

TRABAJO FIN DE GRADO



UCAM

UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITECNICA

Departamento de Ciencias Politécnicas
Grado en Ingeniería Informática

Herramienta para el estudio del alineamiento de
secuencias de ADN mediante DeepLearning

Autor:

Juan Francisco Illán Sánchez

Directores:

Francisco Arcas Túnez

Jesús Soto Espinosa

Murcia, 17 de Mayo de 2021

TRABAJO FIN DE GRADO



UCAM

UNIVERSIDAD CATÓLICA
DE MURCIA

ESCUELA UNIVERSITARIA POLITECNICA

Departamento de Ciencias Politécnicas
Grado en Ingeniería Informática

Herramienta para el estudio del alineamiento de
secuencias de ADN mediante DeepLearning

Autor:

Juan Francisco Illán Sánchez

Directores:

Francisco Arcas Túnez
Jesús Soto Espinosa

Murcia, 17 de Mayo de 2021

AGRADECIMIENTOS

A mi mujer e hijos.

A mis padres y hermanos por la educación básica en los valores de la vida.

A mis tantos profesores, compañeros de estudios, compañeros profesionales y amigos personales.

A todos ellos, por ser partícipes, de las experiencias, la actitud y los objetivos de la vida.

ÍNDICE

1	Introducción	21
1.1	Motivación	21
1.2	Definición.....	21
1.3	Objetivos propuestos.....	22
2	Estado del arte.....	25
3	Estudio de viabilidad	33
3.1	Alcance del proyecto	33
3.2	Estudio de las alternativas y elección de solución.....	33
3.3	Análisis funcional de requisitos	35
4	Metodología	37
5	Tecnologías y herramientas utilizadas en el proyecto	39
6	Estimación de recursos y planificación	41
7	Desarrollo del proyecto.....	43
7.1	Análisis y diseño inicial del sistema.....	43
7.1.1	Base de datos genómica	43
7.1.2	Definición del sistema de alineamiento.....	44
7.1.3	Interfaz de usuario	51
7.1.4	Arquitecturas de deep learning	52
7.2	Configuración del entorno	55
7.3	Implementación	56
7.3.1	PRIMER SPRINT: Creación inicial del proyecto	57
7.3.2	SEGUNDO SPRINT: Alineamiento heurístico.....	59
7.3.3	TERCER SPRINT: Sistema aprendizaje Clasificador Bayes	70
7.3.4	CUARTO SPRINT: Sistema aprendizaje LSTM.....	78
8	Conclusiones	87
8.1	Objetivos alcanzados	87

8.2	Conclusiones del trabajo y personales	87
8.3	Vías futuras	88
9	Referencias bibliográficas	91
10	Anexos.....	95
10.1	Manual de instalación.	95
10.2	Manual de uso.....	97
10.2.1	Alineamiento.....	98
10.2.2	Identificación de proteína	100
10.2.3	Identificación de patógenos.....	102

ÍNDICE DE FIGURAS

Ilustración 1 – Relación entre las distintas áreas del conocimiento: medicina, genética y tecnologías de la información. Imagen tomada de: (Arias, Bahón, & Rodríguez, 2006).....	21
Ilustración 2 - Ácido desoxirribonucleico. Imagen tomada de: www.genome.gov (ADN-acido-Desoxirribonucleico).....	25
Ilustración 3 - Transcripción del ARNm en cadenas de aminoácidos (proteínas). Imagen tomada del National Institutes of Cancer. National Institutes of Health. (NIH)	26
Ilustración 4 - La estructura primaria de una proteína es simplemente una cadena en orden de aminoácidos. Imagen tomada de los documentos didácticos de los profesores: (Raisman & Gonzalez)	26
Ilustración 5 - El código genético es el conjunto de reglas utilizadas para traducir la secuencia de nucleótidos del ARNm a una secuencia de proteína empleado durante el proceso de traducción. Imagen tomada del artículo” 10 Conceptos básicos para introducirte en el mundo de la Genética” en genotipia.com (Garrigues, 2017)	27
Ilustración 6 - Formato FASTA es un formato de fichero informático basado en texto, utilizado para representar secuencias bien de ácidos nucleídos, bien de péptido, y en el que los pares de bases o los aminoácidos se representan usando códigos de una única letra.....	29
Ilustración 7 - Modos del algoritmo BLAST. Imagen tomada del portal de BLAST (NCBI)	30
Ilustración 8 - Ejecución de diferentes algoritmos para el alineamiento de secuencias. Imagen tomada de Chapter 2.3 Searching Sequence Databases; FASTA, BLAST, Computer Science Department. Columbia University (Prof. Yechiam Yemini, 2007)	30
Ilustración 9 - Ejemplo de resultados de alineamiento obtenidos tras ejecutar algoritmo de Smith-Waterman y BLAST.....	31
Ilustración 10 - Esquema de la metodología Agile/SCRUM. Imagen tomada de: (SCRUM- ROLES, ARTIFACTS AND CEREMONIES, 2019)	37
Ilustración 11 – Fichero de datos en formato FASTA utilizado durante el desarrollo del proyecto en las fase tempranas.	43

Ilustración 12 - Ejecución de diferentes algoritmos para el alineamiento de secuencias. Imagen tomada de Chapter 2.3 Searching Sequence Databases; FASTA, BLAST, Computer Science Department. Columbia University (Prof. Yechiam Yemini, 2007).....	44
Ilustración 13 - Ejemplo de resultados de alineamiento obtenidos tras ejecutar algoritmo de Smith-Waterman y BLAST.....	44
Ilustración 14 - Kmers para la secuencia "GTAGAGCTGT" Imagen tomada de: "Bioinformatics 1: K-mer Counting" (Gunavaran Brihadiswaran, 2020).....	46
Ilustración 15 - Combinaciones en kmers para localizar los puntos prometedores de extensión del alineamiento.....	47
Ilustración 16 - Ejemplo animado paso a paso de la ejecución de la matriz y recomponer la solución. Imagen tomada de "Sequence Comparison: Theory and Methods" (Kun-Mao Chao & Louxin Zhang, 2008).....	48
Ilustración 17 - Plot del alineamiento global de 2 secuencias.....	49
Ilustración 18 - Plot del alineamiento local de 2 zonas secuencias	49
Ilustración 19 - Esquema cliente servidor HTTP.....	51
Ilustración 20 - Esquema de la arquitectura de una red neuronal convolucional CNN. Imagen tomada del post "Red neuronal convolucional CNN" de: (Diego Calvo , 2017).....	54
Ilustración 21 - Arranque de servidor web	57
Ilustración 22 - Aspecto de la interfaz de usuario ejecutada sobre el navegador.	58
Ilustración 23 - Contenido de la base de datos sobre ficheros en codificación FASTA.....	59
Ilustración 24 - Pseudocódigo del algoritmo de alineamiento propuesto utilizando el algoritmo de Smith and Waterman.	61
Ilustración 25 - Función read_data() para la lectura de las cadenas del sistema.....	62
Ilustración 26 - Acceso a las columnas de una línea leída desde una fuente .csv delimitada por ";"	62
Ilustración 27 - Función para la extracción en k-mers de una cadena... ..	62
Ilustración 28 - Función para el alineamiento sin huecos entre 2 k-mers.	63

Ilustración 29 - Esquema de la implementación de la función para la localización de las semillas HSP.....	63
Ilustración 30 - Ejemplo animado paso a paso de la ejecución de la matriz y recomponer la solución. Imagen tomada de “Sequence Comparison: Theory and Methods” (Kun-Mao Chao & Louxin Zhang, 2008).....	64
Ilustración 31 - Esquema de la implementación del algoritmo Smith-Waterman.....	64
Ilustración 32 – Heurística del algoritmo de alineamiento. Acotar recorridos sobre la matriz Smith – Waterman.	65
Ilustración 33 - Reconstrucción de la mejor solución de alineamiento encontrado sobre una cadena del sistema.....	65
Ilustración 34 – Presentación de resultados en el proyecto - Bioinformatics Website.....	66
Ilustración 35 - Grafica de resultados que muestra el sistema desarrollado, mostrando de los tiempos de ejecución de una consulta, mostrando una línea el tiempo total en la búsqueda de las semillas HSP y en otra el tiempo de la extensión con huecos de todas esas semillas calculadas.	67
Ilustración 36 - Base de datos de cadenas de nucleótidos y su correlación proteica.	70
Ilustración 37 - Cada cadena de la base de datos se descompone en la combinaciones de los kmers que da lugar	72
Ilustración 38 - La imagen muestra la clasificación de las cadenas en base de datos.....	72
Ilustración 39 – Conunt Vectoricer for sequences of 4 words-Kmers	73
Ilustración 40 - Desarrollo de la bolsa de palabras de 3 palabras de 6 nucleótidos.	74
Ilustración 41 - Clasificador Bayes de sklearn.naive_bayes.MultinomialNB.....	75
Ilustración 42 - Fragmento de código para la transformación de la cadena de consulta al array de entrada al clasificador y su predicción.	76
Ilustración 43 - Arquitectura de una red neuronal LSTM. Imagen tomada de: “Understanding LSTM Network”. (Christopher, 2015)	78

Ilustración 44 - Base de datos para el entrenamiento e identificación del sistema neuronal del patógeno Coronavirus SARS-CoV-2 Severe acute respiratory syndrome coronavirus Wuhan-Hu-1.....	79
Ilustración 45 - Uso del Tokenizer para generar el diccionario de palabras	80
Ilustración 46 - Identificadores de palabra en el diccionario generado por el Tokenizer.....	81
Ilustración 47 - Formato del array de entrenamiento una vez convertido a array de palabras y aplicado relleno para unificar longitudes.....	82
Ilustración 48 - Codificación de la red neuronal LSTM.....	82
Ilustración 49 - Esquema de la arquitectura de la red neuronal LSTM para identificación de un patógeno.....	83
Ilustración 50 - Arquitectura de la red neuronal basada en LSTM para la identificación de cadenas con un patógeno conocido.	84
Ilustración 51 - Código para la consulta de una cadena al clasificador LSTM.....	84
Ilustración 52 – Proceso de instalación de librerías	96
Ilustración 53 - Aspecto del main de sistema Bioinformatics Website ...	97

ÍNDICE DE ECUACIONES

Ecuación 1 - Ecuación de estimación de costes por tres puntos. (Institute, PMBOK - A Guide to the Project Management Body of Knowledge (version 6), 2017)	41
Ecuación 2 - Teorema de Bayes del cálculo de probabilidad condicionada.	70
Ecuación 3 - Numero de palabras máximo del diccionario para un lenguaje de 4 letras y palabras de 6 caracteres.	80

ÍNDICE DE TABLAS

Tabla 1 - Tabla de requisitos funcionales del sistema.	35
Tabla 2 - Detalles del análisis de requisitos del requisito funcional RF_1: Aplicación interfaz web.....	35
Tabla 3 - Tabla de tareas pila de producto del sistema.	38
Tabla 4 - Tabla de product baklog o pipa de producto que describe la lista de tareas a realizar a lo largo del proyecto, con información en relación a la etapa y plazos estimados de ejecución.....	42
Tabla 5 - Matriz de retrospectiva sprint 1.....	59
Tabla 6 - Matriz de retrospectiva sprint 2.....	69
Tabla 7 - Clasificación de cadenas con la proteínas que está relacionada. Dataset obtenido del artículo “Working with DNA sequence data for ML part 2” en kaggle.com (Nelson, 2019).....	71
Tabla 8- Matriz de confusión del sistema Clasificador Naives Bayes	76
Tabla 9 - Matriz de retrospectiva sprint 3.....	78
Tabla 10 - Resultados de precisión y clasificación del modelo de red neuronal LSTM implementado.	85
Tabla 11 - Matriz de retrospectiva sprint 4.....	86

RESUMEN

En bioinformática, el alineamiento e identificación de mensajes biológicos en secuencias es un problema con importantes costos computacionales. Este proyecto desarrolla la implementación de un sistema de laboratorio para el alineamiento de secuencias genómicas y la identificación biológica sobre una secuencia de consulta. Para el desarrollo de estos objetivos, se han aplicado mecanismos tradicionales basados en algorítmica y heurística junto a técnicas en el terreno de la inteligencia artificial como redes neuronales bajo patrones de deep learning.

La primera parte del proyecto, el alineamiento de secuencias de ADN, desarrolla una interfaz web en Python y la implementación de un algoritmo heurístico de alineamiento de secuencias con huecos basado en los algoritmos de Smith-Waterman, FASTA o BLAST, que a partir de una cadena de entrada, nos ofrece los mejores alineamientos contra las secuencias en base de datos. Identificar estos alineamiento sirve en bioinformática para detectar posibles relaciones funcionales, filogénicas o clínicas ente organismos, individuos,...

La segunda parte ha desarrollado la identificación de mensajes biológicos de cadena de consulta de ADN, como son: "la participación en la síntesis de una proteína" o "la clasificación como un patógeno conocido". Estos objetivos se han desarrollado utilizando técnicas de inteligencia artificial basadas en redes neuronales y técnicas de deep learning como ha sido un clasificador multinomial Bayes y una red neuronal LSTM (Long short-term memory).

Los resultados obtenidos en la utilización de técnicas de identificación y clasificación mediante redes neuronales, destacan por una enorme eficiencia y flexibilidad a diferentes objetivos, en comparación con técnicas algorítmicas heurísticas utilizadas. Desde una perspectiva biológica, estos resultados deben ser valorados en el grado de utilidad y eficiencia.

Palabras clave: *Bioinformática, alineamiento de secuencias, algoritmos heurísticos, redes neuronales, deep learning, clasificador Bayes, identificación LSTM, Python.*

ABSTRACT

In bioinformatics, the alignment and identification of biological messages in sequences is a problem with significant computational costs. This project develops the implementation of a laboratory system for the alignment of genomic sequences and the biological identification on a query sequence. For the development of these objectives, traditional mechanisms based on algorithmic and heuristics have been applied together with techniques in the field of artificial intelligence such as neural networks under deep learning patterns.

The first part of the project, DNA sequence alignment, develops a web interface in Python and the implementation of a heuristic algorithm for sequence alignment with gaps based on the Smith-Waterman, FASTA or BLAST algorithms, which from an input chain, it offers us the best alignments against the sequences in the database. Identifying these alignments is useful in bioinformatics to detect possible functional, phylogenic or clinical relationships between organisms, individuals,...

The second part has developed the identification of biological messages of the DNA query chain, such as: "participation in the synthesis of a protein" or "classification as a known pathogen". These objectives have been developed using artificial intelligence techniques based on neural networks and deep learning techniques such as a multinomial Bayes classifier and an LSTM (Long short-term memory) neural networks.

The results obtained in the use of identification and classification techniques through neural networks, stand out for their enormous efficiency and flexibility to different objectives, compared to the algorithmic heuristic techniques used. From a biological perspective, these results must be valued in the degree of utility and efficiency.

Key words: *Bioinformatics, sequence alignment, heuristic algorithms, neural networks, deep learning, Bayes classifier, LSTM identification, Python.*

1 INTRODUCCIÓN

1.1 Motivación

La innovación en bioinformática aún tiene recorrido hasta una meseta de alta productividad. Las técnicas informáticas como deep learning abren un camino prometedor en la identificación y búsqueda de mensajes biológicos en grandes cadenas y bases de datos genómicas para la búsqueda de patrones o interrelaciones funcionales interesantes para el área del conocimiento.

La motivación del proyecto se identifica por un lado poner de manifiesto la complejidad natural del análisis de alineamiento de secuencias y por otro la evaluación y soluciones de nuevas estrategias informáticas basadas en deep learning para la resolución de un problema de alineamiento de secuencias, valorando diferentes alternativas o posibilidades valorando el desempeño y eficiencia de las distintas técnicas desarrolladas.

En la siguiente ilustración podemos la relación entre diferentes disciplinas medicina, genética e informática y disciplinas concretas que surgen en las zonas de convergencia.

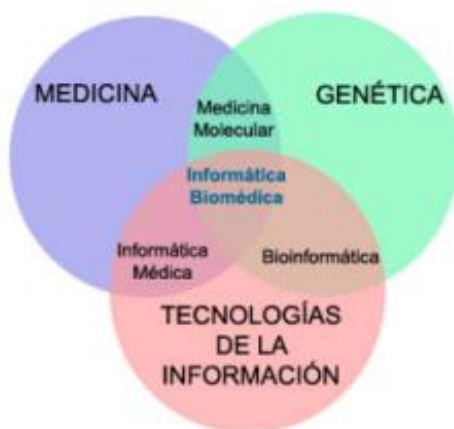


Ilustración 1 – Relación entre las distintas áreas del conocimiento: medicina, genética y tecnologías de la información. Imagen tomada de: (Arias, Bahón, & Rodríguez, 2006)

1.2 Definición

Este proyecto desarrolla la implementación de una suite o sistema de laboratorio para el alineamiento de secuencias, basado por un lado en un

mecanismo heurístico computado sobre la matriz definida por el algoritmo de Smith-Waterman, y por otro lado en la aplicación de técnicas de deep learning para la identificación de significado biológico en cadenas genómicas.

El sistema ofrecerá dada una cadena de entrada información de interrelación con otras cadenas, identificar qué tipo de secuencia pertenece o que codifica a nivel biológico.

Desde la propia definición se canaliza a desarrollarse el proyecto bajo una interfaz web. A través de una interfaz de usuario, se podrán ajustar parámetros y la cadena de consulta contra el sistema para la utilización de la suite desarrollada.

El sistema desarrollara el alineamiento de una cadena de consulta introducida por el usuario contra el sistema para la que este, computando contra el resto de cadenas en la base de datos, devolver las zonas de alto alineamiento o identificación de mensajes biológicos.

1.3 Objetivos propuestos

El objetivo principal del proyecto es la perspectiva del estudio técnico e implementación de un sistema de alineamiento de secuencias genómicas heurístico y comparativamente la implementación de un identificador de patrones o marcadores biológicos basado en arquitecturas de deep learning.

Para este objetivo se identifican los siguientes objetivos secundarios:

1. Se pretende dar utilizando diferentes estrategias computacionales, una orientación práctica, útil y funcional a las problemáticas del alineamiento de secuencias, de forma que sea interpretable y aporte el valor multidisciplinar del proyecto a la hora de identificar el nivel de similitud o clasificación de una

cadena de pregunta y cientos de patrones biológicos de un gran sistema de datos genómicos.

2. Se utilizará una implementación heurística basada en algoritmos de alineamiento de secuencias sobre la matriz de Smith-Watterman para poder realizar una implementación computable eficiente en el alineamiento y cálculo de la similitud de una cadena contra otra de la base de datos del sistema.
3. Por otro lado se realizaran implementaciones de clasificadores de basados en deep learning entrenados para identificar patrones, marcadores, clasificación o significados biológicos.
4. Se implementara el proyecto sobre una interfaz web emulando ser capaz de ser utilizado en un entorno de laboratorio para consultar similitudes o identificación de marcadores biológicos a una cadena de entrada, denominada también a lo largo del proyecto como query o consulta.

2 ESTADO DEL ARTE

El análisis computacional de secuencias biológicas de ADN o proteínas se basa inicialmente en la codificación de estas en datos digitalizados con los que poder trabajar computacionalmente. Tradicionalmente se han utilizado secuencias de caracteres para su representación y computación.

Podemos ver en la siguiente imagen una representación de la ubicación y forma del ADN en uno de los cromosomas del núcleo celular:



Ilustración 2 - Ácido desoxirribonucleico.
Imagen tomada de: www.genome.gov (ADN-acido-Desoxirribonucleico)

Los sistemas de secuencias de nucleótidos tiene 4 tipos de bases nitrogenadas para ADN (Adenina, Timina, Citosina, Guanina) y otras 4 para las cadenas de ARN (Adenina, Uracilo, Citosina y Guanina), cambiando Timina por Uracilo. (Garrigues, 2017).

La siguiente ilustración representa la sintonización de proteínas, es un proceso de traducción a partir de una cadena de ARN mensajero (ARNm) a una cadena de aminoácidos que dará lugar a la proteína resultado. Esta traducción es realizada en los ribosomas celulares.

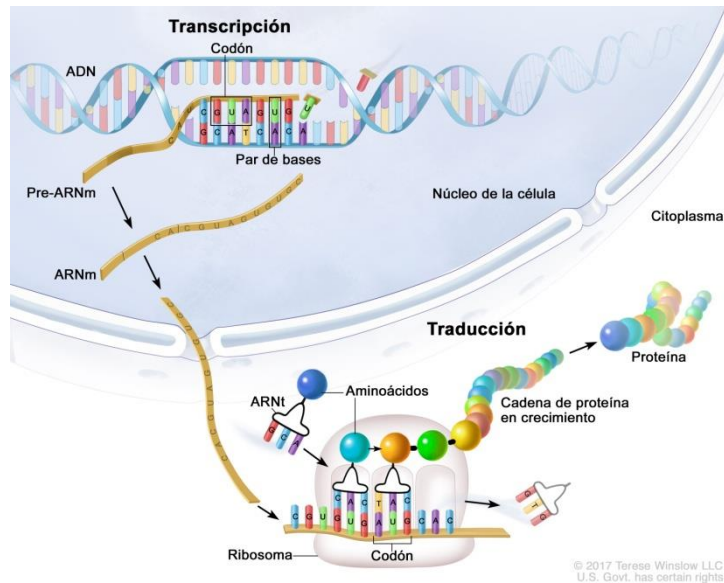


Ilustración 3 - Transcripción del ARNm en cadenas de aminoácidos (proteínas). Imagen tomada del National Institutes of Cancer. National Institutes of Health. (NIH)

Los sistemas proteicos se componen de 20 aminoácidos que conforman las proteínas. Aunque fuera del ámbito de este proyecto, sabemos que estas proteínas participan en la función estructural, de señalización, reguladora a nivel celular y del macro sistema biológico. Puede obtenerse estas conclusiones generales entre otras fuentes en el trabajo “Biología molecular del gen” (James D. Watson, 2006).

Observamos en la siguiente ilustración la estructura de una proteína formada por una cadena de aminoácidos.

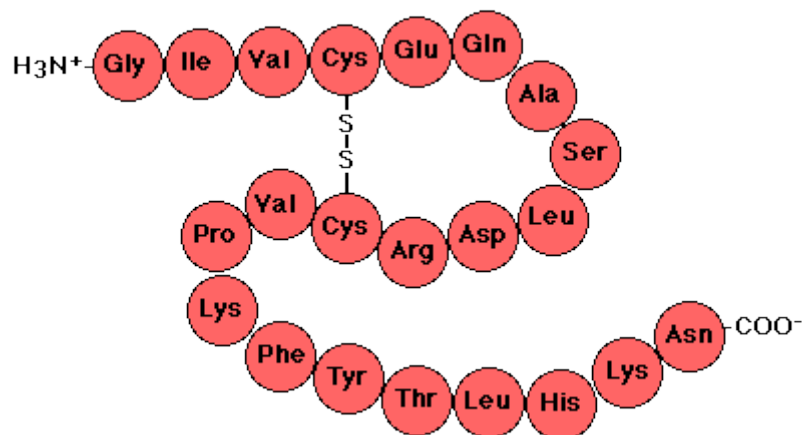


Ilustración 4 - La estructura primaria de una proteína es simplemente una cadena en orden de aminoácidos. Imagen tomada de los documentos didácticos de los profesores: (Raisman & Gonzalez)

La relación entre ADN y proteínas se basa en que la sintonización de proteínas que producida en los ribosomas utiliza como receta, cadenas de ARNm derivadas del ADN del núcleo celular, y se basa en esas cadenas para ir sintetizando partiendo del mensaje genómico la cadena de aminoácidos que dará lugar a las distintas proteínas que intervienen en la vida para el comportamiento celular del sistema biológico. El ADN codifica las proteínas en regiones denominadas genes, y bien interpretadas por la maquinaria celular, y acotando a las secciones intrón-exón que intervienen existe una relación entre cada 3 nucleótidos con un aminoácido en la sintonización de proteínas. Esta relación entre ADN y proteínas es conocida como “Dogma central de la biología molecular”. Fue expresado por Francis Crick en el “Central Dogma of Molecular Biology” en 1958 y publicado posteriormente en la revista Nature: (FRANCIS CRICK, 1970)

La siguiente ilustración representa una herramienta para la traducción de cada codón genético al aminoácido correspondiente en el proceso de sintonización proteico. Por ejemplo podemos observar como el codón “AAA” produce el aminoácido “Lys(K)”, el codón “GCA” sintetiza el aminoácido “Ala(A)”, o el codón “ACG” el aminoácido “Thr(T)”

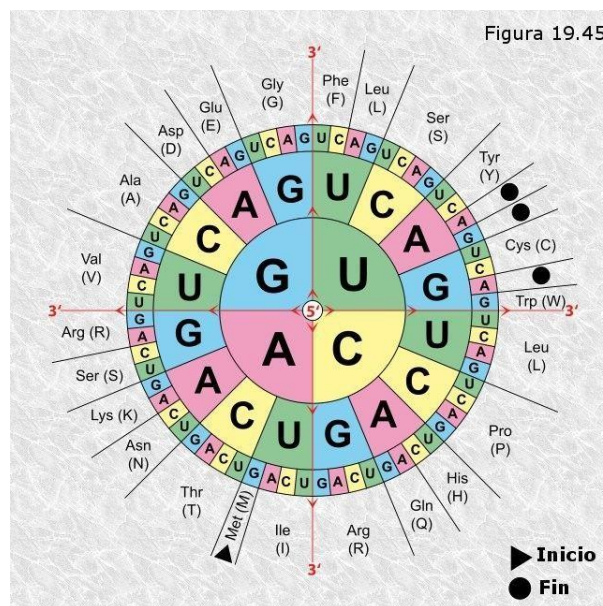


Ilustración 5 - El código genético es el conjunto de reglas utilizadas para traducir la secuencia de nucleótidos del ARNm a una secuencia de proteína empleado durante el proceso de traducción. Imagen tomada del artículo “10 Conceptos básicos para introducirte en el mundo de la Genética” en genotipia.com (Garrigues, 2017)

En este punto entramos al alineamiento en sí de las cadenas, con el objetivo de localizar similitudes que podría significar patrones o marcadores biológicos. Las ideas fundamentales en bioinformática para la realización de este alineamiento puede extraerse entre otros trabajos, de la página divulgativa de “Introducción a la bioinformática” del Bioinformatics & Genomics Group del COMAV Institute la Universidad Politécnica de Valencia: (Introducción a la bioinformática).

El enfoque tradicional de estas comparaciones se basa en dos algoritmos, el de **Needleman-Wunsch** (propuesto por primera vez en 1970 para alineamiento globales) y propuesto más tarde el algoritmo de **Smith-Waterman** (propuesto por Temple Smith y Michael Waterman en 1981 para alineamiento locales). Ambos basados en programación dinámica computadas sobre una matriz. Estas técnicas permiten alinear 2 cadenas considerando huecos en el alineamiento. Localizan entre dos cadenas el alineamiento optimo con huecos.

A finales de los años 80, se desarrolló **FASTA** que fue un paquete de basado en el algoritmo Smith-Waterman. Inicialmente descrito para el alineamiento de secuencias en cadenas proteicas. Fue desarrollado por David J. Lipman y William R. Pearson en 1985 y ampliado en 1988 añadiendo la habilidad de hacer búsquedas entre cadenas ADN:ADN también llamado como FASTN, y proteína:ADN llamado FASTP. Su legado principal es el Formato FASTA el cual es ahora muy popular en Bioinformática por su sencillez y buen funcionamiento representando cadenas proteicas y genómicas como cadenas de caracteres ascii. Introduce técnicas heurísticas para acotar los huecos y el cálculo de la matriz del algoritmo Smith-Waterman, lo que le otorga una mejora en eficiencia a costa de ceder la capacidad de encontrar el alineamiento óptimo entre dos cadenas.

Podemos observar en la plataforma BLAST los 4 modos fundamentales de ejecución.

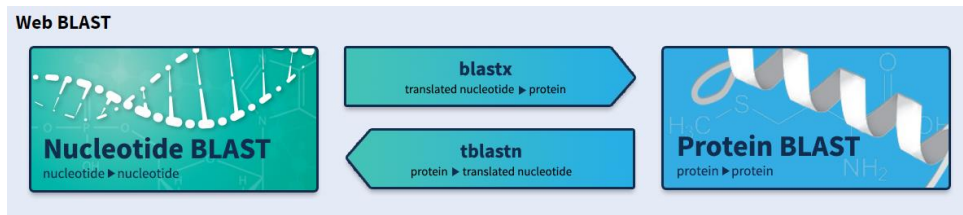


Ilustración 7 - Modos del algoritmo BLAST. Imagen tomada del portal de BLAST (NCBI)

BLAST dispone de diferentes implementaciones que resuelven el alineamiento de cadenas de nucleótidos, de proteínas y de relaciones entre cadenas de nucleótidos y cadenas de proteínas y viceversa.

Comparando las diferentes alternativas observamos como la diferencia principal entre ellos se basa en el cómputo de los huecos durante el alineamiento.

En la siguiente ilustración vemos una representación de alineamiento entre los distintos enfoques algorítmicos al problema del alineamiento.

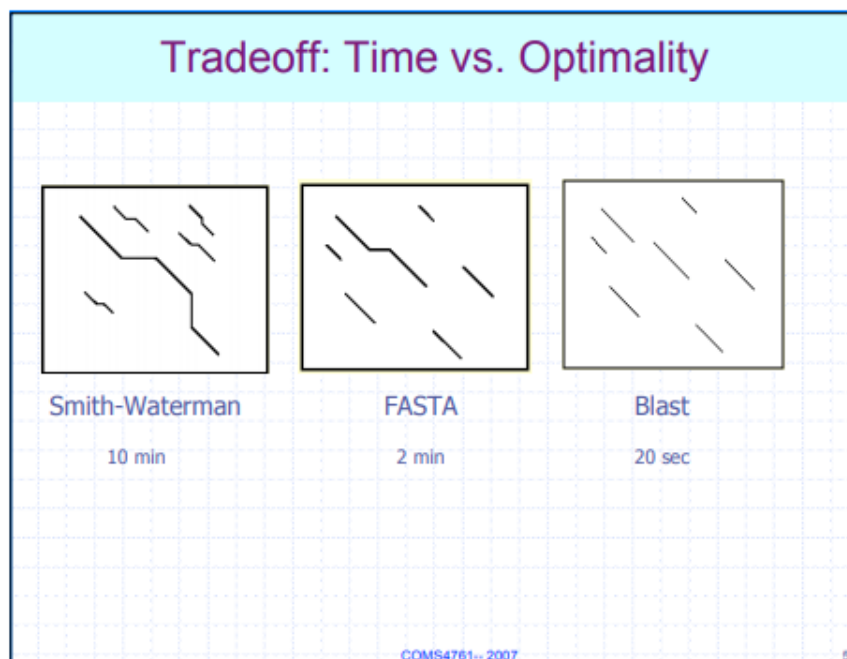


Ilustración 8 - Ejecución de diferentes algoritmos para el alineamiento de secuencias. Imagen tomada de Chapter 2.3 Searching Sequence Databases; FASTA, BLAST, Computer Science Department. Columbia University (Prof. Yechiam Yemini, 2007)

Representación de resultado del alineamiento a nivel de carácter genético.

```
SECUEC NIA BD: AATACCGCGGTATGACCCACCAATAATTACCCCACTACTCTGACACTAATTTCTGTCACCCAACTAAAAATATTAATTCARATTACCATCTACCCCCCTCACCCAAAACCCATAAAA
QUERY: AATACCGCGGTATGACCCACCACTCTCTGACACTAATTTCTGTCACCCAACTAAAAATATTAATTCARATTACCATCTACCCCCCTCACCCAAAACCCATAAAA
-----
ALIGNMENTS: AATACCGCGGTATGACCCACCACTCTCTGACACTAATTTCTGTCACCCAACTAAAAATATTAATTCARATTACCATCTACCCCCCTCACCCAAAACCCATAAAA (Smith-Waterman)
ALIGNMENTS: AATACCGCGGTATGACCCACCACTCTCTGACACTAATTTCTGTCACCCAACTAAAAATATTAATTCARATTACCATCTACCCCCCTCACCCAAAACCCATAAAA (BLAST)
```

Ilustración 9 - Ejemplo de resultados de alineamiento obtenidos tras ejecutar algoritmo de Smith-Waterman y BLAST.

Consideraciones al usar BLAST:

- *BLAST es un programa heurístico y por lo tanto puede que no encuentre la solución óptima.*
- *Los resultados de BLAST requieren evidencia externa (fisiológica, filogenética, genética, etc.) para corroborar un alineamiento con interpretaciones biológicas. Por tanto no garantiza que las secuencias que alinea sean homólogas, simplemente provee posibles candidatos.*
- *La puntuación de BLAST depende del largo de la secuencia, una secuencia muy corta tendrá una puntuación menor que una grande simplemente por la cantidad de caracteres que tiene.*

Existen además otras variantes e implementaciones a esta problemática basadas por lo que ha podido consultar en técnicas similares:

WU BLAST – es una implementación por bioinformáticos de la Universidad de Washington. WU BLAST es software propietario y es gratuito solo para uso académico.

En cuanto a la utilización de estrategias de deep learning no se encuentra información clara que ningún proyecto operativo en producción, utilice soluciones orientadas a técnicas de deep learning. Si existen artículos e implementaciones académicas con ensayos en entornos acotados demostrando interesantes comportamientos de utilidad para la identificación de significados biológicos. (Chin, 2018) (Shi, 2019)

3 ESTUDIO DE VIABILIDAD

3.1 Alcance del proyecto

Se pretende como ya se ha introducido, la implementación de un sistema de consulta de alineamiento contra secuencias de una base de datos genómica basado en diferentes técnicas.

El sistema debe contar con las siguientes características:

- Una interfaz web de uso para la consulta de una secuencia genómica
- Interfaz para la parametrización de las constantes del algoritmo de búsqueda
- Gestión de una base de datos genómica
- Algorítmica y heurística para la computación del alineamiento de la secuencia de consulta contra las secuencias de la base de datos genómica.
- Utilización de arquitecturas de deep learning para la identificación de una cadena de consulta como un patrón biológico.
 - o Identificación de una cadena con la proteína que sintetiza.
 - o Detección de una cadena como patógeno conocido.
- Como resultado en cada caso, se mostrarán las secuencias con mayor puntuación de score y sus zonas de coincidencia, las zonas de no coincidencia, huecos/gaps en el alineamiento o el resultado de la identificación biológica realizado por los sistemas de deep learning así como los datos generales de sus estadísticas en la ejecución de tests.

3.2 Estudio de las alternativas y elección de solución

Los requisitos para la elección de solución son, por un lado los motivos de eficiencia de cómputo de matrices, así como las utilidades y flexibilidad del lenguaje de programación para el manejo de estas matrices de datos. Por otro

lado, la disponibilidad de librerías para la construcción de arquitecturas de deep learning.

Estos requisitos nos hace centrarnos en dos alternativas para la implementación de la solución: Python y Matlab.

La plataforma Matlab es potente y sencilla en cuanto a las operaciones de datos y matrices. Dispone de menos documentación relativa a librerías de lenguaje o arquitecturas de deep learning que Python. Probablemente es más eficiente en las tareas puramente de cómputo, ya que se trata de un sistema especialmente diseñado para el procesamiento de datos dimensionales.

Por otro lado la plataforma Python, dispone de librerías para el manejo de datos matriciales como matrices numéricas, diccionarios, dataframe, tensores. Dispone también de librerías como Scikit o Tensorflow+Keras para la implementación de redes neuronales. Destaca además por las siguientes características: ser libre, ligero, simple, con una amplia documentación y comunidad digital en internet.

La existencia y disponibilidad de mayor librería de funcionalidades, documentación y publicaciones relativas a la plataforma de Python, nos hace decantarnos por esta plataforma para la implementación del sistema.

3.3 Análisis funcional de requisitos

Del análisis de requisitos identificamos los siguientes requisitos funcionales para ser abordados en este proyecto.

REQ. FUNCIONAL	PRIORIDAD	DETALLE REQUISITOS
Aplicación interfaz web	1	> Una interfaz web de uso para la consulta de una secuencia genómica > Interfaz para la parametrización de las constantes del algoritmo de búsqueda"
Codificación de la base de datos	1	> Gestión de una base de datos genómica
Implementación Algoritmo Smith-Waterman	1	> Algorítmica y heurística para la computación del alineamiento de la secuencia de consulta contra las secuencias de la base de datos genómica
Interfaz de usuario y presentación de resultados	1	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades
Funcionalidad red neuronal MNB	2	> Utilización de arquitecturas de deep learning para la identificación de una cadena con la proteína que sintetiza.
Interfaz de usuario y presentación de resultados	2	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades
Funcionalidad red neuronal LSTM	3	> Utilización de arquitecturas de deep learning para la identificación de una cadena como patógeno conocido
Interfaz de usuario y presentación de resultados	3	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades

Tabla 1 - Tabla de requisitos funcionales del sistema.

Hacemos un estudio de los detalles que debe satisfacer cada requisito funcional del sistema.

Identificación del requerimiento	RF_1
Nombre	Aplicación interfaz web
Detalles	> Una interfaz web de uso para la consulta de una secuencia genómica > Se requiere poder implementar diferentes formularios como interfaz para el manejo y utilización de las distintas funcionalidades del sistema. > Las interfaces de usuario permitirán la parametrización de valores para la ejecución de las funcionalidades del sistema.
Prioridad	1

Tabla 2 - Detalles del análisis de requisitos del requisito funcional RF_1: Aplicación interfaz web

4 METODOLOGÍA

Para el desarrollo del proyecto se ha utilizado una metodología incremental Agile, desarrollando los objetivos del proyecto a lo largo de iteraciones aportando incrementos en cada iteración de funcionalidad que ofrece el sistema.

Observemos la representación general de esquema de proceso de la metodología SCRUM.

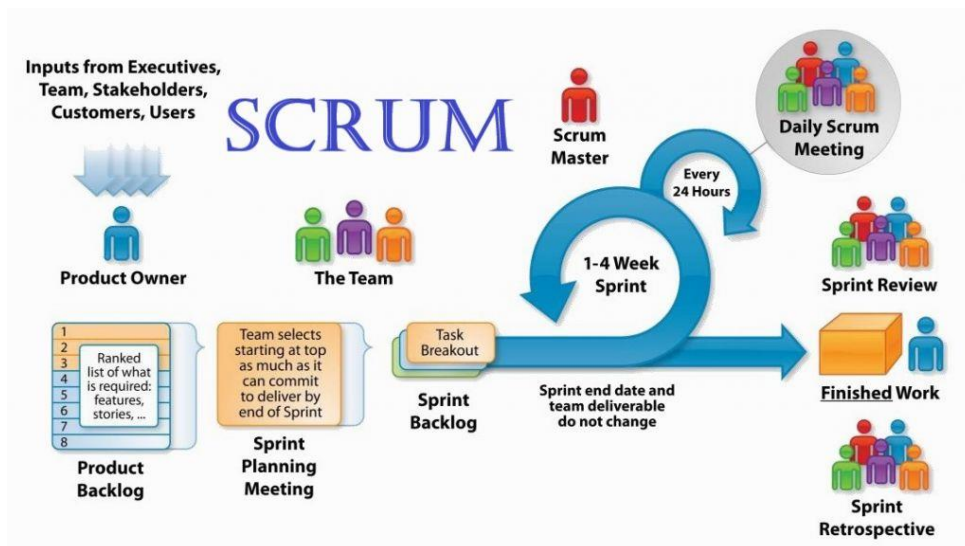


Ilustración 10 - Esquema de la metodología Agile/SCRUM. Imagen tomada de: (SCRUM-ROLES, ARTIFACTS AND CEREMONIES, 2019)

Los requisitos funcionales los organizamos en una pila de producto (product backlog). Al inicio de cada sprint definiremos la pila de sprint (sprint backlog) seleccionando las tareas que se ejecutaran en cada sprint. Para esta selección la metodología ágil se orienta basándose en la utilidad o necesidad que aporta el requisito al negocio.

REQ. FUNCIONAL	PRIORIDAD	DETALLE REQUISITOS
Aplicación interfaz web	1	> Una interfaz web de uso para la consulta de una secuencia genómica > Interfaz para la parametrización de las constantes del algoritmo de búsqueda"
Codificación de la base de datos	1	> Gestión de una base de datos genómica
Implementación Algoritmo Smith-Waterman	1	> Algorítmica y heurística para la computación del alineamiento de la secuencia de consulta contra las secuencias de la base de datos genómica
Interfaz de usuario y presentación de resultados	1	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades
Funcionalidad red neuronal MNB	2	> Utilización de arquitecturas de deep learning para la identificación de una cadena con la proteína que sintetiza.
Interfaz de usuario y presentación de resultados	2	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades
Funcionalidad red neuronal LSTM	3	> Utilización de arquitecturas de deep learning para la identificación de una cadena como patógeno conocido
Interfaz de usuario y presentación de resultados	3	> En cada caso, se mostrarán los resultados obtenidos y sus estadísticas en la ejecución de las funcionalidades

Tabla 3 - Tabla de tareas pila de producto del sistema.

Para el análisis y diseño del sistema hemos utilizado diagramas de flujo de datos, esquemas en pseudocódigo de los algoritmos y diagramas de arquitecturas de deep learning.

5 TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS EN EL PROYECTO

Para el desarrollo del sistema se han desarrollado diferentes tecnologías o herramientas.

- Python 3.7
- Visual Studio Code, como IDE o entorno de programación.
- Librerías Python:
 - Flask y Flask-DebugToolbar
 - Pandas - data analysis and manipulation tool
 - math, sqlite3, numpy
 - SCIKIT: datasets
 - TensorFlow y Keras
 - Bootstrap

6 ESTIMACIÓN DE RECURSOS Y PLANIFICACIÓN

Para realizar una estimación del proyecto utilizamos una técnica de estimación denominada por tres puntos, descrita en la guía PMBOK - Guía de los fundamentos para la dirección de proyectos en su capítulo de estimación de proyectos. (Institute, Guía de los fundamentos para la dirección de proyectos, 1996)

La estimación por tres puntos o tres valores consiste en valorar el coste de una tarea en base a ponderar los costes: Más probable (cM), Optimista (cO) y Pesimista (cP).

$$\text{Coste Esperado } c_e = \frac{c_o + 4c_m + c_p}{6}$$

Ecuación 1 - Ecuación de estimación de costes por tres puntos. (Institute, PMBOK - A Guide to the Project Management Body of Knowledge (version 6), 2017)

Se listan las tareas a realizar y se realiza un ejercicio de análisis de coste basado en esta técnica de cada tarea descrita.

Como puede observarse en la siguiente tabla se ha realizado una secuenciación de tareas para el control de la ejecución del proyecto.

LISTA DE TAREAS				
ETAPA	TAREA	FECHA FINAL PREVISTA	HORAS ESTIMADAS	SITUACIÓN
Sprint 0	Busqueda de informacion y contextualizacion	01/01/2021	43,33	Terminado
Sprint 0	Definicion del proyecto	20/01/2021	21,67	Terminado
Sprint 0	Planificacion del proyecto	25/01/2021	8,67	Terminado
Sprint 1	Configuracion del entorno	30/01/2021	10,83	Terminado
Sprint 1	Aplicación interfaz web	05/02/2021	17,17	Terminado
Sprint 1	Codificación de la base de datos	07/02/2021	12,17	Terminado
Sprint 2	Implementacion Algoritmo Smith-Waterman	05/03/2021	52,50	Terminado
Sprint 2	Validación y Pruebas	10/03/2021	7,83	Terminado
Sprint 2	Análisis algorítmico y mediciones de computo	20/03/2021	7,83	Terminado
Sprint 3	Mejora red neuronal MNB	20/04/2021	45,83	Terminado
Sprint 3	Medicion de resultados	30/04/2021	15,33	Terminado
Sprint 4	Mejora red neuronal LSTM	20/04/2021	45,83	Terminado
Sprint 4	Medicion de resultados	30/04/2021	15,33	Terminado
Final	Documentacion	30/05/2021	35,83	En curso
Final	Entrega y defensa del proyecto	30/06/2021	30,00	En curso

370,17

Tabla 4 - Tabla de product backlog o pipa de producto que describe la lista de tareas a realizar a lo largo del proyecto, con información en relación a la etapa y plazos estimados de ejecución.

La estimación inicial del proyecto nos da un coste de 370 Horas cumpliéndose con los plazos de entrega marcados.

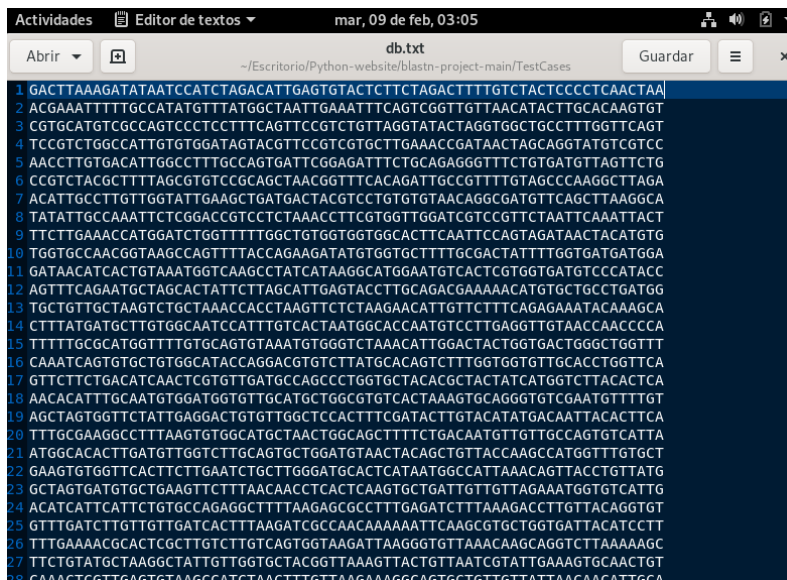
7 DESARROLLO DEL PROYECTO.

7.1 Análisis y diseño inicial del sistema

En esta sección realizaremos un análisis y diseño general del sistema, de cara a poder ejecutar las etapas de implementación de cada Sprint en los que se ha dividido el proyecto.

7.1.1 Base de datos genómica

El desarrollo de la base de datos consistirá en ficheros de texto plano en formato FASTA como puede verse en la ilustración.



```
1 GACTTAAGATATAATCCATCTAGACATTGAGTGTACTCTTAGACTTTTGTCTACTCCCCTCAACTAA
2 ACGAAATTTTGCATATGTTTATGGTAATTGAAATTCAGTCGGTTGTTAACATACTTGACAAAGTGT
3 CGTGCATGTGCGCAGTCCCTCCTTTCAGTCCGTCTGTTAGGTACTAGGTGGCTGCCTTTGGTTCAGT
4 TCCGTCTGGCCATTGTGTGGATAGTACGTTCCGTCTGCTTGA AACCGATAACTAGCAGGTATGTCGTCC
5 AACCTGTGACATTGGCCTTTCAGTGTTCGGAGATTTCTGCAGAGGTTTCTGTGATGTTAGTTCG
6 CCGTCTACGCTTTAGCGTGTCCGACGCTAACGGTTTCACAGATTGCCGTTTGTAGCCCAAGGCTTAGA
7 ACATTGCCTTGTGGTATTGAAGCTGATGACTACGTCCTGTGTGAACAGGCGATGTTACGTTAAGGCA
8 TATATTGCCAAATTCCTGGACCGTCTCTAAACCTTCGTGGTGGATGCTCCGTTCTAATTCAAATTA
9 TTCTTGAACCATGGATCTGGTTTTGGCTGGTGGTGGCACTTCAATTCAGTAGATAACTACATGTG
10 TGGTGCCAAACGGTAAGCCAGTTTTACCAGAAGATATGTTGGTCTTTTGCAGCTATTTTGGTGTGATGGA
11 GATAACATCACTGTAATGGTCAAGCCTATCATAAGGCATGGAATGCTACTCGTGGTGTGTCATACC
12 AGTTTCAGAAATGCTAGCACTATTCTTAGCATTGAGTACCTTTCAGACGAAAAACATGTGCTGCTGATGG
13 TGCTGTTGCTAAGTCTGCTAAACCACTAAGTCTCTAAGAACATTGTTCTTTCAGAGAAATACAAAGCA
14 CTTTATGATGCTTGTGGCAATCCATTTGTCACTAATGGCACCAATGTCCTTGAGGTTGTAACCAACCCCA
15 TTTTTCGCGCATGGTTTTGTGCAAGTGAATGTGGGCTTAAACATTGGACTACTGGTACTGGGCTGGTTT
16 CAAATCAGTGTGCTGTGGCATACCGACGCTGCTTATGCACAGTCTTTGGTGGTGTGCACCTGGTTCA
17 GTTCTCTGACATCAACTCGTGTGATGCCAGCCCTGGTGTACACGCTACTATCATGGTCTTACACTCA
18 AACACATTTGCAATGTGGATGGTGTGATGCTGGCGTGTCACTAAAGTGCAGGGTGTGCAATGTTTGT
19 AGCTAGTGGTCTATTGAGGACTGTGTGGCTCCACTTTCGATACTGTACATATGACAAATACACTTCA
20 TTTGCGAAGGCTTTAAGTGTGGCATGCTAACTGGCAGCTTTTCTGACAATGTTTGGCCAGTGCATTA
21 ATGGCACACTTGATGTTGGTCTTGCAGTGTGGATGTAACACTAGCTGTTTACCAAGCATTGGTTTGGTCT
22 GAAGTGTGGTCTACTTCTGAACTGCTTGGGATGCACATAATGGCATTAAACAGTTACCTGTTATG
23 GCTAGTGTGCTGAAAGTCTTTAACAACCTCACTCAAGTGTGATTGTTGTTAGAAATGGTGTATTG
24 ACATCATTCTGTGCGCAGGCTTTTAAAGAGCCTTTGAGATCTTTAAAGACCTTGTACAGGTTG
25 GTTTGATCTTGTGTTGATCACTTTAAGATCGCCAAACAAAAATTCAAGCGTGTGGTATTACATCCTT
26 TTTGAAAACGCACTCGCTTGTCTGTGAGTGGTAAAGTTAAGGGTGTAAACAGCAGGCTTAAAAAGC
27 TTCTGTATGCTAAGGCTATTGTTGGTCTACGGTTAAAGTTACTGTTAATCGTATTGAAAGTGCACCTGT
28 CAAACTCGTTGAGTGTAAAGCCTAAGCTTAAAGAAAGGCAAGTCTGTTTAAAGCAATTCGA
```

Ilustración 11 – Fichero de datos en formato FASTA utilizado durante el desarrollo del proyecto en las fase tempranas.

Identificamos los siguientes objetivos:

- Cada línea será una secuencia de longitud no definida
- Existirán unos meta-datos iniciales al inicio de la línea para contextualizar la secuencia (origen, clasificación, tipo de cadena, posición/ubicación genómica, observaciones, ...)

7.1.2 Definición del sistema de alineamiento

Existen trabajos y documentos de autores en bioinformática explicando los procesos del alineamiento entre cadenas desde una perspectiva algorítmica, mostrando de forma didáctica paso a paso este alineamiento. (Germán) (John, 2013) (Federico, Puertas, Oswald, & Allende) (Lesk, 2008).

El algoritmo a construir en este trabajo, está basado en una implementación heurística de la especificación del algoritmo de Smith and Waterman (1981) para el alineamiento con huecos entre 2 cadenas.

Recordaremos a través del siguiente gráfico las diferencias de implementación entre las alternativas.

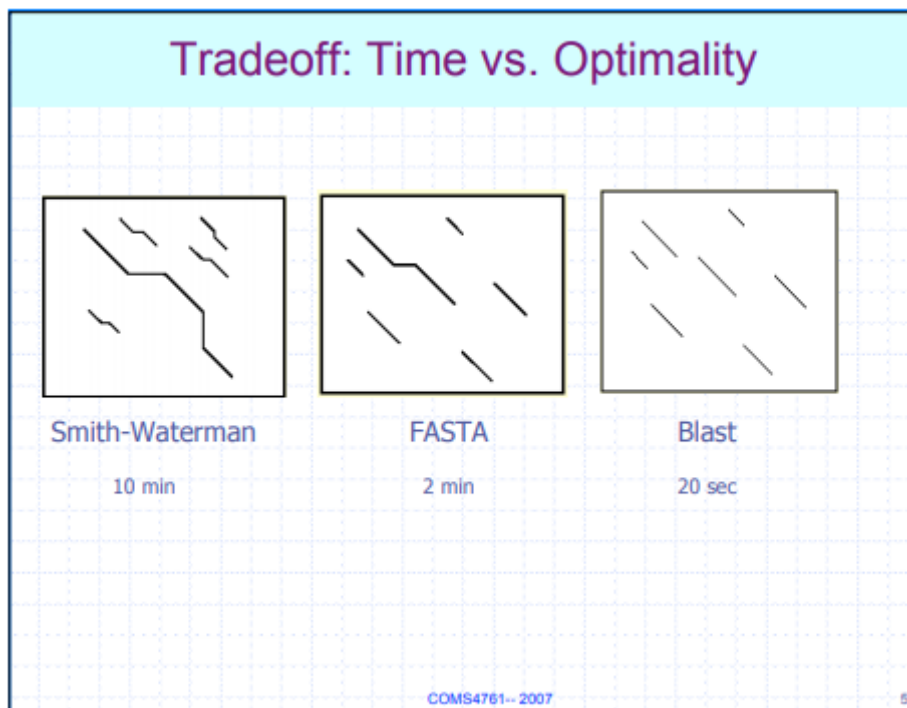


Ilustración 12 - Ejecución de diferentes algoritmos para el alineamiento de secuencias. Imagen tomada de Chapter 2.3 Searching Sequence Databases; FASTA, BLAST, Computer Science Department. Columbia University (Prof. Yechiam Yemini, 2007)

```

SECUCENIA BD: AATACCGCGSTATGACCCACCCTAAATTACCCOCCATCTCTGACACTATTTCTGTGACCCCACTAAAAATATTAATTCARATTACCATCTACCCOCCCTCACCARAAACCCATAAA
QUERY: AATACCGCGSTATGACCCACCCTCTCTGACACTATTTCTGTGACCCCACTAAAAATATTAATTCARATTACCATCTACCCOCCCTCACCARAAACCCATAAA
ALIGNMENTS: AATACCGCGSTATGACCCACCCTCTCTGACACTATTTCTGTGACCCCACTAAAAATATTAATTCARATTACCATCTACCCOCCCTCACCARAAACCCATAAA (Smith-Waterman)
ALIGNMENTS: AATACCGCGSTATGACCCACCCTCTCTGACACTATTTCTGTGACCCCACTAAAAATATTAATTCARATTACCATCTACCCOCCCTCACCARAAACCCATAAA (BLAST)
    
```

Ilustración 13 - Ejemplo de resultados de alineamiento obtenidos tras ejecutar algoritmo de Smith-Waterman y BLAST.

El algoritmo del sistema en su conjunto, se basa en comparar una cadena de entrada "query" contra todas las cadenas de una base de datos usando el alineamiento S&W, almacenando para cada cadena la puntuación de los alineamientos calculados. La comparación de la cadena de consulta y una cadena de la base de datos según una especificación genérica de BLAST se puede describir en 3 etapas (Ian Korf, Mark Yandell, & Joseph Bedell, 2003):

1. *Localización de semillas (HSP) de coincidencia exacta de un tamaño k de query sobre otra cadena, o bien de alta puntuación/coincidencia sin huecos.*¹
2. *Extender las semillas (Algoritmo de Smith-Waterman) que superan cierto umbral en una extensión más amplia con huecos sobre la cadena.*
3. *Ordenar y determinar los valores estadísticos o indicadores del alineamiento*

Esto es necesario realizarlo para todas las secuencias de la base de datos, almacenar los resultados de la comparación de la query contra cada cadena almacenada en el sistema y finalmente quedarnos las mejores secuencias para mostrárselas al usuario.

En la **primera etapa**, a la hora de comparar 2 cadenas para localizar las semillas, nos encontramos con un problema: ¿Dónde empezar a contar en cada cadena? ¿Cuál es el punto óptimo? Pensándolo enseguida nos damos cuenta que no es lo mismo contar desde una posición que otra. La idea tradicional del estudio es que todas las posiciones que den lugar a una combinación hacia adelante de longitud K caracteres (k-mers) deben ser evaluadas entre sí como posibles candidatas a semilla para un alineamiento mayor extendido. En esta línea detallan el proceso entre otros los autores: (Clelia, Pietro, Alessandro, & Roberto, 2014)

¹ HSP - Oxford Reference: (High-scoring Segment Pair): abbr.: HSP; in BLAST searches, an ungapped local pairwise alignment between arbitrary but equal-length fragments of a query and a database sequence that scores above a user-defined threshold. HSPs are the fundamental unit of BLAST.

Representamos concepto de KMERS y combinaciones para cada nivel de K de una cadena dada, representamos la ilustración para la cadena de ejemplo: "GTAGAGCTGT".

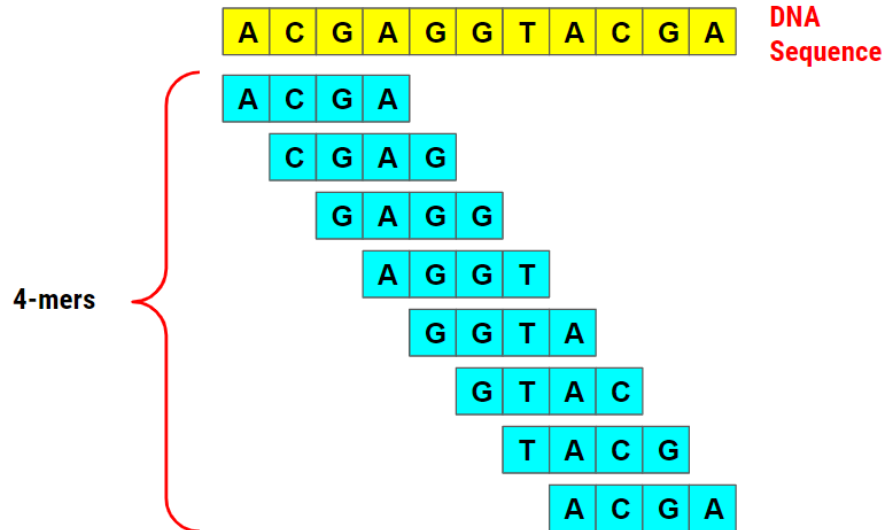


Ilustración 14 - Kmers para la secuencia "GTAGAGCTGT"

Imagen tomada de: "Bioinformatics 1: K-mer Counting" (Gunavaran Brihadiswaran, 2020)

Una vez generados los kmers de las cadenas de consulta y de la cadena del sistema contra la que se está comparando, deberá puntuarse las combinaciones entre los k-mers de la query y de la cadena del sistema, para evaluar que puntuación de alineamiento sin huecos hay. Esto permite que la semilla óptima entre 2 cadenas, pueda ser por ejemplo, la posición 3 de la cadena query y la posición 12 de una cadena en el sistema, seguido a partir de ahí, se tratará de extender una zona de gran alineamiento.

Para comenzar la segunda etapa de alineamiento extendido, seleccionaremos las mejores semillas, aquellas con mejor puntuación, ya que serán las prometedoras, y tendrá sentido generar un alineamiento con huecos expandido. Suele llamarse al punto de umbral de selección de semillas **SEED_THRESHOLD**. Solo a partir de este punto las semillas serán aceptadas y pasaran a la siguiente fase para la extensión con huecos. Estas son identificadas como HSP – high-scoring segment pair

Observamos en el siguiente grafico las combinaciones para buscar las semillas candidatas para una consulta de longitud 7 símbolos, una constante K de 6, y un sistema de 3 cadenas almacenadas en base de datos de longitud entre 7-9.

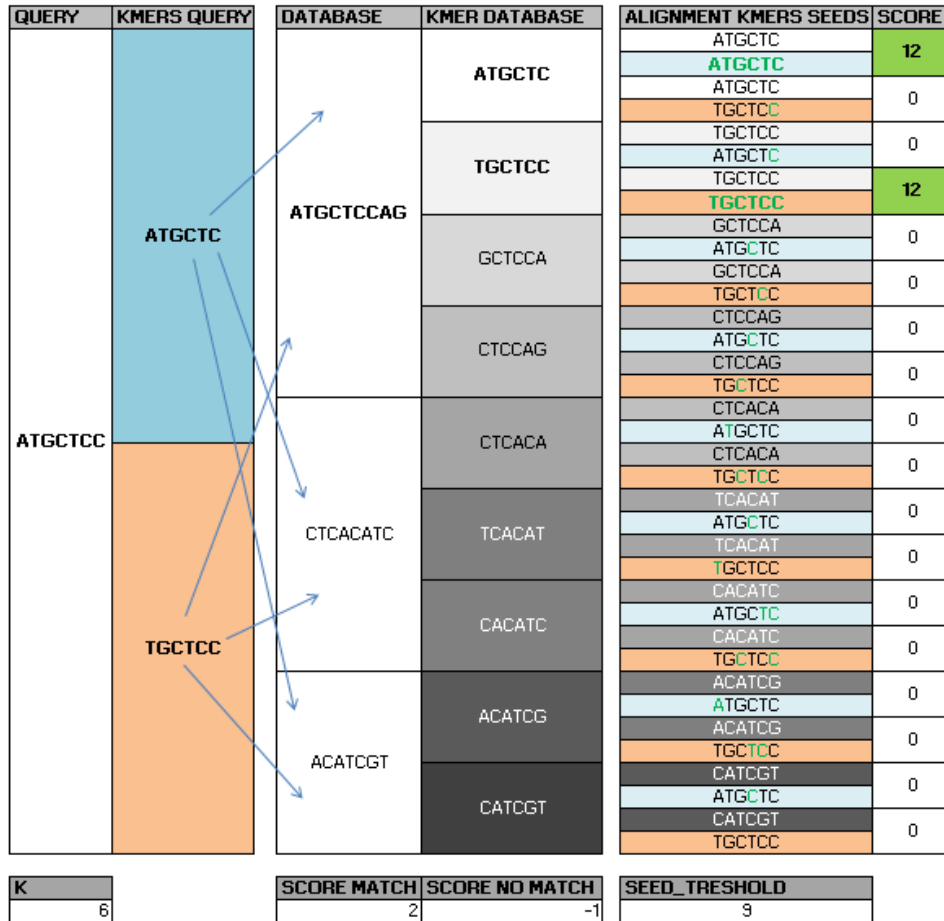


Ilustración 15 - Combinaciones en kmers para localizar los puntos prometedores de extensión del alineamiento

Solo las mejores semillas candidatas para una cadena pasaran a las **segunda etapa**, realizaremos a partir de ellas el alineamiento extendido con huecos sobre el resto de la cadena. Para el desarrollo de este alineamiento extendido con huecos implementaremos el algoritmo de Smith & Waterman como lo describe en "Sequence Comparison: Theory and Methods" (Kun-Mao Chao & Louxin Zhang, 2008). También lo explican otro autores como: (Vinuesa, 2007).

El algoritmo de Smith & Waterman consiste en proyectar sobre una matriz la mejor puntuación del alineamiento en cierto punto dado por las

coordenadas del recorriendo en filas y columnas de las cadenas de consulta y la cadena almacenada en el sistema respectivamente.

Podemos en la siguiente ilustración el cálculo de alineamiento óptimo calculado sobre una matriz en la que se representan las cadenas de consulta en filas y la cadena contra la que se compara en columnas.

		1	2	3	4	5	6	7	8	9	10	11
		C	T	A	T	C	A	T	T	C	T	G
1	G	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	5
2	A	-4	-4	5	-4	-4	5	-4	-4	-4	-4	-4
3	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
4	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
5	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
6	A	-4	-4	5	-4	-4	5	-4	-4	-4	-4	-4
7	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
8	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
9	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
10	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4

Ilustración 16 - Ejemplo animado paso a paso de la ejecución de la matriz y recomponer la solución. Imagen tomada de "Sequence Comparison: Theory and Methods" (Kun-Mao Chao & Louxin Zhang, 2008)

Se intuye que el alineamiento sin huecos sobre la matriz, representa la diagonal. Y considerando huecos, aunque se puedan presentar saltos en vertical, el alineamiento positivo, sigue una trayectoria diagonal. Por tanto la idea no es completar toda la matriz computacionalmente, sino tratar de acotar el coste computacional a esa "diagonal". Esta idea la trataremos de utilizar para acotar el cálculo desde un punto de vista heurístico.

Podemos ver esa representación de diagonal del alineamiento en la siguiente ilustración.

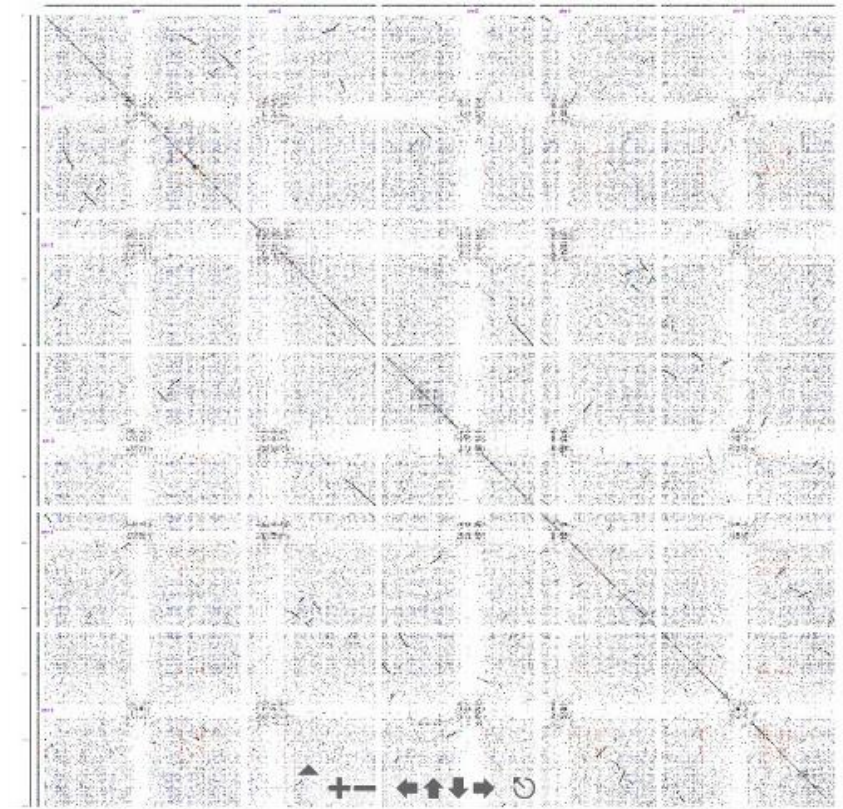


Ilustración 17 - Plot del alineamiento global de 2 secuencias
Imagen tomada del curso Bioinformática básica de (Sales, Jose, & Peio)

Representación de la diagonal a nivel de alineamiento local con huecos.

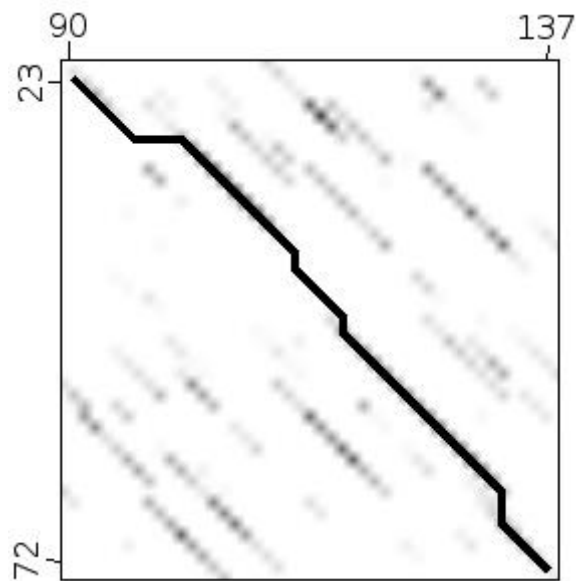


Ilustración 18 - Plot del alineamiento local de 2 zonas secuencias
Imagen tomada del curso Bioinformática básica de (Sales, Jose, & Peio)

En este punto del alineamiento donde la definición de la heurística implementada hará al sistema caer al tratar de superar un máximo local o superarlo y continuar la extensión.

Una vez hechos estos pasos, para una secuencia podremos saber si la query está contenida en ella y qué grado de extensión de similitud hay entre la query y una secuencia almacenada en el sistema.

Podemos ahora describir la receta del algoritmo, a una aproximación con un poco más de detalle sobre la especificación del sistema.

Por cada secuencia almacenada en el sistema, hacer:

1. *Localización de semillas (seeds) de coincidencia exacta de un tamaño k de query sobre otra cadena, o bien de alta puntuación/coincidencia sin huecos.*
 - a. Problema *¿dónde empezar a contar? Todas las combinaciones de longitud k (k-mers) serán candidatas a semilla*
 - b. Heurística: *Selección solo de mejores semillas para cada cadena de la base de datos.*
2. *Extender las semillas que superan cierto umbral en una extensión más amplia con huecos sobre la cadena.*
 - a. Matriz **Smith-Waterman**: *para cada semilla seleccionada, consiste en ir generar una matriz con la mejor puntuación de alineamiento posible a cada celda extendiendo con huecos la puntuación inicial de la semilla sobre cada posición de la matriz calculada.*
 - b. *Finalizada la matriz, se reconstruye la ruta solución desde la celda con la puntuación más alta para esa semilla*
 - c. Heurística: *Acotar cálculo de la Matriz Smith-Waterman a la "frontera" próxima a la diagonal óptima.*
3. *Incluirlo a la lista de resultados de forma ordenada y determinar los valores estadísticos o informadores del alineamiento.*

7.1.3 Interfaz de usuario

Se pretende una herramienta desarrollada con una interfaz web para la que hemos seleccionado como framework (Flask) en Python para el desarrollo de la interfaz de usuario. La librería dispone de documentación y guías de uso, así como comunidad de apoyo libre en internet.¹

Esta arquitectura nos permite la utilización de una interfaz de desarrollo de aplicación web cliente-servidor y la utilización de un patrón MVC para la interacción entre las capas de aplicación. En las páginas de servidor utiliza el lenguaje JINJA² para acceder al modelo.

Mediante el uso de formularios HTML implementaremos la interacción del usuario con los controladores a través de peticiones POST HTTP al servidor para el cálculo del alineamiento a los datos de entrada. La interfaz web, nos permitirá ajustar los parámetros de ejecución del algoritmo de alineamiento. Y una vez ejecutado los servicios del sistema ofrecerá los resultados de forma que favorezca su interpretación.

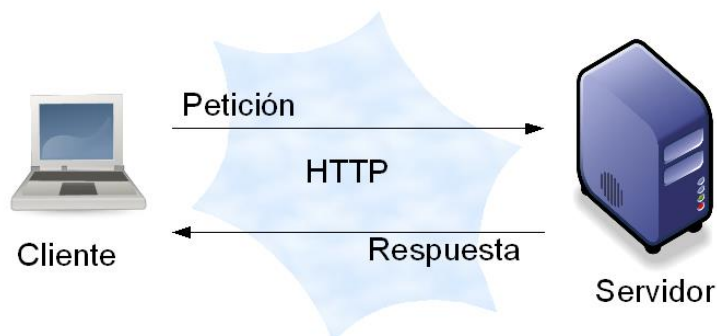


Ilustración 19 - Esquema cliente servidor HTTP.

¹ Flask Documentation - <https://flask.palletsprojects.com/en/1.1.x/>. Dispone de guías de uso, ejemplos y definición de las librerías básicas para una aplicación MVC.

² Jinja Documentation - <https://jinja.palletsprojects.com/en/2.11.x/> Guía y ejemplo de uso de la librería.

El sistema permitirá al usuario la consulta parametrizada, aportando por parte del usuario no solo la cadena de consulta sino algunos datos utilizados como variables para la ejecución del alineamiento.

Parámetros de ejecución:

QUERY: cadena que se desea consultar contra la base de datos genética.

K: tamaño de la búsqueda inicial para la identificación de semillas

MATCH_SCORE: Score del alineamiento correcto de un símbolo de la cadena.

MISMATCH_SCORE: Score del alineamiento incorrecto de un símbolo de la cadena.

GAP_SCORE: Score del alineamiento al computar un carácter de gap en la cadena.

SEED_TRESHOLD: Umbral mínimo que debe cumplir una semilla de tamaño K para ser extendido su cálculo de alineamiento.

7.1.4 Arquitecturas de deep learning

Las arquitecturas deep learning se basan en el aprendizaje a partir de los datos. Estas se comportan procesando datos de entrada que codifican una situación para la que la red neuronal debe ofrecer una respuesta. Inicialmente la respuesta del sistema sería errónea, para ajustar su respuesta, se requiere de una etapa de aprendizaje a raíz de un conjunto de entrenamiento. El sistema se irá ajustando en sus cálculos y pesos internos. Las nociones principales de estos esquemas están descritas de forma precisa por el autor (Russell, 2018).

Podemos clasificar las diferentes arquitecturas para deep learning según describen diferentes autores (Pramod Singh & Avinash Manure, 2019) (María Jesús de la Fuente Aparicio & Teodoro Calonge, 1999):

- Clasificadores estocásticos
- Redes neuronales multicapa
- Modelos redes convolucionales
- Modelos recurrentes
- Combinación secuencial de varias de estas arquitecturas

Estas arquitecturas se nos ofrecen la caja de herramientas para el análisis por deep learning, pero cada problema debe ser analizado concretamente para los datos y los objetivos que persigue el diseño de la arquitectura. Es requisito pensar en la forma que se van a digitalizar los datos de la realidad o problema, en como los codificaremos y pre-procesaremos, y con qué tipo de arquitectura nos permitirá extraer, identificar y resaltar mejor la información intrínseca al mensaje de entrada para el que ofrecer una respuesta por el sistema. Esta tarea es la más importante y difícil al plantear la resolución de un problema a través de deep learning.

Los modelos indicados anteriormente y analizada su arquitectura, podemos ver que son útiles en ciertos escenarios.

Los clasificadores estocásticos y redes neuronales multicapa, son útiles en responder a situaciones de difícil comprensión lógica por el número de parámetros y correlaciones entre existentes entre ellos. Por ejemplo un actuador o alarma industrial.

Las redes convolucionales, responden bien a modelos para tratar reconocimientos de imágenes y voz, resaltando la forma y frontera de las cosas, no su color, sino la forma que define una cara en una imagen por ejemplo.

Los modelos recurrentes son útiles para el procesamiento de datos secuenciales. También en usos de lingüística tanto en reconocimiento como en creación de mensajes. Procesan la información tratando de entender cada símbolo sobre el contexto de la secuencia completa de símbolos o frase.

En la siguiente imagen vemos un sistema neuronal compuesto por 2 capas convolucionales y una etapa final clasificadora implementada con una red neuronal multicapa para generar la salida.

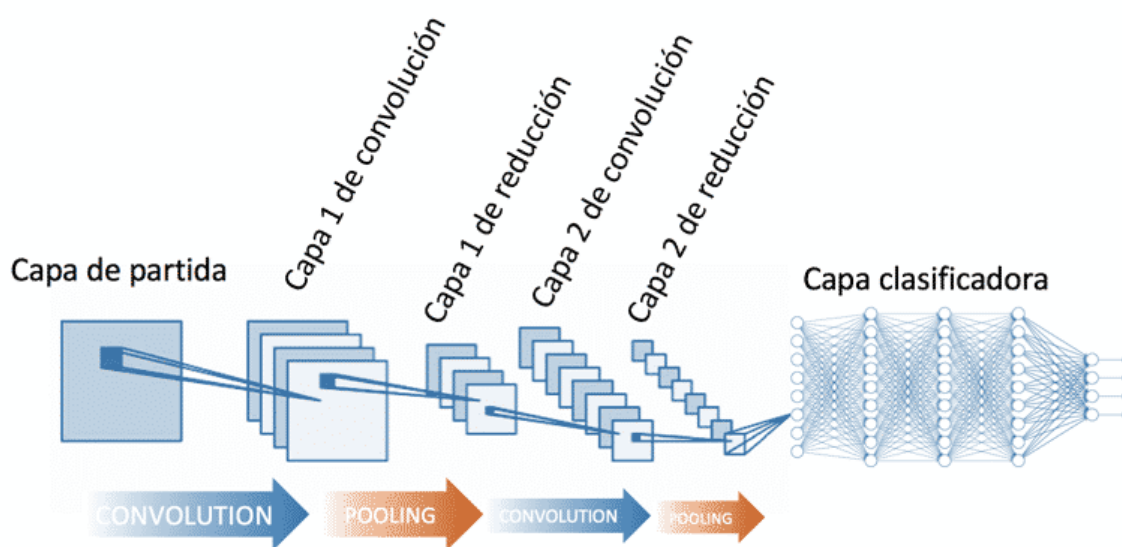


Ilustración 20 - Esquema de la arquitectura de una red neuronal convolucional CNN. Imagen tomada del post “Red neuronal convolucional CNN” de: (Diego Calvo , 2017)

Las diferentes arquitecturas requieren o manejan la información de distinto modo, incluso podemos desarrollar combinaciones de arquitecturas de forma secuencial aumentando la complejidad del modelo.

Referenciando las ideas del autor de las publicaciones “Python deep learning” (Torres, Python deep learning, 2020) y (Torres, Deep learning: Introducción práctica con Keras, 2018), relativas a la forma de los datos son procesados en forma de tensores, esto es una generalización de vector en forma de tupla en un espacio de N dimensiones. De este modo la forma de los datos son tuplas que representan el modelo, indicando el valor en sus N dimensiones

para cada tensor. Por tanto lo que introduce a las capas de un sistema de deep learning son tensores, que pueden verse como matrices de N dimensiones. Así, los datos son pre-procesados y preparados en forma concreta y homogénea para la entrada del sistema neuronal, tanto en su etapa de aprendizaje como en la etapa de producción.

En el desarrollo de los sprints relativos a sistemas neuronales, trataremos la forma concreta de la arquitectura en la que trabajaremos los datos en cada aplicación.

7.2 Configuración del entorno

Para la instalación del entorno de desarrollo, utilizaremos los manuales y documentación de instalación siguiendo los pasos para las siguientes librerías:

Para que podemos comenzar a construir la capa la interfaz web necesitamos las siguientes librerías.

- Visual Studio Code, como IDE o entorno de programación.
- Python 3.7 y gestor de paquetes python-pip
- Flask

Para la capa de servidor, necesitaremos las siguientes librerías. Es aquí donde implementaremos los procesos de alineamiento y deep learning instalaremos las siguientes librerías.

- Tensorflow y Keras
- Pandas
- sqlite
- Matplot
- sklern

Puede consultarse en el ANEXO, sección “Manual de instalación” los detalles e instrucciones de la instalación de cada librería.

A partir de aquí podemos comenzar a desarrollar la capa de software del sistema.

7.3 Implementación

Llegamos a la implementación del sprint del proyecto. El proyecto se programa en 4 sprints para la implementación de los requisitos funcionales. Al inicio de cada sprint re-evaluaremos los requisitos funcionales aún pendientes de la pila de producto. En cada sprint se definirá una lista de requisitos a abordar, conformando la pila de sprint (o sprint backlog) a ejecutar en cada sprint. Para esta selección la metodología ágil se orienta basándose en la utilidad, necesidad de implementación que repercute cada requisito funcional desde una perspectiva del negocio.

- **Sprint 1: Creación inicial del proyecto**
 - Creación aplicación interfaz web
 - Codificación de la base de datos genómica
- **Sprint 2: Alineamiento heurístico**
 - Implementación del Algoritmo Smith-Waterman como sistema de alineamiento heurístico
 - Interfaz de usuario y presentación de resultados
- **Sprint 3: Sistema de aprendizaje Clasificador Bayes**
 - Funcionalidad red neuronal MNB: Implementación de un sistema de aprendizaje para identificar una cadena con una proteína.
 - Interfaz de usuario y presentación de resultados
- **Sprint 4: Sistema de aprendizaje LSTM**
 - Funcionalidad red neuronal LSTM: Implementación de un sistema de aprendizaje para la identificación de una cadena con un patógeno conocido.
 - Interfaz de usuario y presentación de resultados

7.3.1 PRIMER SPRINT: Creación inicial del proyecto

Creación aplicación interfaz web.

Se desarrolla una interfaz web utilizando las librerías FLASK y JINJA una para presentar vía navegador un formulario web con el que enviar los datos de consulta y obtener los resultados calculados.

Por simplificación de este documento, no haremos hincapié en esta capa del proyecto ya que no está en las motivaciones del proyecto. Las tecnologías utilizadas son HTML, controladores de servidor HTTP en Python con Flask, lenguaje JINJA para la compilación de vistas en las peticiones a servidor, y estáticos CSS, JS e imágenes.

Puede consultarse el anexo de este documento “Manual de instalación” para ver los detalles de instalación del entorno de desarrollo.

```
root@debian:/home/jfillan/Escritorio/Python-website# python3 index.py
* Serving Flask app "index" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [22/Jan/2021 05:25:12] "GET / HTTP/1.1" 200 -
```

Ilustración 21 - Arranque de servidor web

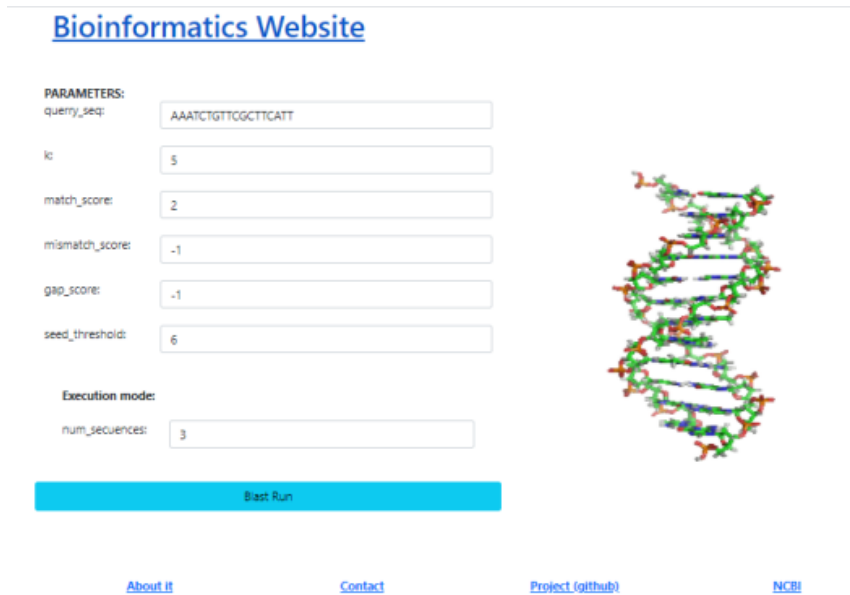


Ilustración 22 - Aspecto de la interfaz de usuario ejecutada sobre el navegador.

Codificación de la base de datos genómica

Uno de los problemas al que nos enfrentamos es la gestión de la base de datos. Se utilizara una base de datos sobre ficheros en disco, en texto plano con los caracteres de cada secuencia a nivel de fila en formato FASTA. Es un método sencillo pero a la vez muy eficaz para procesar las cadenas.

Como ya hemos comentado en el sistema se identifican los siguientes requisitos:

- Cada línea será una secuencia de longitud no determinada
- Existirán unos meta-datos iniciales al inicio de la línea para contextualizar la secuencia (origen, clasificación, tipo de cadena, posición/ubicación genómica, observaciones, ...)

El sistema de almacenamiento lo implementamos sobre ficheros .csv delimitados por “;” por su sencillez, mantenimiento y rapidez de lectura secuencial. Desde el sistema son leídos de forma secuencial línea a línea.

Herramienta para el estudio del alineamiento de secuencias de ADN mediante DeepLearning

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2																	
3																	
4																	
5																	
6																	
7																	
8																	
9																	
10																	
11																	
12																	
13																	
14																	
15																	

Ilustración 23 - Contenido de la base de datos sobre ficheros en codificación FASTA.

Retrospectiva del sprint

El sprint ha sido fructífero, hemos alcanzado los objetivos propuestos: disponiendo de una semilla inicial de la aplicación e interfaz web para poder acometer el desarrollo de las tareas de los siguientes sprints.

Seguir haciendo	Empezar a hacer	Dejar de hacer
- Seguir focalizándonos en requisitos concretos y abordarlos de forma completa	- No se han detectado nuevas estrategias mencionables en el sprint	- No se han detectado problemas mencionables en el sprint

Tabla 5 - Matriz de retrospectiva sprint 1

7.3.2 SEGUNDO SPRINT: Alineamiento heurístico

Implementación del algoritmo de alineamiento

Detallamos ahora los aspectos destacados de la implementación del algoritmo heurístico diseñado para sistema, con el objetivo de calcular el alineamiento con huecos de una secuencia de consulta (query) contra las secuencias de una base de datos, tratando de conseguir a través de la heurística un equilibrio entre calidad y eficiencia.

Como ya se ha descrito en el apartado de Análisis y diseño inicial del sistema, el proyecto de alineamiento lo interpretaremos en tres etapas:

Por cada secuencia almacenada en el sistema, hacer:

1. *Localización de semillas (HSP) de coincidencia exacta de un tamaño k de query sobre otra cadena, o bien de alta puntuación/coincidencia sin huecos.*
2. *Extender las semillas que superan cierto umbral en una extensión más amplia permitiendo de forma heurística huecos sobre la cadena.*
3. *Incluirlo a la lista de resultados de forma ordenada y determinar los valores estadísticos o informadores del alineamiento.*

Para cada HSP seleccionada, desarrollamos la extensión del alineamiento para ver hasta donde es capaz de llegar la puntuación de esta coincidencia. Para calcular este alineamiento con huecos (gaps) de la segunda etapa, realizaremos una implementación heurística propia del algoritmo de Smith-Waterman sobre una matriz en la que calcular como se desarrolla el alineamiento teniendo en cuenta la posibilidad de huecos.

Una aproximación en pseudocódigo del sistema respondería a esta especificación:

```
query ← ENTRADA
ListaHssp : LISTA
ListaBlast : LISTA
Matrix : [][] ENTERO

Para cada secuencia_db en DB
  ListaHssp.limpiar()
  Por cada k-mers Q de query
    Por cada k-mers S de secuencia_db
      Score ← evaluar_sin_huecos(Q,S,match_score,mistch_score)
      Si (score > threshold) // Si S es HSSP
        // Almacenamos semilla hssp
        ListaHssp.añadir(S, score)

// Extendemos las semillas detectadas como buenas candidatas
Por cada Hssp en ListaHssp
  mejor_alig_hsp = 0
  // Extender Alineamiento con huecos
  Desde i = Hssp.index_q ... Hasta len(query)
    Desde j = Hssp.index_db ... Hasta len(secuencia_db)
      Si query[i] == secuencia_db[j]
        D = Matrix[i-1,j-1] + match_score
      Else
        // Evaluar mismatch
        D = Matrix [i-1][j-1] + mismatch_score
        // Evaluar Gap en db sequence
        R = Matrix [i][j-1] + gap_score
        // Evaluar Gap en query sequence
        C = Matrix [i-1][j] + gap_score

      // Calcular la extensión optima
      // de la cadena en Matrix[i,j]
      Matrix[i,j] = max(D,R,C)

      If matrix[i,j] > mejor_alig_hsp
        // Almacenar la mejor solución de semilla
        // más favorable de query y secuecniá_db
        mejor_alig_hsp = Matrix[i,j]
        ListaBlast.añadir(secuencia_db,i,j,Matrix[i,j])

// Ordenar y filtrar los mejores alineamientos contra secuecniás_db.
ordenar(ListaBlast)
mostrarResultados(ListaBlast)
```

Ilustración 24 - Pseudocódigo del algoritmo de alineamiento propuesto utilizando el algoritmo de Smith and Waterman.

Finalmente, los alineamientos mejor valorados serán los mostrados al usuario con la información relativa oportuna para su contextualización.

La implementación de los requisitos del sistema se realizan en python y apoyado en las siguientes librerías de colecciones: listas, arrays, diccionarios y pandas.DataFrame para el manejo y recolección de los datos.

Mostramos ahora algunas implementaciones en Python de las secciones principales del algoritmo surgidas del esquema en pseudocódigo realizado como base de la implementación.

Función para la lectura de las secuencias en fichero csv.

```
""" Read the database in filepath and return a list of sequences found in database """
def read_data(filepath):
    reader = open(filepath, 'r')
    return re.split('\n', reader.read())[1:]
```

Ilustración 25 - Función read_data() para la lectura de las cadenas del sistema

Acceso a las columnas de cada secuencia leída del sistema. En este caso a la propia secuencia sobre la columna 6.

```
# For each sequence in db
for i in range(len(db_sequences)):

    db_seq = db_sequences[i].split(';')[6]
```

Ilustración 26 - Acceso a las columnas de una línea leída desde una fuente .csv delimitada por ";"

Función para la extracción en k-mers de una cadena.

```
""" Extracts every possible kmer from a sequence. """
def extract_kmers(sequence, k):
    kmers = []
    kmer = ""
    for i in range(len(sequence) - k + 1):
        kmer = sequence[i]
        for j in range(1, k):
            kmer += sequence[i + j]
        kmers.append(kmer)
    return kmers
```

Ilustración 27 - Función para la extracción en k-mers de una cadena.

Función para el alineamiento sin huecos entre 2 k-mers

```
""" Calculate ungapped alignment between two kmers. """
def ungapped_alignment(kmer1,kmer2,match_score,mismatch_score):
    scores = np.zeros(len(kmer1))
    for i in range(len(kmer1)):
        if(kmer1[i]==kmer2[i]):
            scores[i] = scores[i-1]+match_score
        else:
            scores[i] = scores[i-1]+mismatch_score

    return scores[-1]
```

Ilustración 28 - Función para el alineamiento sin huecos entre 2 k-mers.

Esquema de la implementación de la función para la localización de las semillas HSP entre query y una cadena de la base de datos.

```
""" Finds the seeds with score > seed_threshold """
def find_seeds(query_seq,db_seq,k,match_score,mismatch_score,seed_threshold):

    # split query_seq in kmers
    query_kmers = extract_kmers(query_seq,k)

    # split db_seq in kmers
    db_kmers = extract_kmers(db_seq,k)

    # for each kmers in query_seq
    for i in range(len(query_kmers)):
        # for each kmers in db_seq
        for j in range(len(db_kmers)):
            # calculate score
            score = ungapped_alignment(query_kmers[i],db_kmers[j],match_score,mismatch_score)
            # if > seed_threshold, is a valid seed
            if(score>=seed_threshold):
                kmers.append(db_kmers[j])
                q_indicies.append(i)
                db_indicies.append(j)
                scores.append(score)
                query_kmer.append(query_kmers[i])
```

Ilustración 29 - Esquema de la implementación de la función para la localización de las semillas HSP.

A la hora de desarrollar la extensión sobre la matriz Smith-Waterman, hay zonas de la matriz que no son interesantes calcularlas como ya hemos comentado a lo largo del trabajo, se intuye que el alineamiento debe avanzar en forma aproximada a la de una diagonal en la matriz.

		1	2	3	4	5	6	7	8	9	10	11
		C	T	A	T	C	A	T	T	C	T	G
1	G	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	5
2	A	-4	-4	5	-4	-4	5	-4	-4	-4	-4	-4
3	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
4	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
5	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
6	A	-4	-4	5	-4	-4	5	-4	-4	-4	-4	-4
7	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
8	C	5	-4	-4	-4	5	-4	-4	-4	5	-4	-4
9	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4
10	T	-4	5	-4	5	-4	-4	5	5	-4	5	-4

Ilustración 30 - Ejemplo animado paso a paso de la ejecución de la matriz y recomponer la solución. Imagen tomada de "Sequence Comparison: Theory and Methods" (Kun-Mao Chao & Louxin Zhang, 2008)

Esquema principal de la implementación del algoritmo Smith-Waterman.

```

cont = True
#Start building table from the kmer position
for i in range(query_index+k, len(query_seq)):
    #if(not cont):
    #    cont = True
    #    break

    # evitar que nos salgamos del recorrido en i sobre query seq
    if (i >= len(query_seq)):
        break

    for j in range(db_index+k, len(db_seq)):

        #if(not cont):
        #    cont = True
        #    break
        R = 0
        C = 0

        if(query_seq[i] == db_seq[j]):
            D = matrix[i-1][j-1] + match_score
        else:
            # Evaluate mismatch
            D = matrix[i-1][j-1] + mismatch_score
            # Evaluate Gap in db sequence
            R = matrix[i][j-1] + gap_score
            # Evaluate Gap in query sequence
            C = matrix[i-1][j] + gap_score
    
```

Ilustración 31 - Esquema de la implementación del algoritmo Smith-Waterman.

Para establecer el corte, utilizaremos 2 estrategias:

- 1) Si hay mach y es solución óptima a la celda, avanzar en la matriz sobre la diagonal

- 2) Si no hay match y el valor cae aun encontrando un próximo match es inferior a la solución óptima encontrada, dejamos de recorrer la fila y continuamos por la siguiente fila.

```
# Acotar recorridos de la matriz, si hay match y es solución óptima, avanzar en la diagonal
if(query_seq[i] == db_seq[j]):
    if (D == np.max([D,R,C])):
        if (i+1 < len(query_seq)):
            matrix[i+1][j] = matrix[i][j] + gap_score
        if (j+1 < len(db_seq)):
            matrix[i][j+1] = matrix[i][j] + gap_score
        j+=1
        break

# Si no hay match, puede llegar el punto que no tenga sentido seguir evaluando posibles GAPS
if(query_seq[i] != db_seq[j] and j > j_aux):
    if (matrix[i][j] + match_score < max_score):
        j = j_aux
        break
```

Ilustración 32 – Heurística del algoritmo de alineamiento. Acotar recorridos sobre la matriz Smith – Waterman.

Reconstrucción de la mejor solución de alineamiento encontrado sobre una cadena del sistema.

```
# reconstruimos desde la mejor puntuación matrix[row][col] hacia atrás
while(row >= query_index+k and col>=db_index+k):
    # retroceder match
    if(matrix[row][col] == matrix[row-1][col-1]+match_score):
        query_alignment += query_seq[row]
        db_alignment += db_seq[col]
        row-=1
        col-=1

    # retroceder mismatch
    elif(matrix[row][col] == matrix[row-1][col-1]+mismatch_score):
        query_alignment += query_seq[row].lower()
        db_alignment += db_seq[col].lower()
        row-=1
        col-=1

    # retroceder GAP de filas
    elif(matrix[row][col] == matrix[row-1][col]+gap_score):
        query_alignment += query_seq[row].lower()
        db_alignment += "_"
        row -=1

    # retroceder GAP de columnas
    elif(matrix[row][col] == matrix[row][col-1]+gap_score):
        db_alignment += db_seq[col].lower()
        query_alignment += "_"
        col -=1

    else:
        query_alignment += 'X'
        db_alignment += 'X'
        row-=1
        col-=1
```

Ilustración 33 - Reconstrucción de la mejor solución de alineamiento encontrado sobre una cadena del sistema

Interfaz de usuario y presentación de resultados

Los datos deben representarse de un modo particular para poder ser interpretados por el usuario del sistema. Se mostraran inicialmente las secuencias de la base de datos ordenadas de mayor a menor score máximo detectado. Una vez pulsada sobre una de esas secuencias para ver el detalle, se mostraran las diferentes combinaciones de alineamiento resultante sobre esa secuencia ordenadas también por sus score obtenidos.

Para ir recolectando los resultados se ha utilizado como librerías los contenedores definidos por panda.DataFrames.

[Bioinformatics Website](#)

PARAMETERS:

query_seq:

k:

match_score:

mismatch_score:

gap_score:

seed_threshold:

Execution mode:

num_sequences:

Blast Run

Results:

idSecuencia	nameSecuencia	strSecuencia	scoreMaxSecuencia
1	MK617223.1 Homo sapiens isolate 205_S0 haplogroup H2a2a mtchondrion, complete genome	ATGAACGGAAATCTGTTGGCTTCATTATGCCCCCAACATAGCGCTACCCGCCGCACTGATCATTCT ATTTCCCCCTCTATTGATCCGACCTCCAAATATCTCATCAACAACCGGACTAATCCACCCCAACAAATGACTAAT CAAACTAACCTCAAACAATGATAACCATACACACACTAAAGGAGCAACCTGATCTCTATACTAGTATCC TTAATCATTTTATTGGCACAACAACCTCCTGGACTCCTGGCTACCTCATTTACAGCAACCCCAACTACT ATAAACCTAGCCATGGCCATCCCTTATGAGCGGGACAGTATTATAGGCTTTCGCTCTAGATTAATAATG CCCTAGCCCACTTTACCAAGGACACCTACACCCCTTATCCCATAGTATTATGGAACCAATCAGG CTACTATTCAACCAATAGCCCTGGCCGTAGCGCTAACCGCTAACATCTAGGAGGCACTACTCTGACCC TAATGGAAGGGCCAGCCCTAGCABATATCAGCATTAACCTTCGCTTACACTATCATCTTCACAACTTAATC TACTGACTATCTAGAAATGGCTGTGGCTTAATCCAGGCTAGGTTTCAGACTCTAGTAAGGCTTACTCTGC AGGACAACACATAA	36.0

'query_seed', 'query_index_seed', 'db_seed', 'db_index_seed', 'score_seed', 'query_alignment_extends', 'db_alignment_extends', 'row_scores'
 1 [AAATCTGTTGGCTTCA, 0, AAATCTGTTGGCTTCA, 8, 32.0, AAATCTGTTGGCTTCATT, AAATCTGTTGGCTTCATT, 36.0]
 2 [AAATCTGTTGGCTTCAT, 1, AAATCTGTTGGCTTCAT, 9, 32.0, AAATCTGTTGGCTTCATT, AAATCTGTTGGCTTCATT, 34.0]
 3 [AAATCTGTTGGCTTCATT, 2, AAATCTGTTGGCTTCATT, 10, 32.0, AAATCTGTTGGCTTCATT, AAATCTGTTGGCTTCATT, 32.0]

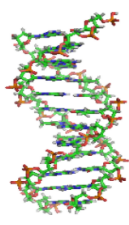


Ilustración 34 – Presentación de resultados en el proyecto - Bioinformatics Website

Estadísticas y análisis de mediciones de cómputo para la optimización

Se realizan pruebas de ejecución observando la importancia de una buena selección de los parámetros de búsqueda. Por lo que se implementa un sistema para la representación de los costes de computación en tiempo para las distintas ejecuciones guardando parámetros como numero de secuencias, longitud de la cadena, valor K, relación entre K y HSP_THRESHOLD.

Se utiliza para esto librerías de Matplotlib para representación de gráficos de representación de tiempos de ejecución de sistema según diferentes parámetros que pueden ser filtrados para ver representación de la variación de un parámetro manteniendo el resto acotados.

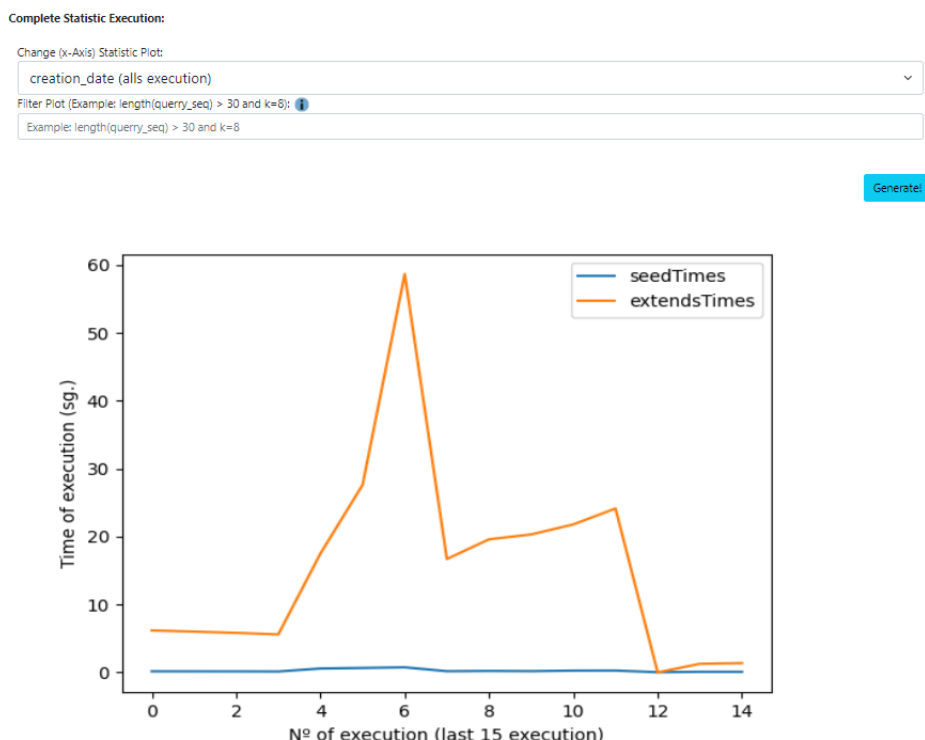


Ilustración 35 - Grafica de resultados que muestra el sistema desarrollado, mostrando de los tiempos de ejecución de una consulta, mostrando una línea el tiempo total en la búsqueda de las semillas HSP y en otra el tiempo de la extensión con huecos de todas esas semillas calculadas.

Validación y pruebas

Se superan primero los problemas de compilación asociados a fallos en la mala utilización o inicialización de algún tipo de dato o utilización de las librerías.

La etapa de validación ha consistido en realizar ejecuciones resolviendo las excepciones de error no controladas que se disparan al procesarse el algoritmo asociado a los clásicos errores en la gestión de los límites en recorridos sobre arrays.

Una vez ajustado comprobamos que funciona correctamente, realizamos ejecuciones controladas en debug para poder comprobar la matriz Smith-Waterman que genera la extensión de cierta semilla HSP y comprobar su correcta ejecución. Realizando esta tarea hemos terminado de ajustar el correcto comportamiento del algoritmo general.

Análisis algorítmico

La comparación de una cadena contra la base de datos va a depender principalmente de estos factores:

- p – número de secuencias de la base de datos
- M – longitud de las secuencias en la base de datos
- N – longitud de la cadena de consulta
- K – tamaño para el cómputo de semillas

La comparación del sistema es comparar la secuencia query contra todas las que haya en la base de datos.

El coste del algoritmo Smith-Waterman es de $O(NM)$, donde n y m son la longitud de las 2 secuencias a alinear, la cadena de consulta y la cadena de base de datos. (Vinuesa, 2007). Este coste solo representa la extensión con huecos de una semilla HSP candidata sobre la cadena de consulta sobre una cadena de la base de datos. Por lo que el coste total de la consulta es la suma del cómputo relativo a la extensión por todo el número de semillas candidatas se hayan detectado.

Gracias las técnicas aplicadas de heurística, el tiempo de ejecución del alineamiento con huecos entre dos cadenas, pasa de $O(NM)$ de la programación dinámica de algoritmo de Smith&Waterman a reducirse a $O(N+M)$, siendo N y M la longitud de las secuencias, esto es enunciado ya en

los documentos de los profesores (F. A., Dr., Oswaldo, & Allende) del centro de biotecnología molecular Severo Ochoa (CBMSO).

Retrospectiva del sprint

Este primer sprint ha llegado a su fin y ha cumplido sus objetivos, ha sido sin duda: duro y largo. En él se han implementado las funcionalidades principales para poder completar un primer paquete funcional completo sobre el sistema.

Se han detectado mediante depuración errores en la ejecución heurística del algoritmo, a los que se les ha dado solución en cada caso.

La complejidad del sprint ha superado la estimación inicial, se ha superado el tiempo previsto para el desarrollo del sprint.

Seguir haciendo	Empezar a hacer	Dejar de hacer
<ul style="list-style-type: none">- Seguir focalizándonos en requisitos concretos y abordarlos de forma completa- Realizar pruebas de validación y depuración de la funcionalidad desarrollada nos ha permitido detectar errores en la codificación.	<ul style="list-style-type: none">- Realizar una adecuada separación en base al acoplamiento y cohesión de las funcionalidades del sistema.	<ul style="list-style-type: none">- No se han detectado problemas mencionables en el sprint

Tabla 6 - Matriz de retrospectiva sprint 2

El enfoque práctico de la funcionalidad es a partir de una cadena de nucleótidos de consulta, identificar con que proteína está relacionada. Observamos en la siguiente tabla la distribución del número de cadenas por cada clasificación proteica.

<u>Gene family</u>	<u>Number</u>	<u>Class label</u>
G protein coupled receptors	531	0
Tyrosine kinase	534	1
Tyrosine phosphatase	349	2
Synthetase	672	3
Synthase	711	4
Ion channel	240	5
Transcription factor	1343	6

Tabla 7 - Clasificación de cadenas con la proteínas que está relacionada. Dataset obtenido del artículo “Working with DNA sequence data for ML part 2” en kaggle.com (Nelson, 2019)¹

En las cadenas genómicas, la codificación biológica se interpreta en las oraciones y párrafos, no en letras aisladas, sino en la interrelación entre ellas.

- Sabemos que la traducción de ADN/ARN a aminoácidos es cada 3 nucleótidos, genera 1 aminoácido en los ribosomas.
- Que hay un lenguaje de 20 aminoácidos para formar las proteínas como cadenas de cientos de aminoácidos.
- La importancia a nivel de frase o párrafo (de 20 - 50 nucleótidos) para la finalidad de encontrar una relación o identificación biológica.

Las ideas principales de la implementación de este clasificador de como el dataset utilizado en forma de cadenas de ADN están recogidas en el artículo “Working with DNA sequence data for ML part 2” en kaggle.com (Nelson, 2019). Han sido fuente principal para afrontar esta implementación.

¹ “Working with DNA sequence data for ML part 2” en kaggle.com. Ejemplo de uso de un clasificador Bayes para la clasificación de cadenas ADN. <https://www.kaggle.com/thomasonelson/working-with-dna-sequence-data-for-ml-part-2>

Estas ideas principales son:

- 1) Genera para las cadenas en la base de datos las combinaciones de kmers. (Utilizamos K=6)

```

Terminal Help neuronal_process.py - BioinformaticsWebsite - Visual Studio Code
neuronal_process.py > createClassifierMNI
114
115 seq_data.head()
116 seq_texts = list(seq_data[
117 for item in range(len(seq
118 seq_texts[item] = '
119 y_data = seq_data.iloc[:,
120
121 # Now we will apply the BA
122 # Creating the Bag of Word
123 # This is equivalent to k-
124 # The n-gram size of 4 was
125 cv = CountVectorizer(ngram
126 X = cv.fit_transform(seq_texts)
127

['atgaac tgaacg gaacga...ata acataa', 'atgccc tgcccc gcccc...cta tctag', 'atgaac tga
> special variables
> function variables
0000: 'atgaac tgaacg gaacga aacgaa acgaaa cgaaaa gaaaaa aaaatc aaatct aat
0001: 'atgccc tgcccc gcccc ccccaa cccaac ccaact caacta aactaa actaaa cta
0002: 'atgaac tgaacg gaacga aacgaa acgaaa cgaaaa gaaaaa aaaatc aaatct aat
0003: 'atgccc tgcccc gcccc ccccaa cccaac ccaact caacta aactaa actaaa cta
0004: 'atgcaa tgcaac gcaaca caacag aacagc acagca cagcat agcatt gcattt cat
0005: 'atgccc tggcct ggcctc gcctca cctcaa ctcaaa tcaaat caaatg aaatga aat
0006: 'atgtgt tgtgtg gtgtgg tgtggc gtggca tggcat ggcatt gcattt catttg att
0007: 'atgtgt tgtgtg gtgtgg tgtggc gtggca tggcat ggcatt gcattt catttg att
0008: 'atgtgt tgtgtg gtgtgg tgtggc gtggca tggcat ggcatt gcattt catttg att
0009: 'atgagg tgaggc gaggcc aggccc ggcccc gcccga cccgag ccgagc cgagcg gag
Hold Alt key to switch to editor language hover

```

Ilustración 37 - Cada cadena de la base de datos se descompone en la combinaciones de los kmers que da lugar

- 2) Por otro lado tenemos la clasificación de proteína de cada cadena o lista de kmers a los que da lugar.

```

16 seq_texts = list(seq_data['words'])
17 for item in range(len(seq_texts)):
18 seq_texts[item] = '
19 y_data = seq_data.iloc[
array([4, 4, 4, ..., 6, 6, 6], dtype=int64)
> special variables
> [0:6061] : [4, 4, 4, 4, 3, 6, 3, 3, 0, 0, 1, 2, 2, ...]
> dtype: dtype('int64')
> max: 6
> min: 0
> shape: (6061,)
size: 6061
Hold Alt key to switch to editor language hover
28 print("y_data: " + y_data)
29 print(X.shape)
30
31 # Splitting the human dataset into the training set and test set

```

Ilustración 38 - La imagen muestra la clasificación de las cadenas en base de datos.

- 3) En base a esto, creamos un contador de palabras CountVectorizer (Bag of Words), pero haremos un uso especial particular del mismo para aportar contexto entre las palabras.

Si pesamos en un clasificado para la detección de email como spam, podemos extrapolar la idea para contar grupos de palabras en lugar de

palabras. Por ejemplo, si aprendemos que cuando aparecen las palabras: win/won, movie, downloads: clasificamos el email como spam, ya tenemos algo que “más o menos” funciona...

Pero... ¿y si el mensaje pone?:

*“Hey, I **won**, I choose!*

*This afternoon we are going to the cinema to see the **movie***

*Here is the link to **downloads** the tickets”*

En este caso no estamos ante un correo SPAM, aunque tenga palabras asociadas a mensajes de SPAM...

La solución a este problema es el contexto: el sistema debe aportar contexto a las palabras. Para ello contamos si aparecen las palabras en un contexto y no de forma aislada.

*“**Free downloads movie**”*

*“game free to play, **now available downloads**”*

*“Your PC has a **virus, download antivirus**”*

Por este motivo, el contador de palabra que utilizaremos será un contador de 3 palabras de 6-mers en cada caso. Podemos así contar grupos de 3 palabras Kmers que aparecen juntos en las cadenas y clasificar con mejor precisión. Y generar a partir de ahí los vectores de entrenamiento.

```
# Now we will apply the BAG of WORDS using CountVectorizer using NLP
# Creating the Bag of Words model using CountVectorizer()
# This is equivalent to k-mer counting
# The n-gram size of 3 was previously determined by testing
# Generate array of sentences for each 3 kmers
cv = CountVectorizer(ngram_range=(3,3))
X = cv.fit_transform(seq_texts)
```

Ilustración 39 – Conunt Vectoricer for sequences of 4 words-Kmers

Esta estrategia es necesaria para que el clasificador responda mejor a variaciones sobre las cadenas, huecos o variaciones con cadenas fuera de la

colección de entrenamiento. El count vectorizer nos generara un array de diccionario por grupos de 3 palabras en lugar de palabras aisladas, de esta forma damos más peso a la interrelación entre nucleótidos próximos en el contexto para la formación de aminoácidos de la proteína esperada.

Ilustración 40 - Desarrollo de la bolsa de palabras de 3 palabras de 6 nucleótidos.

- 4) Generar vectores de entrenamiento. para el conjunto de datos manejado (6061 secuencias), el vector de datos X toma la forma de (6061, 65570), donde 65.570 el número de frases únicas encontradas en el texto. De esta forma tener altas ocurrencias en secuencias de 3 kmers y cuanto más alto sea su aproximación sobre el lenguaje a una cadena, mayor será si índice probabilístico.

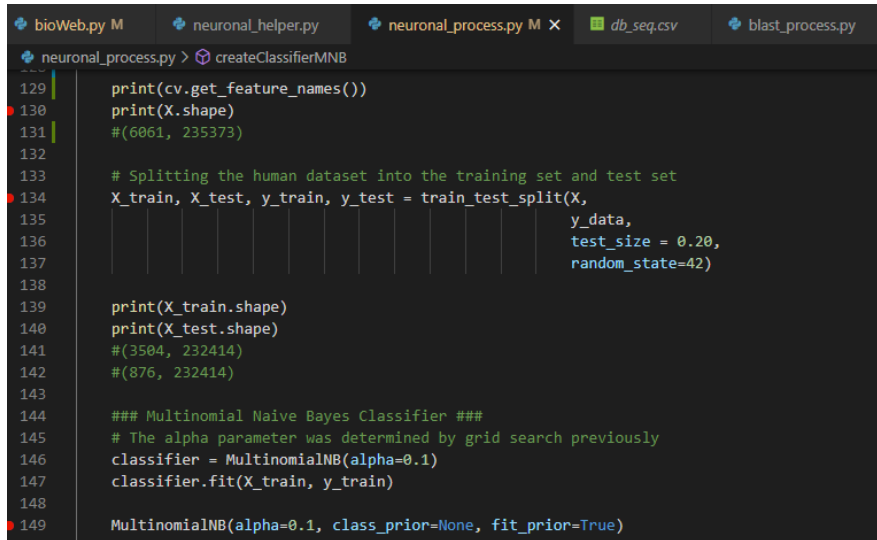
Por tanto, los vectores X de entrenamiento y de test serán de la forma (n, 65570), indicando para cada posición de cadena, el número de veces que aparece cierta palabra en la secuencia completa, y asociado a una clasificación proteica sobre los vectores Y.

- 5) En base a este vector X de entrenamiento generado con la forma de contador de palabras (ngram_range=(3,3)), entrenaremos el clasificador Bayes de probabilidad condicionada. Utilizaremos un clasificador sklearn.naive_bayes.MultinomialNB. El clasificador irá aprendiendo a identificar el mensaje biológico procesando la lista de

Herramienta para el estudio del alineamiento de secuencias de ADN mediante DeepLearning

3words/6mers de cada secuencia a la clasificación que lleve asociada en los metadatos la cadena.

Esquema de código para el pre-procesamiento de los datos iniciales y la ejecución del entrenamiento.



```
neural_process.py > createClassifierMNB
129 |     print(cv.get_feature_names())
130 |     print(X.shape)
131 |     #(6061, 235373)
132 |
133 |     # Splitting the human dataset into the training set and test set
134 |     X_train, X_test, y_train, y_test = train_test_split(X,
135 |                                                       y_data,
136 |                                                       test_size = 0.20,
137 |                                                       random_state=42)
138 |
139 |     print(X_train.shape)
140 |     print(X_test.shape)
141 |     #(3504, 232414)
142 |     #(876, 232414)
143 |
144 |     ### Multinomial Naive Bayes Classifier ###
145 |     # The alpha parameter was determined by grid search previously
146 |     classifier = MultinomialNB(alpha=0.1)
147 |     classifier.fit(X_train, y_train)
148 |
149 |     MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
```

Ilustración 41 - Clasificador Bayes de sklearn.naive_bayes.MultinomialNB

- 6) Una vez el clasificador está entrenado, podrá clasificar haciendo el proceso de:
 - a. Descomponer en kmers (K=6, según el ejemplo desarrollado)
 - b. Contar la bolsa de “frases” de 3 palabras y 6 nucleótidos contenida en la cadena de consulta sobre todo el diccionario identificado por el contador en el entrenamiento previo. Así a cadena de entrada es convertirla en un array de contadores de 3-WORD sobre el lenguaje definido por el CountVectorizer, y tendrá la forma (1, 65570).
 - c. Una vez generado el array de entrada al clasificador podremos saber a qué tipo de clasificación proteica pertenece la cadena.

Esquema de código para la consulta sobre un clasificador entrenado.

```

def clasiffierMNB(cbp, cv, classifier):

    seq_texts = list(getKmers(cbp.query_seq,6))
    seq_texts = ' '.join(seq_texts)

    seq = list()
    seq.append(seq_texts)

    # encode document
    X_seq = cv.transform(seq)

    print(X_seq.shape)
    # clasification
    y_pred = classifier.predict(X_seq)
    print("Predictions x_test: ", str(y_pred[0]))
    
```

Ilustración 42 - Fragmento de código para la transformación de la cadena de consulta al array de entrada al clasificador y su predicción.

Eficiencia del entrenamiento

Tras el entrenamiento y test obtenemos estos datos:

Matriz de confusión

Predicted	0	1	2	3	4	5	6
0	164	0	0	0	1	0	2
1	0	144	0	0	0	0	6
2	0	0	105	0	0	0	4
3	1	1	0	164	1	0	0
4	0	0	0	0	195	0	2
5	0	0	0	0	0	71	1
6	0	0	0	2	0	0	349

Tabla 8- Matriz de confusión del sistema Clasificador Naives Bayes

precisión = 98.30808426399021 %
 time_fit = 14.602660179138184 sg.
 time_test = 0.04393506050109863 sg.

Vemos como tanto la matriz de confusión, el índice de precisión, así como los tiempos de entrenamiento y test son espectaculares.

Predicción y tiempo de consulta

Realizamos consultas con cadenas aleatorias, tomando variaciones de las cadenas que fueron base del entrenamiento. Quitarle pequeñas secuencias o variaciones de nucleótidos.

El sistema responde con acierto a esas variaciones, mostrando como resultado la proteína de la que hemos generado la variación para la consulta.

Resultados:

The sequence is identified with the protein class: 6

Time to prediction: 0.005034446716308594 sg.

El tiempo de consulta es de apenas de 0.005 segundos para identificar la proteína con la que se relaciona una cadena conocida. Comparando con el método de alineamiento heurístico es tremendamente más eficiente. En una estrategia heurística aunque hay muchos factores que pueden influir, en la etapa de análisis de eficiencia del modelo, habíamos obtenido tiempos de consulta de en torno a 10 – 20 segundos para calcular los alineamientos de una cadena contra 10 cadenas o secuencias almacenadas en base de datos de longitud variable de 2000 a 4000 signos. Pues todavía este alineamiento no sería suficiente, puesto que para la comprobación de la relación entre nucleótidos y proteínas es necesaria una traducción correctamente alineada entre tokens de 3 símbolos de la cadena y el aminoácido que codifica.

Retrospectiva del sprint

Los objetivos del sprint se han alcanzado. La etapa más costosa ha sido el análisis y pre-procesamiento de los datos de forma que pudiéramos tratarlo con un sistema clasificador alcanzando resultados esperados. Es probable que no se hayan realizado todos los esfuerzos de análisis y diseño en la etapa de diseño de los requisitos del sprint, y esto nos ha llevado a realizar varias implementaciones en el pre-procesamiento de los datos hasta obtener el pre-

procesamiento en forma de kmer y de bolsa de palabras para que el sistema se comporte según el grado de eficacia esperado.

Seguir haciendo	Empezar a hacer	Dejar de hacer
<ul style="list-style-type: none"> - Seguir focalizándonos en requisitos concretos y abordarlos de forma completa 	<ul style="list-style-type: none"> - Realizar un adecuado análisis de la forma de los datos para la implementación de un sistema neuronal. 	<ul style="list-style-type: none"> - No se han detectado problemas mencionables en el sprint

Tabla 9 - Matriz de retrospectiva sprint 3

7.3.4 CUARTO SPRINT: Sistema aprendizaje LSTM

Para implementar esta funcionalidad utilizaremos una clasificar LSTM. Vamos a utilizar redes recurrentes, que permiten leer secuencialmente un mensaje, incorporando como técnica la disponibilidad de memoria de contexto para analizar el mensaje palabra a palabra y darle un significado o clasificación. Este tipo de redes neuronales son utilizadas por ejemplo para el procesamiento del lenguaje natural.

Clasificador mediante Red Neuronal LSTM:

Observamos el comportamiento secuencial sobre el siguiente diafragma.

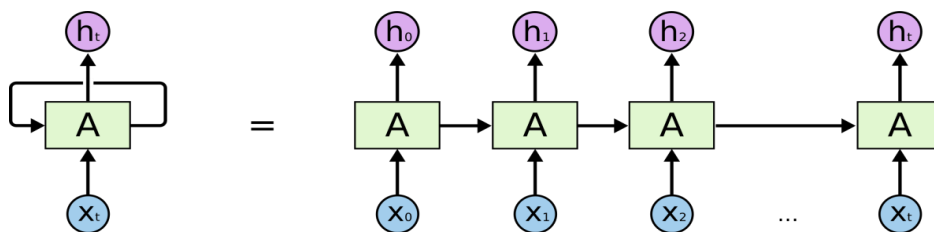


Ilustración 43 - Arquitectura de una red neuronal LSTM. Imagen tomada de: "Understanding LSTM Network". (Christopher, 2015)

El modelo implementado tomas las ideas de esta publicación de Nikoay Oskolkov, "LSTM to Detect Neanderthal DNA", en la que utiliza una red

- 2) Tokenizar cada kmers y generar un diccionario de palabras. A cada combinación de nucleótidos única, se le asignara un id de palabra en el lenguaje.

$$4^K = n^{\circ} \text{ de Palabras maximo del diccionario}$$

$$4^6 = 4096 \text{ Palabras unicas}$$

Ecuación 3 - Numero de palabras máximo del diccionario para un lenguaje de 4 letras y palabras de 6 caracteres.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(seq_texts) # tokenizer the word in each secuencia
encoded_docs = tokenizer.texts_to_sequences(seq_texts) # Transform unique each token in a integer value
max_length = max([len(s) for s in encoded_docs]) # 135 max length of all secuencias
X = pad_sequences(encoded_docs, maxlen = max_length, padding = 'post') # the context is determinate in less 100 nucleotid
```

Ilustración 45 - Uso del Tokenizer para generar el diccionario de palabras


```
tokenizer.word_index:
> special variables
> function variables
'ctgctg': 1
'gctggc': 2
'tgctgc': 3
'aaaaaa': 4
'gctgct': 5
'gcgctg': 6
'ctggcg': 7
'gccagc': 8
'cagcag': 9
'cggcgg': 10
'cgctgg': 11
'gctggt': 12
'gcggcg': 13
'tgctgt': 14
'cgccgc': 15
'tgatga': 16
'gccgcc': 17
'agctgg': 18
'cttttg': 19
'ttttgt': 20
'gctggt': 21
'ttgctg': 22
'ggcggc': 23
'tgctgg': 24
```

Ilustración 46 - Identificadores de palabra en el diccionario generado por el Tokenizer.

El juego de datos nos genera un tamaño de diccionario de 4092, incluida palabra con id=0 utilizada como relleno.

```
vocab_size = 4091 + 1
```

La red LSTM toma como índice para la recurrencia, el tamaño de la frase más larga en palabras. En el dataset preparado la cadena más larga, tiene 135 palabras (kmers). Por esto, todos los array de palabras de entrenamiento, test o las que se utilicen al realizar consultas posteriores al sistema, deberán

rellenarse id=0 hasta tener 135 palabras. Es importante calcular este dato de toda la cadena de entrenamiento, para aportárselo a la función de relleno `keras.preprocessing.sequence.pad_sequences` aportándolo por como parámetro.

```

v X: array([[ 374,  75,  92, ...,  0,  0,  0],
> special variables
v [0:1245] : [array([ 374,  75, ..., 0,  0]), array([ 171, 339, ...55, 186...
> special variables
> function variables
> 000: array([ 374,  75,  92, 143, 632, 593, 2250, 1576,  0,  0,  0,
> 001: array([ 171, 339, 798, 594, 355, 184, 633, 198, 1391, 1859, 3023,
> 002: array([ 310, 596, 1662, 1580, 1118, 1792, 1452, 2724, 3026, 2326, 1216,
> 003: array([ 443, 357, 173, 920, 1220, 247, 131, 2254, 2089, 3832, 3941,

```

Ilustración 47 - Formato del array de entrenamiento una vez convertido a array de palabras y aplicado relleno para unificar longitudes.

Así los array de entrenamiento y test tienen la forma:

```

X_train.shape = (996, 135)
X_test.shape = (249, 135)

```

Donde 135 como decíamos es el máximo tamaño en palabras codificadas con un identificadores única para cada combinación de 6-mer de la base de datos. Por otro lado, 996 y 249 son las muestras del conjunto de entrenamiento y test respectivamente.

```

model = Sequential()
model.add(Embedding(vocab_size, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

```

Ilustración 48 - Codificación de la red neuronal LSTM

La arquitectura de la red es la siguiente:

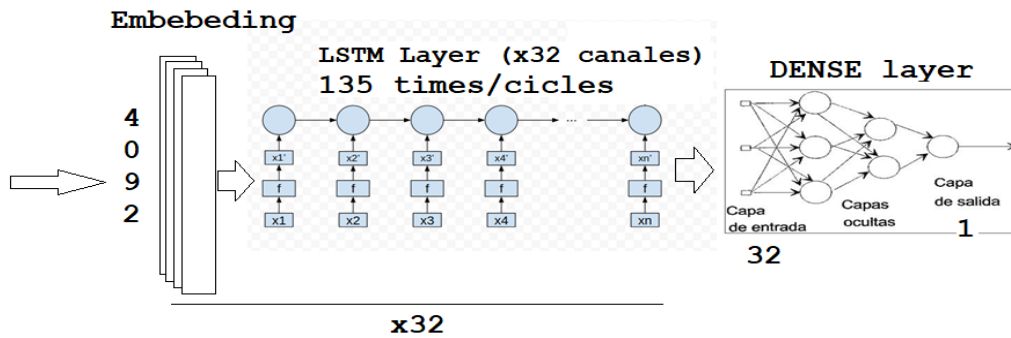


Ilustración 49 - Esquema de la arquitectura de la red neuronal LSTM para identificación de un patógeno.

En esta arquitectura la cadena de entrada es traducida por el tokenizador a un array ids de palabras.

La capa de incrustación, tal y como define en el artículo “CoreML with GloVe Word Embedding and Recursive Neural Network — part 2” (Jacopo Mangiavacchi, 2018) nos permite pasar la capa de entrada de nuestro modelo, a un nuevo vector asignando automáticamente a cada palabra a una representación vectorial que será la que se utilice durante en el proceso de aprendizaje. El tamaño en neuronas de la capa Embedding (incrustación) en el proyecto es de $4092 \times 32 = 130.944$ neuronas.

Podemos ver en la ilustración el tamaño y forma de la red neuronal. Por ejemplo observamos que la red multicapa (Dense) es de 32 neuronas de entrada, y 1 neurona de salida.

```

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
embedding (Embedding)       (None, None, 32)         130944
lstm (LSTM)                  (None, 32)                8320
dense (Dense)                (None, 1)                 33
-----
Total params: 139,297
Trainable params: 139,297
Non-trainable params: 0
    
```

Ilustración 50 - Arquitectura de la red neuronal basada en LSTM para la identificación de cadenas con un patógeno conocido.

Observamos en la ilustración el fragmento de código para la consulta sobre el sistema LSTM ya entrenado.

```

def clasiffierLSTM(cbp, tokenizer, model):

    seq_texts = list(getKmers(cbp.query_seq,6))
    seq_texts = ' '.join(seq_texts)

    seq = list()
    seq.append(seq_texts)

    # encode document
    X_seq = tokenizer.texts_to_sequences(seq)

    y_pred = model.predict_classes(X_seq)
    print("Predictions x_test: ", str(y_pred[0]))

    if y_pred[0]==1:
        return "Identificate pathogen: SARS-CoV-2"
    else:
        return "No identificate pathogen SARS-CoV-2"
    
```

Ilustración 51 - Código para la consulta de una cadena al clasificador LSTM

Eficiencia del entrenamiento

Durante la fase de test se ha obtenido una precisión del 92'77% a las muestras utilizadas para la validación del sistema. Convirtiendo la red LTSM en Bidirectional (LSTM), obtenemos una eficiencia en test del 95'98%. Si que la capa LSTM pasa al doble de neuronas: 16.640 y duplica también las neuronas de la capa Dense posterior.

La salida a la consulta del sistema es identificar si una cadena pertenece o no a un patógeno conocido para el que la red neuronal LSTM ha sido entrenado para su detección.

Matriz de confusión LSTM

	0	1
0	121	9
1	9	110

accuracy = 92.77108311653137 %

time_fit = 10.17677617073059 sg.

time_test = 0.3860020637512207 sg.

Matriz de confusión Bidireccional

	0	1
0	120	0
1	10	119

accuracy = 95.98393440246582 %

time_fit = 10.306645631790161 sg.

time_test = 0.5740888118743896 sg.

Tabla 10 - Resultados de precisión y clasificación del modelo de red neuronal LSTM implementado.

Predicción y tiempo de consulta

Tantos los tiempos de consulta como los resultados obtenidos de clasificación probándolo son también satisfactorios tanto en tiempo de ejecución como en eficacia.

Resultados:

Identificate pathogen: SARS-CoV-2 | Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1(complete genome)

0.3061807155609131 sg.

El tiempo de consulta es de apenas de 0.3 segundos para identificar una cadena como patógeno conocido. Comparando con el método de alineamiento heurístico es tremendamente más eficiente. En una estrategia heurística aunque hay muchos factores que pueden influir, en la etapa de análisis de eficiencia del modelo, habíamos obtenido tiempos de consulta de en torno a 10 – 20 segundos para calcular los alineamientos de una cadena contra 10 cadenas o secuencias almacenadas en base de datos de longitud variable de 2000 a 4000 signos.

Retrospectiva del sprint

Los objetivos del sprint se han alcanzado. En esta ocasión hemos realizado mayores esfuerzos de análisis y diseño inicial de la red neuronal, y nos ha permitido conseguir una implementación eficaz y eficiente sin realizar tantas pruebas en la etapa de codificación y pruebas.

Seguir haciendo	Empezar a hacer	Dejar de hacer
<ul style="list-style-type: none"> - Seguir focalizándonos en requisitos concretos y abordarlos de forma completa - Realizar un adecuado análisis de la forma de los datos para la implementación de un sistema neuronal. 	<ul style="list-style-type: none"> - No se han detectado nuevas estrategias mencionables en el sprint 	<ul style="list-style-type: none"> - No se han detectado problemas mencionables en el sprint

Tabla 11 - Matriz de retrospectiva sprint 4

8 CONCLUSIONES

8.1 Objetivos alcanzados

El objetivo principal del proyecto se ha alcanzado. El objetivo principal era el estudio técnico e implementación de un sistema de alineamiento de secuencias genómicas basado en sistemas heurísticos y en arquitecturas de deep learning. Ambos objetivos se han cumplido.

Poniendo el foco en el análisis de los objetivos secundarios marcados, comprobamos los resultados obtenidos:

- 1) El sistema aporta valor funcional para el estudio del alineamiento de secuencias biológicas como nos proponíamos en el proyecto.
- 2) Se implementa un sistema heurístico computacionalmente eficiente para bases de datos genómicas.
- 3) Se implementa un sistema de clasificación genómica mediante
- 4) El proyecto implementa una interfaz web dispuesta a ser utilizada por un entorno de laboratorio.

El sistema se ha implementado como era uno de los objetivos del proyecto sobre una interfaz web HTML sobre un servidor HTTP dispuesto a ser “utilizado” en un entorno de laboratorio para consultar, ajustar o ampliar funcionalidades a problemas concretos en biotecnología.

8.2 Conclusiones del trabajo y personales

La valoración principal tras el proyecto, es el aprendizaje obtenido durante el desarrollo del proyecto.

La conclusión principal del proyecto, son los beneficios que aporta una estrategia basada en deep learning a problemática concretas en biotecnología.

Los resultados y los tiempos de ejecución son realmente bueno en un entorno de datasets “grande”. La flexibilidad de los sistemas de deep learning, es que sin tocar grandes algoritmos por código y realizando mínimos ajustes a las arquitecturas, se puede aprender a resolver nuevos problemas, ya que el problema donde está definido es en el juego de datos utilizados durante el entrenamiento.

Competitivamente el enfoque basado en deep learning ofrece una serie de mejoras frente a la programación tradicional de algoritmos.

- La flexibilidad, ligereza y facilidad para la programación o adaptación a diferentes problemas.
- El menor tiempo de desarrollo y validación.
- Con una buena definición, se alcanzan índices de acierto superiores torno al 90% de existo en problemas de secuencias en bioinformática.
- Los tiempos de entrenamiento y ejecución en los sistemas de deep learning desarrollados son sorprendentemente eficientes incluso con una simple arquitectura de computación doméstica.

8.3 Vías futuras

La vías futuras que se nos presentan en la utilización de la tecnología que se expone y utiliza en este trabajo, están abiertas a implementaciones tanto en sistemas de laboratorio e investigación, como en sistemas médicos, para el diagnosis a problemas como: la identificación de cadenas de virus en muestras, correlación y diagnosis de patologías genómicas, estudios de compatibilidad genética, y otros estudios clínicos relacionados del contenido del mensaje genético en nuestras células.

Para cubrir los distintos objetivos, la caja de herramientas de deep learning vemos como nos ofrece una alternativa a la algorítmica tradicional, y nos ofrece distintas arquitecturas y modelos. En este trabajo se han utilizado 2 modelos de arquitectura, pero pueden utilizarse otros modelos de redes

profundas como algunos de los mencionados: redes profundas, secuenciales, convolucionales o recurrentes. Cada problema y sus objetivos propuestos, requerirá un diseño que se adapte a la naturaleza del problema para ofrecer los resultados esperados.

9 REFERENCIAS BIBLIOGRÁFICAS

- ADN-acido-Desoxirribonucleico.* (s.f.). Obtenido de genome.gov:
<https://www.genome.gov/es/genetics-glossary/ADN-acido-Desoxirribonucleico>
- Arias, J. A., Bahón, M. d., & Rodríguez, C. G. (2006). *Desarrollo de una plataforma de análisis de datos en Bioinformática basada en Matlab.* Obtenido de Universidad Complutense:
<https://eprints.ucm.es/id/eprint/9056/>
- C. D., P. L., A. N., & R. T. (2014). *Computational Intelligence Methods for Bioinformatics and Biostatistics.* Cambridge.
- C. O. (2015). *Understanding LSTM Networks.* Obtenido de colah's blog:
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Chin, J. (2018). *DCNet — Denoising (DNA) Sequence With a LSTM-RNN and PyTorch.* Obtenido de medium.com: <https://infoecho.medium.com/dcnet-denoising-dna-sequence-with-a-lstm-rnn-and-pytorch-3b454ff727e7>
- Diego Calvo . (2017). <https://www.diegocalvo.es/red-neuronal-convolucional/>. Obtenido de <https://www.diegocalvo.es/red-neuronal-convolucional/>: <https://www.diegocalvo.es/red-neuronal-convolucional/>
- F. A., D. P.-P., O. T., & Allende, R. A. (s.f.). *Alineamiento de secuencias. Búsqueda de parecidos. Alineamientos múltiples.* Obtenido de Centro de Biología Molecular "Severo Ochoa" (CBMSO, CSIC-UAM): http://bioweb.cbm.uam.es/courses/MasterVirol2013/alignment/Alineamiento_secuencias/teoria.html#loc-glob
- F. A., Puertas, P. G., O. T., & Allende, R. A. (s.f.). *Alineamiento de secuencias. Búsqueda de parecidos. Alineamientos múltiples.* Obtenido de Centro de Biología Molecular "Severo Ochoa": http://bioweb.cbm.uam.es/courses/MasterVirol2013/alignment/Alineamiento_secuencias/teoria.html
- Flask.* (s.f.). Obtenido de <https://flask.palletsprojects.com/en/1.1.x/>
- FRANCIS CRICK. (1970). Dogma central de la biología molecular. *Nature.*

- G. G. (s.f.). *Algoritmo Smith-Waterman*. Obtenido de <http://www.bioinformaticos.com.ar/>:
<http://www.bioinformaticos.com.ar/algoritmo-smith-waterman/>
- Garrigues, F. (2017). *10 Conceptos básicos para introducirte en el mundo de la Genética*. Obtenido de El Blog de Genotipia: <https://genotipia.com/conceptos-genetica/>
- Gunavaran Brihadiswaran. (2020). *Bioinformatics 1: K-mer Counting*. Obtenido de medium.com: <https://medium.com/swlh/bioinformatics-1-k-mer-counting-8c1283a07e29>
- Ian Korf, Mark Yandell, & Joseph Bedell. (2003). *BLAST*. O'Reilly Media, Inc.
- Institute, P. M. (1996). *Guía de los fundamentos para la dirección de proyectos*. Institute, P. M. (2017). *PMBOK - A Guide to the Project Management Body of Knowledge (version 6)*. Pensilvania , USA: PMI.
- Introducción a la bioinformática*. (s.f.). Obtenido de Universidad Politecnica de Valencia: https://bioinf.comav.upv.es/courses/intro_bioinf/index.html
- J. T. (2013). *Algoritmos en Bioinformática- Algoritmo de Smith. Waterman - Blast*. Cali: Universidad del Valle.
- J. T. (2019). *Deep Learning, Introducción práctica con Keras*. Watch This Space.
- Jacopo Mangiavacchi. (2018). *CoreML with GloVe Word Embedding and Recursive Neural Network — (part 2)*. Obtenido de medium.com: <https://medium.com/@JMangia/coreml-with-glove-word-embedding-and-recursive-neural-network-part-2-ab238ca90970>
- James D. Watson. (2006). *Biología molecular del gen*. Ed. Médica Panamericana.
- Kun-Mao Chao, & Louxin Zhang. (2008). *Sequence Comparison: Theory and Methods*. Springer Science & Business Media.
- Lesk, A. (2008). *Introduction to bioinformatics, 3rd Ed*. Oxford University Press.
- María Jesús de la Fuente Aparicio, & Teodoro Calonge. (1999). *Aplicaciones de las redes de neuronas en supervisión, diagnóstico y control de procesos*. Equinoccio.
- N. O. (2019). *LSTM to Detect Neanderthal DNA*. Obtenido de towards data science: <https://towardsdatascience.com/lstm-to-detect-neanderthal-dna-843df7e85743>

- NCBI. (s.f.). *BLAST*. Obtenido de BLAST: -
<https://blast.ncbi.nlm.nih.gov/Blast.cgi>
- Nelson, T. (2019). *Working with DNA sequence data for ML part 2*. Obtenido de
<https://www.kaggle.com/>:
<https://www.kaggle.com/thomasnelson/working-with-dna-sequence-data-for-ml-part-2>
- NIH. (s.f.). *Transcripción*. Obtenido de NATIONAL INSTITUTE OF CÁNCER:
<https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/transcripcion>
- Pramod Singh, & Avinash Manure. (2019). *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*. Apress.
- Prof. Yechiam Yemini. (2007). *Searching Sequence Databases; FASTA, BLAST*. Obtenido de Computer Science Department - Columbia University:
<http://web2.cs.columbia.edu/4761/notes07/chapter2.3-sequenceDB.pdf>
- Raisman, D. J., & Gonzalez, D. A. (s.f.). *Proteínas: de la estructura primaria a la cuaternaria*. Obtenido de HIPERTEXTOS DEL ÁREA DE LA BIOLOGÍA :
<http://www.biologia.edu.ar/macromoleculas/structup.htm>
- Rodríguez, S. S. (2018). *LSTM, ¿qué son y cuándo se usan?* Obtenido de slideshare.net: <https://es.slideshare.net/PlainConcepts/lstm-qu-son-y-cundo-se-usan>
- Roman, V. (2019). *Algoritmos Naive Bayes: Fundamentos e Implementación*. Obtenido de medium.com: <https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f>
- Russell, R. (2018). *Deep Learning: Fundamentos Del Aprendizaje Profundo Para Principiantes*. CreateSpace Independent Publishing Platform.
- Sales, J. C., J. B., & P. Z. (s.f.). *Bioinformática básica*. Obtenido de Bioinformatics at COMAV - Universidad Politecnica de Valencia:
https://bioinf.comav.upv.es/courses/intro_bioinf/index.html
- SCRUM- ROLES, ARTIFACTS AND CEREMONIES*. (2019). Obtenido de <https://www.apeironsoftware.com/scrum-roles-artifacts-and-ceremonies/>
- Shi, G. (2019). *“Deep learning” about transcription factor-DNA binding*. Obtenido de <https://towardsdatascience.com/>:

<https://towardsdatascience.com/deep-learning-about-transcription-factor-dna-binding-1d9753eabcc2>

Torres, J. (2018). *Deep learning: Introducción práctica con Keras*. Lulu.

Torres, J. (2020). *Python deep learning*. Marcombo.

Vinuesa, P. (2007). *Algoritmos de Programación Dinámica para alineamientos globales y locales*. Obtenido de BIOLOGÍA FILOGENÉTICA Y EVOLUTIVA, UNAM - UAEM:
https://www.ccg.unam.mx/~vinuesa/Cursos2RMBF/PDFs/C1/Material_su pl1_programacion_dinamica.pdf

10 ANEXOS

10.1 Manual de instalación.

Para utilizar el proyecto en una instalación local es necesario instalar el framework y librerías del proyecto. La instalación de todas estas librerías están descritas en los puntos del apartado:

- Visual Studio Code¹, como IDE o entorno de programación.
- Instalar Python 3.7² y gestor de paquetes python-pip
- Instalar Flask³

```
pip install Flask
```

Para la capa de servidor, necesitaremos las siguientes librerías. Es aquí donde implementaremos los procesos de alineamiento y deep learning instalaremos las siguientes librerías.

- Instalar Tensorflow⁴ y Keras⁵

```
pip install tensorflow
```

```
pip install keras
```

- Instalar Pandas⁶

```
pip install pandas
```

¹ <https://visualstudio.microsoft.com/es/downloads/>

² <https://www.python.org/downloads/>

³ <https://flask.palletsprojects.com/en/1.1.x/>

⁴ <https://www.tensorflow.org/install/pip?hl=es-419>

⁵ <https://pypi.org/project/keras/>

⁶ <https://pypi.org/project/pandas/>


```
> C:\Python37\python.exe .\BioinformaticsWebsite\bioWeb.py
```

Mostrándonos cuando se ha levantado el servicio.

```
Running service ...
* Serving Flask app "bioWeb" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in
a production deployment.
Running service ...
* Debugger is active!
* Debugger PIN: 167-584-505
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Pudiendo acceder al sistema desde el navegador a través de la URL
<http://localhost:5000/>.

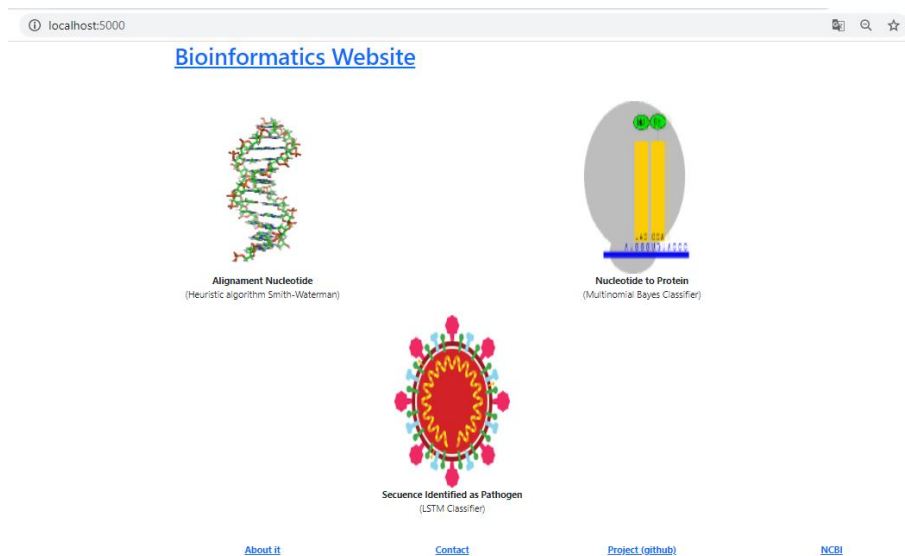


Ilustración 53 - Aspecto del main de sistema Bioinformatics Website

10.2 Manual de uso.

Se requiere realizar previamente las tareas descritas en el anexo
“Manual de instalación”.

Se han implementado 3 funcionalidades para una cadena de consulta: comprobación del alineamiento, identificación de proteínas e identificación como patógeno.

10.2.1 Alineamiento

En este caso, introduciremos una cadena de consulta y los parámetros para la ejecución de la consulta.

Parámetros de ejecución:

QUERY: cadena que se desea consultar contra la base de datos genética.

K: tamaño de la búsqueda inicial para la identificación de semillas

MATCH_SCORE: Score del alineamiento correcto de un símbolo de la cadena.

MISMATCH_SCORE: Score del alineamiento incorrecto de un símbolo de la cadena.

GAP_SCORE: Score del alineamiento al computar un carácter de gap en la cadena.

SEED_THRESHOLD: Umbral mínimo que debe cumplir una semilla de tamaño K para ser extendido su cálculo de alineamiento.

En base a estos, se comprobaba esa cadena contra el resto de cadenas en el sistema calculando en cada caso los alineamientos con huecos de mayor puntuación.

localhost:5000/blast_nucleotide

Bioinformatics Website

PARAMETERS:

query_seq:

k:

match_score:

mismatch_score:

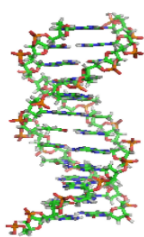
gap_score:

seed_threshold:

Execution mode:

num_sequences:

[Blast Run](#)



[About Us](#) [Contact](#) [Project Github](#) [NCBI](#)

Herramienta para el estudio del alineamiento de secuencias de ADN mediante DeepLearning

Una vez ejecutado se muestran los resultados del alineamiento, y el score obtenido en cada zona de alineamiento.

Results:

	idSecuencia	nameSecuencia	strSecuencia	scoreMaxSecuencia
▶ 1	MK617223.1	Homo sapiens isolate 205_Sb haplogroup H2a2a mitochondrion, complete genome	ATGAACGAAAATCTGTTGCTTCATTATGCCCCACAATCTAGGCTACCCGCCGAGTACTGATCATTCT ATTCCCCCTCTATTGATCCCCACCTCCAAATATCTCATCAACAACGGACTAATCACCCCAACAATGACTAAT CAAACCTAACCTCAAAAACAATGATAACCATACACAACACTAAAGGACGAACTGATCTCTATACTAGTATCC TTAATCATTTTTATTGCCACAACCTAACCTCTCGGACTCTGCCTCACTCATTACACCAACCCCAACTATCT ATAAACCTAGCCATGGCCATCCCTTATGAGCGGGCACAGTATTATAGGCTTTCGCTCTAAGATTAATAATG CCCTAGCCCACTCTTACCCACAAGGCACACCTACACCCCTTATCCCCATACTAGTTATTATCGAAACCATCAGC CTACTCATTCAACCAATAGCCCTGGCCGTAGCCCTAACCCGTAACATTACTGCAGGCCACCTACTCATGCACC TAATTGGAAAGCGCCACCCTAGCAATATCAACCATTAACTTCCCTCTACACTTATCATCTTCACAATTCTAATTC TACTGACTATCTAGAAATCGTGTGCGCTTAATCCAAGCCTACGTTTTCACACTTCTAGTAAGCCTCTACCTGC ACGACAACACATAA	36,0
▶ 2	MW389273.1	"Homo sapiens haplogroup U8b1b1 mitochondrion, complete genome"	ATGCCCCAACTAAATACTACCGTATGGCCCAACATAATTACCCCTACTCCTTACACTATTCTCATCACCCAA CTAAAAATATAAACACAACTACCACCTACCTCCCTCACCACCAAGCCATAAAAAATAAAAAATTATAACAAAC CCTGAGAACCAAAATGAACGAAAATCTGTTGCTTCATTATGCCCCACAATCTAG	36,0
'query_seed', 'query_index_seed', 'db_seed', 'db_index_seed', 'score_seed', 'query_alignment_extends', 'db_alignment_extends', 'row_scores'				
1 ['AAATCTGT', 0, 'AAATCTGT', 169, 16.0, 'AAATCTGTTGCTTCATT', 'AAATCTGTTGCTTCATT', 36.0]				
2 ['AATCTGTT', 1, 'AATCTGTT', 170, 16.0, 'AATCTGTTGCTTCATT', 'AATCTGTTGCTTCATT', 34.0]				
3 ['ATCTGTTC', 2, 'ATCTGTTC', 171, 16.0, 'ATCTGTGCTTCATT', 'ATCTGTGCTTCATT', 32.0]				
4 ['TCTGTTGC', 3, 'TCTGTTGC', 172, 16.0, 'TCTGTTGCTTCATT', 'TCTGTTGCTTCATT', 30.0]				
5 ['CTGTTGCG', 4, 'CTGTTGCG', 173, 16.0, 'CTGTTGCTTCATT', 'CTGTTGCTTCATT', 28.0]				
6 ['TGTTCGCT', 5, 'TGTTCGCT', 174, 16.0, 'TGTTCGCTTCATT', 'TGTTCGCTTCATT', 26.0]				
7 ['GTTGCTTC', 6, 'GTTGCTTC', 175, 16.0, 'GTTGCTTCATT', 'GTTGCTTCATT', 24.0]				
8 ['TTGCTTC', 7, 'TTGCTTC', 176, 16.0, 'TTGCTTCATT', 'TTGCTTCATT', 22.0]				

Como muestra la siguiente ilustración, se muestran además graficas parametrizables de tiempos de consulta para facilitar el análisis computación del algoritmo.

Statistic:

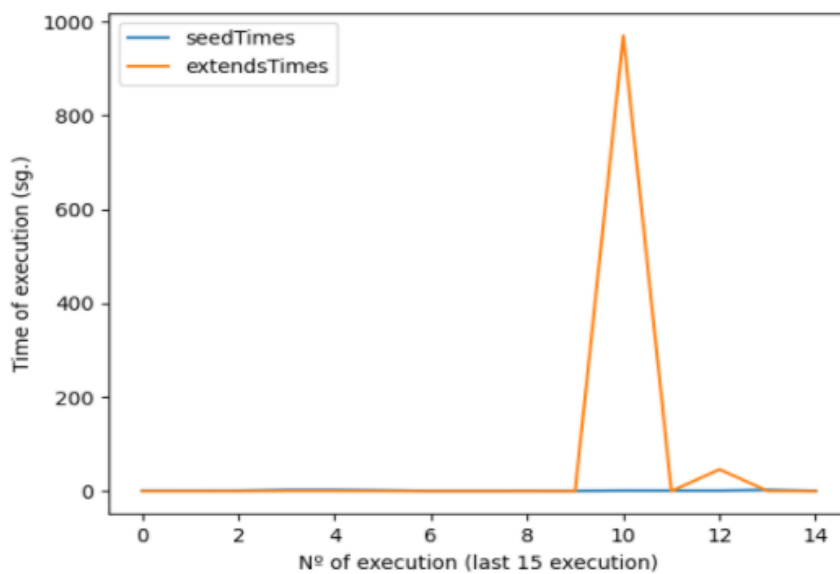
Time to find all seed calculates (score > seed_threshold): 0.7994327545166016
 Time to extends all alignment calculates: 0.04526019096374512

Complete Statistic Execution:

Change (x-Axis) Statistic Plot:

Filter Plot (Example: length(query_seq) > 30 and k=8):

Example: length(query_seq) > 30 and k=8



creation date	len(query seq)	k	mode	numSecuencesDb	numSeedCalculate	numSecuencesCalculate	time seed	time extends
2021-02-28	15	5	1	10	166	166	0.23151111602783203	1.7056653499603271
2021-02-28	15	5	1	10	166	166	0.3034958839416504	1.407651662826538
2021-02-28	15	5	1	10	153	153	0.21088480949401855	1.4765183925628662
2021-02-28	15	5	1	10	153	153	0.24352288246154785	1.3093311786651611

10.2.2 Identificación de proteína

Esta otra funcionalidad nos ofrece la relación de síntesis entre una cadena genética con una proteína relacionada. La implementación esta conseguida a través de un clasificador Bayes. Nos muestra además las estadísticas del tiempo de entrenamiento del sistema y del tiempo de consulta.

Herramienta para el estudio del alineamiento de secuencias de ADN mediante DeepLearning

localhost:5000/blast_protein

Bioinformatics Website

PARAMETERS:

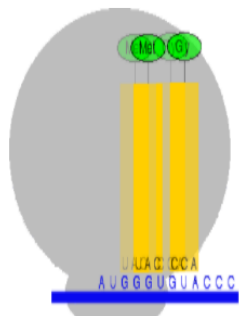
query_seq:

Execution mode:

Bayes

Classification

[Blast Protein Run](#)



[About it](#) [Contact](#) [Project \(github\)](#) [NCBI](#)

Statistics:

Confusion matrix

Predicted	0	1	2	3	4	5	6
Actual							
0	1640	0	0	1	0	2	
1	0	1440	0	0	0	6	
2	0	0	1050	0	0	4	
3	1	1	0	1641	0	0	
4	0	0	0	0	1950	2	
5	0	0	0	0	0	711	
6	0	0	0	2	0	0	349

accuracy = 98.26875515251443 %

precision = 98.30808426399021 %

recall = 98.26875515251443 %

time_fit = 13.080477476119995 sg.

time_test = 0.025997400283813477 sg.

Results:

The sequence is identified with the protein class: 6

Time to prediction: 0.0029866695404052734 sg.

Protein class table:

0 - G protein coupled receptors

1 - Tirosina kynase

2 - Tirosina Phosfate

3 - Synthetases

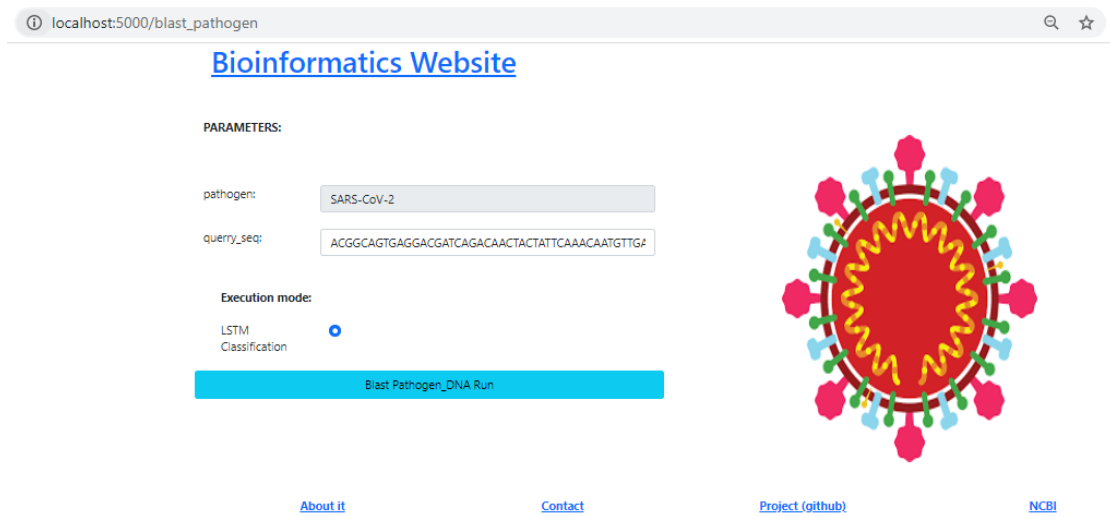
4 - Synthase

5 - Ion channel

6 - Transcription Factor

10.2.3 Identificación de patógenos

En este caso la funcionalidad nos ofrece la identificación de una cadena como parte reconocible de un patógeno conocido. La implementación esta conseguida a través de una red LSTM. Nos muestra además las estadísticas del tiempo de entrenamiento del sistema y del tiempo de consulta.



The screenshot shows a web browser window with the address bar displaying 'localhost:5000/blast_pathogen'. The page title is 'Bioinformatics Website'. Under the heading 'PARAMETERS:', there are two input fields: 'pathogen:' with the value 'SARS-CoV-2' and 'query_seq:' with the value 'ACGGCAGTGAGGACGATCAGACAACACTACTATTCAAAACAATGTTG#'. Below these is the 'Execution mode:' section, which has two radio buttons: 'LSTM' (selected) and 'Classification'. A blue progress bar at the bottom of the form area is labeled 'Blast Pathogen_DNA Run'. To the right of the form is a colorful illustration of a coronavirus particle. At the bottom of the page, there are four links: 'About it', 'Contact', 'Project (github)', and 'NCBI'.

Herramienta para el estudio del alineamiento de secuencias de ADN mediante DeepLearning

Statistics:

Confusion matrix

	0	1
0	126	1
1	14	118

accuracy = 97.99196720123291 %
time_fit = 11.51782512664795 sg.
time_test = 1.2063076496124268 sg.

Results:

Identificate pathogen: SARS-CoV-2 | Severe acute respiratory syndrome coronavirus 2 isolate Wuhan-Hu-1(complete genome)
Time to prediction: 0.6769919395446777 sg.

Neural Network Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 32)	130944
bidirectional (Bidirectional (None, 64))		16640
dense (Dense)	(None, 1)	65

=====
Total params: 147,649
Trainable params: 147,649
Non-trainable params: 0