Predictor de mutaciones patológicas para una familia de proteínas.

Estudiante:
Matías Salinero Delgado
Máster universitario en Bioinformática y Bioestadística

Consultor:
Pau Andrio Balado

6 / 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Predictor de mutaciones patológicas para una familia de proteínas.*
Nombre del autor:	*Matías Salinero Delgado*
Nombre del consultor/a:	*Pau Andrio Balado*
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	*Máster universitario en Bioinformática y Bioestadística*
Idioma del trabajo:	*Castellano*
Palabras clave	*Protein, Python, Machine Learning, SwissVar, Alignment, Mutation Detection, Kinase, Glycerol kinase, Scikit-Learn.*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El desarrollo de las técnicas biológicas actuales, como secuenciación y alineamiento, ha provocado que tengamos a nuestra disposición una cantidad ingente de datos. Esto ha puesto de relieve la necesidad de organizarlos y realizar un estudio exhaustivo y automático que nos permita entender mejor los datos obtenidos y, con ello, descubrir las relaciones hasta ahora ocultas entre ellos.

La aparición de las técnicas de Machine Learning abre la posibilidad de abordar este camino para, por ejemplo, detectar mutaciones patológicas y proporcionar información biomédica que posibilite el desarrollo de terapias.

El objetivo del TFM es desarrollar una aplicación que explore las bondades y limitaciones de este tipo de algoritmos, exponiendo gráfica y documentalmente los resultados alcanzados mediante una ejecución completa del ciclo del dato, desde su importación y tratamiento hasta el resultado final dejando constancia del rendimiento del proceso.

Abstract (in English, 250 words or less):

The development of the current biological techniques, such as sequencing and alignment, has provided us with a huge amount of data. This has highlighted the need to organize them and carry out an exhaustive and automatic study. A study that would allow us to better understand the obtained data and, as a result, discover the relationships that were hidden among them.

The emergence of Machine Learning techniques opens up new possibilities like the detection of pathological mutations, as well as provide biomedical information that enables the development of therapies.

The objective of this Master's Thesis is to develop an application to explore the benefits and limitations of this type of algorithms. It will display graphically and in a documented way the results achieved through a complete execution of the data cycle that will go from its import to the final result, tracking the performance throughout the process.

Índice

1 Introducción

1.1 Contexto y justificación del trabajo.

La gran cantidad de datos biológicos que se producen actualmente y su variedad requiere de nuevas técnicas para obtener resultados. A medida que la dimensionalidad de los datos aumenta, también crece la dificultad para su estudio.

Es aquí donde las técnicas de Machine Learning pueden ofrecer una solución. El desarrollo de estas metodologías durante las últimas décadas puede abrir un camino en la explotación de la cantidad ingente de datos que tenemos a nuestra disposición.

El aumento de la potencia de cómputo también ha permitido que los tiempos de ejecución hayan disminuido de tal forma que se pueden efectuar pequeñas pruebas de manera razonable incluso en un ordenador personal.

Con la finalidad de estudiar lo recién planteado, este TFM se va a centrar en la predicción de la condición patológica, o no, tomando como objeto de estudio el subconjunto de una familia de proteínas, las quinasas, para así explorar las bondades y limitaciones de este tipo de herramientas.

El hacer objetivo de este estudio a una familia de proteínas como son las quinasas tiene su justificación en la cantidad de procesos biológicos en los que están involucradas, teniendo también por ello una alta incidencia en gran cantidad de enfermedades y síndromes cuando sufren una mutación.

1.2 Motivación

Las mutaciones en las cadenas de aminoácidos de las proteínas son causa de una gran cantidad de enfermedades, por lo que es de interés la detección de las proteínas que sufren estas mutaciones.

La creciente cantidad de datos disponible y la potencia de computo actual ponen al alcance la capacidad de efectuar análisis que posibiliten la detección con cierta precisión de las proteínas mutadas patológicamente que son desencadenantes de estas enfermedades. De esta forma, se puede proporcionar información de interés biomédico para el tratamiento de estas patologías.

De entre todas las familias de proteínas, se ha elegido un subconjunto de las proteínas quinasas. Se ha seleccionado esta familia por dos razones:

1. El papel principal de estas proteínas en multitud de funciones celulares y, por tanto, en enfermedades.

2. El artículo de (Seltzer WK, 1989 Apr) que indica una relación entre la GKD y el síndrome de Duchenne. El déficit aislado de glicerol quinasa (GKD) es un trastorno del metabolismo del glicerol muy poco frecuente ligado al cromosoma X, caracterizado bioquímicamente por niveles de glicerol elevados en plasma y orina, y clínicamente por manifestaciones neurometabólicas variables, dependiendo de la edad de aparición, y que varían desde crisis metabólicas que amenazan la vida en la infancia a formas adultas asintomáticas (GKD infantil, GKD juvenil y GKD del adulto. La distrofia muscular de Duchenne (DMD) o distrofia muscular progresiva es una enfermedad hereditaria con un patrón de herencia de tipo recesivo ligado al cromosoma X que produce una deficiencia muscular progresiva y rápida que conduce a la discapacidad física y a una muerte prematura debido a complicaciones respiratorias y cardíacas. Es la enfermedad neuromuscular más frecuente y severa de la infancia.

El estudio y aplicación de las técnicas de Machine Learning en el descubrimiento de nuevas relaciones entre los datos es otra de las razones realizar este proyecto, ya que está posibilitando avances sustanciales en el campo de la bioinformática sobre la explotación de la inmensa cantidad de datos que se generan actualmente. Según (Baldi, 2001), el Machine Learning va a ser capital en los próximos años en el desarrollo e investigación de la Bioinformática.

1.3 Objetivos del trabajo

1.3.1 Objetivos principales

Los objetivos principales de este TFM son:

1. Desarrollo de un predictor de mutaciones patológicas en Python, cumpliendo el ciclo de dato desde su ingesta, tratamiento hasta la presentación de resultados.

2. Estudio e informe de las diferentes técnicas posibles en cada etapa.

3. Estudiar la viabilidad del uso de técnicas de Machine Learning en la predicción de mutaciones proteicas como el GKD.

1.3.2 Objetivos específicos

Los objetivos específicos dentro de cada objetivo principal han sido:

1.1 Familiarizarse con el entorno de desarrollo Python.

1.2 Familiarizarse con las diferentes librerías disponibles para el tratamiento del dato (Biopython, Pandas), Machine Learning (Scikit-Learn) y GUI (Tkinter).

2.1 Presentar de manera gráfica los resultados

2.2 Presentar de manera crítica los resultados.

1.4 Enfoque y método a seguir.

El proyecto se ha abordado diferenciando claramente cada una de las etapas por las que pasan los datos. De esta forma, se ha podido seguir secuencialmente el tratamiento del dato en cada una de sus etapas, considerando previamente las alternativas (si las hubiera), estudiándolas y realizando la implementación en Python.

Siguiendo esta metodología se han ido cumpliendo los dos objetivos principales, el desarrollo de la aplicación y el informe comparativo de las técnicas usadas.

Se ha empleado un repositorio remoto para el correcto desarrollo del código y su control de versiones, Github. (GitHub, s.f.)

https://github.com/msalinero/msd-tfm

1.5 Planificación del trabajo.

Para la planificación temporal del TFM se estimó que la dedicación diaria media sea de 2 horas, incluyendo fines de semana, para así poder tener flexibilidad a la hora de recuperar o anticipar horas de trabajo en caso necesario.

Para realizar el calendario de la planificación de desarrollo se realizó el siguiente desglose de tareas:

1. Desarrollo del predictor en Python

   a. Obtener predicciones mediante Machine Learning.

      1.a.1 Implementar la obtención de las secuencias de las proteínas (16 horas)

      1.a.2 Implementar el alineamiento de las secuencias (16 horas)

1.a.3 Implementar la extracción de features y entrenamiento del modelo (30 horas)

b. Presentar los resultados

1.b.1 Diseño e implementación de las gráficas y plots (24 horas)

2. Presentar y justificar las técnicas usadas

a. Identificar la mejor técnica en cada etapa.

2.a.1 Estudio e implementación de algoritmos de alineamiento (10 horas).

2.a.2 Estudio e implementación de algoritmos de Machine Learning. (10 horas)

b. Presentar una comparativa justificando esta elección

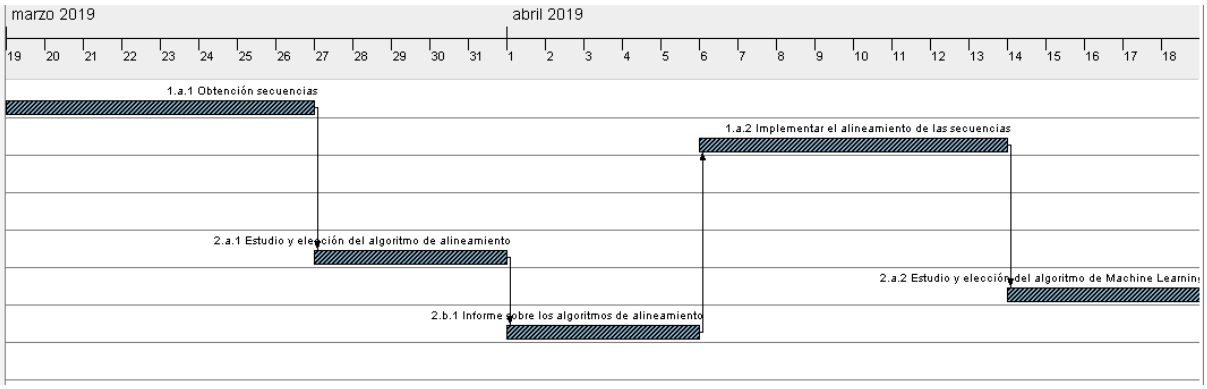
2.b.1 Informe sobre los algoritmos de alineamiento (8 horas).

2.b.2 Informe sobre los algoritmos de Machine Learning (8 horas).

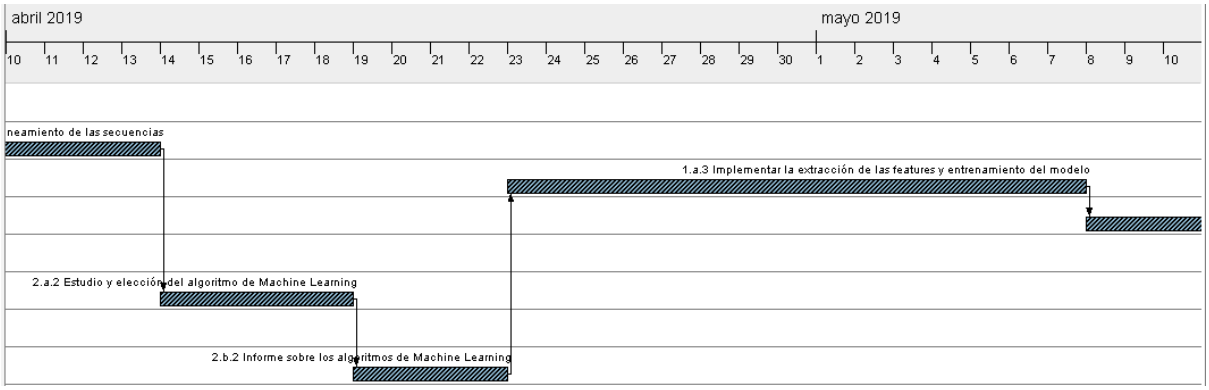
Se ha hecho uso de un diagrama de Gantt para representar la ocupación temporal en el calendario de cada una de las tareas:

Nombre	Fecha de inicio	Fecha de fin
1.a.1 Obtención secuencias	19/03/19	26/03/19
1.a.2 Implementar el alineamiento de las secuencias	6/04/19	13/04/19
1.a.3 Implementar la extracción de las features y entrenamiento del modelo	23/04/19	7/05/19
1.b.1 Diseño e implementación de gráficas y plots	8/05/19	19/05/19
2.a.1 Estudio y elección del algoritmo de alineamiento	27/03/19	31/03/19
2.a.2 Estudio y elección del algoritmo de Machine Learning	14/04/19	18/04/19
2.b.1 Informe sobre los algoritmos de alineamiento	1/04/19	5/04/19
2.b.2 Informe sobre los algoritmos de Machine Learning	19/04/19	22/04/19

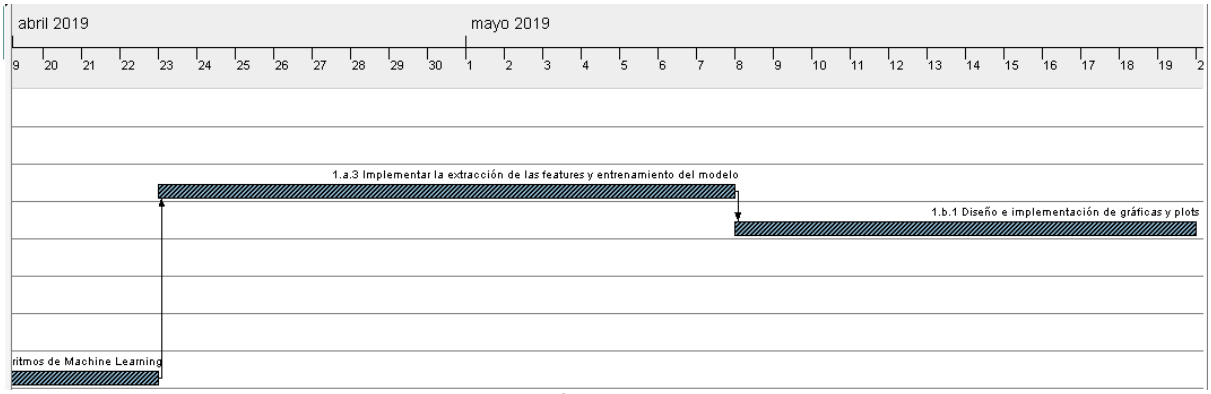
1 Desglose de tareas



2 Planificación temporal (1)



3 Planificacion temporal (2)



4 Planificación temporal (3)

1.6 Sumario de los productos obtenidos.

Como resultado de este TFM se va a elaborar:

- Memoria del TFM, documentando completamente el trabajo realizado.
- Aplicación. (https://github.com/msalinero/msd-tfm)
- Presentación del trabajo.

2 Las proteínas quinasas

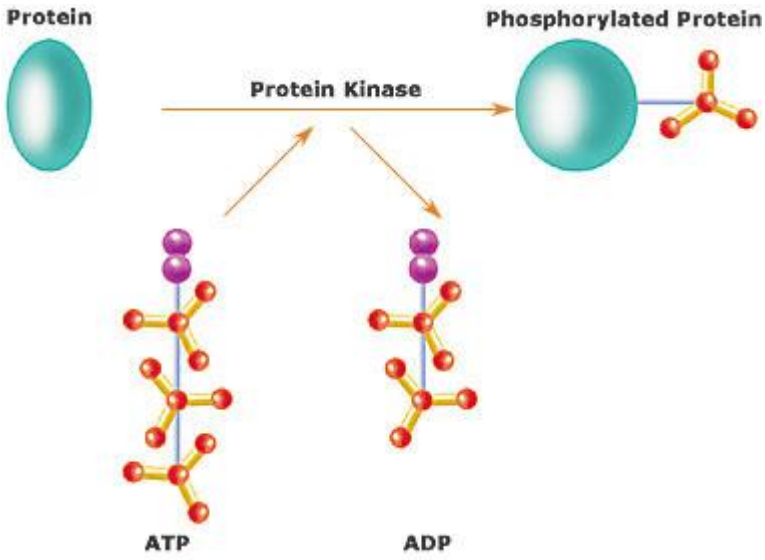
2.1 Introducción

Este apartado tiene el cometido de ser una breve descripción de las quinasas para poner en contexto los datos con los que se ha trabajado y la importancia de esta familia de proteínas.

Las enzimas son moléculas orgánicas cuya misión es la de ejercer como catalizadores de reacciones químicas, acelerando la velocidad de reacción. En su mayoría, son de naturaleza proteica.

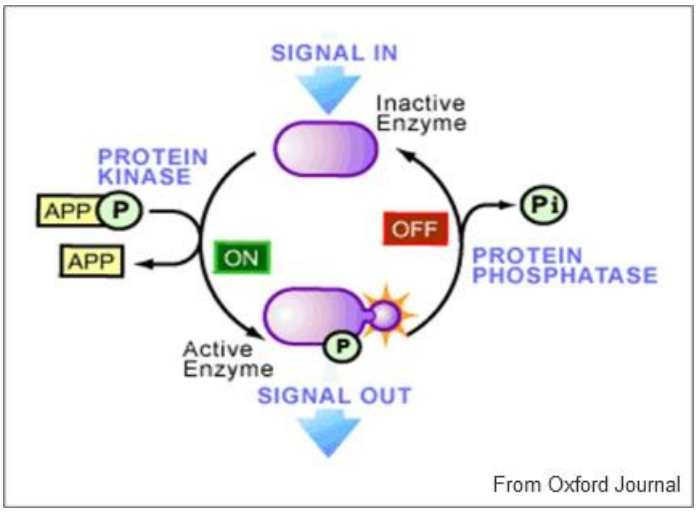
La familia de enzimas que ocupa este TFM es el de las proteínas quinasas. Éstas actúan sobre otras moléculas (sustratos), mayormente proteínas, mediante fosforilación.

La fosforilación consiste en la adición de un grupo fosfato a cualquier otra molécula. Estos grupos fosfatos son proporcionados desde ATP (adenosín trifosfato) a una diana del sustrato. Para que la transmisión del grupo fosfato sea posible, es necesaria la presencia de un ion metálico divalente como el Mg^{2+} o el Mn^{2+}.



5 Fosforilación.

A través de este proceso, las quinasas regulan la actividad biológica de las proteínas provocando cambios en su estado de actividad. Se puede decir que son un "interruptor" para la actividad biológica de las proteínas.



6 Actividad de las quinasas.

Por esto, las quinasas son miembros importantes de la señalización celular y regulan la mayoría de los procesos celulares, como son el metabolismo, transcripción, ciclo celular, apoptosis (muerte celular programada), etc. Alrededor de un 30% del proteoma celular es susceptible de la acción de una proteína quinasa.

2.2 Clasificación

Las proteínas quinasas se pueden clasificar de dos maneras:

1. Por sustratos: tirosina quinasas, serina/treonina quinasa e histidina quinasa.

2. Por dominios catalíticos: AGC quinasa, CAMK quinasa, CK1 quinasa, CMGC quinasa, STE quinasa, Tirosina quinasa y TKL quinasa.

2.3 Implicaciones

Las proteínas quinasas están implicadas en diferentes enfermedades humanas: cáncer, diabetes y enfermedades autoinmunes, entre otras. Por ello, la correcta regulación de las proteínas quinasas es capital para el control de estas enfermedades. La actividad irregular de estas proteínas puede ser patógena, especialmente en el caso del cáncer.

En consecuencia, esta familia de proteínas ha sido objetivo habitual de investigación para el desarrollo de fármacos. En el pasado, el descubrimiento de inhibidores de las proteínas quinasas ha supuesto un avance en la investigación clínica y en la de fármacos contra el cáncer.

3 Entorno de trabajo.

Para poder abarcar los siguientes apartados, es necesario introducir el contexto técnico en el cual se ha realizado este TFM, y así poder explicar cómo se han llevado a cabo las diferentes tareas.

Por ello se efectúa una breve descripción del contexto técnico y de las librerías usadas más relevantes.

3.1 Python.

Python es un lenguaje interpretado, dinámicamente tipado, multiparadigma (soporta programación orientada a objetos, programación imperativa, programación funcional) cuyo principal objetivo es el de conseguir un código legible.

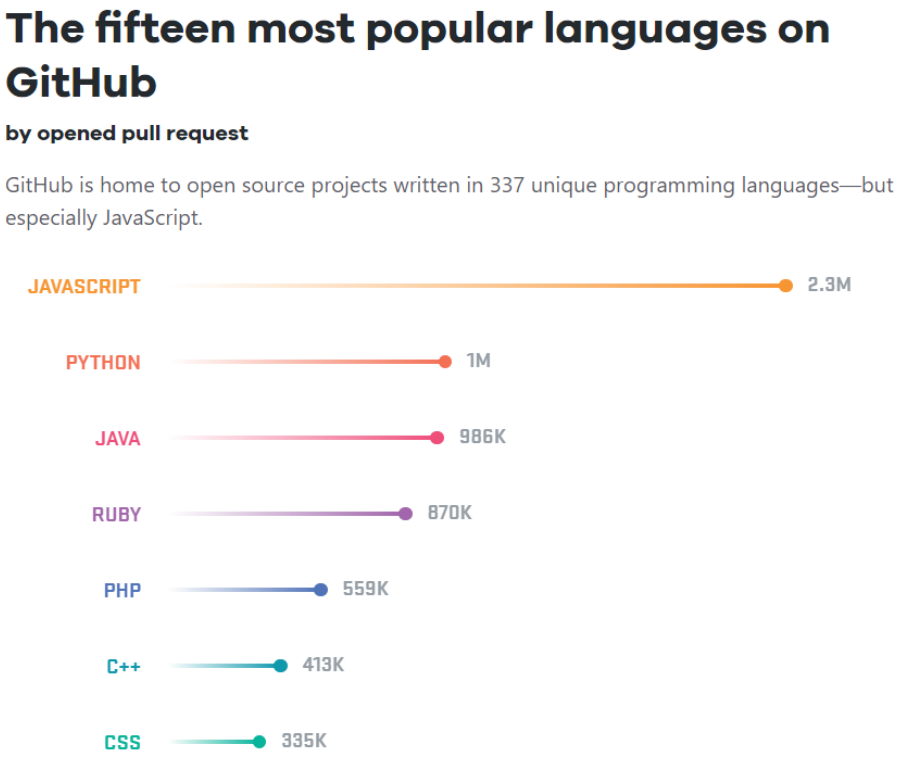
Fue creado a finales de los ochenta por Guido van Rossum.

Posee una licencia de código abierto llamada Python Software License, compatible con la Licencia pública GNU.

Debido a la mencionada legibilidad del código y al ser un lenguaje interpretado, es decir, cualquier máquina que tenga un intérprete puede ejecutar su código, lo que lo hace multiplataforma, Python ha ganado mucha aceptación en los últimos años, siendo uno de los lenguajes más emergentes y populares en la actualidad.

El auge de este lenguaje es debido a la gran comunidad que hay detrás, la cual ha propiciado la creación de una enorme cantidad de proyectos y librerías para el desarrollo de diferentes tareas, tales como Big Data, Data Science, Machine Learning y Bioinformática. Como prueba de ello se encuentran los proyectos Biopython y Scikit-learn, de los que se hace uso en este TFM.

Para este proyecto se ha utilizado la versión 3.7.0



7 Clasificación de lenguajes según pull requests.

3.2 Biopython

Biopython es un proyecto desarrollado en Python por un grupo internacional de desarrolladores que proporciona librerías y herramientas para implementar funcionalidades necesarias dentro del campo de la bioinformática.

Está sujeta a la Biopython License.

Este proyecto se ha realizado con la versión 1.73.

3.3 Scikit-learn

Scikit-learn es una librería para Machine Learning. Está considerada como la biblioteca de referencia para Machine Learning en Python.

Scikit-learn implementa una gran cantidad de algoritmos regresores, clasificadores y de clustering. La librería está preparada para interactuar con las populares librerías de Python NumPy y SciPy.

También implementa otras funcionalidades útiles para el Machine Learning, como funciones generadoras de métricas de rendimiento, datasets de prueba, generadores de muestras, etc.

La versión utilizada en este proyecto es la 0.21.2

3.4 Pandas

Pandas es una biblioteca de código abierto para Python, con licencia BSD, que proporciona estructuras y herramientas para el análisis de datos.

Pandas es una extensión de la librería NumPy y las estructuras de datos que proporciona son compatibles con otras librerías como Scikit-learn.

En este proyecto se ha utilizado la versión 0.24.2

4 Obtención de los datos

4.1 Introducción.

Esta primera parte del desarrollo tiene como objetivo explicar cómo se han obtenido los datos y cuál es su estructura.

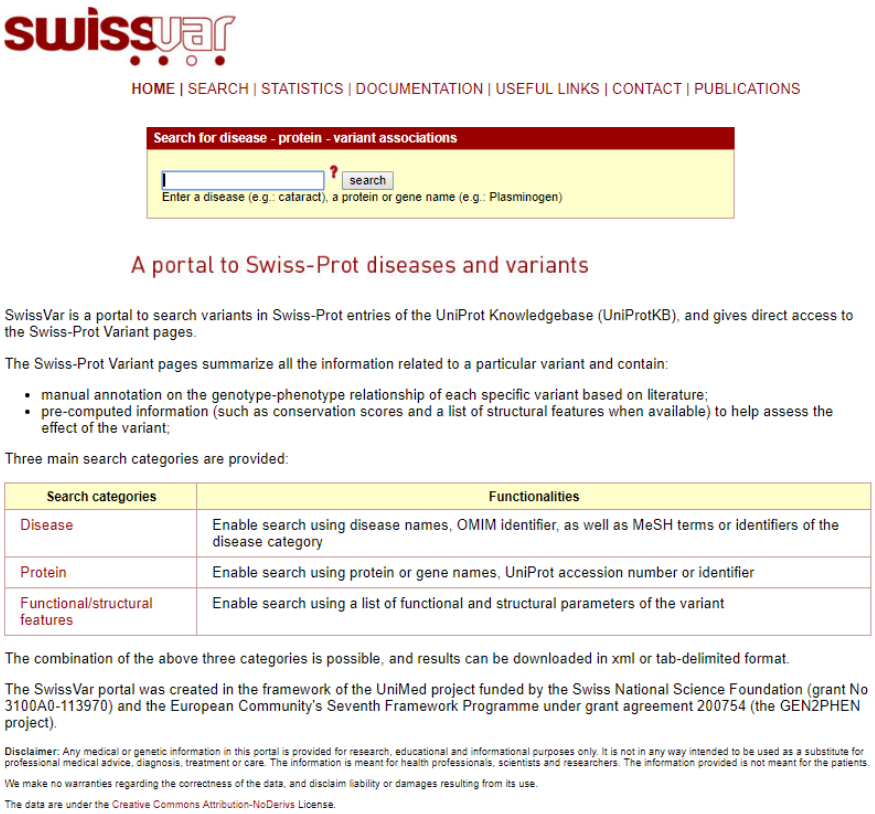
Este primer análisis permitirá comprender mejor cómo se pueden explotar e interpretar los datos sobre los que se trabaja, además de exponer las posibilidades y limitaciones de los datos disponibles.

4.2 Obtención de las secuencias

En este apartado se describe el modo en el que se han obtenido las secuencias proteicas de aminoácidos, formato y tratamiento.

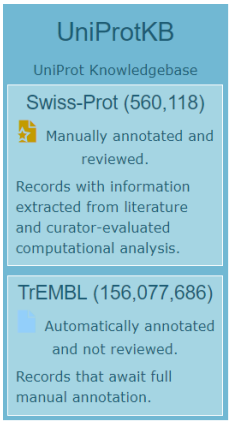
4.2.1 SwissVar

SwissVar (https://swissvar.expasy.org/) es un portal de búsqueda para localizar variantes y enfermedades en entradas Swiss-Prot de la base de datos UniProt Knowledgebase (https://www.uniprot.org/).



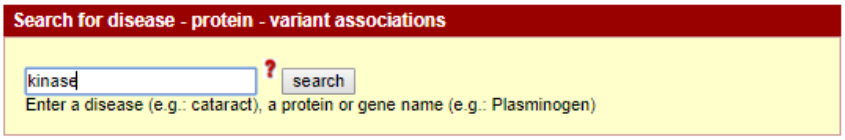
8 Página principal Swiss-Var

En la UniProt Knowledgebase hay dos conjuntos de entradas: Swiss-Prot y TrEMBL. UniProt Knowledgebase ha sido anotada manualmente y revisada. TrEMBL ha sido anotada automáticamente y no revisada manualmente. Por lo tanto, esta última dispone de muchas más entradas.



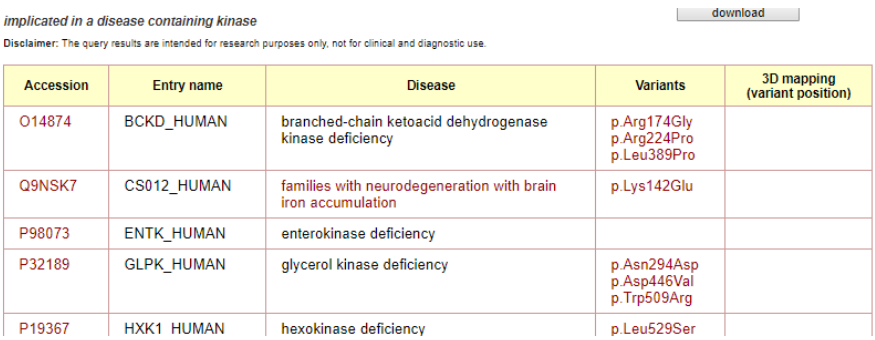
9 Swiss-Prot / TrEMBL

Para obtener las variantes patológicas asociadas a las quinasas que estén contenidas en Swiss-Prot se utiliza el buscador de SwissVar para que las muestre, ya que en él existe la posibilidad de buscar por proteínas, enfermedades y características funcionales y estructurales.



10 Buscador Swiss-Var.

El portal muestra el siguiente resultado de la búsqueda (solo se muestran los primeros resultados):



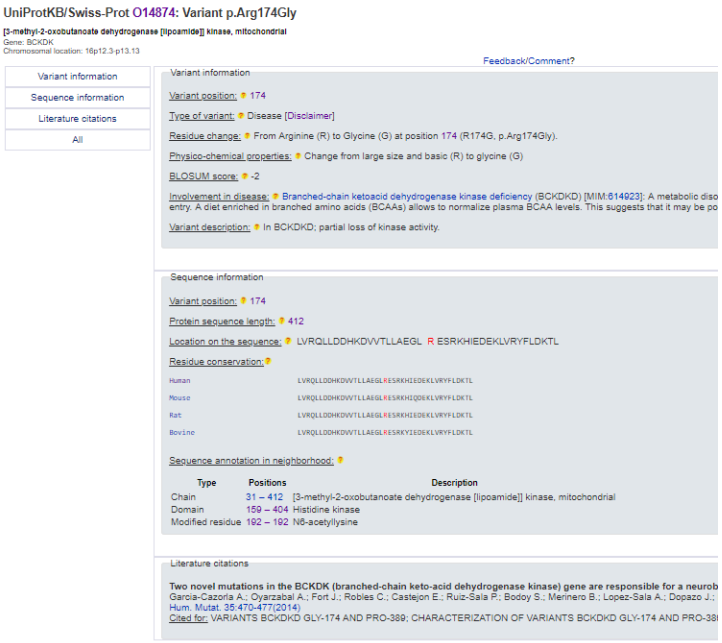
Accession	Entry name	Disease	Variants	3D mapping (variant position)
O14874	BCKD_HUMAN	branched-chain ketoacid dehydrogenase kinase deficiency	p.Arg174Gly p.Arg224Pro p.Leu389Pro	
Q9NSK7	CS012_HUMAN	families with neurodegeneration with brain iron accumulation	p.Lys142Glu	
P98073	ENTK_HUMAN	enterokinase deficiency		
P32189	GLPK_HUMAN	glycerol kinase deficiency	p.Asn294Asp p.Asp446Val p.Trp509Arg	
P19367	HXK1_HUMAN	hexokinase deficiency	p.Leu529Ser	

11. Resultados busqueda Swiss-Var

Las columnas muestran:

- *Accession:* Enlace a la entrada UniProt.
- *Entry Name:* Nombre de la entrada
- *Disease:* Enfermedad asociada.
- *Variants:* Las variantes patológicas asociadas a esta proteína y enfermedad. Siguen el formato p.*AminoácidoOriginalPosiciónAminoácidoFinal* dando información de la mutación efectuada.

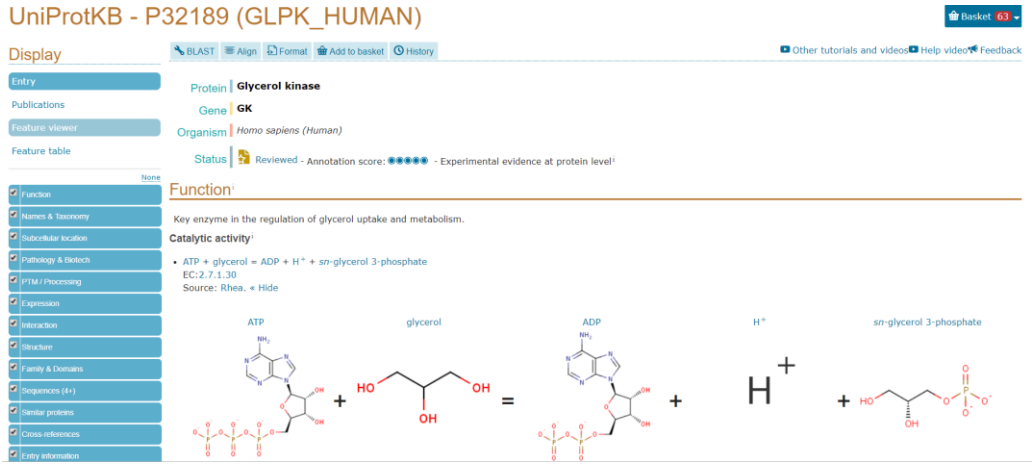
Si se pulsa sobre cualquiera de las opciones de *Variants,* se accede a la información de esa variante, con información sobre la secuencia, enfermedad asociada, referencias, etc.



12 Ejemplo de variante

4.2.2 UniProt

Para obtener las secuencias completas no patógenas, hay que acceder a la entrada en UniProt.



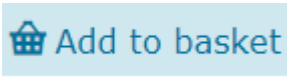
13 Entrada de UniProt.

Entre la información disponible interesan dos apartados principalmente *Pathology and Biotech,* que contiene información sobre las variantes patógenas, y *Sequences,* que proporciona las secuencias de aminoácidos que conforman la proteína.

El apartado *Secuences* contiene, además de la secuencia canónica fijada, las secuencias isoformas alternativas que pueden existir a causa del splicing alternativo. También se recogen isoformas generadas computacionalmente.

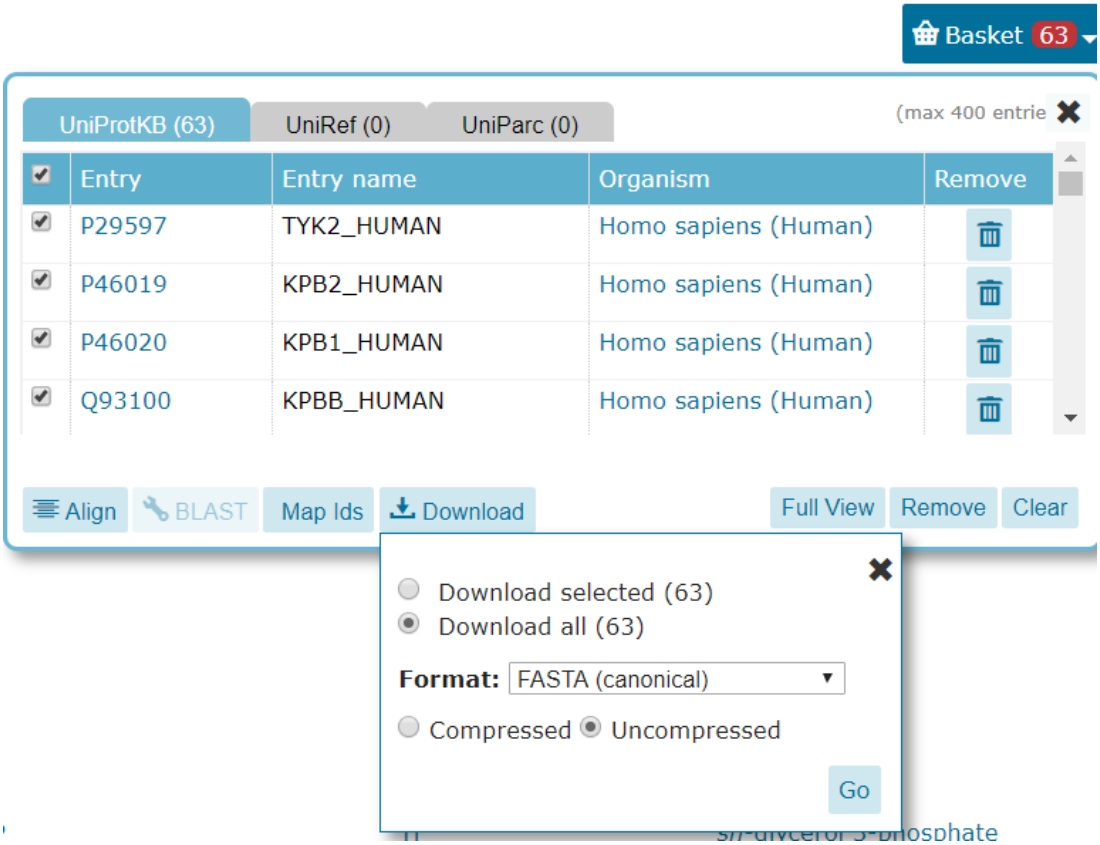
Una isoforma es una de las distintas configuraciones que puede adoptar una misma proteína, a partir de genes relacionados o por el mismo gen cuando se produce el splicing alternativo.

UniProt ofrece la posibilidad de obtener cómodamente las secuencias que se deseen en formato FASTA pulsando en



14 Basket UniProt

Una vez que se tengan las secuencias que se desean en la cesta, hay que dirigirse a la parte superior y descargarlas en el formato deseado



15 Descarga de secuencias desde UniProt

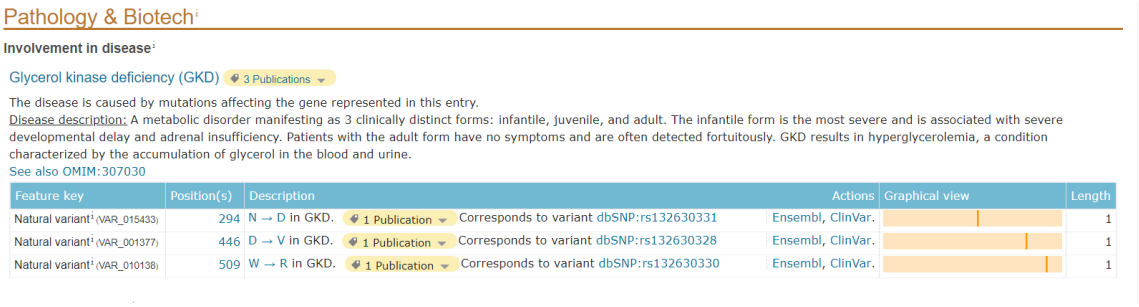
Las secuencias FASTA descargadas tienen el siguiente formato:

```
>sp|Q9NSK7|CS012_HUMAN Protein C19orf12 OS=Homo sapiens OX=9606
GN=C19orf12 PE=1 SV=3
MERLKSHKPATMTIMVEDIMKLLCSLSGERKMKAAVKHSGKGALVTGAMAFVGGLVGGPP
GLAVGGAVGGLLGAWMTSGQFKPVPQILMELPPAEQQRLFNEAAAIIRHLEWTDAVQLTA
LVMGSEALQQQLLAMLVNYVTKELRAEIQYDD
>sp|Q9NSK7-2|CS012_HUMAN Isoform 2 of Protein C19orf12 OS=Homo sapiens
OX=9606 GN=C19orf12
MTSGQFKPVPQILMELPPAEQQRLFNEAAAIIRHLEWTDAVQLTALVMGSEALQQQLLAM
LVNYVTKELRAEIQYDD
>sp|Q9NSK7-3|CS012_HUMAN Isoform 3 of Protein C19orf12 OS=Homo sapiens
OX=9606 GN=C19orf12
MTIMVEDIMKLLCSLSGERKMKAAVKHSGKGALVTGAMAFVGGLVGGPPGLAVGGAVGGL
LGAWMTSGQFKPVPQILMELPPAEQQRLFNEAAAIIRPCSSSCWPCW
>sp|Q9NSK7-4|CS012_HUMAN Isoform 1 of Protein C19orf12 OS=Homo sapiens
OX=9606 GN=C19orf12
MTIMVEDIMKLLCSLSGERKMKAAVKHSGKGALVTGAMAFVGGLVGGPPGLAVGGAVGGL
LGAWMTSGQFKPVPQILMELPPAEQQRLFNEAAAIIRHLEWTDAVQLTALVMGSEALQQQ
LLAMLVNYVTKELRAEIQYDD
```

Aquí se muestran 4 secuencias isoformas, cada una con la misma entrada, pero con el sufijo *-i,* en referencia a cada isoforma. Además, el formato FASTA que proporciona UniProt proporciona, entre más datos, esta información en la descripción.

4.2.3 Secuencias con las variantes patológicas.

En el apartado *Pathology and Biotech* se recogen las mutaciones agrupadas por cada enfermedad así como las mutaciones que las causan.



Pathology & Biotech

Involvement in disease

Glycerol kinase deficiency (GKD) 3 Publications

The disease is caused by mutations affecting the gene represented in this entry.
Disease description: A metabolic disorder manifesting as 3 clinically distinct forms: infantile, juvenile, and adult. The infantile form is the most severe and is associated with severe developmental delay and adrenal insufficiency. Patients with the adult form have no symptoms and are often detected fortuitously. GKD results in hyperglycerolemia, a condition characterized by the accumulation of glycerol in the blood and urine.
See also OMIM:307030

Feature key	Position(s)	Description	Actions	Graphical view	Length
Natural variant (VAR_015433)	294	N → D in GKD. 1 Publication Corresponds to variant dbSNP:rs132630331	Ensembl, ClinVar.		1
Natural variant (VAR_001377)	446	D → V in GKD. 1 Publication Corresponds to variant dbSNP:rs132630328	Ensembl, ClinVar.		1
Natural variant (VAR_010138)	509	W → R in GKD. 1 Publication Corresponds to variant dbSNP:rs132630330	Ensembl, ClinVar.		1

16 Pathology and Biotech

Los apartados recogen:

Feature Key: Identificador único de la variante.
Position: La posición de la cadena donde se produce la mutación.
Description: El SAP (*Single Amino-acid Polymorphism*) producido, enlace a la variante en Swiss-Prot y bibliografía asociada, así como un enlace a la base de datos de SNP (*Single Nucleotide Polymorphism*) de https://www.ncbi.nlm.nih.gov/.
Actions: Estudios realizados sobre esta mutación.
Graphical View: Representación gráfica de la posición de la mutación dentro de la cadena.
Lenght: Cantidad de aminoácidos a los que afecta la mutación.

Para generar las secuencias con las mutaciones patológicas se ha seguido el siguiente procedimiento:

1. Identificar la secuencia canónica, ya que las variaciones se refieren en posición y modificación a esta secuencia.
2. Introducir la modificación descrita.
3. Identificarla, sustituyendo su identificador con el identificador de la variante.

A modo de ejemplo, si se quisiera generar la secuencia conteniendo la variante VAR_015433 de la glicerol quinasa (P32189)

```
>sp|P32189|GLPK_HUMAN Glycerol kinase OS=Homo sapiens OX=9606 GN=GK
PE=1 SV=3
MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI
LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ
STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS
WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY
GLMKISHSVKAGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCNTGHKCV
```

```
FSDHGLLTTVAYKLGRDKPVYYALEGSVAIAGAVIRWLRDNLGIIKTSEEIEKLAKEVGT
SYGCYFVPAFSGLYAPYWEPSARGIICGLTQFTNKCHIAFAALEAVCFQTREILDAMNRD
CGIPLSHLQVDGGMTSNKILMQLQADILYIPVVKPSMPETTALGAAMAAGAAEGVGVWSL
EPEDLSAVTMERFEPQINAEESEIRYSTWKKAVMKSMGWVTTQSPESGDPSIFCSLPLGF
FIVSSMVMLIGARYISGIP
```

Se tendrían que hacer los cambios descritos y quedaría:

```
>sp|VAR_015433|GLPK_HUMAN Glycerol kinase GKD OS=Homo sapiens OX=9606
GN=GK PE=1 SV=3
MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI
LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ
STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS
WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY
GLMKISHSVKAGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCDTGHKCV
FSDHGLLTTVAYKLGRDKPVYYALEGSVAIAGAVIRWLRDNLGIIKTSEEIEKLAKEVGT
SYGCYFVPAFSGLYAPYWEPSARGIICGLTQFTNKCHIAFAALEAVCFQTREILDAMNRD
CGIPLSHLQVDGGMTSNKILMQLQADILYIPVVKPSMPETTALGAAMAAGAAEGVGVWSL
EPEDLSAVTMERFEPQINAEESEIRYSTWKKAVMKSMGWVTTQSPESGDPSIFCSLPLGF
FIVSSMVMLIGARYISGIP
```

De esta forma se han obtenido los dos conjuntos de secuencias para el estudio.

4.2.4 Ampliación del conjunto.

El conjunto de variaciones y secuencias isoformas para las quinasas recogidas por Swiss-Var resultaba en un número insuficiente y desigual de casos: 63 secuencias no patológicas y 144 variaciones. Esto afectaría a los resultados de la ejecución de los algoritmos de Machine Learning, pudiendo llegar a resultados erróneos y sin ser una muestra muy representativa.

Por ello se tomó la decisión de ampliar el campo, y aceptar secuencias TrEMBL y sus variantes, compensando y ampliándolas así: 123 secuencias no patológicas y 125 secuencias patológicas. Este número conforma una base manejable y asequible para las pruebas previstas.

4.3 Alineamiento de secuencias

Antes de abordar cómo se han generado los datos, es necesario explicar los procesos intermedios por los cuales se ha llegado a algunos de ellos. Uno de estos procesos es el alineamiento de secuencias.

El alineamiento de secuencias es un método por el cual se pueden representar dos o más secuencias de ADN, ARN o estructuras primarias de proteínas, y visualizar regiones comunes entre las secuencias. Estas regiones comunes pueden reflejar relaciones que pueden ser de tipo evolutivo, funcional o estructural.

La forma de representación del alineamiento es matricial, introduciendo *gaps* (huecos) para que queden alineados de manera columnar los residuos idénticos o similares.

```
P32189   GLPK_HUMAN          1   MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI        60
P32189-1 GLPK_HUMAN          1   MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI        60
P32189-2 GLPK_HUMAN          1   MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI        60
P32189-4 GLPK_HUMAN          1   MAASKKAVLGPLVGAVDQGTSSTRFLVFNSKTAELLSHHQVEIKQEFPREGWVEQDPKEI        60
                                 ************************************************************

P32189   GLPK_HUMAN         61   LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ       120
P32189-1 GLPK_HUMAN         61   LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ       120
P32189-2 GLPK_HUMAN         61   LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ       120
P32189-4 GLPK_HUMAN         61   LHSVYECIEKTCEKLGQLNIDISNIKAIGVSNQRETTVVWDKITGEPLYNAVVWLDLRTQ       120
                                 ************************************************************

P32189   GLPK_HUMAN        121   STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS       180
P32189-1 GLPK_HUMAN        121   STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS       180
P32189-2 GLPK_HUMAN        121   STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS       180
P32189-4 GLPK_HUMAN        121   STVESLSKRIPGNNNFVKSKTGLPLSTYFSAVKLRWLLDNVRKVQKAVEEKRALFGTIDS       180
                                 ************************************************************

P32189   GLPK_HUMAN        181   WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY       240
P32189-1 GLPK_HUMAN        181   WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY       240
P32189-2 GLPK_HUMAN        181   WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY       240
P32189-4 GLPK_HUMAN        181   WLIWSLTGGVNGGVHCTDVTNASRTMLFNIHSLEWDKQLCEFFGIPMEILPNVRSSSEIY       240
                                 ************************************************************

P32189   GLPK_HUMAN        241   GLMKISHSVKAGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCNTGHKCV       300
P32189-1 GLPK_HUMAN        241   GLMK------AGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCNTGHKCV       294
P32189-2 GLPK_HUMAN        241   GLMK------AGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCNTGHKCV       294
P32189-4 GLPK_HUMAN        241   GLMKISHSVKAGALEGVPISGCLGDQSAALVGQMCFQIGQAKNTYGTGCFLLCNTGHKCV       300
                                 ****     **************************************************
```

17 Alineamiento.

En el alineamiento de proteínas, la similitud en las posiciones de las secuencias denota cuan conservada está esa región o la existencia de motivos y dominios comunes entre ellas.

Se puede clasificar el alineamiento de secuencias en dos tipos: alineamiento local y global. El alineamiento local identifica regiones similares dentro de secuencias que en su total son diferentes. El alineamiento global fuerza a encontrar la similitud a lo largo de toda la longitud de las secuencias, intentando alinear todos los residuos en todas las todas ellas.

Para implementar los algoritmos de alineamiento se ha utilizado una variedad de aproximaciones, como la programación dinámica.

Cuando se quieren alinear más de dos secuencias, se recurre al alineamiento múltiple. Este alineamiento, además de dar la posibilidad de identificar las ya mencionadas propiedades de conservación, detección de motivos y dominios comunes, también permite la representación de árboles filogenéticos.

Los árboles filogenéticos representan las relaciones evolutivas entre las secuencias y son utilizados normalmente sobre alineamientos para ver las regiones conservadas de diferentes especies.

En este caso concreto, va a servir para ver las relaciones entre las secuencias objetivo y como se agrupan las diferentes secuencias con sus variantes patógenas.

Otro resultado del alineamiento múltiple de secuencias es la posibilidad de obtener una secuencia consenso, que es la secuencia que representa los aminoácidos que se encuentran con más frecuencia en cada posición una vez se ha producido el alineamiento.

4.3.1 ClustalW

ClustalW es una herramienta de línea de comandos muy popular usada para el alineamiento múltiple de secuencias. También cuenta con una variante con interfaz gráfica llamada ClustalX. Ha sido desarrollada por el University College Dublin, bajo la autoría de Desmond G. Higgins. (Des Higgins, s.f.)

Una nueva versión, Clustal Omega está siendo desarrollada para ofrecer mayor rendimiento.

ClustalW está implementado como un algoritmo iterativo, por lo que los errores cometidos en los primeros pasos es poco probable que sean corregidos.

En este proyecto se ha empleado la variante de línea de comandos ClustalW, ya que Biopython incluye funciones para invocar a esta herramienta desde Python una vez instalada.

4.3.2 Muscle

MUltiple Sequence Comparison by Log-Expectation (MUSCLE) es otra herramienta para el alineamiento múltiple de secuencias desarrollada originalmente por Robert C. Edgar. (Edgar R. , 2004)

MUSCLE implementa un algoritmo progresivo que permite la corrección de las columnas durante su ejecución, lo que mejora su precisión y velocidad.

Biopython también contiene funciones que integran la llamada a esta herramienta una vez instalada.

4.4 Generación de los datos

Una vez obtenidas las secuencias que van a conformar el punto de partida de los datos, el siguiente paso es generar las *features* sobre las que los algoritmos van a trabajar en base a estas secuencias.

Feature es un término que se define como la propiedad o característica medible de un fenómeno.

Aunque este estudio está ligado a un subconjunto de la familia de las quinasas, se ha pretendido que la elección de *features* no contenga elementos propios de esta familia, es decir, que las *features* sean genéricas para cualquier familia de proteínas y así dar la posibilidad de cambiar a otra familia de proteínas sin suponer ninguna dificultad.

Las *features* generadas se han diseñado de tal manera que pueden ser extraídas de forma automática de la estructura primaria de las proteínas, es decir, su secuencia.

4.4.1 Lista de *features*

A continuación, se detallan la lista de *features* generadas que van a servir de input para la clasificación.

4.4.1.1 Composición por aminoácidos.

Este criterio genera una *feature* por cada uno de los 20 aminoácidos, {A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y} ,donde cada *feature* es la proporción de cada aminoácido dentro de su cadena. Es decir:

$$f(a) = \frac{N_a}{N} \qquad a = \{\mathrm{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}\}$$

Siendo Na el número de cierto aminoácido en la secuencia y N la longitud de esa secuencia.

De este modo se generan 20 *features*.

4.4.1.2 Composición por dipéptidos.

De manera análoga a la composición por aminoácidos, este criterio genera la proporción de cada uno de los 400 dipéptidos posibles, generando 400 *features*. Es decir:

$$f(a,b) = \frac{N_{rs}}{N-1}$$

$$r, s = \{\mathrm{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y}\}$$

Siendo N_{rs} el número de ese dipéptido en esa secuencia y N la longitud total de la cadena.

4.4.1.3 Composition/Transition/Distribution features

Este tipo de descriptores de las proteínas fue desarrollado por (Dubchak, 1995) y (Dubchak, 1999). Los aminoácidos son clasificados en tres grupos siguiendo ciertas propiedades.

	Grupo 1	Grupo 2	Grupo 3
Hidrofobicidad	Polar	Neutral	Hidrófoba
	R, K, E, D, Q, N	G, A, S, T, P, H, Y	C, L, V, I, M, F, W
Van der Waals	0-2.78	2.95-4.0	4.03-8.08
	G, A, S, T, P, D, C	N, V, E, Q, I, L	M, H, K, F, R, Y, W
Polarity	4.9-6.2	8.0-9.2	10.4-13.0
	L, I, F, W, C, M, V, Y	P, A, T, G, S	H, Q, R, K, N, E, D
Polarizability	0-1.08	0.128-0.186	0.219-0.409
	G, A, S, D, T	C, P, N, V, E, Q, I, L	K, M, H, F, R, Y, W
Charge	Positive	Neutral	Negative
	K, R	A, N, C, Q, G, H, I, L, M, F, P, S, T, W, Y, V	D, E
Secondary Structure	Helix	Strand	Coil
	E, A, L, M, Q, K, R, H	V, I, Y, C, W, F, T	G, N, P, S, D
Solvent Accessibility	Buried	Exposed	Intermediate
	A, L, F, C, G, I, V, W	R, K, Q, E, N, D	M, S, P, T, H, Y

Tabla 1 Tabla CDT

Se han implementado las *features* según los criterios de *composition* y *transition*.

Composition

Este criterio pretende describir la composición de la secuencia en base a los componentes de cada grupo y propiedad. Las *features* derivadas de este criterio se generan según:

$$c_r = \frac{n_r}{n} \qquad r = 1,2,3$$

Siendo n_r el número de aminoácidos de cierto grupo para cierta propiedad y n la longitud de la secuencia. Por lo que resultarían 21 features, habiendo 7 propiedades y 3 grupos.

Transition

Una transición del grupo 1 al grupo 2 es la proporción en la cual un aminoácido del grupo 1 viene seguido de un aminoácido del grupo 2 o viceversa en la secuencia.

La generación de *features* sigue la siguiente formula:

$$T_{rs} = \frac{n_{rs}+n_{sr}}{n-1} \qquad rs = 12,13,23$$

Siendo n_{rs} el número de dipéptidos que contiene un aminoácido del grupo r seguido de uno del grupo s y viceversa para n_{sr} . n es la longitud de la secuencia.

4.4.1.4 Gen

Dentro de las secuencias en formato FASTA descargadas desde UniProt, la cabecera contiene una descripción que recoge, entre otra información, el gen implicado en la secuencia.

```
>sp|Q9NSK7|CS012_HUMAN Protein C19orf12 OS=Homo sapiens OX=9606
GN=C19orf12 PE=1 SV=3
MERLKSHKPATMTIMVEDIMKLLCSLSGERKMKAAVKHSGKGALVTGAMAFVGGLVGGPP
GLAVGGAVGGLLGAWMTSGQFKPVPQILMELPPAEQQRLFNEAAAIIRHLEWTDAVQLTA
LVMGSEALQQQLLAMLVNYVTKELRAEIQYDD
```

Este campo permite generar una *feature* cualitativa, dando como valor el nombre del gen.

4.4.1.5 Length

Intuitivamente, esta feature se define como la longitud de la secuencia.

4.4.1.6 State

Esta feature, que va a ser el target en la clasificación de las proteínas, define si la secuencia contiene una variante patológica. Puede tener dos valores: *healthy* y *pathologic*.

4.4.1.7 Features derivadas del alineamiento

Una vez efectuado el alineamiento, se obtiene una secuencia alineada para cada secuencia fuente introducida. En base a esto, se pueden derivar las siguientes features:

nGap

Se define como la proporción de *gaps*(-) respecto a la longitud de la secuencia alineada.

nGaps

Se define como el número de grupos de *gaps* internos de la secuencia alineada.

Consensus

Se define como el tanto por ciento de la secuencia alineada que coincide con la secuencia consenso.

5 Machine Learning.

5.1 Introducción.

Una posible definición de Machine Learning es el conjunto de técnicas y algoritmos que hacen que una máquina pueda aprender de los datos disponibles. Mediante la extracción de patrones, los algoritmos dan sentido a los datos.

En el contexto actual donde la cantidad de datos crece exponencialmente, se requiere una alternativa a la extracción manual de reglas y modelos sobre los datos. Los algoritmos de Machine Learning ofrecen una alternativa más eficiente para hacer predicciones y generar conocimiento.

Es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial. Sus aplicaciones son diversas: motores de búsqueda, detección de fraude, reconocimiento facial y de imágenes, conducción automática, etc.

5.2 Tipos de algoritmos

Se pueden dividir en 3 tipos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

El aprendizaje supervisado tiene como objetivo formar un modelo sobre los datos disponibles (training data) y hacer predicciones sobre datos futuros. El término supervisado hace referencia a que los valores de salidas del algoritmo son valores ya conocidos.

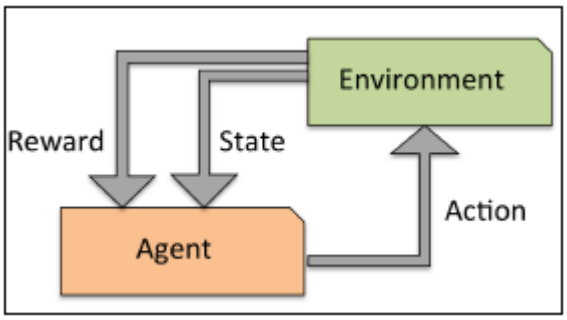
Si el valor de salida pertenece a un rango discreto de valores, el problema debe ser abordado por un clasificador. Si el rango de valores es continuo, el problema debe ser solucionado por una tarea de regresión.

Un clasificador busca predecir la clase de los nuevos datos basándose en datos pasados. Un tipo muy habitual de clasificador es el clasificador binario, que es el que se va a aplicar a este proyecto: clasificar si una secuencia es patógena o no.

Aunque el conjunto de clases no tiene por qué ser binario, puede ser multiclase. Si los valores únicos de clase que tienen los datos conocidos son más de dos, el clasificador puede asignar uno de estos valores a los datos nuevos a clasificar.

Por otra parte, si se quiere predecir una salida continua se tiene que emplear la regresión. La regresión trata de encontrar la relación entre las variables predictoras y la variable continua respuesta para así hacer predicciones.

El aprendizaje por refuerzo tiene por objetivo implementar un sistema (agente) que mejora su rendimiento interactuando con el entorno. El algoritmo interactúa con el entorno y recibe inputs del estado de éste. Este tipo de algoritmos tiene otra señal de entrada, *recompensa,* que mide lo bien que ha sido medida la interacción con el entorno.



18 Aprendizaje por refuerzo

El aprendizaje no supervisado se usa para trabajar con datos no clasificados y con estructura desconocida para extraer información útil sin tener la guía de una respuesta correcta conocida.

5.3 Algoritmos clasificadores.

Para este proyecto se han utilizado los siguientes algoritmos clasificadores, representativos del conjunto de algoritmos de aprendizaje supervisado:

5.3.1 Algoritmo k-NN

El algoritmo k-NN se basa en la noción de "vecino más cercano". Usa la información de los k vecinos más cercanos para clasificar los ejemplos no etiquetados. Para que esto se produzca, el algoritmo requiere un conjunto de ejemplos que hayan sido clasificados. Para cada ejemplo no clasificado, su categoría se asigna por mayoría entre los k vecinos más cercanos.

El término "vecino más cercano" se refiere a los ejemplos clasificados más similares al que se quiere clasificar, que viene dada por una función de similitud que puede ser, por ejemplo, la distancia euclídea, Manhattan, etc.

Fortalezas	Debilidades
Simple y efectivo	No produce un modelo, lo que limita descubrir nuevas lecturas en las relaciones entre características
No hace suposiciones sobre la distribución de los datos	Fase de clasificación lenta
Fase de entrenamiento rápida	Requiere gran cantidad de memoria
	Los atributos nominales y los datos incompletos requieren un procesamiento adicional

Tabla 2 Fortalezas y debilidades KNN

5.3.2 Support Vector Machines

Las máquinas de soporte vectorial, máquinas de vectores de soporte o máquinas de vector soporte (*Support Vector Machines*, SVMs) son un conjunto de algoritmos de aprendizaje supervisado, enfocados a resolución de problemas de clasificación y regresión.

Su objetivo es separar los datos, de forma que exista el mayor margen posible y de forma homogénea. Se basa en el cálculo de los hiperplanos en el espacio multidimensional creado con las distintas características.

Se llama así porque el hiperplano de separación queda definido por las observaciones de cada clase más cercanas a éste.

Algunas de las aplicaciones de los SVM son:

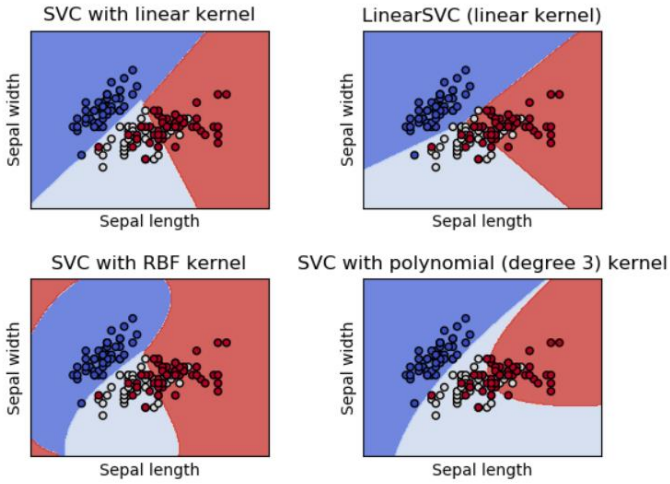
- Clasificar genes diferencialmente expresados a partir de datos de microarrays.

- Clasificación de texto en distintas categorías temáticas.

- Detección de eventos críticos de escasa frecuencia, como terremotos.

Cuando los datos no son separables de forma lineal, es necesario el uso de kernels, o funciones de similitud y especificar un parámetro C para minimizar la función de coste. La elección de este parámetro se realiza por el método de ensayo/error, pero se buscan valores que no sean extremos en la búsqueda del equilibrio sesgo/varianza.

Los kernels más populares son el lineal y el gausiano, aunque hay otros como el polinomial, string kernel, chi-square kernel. etc.



19 Kernels SVM

Fortalezas	Debilidades
Uso en predicción y clasificación. Uso bastante extendido	Requiere especificar parámetro C y función de kernel (prueba y error)
Funciona de forma óptima con ruido	Lento de entrenar, sobre todo a medida que aumenta el número de características.
	Es difícil de interpretar el funcionamiento interno.

Tabla 3 Fortalezas y debilidades SVM

5.3.3 Arboles de decisión.

Los árboles de decisión son clasificadores que usan una estructura de árbol para encontrar relaciones entre las features y las salidas. Usan una estructura de ramificado para predecir la clase de las muestras, cuya predicción se efectúa en los nodos terminales. Cada nodo interno (ramificación), divide los datos indicando una clase potencial para cada conjunto.

Cómo se dividen los nodos puede ser decidido según dos criterios de "impureza" de los nodos: *Gini impurity* y *Entropy.*

Gini Impurity mide la frecuencia en la que un elemento elegido al azar en el subconjunto sería incorrectamente clasificado si se asigna aleatoriamente una clase siguiendo la distribución de clases en ese subconjunto. Alcanza el valor 0 cuando todas las muestras del nodo se asignan a la misma clase.

Entropy (o *Information Gain)* mide la información, o la falta de ella. Es una manera de medir la impureza de un nodo.

Fortalezas	Debilidades
Muy versátil.	Es fácil sufrir overfitting o underfitting.
Puede manejar datos numéricos y nominales, además de datos incompletos.	Pequeños cambios en los datos del modelo pueden cambiar enormemente las producciones.
Mas eficiente que otros modelos más complejos	Árboles grandes pueden ser difíciles de interpretar.

Tabla 4 Fortalezas y debilidades árboles de decisión

5.3.4 Random Forest.

El algoritmo Random Forest implementa la idea de combinar árboles de decisión. Es una técnica de agregación que hace que sea más preciso creando un vector aleatorio, del cual dependen los árboles, con la misma distribución en cada uno de ellos. Los árboles luego se promedian para reducir la variación.

Fortalezas	Debilidades
Uno de los algoritmos más precisos y versátiles.	Posibilidad de overfitting cuando hay ruido.
Eficiente con un gran número de muestras	Es difícil de interpretar.
Buen comportamiento con dimensionalidad grande.	Puede requerir algo de tratamiento previo de los datos.

Tabla 5 Fortalezas y debilidades Random Forest.

5.4 Posibles problemas en Machine Learning y soluciones

Hay ciertos factores que hay que comprobar y tener en consideración cuando se crean los modelos de Machine Learning. Si no se controlan, pueden ocasionar poca precisión en las predicciones o que el modelo se comporte bien para datos vistos, pero no para datos nuevos.

5.4.1 Overfitting.

Overfitting es un término que define cuando un algoritmo se comporta de manera demasiado precisa. Puede parecer una definición un tanto contradictoria, pero se refiere a la situación en la que el modelo se ha ajustado tanto a los datos de entrenamiento que llega a un estado en el que al recibir datos nuevos su

rendimiento es pobre. Esto es debido a que el ruido del training data es cogido como referencia en el modelo y cuando se incorporan nuevos datos que no contienen este ruido el algoritmo es incapaz de generalizar.

Para comprobar si el modelo sufre de *overtfitting* se ha implementado K-fold Cross Validation, técnica que se explicará más adelante.

5.4.2 *Underfitting*

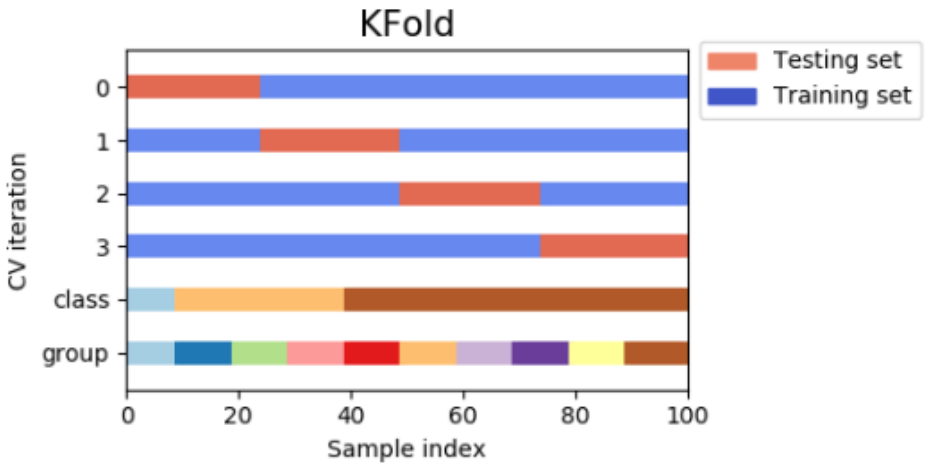
Un modelo que sufre de *underfitting* es un modelo que no se puede construir correctamente con el dato de training ni puede generalizar con datos nuevos. Es fácilmente detectable si los resultados son malos para los datos de training.

K-fold Cross Validation también permite detectar si existe underfitting.

5.4.3 K-fold Cross Validation

K-fold Cross Validation es una técnica que permite comprobar la precisión de los algoritmos, garantizando que los datos son independientes.

Este método divide los datos en k grupos de muestras, llamados folds, del mismo tamaño si es posible. Se itera k veces, usando cada vez un fold para test y los k-1 folds restantes para el conjunto de training.



20 K-fold Cross Validation

5.4.4 *Curse of dimensionality.*

Cuando el espacio dimensional en el que se trabaja es demasiado grande puede ocurrir que el número de *features* impacten en el rendimiento de los algoritmos, e incluso, en algoritmos como KNN se produzca *overfitting*. Este

tamaño del espacio dimensional también tiene repercusión en el uso de la memoria y el tiempo de cómputo.

Para atajar este problema se puede acudir a técnicas de reducción de dimensionalidad, como el PCA.

5.4.5 *Principal Component Analysis.*

PCA es una técnica de reducción de dimensionalidad que disminuye el número de variables existentes, transformándolas en componentes principales, perdiendo la menor información posible.

Esta información se puede interpretar como la varianza que contienen los datos, por lo que el número de componentes principales escogidos debe tener la mayor varianza posible.

5.4.6 Rendimiento pobre.

Cuando los algoritmos no se comportan de una manera suficientemente aceptable se puede recurrir a técnicas para maximizar su rendimiento. Una opción es el ajuste de hiperparámetros.

5.4.7 Ajuste de hiperparámetros.

Los modelos disponen de ciertos parámetros que se fijan antes de su ajuste y que los definen. Buscar el mejor ajuste según los datos del modelo aumenta el rendimiento. Esta búsqueda de la combinación óptima de los parámetros puede ser costosa, pero sirve, además de para mejorar el rendimiento, para comprender la relación entre los datos y estos parámetros y su funcionamiento.

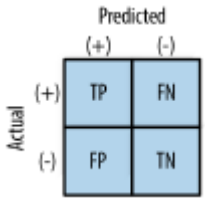
La técnica que se ha implementado para la búsqueda de esta combinación de parámetros es *Grid Search Cross-Validation*, que realiza el ajuste para cada una de las posibles combinaciones entre los valores proporcionados para los hiperparámetros seleccionados y devuelve el modelo con la combinación que ofrece mayor rendimiento, una vez efectuado *Cross-Validation*, para una métrica también definida. Esta métrica puede ser: *accuracy*, *precision*, *recall*, etc. El significado de estas métricas será definido en el siguiente apartado.

5.5 Métricas

Para evaluar el rendimiento del algoritmo existen diferentes métricas que permiten conocer cómo se comporta el modelo:

5.5.1 Confusion matrix

Las *confusion matrix* permiten representar de forma tabular el rendimiento del algoritmo supervisado. Cada fila representa los valores reales del modelo mientras que las columnas representan los valores predichos en el caso de la figura, aunque se pueden configurar de la forma contraria. Los valores TP, FP, FN y TN dependen también de la clase que considerada positiva (+) o negativa (-). En este TFM se ha considerado que la clase positiva es *pathologic(1)* y la negativa *healthy(0).*



21 Confusion Matrix

TP = número de verdaderos positivos. Valores realmente positivos que se han predicho positivos.

TN = número de verdaderos negativos. Valores realmente negativos que se han predicho negativos.

FP = número de falsos positivos. Valores realmente negativos que se han predicho positivos (Error tipo I).

FN = número de falsos negativos. Valores realmente positivos que se han predicho negativos (Error tipo II).

En base a estos valores, se pueden derivar diferentes métricas.

5.5.2 Accuracy

Accuracy define la precisión global del predictor, midiendo la proporción de las muestras correctamente predichas sobre el total de muestras:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

5.5.3 Precision

Precision define la proporción de los positivos predichos correctamente sobre todos los positivos predichos:

$$Precision = \frac{TP}{TP + FP}$$

Un alto valor de esta métrica indica un bajo valor de los falsos positivos.

5.5.4 *Recall* (*Sensitivity*)

Recall define la proporción de los valores positivos predichos en proporción a todos los valores realmente positivos:

$$Recall = \frac{TP}{TP+FN}$$

Tiene un valor aceptable a partir de 0.5.

5.5.5 *F1-score*

F1-score es la media ponderada de *precision* y *recall*. Por lo tanto, implica a los falsos negativos y los falsos positivos:

$$F1 - score = \frac{2(Recall * Precision)}{Recall + Precision}$$

5.5.6 *Roc – Curves*

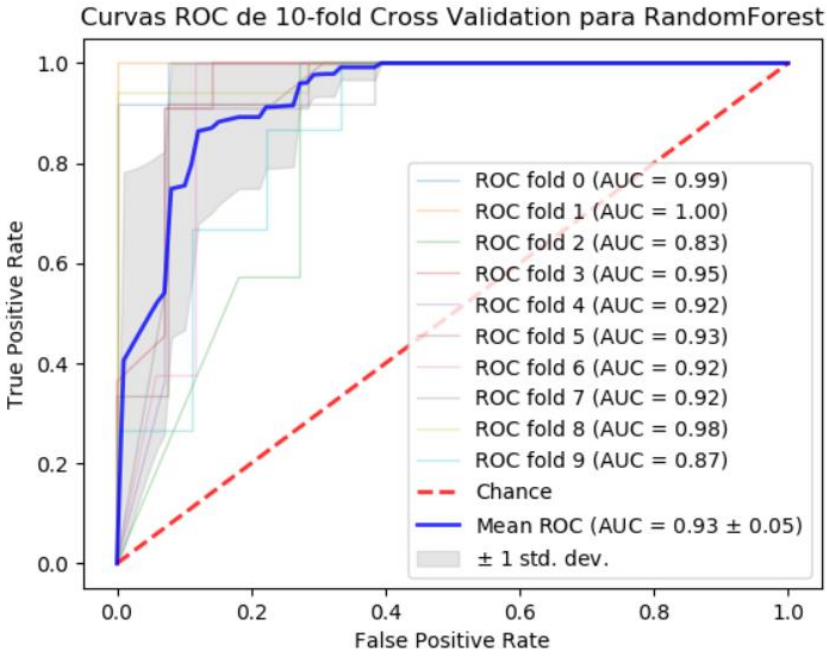
Las curvas ROC (*Receiver Operating Characteristic*) son una representación gráfica de la sensibilidad frente a la especificidad para un clasificador binario según se varía el umbral de discriminación.

Otra interpretación sería que presenta la representación del *ratio* de verdaderos positivos frente al *ratio* de falsos positivos según se varía el umbral de discriminación.

Es deseable que el área debajo de la curva sea lo más amplia posible, ya que eso significa que el *ratio* de verdaderos positivos es elevado.

$$True\ positive\ rate = \frac{TP}{TP + FN}$$

$$False\ positive\ rate = \frac{FP}{FP + TN}$$



22 Ejemplo de curvas ROC

6 La aplicación.

6.1 Introducción

Este apartado tiene como objetivo describir la estructura de la aplicación, seguir el ciclo de ejecución completo y mostrar los resultados obtenidos en forma de reportes y gráficas.

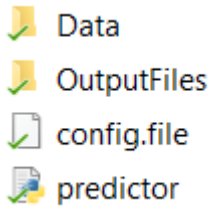
6.2 Requisitos

Para poder ejecutar la aplicación se debe tener instalado:

- Python 3.7.0 https://www.python.org/
- ClustalW (línea de comandos) http://www.clustal.org/clustal2/
- Muscle. https://www.drive5.com/muscle/
- Los paquetes que no son *Batteries Included*, mediante el gestor de paquetes pip, ejecutando *pip3 install nombre_paquete*.

6.3 Archivos de los que consta la aplicación.

La aplicación está compuesta de los siguientes archivos:



23 Archivos de la aplicación

Data: carpeta que contiene los archivos de datos de secuencias. *kinases.fasta* contiene las secuencias sin mutaciones y *patkinases.fasta* las secuencias con las mutaciones patológicas.

OuputFiles: Carpeta que va a recoger todos los archivos de salida que se van generando a lo largo de la ejecución del programa.
Esta carpeta se genera, si no existe, en el mismo directorio que el archivo predictor.py cuando se ejecuta el script.

predictor.py: es la aplicación en sí, que contiene el código que ejecuta la aplicación.

config.file: archivo de texto que recoge los diferentes parámetros de configuración y ejecución de la herramienta. También contiene la configuración de los hiperparámetros. El archivo se encuentra disponible en:

https://github.com/msalinero/msd-tfm/blob/master/Source/config.file

La estructura de este archivo es la siguiente:

```
[DEFAULT]
clustalw_exe = C:/Program Files (x86)/ClustalW2/clustalw2.exe
[ML]
PCA = True
kfolds = 10
target_metric = recall
[RANDOMFOREST]
doRF = True
n_estimators = [200, 500]
max_features = ['auto', 'sqrt', 'log2']
max_depth = [4,5,6,7,8]
criterion = ['gini', 'entropy']
[SVC]
doSVC = True
kernel = ['linear', 'rbf', 'poly']
C = [1,0.25,0.5,0.75]
gamma = [1,2,3,'auto']
decision_function_shape = ['ovo','ovr']
shrinking = [True,False]
[TREE]
doTree = True
criterion = ['gini', 'entropy']
min_samples_leaf = [1, 5, 10]
max_depth = [5,10,15,20,100]
[KNN]
doKNN = True
n_neighbors = [2,3,4,5,6,7,8,9,10]
```

Estos atributos siguen el siguiente esquema jerárquico:

Sección `[DEFAULT]`:

`clustalw_exe`: Dirección del ejecutable ClustalW

Sección `[ML]`:

`kfolds`: Número de folds para k-fold Cross-Validation
`PCA`: Si es "True" se efectúa Principal Component Analisis.
`target_metric`: Métrica que determina los parámetros elegidos cuando se efectúa *grid search*.

Sección `[RANDOMFOREST]`:

`doRF`: Si es "True", se ejecuta este clasificador.
`criterion`: Función para medir la calidad de los split. Valores posibles: "gini", "entropy".
`n_estimators`: número de decisión trees.
`max_features`: número de features que se consideran para elegir el mejor split.
`max_depth`: profundidad máxima del árbol.

Sección `[SVC]`:

`doSVC`: Si es "True", se ejecuta este clasificador.
`kernel`: Especifica el kernel usado en el algoritmo. Los posibles valores son 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' .
`C`: Valor de penalización. Si C es grande, hay poco margen de error.
`gamma`: Coeficiente solo válido para los kernel 'rbf','poly' y 'sigmoid'.
`decision_function_shape` : parámetro que determina si la función de decisión es 'ovr' (one-vs-rest), como todos los demás clasificadores, con dimensión (numero_de_muestras, numero_de_clases) o 'ovo'(one-vs-one) con dimensión (número_de_muestras, número_de_clases *(numero_de_ clases-1)/2)
`shrinking` : Determina si se utiliza esta heurística.

Sección `[TREE]`:

`doTree`: Si es 'True', se ejecuta este clasificador.
`criterion`: Función para medir la calidad de los split. Valores posibles: "gini", "entropy".
`min_samples_leaf` : Número mínimo de muestras para poder ser una hoja.
`max_depth`: profundidad máxima del árbol.

Sección `[KNN]`:

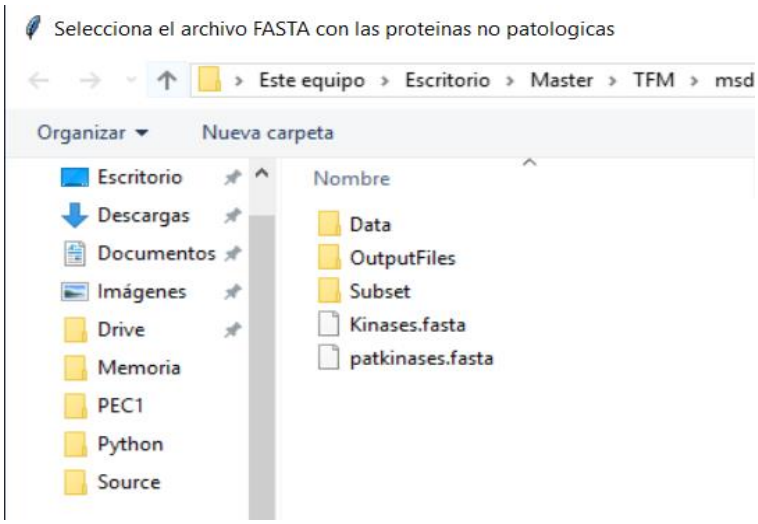
`doKNN`: Si es 'True', se ejecuta este clasificador
`neighbors`: Numero de k para knn. Vecinos a visitar.

6.4 Ciclo de ejecución.

Para comenzar la ejecución, se debe hacer uso del intérprete de Python para el script:

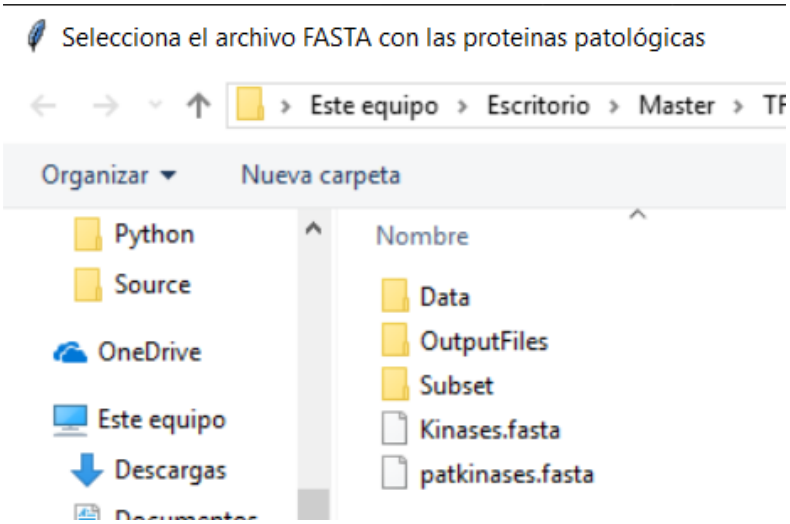
```
Source> python .\predictor.py
```

Una vez ejecutado el script, se presenta un diálogo en el que se pide la ubicación del archivo FASTA que contiene las secuencias libres de las mutaciones patológicas.



24 Selección del archivo FASTA

Una vez seleccionado el archivo se cargan estas secuencias en la aplicación y se repite el diálogo para seleccionar el archivo que contiene las secuencias que abarcan las variaciones patológicas.



25 Selección del archivo FASTA.

```
Source> python .\predictor.py
Carga de las proteinas no patologicas

Cargadas 123 proteinas no patologicas

Carga de las proteinas patologicas
```

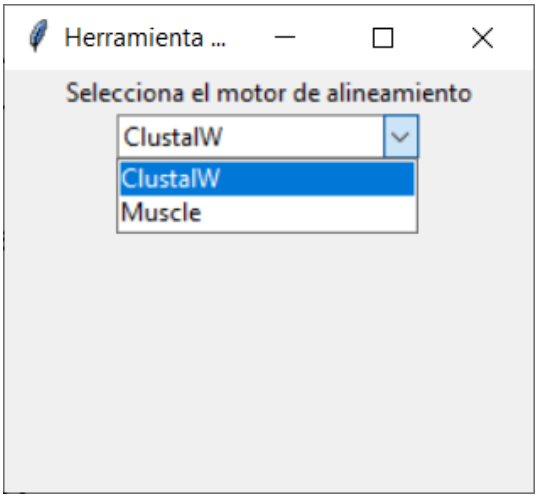
Una vez seleccionado el archivo, se muestran mensajes explicativos informando del número de secuencias cargadas y la generación de un archivo FASTA que contiene la totalidad de las secuencias, mostrando la ruta absoluta del archivo (el archivo se ubica dentro del directorio *OutputFiles*, como la totalidad de los archivos generados durante la ejecución):

```
Cargadas 125 proteinas patológicas

La totalidad de 248 proteinas se han volcado en el archivo
```

Todos los archivos generados llevan un *timestamp* como prefijo que permite identificar los archivos generados en cada ejecución. El *timestamp* tiene formato "%d%m%Y-%H%M%S".

Durante cada carga de estos dos conjuntos de secuencias se crea una primera tanda de *features* para los conjuntos de secuencias importados.

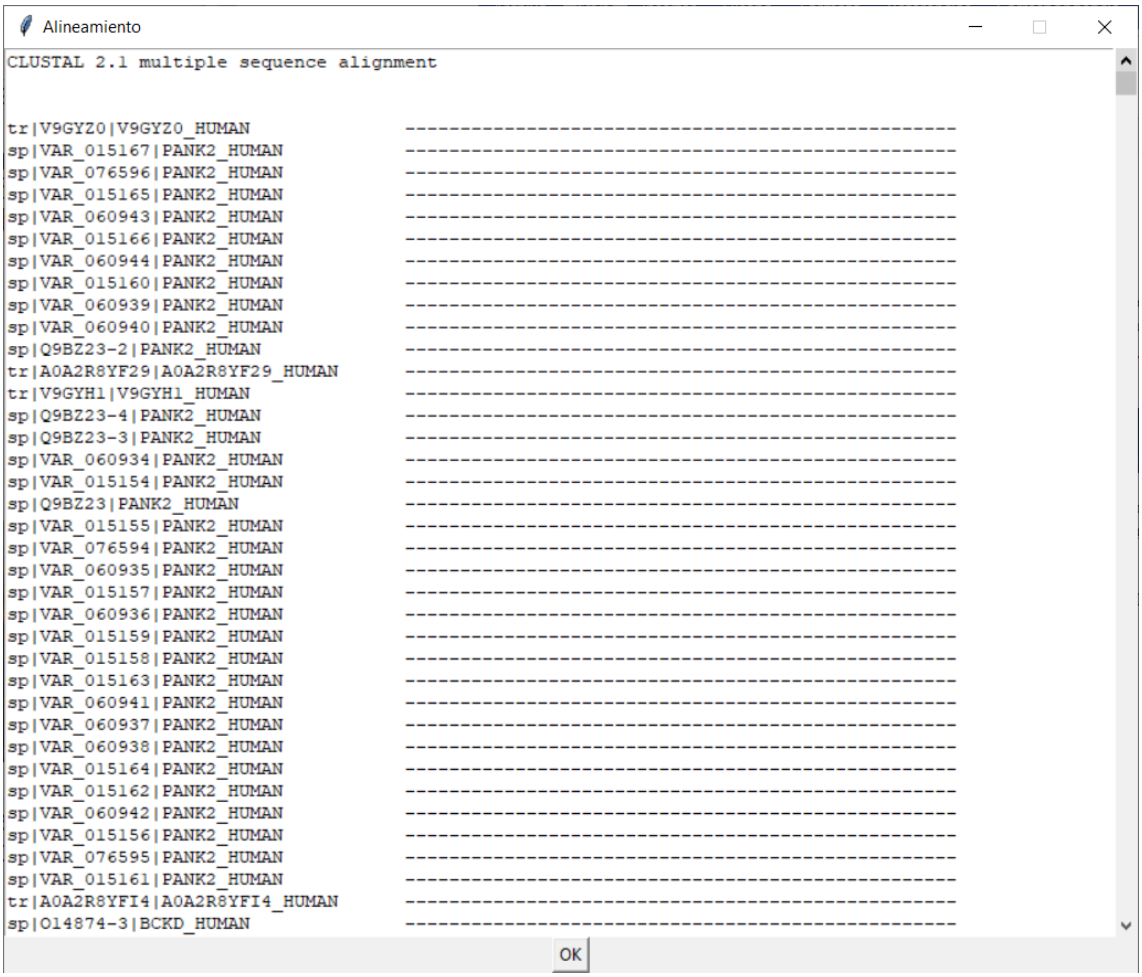
A continuación, se muestra un desplegable donde se elige la herramienta de alineamiento



26 Selección del alineamiento.

Al pulsar en el botón OK, comienza el alineamiento. Una vez finalizado, se muestra el tiempo empleado y una presentación del alineamiento,

```
Ejecucion del alineamiento con ClustalW

El alineamiento con CLustalW ha tardado 137.24 segundos.
```

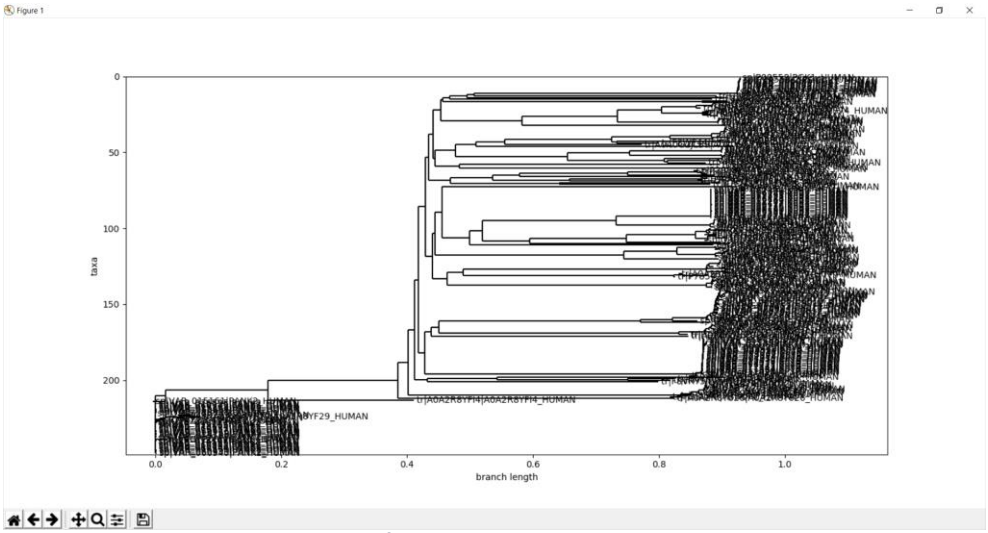


27 Resultado alineamiento.

Esta información está contenida en el archivo generado de nombre *timestamp*AllProt.aln, para su posterior consulta si se desea.

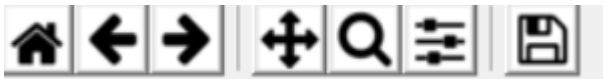
Pulsando OK se muestra el *prompt* y el siguiente gráfico, que es el árbol filogenético resultante del alineamiento

```
Representacion del arbol filogenetico
```

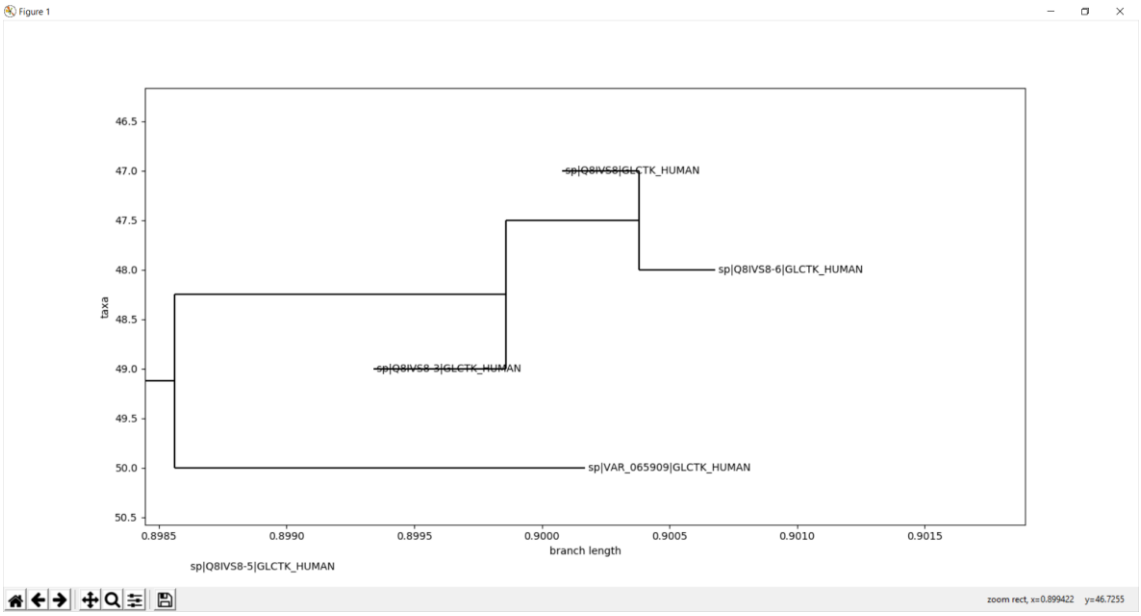


28 Árbol filogenético.

Se puede explorar el contenido del gráfico haciendo uso de las opciones situadas en la parte inferior izquierda:



Así se pueden visualizar de manera más clara las relaciones que se muestran en el árbol:



29 Ampliación árbol filogenético

El archivo sobre el que se construye este árbol filogenético corresponde a *timestampAllProt.dnd*

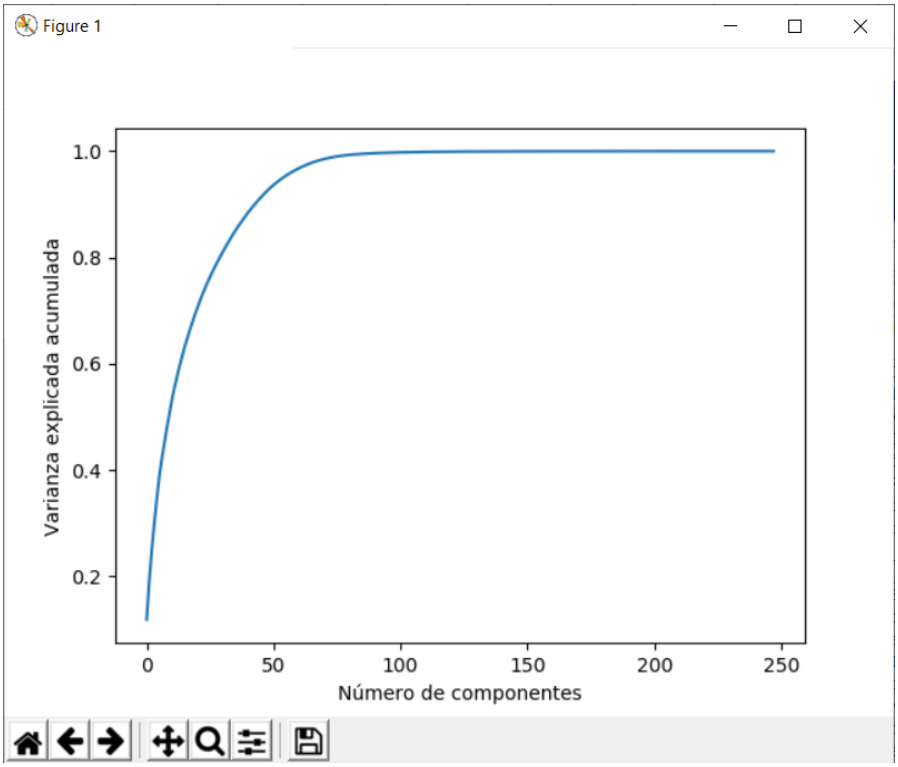
Al cerrar esta ventana, la ejecución continúa con el proceso de creación de las *features* restantes. Cuando este proceso ha terminado, se informa de las dimensiones que conforman el dataframe.

```
Generando las features

El dataframe generado tiene unas dimensiones de 248 muestras y 468 columnas
```

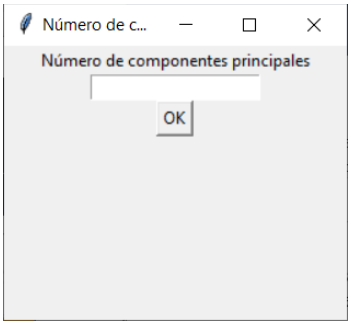
El dataframe se exporta a un archivo Excel para tener una referencia original antes del tratamiento de los datos para así poder explorarlos. El archivo es nombrado como *timestampDataframe.xlsx*. También se da como resultado una exploración de los datos recogida en el archivo *timestampSummary.xlsx*.

```
Dataframe exportado al archivo C:\

Exploracion de los datos exportado al archivo C:\
```

Si la opción de ejecutar el PCA en el config.file está activada se muestra una gráfica con la varianza explicada acumulada en relación al número de componentes principales. También aparece un diálogo para poder introducir el número de componentes principales que se crea conveniente según la varianza explicada acumulada representada en la gráfica:

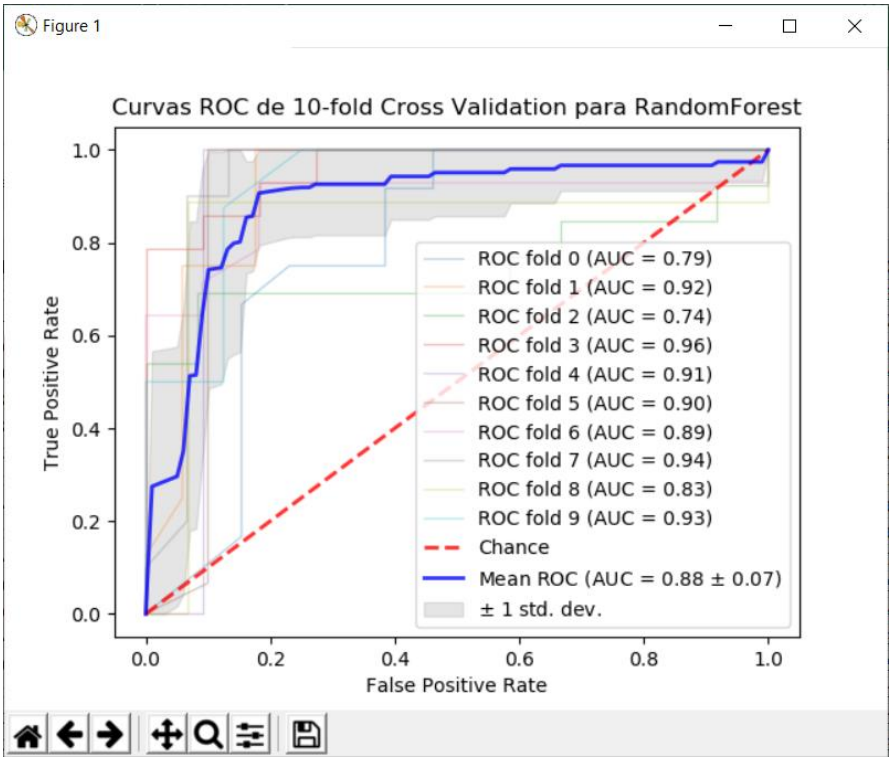


30 Varianza explicada acumulada.



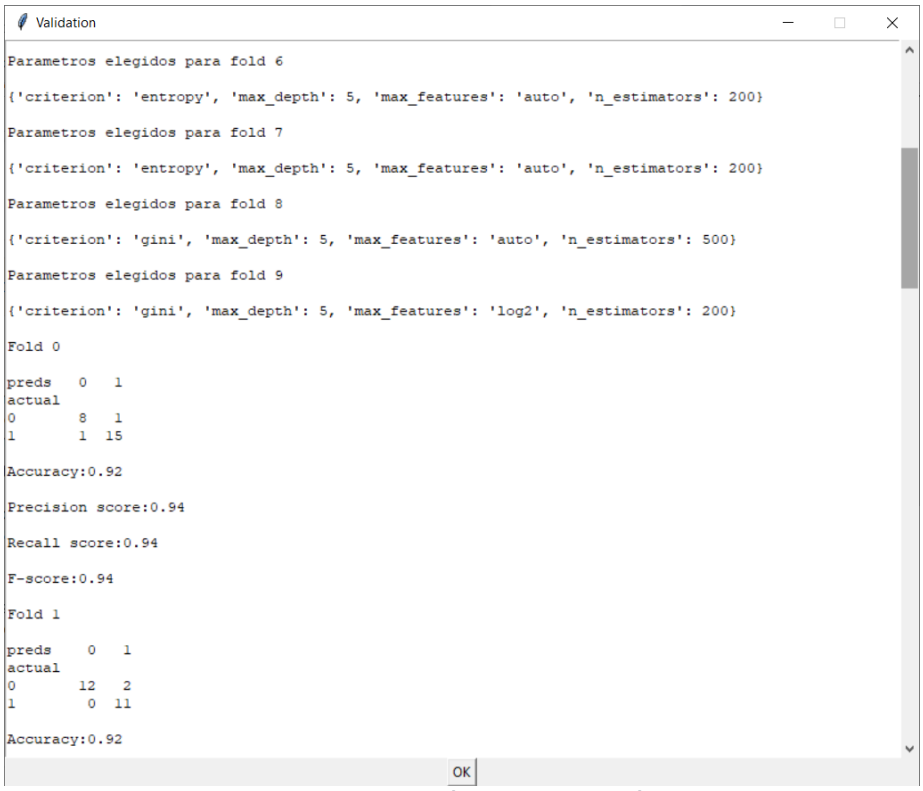
El nuevo dataframe con el PCA realizado también se exporta a un archivo nombrado timestampDfPCA.xlsx.

La parte final de la ejecución muestra los resultados de los clasificadores ejecutados según la configuración. Cada ejecución de cada clasificador muestra dos ventanas: la gráfica de las curvas ROC



31 Ejemplo Curvas ROC ejecución.

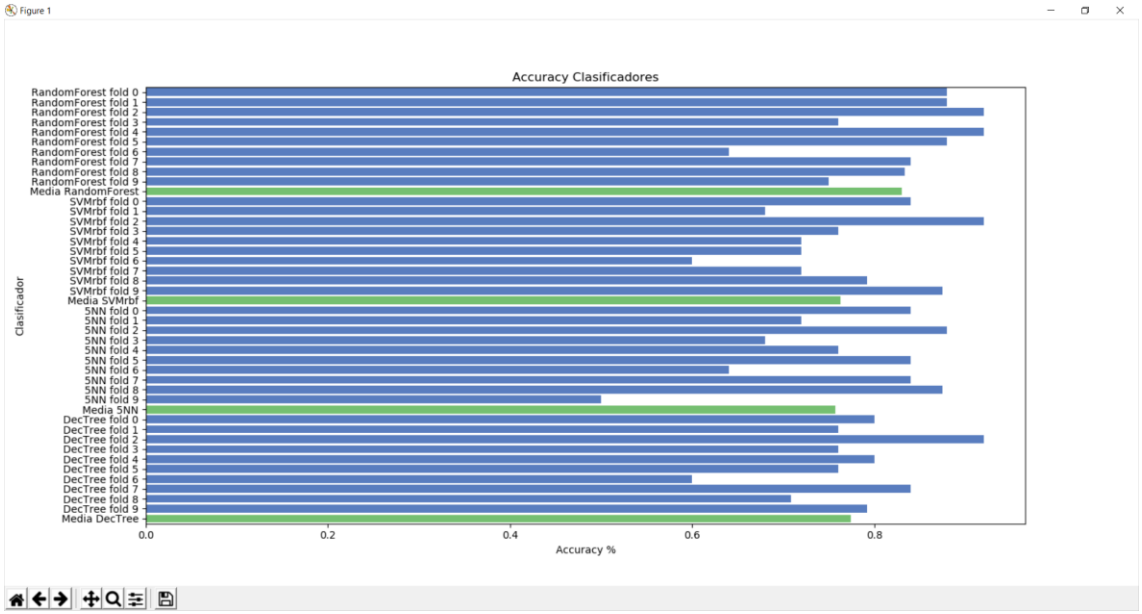
Y la ventana *Validation* que contiene las *confusion matrix* de cada *fold* y sus correspondientes métricas. También se detallan qué hiperparámetros se han elegido para cada *fold* por *grid search*:



32 Ventana con los datos de métricas e hiperparámetros

Si se pulsa *Ok* en la ventana *Validation* se muestran los resultados de la ejecución del siguiente clasificador.

Una vez se han mostrado los resultados individuales de todos los clasificadores, se muestra un histograma que ilustra una comparativa en base al *accuracy* de los clasificadores ejecutados.



33 Ejemplo comparación

Cuando se cierra esta ventana se da por concluida la ejecución de la aplicación.

```
Fin de la ejecucion
```

6.5 Diagrama de clases.

En el diseño de las clases que componen la aplicación se ha buscado la escalabilidad a la hora de incorporar nuevos algoritmos de Machine Learning y nuevas métricas.

La relación de clases es:

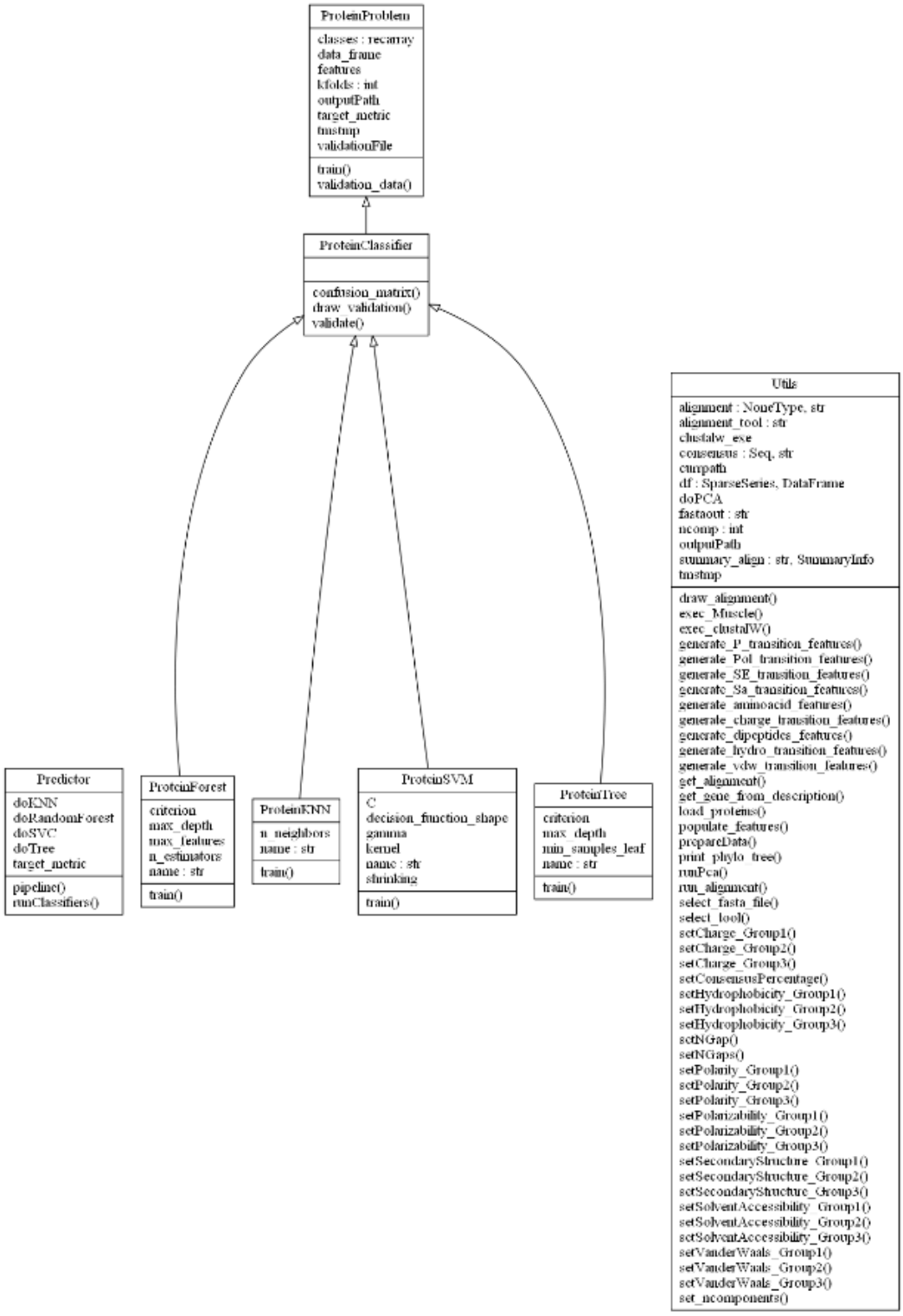
- Util: Clase que contiene los métodos que recopilan y generan los datos que van a ser consumidos por los algoritmos de machine learning.

- ProteinProblem: Clase padre que contiene tiene como atributos los datos comunes a los clasificadores y los datos. Tiene como función principal validation_data() que implementa k-fold Cross Validation.

- ProteinClassifier: Clase intermedia entre la clase padre y las clases de los clasificadores.

- ProteinForest, ProteinKNN, ProteinTree, ProteinSVM: clases que implementan los diferentes clasificadores.

- Predictor: Clase que implementa el pipeline principal de la aplicación.



34 Diagrama UML de clases

6.6 Código fuente.

La totalidad del código fuente se encuentra disponible en el repositorio

https://github.com/msalinero/msd-tfm

A continuación de explican las partes del código más relevantes.

6.6.1 Importación de librerías.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from numpy.random import permutation
from numpy import array_split, concatenate
from sklearn.metrics import mean_squared_error, roc_curve, auc
import sys
import re
import os
import subprocess
import numpy as np
import pandas as pd
import Bio.Align.Applications as ba
from Bio import Phylo
from Bio import SeqIO
from Bio import AlignIO
from Bio.Align import AlignInfo
import tkinter as tk
from tkinter import filedialog
from tkinter.ttk import *
from tkinter import scrolledtext
from tkinter import *
import time
import configparser
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import datetime
import itertools
from scipy import interp
from sklearn.decomposition import PCA
import seaborn as sns
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

pd.options.mode.chained_assignment = None  # default='warn'
```

Destacar las librerías que no son *Batteries Included* en Python: Biopython, Scikit-learn y Pandas. Tkinter está incluida en Python y se hace uso de ella para la GUI.

6.6.2 Clase *Util*().

La clase `Util` se encarga de la importación y preparación de los datos. Genera las *features* que luego servirán de input para los algoritmos.

El método `load_proteins()` lee los archivos FASTA de secuencias mediante el método `parse` de Biopython.SeqIO

```
# Leemos el fichero y creamos la lista de SeqRecord
healthySeqRecords = SeqIO.parse(f1, 'fasta')
```

y crea las primeras *features*. Para configurar el pandas.Dataframe se itera por las secuencias y se crea un diccionario por cada una de las secuencias. Los valores de las entradas de estos diccionarios son las *features*:

```
for seq in healthySeqRecords:
                        numHealth += 1
                        dict1 = {}

                        # Generar features preliminares
                        dict1.update({'id': seq.id,
                                      'state': 'healthy',
                                      'length': len(seq),
                                      'gene':
self.get_gene_from_description(seq.description)
                                      })

                        # Generar features de proporcion de aminoacidos
                        self.generate_aminoacid_features(str(seq.seq),
dict1)

                        # Generar features de proporcion de dipeptidos
                        self.generate_dipeptides_features(str(seq.seq),
dict1)

                        # Generar Transition features
                        self.generate_P_transition_features(
                            str(seq.seq), dict1)
                        self.generate_Pol_transition_features(
                            str(seq.seq), dict1)
                        self.generate_SE_transition_features(
                            str(seq.seq), dict1)
                        self.generate_Sa_transition_features(
                            str(seq.seq), dict1)
                        self.generate_charge_transition_features(
                            str(seq.seq), dict1)
                        self.generate_hydro_transition_features(
                            str(seq.seq), dict1)
                        self.generate_vdw_transition_features(
                            str(seq.seq), dict1)

                        rows_list.append(dict1)

                        outfile.write(">" + seq.id + "\n")
                        outfile.write(str(seq.seq) + "\n")
```

Una función ejemplo de cómo se generan las *features* es:

```
# Transition features para esta propiedad
def generate_hydro_transition_features(self, secuence, dict):
      groups = ['','[RKEDQN]','[GASTPHY]','[CLVIMFW]']

      for r,s in [[1,2],[1,3],[2,3]]:
            grs = re.findall(groups[r]+groups[s], secuence)
            gsr = re.findall(groups[s]+groups[r], secuence)
            column_name = 'hydro.Tr' + str(r) + str(s) + str(s) +
str(r)
            dict[column_name] = (len(grs) + len(gsr))/ (len(secuence)
- 1)
```

Donde se hace uso de expresiones regulares para reconocer los patrones deseados. La función recibe la secuencia y el diccionario con los valores de las *features* ya creadas y crea las nuevas en base a la secuencia entrada.

Dentro del método *runPCA()*, se hace uso de `sklearn.decomposition.PCA` para realizar el PCA.

```
pca = PCA(n_components = self.ncomp)
pca.fit(dfAux)
dfPCA = pd.DataFrame(pca.transform(dfAux), columns=[
                        'PCA%i' % i for i in range(self.ncomp)],index
= dfAux.index)
```

6.6.3 Clase *ProteinProblem()*.

Dentro del método *validation_ data*() se crean los folds:

```
folds = self.kfolds
df = self.data_frame
response = []
# Comprobar que hay mas observaciones que folds
assert len(df.index) > folds
# Generar particiones
perms = array_split(permutation(len(df.index)), folds)
      for i in range(folds):
            train_idxs = list(range(folds))
            train_idxs.pop(i)
            train = []
                  for idx in train_idxs:
                        train.append(perms[idx])
                        train = concatenate(train)

                        test_idx = perms[i]

                        # Observaciones para training
                        training = df.iloc[train]
                        # Observaciones para test
                        test_data = df.iloc[test_idx]
```

Y se obtienen las predicciones con:

```
classifier = self.train(training[self.features], y)
expected = self.__factorize(test_data)
# Predecimos para el conjunto test
predictions = classifier.predict(test_data[self.features])
```

El método devuelve los resultados en una lista:

```
response.append([predictions, expected])
---
---
return response
```

6.6.4 Clase *ProteinClassifier()*.

El método confusion_matrix sirve para representar las confusion_matrix en el archivo

```
@staticmethod
def confusion_matrix(train, test):
                  return pd.crosstab(train, test, rownames=['actual'],
colnames=['preds'])
```

El método validate() hace uso de expected y predicted para calcular las métricas

```
for test, training in self.validation_data():

      print("Fold {}\n".format(ncfs))
      # Pintar confusion matrix
      cm = self.confusion_matrix(training, test)
      print(cm)
      print()
      cm1 = confusion_matrix(training, test, labels=[0, 1])

      # Accuracy
      accuracy = accuracy_score(training,test)
      mean_accuracy = mean_accuracy + accuracy
      print("Accuracy:{0:.2f}\n".format(accuracy))
      #Precision
      precision = precision_score(training, test)
      mean_precision = mean_precision + precision
      print("Precision score:{0:.2f}\n".format(precision))
      # Recall Score
      recall = recall_score(training, test)
      mean_recall = mean_recall + recall
      print("Recall score:{0:.2f}\n".format(recall))
      # F-Score
      f1 = f1_score(training, test)
      mean_f1 = mean_f1 + f1
      print("F-score:{0:.2f}\n".format(f1))
```

6.6.5 Clases relativas a los algoritmos

Estas cuatro clases tienen la misma estructura. Su constructora lee del config.file los hiperparámetros y el método `train()` devuelve el mejor clasificador por *Grid Search* mediante el método `GridSearchCV()`

```
class ProteinForest(ProteinClassifier):

        def __init__(self,data,tmstmp,outputPath):

                self.name = 'RandomForest'

                super().__init__(data, tmstmp, outputPath)

                # Hiperparametros random forest
                config = configparser.ConfigParser()
                config.read(os.path.dirname(os.path.abspath(__file__))
+ '\\config.file')
                self.n_estimators =
eval(config['RANDOMFOREST']['n_estimators'])
                self.max_features =
eval(config['RANDOMFOREST']['max_features'])
                self.max_depth =
eval(config['RANDOMFOREST']['max_depth'])
                self.criterion =
eval(config['RANDOMFOREST']['criterion'])


        def train(self, X, Y):

                # Hacemos grid search para determinar los mejores
parámetros
                param_grid = {
                        'n_estimators': self.n_estimators,
                        'max_features': self.max_features,
                        'max_depth' : self.max_depth,
                        'criterion' : self.criterion
                }

                classifier = RandomForestClassifier()
                CV_rfc = GridSearchCV(
                    estimator=classifier, param_grid=param_grid, cv=3,
scoring = self.target_metric)

                classifier = CV_rfc.fit(X, Y)
                return classifier
```

6.6.6 Clase *Predictor().*

La clase *Predictor()* implementa los métodos que van a:

- Ejecutar las validaciones de los algoritmos.
- Presentar la gráfica comparativa.
- Exportar las métricas.
- Ejecutar el pipeline principal.

```
# Clase que implementa el pipeline principal
class Predictor(object):

        def __init__(self):

                # Parámetros de ejecución.
                config = configparser.ConfigParser()
                config.read(os.path.dirname(
                    os.path.abspath(__file__)) + '\\config.file')

                self.doSVC = config['SVC']['doSVC']
                self.doRandomForest = config['RANDOMFOREST']['doRF']
                self.doTree = config['TREE']['doTree']
                self.doKNN = config['KNN']['doKNN']
                self.target_metric = config['ML']['target_metric']

        # Ejecuta los clasificadores y exporta los resultados
        def runClassifiers(self, ut):

                # Algoritmos ML
                algoritmos_mls = []
                if (self.doRandomForest == 'True'):
                        algoritmos_mls.append(ProteinForest(
                            ut.df, ut.tmstmp, ut.outputPath))
                if (self.doSVC == 'True'):
                        algoritmos_mls.append(ProteinSVM(
                            ut.df, ut.tmstmp, ut.outputPath))
                if (self.doKNN == 'True'):
                        algoritmos_mls.append(ProteinKNN(
                            ut.df, ut.tmstmp, ut.outputPath))
                if (self.doTree == 'True'):
                        algoritmos_mls.append(ProteinTree(
                            ut.df, ut.tmstmp, ut.outputPath))

                # Validar los algoritmos
                log_cols = ["Clasificador", "accuracy",
                            "precision", "recall", "f1"]
                log = pd.DataFrame(columns=log_cols)
```

```
                for ml in algoritmos_mls:
                        for metrics in ml.validate():
                                log_entry = pd.DataFrame(
                                    [metrics], columns=log_cols)
                                log = log.append(log_entry)


                # Tabla de resultados
                sns.set_color_codes("muted")
                clrs = ['g' if ('Media' in y)
                        else 'b' for y in log["Clasificador"]]
                sns.barplot(x=self.target_metric, y="Clasificador",
                            data=log, palette=clrs)

                plt.xlabel(self.target_metric)
                plt.title('{}
Clasificadores'.format(self.target_metric))
                plotFile = ut.outputPath + ut.tmstmp +
"PerformanceComp.png"
                plt.savefig(plotFile, bbox_inches='tight')
                plt.show()

                #Exportar los resultados
                excelLog = ut.outputPath + ut.tmstmp + "AlgLog.xlsx"

                log = log.set_index('Clasificador')
                log.to_excel(excelLog)
```

Método que implementa el pipeline principal

```
        def pipeline(self):

                util = Utils()

                # Preparar los datos
                util.prepareData()

                # Ejecutar clasificadores
                self.runClassifiers(util)

                print("Fin de la ejecución\n")
```

7 Conclusiones y resultados.

7.1 Archivos generados.

Como resultado de un ciclo completo de ejecución se generan un conjunto de archivos que recogen los datos obtenidos durante el proceso, con el objetivo de dejar constancia de los resultados y dar posibilidad a un análisis posterior

El listado de archivos y su contenido es:

-/OutputFiles/*timestamp*AllProt.fasta: la totalidad de las secuencias de proteínas.

-/OutputFiles/*timestamp*AllProt.aln: alineamiento de las secuencias.

-/OutputFiles/*timestamp*AllProt.dnd: árbol filogenético en formato Newick.

-/OutputFiles/*timestamp*Dataframe.xlsx: *dataframe* con las *features* originales sin tratar.

-/OutputFiles/*timestamp*Summary.xlsx: exploración de las *features* originales.

	A	AA	AC	AD
count	248	248	248	248
unique				
top				
freq				
mean	0,085052	0,00916	0,001101	0,002691
std	0,029319	0,008034	0,007235	0,002799
min	0,012821	0	0	0
25%	0,061333	0,004078	0	0
50%	0,07652	0,005495	0	0,001851
75%	0,111498	0,013245	0,00081	0,004756
max	0,2	0,044444	0,111111	0,022222

35 Summary.xlsx

-/OutputFiles/*timestamp*DfScale.xlsx: dataframe con las features originales estandarizadas.

-/OutputFiles/*timestamp*DfPCA.xlsx: dataframe una vez efectuado el PCA, con la clase target, y los componentes principales.

-/OutputFiles/*timestampNombre_Clasificador*Validation.out: hiperparámetros empleados, *confusion matrix* y métricas de un clasificador en cada fold.

-/OutputFiles/*timestampNombre_Clasificador*RocCurves.png: imagen con las curvas ROC de cada *fold* para este clasificador.

-/OutputFiles/*timestampNombre_Clasificador*GridSearch.xlsx: tabla con los parámetros escogidos para cada *fold*.

-/OutputFiles/*timestamp*PerformaceComp.png: imagen con la comparación de *target_score*.

-/OutputFiles/*timestamp*AlgLog: archivo con las métricas de cada *fold*.

7.2 Resultados.

En este apartado se van a discutir los resultados obtenidos durante la ejecución de la aplicación.

7.2.1 Alineamiento.

En la comparativa de tiempo de ejecución entre las herramientas de alineamiento Muscle es más rápida que ClustalW. En este tiempo se incluye la generación del árbol filogenético:

```
El alineamiento con CLustalW ha tardado 137.40 segundos.
```

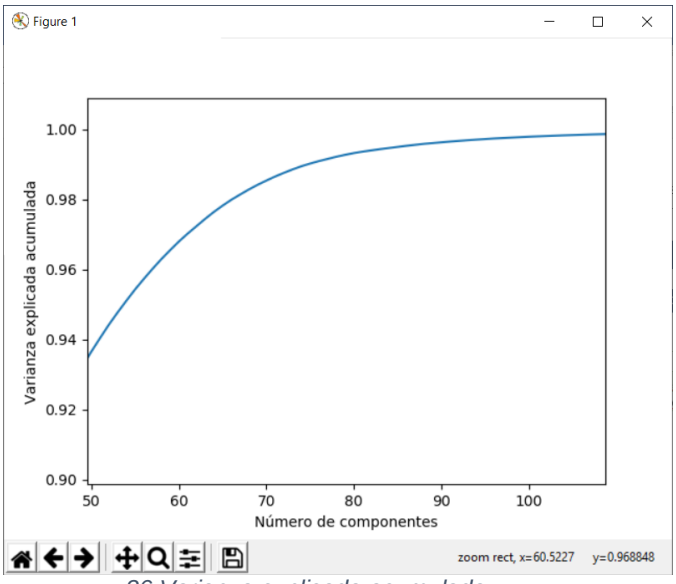
```
El alineamiento con Muscle ha tardado 81.69 segundos.
```

	Tiempo(s)
ClustalW	137.40
Muscle	81.69

Tabla 6 Tiempos alineamiento

7.2.2 Número de componentes principales.

En cuanto a la elección del número de componentes principales, según ilustra la gráfica:



36 Varianza explicada acumulada.

Para 60 componentes se tiene un 96% de la varianza explicada acumulada, disminuyendo de forma notable la dimensionalidad con una pérdida de varianza escasa.

7.2.3 Rendimiento de los clasificadores.

El siguiente apartado presenta el rendimiento y configuraciones en una ejecución de la aplicación de cada uno de los clasificadores cuando se intenta maximizar las métricas individualmente. Cabe destacar que para cada *fold* se ejecuta un *grid search* buscando los hiperparámetros óptimos para la métrica seleccionada.

7.2.3.1 Para accuracy

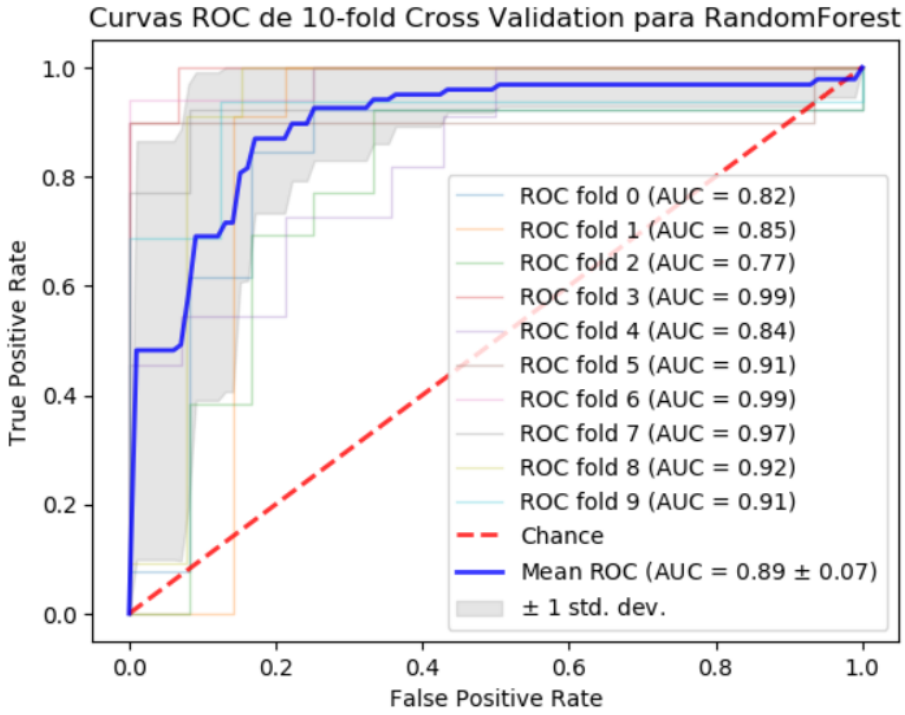
En primer lugar, para RandomForest se obtienen los siguientes hiperparámetros para cada *fold*:

Fold	criterion	max_depth	max_features	n_estimators
0	gini	6	auto	200
1	entropy	7	log2	200
2	gini	6	log2	500
3	gini	4	log2	500
4	entropy	7	sqrt	200
5	gini	6	auto	200
6	gini	5	sqrt	500
7	gini	5	auto	500
8	entropy	5	sqrt	200
9	gini	5	sqrt	200

37 Hiperparámetros Random Forest para accuracy

El *criterion* más usado es *gini*, la profundidad máxima mayoritaria es 5, con una media de 5.6. El número máximo de features está repartido entre las 3 funciones. El número de árboles empleados, *n_estimators*, está repartido casi igualitariamente entre 200 y 500.

Las curvas ROC para *random forest*:



38 Curvas ROC random forest para accuracy

Muestran una sensibilidad alta para la media de las curvas, dejando un área aceptable debajo de la curva, lo cual es un buen indicador.

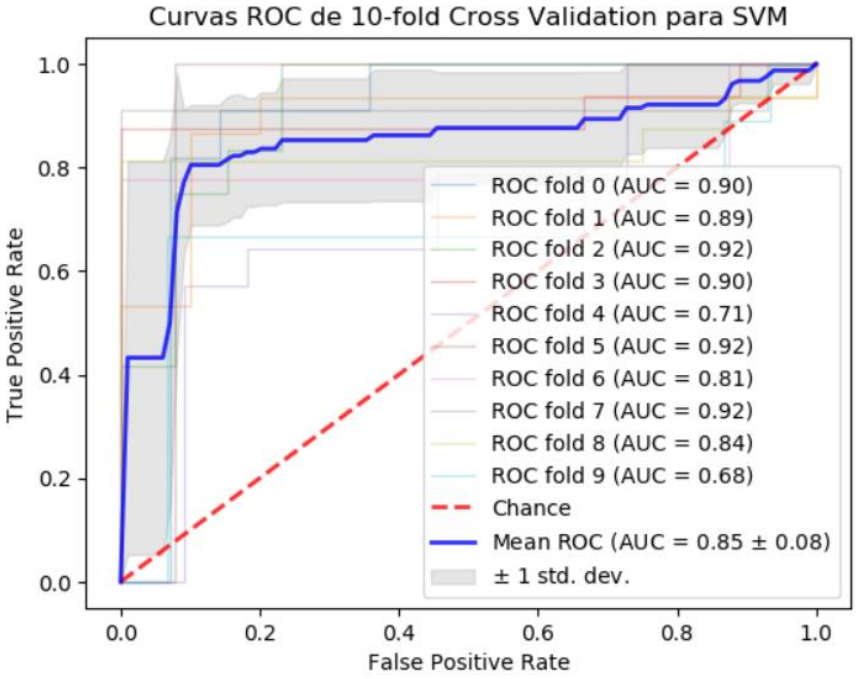
Para SVM, los hiperparámetros elegidos fueron:

fold	C	decision_function_shape	gamma	kernel	shrinking
0	0,5	ovo	1	rbf	VERDADERO
1	1	ovo	3	rbf	VERDADERO
2	0,5	ovo	3	rbf	VERDADERO
3	0,5	ovo	2	rbf	VERDADERO
4	0,25	ovo	2	rbf	VERDADERO
5	0,25	ovo	3	rbf	VERDADERO
6	0,25	ovo	1	rbf	VERDADERO
7	0,25	ovo	3	rbf	VERDADERO
8	0,25	ovo	3	rbf	VERDADERO
9	0,25	ovo	1	rbf	VERDADERO

Tabla 7 Hiperparámetros SVM para accuracy

El valor de C tiene una mayoría de 0,25. Para decision_function_shape el valor indiscutible es 'ovo' (one-versus-one). Destaca también la dominancia del kernel gaussiano *rbf*.

Las curvas ROC para SVM:



39 Curvas ROC SVM para Accuracy

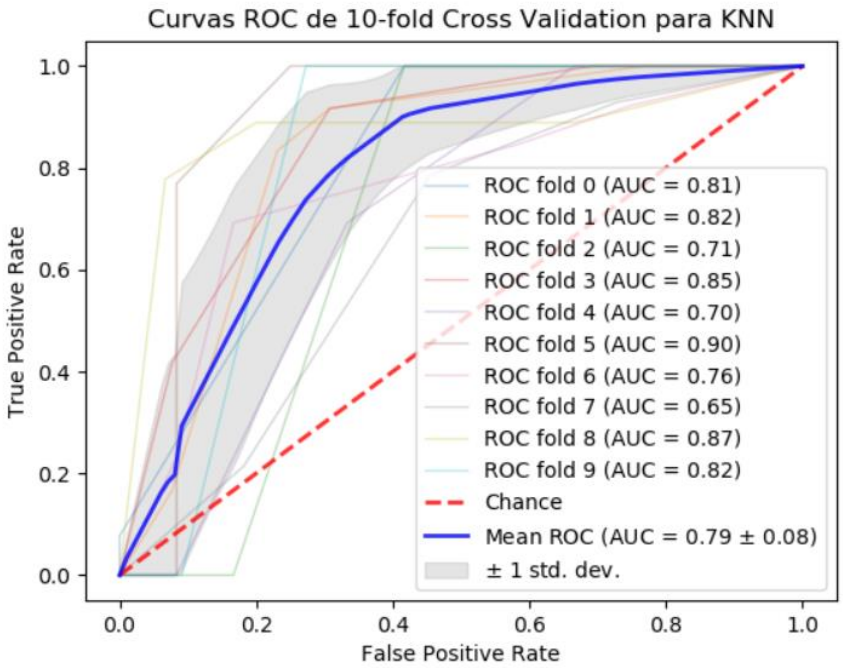
Esta gráfica muestra una buena sensibilidad debido a las áreas existentes bajo las curvas.

El siguiente algoritmo ejecutado es KNN, con una elección del hiperparámetro

fold	n_neighbors
0	3
1	4
2	3
3	3
4	4
5	4
6	4
7	3
8	4
9	3

Se eligen los valores 3 y 4 un número igual de veces.

En la curva ROC para KNN,



40 Curvas ROC KNN para accuracy

Se aprecia que la sensibilidad no es tan pronunciada como en los algoritmos anteriores, dejando un área menor bajo las curvas.

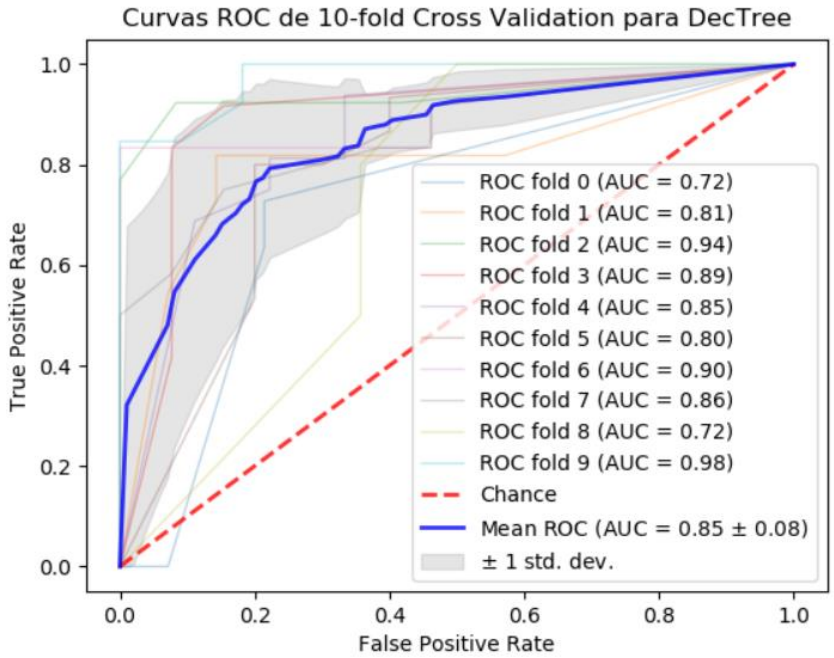
Por último, para un árbol de decisión,

fold	criterion	max_depth	min_samples_leaf
0	entropy	5	1
1	gini	5	10
2	entropy	20	10
3	entropy	5	5
4	entropy	5	10
5	gini	10	10
6	entropy	5	10
7	entropy	10	10
8	gini	15	5
9	gini	5	5

Tabla 8 Hiperparámetros árbol de decisión para accuracy

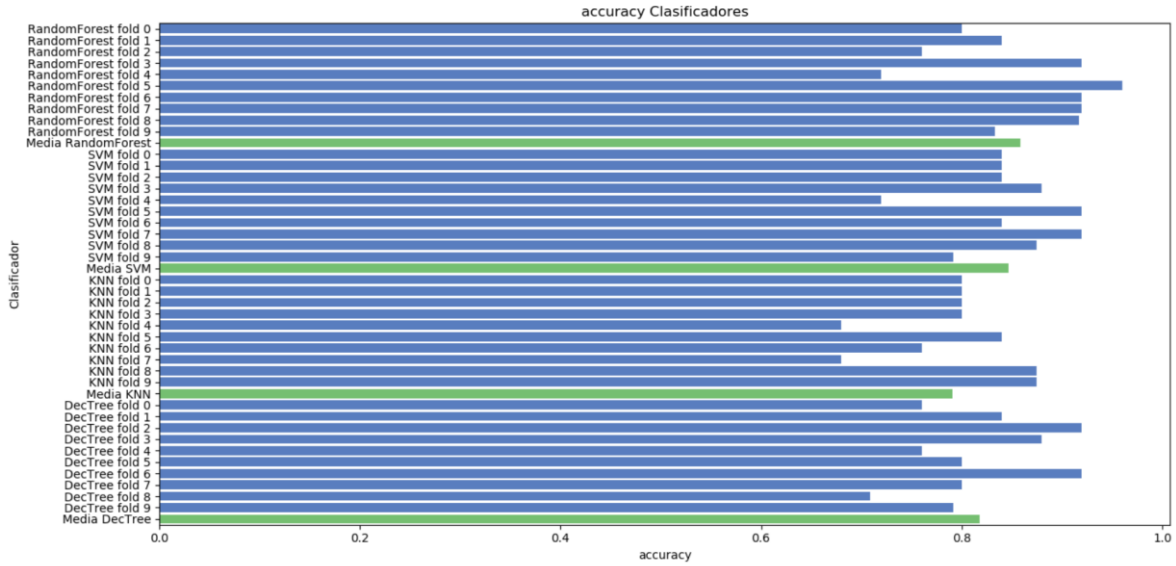
Entropy es el valor dominante en el parámetro *criterion*, la profundidad máxima del árbol tiene un valor mayoritario de 5. En cuanto al número mínimo de muestras necesario para considerar un nodo terminal, el valor predominante ha sido 10.

Las curvas ROC para el árbol de decisión,



41 Curvas ROC árbol de decisión para accuracy

La comparativa de todas las *accuracy* de cada *fold* y cada algoritmo,



42 Comparativa accuracy

Muestra una paridad entre *random forest* y *svm* en cabeza, teniendo *random forest* una media de *accuracy* más alta. KNN y el árbol de decisión muestran un valor medio más bajo que los otros dos clasificadores.

Las métricas en su totalidad fueron,

Clasificador	accuracy	precision	recall	f1
RandomForest fold 0	0,8	0,833333	0,769231	0,8
RandomForest fold 1	0,84	0,769231	0,909091	0,833333
RandomForest fold 2	0,76	0,733333	0,846154	0,785714
RandomForest fold 3	0,92	1	0,8	0,888889
RandomForest fold 4	0,72	0,642857	0,818182	0,72
RandomForest fold 5	0,96	1	0,9	0,947368
RandomForest fold 6	0,92	0,941176	0,941176	0,941176
RandomForest fold 7	0,92	0,923077	0,923077	0,923077
RandomForest fold 8	0,916667	0,846154	1	0,916667
RandomForest fold 9	0,833333	0,928571	0,8125	0,866667
Media RandomForest	0,859	0,861773	0,871941	0,862289
SVM fold 0	0,84	0,769231	0,909091	0,833333
SVM fold 1	0,84	0,866667	0,866667	0,866667
SVM fold 2	0,84	0,833333	0,833333	0,833333
SVM fold 3	0,88	1	0,8125	0,896552
SVM fold 4	0,72	0,888889	0,571429	0,695652
SVM fold 5	0,92	0,916667	0,916667	0,916667
SVM fold 6	0,84	0,777778	0,777778	0,777778
SVM fold 7	0,92	1	0,818182	0,9
SVM fold 8	0,875	1	0,8125	0,896552
SVM fold 9	0,791667	0,833333	0,555556	0,666667
Media SVM	0,846667	0,88859	0,78737	0,82832
KNN fold 0	0,8	0,722222	1	0,83871
KNN fold 1	0,8	0,769231	0,833333	0,8
KNN fold 2	0,8	0,722222	1	0,83871
KNN fold 3	0,8	0,733333	0,916667	0,814815
KNN fold 4	0,68	0,692308	0,692308	0,692308
KNN fold 5	0,84	0,909091	0,769231	0,833333
KNN fold 6	0,76	0,818182	0,692308	0,75
KNN fold 7	0,68	0,6875	0,785714	0,733333
KNN fold 8	0,875	0,875	0,777778	0,823529
KNN fold 9	0,875	0,8125	1	0,896552
Media KNN	0,791	0,774159	0,846734	0,802129
DecTree fold 0	0,76	0,727273	0,727273	0,727273
DecTree fold 1	0,84	0,818182	0,818182	0,818182
DecTree fold 2	0,92	0,923077	0,923077	0,923077
DecTree fold 3	0,88	0,909091	0,833333	0,869565
DecTree fold 4	0,76	0,857143	0,75	0,8
DecTree fold 5	0,8	0,857143	0,8	0,827586
DecTree fold 6	0,92	1	0,833333	0,909091
DecTree fold 7	0,8	0,818182	0,75	0,782609
DecTree fold 8	0,708333	0,615385	0,8	0,695652
DecTree fold 9	0,791667	0,722222	1	0,83871
Media DecTree	0,818	0,82477	0,82352	0,819174

Tabla 9 Métricas para accuracy maximizada

Para las métricas evaluadas (*accuracy*, *precisión*, *recall* y *f1*), *random forest* arroja mejores resultados.

7.2.3.2 Para precision.

Cuando se considera *precision* la métrica a maximizar, se obtienen los siguientes resultados.

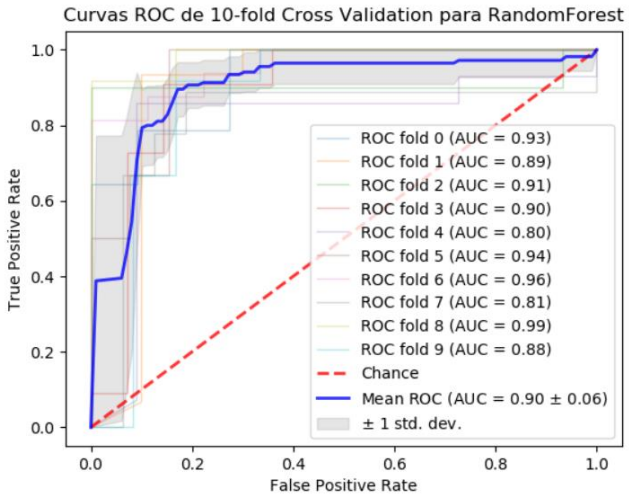
Para *random forest* los hiperparámetros ajustados fueron,

fold	criterion	max_depth	max_features	n_estimators
0	gini	7	auto	500
1	gini	4	log2	500
2	entropy	4	log2	500
3	gini	6	sqrt	500
4	entropy	4	sqrt	200
5	gini	8	log2	200
6	gini	6	sqrt	500
7	gini	7	auto	500
8	entropy	7	log2	200
9	gini	4	sqrt	200

Tabla 10 Hiperparámetros random forest para precision

Gini impurity es el criterio más usado para evaluar la calidad del *split*. En el resto de hiperparámetros los valores están repartidos sin una mayoría clara.

Las curvas ROC,



43 Curvas ROC random forest para precision

Muestran buenos resultados, dejando área por debajo de ellas.

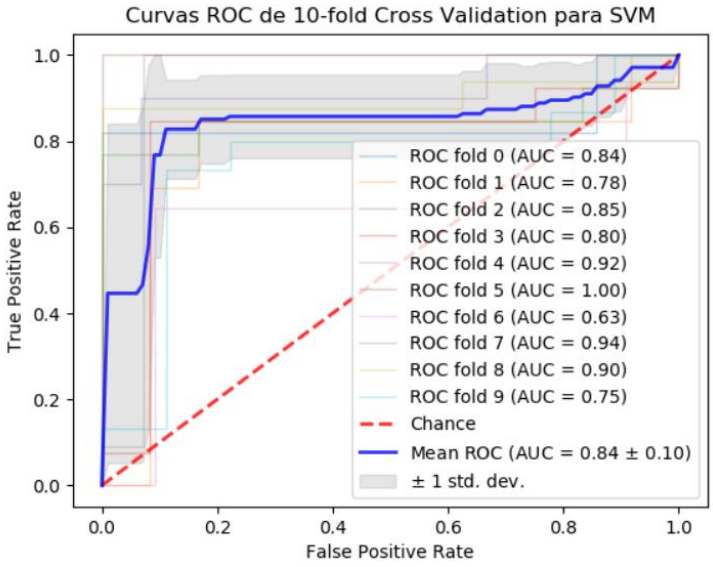
En el algoritmo SVM, los hiperparámetros fueron

fold	C	decision_function_shape	gamma	kernel	shrinking
0	0,25	ovo	3	rbf	VERDADERO
1	0,25	ovo	3	rbf	VERDADERO
2	0,25	ovo	1	rbf	VERDADERO
3	0,25	ovo	3	rbf	VERDADERO
4	0,25	ovo	3	rbf	VERDADERO
5	0,25	ovo	3	rbf	VERDADERO
6	0,25	ovo	3	rbf	VERDADERO
7	0,25	ovo	3	rbf	VERDADERO
8	0,25	ovo	3	rbf	VERDADERO
9	0,25	ovo	3	rbf	VERDADERO

Tabla 11Hiperparámetros SVM para precision.

Se aprecia la mayoría de la configuración.

Las curvas ROC,



44 Curvas ROC SVM para precision

Muestran un buen comportamiento.

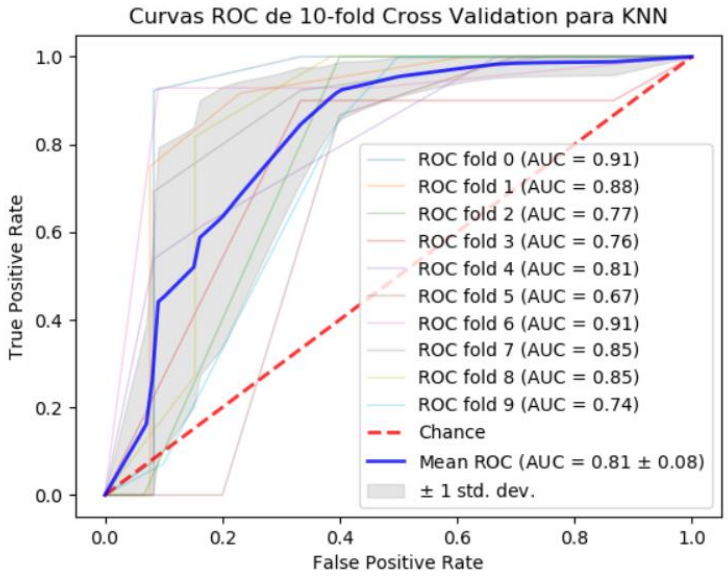
Para KNN,

fold	n_neighbors
0	4
1	4
2	4
3	4
4	4
5	4
6	4
7	4
8	4
9	4

Tabla 12 Hiperparámetros KNN para precision

Se eligen 4 para todos los folds como vecinos a visitar

Las curvas ROC,



45 Curvas ROC KNN para precision

Muestran un buen comportamiento, aunque la sensibilidad es menor que en los anteriores algoritmos.

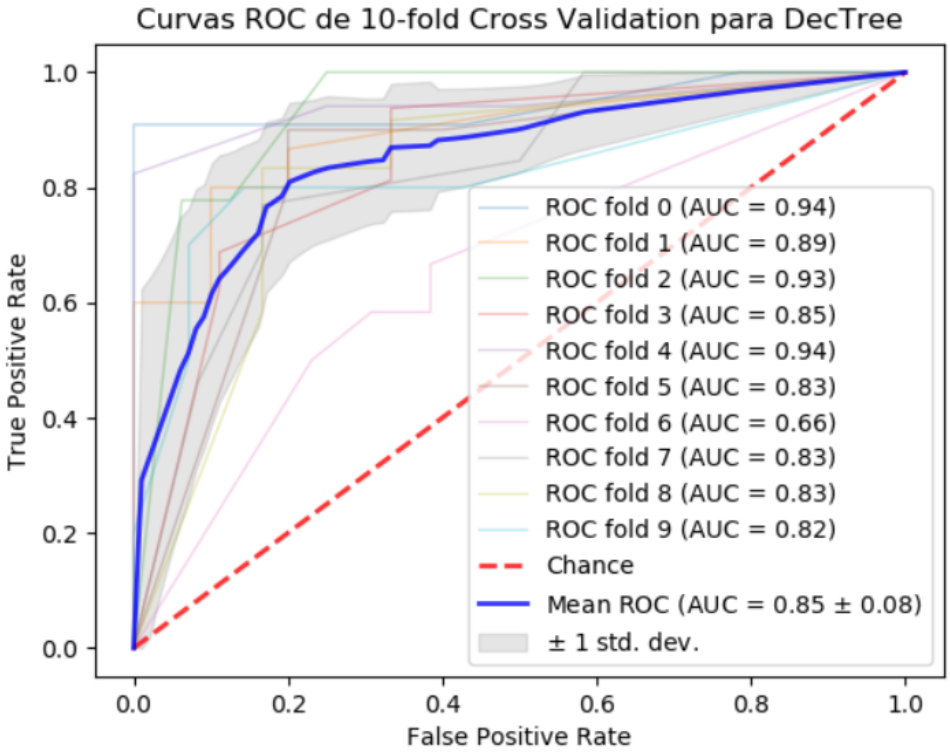
Para el árbol de decisión,

fold	criterion	max_depth	min_samples_leaf
0	gini	5	10
1	gini	15	5
2	gini	5	10
3	gini	20	5
4	gini	5	10
5	gini	20	5
6	entropy	20	10
7	gini	5	10
8	entropy	10	5
9	gini	5	10

Tabla 13 Hiperparámetros árbol de decisión para precision

Gini sigue siendo el criterio más elegido

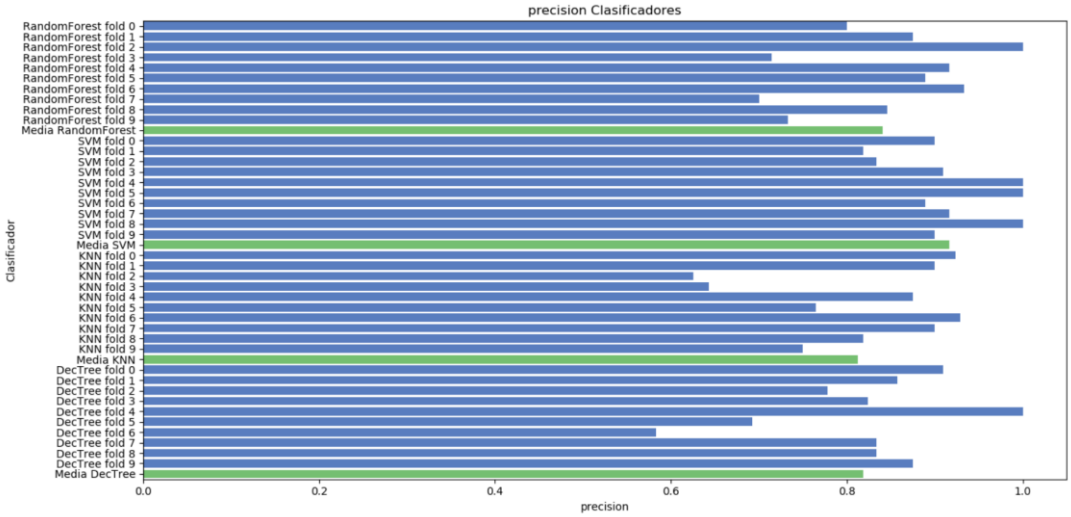
Las curvas ROC para el árbol de decisión,



46 Curvas ROC árbol de decisión para precision

La sensibilidad es alta.

La comparativa de la métrica *precisión* para todos los algoritmos



47 Comparativa según precision

En este caso, el valor medio más alto de *precision* es el obtenido por SVM, seguido de *random forest*. KNN y el árbol de decisión tienen unos resultados algo más bajos.

La totalidad de las métricas fueron:

Clasificador	accuracy	precision	recall	f1
RandomForest fold 0	0,8	0,8	0,857143	0,827586
RandomForest fold 1	0,88	0,875	0,933333	0,903226
RandomForest fold 2	0,92	1	0,8	0,888889
RandomForest fold 3	0,8	0,714286	0,909091	0,8
RandomForest fold 4	0,84	0,916667	0,785714	0,846154
RandomForest fold 5	0,8	0,888889	0,666667	0,761905
RandomForest fold 6	0,88	0,933333	0,875	0,903226
RandomForest fold 7	0,8	0,7	0,777778	0,736842
RandomForest fold 8	0,875	0,846154	0,916667	0,88
RandomForest fold 9	0,791667	0,733333	0,916667	0,814815
Media RandomForest	0,838667	0,840766	0,843806	0,836264
SVM fold 0	0,88	0,9	0,818182	0,857143
SVM fold 1	0,76	0,818182	0,692308	0,75
SVM fold 2	0,8	0,833333	0,769231	0,8
SVM fold 3	0,84	0,909091	0,769231	0,833333
SVM fold 4	0,88	1	0,7	0,823529
SVM fold 5	1	1	1	1
SVM fold 6	0,72	0,888889	0,571429	0,695652
SVM fold 7	0,96	0,916667	1	0,956522
SVM fold 8	0,916667	1	0,875	0,933333
SVM fold 9	0,708333	0,9	0,6	0,72
Media SVM	0,8465	0,916616	0,779538	0,836951
KNN fold 0	0,92	0,923077	0,923077	0,923077
KNN fold 1	0,84	0,9	0,75	0,818182
KNN fold 2	0,76	0,625	1	0,769231
KNN fold 3	0,76	0,642857	0,9	0,75
KNN fold 4	0,72	0,875	0,538462	0,666667
KNN fold 5	0,76	0,764706	0,866667	0,8125
KNN fold 6	0,92	0,928571	0,928571	0,928571
KNN fold 7	0,8	0,9	0,692308	0,782609
KNN fold 8	0,833333	0,818182	0,818182	0,818182
KNN fold 9	0,75	0,75	0,857143	0,8
Media KNN	0,806333	0,812739	0,827441	0,806902
DecTree fold 0	0,92	0,909091	0,909091	0,909091
DecTree fold 1	0,8	0,857143	0,8	0,827586
DecTree fold 2	0,84	0,777778	0,777778	0,777778
DecTree fold 3	0,8	0,823529	0,875	0,848485
DecTree fold 4	0,88	1	0,823529	0,903226
DecTree fold 5	0,8	0,692308	0,9	0,782609
DecTree fold 6	0,6	0,583333	0,583333	0,583333
DecTree fold 7	0,8	0,833333	0,769231	0,8
DecTree fold 8	0,833333	0,833333	0,833333	0,833333
DecTree fold 9	0,833333	0,875	0,7	0,777778
Media DecTree	0,810667	0,818485	0,79713	0,804322

Tabla 14 Métricas para precision maximizada.

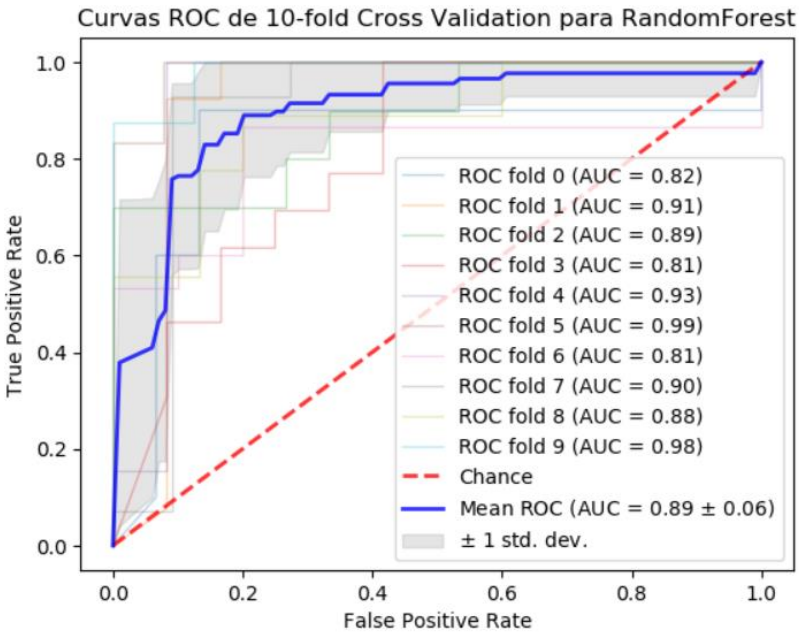
7.2.3.3 Para recall.

Los hiperparámetros ajustados para *random forest* fueron,

fold	criterion	max_depth	max_features	n_estimators
0	entropy	4	auto	200
1	entropy	4	sqrt	200
2	gini	4	sqrt	200
3	gini	5	log2	200
4	gini	4	log2	200
5	entropy	4	sqrt	200
6	gini	5	sqrt	200
7	entropy	5	sqrt	500
8	entropy	5	auto	500
9	gini	4	auto	200

Tabla 15 Hiperparámetros random forest para recall

Hay un equilibrio entre los parámetros ajustados, salvo en el número de árboles utilizados, *n_estimators*, donde se han utilizado una mayoría de 200.



48 Curvas ROC random forest para recall

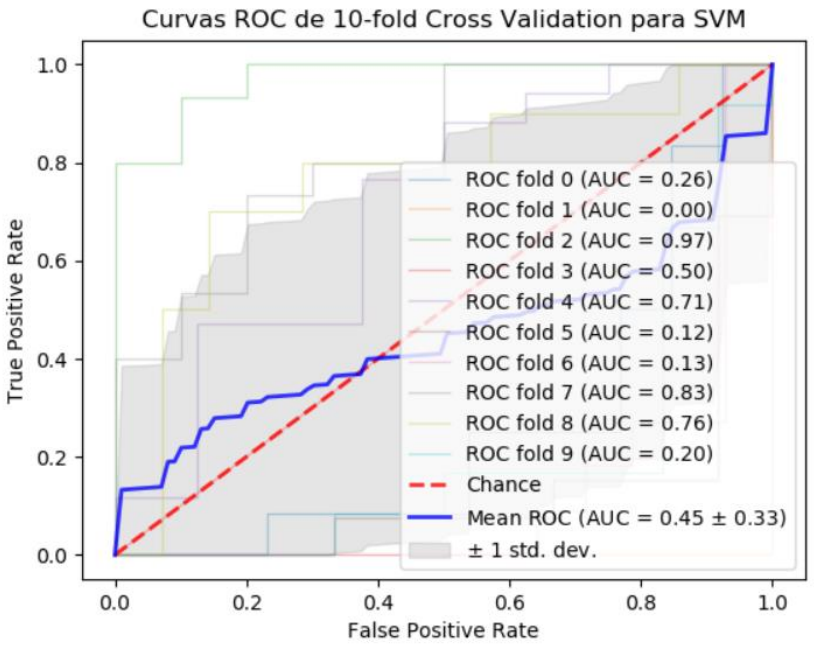
Continúa el buen comportamiento de la sensibilidad para *random forest*.

La relación de los parámetros ajustados para SVM es

fold	C	decision_function_shape	gamma	kernel	shrinking
0	1	ovo	auto	poly	VERDADERO
1	1	ovo	auto	poly	VERDADERO
2	0,5	ovo	auto	rbf	VERDADERO
3	1	ovo	auto	poly	VERDADERO
4	0,5	ovo	auto	rbf	VERDADERO
5	1	ovo	auto	poly	VERDADERO
6	1	ovo	auto	poly	VERDADERO
7	0,75	ovo	auto	rbf	VERDADERO
8	1	ovo	auto	poly	VERDADERO
9	1	ovo	auto	poly	VERDADERO

Tabla 16 Hiperparámetros SVM para recall

Las curvas ROC para SVM muestran el fenómeno de *precision-recall Trade-off*



49 *Curvas ROC SVM para recall*

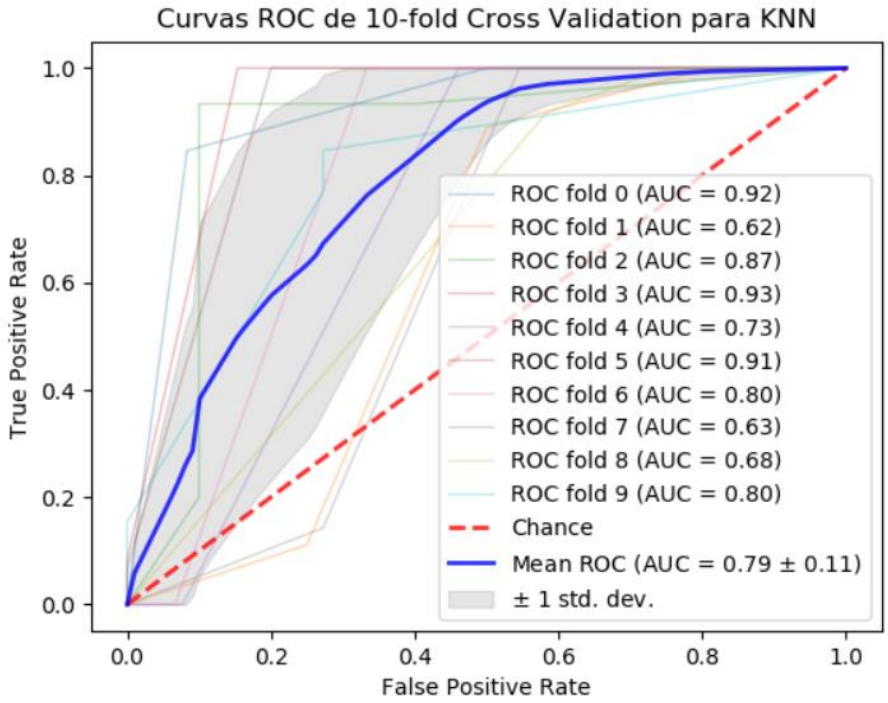
A medida que aumentamos el r*ecall*, *precision* disminuye y viceversa.

Para KNN los ajustes en base al *recall*

folds	n_neighbors
0	3
1	3
2	3
3	3
4	3
5	3
6	3
7	3
8	3
9	3

Tabla 17 Hiperparámetros para KNN para recall

Teniendo como curvas ROC,



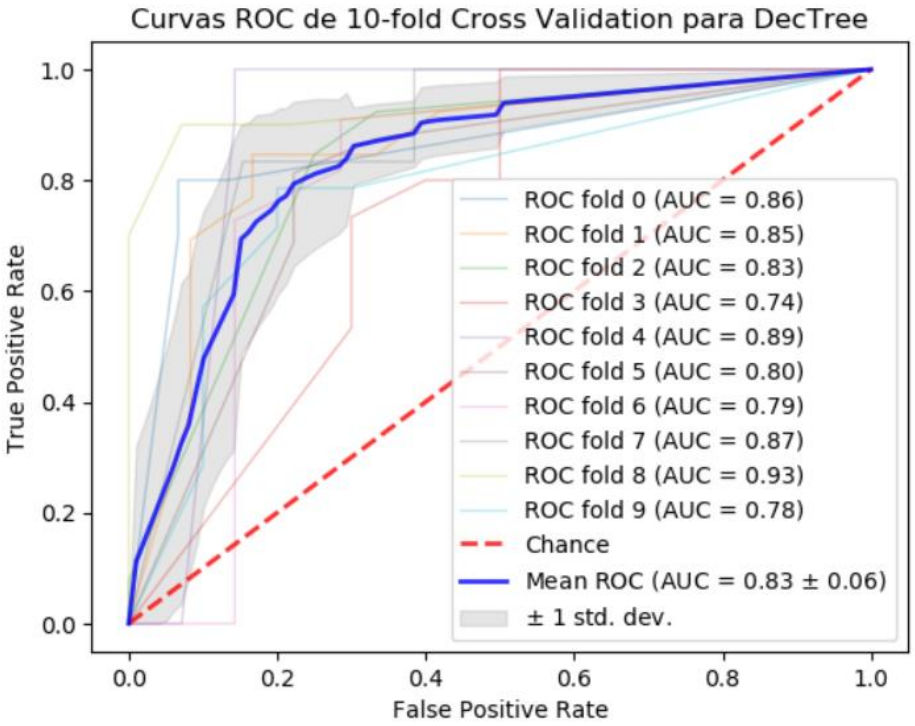
50 Curvas ROC KNN para recall

Para el árbol de decisión

fold	criterion	max_depth	min_samples_leaf
0	entropy	5	5
1	gini	20	5
2	gini	5	1
3	entropy	5	5
4	entropy	5	1
5	gini	5	10
6	entropy	5	1
7	gini	5	5
8	entropy	5	10
9	entropy	5	5

Tabla 18 Hiperparámetros decision tree para recall

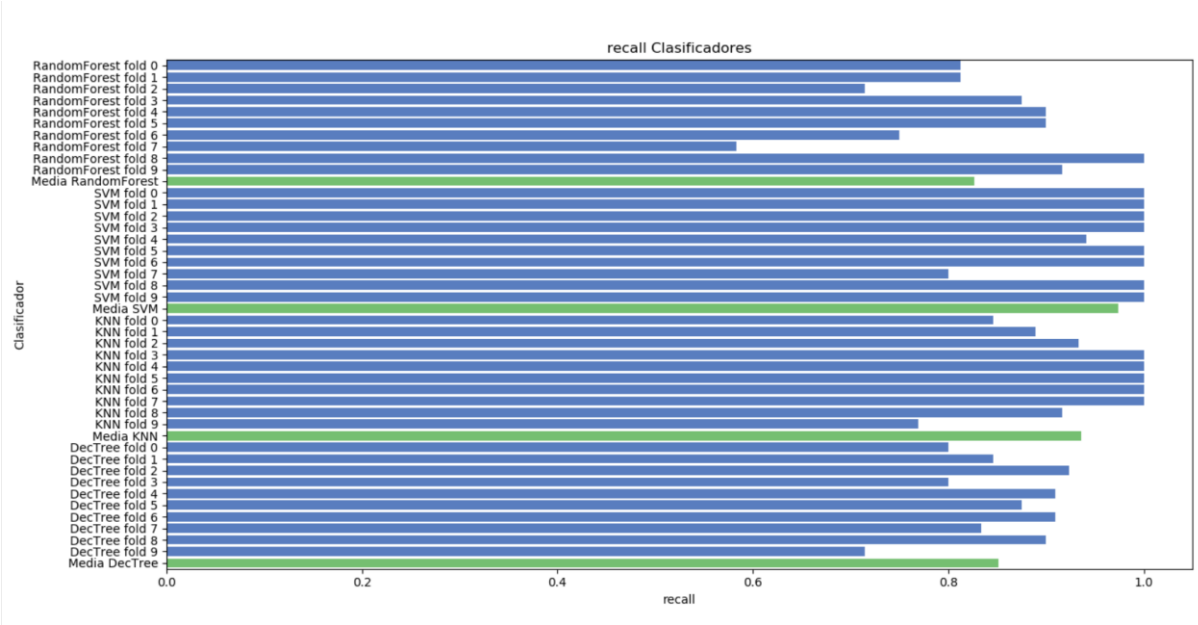
Las curvas ROC,



51 Curvas ROC decision tree para recall

Muestran un buen comportamiento.

La comparativa de los resultados de *recall*,



52 Comparativas clasificadores para recall

La totalidad de los resultados para cada *fold* muestra lo expuesto anteriormente en el clasificador SVM en cuanto a *precision-recall trade off,* dónde los valores extremadamente altos de *recall* coinciden con los valores bajos de *precision.*

Clasificador	accuracy	precision	recall	f1
RandomForest fold 0	0,8	0,866666667	0,8125	0,838709677
RandomForest fold 1	0,76	0,8125	0,8125	0,8125
RandomForest fold 2	0,76	0,833333333	0,71428571	0,769230769
RandomForest fold 3	0,8	0,636363636	0,875	0,736842105
RandomForest fold 4	0,84	0,75	0,9	0,818181818
RandomForest fold 5	0,92	0,9	0,9	0,9
RandomForest fold 6	0,8	0,818181818	0,75	0,782608696
RandomForest fold 7	0,76	0,875	0,58333333	0,7
RandomForest fold 8	1	1	1	1
RandomForest fold 9	0,83333333	0,785714286	0,91666667	0,846153846
Media RandomForest	0,82733333	0,827775974	0,82642857	0,820422691
SVM fold 0	0,48	0,48	1	0,648648649
SVM fold 1	0,4	0,4	1	0,571428571
SVM fold 2	0,92	0,882352941	1	0,9375
SVM fold 3	0,4	0,4	1	0,571428571
SVM fold 4	0,76	0,761904762	0,94117647	0,842105263
SVM fold 5	0,52	0,52	1	0,684210526
SVM fold 6	0,44	0,44	1	0,611111111
SVM fold 7	0,72	0,75	0,8	0,774193548
SVM fold 8	0,41666667	0,416666667	1	0,588235294
SVM fold 9	0,5	0,5	1	0,666666667
Media SVM	0,55566667	0,555092437	0,97411765	0,68955282
KNN fold 0	0,88	0,916666667	0,84615385	0,88
KNN fold 1	0,64	0,5	0,88888889	0,64
KNN fold 2	0,92	0,933333333	0,93333333	0,933333333
KNN fold 3	0,92	0,857142857	1	0,923076923
KNN fold 4	0,76	0,666666667	1	0,8
KNN fold 5	0,92	0,882352941	1	0,9375
KNN fold 6	0,8	0,666666667	1	0,8
KNN fold 7	0,76	0,7	1	0,823529412
KNN fold 8	0,66666667	0,611111111	0,91666667	0,733333333
KNN fold 9	0,75	0,769230769	0,76923077	0,769230769
Media KNN	0,80166667	0,750317101	0,93542735	0,824000377
DecTree fold 0	0,88	0,888888889	0,8	0,842105263
DecTree fold 1	0,84	0,846153846	0,84615385	0,846153846
DecTree fold 2	0,8	0,75	0,92307692	0,827586207
DecTree fold 3	0,72	0,75	0,8	0,774193548
DecTree fold 4	0,88	0,833333333	0,90909091	0,869565217
DecTree fold 5	0,8	0,823529412	0,875	0,848484848
DecTree fold 6	0,8	0,714285714	0,90909091	0,8
DecTree fold 7	0,72	0,666666667	0,83333333	0,740740741
DecTree fold 8	0,91666667	0,9	0,9	0,9
DecTree fold 9	0,75	0,833333333	0,71428571	0,769230769
Media DecTree	0,81066667	0,800619119	0,85100316	0,821806044

Tabla 19 Metricas clasificadores para recall maximizada

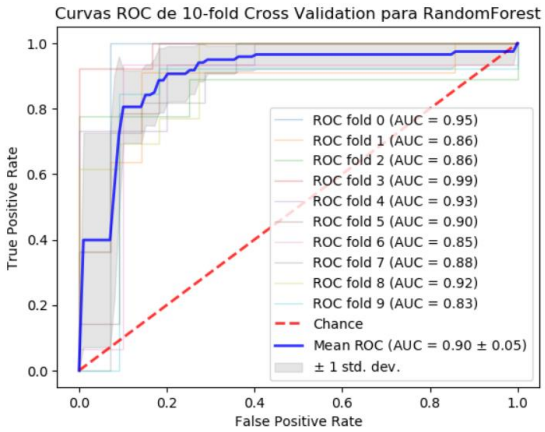
7.2.3.4 Para F1-score

Para *random forest*, se obtiene

fold	Criterion	max_depth	max_features	n_estimators
0	Gini	7	log2	500
1	Entropy	4	sqrt	200
2	Entropy	7	auto	500
3	Gini	5	sqrt	500
4	Gini	4	auto	500
5	Entropy	4	log2	200
6	Gini	5	auto	200
7	Gini	5	sqrt	200
8	Entropy	4	sqrt	500
9	Entropy	6	sqrt	200

Tabla 20 hiperparámetros random forest para F1

Con unas curvas ROC,



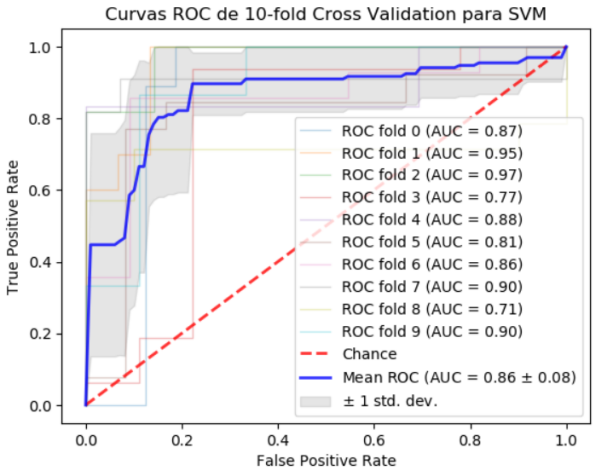
53 Curvas ROC random forest para F1

Para SVM,

fold	C	decision_function_shape	gamma	kernel	shrinking
0	0,75	ovo	3	rbf	VERDADERO
1	1	ovo	3	rbf	VERDADERO
2	1	ovo	3	rbf	VERDADERO
3	1	ovo	3	rbf	VERDADERO
4	1	ovo	3	rbf	VERDADERO
5	0,5	ovo	3	rbf	VERDADERO
6	1	ovo	2	rbf	VERDADERO
7	1	ovo	3	rbf	VERDADERO
8	1	ovo	3	rbf	VERDADERO
9	1	ovo	3	rbf	VERDADERO

Tabla 21 Hiperparametros SVM para f1

Las curvas ROC para SVM,



54 Curvas ROC SVM para f1

Para KNN,

fold	n_neighbors
0	3
1	3
2	3
3	3
4	3
5	3
6	3
7	3
8	4
9	3

Tabla 22 Hiperparámetros KNN para f1

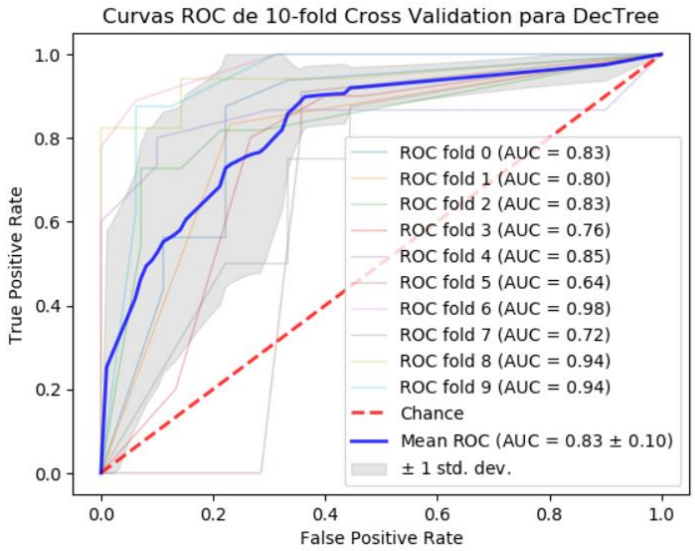
Los hiperparámetros de *decision trees* fueron,

	criterion	max_depth	min_samples_leaf
0	gini	20	5
1	gini	100	1
2	entropy	10	10
3	entropy	5	5
4	gini	20	10
5	gini	5	1
6	entropy	5	10
7	gini	100	5

8	entropy	5	5
9	gini	5	10

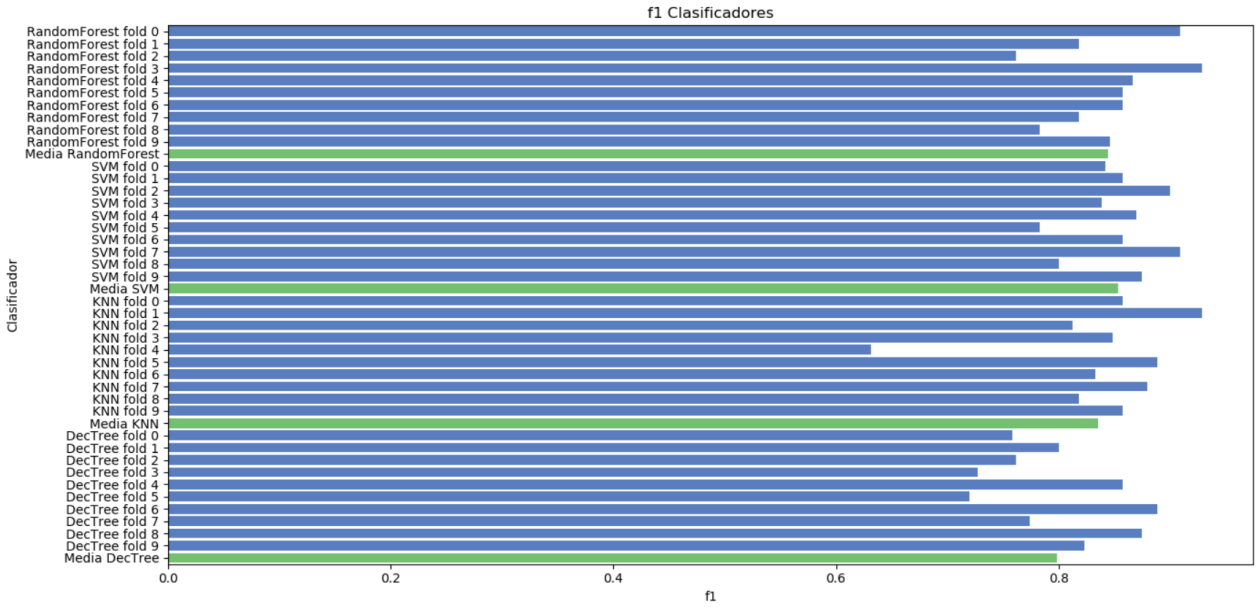
Tabla 23 Hiperparámetros árboles de decisión para f1

Con unas curvas ROC,



55 Curvas ROC árboles de decisión para F1

El resumen de los resultados obtenidos,



56 Comparativa F1

Clasificador	accuracy	precision	recall	f1
RandomForest fold 0	0,92	0,909091	0,909091	0,909091
RandomForest fold 1	0,84	0,818182	0,818182	0,818182
RandomForest fold 2	0,8	0,666667	0,888889	0,761905
RandomForest fold 3	0,92	0,866667	1	0,928571
RandomForest fold 4	0,84	0,866667	0,866667	0,866667
RandomForest fold 5	0,84	0,857143	0,857143	0,857143
RandomForest fold 6	0,84	0,923077	0,8	0,857143
RandomForest fold 7	0,84	0,818182	0,818182	0,818182
RandomForest fold 8	0,791667	0,9	0,692308	0,782609
RandomForest fold 9	0,833333	0,846154	0,846154	0,846154
Media RandomForest	0,8465	0,847183	0,849661	0,844565
SVM fold 0	0,88	0,8	0,888889	0,842105
SVM fold 1	0,88	0,818182	0,9	0,857143
SVM fold 2	0,92	1	0,818182	0,9
SVM fold 3	0,8	0,866667	0,8125	0,83871
SVM fold 4	0,88	0,909091	0,833333	0,869565
SVM fold 5	0,8	0,9	0,692308	0,782609
SVM fold 6	0,84	0,857143	0,857143	0,857143
SVM fold 7	0,92	0,909091	0,909091	0,909091
SVM fold 8	0,791667	0,909091	0,714286	0,8
SVM fold 9	0,833333	0,823529	0,933333	0,875
Media SVM	0,8545	0,879279	0,835906	0,853137
KNN fold 0	0,84	0,923077	0,8	0,857143
KNN fold 1	0,92	0,866667	1	0,928571
KNN fold 2	0,76	0,722222	0,928571	0,8125
KNN fold 3	0,8	0,736842	1	0,848485
KNN fold 4	0,72	0,461538	1	0,631579
KNN fold 5	0,84	0,8	1	0,888889
KNN fold 6	0,84	0,714286	1	0,833333
KNN fold 7	0,88	0,916667	0,846154	0,88
KNN fold 8	0,833333	0,9	0,75	0,818182
KNN fold 9	0,833333	0,75	1	0,857143
Media KNN	0,826667	0,77913	0,932473	0,835582
DecTree fold 0	0,72	0,846154	0,6875	0,758621
DecTree fold 1	0,8	0,769231	0,833333	0,8
DecTree fold 2	0,8	0,8	0,727273	0,761905
DecTree fold 3	0,76	0,666667	0,8	0,727273
DecTree fold 4	0,84	0,923077	0,8	0,857143
DecTree fold 5	0,72	0,642857	0,818182	0,72
DecTree fold 6	0,92	0,888889	0,888889	0,888889
DecTree fold 7	0,72	0,8	0,75	0,774194
DecTree fold 8	0,833333	0,933333	0,823529	0,875

DecTree fold 9	0,875	0,777778	0,875	0,823529
Media DecTree	0,798833	0,804799	0,800371	0,798655

Tabla 24 Métricas F1

7.3 Conclusiones.

A la luz de los resultados obtenidos, entre los diferentes clasificadores el que ha tenido mejor rendimiento medio en las diferentes métricas ha sido *random forest* seguido de cerca por *support vector machines*. Por ejemplo, maximizando la métrica *accuracy* se obtuvo:

Clasificador	accuracy	precision	recall	f1
Media RandomForest	0,859	0,861773	0,871941	0,862289
Media SVM	0,846667	0,88859	0,78737	0,82832
Media KNN	0,791	0,774159	0,846734	0,802129
Media DecTree	0,818	0,82477	0,82352	0,819174

Cabe destacar que para un predictor como este minimizar el error tipo II (falsos negativos) es una tarea importante. Cuando se ha fijado el *target_score* a maximizar a *recall* se han conseguido resultados positivos:

Fold	precision	recall
SVM fold 0	0,48	1
SVM fold 1	0,4	1
SVM fold 2	0,882352941	1
SVM fold 3	0,4	1
SVM fold 4	0,761904762	0,94117647
SVM fold 5	0,52	1
SVM fold 6	0,44	1
SVM fold 7	0,75	0,8
SVM fold 8	0,416666667	1
SVM fold 9	0,5	1
KNN fold 0	0,916666667	0,84615385
KNN fold 1	0,5	0,88888889
KNN fold 2	0,933333333	0,93333333
KNN fold 3	0,857142857	1
KNN fold 4	0,666666667	1
KNN fold 5	0,882352941	1
KNN fold 6	0,666666667	1
KNN fold 7	0,7	1
KNN fold 8	0,611111111	0,91666667
KNN fold 9	0,769230769	0,76923077

Estos valores positivos han sido a costa de unos valores de *precision* en algunos casos pobres. Este fenómeno se denomina como *Precision-Recall Trade-off*, cuanto más aumente el *recall* más pobre es *precision* y viceversa.

También se aprecia la diversidad de los hiperparámetros en cada uno de los folds. Esta variabilidad de los hiperparámetros ajustados en las diferentes ejecuciones demuestran la importancia de esta tarea, debiendo explorar las opciones para obtener el mejor rendimiento.

El uso de *grid search* mejora los resultados y permite explorar todas las combinaciones posibles de hiperparámetros de una forma cómoda. Pero tiene una contrapartida en cuanto al tiempo de ejecución, al probar todas las combinaciones posibles haciendo un uso interno de *Cross-Validation*.

El rendimiento aceptable, aunque no suficiente, de los clasificadores plantea la duda de cómo se comportarían con una mayor cantidad de secuencias. Este incremento de secuencias conllevaría consecuentemente una mayor capacidad de cómputo.

Se podrían también afinar las *features* generadas para que recogieran, por ejemplo, información sobre los dominios. También sería interesante generar *features* que recogieran datos en profundidad sobre la estructura secundaria y terciaria.

Se ha pretendido que la elección de *features* no caractericen de manera particular a las quinasas, para poder expandir estas técnicas a otras familias y probar su desempeño.

Como líneas futuras, se podría explorar el uso de *Deep Learning* y redes neuronales. Estudiar el uso de métodos de agrupamiento de algoritmos, como *Ada Boost*, sería una vía interesante

La usabilidad y *look and feel* de la herramienta admiten mejoras sustanciales. Mejoras en estos dos factores posibilitarían la presentación y comparación de los datos de manera más ágil y clara.

Como bagaje personal, se ha adquirido soltura en el lenguaje Python y en librerías importantes para la Bioinformática y el *Data Science*. Poder aplicar los conocimientos adquiridos durante el máster para un problema en concreto ha sido también una satisfacción personal.

8 Bibliografía

(s.f.). Obtenido de GitHub: https://github.com/

Baldi, P. &. (2001). Bioinformatics: The Machine Learning Approach. . London,England: The MIT Press.

Des Higgins, F. S. (s.f.). ClustalW2. Obtenido de http://www.clustal.org/clustal2/

Dubchak, I. e. (1995). Prediction of protein folding class using global description of amino acid sequence. Proceedings of the National Academy of Sciences, 8700–8704.

Edgar, R. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res., 1792-1797.

Edgar, R. C. (s.f.). Muscle. Obtenido de https://www.drive5.com/muscle/

Kirk, M. (2017). Thoughtful Machine Learning with Python. O'Reilly Media,.

Lantz, B. (2013). Machine Learning with R. Packt Publishing.

Larkin MA, B. G. (2007). Clustal W and Clustal X version 2.0. Bioinformatics, 2947-2948.

Lubanovic, B. (2015). Introducing Python. O'Reilly Media.

Raschka, S. (2015). Python Machine Learning. Packt.

Seltzer WK, A. C. (1989 Apr). Muscle glycerol kinase in Duchenne dystrophy and glycerol kinase deficiency. Muscle Nerve, 12(4), 307-313. doi:https://doi.org/10.1002/mus.880120409

Varios. (s.f.). Towardsdatascience. Obtenido de https://towardsdatascience.com/

Varios. (s.f.). Wikipedia. Obtenido de https://en.wikipedia.org/wiki/Kinase

Varios. (s.f.). Wikipedia. Obtenido de https://en.wikipedia.org/wiki/Sequence_alignment