Desarrollo de una solución mediante Deep Learning para la predicción de secuencias nucleosomales de DNA

Trabajo de fin de grado

Grado en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

Julio 2021

Autor

Alba Vallejo Urruchi

Tutores

Rodrigo Santamaría Vicente
Francisco Antequera

Certificado de los tutores

D. Rodrigo Santamaría Vicente, profesor titular del Departamento de Informática y Automática de la Universidad de Salamanca y Dr. Francisco Antequera, profesor investigador del IBFG, centro CSIC adscrito a la Universidad de Salamanca, en calidad de co-tutor.

Hacen constar:

Que el trabajo titulado “Desarrollo de una solución mediante Deep Learning para la predicción de secuencias nucleosomales de DNA” ha sido realizado por Dña. Alba Vallejo Urruchi, con DNI 71348601W y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado de la Titulación Grado de Ingeniería Informática de esta Universidad.

ANTEQUERA MARQUEZ FRANCISCO MARIA - 06972635R

Firmado digitalmente por ANTEQUERA MARQUEZ FRANCISCO MARIA - 06972635R
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-06972635R, givenName=FRANCISCO MARIA, sn=ANTEQUERA MARQUEZ, cn=ANTEQUERA MARQUEZ FRANCISCO MARIA - 06972635R
Fecha: 2021.07.07 09:26:55 +02'00'

Firmado por SANTAMARIA VICENTE RODRIGO - 70879303L el día 07/07/2021 con un certificado emitido por AC FNMT Usuarios

autofirma

En Salamanca, a 6 julio de 2021

Firmado por VALLEJO URRUCHI ALBA - 71348601W el día 06/07/2021 con un certificado emitido por AC FNMT Usuarios

Resumen

La inteligencia artificial ha ido dando respuesta a problemas que requerían de la inteligencia humana para su resolución. Los métodos de aprendizaje profundo (Deep Learning) ayudan a la clasificación, análisis y reconocimiento de patrones. Estos métodos y otros de Machine Learning se han comenzado a utilizar en el ámbito biológico, por ejemplo, en la identificación de patrones de DNA.

La unidad fundamental de la cromatina es el nucleosoma, complejo proteico formado por DNA e histonas y conforma el primer nivel de compactación del DNA en el núcleo. Se han identificado los patrones de secuencias promedio que contribuyen al posicionamiento de los nucleosomas a lo largo del genoma, pero no se ha desarrollado un método que permita el análisis individual de cadenas de nucleótidos para predecir qué secuencias se asociarán a un nucleosoma o no.

La posible solución propuesta tras la búsqueda de modelos tanto de machine learning y deep learning resultó en un clasificador de secuencias de nucleótidos que predice la probabilidad que sean nucleosomas o no. Este trabajo es un reto multidisciplinar que aplica la inteligencia artificial para dar solución a un problema real biológico.

Palabras clave: inteligencia artificial, machine learning, deep learning, nucleosomas, clasificación

Abstract

Artificial intelligence has been providing answers to problems that required human intelligence. Deep learning methods help in the classification, analysis, and recognition of patterns. These methods and other Machine Learning methods have begun to be used in the biological field, for example, in the identification of DNA patterns.

The fundamental unit of chromatin is the nucleosome, a protein complex formed by DNA and histones that forms the first level of DNA compaction in the nucleus. The average sequence patterns that contribute to the positioning of nucleosomes along the genome have been identified, but no method has been developed that allows the analysis of individual nucleotide chains to predict which sequences will or will not be associated with a nucleosome.

The possible solution proposed after searching both machine learning and deep learning models resulted in a classifier of nucleotide sequences that predicts the probability that they are nucleosomes or not. This work is a multidisciplinary challenge that applies artificial intelligence to solve a real biological problem.

Keywords: artificial intelligence, machine learning, deep learning, nucleosomes, classification

Lista de Abreviaturas

Abreviatura	Significado
URL	Uniform Resource Locator
DNA	Deoxyribonucleic Acid
RNA	Ribonucleic acid
pb	pares de bases
ML	Machine Learning
DL	Deep Learning
IA	Inteligencia artificial
ATP	adenosine triphosphate
NDR	Nucleosome Depletion Region
OHE	One Hot Encoding
Relu	Rectified Linear Unit
MLP	Multilayer Perceptron
DFFN	Deep Feed Forward Networks
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
API	Application Programming Interface

Tabla de contenido

Tabla de ilustraciones

1 Introducción

La tecnología se puede considerar una necesidad básica para el hombre, puede resultar beneficiosa y necesaria para seguir evolucionando en un mundo globalizado. Gracias a la tecnología se ha conseguido automatizar y agilizar procesos que resultaban tediosos para el ser humano, se han acortado distancias y se han disminuido tiempos de actuación. Se intentó ir un poco más allá y llegó la inteligencia artificial, definida como cualquier tarea realizada por una máquina que anteriormente se hubiera considerado que requería inteligencia humana.

La inteligencia va de la mano del conocimiento. Si no existe conocimiento, poco se puede hacer con la inteligencia. Dentro de la informática se conoce como 'Big Data' a los inmensos volúmenes de datos que se crean por personas y empresas y que junto al incremento de las velocidades de procesamiento han propiciado la revolución de la IA (Inteligencia artificial) que los convierten en información. Estos datos pueden ser aprovechados por las máquinas para encontrar programas que generen por sí solos hipótesis, software capaz de traducir en múltiples lenguas de manera instantánea o exhaustivos sistemas de reconocimiento facial.

Por otro lado, la genómica es una de las áreas de investigación donde más información se genera, consiste en el estudio de todos los genes (genoma) de los organismos. El material genético considerado como 'DNI' de un organismo está dentro de las células enormemente compactado, dando la identidad a este. Uno de los grandes retos en genómica ha sido conocer la posición exacta de estas letras, así como sus posibles mutaciones (cambios de letras) para estudiar defectos genéticos que a mayores provoque enfermedades. El primer gran hito en genómica fue el Human Genome Project, en el que se consiguió secuenciar completamente el genoma humano en 2003[1]. Además del genoma humano también se ha descifrado el de otros organismos, plantas, animales, bacterias, hongos, etc. La cantidad de datos que se recopilan es necesario que sea tratada, procesada y convertida en información relevante para las investigaciones científicas.

La inteligencia artificial ha ido resolviendo problemas que suponen un esfuerzo importante para los científicos. Retos que llevaban latentes muchos años. Uno de los más recientes ha sido dar una posible al "problema del plegamiento de proteínas". Las proteínas son largas secuencias de aminoácidos que sufren procesos de torsión y flexión para adoptar una forma tridimensional. Si ese proceso se ve interrumpido, la proteína no tendrá la forma correcta y no realizará su función. Con técnicas de aprendizaje profundo, el grupo DeepMind consiguió un programa que predecía el plegado de las proteínas[2]. Esto ayudará a los investigadores a comprender cómo funcionan las células y cómo las proteínas mal plegadas causan enfermedades.

En conclusión, las técnicas de aprendizaje automático capaces de clasificar, extraer patrones de los datos y explicar fenómenos gracias a los algoritmos subyacentes han podido

[1] https://www.oreilly.com/content/genomics-and-the-role-of-big-data-in-personalizing-the-healthcare-experience/

[2] https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology

resolver problemas científicos. A pesar de la complejidad de los datos biológicos tan heterogéneos, los sistemas actuales de aprendizaje profundo pueden realizar la extracción de características y la predicción de los datos, por ejemplo, una clasificación. (Han & Liu, 2019)

Por todo esto, se presenta en este trabajo fin de grado una posible solución a un problema biológico. El DNA está perfectamente empaquetado en la fibra de cromatina, formada por unas unidades llamadas nucleosomas. Estos nucleosomas ocupan posiciones muy estables a lo largo del genoma. Se ha estudiado el papel de los remodeladores y factores de transcripción que son dos de los determinantes de ese posicionamiento. ¿Pero qué pasa con la propia secuencia DNA en la que se enrrolla el nucleosoma? Este papel no está tan claro, pero es muy probable que la configuración de esa secuencia contribuya a determinar la posición del nucleosoma. Con este problema se planteó el reto de buscar aproximaciones de IA que permitieran identificar secuencias en las que se posicionaría un nucleosoma de las que no.

Si el modelo encontrado pudiera clasificar las secuencias con una exactitud adecuada, podría permitir a los investigadores explicar cómo contribuye la propia secuencia de DNA al posicionamiento nucleosomal. Para la generación del modelo se irá haciendo un recorrido por diferentes técnicas de aprendizaje automático, Machine Learning (ML) y Deep Learning (DL). Se compararán entre ellas con medidas como la exactitud y se mostrarán los resultados obtenidos que den la mejor solución conseguida para el problema. A parte de todo el análisis estadístico y los modelos de inteligencia artificial asociados se creó una página web para poder realizar pruebas con archivos csv para probar el mejor modelo conseguido, la dirección es la siguiente: 37.35.239.182:8080.

1.2. Motivación y Objetivos

El motivo que llevó a plantear este trabajo fue el interés que genera la aplicación de técnicas y avances informáticas en la biología. En los laboratorios empiezan las investigaciones siempre con hipótesis que no están resueltas y necesitan una solución que tras trazar un plan de acción se consigue refutar o apoyar.

El principal objetivo del proyecto es conseguir un modelo de aprendizaje automático que sea capaz de clasificar una secuencia de DNA como nucleosoma o no con un desempeño considerable. Con esto se automatizaría la decisión de si una secuencia de DNA posicionará un nucleosoma. Dentro de este punto se puede decir que también se quiere intentar extraer conocimiento de los datos que se proporcionan, así como realizar un preprocesamiento adecuado de los mismos y conseguir con ellos un modelo que sea capaz de dar una posible respuesta al problema planteado inicialmente.

Otro objetivo es conseguir entender un problema biológico. Conocer las bases biológicas, poderlo presentar y ser capaz de tratar con información que no está dentro del campo informático propiamente.

El último objetivo es aprender a crear modelos tanto de ML y DL, campo que no se ha tratado a lo largo del grado y que suscita gran interés debido a sus posibles aplicaciones.

2 Conceptos teóricos

2.1. Conceptos biológicos

En este apartado se van a explicar los conceptos del campo de la biología más importantes para poder entender el problema que se quiere tratar sin tener un conocimiento previo sobre este campo.

2.1.1. Levadura Schizosaccharomyces pombe

Es un hongo unicelular eucariota. Adopta una forma de bastón y su longitud es de 7 a 14 micrómetros con un diámetro de 3 a 4 micrómetros. Adoptó el nombre "*Pombe*" en 1893 que significa cerveza en el idioma swahili debido a su descubrimiento por primera vez en una cerveza africana.

Contiene el menor número de genes codificantes de proteínas registrado hasta el momento en eucariotas: 4824. Cincuenta genes tienen una similitud significativa con genes de enfermedades humanas; la mitad de ellos están relacionados con el cáncer. También se han encontrado genes altamente conservados que son importantes para la organización de las células eucariotas, incluidos los necesarios para el citoesqueleto, la compartimentación, el control del ciclo celular, la proteólisis, la fosforilación de proteínas y el empalme del RNA. Es por este motivo que se suelen usar como modelos en muchos experimentos de organismos eucarióticos. (V Wood, 2002)

2.1.2 Estructura y función del nucleosoma

Se estima que se podría ir unas 300 veces al Sol y volver con la longitud de DNA que tiene un cuerpo humano. Para ser exactos tenemos 100 trillones de metros de DNA, contenido en 50 trillones de células. Se estima que cada célula diploide tiene dos metros de longitud formado por pares de bases de 0.34 nm. En cada célula hay unos 6 billones de pares de bases por célula empaquetados formando 23 cromosomas con células diploides a excepción de los espermatozoides y los óvulos. (Annunziato, 2008)

Los dos metros de cadena de DNA que contiene la información genética de cada célula diploide y regula las funciones de los seres vivos pasa por un proceso complejo de empaquetamiento de varios órdenes de longitud en el núcleo de las células. **(Figura 1)**

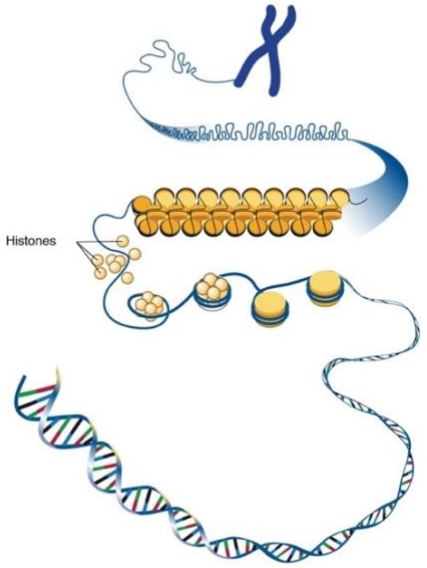


Figura 1. Fases de compactación del genoma desde el cromosoma hasta la doble hélice de DNA. Recuperado de https://www.genome.gov/genetics-glossary/Chromatin

El DNA empaquetado se llama cromatina. El elemento principal de la cromatina es el nucleosoma. Es un complejo de ocho histonas rodeadas de una cadena doble de DNA de aproximadamente 150 nucleótidos (Antequera, 2017). En concreto el octámero de histonas está formado por dos moléculas de cada una de las histonas H2a, H2b, H3 y H4. Además, la unión de la proteína H1 envuelve otros 20 pares de bases, lo que da dos vueltas alrededor del octámero **(Figura 2)** formando la estructura conocida como cromatosoma.

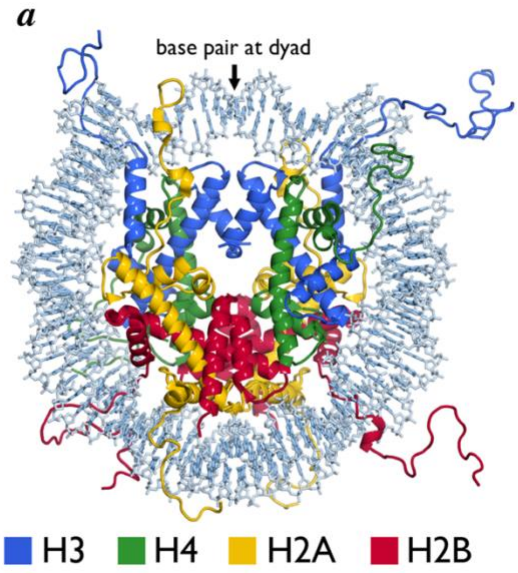


Figura 2. (McGinty & Tan, 2014). Núcleo de la estructura junto a los cuatro heterodímeros de histonas.
La estructura de color gris azul/gris de la periferia es la cadena de DNA. Los cuatro heterodímeros son los que están representados por colores: H3(azul), H4(ve H4(verde), H2A(amarillo) y H2B(rojo). Recuperado de https://pubs.acs.org/doi/pdf/10.1021/cr500373h

El nucleosoma tiene tres funciones principales. La primera de ellas tiene que ver con el primer nivel de compactación del genoma. También actúa como centro de señalización para los procesos que tienen lugar en la cromatina. En tercer lugar, el nucleosoma puede ensamblarse sin ayuda de otras estructuras en cromatina de orden superior con el fin de aumentar la compactación del genoma. (Mc Ginty & Tan, 2014)

2.1.3. Posibles causantes del posicionamiento de los nucleosomas

Los nucleosomas se repiten a lo largo del genoma cada 160 a 240 pares de bases (pb). Para establecer con claridad esa distribución es importante conocer el concepto de posicionamiento. Este se refiere a la localización que tiene el nucleosoma en la cadena del DNA. El nucleosoma se dice que está "bien posicionado" cuando se coloca sobre la misma secuencia de DNA en todas las células de una población. **(Figura 3)**

Existen diferentes factores que favorecen el posicionamiento nucleosomal. Los mismos se describen a continuación.

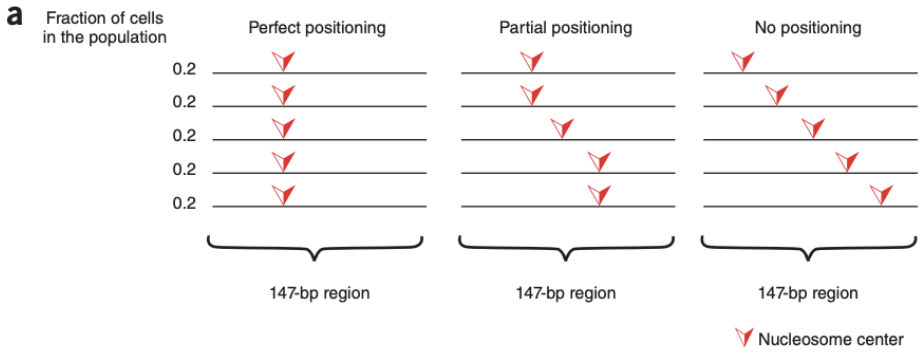


Figura 3. Struhl & Segal (2013). Posicionamiento del centro del nucleosoma de una región de 147 bp a lo largo de las diferentes células de la población. En la imagen de más a la izquierda vemos como están perfectamente posicionados, en la segunda se posicionan de forma parcial y en la última no están posicionados. Recuperado de https://www.nature.com/articles/nsmb.2506.pdf

Secuencia de DNA

No está del todo claro en la literatura si existe una relación directa entre la secuencia del DNA y el posicionamiento de los nucleosomas a lo largo del genoma. Existen estudios que afirman que la secuencia del DNA sólo especifica la localización de las regiones que excluyen a los nucleosomas, distribuyéndose luego estos con cierto criterio, pero de formas más bien difusas sin patrón claro. (Radwan et al., 2008)

Con el fin de permitir el empaquetamiento existen dos características determinantes a la hora de la flexión de la cadena de DNA y, por tanto, la formación de nucleosomas. En primer

lugar, se sabe que los dinucleótidos flexibles (AT y TA) se encuentran cada 10 bp interactuando con las histonas. (Segal K. S., 2013) En segundo lugar, las secuencias homopoliméricas[3] poli (dA:dT) y poli (dG:dC) son rígidas por diferentes razones estructurales y son principalmente supresoras de la formación de nucleosomas.

Al igual que las proteínas de unión al DNA, las preferencias de las secuencias de DNA para los nucleosomas pueden expresarse como una matriz de puntuación para cualquier región de longitud 147 bp. Como la afinidad de los dinucleótidos varía a lo largo de la secuencia, los nucleosomas tendrán una afinidad diferente de unos nucleosomas a otros. **(Figura 4)**

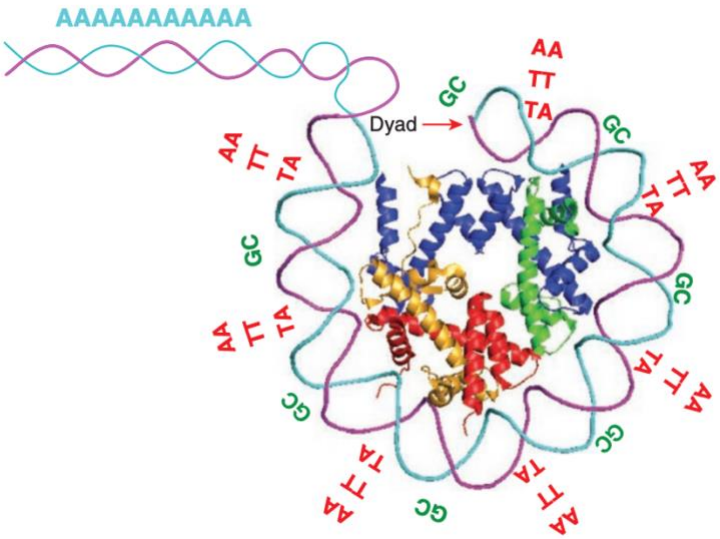


Figura 4. Struhl & Segal (2013). Preferencias de secuencia del nucleosoma.Existe una tendencia hacia un patrón de dinucleótidos en la secuencia de 147 bp En concreto se observan cada ~10 bp los dinucleótidos TA, AA O TT seguidos de GC.

A pesar de que varían las matrices de puntuación, si se tiene en cuenta el posicionamiento de las ubicaciones genómicas a 10 pb de distancia tendrán puntuaciones de formación de nucleosomas similares a menos que las secuencias cercanas tengan secuencias inhibidoras.

No solo se ha visto que influye la frecuencia de los dinucleótidos en la secuencia de DNA, sino que hay otro tipo de perfil de alta frecuencia en estas secuencias de DNA asociado al posicionamiento de los nucleosomas: las *signatures* nucleosomales. (González et al., 2016). Esta investigación se llevó a cabo en el laboratorio con el que se ha colaborado para la realización de este trabajo de fin de grado, el laboratorio del Dr. Antequera del IBFG de Salamanca.

[3] polímero constituido por la repitición de un único monómero (cadena homogénea)

Las *signatures* marcan un perfil de A predominante en el extremo 5' y T en el extremo 3'. En concreto, en *S.Pombe* está caracterizado por mayor contenido de A y T en la posición central (diada) y en los extremos, junto al DNA de enlace al siguiente nucleosoma (linker) entre los que se intercalan regiones de G y C. Sin embargo, no está claro si esto es una marca que se ha generado debido a la estabilidad de los nucleosomas a lo largo de la evolución o realmente contribuyen al posicionamiento de los nucleosomas en las diferentes especies.

Remodeladores de cromatina

Los remodeladores de cromatina son complejos multiproteicos que gracias a su acción ATPasa junto a la liberación de energía por la hidrólisis de ATP, permite al complejo modificar la estructura del nucleosoma. Las observaciones llevadas a cabo proponen que estos complejos no facilitan el movimiento de los nucleosomas a lugares preferentes, sino que son críticos a la hora de establecer la especificidad del sitio de localización de los nucleosomas.

Como los nucleosomas suelen tener una longitud fija, también suelen tener un patrón fijo de separación entre ellos, a lo que pueden contribuir estos remodeladores. Se ha observado una importante alteración del posicionamiento de nucleosomas una vez que se han retirado de la cadena los remodeladores en *S.Pombe*. Esto evidencia el hecho de que hace falta los remodeladores de cromatina para el posicionamiento de nucleosomas en la cadena de DNA.

Factores de transcripción

El posicionamiento de los nucleosomas puede estar determinado por los factores de transcripción que se unen a sitios específicos de la secuencia de DNA. No solo estos factores evidencian los sitios de unión, sino que dejan al descubierto los sitios irrelevantes, es decir, sitios no funcionales. Por eso, una de las posibles hipótesis puede ser que el genoma utiliza la organización de los nucleosomas para establecerlos en zonas no funcionales, para disminuir la accesibilidad de los factores de transcripción. Esto se ha probado en 46 sitios de factores de transcripción. En 17 la ocupación nucleosomal, es decir, la cantidad de nucleosoma presente en una determinada localización era considerablemente más baja comparada con la basal y solo 2% de esos sitios presentaban una ocupación superior a la común. (Segal et al., 2006)

Otros estudios realizados evidencian que la competencia junto a las histonas por la unión del DNA puede generar regiones libres de nucleosomas (Nucleosome Depleted Regions - NDRs) en el genoma. Sin embargo, la eliminación de los factores de transcripción tiene un efecto vago en los NDRs. (Yen et al., 2012)

Con todo esto se puede decir que puede ser que el hecho de que exista una secuencia favorable al posicionamiento nucleosomal evita la unión de factores de transcripción, o por otro lado que el hecho de que existan lugares específicos en la cadena de DNA de unión de

esos factores implique una barrera para el posicionamiento de los nucleosomas. (González Moreno, 2017)

2.2. Aspectos informáticos

En este apartado se van a explicar los términos y técnicas computacionales más relevantes usadas a lo largo del proyecto, así como su desarrollo matemático necesario para una mejor explicación del desarrollo posterior.

2.2.1. Inteligencia artificial

Puede resultar complicado encontrar una definición exacta. Pero quizá, se puede definir la inteligencia artificial como el campo de estudio sobre cómo hacer que los ordenadores hagan cosas, que, por el momento, son realizadas por los humanos. (Rich, 1985)

Alan Turing, famoso matemático británico fue uno de los primeros en encontrar evidencias matemáticas para este aprendizaje automático. Su hipótesis era que al igual que los humanos tienen la capacidad de resolver problemas y tomar decisiones, las máquinas también podrían llegar a hacerlo. Poder hacer realidad ese hecho no era tan fácil. En aquellos años existían varios problemas, uno de ellos era que las computadoras entonces solo podían ejecutar comandos, no almacenarlos y otro gran problema era disponer de la solvencia económica suficiente para adquirir una computadora, teniendo en cuenta la cantidad de ordenadores que se vendían en la época.

En los años 80 hubo un fuerte impulso económico que desencadenó varias tendencias importantes. La primera fueron las "técnicas de aprendizaje profundo" en las que la máquina aprendía de la experiencia. Y después llegaron los llamados "sistemas expertos". Estos programas tendrían almacenada la respuesta de expertos a preguntas pre-formuladas con

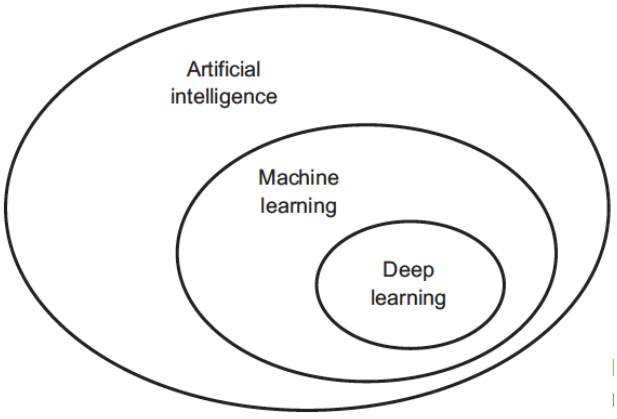


Figura 5. Chollet, F (2018) Artificial intelligence, Machine Learning and Deep Learnig

anterioridad. Una vez aprendida casi la totalidad de los casos, el programa se podría usar por una persona no experta en el ámbito a tratar. Se puede decir que no era inteligencia artificial, sino un conjunto de datos y algoritmos capaces de extraer información a partir de datos de materias concretas, es decir, necesitan conocimiento previo.

No ha habido un salto en cuanto a algoritmos informáticos de IA, pero la Ley de Moore, que calcula que la velocidad y la memoria de los ordenadores se duplica cada año, ha ayudado a este gran salto que no se pudo hacer en los años 70. Así, en 1997 fue la primera vez que un campeón de ajedrez, Gary Kasparov, era derrotado por una máquina, la conocida "Deep Blue" de IBM. Y en 2017 Alpha Go Google pudo derrotar al campeón chino de Go, Ke Jie. (Intelligence, 2017)

Pero ya no solo se habla de IA, sino que también existen dentro de la inteligencia artificial dos campos: machine learning (ML) y deep learning (DL). **(Figura 5)**

2.2.2. Machine Learning

El aprendizaje automático ya no solo se rige por las reglas específicas de un algoritmo que tiene programado, sino que es él mismo es el que busca las reglas tras un proceso de "entrenamiento". En contraste con la programación clásica en la que las personas llaman a unas reglas (programa) junto con unos datos de entrada y se extraen unas respuestas, en ML se proporcionan los datos junto a las respuestas esperadas de esos datos y el programa es el encargado de buscar unas reglas que podrán ser aplicadas en nuevos casos prediciendo la salida. **(Figura 6)**

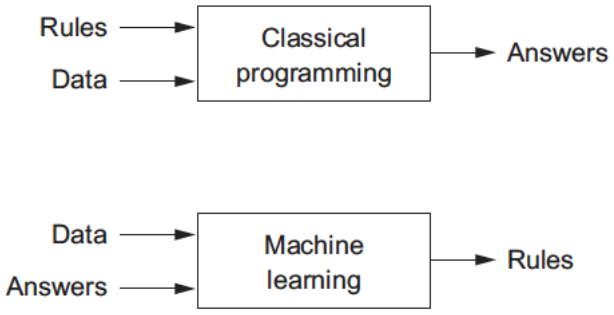


Figura 6. (Chollet. F, 2018) Esquema diferenciador programación clásica y Machine Learning

A los sistemas de ML se les entrena, no programa, con una serie de ejemplos para la tarea que queremos lograr y busca mediante métodos estadísticos una serie de reglas que le permiten llegar a partir de los datos de entrada a las respuestas proporcionadas para posteriormente automatizar esa tarea. A pesar de la carga estadística, los sistemas de ML son

capaces de lidiar con cantidades de datos mucho mayores a lo que sería capaz un método estadístico clásico.

A la hora de enfrentarse a un reto de ML se necesitan tres cosas:

- **Datos de entrada:** pueden estar en diferentes formatos como texto, imágenes o audio.
- **Conjunto de salidas esperadas:** pueden ser una clasificación de imágenes realizadas por humanos, comprobación de experimentos de un laboratorio, datos extraídos de un análisis clínico, etc.
- **Métricas:** es la clave para que el algoritmo "aprenda". Las métricas dan una aproximación de la similitud entre los resultados obtenidos y los esperados. A partir de ellas el algoritmo realiza ajustes para mejorar su aprendizaje.

No solo es importante tener una cantidad considerable de datos sino la representación de estos. Puede haber diferentes tipos de representación para un mismo problema, pero quizá haya una representación que se ajuste mejor a la tarea que queremos resolver. ML busca encontrar técnicas para la búsqueda de la representación óptima de los datos que recibe. Los algoritmos no suelen hacer una búsqueda estocástica, sino que suelen partir de un espacio de posibles hipótesis para el problema planteado. (Chollet, 2018)

2.2.2.1. Tipos de aprendizaje

Existe un amplio abanico de aplicaciones de ML, lo que lleva a diferentes tipos de aprendizaje habiendo una división en dos grupos principales.

Supervisado versus No-Supervisado

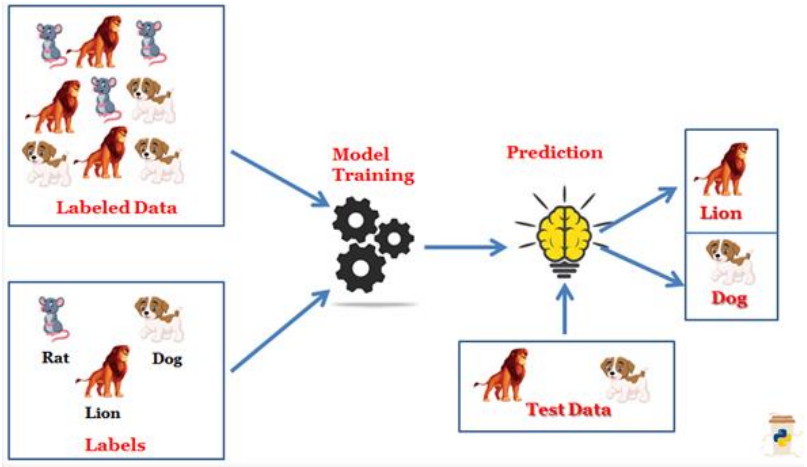


Figura 7. Esquema aprendizaje supervisado. Extraído de https://laptrinhx.com/supervised-and-unsupervised-machine-learning-1391659628/

El aprendizaje supervisado tiene ejemplos al inicio para poder aprender a partir de ellos. El programa etiqueta diferentes categorías (clasificación) y los compara con los datos ya etiquetados desde el principio. **(Figura 7)** En cambio, en el no supervisado hay solo datos de entrenamiento en la entrada con los que el algoritmo de ML tiene que intuir algún patrón o reglas solo con ellos sin un f*eedback* ya creado, es decir, no hay una base previa de conocimiento. Como ejemplo de este tipo de ML está el problema de *clustering* que no es más que crear grupos a partir de ciertas características similares entre sí.

Existe un tercer paradigma de ML que se podría incluir cercano a estas dos categorías. Es el aprendizaje por refuerzo **(Figura 8),** consiste en aprender a decidir, ante una situación determinada, qué acción es la más adecuada para lograr un objetivo. Este aprendizaje está inspirado en la psicología conductista porque es un modelo acción-recompensa, que busca que el algoritmo se ajuste a la "mejor recompensa" que pueda dar el ambiente. Este tipo de paradigma suele estar relacionado con los robots y su ambiente, como el espacio donde tienen capacidad de interactuar. (Shalev-Shwartz et al., 2014)

Figura 8. Diagrama de flujo de aprendizaje por refuerzo

2.2.2.2. Overfitting, Underfitting y Capacidad

El objetivo principal del ML es poder encontrar una serie de reglas que sean capaces de generar buenos resultados para un conjunto de datos nuevos que no se hayan mostrado con anterioridad al sistema, la llamada generalización.

Antes de poder definir los conceptos de esta sección hay que tener claro los dos errores de predicción más comunes: bias y varianza. El bias se asume como la diferencia que existe entre la predicción media de nuestro modelo y la predicción correcta. La varianza indica la dispersión de nuestros datos. Cuanto mayor sea la varianza del modelo mejor se ajustará a nuestros datos, pero no generalizará para datos nuevos. **(Figura 9)**

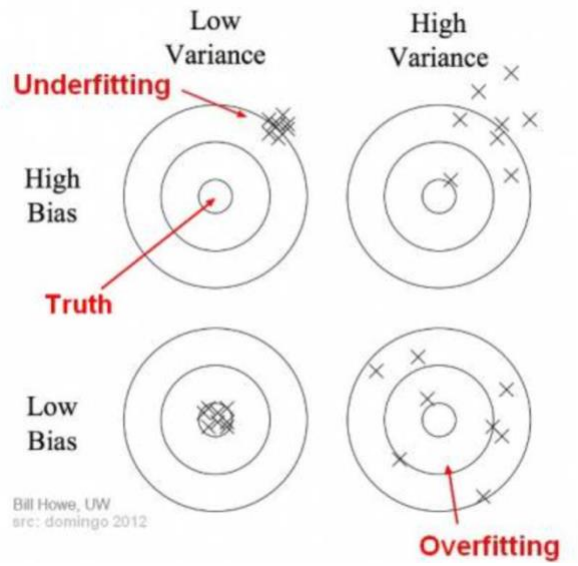


Figura 9. (Howe, Bill, 2012) Diagrama que muestra en el centro con una predicción correcta, cuanto más se aleja del centro la predicción va empeorando.

Como norma general lo primero que se hace es entrenar al sistema y reducir el error de entrenamiento lo máximo posible a lo largo de este, por lo que tiene una similitud con un problema de optimización. Pero, a mayores de la optimización se busca reducir el error de generalización, también conocido como error de test.

Para realizar la distribución de los datos de entrenamiento y de test se deben asumir dos cosas: que los datos de cada conjunto son independientes y que ambos datasets están idénticamente distribuidos. Todo esto permite definir el proceso de generación de datos con una probabilidad estadística conocida como distribución de la generación de datos, denotada por p_{data}.

Tanto el conjunto de test como el conjunto de entrenamiento se llevan a cabo con el mismo proceso de división. El proceso se lleva a cabo dividiendo primero el conjunto de entrenamiento, eligiendo los parámetros para entrenar al modelo y finalmente se divide el conjunto de test. Es por ello por lo que suponemos que el error de test va a ser mayor o igual que el generado en el conjunto de entrenamiento. Con todo esto en cuenta, los dos puntos clave que determinan mejor el desempeño de un algoritmo de ML son:

1. **Minimizar el error de entrenamiento**
2. **Minimizar la brecha entre el error de test y de entrenamiento**

Ahora podemos introducir los dos retos del ML; el underfitting y el overfitting. El underfitting ocurre cuando es posible reducir lo suficiente el error de entrenamiento y en cambio, el overfitting se produce si la diferencia entre el error de test y de entrenamiento es muy grande.

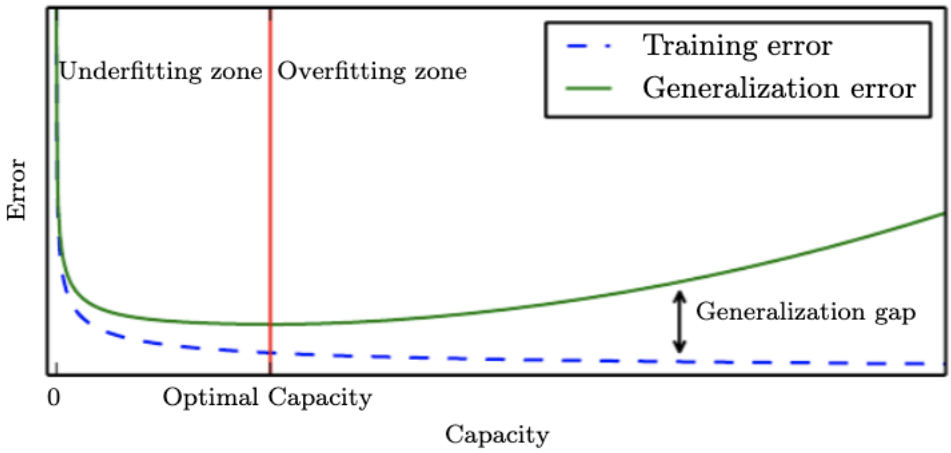


Figura 10. (Goodfellow et al., 2016). Gráfico que relaciona el error frente a la capacidad de un algoritmo de ML.

Se puede reducir tanto el overfitting como el underfitting con la capacidad, que es la habilidad para ajustar el modelo a una gran variedad de funciones. Esa capacidad depende del espacio de hipótesis que se tiene en cuenta como posible solución para entrenar al modelo. El espacio de hipótesis se define como el conjunto de funciones que se pueden considerar a la hora del aprendizaje como posibles soluciones del problema que se está tratando. Si la capacidad es muy baja el modelo puede tener problemas de ajuste y si por el contrario la capacidad es muy alta el modelo puede sufrir overfitting.

	Underfitting	Overfitting
↑	Bias	Varianza
↑	Error de entrenamiento	↓ Error de entrenamiento
↑	Error de test	Error de test

Teniendo en cuenta todo esto podríamos decir que la capacidad óptima de un sistema es el punto en el que el error es suficientemente pequeño sin que aumente el error de generalización en comparación con el de entrenamiento.

En el gráfico superior **(Figura 10)** se pueden distinguir diferentes casos posibles. A la izquierda tanto el error de entrenamiento como el error de generalización son altos, por tanto, estamos en la región de underfitting. A medida que se aumenta la capacidad el error de

entrenamiento disminuye, pero la brecha entre ambos errores es mayor, generando overfitting.

2.2.2.3. Conjuntos de datos de entrenamiento, validación y testeo

En ML con aprendizaje supervisado se suele diferenciar entre conjuntos de datos:

- Conjunto de entrenamiento: será el conjunto de datos con los que el modelo se entrenará y aprenderá.
- Conjunto de validación: será el que dará el error de pérdida, diferencia entre valor real y valor predicho; gracias al que se volverán a calcular los parámetros del modelo.
- Conjunto de test: será el que quede sin usarse durante el entrenamiento del modelo y con el que se hará las predicciones para obtener luego las métricas para comparar los diferentes modelos.

Los conjuntos de validación y test son considerablemente más pequeños del conjunto de entrenamiento, aunque su tamaño no está prefijado de antemano; sino que se elige en función

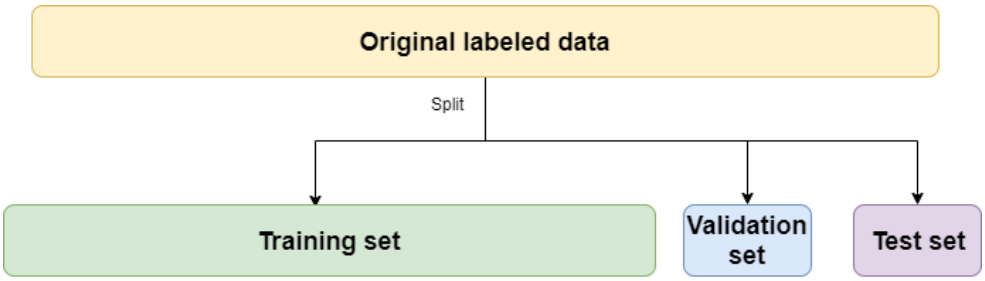


Figura 11. Representación del conjunto de entrenamiento, validación y testeo a partir del dataset original

del número de datos que tengamos. También los tres conjuntos deberían ser representativos, debería estar equilibrado el número de datos de las diferentes categorías (en el caso de una clasificación).

2.2.2.4. Optimización de hiperparámetros

Los algoritmos de machine learning estiman variables a partir de los datos que se le pasan. Estos no se pasan al modelo, sino que son intrínsecos al mismo. Estas variables son los **parámetros** del modelo. Sin embargo, hay otras variables que se deben fijar antes de entrenar el modelo. Estos son los **hiperparámetros** y de ellos depende, en muchas ocasiones, el rendimiento del algoritmo. Por ello, el científico de datos puede optimizar y cambiar estos valores. Los métodos para optimizar los hiperparámetros requieren de un espacio de búsqueda entre posibles valores, a parte de los que se dejen por defecto de acuerdo con las librerías de Python.

Dejar los hiperparámetros a su valor por defecto puede a veces no interferir en el resultado, pero normalmente es necesario modificar alguno de ellos para mejorar los resultados obtenidos. Todo depende del algoritmo y qué es lo que se busca. (Weerts et al.)

El proceso de elección tiene como meta encontrar los mejores hiperparámetros. Para esta búsqueda se puede realizar de forma manual, un trabajo tedioso si se hace sin mirar la literatura anterior o basándonos en la literatura con modelos ya realizados de prueba-error. O también hay métodos que automatizan este proceso.

En el desarrollo de los modelos de ML, se verá que se ha usado **Grid Search**, pero, también son conocidas Random Search, que se hace por aleatorización o optimización bayesiana. Este método, presente en la biblioteca Scikit-Learn tiene un diccionario con el espacio de búsqueda de los diferentes hiperparámetros, que se irán recorriendo con todas las diferentes combinaciones de posibilidades para encontrar los mejores valores para el modelo. Lo que facilita la tarea de tener que ir introduciendo manualmente las diferentes combinaciones, ya que, se realiza de forma automática. **(Figura 12)**

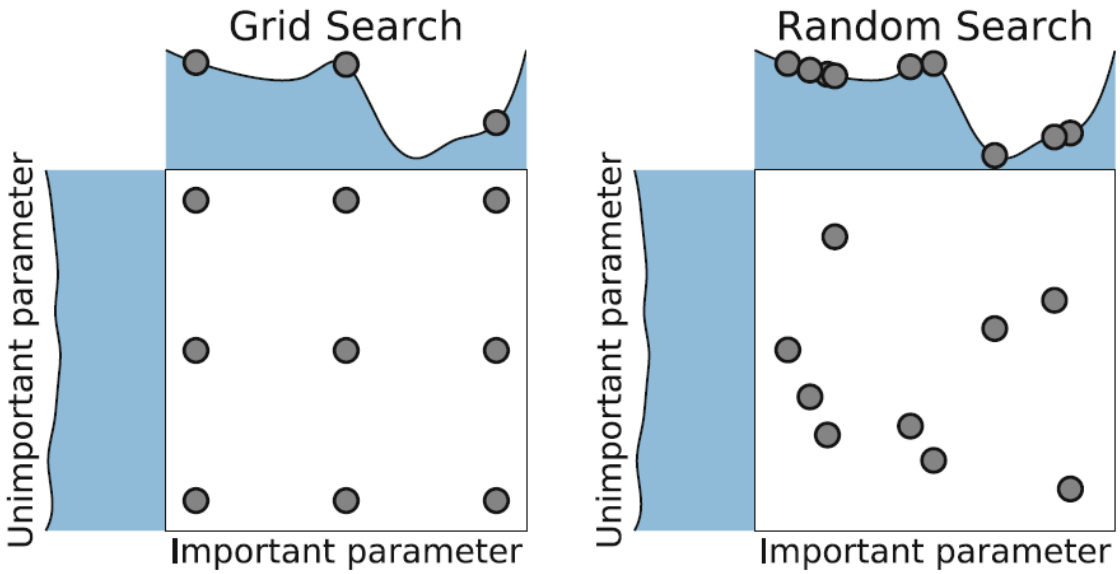


Figura 12. Grid Search vs Random Search en optimización de hiperparámetros. Recuperada de https://tex.stackexchange.com/questions/571121/grid-search-vs-random-search-or-how-to-draw-multiple-functions-near-a-matrix-in

2.2.2.5. Técnicas de Machine Learning

En este apartado se van a introducir las diferentes técnicas de ML que posteriormente se han usado para dar una posible solución al problema planteado en este trabajo. Todas estas técnicas pertenecen al grupo de aprendizaje supervisado.

Random Forest

El algoritmo de Random Forest (Breiman, 2001) surge como solución a la necesidad de mejorar los modelos de árboles de decisión. Estos son en esencia un diagrama donde un nodo

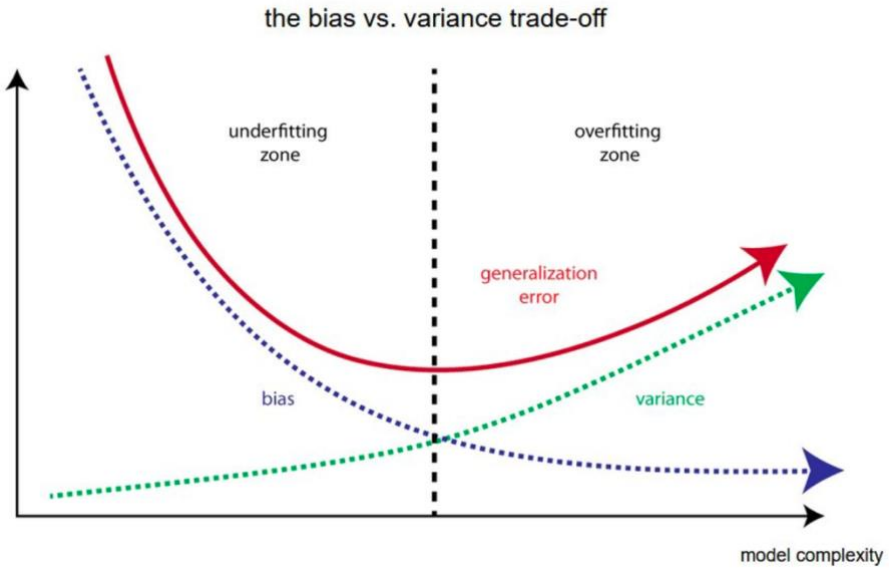


Figura 13. Bias vs variance trade off. Se muestra en la imagen cómo cuando hay bajo bias (sesgo) y alta varianza se produce over-fitting, de modo contrario al under-fitting.

interno representa una característica (o atributo), la rama representa una regla de decisión y cada nodo hoja el resultado en función de aplicar la regla al nodo interno. Esta estructura se basa en toma de decisiones, pero sufre problemas de sesgo o bias (cuánto en promedio son de diferentes los valores predecidos de los valores reales) y varianza (cuánto de diferentes serán las predicciones de un modelo en un punto si las muestras se tomaran de la misma población). Si se construye un árbol pequeño tendrá una baja varianza y alto sesgo. Cuanto más complejo sea el modelo habrá menos sesgo, pero puede llegar a haber un sobreajuste (over-fitting). El modelo óptimo debería ser un balance entre estos dos errores y se conoce como "trade off" **(Figura 13)**

El modelo se inicia con un conjunto de entrenamiento que tiene n observaciones, la variable de interés Y (cada una de las categorías) y las predictoras que es un array de X El proceso es el siguiente:

1.- Se construye un conjunto nuevo de entrenamiento con la técnica **Bootstrap**. Se toma una muestra de N casos aleatoriamente con reemplazo y esta será la muestra para construir el árbol i. **(Figura 14)**

Figura 14. Técnica Bootstrap

2.- Se construye un árbol usando en cada partición un subconjunto de las variables de entrada tomadas.

3.- Se repiten los pasos R veces.

Para predecir la variable Y que es la que luego se comparará con el conjunto de datos de test para evaluar la precisión se toman los R árboles creados y se predice la variable Y. Teniendo tantas predicciones de Y como árboles se hayan creado (R). Luego en el caso de problemas de clasificación se toma la clase que ha salido más veces para predecir la variable Y para ese conjunto de Bootstrap.

Es un buen modelo para clasificación, pero tiene el problema de su naturaleza de caja negra que es difícil de interpretar cómo ha realizado la predicción de variable Y.

Logistic Regression

Este modelo de clasificación estadístico mide la probabilidad de que cierto evento suceda (nucleosoma/no nucleosoma). Se llaman modelos binarios porque también puede existir otros que sean capaces de clasificar varias categorías, multinomial.

Matemáticamente, la regresión logística binaria tiene una variable dependiente (y) con dos posibles valores representados por "0" y "1" (variables categóricas). El logaritmo de la probabilidad para que la variable dependiente tome el valor 1 es una combinación lineal de una o más variables independientes (X), que pueden ser binarias (números enteros) o continuas (valores reales). La probabilidad para que sea el valor "1" puede variar en un intervalo entre 0-1. La función que convierte las probabilidades logarítmicas en probabilidades es la función logística. (Tolles & Meurer, 2016)

Función logística

La entrada a esta función es un conjunto de valores reales de entrada que darán una salida que va entre 0 y 1. La función logística se representa:

$$\sigma: \mathbb{R} -> (0,1)$$

Siendo esta:

$$\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$$

La función sigmoide σ en un intervalo t se ve en la **figura 15.**

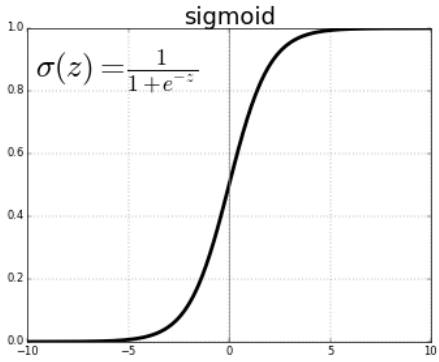


Figura 15. Función sigmoide. Extraído de: https://ml4a.github.io/images/figures/sigmoid.png

Suponemos que t es una función linear con una variable independiente x, por tanto, la función t se representa como:

$$t = \beta_0 + \beta_1 x$$

Siendo β_0 el valor de t cuando la variable independiente (x) es 0. β_1 son coeficientes fijos que multiplican a las diferentes variables independientes del modelo para poder hallar el valor de t en cada momento. Ahora podemos escribir ya la función logística como:

$$p(x) = \sigma(t) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

La probabilidad p(x) de que la variables dependiente Y sea éxito ("1") frente a fracaso ("0") dependiendo de cuál sean las variables independientes (x) y los parámetros fijados $(\beta_0, \beta_1, \beta_2 \ldots)$ (Freedman, 2009)

Representación modelo

Entendido dentro del campo de la inteligencia artificial es una red neuronal con una única neurona y su función de activación es una sigmoide. En nuestro caso los valores de entrada

(X) son las diferentes secuencias de DNA de las que se extraerán las características. Como se ha visto en el apartado anterior la regresión logística se compone de dos partes:

- combinación lineal con los diferentes parámetros y las variables independientes
- función logística (sigmoide)

De estas dos funciones viene el nombre regresión logística. "Regresión" de la primera parte que es similar a una regresión lineal y después logística por la sigmoide.

KNN

Uno de los modelos más sencillos de entender y más extendidos. La idea básica sobre la que se fundamenta el algoritmo es que un nuevo dato se va a clasificar como la clase más frecuente que presentan sus K vecinos más cercanos.

Como entrada tenemos una serie de variables X con su correspondiente clase C para todos los datos de entrada N. Para cada nuevo caso X se calculan las distancias de los casos ya calculados al caso actual y una vez que ya tenemos los K casos seleccionados más cercano, se le asigna la clase C más frecuente de entre esas K. **(Figura 16)**

COMIENZO
Entrada: $D = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_N, c_N)\}$
$\mathbf{x} = (x_1, \ldots, x_n)$ nuevo caso a clasificar
PARA todo objeto ya clasificado (x_i, c_i)
calcular $d_i = d(\mathbf{x}_i, \mathbf{x})$
Ordenar $d_i (i = 1, \ldots, N)$ en orden ascendente
Quedarnos con los K casos $D_{\mathbf{x}}^K$ ya clasificados más cercanos a $\mathbf{x}$
Asignar a $\mathbf{x}$ la clase más frecuente en $D_{\mathbf{x}}^K$
FIN

Figura 16. Pseudocódigo para el algoritmo K-NN

En caso de que se produzca un empate en las clases más frecuentes de los K vecinos más cercanos, se debería implementar una heurística[4]. Esta puede ser, por ejemplo, coger la clase del vecino más cercano. (Cost & Salzberg, 1993)

Decision Trees

Es un modelo predictivo usado en minería de datos, estadística o aprendizaje automático. Si la variable destino que se busca toma una serie de variables categóricas o finitas se trata de árboles de clasificación. Si, por el contrario, se trata de variables continuas son árboles de

[4] Heurística en IA se conoce como un conjunto de métodos o reglas pertenecientes al problema

regresión. Su uso se ha extendido debido a su sencillez y posibilidad de entendimiento en comparación de otros algoritmos que son prácticamente "cajas negras".

El objetivo es encontrar una variable dependiente o categoría que corresponda a los datos iniciales. Para conseguirlo se hacen podas continuas al espacio de datos inicial. Existen un nodo raíz, es el primer nodo del árbol que no tiene predecesores y a partir del cual se hace la primera poda. Además, están los nodos internos que son nodos donde se divide el dataset en subconjuntos de acuerdo con algún criterio o el valor que toma cierta variable, es el punto donde se va realizando la toma de decisiones. Se va avanzando por las diferentes ramas hasta llegar a un nodo hoja que ya no tiene sucesores, este nos dará la categoría asociada. Se supone que tiene que haber tantos nodos hojas como categorías a clasificar. **(Figura 17)** (Rokach & Maimon, 2010)

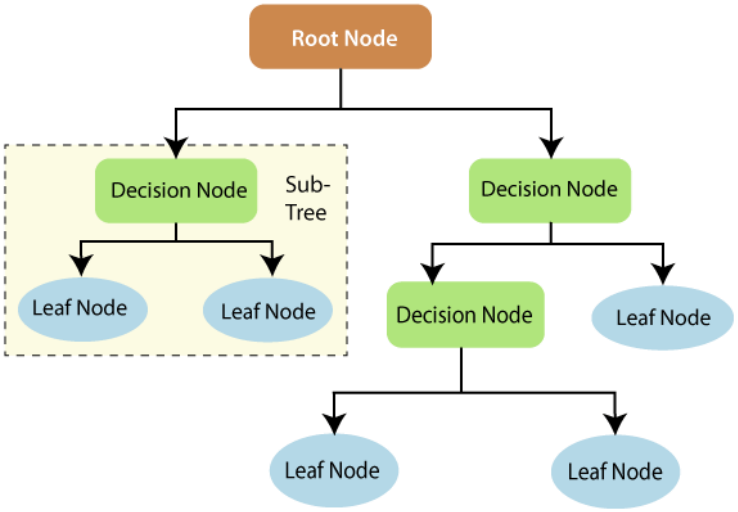


Figura 17. Representación árbol de decisión. Extraído de: https://laptrinhx.com/decision-trees-in-machine-learning-1041658267/

La complejidad del árbol se puede deber a diversos factores: número de nodos, número de nodos hojas, profundidad del árbol y número de atributos del dataset inicial. Como señaló Breiman et al. (1984) esta complejidad puede afectar al accuracy del modelo posteriormente. Cada camino del nodo raíz a uno de los nodos hoja se podría transformar en una regla heurística. La finalidad más útil de estas reglas sería la compresión por parte de los seres humanos de las reglas para aplicar el modelo.

Ventajas	Desventajas
• Si el número de nodos internos no es muy grande se puede interpretar el modelo de manera relativamente sencilla	• Usa el método "divide y venceras" que suele funcionar bien si existen muchas características iniciales (variables independientes) sino tiende a fallar
• Puede tener variables enteras y continuas	• Problemas de overfitting
• Data set puede tener valores nulos • Modelo no paramétrico, esto significa que los datos inicialmente no se ajustan a una distribución definida a priori.	• Difícil cálculo si existen muchas clases a clasificar • Suele presentar un accuracy inferior a otros modelos de aprendizaje supervisado

Naïve Bayes

Técnica muy usada como clasificador debido al uso de la métrica de máxima verosimilitud que toma la categoría con más probabilidad, aunque existen modelos que la están usando también para regresión. Este conjunto de métodos se basa en el teorema de Bayes y asumen que el valor de una característica es independiente de cualquier otra característica. Todas contribuyen de forma autónoma a la probabilidad de que la clase resultante sea 'X'.

Es bastante eficaz en el aprendizaje supervisado y requiere un conjunto no muy grande de datos de entrada para poder inferir los parámetros necesarios para realizar la clasificación en categorías.

Teorema de Bayes

Describe la probabilidad de un evento teniendo conocimiento de eventos previos relativos al evento a determinar. Se expresa mediante la siguiente fórmula: (Stuart & K.Ord, 1994)

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

- A y B son dos eventos, con la probabilidad $P(B) \neq 0$
- P(A|B) es una probabilidad condicional a posteriori, la distribución de la probabilidad final.
- P(B|A) es la probabilidad de que ocurra B siendo A verdadera
- P(A) y P(B) es la probabilidad de observar ambos eventos sin más condiciones. Ambos tienen que ser eventos diferentes

La variable dependiente A, con cierto número de clases (a clasificar), la variable está condicionada por varias variables independientes B. Como se ha indicado se asume que las diferentes variables B para el problema son independientes entre ellas.

Bayes lo que afirma es que se puede dar una cierta hipótesis A si algún evento B se ha producido. Al contrario de otros modelos que lo que hacen es inferir los efectos dadas las causas. Bayes lo que intenta es conocer la probabilidad de las causas sabiendo el efecto que produce.

Una vez se ha introducido los conceptos básicos del teorema de Bayes, ampliamente utilizado por su asunción de que cada variable es independiente del resto y su contribución a la probabilidad también, se ha construido un modelo de probabilidad Naive Bayes. Dicho modelo combina el teorema de Bayes con una regla de decisión. Normalmente esa regla suele ser coger la hipótesis con mayor probabilidad, conocida como la regla "maximum a posteriori" o MAP.

Como en el problema que nos concierne se tienen características que no se conoce el grado de independencia entre ellas, ya que se trata de secuencias de DNA, se decidió implementar este modelo. En concreto con una distribución de Bernoulli porque las variables objetivo (dependientes) son categóricas, solo pueden tomar valor 0 o 1. La distribución de los datos de entrada se conoce como los modelos de evento. La prioridad de una clases puede calcularse asumiendo clases equiprobables o calculando la probabilidad de esa clase a partir de los datos del conjunto de entrenamiento. Para estimar la distribución de las características de entrada, se tiene que asumir alguna distribución o modelo estadístico no paramétrico, esto se refiere a esas distribuciones que no están caracterizadas por ciertos parámetros, no siguen una distribución normal con σ y μ. (John & Langley, 1995)

Bernoulli Naïve Bayes

Las características singulares de este modelo residen en 2 cosas:

- Se tiene que tratar de un problema de clasificación
- Las variables objetivos tienen que ser binarias [0,1]

Si se cumplen estas dos premisas se podrá intentar aplicar el modelo de Bernoulli. Se suele usar en clasificación de textos. (McCallum & Nigam, 1998)

2.2.3. Deep Learning

Considerado como un subcampo dentro del ML, es una nueva forma de entender el aprendizaje basado en la forma de representar los datos para obtener una salida. El término profundo no tiene nada que ver con la dificultad de la arquitectura de las redes, sino con el número de capas sucesivas que tendrán estos modelos. El número de capas nos indica la profundidad del modelo. Estas capas suelen formar parte normalmente del modelo más común que es la red neuronal.

Cuando se habla de DL parece que es algo nuevo y estamos hablando de que las CNN, usadas para reconocimiento de imágenes y procesamiento de texto ya habían surgido en la

década de los 90. Sin embargo, no fue hasta 2012 cuando por fin se hicieron populares. Hubo dos detonantes realmente significativos: hardware y datos.

Cuando se habla de hardware en cuanto a inteligencia artificial uno se refiere a la incorporación de máquinas que tienen GPU (Graphical proccesing units). Empresas como NVIDIA y AMD lanzaron al mercado potentes tarjetas gráficas para mejorar la experiencia del usuario de los videojuegos y los científicos de datos las usaron para el entrenamiento de sus modelos. La velocidad de cálculo mejoró en varios órdenes de magnitud, lo que permitía probar muchos más modelos en menos tiempo, mejorando los resultados.

En cuanto a los datos, qué decir, actualmente en cualquier ámbito de estudio y de negocio la cantidad de datos que se maneja es incalculable. Sin ir más lejos, el campo de la biología genera últimamente millones de datos, sobre todo genómicos, posicionándose como uno de los campos de estudio con una de las mayores bases de datos disponibles. Internet ha sido el mensajero de esos datos, permitiendo compartir las bases de datos con mayor facilidad entre la comunidad. El propio contenido que se genera día a día en Internet (YouTube, Instagram, Wikipedia, etc.) es una fuente de datos de entrada para modelos de reconocimiento de imágenes y NLP. (Chollet, 2018)

Todo esto, junto a la mejora de los algoritmos de los que se partía en ML ha propiciado que sea DL la nueva baza a jugar por los científicos de datos.

2.2.3.1. Tipos de datos de entrada

La importancia que tiene en ML la ingeniería de características y el preprocesado de los datos no es tan importante en DL. De hecho, los modelos de redes neuronales están preparados para extraer de los datos de entrada las características adecuadas para entrenar posteriormente al modelo.

La librería TensorFlow debe su nombre al tipo de almacenamiento de datos que se hace en *tensores* o los conocidos en Python como Numpy arrays. Estos tensores son contenedores de datos (normalmente numéricos) que pueden tener diferentes dimensiones:

- Escalares (0D tensores): son los números propiamente (enteros, decimales, reales, etc.)
- Vectores (1D): tiene una sola dimensión
- Matrices (2D): utilizados también en ML. Tiene dos dimensiones (filas y columnas)
- Tensores 3D o mayor dimensión: tiene tres dimensiones. Por ejemplo, se utilizan en reconocimiento de imágenes (alto, ancho, número de canales (RGB, escala grises).

2.2.3.1. Red neuronal

El término red neuronal viene de la neurobiología, ya que las redes neuronales están compuestas por capas que contienen los elementos llamadas neuronas artificiales similares a los elementos básicos del sistema nervioso del ser humano. **(Figura 18)**

A pesar de que pudo haber una inspiración inicial en la neurona biológica para la creación de estos elementos ficticios, los modelos de deep learning no están inspirados en modelos de aprendizaje del cerebro humano. Al contrario que el cerebro que trabaja de forma paralela con las conexiones de todas las neuronas cuando se produce un estímulo, los ordenadores tienen mucho menor procesamiento paralelo. Eso sí la capacidad de cómputo de cada uno de los cores de una computadora es enorme en comparación con la capacidad de procesamiento una única neurona biológica. Por eso, hay que saber diferenciar entre core y neurona artificial, ya que un core puede modelar muchas neuronas al mismo tiempo. Dado que se busca automatizar la resolución de problemas que realiza el ser humano como reconocimiento de imágenes, procesamiento de audio, etc. se ha hecho necesario intentar buscar un mecanismo similar al de nuestro cerebro.

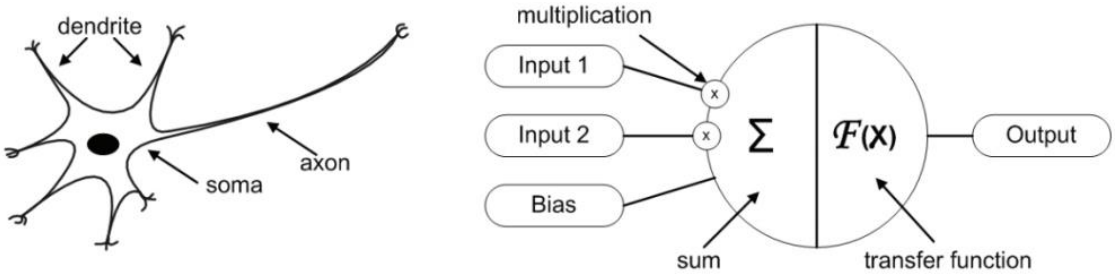


Figura 18. (Krenker et al., 2011). Estructura de una neurona biológica versus una artificial

En la neurona artificial las diferentes entradas se multiplican por un peso diferente, una vez que estén todos, se sumarán y pasarán por una función de transferencia o función de activación que dará las salidas. Esto se puede simplificar en el siguiente modelo matemático:

- $x_i(k)$ es el valor de entrada para un tiempo k para la entrada i que va de 0 a m,

$$y(k) = F\left(\sum_{i=0}^{m} w_i(k) \cdot x_i(k) + b\right)$$

  siendo m el número total de entradas a la neurona.
- $w_i(k)$ es el peso asignado en un tiempo k para una entrada i que va de 0 a m.
- b es el bias
- F es la función de transferencia que se aplica
- $y_i(k)$ es el valor de salida para una entrada i en un tiempo k determinado.

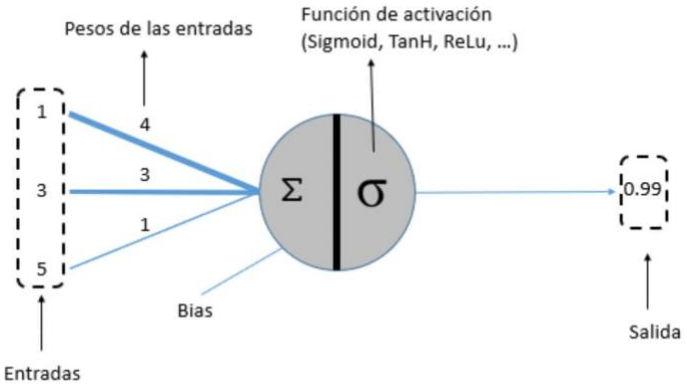


Figura 19. Funcionamiento de una neurona artificial.

Para evitar la linealidad entre la entrada y la salida de las neuronas se utiliza la función de transferencia que puede ser cualquier función matemática. **(Figura 20)** Las más comunes son:

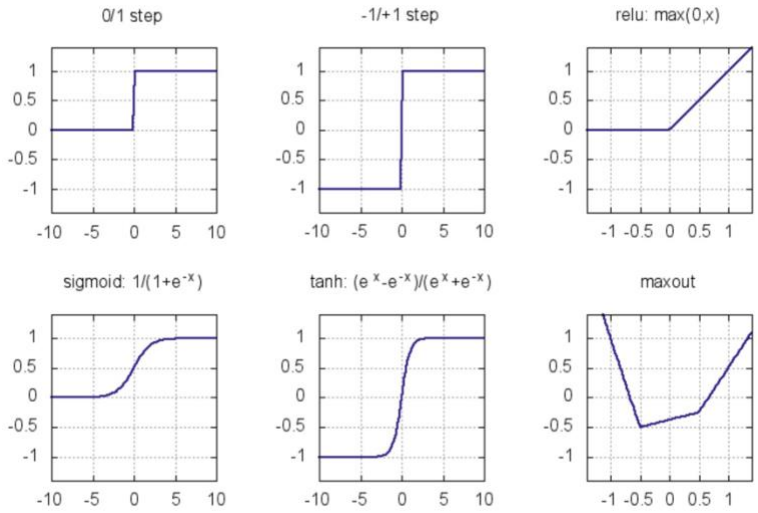


Figura 20. Funciones de transferencia o activación más comunes.

La función escalón (step) solo tiene dos valores de salida posibles (1 y 0), se puede considerar como un umbral para diferenciar dos únicas salidas de entre todas las posibles entradas. Si supera esa barrera se le asignará el 1 y sino el 0. La segunda función escalón (-1/1) funciona igual que la primera, pero con los valores -1 y 1. Otra función también muy utilizada es la relu, que lo que hace es mantener el mismo valor si es mayor que 0 y los negativos los deja con un valor 0. (Suzuki, 2011)

Una vez que sabemos cuáles son los elementos básicos, vamos a subir de nivel. El conjunto de las diferentes neuronas artificiales forma una capa. Su función principal es

almacenar los pesos de salida de cada una de las neuronas de la capa. Los pesos se representan por números agrupados en una lista, también conocidos como los parámetros del modelo. Por tanto, el aprendizaje consiste en aprender los mejores valores de esos pesos en las diferentes capas para poder obtener el mejor resultado posible tras el aprendizaje. La cantidad de parámetros puede ser del orden de cientos o miles de parámetros, que además cambian si se realiza una modificación en uno de ellos a través del algoritmo de backpropagation.

Algoritmo de backpropagation

Con el fin de que las capas puedan cambiar sus pesos se hace a través del algoritmo de backpropagation, que va cambiando los pesos de las neuronas a partir de la función de pérdida en cada interacción de entrenamiento.

El objetivo final es conseguir minimizar al máximo la función de pérdida, es decir, que los resultados obtenidos sean cercanos a los que se esperaba inicialmente. Esto se consigue encontrando los pesos adecuados. Esta búsqueda se lleva a cabo mediante backpropagation **(Figura 21)**, determinando el error o la pérdida en la salida y luego propagando hacia atrás el error. Así, se van actualizando los pesos desde la última capa hasta la primera para reducir el error resultante de cada neurona. La mejora de los pesos se hace a través del optimizador, que será una de las variables que habrá que decididir a la hora de entrenar el modelo. En

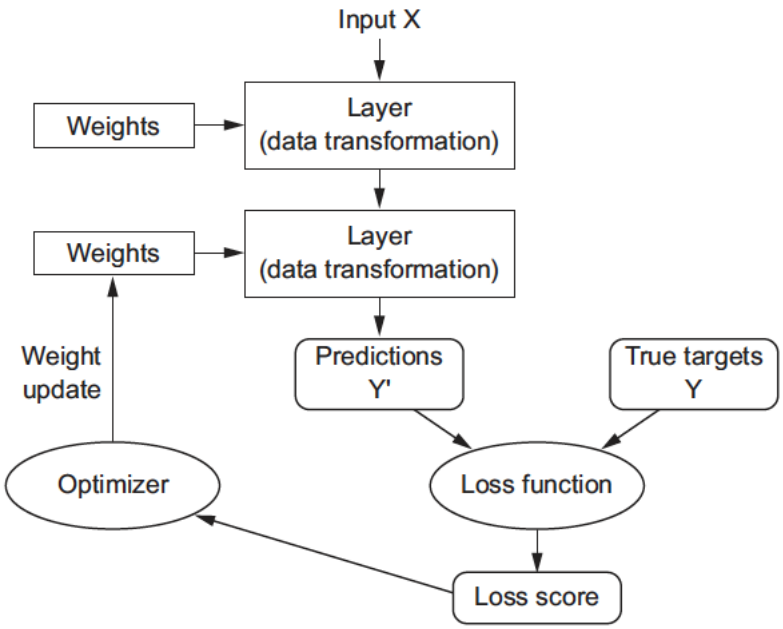


Figura 21. (Chollet, 2011) Representación esquemática de una red neuronal y el algoritmo de backpropagation

concreto, el DL se caracteriza porque puede usar funciones de retropropagación más efcienetes, sin overfitting y sin elevado coste computacional.

- ***'Loss function'***: es un método para evaluar lo bien que lo hace el modelo con respecto a los datos de entrada. Si el número que proporciona esta función es alto, el modelo no funciona bien, en cambio, si el número es bajo, el funcionamiento está siendo correcto. Nos permite ver cuánto difiere el dato predicho por el modelo del valor real. Tipos:
    - Binary Cross-Entropy
    - Categorical Cross-Entropy
    - MSE (Mean Squared Error)
    - Mean Absolute Error

Este resultado se usa como realimentación a la red neuronal a través del optimizador.

- **Optimizador**: este parámetro es una función que tiene como objetivo encontrar los mejores pesos para minimizar la función de pérdida en cada interacción de entrenamiento. La función de coste aporta el dato de cuán de mal se encuentra el modelo y el optimizador es el encargado de remodelarlo para mejorar su desempeño para el problema.

El optimizador más conocido es el descenso de gradiente. Su funcionamiento es el siguiente. Primeramente, calcula pequeños cambios y cómo afectaría al resultado de la función de pérdida estos cambios. Ajusta cada peso basándose en el gradiente y vuelve a realizar repetidamente esos dos pasos hasta encontrar un mínimo adecuado de la función de coste. Los gradientes son derivadas parciales (pequeños cambios en los pesos de las neuronas). Existe un problema con este algoritmo y es que, a veces, crea una falsa sensación de haber llegado al mínimo de la función de coste y resulta que es un mínimo local, no absoluto. Para evitar esto se suele establecer un *'learning rate'*, hiperparámetro que sirve para indicar cómo de grandes van a ser los cambios que va a hacer el optimizador. **(Figura 22)** Un *learning rate* muy grande puede no alcanzar nunca el mínimo de la función y uno muy pequeño puede encontrarse con mínimos locales considerándolos como absolutos. (Velasco, 2020)

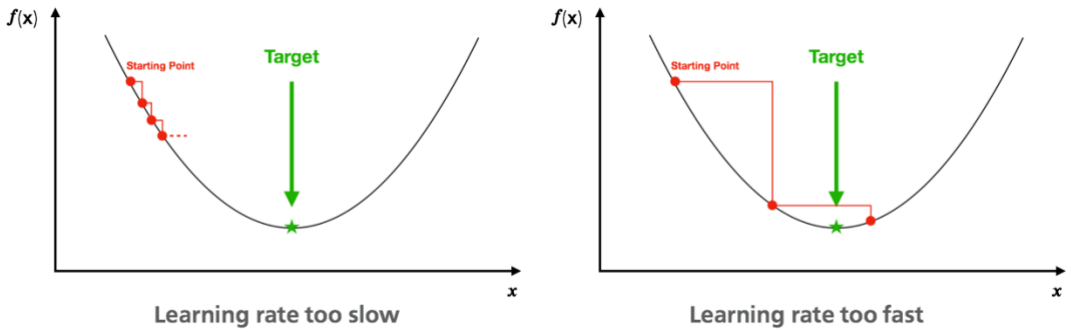


Figura 22. Gráfico de cómo influye el 'learning rate' para encontrar el mínimo absoluto. Extraída de: https://wandb.ai/site/tutorial/build-a-neural-network

2.2.3.2. Deep Feedforward Networks

El perceptrón es la red neuronal más sencilla con una única neurona y con una arquitectura y funcionamiento al presentado en la **Figura 19**, pero a medida que las tareas son más complejas se necesita añadir más capas con más neuronas. Las redes neuronales prealimentadas se conocen también como MLPs (Multilayer perceptrons). **(Figura 23)** Fueron las primeras redes neuronales ideadas y las más sencillas. Al igual que antes el objetivo es aproximar una función f*, realizando un mapa que conecte las entradas x con las salidas, intentando que el modelo aprenda los mejores parámetros para ellos.

$$y = f*(x) \approx f(x,w) \approx \phi(x^T w)$$

- y es la salida resultante
- f es la función que proporciona una salida y para una entrada x con el peso w de la neurona en cuestión
- ϕ es la función de activación no linear para una capa

Si tenemos diferentes capas ocultas:

$$\phi = \phi^{(n)}(\dots\phi^{(2)}(\phi^{(1)}(x))\dots)$$

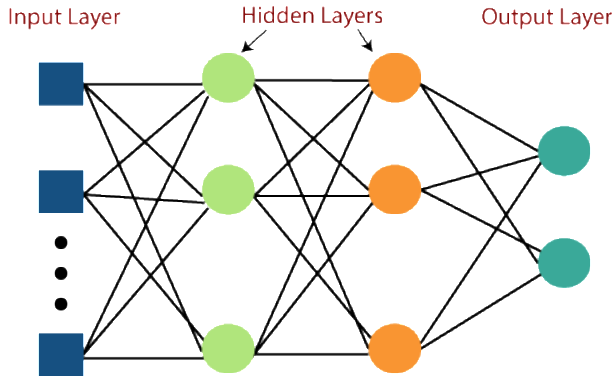


Figura 23. Diagrama Multilayer Perceptron (MLP). Extraído de https://github.com/Thanasis1101/MLP-from-scratch

En este caso, en las DFFN en vez de adivinar la f correcta de un conjunto de ellas, la red consigue aprender la función aproximando la misma a través de la función ϕ, que también irá cambiando a lo largo de las capas ocultas.

Se llaman redes prealimentadas hacia delante porque los datos van de la entrada a la salida sin que haya retroalimentación a lo largo de la red, es decir, el flujo de datos siempre va de la entrada a la salida.

La arquitectura consiste en una red de neuronas artificiales en diferentes capas, existiendo tres capas de neuronas: capa de entrada, capas ocultas y la capa de salida. La capa de entrada tiene tantas neuronas como características o parámetros de entrada haya en el conjunto de

entrada. En las capas ocultas todavía no hay un criterio claro de cuántas neuronas se necesitan en cada problema. Hay estudios de algoritmos genéticos que permiten determinar cuántas neuronas son necesarias para cada caso de estudio. Esto es algo que se escapa un poco del objetivo de este trabajo, por lo que la metodología seguida para determinar el número adecuado de neuronas ha sido prueba-error hasta obtener unos resultados que se podrían considerar optimistas. Y en la capa de salida, si se trata de un clasificador, como es el caso, se precisan de tantas neuronas como categorías haya que distinguir (2, en nuestro caso). (Sewak et al., 2020)

La literatura propone en muchos casos que el número de capas ocultas tiene que ir acorde a la complejidad de los datos. Sin embargo, encontrando las los hiperparámetros adecuadas puede ser posible dar con resultados muy prometedores con una o dos capas ocultas.

Capa densa

En esta capa todos los datos de entrada tienen una conexión con cada una de las neuronas, es decir, se conectan una a una. Luego la salida de cada una de las neuronas pasa a la siguiente capa. Debido a esto, el número de datos de salida suele ser igual al número de neuronas. Se multiplica cada uno de los datos de entradas por el peso de la neurona (parámetro del modelo). En las capas de salida suele haber una capa densa con tantas neuronas como categorías se quiera clasificar.

Además, cada capa densa tiene una función de activación. Esta es diferente en las capas de entrada/ocultas y en las capas de salida. Cada neurona tiene la capacidad de decidir si se encuentra activa o no, esta decisión está condicionada por la función de activación. ¿Cómo se decide si una neurona se activa o no? Para eso está la función de activación que establecerá un margen para tomar una vía u otra para que el resto de las neuronas conozcan su estado. Las funciones más usadas son:

1. **Softmax:** transforma las salidas a una representación de probabilidades, siendo el sumatorio de todas las probabilidades de salida 1. Es muy usada en las capas de salida, por lo que, se suele usar para clasificación.
2. **Sigmoide o tanh:** son de las primeras que se implementaron. Son poco recomendables para entrenar DNN porque las neuronas estarían en las zonas extremos de la función sigmoide. Debido a este problema, las redes neuronales no pudieron resolver algunos problemas hasta que se encontraron nuevas funciones de activación.
3. **Relu:** función bastante utilizada en las capas de entrada y ocultas. Se comporta como una función de identidad dando el valor que tiene si se trata de un valor positivo o anulando si es negativo.

$$f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$$

Capa dropout

Es una capa sin coste computacional, es decir, no va a afectar al rendimiento del modelo. Se suele introducir cuando hay overfitting como método de regularización. Durante el entrenamiento del modelo una proporción de los nodos de una capa de la red se inactivarán. Existen diferentes formas de inactivar los nodo/neuronas de la red, pero la más común es quitarla de la red durante ese entrenamiento. Esto se hace multiplicando su salida por un valor cero. Este proceso solo se realiza durante el entrenamiento y no cuándo se valida el modelo. Por tanto, puede que haya veces en los que el error de test sea menor que el de entrenamiento y haya que tener en cuenta la introducción de una capa de Dropout. (Srivastava et al., 2014)

2.2.3.3. Redes Neuronales Convolucionales

En las redes neuronales artificiales normalmente todas las neuronas estaban conectadas una a una con la siguiente capa, lo que resulta en un gran número de parámetros. En las convolucionales cada neurona está únicamente conectada con unas pocas neuronas contiguas, reduciendo considerablemente el número de parámetros de la red. **(Figura 24)**

Las conexiones entre una capa y otra utilizan también pesos, pero esos pesos irán en un kernel o filtro. El kernel se compartirá entre todas las neuronas conectadas entre sí adyacentes. El resultado de los cálculos que se lleven a cabo con ese kernel dará lugar a una matriz, el mapa de activación. (LeCun et al., 2015). Cada mapa de activación será diferente, según el kernel usado en cada caso. El número de kernels es un hiperparámetro que se le pasará al modelo a la hora del entrenamiento.

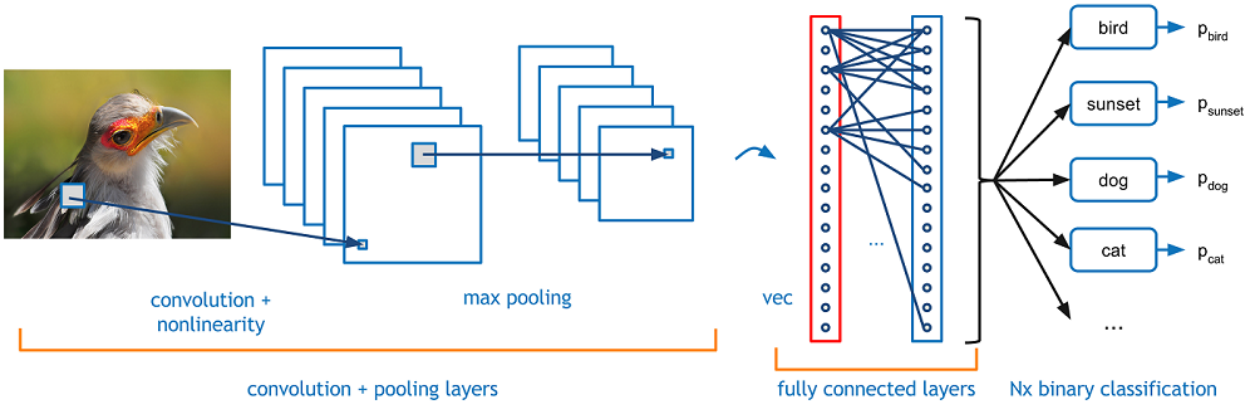


Figura 24. Esquema Convolutional neural network (CNN)

Capa convolucional

Esta capa es la más importante y, por ello da nombre a este tipo de redes neuronales artificiales.

Existen tres hiperparámetros, a diferencia de los parámetros, estos son ajenos al modelo y se deben establecer por el ingeniero de datos que diseña la red neuronal. En este caso los hiperparámetros que se pueden modificar son:

1- Tamaño del kernel. El kernel, filtro o núcleo es un vector o matriz que irá recorriendo los datos de entrada para la extracción de características relevantes. Puede ser de 3x3, 5x5, 7x7, etc. Cuanto menor sea el tamaño se podrá hacer una extracción de características más fina, pero mayor será la dimensión del mapa de activación resultante.

2- Stride. Podemos mover el kernel de uno en uno en los parámetros de la matriz de entrada o con saltos mayores. Es decir, si se elige stride = 1, se realizará la operación de convolución de uno en uno en los elementos de la matriz de entrada. Cuanto mayor sea el tamaño del stride menor será el valor del mapa.

3- Padding. Este parámetro se usa para especificar el número de ceros que se añaden en los bordes de la entrada. Esto permite conservar el tamaño inicial de la entrada, sin perder los bordes.

La **operación de convolución** es la operación central en las capas de convolución. Para una explicación más fácil se va a usar una imagen, que suele estar formada por tres planos: canales RGB (red, green, blue). Cada uno de los canales está formado por matrices de un ancho y alto determinado. Sin embargo, la entrada a una capa de convolución puede tener un solo canal o incluso ser una matriz como entrada. Los datos de entrada de estas capas se suelen llamar **tensores**. (Chollet, 2018) Un tensor se define como una entidad algebraica formado por diferentes componentes, se clasifica según su dimensión:

- Tensor0D: son los escalares, es decir, un valor numérico.
- Tensor1D: tiene una dimensión, son los vectores.
- Tensor2D: son las matrices.
- Tensor3D: tiene tres dimensiones (altura, anchura y profundidad)

Dependiendo de cómo sea la entrada de cada capa convolucional, se usarán arquitecturas con capas Conv1D, capas Conv2D, etc.

El filtro de 2x2 usado en la **Figura 25** se desplaza por toda la matriz de 4x3 generando una matriz 3x2 suponiendo que el *stride* es 1 en este caso, es decir, el *kernel* se desplaza 6 veces. Como se ve lo que se hace es multiplicar cada uno de los valores del *kernel* por la matriz de entrada y el sumatorio será el valor del mapa de activación en esa posición. Este proceso se hace desplazando el *kernel* con un determinado *stride* hacia la derecha hasta completar el ancho y luego se hace un salto empezando de nuevo de izquierda a derecha hasta completar toda la imagen, en este ejemplo.

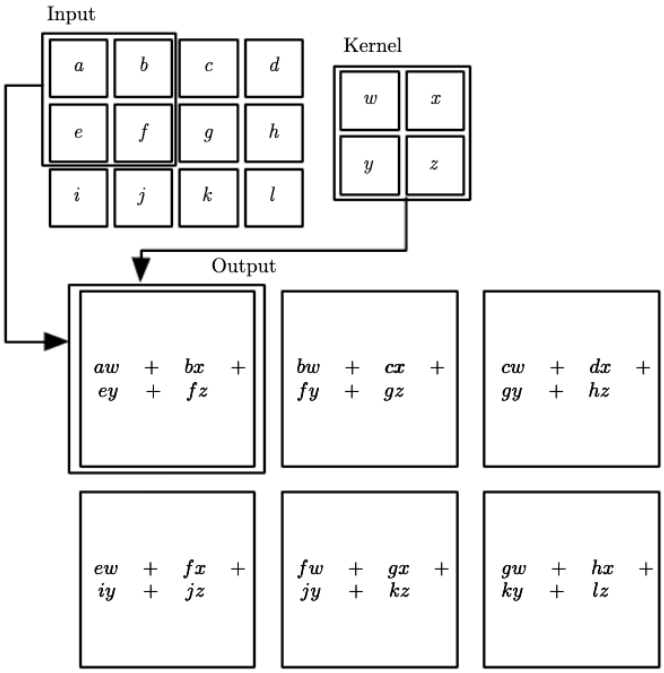


Figura 25. Ejemplo de convolución 2D con un filtro de 2*2 y con un stride=1. (Goodfellow et al., 2017)

En caso de que hubiera varios canales de entrada, como RGB, el *kernel* tiene la misma profundidad que número de canales.

Dependiendo del tipo de *padding* podemos aumentar o reducir la dimensionalidad de los resultados con respecto a la entrada:

- **Same-padding**: mantiene o aumente la dimensionalidad.
- **Valid-padding**: disminuye la dimensionalidad porque no ha aplicado relleno.

Esta capa se usa muchas veces porque reduce considerablemente la capacidad de almacenamiento y de cálculo. En las primeras capas convolucionales se suele hacer la extraccción de características más evidentes y a medida que se unen más capas convolucionales suele ser para extraer patrones más difíciles en alto nivel.

Capa de pooling

El fin último de esta capa es reducir la dimensión de la entrada con cualquier técnica de pooling que exista sin perder demasiada información. Los hiperparámetros que se pueden modificar son el zero-padding. stride y la dimensión de la ventana de pading. Por ejemplo, una ventana de dimensión 2, 2 de stride y 0 zero-padding reduciría a la mitad el tamaño de los datos de entrada a la capa.

La técnica más conocida es la de *max-pooling*, que consiste en coger el máximo valor de las matrices de activación creadas por el *kernel*. Esta capa es invariante a los pequeños cambios en traslación, por lo tanto, es más importante el hecho de que encuentre cierta característica en el conjunto de entrada, que la localización concreta de la misma (Scherer et al., 2010). Esta técnica reduce la dimensión de entrada de la capa, lo que produce un rendimiento en la eficiencia estadística y reduce el tamaño de memoria requerido. También se puede usar la técnica de *average pooling* que devuelve el valor medio de la parte de los datos de entrada cubiertos por el *kernel*. **(Figura 26)**

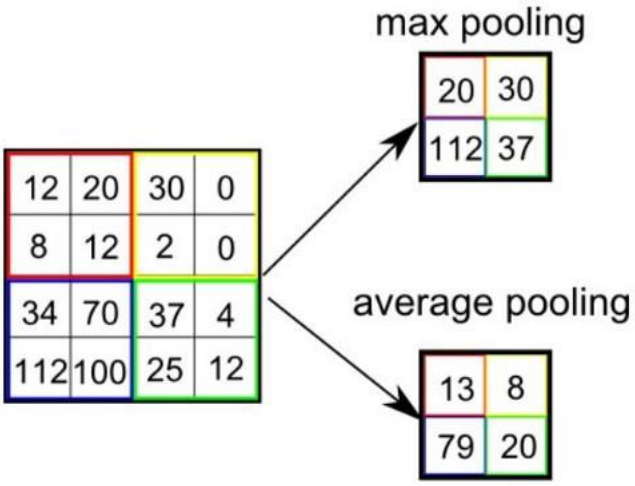


Figura 26. Tipos de pooling

2.2.3.1 Hiperparámetros

Cada una de las capas que se han ido viendo de las diferentes arquitecturas tienen sus hiperparámetros, pero hay dos hiperparámetros que son comunes a todas las arquitecturas de DL a la hora de entrenarlo:

- '**Batch size**': las redes neuronales no se entrenan de forma completa, sino que se dividen los datos en pequeños batches. No existe un número ideal del tamaño del batch pero suele rondar entre 32-512, esto es que hay de 32 - 512 datos cada vez que se entrena el modelo.

- **Epochs**: es un hiperparámetro numérico que determina el número de veces que se entrenará el algoritmo con la totalidad de los datos, es decir, el número de veces que se modificarán los parámetros internos del modelo. Cada epoch tiene diferentes batches de datos. Normalmente la forma de evaluar el desempeño de un modelo de DL se hace con una gráfica que se representa el accuracy frente al número de epochs entrenados.

2.2.4 Métricas

Para poder comparar los diferentes modelos tanto de ML como DL se necesitan métricas que indique qué modelo es más adecuado en cada caso de estudio.

Accuracy

Los modelos de clasificación se suelen comparar con la métrica de exactitud (accuracy). Pero ¿qué es la exactitud? La exactitud es el número de predicciones que el modelo realizó correctamente en función del total de predicciones realizadas. (Developers Google, 2021) En concreto en la clasificación binaria se calcula de la siguiente forma:

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

- VP: verdaderos positivos, es decir, categorías positivas bien predichas
- VN: verdaderos negativos, categorías negativas bien predichas
- FP: falsos positivos. Cuando se clasifica como positivo algo que es negativo
- FN: falsos negativos. Se clasifica como negativo un valor positivo

Matriz de confusión (MC)

La matriz de confusión sirve para ver el desempeño de clasificación de un clasificador con respecto a los datos de test reservados a la hora de trocear el conjunto de datos original. Es una matriz bidimensional, en la que las columnas representan los valores predichos y las filas los valores verdaderos. En la **figura 27** se representa la clasificación en dos categorías (positiva y negativa). Con esta matriz se puede ver cuántos valores se han predicho bien de cada categoría con respecto a los mal predichos para evaluar el desempeño del modelo. (K.M, 2011)

		Predicted	
		Negative	Positive
Actual	Negative	**a**	**b**
	Positive	**c**	**d**

Figura 27. Ejemplo matriz de confusión

Precisión y exactitud

La exactitud es el grado de dispersión de los valores verdaderos. Es decir, una aproximación de cuántos valores verdaderos predice de forma correcta el modelo. En cambio, la precisión también conocida como ratio de verdaderos positivos mide el conjunto de verdaderos positivos que estima bien el modelo en comparación con todos los valores positivos (reales + predichos). Puede que el modelo sea preciso para una de las dos clases, pero no sea exacto, ya que simplemente tiene precisión correcta en los no-nucleosomales.

Sensibilidad y especificidad

La sensibilidad y especifidad se usan para descriminar los casos positivos, de los negativos. La sensibilidad ("recall") o tasa de verdaderos positivos ("tp") es la proporción de casos positivos que fueron bien identificados por el modelo. **(Figura 29)**

La especificidad o tasa de verdaderos negativos ("tn") es la tasa de casos negativos que el algoritmo ha clasificado de forma correcta.

$$\text{fp rate} = \frac{FP}{N} \qquad \text{tp rate} = \frac{TP}{P}$$

$$\text{precision} = \frac{TP}{TP+FP} \qquad \text{recall} = \frac{TP}{P}$$

$$\text{accuracy} = \frac{TP+TN}{P+N} \qquad \text{F-measure} = \frac{2}{1/\text{precision}+1/\text{recall}}$$

Figura 28. Métricas usadas para ver el desempeño de modelos de IA

Curva ROC

A modo de reafirmar los resultados se ha decidido usar otra técnica para visualizar, analizar y comparar algoritmos de machine learning que se ha venido usando últimamente en ámbitos como diagnóstico médico y cada vez más en el ámbito científico en general. Se trata de la curva ROC.

La gráfica representa en el eje de la x los FP y en el eje de la y los TP. Mide el porcentaje de verdaderos positivos frente a falsos positivos, es decir, relación entre beneficios y costes del modelo. El punto [0,1] es la mejor clasificación, el ideal. Los puntos que tenga una x más baja (bajo ratio fp) y mayor y (alto ratio tp) serán mejores que los contrarios: alta x y bajo y. Para poder ver el rendimiento del modelo de un primer vistazo al mirar la gráfica se suele añadir una recta que sea y=x, llamada rendimiento aleatorio. Por ejemplo, si el modelo aleatoriamente acierta la mitad de TP, también tendrá una tasa de 0.5 de FP [o.5, 0.5]. Puede ser que también clasifique muy bien 0.9 tp rate, pero tendrá un fp rate de 0.9. Por tanto, todo clasificador que aparezca en el triángulo inferior derecho debajo de la curva aleatoria será un mal clasificador. Todo lo que quede en el triángulo superior izquierdo es indicador de un clasificador adecuado. **(Figura 29)**

La curva tiene dos dimensiones, por lo que para realizar comparaciones se debería obtener un valor numérico. En este caso se suele calcular el área bajo la curva (AUC). Es un valor entre 0 y 1. En concreto, la curva aleatoria tiene un AUC = 0.5. Cualquier valor por debajo de eso no representará un clasificador realista. (Fawcett, 2004)

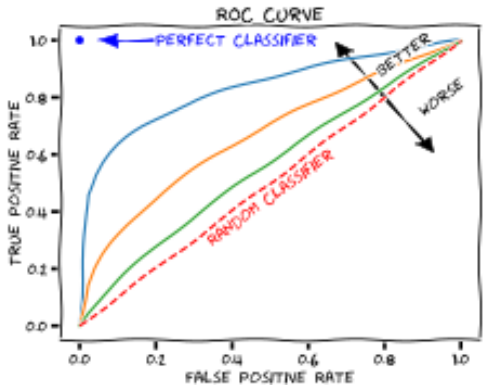


Figura 29. ROC Curve

2.2.5 Bioinformática

La bioinformática es un campo interdisciplinar que se ocupa de la aplicación de la informática a la recopilación, almacenamiento, organización, análisis, presentación y distribución de datos biológicos o médicos (por ejemplo, secuencias de DNA).

Tiene tres propósitos claros (Luscombe et al., 2001):

1- Organizar los datos en bases de datos biológicas que sean accesibles a los investigadores, además de tener la posibilidad estos de aumentar la base de datos introduciendo nuevos registros. La información de estas bases de datos no tiene una utilidad hasta que no se analiza. Por lo tanto, la comprobación y análisis de estos datos es otra de las tareas de la bioinformática.

2- Desarrollar herramientas y recursos que sirvan para poder realizar el análisis de los datos. Por ejemplo, la comparación de secuencias por su similitud, con programas como BLAST (Basic Local Alignment Search Tool) que encuentra las regiones de similitud entre las secuencias para futuras predicciones.

3- Analizar los datos e interpretar los resultados para que haya una correlación entre los resultados estadísticos obtenidos por la computadora y la interpretación biológica.

Como hemos visto anteriormente en el apartado 3.2 y 3.3 ML y DL son técnicas basadas en el aprendizaje de los datos proporcionados tanto para realizar técnicas como clustering, clasificación o regresión. La combinación de estas dos subcampos de la IA junto a las herramientas de la bioinformática ayuda a predecir ciertos resultados sin necesidad de diseñar tantos experimentos en el laboratorio con su posterior ejecución.

2.3. Estado actual

La función de compactación que realiza la cromatina es importante, pero además actúa como un "semáforo" de acceso a la secuencia de DNA por parte de otras macromoléculas y regula procesos como transcripción, replicación, recombinación y reparación. En consecuencia, el control de la cromatina tiene que ver con las modificaciones histónicas, enzimas remodeladoras de cromatina y la presencia de proteínas estructurales. Actualmente, se sabe que el posicionamiento de los nucleosomas está relacionado con estos procesos **(ver fig. 30).**

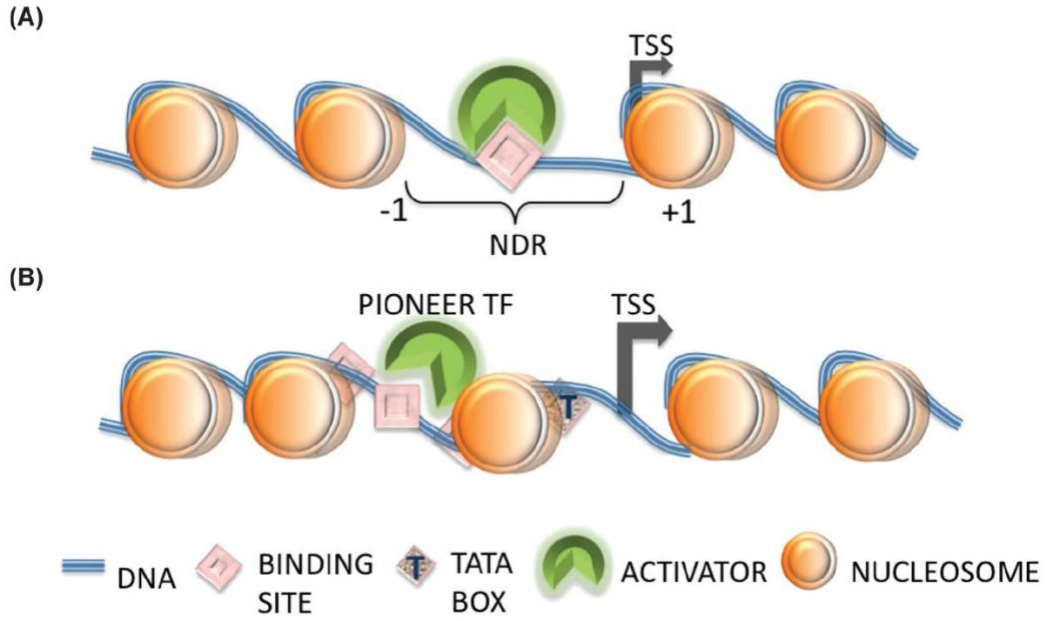


Figura 30. (Arya et al., 2012) Esquema de los promotores abiertos (A) con el nucleosoma "+1" y "-1", el NDR antes del TSS. En (B) se muestra el promotor cerrado o ocupado, con la presencia de la TATA box antes del TSS.

Se sabe que existen patrones de presencia/ausencia de nucleosomas que facilitan o dificultan la actuación de diferentes agentes sobre el DNA. (Arya et al., 2012)

Los experimentos acerca de la posición de los nucleosomas han sufrido un incremento en los últimos años por el avance de técnicas que permiten pintar mapas de nucleosomas, que sirven para identificar donde se encuentran los nucleosomas a lo largo del genoma. Con estos mapas se puede ver la tendencia de los nucleosomas por ciertas regiones específicas del genoma; además de la presencia de proteínas o remodeladores de cromatina como hemos visto en el apartado 2.3.

La ocupación de nucleosomas varía de unas especies a otros en las regiones cercanas al inicio de la transcripción. Se ha visto que puede deberse al número de genes activos. Los genes que no se usan, se encuentran inactivos, solo activando aquellos que son necesarios en

cada proceso en organismos pluricelulares. En cambio, en los unicelulares al tener una menor complejidad, menor número de células diferentes, la mayoría de los genes están en uso de forma continua, por lo que es necesario que sean más fácilmente accesibles con zonas desnudas de nucleosomas, NDRs. (Tompitak, Vaillant, & Schiessel, 2017)

También se han comparado secuencias de nucleosomas de organismos *in vivo* con otras nucleosomas y secuencias de DNA purificados en el laboratorio previamente, *in vitro*. **(Figura 31)** La organización de los nucleosomas en el genoma en vivo y en vitro es bastante similar. Lo único que se remarca es una mayor ocupación de nucleosomas in vivo. La correlación entre ambos modelos tampoco es uniforme a lo largo del genoma, aumentando en las regiones intergénicas al final de un gen transcrito o disminuyendo en los promotores o regiones codificantes. Estos resultados remarcan que *in vivo* hay más factores que provocan el posicionamiento del nucleosoma que la propia cadena de DNA, como son factores de transcripción, los remodeladores de cromatina o un proceso de transcripción activa.

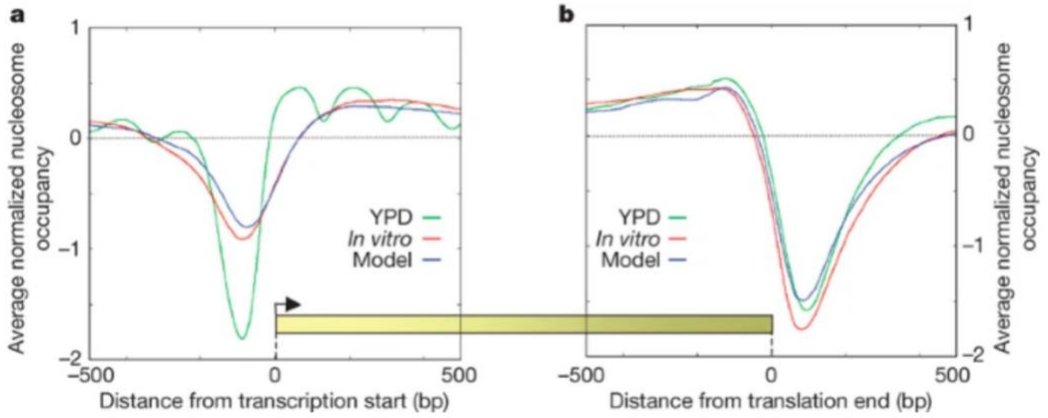


Figura 31. (Kaplan et al., 2008) Representa la ocupación nucleosomal frente a la distancia del inicio de la transcripción (a) y al final de la traducción (b). La tendencia más clara de las secuencias in vivo puede ser explicada por la mayor concentración de nucleosomas que in vitro, lo que provoca un mapa más claro de posicionamiento. El agotamiento al final de los genes puede deberse a señales de terminación.

Para analizar el papel que ejerce la secuencia de DNA al posicionamiento de nucleosomas podemos mirar únicamente las secuencias *in vitro* que solo están determinadas por el DNA, sin la influencia de otros factores o procesos celulares. Gracias a un modelo estadístico se predijeron las regiones ricas en nucleosomas, así como las que están desnudas. Por tanto, la secuencia en crudo de DNA se vió que puede favorecer los sitios de unión de factores de transcripción y procesos celulares como la transcripción. (Kaplan et al., 2008)

El elevado coste de representar los mapas de nucleosomas, ha llevado a la comunidad científica a buscar soluciones computacionales que ayuden a encontrar las secuencias de nucleosomas. Como se ha ido evidenciando la secuencia de DNA es uno de los elementos clave para el posicionamiento.

Las primeras soluciones vinieron de la mano de técnicas estadísticas. Pero, con el fin de facilitar la experiencia del usuario desarrollaron una aplicación web nuMap con una interfaz gráfica donde se puede introducir la secuencia a predecir bien por comando o a través de algún formato de fichero. Puede haber cuatro diseños de salida diferentes: "Single Layout" muestra el mapa de nucleosoma simple, "Superimpose Layout" que muestra más de un gráfico si varios secuencias se han introducido. "Average Layout" calcula las puntuaciones medias de las secuencias dadas. "Average Layout (Symmetric)" produce el gráfico simétrico complementario de todas las secuencias de entrada y las puntuaciones medias de cada posición de pares de bases. (Alharbi et al., 2014)

Además de los métodos estadísticos también se han añado soluciones que contemplen modelos de inteligencia artificial. Como se ha visto en la sección 3 existen diferentes tipo de redes neuronales dependiendo normalmente la tarea que resuelvan. Las CNN se usan normalmente para interpretar imágenes, visión por ordenador; pero si se usa algún tipo concreto de red como Conv1D se pueden emplear para la clasificación de series de tiempo o señales de audio. Las RNN suelen ser comunes en generación de texto o procesamiento de lenguaje natural (NLP).

Dentro del campo de CNN se ha creado un modelo de Deep Learning para aprender el posicionamiento de secuencias de DNA. LeNup es un modelo con una arquitectura de capas convolucionales, pasando todas por una capa de rectificación linear (ReLU) para la activación, seguida por una MaxPooling, una capa de DropOut y finalmente una capa lineal aplicando función de pérdida sigmoide como clasificador. (Zhang et al., 2018)

La combinación de RNN y CNN es lo propuesto en CORENup, un modelo de identificación de nucleosomas. Como entrada recibe las secuencias de DNA que pasará por un método de preprocesamiento de one-hot encoding con dos niveles paralelos, CNN y capa recurrente para coger las características de las secuencias de DNA periódico y no periódico. Este modelo tiene 4 capas en comparación de las 5 capas convolucionales más las dos totalmente conectadas de LeNup. Las capas convolucionales pueden extraer las características de las secuencias en crudo como patrones. La capa recurrente puede completar esa extracción de características repetidas en los patrones. (Amato et al., 2020)

3 Método, materiales y herramientas

3.1. Método

El proceso unificado (UP) y Scrum son metodologías utilizadas para el desarrollo software. En la última década con la mejora y abaratamiento de los mecanismos para almacenar información, ha habido un auge en la cantidad de información disponible. Lo que ha dado lugar a una nueva disciplina conocida como "data mining" o minería de datos.

Piatetski-Shapiro en 1991 definió Data Mining como el conjunto de técnicas y herramientas aplicadas al proceso no trivial de extraer y presentar conocimiento implícito, previamente desconocido, potencialmente útil y humanamente comprensible, a partir de grandes conjuntos de datos, con objeto de predecir de forma automatizada tendencias y comportamientos y/o descubrir de forma automatizada modelos previamente desconocidos.

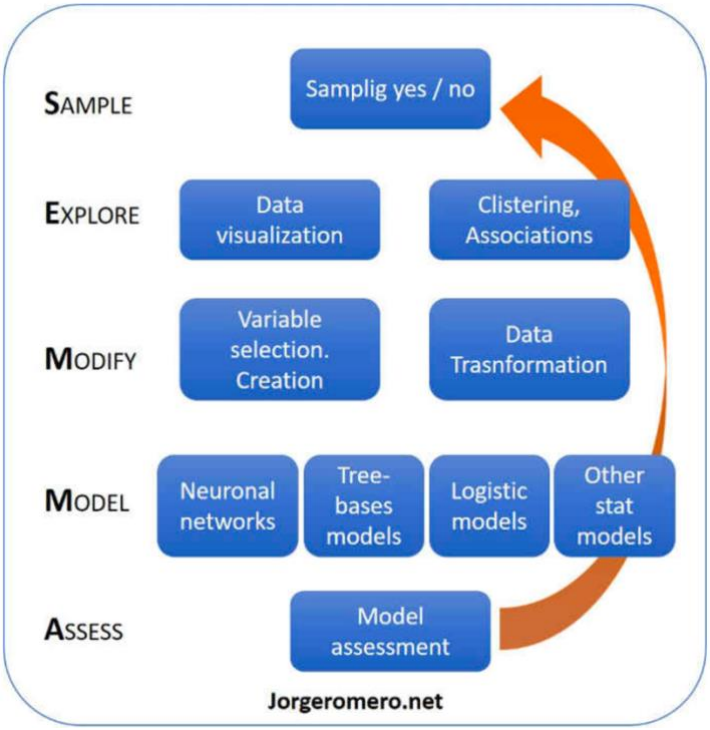


Figura 32. Metodología SEMMA. Extraido de https://jorgeromero.net/metodologias-de-mineria-de-datos/

Este tipo de proyectos no puede seguir un desarrollo incremental porque la mayor carga de trabajo (75%) recae en las primeras fases (preprocesamiento de los datos). La necesidad de un método de trabajo diferente dio lugar a diferentes metodologías como son SEMMA, KDD y CRISP-DM. (Rodríguez Montequín et al., 2016)

El método elegido ha sido SEMMA, cuyas iniciales corresponden a las diferentes fases: muestreo (sample), exploración (explore), manipulación (modify), modelado (model) y valoración (asses). **(Figura 32)**

Se inicia cogiendo una muestra (es opcional) del conjunto total de los datos. La elección de la población muestral es determinante para la exploración posterior. Normalmente se suele hacer de forma aleatoria, porque suponemos que todos los individuos tienen la misma posibilidad de ser elegidos grupo representativo, esto se denomina muestreo aleatorio simple. Una vez hecho el muestreo se hace una exploración de las variables o bien por métodos gráficos o métodos estadísticos para quedarnos con las variables explicativas que serán el input del modelo. Una vez tenemos seleccionados los datos de entrada, se modificarán para que tengan el formato adecuado para el modelo (por ejemplo, one-hot encoding). La cuarta fase es la más importante porque se define el modelo que se va a elegir para el problema planteado, pueden ser machine learning, deep learning o métodos estadísticos. Finalmente, en la última fase se hace una validación de los datos de los grupos entrenados frente a los datos de testeo.

El actual trabajo está basado en esta metodología, pero con ciertas modificaciones adaptadas al proyecto de investigación científica que se está haciendo porque normalmente suelen estar enfocadas al ámbito empresarial.

3.2. Materiales

3.2.1. Ficheros FASTA

Se dispone de tres archivos con las secuencias de DNA almacenados con la extensión '.fasta'. Estos archivos '.fasta' están disponibles en un drive a través del siguiente enlace: https://drive.google.com/drive/folders/1DXt8y0fxGyEeee6a42MrcM8nmTfBc7uJ?usp=sharing.

El formato FASTA es un tipo de formato de fichero computarizado que se utiliza para representar secuencias tanto de ácidos nucleicos como de péptidos, popularmente utilizado en ámbitos relacionados con la biología computacional. La forma para representar los pares de bases o aminoácidos es a través de una sola letra. La primera línea se diferencia del resto porque va precedida del símbolo '>' que incluirá el comienzo de la descripción de la secuencia presentada en las siguientes líneas. A parte de permitir la separación a la hora de poder leer el fichero, también facilita la labor de extraer los datos necesarios a través de un procesador de texto o lenguajes de programación como Python. (Mount, 2004)

Para la correcta escritura del fichero las secuencias se tienen que regir según lo especificado en el estándar IUB/IUPAC tanto para códigos de aminoácidos como de ácidos nucleicos. **(Figura 33)**

A	adenosine	C	cytidine	G	guanine
T	thymidine	N	A/G/C/T (any)	U	uridine
K	*G/T (keto)*	*S*	*G/C (strong)*	*Y*	*T/C (pyrimidine)*
M	*A/C (amino)*	*W*	*A/T (weak)*	*R*	*G/A (purine)*
B	*G/T/C*	*D*	*G/A/T*	*H*	*A/C/T*
V	*G/C/A*	*-*	*gap of indeterminate length*		

Figura 33. Tabla del estándar IUB/IUPAC [captura de pantalla], por National Center for Biotechnology Information. Extraído de https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp

Los tres ficheros han sido proporcionados por el laboratorio del profesor de investigación del CSIC, Francisco Antequera, del IBFG (Instituto de Biología Funcional y Genómica); cotutor del presente trabajo de fin de grado.

El primer fichero "Spombe.FASTA" contiene en su cabecera, tras '>', el nombre del cromosoma y, a continuación, toda la secuencia del cromosoma correspondiente de S. Pombe. **(Figura 34)**

```
>chromosome3
GATCAGCCAAAATGGCTGATCCAGCTATTTAGCAGGTTAAGGTCTCGTTCGTTATCGCAA
TTAAGCAGACAAATCACTCCACCAACTAAGAACGGCCATGCACCACCACCCATAGAATCA
AGAAAGAGCTCTCAATCTGTCAATCCTTACTATGTCTGGACCTGGTGAGTTTCCCCGTGT
TGAGTCAAATTAAGCCGCAGGCTCCACTCCATTGTGGTGCCCTTCCGTCAATTCCTTTAA
GTTTCAGCCTTGCGACCATACTCCCCCCGGAACCCAAAGACTTTGATTTCTCGTAAGGTG
```

Figura 34. Cabecera del fichero "Spombe.FASTA"

El segundo fichero "Nuc_SP3.FASTA" contiene las secuencias de nucleosomas. En la cabecera de la primera línea de cada entrada tiene la información del cromosoma, así como la posición de inicio del nucleosoma dentro del cromosoma indicado**. (Figura 35)**

```
>chromosome1:3786417
CGAACCTCATGAAATCGTTTACCGCTTCTCCTTAATCCATTTGTGTAATACTGAATGCTAAGTAAGAGTGGAAAGCTTCCACCATCCACCAGACCATTACAAGCA
CTACATACGCCATCTTCAATATCGTATATTTTCAGTAGTCAACTT
```

Figura 35. Cabecera del fichero "Nuc_SP3.FASTA"

```
>SPAC212.01c
AAATTGAGAAAAACTGAAAAATAAAGAACGAGGTTCAGCAATAAGCTACAGCTACTTTAGTGATTAGGTAACAATTTAATTCATATATAAAAATATCAATAGCAGA
TGCTACTTGTATTACTGTGTACTGTAATATACTAGGTCACAAGGAAAACACATCGTACTTAAACGTATCGTTAAACAGATACCAAACTGCGTAGCTGTTAAATTGT
GAAAAGAAAAAAGTTGTAAGGATTACGAGATGCCA
```

Figura 36. Cabecera del fichero " NDRs_total_Spombe.FASTA "

Y el tercer y último fichero proporcionado "NDRs_total_Spombe.FASTA" contiene los NDR de *S.Pombe*. Esta vez en la cabecera se proporciona el nombre del gen asociado a esa secuencia de NDR**. (Figuro 36)** El tamaño de las secuencias es diferente en cada uno de ellos. En el archivo "Nuc_SP3.FASTA" las secuencias tienen 150 pares de bases, tamaño que tiene normalmente un nucleosoma.

3.3. Herramientas

En este apartado se presentará las principales herramientas utilizadas para llevar a cabo el proyecto. A lo largo del desarrollo se han ido utilizando otras herramientas, por ejemplo, servicios web como Drive o GitHub para llevar a cabo las diferentes copias de seguridad tanto de los ficheros de código como documentación escrita.

3.3.1. Python

El 20 de Febrero de 1991, Guido van Rossum publicó el código de Python por primera vez en "alt.sources". Él es conocido como la primera persona que se convirtió en Dictador Benevolente, eso significa que es la persona encargada de la toma de decisiones final y la aplica con un uso racional para el beneficio de la comunidad. Se creo este título dentro del software libre para centralizar la toma de decisiones teniendo en cuenta la opinión de la comunidad de programadores. Sin embargo, van Rossum dejó en 2018 este cargo que se le había otorgado.

Es un lenguaje de programación interpretado, dinámico y multiplataforma. El programador puede utilizar un estilo orientado a objetos, programación imperativa o programación funcional, todos soportados para Python.

Considerado como uno de los lenguajes de programación más fáciles de aprender si estás empezando a programar y al mismo tiempo uno de los más potentes, es ampliamente utilizado en el mundo científico, gracias a su uso en procesamiento de imágenes o análisis numérico. Ofrece solución a dos problemas científicos recurrentes como es el cálculo numérico y simbólico.

Tiene licencia "Python Software Foundation License" que permite su libre distribución sin ningún tipo de coste adicional. Esto lo hace ideal para el ámbito educativo y científico. Además, gracias a su infinidad de librerías es bastante más compacto que otros lenguajes, ahorro de memoria y facilidad de lectura de código. (M. Ayer et al., 2014)

3.3.2. NumPy

Librería con un tipo principal de dato que representa matrices multidimensionales, equivalentes a las matrices en R. Se utiliza para cálculo numérico con funciones matemáticas, generador de números aleatorios, transformadas de Fourier o funciones algebraicas. Está muy optimizada, ya que se ha creado a partir de código compilado en C.

3.3.3. Pandas

Pandas es una librería para análisis de datos que cuenta con librerías para la limpieza de datos para conseguir datos aptos para su posterior análisis. Pandas se utiliza para importar y exportar ficheros de textos (.csv. .txt...), función que permanece de su función original que era el análisis de hojas de cálculo.

Las estructuras de datos más importantes de esta librería son el DataFrame y las Series. El DataFrame es una colección ordenadas de columnas con nombre y tipos de datos concretos, donde cada fila representa un caso y las columnas los atributos, es decir tiene 2D. En cambio, las Series es un objeto unidimensional (1D).

3.3.4. Scikit-learn

Para los modelos de machine learning se ha usado esta librería de código abierto. Soporta aprendizaje supervisado y no supervisado. Tiene algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad. Es compatible con otras librerías como NumPy o matplotlib, para la representación a través de gráficos.

3.3.5. TensorFlow

TensorFlow es una librería de Python, propiedad de Google con licencia de código abierto, para hacer cálculos numéricos. Como su nombre indica utiliza gráficas de flujo de datos. Los nodos de las gráficas son las operaciones matemáticas y los bordes de estas son las matrices de datos (tensores) comunicados entre ellos.

Se utiliza para la construcción y entrenamiento de redes neurales destinadas a aplicaciones como procesamiento de imágenes, diagnóstico médico y mejora de la fotografía en smartphones.

3.3.6. Keras

Keras se presenta como una API creada para minimizar la carga cognitiva de los usuarios: ofrece APIs consistentes y simples, minimiza el número de acciones del usuario para casos de uso usuales y ofrece un sistema de mensajes de errores claro. Es una librería de alto nivel, capaz de correr por encima del framework de TensorFlow 2.0. Se puede exportar a modelos de JavaScript, iOS, Android, etc. Ofrece una interfaz sencilla, amigable e intuitiva para la implementación de modelos de redes neuronales.

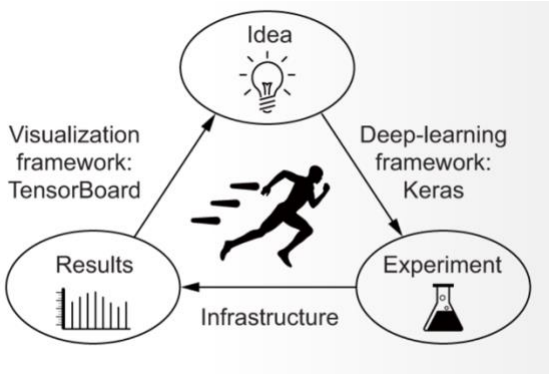


Figura 37. Relación entre Keras API y TensorFlow. Extraído de https://keras.io/

3.3.7. Servidor Artemisa

Gracias al CSIC (Centro de Investigaciones Científicas), y en concreto al grupo de investigación del Profesor Francisco Antequera del IBFG (Instituto de Biología Funcional y Genómica) para este proyecto de fin de grado he podido tener acceso al servidor ARTEMISA. Las siglas significan "Artificial Environment for ML and Innovation in Scientific Advanced Computing". Esta máquina CentOS 7.7.1908 me permitió ejecutar los diferentes modelos de machine learning y deep learning aprovechando los 64 núcleos del servidor, además de paralelizar ciertas tareas. Se puede encontrar el proyecto en la siguiente dirección: https://artemisa.ific.uv.es/web/

4 Desarrollo del proyecto

En el presente trabajo, el objetivo es crear un modelo que sea capaz de clasificar secuencias de DNA indicando si son nucleosomas. Para ello se ha intentado seguir un método de Ciencia de Datos que se llama SEMMA presentado en el apartado 3.1. a la hora de elegir el modelo adecuado y presentar los resultados.

El deep learning es un tipo específico de machine learning, por lo que vi adecuado empezar aplicando modelos de machine learning y posteriormente dar el salto. Los modelos de machine learning suelen ser más fáciles de comprender porque se basan en modelos estadísticos más concisos.

Para que este código sea accesible y se pueda usar en investigaciones posteriores se ha creado un repositorio de GitHub donde está colgados los archivos de los modelos explicados en las siguientes páginas: https://github.com/albavu/TFG_USAL.git

4.1. Problema planteado

Como hemos visto en el apartado 2.3 ya existen algunas soluciones o aplicaciones para intentar dar una solución al problema de posicionamiento de nucleosomas. Algunas de ellas incluso utilizan modelos de aprendizaje automático para dar con una respuesta.

En este caso queremos comprobar la contribución de la secuencia de DNA a este posicionamiento. Lo ideal sería conseguir un clasificador[5] de nucleosomas sin necesidad de comparar secuencias, es decir, que se introduzca una cadena de 150 pb y obtengamos una respuesta afirmativa o negativa.

4.2. Creación ficheros de trabajo

Los ficheros FASTA proporcionados no suelen ser formatos compatibles con las librerías de Python[6] utilizadas en análisis de datos. Por eso a partir de los archivos FASTA se han creado archivos con extensión CSV ('Comma Separated Values'), que tal como su nombre indica son valores separados por comas u otro separador en columnas. Además, este tipo de archivos también se podrían leer con programas que utilizan hojas de cálculo.

Los ficheros csv que se querían conseguir tenían dos columnas: una con las secuencias de DNA de 150 pb. No todos los ficheros presentados anteriormente (apartado 3.2.1) tenían ya en cada entrada secuencias de esta longitud. Se va a explicar a continuación cómo cada uno de los ficheros FASTA han pasado a un formato csv.

[5] Un clasificador, en minería de datos, es un elemento que proporciona una clase etiquetada como salida a partir de un conjunto de datos de entrada.

[6] Keras, TensorFlow, Scikit-Learn,etc.

El fichero "Nuc_SP3.fasta" contenía las secuencias de nucleosomas con una longitud de 150 pb, por lo que se metieron cada una de las secuencias en las filas de un archivo csv. Este archivo tenía 49998 filas y 3 columnas, cada una con los datos del FASTA. **(Figura 38)**

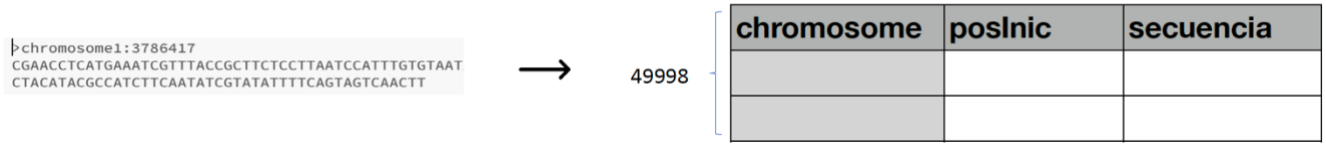


Figura 38. Esquema paso del fichero "Nuc_SP3.fasta" a un archivo con formato csv de las secuencias de nucleosomas.

El fichero "NDRs_total_Spombe.fasta" contiene las secuencias de NDR correspondiente a los diferentes genes. Estas secuencias tienen diferentes longitudes. Se quiere un csv en la que la columna de las secuencias tenga la misma longitud en todas.

Se había obtenido 1468 secuencias de NDRs con una longitud de 150 pares de bases. Se decidió realizar otro método para poder tener más secuencias de NDRs. La importancia de tener más de estas secuencias reside en el hecho de que son secuencias desnudas de nucleosomas completamente, por lo que podría ayudar en el posterior aprendizaje de los modelos. **(Figura 39)** El proceso seguido fue el siguiente:

1º- Unir todas las secuencias crudas de DNA del archivo para tenerlo en un solo array.
2º- Aleatorizar un número que vaya de 0- (longitud cadena -150). Esto es porque el número va a ser el comienzo de una cadena de 150 pb.
3º- La posición de inicio la da el número obtenido y las 150 pb siguientes.

El problema de esta solución reside en que puede que algunas secuencias no coincidan con NDRs puros y sean la mezcla de dos. Sin embargo, a priori puede parecer que estamos sesgando los datos, pero ante la imposibilidad de tener un conjunto de NDRs mayor surge la necesidad de crear otros conjuntos. Finalmente se consiguió 1468 filas de NDRs que ya tenían desde un principio 150 pb y luego 10000 secuencias a partir del método que se ha explicado anteriormente.

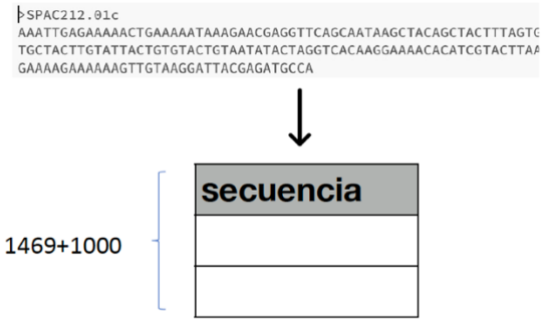


Figura 39. Esquema paso del fichero " NDRs_total_Spombe.fasta " a un archivo con formato csv

Con el fichero que contiene todo el genoma de *S.Pombe* se han creado secuencias de longitud 150 pb de cada cromosoma, habiendo tres cromosomas. Esto se ha conseguido aleatorizando un número de 0-(longitud DNA cromosoma -150) y cogiendo los 150 pb siguientes. En total 21000 secuencias, 7000 de cada uno de los 3 cromosomas que hay en *S.Pombe.* **(Figura 40)**

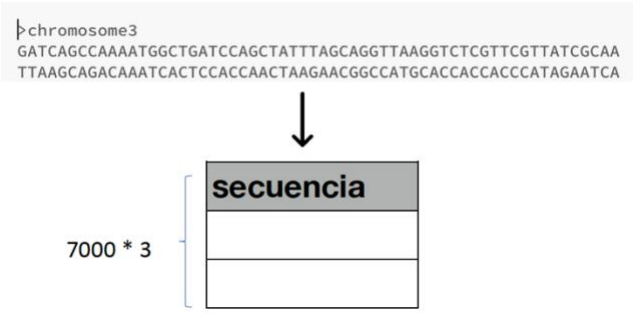


Figura 40. Esquema paso del fichero con todo el genoma de *S.Pombe* a un archivo con formato csv

Finalmente se han obtenido 49998 secuencias del FASTA que contenía las secuencias de nucleosomas; así como 11468 secuencias de NDRs y 21000 secuencias del genoma de *S.Pombe.* Con estos tres csv ya se puede empezar a trabajar en el preprocesamiento para conseguir conjuntos de datos de entrenamiento, validación y testeo que se pasarán a los modelos de IA.

Todo el código con el que se han conseguido estos datasets está recogido en el archivo **DataSet_Edition.py** del repositorio de GitHub y los propios csv están disponibles para poderse descargar en el siguiente enlace:
https://drive.google.com/drive/folders/1cnjcm4FcpLFw1V8zl9HYqd158jTnwrY5?usp=sharing

4.3 Preprocesamiento de datos

Las secuencias de DNA no son aleatorias y desordenadas, simples secuencias de caracteres formadas por cuatro bases desoxirribonucleicas (A, C, G, T). La interpretación de estas secuencias podría asimilarse a lectura de una frase, en la que el orden de las palabras y las letras es crucial para comprender el significado.

Los algoritmos de ML no pueden procesar las secuencias de DNA, antes es necesario realizar una transformación a valor numérico para formar la matriz de entrada. (Yang et al., 2020). El tratamiento previo de los datos es importante para poder entrenar de forma correcta los modelos de ML después.

Antes de empezar con el preprocesamiento de los datos se juntaron todos los csv que se habían sacado de los FASTA en un solo dataset con dos columnas ['secuencia', 'nucleosoma']. En la columna 'secuencia' se tendrían en cada fila las diferentes secuencias y en la columna

'nucleosoma' hay dos valores: 0 si la secuencia corresponde a un no-nucleosoma y 1 si corresponde a un nucleosoma.

En el presente estudio no hizo falta realizar técnicas de limpieza de datos, eliminación de valores nulos (no existían) o reescalado, comunes en aprendizaje supervisado (Kotsiantis et al., 2006). Las técnicas que se han usado en el presente estudio se explican a continuación.

Codificación ordinal

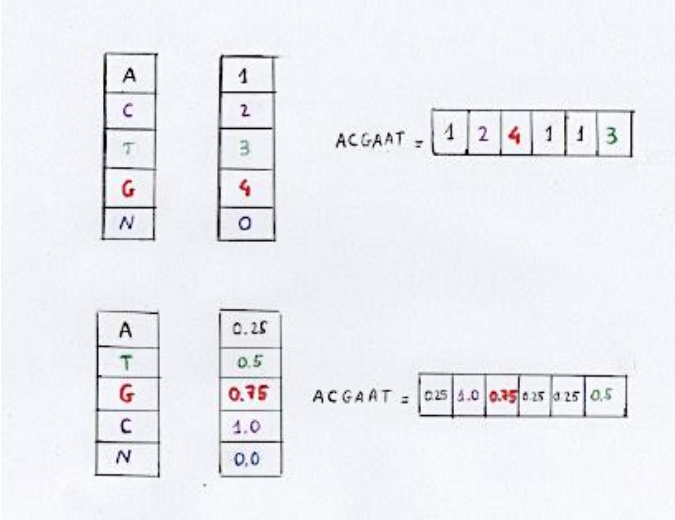
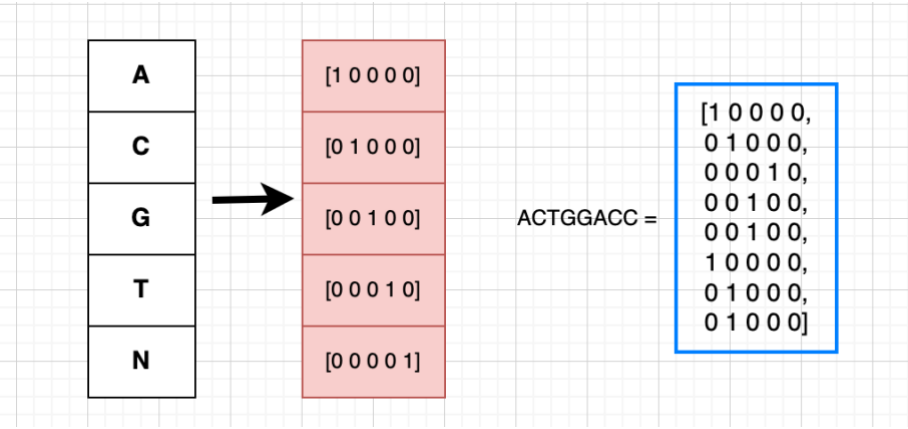


Figura 41. Codificación ordinal

Con este método se codifica cada base como un número. La letra N, no corresponde a ningún nucleótido concreto, sino que puede ser cualquiera de los cuatro nucleótido [A, C, T, G]. Se ha realizado dos tipos de codificaciones diferentes, una con números decimales dentro del rango [0-1] y otra con números enteros (0,1,2,3,4) como se muestra en **Figura 41.**

One-Hot Encoding (OHE)



Es el procedimiento más usado de entre todas las posibilidades de preprocesamiento que pueden existir. Transforma una variable de tamaño m con n posibles valores a asignar en una matriz de m * n donde se le asigna un 1 si está presente esa característica o 0 en caso contrario como se muestra en la **Figura 42.** En cada fila ahora tenemos una lista con las ocurrencias de cada nucleótido en su posición correcta.

Dinucleótidos

Utilizando el método de k-mer counting se hizo una aproximación para el preprocesamiento de las secuencias de DNA. Antes de nada, se va a explicar qué es el k-mer counting. Consiste básicamente en contar las frecuencias con las que aparece una subcadena de DNA de un tamaño mayor de 0.

K-mer counting se utiliza en bioinformática para tareas de análisis de las secuencias de DNA. Se ilustra con un ejemplo:

Inicialmente supongamos que hay un dataset con tres secuencias de 6 nucleótidos: {ACGTTA, ACGTTA, ACGTTT}. De estas hay dos secuencias de 6 diferentes: {ACGTTA, ACGTTT}. Si buscáramos 4-mers existen 9 subcadenas presentes: {ACGT, CGTT, GTTA, ACGT, CGTT, GTTA, ACGT, CGTT, GTTT}. Esto se podría representar viendo la frecuencia de las diferentes subcadenas como: {ACGT: 3, CGTT: 3, GTTA: 2, GTTT: 1} (Manekar & Sathe, 2018)

Con esta idea en la cabeza se cogieron todas las secuencias de DNA de la columna 'secuencia' del dataset y se fue aplicando 2-mers a todas para extraer todas las posibles subcadenas de 2 nucleótidos (dinucléotidos presentes). Una vez que teníamos un dataset con

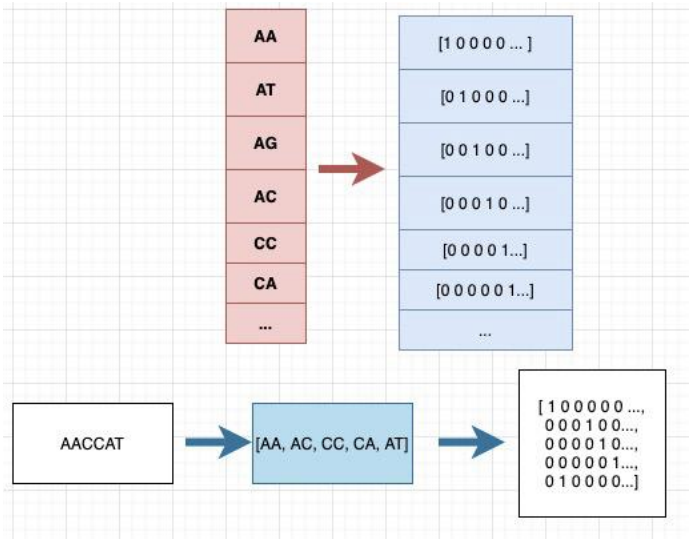


Figura 43. Preprocesamiento dinucleótidos

las secuencias de DNA fraccionadas en subcadenas de 2 nucleótidos se hizo OHE. De esta forma tenían que darse dos condiciones:

- Que el dinucléotido estuviera presente
- Que la posición del dinucleótido en la cadena fuera correcta

4.4. Modelos machine learning

En nuestro caso estamos ante un aprendizaje supervisado, puesto que los datos de entrada están categorizados de antemano. Además, es un problema de clasificación en dos categorías, nucleosoma y no nucleosoma.

Los conjunto de entrenamiento, validación y testeo se crearon en este trabajo aleatorizando el dataset inicial y usando la función split de la librería numpy.

Con cada uno de los datos preprocesados inicialmente se realiza la misma división del DataFrame para tener una consistencia a la hora de comparar los resultados posteriormente. En el caso del preprocesamiento ordinal y OHE se tenían un conjunto de datos de 49998 nucleosomas y 49998 no nucleosoma sobre el que se hará la división en entrenamiento (80%), validación (10%) y test (10%). En cambio, en el caso de preprocesamiento como dinucleótidos había 49998 de nucleosomas y 60000 de no-nucleosomas. **(Figura 44)**

Figura 44. Train, Validation y Test Data

4.4.1. Random Forest

El primer algoritmo de ML que se usó fue Random Forest con los tres tipos de preprocesamiento: ordinal, OHE y dinucleótidos.

Elección de hiperparámetros

Para la elección de hiperparámetros se ha usado el Grid Search de la librería Scikit-Learn explicado en el apartado 2.2.2.4 con un espacio de búsqueda concreto:

- **n_estimators:** es el número de árboles. En mi caso he elegido los siguientes valores [10,20,30,40,50,70,90,100,500,1000].
- **max_features**: el número de características que se miran para la elección de una rama.
"auto" == "sqrt" -> max_features = sqrt(n_features)
"log2" -> max_features = log2(n_features)
- **max_depth:** la máxima profundidad del árbol, es decir, el número de nodos internos que hay antes de llegar a un nodo hoja. Yo he elegido [4,5,6,7,8].
- **criterion**: es la función para medir la calidad de decisión de ramas ['gini', 'entropy']

El índice de 'gini' tiene como objetivo 'medir la frecuencia con la que un elemento elegido al azar del conjunto sería etiquetado de forma incorrecta'. Una puntuación de 0 indica una pureza total, es decir, que no hay datos etiquetados de forma incorrecta. Sus valores están entre [0-0.5]. En cambio, la entropía utiliza logaritmos para medir también el desorden de una variable objetivo. Sus valores están entre [0-1] siendo 0 el valor con menos entropía, menor desorden entre los datos.

Resultados

Se presenta en una tabla los resultados obtenidos en los diferentes preprocesamiento realizados (binario (2 formas), one-hot encoding y dinucleótidos).

Para presentar los resultados obtenidos del preprocesamiento binario entero se coloca en forma de tabla con los 10 primeros hiperparámetros que han dado la mejor accuracy.

criterion	max_depth	max_features	n_estimators	Accuracy
entropy	8	sqrt	1000	0,649
gini	8	sqrt	1000	0,649
entropy	7	sqrt	1000	0,645
gini	7	sqrt	1000	0,644
gini	8	sqrt	500	0,644
entropy	8	sqrt	500	0,644
entropy	7	sqrt	500	0,641
gini	7	sqrt	500	0,640
entropy	8	log2	1000	0,639
gini	8	log2	1000	0,639

Figura 45. Tabla resultados RF para procesamiento binario entero

A pesar de tener un 64,8% de accuracy, hay que tener en cuenta cuántas secuencias de nucleosomas posiciona bien y cuántas mal de cada uno, porque si realiza muy bien una de las dos tareas de clasificación, pero es nefasto en la otra estamos ante un modelo muy poco útil en la práctica. Para poder medir eso se ha realizado la llamada **matriz de confusion**

(MC). La primera fila representa los no nucleosomas y la segunda los nucleosomas. Para una explicación más clara vamos a ver la matriz de confusión de los resultados anteriores de random Forest con la mejor accuracy.

Hay que ver cuántas filas de nucleosomas y no nucleosomas había en el conjunto de test antes de poder realizar la matriz de confusión. En el caso de preprocesamiento binario con enteros ({1,2,3,4,0}) se distribuían en el conjunto de test de la siguiente forma en las dos categorías: {1: 5035} y {0:4965}

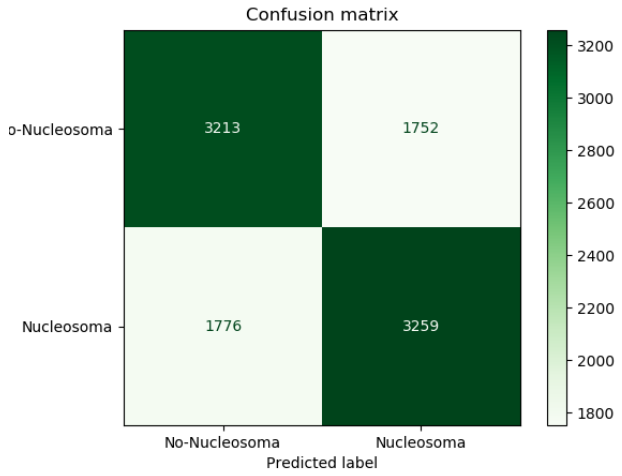


Figura 46. Matriz de confusión Random Forest preprocesamiento binario entero

Partíamos de los datos de testeo: 5035 para nucleosomas y 4965 no-nucleosomas. De estos el modelo ha clasificado:

- VP: ha clasificado bien 3259 nucleosomas.
- VF: 3213 como no nucleosomas
- FP: ha clasificado mal 1776 secuencias como nucleosomas y en realidad eran no nucleosomas
- FN: 1752 clasificados como nucleosomas cuando eran nucleosomas.

También uno se puede percatar de que si hacemos la operación accuracy manualmente de los datos obtenidos:

$$Accuracy = \frac{3213 + 3259}{1752 + 3213 + 1776 + 3259} = 0{,}647$$

Este valor es algo diferente al obtenido en el supuesto mejor resultado del modelo que era 64,9%. Esta inconsistencia de los datos es debida a que el modelo realizado con un el método Grid Search tiene un parámetro cv que es la validación cruzada del modelo. En nuestro caso es 5, por tanto, se harán 5 veces las predicciones con cada conjunto de hiperparámetros y el resultado será la media obtenida de las cinco. Es por eso, que quizá al entrenar el modelo con los mejores hiperparámetros nos de un resultado algo superior.

Ahora que ya se ha visto cómo es el proceso que se ha seguido, se presenta en una tabla los mejores resultados de todos los preprocesamientos para tenerlos en un golpe de vista. A pesar de tener aquí un resumen de los mejores resultados los archivos con los csv de los resultados están subidos junto al código.

	criterion	max_depth	max_features	n_estimators	Accuracy
OHE	gini	8	sqrt	1000	0,638
B-decimal	gini	8	sqrt	1000	0,650
B-entero	entropy	8	sqrt	1000	0,649
dinuc	entropy	8	sqrt	10	0,580

Figura 47. Tabla comparatativa resultados Random Forest

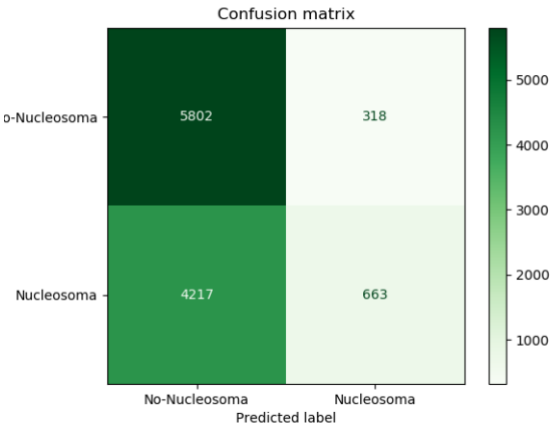


Figura 48. Matriz de confusión dinucleótidos

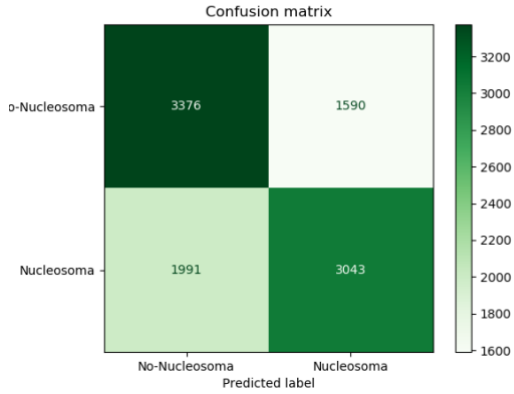


Figura 49. Matriz de confusión binario-decimal

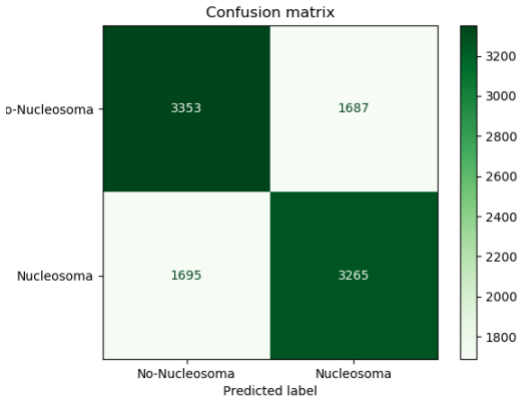


Figura 50. Matriz de confusión OHE

Al observar la tabla de resultados se puede ver que los mejores resultados de tres de los cuatro preprocesamiento comparten tres hiperparámetros igual de todo el espacio de búsqueda que había incialmente:

- max_depth: 8
- max_features: sqrt
- n_estimators: 1000

Aún así ninguno de los casos supera la barrera del 70% de accuracy lo que motivó a seguir probando modelos. El "n_estimators" nos indica el número de árboles que se van a representar en el modelo, usar 1000 árboles podría suponer un trabajo arduo par poder ver cómo trabaja el modelo.

4.4.2. Logistic Regression

Como ya se había explicado previamente el modelo, solo apuntaré que para suu implementación se ha usado la librería de scikit-learn de Python.

Hiperparámetros

Usando la función Grid Search de Scikit-Learn el espacio de búsqueda en este caso es el siguiente:

- **penalty:** diferentes tipos de penalizaciones que se usan en función del valor de C. Normalment 'l1' es la que produce resultados más dispersos y 'Elastic-net' está entre l1 y l2.
- **C**: bastante usado en modelos de clasificación lineal como SVM[7] para determinar el límite de decisión entre dos clases. Cuanto más bajo es C, los errores de clasificación son mayores porque el margen es más amplio. En cambio, si C es grande se empieza a reducir el número de ejemplos mal clasificados. Tiene que ser un número positivo. En mi caso he elegido [0.1,0.3,0.7,0.5,1.0]
- **class_weight:** es el peso que se le da a cada clase, si no se especifica ninguna o 'None' se les da a todas las clases peso 1. Si se pone 'balanced' se ajustan los pesos de la siguiente forma en función del número de clases y de los valores de y:

$$peso\_clase = \frac{numero\_ejemplos}{numero\_clases * valores(y)}$$

- **solver**: algoritmo usado para la optimización del probelma.
    - 'newton-cg', 'lbfgs','sag','saga' funcionan con penalty = l2 o con penalty=None
    - 'liblinear' suelen ser adecuado cuando el datset no tiene muchas entradas.
    - 'sag' y 'saga' funcionan mejor con datasets más grandes
    - 'saga' también soporta el penalty = 'elastic-net'

Resultados

[7] Support Vector Machine: modelo de machine learning

Una vez entrenados todos los modelos con los diferentes Grid Search con sus respectivos modelos y espacios de búsqueda se obtuvieron los reultados en tablas .csv. Todas están subidas junto al código. En la siguiente tabla **(Figura 39)** se van a mostrar los resultados más prometedores.

	C	class_weight	penalty	solver	Accuracy
OHE	0.5	None	l2	sag	0,585
B-decimal	0.1	None	l2	sag	0,545
B-entero	0.1	balanced	l2	liblinear	0,542
dinuc	1.0	None	l2	lbfgs	0,767

Figura 47. Tabla comparativa resultados Logistic Regression

Sin lugar a duda el mejor preprocesamiento para este modelo es el preprocesamiento de dinucleótidos. Para ver si realmente ese 76,7% clasifica bien ambas categorías hay que mirar la matriz de confusión.

Ahora bien, si nos fijamos se encuentran 6120 secuencias de no-nucleosomas en total, entre los VP y FP **(Figura 40).** Además, hay 4880 secuencias de nucleosoma, Por lo que en total hay 11000 secuencias, dato que no coincide con el resto de preprocesamientos que eran 10000 secuencias para el conjunto de testeo. Esto es porque el dataset inicial de dinucleótidos que formamos era mayor que en el resto de preprocesamiento (indicado anteriormente), por lo tanto, al hacer el train-test split con un 10% para los datos de validación y 10% para datos de test nos quedaba de esta forma. Si se realiza el cálculo de la precisión con estos datos obtenemos lo siguiente:

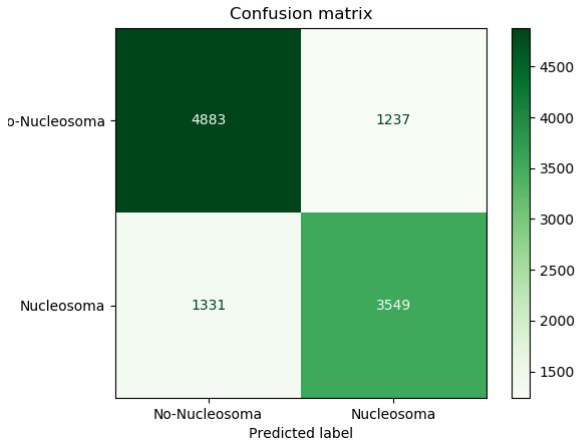


Figura 48. Matriz de confusión Logistic Regression preprocesamiento dinucleótidos.

$$Accuracy = \frac{4883 + 3549}{4883 + 3549 + 1331 + 1237} = 0{,}7665$$

4.4.3. K-NN

En este caso el único hiperparámetro que se ha elegido es K, el número de vecinos. Se ha establecido que puede ser un número entre el intervalo (10,51).

Resultados

Se presenta al igual que con los anteriores modelos los mejores resultados.

	n_neighbors	Accuracy
B-float	50	0,589
B-entero	48	0,575
OHE	50	0,63

Figura 49. Tabla comparativa resultados KNN

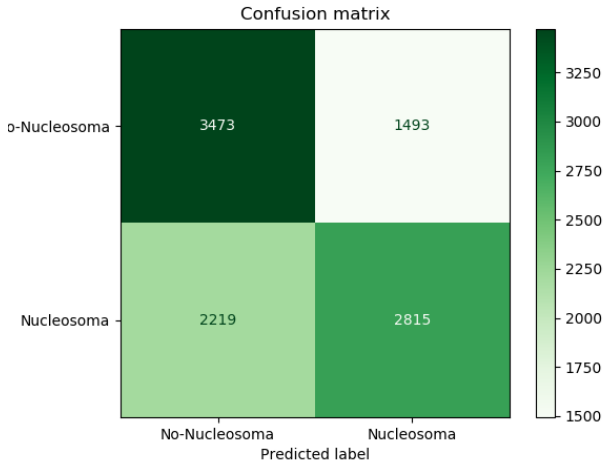


Figura 50. Matriz de confusión KNN con OHE

Se ha realizado la matriz de confusión con los hiperparámetros del modelo con los mejores resultados. Aún siendo un accuracy (63%) resulta que para los no nucleosomas realiza una predicción más menos adecuada, pero en cambio de nucleosomas ronda un 50% las veces que acierta la categoría.

También hay que anotar que aquí el modelo de preprocesamiento de dinucleótidos no era adecuado porque no entrenaba correctamente los modelos. Con todas estas dificultades se siguió probando nuevos modelos de ML.

4.4.4. Decision Trees

Hiperparámetros

En este modelo también implementado en la librería de Scikit-Learn de Python existen una serie de hiperparámetros que se pueden modificar según diferentes criterios. En este caso se eligieron dos hiperparametros dentro del espacio de búsqueda del Grid Search:

- **Criterion**: es el criterio elegido para la poda. Puede ser gini o entropy que es para la ganancia de información.

1. Ganancia de información

Esta medida se basa en la entropía[8] que se genera cuando se divide el dataset en función de algún atributo. Calcula cuánta información nos proporciona una característica acerca de una clase. Según este valor se decide dividir el dataset en un nodo interno y seguir construyendo el árbol de decisión.

$$Information\ Gain\ =\ Entropía\ (S)\ -\ [(peso\ medio)\ *\ Entropía\ (cada\ característica)]$$

2. Índice Gini

También es una medida de la impureza o pureza cuando se está construyendo el árbol de decisión. Una característica con un índice bajo de Gini suele ser mejor que uno alto.

- **class_weight:** peso que se le da a las clases. En este caso se ha elegido hacer un balanceo de las clases. "balanced"

Resultados

Se presenta en una tabla los resultados obtenidos con el método de búsqueda Grid Search con las combinaciones posibles del espacio de búsqueda marcado.

	class_weight	criterion	Accuracy
B-entero	balanced	entropy	0,522
B-float	balanced	gini	0,519
OHE	balanced	gini	0,519

Figura 51. Tabla resultados del modelo de árboles de decisión

Como se apuntaba una de las desventajas del modelo es que no suele tener un accuracy muy bueno comparado con otros algoritmos. No se va a mostrar la matriz de confusión porque los resultados no son en ninguno de los preprocesamientos demasiado correctos, algo similar a hacer de forma aleatoria la clasificación. Sin embargo, todas las tablas csv con los resultados y las matrices de confusión están accesibles para que puedan ser consultadas.

4.4.5. Naive Bayes

[8] Medida de la impureza de un atributo. Especifica la aleatoriedad presente en los datos

La clase *sklearn.naive_bayes.BernoulliNB* usada para implementar el modelo con GridSearch como en los previos casos perteneciente a la librería Scikit-Learn tiene una serie de hiperparámetros para modelar y optimizar los resultados. En mi caso se ha elegido dos hiperparámetros que conformarán el espacio de búsqueda:

- **alpha**: valores elegidos [0,0.25,0.4,0.7,1.0]
- **binarize**: valor para determinar el margen en el que se clasifican los valores en dos categorías sino eran clases binarias desde un principio. Se ha elegido [0,0.2,0.3,0.5,1.0]

Resultados

Como suele ser la tónica habitual se aglutinó en una tabla los mejores resultados obtenidos con los diferentes preprocesamientos.

	alpha	binarize	Accuracy
OHE	1.0	0.5	0,584
B-entero	1.0	1.0	0,550
B-decimal	0.7	0.3	0,548

Figura 52. Tabla resultados Bernoulli Naïve Bayes

Quiero realizar una serie de observaciones acerca de los datos obtenidos en este modelo. No se ha visto una mejora considerable de unos hiperparámetros a otros, de hecho, en el caso del preprocesamiento OHE no hay más que dos accuracy de los 24 resultados que ha dado el Grid Search al hacer las diferentes combinaciones posibles con Cross-Validation.

Se muestra la matriz de confusión para el mejor resultado. No podríamos considerar este modelo como un posible éxito porque prácticamente el 50% de las veces realizó una clasificación errónea de los datos, lo que en ningún caso nos serviría como modelo fiable para poder usar en un ámbito científico donde la rigurosidad es uno de los máximos exponentes.

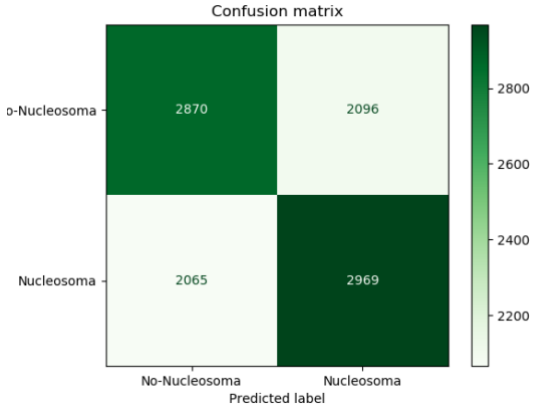


Figura 53. Matriz de confusión Bernoulli Naïve Bayes

Mejor modelo de Machine Learning

Los resultados expuestos en las páginas previas es un resumen de los modelos más importantes probados teniendo en cuenta la clasificación binaria que queremos lograr. La mayoría dejan mucho que desear, estando bastante por debajo de lo que se quería a priori.

Resultó complicado encontrar la forma de codificar como números las diferentes cadenas en crudo de DNA que fueron facilitadas. Las aproximaciones que se hicieron quizá no fueron las más acertadas. Se pueden destacar fallos relevantes como la posibilidad de haber tenido en consideración en qué orden se daba los diferentes valores numéricos a los nucleótidos, no se tuvo en cuenta si eso tenía algún tipo de importancia para los resultados obtenidos. Quizá hubiese sido adecuado haber visto la frecuencia con la que aparecen los diferentes pares de bases para dar un peso previo a su entrenamiento en los modelos. Se partió de un modelo en crudo porque al fin y al cabo todavía no hay evidencias claras y definitivas de cómo la secuencia de DNA contribuye al posicionamiento de los nucleosomas.

- **Preprocesamiento**: dinucleótidos
- **Algoritmo de machine learning**: Logistic Regression
- **Hiperparámetros:**
  - C: 1.0
  - class_weight: None
  - penalty: l2
  - solver: lbfgs
- **Accuracy:** 0,767

Aún teniendo las primeras dificultades en el preprocesamiento y en la computación en paralelo para exprimir al máximo los recursos que se habían prestado, se obtuvo un modelo con un accuracy aceptable.

La comparación de los modelos de machine learning se hizo con la exactitud porque es la medida más entendible y fácil de aplicar para después elegir la mejor solución. Como esta a veces se queda corta se usó la matriz de confusión, ya explicada anteriormente. Viendo que en este modelo ambas cosas daban un resultado adecuado se decidió escoger otras métricas como sensibilidad, precisión o F-1.

Todas las medidas que se presentan tienen mucho que ver con que el modelo sea de clasificación binario, si hubiésemos tenido una clasificación en diferentes categorías las

```
Matriz de confusion

[[4883 1237]
 [1331 3549]]
```

métricas son diferentes. Partiendo de los datos obtenidos de la matriz de confusión [TP, FP, TN, FN] que en nuestro caso eran los siguientes:

{TP, 4883} {TN, 3549} {FP, 1237} {FN, 1331} Con estos datos se van a obtener el resto de las métricas.

El uso de estas métricas va a depender de qué es lo que se busque. En nuestro caso concreto lo que queremos encontrar es un modelo que clasifique bien los nucleosomas. Tal como hemos representado la matriz de confusión en nuestro caso se busca un ratio de FN alto, es decir, que haya identificado el mayor número de clases de nucleosomas adecuadamente

Para hacerlo más visual se ha realizado una tabla con las diferentes métricas asociando cada una a la categoría que corresponde en cada caso y que posteriormente se comparará con el resultado obtenido en Python usando la clase *sklearn.metrics.classifiction_report*.

	No-nucleosomas [0]	Nucleosomas [1]
Precision	$\frac{4883}{4883+1237} = 0{,}79$	$\frac{3549}{3549+1331} = 0{,}73$
Recall	$\frac{4883}{4883+1331} = 0{,}8$	$\frac{3549}{3549+1237} = 0{,}74$
F-1	$\frac{2}{\frac{1}{0{,}79}+\frac{1}{0{,}8}} = 0{,}79$	$\frac{2}{\frac{1}{0{,}73}+\frac{1}{0{,}74}} = 0{,}73$

Figura 54. Tabla comparativa métricas clasificación binaria

Ahora se muestran los resultados obtenidos con las propias librerías de scikit-learn de Python.

Classification report

	precision	recall	f1-score	support
0	0.79	0.80	0.79	6120
1	0.74	0.73	0.73	4880
accuracy			0.77	11000
macro avg	0.76	0.76	0.76	11000
weighted avg	0.77	0.77	0.77	11000

Figura 55. Resultados obtenidos del modelo con Python

También se representó la curva ROC para ver cómo de bueno es el clasificador frente a un clasificador aleatorio (clasifica 50% de las categorías de forma correcta).

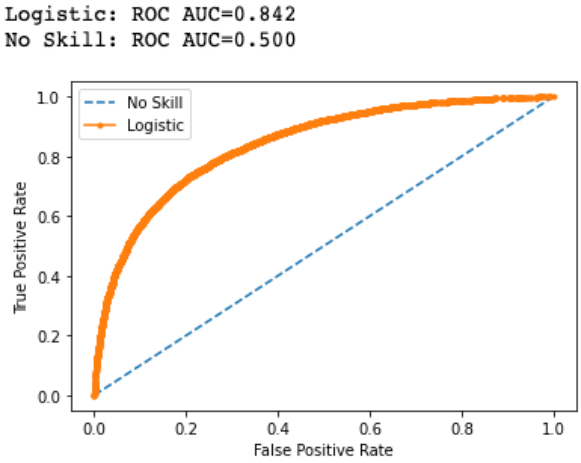


Figura 56. Representación ROC del modelo Logistic Regression

Los algoritmos de ML son de utilidad para una gran cantidad de problemas, pero quizá sean ineficientes para otros muchos casos. A medida que ha avanzado la rama de la inteligencia artificial ha habido casos como reconocimiento de imágenes o procesamiento de lenguaje natural (NPL) en los que se ha visto la necesidad de probar nuevos algoritmos. Ahora bien, ¿qué problemas asociados al machine learning han contribuido a este cambio?

Como dijo Thomas Dietterich, profesor de la Universidad del estado de Oregón, "Con los recientes avances en machine learning, podemos tener sistemas que pueden [reconocer objetos en imágenes, reconocimiento del lenguaje, y traducir diferentes idiomas] con una exactitud similar a la del ser humano." Bien es cierto que ML requiere de una selección o adaptación de las características previo a entrenar el modelo. Esta limitación se ha mejorado con el DL que permite introducir datos en crudo como imágenes, texto o audio sin definir o extraer características previamente. DL es capaz de aprender esas características a partir de los datos de entrada. Sin embargo, requiere una capacidad de cómputo mayor y mayores habilidades. Los algoritmos de DL son más complicados de entender y requieren mayor conocimientos estadísticos e informáticos. Sin embargo, hay formas de combinar ambos tipos. (Dietterich, 2017)

Por todo esto, se decidió empezar a probar modelos de DL y abandonar ML. Bien es cierto que el planteamiento inicial de este trabajo era únicamente trabajar con modelos de DL, pero viendo la dificultad de poder entender cómo funcionan estas arquitecturas se empezó por modelos de ML con la esperanza de mejorar el desempeño de la solución.

4.5. Modelos de Deep Learning

En este apartado se va a explicar qué modelos de DL se han usado para intentar mejorar los resultados de clasificación obtenidos y comparar también varios tipos de algoritmos de DL para finalizar con el modelo que a priori hasta ahora mejor clasifica para nuestro caso de estudio.

4.5.1. Representación de los datos para las redes neuronales

A priori con que las secuencias estuvieran almacenadas en alguno de estos tensores es

	OHE	**Dinucleótidos**
Nº datos (shape[0])	99996	109998
Alto (shape[1])	150	149
Ancho (shape[2])	5	17

condición más que suficiente para alimentar a la red neuronal. Pero después de hacer alguna prueba con las secuencias en crudo de DNA y con redes neuronales Deep-Forward simples con un par de capas densas y ver que los resultados rondaban el 50% de accuracy se decidió cambiar un poco las estructuras de los datos para convertirlos a tensores 3D.

Si se recuerda en los modelos de ML se realizaron cuatro tipos de preprocesamiento diferentes: codificación binaria [0,1,2,3,4], codificación decimal [0.25, 0.5, 0.75, 1.0], OHE y dinucleótidos. Para los modelos de Deep Learning se probó con dos de ellas modificadas ligeramente:

1. OHE
2. Dinucleótidos

1. OHE: se eligió esta porque simplemente consiste en asignar un 0 o 1 a cada una de las posiciones de la secuencias de DNA si se trata del nucleótido en cuestión. Prácticamente es como si tuviéramos la secuencia en crudo de DNA con una pequeña modificación a la hora de representar la propia cadena. Ahora la dimensión va a ser (nº datos, 150, 5), siendo 150 la longitud de las cadenas y 5 los nucleótidos [A, G, C, T, N]. Cada una de las filas del dataset pasaría a ser:

Ejemplo, nucleótido A = ([1.0, 0, 0, 0, 0] * 150 numpy arrray) * N (nº datos)

2. Dinucleótidos: al igual que en ML se usa 2-mers para sacar los dinucleótidos contenidos en todas las cadenas de DNA de 150 nucleótidos, quedando 149 parejas de dinucleótidos por nucleosoma con 17 posibles dinucleótidos. El diccionario de dinucleótidos obtenido es [aa, ag, ac, at, tt, ta, tc, tg, gg, gt, ga, gc, cc, ca, cg, ct, nn].

Con esto y la librería *sklearn.preprocessing.LabelEncoder* se asigna a cada dinucleótido un número para luego volver a hacer OHE. La dimensión final del tensor va a ser (nº datos, 149, 17).

Figura 57. Tabla de datos de entrada modelos Deep Learning

En la anterior tabla se muestra el conjunto de datos inicial. Por lo que en ambos casos se ha buscado un tensor3D que luego se podrá cambiar las dimensiones para ajustarlas a las diferentes capas de entrada de las redes neuronales que solo soportan ciertas dimensiones como input.

La división de los conjuntos en entrenamiento, validación y test se hace posteriormente. En nuestro caso, teniendo ya la experiencia previa de los modelos de machine learning y habiendo observado que normalmente se cumple una tónica habitual de que no es necesario un conjunto de test y validación demasiado grande. La división se hace de 80% entrenamiento, 10% test y 10% validación.

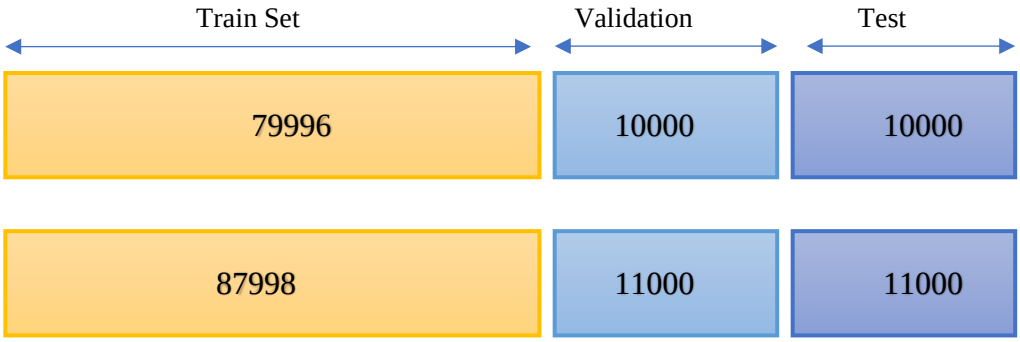


Figura 58. Reparto del conjunto de datos original en entrenamiento, validación y test.

4.5.2. Primer modelo

En DL encontrar los mejores hiperparámetros para el modelo resultó una tarea tediosa. El número de hiperparámetros es superior a los modelos de ML: número de capas, número de batches, epochs, dropout, número de neuronas/capa y un largo etc. Esto hace aún más complicado encontrar el modelo que encaje perfectamente.

Se ha seguido en todos los modelos de DL una estrategia que, aunque sea costosa y larga, podría considerarse una de las más útiles en ámbitos académicos y científicos porque favorecen el aprendizaje y el entendimiento del proceso que se sigue. Esta técnica es prueba y error o también llamada "Babysitting". Es decir, primero se hace un estudio de los datos de entrada, se conoce cuál es el problema y finalmente se prueban diferentes combinaciones y en función de los resultados se decide el camino a seguir. Puede verse las similitudes con el método científico y sus diferentes pasos hasta llegar a una solución partiendo de la hipótesis planteada.

La arquitectura del primer modelo se muestra en la **figura 63**.

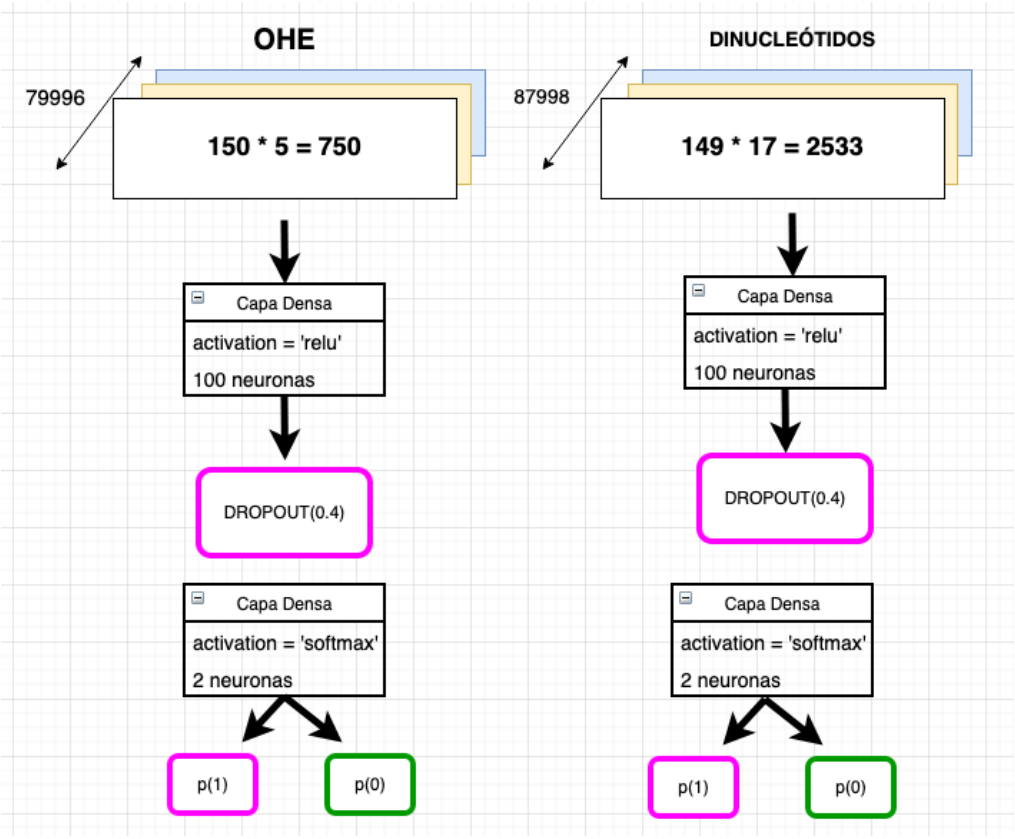


Figura 59. Arquitectura primer modelo DL con una capa densa, dropout y una última capa densa.

En la propia imagen se pueden ver las diferencias que residen básicamente en la capa de entrada, teniendo cada una las dimensiones de sus secuencias calculadas multiplicando el número de nucleótidos o dinucleótidos por las posibles valores que puede tomar.

Además de los hiperparámetros de cada una de las capas elegidos como se muestra en la figura para el entrenamiento se tomaron:

-**'loss'**: categorical_crossentropy
-**'optimizer'**: SGD
-**epochs**: 25 (dinucleótidos) y 100 (OHE)
-**batch size**: 256

En DL se empezó por adoptar el optimizador de 'adam' presentado por Diedrik Kigma y Jimmy Ba en 2015 en el artículo "Adam: a method for stochastic optimization". Se utiliza en redes neuronales convolucionales y se adapta a datos de entrada grandes o con muchos parámetros. En el estudio se muestran gráficas muy alentadoras sobre su rendimiento y grandes velocidades de entrenamiento, por lo que muchos científicos de datos lo han usado como optimizador para sus modelos de aprendizaje profundo. Pero Wilson et al mostró en su

artículo 'El valor marginal de los métodos de gradiente adaptativo en el aprendizaje automático' que los modelos adaptativos (como Adam o Adadelta) no generalizan tan bien como SGD con impulso cuando se experimenta con diferentes tareas de aprendizaje, produciendo overfitting. Sobre todo, se resaltaba el hecho de que llegado ciertas etapas de entrenamiento se satura.

Después de leer estos artículos decidí cambiar mi optimizador porque a pesar de que el accuracy era bueno, había una gran diferencia entre el valor de validación y de entrenamiento, produciendo problemas de *overfitting.* No pensé en un principio que fuera un problema del optimizador, pero se puede ver la diferencia considerable en las gráficas. **(Figura 64)**

Viendo las dos gráficas uno se puede dar cuenta del 'overfitting' que se produce al usar

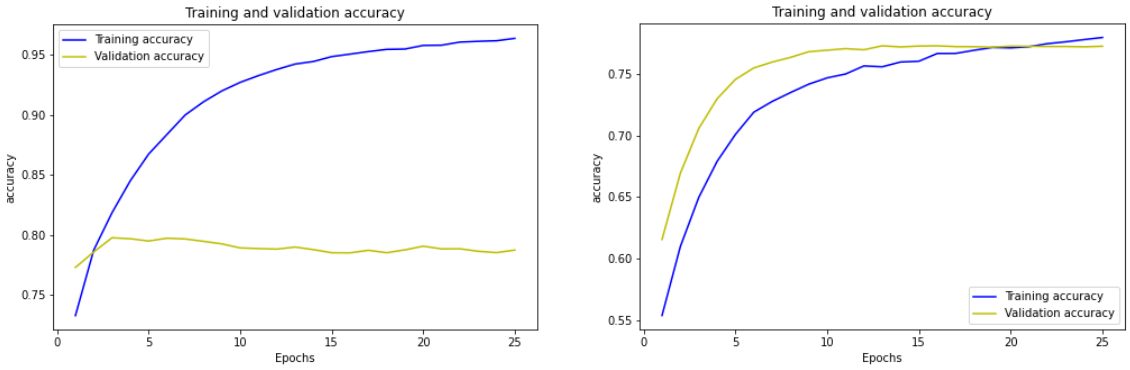


Figura 60. Gráficas de accuracy frente al número de epochs entrenados. [1] se trata del modelo con el optimizador Adam [2] El mismo modelo usando el optimizador SGD

'adam', cosa que mejora considerablemente al usar SGD. A partir de este momento se tuvo en cuenta el optimizador de SGD si se observaba 'overfitting' en algunos de los resultados.

El optimizador SGD funciona de forma similar al algoritmo de descenso de gradiente que se explicó en el apartado 2.2.3.1, pero con la diferencia que no se hace el cálculo del gradiente (derivada parcial) por cada dato de entrada y característica, sino que elige de forma aleatoria un dato, de esta forma se reduce la carga computacional de forma considerable.

Resultados

Tras ver la arquitectura del modelo con solo tres capas: dos densas y un Dropout, se presentarán los entrenamientos de ambos modelos para el preprocesamiento de dinucleótidos y OHE. Se presentan las gráficas de la métrica de pérdida, que el objetivo es lograr que sea la menor posible y la de accuracy que se intenta que sea la máxima posible.

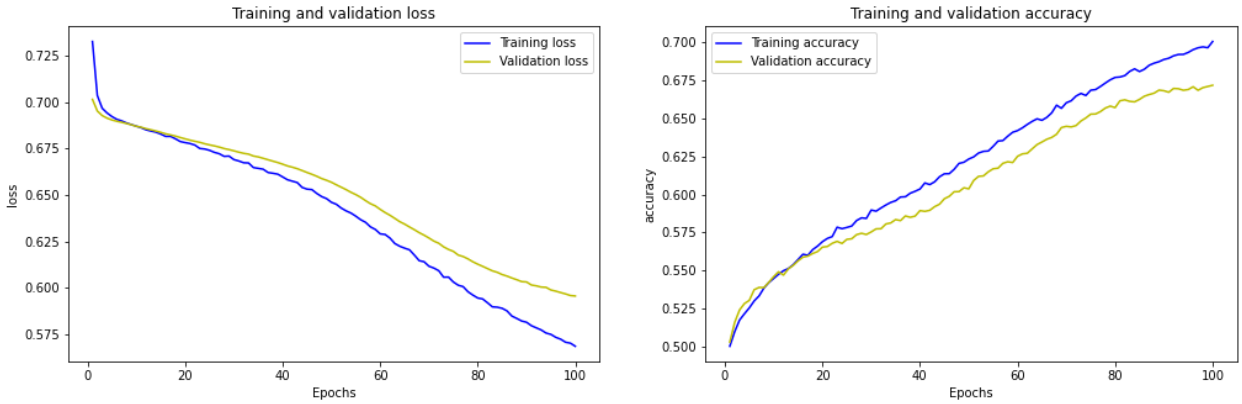


Figura 61. Gráficas de desempeño primer modelo DL con preprocesamiento OHE

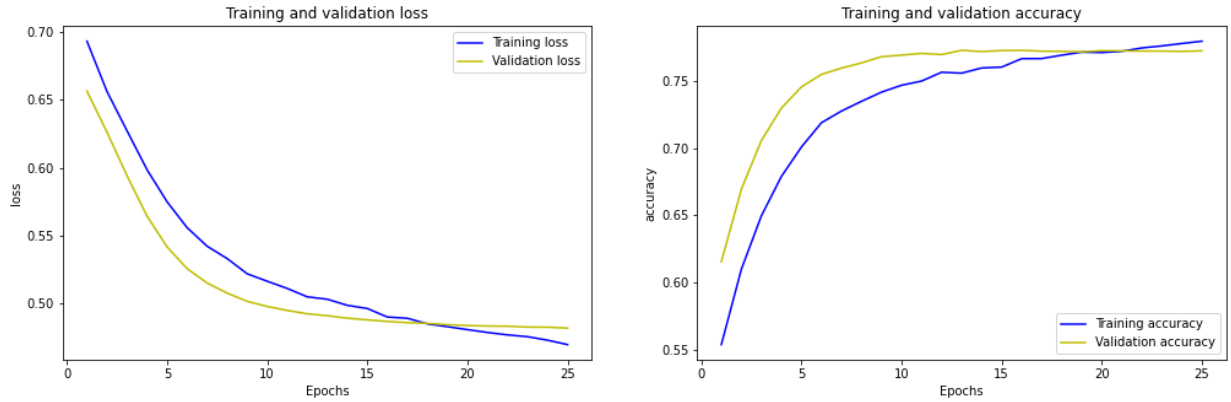


Figura 62. Gráficas de desempeño primer modelo DL con preprocesamiento dinucleótidos

Al ver las gráficas se observa en las primeras de ellas (**Figura 65)** un buen modelo debido a que no hay prácticamente 'overfitting'. Se empieza a intuir un poco a partir de la época 22, pero hasta entonces parece que evoluciona bien el modelo. En cambio, en el segundo preprocesamiento **(Figura 66)** sí que vemos 'overfitting' debido a que la exactitud de el conjunto de validación es un poco inferior al entrenamiento, aún así no tiene un mal desempeño. Hay que anotar que estos dos modelos previamente con el optimizador 'adam' daban resultados mucho peores, produciendo 'overfitting' en ambos casos. También es destacable que en el segundo caso se hizo con 100 épocas y en el primero con 25. Esto es debido a que los modelos a base de prueba-error se fueron probando diferentes épocas para ver cómo evolucionaban las métricas. En el caso de OHE, la evolución iba aumentando sin

producir 'overfitting', por lo que se fueron ampliando hasta ver cuándo se llegaba a un punto que ya no pudiera mejorar más.

Para comprobar si realmente el modelo clasifica bien, se volvió a usar la matriz de confusión en ambos casos para ver su precisión y la tasa de verdaderos negativos también.

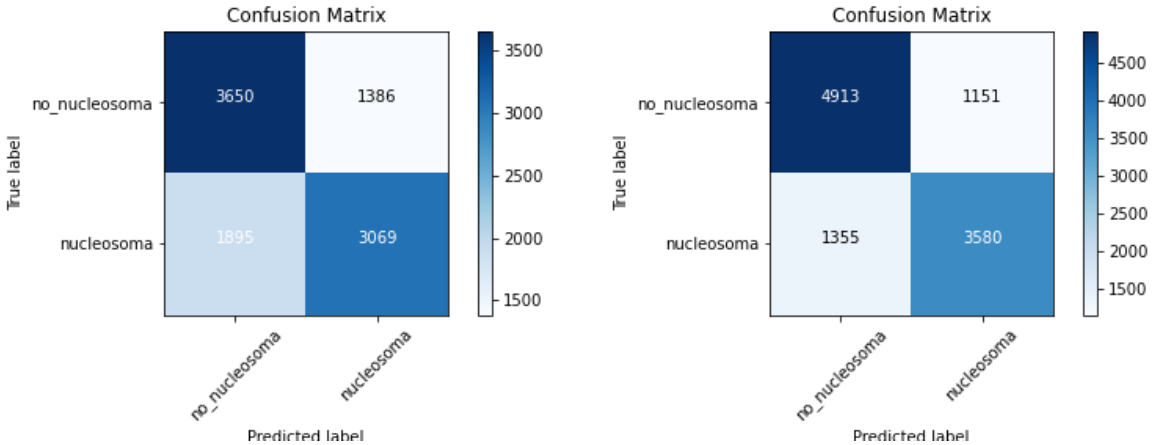


Figura 63. Matrices de confusión del primer modelo DL

La primera matriz de confusión corresponde a OHE con un accuracy de 67,18% y 'loss' de 0.59. Aún teniendo este *accuracy* no demasiado bueno realiza una clasificación similar a ambas categorías. Algo mejor resulta el preprocesamiento de dinucleótidos con un accuracy de 77,22% y 'loss' de 0.48. Este supera un poco al modelo de machine learning, sin embargo; se siguió probando con redes neuronales convolucionales (CNN).

4.5.3. Segundo modelo DL

En la **figura 68** se puede ver la arquitectura del modelo CNN con capas de convolución 1D, que a priori son las adecuadas para secuencias (DNA). Por otro lado, existe una segunda capa de MaxPooling, seguida de dos capas densas, una oculta y otra de salida.

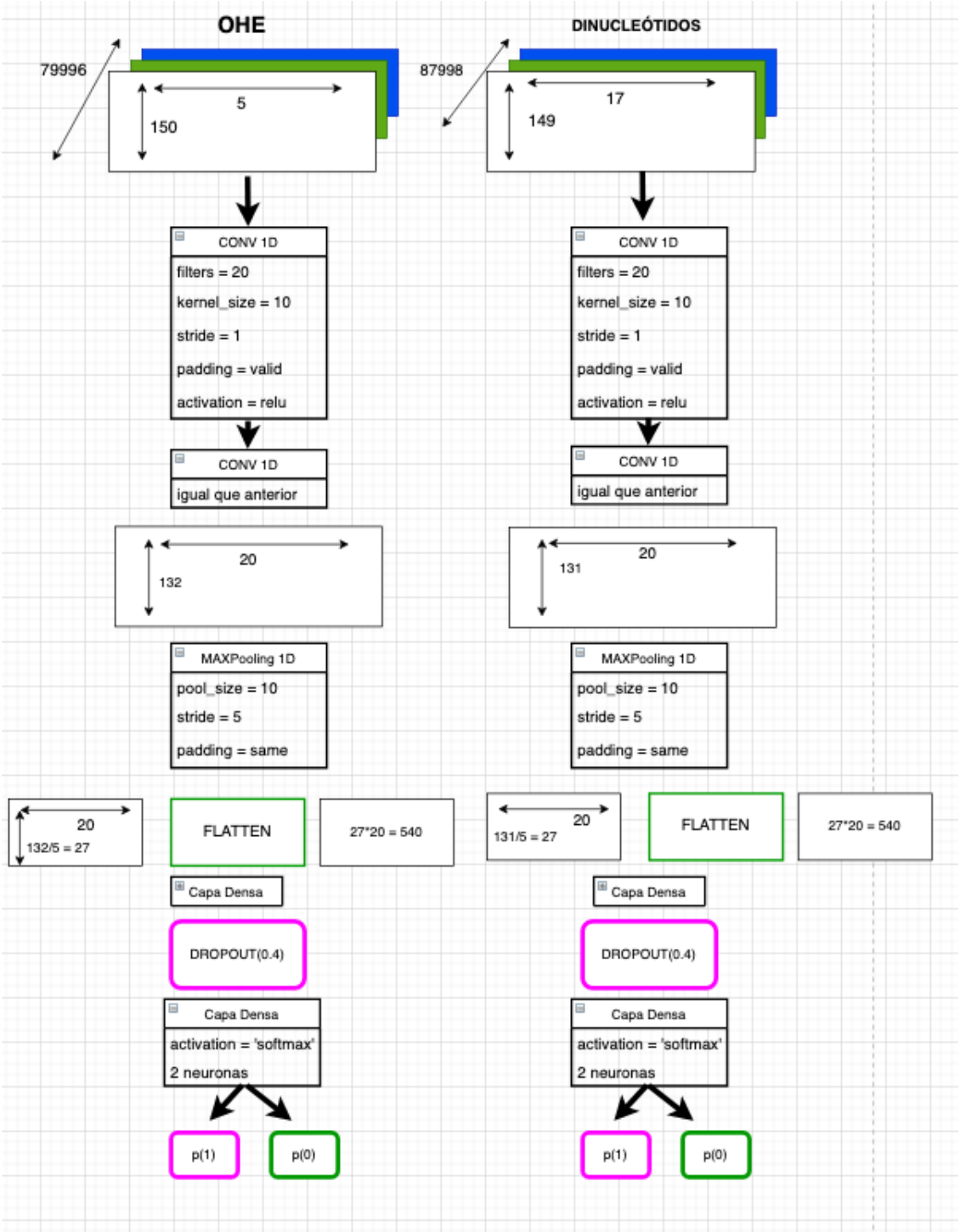


Figura 64. Arquitectura modelo 2 DL con capas convolucionales 1D.

Se va a explicar la red tomando como ejemplo el preprocesamiento de los dinucleótidos, arquitectura de la parte izquierda. Para el caso de OHE sería lo mismo solo que con las dimensiones propias para ese caso.

- Capa Conv1D: se utilizan 20 filtros con anchura 10. Se irá realizando la operación de convolución ya explicada por todos los datos de entrada. El desplazamiento del filtro por toda la muestra generando un mapa de activación que se pasará a la siguiente capa. Los 20 vienen de los 20 filtros usados y 149 (tamaño inicial) menos los 10 del tamaño del filtro es 140. Hay una segunda capa también 'Conv1D' igual que la anterior, por tanto, el output que se pasará a la capa 'MaxPooling' es de (131, 20).

Como se puede ver en la figura, el stride es 1, desplazándose de uno en uno; mientrás que el padding es valid, esto recordamos que era que disminuía la dimensión de los datos de salida repecto a los de entrada de la capa correspondiente.

- Capa 'MaxPooling1D': La capa 'MaxPooling' reduce la dimensionalidad en función del tamaño del 'stride' (la ventana), en este caso en 5, por tanto, 131/5 = 27 generando una dimensión de salida de la siguiente capa (27,20).

- Capa 'Flatten': La capa 'Flatten' forma un mapa de una sola dimensión con el mapa de características de la anterior capa formando un vector (27*20) -> (540), que entrará en la capa densa totalmente conectada de 128 neuronas del modelo MLP, con salida 128 que ya entrarán en la última capa de la arquitectura.

- Capa 'Dense': son capas totalmente conectadas, la primera de 128 neuronas y la segunda será la que clasificará en las dos categorías pertinentes (nucleosoma y no-nucleosoma).

Hiperparámetros compilación y entrenamiento del modelo:

	OHE	**dinucleótidos**
Batch size	**64**	**64**
Epochs	**15**	**30**

Por lo tanto, esta arquitectura añade con respecto a la anterior dos capas Conv1D y una capa 'MaxPooling1D'. Esto implica aplicar el proceso de convolución y una reducción de dimensiones con la estrategia *max pooling*.

Resultados

El proceso seguido para comparar los resultados fue el mismo que en el modelo 1, primero se hicieron pruebas para tantear los mejores hiperparámetros a través de un proceso de prueba y error y cuando ya se tenían los hiperparámetros decididos se entrenó con los datos de preprocesamiento de dinucleótidos y OHE.

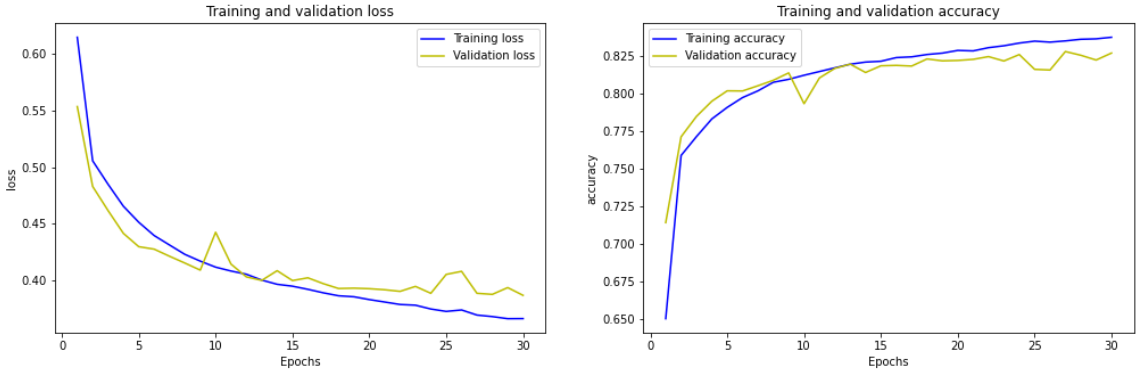


Figura 65. Gráfica modelo 2 DL preprocesamiento dinucleótidos

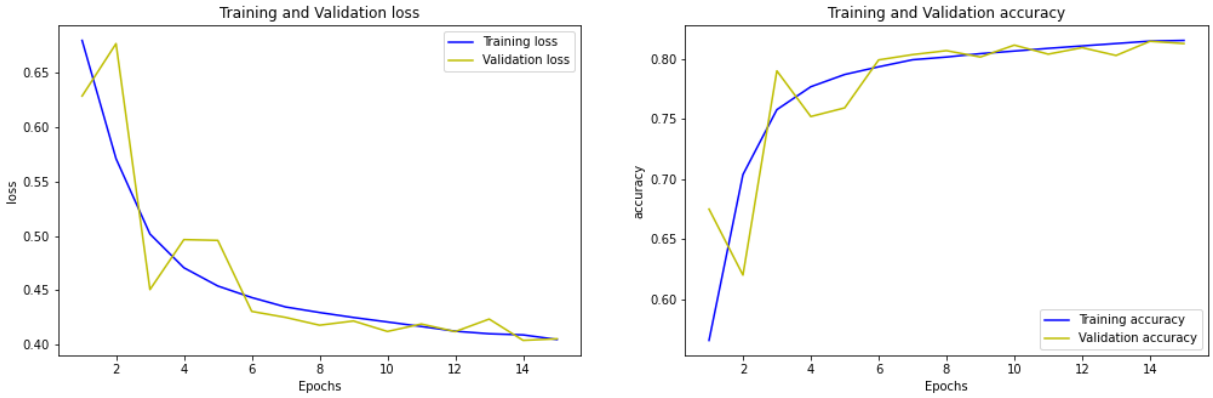


Figura 66. Gráfica modelo 2 DL preprocesamiento OHE

Si analizamos las gráficas se puede ver cómo en la **Figura 69** se produce un poco de 'overfitting', el modelo está aprendiendo sólo los datos de entrenamiento. Aún así no es muy marcado, por lo que no estaríamos ante un mal modelo. Por otra parte, el *accuracy* a lo largo de los epochs va aumentando, por eso, se probó hasta 30 épocas con este modelo. El *accuracy* queda cercano al 80%. En la **Figura 70** no existe ningún indicio aún de 'overfitting', pero ya se ve cómo la línea de la exactitud se empieza a estabilizar por lo que tampoco tiene sentido entrenar más épocas. Lo que si se puede observar en ambos casos que los resultados de la validación tienen picos y valles donde mejoran y empeoran los resultados no siguiente un entrenamiento lineal.

En cuanto al accuracy conseguido, en dinucleótidos es de 82,85% y en OHE de 81,7%. Aunque es muy similar en ambos, vuelve a ser mejor para los dinucleótidos. Si vemos en las matrices de confusión **(Figura 71)** que existe una clasificación correcta de ambas categorías.

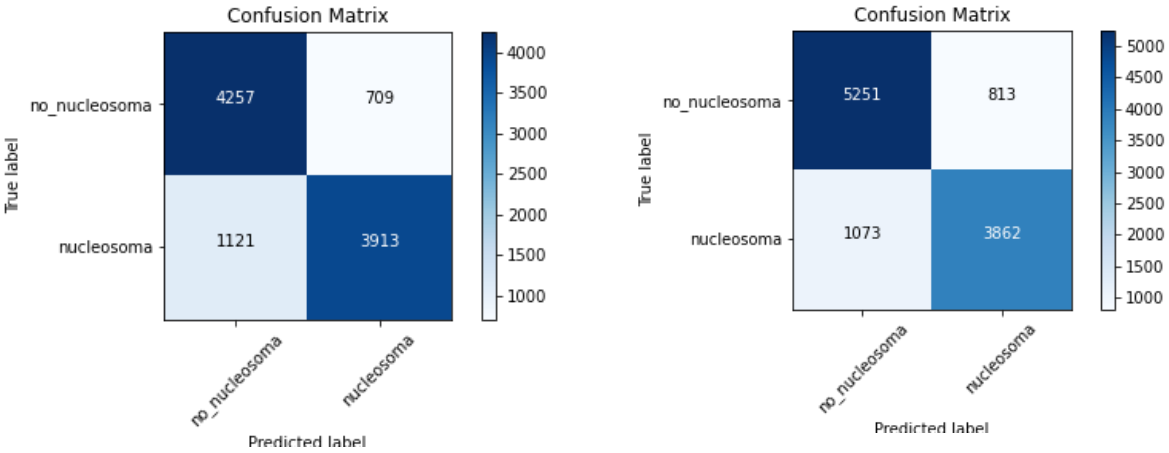


Figura 67. Matrices de confusión modelo 2 DL. [1] OHE. [2] dinucleótidos

4.5.3. Tercer modelo DL

Uno de los campos de implementación de modelos DL es el reconocimiento de imágenes. A priori no parece muy factible convertir una secuencia de DNA en una imagen. Sin embargo, ya hay datos en la literatura sobre estas transformaciones. Procesaron secuencias de DNA como imágenes en escala de grises. Cada nucleótido se representaba por un número decimal entre 0 y 1 para después entrenar una CNN. (Santamaría. H et al.)

Puede que sus resultados fueran buenos, pero no estaba claro cuál era el criterio para elegir los diferentes valores para los nucleótidos; así que, se decidió entrenar los modelos de CNN 2D con las estructuras de datos que ya se tenían, pero cambiando la dimensión de estas. Para hacer el cambio de dimensión se hizo lo siguiente:

1. Extraer la longitud de la secuencia en un vector único. Se multiplican los valores para ello. En el caso de OHE = 150*5 = 750 y dinucleótidos 149*17 = 2533.
2. Cambiar las dimensiones. En el primer caso se pudo cambiar por la siguiente dimensión (25, 10, 3) siendo si se compara con imágenes (alto, ancho, número de canales). Lo más normal es que haya tres canales (Red, Green, Blue). Para dinucleótidos no era tan sencillo porque no había divisores enteros. Por tanto, simplemente se puso los valores en un canal -> (149, 17, 1) **(Figura 72)**

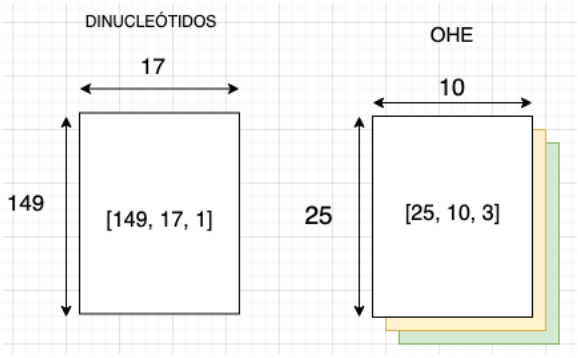


Figura 68. Representación datos de entrada tercer modelo de DL

Con esto ya se tenían los datos con la dimensión adecuada para alimentar a la CNN 2D que se quería construir. La diferencia de este modelo con respecto al modelo 2 está en que la dimensión de los filtros es de dos dimensiones en vez de una, es decir, tiene ancho y alto. Por eso, los datos de entrada para este tipo de redes tienen que ser tensores 3D (alto, ancho, número de canales). La explicación de la arquitectura del modelo se va a hacer con el preprocesamiento de OHE, pero con dinucleótidos solo varían los números y el número de filtros o neuronas en alguna capa, estos cambios se mencionarán:

- Conv2D: con tamaño de filtro (3, 3), es decir, 3 de alto por 3 de ancho. Será el que realizará la convolución por todos los datos de entrada por cada uno de los filtros.

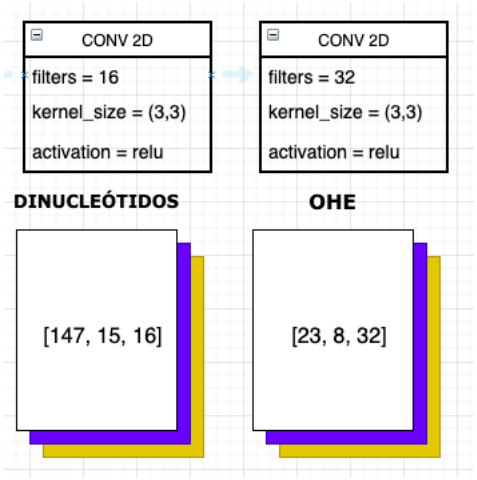


Figura 69. Diagrama para representar la capa Conv2D

-MaxPooling2D: hace lo mismo que MaxPoolign1D, pero ahora la ventana es de (2, 2).

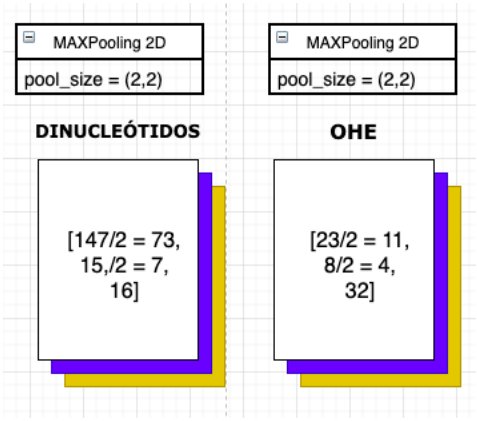


Figura 70. Diagrama capa MaxPooling 2D

- Conv2D: con otro filtro de (3,3). Dejando la salida en (9, 2, 256) En este capa se usaron 256 filtros y 64, respectivamente.

- Flatten, Dense (128) y Dense (2): Primero juntamos todo el conjunto de características en un vector y luego hay dos capas de neuronas totalmente conectadas. La última es la que clasifica en las dos categorías.

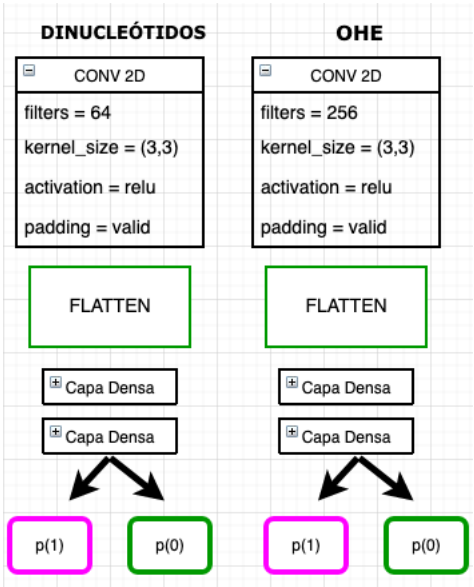


Figura 71. Diagrama con el resto de capas del tercer modelo DL.

	OHE	**dinucleótidos**
Batch size	**64**	**128**
Epochs	**20**	**20**

Una cosa que me gustaría destacar es la elección otra vez del optimizador SGD puesto que se probó con 'rmsprop' que daba una exactitud de entrenamiento cercana al 90%, pero, sin embargo, generaba un gran 'overfitting'. **(Figura 63)** No tiene sentido un modelo que clasifique muy bien para los datos que se le pasan como entrenamiento, pero no sea capaz de generalizar para nuevos datos. La elección estaba clara porque con SGD hay un ajuste mayor entre entrenamiento y validación, a pesar de tener una exactitud menor de entrenamiento.

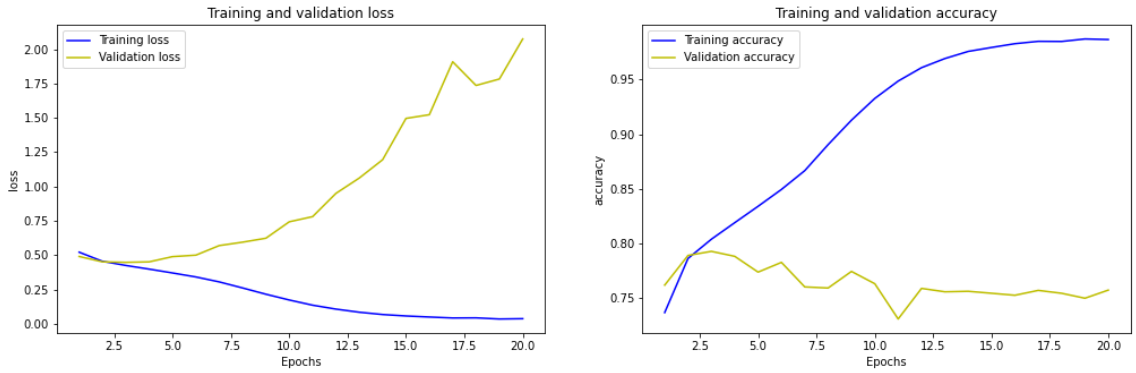


Figura 72. Gráfica del modelo 3 con optimizador 'rmsprop'

Resultados

Teniendo en cuenta el modelo presentado y los hiperparámetros elegidos se van a ir mostrando las gráficas que evidencian el resultado del entrenamiento del modelo DL número 3 desarrollado para encontrar la mejor solución al problema de clasificación de nucleosomas.

En ambos casos se puede observar cómo la línea de validación tiene picos, es decir, que

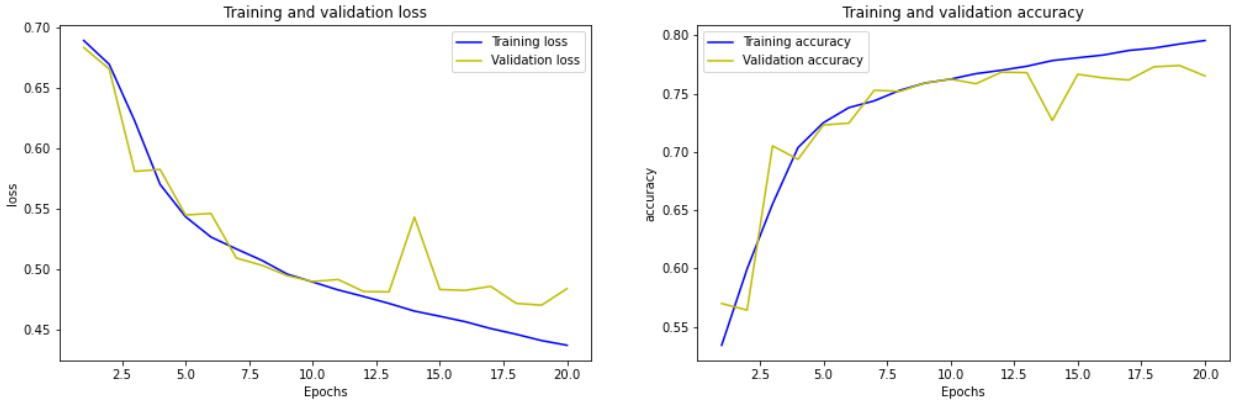


Figura 73. Gráfica resultados obtenidos modelo 3 DL con datos OHE

no sigue un entrenamiento lineal a lo largo de las épocas. Teniendo diferentes alteraciones entre buenos resultados y bajadas de la tendencia positiva. En la **Figura 77** se puede observar

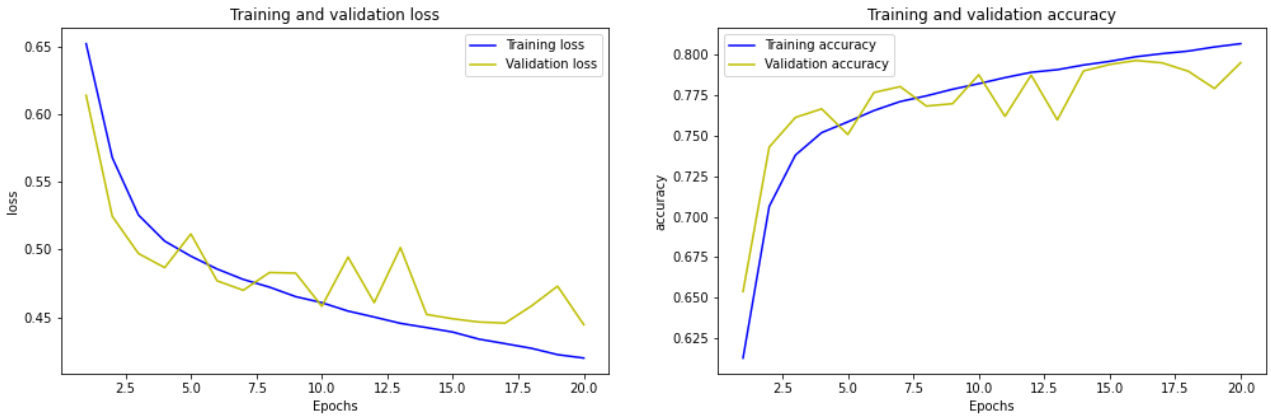


Figura 74. Gráfica resultados obtenidos modelo 3 DL con datos dinucleótidos

lo dicho hacia la época 15 que hay una bajada considerable del desempeño del modelo con un peor *accuracy* y subida de la función de pérdida. No hay una gran diferencia de resultados de un preprocesamiento a otro, rondando el 75% sin llegar a conseguir el 80%. En las matrices de confusión podemos ver como el ratio de verdaderos positivos y verdaderos negativos es similar, por lo que hay una clasificación equitativa de las categorías. **(Figura 79)**

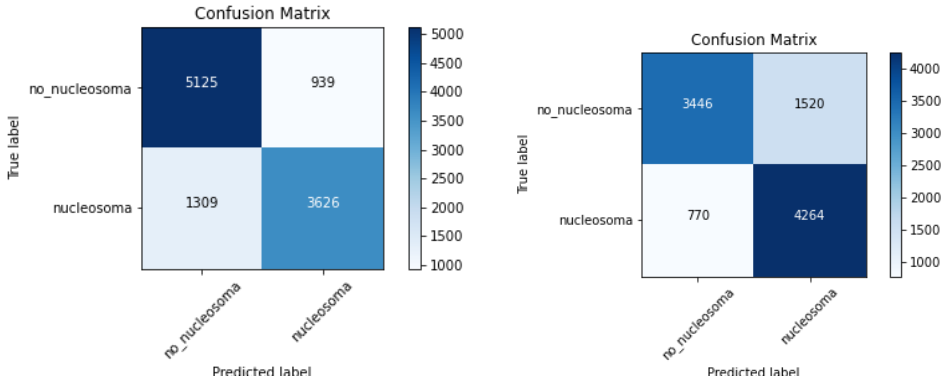


Figura 75. Matriz de confusión modelo 3 DL. [1] dinucleótidos. [2] OHE

4.6 Página web

Una vez ya probados varios modelos de ML y DL y habiendo obtenido un accuracy de 82,85% se intentó acercar un poco más el resultado conseguido de los modelos a los usuarios que no tuvieran un conocimiento de programación suficiente porque, aunque el objetivo no era, en este caso, conseguir una aplicación que estuviera disponible, siempre ayuda el hecho de hacer una pequeña aproximación para que el usuario pueda hacer pequeñas pruebas.

Para ello se optó por realizar una página web sencilla en la que se usó Python y Flask. La página web está alojada en la siguiente dirección: 37.35.239.182:8080. Una vez entramos podemos ver la página principal en la que simplemente hay un pequeño resumen y un diagrama de los resultados de los diferentes modelos probados. En la segunda pestaña "Modelo" es en la que se puede probar el modelo con la mejor accuracy de DL. Para probarlo tendríamos que haber realizado un csv que contenga en una columna tantas filas con secuencias de 150 pares de bases se quieran predecir. Existe un archivo de prueba que se puede descargar y probar. El resultado será una tabla con la probabilidad de que sea no nucleosoma, por tanto, en verde tenemos los que se predicen como secuencias que pertenecen a un nucleosoma y en rojo las que no pertenecen según el modelo. Además, hay un enlace al GitHub donde está el código de los modelos de inteligencia artificial.

5 Resultados y discusión

Conocer el posicionamiento de los nucleosomas a lo largo de la cadena de DNA puede ser una de las claves para descubrir cómo se regulan los genes. No existe un único determinante que contribuye a que un nucleosoma se posicione 'bien' en el genoma. Los más destacados son los remodeladores de cromatina, los factores de transcripción y finalmente, la cadena de DNA. En este caso se ha centrado la investigación en la influencia de esta última. Para poder conocer cómo afecta la cadena en crudo de DNA al posicionamiento se ha intentado buscar una solución con aprendizaje automático. Gracias a las técnicas de DL y ML para clasificar, se ha modelado una solución que clasifique secuencias de DNA en dos categorías: nucleosoma y los que no son.

Lo primero se han extraído conjuntos de datos de los archivos facilitados de *S. Pombe*, hongo unicelular eucariota que sirve como modelo para análisis en diversos campos de la biología y con un genoma de un tamaño relativamente pequeño comparado con otras especies para poder realizar análisis con el mismo. Se han extraído una cantidad de datos considerable que después se han sometido a métodos de preprocesamiento, ya que un modelo de ML o DL no puede alimentarse de secuencias de texto, sino que tienen que cambiar por valores numéricos. Al fin y al cabo, los modelos de inteligencia artificial optimizan una función matemática que necesita lidiar en última instancia con números. Es una de las tareas que llevaron mayor tiempo de desarrollo.

La segunda parte fue la elección de modelos con sus correspondientes hiperparámetros para conseguir la mejor combinación posible. A la hora de probar el modelo se escogió un 80% para su entrenamiento, un 10% para la validación del modelo y otro 10% para realizar predicciones y comprobaciones del modelo, así como las matrices de confusión. Se han probado diferentes algoritmos con un espacio de búsqueda de hiperparámetros diferente para cada uno de ellos. Un resumen de los resultados obtenidos se puede observar en el diagrama de la **figura 80**. Cada uno de los rectángulos del diagrama muestra el nombre el algoritmo de ML con sus hiperparámetros para el mejor caso de ese algoritmo concreto y los modelos de DL con su nombre tal como aparece en la memoria. No se han incluido todas las capas con sus diferentes hiperparámetros porque ya se razonó en el apartado correspondiente.

Se observa en el diagrama la medida de exactitud, medida que se ha escogido como contraste a lo largo de todos los modelos. Pude resultar una medida engañosa, puesto que calcula el porcentaje de resultados positivos, pudiendo existir un desequilibrio en el número de datos vinculados a las categorías que hace que mejore esta medida si se aumenta los datos de entrada de la categoría que clasifica bien. Sin embargo, en el presente trabajo están cotejadas con sus correspondientes matrices de confusión presentadas en cada uno de los modelos correspondientes para evitar confusión en la métrica. Todos los algoritmos se entrenaron con una medida balanceada o más bien balanceada de los datos objetivo, es decir, alrededor del 50-60% de datos de cada clase objetivo a fin de evitar predicciones desequilibradas de los modelos.

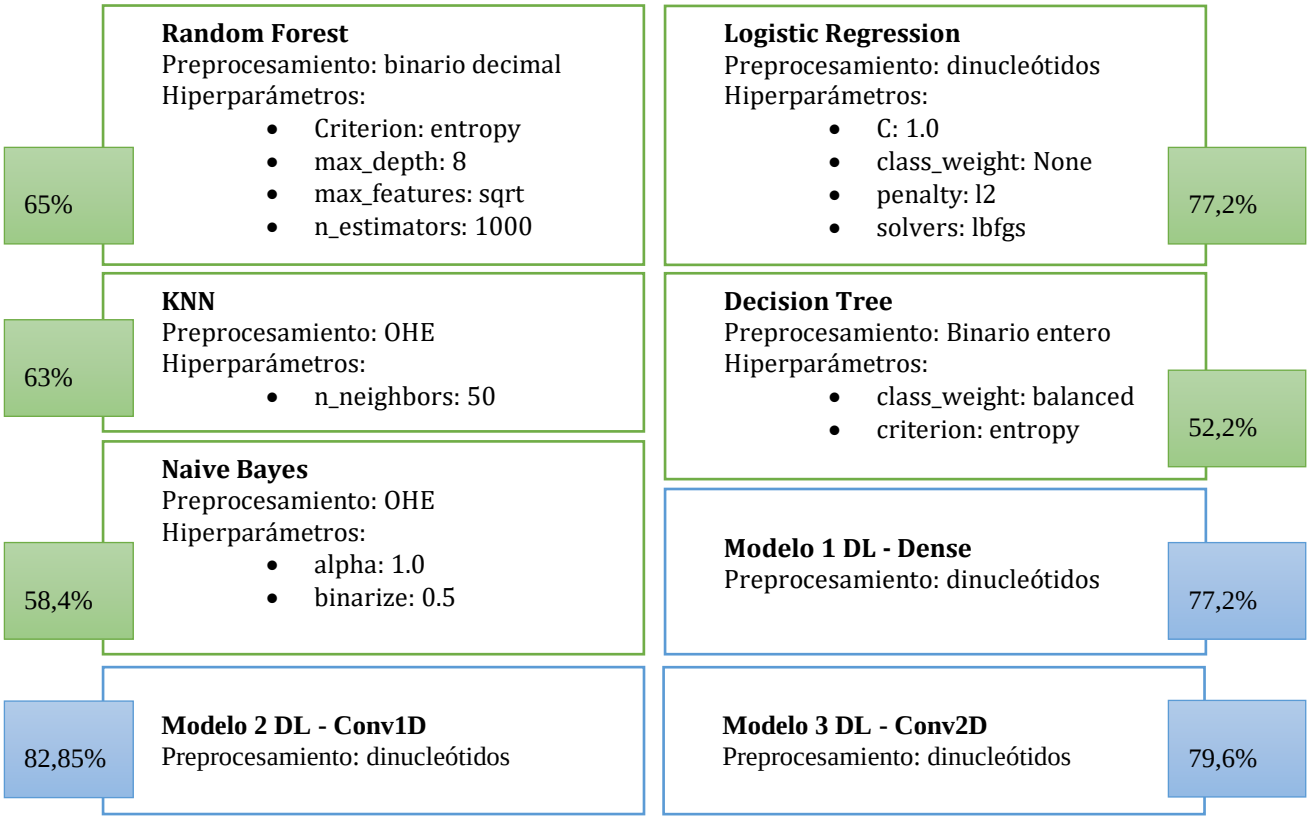


Figura 76. Diagrama de comparación modelos usados en el trabajo

Tras decidir que el mejor modelo era el de Deep Learning se realizó una pequeña web a modo de presentación en la que se puede realizar una prueba del modelo introduciendo un archivo con extensión 'csv' con las secuencias de 150 nucleótidos que se quieran predecir. Se puede acceder a través de la siguiente dirección: 37.35.239.182:8080.

En la literatura ya existían modelos de aprendizaje profundo para la identificación de nucleosomas. Un ejemplo de ello es CORENup (Amato et al., 2020), que procesa una secuencia de DNA como entrada con una representación numérica asociada y combina una red convolucional con una recurrente. Los resultados presentados en este proceso se basan en la métrica de AUC, por lo que a modo de comparar con nuestro resultado se calculó el AUC del modelo de red convolucional de 1D. Los resultados del presente trabajo con un AUC de 0,908 está algún punto por debajo del desempeño mostrado en CORENup de 0,933.

Su predecesor seguramente sea LeNup (Zhang et al., 2018), modelo de aprendizaje profundo que mezcla redes de tipo *Inception* para identificar secuencias de nucleosomas. Se entrenó este modelo con genomas de *Homo sapiens, Caenorhabditis elegans, Drosophila melanogaster y Saccharomyces cerevisiae*. El rendimiento que se produjo con LeNup fue de un *accuracy* de 0,89 para *H. sapiens,* 0,92 para *C.elegans* y 0,88 para *D.melanogaster*. Estos

datos son superiores al obtenido en nuestro caso 0,82, aún así no puede ser directamente comparable debido a que estamos hablando lo primero de genomas distintos, formas de tratar con los datos de entrada diferentes y redes con otra distribución de la arquitectura.

6 Conclusiones

El principal resultado de este trabajado ha sido el desarrollo de un modelo de aprendizaje profundo capaz de resolver con cierta solidez el problema de identificación de nucleosomas. Los resultados obtenidos de las pruebas sustentan la idea de que muchas veces menos, es más. El modelo vencedor presenta dos capas convolucionales, junto a una capa de pooling con método 'Max Pooling' y dos capas totalmente conectadas intercalando una capa de regularización ('Dropout') entre ellas. No han sido necesarias muchas capas ocultas con gran cantidad de neuronas para acercarse a la optimización de la red. Las redes neuronales profundas presentaron un mejor desempeño que ML

Las técnicas de análisis de datos se han aplicado especialmente a campos económicos y de negocios. La biología no es un área de actuación más común para el análisis de datos, pero se ha logrado una herramienta que podría relativamente útil para poder aplicar en laboratorios o líneas de investigación futuras. La informática ha demostrado poder estar al servicio de ámbitos de investigación como es la biología. Lograr entender un pequeño problema biológico y combinar las técnicas computacionales para resolver ha sido un gran reto.

El desarrollo de diversos modelos y su comparativa supone haber podido comprender qué es lo que hace que un modelo pueda tener un mejor desempeño. La ingeniería intenta buscar una solución al problema, pero a veces tiene dificultad en encontrar el motivo que ha llevado a ese resultado final. Por eso, una de los mayores limitaciones para modelar nuevos algoritmos ha sido poder comprender qué hiperparametros eran más adecuados con su correspondiente justificación y ha podido desencadenar en no adentrarse en el desarrollo de redes muy profundas complicadas de explicar. La ciencia de datos es una herramienta muy útil pero también oscura, no sabiendo qué es lo que internamente hace para mejorar los modelos matemáticamente hablando. Esto es una problemática que se presenta a la hora de exponer una solución a un problema biológico que necesita de una rigurosidad mucho mayor que la informática debido a sus implicaciones posteriores.

La potencia software es importante para entrenar modelos de IA, pero disponer de un hardware adecuado era también un reto. He aprendido a lidiar con un servidor externo a mi ordenador personal con un número mayor de núcleos para poder entrenar los modelos con sus dificultades y aprender a implementar paralelización. El modelo puede tener limitaciones en cuanto a tiempos de entrenamiento porque no se ha centrado el desarrollo en realizar un modelo óptimo en tiempo, focalizando toda la atención en el desempeño.

La conclusión de este trabajo es que la combinación técnicas de aprendizaje automático, tanto ML y DL, facilita encontrar soluciones a problemas de otra índole como es el caso de la biología reduciendo el tiempo para lograr una solución y facilitando herramientas que se podrían implementar en un área de trabajo de investigación si se consigue comprender el problema y exponer el desarrollo seguido para lograr una solución.

6.1. Trabajo futuro

Se ha dado una solución al problema tras probar un conjunto de modelos con diferentes características, pero existen varios puntos que se podrían mejorar de cara al desarrollo de nuevos modelos y mejora de este trabajo en un futuro:

- La elección de codificación ha sido en base a criterios no especialmente rigurosos biológicamente hablando. El orden en el que se han codificado los nucleótidos no sigue un patrón sustentado en experimentos científicos. Se podría investigar más en iteraciones entre nucleótidos, realizar un estudio previo de estas cadenas de 150 pb para ver si pudiera haber algún patrón para luego construir las estructuras de entrada a modelos. Además, aplicar otras técnicas como pudiera ser la matriz de doblabilidad (Salih, Tripathi, & Trifonov, 2013), en la que se observa la periodicidad de dinucleótidos cada un número fijo de pares de bases. Se propone realizar un conteo de cuántas veces se repite cada pareja de nucleótidos a lo largo de los nucleosomas y el genoma en general de S. Pombe podría ser el punto de partida de nuevos modelos. Como está se podría observar en la literatura otras aproximaciones matemáticas para predecir el posicionamiento de nucleosomas (Liu, et al., 2014) que podrían incorporarse como forma de codificar las secuencias de entrada a modelos de aprendizaje automático.

- Otro punto interesante podría ser probar el modelo elegido por su mejor desempeño en un ambiente propiamente científico, con secuencias de 150 pb que se vayan a someter a experimentos posteriores validando así de una forma más exhaustiva el modelo computacional propuesto.

- El rendimiento del aprendizaje profundo tiene como punto de inflexión la cantidad de datos con que se trabaja. Como punto de mejora se podría intentar incorporar genomas de nuevas especies para comprobar su exactitud con mayor cantidad de datos. Así también se podría hacer un estudio comparativo entre especies y posibilitar que la red tenga más datos de los que aprender patrones para evitar el 'overfitting' de los datos de entrada y perseguir una mayor generalización del modelo.

- Las herramientas informáticas tienen que poderse usar por el público al que van dirigidas con sus capacidades y limitaciones. Es de suponer que un biólogo, destinatario de esta herramienta, no sepa cómo programar en Python o extraer el código, compilarlo y utilizarlo para predecir nuevos datos. Con este trabajo se ha conseguido el modelo informático que era el reto de este trabajo, pero sin disponer de un conocimiento informático previo resulta difícil de implementar y usar. Para que sea accesible a todos los usuarios y sea usada como herramienta software habría que pensar en una aplicación con interfaz de usuario aún más potente que la página web ya facilitada.

Bibliografía

1. A, S., & K, O. (1994). *Kendall's Advanced Theory of Statistics - Volume I- Ditribution Theory.*
2. Alharbi, B. A., Alshammari, T. H., Felton, N. L., Zhurkin, V. B., & Cui, F. (2014). nuMap: A Web Platform for Accurate Prediction of Nucleosome Positioning. *Genomics Proteomics Bioinformatics*, 249-253.
3. Amato, D., Bosco, G. L., & Rizzo, R. (2020). CORENup: a combination of convolutional and recurrent deep neural networks for nucleosome positioning identification. *BMC Bioinformatics*, articulo 26.
4. Annunziato, A. T. (2008). DNA Packaging: Nucleosomes and Chromatin. *Nature Education*, 2.
5. Antequera, F. (Mayo de 2017). Las huellas nucleosómicas, un nuevo nivel de información en el ADN. *Investigación y Ciencia*(488), 3.
6. Arya, G., Maitra, A., & Grigoryev, S. A. (2012). A Structural Perspective on the Where, How, Why, and What of Nucleosome Positioning. *Full Terms & Conditions of access and use can be found at https://www.tandfonline.com/action/journalInformation?journalCode=tbsd20 Journal of Biomolecular Structure and Dynamics*, 3-5.
7. Bilofsky, H. S., Burks, C., Fickett, J. W., Goad, W. B., Lewitter, F. I., & Rindone, W. P. (1986). The GenBank genetic sequence databank. *Nucleic Acids Res. 14.*
8. Chen, W., Feng, P., Ding, H., Lin, H., & Chou, K.-C. (Marzo de 2016). Using deformation energy to analyze nucleosome positioning in genomes. *Genomics, 107*, 69-75.
9. Chollet, F. (2018). *Deep Learning with Python.* New York: Manning Publications.
10. Cost, S., & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Department of Computer Science, Johns Hopkins University, Baltimor*, 57-78.

11. *Developers Google*. (2021). Retrieved from Clasificación: Exactitud: https://developers.google.com/machine-learning/crash-course/classification/accuracy
12. Dietterich, T. (2 de May de 2017). Machine learning challenges and impact: an interview with Thomas Dietterich. (Z.-H. Zhou, Entrevistador)
13. Fawcett, T. (2004). ROC Graphs: Notes and Practical Considerations for Researchers. 1-18.
14. Freedman, D. A. (2009). *Statistical Models: Theory and Practice.* Cambridge University Press.
15. González Moreno, S. (2017). *Contribución de la secuencia del DNA al posicionamiento de los nucleosomas.* Salamanca: Tesis Doctoral.
16. Gonzalez, S., Garcia, A., Vazquez , E., Serrano , R., Sanchez, M., Quintales, L., & Antequera, F. (2016). Nucleosomal signatures impose nucleosome positioning in coding and noncoding sequences in the genome. *Genome Research*, 12.
17. Goodfellow, I., Bengio, Y., & Courville, A. (2017). *Deep Learning.* Cambridge, MA: MIT Press.
18. Han, H., & Liu, W. (2019). The coming era of artificial intelligence in biological data science. *BMC Bioinformatics, 20*(712).
19. Intelligence, T. h. (28 de Agosto de 2017). *Harvard University, The Graduate School of Arts and Sciences.* Obtenido de Blog, Special Edition on Artificial Intelligence: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/
20. John, G. H., & Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. *Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence*, 338-345.
21. K.M, T. (2011). Confusion Matrix. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. *Springer*.
22. Kaplan, N., Moore, I. K., Fondufe-Mittendorf, Y., Gossett, A. J., Tillo, D., Field, Y., . . . Segal, E. (2008). The DNA-encoded nucleosome organization of a eukaryotic genome. *Nature*, 362-366.
23. Kotsiantis, S., D.Kanellopoulos, & Pintelas, P. (2006). Data Preprocessing for Supervised Learning. *International Journal of Computer Science, 1*(1).

24. Le Cun, Y. (s.f.). *Generalization and Network Design Strategies.*
25. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, 521:436.
26. Liu, H., Zhang, R., Xiong, W., Guan, J., Zhuang, Z., & Zhou, S. (2014). A comparative evaluation on prediction methods of nucleosome positioning. *Briefings in Bioinformatics, 15*(6), 1014-1027.
27. Luscombe, N., Greenbaum, D., & Gerstein, M. (2001). What is bioinformatics? An introduction and overview. *Yearbook of Medical Informatics* , 1-3.
28. M.Ayer, V., Miguez, S., & H.Toby, B. (2014). Why scientists should learn to program in Python. *Cambridge University Press*.
29. Manekar, S. C., & Sathe, S. R. (December de 2018). A benchmark study of k-mer counting methods for high-throughput sequencing. *GigaScience, 7*(12).
30. Mc Ginty, R., & Tan, S. (2014). Nucleosome Structure and Function. *American Chemical Society*, 19.
31. McCallum, A., & Nigam, K. (1998). A comparison of event models for Naive Bayes text classification. *AAAI-98 workshop on learning for text categorization*.
32. Mount, D. W. (2004). *Bionformatics: Sequence and Genome Analysis.* Spring Harbor Press.
33. Python. (s.f.). *Wikipedia.* Obtenido de Python - Wikipedia: https://es.wikipedia.org/wiki/Python
34. Radwan, A., Akmal Younis, Luykx, P., & Khuri, S. (2008). Prediction and analysis of nucleosome exclusion regions in the human genom. *BMC Genomics*.
35. Rich, E. (1985). Artificial Intelligence and the Humanities. *Natural Language Processing*, 117-122.
36. Rodríguez Montequín, M. T., Álvarez Cabal, J. V., Mesa Fernández, J. M., & González Valdés, A. (2016). *METODOLOGÍAS PARA LA REALIZACIÓN DE PROYECTOS DE DATA MINING.* Oviedo: Correspondencia Universidad de Oviedo Área de Matemática Aplicada.
37. Rokach, L., & Maimon, O. (2010). Classification Trees. En L. Rokach, & O. Maimon, *Data Mining and knowledge discovery handbook* (págs. 149-175). Longon : Springer.

38. Salih, B., Tripathi, V., & Trifonov, E. N. (2013). *Visible periodicity of strong nucleosome DNA sequences.* London: Journal of Biomolecular Structure and Dynamics.
39. Santamaría, L. A., H, S. Z., T, I. H., Somodevilla, M. J., & L., M. R. (s.f.). DNA Sequence Recognition using Image Representation.
40. Scherer, D., Müller, A., & and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. *International Conference on Artificial Neural Networks*, 92-100.
41. Segal, E., Fondufe-Mittendorf, Y., Chen, L., Thastrom, A., Field, Y., Moore, I. K., & Widom, J.-P. Z. (Agosto de 2006). A genomic code for nucleosome positioning. *Nature, 442*, 7.
42. Segal, K. S. (Marzo de 2013). Determinants of nucleosome positioning. *nature structural & molecular biology, 20*(3), 7.
43. Sewak, M., Sahay, S. K., & Rathore, H. (2020). An Overview of Deep Learning Architecture of Deep Neural Networks and Autoencoders. *Journal of Computational and Theoretical Nanoscience*, 15.
44. Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms.* Cambridge: Cambridge University Press.
45. Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. *Cambridge University Press*, 19-21.
46. Stuart, A., & K.Ord. (1994). *Kendall's Advanced Theory Statistics: Volume I - Distribution Theory.* Edward Arnold Publishers.
47. Suzuki, K. (2011). Artificial Neural Networks - Methodological Advances and Biomedical Applications. *Artificial Neural Networks - Methodological Advances and Biomedical Applications* (pág. 374). Rijeka: InTech.
48. Tolles, J., & Meurer, W. J. (2016). Logistic Regression Relating Patient Characteristics to Outcomes. *JAMA*, 533-534.
49. Tompitak, M., Vaillant, C., & Schiessel, H. (2017). Genomes of Multicellular Organisms Have Evolved to Attract Nucleosomes to Promoter Regions. *Biopyhs J.*, 505-511.

50. V Wood. (2002). The genome sequence of Schizosaccharomyces pombe. *Nature*, 11.
51. Velasco, L. (26 de Abril de 2020). *Medium.* Obtenido de Optimizadores en redes neuronales profundas: un enfoque práctico: https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5
52. Weerts, H. J., Müller, A. C., & Vanschoren, J. (15 julio 2020). Importance of Tuning Hyperparameters of Machine Learning Algorithms.
53. Yang, A., Zhang, W., Wang, J., Yang, K., Han, Y., & Zhang, L. (2020). Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA. *Front. Bioeng. Biotechnol*.
54. Yen, K., Vinayachandran, V., Batta, K., Koerber, R. T., & Pugh, B. F. (2012). Genome-wide nucleosome specificity and directionality of chromatin remodelers . *PubMed*.
55. Zhang, J., Peng, W., & Wang, L. (Mayo de 2018). LeNup: learning nucleosome positioning from DNA sequences with improved convolutional neural network. *Bioinformatics, 34*, 1705-1712.