**PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ**

**FACULTAD DE CIENCIAS E INGENIERÍA**

****

**Desarrollo de un modelo de aprendizaje profundo para identificar moléculas ARN no codificantes**

**Tesis para optar por el Título de Ingeniero Informático que presenta el bachiller:**

**José Luis Santillán Escudero**

**20030307**

**Asesor: Dr. Edwin Rafael Villanueva Talavera**

Lima, abril de 2019

# Tabla de Contenido

[Tabla de Contenido 2](#_Toc10267498)

[Índice de Figuras 6](#_Toc10267499)

[Índice de Tablas 7](#_Toc10267500)

[Capítulo 1. Generalidades 8](#_Toc10267501)

[1.1 Problemática 8](#_Toc10267502)

[1.2 Objetivos 10](#_Toc10267503)

[1.2.1 Objetivo general 10](#_Toc10267504)

[1.2.2 Objetivos específicos 10](#_Toc10267505)

[1.2.3 Resultados esperados 10](#_Toc10267506)

[1.2.4 Mapeo de objetivos, resultados y verificación 11](#_Toc10267507)

[1.3 Herramientas y Métodos 12](#_Toc10267508)

[1.3.1 Bases de datos de transcriptomas 13](#_Toc10267509)

[1.3.2 Lenguajes de programación y librerías 14](#_Toc10267510)

[1.3.3 Aprendizaje de Máquina 16](#_Toc10267511)

[1.3.4 Métodos de Validación 16](#_Toc10267512)

[Capítulo 2. Marco Conceptual 19](#_Toc10267513)

[2.1 Biología Molecular 19](#_Toc10267514)

[2.1.1 ADN 19](#_Toc10267515)

[2.1.2 ARN – proceso de transcripción 19](#_Toc10267516)

[2.1.3 Proteínas – síntesis mediante la traducción de ARN 20](#_Toc10267517)

[2.2 ARN no codificante 20](#_Toc10267518)

[2.2.1 Historia 20](#_Toc10267519)

[2.2.2 Clasificación 21](#_Toc10267520)

[2.2.3 Proyectos de secuenciación de transcriptomas 22](#_Toc10267521)

[2.3 Aprendizaje de máquina 22](#_Toc10267522)

[2.3.1 Definición 22](#_Toc10267523)

[2.3.2 Categorización 22](#_Toc10267524)

[2.3.3 Clasificadores con aprendizaje de máquina 23](#_Toc10267525)

[Capítulo 3. Estado del Arte 25](#_Toc10267526)

[3.1 Preguntas de investigación 25](#_Toc10267527)

[3.2 Estrategia de búsqueda 26](#_Toc10267528)

[3.2.1 Términos de búsqueda 26](#_Toc10267529)

[3.2.2 Proceso de búsqueda 26](#_Toc10267530)

[3.3 Criterios de inclusión y exclusión 26](#_Toc10267531)

[3.4 Extracción de la información 27](#_Toc10267532)

[3.5 Otras Tesis encontradas 29](#_Toc10267533)

[3.6 Resultados 29](#_Toc10267534)

[3.6.1 Métodos más usados 29](#_Toc10267535)

[3.6.2 Principales limitaciones 31](#_Toc10267536)

[3.7 Conclusiones 32](#_Toc10267537)

[Capítulo 4. Recopilación y preprocesamiento de datos 34](#_Toc10267538)

[4.1 Fuentes de información 34](#_Toc10267539)

[4.2 Selección de especies 34](#_Toc10267540)

[4.3 Método de obtención de las fuentes 38](#_Toc10267541)

[4.3.1 Ensembl 38](#_Toc10267542)

[4.3.2 CantataDB 39](#_Toc10267543)

[4.3.3 GreeNC 39](#_Toc10267544)

[4.4 Selección de transcritos a utilizar en el modelo 39](#_Toc10267545)

[4.5 Selección de características a utilizar para el modelo 39](#_Toc10267546)

[4.5.1 Características calculadas por scripts propios 40](#_Toc10267547)

[4.5.2 Características calculadas a través de CPAT 41](#_Toc10267548)

[4.5.3 Características calculadas a través de Diamond 45](#_Toc10267549)

[4.6 Resumen 47](#_Toc10267550)

[Capítulo 5. Modelo implementado 49](#_Toc10267551)

[5.1 Modelos referenciales 49](#_Toc10267552)

[5.2 Resultados de los modelos referenciales 50](#_Toc10267553)

[5.3 Arquitectura del modelo de especies múltiples 53](#_Toc10267554)

[5.4 Resultados del modelo de especies múltiples 54](#_Toc10267555)

[5.5 Comparación modelos referenciales contra modelo final 56](#_Toc10267556)

[5.6 Resumen 58](#_Toc10267557)

[Capítulo 6. Comparación contra otros modelos 60](#_Toc10267558)

[Referencias 61](#_Toc10267559)

[Anexos a](#_Toc10267560)

[Anexo A : Plan de Proyecto a](#_Toc10267561)

[Anexo B : Descarga de Ensembl a través de biomaRt i](#_Toc10267562)

[Anexo C : Procesamiento de descargas manuales de Ensembl k](#_Toc10267563)

[Anexo D : Procesamiento de archivos en formato fasta de CantataDB m](#_Toc10267564)

[Anexo E : Procesamiento de archivos en formato fasta de GreeNC o](#_Toc10267565)

[Anexo F : Selección aleatoria de transcritos para el modelo q](#_Toc10267566)

[Anexo G : Características – Longitud del transcrito r](#_Toc10267567)

[Anexo H : Características – Porcentaje de nucleótidos GC s](#_Toc10267568)

[Anexo I : Generación de archivos en formato fasta para CDS y lncRNA t](#_Toc10267569)

[Anexo J : CPAT – Generación de tabla de frecuencias de hexámeros v](#_Toc10267570)

[Anexo K : Generación de archivos en formato fasta para ARN codificantes w](#_Toc10267571)

[Anexo L : CPAT – Generación de regresión logística x](#_Toc10267572)

[Anexo M : CPAT – Ejecución de CPAT y exportación a la base de datos y](#_Toc10267573)

[Anexo N : Diamond – Generación de la base de datos aa](#_Toc10267574)

[Anexo O : Diamond – Búsqueda de alineamientos bb](#_Toc10267575)

[Anexo P : Diamond – Extracción de características cc](#_Toc10267576)

[Anexo Q : Librería util\_bd.py ff](#_Toc10267577)

[Anexo R : Librería util\_fasta.py gg](#_Toc10267578)

[Anexo S : Librería util\_caracteristicas.py ii](#_Toc10267579)

[Anexo T : Librería util\_modelo\_final.py mm](#_Toc10267580)

[Anexo U : Generación de modelos referenciales y finales ggg](#_Toc10267581)

Índice de Figuras

Figura 1 - Síntesis de proteínas y ARN no codificante. Elaboración propia. 8

Figura 2 - Árbol de problema. Elaboración propia. 10

Figura 3 – Validación cruzada. Elaboración propia. 17

Figura 4 - Tipos de resultados de un predictor. Elaboración propia. 18

Figura 5 - Esquema del ADN. Adaptado de (Risueño Pérez, 2012). 19

Figura 6 - Estructura del ARN. Adaptado de (Vieira, 2018). 20

Figura 7 - Diferencia entre aprendizaje tradicional y aprendizaje profundo. Adaptado de (Rouse, 2017). 23

Figura 8 - Número de investigaciones relacionadas a aprendizaje profundo en Bioinformática en Scopus. Elaboración propia. 24

Figura 9 - Generación del modelo de regresión logística para CPAT. Elaboración propia. 42

Figura 10 - Diferencia visualizada en Biomart (Ensembl) entre transcrito y secuencia codificante. Elaboración propia. 44

Figura 11 - Resumen de la extracción y preprocesamiento de los datos. Elaboración propia. 48

Figura 12 - Arquitectura de modelos de referencia, tomando como ejemplo una especie. Elaboración propia. 50

Figura 13 - Gráfico de cajas de la exactitud, precisión y recuperación de los 30 modelos referenciales. Elaboración propia. 51

Figura 14 - Arquitectura del modelo de especies múltiples. Elaboración propia. 54

Figura 15 - Estructura de descomposición del trabajo. Elaboración propia. d

Figura 16 - Cronograma del proyecto por semanas. Elaboración propia. f

Índice de Tablas

Tabla 1 - Mapeo de objetivos, resultados y verificación. 11

Tabla 2 - Herramientas y métodos. 12

Tabla 3 - Clasificación del ARN. Adaptado de (Genomics, 2015). 21

Tabla 4 - Conceptos generales definidos con el método PICOC. 25

Tabla 5 - Componentes de la cadena de búsqueda. 26

Tabla 6 - Artículos encontrados por base de datos. 27

Tabla 7 - Artículos que discuten el estado del arte. 28

Tabla 8 - Métodos de aprendizaje de máquina más usados para la clasificación de lncRNA. 29

Tabla 9 - Número de especies en las bases de datos de transcriptomas 34

Tabla 10 - Especies con el mayor conteo de clases positivas, y sus fuentes 35

Tabla 11 - Número total de transcritos a utilizar según el modelo seleccionado 36

Tabla 12 - Ejemplo de tabla de frecuencia de hexámeros para Amborella trichopoda generada por CPAT 44

Tabla 13 - Resultados de exactitud, precisión y recuperación de los modelos referenciales 51

Tabla 14 - Hiperparámetros buscados para el modelo final 55

Tabla 15 - Comparación de modelos referenciales y modelos finales 56

Tabla 16 - Riesgos del proyecto de investigación. c

Tabla 17 - Lista de tareas del proyecto. e

Tabla 18 - Costos del proyecto. h

# Generalidades

## Problemática

El ADN (ácido desoxirribonucleico) es una molécula compuesta principalmente por nucleótidos, que se encuentra en el núcleo de la célula y es responsable de almacenar información para la generación de proteínas y ARN (ácido ribonucleico) para que cumplan todas las funciones celulares (NHGRI, 2015).

Esta generación de proteínas en las células es posible a través del proceso de síntesis de proteínas. Para ello, se genera una copia de un segmento del ADN (gen) que contiene la información de la proteína a sintetizar a través del proceso de transcripción, produciéndose una molécula intermediaria de ARN con la información del gen, la cual mediante el proceso de traslación se puede expresar en proteína (NHGRI, 2015). Sin embargo, solo una pequeña proporción del ARN producido por el ADN se expresa en proteínas, y a estos se les conoce como ARN codificantes (o en inglés, como *protein coding transcripts* – *PCT*) (Kung, Colognori, & Lee, 2013). El ARN que no se expresa en proteínas es conocido como ARN no codificante (ver Figura 1).

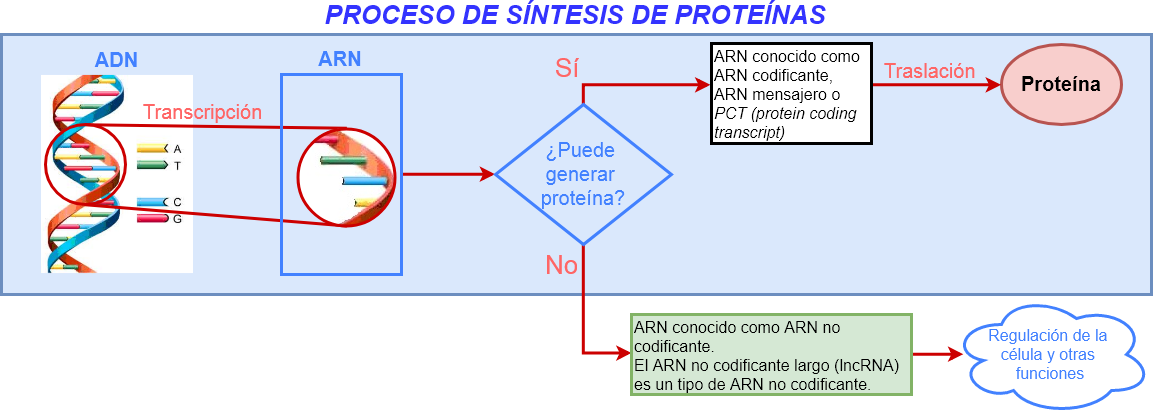


Figura 1 - Síntesis de proteínas y ARN no codificante. Elaboración propia.

Históricamente se pensó que las secciones del ADN que se transcriben a ARN no codificantes eran simplemente rezagos evolutivos de material genético, que perdió utilidad, y por lo tanto no tenían ninguna función biológica (Ponjavic, Ponting, & Lunter, 2007). Pero con el pasar del tiempo se fue descubriendo que estas secciones sí cumplían funciones, y en particular hoy en día se sabe que los ARN no codificantes largos (del inglés *long non-coding* *RNA*, o *lncRNA*) cumplen funciones reguladoras en la célula y que tienen implicancias en el desarrollo de enfermedades como el cáncer (Kung et al., 2013). Es así que estas regiones no codificantes toman importancia para el entendimiento del genoma, sobre todo al ser un componente mayoritario del mismo (Uszczynska-Ratajczak, Lagarde, Frankish, Guigo, & Johnson, 2018).

La definición clásica que se tiene sobre los *lncRNAs* es que son secciones no codificantes conformadas por más de 200 nucleótidos (Yao et al., 2018). Sin embargo, diferenciar *lncRNAs* de *PCTs* es una tarea desafiante, dada su similitud estructural (Quinn & Chang, 2016). La correcta clasificación de estas moléculas es importante, ya que pueden tener un impacto profundo en proyectos que se apoyan en estas anotaciones (Uszczynska-Ratajczak et al., 2018).

En la literatura existen técnicas basadas en Aprendizaje de Máquina (AM) para discriminar *lncRNAs* de *PCTs*. Sin embargo, el objetivo de tales estudios ha sido generar predictores para especies específicas, con base a muestras experimentalmente anotadas en esas mismas especies. El problema se presenta cuando se secuencian transcriptomas en especies poco estudiadas (Schneider, Raiol, Brigido, Walter, & Stadler, 2017). En tales casos, no se dispone de un conjunto de muestras anotadas para realizar el aprendizaje de modelos predictivos, y justamente es en esos casos donde más se necesita contar con tales modelos para hacer una discriminación inicial de *lncRNAs* y *PCTs* y ayudar así a focalizar la validación experimental (Simopoulos, Weretilnyk, & Golding, 2018).

En el presente trabajo se pretende aplicar técnicas computacionales de aprendizaje de máquina (AM) para la clasificación de *lncRNA* en especies desconocidas para el modelo. Dentro de este contexto, se propone el desarrollo de un modelo computacional de aprendizaje profundo, para investigar la viabilidad de crear modelos de predicción de *lncRNAs* para especies desconocidas, entrenándolo con información de otras especies ya anotadas.

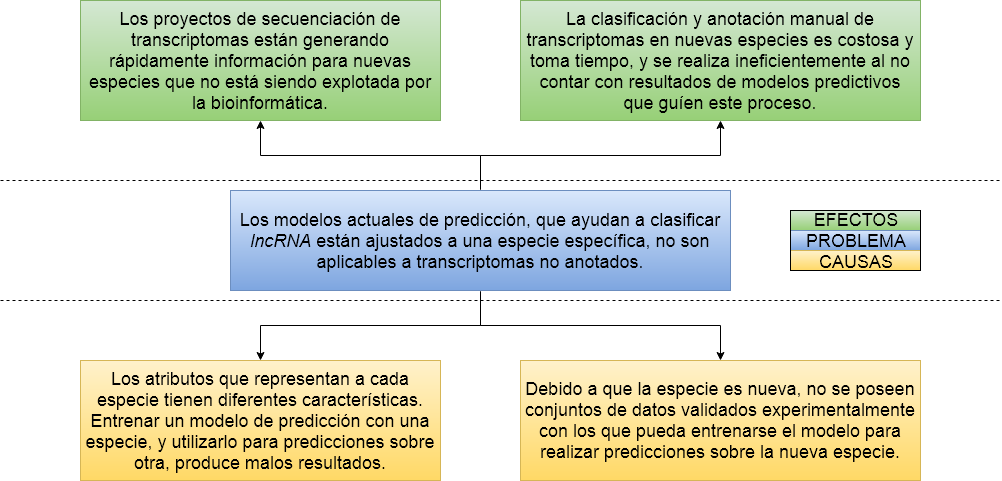


Figura 2 - Árbol de problema. Elaboración propia.

## Objetivos

### Objetivo general

Elaborar y evaluar un modelo computacional, basado en aprendizaje profundo, para la identificación de moléculas ARN no codificantes en transcritos de especies desconocidas al modelo.

### Objetivos específicos

1. Recopilar y preprocesar un conjunto de datos de transcritos para el entrenamiento y validación del modelo algorítmico.
2. Implementar la arquitectura del proceso de aprendizaje de máquina y que la misma produzca predictores aceptables.
3. Realizar un análisis comparativo de la precisión de los predictores generados contra otro modelo del estado del arte.

### Resultados esperados

1. Conjuntos de datos estructurados recolectados y preprocesados de transcritos (O1).
2. Arquitectura del proceso de Aprendizaje de Máquina (O2).
3. Análisis comparativo de la precisión de los predictores generados y otro modelo referencial del estado del arte (O3).

### Mapeo de objetivos, resultados y verificación

En la Tabla 1 se presentan los objetivos específicos planteados, juntos a sus resultados esperados, mapeados con los entregables y su respectiva verificación.

Tabla 1 - Mapeo de objetivos, resultados y verificación.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Objetivo:** Recopilar y preprocesar un conjunto de datos de transcritos para el entrenamiento y validación del modelo algorítmico. | | | | |
| **Resultado** | | **Meta física** | **Medio de verificación** | |
| Conjuntos de datos estructurados recolectados y preprocesados de transcritos. | | Conjunto de datos listo para su uso.  Cuadros y gráficos estadísticos. | * Documento del proceso de recolección y preprocesamiento de transcritos. * Estadísticas descriptivas de los datos. | |
| **Objetivo:** Implementar la arquitectura del proceso de aprendizaje de máquina y que la misma produzca predictores aceptables. | | | | |
| **Resultado** | **Meta física** | | | **Medio de verificación** |
| Arquitectura del proceso de Aprendizaje de Máquina. | Arquitectura implementada.  Documento de reporte de resultados.  Cuadros y gráficos estadísticos. | | | * Documento detallando la arquitectura implementada. * Precisión de los predictores generados para clasificar especies para las cuales no fue entrenado. |
| **Objetivo:** Realizar un análisis comparativo de la precisión de los predictores generados contra otro modelo del estado del arte. | | | | |
| **Resultado** | | **Meta física** | **Medio de verificación** | |
| Análisis comparativo de la precisión de los predictores generados y otro modelo similar del estado del arte. | | Documento de reporte comparativo.  Cuadros y gráficos estadísticos. | * Comparación de la precisión de los predictores generados y otro modelo del estado del arte para clasificar especies desconocidas al modelo, al ser entrenados en otras. | |

## Herramientas y Métodos

A continuación, en Tabla 2, se detallan las herramientas y métodos que ayudarán con la obtención y/o validación de cada uno de los resultados esperados planteados.

Tabla 2 - Herramientas y métodos.

|  |  |
| --- | --- |
| **Objetivo:** Recopilar y preprocesar un conjunto de datos de transcritos para el entrenamiento y validación del modelo algorítmico. | |
| **Resultado** | **Herramientas o métodos** |
| Conjuntos de datos estructurados recolectados y preprocesados de transcritos. | * Ensembl * CantataDB * GreeNC * Anaconda * Python * R * BiomaRt * Jupyter Notebook * Matplotlib y Seaborn |
| **Objetivo:** Implementar la arquitectura del proceso de aprendizaje de máquina y que la misma produzca predictores aceptables. | |
| **Resultado** | **Herramientas o métodos** |
| Arquitectura del proceso de Aprendizaje de Máquina. | * Redes neuronales profundas * Anaconda * Tensorflow * Keras * Python * Jupyter Notebook * Matplotlib y Seaborn * Validación cruzada * Medidas de precisión: exactitud, precisión y recuperación |
| **Objetivo:** Realizar un análisis comparativo de la precisión de los predictores generados contra otro modelo similar del estado del arte. | |
| **Resultado** | **Herramientas o métodos** |
| Análisis comparativo de la precisión de los predictores generados y otro modelo del estado del arte. | * Anaconda * Python * R * Jupyter Notebook * Matplotlib y Seaborn * Validación cruzada * Medidas de precisión: exactitud, precisión y recuperación |

### Bases de datos de transcriptomas

#### Ensembl

Los proyectos de secuenciación de transcriptomas buscan generar la secuencia completa de ARN de un organismo (Vieira, 2018). Debido a la aparición de nuevos métodos de secuenciación, los proyectos de secuenciación de transcriptomas ahora son más accesibles (Uszczynska-Ratajczak et al., 2018), y existe una multitud de fuentes disponibles para transcritos generados por estos proyectos.

Dentro de estas fuentes, Ensembl (http://www.ensembl.org) posee una gran cantidad de información de transcritos de diferentes especies. Adicionalmente, activamente desarrollan APIs públicas para acceder de forma más directa a sus servicios (Zerbino et al., 2018).

#### CantataDB

CantataDB es una base de datos de *lncRNAs* en línea ([http://cantata.amu.edu.pl](http://cantata.amu.edu.pl/)), que nació para suplir la necesidad de bases de datos más integrales de transcritos, específicamente de *lncRNAs*, para plantas (Szcześniak, Rosikiewicz, & Makałowska, 2016). Actualmente posee información para 36 especies de plantas, y permite realizar búsquedas en línea y descarga de transcritos.

#### GreeNC

GreeNC es otra una base de datos de *lncRNAs* en línea ([http://greenc.sciencedesigners.com](http://greenc.sciencedesigners.com/)), que también nació de la necesidad de tener anotaciones de ARN no codificantes largos en plantas, para lo que desarrollaron un flujo para la anotación de estas moléculas y lo aplicaron a 37 especies (Paytuví Gallart, Hermoso Pulido, Anzar Martínez de Lagrán, Sanseverino, & Aiese Cigliano, 2016). Actualmente posee información para 45 especies de plantas, y permite realizar búsquedas en línea y descarga de transcritos.

### Lenguajes de programación y librerías

#### Anaconda

Anaconda (<https://www.anaconda.com>) es una plataforma de código abierto que puede gestionar la instalación completa de todos los elementos necesarios (incluyendo dependencias) para la elaboración de trabajos científicos. Debido a que centraliza la gestión de toda la plataforma necesaria para la investigación, se optó por su uso.

Todos los elementos mencionados en esta sección son instalables y administrables a través de Anaconda.

#### Python

Python es un lenguaje de programación que en los últimos años ha ganado popularidad. Entre sus principales ventajas está su facilidad de uso, el soporte que se puede encontrar para el lenguaje en la comunidad y la gran cantidad de librerías con integraciones a otros lenguajes o plataformas, como con el lenguaje R y con Tensorflow (Willems, 2015).

#### R

R es un lenguaje de programación está enfocado al análisis de datos, estadísticas y gráficos, que también está gozando de una creciente popularidad en los últimos años (Willems, 2015). Su facilidad de uso para el análisis estadístico fue un factor clave para su elección de cara al análisis que se ha de realizar de los distintos modelos predictivos en la presente investigación.

#### BiomaRt

Este paquete para R forma parte del proyecto de código abierto Bioconductor, que desarrolla una multitud de herramientas estadísticas y de gráficos orientadas al análisis de información genómica, para el lenguaje R (Durinck et al., 2005). En particular el paquete biomaRt ofrece una interfaz de programación que permite consultar bases de datos a través de internet, como por ejemplo Ensembl (Durinck, Spellman, Birney, & Huber, 2009).

#### Tensorflow

Tensorflow es una librería de código abierto para aprendizaje de máquina, desarrollada por Google. Se ha optado por su uso debido a que se especializa en modelos con aprendizaje profundo, permitiendo la manipulación de estos a un bajo nivel y un gran nivel de flexibilidad en la implementación de distintos modelos (Abadi et al., 2016).

#### Keras

Keras es una API de alto nivel, orientada a facilitar la construcción y el manejo de modelos de redes neuronales profundas. Tensorflow posee una implementación específica de esta API, que facilita en gran medida el uso y ajustes que se necesiten realizar al modelo implementado en Tensorflow (Chollet, 2017).

#### Jupyter Notebook

Jupyter Notebook (<http://jupyter.org>) es una aplicación web que tiene varias funciones, como editar código en línea, compartir documentos y otras tareas especializadas para trabajos científicos. Se utilizará porque es ideal para el trabajo colaborativo científico, ya que expone códigos de programas con textos e imágenes descriptivos de los mismos, que pueden ser editados y ejecutados en línea (Shen, 2014).

#### Matplotlib y Seaborn

Matplotlib es una librería de código abierto para Python que permite generar gráficos y visualizaciones de datos. Tiene una gran base de usuarios y soporte, además de varias otras APIs construidas sobre esta librería (VanderPlas, 2016).

Una de estas APIs es Seaborn, que ayuda a extender la funcionalidad de Matplotlib, que se estaba quedando desfasada, sobre todo en términos de la calidad de los gráficos que genera, además de ofrecer un uso más simple de Matplotlib (VanderPlas, 2016).

### Aprendizaje de Máquina

#### Redes neuronales profundas

Las redes neuronales son algoritmos de aprendizaje de máquina que fueron modelados en base al cerebro humano. La unidad básica de estas redes es la neurona, que recibe entradas numéricas y devuelve una salida entre 0 y 1. Estas neuronas se organizan en capas, y cuando la red neuronal posee más de dos capas, se le conoce como red neuronal profunda (Nielsen, 2018).

El uso de las redes neuronales profundas, y sus variantes, es cada vez mayor en el campo de la biología, apoyándose en su poder de abstracción y en el beneficio que obtienen de la cantidad de información disponible gracias a los proyectos de secuenciación de transcriptomas (Jabeen, Ahmad, & Raza, 2018).

Dentro de los distintos tipos de redes neuronales profundas, en el presente proyecto se hará uso de la red neuronal prealimentada (del inglés *feed-forward network*). Son el tipo más básico de red neuronal, en la que las conexiones no forman ciclos y viajan hacia adelante.

### Métodos de Validación

#### Validación cruzada

Es un método estadístico ampliamente usado en modelos de aprendizaje de máquina, que consiste, en su versión más simple, en la división de un conjunto de datos en un número k de grupos, de los cuales uno será tomado para la validación del modelo, y el resto para el entrenamiento (Brownlee, 2018).

El modelo será evaluado k veces, hasta que cada grupo haya sido usado para la validación del modelo (ver Figura 3). Al final del proceso obtendremos k mediciones de la precisión del modelo predictivo, de los cuales podemos extraer medidas representativas a través, por ejemplo, de un promedio simple de las mediciones (Brownlee, 2018).

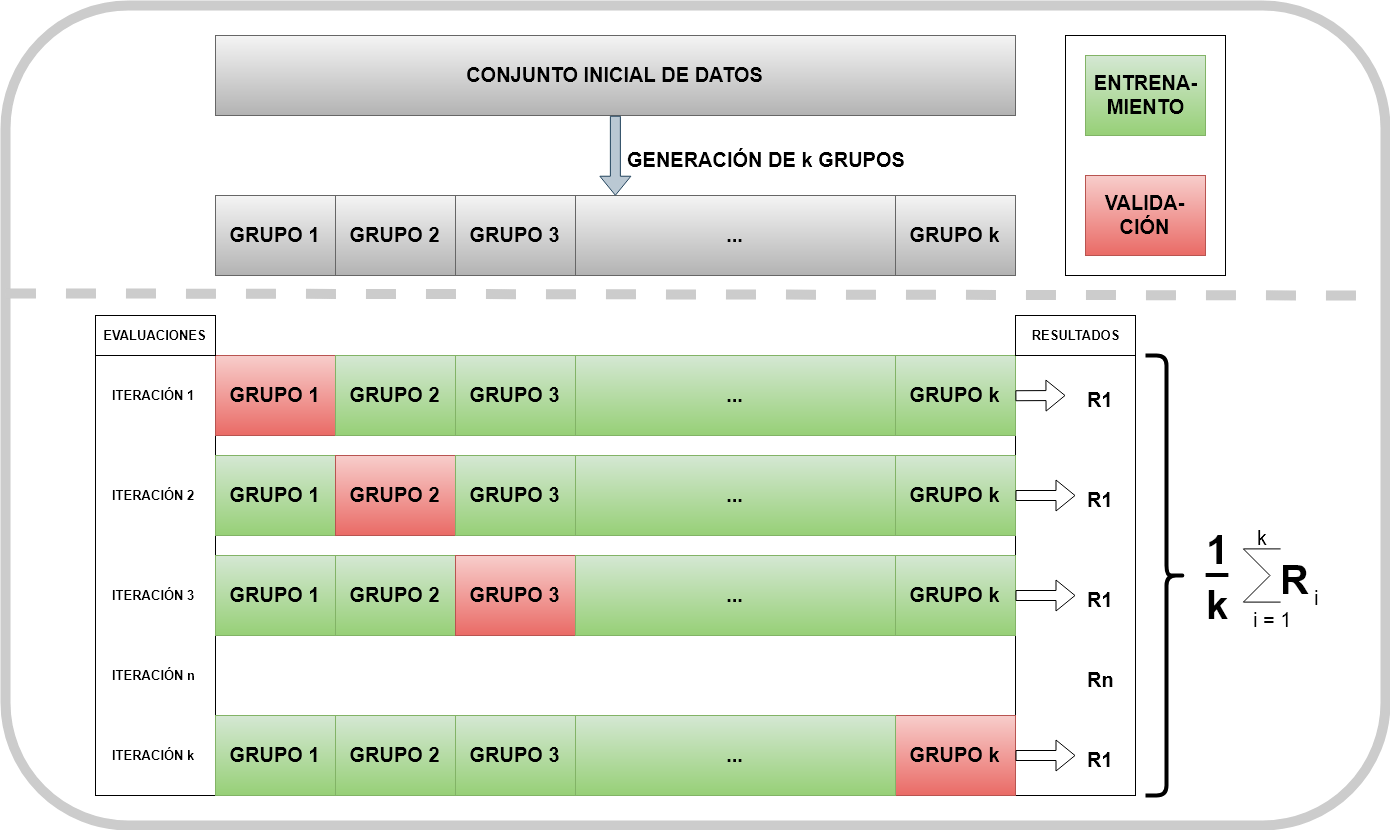


Figura 3 – Validación cruzada. Elaboración propia.

#### Medidas de precisión: exactitud, precisión y recuperación

Para medir la precisión del modelo predictivo, se utilizarán tres medidas: la exactitud, la precisión y la recuperación. Estas medidas son clásicamente usadas para evaluar el desempeño de modelos predictivos (Wucher et al., 2017).

Cuando se genera un predictor binario (se llama así porque solamente predice dos clases, en este caso: *lncRNA* o *PCT*), los resultados pueden dividirse en cuatro secciones: verdaderos positivos (VP), falsos positivos (FP), falsos negativos (FN) y verdaderos negativos (VN). La Figura 4 detalla el significado de cada uno de ellos en el contexto del predictor de *lncRNA*.

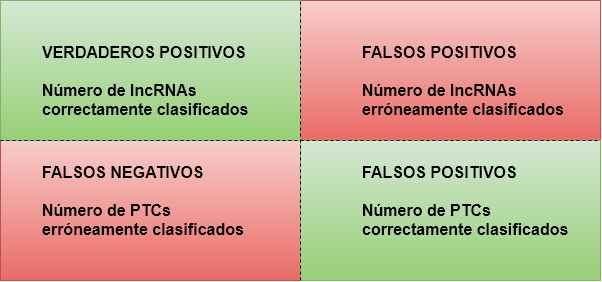


Figura 4 - Tipos de resultados de un predictor. Elaboración propia.

Entonces, para realizar la validación del modelo, las tres medidas se definen de la siguiente manera:

* Exactitud = (VP + VN) / (VP + FP + FN + VN)
* Precisión = (VP) / (VP + FP)
* Recuperación = (VP) / (VP + FN)

# Marco Conceptual

## Biología Molecular

La biología molecular estudia el funcionamiento de las moléculas que posibilitan los procesos biológicos. Entre las principales moléculas que estudia están el ADN (ácido desoxirribonucleico) y las proteínas (Setubal & Meidanis, 1997).

### ADN

El ADN está formado por un polímero de nucleótidos unidos en una doble hélice, y contiene la información genética del organismo. Un nucleótido está formado por una base nitrogenada, un grupo fosfato y un azúcar (ver Figura 5). Las bases nitrogenadas que puede contener un nucleótido en el ADN son la adenina (A), la guanina (G), la timina (T) y la citosina (C). El orden en el que se encuentran finalmente determina la secuencia genómica (Risueño Pérez, 2012).

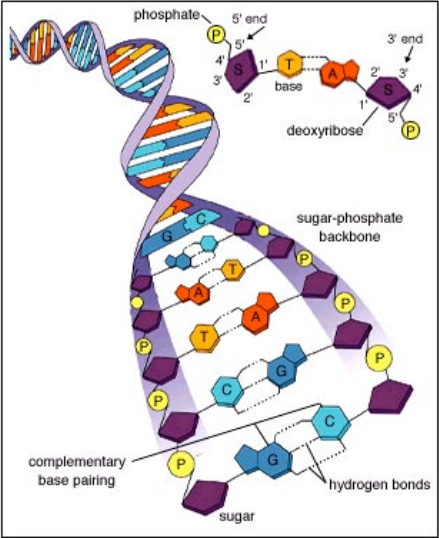


Figura 5 - Esquema del ADN. Adaptado de (Risueño Pérez, 2012).

La doble hélice está unida por puentes de hidrógeno formados por las bases nitrogenadas, y unen estos nucleótidos siempre en pares que se limitan a A-T (adenina con timina) y G-C (guanina con citosina).

El ADN no se usa de forma directa para la generación de proteínas, sino que su información se copia a una molécula de ARN intermediaria mediante el proceso de transcripción.

### ARN – proceso de transcripción

En el proceso de transcripción, secuencias de ADN son duplicadas en lo que se conoce como ARN (ácido ribonucleico). El ARN, al igual que el ADN, se compone de nucleótidos (ver Figura 6). Sin embargo, el ARN forma una sola hebra, y en el proceso de transcripción se reemplaza la timina, por el uracilo (U).

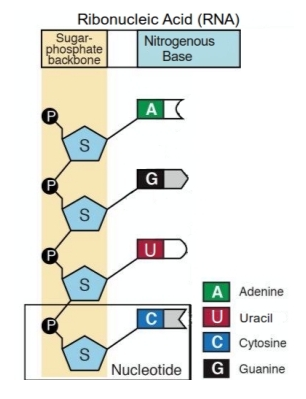


Figura 6 - Estructura del ARN. Adaptado de (Vieira, 2018).

En este punto, el ARN puede llegar o no a expresarse en proteínas. Los ARN que generarán proteínas mediante la traducción se conocen como ARNs mensajeros (*mRNA*), y los que no, se conocen como ARNs no codificantes.

### Proteínas – síntesis mediante la traducción de ARN

El ARN transcrito, viaja del núcleo al ribosoma (sección de la célula dónde se lleva a cabo la síntesis de proteínas), ubicado en el citoplasma de la célula. El ribosoma se encarga entonces de la traducción del ARN a cadenas de aminoácidos, que finalmente son los que forman la proteína.

## ARN no codificante

### Historia

Como se mencionó, existe ARN transcrito que no se expresa en proteínas y se conoce como ARN no codificante. Al descubrirse inicialmente que gran parte del genoma no se expresaba en proteínas, y al no encontrarse ninguna función para estas secciones del ADN que no se codificaban (la biología molecular estaba en sus etapas iniciales), se asumió que estas secciones sin función existían debido a que en el pasado mutaron y hoy en día ya no cumplían ninguna función. También se teorizó que eran secciones que podían evolucionar y volverse codificantes, generando mutaciones en el organismo. En todo caso, esto originó que se acuñe el término “ADN basura” para estas secciones del ADN (Kung et al., 2013).

Entre 1990 y 2000, con la llegada de los proyectos de secuenciación del genoma, se haría evidente que una gran cantidad de este “ADN basura” era en realidad transcrito en algún punto (Kung et al., 2013), lo cual generó la pregunta de que si estas transcripciones, que no finalizaban en la síntesis de proteínas, servían algún propósito.

Así fue como el estudio del ARN no codificante se volvió relevante en la biología molecular. Sobre todo, a la luz de que se identificó que mientras un organismo es más complejo, posee mayor cantidad de ARN no codificante (Sebastián, 2017).

### Clasificación

Se presenta una tabla resumen con la clasificación del ARN:

Tabla 3 - Clasificación del ARN. Adaptado de (Genomics, 2015).

|  |  |
| --- | --- |
| Clasificación | Función |
| ARN mensajero (*mRNA*) | Lleva información genética transcrita del ADN con potencial codificante. |
| ARN de la transferencia (*tRNA*) | Participa en el proceso de traducción del ARN a proteína, transportando aminoácidos al proceso de síntesis de proteica. |
| ARN ribosomal (*rRNA*) | Se encuentran en el ribosoma y forman el 80% del ARN de la célula. Participan en la síntesis de la proteína. |
| Micro ARN (*miRNA*) | Pequeños ARNs (20~25 nucleótidos). Tiene funciones regulatorias. |
| ARN pequeño (*small* *RNA*) | Participa en el proceso de traducción del ARN. |
| ARN de telomerasa (*telomeraseRNA*) | Se relaciona con el proceso de replicación de cromosomas. |
| ARN anti-sentido (*antisense* *RNA*) | Directamente involucrado en la regulación de la expresión genética. |
| ARN largo no codificante (*lncRNA*) | ARN no codificante, que participa en la regulación de la célula. Suelen tener una longitud no menor a 200 nucleótidos y son difícil de distinguir de los codificantes. |

### Proyectos de secuenciación de transcriptomas

Los métodos más usados actualmente para el perfilado de transcriptomas son los microarreglos y la secuenciación de ARN (*RNA-Seq*). En especial, el secuencionamiento de ARN permite obtener millones de secuencias con gran precisión estadística, y se pueden utilizar en nuevas especies para las cuáles no existe una secuencia genómica de referencia.

En los últimos años han habido muchas iniciativas de proyectos de secuenciación de transcriptomas, que han generado una enorme cantidad de información de ARN (Schneider et al., 2017). Estos proyectos son de gran importancia para el estudio del ARN, por ejemplo, son útiles para:

* El diagnóstico de enfermedades (Tolosa, 2017).
* La comprensión de cómo cada sección del ADN afecta el funcionamiento de distintos tipos de células para un mismo organismo.

## Aprendizaje de máquina

### Definición

El campo del aprendizaje de máquina se encarga del desarrollo de técnicas computacionales que aprenden a realizar tareas a partir del entrenamiento de conjuntos de datos preexistentes (Jabeen et al., 2018).

### Categorización

Se pueden dividir los métodos de aprendizaje de máquina en dos grandes grupos: el aprendizaje tradicional y el aprendizaje profundo.

El aprendizaje tradicional implica la extracción de atributos de las entidades que se quieren modelar, el entrenamiento del modelo en base a los atributos extraídos y la generación de resultados. La generación de atributos no es un proceso automático, y suele requerir el juicio de un experto.

El aprendizaje profundo integra todos estos pasos en el proceso de aprendizaje del modelo, aprovechando la gran cantidad de información disponible para inferir atributos y representarlos en un bajo nivel, y generar el modelo de predicción en base a ellas (ver Figura 7).



Figura 7 - Diferencia entre aprendizaje tradicional y aprendizaje profundo. Adaptado de (Rouse, 2017).

### Clasificadores con aprendizaje de máquina

Los métodos de clasificación de aprendizaje de máquina se favorecen de grandes cantidades de información, y dado que los proyectos de secuenciación de transcriptomas están generando una gran cantidad de información de transcriptomas, se prestan para su análisis (Min, Lee, & Yoon, 2017).

En particular, la generación de investigaciones relacionadas a algoritmos de aprendizaje profundo ha ido en aumento en el área de la bioinformática, debido a que apoyándose en la existencia de grandes cantidades de información, puede simplificar procesos al extraer por su propia cuenta atributos relevantes de las clases que estudia (Min et al., 2017).

Para visualizar este incremento en las investigaciones, se realizó un gráfico (ver Figura 8) de las investigaciones encontradas en la base de datos Scopus con dos cadenas de búsqueda:

* “Deep learning” (aprendizaje profundo).
* “Deep learning” AND “bio” (aprendizaje profundo en bioinformática).

Figura 8 - Número de investigaciones relacionadas a aprendizaje profundo en Bioinformática en Scopus. Elaboración propia.

La distinción de moléculas no codificantes, entonces, sería una tarea ideal de aprendizaje de máquina profundo, desde el punto de vista computacional (Schneider et al., 2017).

# Estado del Arte

En este capítulo se presentan estudios sobre la clasificación de ARNs no codificantes largos a través de métodos de aprendizaje de máquina en artículos científicos, tesis u otros trabajos de investigación relacionados. Se realiza esta revisión sistemática del estado del arte mediante los lineamientos definidos en el reporte “*Guidelines for performing systematic literature reviews in software engineering*” (Kitchenham & Charters, 2007).

## Preguntas de investigación

Primero se definirán con ayuda del método PICOC (*population, intervention, comparison, outcomes, context*) los conceptos generales. Estos servirán de apoyo para formular las peguntas de investigación.

Tabla 4 - Conceptos generales definidos con el método PICOC.

|  |  |
| --- | --- |
| Criterio | Descripción |
| Población | ARNs no codificantes largos. |
| Intervención | Clasificación a través de métodos de aprendizaje de máquina / aprendizaje profundo. |
| Comparación | *(\*) No aplica. Más que hacer una comparación de los métodos actuales, se desea saber cuáles son y qué necesidades hay en el campo de estudio.* |
| Resultados | Aplicables a múltiples especies. |
| Contexto | Contexto académico, bioinformática. |

En base a lo anterior, se plantearon las siguientes preguntas de investigación:

* ¿Cuáles son los métodos de aprendizaje de máquina más representativos que se han usado para la clasificación de *lncRNAs*?
* ¿Cuáles son las principales limitaciones de los métodos utilizados actualmente para la clasificación de *lncRNAs*?

## Estrategia de búsqueda

### Términos de búsqueda

A partir de los conceptos extraídos con el método PICOC, de las preguntas de investigación planteadas y tomando en cuenta que la literatura buscada se encuentra principalmente en inglés, se plantearon los siguientes componentes para la cadena de búsqueda (Tabla 5):

Tabla 5 - Componentes de la cadena de búsqueda.

|  |  |
| --- | --- |
| # | Componentes de la cadena de búsqueda |
| C1 | “lncRNA” OR “long non-coding RNA” OR “long noncoding RNA” OR “long non coding RNA” |
| C2 | “classification” OR “prediction” |
| C3 | “machine learning” OR “deep learning” |

Se usará la cadena de búsqueda:

C1 AND C2 AND C3

### Proceso de búsqueda

Se realizará la búsqueda en las siguientes bases de datos:

* Web of Science
* Scopus
* Springer Link
* BMC Bioinformatics

## Criterios de inclusión y exclusión

Los criterios de inclusión utilizados fueron los siguientes:

* Los artículos que en sí mismos sean una revisión del estado del arte son automáticamente incluidos.
* Los artículos que sean sobre la implementación de algún algoritmo de aprendizaje de máquina específico son revisados, y su inclusión se basa en:
  + Fecha de publicación, favoreciendo los más recientes.
  + Número de veces citados por otros artículos. Tomando en cuenta que el peso de este criterio es menor para los artículos más recientes, pero mayor para los más antiguos.
  + Se favorecen aquellos que tengan una implementación pública, con la intención de que más adelante se puedan realizar comparaciones. Pero si no la tienen no implica que serán excluidos.
* El artículo está en inglés o español (sin embargo, esto no modifica la decisión de que la búsqueda se realizará en inglés).

Los criterios de exclusión que se utilizaron fueron los siguientes:

* Artículos previos al 2008.
* Idioma diferente al inglés o español.
* Método de clasificación diferente a los de aprendizaje de máquina.

## Extracción de la información

Se presenta a manera de resumen, la cantidad de artículos encontrados en cada una de las bases de datos buscadas:

Tabla 6 - Artículos encontrados por base de datos.

|  |  |  |  |
| --- | --- | --- | --- |
| Base de datos | Artículos Encontrados | Artículos Duplicados | Artículos Seleccionados |
| Web of Science | 20 | 0 | 8 |
| Scopus | 40 | 14 | 2 |
| Springer Link | 154 | 14 | 4 |
| BMC Bioinformatics | 56 | 56 | 0 |
| Total | 270 | 84 | 14 |

Podemos dividir los artículos encontrados en dos clases:

* Artículos con implementaciones específicas de clasificadores de lncRNAs. De esta clase de artículos se encontraron 9.
* Artículos que discuten el estado del arte. De esta clase de artículos se encontraron 5.

De los artículos que discuten el estado del arte se extraerán otras implementaciones adicionales para expandir la lista de artículos con implementaciones específicas.

El objetivo final será, además de revisar la literatura para obtener el estado del arte con respecto a la clasificación de *lncRNAs* mediante métodos de aprendizaje de máquina, el generar una lista con los métodos más recientes que dispongan de implementaciones públicas que nos permita realizar comparaciones con lo desarrollado en la presente tesis para evaluar la utilidad del modelo predictivo que se quiere realizar.

Tomando en cuenta lo anterior, se presentan los artículos que discuten el estado del arte:

Tabla 7 - Artículos que discuten el estado del arte.

|  |  |  |
| --- | --- | --- |
| Año | Autor | Título |
| Yao, Yuhua Li, Xianhong Geng, Lili Nan, Xuying Qi, Zhaohui Liao, Bo | 2018 | Recent Progress in Long Noncoding RNAs Prediction (Yao et al., 2018) |
| Han, Siyu Liang, Yanchun Li, Ying Du, Wei | 2016 | Long Noncoding RNA Identification: Comparing Machine Learning Based Tools for Long Noncoding Transcripts Discrimination (Han, Liang, Li, & Du, 2016) |
| Jia-Pei, Yuan Hao-Wen, Zhang Zhi, Lu | 2013 | Progress on Bioinformatic Research of lncRNA (Jia-Pei, Hao-Wen, & Zhi, 2013) |
| Jabeen, Almas Ahmad, Nadeem Raza, Khalid | 2018 | Machine Learning-Based State-of-the-Art Methods for the Classification of RNA-Seq Data (Jabeen et al., 2018) |
| Yip, Kevin Y. Cheng, Chao Gerstein, Mark | 2013 | Machine learning and genome annotation: a match meant to be? (Yip, Cheng, & Gerstein, 2013) |

## Otras Tesis encontradas

Se realizó la búsqueda de tesis de bioinformática similares en el repositorio de tesis PUCP, pero no se encontraron resultados.

Por otro lado, se ha tenido acceso a una tesis de Brasil, “Métodos baseados em aprendizagem de máquina para distinguir RNAs longos não-codificadores intergênicos de transcritos codificadores de proteínas” (Vieira, 2018), donde se desarrolla un predictor nombrado PlantSniffer.

## Resultados

### Métodos más usados

A continuación, se discuten los métodos más usados para la clasificación de *lncRNAs* a través de predictores de aprendizaje de máquina. Para ello en la Tabla 8 se presentan los métodos agrupados según el tipo de clasificador que utilizan:

Tabla 8 - Métodos de aprendizaje de máquina más usados para la clasificación de lncRNA.

|  |  |  |
| --- | --- | --- |
| Método | Año | Clasificadores |
| Máquina de vectores de soporte (*support vector machine - SVM*) | 2007 | CPC (Kong et al., 2007) |
| **Ensamble de bosques aleatorios (*random forest ensemble*)** | **2015** | **LncRNA-ID (Achawanantakun, Chen, Sun, & Zhang, 2015)** |
| *Deep* *stacking* *networks* (*DSN*) | 2015 | lncRNA-MFDL (Fan & Zhang, 2015) |
| Redes neuronales profundas (*deep* *neural* *networks* - *DNN*) | 2016 | DeepLNC (Tripathi, Patel, Kumari, Chakraborty, & Varadwaj, 2016) |
| Máquina de vectores de soporte | 2017 | Clasificador de máquina de vectores de soporte de *lncRNA* (Schneider et al., 2017) |
| **Ensamble de bosques aleatorios** | **2017** | **PLncPRO (Singh, Khemka, Rajkumar, Garg, & Jain, 2017)** |
| **Ensamble de bosques aleatorios** | **2017** | **FEELnc** (Wucher et al., 2017) |
| **Ensamble de bosques aleatorios** | **2018** | **Predictor de *lncRNA* para plantas (Simopoulos et al., 2018)** |
| Máquina de vectores de soporte | 2018 | PlantSniffer (Vieira, 2018) |

En la Tabla 8 se resaltaron los métodos más usados (método por ensamble). Notar además que también son los más recientes. Los métodos por ensamble funcionan combinando múltiples clasificadores en un solo modelo, lo cual suele dar mayor generalización al clasificador que se obtiene como resultado y mejores tasas de predicción (Simopoulos et al., 2018).

Se encontraron también clasificadores que utilizan el modelo de máquina de vectores de soporte (*SVM*). Son métodos de aprendizaje supervisado, que requieren de la extracción de atributos de las entidades a clasificar por parte de expertos (Schneider et al., 2017). Esto difiere del método de aprendizaje profundo propuesto en la presente investigación, que incorpora de manera automática esta etapa dentro del proceso de aprendizaje.

Dentro de los métodos de aprendizaje profundo, encontramos el clasificador lncRNA-MFDL (Fan & Zhang, 2015) , que hace uso de *deep* *stacking* *networks* (*DSN*). Un segundo método de aprendizaje profundo encontrado es el clasificador DeepLNC, que hace uso de redes neuronales profundas (*DNN*), ideales para el reconocimiento de patrones (Tripathi et al., 2016).

### Principales limitaciones

Dentro de los modelos de predicción encontrados, analizaremos caso por caso sus limitaciones, prestando atención a la capacidad de los predictores de realizar la clasificación en especies desconocidas al modelo, es decir, especie bajo las cuales no han sido entrenados.

CPC (Kong et al., 2007) es uno de los primeros clasificadores popularizados. Utiliza un modelo máquina de vectores de soporte (*SVM*) para determinar la capacidad de un transcrito de ser codificante. Sin embargo, no es bueno para clasificar *lncRNAs*, ya que un transcrito con baja capacidad para codificarse no es sinónimo de ser no codificante.

LncRNA-ID (Achawanantakun et al., 2015): A través de un ensamble de bosques aleatorios se buscó construir un modelo, que se entrenó en humanos y ratones por separado (dos modelos diferentes), que pueda manejar el problema de desbalance de información (la cantidad del conjunto de datos de una clase es mucho mayor a la de otra). Es posible entrenar este modelo para otras especies, pero se desconoce su utilidad para clasificar especies desconocidas al modelo.

lncRNA-MFDL (Fan & Zhang, 2015): Este predictor utiliza *deep stacking networks* (*DSN*) con un modelo de aprendizaje profundo. Posee un link público con su implementación, sin embargo, no está disponible. El modelo se usó para predicciones en humanos y no ha sido probado en otras especies.

DeepLNC (Tripathi et al., 2016): Este clasificador utiliza redes neuronales profundas (*DNN*), y su objetivo principal es el poder armar un clasificador para una especie sin tener que hacer una selección de atributos. Esto permite entrenarlo fácilmente para otras especies, pero no se probó su utilidad al clasificar especies desconocidas al modelo. Existe un link público para acceder al clasificador, pero no está disponible.

Clasificador de máquina de vectores de soporte de *lncRNA* (Schneider et al., 2017): A través de un clasificador de máquina de vectores de soporte (*SVM*) se buscó construir un modelo que pueda generalizar sus predicciones a otras especies con buenos resultados. El modelo se entrenó con información de humanos, ratones y danios rerios (pez cebra); obteniendo buenos resultados con especies cercanas.

PLncPRO (Singh et al., 2017): A través de un ensamble de bosques aleatorios se entrenó el modelo específicamente para predecir *lncRNAs* en plantas. También se realizaron pruebas en humanos (mediante una nueva extracción de atributos). No se ha comprobado su utilidad para especies desconocidas al modelo.

FEELnc (Wucher et al., 2017): A través de un ensamble de bosques aleatorios, este clasificador propone un método alternativo para el entrenamiento del modelo cuando se poseen *PCTs*, pero no se dispone de *lncRNAs* para la especie deseada: generar artificialmente *lncRNAs* para el entrenamiento del modelo a partir de las secuenciaciones iniciales.

Predictor de *lncRNA* para plantas (Simopoulos et al., 2018): A través de un ensamble de bosques aleatorios y potenciación del gradiente (*gradient boosting*), entrenado en base a información empíricamente confirmada de animales, plantas y virus, se buscó construir un clasificador para plantas. Este predictor muestra si un transcriptor es codificante o no, con un nivel de confianza. De esta forma se puede priorizar qué transcriptores pasarán por futuras validaciones empíricas, y retroalimentar el modelo con esta nueva información para que mejore continuamente.

PlantSniffer (Vieira, 2018): A través de un ensamble de máquina de vectores de soporte, cuyos resultados se cruzan contra BLAST, se busca predecir *lincRNAs* en plantas para las que no existen bases de datos de genomas y/o transcriptomas (caña de azúcar y maíz), utilizando en su lugar información de especies cercanas.

## Conclusiones

En los últimos años los proyectos de secuenciación de transcriptomas están generando una enorme cantidad de información que necesita ser explotada. La anotación y clasificación manual de los transcritos de estos proyectos es lenta y costosa. En este contexto los métodos computacionales de inteligencia artificial y aprendizaje de máquina son ventajosos, ya que se apoyan en la gran cantidad de información producida y generan resultados rápidos y sin mucho costo (Jabeen et al., 2018).

Dentro de los métodos de clasificación existentes, nos interesa saber su utilidad para la clasificación de especies desconocidas al modelo, y si se tiene acceso al clasificador con la intención de poder realizar comparaciones.

Los métodos más usados en los últimos años son los ensambles, debido a que combinan varios clasificadores y permiten evitar problemas de sobreajuste (ser muy específicos, del inglés *overfitting*). Aunque aún no son tan usados, los métodos de aprendizaje profundo también están ganando popularidad, ya que la extracción de atributos (en la que normalmente participa un experto) es un paso que realiza el algoritmo durante su entrenamiento, aprendiendo los atributos por su cuenta a partir de la información de la que se alimenta.

Sólo se encontraron dos modelos orientados a generalizar sus resultados a otras especies – el clasificador de máquina de vectores de soporte de *lncRNA* (Schneider et al., 2017) y PlantSniffer (Vieira, 2018).

Los clasificadores suelen construirse con el objetivo de tener las mejores predicciones para las especies bajo las cuales son entrenadas. La carencia de estudios de modelos que intenten clasificar especies en base a información de otras especies, motiva la presente investigación de un modelo predictivo que pueda usarse en especies diferentes a la que se posee en el conjunto de entrenamiento, pudiendo estas ser nuevas especies.

# Recopilación y preprocesamiento de datos

## Fuentes de información

El conjunto de datos está compuesto de transcritos identificados como ARN codificantes y no codificantes largos. Para la extracción de ARN codificante (PCT) se utilizó como fuente de información Ensembl (Zerbino et al., 2018), mientras que para los ARN no codificantes largos (lncRNA) se utilizaron las bases de datos especializadas en plantas CantataDB (Szcześniak et al., 2016) y GreeNC (Paytuví Gallart et al., 2016).

Se construirá un modelo de clasificación binaria en base a la información de varias especies ya conocidas, con el objetivo de realizar predicciones sobre especies no modeladas. Se identifica como la clase positiva los ARN no codificantes largos, y como clase negativa los ARN codificantes.

Cada fuente de información posee un conjunto de datos para diferentes especies (ver Tabla 9). En este contexto, se desea que el conjunto de datos sea balanceado, por lo que el número de transcritos de cada especie a utilizar para la construcción del modelo deberá ser el mismo. Inclusive, dentro de cada especie, el número de casos positivos y negativos tendrá que ser igual.

Tabla - Número de especies en las bases de datos de transcriptomas

|  |  |  |
| --- | --- | --- |
| Fuente | Tipo | Conteo de especie |
| Ensembl v43 | PCT (clase negativa) | 59 especies |
| CantataDB v2.0 | lncRNA (clase positiva) | 43 especies |
| GreeNC v1.12 | lncRNA (clase positiva) | 45 especies |

## Selección de especies

Con el objetivo de construir un modelo que generalice adecuadamente a especies no modeladas, interesa cumplir con cuatro restricciones y/u objetivos:

* Elegir el mismo número de transcritos de cada especie.
* Dentro de cada especie, utilizar el mismo número de casos positivos y negativos.
* Hacer uso de la mayor cantidad de especies.
* Hacer uso del mayor número de transcritos.

En el conjunto de datos extraído, el número de ARN codificantes disponibles es mucho mayor al número de ARN no codificantes largos disponibles, por lo que se realizará la contabilización de la clase positiva, lncRNA.

Se debe tomar en cuenta que, para una misma especie, puede existir información de ARN no codificante tanto en CantataDB como en GreeNC. En estas casuísticas, se optó por utilizar solamente los transcritos de la fuente que posea el mayor número de transcritos, descartando los transcritos de la otra.

En base a lo anterior, en la Tabla 10 se pueden visualizar las 10 especies con la mayor cantidad de ARN no codificantes disponibles, junto con la fuente de dónde provienen estos transcritos.

Tabla - Especies con el mayor conteo de clases positivas, y sus fuentes

|  |  |  |  |
| --- | --- | --- | --- |
| Especie | Conteo PCT (clase negativa) | Conteo lncRNA (clase positiva) | Fuente de los lncRNA |
| Triticum aestivum | 107,545 | 38,820 | GreeNC |
| Brassica napus | 101,040 | 12,010 | CantataDB |
| Oryza rufipogon | 37,071 | 10,261 | CantataDB |
| Trifolium pratense | 39,917 | 10,179 | CantataDB |
| Physcomitrella patens | 32,447 | 9,690 | GreeNC |
| Medicago truncatula | 50,444 | 9,676 | GreeNC |
| Manihot esculenta | 33,044 | 9,504 | CantataDB |
| Oryza nivara | 36,313 | 8,955 | CantataDB |
| Brassica rapa | 41,018 | 8,501 | CantataDB |
| Hordeum vulgare | 37,705 | 7,970 | CantataDB |

En base a la tabla anterior, si se elige trabajar con 3 especies, se seleccionarían las primeras tres especies de la tabla, hasta Oryza rufipogon. El límite en la cantidad de clases negativas y positivas a utilizar por especie sería establecido por la cantidad de lncRNA disponibles en esta última especie (10,261). Entonces, el número total de transcritos a utilizar sería 10,261 multiplicado por 2 (tomando en cuenta son 10,261 transcritos para la clase positiva y 10,261 para la clase negativa) y multiplicado por 3 (debido al número de especies seleccionadas), totalizando 61,566 transcritos.

En cambio, si se opta por utilizar las 10 especies listadas, esta vez el límite viene dado por la última especie, Hordeum vulgare, y el total de transcritos a utilizar en el modelo sería 7,970 (transcritos en Hordeum vulgare) x 2 (clase positiva y negativa) x 10 (número de especies), en total 159,400 transcritos.

De esta forma, al ordenar las especies según el número de transcritos ARN no codificantes que tienen disponibles, se puede construir la fórmula del total de transcritos que se utilizaría en el modelo. Al seleccionar una de ellas, la fórmula vendría dada por la cantidad de transcritos ARN no codificantes que posee, multiplicado por dos y multiplicado por su posición o ranking:

Transcritos lncRNA de la especie x 2 x Ranking de la especie

Si se realiza esta operación para todas las especies disponibles, ordenándolas de mayor a menor de acuerdo al resultado de la fórmula anterior, se obtiene como resultado lo mostrado en la Tabla 11 (se muestran los cinco primeros resultados).

Tabla - Número total de transcritos a utilizar según el modelo seleccionado

|  |  |  |  |
| --- | --- | --- | --- |
| Especie | Ranking | Conteo lncRNA | Número de transcritos total para el modelo |
| Oryza sativa Japonica Group | 25 | 5,237 | 261,850 |
| Setaria italica | 30 | 4,208 | 252,480 |
| Musa acuminata | 31 | 4,071 | 252,402 |
| Theobroma cacao | 24 | 5,256 | 252,288 |
| Vitis vinifera | 27 | 4,542 | 245,268 |

El número de transcritos a utilizar, viene maximizado al seleccionar las primeras 25 especies hasta Oryza sativa Japonica Group, especie para la cual se poseen 5,237 lncRNA, con lo cual el modelo utilizaría un total de 261,850 transcritos.

Sin embargo, si se selecciona el siguiente candidato, Setaria itálica, se tendría como total de transcritos disponibles para la construcción del modelo 252,480. Serían 9,370 transcritos menos, pero con la ventaja de aumentar la variedad de especies a utilizar de 25 a 30. Como se mencionó anteriormente, además de utilizar el mayor número de transcritos disponibles, otro de los objetivos en la selección de especies es maximizar el número de especies utilizadas.

Debido a lo anterior, se optó por utilizar 30 especies para el modelo, teniendo cada especie un total de 4,208 instancias en la clase positiva y 4,208 instancias en la clase negativa. Se presenta el resumen de las especies seleccionadas y la fuente de donde se obtendrán los ARN no codificantes para las mismas (la obtención de ARN codificantes será siempre de Ensembl):

* E1: Amborella trichopoda (GreeNC)
* E2: Arabidopsis lyrata (CantataDB)
* E3: Arabidopsis thaliana (CantataDB)
* E4: Brachypodium distachyon (GreeNC)
* E5: Brassica napus (CantataDB)
* E6: Brassica oleracea (CantataDB)
* E7: Brassica rapa (CantataDB)
* E8: Corchorus capsularis (CantataDB)
* E9: Cucumis sativus (CantataDB)
* E10: Glycine max (GreeNC)
* E11: Gossypium raimondii (GreeNC)
* E12: Hordeum vulgare (CantataDB)
* E13: Leersia perrieri (CantataDB)
* E14: Manihot esculenta (CantataDB)
* E15: Medicago truncatula (GreeNC)
* E16: Oryza barthii (CantataDB)
* E17: Oryza brachyantha (CantataDB)
* E18: Oryza nivara (CantataDB)
* E19: Oryza rufipogon (CantataDB)
* E20: Oryza sativa Japonica Group (GreeNC)
* E21: Physcomitrella patens (GreeNC)
* E22: Populus trichocarpa (GreeNC)
* E23: Setaria italica (CantataDB)
* E24: Solanum lycopersicum (CantataDB)
* E25: Solanum tuberosum (GreeNC)
* E26: Sorghum bicolor (GreeNC)
* E27: Theobroma cacao (CantataDB)
* E28: Trifolium pratense (CantataDB)
* E29: Triticum aestivum (GreeNC)
* E30: Vitis vinífera (CantataDB)

## Método de obtención de las fuentes

### Ensembl

Para el acceso a Ensembl Plants ([http://plants.ensembl.org](http://plants.ensembl.org/)) se utilizó el paquete en R biomaRt (Durinck et al., 2005) para migrar los transcritos de las especies a una base de datos. Los parámetros utilizados para la extracción de las clases negativas fueron:

* Como atributos a extraer se especificó ensembl\_transcript\_id y transcript\_exon\_intron. Esto quiere decir que se indicó como campos devueltos en la llamada a biomaRt el identificador del transcrito, y la secuencia de ARN completa.
* Como filtros utilizados, se usó transcript\_biotype, con el valor protein\_coding. Esto quiere decir que se filtraron los transcritos que codifican en ARN mensajero.
* Se extrajo la información de las bases de datos de cada una de las especies seleccionadas.

Se presenta en el Anexo B el script utilizado para la descarga de la información de Ensembl, a través del paquete biomaRt.

Sin embargo, debido a la cantidad de información contenida en la consulta, la extracción de la información no se pudo realizar enteramente de forma automática, por lo cual se tuvo que descargar manualmente la información de la web [https://plants.ensembl.org/biomart/ martview/](https://plants.ensembl.org/biomart/%20martview/), en archivos en formato fasta, para luego procesarlas a través de un script, disponible en el Anexo C.

Se limitó la longitud de las secuencias importadas a un máximo de 65,000 nucleótidos. Aquellas secuencias con una longitud mayor fueron descartadas.

### CantataDB

Los transcritos de CantataDB se descargaron en formato fasta de la web <http://cantata.amu.edu.pl/>, y se importaron a la base de datos, descartando secuencias con una longitud mayor a 65,000 nucleótidos. En el Anexo D se encuentra el script utilizado para procesar estos transcritos descargados.

### GreeNC

Los transcritos de GreeNC se descargaron en formato fasta de la web [http://greenc.sciencedesigners.com](http://greenc.sciencedesigners.com/), y se importaron a la base de datos, descartando secuencias con una longitud mayor a 65,000 nucleótidos. En el Anexo E se encuentra el script utilizado para procesar estos transcritos descargados.

## Selección de transcritos a utilizar en el modelo

Se realizó la descarga de información de todas las especies y transcritos disponibles en cada una de las fuentes listadas. Dentro de cada especie, se procedió con el proceso de selección de los 4208 transcritos de ARN codificante y 4208 transcritos de ARN no codificante a utilizar para la construcción del modelo. El criterio de selección de los transcritos fue completamente aleatorio. Se anexa el script utilizado para este proceso (ver Anexo F).

## Selección de características a utilizar para el modelo

Para la representación de los transcritos a través de características (en inglés *features*), se hizo uso de una de las fuentes referenciadas durante la revisión del estado del arte, el predictor de lncRNA para plantas (Simopoulos et al., 2018), que tiene código fuente disponible en <https://github.com/gbgolding/crema>.

En el trabajo referenciado, las características utilizadas fueron:

1. Longitud del transcrito
2. Longitud del marco abierto de lectura (del inglés *open reading frame*, u *ORF*) y porcentaje del marco abierto de lectura con respecto a la longitud del transcrito
3. Porcentaje de nucleótidos GC presentes
4. Puntaje Fickett
5. Puntaje de hexámeros
6. Puntaje de identidad contra la base de datos Swiss-Prot
7. Longitud del alineamiento contra la base de datos Swiss-Prot
8. Proporción de la longitud del alineamiento con respecto a la longitud del transcrito
9. Proporción de la longitud del alineamiento con respecto a la longitud del marco abierto de lectura
10. Presencia de elementos transponibles (descartado)
11. Divergencia porcentual de la secuencia con respecto a los elementos transponibles (descartado)

El cálculo de características como la longitud del transcrito y el porcentaje de nucleótidos GC se puede realizar a través de scripts relativamente simples.

Por otro lado, para el cálculo del marco abierto de lectura, su porcentaje con respecto a la longitud del transcrito, el puntaje Fickett y el puntaje de hexámeros, se utilizó el programa CPAT (Wang et al., 2013).

Características relacionadas al alineamiento contra la base de datos Swiss-Prot se realizaron con la aplicación del algoritmo Diamond (Buchfink, Xie, & Huson, 2015).

Las características relacionadas a elementos transponibles fueron extraídas con RepeatMasker (<http://www.repeatmasker.org/>), pero en el trabajo referenciado fueron descartadas debido a que de hecho afectaban negativamente el desempeño de los clasificadores (Simopoulos et al., 2018).

### Características calculadas por scripts propios

Se calcularon dos características a través de scripts propios: la longitud del transcrito y el porcentaje de nucleótidos GC.

Se anexa el script utilizado para el cálculo de la longitud del transcrito (ver Anexo G). A pesar de no ser una característica especialmente discriminante, es frecuentemente utilizado en la literatura, ya que se puede usar en conjunción con otras características.

El porcentaje de nucleótidos GC (se adjunta script en el Anexo H) es otra característica comúnmente utilizada en la literatura (Simopoulos et al., 2018).

### Características calculadas a través de CPAT

#### Definición de CPAT

CPAT es una herramienta orientada a predecir el potencial codificante de transcritos (Wang et al., 2013). Al utilizar como entrada un archivo en formato fasta, CPAT genera como salida sus predicciones con respecto al potencial codificante de cada transcrito que contiene, pero junto a esta información genera características calculadas de estos transcritos, que son de interés para el modelo del presente trabajo de investigación: longitud del marco abierto de lectura, porcentaje de la longitud del marco abierto de lectura con respecto a la longitud total del transcrito, puntaje Fickett y el puntaje de hexámeros.

CPAT realiza sus predicciones a través de un modelo de regresión logística, que debe ser construido previo al uso de la herramienta. Este modelo se genera con transcritos ARN no codificantes largos, transcritos de secuencias codificantes (CDS) y transcritos ARN codificantes (ver Figura 9).

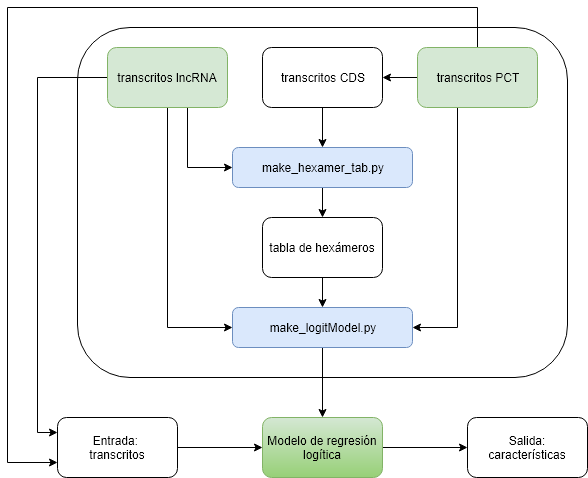


Figura - Generación del modelo de regresión logística para CPAT. Elaboración propia.

El modelo de regresión logística de CPAT será dependiente del conjunto de datos utilizado para su generación. Si generamos un único modelo de regresión logística CPAT en base a las 30 especies seleccionadas, al realizar la validación cruzada existirá fuga de información del conjunto de datos utilizado como validación, ya que este último habría sido utilizado para generar el modelo de regresión logística de CPAT. Entonces, dentro de cada iteración de la validación cruzada será necesario generar un modelo de regresión logística para CPAT ajustado al conjunto de datos utilizado como entrenamiento en la iteración actual de la validación cruzada.

#### Características extraídas

El marco abierto de lectura son secciones del transcrito con posibilidad de ser codificados, que suelen iniciar con codón de inicio (AUG) y se encuentran limitadas por un codón de parada. El marco abierto de lectura es una característica comúnmente utilizada en modelos predictivos de ARN no codificantes y ARN codificantes (Wang et al., 2013). Un transcrito puede poseer múltiples marcos abiertos de lectura, por lo que CPAT utilizará la longitud del más largo.

El puntaje Fickett (en inglés *Fickett Testcode score*) se basa en la composición de cada nucleótido dentro del ARN codificante y no codificante, combinado con la preferencia del uso de cada nucleótido en una de las tres posiciones que puede ocupar en un codón (Fickett, 1982).

El puntaje de hexámeros (en inglés *hexamer usage bias*) es calculado en base a la probabilidad de la aparición de cada secuencia de seis nucleótidos en un ARN codificante/no codificante (Wang et al., 2013).

#### Requerimientos para CPAT

Instrucciones para la instalación y uso de CPAT se encuentran disponibles en <http://rna-cpat.sourceforge.net/>. Para hacer uso de CPAT se requieren las siguientes entradas:

* Archivo en formato fasta con los transcritos a predecir
* Tabla de frecuencias de hexámeros
* Modelo de regresión logística

La tabla de frecuencia de hexámeros y el modelo de regresión logística deben ser generados según el conjunto de datos de entrenamiento utilizado. CPAT ya tiene estas entradas preconstruidas para algunas especies mamíferas, pero para la presente investigación tendrán que ser generados (ver Figura 9).

#### Tabla de frecuencia de hexámeros

Para la generación de la tabla de frecuencias de hexámeros, se requiere como entrada las secuencias codificantes de los ARN codificantes (diferente de los transcritos de los ARN codificantes ya descargados de Ensembl, ver Figura 10), y los transcritos de ANR no codificantes. En el Anexo I se muestra el script utilizado para generar los archivos en formato fasta de estas dos entradas.

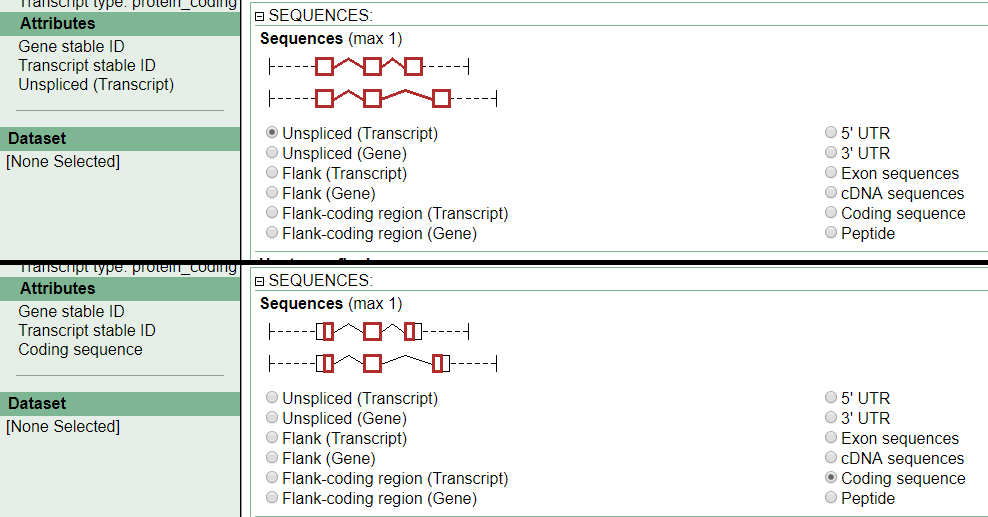


Figura - Diferencia visualizada en Biomart (Ensembl) entre transcrito y secuencia codificante. Elaboración propia.

Con los archivos de entradas preparados, se debe ejecutar el script de CPAT “make\_hexamer\_tab.py” para generar la tabla de frecuencias de hexámeros. En el Anexo J se tiene el script utilizado para la generación de la tabla de frecuencia de hexámeros.

La salida de este script es un archivo con la tabla de hexámeros calculada (ver Figura 9). En esta tabla se pueden apreciar todas las combinaciones posibles de secuencias de 6 nucleótidos, acompañadas de la probabilidad de que formen parte de una secuencia codificante o un ARN no codificante. Se presenta un ejemplo en la Tabla 12. Todas las entradas en la segunda columna con la probabilidad de que la secuencia forme parte de secuencias codificantes, sumarán en total 1. De forma similar, todas las entradas en la tercera columna para la probabilidad de formar parte de ARN no codificantes también suman 1.

Tabla - Ejemplo de tabla de frecuencia de hexámeros para Amborella trichopoda generada por CPAT

|  |  |  |
| --- | --- | --- |
| Hexamer | coding | noncoding |
| AAAAAA | 0.0004820376938412906 | 0.00044355770099859063 |
| AAAAAC | 0.0003494773280349357 | 0.00030714278540845805 |
| AAAAAG | 0.0006740995874811799 | 0.0005264109319398369 |
| … | … | … |
| TTTTTC | 0.00030277992644406066 | 0.00027868814043873716 |
| TTTTTG | 0.000454923073562718 | 0.0003657258779931776 |
| TTTTTT | 0.0003095585815137038 | 0.00024270138356526658 |

#### Modelo de regresión logística

La generación del modelo de regresión logística requiere como entrada la tabla de hexámeros anteriormente calculada, los ARN codificantes (en esta ocasión el transcrito completo y no solamente la secuencia codificante) y los ARN no codificantes (ver Figura 9).

El archivo en formato fasta para ARN no codificantes ya se había generado con anterioridad, para la construcción de la tabla de hexámeros. En el Anexo K se puede visualizar el script utilizado para la generación del archivo en formato fasta de ARN codificantes faltante.

Para generar el modelo de regresión logística, se hace uso del script de CPAT “make\_logitModel.py”. Se anexa el script utilizado para la generación de la regresión logística en el Anexo L.

#### Ejecución de CPAT

Teniendo todos los archivos de entrada preparados (archivos en formato fasta a predecir, archivo de tabla de frecuencias de hexámeros y modelo de regresión logística), procedemos con la ejecución de CPAT para obtener como salida las características de interés de los transcritos de entrada. Se anexa el script utilizado para ello en el Anexo M.

### Características calculadas a través de Diamond

#### Definición de Diamond

Diamond (Buchfink et al., 2015) es un algoritmo que realiza búsquedas de alineamientos en base de datos de proteínas de manera rápida, en comparación con otras herramientas disponibles (Simopoulos et al., 2018).

#### Características Extraídas

Las características extraídas mediante el algoritmo Diamond serán aquellas relacionadas con alineamientos contra la base de datos de proteínas Swiss-Prot, específicamente: el puntaje de identidad, la longitud del alineamiento, la proporción de la longitud del alineamiento con respecto a la longitud del transcrito, y la proporción de la longitud del alineamiento con respecto a la longitud del marco abierto de lectura antes calculado.

El puntaje de identidad se refiere al porcentaje del transcrito encontrado en la búsqueda en la base de datos Swiss-Prot. Por otro lado, la longitud del alineamiento se refiere a la longitud de la coincidencia, y las dos medidas restantes (proporción con respecto a la longitud del transcrito, y proporción con respecto a la longitud del marco abierto de lectura) se obtienen de características previamente calculadas.

#### Requerimientos para Diamond

Para ejecutar Diamond se requieren dos pasos. El primero será obtener una base de datos de proteínas en un archivo en formato fasta, que será utilizado como entrada para la generación de una base de datos en un formato propio utilizado por el algoritmo. Con esta base de datos construida, como segundo paso, podemos tomar como entrada los transcritos para los que se realizará la búsqueda de alineamientos, ejecutar Diamond, y tomar como salida los puntajes de alineamientos.

#### Construcción de la base de datos de proteínas desde Swiss-Prot

Para la construcción de la base de datos de proteínas, se descargó esta información desde UniProt (<https://www.uniprot.org/>), específicamente de la base de datos con anotaciones manuales Swiss-Prot. Dado que el presente trabajo se enfoca en realizar predicciones en plantas, se filtró la información para incluir solamente esta especie (taxonomía Viridiplantae). Se puede visualizar el resultado de la búsqueda en [https://www.uniprot.org/uniprot/?query=taxonomy:viridiplantae&fil=reviewed%3Ayes&sort=score](https://www.uniprot.org/uniprot/?query=taxonomy:viridiplantae&fil=reviewed%3Ayes&%20sort=score).

La descarga del resultado de la búsqueda genera un archivo en formato fasta, que es utilizado para generar la base de datos de proteínas a través del comando de Diamond “makedb”. Se muestra el script utilizado para esta tarea en el Anexo N.

#### Ejecución de Diamond

Con el objetivo de obtener las características deseadas, se realizó la búsqueda de alineamientos de los ARN codificantes y ARN no codificantes en la base de datos generada con el comando “makedb”. Se anexa el script utilizado para esta sección en el Anexo O.

Los parámetros utilizados en la ejecución de Diamond serán similares a los utilizados en el predictor lncRNA para plantas (Simopoulos et al., 2018) referenciado. El resultado del comando será un archivo tabulado, con todas las coincidencias encontradas, por lo que cada transcrito puede aparecer en ninguna, en una, o en varias filas, según el número de coincidencias encontradas. Entonces, para transformar esta información en características, será necesario un script adicional.

#### Extracción de características desde el archivo de resultados

Se adjunta en el Anexo P el script utilizado para procesar el resultado de la búsqueda de alineamientos a través de Diamond. Para un transcrito, el script buscará el alineamiento con el mayor puntaje asignado por Diamond, y serán las características de identidad y longitud del alineamiento de este alineamiento los asignados al transcrito.

## Resumen

Se utilizó como fuente de información de transcriptomas ARN codificantes la base de datos Ensembl, y para información de transcriptomas ARN no codificantes las bases de datos CantataDB y GreeNC.

El conjunto de datos para el entrenamiento del modelo se originó de las 30 especies con la mayor cantidad disponible de transcritos. Por cada especie se utilizaron 4208 transcritos de ARN codificante, y 4208 transcritos de ARN no codificantes, utilizando un total de 252,480 transcritos.

Las características extraídas de los transcritos fueron la longitud del transcrito, el porcentaje de nucleótidos GC, características extraídas de la herramienta CPAT como aquellas relacionadas al marco abierto de lectura, al puntaje de hexámeros y al puntaje Ficket; y características de alineamientos contra Swiss-Prot obtenidas con la herramienta Diamond.

Se presenta en la Figura 11, el resumen de los resultados del primer objetivo.

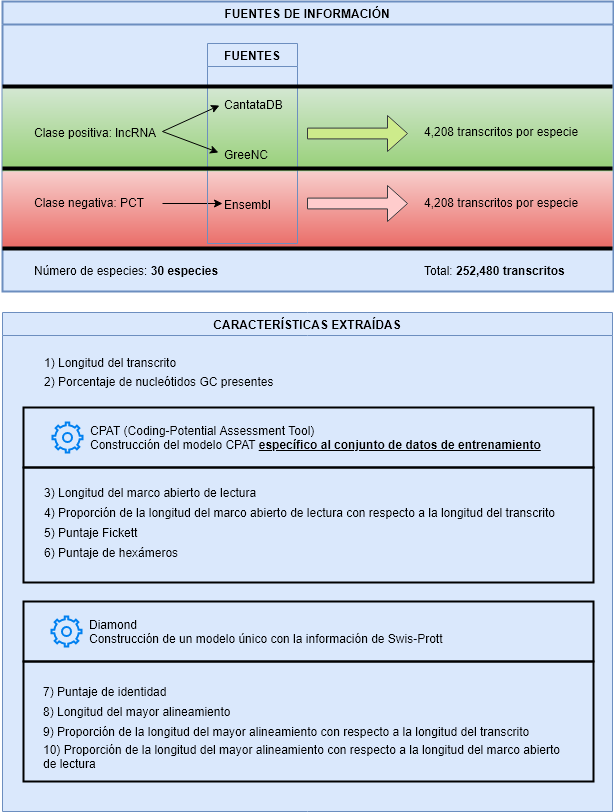


Figura - Resumen de la extracción y preprocesamiento de los datos. Elaboración propia.

# Modelo implementado

## Modelos referenciales

Como paso previo a la implementación del modelo de la presente investigación, se generaron 30 modelos, cada uno entrenado con datos de una sola especie. Sus medidas de precisión se utilizarán como referencia, para realizar comparaciones con las medidas obtenidas en el modelo entrenado con múltiples especies.

Se espera que la precisión de estos modelos de referencia sea mejor, ya que para su entrenamiento utilizarán únicamente datos de la especie que predicen, pero también se desea obtener información de la diferencia de la precisión de estos modelos con respecto a la precisión del modelo final entrenado con múltiples especies.

Para cada especie, la información de sus 8416 transcritos se utilizó para el entrenamiento y validación del modelo mediante la validación cruzada, con k igual a 10. Sin embargo, en cada iteración de la validación cruzada, se generó un modelo CPAT específico al conjunto de datos utilizado como entrenamiento, con el objetivo de evitar la fuga de información del conjunto de datos de validación. Con esto podremos extraer las características y alimentarlas al modelo (en la iteración de la validación cruzada actual).

Adicionalmente, se realizó la búsqueda de hiperparámetros a través del muestreo de cuadrícula (del inglés *grid search*). De la mejor selección de parámetros, se utilizó el promedio de las 10 medidas generadas en la validación cruzada, como medida final del modelo. Este resultado se utilizará como referencia para compararla con el modelo de especies múltiples generado en la presente investigación.

Se puede visualizar en la Figura 12 un resumen de la arquitectura del modelo referencial para una de las especies, Amborella trichopoda.

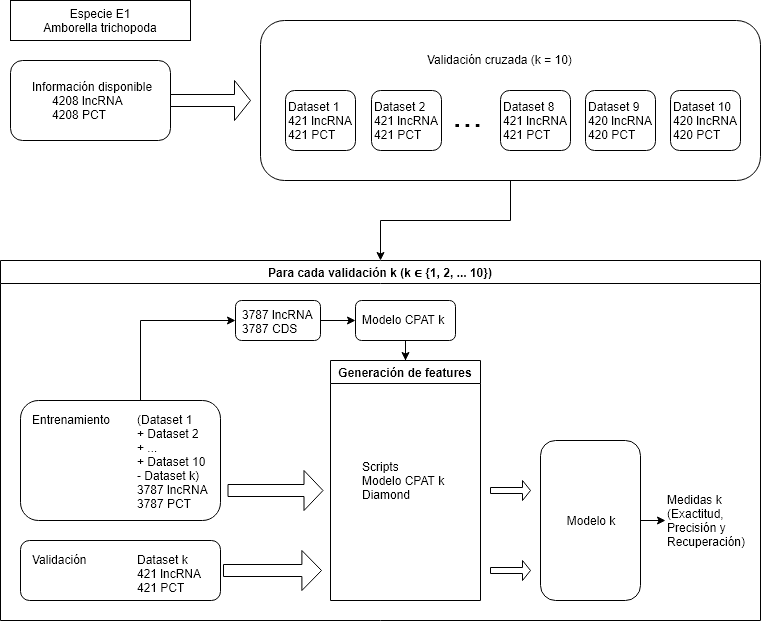


Figura - Arquitectura de modelos de referencia, tomando como ejemplo una especie. Elaboración propia.

## Resultados de los modelos referenciales

Para los modelos referenciales de cada una de las 30 especies seleccionadas, se entrenaron máquinas de soporte de vectores con kernel de función de base radial (*RBF*), valor gamma de 1e-3, y selección del mejor valor C a través de muestreo de cuadrícula con opciones 0.1, 0.5, 0.9 y 2.

La exactitud promedio lograda a través de los modelos referenciales generados para las 30 especies fue de 89.9% (ver Figura 13).

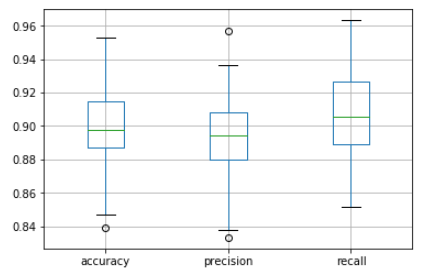


Figura - Gráfico de cajas de la exactitud, precisión y recuperación de los 30 modelos referenciales. Elaboración propia.

Se presenta en la Tabla 13 el detalle de los resultados para todas las especies. La menor exactitud (83.92%), precisión (83.31%) y recuperación (85.14%) se obtuvieron con la especie Solanum tuberosum (papa o patata).

Tabla - Resultados de exactitud, precisión y recuperación de los modelos referenciales

|  |  |  |  |
| --- | --- | --- | --- |
| Especie | Exactitud | Precisión | Recuperación |
| Amborella trichopoda | 0.901616 | 0.857721 | 0.963403 |
| Arabidopsis lyrata | 0.893061 | 0.896691 | 0.891635 |
| Arabidopsis thaliana | 0.929183 | 0.930819 | 0.927519 |
| Brachypodium distachyon | 0.899596 | 0.887825 | 0.916112 |
| Brassica napus | 0.878089 | 0.868310 | 0.891397 |
| Brassica oleracea | 0.882605 | 0.874581 | 0.894011 |
| Brassica rapa | 0.901141 | 0.908450 | 0.892586 |
| Corchorus capsularis | 0.884981 | 0.883904 | 0.886407 |
| Cucumis sativus | 0.874525 | 0.864836 | 0.888546 |
| Glycine max | 0.916825 | 0.908439 | 0.927281 |
| Gossypium raimondii | 0.931440 | 0.928359 | 0.935361 |
| Hordeum vulgare | 0.891160 | 0.896778 | 0.884268 |
| Leersia perrieri | 0.937381 | 0.936394 | 0.938688 |
| Manihot esculenta | 0.880466 | 0.886434 | 0.873337 |
| Medicago truncatula | 0.903042 | 0.883845 | 0.928470 |
| Oryza barthii | 0.895556 | 0.885706 | 0.908745 |
| Oryza brachyantha | 0.918132 | 0.911707 | 0.926093 |
| Oryza nivara | 0.907913 | 0.907365 | 0.908745 |
| Oryza rufipogon | 0.899358 | 0.903400 | 0.894724 |
| Oryza sativa Japonica Group | 0.921459 | 0.931990 | 0.909458 |
| Physcomitrella patens | 0.892942 | 0.870535 | 0.923479 |
| Populus trichocarpa | 0.926331 | 0.909877 | 0.946768 |
| Setaria italica | 0.846958 | 0.837562 | 0.860979 |
| Solanum lycopersicum | 0.902091 | 0.901187 | 0.903755 |
| Solanum tuberosum | 0.839235 | 0.833151 | 0.851473 |
| Sorghum bicolor | 0.890090 | 0.878365 | 0.906844 |
| Theobroma cacao | 0.952947 | 0.956685 | 0.948907 |
| Trifolium pratense | 0.886288 | 0.889651 | 0.882367 |
| Triticum aestivum | 0.893774 | 0.898691 | 0.887833 |
| Vitis vinífera | 0.893893 | 0.892367 | 0.895913 |

## Arquitectura del modelo de especies múltiples

Para el entrenamiento y validación del modelo, se utilizaron los 252,480 transcritos de las 30 especies disponibles. A través de la validación cruzada, se dividió el conjunto de datos en 30 particiones, teniendo cada partición información de una sola especie. Entonces, los datos de entrenamiento en la validación cruzada estuvieron conformados por 122,032 transcritos ARN no codificantes y 122,032 transcritos codificantes de 29 especies, mientras que los datos de validación por 4208 transcritos ARN no codificantes y 4208 transcritos ARN codificantes de una sola especie.

En cada iteración de la validación cruzada, fue necesario generar un modelo CPAT ajustado al conjunto de datos de entrenamiento, para evitar fuga de información del conjunto de datos de validación. Para ello, de la clase negativa de los datos de entrenamiento (ARN codificante) se extrajeron las secuencias codificantes (CDS), y utilizando la clase positiva (122,032 lncRNA), la clase negativa (122,032 PCT) y las secuencias codificantes (122,032 CDS) se generaron los modelos CPAT para cada iteración.

Se realizó la búsqueda de los mejores hiperparámetros a través del muestreo de cuadrícula. Del modelo con los mejores hiperparámetros resultantes, se realizó la comparación de las medidas de precisión en la iteración de la validación cruzada que utilizó a la especie E como conjunto de datos de validación, contra los resultados de la precisión obtenida en el modelo referencial para esta misma especie E. Se realizó esta comparación para las 30 especies.

Finalmente, se ajustó el modelo final con los datos de las 30 especies (con su propio modelo CPAT generado a partir del conjunto de datos completo de las 30 especies para la extracción de características), para obtener el modelo final utilizado para comparaciones contra otro modelo del estado del arte.

En la Figura 14 se puede ver un resumen de la arquitectura utilizada para el modelo de especies múltiples.

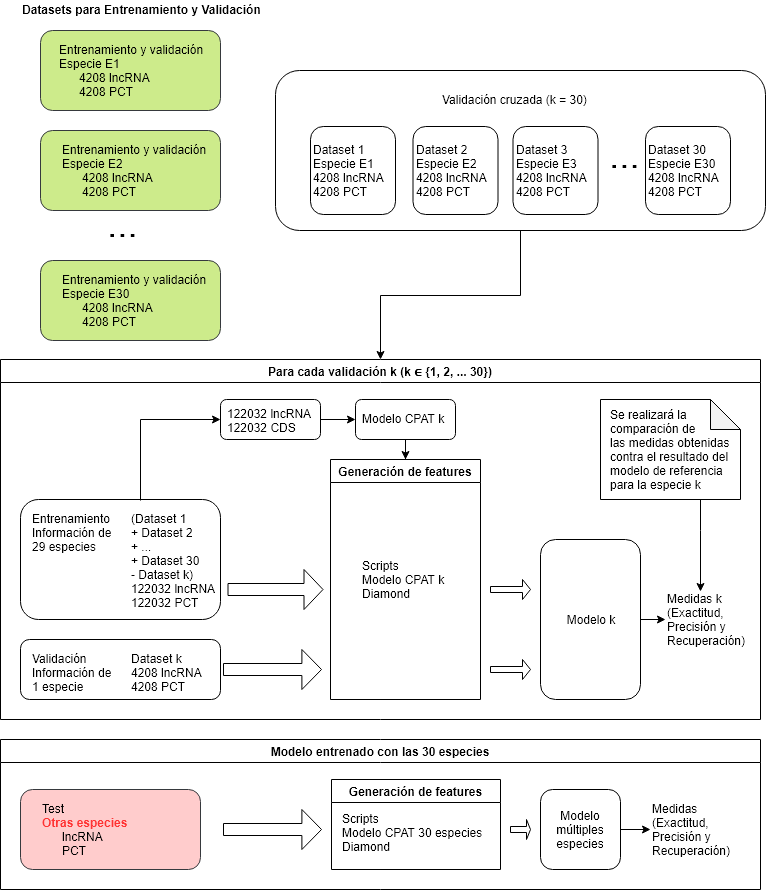


Figura - Arquitectura del modelo de especies múltiples. Elaboración propia.

## Resultados del modelo de especies múltiples

Para el modelo final, se entrenó una red neuronal, realizando la búsqueda de los mejores hiperparámetros a través del muestreo de cuadrículas. Los hiperparámetros buscados fueron el número de épocas (del inglés *epochs*), la función de activación de la última capa, la probabilidad de deserción (del inglés *dropout*), la limitación de pesos en las capas ocultas (logrado en Keras mediante el parámetro kernel\_constraint de la clase Dense), la cantidad de capas ocultas adicionales (la red neuronal siempre tendrá una capa oculta, si este parámetro es fijad en uno, tendría dos capas ocultas) y el número de neuronas para las capas ocultas adicionales. En la Tabla 14 se muestra un resumen de todas las combinaciones buscadas. Otros hiperparámetros que se fijaron en un solo valor son el tamaño de lote para la actualización de parámetros del modelo (en inglés *batch size*, para el cual se usó un tamaño de 1024), el optimizador (Adam, con *learning rate* por defecto de 0.001), la función de activación de la capa de entrada y las capas intermedias (unidades de rectificación lineal, o *relu*), la inclusión de una capa oculta por defecto y la cantidad de neuronas para esta primera capa oculta.

Tabla - Hiperparámetros buscados para el modelo final

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Número de epochs | Activación de la última capa | Probabilidad de dropout | Maxnorm para los pesos | Capas ocultas adicionales | Neuronas en capas ocultas |
| 1024 | sigmoid | 0 | 1 | 1 | 10 |
| 128 | sofmax | 0.3 0.6 0.9 | 2 4 | 1 2 | 10 |
| 128 | softmax | 0.3 0.6 | 2 4 | 4 | 100 |

El mejor resultado se obtuvo al utilizar 1024 épocas, con la función de activación sigmoidal en la útima capa, sin dropout, regularización maxnorm para los pesos con valor igual a uno, una capa oculta adicional, y 10 neuronas para la capa oculta. Se obtuvieron métricas de exactitud de 87.79%, precisión de 83.515 y recuperación de 94.47%. Entonces, el mejor modelo posee una capa de entrada de 10 dimensiones (cantidad de características), dos capas ocultas de 10 neuronas cada una con unidades de rectificación lineal como función de activación, una capa de salida de una neurona con función de activación sigmoidal.

Las librerías generadas y utilizadas para la generación de los modelos referenciales y finales se puede encontrar en el Anexo Q, Anexo R, Anexo S y Anexo T. El código con la generación de los modelos, que hacen uso de las librerías anteriormente mencionadas, se encuentra en el Anexo U.

## Comparación modelos referenciales contra modelo final

Como se mencionó, se realizará la comparación de la exactitud los modelos referenciales contra el modelo final. Para realizar la comparación, se tomaron los resultados del modelo de una especie, y se compararon contra los resultados de la validación cruzada del modelo final al utilizar la mejor combinación de hiperparámetros, en específico, se compara contra las medidas obtenidas de la validación cruzada en la iteración en la que la especie en referencia fue utilizada como conjunto de datos de validación. Se presenta esta comparación en la Tabla 15.

Tabla - Comparación de modelos referenciales y modelos finales

|  |  |  |
| --- | --- | --- |
| Especie | Modelo referencial | Modelo final |
| Amborella trichopoda | 0.901616 | 0.8159136 |
| Arabidopsis lyrata | 0.893061 | 0.87152433 |
| Arabidopsis thaliana | 0.929183 | 0.91857001 |
| Brachypodium distachyon | 0.899596 | 0.90355015 |
| Brassica napus | 0.878089 | 0.86953823 |
| Brassica oleracea | 0.882605 | 0.85985601 |
| Brassica rapa | 0.901141 | 0.87226912 |
| Corchorus capsularis | 0.884981 | 0.86283515 |
| Cucumis sativus | 0.874525 | 0.85153923 |
| Glycine max | 0.916825 | 0.91621152 |
| Gossypium raimondii | 0.931440 | 0.92241807 |
| Hordeum vulgare | 0.891160 | 0.88666832 |
| Leersia perrieri | 0.937381 | 0.9189424 |
| Manihot esculenta | 0.880466 | 0.87884806 |
| Medicago truncatula | 0.903042 | 0.85253227 |
| Oryza barthii | 0.895556 | 0.87524826 |
| Oryza brachyantha | 0.918132 | 0.86357994 |
| Oryza nivara | 0.907913 | 0.88505462 |
| Oryza rufipogon | 0.899358 | 0.88269613 |
| Oryza sativa Japonica Group | 0.921459 | 0.80250745 |
| Physcomitrella patens | 0.892942 | 0.88331678 |
| Populus trichocarpa | 0.926331 | 0.91074975 |
| Setaria italica | 0.846958 | 0.84793942 |
| Solanum lycopersicum | 0.902091 | 0.87847567 |
| Solanum tuberosum | 0.839235 | 0.85613208 |
| Sorghum bicolor | 0.890090 | 0.89411619 |
| Theobroma cacao | 0.952947 | 0.92887289 |
| Trifolium pratense | 0.886288 | 0.87909633 |
| Triticum aestivum | 0.893774 | 0.86295929 |
| Vitis vinífera | 0.893893 | 0.88741311 |

En todos los casos, excepto en 4, las medidas referenciales son mejores a las del modelo final (dentro de la validación cruzada al usar la especie como conjunto de validación). Las especies en este grupo son Solanum tuberosum (papa o patata, con diferencia de 1.69% de diferencia), Sorghum bicolor (sorgo o zahína, con 0.4% de diferencia), Brachypodium distachyon (especie de hierba, con 0.4% de diferencia) y Setaria italica (mijo o la moha, especie de cereal, con 0.1% de diferencia).

De los cuatro casos anteriores, dos especies (Solanum tuberosum y Setaria italica) obtuvieron las métricas más bajas de los modelos referenciales, lo que puede significar que las muestras poseían casos muy heterogéneos, y al momento de clasificarlos con el modelo final entrenado en otras especies, se capturaron más propiedades. Por otro lado, las otras dos (Sorghum bicolor y Brachypodium distachyon), estuvieron dentro de las mejores métricas obtenidas durante la validación cruzada del modelo final, además de que la diferencia registrada en ambos casos era insignificante (menor al 1%).

En promedio, el modelo final mostró medidas de exactitud 2.11% por debajo de la medida de exactitud lograda en el modelo referencial.

Las cuatro mayores diferencias en las que los modelos referenciales mostraron mejor desempeño que el modelo final, se encontraron en cuatro especies: Oryza sativa Japonica Group (arroz, con 11.9% de diferencia), Amborella trichopoda (amboreláceas, con 8.57% de diferencia), Oryza brachyantha (especie de pasto relacionado al arroz, con 5.46% de diferencia) y Medicago truncatula (leguminosa, con 5.05% de diferencia).

Estas diferencias pueden darse por una serie de factores. En principio, las precisiones de los modelos referenciales de estas cuatro especies se encuentran ligeramente por encima del promedio de la precisión media para todos los modelos referenciales. Por otro lado, las precisiones de la validación cruzada para el modelo final correspondiente a estas cuatro especies se encuentran dentro del tercio inferior, inclusive dos de ellos son las dos menores precisiones alcanzadas (Oryza sativa Japonica Group y Amborella trichopoda).

Añadido a lo anterior, al observar el árbol filogenético disponible en la web de Ensembl (<http://plants.ensembl.org/prot_tree_stats.html>), la mayoría de especies poseen varias especies cercanas al subir seis niveles en el árbol, pero en el caso de Medicago truncatula solo se encontraron seis especies cercanas, para Oryza sativa Japonica Group se encontraron 3 especies cercanas, y para Amborella trichopoda y Oryza brachyantha ninguna.

## Resumen

Se entrenaron modelos referenciales (máquinas de soporte de vectores), entrenados bajo una sola especie, con las mismas características definidas para el modelo final, alcanzando en promedio una exactitud de 89.9%.

Para la generación del modelo final se utilizaron redes neuronales profundas, con un mínimo de dos capas (el número de capas adicionales a la primera se trabajó como un hiperparámetro), realizando la búsqueda de los mejores hiperparámetros para el modelo a través del muestreo de grilla, llegando a probar 17 combinaciones de hiperparámetros. La precisión lograda para la mejor combinación de hiperparámetros fue de 87.79%.

Al comparar el modelo final contra los modelos referenciales, en casi todos los casos lo modelos referenciales mostraron mejores tasas de predicción, como era de esperarse, al estar entrenados y validados bajo una sola especie. Sin embargo, los resultados del modelo final en promedio mostraron solamente una exactitud menor en un 2.11%.

# Comparación contra otros modelos

## Modelo final con máquina de soporte de vectores

En base a las mismas características definidas para nuestro modelo final, y la misma arquitectura, se construyó un modelo adicional con máquina de soporte de vectores, con kernel de función de base radial (RBF), valor gamma de 1e-3, y selección del mejor valor C a través de muestreo de cuadrícula con opciones 0.1, 0.5, 0.9 y 2 (similar al mismo espacio buscado para los modelos referenciales.

Los resultados logrados fueron similares al modelo final con redes neuronales profundas, lo cual se puede explicar por la baja dimensionalidad de los datos. Se muestra el detalle de los resultados para cada una de las validaciones cruzadas finales en la Tabla 16.

Tabla - Resultado del modelo final con redes neuronales contra el modelo final con máquina de soporte de vectores.

|  |  |  |
| --- | --- | --- |
| Especie | Modelo final  (redes neuronales) | Modelo final (SVM) |
| Amborella trichopoda | 0.8159136 | 0.79039924 |
| Arabidopsis lyrata | 0.87152433 | 0.85194867 |
| Arabidopsis thaliana | 0.91857001 | 0.90922053 |
| Brachypodium distachyon | 0.90355015 | 0.89436787 |
| Brassica napus | 0.86953823 | 0.83721483 |
| Brassica oleracea | 0.85985601 | 0.83032319 |
| Brassica rapa | 0.87226912 | 0.86264259 |
| Corchorus capsularis | 0.86283515 | 0.7940827 |
| Cucumis sativus | 0.85153923 | 0.83875951 |
| Glycine max | 0.91621152 | 0.902923 |
| Gossypium raimondii | 0.92241807 | 0.89995247 |
| Hordeum vulgare | 0.88666832 | 0.90042776 |
| Leersia perrieri | 0.9189424 | 0.91076521 |
| Manihot esculenta | 0.87884806 | 0.86573194 |
| Medicago truncatula | 0.85253227 | 0.8398289 |
| Oryza barthii | 0.87524826 | 0.86204848 |
| Oryza brachyantha | 0.86357994 | 0.84660171 |
| Oryza nivara | 0.88505462 | 0.88177281 |
| Oryza rufipogon | 0.88269613 | 0.871673 |
| Oryza sativa Japonica Group | 0.80250745 | 0.88545627 |
| Physcomitrella patens | 0.88331678 | 0.87939639 |
| Populus trichocarpa | 0.91074975 | 0.9045865 |
| Setaria italica | 0.84793942 | 0.84173004 |
| Solanum lycopersicum | 0.87847567 | 0.86169202 |
| Solanum tuberosum | 0.85613208 | 0.83662072 |
| Sorghum bicolor | 0.89411619 | 0.89020913 |
| Theobroma cacao | 0.92887289 | 0.9319154 |
| Trifolium pratense | 0.87909633 | 0.86965304 |
| Triticum aestivum | 0.86295929 | 0.8599097 |
| Vitis vinífera | 0.88741311 | 0.87595057 |

La exactitud final lograda fue de 86.75% (en comparación a 87.79% para la el modelo final con red neuronal). En casi todas las especies, la exactitud fue ligeramente menor para el modelo con máquina de soporte de vectores. Los resultados para la red neuronal fueron ligeramente más estables, con una desviación estándar de 0.029376367, contra el 0.033320544 de la máquina de soporte de vectores.

Se realizó una prueba adicional para ambos modelos con información de maíz (zea mays). Para ello utilizó una muestra de 19,796 ARNs codificantes y 2,532 ARNs no codificantes.

Las métricas logradas con redes neuronales son ligeramente mejores, teniendo una exactitud 0.86% adicional al modelo de máquina de soporte de vectores. Se presenta en la Figura 15 la matriz de confusión para ambos modelos.

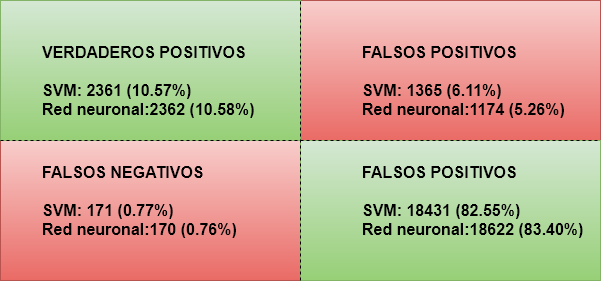


Figura - Matriz de confusión para maíz, para el modelo final con máquina de vectores de soporte y redes neuronales. Elaboración propia.

## Predictor de lncRNA para plantas

Este predictor es un ensamble de bosques aleatorios, entrenado bajo datos empíricamente validados de especies de plantas, animales y virales (Simopoulos et al., 2018). También busca resolver la problemática de construir un modelo para plantas haciendo uso de información de otras especies.

Se realizaron predicciones con este modelo según las indicaciones en <https://github.com/gbgolding/crema>. El código fuente utilizado para realizar estas predicciones se puede encontrar en el Anexo V. En los resultados mostrados en la Tabla 17, se puede observar que a exactitud lograda en el modelo final es mayor en todos los casos.

Tabla - Comparación del modelo final y el predictor para plantas.

|  |  |  |
| --- | --- | --- |
| Especie | Modelo final | Predictor para plantas |
| Amborella trichopoda | 0.8159136 |  |
| Arabidopsis lyrata | 0.87152433 |  |
| Arabidopsis thaliana | 0.91857001 |  |
| Brachypodium distachyon | 0.90355015 |  |
| Brassica napus | 0.86953823 |  |
| Brassica oleracea | 0.85985601 |  |
| Brassica rapa | 0.87226912 |  |
| Corchorus capsularis | 0.86283515 |  |
| Cucumis sativus | 0.85153923 |  |
| Glycine max | 0.91621152 |  |
| Gossypium raimondii | 0.92241807 |  |
| Hordeum vulgare | 0.88666832 |  |
| Leersia perrieri | 0.9189424 |  |
| Manihot esculenta | 0.87884806 |  |
| Medicago truncatula | 0.85253227 |  |
| Oryza barthii | 0.87524826 |  |
| Oryza brachyantha | 0.86357994 |  |
| Oryza nivara | 0.88505462 |  |
| Oryza rufipogon | 0.88269613 |  |
| Oryza sativa Japonica Group | 0.80250745 |  |
| Physcomitrella patens | 0.88331678 |  |
| Populus trichocarpa | 0.91074975 |  |
| Setaria italica | 0.84793942 |  |
| Solanum lycopersicum | 0.87847567 |  |
| Solanum tuberosum | 0.85613208 |  |
| Sorghum bicolor | 0.89411619 |  |
| Theobroma cacao | 0.92887289 |  |
| Trifolium pratense | 0.87909633 |  |
| Triticum aestivum | 0.86295929 |  |
| Vitis vinífera | 0.88741311 |  |

Referencias

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … Zheng, X. (2016). *TensorFlow: A system for large-scale machine learning*. 21.

Achawanantakun, R., Chen, J., Sun, Y., & Zhang, Y. (2015). LncRNA-ID: Long non-coding RNA IDentification using balanced random forests. *Bioinformatics*, *31*(24), 3897–3905. https://doi.org/10.1093/bioinformatics/btv480

Brownlee, J. (2018, mayo 22). A Gentle Introduction to k-fold Cross-Validation. Recuperado el 4 de noviembre de 2018, de Machine Learning Mastery website: https://machinelearningmastery.com/k-fold-cross-validation/

Buchfink, B., Xie, C., & Huson, D. H. (2015). Fast and sensitive protein alignment using DIAMOND. *Nature Methods*, *12*(1), 59–60. https://doi.org/10.1038/nmeth.3176

Chollet, F. (2017). *Deep Learning with Python* (1st ed.). Greenwich, CT, USA: Manning Publications Co.

Durinck, S., Moreau, Y., Kasprzyk, A., Davis, S., De Moor, B., Brazma, A., & Huber, W. (2005). BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, *21*(16), 3439–3440. https://doi.org/10.1093/bioinformatics/bti525

Durinck, S., Spellman, P. T., Birney, E., & Huber, W. (2009). Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt. *Nature Protocols*, *4*(8), 1184–1191. https://doi.org/10.1038/nprot.2009.97

Fan, X.-N., & Zhang, S.-W. (2015). lncRNA-MFDL: identification of human long non-coding RNAs by fusing multiple features and using deep learning. *Molecular BioSystems*, *11*(3), 892–897. https://doi.org/10.1039/C4MB00650J

Fickett, J. W. (1982). Recognition of protein coding regions in DNA sequences. *Nucleic Acids Research*, *10*(17), 5303–5318.

Genomics. (2015, septiembre 1). RNA Class:The Classification of RNA | CD Genomics Blog. Recuperado el 30 de septiembre de 2018, de https://www.cd-genomics.com/blog/index.php/rna-classthe-classification-of-rna/

Han, S., Liang, Y., Li, Y., & Du, W. (2016). Long Noncoding RNA Identification: Comparing Machine Learning Based Tools for Long Noncoding Transcripts Discrimination. *Biomed Research International*, 8496165. https://doi.org/10.1155/2016/8496165

Jabeen, A., Ahmad, N., & Raza, K. (2018). Machine Learning-Based State-of-the-Art Methods for the Classification of RNA-Seq Data. En *Lecture Notes in Computational Vision and Biomechanics*. *Classification in BioApps: Automation of Decision Making* (pp. 133–172). https://doi.org/10.1007/978-3-319-65981-7\_6

Jia-Pei, Y., Hao-Wen, Z., & Zhi, L. (2013). Progress on Bioinformatic Research of lncRNA. *Progress in Biochemistry and Biophysics*, *40*(7), 634–640. https://doi.org/10.3724/SP.J.1206.2013.00266

Kitchenham, B., & Charters, S. (2007). *Guidelines for performing Systematic Literature Reviews in Software Engineering*.

Kong, L., Zhang, Y., Ye, Z.-Q., Liu, X.-Q., Zhao, S.-Q., Wei, L., & Gao, G. (2007). CPC: assess the protein-coding potential of transcripts using sequence features and support vector machine. *Nucleic Acids Research*, *35*(Web Server issue), W345–W349. https://doi.org/10.1093/nar/gkm391

Kung, J. T. Y., Colognori, D., & Lee, J. T. (2013). Long Noncoding RNAs: Past, Present, and Future. *Genetics*, *193*(3), 651–669. https://doi.org/10.1534/genetics.112.146704

Min, S., Lee, B., & Yoon, S. (2017). Deep learning in bioinformatics. *Briefings in Bioinformatics*, *18*(5), 851–869. https://doi.org/10.1093/bib/bbw068

NHGRI. (2015, octubre 21). Ácido desoxirribonucleico (ADN). Recuperado el 20 de octubre de 2018, de National Human Genome Research Institute (NHGRI) website: https://www.genome.gov/27562614/cido-desoxirribonucleico-adn/

Nielsen, M. A. (2018). *Neural Networks and Deep Learning*. Recuperado de http://neuralnetworksanddeeplearning.com

Paytuví Gallart, A., Hermoso Pulido, A., Anzar Martínez de Lagrán, I., Sanseverino, W., & Aiese Cigliano, R. (2016). GREENC: a Wiki-based database of plant lncRNAs. *Nucleic Acids Research*, *44*(D1), D1161–D1166. https://doi.org/10.1093/nar/gkv1215

Ponjavic, J., Ponting, C. P., & Lunter, G. (2007). Functionality or transcriptional noise? Evidence for selection within long noncoding RNAs. *Genome Research*, *17*(5), 556–565. https://doi.org/10.1101/gr.6036807

Quinn, J. J., & Chang, H. Y. (2016). Unique features of long non-coding RNA biogenesis and function. *Nature Reviews Genetics*, *17*(1), 47–62. https://doi.org/10.1038/nrg.2015.10

Risueño Pérez, A. (2012). *Bioinformática aplicada a estudios del transcriptoma humano: análisis de expresión de genes, isoformas génicas y ncRNAs en muestras sanas y en cáncer*. Recuperado de https://gredos.usal.es/jspui/handle/10366/121408

Rouse, M. (2017, abril). ¿Qué es Aprendizaje profundo (deep learning)? - Definición en WhatIs.com. Recuperado el 30 de septiembre de 2018, de SearchDataCenter&nbsp;en&nbsp;Español website: https://searchdatacenter.techtarget.com/es/definicion/Aprendizaje-profundo-deep-learning

Schneider, H. W., Raiol, T., Brigido, M. M., Walter, M. E. M. T., & Stadler, P. F. (2017). A Support Vector Machine based method to distinguish long non-coding RNAs from protein coding transcripts. *Bmc Genomics*, *18*, 804. https://doi.org/10.1186/s12864-017-4178-4

Sebastián, I. (2017, diciembre 15). ADN codificante Archives. Recuperado el 5 de octubre de 2018, de Genética Médica Blog website: https://revistageneticamedica.com/blog/tag/adn-codificante/

Setubal, J. C., & Meidanis, J. (1997). *Introduction to computational molecular biology*. Boston: PWS Pub.

Shen, H. (2014). Interactive notebooks: Sharing the code. *Nature News*, *515*(7525), 151. https://doi.org/10.1038/515151a

Simopoulos, C. M. A., Weretilnyk, E. A., & Golding, G. B. (2018). Prediction of plant lncRNA by ensemble machine learning classifiers. *Bmc Genomics*, *19*, 316. https://doi.org/10.1186/s12864-018-4665-2

Singh, U., Khemka, N., Rajkumar, M. S., Garg, R., & Jain, M. (2017). PLncPRO for prediction of long non-coding RNAs (lncRNAs) in plants and its application for discovery of abiotic stress-responsive lncRNAs in rice and chickpea. *Nucleic Acids Research*, *45*(22), e183–e183. https://doi.org/10.1093/nar/gkx866

Szcześniak, M. W., Rosikiewicz, W., & Makałowska, I. (2016). CANTATAdb: A Collection of Plant Long Non-Coding RNAs. *Plant and Cell Physiology*, *57*(1), e8–e8. https://doi.org/10.1093/pcp/pcv201

Tolosa, A. (2017, mayo 19). Utilidad diagnóstica de la secuenciación del transcriptoma. Recuperado el 30 de septiembre de 2018, de Genética Médica website: https://revistageneticamedica.com/2017/05/19/secuenciacion-transcriptoma/

Tripathi, R., Patel, S., Kumari, V., Chakraborty, P., & Varadwaj, P. K. (2016). DeepLNC, a long non-coding RNA prediction tool using deep neural network. *Network Modeling Analysis in Health Informatics and Bioinformatics*, *5*(1), 21. https://doi.org/10.1007/s13721-016-0129-2

Uszczynska-Ratajczak, B., Lagarde, J., Frankish, A., Guigo, R., & Johnson, R. (2018). Towards a complete map of the human long non-coding RNA transcriptome. *Nature Reviews Genetics*, *19*(9), 535–548. https://doi.org/10.1038/s41576-018-0017-y

VanderPlas, J. (2016). *Python Data Science Handbook*. Recuperado de http://shop.oreilly.com/product/0636920034919.do

Vieira, L. M. (2018). *Métodos baseados em aprendizagem de máquina para distinguir RNAs longos não-codificadores intergênicos de transcritos codificadores de proteínas*. Recuperado de http://repositorio.unb.br/handle/10482/32463

Wang, L., Park, H. J., Dasari, S., Wang, S., Kocher, J.-P., & Li, W. (2013). CPAT: Coding-Potential Assessment Tool using an alignment-free logistic regression model. *Nucleic Acids Research*, *41*(6), e74. https://doi.org/10.1093/nar/gkt006

Willems, K. (2015, mayo 12). Choosing R or Python for data analysis? An infographic. Recuperado el 3 de noviembre de 2018, de DataCamp Community website: https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis

Wucher, V., Legeai, F., Hédan, B., Rizk, G., Lagoutte, L., Leeb, T., … Derrien, T. (2017). FEELnc: a tool for long non-coding RNA annotation and its application to the dog transcriptome. *Nucleic Acids Research*, *45*(8), e57. https://doi.org/10.1093/nar/gkw1306

Yao, Y., Li, X., Geng, L., Nan, X., Qi, Z., & Liao, B. (2018). Recent Progress in Long Noncoding RNAs Prediction. *Current Bioinformatics*, *13*(4), 344–351. https://doi.org/10.2174/1574893612666170905153933

Yip, K. Y., Cheng, C., & Gerstein, M. (2013). Machine learning and genome annotation: a match meant to be? *Genome Biology*, *14*(5), 205. https://doi.org/10.1186/gb-2013-14-5-205

Zerbino, D. R., Achuthan, P., Akanni, W., Amode, M. R., Barrell, D., Bhai, J., … Flicek, P. (2018). Ensembl 2018. *Nucleic Acids Research*, *46*(D1), D754–D761. https://doi.org/10.1093/nar/gkx1098

# Anexos

## : Plan de Proyecto

* **Justificación**

El presente proyecto proveerá un modelo predictivo para clasificar *lncRNAs* y *PCTs*, diferenciándose de otros clasificadores existentes, en que para realizar predicciones sobre una especie específica puede ser entrenado en base a especies diferentes, y producir resultados aceptables en el proceso.

También, dentro del contexto de los proyectos de secuenciación de transcriptomas, beneficiará a otros proyectos de anotación experimental sobre nuevas especies, que no poseen anotaciones previas. Los proyectos de secuenciación del transcriptoma generan una enorme cantidad de información, y las predicciones iniciales del modelo pueden guiar la validación y anotación experimental.

* **Viabilidad**

El proyecto es viable desde el aspecto técnico, ya que en el campo de la bioinformática el aprendizaje profundo está siendo empleado con éxito. Adicionalmente, el aprendizaje profundo es ideal para problemas en los cuales se tiene gran cantidad de información (bases de datos de transcriptomas), y para el análisis de información secuencial (secuencias de nucleótidos); ambos factores están presentes en este proyecto.

Otro aspecto de viabilidad técnica, es el auge del aprendizaje de máquina en la actualidad. Se tienen herramientas, como la suite Anaconda, que centralizan la instalación, configuración, control de dependencias, edición de código y generación de gráficos y anotaciones para proyectos de investigación científica.

Adicionalmente, se tiene el apoyo del asesor de proyecto, especialista en bioinformática, para absolver posibles dudas que puedan surgir durante la elaboración del proyecto de fin de carrera.

Con respecto a la viabilidad temporal, se posee de medio ciclo para todas las labores de investigación previas a la presentación de resultados (planteamiento del problema y objetivos, marco teórico, estado del arte y planificación de actividades para llegar a los resultados esperados). También durante medio semestre universitario, que se traduce a dos meses y medios, se realizará en base al trabajo previo la implementación y análisis de los resultados. Se estima que estos tiempos serán suficientes para la realización de este proyecto de fin de carrera.

Con respecto a la viabilidad económica, la PUCP ofrece acceso gratuito a bases de datos de investigación. Adicionalmente, el acceso a las bases de datos transcriptómicas es gratuito, y se trabajará con software libre para la elaboración de la solución de aprendizaje de máquina. Finalmente, se requerirá equipo especializado para el entrenamiento y validación del modelo predictivo, el cuál será brindado por el grupo de inteligencia artificial de la PUCP (IA-PUCP), a través de servidores con GPUs especialmente orientados a estas labores.

* **Alcance**

El presente proyecto de investigación se encuentra enmarcado dentro de las Ciencias de la Computación, y es de carácter exploratorio, buscando una alternativa nueva para la predicción de moléculas ARN no codificantes en especies bajo las cuales el modelo no ha sido entrenado. Para ello se planteará una arquitectura para el proceso de aprendizaje de máquina, mediante el cual se defina el entrenamiento y validación del modelo y se generen predictores. Se probará el desempeño del modelo y sus predictores en tres instancias:

* La primera será la validación base del modelo predictivo bajo la técnica del aprendizaje profundo.
* La segunda será la generación de modelos entrenados y validados en una misma especie. Se espera que la precisión de esta especie sea mejor que la del modelo propuesto, y que sirva de referencia.
* Finalmente, se realizará una comparación de la precisión de los predictores generados y el modelo del estado del arte para clasificar especies desconocidas al modelo, al ser entrenado en otras.
* **Limitaciones**

Las limitaciones identificadas durante la revisión del estado del arte y la elaboración del proyecto fueron las siguientes:

* Debido a límites de tiempo la validación contra otros modelos no será exhaustiva. Es decir, a pesar de existir en la literatura una gran cantidad de modelos ya elaborados, se ha seleccionado un modelo específico para la comparación.
* Debido al punto anterior, para la evaluación del modelo, se utilizará como datos de entrenamiento transcritos (ARN no codificante, *lncRNA*) de plantas, y en específico, se simulará la predicción en maíz, como la especie desconocida al modelo.
* **Identificación de los riesgos del proyecto**

Los riesgos identificados en el presente proyecto y sus respectivas contingencias se presentan en la Tabla 16.

Tabla 18 - Riesgos del proyecto de investigación.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Descripción | Probabilidad | Impacto | Severidad | Síntomas, Mitigación y Contingencia |
| Cantidad insuficiente de transcritos para el entrenamiento y validación de los predictores. | Baja  10% | Alto  90% | Baja  9% | Síntomas: Bajo desempeño del modelo predictivo.  Mitigación: Evaluación del estado del arte, analizando si se dio el problema antes.  Contingencia: Uso de otras bases de datos de secuenciación de transcriptomas. |
| Pérdida de información del proyecto por malfunciona-miento o pérdida del equipo donde se trabaja el proyecto. | Media  50% | Alto  90% | Media  45% | Síntomas: Malfuncionamiento de Laptop de trabajo. Hurto.  Mitigación: Uso de sistemas de control de versiones para el almacenamiento en la nube de la información.  Contingencia: Se tiene una PC adicional de trabajo. Recuperación de fuentes de repositorios en la nube. |

* **Estructura de descomposición del trabajo (EDT)**

Se presenta en la Figura 15 la estructura de descomposición del trabajo para el proyecto de fin de carrera.

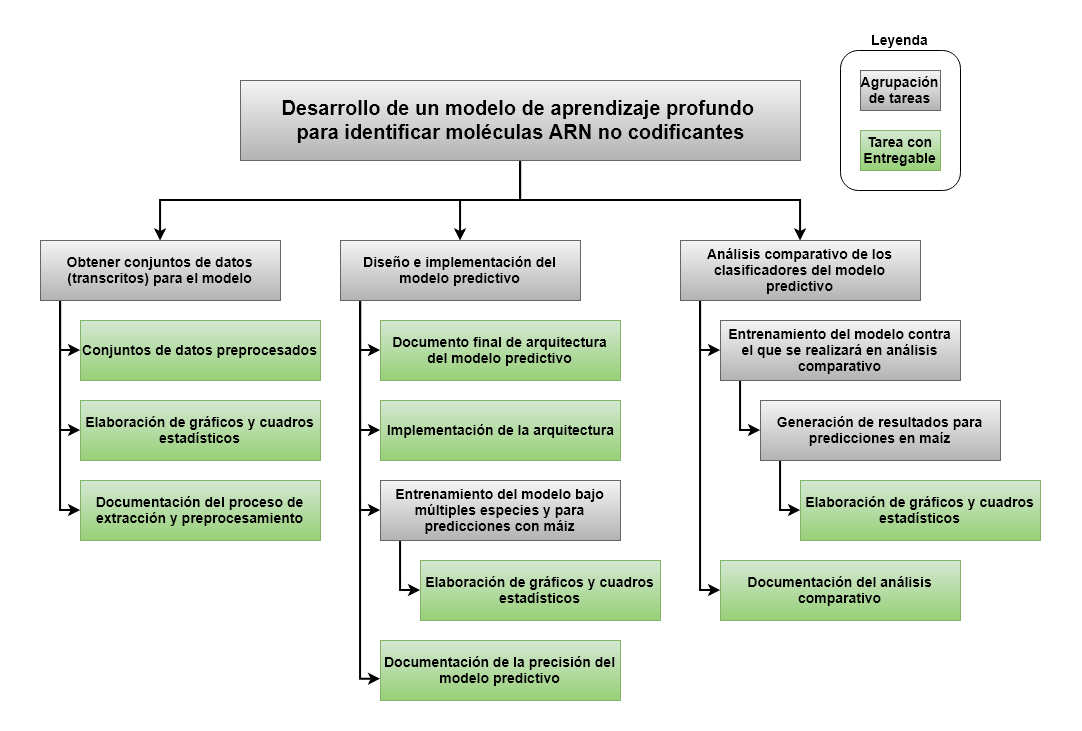


Figura 16 - Estructura de descomposición del trabajo. Elaboración propia.

* **Lista de tareas**

En la Tabla 17 se presenta la lista de tareas.

Se tomó en cuenta un solo recurso, con un costo mensual de mil soles, que se traslada en un costo por hora de 5.68 soles.

Tabla 19 - Lista de tareas del proyecto.



* **Cronograma del proyecto**

En la Figura 16 se presenta el cronograma del proyecto.

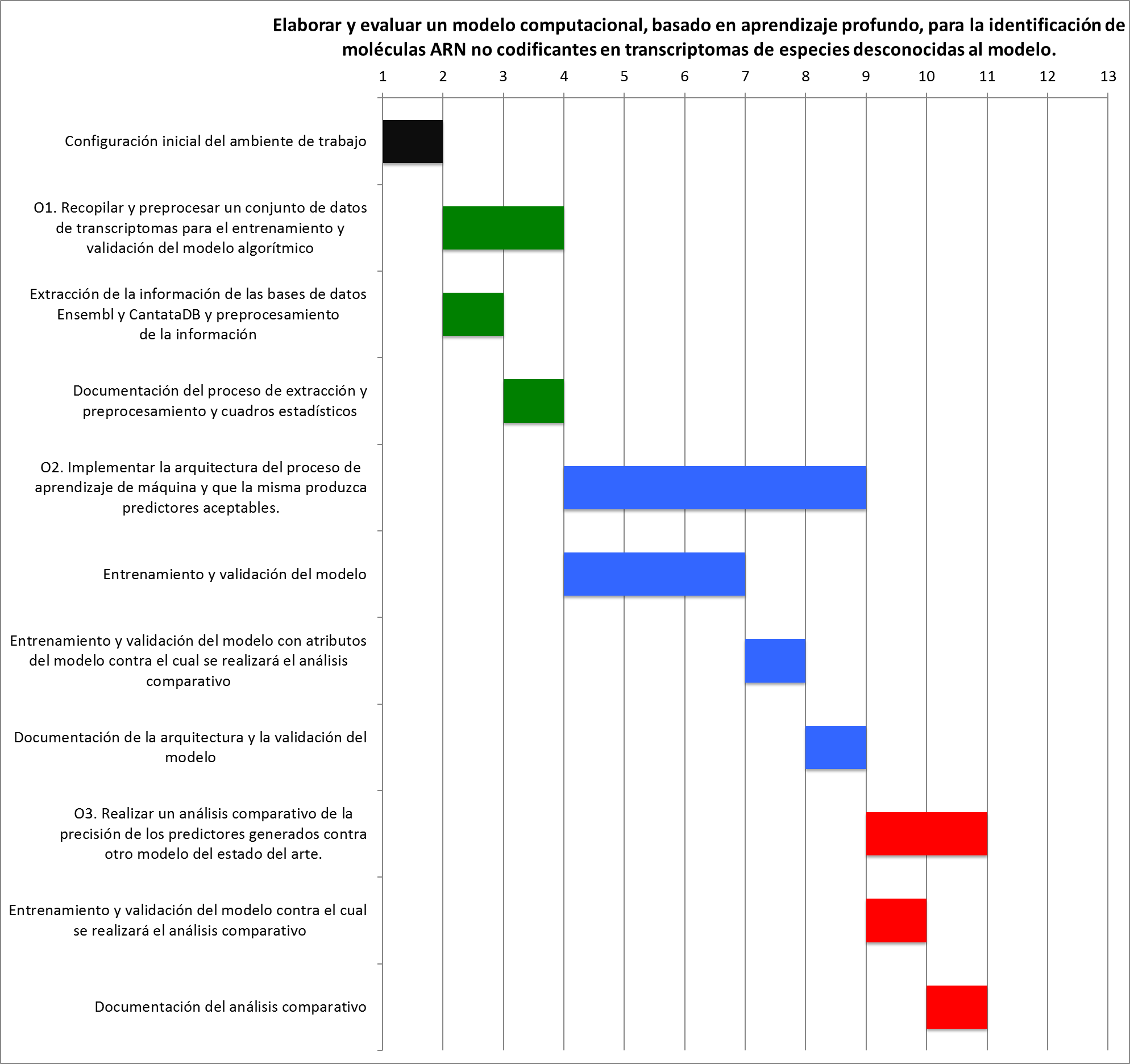


Figura 17 - Cronograma del proyecto por semanas. Elaboración propia.

* **Lista de recursos**
  + **Personas involucradas y necesidades de capacitación**
* Estudiante José Luis Santillán Escudero, quien elabora el proyecto de tesis. Debido a que el proyecto cubre el campo de la bioinformática, se requiere capacitación en temas relacionados a la biología, que será brindada por el asesor del presente proyecto.
* Dr. Edwin Rafael Villanueva Talavera, asesor del presente proyecto, brindará guía con respecto a temas de biología y el uso de aprendizaje de máquina para modelos predictivos.
  + **Materiales requeridos para el proyecto**
* Acceso a bases de datos de investigación, ofrecidos gratuitamente por la PUCP.
* Acceso a bases de datos de transcriptomas (Ensembl y CantataDB).
  + **Estándares utilizados en el proyecto**

No aplica.

* + **Equipamiento requerido**
* Equipo capaz de procesar gran cantidad de información para el entrenamiento y validación de algoritmos de aprendizaje de máquina. Se tendrá acceso al mismo a través del grupo IA-PUCP, que brindará servidores con GPUs especialmente diseñados para la labor.
* Laptop o PC para la elaboración del proyecto de tesis, y la interacción con el servidor durante la ejecución del proyecto de tesis. Se poseen ambos.
  + **Herramientas requeridas**
* Las herramientas requeridas, será un ambiente de trabajo especialmente diseñado para la investigación científica, en específico, para el trabajo con algoritmos de aprendizaje de máquina. Anaconda ofrece gratuitamente una suite, a través de la cual se pueden instalar los lenguajes de programación, librerías, dependencias, herramientas de análisis y visualización de datos y soporte a través de una comunidad activa.
* **Costeo del Proyecto**

En la Tabla 18 se presentan los costos que se identificaron para el presente proyecto.

Tabla - Costos del proyecto.

| **Ítem** | **Descripción** | | | **Unidad** | **Cantidad** | **Valor Unidad (S/.)** | **Monto Parcial (S/.)** | **Monto**  **Total (S/.)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **Costo total del proyecto** | | | **---** | **---** | **---** | **---** | **5,209** |
| **1.** | **Estudiantes o tesistas** | | | **---** | **---** | **---** | **---** | **1,909** |
| 1.1 | José Luis Santillán Escudero | | | Horas | 336 | 5.68 | 1,909 |  |
| **2.** | **Otros participantes** | | | **---** | **---** | **---** | **---** | **800** |
| 2.1 | Asesor: Dr. Edwin Rafael Villanueva Talavera | | | Horas | 40 | 20 | 800 |  |
| **3.** | **Servicios y consultoría** | | | **---** | **---** | **---** | **---** | **0** |
|  | No aplica. | | |  |  |  |  |  |
| **4.** | **Materiales e insumos** | | | **---** | **---** | **---** | **---** | **0** |
| 4.1 | Acceso a base de datos PUCP | | | Costo desconocido | | | |  |
| 4.2 | Acceso a bases de datos de transcriptomas | | | Costo desconocido | | | |  |
| **5.** | **Bienes y equipos** | **Unid1** | **Cant1-** | **Unid2** | **Cant2-** | **-** | **-** | **2,500** |
| 5.1 | Computadora | Equipo | 1 | Horas | 100 | 5 | 500 |  |
| 5.2 | Servidor  GPU grupo IA-PUCP | Equipo | 1 | Horas | 200 | 10 | 2,000 |  |
| **6.** | **Pasajes y viáticos** | **Unid1** | **Cant1-** | **Unid2** | **Cant2-** | **-** | **-** | **0** |
|  | No aplica |  |  |  |  |  |  |  |

## : Descarga de Ensembl a través de biomaRt

%%time

%%R

library(RMySQL)

library(curl)

library(biomaRt)

library(stringr)

cat("Iniciando proceso...**\n**")

dsn\_database = "tesis2"

dsn\_hostname = "localhost"

dsn\_port = 3306

dsn\_uid = "tesis2"

dsn\_pwd = "tesis2"

conn = dbConnect(MySQL(), user=dsn\_uid, password=dsn\_pwd, dbname=dsn\_database, host=dsn\_hostname)

rs = dbSendQuery(conn, "SELECT id\_especie, especie FROM maestra\_especies m WHERE especie IN (SELECT c1.especie FROM conteo\_lncrna\_especie c1 LEFT JOIN conteo\_lncrna\_especie c2 ON c1.especie = REPLACE(c2.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c2.fuente = 'CantataDB' LEFT JOIN conteo\_lncrna\_especie c3 ON c1.especie = REPLACE(c3.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c3.fuente = 'GreeNC' WHERE c1.fuente = 'Ensembl' AND (c2.fuente IS NOT NULL OR c3.fuente IS NOT NULL)) AND NOT EXISTS (SELECT 1 FROM secuencias s WHERE s.id\_especie = m.id\_especie AND s.flg\_pct = 1)")

especies = dbFetch(rs)

dbClearResult(rs)

ensembl\_mart=useMart("plants\_mart", host="plants.ensembl.org")

lista\_datasets = listDatasets(ensembl\_mart)

**if** (nrow(especies) > 0) {

**for**(i **in** 1:nrow(especies)) {

tryCatch(

expr = {

cat("**\r**Procesando especie #", i, " ", especies[i,2], " ", sep = "")

la\_especie = especies[i,2]

especie = paste("^", la\_especie, sep="")

dataset = lista\_datasets[which(str\_detect(lista\_datasets$description, especie)), 1]

ensembl\_ds = useDataset(dataset,mart=ensembl\_mart)

genes = getBM(attributes=c('ensembl\_transcript\_id', 'transcript\_exon\_intron'), filters = 'transcript\_biotype', values = 'protein\_coding', mart = ensembl\_ds)

**for**(j **in** 1:nrow(genes)) {

**if** (j %% 1000 == 0) {

cat("**\r**Procesando especie #", i, " ", especies[i,2], " - procesadas ", j, " filas ", sep = "")

}

**if** (!(genes[j,2] == "Sequence unavailable" | str\_length(genes[j,2]) > 65000)) {

query = sprintf("INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%d**, '**%s**', '**%s**', 1, NULL)", especies[i,1], genes[j,1], genes[j,2])

dbSendQuery(conn, query)

}

}

cat("**\r**Procesando especie #", i, " ", especies[i,2], " - procesadas ", nrow(genes), " filas ", sep = "")

},

error = function(e) {

cat("**\r**Error al procesar la especie ", especies[i,2], " **\n**", sep = "")

}

)

}

}

dbDisconnect(conn)

cat("**\r**Proceso finalizado... ")

## : Procesamiento de descargas manuales de Ensembl

%%time

**import** **mysql.connector**

**import** **os**

*#método para leer los transcriptomas de un archivo fasta de ensembl*

**def** leer\_transcriptomas\_ensembl(archivo, cur, id\_especie):

cabecera = ""

transcriptoma = ""

f = open(archivo, "r")

**for** linea **in** f:

**if** linea.startswith(">"):

**if** transcriptoma != "":

**if** transcriptoma != "Sequence unavailable" **and** len(transcriptoma) <= 65000:

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 1, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

cabecera = linea.rstrip("**\n**").lstrip(">")

**else**:

transcriptoma += linea.rstrip("**\n**")

**if** transcriptoma != "" **and** len(transcriptoma) <= 65000:

**if** transcriptoma != "Sequence unavailable":

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 1, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cur = conn.cursor()

directory = '../Semana 05/descarga\_manual\_ensembl'

**for** filename **in** os.listdir(directory):

**if** filename.endswith(".fasta"):

especie = os.path.splitext(filename)[0]

query = "SELECT COUNT(\*) FROM maestra\_especies m WHERE especie IN (SELECT c1.especie FROM conteo\_lncrna\_especie c1 LEFT JOIN conteo\_lncrna\_especie c2 ON c1.especie = REPLACE(c2.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c2.fuente = 'CantataDB' LEFT JOIN conteo\_lncrna\_especie c3 ON c1.especie = REPLACE(c3.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c3.fuente = 'GreeNC' WHERE c1.fuente = 'Ensembl' AND (c2.fuente IS NOT NULL OR c3.fuente IS NOT NULL)) AND m.especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

result=cur.fetchone()

**if** result[0] > 0:

print("**\r**Procesando especie ", especie, " ", end=" ")

query = "SELECT e.id\_especie FROM maestra\_especies e WHERE e.especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

id\_especie=cur.fetchone()

query = "delete from secuencias where id\_especie = **%s**"

cur.execute(query, [id\_especie[0]])

leer\_transcriptomas\_ensembl(directory + "/" + filename, cur, id\_especie[0])

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Procesamiento de archivos en formato fasta de CantataDB

%%time

**import** **mysql.connector**

**import** **os**

*#método para leer los transcriptomas de un archivo fasta en cantatadb*

**def** leer\_transcriptomas\_cantatadb(archivo, cur, id\_especie):

cabecera = ""

transcriptoma = ""

f = open(archivo, "r")

**for** linea **in** f:

**if** linea.startswith(">"):

**if** transcriptoma != "":

**if** len(transcriptoma) <= 65000:

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 0, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

cabecera = linea.rstrip("**\n**").lstrip(">")

**else**:

transcriptoma += linea.rstrip("**\n**")

**if** transcriptoma != "":

**if** len(transcriptoma) <= 65000:

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 0, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cur = conn.cursor()

directory = '../Semana 02/httpcantata.amu.edu.pldownload.php/Otros'

**for** filename **in** os.listdir(directory):

**if** filename.endswith(".fasta"):

especie = os.path.splitext(filename)[0].split("\_lncrnas")[:-1][0]

query = "SELECT COUNT(\*) FROM maestra\_especies m WHERE especie IN (SELECT c1.especie FROM conteo\_lncrna\_especie c1 LEFT JOIN conteo\_lncrna\_especie c2 ON c1.especie = REPLACE(c2.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c2.fuente = 'CantataDB' LEFT JOIN conteo\_lncrna\_especie c3 ON c1.especie = REPLACE(c3.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c3.fuente = 'GreeNC' WHERE c1.fuente = 'Ensembl' AND (c2.fuente IS NOT NULL OR c3.fuente IS NOT NULL)) AND NOT EXISTS (SELECT 1 FROM secuencias s WHERE s.id\_especie = m.id\_especie AND s.flg\_pct = 0) AND m.especie = REPLACE(**%s**, '\_', ' ') AND EXISTS (SELECT 1 FROM especies\_seleccionadas es WHERE es.especie = m.especie AND fuente = **%s**)"

cur.execute(query, (especie, "CantataDB"))

result=cur.fetchone()

**if** result[0] > 0:

print("**\r**Procesando especie ", especie, " ", end=" ")

query = "SELECT e.id\_especie FROM maestra\_especies e WHERE e.especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

id\_especie=cur.fetchone()

leer\_transcriptomas\_cantatadb(directory + "/" + filename, cur, id\_especie[0])

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Procesamiento de archivos en formato fasta de GreeNC

%%time

**import** **mysql.connector**

**import** **os**

*#método para leer los transcriptomas de un archivo fasta en cantatadb*

**def** leer\_transcriptomas\_cantatadb(archivo, cur, id\_especie):

cabecera = ""

transcriptoma = ""

f = open(archivo, "r")

**for** linea **in** f:

**if** linea.startswith(">"):

**if** transcriptoma != "":

**if** len(transcriptoma) <= 65000:

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 0, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

cabecera = linea.rstrip("**\n**").lstrip(">")

**else**:

transcriptoma += linea.rstrip("**\n**")

**if** transcriptoma != "":

**if** len(transcriptoma) <= 65000:

query = "INSERT INTO secuencias (id\_especie, cod\_secuencia, secuencia, flg\_pct, flg\_seleccionado) VALUES (**%s**, **%s**, **%s**, 0, NULL)"

cur.execute(query, (id\_especie, cabecera, transcriptoma))

transcriptoma = ""

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cur = conn.cursor()

directory = '../Semana 03/httpgreenc.sciencedesigners.com'

**for** filename **in** os.listdir(directory):

**if** filename.endswith(".fasta"):

especie = os.path.splitext(filename)[0]

query = "SELECT COUNT(\*) FROM maestra\_especies m WHERE especie IN (SELECT c1.especie FROM conteo\_lncrna\_especie c1 LEFT JOIN conteo\_lncrna\_especie c2 ON c1.especie = REPLACE(c2.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c2.fuente = 'CantataDB' LEFT JOIN conteo\_lncrna\_especie c3 ON c1.especie = REPLACE(c3.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c3.fuente = 'GreeNC' WHERE c1.fuente = 'Ensembl' AND (c2.fuente IS NOT NULL OR c3.fuente IS NOT NULL)) AND NOT EXISTS (SELECT 1 FROM secuencias s WHERE s.id\_especie = m.id\_especie AND s.flg\_pct = 0) AND m.especie = REPLACE(**%s**, '\_', ' ') AND EXISTS (SELECT 1 FROM especies\_seleccionadas es WHERE es.especie = m.especie AND fuente = **%s**)"

cur.execute(query, (especie, "GreeNC"))

result=cur.fetchone()

**if** result[0] > 0:

print("**\r**Procesando especie ", especie, " ", end=" ")

query = "SELECT id\_especie FROM maestra\_especies WHERE especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

id\_especie=cur.fetchone()

leer\_transcriptomas\_cantatadb(directory + "/" + filename, cur, id\_especie[0])

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Selección aleatoria de transcritos para el modelo

%%time

**import** **mysql.connector**

**import** **pandas** **as** **pd**

print("Iniciando proceso...")

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

cursor = conn.cursor()

query = "UPDATE secuencias SET flg\_seleccionado = NULL"

cursor.execute(query)

query = "SELECT id\_especie FROM maestra\_especies m JOIN especies\_seleccionadas s ON m.especie = s.especie"

cursor.execute(query)

ids = cursor.fetchall()

**for** id\_especie **in** ids:

print("**\r**PCT for " + str(id\_especie[0]) + " ", end=" ")

query = "UPDATE secuencias SET flg\_seleccionado = 1 WHERE id\_especie = **%s** AND flg\_pct = 1 ORDER BY RAND() LIMIT 4208"

cursor.execute(query, [id\_especie[0]])

print("**\r**lncRNA for " + str(id\_especie[0]) + " ", end=" ")

query = "UPDATE secuencias SET flg\_seleccionado = 1 WHERE id\_especie = **%s** AND flg\_pct = 0 ORDER BY RAND() LIMIT 4208"

cursor.execute(query, [id\_especie[0]])

print("**\r**Proceso terminado... **\n**")

conn.close()

## : Características – Longitud del transcrito

%%time

**import** **mysql.connector**

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

query = "SELECT id\_especie FROM maestra\_especies"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Features para id-especie " + str(especie[0]), " ", end="")

query = "INSERT INTO secuencias\_features (id\_especie, cod\_secuencia, flg\_pct, flg\_seleccionado, longitud) SELECT id\_especie, cod\_secuencia, flg\_pct, flg\_seleccionado, LENGTH(secuencia) FROM secuencias WHERE id\_especie = **%s**"

cursor.execute(query, [especie[0]])

print("**\r**Proceso finalizado. ")

conn.close()

## : Características – Porcentaje de nucleótidos GC

%%time

**import** **mysql.connector**

**import** **os**

**from** **Bio.SeqUtils** **import** GC

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Generando GC content " + str(especie[0]), " ", end="")

query = "SELECT cod\_secuencia, secuencia FROM secuencias WHERE id\_especie = **%s** AND flg\_seleccionado = 1"

cursor.execute(query, [especie[0]])

transcriptomas = cursor.fetchall()

**for** transcriptoma **in** transcriptomas:

gc\_content = GC(transcriptoma[1])

query = "UPDATE secuencias\_features SET gc\_content = **%s** WHERE id\_especie = **%s** and cod\_secuencia = **%s**"

cur.execute(query, (gc\_content, especie[0], transcriptoma[0]))

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Generación de archivos en formato fasta para CDS y lncRNA

%%time

**import** **mysql.connector**

**import** **os**

**def** generar\_fasta(secuencias, especie, tipo, directory):

t\_tamanio = 80

f = open(directory + especie + "." + tipo + ".fasta" ,"w+")

**for** transcriptoma **in** secuencias:

f.write(">**%s\n**" % (transcriptoma[0]))

seq = transcriptoma[1]

t\_partes = [seq[i:i+t\_tamanio] **for** i **in** range(0, len(seq), t\_tamanio)]

**for** t\_parte **in** t\_partes:

f.write("**%s\n**" % (t\_parte))

f.close()

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

directory = '../Semana 05/fasta\_para\_hexamer/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Generando fasta CDS para " + str(especie[0]), " ", end="")

query = "select c.cod\_secuencia, c.coding from secuencias\_CDS c JOIN secuencias\_features f ON c.id\_especie = f.id\_especie AND c.cod\_secuencia = f.cod\_secuencia WHERE c.id\_especie = **%s** AND f.flg\_pct = 1 AND f.flg\_seleccionado = 1"

cursor.execute(query, [especie[0]])

secuencias = cursor.fetchall()

generar\_fasta(secuencias, especie[1], "CDS", directory)

print("**\r**Generando fasta lncRNA para " + str(especie[0]), " ", end="")

query = "SELECT cod\_secuencia, secuencia FROM secuencias WHERE id\_especie = **%s** AND flg\_pct = 0 AND flg\_seleccionado = 1"

cursor.execute(query, [especie[0]])

secuencias = cursor.fetchall()

generar\_fasta(secuencias, especie[1], "lncRNA", directory)

conn.close()

print("**\r**Proceso finalizado... ")

## : CPAT – Generación de tabla de frecuencias de hexámeros

%%time

**import** **mysql.connector**

**import** **os**

**from** **IPython.utils** **import** io

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

input\_directory = '../Semana 05/fasta\_para\_hexamer/'

output\_directory = '../Semana 05/tablas\_hexamer/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Generando tabla hexamer para " + str(especie[0]), " ", end="")

script = "~/anaconda3/bin/make\_hexamer\_tab.py"

fasta\_cds = "'" + input\_directory + especie[1] + ".CDS.fasta" + "'"

lncRNA\_cds = "'" + input\_directory + especie[1] + ".lncRNA.fasta" + "'"

salida = output\_directory + especie[1] + ".tsv"

**with** io.capture\_output() **as** captured:

%run $script -c $fasta\_cds -n $lncRNA\_cds

f = open(salida ,"w+")

f.write(captured.stdout)

f.close()

conn.close()

print("**\r**Proceso finalizado... ")

## : Generación de archivos en formato fasta para ARN codificantes

%%time

**import** **mysql.connector**

**import** **os**

**def** generar\_fasta(secuencias, especie, tipo, directory):

t\_tamanio = 80

f = open(directory + especie + "." + tipo + ".fasta" ,"w+")

**for** transcriptoma **in** secuencias:

f.write(">**%s\n**" % (transcriptoma[0]))

seq = transcriptoma[1]

t\_partes = [seq[i:i+t\_tamanio] **for** i **in** range(0, len(seq), t\_tamanio)]

**for** t\_parte **in** t\_partes:

f.write("**%s\n**" % (t\_parte))

f.close()

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

directory = '../Semana 05/fasta\_pct\_para\_logit/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Generando fasta CDS para " + str(especie[0]), " ", end="")

query = "SELECT cod\_secuencia, secuencia FROM secuencias WHERE id\_especie = **%s** AND flg\_pct = 1 AND flg\_seleccionado = 1"

cursor.execute(query, [especie[0]])

secuencias = cursor.fetchall()

generar\_fasta(secuencias, especie[1], "PCT", directory)

conn.close()

print("**\r**Proceso finalizado... ")

## : CPAT – Generación de regresión logística

%%time

**import** **mysql.connector**

**import** **os**

**from** **IPython.utils** **import** io

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cursor = conn.cursor()

input\_directory = '../Semana 05/tablas\_hexamer/'

input\_directory\_pct = '../Semana 05/fasta\_pct\_para\_logit/'

input\_directory\_lncRNA = '../Semana 05/fasta\_para\_hexamer/'

output\_directory\_logit = '../Semana 05/modelo\_regresion\_logistica/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Generando modelo de regresión para " + str(especie[0]), " ", end="")

script = "~/anaconda3/bin/make\_logitModel.py"

archivo\_hexamer = "'" + input\_directory + especie[1] + ".tsv" + "'"

fasta\_pct = "'" + input\_directory\_pct + especie[1] + ".PCT.fasta" + "'"

lncRNA\_cds = "'" + input\_directory\_lncRNA + especie[1] + ".lncRNA.fasta" + "'"

salida = "'" + output\_directory\_logit + especie[1] + "'"

**with** io.capture\_output() **as** captured:

%run $script -x $archivo\_hexamer -c $fasta\_pct -n $lncRNA\_cds -o $salida

conn.close()

print("**\r**Proceso finalizado... ")

## : CPAT – Ejecución de CPAT y exportación a la base de datos

%%time

**import** **mysql.connector**

**import** **os**

**import** **csv**

**def** procesar\_features\_cpat(archivo, cur, id\_especie):

**with** open(archivo, "r") **as** tabular:

cpat\_reader = csv.reader(tabular, delimiter=("**\t**"))

**for** row **in** cpat\_reader:

cod\_secuencia = row[0]

orf\_length = float(row[2])

fickett = float(row[3])

hexamer = float(row[4])

query = "UPDATE secuencias\_features SET orf\_length = **%s**, orf\_coverage = **%s**/longitud, ficket\_score = **%s**, hexamer\_score = **%s** WHERE id\_especie = **%s** and cod\_secuencia = **%s**"

cur.execute(query, (orf\_length, orf\_length, fickett, hexamer, id\_especie, cod\_secuencia))

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

print("Iniciando proceso...")

cur = conn.cursor()

directory = '../Semana 05/cpat\_features'

**for** filename **in** os.listdir(directory):

**if** filename.endswith(".dat"):

especie = filename.split(".")[0]

query = "SELECT COUNT(\*) FROM maestra\_especies m WHERE especie IN (SELECT c1.especie FROM conteo\_lncrna\_especie c1 LEFT JOIN conteo\_lncrna\_especie c2 ON c1.especie = REPLACE(c2.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c2.fuente = 'CantataDB' LEFT JOIN conteo\_lncrna\_especie c3 ON c1.especie = REPLACE(c3.especie, '\_', ' ') AND c1.fuente = 'Ensembl' AND c3.fuente = 'GreeNC' WHERE c1.fuente = 'Ensembl' AND (c2.fuente IS NOT NULL OR c3.fuente IS NOT NULL)) AND m.especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

result=cur.fetchone()

**if** result[0] > 0:

print("**\r**Procesando especie", especie, filename.split(".")[1]," ", end=" ")

query = "SELECT e.id\_especie FROM maestra\_especies e WHERE e.especie = REPLACE(**%s**, '\_', ' ')"

cur.execute(query, [especie])

id\_especie=cur.fetchone()

procesar\_features\_cpat(directory + "/" + filename, cur, id\_especie[0])

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Diamond – Generación de la base de datos

%%time

**import** **os**

!~/anaconda3/bin/diamond makedb --**in** '../Semana 05/diamond/uniprot-viridiplantae-reviewed.fasta' -d '../Semana 05/diamond/uniprot-viridiplantae-reviewed'

## : Diamond – Búsqueda de alineamientos

%%time

**import** **mysql.connector**

**import** **os**

**from** **IPython.utils** **import** io

print("Iniciando proceso...")

cursor = conn.cursor()

input\_directory\_pct = '../Semana 05/fasta\_pct\_para\_logit/'

input\_directory\_lncRNA = '../Semana 05/fasta\_para\_hexamer/'

output\_directory = '../Semana 05/diamond\_features/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Realizando búsqueda para " + str(especie[0]), " PCT ", end="")

diamond\_db = "'" + '../Semana 05/diamond/uniprot-viridiplantae-reviewed.dmnd' + "'"

fasta\_pct = "'" + input\_directory\_pct + especie[1] + ".PCT.fasta" + "'"

lncRNA\_cds = "'" + input\_directory\_lncRNA + especie[1] + ".lncRNA.fasta" + "'"

salida = "'" + output\_directory + especie[1] + ".PCT.tsv" + "'"

**with** io.capture\_output() **as** captured:

!~/anaconda3/bin/diamond blastx -d $diamond\_db -q $fasta\_pct -o $salida -k 5 --gapopen 11 --gapextend 1 --more-sensitive -f 6 qseqid pident length qframe qstart qend sstart send evalue bitscore

print("**\r**Realizando búsqueda para " + str(especie[0]), " lncRNA ", end="")

lncRNA\_cds = "'" + input\_directory\_lncRNA + especie[1] + ".lncRNA.fasta" + "'"

salida = "'" + output\_directory + especie[1] + ".lncRNA.tsv" + "'"

**with** io.capture\_output() **as** captured:

!~/anaconda3/bin/diamond blastx -d $diamond\_db -q $lncRNA\_cds -o $salida -k 5 --gapopen 11 --gapextend 1 --more-sensitive -f 6 qseqid pident length qframe qstart qend sstart send evalue bitscore

conn.close()

print("**\r**Proceso finalizado... ")

## : Diamond – Extracción de características

%%time

**import** **mysql.connector**

**import** **csv**

**import** **os**

conn = mysql.connector.connect(

host="localhost",

database="tesis2",

user="tesis2",

passwd="tesis2"

)

*#adaptado de https://github.com/gbgolding/crema/blob/master/bin/featuresetup\_module.py*

**def** transcript\_info\_dict(id\_especie, blast\_file, cursor, flg\_pct):

transcript\_dict = {}

**with** open(blast\_file, "r") **as** f:

tab\_reader = csv.reader(f, delimiter=("**\t**"))

line\_1 = next(tab\_reader)

first = line\_1[0]

score = [float(line\_1[9])]

with\_len = [[first, float(line\_1[1]), float(line\_1[2]), float(line\_1[3]), float(line\_1[9])]] *# name identity length frame score*

**for** row **in** tab\_reader:

**if** row[0] == first:

score.append(float(row[9]))

with\_len.append([row[0], float(row[1]), float(row[2]), float(row[3]), float(row[9])])

**else**:

transcript\_dict[first] = {}

transcript\_dict[first]["identity"] = float(0)

transcript\_dict[first]["align\_length"] = float(0)

max\_value = max(score)

max\_index = score.index(max\_value)

max\_len\_ident = with\_len[max\_index]

**if** max\_len\_ident[3] > 0:

transcript\_dict[first]["identity"] = float(max\_len\_ident[1])

transcript\_dict[first]["align\_length"] = float(max\_len\_ident[2])

score = [float(row[9])]

first = row[0]

with\_len = [[first, float(row[1]), float(row[2]), float(row[3]), float(row[9])]]

transcript\_dict[first] = {}

transcript\_dict[first]["identity"] = float(0)

transcript\_dict[first]["align\_length"] = float(0)

max\_value = max(score)

max\_index = score.index(max\_value)

max\_len\_ident = with\_len[max\_index]

**if** max\_len\_ident[3] > 0:

transcript\_dict[first]["identity"] = float(max\_len\_ident[1])

transcript\_dict[first]["align\_length"] = float(max\_len\_ident[2])

*#fin de código adaptado de https://github.com/gbgolding/crema/blob/master/bin/featuresetup\_module.py*

query = "SELECT cod\_secuencia, IFNULL(longitud, 0), IFNULL(orf\_length, 0) FROM secuencias\_features WHERE flg\_seleccionado = 1 AND id\_especie = **%s** AND flg\_pct = **%s**"

cursor.execute(query, (id\_especie, flg\_pct))

transcriptomas = cursor.fetchall()

**for** transcriptoma **in** transcriptomas:

cod\_secuencia = transcriptoma[0]

longitud = transcriptoma[1]

orf\_length = transcriptoma[2]

**if** transcriptoma[0] **in** transcript\_dict:

transcript\_dict[cod\_secuencia]["align\_perc\_len"] = float(transcript\_dict[cod\_secuencia]["align\_length"]/longitud)

**if** orf\_length == 0:

transcript\_dict[cod\_secuencia]["align\_perc\_ORF"] = float(0)

**else**:

transcript\_dict[cod\_secuencia]["align\_perc\_ORF"] = float(transcript\_dict[cod\_secuencia]["align\_length"]/orf\_length)

**else**:

transcript\_dict[cod\_secuencia] = {}

transcript\_dict[cod\_secuencia]["identity"] = float(0)

transcript\_dict[cod\_secuencia]["align\_length"] = float(0)

transcript\_dict[cod\_secuencia]["align\_perc\_len"] = float(0)

transcript\_dict[cod\_secuencia]["align\_perc\_ORF"] = float(0)

query = "UPDATE secuencias\_features SET alignment = **%s**, alignment\_length = **%s**, alignment\_proportion = **%s**, alignment\_proportion\_orf = **%s** WHERE id\_especie = **%s** AND cod\_secuencia = **%s**"

cursor.execute(query, (transcript\_dict[cod\_secuencia]["identity"], transcript\_dict[cod\_secuencia]["align\_length"], transcript\_dict[cod\_secuencia]["align\_perc\_len"], transcript\_dict[cod\_secuencia]["align\_perc\_ORF"], id\_especie, cod\_secuencia))

print("Iniciando proceso...")

cursor = conn.cursor()

output\_directory = '../Semana 05/diamond\_features/'

query = "SELECT m.id\_especie, REPLACE(m.especie, ' ', '\_') FROM maestra\_especies m JOIN especies\_seleccionadas es ON m.especie = es.especie"

cursor.execute(query)

especies = cursor.fetchall()

**for** especie **in** especies:

print("**\r**Exportando features para " + str(especie[0]), " PCT ", end="")

salida = output\_directory + especie[1] + ".PCT.tsv"

transcript\_info\_dict(especie[0], salida, cursor, 1)

conn.commit()

print("**\r**Exportando features para " + str(especie[0]), " lncRNA ", end="")

salida = output\_directory + especie[1] + ".lncRNA.tsv"

transcript\_info\_dict(especie[0], salida, cursor, 0)

conn.commit()

conn.close()

print("**\r**Proceso finalizado... ")

## : Librería util\_bd.py

%%writefile ./libs/util\_bd.py

**import** **mysql.connector**

**import** **myloginpath**

**import** **pandas** **as** **pd**

**def** resultados\_query(query):

conf = myloginpath.parse('tesis2')

conn = mysql.connector.connect(\*\*conf, db="tesis2")

cursor = conn.cursor()

cursor.execute(query)

resultado = cursor.fetchall()

conn.close()

**return** resultado

**def** ejecutar\_query(query):

conf = myloginpath.parse('tesis2')

conn = mysql.connector.connect(\*\*conf, db="tesis2")

cursor = conn.cursor()

cursor.execute(query)

conn.close()

**def** mostrar\_resultado\_query(query):

conf = myloginpath.parse('tesis2')

conn = mysql.connector.connect(\*\*conf, db="tesis2")

df = pd.read\_sql\_query(query, conn)

display(df)

conn.close()

## : Librería util\_fasta.py

%%writefile ./libs/util\_fasta.py

**import** **os**

**def** generar\_fasta(secuencias, archivo, tamanio\_por\_linea=80):

t\_tamanio = tamanio\_por\_linea

f = open(archivo ,"w+")

**for** transcrito **in** secuencias:

f.write(">**%s\n**" % (transcrito[0].strip().upper()))

seq = transcrito[1]

t\_partes = [seq[i:i+t\_tamanio] **for** i **in** range(0, len(seq), t\_tamanio)]

**for** t\_parte **in** t\_partes:

f.write("**%s\n**" % (t\_parte))

f.close()

**def** leer\_fasta(archivo, limite = 0):

transcritos = {}

cod\_secuencia = ""

secuencia = ""

f = open(archivo, "r")

**for** linea **in** f:

**if** linea.startswith(">"):

**if** secuencia != "":

transcritos[cod\_secuencia] = secuencia

secuencia = ""

limite -= 1

**if** limite == 0:

**break**

cod\_secuencia = linea.rstrip("**\n**").lstrip(">").strip().upper()

**else**:

secuencia += linea.rstrip("**\n**")

**if** secuencia != "":

transcritos[cod\_secuencia] = secuencia

secuencia = ""

f.close()

**return** transcritos

**def** leer\_fasta\_list(archivo, limite = 0):

transcritos = list()

cod\_secuencia = ""

secuencia = ""

f = open(archivo, "r")

**for** linea **in** f:

**if** linea.startswith(">"):

**if** secuencia != "":

transcritos.append((cod\_secuencia, secuencia))

secuencia = ""

limite -= 1

**if** limite == 0:

**break**

cod\_secuencia = linea.rstrip("**\n**").lstrip(">").strip().upper()

**else**:

secuencia += linea.rstrip("**\n**")

**if** secuencia != "":

transcritos.append((cod\_secuencia, secuencia))

secuencia = ""

f.close()

**return** transcritos

## : Librería util\_caracteristicas.py

%%writefile ./libs/util\_caracteristicas.py

**import** **os**

**import** **util\_fasta**

**from** **Bio.SeqUtils** **import** GC

**import** **csv**

**from** **sklearn.externals.joblib** **import** dump, load

**def** generar\_modelo\_CPAT(archivo\_lncRNA, archivo\_PCT, archivo\_CDS, carpeta\_cpat):

\_generar\_modelo\_CPAT\_hexamer(archivo\_lncRNA, archivo\_CDS, carpeta\_cpat)

\_generar\_modelo\_CPAT\_logit(archivo\_lncRNA, archivo\_PCT, carpeta\_cpat)

**def** \_generar\_modelo\_CPAT\_hexamer(archivo\_lncRNA, archivo\_CDS, carpeta\_cpat):

script = "~/anaconda3/bin/make\_hexamer\_tab.py"

fasta\_cds = "'" + archivo\_CDS + "'"

fasta\_lncRNA = "'" + archivo\_lncRNA + "'"

salida = "'" + carpeta\_cpat + "/hexamer.tsv" + "'"

comando = "**{}** -c **{}** -n **{}** > **{}**".format(script, fasta\_cds, fasta\_lncRNA, salida)

os.system(comando)

**def** \_generar\_modelo\_CPAT\_logit(archivo\_lncRNA, archivo\_PCT, carpeta\_cpat):

script = "~/anaconda3/bin/make\_logitModel.py"

hexamer = "'" + carpeta\_cpat + "/hexamer.tsv" + "'"

fasta\_pct = "'" + archivo\_PCT + "'"

fasta\_lncRNA = "'" + archivo\_lncRNA + "'"

salida = "'" + carpeta\_cpat + "/fold" + "'"

comando = "**{}** -x **{}** -c **{}** -n **{}** -o **{}**".format(script, hexamer, fasta\_pct, fasta\_lncRNA, salida)

os.system(comando)

**def** ejecutar\_diamond(archivo\_entrada, diamond\_db, archivo\_salida):

script = "~/anaconda3/bin/diamond"

diamond\_bd = "'" + diamond\_db + "'"

salida = "'" + archivo\_salida + "'"

comando = "**{}** blastx -d **{}** -q **{}** -o **{}** -k 5 --gapopen 11 --gapextend 1 --more-sensitive -f 6 qseqid pident length qframe qstart qend sstart send evalue bitscore".format(script, diamond\_bd, archivo\_entrada, salida)

os.system(comando)

**def** ejecutar\_cpat(archivo\_entrada, carpeta\_cpat, archivo\_salida):

script = "~/anaconda3/bin/cpat.py"

logit = "'" + carpeta\_cpat + "/fold.logit.RData" + "'"

hexamer = "'" + carpeta\_cpat + "/hexamer.tsv" + "'"

salida = "'" + archivo\_salida + "'"

comando = "**{}** -g **{}** -d **{}** -x **{}** -o **{}**".format(script, archivo\_entrada, logit, hexamer, salida)

os.system(comando)

**def** generar\_features\_base(archivo\_entrada, archivo\_cpat, archivo\_diamond, archivo\_salida):

transcritos = util\_fasta.leer\_fasta(archivo\_entrada)

transcript\_dict = {}

**for** k **in** transcritos.keys():

transcript\_dict[k.strip().upper()] = {

"length" : len(transcritos[k]),

"gc" : GC(transcritos[k]),

"orf\_length" : 0,

"orf\_coverage" : float(0),

"hexamer\_score" : float(0),

"fickett\_score" : float(0),

"identity" : float(0),

"align\_length" : float(0),

"align\_perc\_len" : float(0),

"align\_perc\_orf" : float(0)

}

*#adaptado de https://github.com/gbgolding/crema/blob/master/bin/featuresetup\_module.py*

**with** open(archivo\_cpat, "r") **as** f:

cpat\_reader = csv.reader(f, delimiter=("**\t**"))

next(cpat\_reader, **None**) *# skip header*

**for** row **in** cpat\_reader:

cod\_secuencia = row[0]

transcript\_dict[cod\_secuencia]["orf\_length"] = float(row[2])

transcript\_dict[cod\_secuencia]["orf\_coverage"] = float(row[2])/float(transcript\_dict[cod\_secuencia]["length"])

transcript\_dict[cod\_secuencia]["fickett\_score"] = float(row[3])

transcript\_dict[cod\_secuencia]["hexamer\_score"] = float(row[4])

**with** open(archivo\_diamond, "r") **as** f:

tab\_reader = csv.reader(f, delimiter=("**\t**"))

line\_1 = next(tab\_reader)

first = line\_1[0].upper()

score = [float(line\_1[9])]

with\_len = [[first, float(line\_1[1]), float(line\_1[2]), float(line\_1[3]), float(line\_1[9])]] *# name identity length frame score*

**for** row **in** tab\_reader:

**if** row[0].upper() == first:

score.append(float(row[9]))

with\_len.append([row[0].upper(), float(row[1]), float(row[2]), float(row[3]), float(row[9])])

**else**:

transcript\_dict[first]["identity"] = float(0)

transcript\_dict[first]["align\_length"] = float(0)

max\_value = max(score)

max\_index = score.index(max\_value)

max\_len\_ident = with\_len[max\_index]

**if** max\_len\_ident[3] > 0:

transcript\_dict[first]["identity"] = float(max\_len\_ident[1])

transcript\_dict[first]["align\_length"] = float(max\_len\_ident[2])

transcript\_dict[first]["align\_perc\_len"] = float(transcript\_dict[first]["align\_length"]/transcript\_dict[first]["length"])

transcript\_dict[first]["align\_perc\_orf"] = (0 **if** transcript\_dict[first]["orf\_length"] == 0 **else** float(transcript\_dict[first]["align\_length"]/transcript\_dict[first]["orf\_length"]))

score = [float(row[9])]

first = row[0].upper()

with\_len = [[first, float(row[1]), float(row[2]), float(row[3]), float(row[9])]]

transcript\_dict[first]["identity"] = float(0)

transcript\_dict[first]["align\_length"] = float(0)

max\_value = max(score)

max\_index = score.index(max\_value)

max\_len\_ident = with\_len[max\_index]

**if** max\_len\_ident[3] > 0:

transcript\_dict[first]["identity"] = float(max\_len\_ident[1])

transcript\_dict[first]["align\_length"] = float(max\_len\_ident[2])

*#fin de código adaptado de https://github.com/gbgolding/crema/blob/master/bin/featuresetup\_module.py*

dump(transcript\_dict, archivo\_salida)

**def** generar\_features(archivo\_entrada, features\_base, archivo\_cpat, archivo\_salida):

transcritos = util\_fasta.leer\_fasta(archivo\_entrada)

features\_globales = load(features\_base)

transcript\_dict = {}

**for** k **in** transcritos.keys():

transcript\_dict[k.strip().upper()] = {

"length" : features\_globales[k.strip().upper()]["length"],

"gc" : features\_globales[k.strip().upper()]["gc"],

"orf\_length" : features\_globales[k.strip().upper()]["orf\_length"],

"orf\_coverage" : features\_globales[k.strip().upper()]["orf\_coverage"],

"hexamer\_score" : float(0),

"fickett\_score" : float(0),

"identity" : features\_globales[k.strip().upper()]["identity"],

"align\_length" : features\_globales[k.strip().upper()]["align\_length"],

"align\_perc\_len" : features\_globales[k.strip().upper()]["align\_perc\_len"],

"align\_perc\_orf" : features\_globales[k.strip().upper()]["align\_perc\_orf"]

}

**with** open(archivo\_cpat, "r") **as** f:

cpat\_reader = csv.reader(f, delimiter=("**\t**"))

next(cpat\_reader, **None**) *# skip header*

**for** row **in** cpat\_reader:

cod\_secuencia = row[0].strip().upper()

**if** cod\_secuencia **in** transcript\_dict:

transcript\_dict[cod\_secuencia]["fickett\_score"] = float(row[3])

transcript\_dict[cod\_secuencia]["hexamer\_score"] = float(row[4])

dump(transcript\_dict, archivo\_salida)

## : Librería util\_modelo\_final.py

%%writefile ./libs/util\_modelo\_final.py

**import** **os**

**import** **shutil**

**import** **util\_fasta**, **util\_caracteristicas**

**from** **sklearn.externals.joblib** **import** Parallel, delayed, dump, load

**import** **hashlib**

**from** **sklearn.pipeline** **import** Pipeline

**from** **sklearn.svm** **import** SVC

**from** **sklearn.base** **import** BaseEstimator, TransformerMixin

**from** **sklearn.preprocessing** **import** RobustScaler

**from** **sklearn.model\_selection** **import** LeaveOneGroupOut, GridSearchCV, StratifiedKFold

**from** **sklearn.metrics** **import** classification\_report, confusion\_matrix, precision\_recall\_fscore\_support, precision\_recall\_curve, average\_precision\_score, roc\_curve, roc\_auc\_score

**import** **hashlib**

**import** **util\_fasta**

**import** **os**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.utils** **import** shuffle

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Dropout

**from** **keras.wrappers.scikit\_learn** **import** KerasClassifier

**from** **keras.constraints** **import** maxnorm

**from** **keras.utils** **import** to\_categorical

**class** **Tesis2**():

**def** \_\_init\_\_(self, carpeta\_base=".", n\_jobs=-1, verbose=0, tuned\_parameters=[{'svc\_\_kernel': ['rbf'], 'svc\_\_gamma': [1e-3], 'svc\_\_C': [0.1,0.5,0.9,2]}], score = ['accuracy','precision','recall']):

self.carpeta\_base = carpeta\_base

self.n\_jobs = n\_jobs

self.verbose = verbose

self.tuned\_parameters = tuned\_parameters

self.score = score

self.carpeta\_data\_base = self.carpeta\_base + "/data"

self.carpeta\_fold\_base = self.carpeta\_base + "/folds"

self.carpeta\_modelo\_base = self.carpeta\_base + "/modelo\_final"

**if** **not** os.path.isdir(self.carpeta\_base):

os.mkdir(self.carpeta\_base)

self.diamond\_db = "./feature\_engine/Diamond\_BD/uniprot-viridiplantae-reviewed.dmnd"

self.modelo\_final\_generado = **False**

self.modelo\_referencial\_generado = **False**

**def** generar\_modelo\_final(self):

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generando llaves \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_llaves\_clases()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Armando folds \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.armar\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\* Generando modelo cpat para cada fold \*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_cpats\_de\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\* Ejecutando cpat y diamond sobre los folds \*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.ejecutar\_cpat\_diamond\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\* Serializando features \*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_features\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\* Generando modelo final \*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.entrenar\_modelo\_final()

self.modelo\_final\_generado = **True**

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\* Limpiando archivos intermedios \*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.limpiar\_archivos\_intermedios()

**if** (self.verbose > 1):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\* Mostrando resultados \*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.mostrar\_resultados()

**def** generar\_modelo\_final\_keras(self):

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Generando llaves \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_llaves\_clases()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Armando folds \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.armar\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\* Generando modelo cpat para cada fold \*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_cpats\_de\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\* Ejecutando cpat y diamond sobre los folds \*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.ejecutar\_cpat\_diamond\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\* Serializando features \*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.generar\_features\_folds()

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\* Generando modelo final \*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.entrenar\_modelo\_final\_keras()

self.modelo\_final\_generado = **True**

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\* Limpiando archivos intermedios \*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.limpiar\_archivos\_intermedios()

**if** (self.verbose > 1):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\* Mostrando resultados \*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.mostrar\_resultados()

**def** generar\_modelos\_referenciales(self):

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\* Generando modelos referenciales \*\*\*\*\*\*\*\*\*")

**if** (self.verbose > 1): print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.entrenar\_modelos\_referenciales()

self.modelo\_referencial\_generado = **True**

**if** (self.modelo\_final\_generado **and** self.verbose > 1):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\* Mostrando resultados \*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

self.mostrar\_resultados\_referencial\_vs\_final()

**def** carpeta\_data(self):

**return** self.carpeta\_data\_base

**def** carpeta\_fold(self):

**return** self.carpeta\_fold\_base

**def** carpeta\_modelo(self):

**return** self.carpeta\_modelo\_base

**def** folder\_clase(self, num\_clase):

**return** self.carpeta\_data() + "/clase\_" + str(num\_clase)

**def** archivo\_clase(self, num\_clase, tipo):

**return** self.folder\_clase(num\_clase) + "/" + tipo + ".fa"

**def** obtener\_num\_clases(self):

num\_clases = 0

**if** **not** os.path.isfile(self.carpeta\_base + "/num\_clases.bin"):

**while** os.path.isdir(self.folder\_clase(num\_clases + 1)):

num\_clases = num\_clases + 1

dump(num\_clases, self.carpeta\_base + "/num\_clases.bin")

num\_clases = load(self.carpeta\_base + "/num\_clases.bin")

**return** num\_clases

**def** iterador\_clases(self):

**return** range(1, self.obtener\_num\_clases() + 1)

**def** obtener\_primera\_secuencia(self, num\_clase):

secuencias = util\_fasta.leer\_fasta(self.archivo\_clase(num\_clase, "lncRNA"), 1)

**return** list(secuencias.keys())[0]

**def** obtener\_todas\_las\_secuencias(self):

**return** {num\_clase : self.obtener\_primera\_secuencia(num\_clase) **for** num\_clase **in** self.iterador\_clases()}

**def** generar\_llaves\_clases(self):

secuencias = self.obtener\_todas\_las\_secuencias()

llaves = {num\_clase : "" **for** num\_clase **in** secuencias.keys()}

llaves[0] = "" *#llave cero corresponde a todo el universo*

**for** i\_clase **in** self.iterador\_clases():

llaves[0] += secuencias[i\_clase]

**for** j\_clase **in** self.iterador\_clases():

**if** (i\_clase != j\_clase):

llaves[j\_clase] += secuencias[i\_clase]

llaves[0] = hashlib.sha224(llaves[0].encode()).hexdigest()

**for** i\_clase **in** self.iterador\_clases():

llaves[i\_clase] = hashlib.sha224(llaves[i\_clase].encode()).hexdigest()

dump(llaves, self.carpeta\_base + "/llaves\_clases.bin")

**def** obtener\_llaves\_clases(self):

**return** load(self.carpeta\_base + "/llaves\_clases.bin")

**def** carpeta\_fold\_clase(self, llave):

**return** self.carpeta\_fold() + "/fold\_clase\_" + str(llave)

**def** archivo\_fold\_clase(self, llave, tipoTrainTest, tipoRNA):

**return** self.carpeta\_fold\_clase(llave) + "/" + tipoTrainTest + "/" + tipoRNA + ".fa"

**def** armar\_fold\_final(self, tipo):

llave = self.obtener\_llaves\_clases()[0]

**with** open(self.archivo\_fold\_clase(llave, "train", tipo), "w+") **as** outfile:

**for** num\_clase **in** self.iterador\_clases():

**with** open(self.archivo\_clase(num\_clase, tipo), "r") **as** infile:

**for** inline **in** infile:

outfile.write(inline)

**def** armar\_fold\_clase(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

os.mkdir(self.carpeta\_fold\_clase(llave))

os.mkdir(self.carpeta\_fold\_clase(llave) + "/train")

**for** tipo **in** ["lncRNA", "PCT", "CDS"]:

**with** open(self.archivo\_fold\_clase(llave, "train", tipo), "w+") **as** outfile:

**for** j\_num\_clase **in** self.iterador\_clases():

**if** num\_clase != j\_num\_clase:

**with** open(self.archivo\_clase(j\_num\_clase, tipo)) **as** infile:

**for** inline **in** infile:

outfile.write(inline)

os.mkdir(self.carpeta\_fold\_clase(llave) + "/test")

**for** tipo **in** ["lncRNA", "PCT"]:

**with** open(self.archivo\_fold\_clase(llave, "test", tipo), "w+") **as** outfile:

**with** open(self.archivo\_clase(num\_clase, tipo)) **as** infile:

**for** inline **in** infile:

outfile.write(inline)

**def** armar\_folds(self):

**if** os.path.isdir(self.carpeta\_fold()):

shutil.rmtree(self.carpeta\_fold())

os.mkdir(self.carpeta\_fold())

llave = self.obtener\_llaves\_clases()[0]

**if** **not** os.path.isdir(self.carpeta\_fold\_clase(llave)):

os.mkdir(self.carpeta\_fold\_clase(llave))

**if** **not** os.path.isdir(self.carpeta\_fold\_clase(llave) + "/train"):

os.mkdir(self.carpeta\_fold\_clase(llave) + "/train")

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_armar\_fold\_final)(self, tipo) **for** tipo **in** ["lncRNA", "PCT", "CDS"])

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_armar\_fold\_clase)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

**def** carpeta\_fold\_cpat(self, llave):

**return** self.carpeta\_fold\_clase(llave) + "/cpat"

**def** generar\_cpat\_fold(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

archivo\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT")

archivo\_CDS = self.archivo\_fold\_clase(llave, "train", "CDS")

carpeta\_cpat = self.carpeta\_fold\_cpat(llave)

**if** **not** os.path.isdir(carpeta\_cpat):

os.mkdir(carpeta\_cpat)

util\_caracteristicas.generar\_modelo\_CPAT(archivo\_lncRNA, archivo\_PCT, archivo\_CDS, carpeta\_cpat)

**def** generar\_cpat\_final(self):

llave = self.obtener\_llaves\_clases()[0]

archivo\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT")

archivo\_CDS = self.archivo\_fold\_clase(llave, "train", "CDS")

carpeta\_cpat = self.carpeta\_fold\_cpat(llave)

**if** **not** os.path.isdir(carpeta\_cpat):

os.mkdir(carpeta\_cpat)

util\_caracteristicas.generar\_modelo\_CPAT(archivo\_lncRNA, archivo\_PCT, archivo\_CDS, carpeta\_cpat)

**def** limpieza\_archivos\_CDS(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

os.remove(self.archivo\_fold\_clase(llave, "train", "CDS"))

**def** generar\_cpats\_de\_folds(self):

self.generar\_cpat\_final()

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_generar\_cpat\_fold)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

self.limpieza\_archivos\_CDS(0)

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_limpieza\_archivos\_CDS)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

**def** ejecutar\_cpat\_fold(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

archivo\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT")

carpeta\_cpat = self.carpeta\_fold\_cpat(llave)

util\_caracteristicas.ejecutar\_cpat(archivo\_lncRNA, carpeta\_cpat, archivo\_lncRNA.replace(".fa", ".cpat"))

os.remove(archivo\_lncRNA.replace(".fa", ".cpat") + ".dat")

os.remove(archivo\_lncRNA.replace(".fa", ".cpat") + ".r")

util\_caracteristicas.ejecutar\_cpat(archivo\_PCT, carpeta\_cpat, archivo\_PCT.replace(".fa", ".cpat"))

os.remove(archivo\_PCT.replace(".fa", ".cpat") + ".dat")

os.remove(archivo\_PCT.replace(".fa", ".cpat") + ".r")

**def** ejecutar\_cpat\_diamond\_final(self):

llave = self.obtener\_llaves\_clases()[0]

archivo\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT")

diamond\_db = self.diamond\_db

carpeta\_cpat = self.carpeta\_fold\_cpat(llave)

util\_caracteristicas.ejecutar\_diamond(archivo\_lncRNA, diamond\_db, archivo\_lncRNA.replace(".fa", ".dmnd"))

util\_caracteristicas.ejecutar\_diamond(archivo\_PCT, diamond\_db, archivo\_PCT.replace(".fa", ".dmnd"))

util\_caracteristicas.ejecutar\_cpat(archivo\_lncRNA, carpeta\_cpat, archivo\_lncRNA.replace(".fa", ".cpat"))

os.remove(archivo\_lncRNA.replace(".fa", ".cpat") + ".dat")

os.remove(archivo\_lncRNA.replace(".fa", ".cpat") + ".r")

util\_caracteristicas.ejecutar\_cpat(archivo\_PCT, carpeta\_cpat, archivo\_PCT.replace(".fa", ".cpat"))

os.remove(archivo\_PCT.replace(".fa", ".cpat") + ".dat")

os.remove(archivo\_PCT.replace(".fa", ".cpat") + ".r")

**def** ejecutar\_cpat\_diamond\_folds(self):

self.ejecutar\_cpat\_diamond\_final()

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_ejecutar\_cpat\_fold)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

**def** archivo\_features\_clase(self, llave, tipoTrainTest, tipoRNA):

**return** self.archivo\_fold\_clase(llave, tipoTrainTest, tipoRNA).replace(".fa", ".ft")

**def** generar\_features\_fold(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

features\_base\_lncRNA = self.archivo\_features\_clase(self.obtener\_llaves\_clases()[0], "train", "lncRNA")

features\_base\_PCT = self.archivo\_features\_clase(self.obtener\_llaves\_clases()[0], "train", "PCT")

**for** tipo **in** ["train", "test"]:

archivo\_lncRNA = self.archivo\_fold\_clase(llave, tipo, "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, tipo, "PCT")

archivo\_cpat\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA").replace(".fa", ".cpat")

archivo\_cpat\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT").replace(".fa", ".cpat")

salida\_lncRNA = self.archivo\_features\_clase(llave, tipo, "lncRNA")

salida\_PCT = self.archivo\_features\_clase(llave, tipo, "PCT")

util\_caracteristicas.generar\_features(archivo\_lncRNA, features\_base\_lncRNA, archivo\_cpat\_lncRNA, salida\_lncRNA)

util\_caracteristicas.generar\_features(archivo\_PCT, features\_base\_PCT, archivo\_cpat\_PCT, salida\_PCT)

**def** generar\_features\_final(self):

llave = self.obtener\_llaves\_clases()[0]

archivo\_lncRNA = self.archivo\_fold\_clase(llave, "train", "lncRNA")

archivo\_PCT = self.archivo\_fold\_clase(llave, "train", "PCT")

salida\_lncRNA = self.archivo\_features\_clase(llave, "train", "lncRNA")

salida\_PCT = self.archivo\_features\_clase(llave, "train", "PCT")

util\_caracteristicas.generar\_features\_base(archivo\_lncRNA, archivo\_lncRNA.replace(".fa", ".cpat"), archivo\_lncRNA.replace(".fa", ".dmnd"), salida\_lncRNA)

util\_caracteristicas.generar\_features\_base(archivo\_PCT, archivo\_PCT.replace(".fa", ".cpat"), archivo\_PCT.replace(".fa", ".dmnd"), salida\_PCT)

**def** generar\_features\_folds(self):

self.generar\_features\_final()

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_generar\_features\_fold)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

**def** obtener\_data\_entrenamiento(self):

llave = self.obtener\_llaves\_clases()[0]

codigos\_lncRNA = util\_fasta.leer\_fasta\_list(self.archivo\_fold\_clase(llave, "train", "lncRNA"))

codigos\_PCT = util\_fasta.leer\_fasta\_list(self.archivo\_fold\_clase(llave, "train", "PCT"))

codigos\_lncRNA = [(x[0],"") **for** x **in** codigos\_lncRNA]

codigos\_PCT = [(x[0],"") **for** x **in** codigos\_PCT]

cantidad\_transcritos = len(codigos\_lncRNA)//self.obtener\_num\_clases()

y = ([1] \* len(codigos\_lncRNA)) + ([0] \* len(codigos\_PCT))

groups = list()

**for** \_ **in** range(2):

**for** num\_clase **in** self.iterador\_clases():

groups += ([num\_clase] \* (cantidad\_transcritos))

**return** codigos\_lncRNA + codigos\_PCT, y, groups, cantidad\_transcritos

**def** obtener\_data\_entrenamiento\_keras(self):

llave = self.obtener\_llaves\_clases()[0]

codigos\_lncRNA = util\_fasta.leer\_fasta\_list(self.archivo\_fold\_clase(llave, "train", "lncRNA"))

codigos\_PCT = util\_fasta.leer\_fasta\_list(self.archivo\_fold\_clase(llave, "train", "PCT"))

codigos\_lncRNA = [(x[0],"") **for** x **in** codigos\_lncRNA]

codigos\_PCT = [(x[0],"") **for** x **in** codigos\_PCT]

cantidad\_transcritos = len(codigos\_lncRNA)//self.obtener\_num\_clases()

y = ([1] \* len(codigos\_lncRNA)) + ([0] \* len(codigos\_PCT))

groups = list()

**for** \_ **in** range(2):

**for** num\_clase **in** self.iterador\_clases():

groups += ([num\_clase] \* (cantidad\_transcritos))

x = codigos\_lncRNA + codigos\_PCT

reserva\_x = list()

reserva\_y = list()

reserva\_groups = list()

iremove = list()

num\_transcritos = len(x)

num\_transcritos\_por\_grupo = cantidad\_transcritos

**for** i **in** range(num\_transcritos//(num\_transcritos\_por\_grupo\*2)):

reserva\_x.append(x[i \* num\_transcritos\_por\_grupo])

reserva\_y.append(y[i \* num\_transcritos\_por\_grupo])

reserva\_groups.append(groups[i \* num\_transcritos\_por\_grupo])

iremove.insert(0, i \* num\_transcritos\_por\_grupo)

**for** i **in** iremove:

**del** x[i]

**del** y[i]

**del** groups[i]

x, y, groups = shuffle(x, y, groups, random\_state=7)

**for** i **in** range(num\_transcritos//(num\_transcritos\_por\_grupo\*2)):

x.insert(i, reserva\_x[i])

y.insert(i, reserva\_y[i])

groups.insert(i, reserva\_groups[i])

**return** x, y, groups, cantidad\_transcritos

**def** entrenar\_modelo\_final(self):

**if** os.path.isdir(self.carpeta\_modelo()):

shutil.rmtree(self.carpeta\_modelo())

os.mkdir(self.carpeta\_modelo())

X\_train, y\_train, groups, cantidad\_transcritos = self.obtener\_data\_entrenamiento()

svm\_pipeline = Pipeline(steps=[('features', GeneradorFeatures(self, cantidad\_transcritos, self.obtener\_num\_clases())), ('scaler', RobustScaler()), ('svc', SVC())])

logo = LeaveOneGroupOut()

clf = GridSearchCV(svm\_pipeline, self.tuned\_parameters, cv=logo, scoring=self.score, n\_jobs=self.n\_jobs, refit="accuracy", return\_train\_score = **True**, verbose=self.verbose)

clf.fit(X\_train, y\_train, groups) *#requerido por LeaveOneGroupOut*

resultado = {

"accuracy" : clf.cv\_results\_['mean\_test\_accuracy'][clf.best\_index\_],

"precision" : clf.cv\_results\_['mean\_test\_precision'][clf.best\_index\_],

"recall" : clf.cv\_results\_['mean\_test\_recall'][clf.best\_index\_]

}

dump(resultado, self.carpeta\_modelo() + "/resultado.bin")

dump(clf.best\_params\_, self.carpeta\_modelo() + "/params.bin")

dump(clf.cv\_results\_, self.carpeta\_modelo() + "/cv\_results.bin")

dump(clf.best\_estimator\_, self.carpeta\_modelo() + "/modelo.plk")

**def** entrenar\_modelo\_final\_keras(self):

**if** os.path.isdir(self.carpeta\_modelo()):

shutil.rmtree(self.carpeta\_modelo())

os.mkdir(self.carpeta\_modelo())

X\_train, y\_train, groups, cantidad\_transcritos = self.obtener\_data\_entrenamiento\_keras()

*#X\_train, y\_train, groups = shuffle(X\_train, y\_train, groups, random\_state=7)*

keras\_pipeline = Pipeline(steps=[('features', GeneradorFeaturesKeras(self, cantidad\_transcritos, self.obtener\_num\_clases())), ('scaler', RobustScaler()), ('keras', KerasClassifier(build\_fn=crear\_modelo\_keras, verbose=0))])

logo = LeaveOneGroupOut()

clf = GridSearchCV(keras\_pipeline, self.tuned\_parameters, cv=logo, scoring=self.score, n\_jobs=self.n\_jobs, refit="accuracy", return\_train\_score = **True**, verbose=self.verbose)

clf.fit(X\_train, y\_train, groups) *#requerido por LeaveOneGroupOut*

resultado = {

"accuracy" : clf.cv\_results\_['mean\_test\_accuracy'][clf.best\_index\_],

"precision" : clf.cv\_results\_['mean\_test\_precision'][clf.best\_index\_],

"recall" : clf.cv\_results\_['mean\_test\_recall'][clf.best\_index\_]

}

dump(resultado, self.carpeta\_modelo() + "/resultado.bin")

dump(clf.best\_params\_, self.carpeta\_modelo() + "/params.bin")

dump(clf.cv\_results\_, self.carpeta\_modelo() + "/cv\_results.bin")

dump(clf.best\_estimator\_, self.carpeta\_modelo() + "/modelo.plk")

**def** limpieza\_archivos\_finales\_fasta\_ruta(self, llave):

shutil.rmtree(self.carpeta\_fold\_clase(llave))

**def** limpieza\_archivos\_finales\_fasta(self, num\_clase):

llave = self.obtener\_llaves\_clases()[num\_clase]

**if** num\_clase == 0:

shutil.rmtree(self.carpeta\_fold\_clase(llave) + "/train")

**else**:

self.limpieza\_archivos\_finales\_fasta\_ruta(llave)

**def** limpiar\_archivos\_intermedios(self):

self.limpieza\_archivos\_finales\_fasta(0)

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_limpieza\_archivos\_finales\_fasta)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

self.features\_pre\_generados = **False**

**def** mostrar\_resultados(self):

**if** (**not** self.modelo\_final\_generado):

print("Debe generar el modelo final")

**return**

display(load(self.carpeta\_modelo() + "/cv\_results.bin"))

display(load(self.carpeta\_modelo() + "/params.bin"))

display(load(self.carpeta\_modelo() + "/resultado.bin"))

**if** (self.modelo\_referencial\_generado):

self.mostrar\_resultados\_referencial\_vs\_final()

**def** devolver\_resultado(self):

**return** load(self.carpeta\_modelo() + "/resultado.bin")

**def** devolver\_mejor\_parametro(self):

**return** load(self.carpeta\_modelo() + "/params.bin")

**def** devolver\_cv\_results(self):

**return** load(self.carpeta\_modelo() + "/cv\_results.bin")

**def** devolver\_mejor\_modelo(self):

**return** load(self.carpeta\_modelo() + "/modelo.plk")

**def** preparar\_data\_modelo\_referencial(self, num\_clase):

carpeta\_base\_referencial = self.carpeta\_base + "/modelos\_referenciales/clase\_" + str(num\_clase)

**if** **not** os.path.isdir(carpeta\_base\_referencial):

os.mkdir(carpeta\_base\_referencial)

**if** **not** os.path.isdir(carpeta\_base\_referencial + "/data"):

os.mkdir(carpeta\_base\_referencial + "/data")

clase\_positiva = util\_fasta.leer\_fasta\_list(self.archivo\_clase(num\_clase, "lncRNA"))

PCT = util\_fasta.leer\_fasta(self.archivo\_clase(num\_clase, "PCT"))

CDS = util\_fasta.leer\_fasta(self.archivo\_clase(num\_clase, "CDS"))

clase\_negativa = list()

**for** k **in** PCT.keys():

clase\_negativa.append((k, PCT[k], CDS[k]))

X = clase\_positiva + clase\_negativa

y = ([1] \* len(clase\_positiva)) + ([0] \* len(clase\_negativa))

skf = StratifiedKFold(n\_splits=10)

isplit = 1

**for** \_, test **in** skf.split(X, y):

split\_lncRNA = list()

split\_PCT = list()

split\_CDS = list()

**for** itest **in** test:

**if** y[itest] == 1:

split\_lncRNA.append(X[itest])

**else**:

split\_PCT.append((X[itest][0], X[itest][1]))

split\_CDS.append((X[itest][0], X[itest][2]))

**if** **not** os.path.isdir(carpeta\_base\_referencial + "/data/clase\_" + str(isplit)):

os.mkdir(carpeta\_base\_referencial + "/data/clase\_" + str(isplit))

util\_fasta.generar\_fasta(split\_lncRNA, carpeta\_base\_referencial + "/data/clase\_" + str(isplit) + "/lncRNA.fa")

util\_fasta.generar\_fasta(split\_PCT, carpeta\_base\_referencial + "/data/clase\_" + str(isplit) + "/PCT.fa")

util\_fasta.generar\_fasta(split\_CDS, carpeta\_base\_referencial + "/data/clase\_" + str(isplit) + "/CDS.fa")

isplit += 1

**def** instanciar\_modelo\_referencial(self, num\_clase):

carpeta\_base\_referencial = self.carpeta\_base + "/modelos\_referenciales/clase\_" + str(num\_clase)

**return** Tesis2(carpeta\_base=carpeta\_base\_referencial, n\_jobs=self.n\_jobs, verbose=0, tuned\_parameters=self.tuned\_parameters, score=self.score)

**def** crear\_modelo\_referencial(self, num\_clase):

carpeta\_base\_referencial = self.carpeta\_base + "/modelos\_referenciales/clase\_" + str(num\_clase)

self.preparar\_data\_modelo\_referencial(num\_clase)

tesis2 = self.instanciar\_modelo\_referencial(num\_clase)

tesis2.generar\_modelo\_final()

shutil.rmtree(carpeta\_base\_referencial + "/data")

**def** entrenar\_modelos\_referenciales(self):

**if** **not** os.path.isdir(self.carpeta\_base + "/modelos\_referenciales"):

os.mkdir(self.carpeta\_base + "/modelos\_referenciales")

Parallel(n\_jobs=self.n\_jobs, verbose=self.verbose)(delayed(wrapper\_crear\_modelo\_referencial)(self, num\_clase) **for** num\_clase **in** self.iterador\_clases())

**def** obtener\_resultados\_referencial\_vs\_final(self):

resultado\_referencial = list()

resultado\_final = list()

**if** (**not** self.modelo\_final\_generado) **or** (**not** self.modelo\_referencial\_generado):

print("Debe generar ambos modelos para obtener resultados comparativos")

**return** resultado\_referencial, resultado\_final

parametros = self.devolver\_mejor\_parametro()

resultados = self.devolver\_cv\_results()

i\_seleccionado = 0

i\_iter = 0

**for** param **in** resultados["params"]:

**if** parametros == param:

i\_seleccionado = i\_iter

i\_iter += 1

**for** i **in** range(self.obtener\_num\_clases()):

resultado\_referencial.append(self.instanciar\_modelo\_referencial(i+1).devolver\_resultado()["accuracy"])

resultado\_final.append(resultados["split" + str(i) + "\_test\_accuracy"][i\_seleccionado])

**return** resultado\_referencial, resultado\_final

**def** mostrar\_resultados\_referencial\_vs\_final(self):

**if** (**not** self.modelo\_final\_generado) **or** (**not** self.modelo\_referencial\_generado):

print("Debe generar ambos modelos para obtener resultados comparativos")

**return**

resultado\_referencial, resultado\_final = self.obtener\_resultados\_referencial\_vs\_final()

**for** i **in** range(self.obtener\_num\_clases()):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\* CLASE " + str(i+1) + " \*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

acc\_mr = resultado\_referencial[i]

acc\_mf = resultado\_final[i]

print("Accuracy modelo referencial: " + '**{:.1%}**'.format(acc\_mr))

print("Accuracy modelo final: " + '**{:.1%}**'.format(acc\_mf))

print("")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\*\*\* MODELO FINAL \*\*\*")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Accuracy modelo final: " + '**{:.1%}**'.format(self.devolver\_resultado()["accuracy"]))

print("")

**def** generar\_predictor\_final(self):

predictor = self.devolver\_mejor\_modelo()

llave\_fold\_final = self.obtener\_llaves\_clases()[0]

nuevo\_generador\_features = GeneradorFeaturesParaPredicciones(carpeta\_base=self.carpeta\_base, diamond\_db=self.diamond\_db, carpeta\_cpat=self.carpeta\_fold\_cpat(llave\_fold\_final))

predictor.steps.pop(0)

predictor.steps.insert(0,['features', nuevo\_generador\_features])

dump(predictor, self.carpeta\_modelo() + "/modelo\_final.plk")

**def** reportar\_predicciones(self, archivo\_lncRNA, archivo\_PCT):

y\_pred\_lncRNA = self.realizar\_predicciones(archivo\_lncRNA)

probs\_lncRNA = self.realizar\_predicciones\_proba(archivo\_lncRNA, features\_calculados=**True**)

y\_pred\_PCT = self.realizar\_predicciones(archivo\_PCT)

probs\_PCT = self.realizar\_predicciones\_proba(archivo\_PCT, features\_calculados=**True**)

y\_true = ([1] \* len(y\_pred\_lncRNA)) + ([0] \* len(y\_pred\_PCT))

y\_pred = np.concatenate((y\_pred\_lncRNA, y\_pred\_PCT))

plt.figure(1)

probs = np.concatenate((probs\_lncRNA[:,1], probs\_PCT[:,1]))

precision, recall, \_ = precision\_recall\_curve(y\_true, probs)

average\_precision = average\_precision\_score(y\_true, probs)

plt.step(recall, precision, color='b', alpha=0.2, where='post')

plt.fill\_between(recall, precision, step='post', alpha=0.2, color='b')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('2-class Precision-Recall curve: AP=**{0:0.2f}**'.format(average\_precision))

plt.figure(2)

fpr, tpr, \_ = roc\_curve(y\_true, probs)

roc\_auc = roc\_auc\_score(y\_true, probs)

plt.plot(fpr, tpr, label='ROC curve (area = **%0.3f**)' % roc\_auc)

plt.plot([0, 1], [0, 1], linestyle='--') *# random predictions curve*

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('False Positive Rate or (1 - Specifity)')

plt.ylabel('True Positive Rate or (Sensitivity)')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

**return** classification\_report(y\_true, y\_pred, target\_names=["PCT", "lncRNA"]), confusion\_matrix(y\_true, y\_pred), precision\_recall\_fscore\_support(y\_true, y\_pred, average='binary')

**def** reportar\_predicciones\_keras(self, archivo\_lncRNA, archivo\_PCT):

y\_pred\_lncRNA = self.realizar\_predicciones(archivo\_lncRNA)

probs\_lncRNA = self.realizar\_predicciones\_proba\_keras(archivo\_lncRNA, features\_calculados=**True**)

y\_pred\_PCT = self.realizar\_predicciones(archivo\_PCT)

probs\_PCT = self.realizar\_predicciones\_proba\_keras(archivo\_PCT, features\_calculados=**True**)

y\_true = ([1] \* len(y\_pred\_lncRNA)) + ([0] \* len(y\_pred\_PCT))

y\_pred = np.concatenate((y\_pred\_lncRNA, y\_pred\_PCT))

plt.figure(1)

probs = np.concatenate((probs\_lncRNA[:,1], probs\_PCT[:,1]))

precision, recall, \_ = precision\_recall\_curve(y\_true, probs)

average\_precision = average\_precision\_score(y\_true, probs)

plt.step(recall, precision, color='b', alpha=0.2, where='post')

plt.fill\_between(recall, precision, step='post', alpha=0.2, color='b')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.ylim([0.0, 1.05])

plt.xlim([0.0, 1.0])

plt.title('2-class Precision-Recall curve: AP=**{0:0.2f}**'.format(average\_precision))

plt.figure(2)

fpr, tpr, \_ = roc\_curve(y\_true, probs)

roc\_auc = roc\_auc\_score(y\_true, probs)

plt.plot(fpr, tpr, label='ROC curve (area = **%0.3f**)' % roc\_auc)

plt.plot([0, 1], [0, 1], linestyle='--') *# random predictions curve*

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.xlabel('False Positive Rate or (1 - Specifity)')

plt.ylabel('True Positive Rate or (Sensitivity)')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

**return** classification\_report(y\_true, y\_pred, target\_names=["PCT", "lncRNA"]), confusion\_matrix(y\_true, y\_pred), precision\_recall\_fscore\_support(y\_true, y\_pred, average='binary')

**def** realizar\_predicciones(self, archivo\_fasta, features\_calculados=**False**):

predictor = load(self.carpeta\_modelo() + "/modelo\_final.plk")

X\_test = util\_fasta.leer\_fasta\_list(archivo\_fasta)

predictor.set\_params(features\_\_features\_calculados=features\_calculados)

**return** predictor.predict(X\_test)

**def** realizar\_predicciones\_proba(self, archivo\_fasta, features\_calculados=**False**):

predictor = load(self.carpeta\_modelo() + "/modelo\_final.plk")

X\_test = util\_fasta.leer\_fasta\_list(archivo\_fasta)

predictor.set\_params(features\_\_features\_calculados=features\_calculados)

**return** predictor.decision\_function(X\_test)

**def** realizar\_predicciones\_proba\_keras(self, archivo\_fasta, features\_calculados=**False**):

predictor = load(self.carpeta\_modelo() + "/modelo\_final.plk")

X\_test = util\_fasta.leer\_fasta\_list(archivo\_fasta)

predictor.set\_params(features\_\_features\_calculados=features\_calculados)

**return** predictor.predict\_proba(X\_test)

*#wrappers para ejecución en paralelo*

**def** wrapper\_armar\_fold\_final(tesis2, tipo):

tesis2.armar\_fold\_final(tipo)

**def** wrapper\_armar\_fold\_clase(tesis2, num\_clase):

tesis2.armar\_fold\_clase(num\_clase)

**def** wrapper\_generar\_cpat\_fold(tesis2, num\_clase):

tesis2.generar\_cpat\_fold(num\_clase)

**def** wrapper\_limpieza\_archivos\_CDS(tesis2, num\_clase):

tesis2.limpieza\_archivos\_CDS(num\_clase)

**def** wrapper\_ejecutar\_cpat\_fold(tesis2, num\_clase):

tesis2.ejecutar\_cpat\_fold(num\_clase)

**def** wrapper\_generar\_features\_fold(tesis2, num\_clase):

tesis2.generar\_features\_fold(num\_clase)

**def** wrapper\_limpieza\_archivos\_finales\_fasta(tesis2, num\_clase):

tesis2.limpieza\_archivos\_finales\_fasta(num\_clase)

**def** wrapper\_crear\_modelo\_referencial(tesis2, num\_clase):

tesis2.crear\_modelo\_referencial(num\_clase)

**def** crear\_modelo\_keras(optimizer='adam', learn\_rate=0.01, momentum=0, init\_mode='uniform', activation='relu', activation2='softmax', activation\_final='sigmoid', dropout\_rate=0.0, weight\_constraint=1, neurons=10, hidden\_layers=1, hidden\_neurons=10):

*#optimizer = SGD(lr=learn\_rate, momentum=momentum)*

model = Sequential()

model.add(Dense(units=neurons, activation=activation, input\_dim=10, kernel\_initializer=init\_mode, kernel\_constraint=maxnorm(weight\_constraint)))

**for** \_ **in** range(hidden\_layers):

model.add(Dropout(dropout\_rate))

model.add(Dense(units=hidden\_neurons, activation=activation2, kernel\_initializer=init\_mode, kernel\_constraint=maxnorm(weight\_constraint)))

model.add(Dropout(dropout\_rate))

**if** activation\_final == "softmax":

model.add(Dense(2, activation=activation\_final, kernel\_initializer=init\_mode))

model.compile(loss='sparse\_categorical\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

**else**:

model.add(Dense(activation=activation\_final, kernel\_initializer=init\_mode))

model.compile(loss='binary\_crossentropy', optimizer=optimizer, metrics=['accuracy'])

**return** model

**class** **GeneradorFeatures**(BaseEstimator, TransformerMixin):

**def** \_\_init\_\_(self, tesis2=**None**, cantidad\_transcritos=**None**, num\_clases=**None**):

**if** cantidad\_transcritos **is** **None**:

**return**

self.cantidad\_transcritos = cantidad\_transcritos

self.num\_clases = num\_clases

self.tesis2 = tesis2

**def** fit(self, X, y=**None**):

self.\_llave\_fold = self.obtener\_llave\_fold(X)

**return** self

**def** transform(self, X):

**return** self.obtener\_features\_pre\_calculados(X)

**def** obtener\_llave\_fold(self, X):

cod\_secuencias = ""

num\_transcritos = len(X)

num\_transcritos\_por\_grupo = self.cantidad\_transcritos

**for** i **in** range(num\_transcritos//(num\_transcritos\_por\_grupo\*2)):

cod\_secuencias += X[i \* num\_transcritos\_por\_grupo][0]

llave = hashlib.sha224(cod\_secuencias.encode()).hexdigest()

**return** llave

**def** obtener\_features\_pre\_calculados(self, X):

llave = self.\_llave\_fold

tipo = "train"

**if** os.path.isfile(self.tesis2.archivo\_fold\_clase(llave, "test", "lncRNA")):

secuencias = util\_fasta.leer\_fasta(self.tesis2.archivo\_fold\_clase(llave, "test", "lncRNA"), 1)

**if** list(secuencias.keys())[0] == X[0][0]:

tipo = "test"

features = list(load(self.tesis2.archivo\_features\_clase(llave, tipo, "lncRNA")).values())

features += list(load(self.tesis2.archivo\_features\_clase(llave, tipo, "PCT")).values())

**return** [list(x.values()) **for** x **in** features]

**class** **GeneradorFeaturesKeras**(BaseEstimator, TransformerMixin):

**def** \_\_init\_\_(self, tesis2=**None**, cantidad\_transcritos=**None**, num\_clases=**None**):

**if** cantidad\_transcritos **is** **None**:

**return**

self.cantidad\_transcritos = cantidad\_transcritos

self.num\_clases = num\_clases

self.tesis2 = tesis2

**def** fit(self, X, y=**None**):

self.\_llave\_fold = self.obtener\_llave\_fold(X)

**return** self

**def** transform(self, X):

**return** self.obtener\_features\_pre\_calculados(X)

**def** obtener\_llave\_fold(self, X):

cod\_secuencias = ""

num\_transcritos = len(X)

num\_transcritos\_por\_grupo = self.cantidad\_transcritos

**for** i **in** range(num\_transcritos//(num\_transcritos\_por\_grupo\*2)):

cod\_secuencias += X[i][0]

llave = hashlib.sha224(cod\_secuencias.encode()).hexdigest()

**return** llave

**def** obtener\_features\_pre\_calculados(self, X):

llave = self.\_llave\_fold

tipo = "train"

**if** os.path.isfile(self.tesis2.archivo\_fold\_clase(llave, "test", "lncRNA")):

secuencias = util\_fasta.leer\_fasta(self.tesis2.archivo\_fold\_clase(llave, "test", "lncRNA"), 1)

**if** list(secuencias.keys())[0] == X[0][0]:

tipo = "test"

features = {\*\*load(self.tesis2.archivo\_features\_clase(llave, tipo, "lncRNA")), \*\*load(self.tesis2.archivo\_features\_clase(llave, tipo, "PCT"))}

**return** [list(features[x[0]].values()) **for** x **in** X]

**class** **GeneradorFeaturesParaPredicciones**(BaseEstimator, TransformerMixin):

**def** \_\_init\_\_(self, carpeta\_base=**None**, diamond\_db=**None**, carpeta\_cpat=**None**, features\_calculados=**False**):

**if** carpeta\_base **is** **None**:

**return**

self.carpeta\_base = carpeta\_base

self.diamond\_db = diamond\_db

self.carpeta\_cpat = carpeta\_cpat

self.features\_calculados = features\_calculados

**def** fit(self, X, y=**None**):

**raise** **Exception**('Este modelo no admite fit')

**return** self

**def** transform(self, X):

carpeta\_transform = self.carpeta\_base + "/transform"

**if** **not** os.path.isdir(carpeta\_transform):

os.mkdir(carpeta\_transform)

archivo\_fasta = carpeta\_transform + "/secuencias.fa"

**if** (**not** self.features\_calculados):

self.generar\_archivos\_fasta(archivo\_fasta, X)

self.ejecutar\_diamond\_cpat(archivo\_fasta)

self.generar\_features(archivo\_fasta)

**return** self.obtener\_features\_pre\_calculados(archivo\_fasta)

**def** generar\_archivos\_fasta(self, archivo\_fasta, X):

util\_fasta.generar\_fasta(X, archivo\_fasta)

**def** ejecutar\_diamond\_cpat(self, archivo\_fasta):

diamond\_db = self.diamond\_db

carpeta\_cpat = self.carpeta\_cpat

util\_caracteristicas.ejecutar\_diamond(archivo\_fasta, diamond\_db, archivo\_fasta.replace(".fa", ".dmnd"))

util\_caracteristicas.ejecutar\_cpat(archivo\_fasta, carpeta\_cpat, archivo\_fasta.replace(".fa", ".cpat"))

os.remove(archivo\_fasta.replace(".fa", ".cpat") + ".dat")

os.remove(archivo\_fasta.replace(".fa", ".cpat") + ".r")

**def** generar\_features(self, archivo\_fasta):

util\_caracteristicas.generar\_features\_base(archivo\_fasta, archivo\_fasta.replace(".fa", ".cpat"), archivo\_fasta.replace(".fa", ".dmnd"), archivo\_fasta.replace(".fa", ".ft"))

**def** obtener\_features\_pre\_calculados(self, archivo\_fasta):

features = list(load(archivo\_fasta.replace(".fa", ".ft")).values())

**return** [list(x.values()) **for** x **in** features]

## : Generación de modelos referenciales y finales

*#Generación de archivos fasta para las 30 especies*

**import** **sys**

sys.path.append('./libs')

**import** **util\_bd**, **util\_fasta, util\_modelo\_final**

**import** **os**

**import** **shutil**

**from** **sklearn.externals.joblib** **import** Parallel, delayed, dump

*#tuned\_parameters = [{'keras\_\_batch\_size': [2\*\*10], 'keras\_\_epochs': [2\*\*7], 'keras\_\_optimizer': ['Adam'], 'keras\_\_learn\_rate': [0.01], 'keras\_\_momentum': [0], 'keras\_\_init\_mode': ['uniform'], 'keras\_\_activation': ['relu'], 'keras\_\_activation2': ['softmax'], 'keras\_\_dropout\_rate': [0.0], 'keras\_\_weight\_constraint': [1], 'keras\_\_neurons': [10, 100]}]*

tesis2 = util\_modelo\_final.Tesis2(carpeta\_base="real",verbose=100,tuned\_parameters=tuned\_parameters)

cantidad\_especies = 30

cantidad\_transcritos = 4208

**def** obtener\_especies():

query = "SELECT \* FROM (SELECT CONVERT(@row\_number:=@row\_number + 1, UNSIGNED) AS orden, m.id\_especie, m.especie FROM especies\_seleccionadas s JOIN maestra\_especies m ON s.especie = m.especie, (SELECT @row\_number:=0) AS rn ORDER BY m.id\_especie) a ORDER BY 1 LIMIT " + str(cantidad\_especies)

**return** util\_bd.resultados\_query(query)

**def** generar\_fasta\_especie(tesis2, row\_especie):

os.mkdir(tesis2.carpeta\_data() + "/clase\_" + str(row\_especie[0]))

*# lncRNA*

query = "SELECT cod\_secuencia, secuencia FROM secuencias WHERE flg\_pct = 0 AND flg\_seleccionado = 1 AND id\_especie = " + str(row\_especie[1]) + " ORDER BY cod\_secuencia LIMIT " + str(cantidad\_transcritos)

secuencias = util\_bd.resultados\_query(query)

util\_fasta.generar\_fasta(secuencias, tesis2.carpeta\_data() + "/clase\_" + str(row\_especie[0]) + "/lncRNA.fa")

*# PCT*

query = "SELECT cod\_secuencia, secuencia FROM secuencias WHERE flg\_pct = 1 AND flg\_seleccionado = 1 AND id\_especie = " + str(row\_especie[1]) + " ORDER BY cod\_secuencia LIMIT " + str(cantidad\_transcritos)

secuencias = util\_bd.resultados\_query(query)

util\_fasta.generar\_fasta(secuencias, tesis2.carpeta\_data() + "/clase\_" + str(row\_especie[0]) + "/PCT.fa")

*# CDS*

query = "SELECT f.cod\_secuencia, cds.coding FROM secuencias\_CDS cds JOIN secuencias\_features f ON cds.id\_especie = f.id\_especie AND cds.cod\_secuencia = f.cod\_secuencia WHERE f.flg\_pct = 1 AND f.flg\_seleccionado = 1 AND f.id\_especie = " + str(row\_especie[1]) + " ORDER BY f.cod\_secuencia LIMIT " + str(cantidad\_transcritos)

secuencias = util\_bd.resultados\_query(query)

util\_fasta.generar\_fasta(secuencias, tesis2.carpeta\_data() + "/clase\_" + str(row\_especie[0]) + "/CDS.fa")

**if** os.path.isdir(tesis2.carpeta\_data()):

shutil.rmtree(tesis2.carpeta\_data())

os.mkdir(tesis2.carpeta\_data())

%time dump(obtener\_especies(), tesis2.carpeta\_data() + "/info\_clases.bin")

%time Parallel(n\_jobs=tesis2.n\_jobs, verbose=tesis2.verbose)(delayed(generar\_fasta\_especie)(tesis2, row\_especie) **for** row\_especie **in** obtener\_especies())

*#GENERACION DEL MODELO FINAL*

%time tesis2.generar\_modelo\_final\_keras()

*#GENERACION DE LOS MODELOS REFERENCIALES*

%time tesis2.generar\_modelos\_referenciales()

*#GUARDAR EL MODELO FINAL EN UN ARCHIVO modelo\_final.plk*

%time tesis2.generar\_predictor\_final()

print("Proceso finalizado")

## : Predicciones con el predictor para plantas