para poder ejecutar los algoritmos de selección.
- De la compilación de algoritmos examinados, identificar 5 algoritmos prometedores para ejecutarlos con los conjuntos preparados y generar subconjuntos.
- Clasificación de subconjuntos generados con los clasificadores más utilizados como Naïve bayes, SVM, Random Tree y Random Forest.
- Evaluar los resultados de la selección de características y de la clasificación para decidir qué algoritmo y clasificador son mejores para predecir los tipos de las proteínas del conjunto.

1.4. Organización

Este proyecto de título se encuentra dividido en los siguientes capítulos.

Capítulo 1: Se presenta el proyecto de título y se explica lo que es el Aprendizaje Automático y su enfoque hacia la Bioinformática, con algunos ejemplos simples sobre sus aplicaciones, se presenta la definición del problema, la hipótesis, los objetivos general y específico, y la organización.

Capítulo 2: En este capítulo se profundiza sobre los procesos utilizados en el Aprendizaje Automático, sus etapas y distintas herramientas.

Capítulo 3: Explica qué es la Bioinformática, qué son las Proteínas junto con sus propiedades, funciones e interacción.

Capítulo 4: Muestra la metodología utilizada para ordenar el trabajo a realizar, cómo se ordenan y se componen las matrices de datos a utilizar.

Capítulo 5: Se explica qué son los algoritmos de selección de características, además se presenta una compilación de algoritmos y se explican cómo funcionan algunos.

Capítulo 6: En este capítulo se explica el funcionamiento a detalle de los algoritmos elegidos para implementar.

Capítulo 7: Resultados de la ejecución de los algoritmos de selección de características y de la clasificación de los conjuntos generados.

Capítulo 8: En esta sección se da a conocer el estado final de los resultados, el cumplimiento de objetivos y si se confirmó la hipótesis. También se da a conocer los posibles caminos a seguir que surgieron a través de el proceso de investigación y opciones para el trabajo futuro.

Capítulo 2

Aprendizaje Automático

El Aprendizaje Automático es una rama de la Inteligencia Artificial en donde se construyen programas computacionales que desempeñen tareas específicas y que puedan automáticamente mejorar con la experiencia [17]. Este proceso se conoce como aprendizaje y se puede utilizar para que un sistema pueda aprender cuales son los patrones de los objetos de estudio. Esto quiere decir que se tiene un modelo predictivo matemático definido con algunos parámetros y se ejecuta un programa para optimizar estos parámetros a través de los datos de entrenamiento o experiencia, para poder realizar mejores predicciones [18].

2.1. Enfoques del reconocimiento de patrones

Cuando se trata de reconocimiento de patrones puedo mencionar los siguientes enfoques con los que se trata de asignar una clase a los diferentes objetos de estudio.

El primer enfoque es el **reconocimiento estadístico**, en donde cada patrón está representado por d características y es visto como un punto en un espacio de dimensión d. La efectividad del conjunto va a depender de qué tan bien se pueden separar los patrones de las distintas clases. Entonces se deben establecer decisiones para limitar el espacio de características para separar los patrones correspondientes a cada clase. Luego de esto se pueden establecer decisiones para asignar las clases basados en las distribuciones de patrones pertenecientes a cada clase, lo cual puede ser especificado por el usuario o aprendido a partir de los ejemplos [19].

El segundo enfoque es llamado **reconocimiento sintáctico** y busca analizar textos para determinar la estructura de cada oración, para poder interpretar o describir lo que está

escrito [20]. De la misma forma se puede utilizar este enfoque para analizar secuencias de proteínas para clasificar una estructura del universo de estructuras clasificables.

El tercer enfoque y último es el **lógico combinatorio**, ya que para poder realizar la clasificación se forman grupos de los elementos del conjunto de acuerdo a su semejanza y luego se da un valor específico a cada semejanza [21]. Esto se puede realizar a través de formalismos matemáticos para derivar nuevos conocimientos a partir de los existentes.

Para este proyecto de título se utiliza el enfoque estadístico debido a que se desconoce el número de patrones que puedan haber y cómo poder separar efectivamente los datos de los conjuntos, por lo tanto se utilizan distribuciones para poder generar rangos de parámetros como patrones.

2.2. Etapas del reconocimiento de patrones

Un sistema que reconoce patrones se puede componer de varias etapas, pero en este trabajo se han utilizado las siguientes:

2.2.1. Obtención de datos

La obtención de los datos se refiere a cualquier sistema que entregue información o datos, como puede ser un termómetro, bases de datos o una cámara. Por lo tanto, la temperatura, distancia, aceleración, humedad, entre otras, son utilizados como entrada para ser procesadas un sistema que reconoce patrones.

2.2.2. Representación de datos

En esta etapa se busca reducir la dimensión de los datos y encontrar un conjunto el cual pueda representar de manera óptima los datos obtenidos originalmente. Este paso es importante cuando se trabaja con datos muy complejos ya que ayuda a óptimamente reducir la dimensión de estos datos mientras preserva su información sobre la vecindad entre cada uno de los datos [22], en otras palabras, se crea una representación de los datos para trabajarlos más fácilmente.

2.2.2.1. Normalización

Es un proceso el cual ayuda que las características sean medibles en igual proporción, esto quiere decir, que si existen características con caracteres y otras con números, filtrarlas

y dejar las que poseen sólo números es una forma de normalizar. Normalización significa regularizar, lo cual se puede aplicar, como cuando los valores poseen rangos muy distintos [23].

2.2.2.2. Selección de características

La selección de características es una técnica para poder encontrar qué características son más relevantes, cuales representan mejor la clase y por lo tanto, cuáles van a entregar un mayor porcentaje de exactitud al momento de clasificar los datos debido a que la calidad de los patrones va a ser mejor. Es necesaria aplicar en muchos casos para reducir las entradas a un tamaño apropiado para poder procesar los datos y analizarlos.

El efecto Hughes (también conocido como la maldición de la dimensionalidad) [24] es un fenómeno que ocurre debido a que el proceso de aprendizaje requiere una cantidad de ejemplos basados en la cantidad exponencial de características, eso quiere decir que si hay muchas características el aprendizaje se vuelve muy costoso en recursos computacionales.

2.2.3. Clasificación

La clasificación es un proceso mediante el cual un algoritmo clasificador detecta patrones de un objeto de estudio y a partir de estos le asigna una clase a dicho objeto. Realiza un proceso de aprendizaje mediante el cual ajusta los datos a un modelo, el cual está definido por un conjunto de parámetros para poder discriminar sobre los datos según su clase [18].

Tipo de aprendizaje

Los tipos de aprendizaje varían según la información que se le proporciona a un clasificador.

Supervisado El aprendizaje supervisado corresponde a la clasificación de nuevos objetos basándose en la función generada al clasificar objetos los cuales se conocía la clase de cada uno. Éste es el tipo de aprendizaje utilizado en este proyecto de título y se emplea para poder utilizar el conjunto de entrenamiento como prueba de la eficacia del clasificador al comparar los resultados de la clasificación con las clases de cada ejemplo del conjunto de entrenamiento.

No-supervisado Corresponde a una clasificación sin tener pistas sobre el conjunto que se está tratando de clasificar (no tiene conocimiento a priori). En este tipo de aprendizaje se consideran las variables de entrada como aleatorias para luego calcular las probabilidades condicionales de las variables. Otra forma es el clustering, el cual agrupa los objetos basados en un criterio de algún patrón que posean [25].

2.2.4. Evaluación

En esta etapa se evalúan los resultados para tomar la decisión final con respecto a la asignación de las clases a el conjunto, para generar un vector de clases correspondiente a los ejemplos.

2.3. Métodos de clasificación

A continuación se muestran los métodos de clasificación utilizados para este trabajo, los cuales fueron seleccionados a partir los trabajos anteriores [11, 14, 16].

2.3.1. Naïve Bayes

Un ejemplo de Naïve bayes [1] es que teniendo un problema de 2 clases, se calcula la probabilidad de que un ejemplo sea de cierta clase, sólo si la probabilidad condicional de que el ejemplo dado la probabilidad de la clase, dividido por la probabilidad condicional del ejemplo dado la probabilidad de la otra clase, sea mayor o igual a 1.

Entonces, teniendo aquel criterio de clasificación, se calcula la Ecuación 2.1.

$$f_{nb}(E) = \frac{p(C1)}{p(C2)} \prod_{i=1}^{n} \frac{p(x_i|C1)}{p(x_i|C2)} \tag{2.1}$$

Para esta ecuación, *C1* representa la clase 1, *C2* representa la clase 2 y x_i un valor de la característica de un ejemplo.

Luego, las características de los ejemplos clasificados como clase 1 se añaden a un árbol como hijas del nodo de clase 1 y lo mismo con la clase 2, todas independientes, según muestra la Figura 2.1, en donde A representa un atributo o característica.

Entonces, cuando se clasifique un nuevo ejemplo, calcula independientemente si es que tiene el atributo *A1* o si tiene el atributo *A2* y si tiene el atributo *An*, y compara con la probabilidad de la otra clase para tomar la decisión de a qué clase asignar el ejemplo.

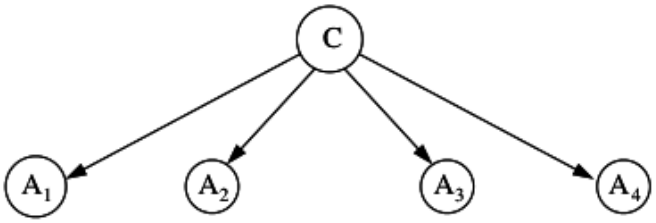


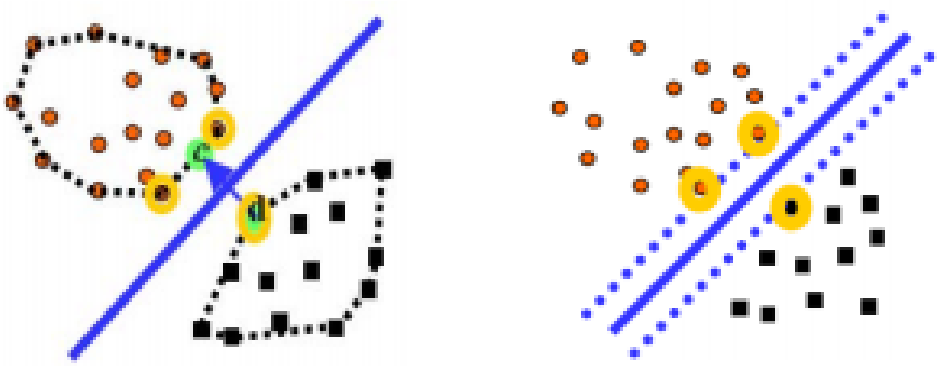
FIGURA 2.1: Nodos de árbol de Naïve Bayes [1]

2.3.2. Máquinas de Vector de Soporte (SVM)

Para realizar la clasificación se deben graficar los ejemplos en un plano de dimensión de n características y luego se trata de encontrar un margen que pueda dividir el espacio de los ejemplos para separar los que son de una clase de otra [2, 26].

Cuando se habla de margen, se refiere a la línea que separa las clases con un hiper-plano lineal separador, lo cual hace que SVM sea para problemas de 2 clases (binario), aunque existen técnicas las cuales permiten trabajar con más. Para encontrar el hiper-plano de decisión óptimo (que maximiza la separabilidad) se pueden emplear 2 técnicas:

- La Figura 2.2(a) muestra los puntos más cercanos de los que forman una envolvente convexa y que como se ve en la imagen, los puntos $\boldsymbol{c}$ y $\boldsymbol{d}$ bisectan.
- Y en la Figura 2.2(b) se muestra que al agrandar el margen de los dos planos de soporte paralelos consiguen el mismo resultado.

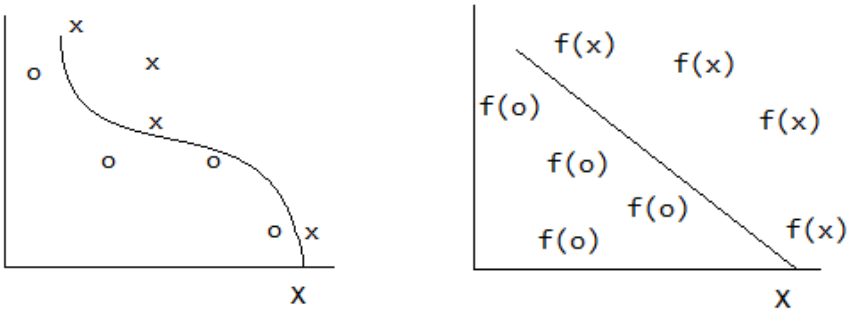


(a) Hiperplano 1 (b) Hiperplano 2

FIGURA 2.2: Hiper-planos lineal separadores [2]

Hay dos criterios con los cuales se pueden separar los datos, el caso lineal y el kernel (no lineal) como se puede observar en la Figura 2.3(a) y 2.3(b). En el caso lineal se emplea un hiperplano lineal separador. Como se habló anteriormente, se trata de generar un hiper-plano el cual maximice la distancia del margen que separa las 2 clases.

La razón de porqué se utiliza el kernel porque al no ser lineal, permite trabajar con datos que no se pueden separar linealmente y que pueden poseer ruido, los cuales corresponden a datos del mundo real.

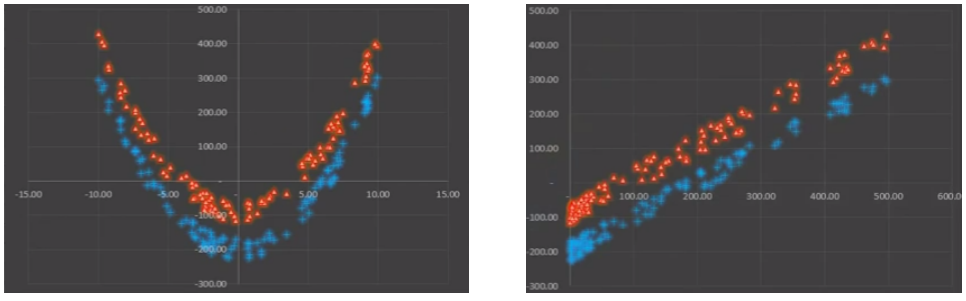


(a) Separación no lineal

(b) Separación lineal

FIGURA 2.3: Separación no lineal y lineal

Hay una función llamada función kernel [3, 27], que permite introducir nuevas características a un nuevo espacio al modificar sus valores darle forma al espacio de características y que puedan procesarse como un espacio lineal como lo se muestra en las Figuras 2.4(a) y 2.4(b).

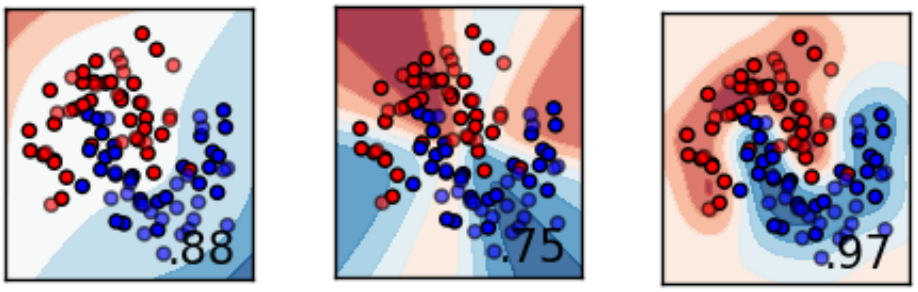


(a) Ejemplo de espacio no lineal

(b) Ejemplo de espacio no lineal transformado a lineal utilizando la función kernel

FIGURA 2.4: Resultado de la aplicación de una función kernel lineal a un espacio no lineal [3]

De algunas transformaciones que se pueden aplicar para diferenciar las características según su clase [26] puedo mencionar la **polinomial**, que corresponde a la Figura 2.5(a), la **sigmoideo** que corresponde a la Figura 2.5(b) y **base radial** como se muestra en la Figura 2.5(c).



(a) Ejemplo de espacio polinomial

(b) Ejemplo de espacio sigmoideo

(c) Ejemplo de espacio base radial

FIGURA 2.5: Representación gráfica de espacios no lineal para una máquina vector de soporte [4]

2.3.3. Árbol Aleatorio

El término Árbol Aleatorio (Random Tree) [28] no tiene que ver con clasificación, sin embargo es utilizado para referirse a árboles de decisiones construidos a partir de un conjunto aleatorio de columnas. El clasificador basado en un árbol de decisión es un enfoque utilizado para poder realizar decisiones complejas al unir decisiones simples.

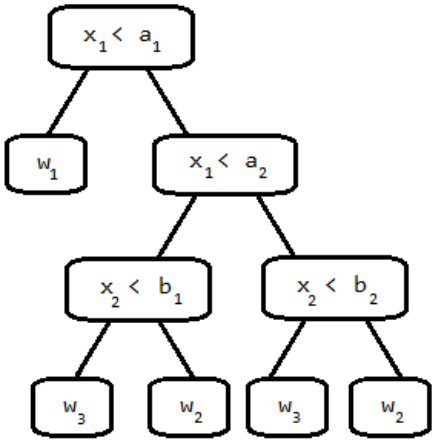


FIGURA 2.6: Representación de árbol de decisión [5].

En la Figura 2.6 se puede apreciar una representación simple de un árbol de decisión, en donde w_1 es la clase 1, w_2 es la clase 2 y w_3 es la clase 3. Las x son características que evalúa con respecto a cierto patrón encontrado en el conjunto de entrenamiento, por ejemplo si una característica x posee un valor menor que la variable a, entonces quiere decir que pertenece a la clase w, sino se continúa la evaluación recorriendo todas las características del conjunto.

Por esto es que requiere una selección de características eficiente, porque puede darse que los valores de una característica son completamente aleatorios y no se puede concluir a qué clase pertenece a partir de ella, entonces no debería estar en el conjunto. Además es importante la forma en la que se va construyendo el árbol, el criterio de evaluación de las características y el promedio máximo de ganancia de información mutual con respecto a la clase [5].

Un aspecto importante de los árboles de decisiones es que son fácil de comprender e interpretar el porqué el clasificador toma sus decisiones.

Un ejemplo simple [29] se puede apreciar al utilizar un árbol de decisión para clasificar el tipo de flor de Iris, si son Setosa, Versicolor o Virginica, basándose en información como ancho y largo de pétalos.

Tras entrenar el clasificador con los datos de ejemplo, se genera el árbol que muestra la Figura 2.7.

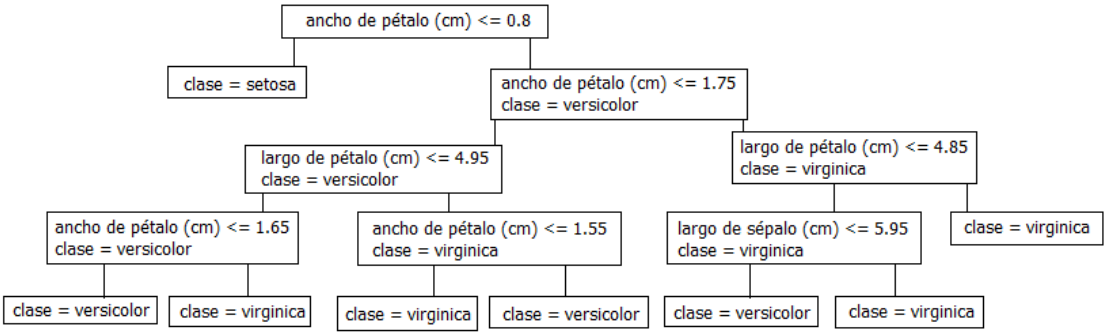


FIGURA 2.7: Árbol de decisión [5].

Y si por ejemplo se presenta un nuevo dato con un ancho de pétalo de 1.4 cm, un largo de pétalo de 4.7 cm, un ancho de sépalo de 3.2 cm, y un largo de sépalo de 7 cm. Se comienza por el primer nodo y se evalúa la variable del ancho del pétalo. Como 1.4 cm no es menor o igual a 0.8 cm y se avanza al siguiente nivel del árbol, al nodo derecho puesto que la evaluación indica que no es verdad que el nuevo dato es setosa. Y de esta forma se continúa evaluando hasta llegar a un nodo que no tenga continuación, entonces se le asigna la clase al ejemplo dependiendo en qué nodo quedó. Con el nuevo dato de ejemplo si se continúa el proceso, terminaría siendo clasificado como versicolor.

2.3.4. Bosque Aleatorio (Random Forest)

El clasificador Bosque Aleatorio utiliza como base el árbol de decisión aleatorio (random tree) y básicamente lo que hace es entrenar varios árboles aleatorios y luego cuando se trate de clasificar un nuevo ejemplo utilizándolos, asigna la clase obtenida de ellos que más se repita.

Utiliza la idea de bagging [30] la cual dice que la combinación de modelos de aprendizaje aumenta a exactitud de la clasificación.

2.4. Evaluación

La parte de evaluación corresponde a técnicas las cuales sirven para validar los resultados obtenidos, para tratar de garantizar un buen rendimiento de los clasificadores. Es importante realizar una evaluación no sólo para saber el porcentaje de acierto de sus futuras predicciones, si no que también para elegir un clasificador que se pueda conseguir el mejor modelo para los datos, o también para combinar clasificadores [31].

Para calcular el porcentaje de acierto del clasificador, se compara los resultados de la clasificación con las clases correspondientes a cada ejemplo y se obtiene un porcentaje de cuan exacta fue la clasificación. En cambio sólo utilizando este tipo de evaluación no se garantizan resultados robustos. Por lo tanto se utilizan métodos como la validación cruzada y división por porcentaje, los cuales fueron usados en este trabajo y se explican a continuación.

2.4.1. Entrenamiento normal

Se utiliza el conjunto completo para el entrenamiento, luego se aplica un algoritmo de inducción para construir el clasificador a partir del conjunto y finalmente se clasifica el mismo conjunto con el clasificador generado. Aunque esta forma de entrenamiento no es recomendada utilizarla porque el clasificador construido probablemente sirva sólo para el mismo conjunto y no para otros ejemplos futuros.

2.4.2. Validación Cruzada (Cross-Validation)

Lo que hace esta técnica es dividir el conjunto en n partes iguales. Sirve para asegurar que cada instancia del conjunto sea usada para el entrenamiento y evitar una alta varianza de la exactitud. Normalmente se utiliza una división de 10 (k pliegues) en la cual se toma el 90 por ciento inicial del conjunto para entrenar el clasificador y el 10 por ciento final para probarlo como se ve en la primera iteración en la Figura 2.8. Y luego se elije el siguiente conjunto de pruebas con la penúltima división y el resto del conjunto para entrenamiento, hasta que se han completado las 10 iteraciones.

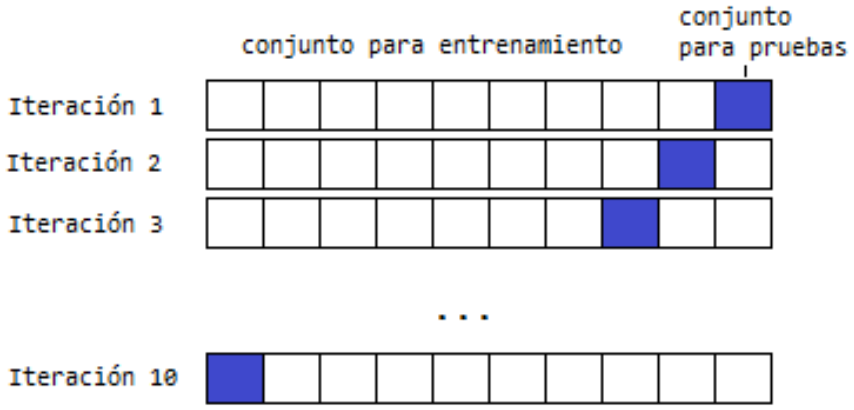


FIGURA 2.8: Representación de validación cruzada.

2.4.3. División por Porcentaje (Percantage-Split)

La división por porcentaje es parecida a la validación cruzada, por ejemplo, una de las más utilizadas es la de división de 66 % y 90 %. En la de 90 % se divide el conjunto de atributos en 90 % para entrenar el algoritmo y 10 % para probarlo, normalmente se reordena el conjunto aleatoriamente antes de realizar este proceso [32].

2.4.4. Evaluación de la Clasificación y Medidas de Desempeño

Evaluar la clasificación parte en cuestionar el porqué suceden ciertos resultados y cómo mejorarlos. En algunos casos sucede que la exactitud de predicción de una clase es mucho mayor que la exactitud para la otra clase. Por ejemplo, puede ocurrir que el total del conjunto de entrenamiento para una clase A supera en cantidad al conjunto de entrenamiento otra clase B, lo que se conoce como el problema de clases no balanceadas [33]. Esto genera que se sobre ajuste el modelo porque al tratar de representar todo el espacio, mejora la clasificación de los ejemplos que predominan y ocurre que la clase A

predominante tiene 95 por ciento de ser clasificada correctamente y la clase B un 0 por ciento.

Una de las primeras opciones para cuestionar si surge este problema es si se puede conseguir más datos para el conjunto de la clase minoritaria. Reducir el conjunto para poder tener una proporción 1:1 de los datos es una opción sólo si se tienen muchos ejemplos o si la cantidad de características es pequeña, pero no es recomendable, sobre todo si se descartan datos sin algún criterio, ya que podrían poseer información relevante.

Por lo tanto para combatir estos problemas se utilizan diferentes medidas del desempeño del clasificador como las siguientes.

2.4.4.1. Matriz de Confusión

Consiste en una matriz como muestra la Tabla 2.1 en donde se ubica la cantidad de instancias clasificadas correctamente e incorrectamente para ambas clases (también sirve para problemas con más clases). La importancia radica en que utiliza más variables que cuando se calcula la exactitud del clasificador.

- Verdadero Positivo (True Positive) (TP), es cuando se clasifica correctamente un objeto de clase A como clase A.
- Falso Positivo (False Positive) (FP) es cuando se clasifica un objeto como clase B, siendo que es de clase A.
- Falso Negativo (False Negative) (FN), es cuando se clasifica un objeto de clase B como clase A.
- Verdadero Negativo (True Negative) (TN), es cuando clasifica un objeto como clase B, siendo que es de clase B (o más bien dicho cuando se rechaza la hipótesis de que el objeto era de clase A y acierta, ya que era de clase B).

	Positivo	**Negativo**
Positivo	Verdadero Positivo (TP)	Falso Positivo (FP)
Negativo	Falso Negativo (FN)	Verdadero Negativo (TN)

Tabla 2.1: Tabla de representación de una matriz de confusión

Las siguientes medidas son algunas de las utilizadas para evaluar la clasificación de clases no balanceadas [34].

2.4.4.2. Precisión

Indica qué tanto predice correctamente, por lo que una baja precisión quiere decir que existen muchos intentos fallidos de predecir la clase A, o falsos positivos.

$$\frac{TP}{TP + FP} \tag{2.2}$$

2.4.4.3. Recall

También llamado sensitividad, es la probabilidad de acertar correctamente la clase A, de forma que si existen muchos objetos mal clasificados como A, entonces disminuye el valor de recall.

$$\frac{TP}{TP + FN} \tag{2.3}$$

2.4.4.4. F1 score

Se basa en un equilibrio entre la medida de Precisión y Recall, esto se conoce como Harmonic Mean (promedio harmónico de precisión y sensitividad).

$$2 * (\frac{Precision * Recall}{Precision + Recall}) \tag{2.4}$$

2.4.4.5. Coeficiente Kappa

Mide la proporción de objetos correctamente clasificados para después poder contabilizar la probabilidad del porcentaje de acuerdo [35]. Para realizar esto se debe calcular un acuerdo en la Observación de los datos clasificados:

$$OA = \frac{TP + TN}{P + N} \tag{2.5}$$

P = Todas las clases positivas, o clase 1. N = Todas las clases negativas, o clase 2, y $P+N$ es el total de objetos clasificados.

Luego se calcula el acuerdo de posibilidad de los positivos con los negativos:

$$AC = \frac{TP + FN}{P + N} * \frac{TP + FP}{P + N} + \frac{FP + TN}{P + N} * \frac{FN + TN}{P + N} \tag{2.6}$$

Y entonces se calcula el coeficiente Kappa:

$$K = \frac{OA - AC}{1 - AC} \tag{2.7}$$

Las interpretaciones del coeficiente kappa no suelen ser estandarizadas, esto quiere decir que se pueden concluir distintos resultados con los mismos datos dependiendo de qué es lo que se considera relevante, por ejemplo si tuviera 95 por ciento del total de las instancias de clase A clasificadas como clase A y sólo un 40 por ciento del total de la clase B clasificada como clase B, y el número de instancias de la clase A excede por mucho a la clase B, entonces se preferiría un clasificador que mejore la exactitud para la clase B y esperar un valor de K menor.

El valor de K puede ir entre -1 y 1, generalmente se considera como casi perfecto un valor de 0.81 a 1 de concordancia, y entre 0 a 0.21 a casi sin concordancia, y los valores negativos quiere decir que no existe un acuerdo efectivo entre las razones [36].

2.4.4.6. Receiver Operating Characteristic (ROC)

Eles un gráfico el cual se crea trazando los valores Verdadero Positivo contra los Falso Positivo a distintas configuraciones de límites y sólo sirve para problemas binarios [37].

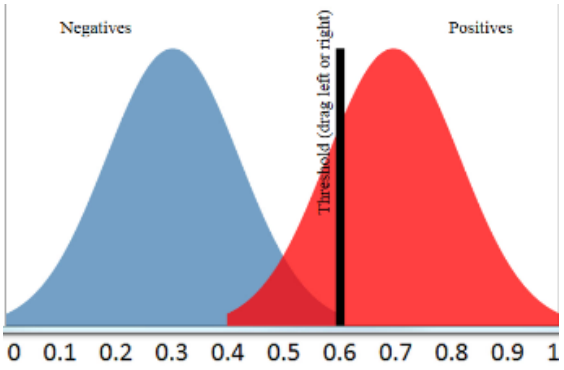


FIGURA 2.9: Ejemplo de distribuciones de 2 clases del evaluador ROC [6].

La Figura 2.9 es un ejemplo de una distribución entre 2 clases, la izquierda son las instancias clasificadas como negativas y la derecha son las positivas. El punto intermedio son un conjunto de instancias clasificadas como positivas y otras negativas, y debido a eso, colocar un límite justo ese punto en donde se intersectan (0.5) sirve para maximizar la separación de las clases.

El gráfico que muestra la Figura 2.10 se genera con los datos de las instancias clasificadas.

A distintos niveles que se pueden fijar el límite varía la forma de esta curva.

Para comparar clasificadores lo que se utiliza es el área bajo la curva (Area Under the Curve (AUC)) de forma que un clasificador con mayor AUC puede fijar mejor un límite que separe las clases y alcance un mayor potencial para clasificar las instancias. El mejor caso sería que el AUC se asemeja al área de un triángulo que se forma con el eje Verdadero Positivo. En la figura 2.10 se puede apreciar 3 curvas distintas, siendo la que está más cerca del eje de la Razón de Falso Positivo, la que posee un valor poco útil de separabilidad de las clases, la curva que está más cerca del eje de la Razón de Verdadero Positivo es la que posee el mayor valor del área bajo la curva, mientras que la del medio es un valor bueno pero cercano a 0.

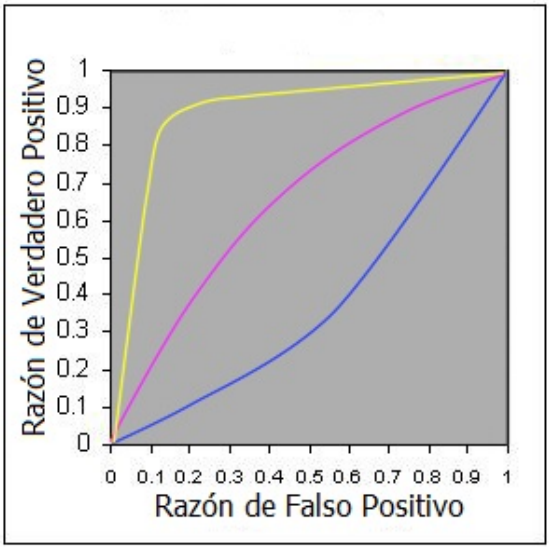


FIGURA 2.10: Gráfico de ejemplo de curva ROC [7].

2.5. Conclusiones

Para evaluar la clasificación es importante tener varias alternativas de medir el desempeño del clasificador ya que de esta forma se puede evaluar de forma más completa y conocer qué tan bien puede predecir las clases de los datos de entrenamiento y por lo tanto, qué tan bien se desempeñará con datos futuros que se presenten. El programa Weka evalúa sus clasificadores con estas medidas de desempeño, siendo la exactitud final el resultado en combinación de estos.

Capítulo 3

Bioinformática

La bioinformática trata de comprender procesos biológicos a través de distintas técnicas multidisciplinarias con las cuales se desarrollan herramientas para analizar, simular, almacenar e interpretar los datos [38].

¿Porqué es necesaria la bioinformática?, si bien el trabajo de analizar procesos biológicos puede ser realizado manualmente, como en un laboratorio, la gran motivación de unir la informática con la biología surge por la enorme cantidad de datos disponibles que se generan cada año a un nivel casi exponencial como lo muestra la Figura 3.1 y que se almacenan en bases de datos biológicas públicas.

La potencia de los computadores permite realizar un mayor nivel de análisis en menor tiempo, lo cual incentiva el desarrollo de aplicaciones y herramientas para resolver distintos problemas de la realidad. Un ejemplo de esto es el Protein Data Bank, el cual fue utilizado para obtener las estructuras atómicas de los complejos de proteínas para esta tesis.

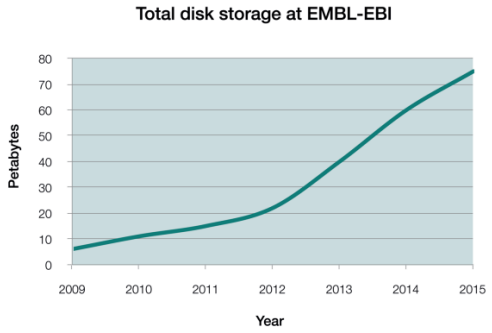


FIGURA 3.1: Gráfico de crecimiento de almacenamiento de datos biológicos [8].

Las IPP son un tema clave cuando se trata de describir las funcionalidades de las proteínas, ya que estas actúan y cooperan en conjunto. Dado esto, se clasifican grupos de proteínas que interactúan a conjuntos funcionales, designados a complejos físicos, vías de señalización o módulos estrechamente vinculados [8].

3.1. Proteínas

Los aminoácidos son moléculas orgánicas que se unen para formar cadenas llamadas péptidos o polipéptidos, que al superar cierta longitud, o su masa molecular supera cierto valor, y cuando su estructura tridimensional es estable, se les atribuye el nombre de proteína.

Las proteínas están codificadas en secciones del ADN llamadas genes. Para poder crear una proteína, una enzima llamada ARN-Polimerasa se mueve a lo largo del ADN creando una nueva cadena utilizando bases que se encuentran dentro del núcleo, ordenados según el código del ADN en un proceso llamado transcripción [9] como se puede apreciar en la Figura 3.2.

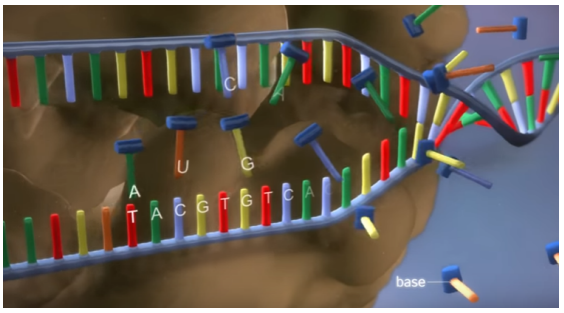


FIGURA 3.2: Representación de construcción de ARN mensajero [9].

La nueva cadena generada se llama ARN mensajero, y para que pueda ser utilizada como plantilla para crear una proteína, debe ser procesada, en donde sufre la eliminación de secuencias internas. Luego el ARN mensajero se mueve fuera del núcleo, al citoplasma de la célula.

Entonces, máquinas moleculares llamadas ribosomas (se les atribuye el nombre de máquinas por que producen movimientos casi mecánicos en respuesta a estímulos) leen el código en el ARN mensajero y cada 3 nucleótidos (codon) los ARN de transferencia son llamados para obtener los aminoácidos que portan para que el ribosoma vaya uniéndolos en una cadena.

Cuando el último aminoácido es añadido a la cadena, la cadena se pliega a si misma para formar la proteína [9].

Por lo tanto se pueden diferenciar 4 instancias de la estructura de las proteínas como se aprecia en la Figura 3.3.

La estructura **primaria** es la secuencia lineal de los aminoácidos que la componen; la estructura **secundaria** son sub-estructuras regulares definidas por patrones de los enlaces de hidrógeno, que pueden dividirse en hélices o hebras; la **terciaria** es una estructura tridimensional en donde las hebras y hélices se pliegan y se compactan en una estructura globular; la estructura **cuaternaria** se refiere a cómo las sub-estructuras interactúan entre ellas y se reordenan para formar un complejo.

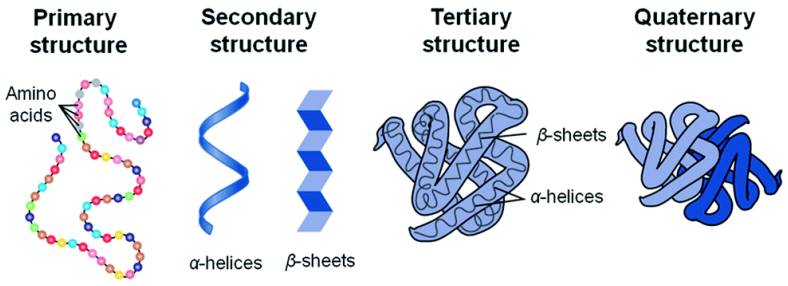


FIGURA 3.3: Lasa 4 estructuras de las proteínas [10].

3.2. Propiedades de las Proteínas

Las propiedades de las proteínas son características que le permiten a estas formarse, funcionar y existir [14]. A continuación se desriben 5 de ellas.

- **Especificidad**

  La función específica de las proteínas está determinada por su estructura primaria y conformación espacial. Lo cual significa que cualquier cambio en su forma representa una pérdida de su función.

- **Amortiguador de pH**

  Cuando las proteínas pierden su estructura tridimensional al romper sus enlaces se le llama desnaturalización. Puede ocurrir debido a cambios en la temperatura o pH. Por lo tanto debe ser capaz de mantener su pH regulado.

- **Solubilidad**

  La proteína debe ser soluble en medios, por lo tanto debe mantener presentes sus enlaces fuertes y enlaces débiles. La solubilidad se puede perder debido a efectos de la temperatura y cambios en el pH.

- **Estabilidad**

  Debe ser estable en el medio en donde desempeña su función. Para ello las proteínas crean un núcleo hidrofóbico empaquetado. Esto significa que repelen el agua y que evitan interactuar con otras proteínas que habitan el mismo medio acuoso, y por lo tanto evitan modificar su estructura.

3.3. Funciones de las Proteínas

De las funciones más importantes que se puede mencionar son:

- **Estructural:** las proteínas estructurales ayudan a mantener la forma de las células, como también formar tejido conjuntivo el cual conecta o separa diferentes tipos de tejido y órganos.
- **Catálisis:** son enzimas proteicas las cuales pertenecen a un grupo que son químicamente reactivas y funcionan como catalizador para reacciones químicas específicas en sistemas biológicos, como la pepsina que es una enzima digestiva que se encuentra en el estómago y que ayuda a digerir los alimentos.
- **Reguladoras:** las hormonas son proteínas que equilibran las funciones del cuerpo, como la insulina, la cual regula los niveles de azúcar en la sangre.
- **Transporte:** estas proteínas llevan sustancias hacia donde son requeridas como la hemoglobina la cual lleva el oxigeno por medio de la sangre.
- **Defensoras:** son las Proteínas encargadas de proteger el organismo. Entre sus distintas funciones se puede mencionar la queratina, la cual forma las capas externas de la epidermis y tejidos como uñas y pelo.

3.4. Interacciones de Proteínas

Las interacciones de las proteínas son muy importantes para los humanos y para otras formas de vida también, ya que participan en muchos procesos biológicos vitales, como por ejemplo: contracción muscular; señalización entre células, lo cual significa que forman

parte de las actividades básicas de las células y coordinan todas las acciones celulares; transporte celular; entre otras.

Además, la complejidad de los organismos no depende del número de proteínas codificadas en sus genomas, sino de el número de interacciones en ellas [39]. Por esta misma razón es que no se puede predecir los tipos de las interacciones sólo conociendo la estructura de las proteínas por si solas, por lo que deben ser estudiadas en conjunto. La interacción se realiza entre los átomos de las cadenas de aminoácidos que componen a las proteínas al hacer contacto.

En las interacciones ocurren fuerzas de atracción y repulsión las cuales ayudan a formar la proteína y a que esta no tome una forma que no debería, resultado en una estabilización entre estas fuerzas que definen el estado final de la proteína y su función específica [14].

Tipos de interacción

Las proteínas interactúan entre ellas con un amplio intervalo de afinidades y escalas de tiempo. Los tipos de interacción con los que se trabaja en este proyecto de título son las transientes y las permanentes.

Nooren y Thornton [40] realizaron una clasificación basados en el tiempo de vida de los complejos. Las interacciones permanentes son muy estables y por lo tanto sólo existen en su forma de complejo, en cambio la interacción transiente se asocia y disocia *in vivo*, lo cual significa que la interacción se rompe y se forma continuamente.

Debido a que las proteínas dependen de variables como las condiciones fisiológicas y el ambiente o medio en donde se encuentran, pueden ser transiente *in vivo* o ser permanente en otras condiciones. Por ende su relevancia biológica es necesaria para identificarlas, como por ejemplo las interacciones intercelulares de señalización se espera que sean transientes ya que su función requiere que se asocien y disocien.

Capítulo 4

Metodología y datos utilizados

La metodología a seguir se basa en la presentada por Gutiérrez-Bunster et al [11].

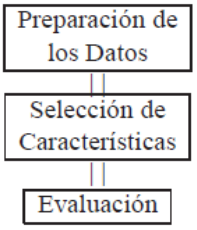
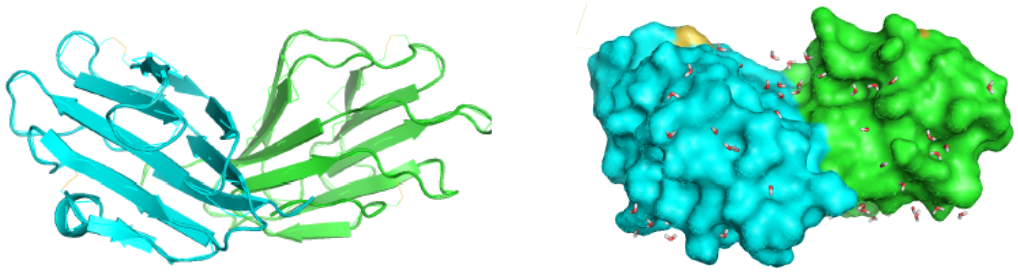


FIGURA 4.1: Metodología desarrollada [11].

Los datos de interacción proteína-proteína son obtenidos de bases de datos biológicas [41]. Como se menciona en Gutiérrez-Bunster et al [11], tras la investigación realizada por Jones [42], identifican que a partir de los residuos de la interacción, los complejos transientes contenían mayor residuos hidrofílicos en sus interfaces que en los complejos permanentes. Después se realiza la selección de características sobre los datos para generar los subconjuntos. Se registran los tiempos de ejecución de cada algoritmo con cada conjunto. Y luego en la evaluación se utiliza el programa Weka [28] para entrenar un modelo y clasificar los datos dividiéndolos en entrenamiento y pruebas, para finalmente evaluar los resultados obtenidos de cada clasificación.

4.1. Complejos

Cada complejo proteico se compone de 2 proteínas como se aprecia en la Figura 4.2.



(a) Cadenas del complejo (b) Superficie del complejo

FIGURA 4.2: Representación tridimensional de un complejo de proteínas extraido del Protein Data Bank [12].

Cuando las proteínas forman un complejo, una es llamada Ligando y la otra Receptor. Las proteínas interactúan entre sí al hacer contacto físico debido a la conformación preferida que las moléculas poseen como lo muestra la Figura 4.3.

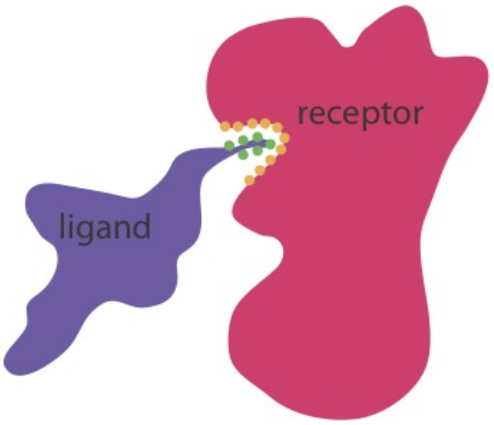


FIGURA 4.3: Representación de acoplamiento de proteínas [13].

4.2. Preparación de Datos

Los datos utilizados en este trabajo se título se componen de matrices de características de los complejos de las proteínas, que fueron generados en los trabajos previos de Gutiérrez-Bunster [11, 14] y el trabajo de Gutiérrez-Víctor [16]. Cada matriz posee 296 filas correspondiente a los ejemplos de los complejos proteicos y la cantidad de características están dadas según lo muestran la Tabla 4.1 y la Tabla 4.2.

4.2.1. Características de Complejos de Proteínas

Cada uno de los complejos de proteínas utilizados se encuentran previamente clasificados por el trabajo de Mintseris [43] y se considera sólo la zona de interacción, la cual es el área de contacto entre las dos proteínas.

Los tipos de energías de las matrices son las siguientes:

- **Energía electrostática**: es la energía de una colección de puntos que están cargados positiva o negativamente definido como un sistema, los cuales se atraerán o repelerán.
- **Energía de desolvatación:** en la formación de complejos de proteínas se remueven moléculas de agua de la región de su interfase, por ende, regiones de la superficie con una baja cantidad de energía requerida para remover el agua corresponden a sitios de interacción preferidos [44].
- **Energías libres de unión:** es la energía requerida para desmontar un sistema en partes separadas. Dependiendo de si el total de la energía del sistema es positivo se favorecerá la disociación del complejo, si es negativo entonces se favorecerá la formación del complejo [14].

A continuación se dan a conocer los nombres de las matrices utilizadas, se muestra la Tabla 4.1 que explica la composición de la matriz de datos principal, y luego en las otras matrices se explican los cambios que se le realizan.

DOC Top+-

Esta matriz es la más grande de las que fueron utilizadas para este trabajo. Contiene las mejores energías positivas y negativas encontradas en el trabajo de Gutiérrez-Bunster [14], como lo muestra la Tabla 4.1.

DOC Electrostatic Top+-

En el trabajo de Gokul [45] utilizan sólo energías electrostáticas como propiedades para la predicción de los tipos de las interacciones entre proteínas, para poder distinguir entre interacciones obligadas (proteínas que no pueden formar una estructura cristalina por si

Cadena	Tipo de energía	Características	Matrices	Total
Complejo	Energía de desolvatación			1
	Energía electrostática			1
	Energía libre de unión	2(E & R)	29(14+, 15-)	58
Ligando	Energía de desolvatación	2 (E & R)	21(10+, 11+)	42
	Energía electrostática	2 (E & R)	21(10+, 11+)	42
Receptor	Energía de desolvatación	2 (E & R)	21(10+, 11+)	42
	Energía electrostática	2 (E & R)	21(10+, 11+)	42
Receptor-Ligada	Contactos de energía electrostática	3 (E & R & R)	268(134+,134-)	804
	Contactos de energía libre de unión	3 (E & R & R)	268(134+,134-)	804
Clase				1
Total de características				1837

TABLA 4.1: Representación de los datos dentro de la matriz DOC top+- [14].

solas) y no-obligadas (interacciones que son formadas por proteínas que pueden formar una estructura cristalina por si solas). Proponen que como las interacciones electroestáticas tienen una mayor distancia de interacción y que poseen una mayor influencia en moléculas cargadas, pueden ser mejores candidatos para la predicción.

Entonces se propone utilizar este conjunto para explorar su potencial de predicción. Para construir este conjunto se tuvieron que extraer las energías de desolvatación y de energía libre del conjunto DOC top+-. En total, luego de extraer las características, el conjunto se redujo de 1837 características a 889.

DOC Top+- sin residuos

Esta matriz fue explorada y generada en el trabajo de Gutiérrez-Víctor [16] para probar nuevas opciones de conjuntos de datos utilizando la herramienta FastContact para sólo clasificar en base a las energías y no los residuos.

DOC Top– sin residuos

Esta matriz también fue generada en el trabajo de Gutiérrez-Víctor [16] a base de la está basada en la DOC top+- sin residuos pero con la diferencia que fueron consideradas sólo las energías negativas las cuales son las que contribuyen más en la interacción.

MatrizMag ó Top20+-

Está compuesta por la 20 mejores energías según su aporte en la interacción, para cada tipo de energía, esta matriz al igual que DOC top+- es calculada independientemente

por la aplicación FastContact, para obtener las mejores energías que aportan en la interacción como lo muestra la Tabla 4.2.

Cadena	Tipo de energía	Características	Matrices	Total
Complejo	Energía de desolvatación			1
	Energía electrostática			1
	Energía libre de unión	2(E & R)	40(20+, 20-)	80
Ligando	Energía de desolvatación	2 (E & R)	40(20+, 20-)	80
	Energía electrostática	2 (E & R)	40(20+, 20-)	80
Receptor	Energía de desolvatación	2 (E & R)	40(20+, 20-)	80
	Energía electrostática	2 (E & R)	40(20+, 20-)	80
Receptor-Ligada	Contactos de energía electrostática	3 (E & R & R)	40(20+, 20-)	120
	Contactos de energía libre de unión	3 (E & R & R)	40(20+, 20-)	120
Clase				1
Total de características				643

TABLA 4.2: MatrizMag ó Top20+- tipos de energías calculadas por FastContact para cada complejo [11].

Capítulo 5

Selección de características

La selección de características es una técnica empleada con el fin de reducir un conjunto o espacio de búsqueda para dejar como resultado un subconjunto que represente mejor al original, descartando características redundantes o con poco poder de predicción, esto además reduce los requerimientos de almacenamiento, tiempos de entrenamiento, utilización, y trata de sobrellevar el efecto Hughes para mejorar el rendimiento de la predicción [23].

El efecto Hughes [24] es que cuando las cantidades de características incrementan, los ejemplos requeridos para poder generalizar sobre el conjunto crece exponencialmente, lo cual para muchos algoritmos significa volverse completamente ineficientes, como los que iteran muchas veces sobre el conjunto. Intentar entrenar un clasificador con grandes cantidades de datos o conjuntos sin procesar resulta en un clasificador con una capacidad de predicción reducida, ya que el algoritmo del clasificador tratará de aprender sobre datos que tal vez sean redundantes [46].

Para entender cómo funcionan los algoritmos de selección de características se revisaron varias referencias bibliográficas.

5.1. Tipos de algoritmos de selección de características

Existen 3 enfoques para los algoritmos de selección de características los cuales son Filtro (Filter), Envoltorio (Wrapper) y Embebido (Embedded), los cuales, dependiendo de su funcionamiento se revisará su eficiencia en base a su capacidad para trabajar con grandes cantidades de características y si son capaces de seleccionar características relevantes, o de forma contraria, descartar las características más redundantes.

5.1.1. Filtro (Filter)

Los algoritmos de tipo filtro se caracterizan por no ser de tipo inductivo sino que de ser un pre-procesamiento de los datos. Son más rápidos que los wrapper puesto que no buscan un conjunto que optimice la función del clasificador, aunque se puede utilizar un clasificador simple como forma de evaluación del conjunto. Una de las observaciones que se puede hacer al utilizar este tipo de algoritmos es que si los datos seleccionados alcanzan un buen nivel de exactitud al entrenar un clasificador, quiere decir que los datos ya poseían un buen nivel de predicción y requerían ser estandarizados [23].

5.1.2. Envoltorio (Wrapper)

Para este trabajo se prefiere utilizar la palabra en inglés Wrapper debido a que es ampliamente utilizada en vez de su traducción. Se utiliza un algoritmo de aprendizaje para calificar a subconjuntos de características según su poder predictivo [23]. Ven al algoritmo de aprendizaje como una caja negra para evaluar los resultados. Suelen tener mejor desempeño que los de tipo filtro pero a un mayor costo computacional.

5.1.3. Embebido (Embedded)

Los algoritmos embebidos realizan la selección de características en el proceso de entrenamiento [23]. Se suele utilizar el criterio de qué tan bien se ajustan las observaciones al modelo estadístico. Además suelen tener aplicar penalizaciones cuando el conjunto es muy grande.

5.2. Revisión de Algoritmos de Selección de Características

Se revisó la compilación de algoritmos de estrategia completa realizada por Molina [15], para poder comprender el funcionamiento de distintos algoritmos de selección de características.

La estrategia completa se refiere a que estos algoritmos utilizan todo el espacio de búsqueda aunque hay algunos que permiten que el usuario ingrese un parámetro para detener la selección en algún punto para así ahorrar tiempo si resulta ser muy extensa o las características satisfacen cierta medida.

Junto a esta compilación, se revisaron los algoritmos de selección de características de un repositorio open-source para investigaciones y estudios comparativos, llamado Scikit-Feature [47, 48], de donde se exploraron otros algoritmos de selección de características y que fueron añadidos a la compilación de la Tabla 5.1.

Medida de evaluación	Algoritmos de estratégia completa
Distancia	B&B [49] BF [50]
Información	MDLM [51] CARDIE [52] ReliefF [53] CIFE [54]
Dependencia	Bobrowski [55] CFS [56] t score [57] low variance[57]
Consistencia	AB&B [58] Focus [59] Schlimmer [60] MIFES-1 [61]
Exactitud	BF [62] AMB&B [63] IchiSkla [64] DavRus [65]
Distancia + Exactitud	Oblivion [66]

TABLA 5.1: Tabla de algoritmos de selección de características de estrategia completa [15]

A continuación se describen cada uno de ellos según su funcionamiento y el criterio que utilizan para realizar la selección de las características. En la Tabla 5.1 se encuentra un resumen de los algoritmos revisados, separados por el tipo de medida de evaluación.

5.2.1. Branch and Bound (B&B)

El algoritmo B&B se caracteriza por ser una variación de la búsqueda en profundidad, la cual se basa en recorrer los nodos de un árbol avanzando 1 nivel por cada nodo hasta llegar al último. B&B uno de los más antiguos y de los más usados el cual varía dependiendo del criterio de evaluación o enfoques que se apliquen. Se emplea un árbol de subconjuntos de características, siendo el primer nodo el nivel 0, el que posee N características, y los nodos hijos del nivel 1, el mismo conjunto de características del primer nodo menos una característica y restando de la misma forma para los niveles menores hasta llegar al último nivel lo cual se llama selección secuencial hacia atrás (Sequential Backward Selection).

Para la correcta implementación de este algoritmo, el árbol no debe ser simétrico, debe ser generado de un lado hacia otro (Ej: derecha a izquierda) y comenzar la búsqueda por el lado más pequeño. Entonces, en la primera búsqueda se hace en profundidad por la rama del árbol que tiene menos hijos y al final, cuando se obtiene un subconjunto de características de tamaño m (que es definido por el usuario), se compara con el primer nodo de características de las ramas no exploradas. Si el subconjunto obtenido es mejor que con el que se desea comparar (según el criterio/medida de evaluación, el cual debe ser monótona, lo que quiere decir que sólo el criterio elegido debe ser admisible), entonces la rama del nodo comparado se corta y se repite el proceso. Si se encuentra un subconjunto mejor, entonces se explora la rama nueva y se repite el proceso hasta acabar con todas las ramas [49].

5.2.2. Best First (BF)

BF o Primero el mejor, propone una búsqueda informada la cual reduce la cantidad de subconjuntos que son evaluados en el algoritmo Branch and Bound, garantizando también el mejor subconjunto pero siendo menos exhaustivo. A diferencia de Branch and Bound, expande los nodos que son más prometedores por cada nivel que va avanzando hasta llegar a un subconjunto óptimo [50].

5.2.3. Minimum Description Length Method (MDLM)

MDLM o Método de longitud de descripción mínima, es un algoritmo que detecta características que son inservibles, de forma que no se tengan que utilizar para el clasificador y como resultado mejorar el rendimiento de este, ahorrar mediciones y tiempo computacional. Se basa en un criterio teorético de información en el cual se definen las características inservibles y luego se construye un modelo probabilístico [51].

5.2.4. CARDIE

CARDIE es un algoritmo creado para un sistema de procesamiento de lenguaje natural el cual se emplea para procesar textos y realizar un análisis semántico en caso de encontrar una palabra que el sistema desconoce. Para ello se dividen las características en 3 grupos: características de definición de palabras, características de contexto local y características de contexto global. Utiliza árboles de decisiones, aprendizaje basado en casos y un enfoque híbrido mezclando ambos [52].

5.2.5. ReliefF

El algoritmo ReliefF se basa en la medida de evaluación de distancia y calcula la norma de manhattan que se puede apreciar en la Ecuación 5.1 en donde $\boldsymbol{p}$ y $\boldsymbol{q}$ son dos vectores de características para determinar la similaridad entre ellas [48, 53].

$$d_1(p,q) = ||p-q||_1 = \sum_{n=1}^{n} |p_i - q_i| \tag{5.1}$$

5.2.6. Conditional Infomax Feature Extraction (CIFE)

El algoritmo CIFE busca maximizar la información entre las características y la clase en un proceso de aprendizaje de características para reducir las que son poco útiles y puede ser utilizado para problemas de múltiples clases. Estima la información mutual discreta de las características con respecto a la clase. Se basa en que mientras más sepamos sobre las características, se podrán clasificar con mayor exactitud [54].

5.2.7. Bobrowski (BOBRO)

BOBRO se basa en un coeficiente de homogeneidad entre las características, para lo cual utiliza un hiper-plano y una función de criterio. De forma que se plantea si el criterio

puede ser usado para examinar la pregunta de, qué tan bien el conjunto de datos puede encajar en un hiper-plano. Al igual que los otros algoritmos, Bobrowski puede omitir mediciones que son linealmente dependiente de otras, y además al tener un coeficiente de mono-tonicidad (regla de inferencia) se puede encontrar un subconjunto óptimo sin una búsqueda exhaustiva [55].

5.2.8. Correlation based Feature Selection (CFS)

CFS o Selección de característica basado en correlación, es un algoritmo filtro que categoriza (Ranking) los subconjuntos de características de acuerdo a una correlación basada en una función heurística. El criterio de evaluación se inclina hacia subconjuntos que contienen características que están altamente correlacionadas con la clase y no correlacionadas con cada una. Se ignoran los características irrelevantes ya que poseen bajo nivel de correlación con la clase. Si los características están fuertemente correlacionadas con unos a otros entonces son sacados fuera por ser redundantes. La aceptación de una característica dependerá entonces si es capaz de predecir la clase en áreas del espacio que no hayan sido predichas por alguna otra característica [56].

5.2.9. T score

El algoritmo de T score utiliza una métrica de la desviación estándar que por cada característica debe calcular individualmente la Ecuación 5.2 para poder compararlos con las demás características y en base a los valores obtenidos realiza ranking. Calcula el promedio (mean) de los valores de las características para cada clase y luego la desviación estándar para las dos clases (std) y el número de cada clase correspondiente (clase A es n1 y clase B es n2) [57].

$$tscore = |mean1 - mean2|\sqrt{((std1^2)/n1) + ((std2^2)/n2))} \tag{5.2}$$

5.2.10. Low Variance

El algoritmo de baja varianza es de tipo estadístico y calcula la media de cada característica para obtener una varianza promedio y a partir de esto calcula la varianza para cada característica, para luego filtrar las que posean varianza mayor al límite de la media de varianza. De esta forma, si las características poseen baja varianza, es indicio de que son similares y son más fácil de generalizar [57].

5.2.11. Automatic Branch and Bound (AB&B)

Automatic Branch and Bound es una variación del algoritmo Branch and Bound que como su nombre indica es automático, debido a que no es necesario especificar el tamaño de un subconjunto deseado (llamado m en B&B) ya que al final de la búsqueda entrega los subconjuntos de menor cardinalidad (cantidad de características) como los subconjuntos óptimos. Para esto se emplea una medición de inconsistencia (denotado como U) con la cual se comparan 2 subconjuntos y uno es preferido sobre el otro dependiendo la razón de inconsistencia, la cual prueba ser monotónica [58].

5.2.12. Focus

Focus es un algoritmo a favor de la simplicidad ya que se basa en la idea de características mínimas, lo que quiere decir que si dos funciones son consistentes con los ejemplos de entrenamiento, se debe preferir la función que involucra menos características de entrada. Realiza una búsqueda exhaustiva en anchura para medir si existen ejemplos en la muestra que coincidan con todas las características en un subconjunto A dado. Además que coincida con la clase, de forma que sólo las características en A sean relevantes con cada concepto encontrado consistente con los ejemplos. Este algoritmo es prohibitivo si es que no se puede reducir el número de características relevantes [59].

5.2.13. Schlimmer

Schlimmer es un algoritmo similar a Focus y busca determinaciones cuyos dominios son de tamaño menor o igual a una variable k, en donde k es un parámetro entregado por el usuario. También deben indicarse ciertas reglas de poda de las ramas del árbol. Realiza una búsqueda sistemática basada en cierta heurística. Comienza con un estado inicial vacío y va añadiendo propiedades hasta satisfacer cierto numero k, o todas las propiedades han sido añadidas, o se ha generado una excepción dentro de las reglas de poda [60].

5.2.14. MIFES-1

MIFES-1 es un algoritmo el cual selecciona un conjunto de características de mínima cardinalidad (menor cantidad de elementos) que sea suficiente para clasificar todos los ejemplos presentados en el conjunto de entrenamiento. Se emplea una matriz en donde, cada columna corresponde a un ejemplo positivo en el conjunto de entrenamiento y cada fila corresponde a un ejemplo negativo en el mismo conjunto. Luego se considera que

una característica la cual tendrá un valor 1 para algunos ejemplos y 0 para otros en el conjunto de entrenamiento. Entonces, una característica cubre a un elemento de la matriz si adopta valores opuestos para una instancia positiva y otra negativa [61].

5.2.15. Beam Search (BS)

BS o Búsqueda en haz, es un algoritmo de tipo de búsqueda Best First el cual utiliza una cola limitada para reducir el alcance de la búsqueda. Los estados de la cola son ordenados de mejor a peor, teniendo en cuenta que los mejores van al inicio. El algoritmo toma un estado del frente de la cola y extiende la búsqueda desde ese estado. Por cada nivel avanzado se va añadiendo una característica. Cuando se extiende un estado, se reemplaza la cola con los nuevos valores. La cola debe ser limitada, de lo contrario la búsqueda sería exhaustiva. Si la cola es de tamaño 1 entonces la búsqueda es secuencial [62].

5.2.16. Approximate Monotonic Branch & Bound (AMB&B)

El algoritmo B&B de aproximación monotónica, es una mejora del algoritmo Branch and Bound debido a que permite el uso de funciones de evaluación no monotónicas o clasificadores. Mejora el desempeño de clasificadores al remover características irrelevantes o ruidosas(información no útil). Se emplea una variable de tolerancia la cual define el límite de la búsqueda. De forma que, si en Branch and Bound se llega a un conjunto de características con cierto porcentaje de error, y luego se quiere seguir expandiendo el nodo, pero el siguiente nodo posee mayor porcentaje de error entonces la rama se corta, en cambio en AMBB si el porcentaje de error del siguiente nodo es menor a la variable de tolerancia, entonces se continúa expandiendo el nodo [63].

5.2.17. IchiSkla

IchiSkla es un algoritmo el cual emplea un patrón de clasificación llamado clasificador de caja, el cual es un clasificador cuya superficie de decisión es un paralelepípedo rectangular donde en cada cara es perpendicular a una característica en eje de coordenada. Se utiliza Branch and Bound como criterio de evaluación, y también emplean la técnica de volver hacia atrás (backtrack) [64].

5.2.18. DavRus

DavRus se basa en FOCUS y explica que encontrar determinaciones con la menor cardinalidad es mas fácil en la práctica que encontrar determinaciones mínimas. Al procesar

los ejemplos, usa una lista L de todos los posibles conjuntos de características S, de la menor posible cardinalidad n (ya que utiliza un enfoque de mínima cardinalidad), de forma que cualquier ejemplo visto que coinciden en las características en S tienen valores de característica de salida idénticas, entonces se van eliminando conjuntos que no clasifican correctamente los ejemplos utilizados, y si la lista se vacía, entonces incrementa la variable n (cardinalidad) en 1 hasta llegar a la lista con todos las características, lo cual correspondería al peor caso [65].

5.2.19. Oblivion

Oblivion es un algoritmo basado en Branch and Bound y Schlimmer el cual emplea una variación de un árbol de decisión, llamado árbol de decisión olvidadizo el cual, todos los nodos del mismo nivel prueban la misma característica. Para construir uno, se puede utilizar ambas técnicas, ya sea empezar con un conjunto vacío de características o empezar con todas las características e ir podando o utilizando la técnica de eliminación de volver atrás para remover las características que no ayuda a la precisión de la clasificación. La ventaja es que la precisión baja sustancialmente si uno remueve una sola característica relevante, incluso si interacciona con otras características pero permanece inafectada cuando uno poda una característica redundante [66].

5.3. Análisis, elección de los algoritmos de selección de características a utilizar y conclusiones

Varios de los algoritmos mostrados en la Tabla 5.1 fueron diseñados para trabajar con conjuntos pequeños (menos de 30 características) en comparación a los que se desea trabajar hoy (en este caso, 1800 características), por lo tanto es necesario descartar los que sean prohibitivos en términos de tiempo computacional.

Focus es un algoritmo basado en un criterio llamado MIN-FEATURES, el cual intenta encontrar un subconjunto óptimo usando la mínima cantidad de características de entrada, en cambio en este trabajo se desea encontrar el mejor subconjunto, por lo tanto queda descartado debido a esta limitante. Lo mismo aplica para los que se basan en Focus como Schlimmer y DavRus.

Como se dijo antes, los algoritmos exhaustivos que tengan un tiempo computacional muy elevado como los de orden exponencial como B&B quedan descartados; en cambio como se ha mostrado en la descripción de los algoritmos de la Tabla 5.1, existen enfoques distintos y variaciones para el algoritmo B&B tales como AB&B y AMB&B los cuales

emplean técnicas de búsqueda informada. Pero quedan descartados porque además de ser casi exponencial, se prefieren algoritmos que reduzcan las variables poco útiles en una etapa preliminar, no que busquen exhaustivamente sobre características que puedan ser redundantes.

CARDIE es un algoritmo que se basa en diferenciar las características en distintos grupos para realizar un análisis semántico para ayudar a ubicar el sentido de las palabras desconocidas en textos, lo cual no se aplica muy bien al tipo del problema que se quiere abordar en este trabajo.

Mifes-1 es un algoritmo que optimiza una función de utilidad, como la de cardinalidad. En cambio, este algoritmo indica que tal vez falle al intentar encontrar características útiles si es que la información de clasificación está correlacionada entre las características, por lo tanto queda descartado debido a tal posible fallo.

En Shardlow [67] realizan un análisis sobre varios algoritmos de selección de características como Naive Forward, Ranked Forward, Refined Exhaustive y Control, para lo cual utilizan un clasificador SVM simple, con el fin de ver la exactitud que poseen. Luego de los experimentos, se concluye que entre los algoritmos el mejor de todos era Ranked Forward Search debido al porcentaje de exactitud encontrado y por la reducción de la dimensión del espacio alcanzado. Un aspecto interesante del experimento realizado es que de un total de 325 características del conjunto, los algoritmos alcanzaron un porcentaje de exactitud alto con menos de 33 características seleccionadas, esto se debe a la existencia de características irrelevantes o redundantes.

Ranking [23] es una técnica empleada para poder obtener el subconjunto con mayor poder de predicción. Se tiene cierto subconjunto generado por un algoritmo de selección de características de unas 250 características, pero tal vez puede ocurrir que el 50 % de las características tengan un gran valor de predicción y el restante quizá no es tan bueno en comparación al otro 50 %, entonces lo que se puede hacer es ordenarlos por su poder de predicción y seleccionar por ejemplo las 100 mejores. Aunque también puede ocurrir que haya características que tienen exactamente el mismo poder de predicción y que el incluir una o ambas al conjunto final no mejore los resultados de la clasificación, por lo tanto una es removida (vector tunning operations [68]).

Tras analizar los algoritmos, se decidió optar por aquellos de tipo filtro debido a que sus evaluaciones son más simples que los de tipo wrapper y por el hecho que los de tipo wrapper funcionan mejor con menor cantidad de características y con una mayor cantidad de ejemplos.

Por ejemplo, los algoritmos basados en mediciones estadísticas como varianza y desviación estándar, sirven para poder seleccionar las características las cuales se encuentran

dentro de ciertos parámetros, por ejemplo, el algoritmo de baja varianza, calcula la varianza media de las características y en base al valor obtenido, filtra las características de alta varianza.

De igual forma, el algoritmo T score, asigna un valor a las características basado en métricas estadísticas como desviación estándar y media. Ambos algoritmos aseguran recorrer todo el conjunto y la función criterio para decidir qué características seleccionar es simple, por lo que los tiempos de ejecución son mínimos.

Otro tipo de algoritmos que aseguran tiempos tiempos de ejecución bajos son los que exploran el espacio de búsqueda como un conjunto de puntos en un plano, en donde se realizan mediciones de distancia entre puntos, como la forma en que trabaja el algoritmo RealiefF, que además se limita a medir los 5 puntos vecinos por cada característica.

Algoritmos que maximicen la información al tratar de reducir la redundancia de las características o de la correlación entre las mismas características y no la clase, pueden tardar más que los algoritmos estadísticos, puesto que tratan de realizar la búsqueda de características basadas en una función criterio que considera un conjunto de características, y a medida que se va avanzando la búsqueda se va añadiendo características y descartando otras, o se van tomando características hasta satisfacer una medida o límite. Por ende una solución filtro como CIFE y CFS, es elegida, ya que como su evaluación puede ser costosa en tiempo, utilizan técnicas para ahorrar mediciones al considerar parámetros limitadores.

Capítulo 6

Implementación y ejecución de algoritmos de selección de características

6.1. Implementación de los algoritmos de selección

A continuación se muestran los algoritmos seleccionados. Además se explica brevemente su funcionamiento y funciones criterio, junto con un pseudocódigo explicando cómo funcionan, para lo cual algunas partes fueron resumidas pero son explicadas a detalle en la descripción de los algoritmos de selección de características.

6.1.1. Baja varianza

Este algoritmo se basa sólo en las varianzas de los datos de los ejemplos por cada característica y los filtra si las varianzas son menores a la varianza promedio de todas las características. Este algoritmo no efectúa evaluaciones sobre el poder de predicción de los datos a diferencia del resto de los algoritmos, no incluye una clasificación para comprobar cual subconjunto generado es mejor que los demás. Para esto tiene que calcular la media de los valores de cada característica y luego la media promedio entre todas las características. Luego se resta la media promedio a todas las características, se eleva al cuadrado para obtener la varianza y se hace lo mismo con todas las características. Luego se obtiene la varianza promedio de cada característica. A partir de esta se calcula la varianza promedio de todas las varianzas para poder comparar. Luego se construye un nuevo conjunto con las características que posean varianza menor a la varianza promedio general.

Algoritmo 1 Baja varianza

Entrada: Matriz de características X de complejos proteicos.
Salida: SubMatriz x de características de los complejos proteicos ordenada por ranking.

```
1: nFilas = dimensión de X
2: nColumnas = dimensión de X[0]
3: para i = 0 hasta nColumnas hacer
4:    para j = 0 hasta nFilas hacer
5:       Calcular media para todas las características de X[j][i].
6:    fin para
7: fin para
8: Obtener media general de todas las medias de las características.
9: para i = 0 hasta nColumnas hacer
10:   para j = 0 hasta nFilas hacer
11:      Calcular diferencia entre los valores de la matriz: M[j][i] - media general.
12:      Elevar al cuadrado el valor de la diferencia para obtener su varianza.
13:   fin para
14: fin para
15: varianza media general = 0
16: para i = 0 hasta nColumnas hacer
17:   para j = 0 hasta nFilas hacer
18:      Sumar los valores de varianza correspondientes a cada característica y dividirlo
         por nColumnas para obtener la varianza media de la característica.
19:
20:   fin para
21:   Añadir a una nueva matriz V el número de la posición de cada característica y su
      valor de varianza.
22:   Sumar el valor de varianza media al valor de la varianza media general.
23: fin para
24: Dividir el valor de la varianza media general en nColumnas.
25: Ordenar la matriz V de menor a mayor valor de varianza.
26: para i = 0 hasta nColumnas hacer
27:   si El valor de varianza de V[1][i] es menor a la varianza media general entonces
28:      Añadir V[0][i] a una lista L.
29:   fin si
30: fin para
31: Se instancia una nueva matriz m basándose en los datos de la matriz M con los
    índices de columnas en la lista L.
32: devolver Matriz m.
```

6.1.2. T score

Este algoritmo al igual que los otros, exceptuando a baja varianza, utiliza la librería de scikit [47] para implementar un clasificador SVM lineal con el cual poder evaluar los subconjuntos generados al utilizar validación cruzada de 10 pliegues aleatoria, por lo que puede suceder que no siempre aparezca el mismo subconjunto con un alto poder predictivo. Calcula la media de las clases para los valores de cada característica, luego estima la desviación estándar para las 2 clases y finalmente calcula el T score utilizando estos datos y los guarda en un vector, para luego ordenarlos en forma descendiente.

Algoritmo 2 T score

Entrada: Matriz de características X de complejos proteicos.
Salida: SubMatriz x de características de los complejos proteicos ordenada por ranking.

1: nFilas = dimensión de X
2: nColumnas = dimensión de X[0]
3: Inicializar vector F de tamaño nColumnas, con ceros
4: Generar rangos de valor en un arreglo S para el conjunto de entrenamiento y prueba para 10 iteraciones de validación cruzada.
5: **para** cada valor de entrenamiento y prueba en S **hacer**
6: **para** $i = 0$ hasta nColumnas **hacer**
7: Crear vector f con todas los datos de X para la característica número i
8: Calcular la media para todos los valores de la clase 1 y clase 2 de la característica i
9: Calcular la desviación estándar D1 de los valores para la clase 1 y D2 para la clase 2
10: t = media de la clase 1 - media de la clase 2
11: t0 = División de $D1^2$ por la cantidad de instancias para la clase 1
12: t1 = División de $D2^2$ por la cantidad de instancias para la clase 2
13: F[i] = t / $\sqrt{(t0 + t1)}$
14: **fin para**
15: Almacenar en un vector IDX con los índices de las características en órden descendiente según su valor en F
16: Instanciar una sub matriz SF de la matriz X con los índices de las primeras 100 características en IDX
17: Utilizar SF para entrenar un clasificador SVM lineal con los valores de entrenamiento.
18: Utilizar el clasificador generado para predecir las clases de las instancias, con los valores de prueba.
19: Calcular el porcentaje de exactitud de la predicción
20: **si** si la exactitud de la predicción es mayor a la anterior **entonces**
21: Almacenar el conjunto SF
22: Utilizar el nuevo valor de predicción para comparar
23: **fin si**
24: **fin para**
25: **devolver** Matriz SF

6.1.3. CIFE

Este algoritmo utiliza una función de criterio llamada infomax. Para esto utiliza 4 vectores F, t1, t2 y t3. En el vector t1 almacena el resultado de la estimación de la información mutual discreta de los ejemplos de cada característica con respecto a su clase. Toma la característica con el mayor valor de información mutual de t1, añade su índice a F y comienza un ciclo hasta que se hayan añadido las características límite a F, las cuales para este proyecto de título fueron 100. Revisa todas las características y si no se encuentran en F calcula la información mutual discreta entre la última característica añadida a F (que posee el mayor valor de información mutual discreta) y suma el valor de la estimación al valor antiguo en t2 (que inicialmente es 0), luego calcula la información mutual discreta en 3 dimensiones considerando la característica que se está evaluando, la última característica evaluada y la clase. Luego a una variable t le asigna el valor de el valor de t1[i] que se está evaluando, menos el valor de t2[i] multiplicado por una variable beta que en este caso es 1 así que no significa ninguna diferencia, más el valor de t3[i] multiplicado por una variable gamma también posee valor 1 par este caso. Finalmente se evalúa si el valor encontrado es mayor al último valor encontrado, si lo es entonces se almacena y se continúa hasta conseguir el mayor valor, se agrega F y se continúa buscando hasta que el tamaño de F sea 100.

Algoritmo 3 CIFE

Entrada: Matriz de características X de complejos proteicos.
Salida: SubMatriz x de características de los complejos proteicos ordenada por ranking.

1: nFilas = dimensión de X
2: nColumnas = dimensión de X[0]
3: Generar rangos de valor en un arreglo S para el conjunto de entrenamiento y prueba para 10 iteraciones de validación cruzada.
4: **para** cada rango de valor de entrenamiento y prueba en S **hacer**
5: Utilizar la función de criterio de calificación de combinación lineal con el conjunto de entrenamiento para un límite de 100 características y generar un arreglo IDX con los índices de las características.
6: Generar una submatriz x de la matriz de entrada X, utilizando los índices en IDX.
7: Entrenar un clasificador SVM Lineal con la matriz x utilizando los rangos de valor para entrenamiento
8: Utilizar el clasificador generado para predecir la clase del conjunto de prueba.
9: Calcular el porcentaje de exactitud de la predicción.
10: **si** la exactitud es mayor que la anterior **entonces**
11: SF = subconjunto x.
12: Almacenar el valor de la exactitud para comparar con la siguiente iteración.
13: **fin si**
14: **fin para**
15: **devolver** Matriz SF

6.1.4. CFS

Este algoritmo utiliza un criterio de correlación como heurística. Comienza explorando el espacio de características, comprueba si el índice de una característica no está en un vector F, entonces lo añade. Luego calcula el mérito para todas las características en F.

Para calcular el mérito de las características de F, se deben explorar las características y por cada una se calcula su incerteza simétrica lo cual se estima con la ganancia de información y estimadores de la entropía, luego se suma al valor de una variable rcf, después se calcula la suma de la incerteza simétrica de las características que vienen después de la característica que se está evaluando y se le asigna el valor a la variable rff. Entonces para obtener el mérito se calcula $rcf/\sqrt{features + rff})$.

Se compara el mérito obtenido con el antiguo mérito mayor y si es mayor se guarda el valor de mérito y el índice de la característica revisada. Finalmente se almacena en F el índice de la característica explorada y el mérito obtenido en un vector M. Entonces se calcula si es que el tamaño del vector M es mayor a 5, entonces si el valor del mérito va en orden descendente, si lo está, entonces regresa el vector F.

Algoritmo 4 CFS

Entrada: Matriz de características X de complejos proteicos.
Salida: SubMatriz x de características de los complejos proteicos ordenada por ranking.

```
nFilas = dimensión de X
nColumnas = dimensión de X[0]
Arreglo F = []
Arreglo M = []
merito = -100000000000
idx = 0
Generar rangos de valor en un arreglo S para el conjunto de entrenamiento y prueba para 10 iteraciones de validación cruzada.
para cada rango de valor de entrenamiento y prueba en S hacer
    mientras 1 == 1 hacer
        para I = 0 hasta nColumnas hacer
            si i no esta en F entonces
                Añadir i a F
                t = Cálculo del mérito de las características con índices en F.
                si t es mayor al último mérito calculado entonces
                    Hacer t el nuevo mérito
                    idx =i
                fin si
                Extraer i de F.
            fin si
        fin para
        Añadir idx a F
        Añadir merito a M
        si Largo de M ¿5 entonces
            si M[tamaño de (M)-1] <= M[tamaño de (M)-2] entonces
                si M[tamaño de (M)-2] <= M[tamaño de (M)-3] entonces
                    si M[tamaño de (M)-3] <= M[tamaño de (M)-4] entonces
                        si M[tamaño de (M)-4] <= M[tamaño de (M)-5] entonces
                            Salir del ciclo while.
                        fin si
                    fin si
                fin si
            fin si
        fin si
    fin mientras
    Extraer las 100 mejores características de X utilizando los valores de índices de F.
    Entrenar un SVM lineal con los rangos de entrenamiento de las características seleccionadas.
    Utilizar el clasificador para predecir las clases y generar un vector de los resultados.
    Comparar resultados para calcular el porcentaje de exactitud.
    si el porcentaje de exactitud es mayor al anterior entonces
        SF va a ser una matriz igual a las características seleccionadas.
        Se reemplaza el último porcentaje de exactitud por el reciente.
    fin si
fin para
devolver Matriz SF
```

6.1.5. ReliefF

Relieff trabaja con una función criterio de distancia. Toma la matriz de entrada sin las clases y calcula la matriz de distancia, obteniendo así una matriz D, teniendo que D[i][j] es la distancia entre el vector número i y el vector número j de la matriz X.

Por cada característica añade a un vector de distancias ordenadas todas las distancias correspondientes con todas las otras características.

Luego si la distancia ordenada de las características posee la clase de la característica que se está revisando añade el índice de la característica a un vector de aciertos más cercanos y si la cantidad de aciertos es igual a 5 entonces añade la característica a un vector de

revisados. Si no es de la misma clase entonces encuentra los fallos más cercanos por cada clase. Luego de obtener todos los aciertos cercanos y fallos cercanos de las clases, calcula la calificación de Relieff y devuelve los índices de las características ordendas según su calificación en orden descendiente.

Algoritmo 5 ReliefF

Entrada: Matriz de características X de complejos proteicos.
Salida: SubMatriz x de características de los complejos proteicos ordenada por ranking.

nFilas = dimensión de X
nColumnas = dimensión de X[0]
k = 5
Vector W de tamaño nColumnas
Generar rangos de valor en un arreglo S para el conjunto de entrenamiento y prueba para 10 iteraciones de validación cruzada.
distancia = Resultado del cálculo de distancia de pares entre instancias utilizando el criterio de la distancia de manhattan.
para cada rango de valor de entrenamiento y prueba en S **hacer**
    para $i = 1$ hasta nColumnas **hacer**
        Seleccionar aleatoriamente una instancia R_i
        Encontrar los k puntos más cercanos y añadir a vector H
        para cada clase C distinta de R_i **hacer**
            Encontrar los k fallos más cercanos de la clase C y añadir a vector M.
        fin para
        para $a = 1$ hasta nColumnas **hacer**
            para $j = 1$ hasta k **hacer**
                dif1 = diferencia entre las instancia a, R_i y H_j divididos por $(nColumnas * k)$
            fin para
            C es la clase contraria de R_i
            para $x = 1$ hasta k **hacer**
                dif2 += diferencia entre las instancia a, R_i y M_x divididos por $(nColumnas * k)$
            fin para
            dif2 = $dif2 * \frac{P(C)}{1-P(clase(R_i))}$
            W[a] = W[a] - dif1 + dif2
        fin para
    fin para
    Generar un arreglo IDX con los valores de W ordenados en forma descendiente.
    SF es una submatriz de X con los primeros 100 índices de IDX
    Entrenar un clasificador SVM lineal con los rangos de entrenamiento con el conjunto SF
    Predecir las instancias de SF con los rangos de índices de entrenamiento
    Estimar la exactitud de la predicción
    si la exactitud obtenida es mayor a la anterior **entonces**
        Instanciar una matriz m con los datos de la submatriz SF.
        Almacenar la exactitud obtenida para comparar con la siguiente.
    fin si
fin para
devolver Matriz m

6.2. Conclusiones

El algoritmo de baja varianza fue implementado con sólo la función criterio de varianza para evaluar las características sin considerar su clase para motivo de prueba en comparación con otro algoritmo de tipo estadístico como T score el cual además como se encuentra dentro de los algoritmos de la compilación de Scikit-Feature [48] su implementación incluyó un clasificador SVM lineal para evaluar los subconjuntos de entrenamiento y pruebas generados, los cuales fueron seleccionados aleatoriamente, lo que quiere decir que si bien el algoritmo puede conseguir cierto porcentaje de exactitud, puede que le

vaya mejor una segunda vez pero no se puede tener la certeza de cuantas veces sería necesario; lo mismo aplica para los siguientes algoritmos.

Capítulo 7

Resultados y Discusión

7.1. Resultados de la clasificación

Para representar las técnicas de evaluación del entrenamiento del clasificador se utilizará las siguientes siglas junto al nombre de cada algoritmo para cada conjunto. split90 se refiere a división por porcentaje de 90 %, de la misma forma, split66 se refiere a una división por porcentaje de 66 %, cross10 significa una validación cruzada de 10 particiones, y cross15 una validación cruzada de 15 particiones.

7.1.1. Clasificación de conjuntos antes de la selección de características

Se realizó una clasificación con los conjuntos originales para probar cuánto porcentaje de exactitud se puede alcanzar sin realizar una selección de las características.

Como lo muestra la Tabla 7.1, alcanzó un máximo de 90 % de exactitud lo cual es bastante bueno considerando sólo ese valor, en cambio al ver el resto de los resultados, en promedio suelen ser bajos. Lo ideal es que se presente en varias instancias valores altos de forma que en promedio el conjunto tenga un buen desempeño, pero es un buen punto de partida para comparar con los demás resultados después de la selección de características.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
DOC top+- split90	80 %	70 %	63.3 %	80 %
DOC top+- split66	71.2 %	66.3 %	70.2 %	74.2 %
DOC top+- cross10	72.8 %	66.4 %	71.4 %	79.8 %
DOC top+- cross15	72.8 %	67.1 %	71.1 %	79.9 %
Elec. DOC top+- split90	80 %	66.6 %	70 %	76.6 %
Elec. DOC top+- split66	70 %	69.3 %	69.3 %	70.2 %
Elec. DOC top+- cross10	68.7 %	68 %	66.1 %	79 %
Elec. DOC top+- cross15	70.4 %	67.1 %	64.7 %	78.7 %
top20 +- split90	83.3 %	53.3 %	70 %	86.6 %
top20 +- split66	81.1 %	65.3 %	74.2 %	84.1 %
top20 +- cross10	64.1 %	67.2 %	73.3 %	79 %
top20 +- cross15	65.2 %	64.5 %	70 %	78.9 %
DOC top+- sin Res. split90	83.3 %	73.3 %	73.3 %	83.3 %
DOC top+- sin Res. split66	71.2 %	73.2 %	63.3 %	77.2 %
DOC top+- sin Res. cross10	74.1 %	75.8 %	67.4 %	81.2 %
DOC top+- sin Res. cross15	74.1 %	75.1 %	71.8 %	78.8 %
DOC top– sin Res. split90	90 %	80 %	66.6 %	80 %
DOC top– sin Res. split66	73.2 %	77.2 %	63.3 %	76.2 %
DOC top– sin Res. cross10	74.1 %	81.5 %	70.1 %	79.5 %
DOC top– sin Res. cross15	74.8 %	80.5 %	71.8 %	78.5 %
Mayor	90 %	81.5 %	74.2 %	86.6 %

TABLA 7.1: Resultados de clasificación de las matrices originales antes de realizar la selección de características

7.1.2. Conjunto top20+-

Los algoritmos de selección al utilizar el conjunto top20+- no superaron el resultado máximo obtenido en la clasificación del mismo conjunto sin selección, en cambio mejoró bastante el promedio de exactitud para el clasificador SVM lineal en comparación al original al utilizar los algoritmos CFS y t score, como se muestra en la Tabla 7.2.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
CFS split90	80 %	83.3 %	66.6 %	73.3 %
CFS split66	82.1 %	84.1 %	75.2 %	79.2 %
CFS cross10	82.1 %	79.3 %	66.8 %	75 %
CFS cross15	79.3 %	79.7 %	66.8 %	75 %
CIFE split90	70 %	73.3 %	63.3 %	83.3 %
CIFE split66	72.2 %	63.3 %	60.3 %	75.2 %
CIFE cross10	36.8 %	71.9 %	66.8 %	75 %
CIFE cross15	35.1 %	70.6 %	67.2 %	74.3 %
lowvariance split90	83.3 %	63.3 %	76.6 %	86.6 %
lowvariance split66	79.2 %	59.4 %	69.3 %	79.2 %
lowvariance cross10	46.9 %	61.4 %	71.9 %	79.3 %
lowvariance cross15	47.2 %	60.8 %	72.6 %	81 %
ReliefF split90	73.3 %	63.3 %	56.6 %	73.3 %
ReliefF split66	79.2 %	57.4 %	73.2 %	80.1 %
ReliefF cross10	72.2 %	59.1 %	61.1 %	75 %
ReliefF cross15	72.9 %	62.8 %	71.6 %	77 %
t score split90	83.3 %	80 %	73.3 %	80 %
t score split66	79.2 %	75.2 %	76.2 %	80.1 %
t score cross10	77.3 %	80 %	75 %	78.3 %
t score cross15	78 %	79.3 %	70 %	79 %
Mayor	83.3 %	84.1 %	76.6 %	86.6 %

Tabla 7.2: Resultados de clasificación utilizando subconjuntos de top20+-

7.1.3. Conjunto DOC top+-

El conjunto DOC top+- fue de los que obtuvieron mejor resultado con un 93 % de exactitud como se puede apreciar en la Tabla 7.3. Este resultado se debe principalmente a que el conjunto es el de mayor dimensión el cual es explorado en su totalidad, de esta forma es que es más probable encontrar mejores características que posean mayor poder predictivo o describan mejor la clase a la que pertenecen.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
CFS split90	80 %	80 %	70 %	80 %
CFS split66	77.2 %	77.2 %	71.2 %	75.2 %
CFS cross10	75.8 %	76.5 %	69.1 %	77.5 %
CFS cross15	76.1 %	76.1 %	70.1 %	77.5 %
CIFE split90	90 %	73.3 %	70 %	83.3 %
CIFE split66	71.2 %	67.3 %	66.6 %	73.2 %
CIFE cross10	65.1 %	70.1 %	65.7 %	75.1 %
CIFE cross15	70.8 %	70.8 %	71.1 %	76.1 %
lowvariance split90	83.3 %	56.6 %	63.3 %	88 %
lowvariance split66	73.2 %	68.3 %	68.3 %	77.2 %
lowvariance cross10	73.1 %	65.7 %	71.8 %	78.1 %
lowvariance cross15	74.8 %	65.4 %	67.4 %	79.1 %
ReliefF split90	73.3 %	53.3 %	66.6 %	83.3 %
ReliefF split66	69.3 %	53.4 %	70.2 %	80.1 %
ReliefF cross10	69.7 %	57.7 %	61.7 %	75 %
ReliefF cross15	69.7 %	60.7 %	69.4 %	77 %
t score split90	86.6 %	80 %	66.6 %	93.3 %
t score split66	77.2 %	75.2 %	68.3 %	78.2 %
t score cross10	79.1 %	77.1 %	67.1 %	80 %
t score cross15	78.8 %	76.8 %	66.1 %	79.1 %
Mayor	90 %	80 %	71.8 %	93.3 %

Tabla 7.3: Resultados de clasificación utilizando subconjuntos de DOC top+-

7.1.4. Conjunto Electrostatic DOC top+-

Este conjunto presentó el desempeño más bajo en comparación a los demás como se ve en la Tabla 7.4. Esto se puede deber a que se utilizó sólo las energías electrostáticas y no las de desolvatación y de energía libre de unión, lo cual puede sugerir que, o las energías electrostáticas tienen poco poder predictivo, o que su poder predictivo depende de la consideración de las demás energías y por lo tanto funcionan mejor en conjunto como indica el trabajo de Guyon [23], en donde explican que una característica redundante puede ser útil en presencia de otras características.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
CFS split90	83.3 %	83.3 %	63.3 %	73.3 %
CFS split66	73.2 %	77.2 %	73.2 %	76.2 %
CFS cross10	74.8 %	78.1 %	66.4 %	76.5 %
CFS cross15	73.4 %	77.8 %	66.1 %	75.8 %
CIFE split90	80 %	76.6 %	53.3 %	76.6 %
CIFE split66	70.2 %	68.3 %	67.3 %	77.2 %
CIFE cross10	63.7 %	68.4 %	67.7 %	74.4 %
CIFE cross15	66.4 %	72.8 %	66.1 %	74.1 %
lowvariance split90	80 %	70 %	60 %	76.6 %
lowvariance split66	73.2 %	64.3 %	65.3 %	75.2 %
lowvariance cross10	69.4 %	61.4 %	68.4 %	75.5 %
lowvariance cross15	69.1 %	61.4 %	66.7 %	77.1 %
ReliefF split90	76.6 %	63.3 %	50 %	86.6 %
ReliefF split66	70.2 %	59.4 %	68.3 %	73.2 %
ReliefF cross10	70.4 %	58 %	67.4 %	74.1 %
ReliefF cross15	70.4 %	59 %	69.7 %	74.1 %
t score split90	76.6 %	63.3 %	63.3 %	80 %
t score split66	70 %	62.3 %	69.3 %	75.4 %
t score cross10	70.8 %	57.3 %	60 %	76.1 %
t score cross15	71.1 %	65.7 %	64.4 %	75.1 %
Mayor	83.3 %	83.3 %	73.2 %	86.6 %

TABLA 7.4: Resultados de clasificación utilizando subconjuntos de Electrostatic DOC top+-

7.1.5. Conjunto DOC top+- Sin Residuos

Este conjunto obtuvo una exactitud máxima de 93.3 % y en varias instancias alcanzó no menos que 90 % de exactitud como se puede apreciar en la Tabla 7.5. Esto se debe a que se extrajo los residuos del conjunto y dejaron sólo las energías. Esto puede significar que los residuos eran redundantes para la clasificación del tipo de proteína y afectaban a la predicción.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
CFS split90	83.3 %	86.6 %	76.6 %	76.6 %
CFS split66	78.2 %	81.1 %	64.3 %	73.2 %
CFS cross10	77.1 %	79.1 %	71.4 %	78.1 %
CFS cross15	77.1 %	79.1 %	69.1 %	77.1 %
CIFE split90	90 %	90 %	63.3 %	83.3 %
CIFE split66	71.2 %	74.2 %	72.2 %	76.2 %
CIFE cross10	67.7 %	74.4 %	63 %	77.1 %
CIFE cross15	71.4 %	73.8 %	68.4 %	76.5 %
lowvariance split90	83.3 %	73.3 %	66.6 %	83.3 %
lowvariance split66	71.2 %	75.2 %	66.3 %	75.2 %
lowvariance cross10	74.4 %	75.8 %	67.7 %	81.5 %
lowvariance cross15	73.8 %	77.1 %	70.1 %	80.8 %
ReliefF split90	90 %	76.6 %	80 %	83.3 %
ReliefF split66	77.2 %	72.2 %	66.3 %	78.2 %
ReliefF cross10	76.1 %	78.8 %	69.7 %	80.5 %
ReliefF cross15	75.8 %	78.5 %	68.1 %	79.5 %
t score split90	86.6 %	80 %	66.6 %	93.3 %
t score split66	77.2 %	75.2 %	68.3 %	78.2 %
t score cross10	79.1 %	77.1 %	67.1 %	80.2 %
t score cross15	78.8 %	76.8 %	66.1 %	79.1 %
Mayor	90 %	90 %	80 %	93.3 %

Tabla 7.5: Resultados de clasificación utilizando subconjuntos de DOC top+- Sin Residuos

7.1.6. Conjunto DOC top - - Sin Residuos

Este conjunto obtuvo un desempeño menor en comparación al conjunto DOC top+- sin residuos como se aprecia en la Tabla 7.6 debido a que se consideraron sólo energías negativas. Si bien se pensaba que las energías negativas contribuían más en la interacción, puede ser que estas energías se complementen.

Conjuntos	Naïve Bayes	linear SVM	Random Tree	Random Forest
CFS split90	86.6 %	83.3 %	66.6 %	73.3 %
CFS split66	79.2 %	78.2 %	69.3 %	78.2 %
CFS cross10	78.1 %	79.1 %	72.8 %	74.4 %
CFS cross15	78.5 %	79.5 %	74.4 %	76.1 %
CIFE split90	90 %	86.6 %	80 %	83.3 %
CIFE split66	71.2 %	77.2 %	62.3 %	73.2 %
CIFE cross10	66.7 %	74.1 %	71.8 %	75.8 %
CIFE cross15	70.8 %	74.4 %	69.7 %	77.5 %
lowvariance split90	90 %	86.6 %	73.3 %	80 %
lowvariance split66	73.2 %	75.2 %	69.3 %	74.2 %
lowvariance cross10	74.1 %	81.5 %	71.8 %	78.1 %
lowvariance cross15	74.8 %	80.8 %	70.1 %	78.1 %
ReliefF split90	80 %	73.3 %	63.3 %	73.3 %
ReliefF split66	74.2 %	73.2 %	69.3 %	78.2 %
ReliefF cross10	75.8 %	79.5 %	69.4 %	76.8 %
ReliefF cross15	76.5 %	80.2 %	74.8 %	76.1 %
t score split90	83.3 %	86.6 %	63.3 %	83.3 %
t score split66	78.2 %	78.2 %	67.3 %	77.2 %
t score cross10	78.1 %	79.1 %	70.8 %	78.1 %
t score cross15	78.1 %	79.1 %	71.4 %	77.1 %
Mayor	90 %	86.6 %	74.8 %	83.3 %

TABLA 7.6: Resultados de clasificación utilizando subconjuntos de DOC top - - Sin Residuos

7.2. Evaluación

Los algoritmos con mejor rendimiento según cada conjunto se encuentran en la Tabla 7.7:

Conjuntos	Algoritmo	Resultado
top20+-	lowvariance split90	86.6 %
DOC top+-	t score split90	93.3 %
Electrostatic DOC top+-	ReliefF split90	86.6 %
DOC top+- Sin Residuos	t score split90	93.3 %
DOC top - - Sin Residuos	CIFE split90	90 %

Tabla 7.7: Algoritmos con los mejores resultados para cada conjunto.

La tabla 7.8 muestra el cruce de los resultados obtenidos por cada clasificador para cada conjunto.

Conjunto	Naïve Bayes	linear SVM	Random Tree	Random Forest
top20+-	83.3 %	84.1 %	76.6 %	86.6 %
DOC top+-	90 %	80 %	71.8 %	93.3 %
Electrostatic DOC top+-	83.3 %	83.3 %	73.2 %	86.6 %
DOC top+- sin Residuo	90 %	90 %	80 %	93.3 %
DOC top- - sin Residuo	90 %	86.6 %	74.8 %	83.3 %

Tabla 7.8: Resumen de los mejores resultados de cada clasificador para cada conjunto.

El clasificador **Random Forest** alcanza un mayor desempeño que los demás clasificadores y del máximo valor obtenido de exactitud de la clasificación se demuestra que se pudo alcanzar un mayor valor que en el trabajo de Gutiérrez-Víctor [16] a un menor costo de recursos computacionales.

Hubieron casos en donde la exactitud de la clasificación también fue alta, lo cual no es de ignorar ya que su desempeño fue bueno (90 %) pero quedaron opacados por el algoritmo **t score** que alcanzó una exactitud de 93.3 % con Random Forest.

El conjunto más grande **DOC top+-** del que se esperaba mayor poder predictivo, se desempeñó bastante bien al considerar el resultado del algoritmo **t score** con el clasificador Random Forest.

En cambio el mejor desempeño fue obtenido al utilizar el conjunto **DOC top+- Sin Residuos** debido a que al igual que el conjunto DOC top+-, t score consiguió un valor

máximo de 93.3 %, pero DOC top+- Sin Residuos en promedio fue mejor al conseguir 90 % de exactitud en 3 ocasiones: 2 con el clasificador Naïve Bayes con los algoritmos ReliefF y CIFE, y 1 por el algoritmo CIFE con el clasificador **SVM lineal**, además cabe mencionar que la técnica de división por porcentaje (percentage split) de 90 % fue la que entregó los valores más altos.

En la Tabla 7.9 se muestra el tiempo de ejecución de los algoritmos de selección de características para cada conjunto expresados en segundos y se encuentran en el mismo orden en que se presentaron los conjuntos. A primera vista se puede notar una gran diferencia entre los tiempos de ejecución de los algoritmos, tomando en cuenta uno de los algoritmos que se ejecutó en menor tiempo, **t score**, que alcanzó una exactitud máxima de 93.3 %, al compararlo con el algoritmo que tardó más tiempo en ejecutarse, **CFS** y que no pudo superar a los demás algoritmos en la exactitud de la clasificación, puedo indicar que el criterio de correlación no fue tan útil para seleccionar características de IPP, pero que en cambio utilizar un criterio estadístico mucho más rápido proporcionó las características con el mayor poder representativo para que los clasificadores pudieran desempeñar mejor su función.

Algoritmo	top20+-	DOC+-	E.DOC+-	DOC+-s.res	DOC–s.res
t score	1.36	2.59	1.47	1.65	1.30
ReliefF	2.58	4.71	2.51	3.23	3.04
CFS	1302.92	7168.18	995.28	2341.0	1489.61
CIFE	865.93	3837.94	1358.65	838.66	751.41
low variance	0.42	1.12	0.56	0.57	0.53

Tabla 7.9: Tiempos de ejecución de los algoritmos para todos los conjuntos expresado en segundos.

7.2.1. Comparación de resultados con trabajos previos

En el trabajo de Gutiérrez-Víctor [16] se muestran los mejores resultados de clasificación en la Tabla 7.10, con los diferentes conjuntos generados por los algoritmos de búsqueda secuencial hacia delante (SFS) y búsqueda secuencial hacia atrás (SBS), entre otros. En su trabajo nombra a un conjunto TopE+-, el cual en este trabajo es nombrado DOC Top+- sinResiduos, aunque si bien ambos nombres son válidos ya que el conjunto sin residuos se compone de sólo las energías, y es debido a eso su nombre. Utilizó los clasificadores de bosque aleatorio y máquinas de vector de soporte con los kernel lineal, polinomial 2, polinomial 3, radial y sigmoideo. Como se puede apreciar, el valor máximo obtenido fue con el conjunto Top20+- y el algoritmo SFS con el clasificador de bosques

aleatorios con un 87.88 % de exactitud. Por lo tanto, se genera la Tabla 7.11 con la comparación de los mejores resultados obtenidos en este proyecto de título con los de la Tabla 7.10.

En la comparación se puede apreciar en los conjuntos utilizados por ambos trabajos que para el conjunto Top20+- el porcentaje de exactitud bajó aproximadamente un 1.28 %, lo cual no es una diferencia muy significativa, en cambio para el conjunto DOC Top+- sinResiduos, la exactitud se vio incrementada aproximadamente en un 6.63 %.

Conjunto de datos	Lineal	Polyn 2	Polyn 3	Radial	Sigmoid	Bosques aleatorios
Top20+- SFS	75 %	72.57 %	69.2 %	72.28 %	72.28 %	87.88 %
Top20+- SBS	73.33 %	82.14 %	74.58 %	77.28 %	72.28 %	86.67 %
Top20+- SFFS	76.67 %	70.27 %	76.67 %	72.28 %	72.28 %	80 %
DOC Top+- sinResiduos SFS	71.81 %	70.47 %	71.14 %	80 %	83.33 %	86.67 %
DOC Top- sinResiduos SFS	80 %	80 %	76.17 %	80 %	83.33 %	86.11 %

Tabla 7.10: Resultados de clasificación del trabajo de Guitérrez-Víctor[16]

Conjuntos de datos	Algoritmos utilizados por Gutiérrez-Víctor	Bosques Alteatorios		Algoritmos utilizados en este proyecto
Top20+-	SFS	87.88 %	86.6 %	baja varianza
DOC Top+- sinResiduos	SFS	86.67 %	93.3 %	T score
DOC Top- sinResiduos	SFS	86.11 %	90.0 %	CIFE

Tabla 7.11: Cruce de resultados con los del trabajo de Gutiérrez-Víctor [16] del clasificador Bosque Aleatorio.

Capítulo 8

Conclusión

Se comenzó realizando pruebas de selección de características utilizando algoritmos filtro para medir su desempeño y conocer cuanto porcentaje de exactitud podían lograr para luego implementar algoritmos más complejos debido a que se esperaba un desempeño menor por parte de los de tipo filtro, pero se demostró que estos algoritmos pueden seleccionar características de interacciones de proteína-proteína relevantes para la clasificación o por lo menos reducir las características redundantes, como se puede observar en los resultados obtenidos, ya que se alcanzó un 93.3 % de exactitud. Se encontró que al utilizar un clasificador para la evaluación o pre-clasificación no aumentó considerablemente el tiempo de ejecución y además mejoró sustancialmente la exactitud final de la clasificación.

Para ejecutar los algoritmos y posteriormente realizar el entrenamiento del modelo y clasificación se utilizó un computador (laptop) con las siguientes características.

Procesador: Intel Core i5-4210U CPU 1.70 GHz hasta 2.4 GHz.

Memoria Ram: 8 GB.

Arquitectura: 64 bits.

Sistemas Operativos: Ubuntu 14, Windows 8.1.

Las versiones utilizadas de los programas utilizados fueron **Python 2.7 y 3.5**, **Weka 3.8**, **Numpy 1.6**, **Numpy 1.13** y **Sci-Py Stack 0.16.1**.

De las matrices utilizadas 4 fueron de trabajos previos y fue explorada otra posibilidad con la matriz de sólo energías electrostáticas al extraer los otros tipos de energías del conjunto DOC top+-.

Además se reafirma que las energías de las interacciones sin residuo son importantes para discriminar entre tipos de proteínas transientes con permanentes ya que fue el conjunto de datos que obtuvo el mejor desempeño en promedio con cada algoritmo utilizado.

8.1. Trabajo futuro

Un nuevo trabajo de investigación puede ser realizado si se logran conseguir nuevos datos de complejos de interacción de proteína-proteína, de forma que se pueda probar el real poder predictivo de las características seleccionadas por los algoritmos utilizados en éste trabajo, lo que también significa que sepuedan realizar pruebas con algoritmos de tipo wrapper los cuales requieren más ejemplos.

Se recolectó una gran variedad de algoritmos y se encontró otros recientes como por ejemplo el de Moayedikia [68] para que sirvan de referencia de trabajos posteriores que puedan surgir.

El poder predictivo del clasificador **Random Tree** en general fue el más bajo a pesar de ser el bloque de construcción del clasificador de Random Forest el cual tuvo los mejores resultados. Esto incentiva a investigar el poder de los clasificadores en combinación.

Apéndice A

Códigos de algoritmos de selección de características utilizados

Algoritmo de Baja Varianza

```
import numpy as np
import random
from math import pow
from timeit import default_timer as timer

file = open("input.txt", "r")
data = file.readlines()
completeSet = np.loadtxt(data)

nRows = len(completeSet)
nColumns = len(completeSet[0])

differenceSet = np.empty([nRows, nColumns - 1], dtype=float)

def variance_measurement(arg, p, set, diffValues, nRows, nColumns):
        percentage = p # porcentaje del conjunto a medir para obtener media

        if p > 100:
                p = 100
        elif p < 1:
                p = 1

        nColumnsToMeasure = int(nColumns / percentage)
        randomColumns = random.sample(xrange(1, nColumns), nColumnsToMeasure)

        # media general
        sum = 0
        totalSum = 0
```

```
        for i in range(0, nColumnsToMeasure):
                for j in range(0, nRows):
                        sum += set[j][randomColumns[i]]
                sum = sum / nRows
                totalSum += sum
                sum = 0
        mean = totalSum / nColumnsToMeasure

        ##calculando la varianza
        varianceValues = []

        varianceSum = 0;

        #estimar varianza
        for i in range(1, nColumns):
                for j in range(0, nRows):
                        diffValues[j][i - 1] = set[j][i] - mean
        for i in range(1, nColumns):
                for j in range(0, nRows):
                        varianceSum += pow(diffValues[j][i - 1], 2)
                varianceValues.append(varianceSum/nRows)
                varianceSum = 0
        #estimar varianza general
        sum = 0
        for i in range(0, len(varianceValues)):
                sum += varianceValues[i]

        meanVariance = sum / len(varianceValues)

        # Extraer valores de varianza baja y alta
        lowestVariancePositions = []
        highestVariancePositions = []
        for i in range(0, len(varianceValues)):
                if varianceValues[i] < meanVariance:
                        lowestVariancePositions.append(i+1)
                else:
                        if varianceValues[i] > meanVariance:
                                highestVariancePositions.append(i+1)
        if arg == 0:
                return lowestVariancePositions
        else:
                return highestVariancePositions
t = timer()
##Call
outputSet = variance_measurement(0,10,completeSet,differenceSet,nRows,nColumns)

##Filtering
finalMatrix = completeSet[:, [0] + outputSet]

##Writing to file
output = open("output.txt", "w")
text = ""
for i in range(0, len(finalMatrix)):
        text = str(finalMatrix[i][0])
        for j in range(1, len(finalMatrix[0])):
```

```
                text += "," + str(finalMatrix[i][j])
        output.write(text+"\n")

output.close()
print timer() - t, " seconds"
```

Algoritmo T score

```
import scipy.io
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn import svm
from timeit import default_timer as timer
import numpy as np

def t_score(X, y):
    n_samples, n_features = X.shape
    F = np.zeros(n_features)
    c = np.unique(y)
    if len(c) == 2:
        for i in range(n_features):
            f = X[:, i]
            class0 = f[y == c[0]]
            class1 = f[y == c[1]]
            mean0 = np.mean(class0)
            mean1 = np.mean(class1)
            std0 = np.std(class0)
            std1 = np.std(class1)
            n0 = len(class0)
            n1 = len(class1)
            t = mean0 - mean1
            t0 = np.true_divide(std0**2, n0)
            t1 = np.true_divide(std1**2, n1)
            F[i] = np.true_divide(t, (t0 + t1)**0.5)
    else:
        print('debe contener clases binarias solamente')
        exit(0)
    return np.abs(F)


def feature_ranking(F):
    idx = np.argsort(F)
    return idx[::-1]

def formatData(name, setX, setY):
    file = open(name,"w")
    for i in range(len(setX)):
        file.write(str(setY[i])[0])
        for j in range(len(setX[0])):
            file.write("," + str(setX[i][j]))
        file.write("\n")
    file.close()

def main():
    mat = scipy.io.loadmat('input.mat')
    X = mat['X']
    X = X.astype(float)
    y = mat['Y']
    y = y[:, 0]
    n_samples, n_features = X.shape
    ss = cross_validation.KFold(n_samples, n_folds=10, shuffle=True)
    num_fea = 100
```

```
    clf = svm.LinearSVC()
    maxval = 0
    lastvalue = -1
    features = None

    for train, test in ss:
        score = t_score(X, y)
        idx = feature_ranking(score)
        selected_features = X[:, idx[0:num_fea]]
        clf.fit(selected_features[train], y[train])
        y_predict = clf.predict(selected_features[test])
        acc = accuracy_score(y[test], y_predict)

        if lastvalue == -1:
                lastvalue = acc
        else:
                if acc > lastvalue:
                    features = selected_features
                    lastvalue = acc
                    print acc
    formatData("output.txt", features, y)

if __name__ == '__main__':
    t = timer()
    main()
    print timer() - t, " seconds"
```

Algoritmo CIFE

```
import scipy.io
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
from sklearn import svm
from timeit import default_timer as timer
from skfeature.utility.entropy_estimators import *

def lcsi(X, y, **kwargs):
    n_samples, n_features = X.shape
    F = []
    is_n_selected_features_specified = False
    if 'beta' in kwargs.keys():
        beta = kwargs['beta']
    if 'gamma' in kwargs.keys():
        gamma = kwargs['gamma']
    if 'n_selected_features' in kwargs.keys():
        n_selected_features = kwargs['n_selected_features']
        is_n_selected_features_specified = True

    t1 = np.zeros(n_features)
    t2 = np.zeros(n_features)
    t3 = np.zeros(n_features)

    for i in range(n_features):
        f = X[:, i]
        t1[i] = midd(f, y)
    j_cmi = 1

    while True:
        if len(F) == 0:
            idx = np.argmax(t1)
            F.append(idx)
            f_select = X[:, idx]
        if is_n_selected_features_specified is True:
            if len(F) == n_selected_features:
                break
        if is_n_selected_features_specified is not True:
            if j_cmi < 0:
                break
        j_cmi = -1000000000000
        for i in range(n_features):
            if i not in F:
                f = X[:, i]
                t2[i] += midd(f_select, f)
                t3[i] += cmidd(f_select, f, y)
                t = t1[i] - beta*t2[i] + gamma*t3[i]

                if t > j_cmi:
                    j_cmi = t
                    idx = i
        F.append(idx)
        f_select = X[:, idx]
    return np.array(F)
```

```
def cife(X, y, **kwargs):
    if 'n_selected_features' in kwargs.keys():
        n_selected_features = kwargs['n_selected_features']
        F = lcsi(X, y, beta = 1, gamma = 1, n_selected_features = n_selected_features)
    else:
        F = lcsi(X, y, beta=1, gamma=1)
    return F

def formatData(name, setX, setY):
    file = open(name,"w")
    for i in range(len(setX)):
        file.write(str(setY[i])[0])
        for j in range(len(setX[0])):
            file.write("," + str(setX[i][j]))
        file.write("\n")
    file.close()

def main():
    mat = scipy.io.loadmat('input.mat')
    X = mat['X']
    X = X.astype(float)
    y = mat['Y']
    y = y[:, 0]
    n_samples, n_features = X.shape
    ss = cross_validation.KFold(n_samples, n_folds=10, shuffle=True)
    num_fea = 100
    clf = svm.LinearSVC()

    selfeatures = None
    maxacc = 0
    for train, test in ss:
        idx = cife(X[train], y[train], n_selected_features=num_fea)
        features = X[:, idx[0:num_fea]]
        clf.fit(features[train], y[train])
        y_predict = clf.predict(features[test])
        acc = accuracy_score(y[test], y_predict)

        if acc > maxacc:
            maxacc = acc
            print acc
            selfeatures = features

    formatData("output.txt", selfeatures, y)

if __name__ == '__main__':
    t = timer()
    main()
    print timer() - t, " seconds"
```

Algoritmo CFS

```
import scipy.io
from sklearn import svm
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from timeit import default_timer as timer
import numpy as np
from skfeature.utility.mutual_information import su_calculation


def merit_calculation(X, y):
    n_samples, n_features = X.shape
    rff = 0
    rcf = 0
    for i in range(n_features):
        fi = X[:, i]
        rcf += su_calculation(fi, y)
        for j in range(n_features):
            if j > i:
                fj = X[:, j]
                rff += su_calculation(fi, fj)
    rff *= 2
    merits = rcf / np.sqrt(n_features + rff)
    return merits


def cfs(X, y):
    n_samples, n_features = X.shape
    F = []
    M = []
    while True:
        merit = -100000000000
        idx = -1
        for i in range(n_features):
            if i not in F:
                F.append(i)
                t = merit_calculation(X[:, F], y)
                if t > merit:
                    merit = t
                    idx = i
                F.pop()
        F.append(idx)
        M.append(merit)
        if len(M) > 5:
            if M[len(M)-1] <= M[len(M)-2]:
                if M[len(M)-2] <= M[len(M)-3]:
                    if M[len(M)-3] <= M[len(M)-4]:
                        if M[len(M)-4] <= M[len(M)-5]:
                            break
    return np.array(F)

def formatData(name, setX, setY):
    file = open(name,"w")
    for i in range(len(setX)):
        file.write(str(setY[i])[0])
```

```
        for j in range(len(setX[0])):
            file.write("," + str(setX[i][j]))
        file.write("\n")
    file.close()

def main():
    mat = scipy.io.loadmat('input.mat')
    X = mat['X']
    X = X.astype(float)
    y = mat['Y']
    y = y[:, 0]
    n_samples, n_features = X.shape
    ss = cross_validation.KFold(n_samples, n_folds=10, shuffle=True)
    num_fea =  100
    clf = svm.LinearSVC()
    correct = 0
    features = None
    maxacc = 0

    for train, test in ss:
        idx = cfs(X[train], y[train])
        selected_features = X[:, idx[0:num_fea]]
        clf.fit(selected_features[train], y[train])
        y_predict = clf.predict(selected_features[test])
        acc = accuracy_score(y[test], y_predict)
        correct = correct + acc

        if acc > maxacc:
            maxacc = acc
            print acc
            features = selected_features
    formatData("output.txt", features, y)

if __name__ == '__main__':
    t = timer()
    main()
    print timer() - t, " seconds"
```

Algoritmo ReliefF

```
import scipy.io
from sklearn import cross_validation
from sklearn import svm
from sklearn.metrics import accuracy_score
from timeit import default_timer as timer
import numpy as np
from sklearn.metrics.pairwise import pairwise_distances

def reliefF(X, y, **kwargs):
    if "k" not in kwargs.keys():
        k = 5
    else:
        k = kwargs["k"]

    n_samples, n_features = X.shape
    distance = pairwise_distances(X, metric='manhattan')
    score = np.zeros(n_features)

    for idx in range(n_samples):
        near_hit = []
        near_miss = dict()

        self_fea = X[idx, :]
        c = np.unique(y).tolist()

        stop_dict = dict()
        for label in c:
            stop_dict[label] = 0
        del c[c.index(y[idx])]

        p_dict = dict()
        p_label_idx = float(len(y[y == y[idx]]))/float(n_samples)

        for label in c:
            p_label_c = float(len(y[y == label]))/float(n_samples)
            p_dict[label] = p_label_c/(1-p_label_idx)
            near_miss[label] = []

        distance_sort = []
        distance[idx, idx] = np.max(distance[idx, :])

        for i in range(n_samples):
            distance_sort.append([distance[idx, i], int(i), y[i]])
        distance_sort.sort(key=lambda x: x[0])

        for i in range(n_samples):
            if distance_sort[i][2] == y[idx]:
                if len(near_hit) < k:
                    near_hit.append(distance_sort[i][1])
                elif len(near_hit) == k:
                    stop_dict[y[idx]] = 1
            else:
                if len(near_miss[distance_sort[i][2]]) < k:
                    near_miss[distance_sort[i][2]].append(distance_sort[i][1])
```

```
                else:
                    if len(near_miss[distance_sort[i][2]]) == k:
                        stop_dict[distance_sort[i][2]] = 1
            stop = True
            for (key, value) in stop_dict.items():
                    if value != 1:
                        stop = False
            if stop:
                break

        near_hit_term = np.zeros(n_features)
        for ele in near_hit:
            near_hit_term = np.array(abs(self_fea-X[ele, :]))+np.array(near_hit_term)

        near_miss_term = dict()
        for (label, miss_list) in near_miss.items():
            near_miss_term[label] = np.zeros(n_features)
            for ele in miss_list:
                near_miss_term[label] = np.array(abs(self_fea-X[ele, :]))
                near_miss_term[label] += np.array(near_miss_term[label])
            score += near_miss_term[label]/(k*p_dict[label])
        score -= near_hit_term/k
    return score

def feature_ranking(score):
    idx = np.argsort(score, 0)
    return idx[::-1]

def formatData(name, setX, setY):
    file = open(name,"w")
    for i in range(len(setX)):
        file.write(str(setY[i])[0])
        for j in range(len(setX[0])):
            file.write("," + str(setX[i][j]))
        file.write("\n")
    file.close()

def main():
    mat = scipy.io.loadmat('input.mat')
    X = mat['X']
    X = X.astype(float)
    y = mat['Y']
    y = y[:, 0]
    n_samples, n_features = X.shape
    ss = cross_validation.KFold(n_samples, n_folds=10, shuffle=True)
    num_fea = 100
    clf = svm.LinearSVC()
    features = None
    maxacc = 0
    for train, test in ss:
        score = reliefF(X[train], y[train])
        idx = feature_ranking(score)
        selected_features = X[:, idx[0:num_fea]]
        clf.fit(selected_features[train], y[train])
        y_predict = clf.predict(selected_features[test])
```

```
        acc = accuracy_score(y[test], y_predict)
        if acc > maxacc:
            maxacc = acc
            print acc
            features = selected_features
    formatData("output.txt", features, y)

if __name__ == '__main__':
    t = timer()
    main()
    print timer() - t, " seconds"
```

Bibliografía

[1] Harry Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.

[2] Zhang Xuegong. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica*, 26(1):32–42, 2000.

[3] Gopal Malakar. Introduction to support vector machine (svm) and kernel trick (how does svm and kernel work?), 2016. URL `https://www.youtube.com/watch?v=ikt7Qze0czE`.

[4] WittmannF. Visualization of svm kernels linear, rbf, poly and sigmoid on python, 2016. URL `https://gist.github.com/WittmannF/60680723ed8dd0cb993051a7448f7805`.

[5] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[6] Data School. Roc curves and area under the curve (auc) explained, 2014. URL `https://www.youtube.com/watch?v=OAl6eAyP-yo`.

[7] MD Thomas G. Tape. The area under an roc curve. URL `http://gim.unmc.edu/dxtests/roc3.htm`.

[8] Charles E Cook, Mary Todd Bergman, Robert D Finn, Guy Cochrane, Ewan Birney, and Rolf Apweiler. The european bioinformatics institute in 2016: data growth and integration. *Nucleic acids research*, 44(D1):D20–D26, 2015.

[9] yourgenome. From dna to protein - 3d, 2015. URL `https://www.youtube.com/watch?v=gG7uCskUOrA`.

[10] majordifferences. Difference between protein structures, 2015. URL `http://www.majordifferences.com/2013/02/difference-between-primary-and.html#.WYQ_5Yg2sdU`.

[11] Tatiana Gutiérrez Bunster et al. *Estudio de características energéticas en zonas de interacción proteína-proteína, para identificación de interacciones transitorias*

y permanentes. PhD thesis, Universidad de Concepción. Facultad de Ingeniería. Departamento de Ingeniería Informática y Ciencias de la Computación, 2008.

[12] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank, 1999–. In *International Tables for Crystallography Volume F: Crystallography of biological macromolecules*, pages 675–684. Springer, 2006.

[13] Gem. Coevolution, from hummingbirds to proteins, 2011. URL `http://proteinevolution.fieldofscience.com/2011/10/`.

[14] Tatiana A Gutiérrez-Bunster. *Towards a better Understanding of Protein-Protein Interaction Networks*. PhD thesis, University of Victoria, 2014.

[15] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 306–313. IEEE, 2002.

[16] Víctor Gutiérrez. Estudio y selección de algoritmos discriminantes para jerarquizar características de interacción de proteínas de acuerdo a su relevancia en la interacción, 2016.

[17] Tom M Mitchell et al. Machine learning. wcb, 1997.

[18] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[19] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1): 4–37, 2000.

[20] Sofía Natalia Galicia Haro. Análisis sintáctico conducido por un diccionario de patrones de manejo sintáctico para lenguaje español. *Computación y Sistemas*, 6 (2):143–152, 2002.

[21] Luis Eduardo Quintos Vázquez. *Reconocimiento de patrones, el enfoque lógico combinatorio*. PhD thesis, Instituto politecnico nacional. Escuela superior de fisica y matematicas, 2013.

[22] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[23] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[24] Thomas Oommen, Debasmita Misra, Navin KC Twarakavi, Anupma Prakash, Bhaskar Sahoo, and Sukumar Bandopadhyay. An objective analysis of support vector machine based classification for remote sensing. *Mathematical geosciences*, 40(4): 409–424, 2008.

[25] Sergios Theodoridis and Konstantinos Koutroumbas. Pattern recognition. *None*, 2003.

[26] Steve R Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14:85–86, 1998.

[27] JA Resendiz Trejo. Las máquinas de vectores de soporte para identificación en línea, 2006.

[28] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[29] Google Developers. Visualizing a decision tree - machine learning recipes 2, 2016. URL `https://www.youtube.com/watch?v=tNa99PG8hR8`.

[30] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[31] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[32] Weka. Lesson 2.2 in data mining with weka, 2014. URL `http://www.cs.waikato.ac.nz/ml/weka/mooc/dataminingwithweka/transcripts/Transcript2-2.txt`.

[33] Jason Brownlee. 8 tactics to combat imbalanced classes in your machine learning dataset, 2015. URL `http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/`.

[34] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[35] Elizabeth A Freeman and Gretchen G Moisen. A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa. *Ecological Modelling*, 217(1):48–58, 2008.

[36] Steve Simon. What is a kappa coefficient? (cohen's kappa), 2007. URL `http://www.pmean.com/definitions/kappa.htm`.

[37] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006.

[38] Nicholas M Luscombe, Dov Greenbaum, Mark Gerstein, et al. What is bioinformatics? a proposed definition and overview of the field. *Methods of information in medicine*, 40(4):346–358, 2001.

[39] Victor de la Torre Russis, Alfredo Valles, Raúl Gómez, Glay Chinea, and Tirso Pons. Interacciones proteína-proteína: bases de datos y métodos teóricos de predicción. *Biotecnología Aplicada*, 20(3):201–208, 2003.

[40] Irene MA Nooren and Janet M Thornton. Diversity of protein–protein interactions. *The EMBO journal*, 22(14):3486–3492, 2003.

[41] Kengo Kinoshita and Haruki Nakamura. Identification of the ligand binding sites on the molecular surface of proteins. *Protein Science*, 14(3):711–718, 2005.

[42] Susan Jones and Janet M Thornton. Principles of protein-protein interactions. *Proceedings of the National Academy of Sciences*, 93(1):13–20, 1996.

[43] Julian Mintseris and Zhiping Weng. Atomic contact vectors in protein-protein recognition. *Proteins: Structure, Function, and Bioinformatics*, 53(3):629–639, 2003.

[44] Sébastien Fiorucci and Martin Zacharias. Prediction of protein-protein interaction sites using electrostatic desolvation profiles. *Biophysical journal*, 98(9):1921–1930, 2010.

[45] Gokul Vasudev and Luis Rueda. A model to predict and analyze protein-protein interaction types using electrostatic energies. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–5. IEEE, 2012.

[46] Shenkai Gu, Ran Cheng, and Yaochu Jin. Feature selection for high-dimensional classification using a competitive swarm optimizer. *Soft Computing*, pages 1–12, 2016.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *arXiv preprint arXiv:1601.07996*, 2016. URL `https://github.com/jundongl/scikit-feature/`.

[49] Patrenahalli M. Narendra and Keinosuke Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 9(C-26):917–922, 1977.

[50] Lei Xu, Pingfan Yan, and Tong Chang. Best first strategy for feature selection. In *Pattern Recognition, 1988., 9th International Conference on*, pages 706–708. IEEE, 1988.

[51] Jacob Sheinvald, Byron Dom, and Wayne Niblack. A modeling approach to feature selection. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume 1, pages 535–539. IEEE, 1990.

[52] Claire Cardie. Using decision trees to improve case-based learning. In *Proceedings of the tenth international conference on machine learning*, pages 25–32, 1993.

[53] Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, 53(1-2):23–69, 2003.

[54] Dahua Lin and Xiaoou Tang. Conditional infomax learning: an integrated framework for feature extraction and fusion. *Computer Vision–ECCV 2006*, pages 68–82, 2006.

[55] Leon Bobrowski. Feature selection based on some homogeneity coefficient. In *Pattern Recognition, 1988., 9th International Conference on*, pages 544–546. IEEE, 1988.

[56] Mark Andrew Hall. *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato Hamilton, 1999.

[57] Timothy C Urdan. *Statistics in plain English*. Taylor & Francis, 2016.

[58] Huan Liul, Hiroshi Motoda, and Manoranjan Dash. A monotonic measure for optimal feature selection. *Machine learning: ECML-98*, pages 101–106, 1998.

[59] Hussein Almuallim and Thomas G Dietterich. Learning with many irrelevant features. In *AAAI*, volume 91, pages 547–552, 1991.

[60] Jeffrey C Schlimmer et al. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *ICML*, pages 284–290, 1993.

[61] Alberto Sangiovanni-Vincentelli. Constructive induction using a non-greedy strategy for feature selection. In *Machine Learning Proceedings 1992: Proceedings of the Ninth International Workshop (ML92)*, page 355. Morgan Kaufmann, 2014.

[62] Justin Doak. Cse-92-18-an evaluation of feature selection methodsand their application to computer security. *UC Davis Dept of Computer Science tech reports*, 1992.

[63] Iman Foroutan and Jack Sklansky. Feature selection for automatic classification of non-gaussian data. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(2): 187–198, 1987.

[64] Manabu Ichino and Jack Sklansky. Optimum feature selection by zero-one integer programming. *IEEE Transactions on Systems, Man, and Cybernetics*, 14(5):737–746, 1984.

[65] Scott Davies and Stuart Russell. Np-completeness of searches for smallest possible feature sets. In *Proceedings of the 1994 AAAI fall symposium on relevance*, pages 37–39. AAAI Press, 1994.

[66] Pat Langley and Stephanie Sage. Oblivious decision trees and abstract cases. In *Working notes of the AAAI-94 workshop on case-based reasoning*, pages 113–117. Seattle, WA, 1994.

[67] Matthew Shardlow. An analysis of feature selection techniques. *The University of Manchester*, 2016.

[68] Alireza Moayedikia, Kok-Leong Ong, Yee Ling Boo, William GS Yeoh, and Richard Jensen. Feature selection for high dimensional imbalanced class data using harmony search. *Engineering Applications of Artificial Intelligence*, 57:38–49, 2017.