Predicción de la toxicidad y de la actividad antimicrobiana a partir de la secuencia aminoacídica

Ramón Mariño Solís
Máster de Bioinformática y Bioestadística
Bioinformática Farmacéutica

Melchor Sánchez Martínez
Profesor Responsable

24/05/2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	Predicción de la toxicidad y de la actividad antimicrobiana a partir de la secuencia aminoacídica
Nombre del autor:	*Ramón Mariño Solís*
Nombre del consultor/a:	*Melchor Sánchez Martínez*
Nombre del PRA:	-
Fecha de entrega (mm/aaaa):	05/2017
Titulación:	*Bioinformática y Bioestadística*
Área del Trabajo Final:	Bioinformática Farmacéutica
Idioma del trabajo:	*Castellano*
Palabras clave	*Máximo 3 palabras clave, validadas por el director del trabajo (dadas por los estudiantes o en base a listados, tesauros, etc.)*

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

Vimos la necesidad de crear una herramienta informática que pudiera predecir la actividad antimicrobiana y si pudiera causar toxicidad de un péptido. ¿Por qué era importante? Porque hay un creciente problema con la resistencia bacteriana a los antibióticos, que conlleva que cada vez hay menos moléculas que podrán utilizarse contra las infecciones problemáticas y, por lo tanto, es importante crear o descubrir nuevos fármacos con una fuerte actividad antibacteriana.

Para ello, creamos dos modelos SVM, uno que clasifique entre péptidos tóxicos y no tóxicos, y otro que puede detectar si un péptido tiene actividad antimicrobiana, o no. Primero obtuvimos la información de las secuencias en la base de datos SwissProt. Éstas fueron utilizadas para calcular un grupo de variables explicativas que aportaran información al algoritmo para entrenar el modelo y hacer la mejor separación por hiperplanos.

Utilizamos el lenguaje Python para crear algunos programas que crean las diferentes bases de datos, que contienen la toda la información, y a través de Biopython calculamos las variables. Luego, usando el paquete scikit-learn, hacemos el código para crear, entrenar y probar los modelos.

Se obtienen malos resultados en el clasificador tóxico, pero la actividad antimicrobiana se puede predecir con un 93% de precisión.

Abstract (in English, 250 words or less):

We saw a necessity to create a informatic tool which it could predict the antimicrobial activity and if it could cause a toxic effect. Why it was important? Because there is problem with the bacterial resistance to antibiotics. Actually, there are less molecules that it could be used in the problematic infections and it is hardly important to create or discover new drugs with a strong antibacterial activity.

To do this, we thought to create two SVM models, one which classify between toxic and no toxic peptides, and other which can detect if a peptide has antimicrobial activity. First, we obtain the information in UniProt database, but only with the SwissProt sequences. We utilized this data to calculate a group of explicative variables that could bring the best information to the algorithm to train it and make the best hyperplane separation.

We used the Python language to create some programs which create the different databases and Biopython to calculate the variables. Then, using the scikit-learn package, we make the code to create, train and test the models.

We obtain bad results in the toxic classifier, but the antimicrobial activity could be predicted with a 93% of accuracy.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Los antimicrobianos son moléculas que eliminan o inhiben el crecimiento de microorganismos, como hongos, bacterias o parásitos. Son ampliamente utilizados para combatir infecciones producidas por estos organismos y, desde el descubrimiento de los antibióticos, la mortalidad debido a estos patógenos ha disminuido hasta ser un problema menor en países desarrollados. Sin embargo, en los últimos años se ha registrado un aumento de la resistencia a los antimicrobianos (RAM), un fenómeno por el cual un microorganismo deja de ser afectado por este tipo de agentes, al que antes era sensible, y que surge por la mutación del microorganismo o por la adquisición de un gen que se la confiere (OMS).

En la última década la preocupación por la RAM ha ido en aumento hasta convertirse en un riesgo a escala global. El incremento de este fenómeno puede llegar a inutilizar los fármacos de uso más común contra enfermedades infecciosas comunes e, incluso, los medicamentos empleados como último recurso pueden no ser suficientes para salvar vidas. A principios de 2016 se conocía la muerte de una persona debido a una bacteria súper-resistente en EEUU, un caso que llama la atención ya que se probaron hasta 26 fármacos diferentes que, finalmente, no tuvieron el efecto deseado (De Benito, E., 2017).

Este suceso abrió la puerta a la preocupación fuera de la comunidad científico-sanitaria, ya que una de las grandes potencias del mundo era incapaz de derrotar a la pequeña *Klebsiella pneumoniae*. El mismo artículo da pistas sobre el problema, la infección original se produjo en la India y se conocía que la bacteria tenía una mutación que le proporcionaba multi-resistencia, motivo por el cual se pudo rastrear, pero no tratar. La globalización es por tanto una vía de transmisión de bacterias a través de todo el mundo, dejando patente que no será solo una crisis localizada en países en fase de desarrollo.

La RAM es un proceso natural que ocurre con el tiempo y si hay una presión selectiva, es un fenómeno evolutivo por lo que necesita ambas cosas. Cuando se usan estos medicamentos se eliminan a los que son sensibles, pero quedan aquellos que tienen resistencia, o mayor tolerancia, y que podrán transmitir esta cualidad a la descendencia. Un uso inadecuado de los fármacos aumenta la farmacorresistencia, tanto por un exceso en su uso, como si lo hacemos de forma insuficiente. Se acelera de esta forma la aparición de lo que hoy ya se conocen como súper-bacterias.

No es un problema que se pueda achacar a un país o región del planeta. Este problema ha alcanzado ya el estatus de problema de sanidad a nivel mundial. La movilización y la globalización permiten que, como en el caso comentado anterior, una bacteria de la India produzca una muerte en EEUU. Enfermedades infecciosas hasta ahora controladas en el primer mundo pueden volverse incontrolables si los medicamentos que se usaban hasta el momento se vuelven ineficaces.

La Organización Mundial de la Salud tiene claro la relevancia de este problema y, en 2016, realizó una importante campaña de comunicación y exposición de las causas, problemas y medidas de actuación. Sin embargo, las medidas son sobre todo de aumento de la información y la publicación de distintas recomendaciones a seguir para mejorar la situación. Este organismo, que muchas veces es usado como una fuente meramente consultiva, no puede exigir a las empresas y/o estados que presionen a los sectores, a los profesionales y, mucho menos, a una población que muchas veces no tiene un acceso a una medicación adecuada.

Ilustración 1. *Causas del aumento de la resistencia a los antibióticos. Fuente: Organización Mundial de la Salud (OMS).*

La búsqueda de alternativas a los tratamientos habituales para las enfermedades infecciosas se convierte así en una necesidad que, si no se cambian los patrones de conducta actuales, se hará cada vez más urgente. Para incentivar que el sector privado se involucre en la solución a esta problemática se ha puesto en marcha una iniciativa que premia económicamente al laboratorio que consiga nuevos antibióticos.

En medio de esta situación ha comenzado a aumentar la popularidad de los conocidos como péptidos antimicrobianos (AMPs, por su nombre en inglés). Estas moléculas son parte del sistema inmune de los seres vivos, no solo de mamíferos sino de muchos otros grupos de animales, como los hexápodos. Su presencia es conspicua, debido a que cada grupo de organismos genera sus propios péptidos para combatir enfermedades bacterianas, víricas, fúngicas e, incluso, distintos tumores.

Se postulan entonces como una alternativa al uso masivo de antibióticos, no como sus sustitutos, para permitir un menor uso de estos fármacos. Este tipo de péptidos tienen una naturaleza muy variable y tienen ventajas en su uso clínico, pero también hay que ser cautos si se piensa en su incorporación al uso habitual en el entorno médico. Sin embargo, abren la puerta a una familia de compuestos con usos tan variados como organismos que los genera, incluyendo al propio *Homo sapiens*, y que actúan con mecanismos variopintos y sobre dianas igualmente variadas.

Pese a esta amplia variedad, tanto a nivel de secuencia aminoacídica como de estructura, este tipo de péptidos tienen propiedades comunes. La más característica es su carga, son mayoritariamente catiónicos en condiciones fisiológicas, por lo que también son conocidos como *Cationic Antimicrobial Peptides* (CAMPs). Otra principal cualidad es su fracción de residuos hidrofóbicos y un carácter anfipático (Soraya et al., 2016). El mecanismo más común de actuación contra microorganismos es el ataque a la membrana plasmática, a la cual se unen aprovechando tanto su carga como su fracción hidrofóbica. Consiguen desestabilizarla y, debido a la vulnerabilidad que eso ocasiona, acaban provocando la lisis celular y la muerte. Nuevos estudios sugieren que también podrían tener dianas intracelulares, para lo cual necesitan atravesar la bicapa lipídica (Xiong et al., 2005).

Como una última variable que afecta a este entorno que estamos describiendo tenemos las nuevas tecnologías. La bioinformática ha entrado en los últimos tiempos como la solución a muchos de los grandes desafíos a los que se enfrenta la biología. La secuenciación masiva de genomas, permitida gracias al abaratamiento de costes y al aumento de la velocidad del proceso, genera un volumen de información que solo puede ser procesado con ordenadores. La informática se integra como un nuevo recurso para la interpretación y el estudio de nuevas moléculas, muchas de las cuales solo son conocidas por su secuencia, y que son muy complicadas de obtener por técnicas tradicionales de aislamiento.

Si cada día podemos predecir nuevas proteínas a partir de genomas secuenciados de distintos organismos, y conseguimos también una mayor facilidad del tratamiento de esta información, se pueden originar herramientas muy potentes a la hora de anticipar resultados en los laboratorios. Los ensayos clínicos que realizan las empresas farmacéuticas son procesos largos y costosos, en los cuales miles de compuestos son analizados y valorados para su uso clínico. Muchas veces a penas unos pocos consiguen llegar a las últimas fases, lo que implica un porcentaje de éxito bajo que, sin embargo, puede ser mejorado creando programas de predicción.

La búsqueda de nuevos antibióticos, y de nuevos AMPs, potentes y no tóxicos, puede ser mejorada con la bioinformática. Puede anticiparse que péptidos tienen una mayor probabilidad de ser viables para combatir bacterias súper-resistentes en medicina. Es entonces cuando, aprovechando las bases de datos existentes hoy día, se pueden utilizar para predecir las características de proteínas e, incluso, ver si las distintas variantes de una misma molécula presentan una actividad diferente.

Actualmente se busca solucionar los problemas derivados de la resistencia a los antibióticos a través del diagnóstico; analizando el microorganismo y descubriendo sus resistencias. Pero es solo un parche, porque el proceso es continuo y, si no se consiguen alternativas, los antibióticos tradicionales finalmente dejarán de tener efecto.

En este trabajo buscamos crear una herramienta bioinformática que sea capaz de anticipar un efecto antimicrobiano de una secuencia de aminoácidos. Su utilidad

radica en que en poco tiempo podemos analizar una gran cantidad de secuencias de proteínas y descartar rápidamente aquellas que no se cataloguen como activas. De esta forma se pueden centrar los esfuerzos en moléculas con una mayor probabilidad de acabar en el mercado, y/o en el uso clínico contra enfermedades infecciosas.

1.2. Objetivos del Trabajo

- Desarrollo de un modelo que permita predecir la actividad antimicrobiana a partir de la secuencia de aminoácidos de un péptido.

- Desarrollo de un modelo que permita anticipar la toxicidad de un péptido a partir de su estructura secundaria.

- Crear una herramienta bioinformática que combine ambos modelos.

1.3. Enfoque y método seguido

La predicción de la actividad antimicrobiana es un tema que ha ido cobrando interés en los últimos años dentro de la comunidad científica ligada a la bioinformática. Han surgido varias herramientas propuestas por distintos grupos y con diferentes estrategias a la hora de cómo clasificar la actividad y qué tipo de variables explicativas son las que aportan más información.

Un ejemplo es AMPA (Torrent *et al., 2012*), incorporado en el servidor web T-Coffee y que se basa, al igual que dicha plataforma, en la predicción de zonas activas en proteínas antimicrobianas a través del alineamiento de secuencias. El portal antiBP (Lata *et al., 2007)* emplean *Support Vector Machine* (SVM), *Quantitative Matrices* (QM) y *Artificial Neural Network* (ANN) para catalogar los péptidos usando como variables la posición y porcentaje de aminoácidos, alcanzando una precisión cercana al 92%. Sin embargo, su base de datos no alcanza los 500 péptidos.

En este trabajo se plantearán una serie de variables que, tras un estudio de la naturaleza y características de los AMPs, se cree que serán buenas predictoras de la actividad. Estas serán siempre numéricas, en caso de ser de carácter cualitativo se

transformarán a números, y describirán aspectos físico-químicos deducibles a partir de su secuencia. La elección de estas variables se discute ampliamente en el capítulo 2.6

Este planteamiento se considera el adecuado para cumplir, no solo los objetivos planteados en este trabajo, sino que también permitirá al alumno alcanzar competencias previstas durante la realización de un trabajo final de máster (TFM). Gracias a enfrentarse a un problema bioinformático desde cero se ve obligado a buscar y explorar diferentes ramas usadas en la bioinformática, como el *machine learning,* y distintos lenguajes de programación, como *Python* y R, para generar un modelo funcional con una aplicabilidad real. También aumenta las competencias profesionales del estudiante al aportarle los conocimientos para poder crear algoritmos de clasificación en el futuro, los cuales son herramientas muy versátiles y con gran aplicabilidad.

1.4. Planificación del Trabajo

El trabajo se planifica para llevar a cabo las distintas tareas que son necesarias para construir un modelo con todas las garantías. Los recursos necesarios se limitan a un ordenador portátil personal y las 325 horas asignadas para la realización del TFM. Al ser un trabajo individual no se contempla la participación de ninguna otra persona en el proyecto.

	Nombre	Duracion	Inicio	Terminado
1	Definición del proyecto	6,5 days	1/03/17 8:00	9/03/17 13:00
2	Planificación Temporal	3,5 days	6/03/17 8:00	9/03/17 13:00
3	Redacción de la PEC1	3 days	10/03/17 8:00	14/03/17 17:00
4	Búsqueda bibliográfica	12,75 days?	15/03/17 8:00	31/03/17 15:00
5	Creación de las bases de datos	17 days?	23/03/17 8:00	14/04/17 17:00
6	Base de datos	0 days	14/04/17 8:00	14/04/17 8:00
7	Creación del modelo SVM	25 days?	17/04/17 8:00	19/05/17 17:00
8	Modelo SVM (Entregable)	0 days	19/05/17 8:00	19/05/17 8:00
9	Testeo del modelo SVM	4,5 days?	22/05/17 8:00	26/05/17 13:00
10	Informe del testeo	0 days	29/05/17 8:00	29/05/17 8:00
11	Análisis de mejora del algoritmo/SVM	5 days?	29/05/17 13:00	5/06/17 13:00
12	Informe del análisis de mejora(Entregable)	1 day?	5/06/17 13:00	6/06/17 13:00
13	PEC1	11 days?	1/03/17 8:00	15/03/17 17:00
14	PEC2	15 days?	16/03/17 8:00	5/04/17 17:00
15	PEC3	25 days?	6/04/17 8:00	10/05/17 17:00
16	Memoria	10 days	11/05/17 8:00	24/05/17 17:00
17	Presentación visual	0 days	10/05/17 8:00	10/05/17 8:00
18	Presentación final y memoria	10 days?	5/06/17 13:00	19/06/17 13:00

Ilustración 2. *Lista de las tareas asignadas al TFM. La primera columna corresponde con el título de la tarea, la segunda es el tiempo para su realización en días y las dos últimas corresponden a la fecha de inicio y finalización respectivamente.*

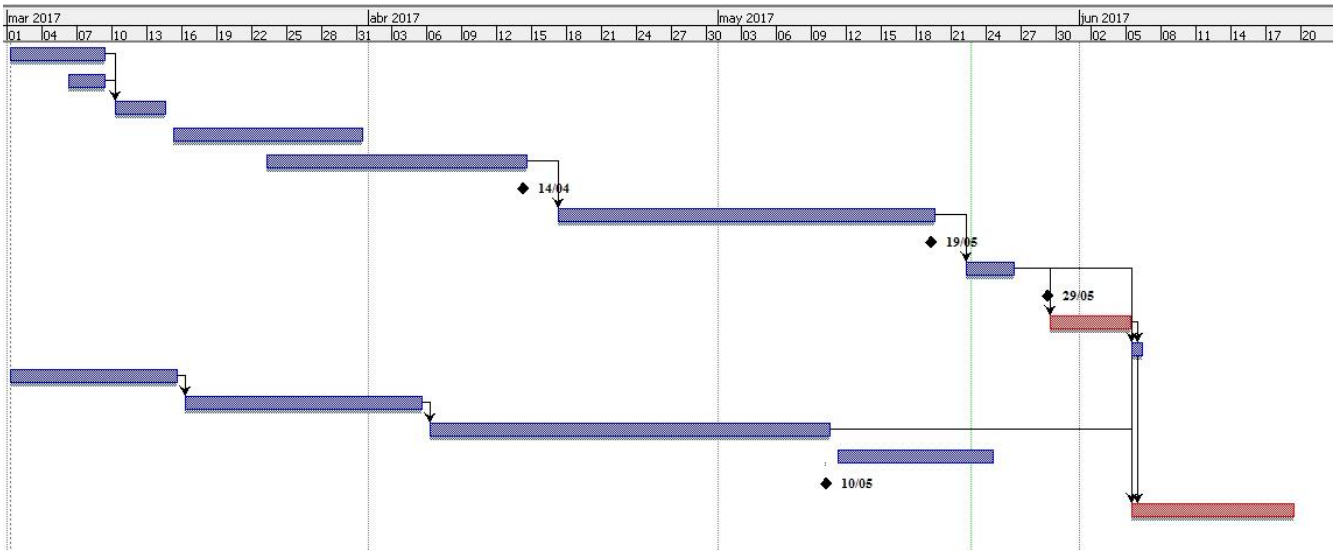


***Ilustración 3**. Diagrama de Gant de la planificación temporal de las tareas necesarias para realizar el TFM.*

1.5. Breve sumario de productos obtenidos

Al finalizar el trabajo se han obtenido varios productos. Destacan los programas escritos en Python, ya que se pueden reutilizar más adelante, y un modelo optimizado que era uno de los objetivos. A continuación, se listan los entregables generados:

- Database.py
- Svmclassification.py
- Distintas bases de datos (incluyendo las de testeo y entrenamiento).
- ampModel.pkl
- ampModelPoly.pkl
- ampModelRbf.pkl

1.6. Breve descripción de los otros capítulos de la memoria

Los próximos apartados describen más en profundidad los apartados anteriores, los cuales sirven como un resumen de la justificación y realización del trabajo. Para poder dar información lo más comprensible posible se estructuran los siguientes capítulos siguiendo el orden de los apartados anteriores, como se describe a continuación:

2.1. Estudio de los AMPs

Este capítulo continúa con una explicación más extensa sobre los péptidos antimicrobianos, aportando más datos sobre su composición, clasificación y estructura. Sirve de avanzadilla para entender la selección de las variables explicativas, que se aborda más adelante.

2.2. Investigación de los algoritmos de selección y del lenguaje de programación

Como en el caso anterior, este capítulo explora más en profundidad las diferentes estrategias y modelos que se podrían desarrollar para llevar a cabo este trabajo. Un análisis de los algoritmos de selección más utilizados, así como de los distintos lenguajes de programación que se emplearon, ayudará a entender las decisiones finales y aportará una mayor perspectiva a la hora de buscar mejoras.

2.3. Selección de las variables explicativas para el modelo

Conseguir un porcentaje elevado de precisión está ligado a la selección de las variables explicativas. Si éstas no tienen relación con la propiedad, o con la clase, que queremos clasificar no tendremos buenos resultados en ningún caso.

2.4. Creación de las bases de datos

La recolección de la información, la obtención de las variables y, si es necesario, su transformación, cristaliza en la construcción de las bases de datos. Estas serán luego utilizadas para entrenar y testar el modelo.

2.5. Generación de los modelos predictivos

Se ahonda en los programas que generan los SVM de la toxicidad y la actividad antimicrobiana. Se repasa el código de los distintos módulos dando una explicación de cada uno.

2.6. Testeo del modelo

Después de generar el modelo y entrenarlo se hace una comprobación de su capacidad predictiva.

2.7. Productos

Se listan los productos generados durante la elaboración del trabajo. También se acompaña una breve descripción de su función.

2.8. Resultados

Se presentan los resultados obtenidos en los capítulos anteriores, incluyendo los programas y las bases de datos, entrando en detalle.

2. Cuerpo del trabajo

2.1. Estudio de los AMPs.

2.1.1. Origen

Los AMPs son moléculas efectoras del sistema inmune innato presentes en casi todos los organismos, desde bacterias hasta mamíferos. Se sintetizan de forma constitutiva, o inducible, en diferentes tipos celulares y tejidos. Habitualmente provienen de genes ribosomales, aunque existen péptidos codificados en el núcleo o que son metabolitos secundarios.

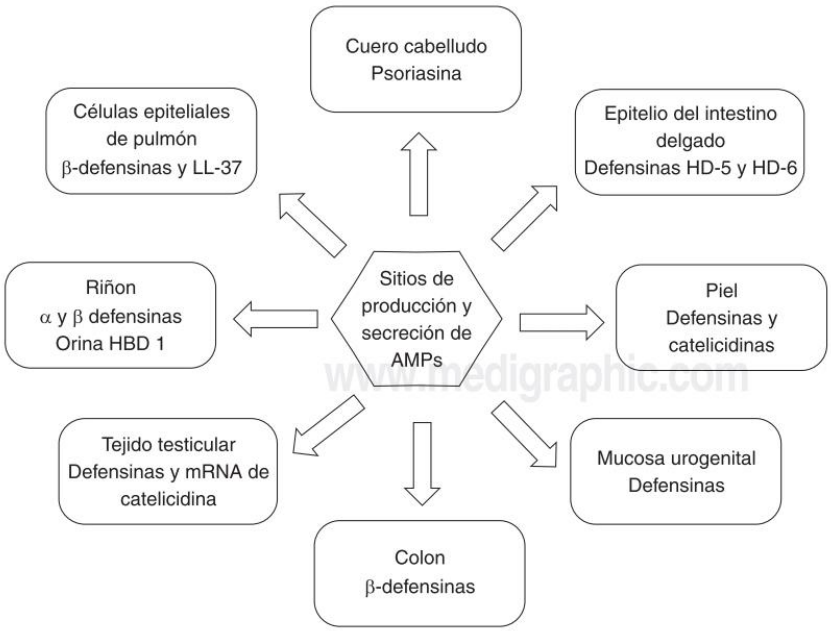


Ilustración 4.*Distintos tejidos humanos donde se producen péptidos antimicrobianos. Se añaden también algunos ejemplos de este tipo de péptidos.*

2.1.2. Características físicas

Los péptidos antimicrobianos son muy conocidos por tener una carga positiva en condiciones fisiológicas. Esta carga suele oscilar entre +2 y +9 y se debe a que hay una mayor cantidad de residuos de arginina y lisina, cargados positivamente, que residuos de ácido aspártico y ácido glutámico, que poseen carga negativa.

Como se describió en la introducción, este tipo de proteínas tienen una fracción hidrofóbica importante, concretamente, suele haber cerca de un 30% de aminoácidos con esta característica. Esto confiere a la molécula un carácter anfipático, es decir, que tiene carácter hidrofóbico e hidrofílico al mismo tiempo.

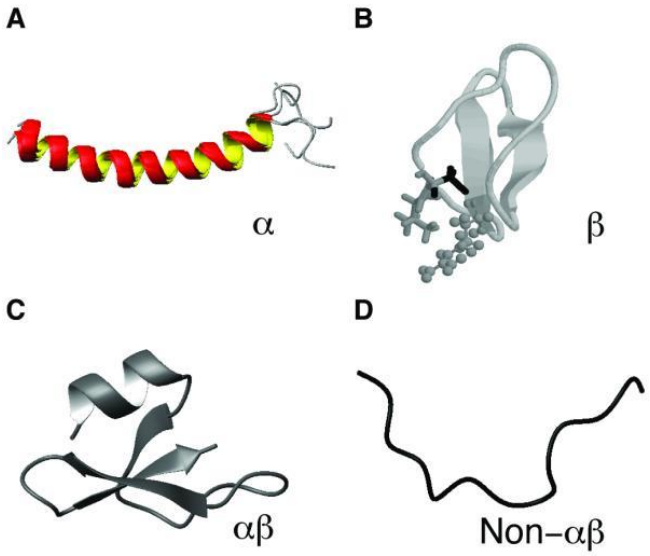


Ilustración 5. *Principales estructuras secundarias de los péptidos antimicrobianos. Esta característica se ha usado ampliamente como método de clasificación junto con otras características como la carga. Fuente: Wang, G., 2015.*

2.1.3. Clasificación

Hay diversos criterios para clasificar este tipo de péptidos como su origen, su función biológica, sus mecanismos de biosíntesis, las propiedades del péptido… En este trabajo nos ceñiremos a la clasificación por la estructura secundaria aportada por Guangshum Wang (Wang, G., 2015). Esta clasificación se basa en los patrones de conexión de las cadenas polipeptídicas e incluye un total de cuatro clases.

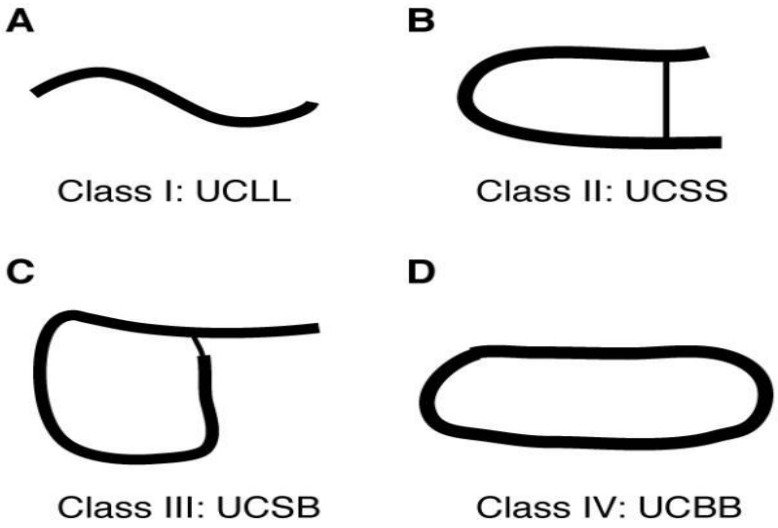


Ilustración 6. *Clasificación de universal propuesta por G. Wang. La clase I engloba a los péptidos lineales, la clase II reúne a aquellos que tienen enlaces químicos que unen los extremos de la cadena. En la clase III las cadenas tienen un enlace con el esqueleto principal de la proteína y la clase IV abarca a los AMPs en los que se unen el extremo carbonilo y el extremo amino, proporcionando una estructura circular continua. Fuente: Want. G, 2015.*

Esta clasificación busca unificar los criterios para un grupo muy amplio de compuestos. Esta enorme diversidad complica la caracterización y la homogenización que, después, puede permitir una mejor compresión de esta familia y una mayor facilidad para su alteración.

2.1.4. Actividades principales

La familia de los péptidos antimicrobianos es conocida por tener un amplio número de actividades biológicas, destacando su actividad contra microorganismos. La diversidad de este tipo de moléculas permite tener un gran número de objetivos y tampoco es extraño que una misma proteína tenga más de un tipo de actividad.

A continuación, se enumeran los distintos tipos de actividades descritas, se comenta brevemente el mecanismo de acción y se menciona algún ejemplo:

- **Act. Antibacteriana:** Las características físico-químicas comunes de estos péptidos provocan que interaccionen con la membrana bacteriana y acaba por desestabilizarla. Se han descrito muchos modelos para explicar este proceso, como el modelo de hoyo de polilla, el modelo de agregación o el

modelo del poro toroidal. La base de datos APD3 tiene muchos ejemplos de este tipo de AMPs.

- **Act. Antifúngica:** Algunos tienen la capacidad de eliminar hongos ya que, como en el caso anterior, pueden desestabilizar su pared, aunque también pueden interferir en la síntesis proteica. Como detalle, los AMPs de origen vegetal con esta actividad son ricos en aminoácidos polares y neutros, además de que no se han encontrado dominios que se puedan vincular con este tipo de actividad. Los derivados de lactoferrinas, como el P18, o defensinas de plantas.
- **Act. Antiviral:** Una de las funciones principales más interesantes que tienen estos elementos del sistema inmune innato es la capacidad de combatir virus. Las distintas estrategias que se han investigado se centran en el bloqueo de la entrada a la célula, por ejemplo, uniéndose a glucoproteínas virales o bloqueando la interacción con el heparán sulfato (una glucosamina sulfato muy relevante para la unión viral). También pueden evitar la diseminación de los virus a nuevas células y unirse a la envoltura viral impidiendo que actúe de forma normal. Un ejemplo sería la defensina humana LL-37.
- **Act. Antiparasitaria:** La maganina2, un péptido antiprotozoa, actúa de forma similar, destruye las paredes de estos microorganismos e incluso es capaz de atacar a los huevos impidiendo su expansión.
- **Act. Inmunomoduladora:** Algunos tienen la capacidad de controlar la expresión ciertos genes en monocitos y células epiteliales. Sobre otras células inmunes tienen capacidad quimiotáctica, y puede provocar la inducción de citoquinas y la diferenciación celular promoviendo la angiogénesis, vinculada a la curación de heridas y el ataque a infecciones. Un ejemplo muy estudiado es la catelicidina LL37.

2.1.5. Resistencia a los péptidos antimicrobianos

La fuente original de la necesidad que queremos solventar es el problema de la resistencia a los antibióticos. El uso inadecuado de este tipo de sustancias genera un aumento de las cepas resistentes, ya que la presión producida elimina

a las bacterias susceptibles y estas son las que siguen replicándose. Si se quiere que los AMPs puedan ayudar a atajar este problema es lógico que se indague sobre si no se repetirá el mismo error de nuevo.

Los mecanismos generales para la adquisición de esta tolerancia pueden ser intrínsecos, los factores involucrados estaban ya presentes en el microorganismo; o adaptativos, los factores deben ser activados (debe existir presión selectiva). Es cierto que existe la posibilidad de que el uso reiterado de AMPs acabe causando resistencia, hay que recordar que es un proceso evolutivo natural. Sin embargo, varios estudios sugieren que esta probabilidad es menor que con los antibióticos de uso común (Castañeda *et al., 2009;* Tellez *et al., 2010).*

Normalmente, son propiedades bacterianas las que permiten adquirir la resistencia, ya que se vincula a la carga de la membrana, la estructura de los componentes de la membrana externa, la composición lipídica… Incluso hay mecanismos que, como con los antibióticos, ayudan a adquirir protección frente a estas sustancias. Un ejemplo de estos últimos es la segregación de enzimas que destruyen los AMPs, la existencia de mecanismos de transporte de flujo y transporte al exterior celular…

Estas estrategias son generalidades y cada bacteria tiene su propia forma de adaptarse al ambiente y a los cambios que en él se producen. Los péptidos antimicrobianos son parte del sistema inmune innato y llevan controlando infecciones desde hace millones de años, es decir, que las especies patógenas se han ido adaptando para poder ser más eficaces evitándolas. Si siguen siendo efectivas es porque han sido capaces de seguir funcionando pese a estos cambios.

2.1.6. Uso terapéutico

El objetivo final del estudio de los péptidos antimicrobianos es conseguir su viabilidad en el uso terapéutico, lo cual implica que sean seguros y eficaces. Los factores más importantes son la estabilidad, la toxicidad y la inmunogenicidad en el huésped.

Los AMPs tienen la desventaja de necesitar una elevada concentración para conseguir resultados similares a los antibióticos, lo cual causa toxicidad.

Para mejorar su aplicabilidad se han diseñado péptoides sintéticos, derivados de los que se encuentran en la naturaleza, que mejoraban la actividad de los originales permitiendo usar una menor cantidad. Esto abrió la vía al desarrollo de nuevos agentes antibacterianos relacionados con esta familia de compuestos.

La estabilidad se relaciona con la acción de proteasas que pueden degradar a los AMPs e inutilizarlos. Este tipo de enzimas están presentes en el torrente sanguíneo y el sistema gastrointestinal, limitando una aplicación directa por estas vías. Además, la unión a otras proteínas, por ejemplo del sistema inmune, puede tener el mismo efecto. Se han encontrado péptidos con actividad en membranas eucariotas, aunque mucho menor que la que presentan para bacterias. Estos problemas han llevado a que se usen de forma tópica, evitando así la degradación, pero el futuro está de nuevo en la creación de derivados resistentes a este tipo de proteasas (Tellez *et al.,* 2010).

2.2. Investigación de los algoritmos de selección y del lenguaje de programación.

Este trabajo ataja un problema biológico a través de la informática, concretamente aprovecha el *machine learning* y la minería de datos. Ambas disciplinas pueden funcionar en ámbitos muy distintos al planteado aquí, ya que la naturaleza de los datos con los que se trabaja no es relevante a la hora de su aplicabilidad. Sin embargo, nunca se debe perder de vista que si no existe un conocimiento previo de con qué se trabaja, no se puede construir un modelo, ni elegir un algoritmo, que sea apropiado.

La minería de datos es una ciencia computacional que busca patrones a lo largo de enormes conjuntos de datos. Para ello se vale de otras ciencias como la estadística, la inteligencia artificial o, el *machine learning*. Esta última disciplina fue definida por Arthur Samuel, en 1959, como aportar la habilidad de aprender a los ordenadores sin haberlos programado de forma explícita previamente.

En conjunto, forman una poderosa herramienta que permite analizar las enormes bases de datos biológicas e inferir patrones, o clasificar las entradas según

alguna característica. Una de las funciones principales de ambas ciencias es la construcción de algoritmos que puedan aprender y hacer predicciones sobre un conjunto de datos. Sin embargo, hay distintas formas de conseguir estos algoritmos y es importante conocer las diferentes opciones para poder elegir la más adecuada (Rajagopalan, B., & Krovi, R.,2002). Por ello, se analizan algunos de los más relevantes en la bioinformática farmacéutica:

- **Árboles de decisión:** Es una herramienta de apoyo en la toma de decisiones, que utiliza un gráfico o un modelo similares a un árbol de decisiones. Esto incluye los resultados fortuitos, las consecuencias, los costes de los recursos y la utilidad.

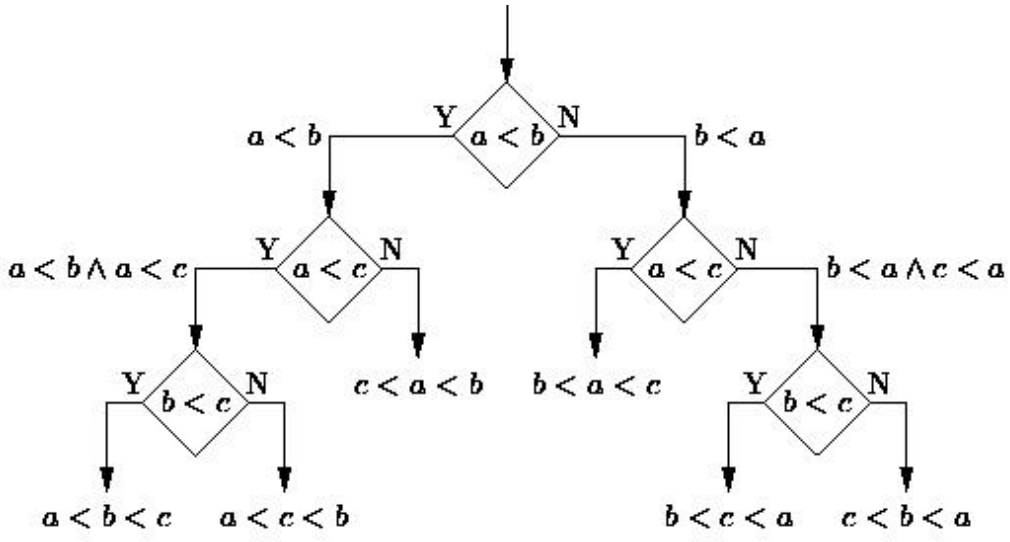


Ilustración 7. *Ejemplo general de un árbol de toma de decisiones.*
Fuente: http://www.kdnuggets.com/2016/08/10-algorithms-machine-learning-engineers.html

- **Naïves Bayes Classification:** Son una familia de clasificadores probabilísticos simples basados en el teorema de Bayes, asumiendo que hay independencia total entre las dos categorías. La siguiente fórmula representa este teorema:

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

Aplicado a nuestro caso:

  - P(A): Probabilidad de ser de la categoría A.
  - P(B): Probabilidad de ser la categoría B.
  - P(A|B): Probabilidad de ser A condicionada por la probabilidad de ser B.

- **Least Squares Regression:** La regresión lineal es ampliamente conocida en estadística. Este método genera una regresión lineal a partir del valor mínimo de los cuadrados de los residuos. Se puede aprovechar para trazar una línea que separe datos de categorías distintas.

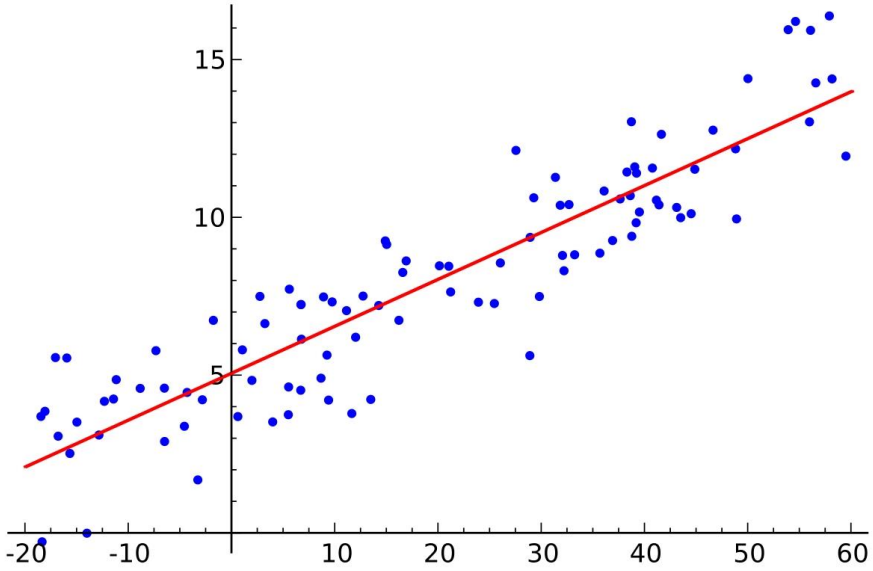


***Ilustración 8**. Ejemplo de separación de datos por mínimos cuadrados. Fuente:Wikipedia.*

- **Logistic Regression:** Es un modelo de regresión donde la variable dependiente es categórica. En el ejemplo de más abajo se puede comprobar que esta variable es binaria, es decir, que tiene solo dos categorías o valores.

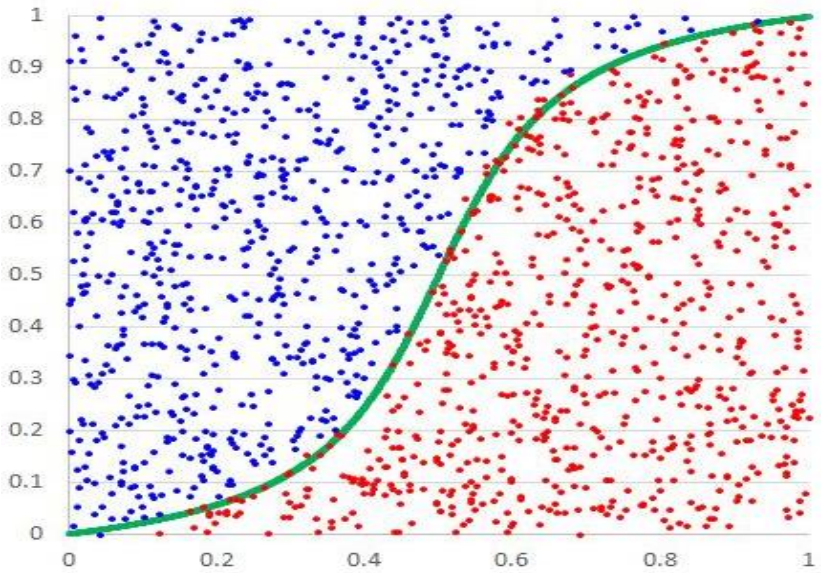


***Ilustración 9**. Ejemplo de regresión logística. Fuente: MSSQLTips.com*

- **Support Vector Machine:** Es un clasificador discriminativo basado en la separación por un hiperplano. Para funcionar necesita de un conjunto de datos marcados, en los que se indica a cuál de las clases pertenecen, de forma que se optimiza el hiperplano, para poder clasificar a nuevos ejemplos.

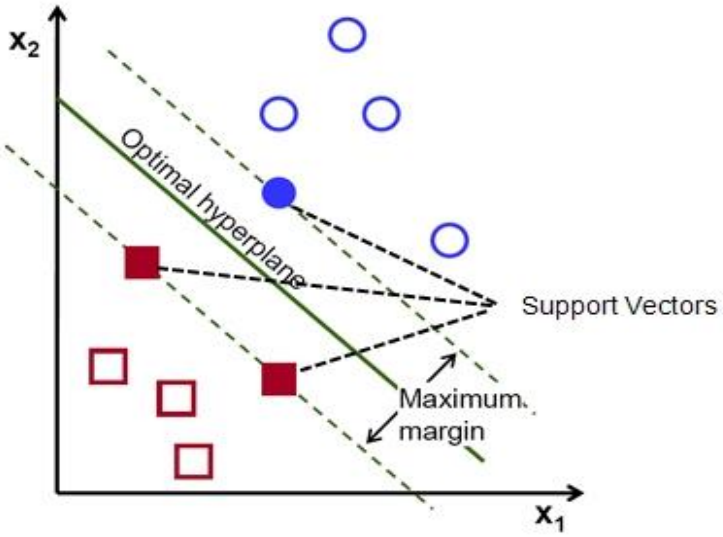


***Ilustración 10**. Ejemplo de la clasificación por un SVM. Fuente: docsopencv.org*

Por supuesto, hay más de algoritmos que podrían utilizarse para este trabajo, pero estos son los que se valoraron debido a que en la literatura son los más utilizados. En un contexto más amplio, con más tiempo y recursos, lo adecuado sería trabajar con varios para poder comprobar cuál consigue los mejores resultados, pero por logística se restringió a uno. De entre ellos fue el último, el *Support Vector Machine* que se consideró más adecuado ya que permitía comparar varios *kernels* y optimizar los parámetros de una forma sencilla.

Más motivos para elegir este grupo de algoritmos están basados en los objetivos y las características del mismo. El SVM está especializado en clasificar entre dos categorías, en el caso de este trabajo se quiere saber si las secuencias muestran actividad antimicrobiana o no, o si presentan toxicidad o no. Sin embargo, estas categorías podrían no tener una separación lineal, ya que su relación no es clara.

Este tipo de algoritmo ayudaría a dilucidar cuál es la relación que tendremos entre las variables y la actividad. Se pueden seleccionar diferentes tipos de funciones,

no solo la lineal, variando el *kernel* que se proporciona a los modelos se puede mejorar la precisión.

Además, la implementación de estos modelos puede llevarse a cabo en distintos lenguajes de programación, siendo los más comunes en *machine learning* y la minería de datos R y *Pyhton*, ambos muy presentes a lo largo del máster. Ambos tienen paquetes, o módulos para *Python*, que contienen funciones necesarias para construir el modelo. Así mismo, se utilizan para analizar, transformar y organizar datos; por lo que ambos pueden ser usados para manejar la información que se extraiga de las distintas bases para luego poder crear nuestra propia base de datos.

2.3. Selección de las variables explicativas para el modelo.

En distintos trabajos analizados se utilizan diferentes variables para poder generar un modelo con una buena precisión. En este trabajo se utiliza el conocimiento de la naturaleza físico-química previamente recabada sobre los AMPs para seleccionar los criterios más apropiados.

La idea original es poder crear las bases de datos utilizando simplemente la secuencia de aminoácidos, de forma que la herramienta final pueda utilizarse con poca información y de forma sencilla por el usuario. Un investigador sabe que cada aminoácido posee información intrínseca: carga neta, masa, aromaticidad…, pero un ordenador no es capaz de interpretarla directamente. Por lo que es necesario que se extraiga para cada aminoácido, incluso aquellos los datos derivados de la secuencia como conjunto (longitud, masa total...).

Para la creación del modelo se emplearon las siguientes variables:

- *Sequence.*
- *Mass.*
- *Length.*
- *Isoelectric Point.*
- *Aromaticity.*
- *Aminoacid Percent (cada uno en una columna).*
- *Effect.*

Como se mencionó anteriormente, una de las características destacadas de los AMPs es que presentan una carga positiva, es decir, que son catiónicos. El punto isoeléctrico determina la carga a pH fisiológico y, por tanto, aporta esta información. Otro factor importante es la hidrofobicidad, pues permite interactuar de manera adecuada con las membranas bacterianas. Estos péptidos tenían una importante fracción hidrofóbica que se puede medir con variable aromaticidad, una propiedad de los hidrocarburos cíclicos que presentan los aminoácidos.

El porcentaje de cada aminoácido aporta información sobre la composición de la secuencia y está vinculada también con las otras variables. La relación de los residuos es clave en la carga neta y en las fracciones hidrofóbicas e hidrofílicas que dan el carácter anfipático. A mayores, tanto la longitud de la secuencia, como el peso total del péptido, se utilizan al ser variables físicas fáciles de conseguir y que suelen ser muy explicativas. Finalmente, la variable *Effect*, es la que se utilizará en las bases de datos de entrenamiento y, también, será útil para las pruebas de test del modelo.

2.4. Creación de las bases de datos.

Para obtener las secuencias que nos interesaban utilizamos la base de datos SwissProt, recurriendo siempre a los datos manualmente anotados y revisados. Utilizando las *keywords* "KW-0800" (Toxin), y "NOT KW-0800", se recogen un total de 6,700 y 547,815 entradas, respectivamente. Se descargan los campos: *ID*, *Protein* names, *Organism*, *Length*, *Mass* y *Sequence*. Se denomina ToxicDatabase y NotToxicDatabase, según el tipo de proteínas extraídas.

De nuevo, desde SwissProt, se utiliza la negación del término "KW-0929", *Antimicrobial"*, para adquirir los péptidos que no tengan esta actividad. Se obtienen un total de 551,564 secuencias, que se descargan igual que en el caso anterior. Para reducir esta cantidad se recogen solo aquellos péptidos de origen humano, reduciéndose a solo 20,114. La base de datos descargada se nombra como NotAntimicrobialDatabase.

Para hacer nuestra siguiente base de datos se indaga en la categoría *Antimicrobial*, ya que el sistema de etiquetado por palabras clave es jerárquico, y encontramos que engloba distintas actividades: *Antibiotic, Bacteriolitic Enzyme, Defensin y Fungicide*. Se seleccionan las tres primeras, cuyo código es "KW-0044", "KW-

0081" y "KW-0211", respectivamente, ya que se centrará la atención en la actividad contra las bacterias y no contra otros organismos.

Las secuencias de cada término se exportan en formato Excel, con los mismos campos que para las bases de datos anterior, y se crean las bases: AntibioticDatabase, BacterioliticEnzymeDatabase y DefensineDatabase. Las tres se fusionan en una sola que se nombra AntimicrobialDatabase.

Para generar las variables de las bases de datos finales se usa la herramienta *ProteinAnalysis* de *ProtParam*, presente en el paquete *SeqUtils* del módulo *Bio* de Biopython. Esta herramienta convierte una secuencia en un objeto que tiene distintas funciones, cada una de ellas permite obtener los valores de las variables explicativas que se utilizan para el modelo.

Las distintas variables se recogen utilizando en el código de más abajo, cuyo objetivo es obtener los valores obtenidos de cada función y almacenarlos en una base de datos, que luego se utilizará para la generación del modelo.

```
Antimicrobial_Database = pd.read_excel(file)
NotAntimicrobial_Database = pd.read_excel(file2)
Antimicrobial_Analysis = Antimicrobial_Database['Sequence'].apply(ProteinAnalysis)
NotAntimicrobial_Analysis = NotAntimicrobial_Database['Sequence'].apply(ProteinAnalysis)

aminoacids = list()
isoelectricpoint = list()
aromaticity = list()
seq = range(0, len(Antimicrobial_Analysis))

for i in list(seq):
   aminoacids.append(Antimicrobial_Analysis[i].get_amino_acids_percent())
   isoelectricpoint.append(Antimicrobial_Analysis[i].isoelectric_point())
   aromaticity.append(Antimicrobial_Analysis[i].aromaticity())

aminoacidosFinalDict = dict([(k, [x[''+k+''] \
            for x in aminoacids]) for k in aminoacids[0]])

AntimicrobialPeptides = pd.DataFrame()
AntimicrobialPeptides['Mass'] = Antimicrobial_Database['Mass']
AntimicrobialPeptides['Length'] = Antimicrobial_Database['Length']
AntimicrobialPeptides['Aromaticity'] = aromaticity
AntimicrobialPeptides['IsoelectricPoint'] = isoelectricpoint

for k,value in aminoacidosFinalDict.items():
    AntimicrobialPeptides[k] = value

AntimicrobialPeptides['AntimicrobialActivity']=[1 for x in seq]

```

Ilustración 11. *Código de Python para la generación de la base de datos. Este código es análogo al que se utiliza para el resto de bases de datos. Fuente: Propia.*

Cabe destacar que, debido al número tan elevado de algunos resultados, se generan las muestras de 1000 entradas para usar en el modelo. Esto resulta en la generación de las FinalAntimicrobialSample y FinalToxicSample.

2.5. Generación de los modelos predictivos.

Una vez diseñadas las bases de datos con las variables explicativas que nos interesan se genera el modelo. El código se describe abajo

```
14
15 # load the CSV file as a numpy matrix
16 dataset = np.array(x).astype('float')
17
18 # separate the data from the target attributes
19 xRaw = dataset[:,0:dataset.shape[1]-1]
20 yRaw = dataset[:,dataset.shape[1]-1]
21
22 # Set labels for wighting
23 le = preprocessing.LabelEncoder()
24 y = le.fit_transform(yRaw)
25
26 # missing values
27 imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
28 xPrep = imp.fit_transform(xRaw)
29 #Standardize data
30 scaler = preprocessing.StandardScaler().fit(xPrep)
31 x=scaler.transform(xPrep)
32
33 #Divide in training and test, shuffling the examples and keeping the proportion of examples of each class
34 xTrain, xTest, yTrain, yTest = cross_validation.train_test_split(x, y, test_size=0.2, random_state=1)
35 #print('xTrain', xTrain, "\nxTest", xTest, "\nyTrain", yTrain, "\nyTest", yTest)
36
37 #Generate grid search
38 kernels = ['linear', 'poly', 'rbf']
39
40 for k in kernels:
41     errList, devList, cValues, gValues = [], [], [], []
42     gamma = [1]
43     cList = list(np.arange(0.1, 1, 0.05))
44     if k == 'rbf':
45         gamma = list(np.arange(0.1, 1, 0.1))
46     hyperParams = {'kernel': [k], 'C': cList, 'gamma': gamma}
47
48     #Create an instance of Tree Classifier and fit the data for the grid parameters
49     modelCV = GridSearchCV(svm.SVC(), param_grid=hyperParams, cv=5)
50     modelCV.fit(xTrain, yTrain)
51     print("Best hyperparameters", modelCV.best_params_)
52     errList2, devList2 = [], []
53     for hyperP, mean_score, scores in modelCV.grid_scores_:
54         print("%0.3f (+/-%0.03f) for %r"
55                   % (mean_score, scores.std(), hyperP))
56         cValues.append(hyperP['C'])
57         gValues.append(hyperP['gamma'])
58         errList.append(mean_score)
59         devList.append(scores.std())
60     #Create an instance of Tree Classifier with the best hyperparameters and weight: auto and the full training set
61     model = svm.SVC(kernel = modelCV.best_params_['kernel'],
62         C = modelCV.best_params_['C'],
63         gamma = modelCV.best_params_['gamma'])
64     model.fit(xTrain, yTrain)
65
```

Ilustración 12. *Código en python para la generación del modelo. Los distintos apartados están brevemente explicados en las líneas superiores. Fuente: Propia.*

En líneas generales, para hacer funcionar el modelo se necesita una base de datos que tenga los elementos, de las clases que se van predecir, debidamente clasificados. Esta base de datos se separará, una parte para entrenar el modelo y la otra para testarlo. Ambos procesos se hacen de forma aleatoria respetando una proporción dada y están reflejados en la línea 34.

Después, se utilizan distintos parámetros del modelo: *gamma*, *C* y el *kernel*, para optimizar el clasificador. Lo más destacable en este código es la elección de los *kernels*, que determinan qué tipo de función de similitud se aporta al algoritmo. Se probarán 3 tipos: lineal, polinomial (*poly)* y radial (radial basis function, *rbf).* Se entrenará con los tres y se obtendrán los resultados de forma gráfica, además de que se sacarán los mejores parámetros, definidos como '*C*', para cada uno de ellos.

Finalmente, los mejores modelos (parámetros incluidos) se guardarán con el código que se enseña a continuación:

```
46
47 def saveModel(kernel, c, g, name):
48     model = svm.SVC(kernel = kernel, C = c, gamma = g)
49     model.fit(xTrain, yTrain)
50
51     yPred=model.predict(xTest)
52     cnf_matrix = confusion_matrix(yTest, yPred)
53     print(cnf_matrix)
54
55     joblib.dump(model, 'C:/TFM/'+ name +'.pkl')
56
57 saveModel('rbf',0.9, 0.1, 'ampModelRbf')
58 saveModel('poly',0.1, 1, 'ampModelPoly')
59
```

Ilustración 13. *Código de python para guardar los mejores parámetros. En este caso se guardaron solo aquellos que usan los kernels rbf y poly. Fuente: Propia.*

2.6. Testo del modelo.

Una vez entrenado el modelo se hacen pruebas de su eficacia con el set de test (*Xtest* en el código), que es el resto de la base de datos que no fue utilizada para el entrenamiento. De esta forma, se utilizan diferentes muestras para que las clasifique y así poder comparar cuáles funcionan mejor.

```
67     #Test
68     for g in gamma:
69       testError = []
70       for c in cList:
71           model = svm.SVC(kernel = k, C = c, gamma = g)
72           model.fit(xTrain, yTrain)
73           testError.append(model.score(xTest, yTest))
74
75       cValuesForGamma = [x for index, x in enumerate(cValues) if gValues[index] == g]
76
77       print("Best hyperparameters", modelCV.best_params_)
78       plt.title("kernel = %s, gamma = '%s'" % (k, g))
79
80       model.predict(xTest)
81
82       plt.errorbar(cValues, errList, yerr = devList)
83       plt.errorbar(cValuesForGamma, testError)
84       plt.xlim(cValues[0]-0.2, cValues[len(cValues)-1]+0.2)
85       plt.show()
86
```

Ilustración 14. *Código de Python para testear los modelos entrenados previamente. Fuente: Propia*

Los resultados se muestran en forma de gráficas que exhiben la precisión obtenida durante el entrenamiento y durante la fase de test. También se escriben los mejores parámetros para cada uno, aunque solo en *rbf* se prueban distintos tipos de alguno que no sea *C*

A mayores, se realizan otras pruebas con los modelos obtenidos. En este caso se utiliza una matriz de confusión para saber cuántas veces se equivoca a la hora de clasificar cada categoría. Se reutiliza el código de programas anteriores para cargar los datos.

```
16
17 def preprocessData(dataset):
18     raw_data = open(dataset, 'rt')
19     reader = csv.reader(raw_data, delimiter = ',', quoting=csv.QUOTE_NONE)
20     next(reader, None)
21     x = list(reader)
22
23     # load the CSV file as a numpy matrix
24     dataset = np.array(x).astype('float')
25
26     # separate the data from the target attributes
27     xRaw = dataset[:,0:dataset.shape[1]-1]
28     yRaw = dataset[:,dataset.shape[1]-1]
29
30     # Set labels for wighting
31     le = preprocessing.LabelEncoder()
32
33     # missing values
34     imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
35     xPrep = imp.fit_transform(xRaw)
36     #Standardize data
37     scaler = preprocessing.StandardScaler().fit(xPrep)
38     x=scaler.transform(xPrep)
39     return x
40
41 def getResults(x):
42     prediction = model.predict(x)
43     predictionAccuracy = len([x for x in prediction if x == 1]) / len(prediction)
44
45     print(len([x for x in prediction if x == 1]))
46     print(len(prediction))
47     print(predictionAccuracy)
48
49 x = preprocessData(dataset)
50 x2 = preprocessData(dataset2)
51
52 getResults(x)
53 getResults(x2)
54
```

***Ilustración 15**. Código de Python para realizar nuevos exámenes de los modelos. Fuente: Propia.*

2.7. Productos obtenidos

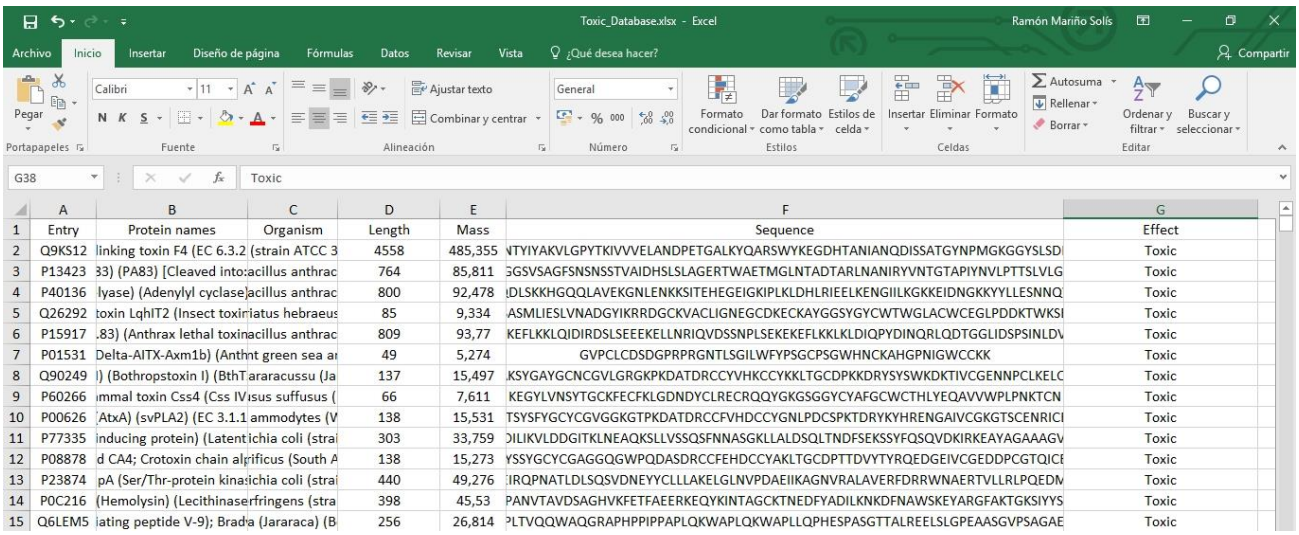
Los resultados de este trabajo han sido de diversa índole. En cada uno de los pasos se han ido generando distintos archivos, como las distintas bases de datos o los

programas en Python, pero los más relevantes son los modelos generados y las validaciones hechas sobre ellos.

2.7.1. Bases de datos

Se han creado varias bases de datos que reúnen péptidos que poseen, o no, cierta actividad concreta. En este caso, todas tienen los mismos campos y se diferencian unas de otras por el contenido. Como el tamaño es de miles de entradas cada una, continuación se expondrá solo una muestra de las primeras 15 entradas.

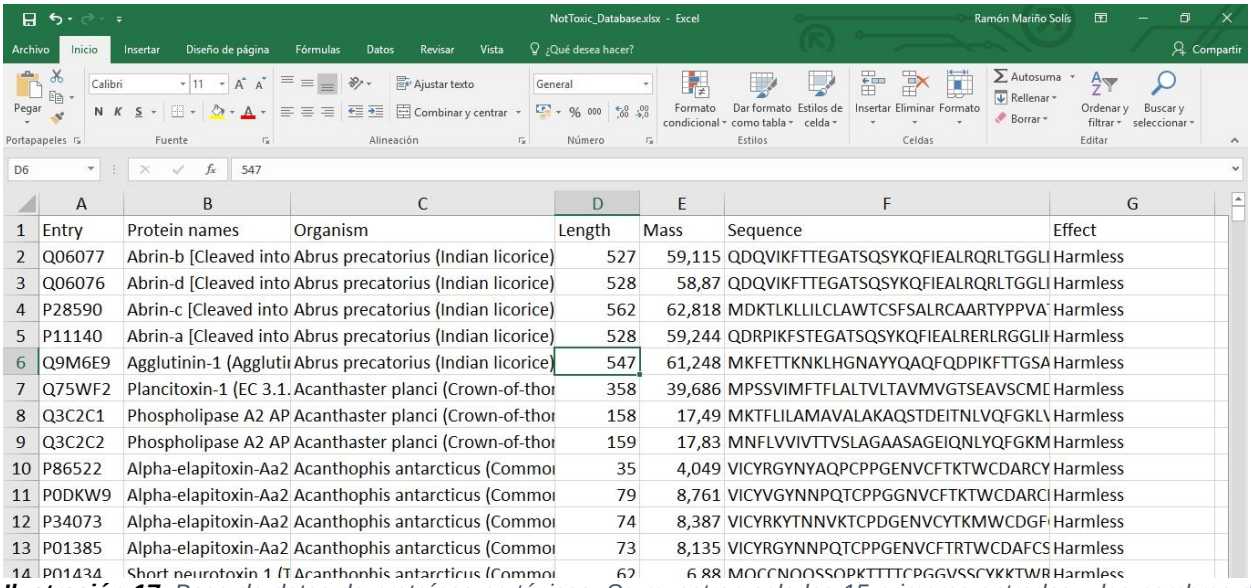
ToxicDatabase:



	A	B	C	D	E	F	G
1	Entry	Protein names	Organism	Length	Mass	Sequence	Effect
2	Q9KS12	linking toxin F4 (EC 6.3.2	(strain ATCC 3	4558	485,355	NTYIYAKVLGPYTKIVVVELANDPETGALKYQARSWYKEGDHTANIANQDISSATGYNPMGKGGYSLSD	Toxic
3	P13423	33) (PA83) [Cleaved into	acillus anthrac	764	85,811	GGSVSAGFSNSNSSTVAIDHSLSLAGERTWAETMGLNTADTARLNANIRYVNTGTAPIYNVLPTTSLVLG	Toxic
4	P40136	lyase) (Adenylyl cyclase	acillus anthrac	800	92,478	DLSKKHGQQLAVEKGNLENKKSITEHEGEIGKIPLKLDHLRIEELKENGIILKGKKEIDNGKKYYLLESNNQ	Toxic
5	Q26292	toxin LqhIT2 (Insect toxi	riatus hebraeus	85	9,334	ASMLIESLVNADGYIKRRDGCKVACLIGNEGCDKECKAYGGSYGYCWTWGLACWCEGLPDDKTWKSI	Toxic
6	P15917	.83) (Anthrax lethal toxin	acillus anthrac	809	93,77	KEFLKKLQIDIRDSLSEEEKELLNRIQVDSSNPLSEKEKEFLKKLKLDIQPYDINQRLQDTGGLIDSPSINLDV	Toxic
7	P01531	Delta-AITX-Axm1b) (Anth	nt green sea ar	49	5,274	GVPCLCDSDGPRPRGNTLSGILWFYPSGCPSGWHNCKAHGPNIGWCCKK	Toxic
8	Q90249	l) (Bothropstoxin I) (Bth	T ararauss u (Ja	137	15,497	KSYGAYGCNCGVLGRGKPKDATDRCCYVHKCCYKKLTGCDPKKDRYSYSWKDKTIVCGENNPCLKELC	Toxic
9	P60266	mmal toxin Css4 (Css IV	isus suffusus (	66	7,611	KEGYLVNSYTGCKFECFKLGDNDYCLRECRQQYGKGSGGYCYAFGCWCTHLYEQAVVWPLPNKTCN	Toxic
10	P00626	AtxA) (svPLA2) (EC 3.1.1	ammodytes (W	138	15,531	TSYSFYGCYCGVGGKGTPKDATDRCCFVHDCCYGNLPDCSPKTDRYKYHRENGAIVCGKGTSCENRIC	Toxic
11	P77335	inducing protein) (Latent	ichia coli (strai	303	33,759	DILIKVLDDGITKLNEAQKSLLVSSQSFNNASGKLLALDSQLTNDFSEKSSYFQSQVDKIRKEAYAGAAAGV	Toxic
12	P08878	d CA4; Crotoxin chain a	rificus (South A	138	15,273	YSSYGCYCGAGGQGWPQDASDRCCFEHDCCYAKLTGCDPTTDVYTYRQEDGEIVCGEDDPCGTQICE	Toxic
13	P23874	pA (Ser/Thr-protein kina	ichia coli (strai	440	49,276	IRQPNATLDLSQSVDNEYYCLLLAKELGLNVPDAEIIKAGNVRALAVERFDRRWNAERTVLLRLPQEDM	Toxic
14	P0C216	(Hemolysin) (Lecithinase	rfringens (stra	398	45,53	PANVTAVDSAGHVKFETFAEERKEQYKINTAGCKTNEDFYADILKNKDFNAWSKEYARGFAKTGKSIYYS	Toxic
15	Q6LEM5	iating peptide V-9); Brad	a (Jararaca) (B	256	26,814	PLTVQQWAQGRAPHPPIPPAPLQKWAPLQKWAPLLQPHESPASGTTALREELSLGPEAASGVPSAGAE	Toxic

Ilustración 16. *Base de datos de péptidos tóxicos. Se muestran solo las 15 primeras entradas y el nombre de cada uno de los campos. Fuente: Propia*

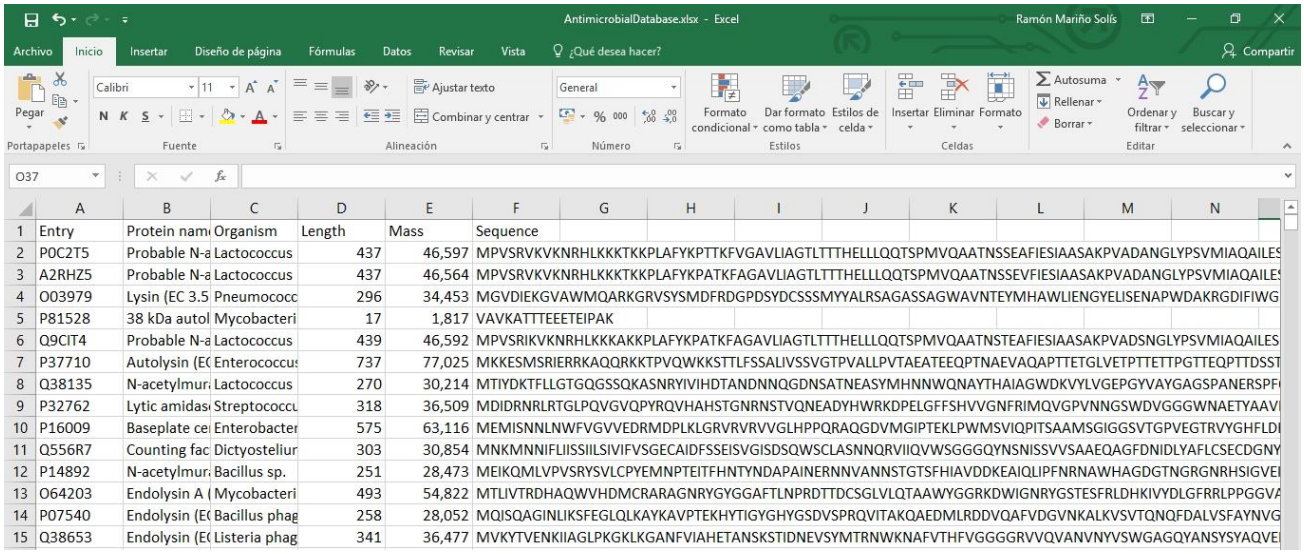
NotToxicDatabase:



	A	B	C	D	E	F	G
1	Entry	Protein names	Organism	Length	Mass	Sequence	Effect
2	Q06077	Abrin-b [Cleaved into	Abrus precatorius (Indian licorice)	527	59,115	QDQVIKFTTEGATSQSYKQFIEALRQRLTGGLI	Harmless
3	Q06076	Abrin-d [Cleaved into	Abrus precatorius (Indian licorice)	528	58,87	QDQVIKFTTEGATSQSYKQFIEALRQRLTGGLI	Harmless
4	P28590	Abrin-c [Cleaved into	Abrus precatorius (Indian licorice)	562	62,818	MDKTLKLLILCLAWTCSFSALRCAARTYPPVA	Harmless
5	P11140	Abrin-a [Cleaved into	Abrus precatorius (Indian licorice)	528	59,244	QDRPIKFSTEGATSQSYKQFIEALRERLRGGLIF	Harmless
6	Q9M6E9	Agglutinin-1 (Agglutir	Abrus precatorius (Indian licorice)	547	61,248	MKFETTKNKLHGNAYYQAQFQDPIKFTTGSA	Harmless
7	Q75WF2	Plancitoxin-1 (EC 3.1.	Acanthaster planci (Crown-of-thor	358	39,686	MPSSVIMFTFLALTVLTAVMVGTSEAVSCMD	Harmless
8	Q3C2C1	Phospholipase A2 AP	Acanthaster planci (Crown-of-thor	158	17,49	MKTFLILAMAVALAKAQSTDEITNLVQFGKLV	Harmless
9	Q3C2C2	Phospholipase A2 AP	Acanthaster planci (Crown-of-thor	159	17,83	MNFLVVIVTTVSLAGAASAGEIQNLYQFGKM	Harmless
10	P86522	Alpha-elapitoxin-Aa2	Acanthophis antarcticus (Commor	35	4,049	VICYRGYNYAQPCPPGENVCFTKTWCDARCY	Harmless
11	P0DKW9	Alpha-elapitoxin-Aa2	Acanthophis antarcticus (Commor	79	8,761	VICYVGYNNPQTCPPGGNVCFTKTWCDARCI	Harmless
12	P34073	Alpha-elapitoxin-Aa2	Acanthophis antarcticus (Commor	74	8,387	VICYRKYTNNVKTCPDGENVCYTKMWCDGF	Harmless
13	P01385	Alpha-elapitoxin-Aa2	Acanthophis antarcticus (Commor	73	8,135	VICYRGYNNPQTCPPGENVCFTRTWCDAFCS	Harmless
14	P01434	Short neurotoxin 1 (T	Acanthophis antarcticus (Commor	62	6,88	MQCCNQQSSQPKTTTTCPGGVSSCYKKTWR	Harmless

Ilustración 17. *Base de datos de proteínas no tóxicas. Se muestran solo las 15 primeras entradas y los nombres de las columnas. Fuente: Propio*

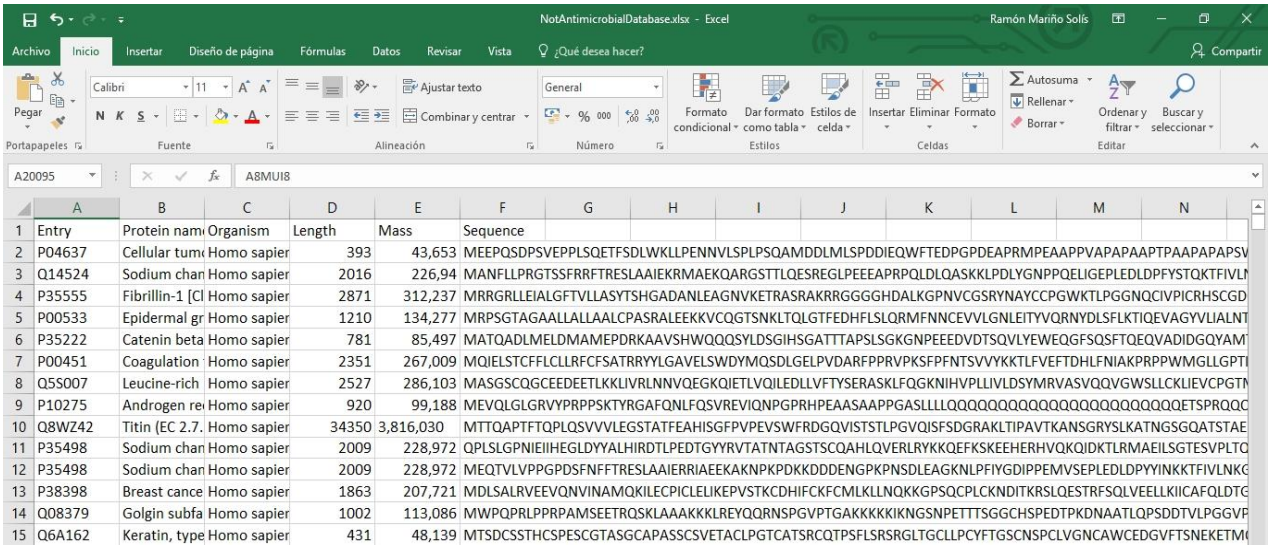
Antimicrobial:



	A	B	C	D	E	F
1	Entry	Protein nam	Organism	Length	Mass	Sequence
2	P0C2T5	Probable N-a	Lactococcus	437	46,597	MPVSRVKVKNRHLKKKTKKPLAFYKPTTKFVGAVLIAGTLTTTHELLLQQTSPMVQAATNSSEAFIESIAASAKPVADANGLYPSVMIAQAILES
3	A2RHZ5	Probable N-a	Lactococcus	437	46,564	MPVSRVKVKNRHLKKKTKKPLAFYKPATKFAGAVLIAGTLTTTHELLLQQTSPMVQAATNSSEVFIESIAASAKPVADANGLYPSVMIAQAILES
4	O03979	Lysin (EC 3.5	Pneumococc	296	34,453	MGVDIEKGVAWMQARKGRVSYSMDFRDGPDSYDCSSSMYYALRSAGASSAGWAVNTEYMHAWLIENGYELISENAPWDAKRGDIFIWG
5	P81528	38 kDa autol	Mycobacteri	17	1,817	VAVKATTTEEETEIPAK
6	Q9CIT4	Probable N-a	Lactococcus	439	46,592	MPVSRIKVKNRHLKKKAKKPLAFYKPATKFAGAVLIAGTLTTTHELLLQQTSPMVQAATNSTEAFIESIAASAKPVADSNGLYPSVMIAQAILES
7	P37710	Autolysin (E(	Enterococcus	737	77,025	MKKESMSRIERRKAQQRKKTPVQWKKSTTLFSSALIVSSVGTPVALLPVTAEATEEQPTNAEVAQAPTTETGLVETPTTETTPGTTEQPTTDSST
8	Q38135	N-acetylmur	Lactococcus	270	30,214	MTIYDKTFLLGTGQGSSQKASNRYIVIHDTANDNNQGDNSATNEASYMHNNWQNAYTHAIAGWDKVYLVGEPGYVAYGAGSPANERSPF
9	P32762	Lytic amidas	Streptococcu	318	36,509	MDIDRNRLRTGLPQVGVQPYRQVHAHSTGNRNSTVQNEADYHWRKDPELGFFSHVVGNFRIMQVGPVNNGSWDVGGGWNAETYAAVI
10	P16009	Baseplate ce	Enterobacter	575	63,116	MEMISNNLNWFVGVVEDRMDPLKLGRVRVRVVGLHPPQRAQGDVMGIPTEKLPWMSVIQPITSAAMSGIGGSVTGPVEGTRVYGHFLDI
11	Q556R7	Counting fac	Dictyosteliur	303	30,854	MNKMNNIFLIISSIILSIVIFVSGECAIDFSSEISVGISDSQWSCLASNNQRVIIQVWSGGGQYNSNISSVVSAAEQAGFDNIDLYAFLCSECDGNY
12	P14892	N-acetylmur	Bacillus sp.	251	28,473	MEIKQMLVPVSRYSVLCPYEMNPTEITFHNTYNDAPAINERNNVANNSTGTSFHIAVDDKEAIQLIPFNRNAWHAGDGTNGRGNRHSIGVEI
13	O64203	Endolysin A (	Mycobacteri	493	54,822	MTLIVTRDHAQWVHDMCRARAGNRYGYGGAFTLNPRDTTDCSGLVLQTAAWYGGRKDWIGNRYGSTESFRLDHKIVYDLGFRRLPPGGVA
14	P07540	Endolysin (E(	Bacillus phag	258	28,052	MQISQAGINLIKSFEGLQLKAYKAVPTEKHYTIGYGHYGSDVSPRQVITAKQAEDMLRDDVQAFVDGVNKALKVSVTQNQFDALVSFAYNVG
15	Q38653	Endolysin (E(	Listeria phag	341	36,477	MVKYTVENKIIAGLPKGKLKGANFVIAHETANSKSTIDNEVSYMTRNWKNAFVTHFVGGGGRVVQVANVNYVSWGAGQYANSYSYAQVEI

Ilustración 18. *Base de datos de péptidos antimicrobianos. Se muestran solo las 15 primeras entradas y el nombre de las columnas. Fuente: Propia*

NotAntimicrobial:



	A	B	C	D	E	F
1	Entry	Protein nam	Organism	Length	Mass	Sequence
2	P04637	Cellular tum	Homo sapier	393	43,653	MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLSPDDIEQWFTEDPGPDEAPRMPEAAPPVAPAPAAPTPAAPAPAPSV
3	Q14524	Sodium chan	Homo sapier	2016	226,94	MANFLLPRGTSSFRRFTRESLAAIEKRMAEKQARGSTTLQESREGLPEEEAPRPQLDLQASKKLPDLYGNPPQELIGEPLEDLDPFYSTQKTFIVLN
4	P35555	Fibrillin-1 [Cl	Homo sapier	2871	312,237	MRRGRLLEIALGFTVLLASYTSHGADANLEAGNVKETRASRAKRRGGGGHDALKGPNVCGSRYNAYCCPGWKTLPGGNQCIVPICRHSCGD
5	P00533	Epidermal gr	Homo sapier	1210	134,277	MRPSGTAGAALLALLAALCPASRALEEKKVCQGTSNKLTQLGTFEDHFLSLQRMFNNCEVVLGNLEITYVQRNYDLSFLKTIQEVAGYVLIALNT
6	P35222	Catenin beta	Homo sapier	781	85,497	MATQADLMELDMAMEPDRKAAVSHWQQQSYLDSGIHSGATTTAPSLSGKGNPEEEDVDTSQVLYEWEQGFSQSFTQEQVADIDGQYAMT
7	P00451	Coagulation	Homo sapier	2351	267,009	MQIELSTCFFLCLLRFCFSATRRYYLGAVELSWDYMQSDLGELPVDARFPPRVPKSFPFNTSVVYKKTLFVEFTDHLFNIAKPRPPWMGLLGPTI
8	Q5S007	Leucine-rich	Homo sapier	2527	286,103	MASGSCQGCEEDEETLKKLIVRLNNVQEGKQIETLVQILEDLLVFTYSERASKLFQGKNIHVPLLIVLDSYMRVASVQQVGWSLLCKLIEVCPGTN
9	P10275	Androgen re	Homo sapier	920	99,188	MEVQLGLGRVYPRPPSKTYRGAFQNLFQSVREVIQNPGPRHPEAASAAPPGASLLLLQQQQQQQQQQQQQQQQQQQQQQQQQETSPRQQC
10	Q8WZ42	Titin (EC 2.7.	Homo sapier	34350	3,816,030	MTTQAPTFTQPLQSVVVLEGSTATFEAHISGFPVPEVSWFRDGQVISTSTLPGVQISFSDGRAKLTIPAVTKANSGRYSLKATNGSGQATSTAE
11	P35498	Sodium chan	Homo sapier	2009	228,972	QPLSLGPNIEIIHEGLDYYALHIRDTLPEDTGYYRVTATNTAGSTSCQAHLQVERLRYKKQEFKSKEEHERHVQKQIDKTLRMAEILSGTESVPLTQ
12	P35498	Sodium chan	Homo sapier	2009	228,972	MEQTVLVPPGPDSFNFFTRESLAAIERRIAEEKAKNPKPDKKDDDENGPKPNSDLEAGKNLPFIYGDIPPEMVSEPLEDLDPYYINKKTFIVLNKG
13	P38398	Breast cance	Homo sapier	1863	207,721	MDLSALRVEEVQNVINAMQKILECPICLELIKEPVSTKCDHIFCKFCMLKLLNQKKGPSQCPLCKNDITKRSLQESTRFSQLVEELLKIICAFQLDTG
14	Q08379	Golgin subfa	Homo sapier	1002	113,086	MWPQPRLPPRPAMSEETRQSKLAAAKKKLREYQQRNSPGVPTGAKKKKKIKNGSNPETTTSGGCHSPEDTPKDNAATLQPSDDTVLPGGVP
15	Q6A162	Keratin, type	Homo sapier	431	48,139	MTSDCSSTHCSPESCGTASGCAPASSCSVETACLPGTCATSRCQTPSFLSRSRGLTGCLLPCYFTGSCNSPCLVGNCAWCEDGVFTSNEKETM

Ilustración 19. *Base de datos de péptidos no antimicrobianos. Se muestran solo las 15 primeras entradas y el nombre de las columnas. Fuente: Propia*

2.7.2. Programas

Para realizar el programa se crearon varios programas en Python. Estos códigos deben ser retocados para el uso de forma generalizada, ya que actualmente están implementados para trabajar en un ordenador concreto al utilizar directorios específicos. La mayor parte de ellos ya han sido descritos en el apartado anterior, por lo que aquí solo se citarán. Se presentará una copia en el apartado de anexos.

Lista de programas:

- **Database.py** (crea la base de datos de péptidos tóxicos y no tóxicos).
- **AntimicrobialDatabse.py** (crea la base de datos péptidos antibacterianos y no antibacterianos).
- **svmClasificacion1.py** (crea, entrena y testea el modelo para encontrar los parámetros más apropiados).
- **Save_Train_Model.py** (guarda los modelos óptimos)
- **Test.py** (se comprueban la efectividad de los modelos optimizados).

2.7.3. Modelos

Los modelos generados se guardan con el programa **Save_Train_Model.py**, mencionado en el apartado anterior, y se hace en formato *pkl*. Son los siguientes:

- **ampModelPoly.pkl**
- **ampModelRbf.pkl**

Solo se guardaron aquellos que modelos con resultados de precisión útiles para su uso posterior. El predictor de toxicidad no se guardó, aunque sí que se analizarán los resultados obtenidos para analizar los motivos que llevaron a no conservar el modelo.

2.8. Resultados

2.8.1. Modelo de toxicidad

Los resultados gráficos obtenidos por el programa svmClasification1.py muestran que la precisión es muy baja. En las siguientes ilustraciones se muestran los resultados para los distintos *kernels* y, en el caso del *rbf*, también para los distintos valores de gamma entre 0.1 y 1.

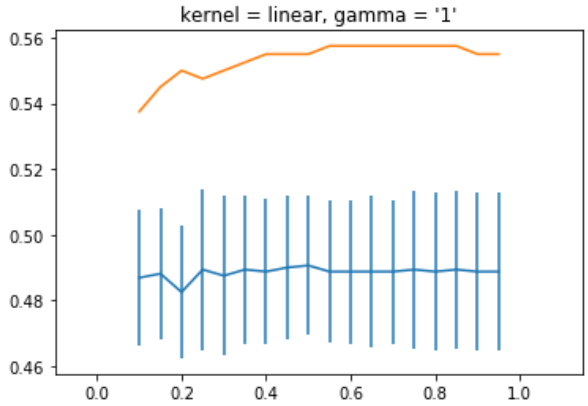
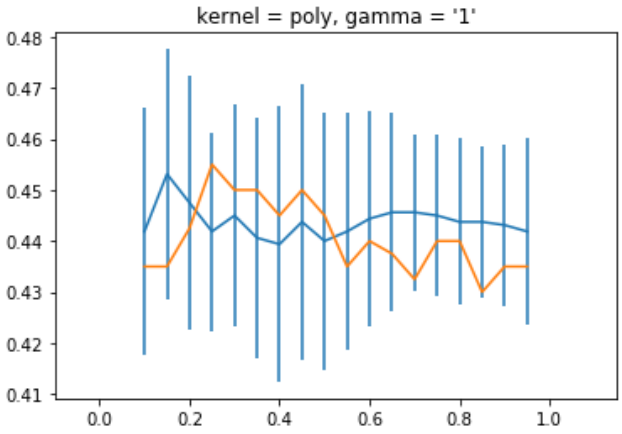


Ilustración 20. *Precisión del modelo SVM para la clasificación de la toxicidad con kernel linal y gamma=1. La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el test.*



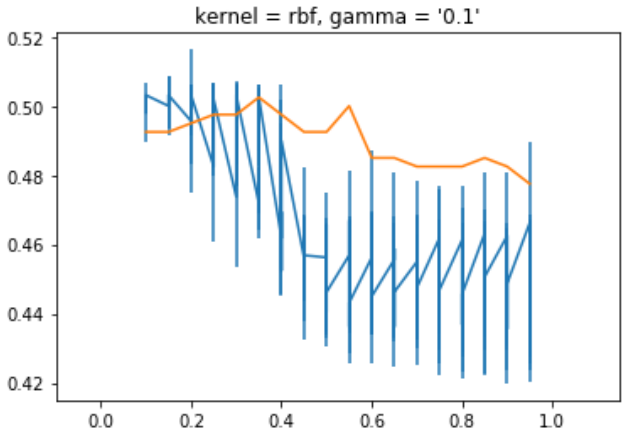
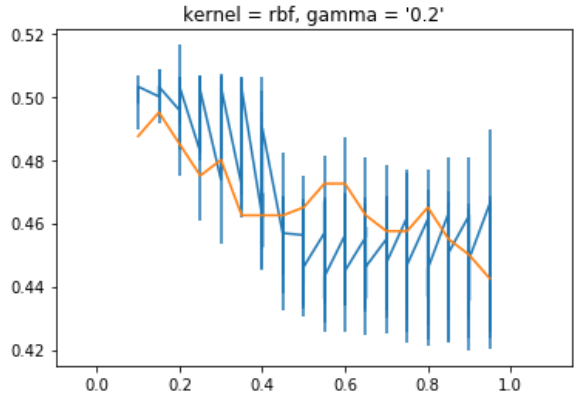
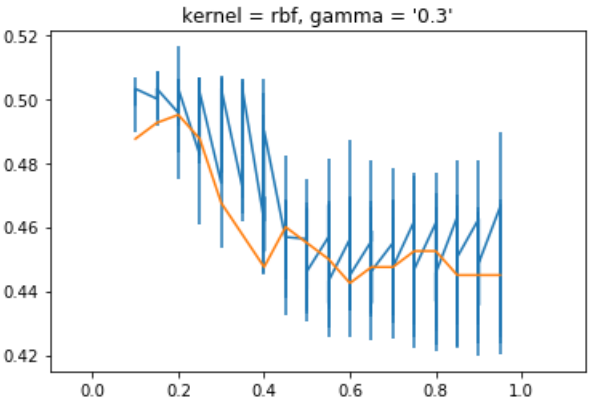
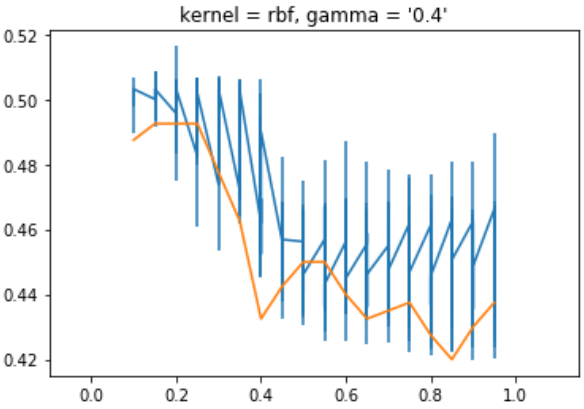
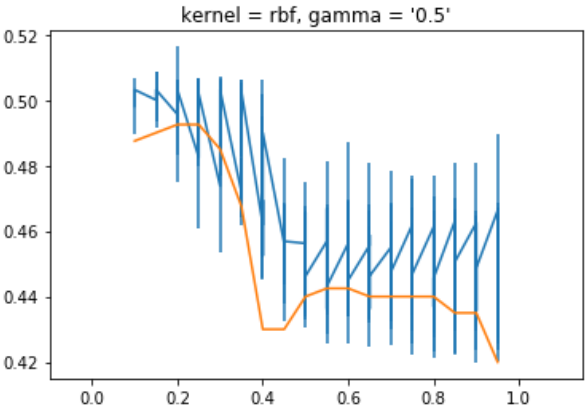


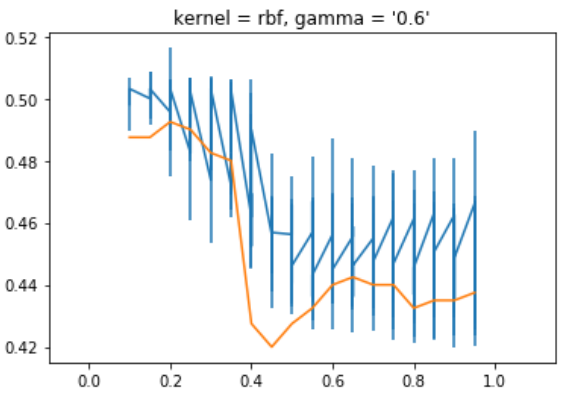
***Ilustración 21**. Precisión del modelo SVM para la clasificación de la toxicidad con kernel polinomial y gamma=1. La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el testeo.*

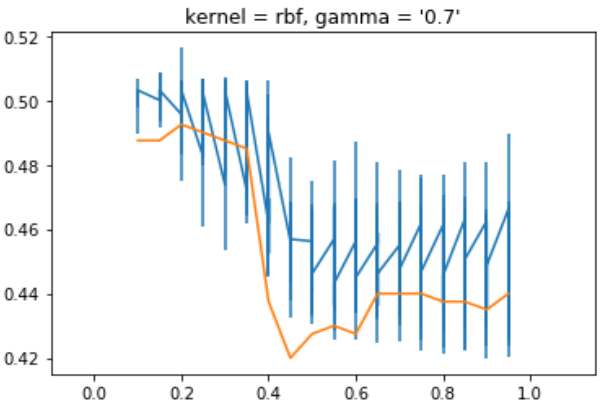












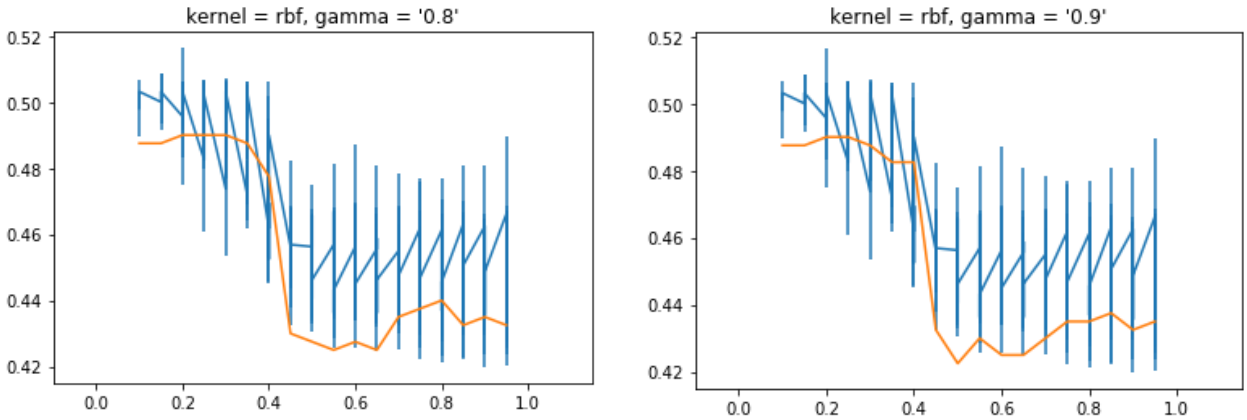


Ilustración 22. *Las diez imágenes anteriores muestran la precisión del modelo SVM para la clasificación de la toxicidad con kernel rbf y gamma=0.1 – 0.9 (cada imagen con un aumento de 0.1 con respecto a la anterior). La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el testeo de modelo.*

Estos resultados muestran que, en ninguno de los kernels utilizados, se alcanza una precisión demasiado elevada. El lineal, que obtiene los mejores porcentajes, apenas se roza el 56%, y el polinomial y el rbf, independientemente del valor de gamma, tienen variaciones descendentes muy abruptas.

Si nos fijamos en los parámetros *C* en los distintos, modelos su aumento reduce todavía más la precisión. Al tener mayor valor se intenta clasificar todas las entradas del *set* de entrenamiento de forma correcta, se aumenta el peso de cada observación y se aceptan menos errores lo que redunda en un modelo más estricto.

En el caso de *rbf* hay que ponderar también *gamma*, que actúa junto con *C* para determinar el margen de separación. De nuevo, si tenemos valores altos de cada uno tenemos una exigencia alta y, otra vez, se observa que la predicción se reduce. En las últimas gráficas, las que tienen un gamma elevado (0.6-0-9) cuando sobre pasamos el valor de 0.4 para *C* la precisión desciende abruptamente.

Lo que se indica es un modelo que no tiene una buena capacidad clasificatoria. Al tener que elegir entre dos categorías, si se considera que sin ningún tipo de información la probabilidad de acertar es de un 50%, que el clasificador consiga valores del 40% sugiere que resulta más útil hacerlo al azar.

Los resultados del modelo nos señalan que los datos no pueden separarse de forma lineal, radial o polinomial y, por lo tanto, no se puede conseguir una buena precisión. Estos porcentajes pueden ser tan bajos por las variables explicativas utilizadas o porque el algoritmo empleado no es adecuado. La utilización de otro tipo de algoritmos, como los mencionados anteriormente, o conseguir unas variables más

vinculadas con la toxicidad, pueden permitir la mejora del modelo. También puede emplearse alguno de los modelos ya implementados por otros investigadores o grupos, citados anteriormente en este trabajo.

2.8.2. Modelo de antimicrobianos

Al igual que en el caso anterior, se obtienen las gráficas de la precisión del modelo que se construyó para predecir la actividad antibacteriana.

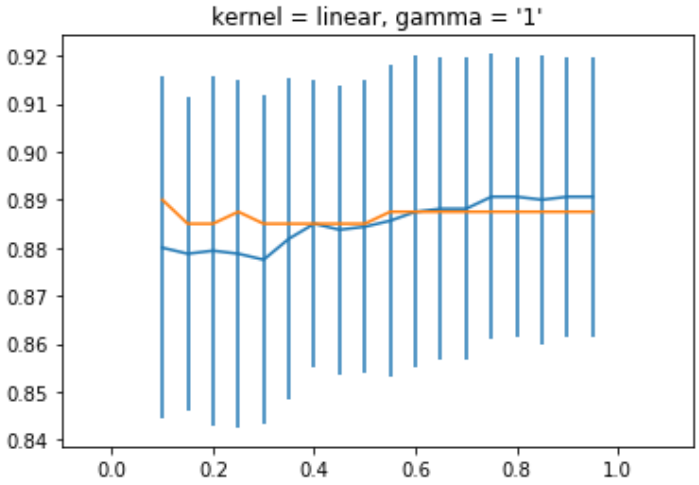


Ilustración 23. *Precisión del modelo SVM para la clasificación de la actividad antibacteriana, con kernel lineal y gamma=1. La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el testeo.*

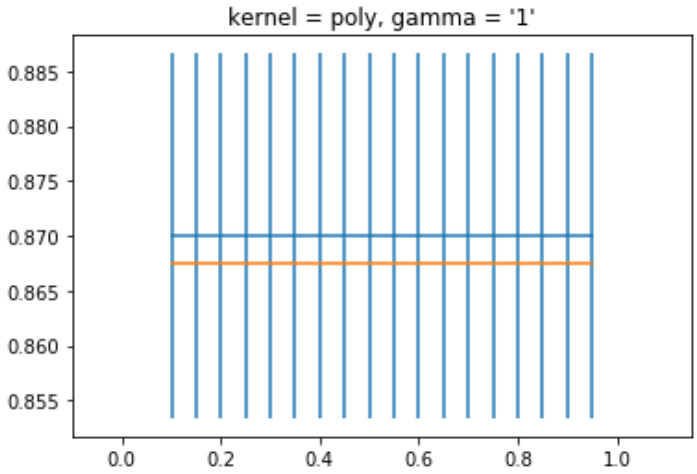
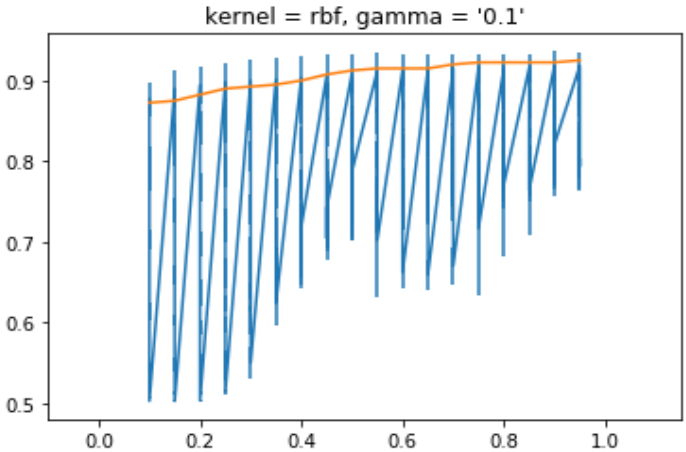
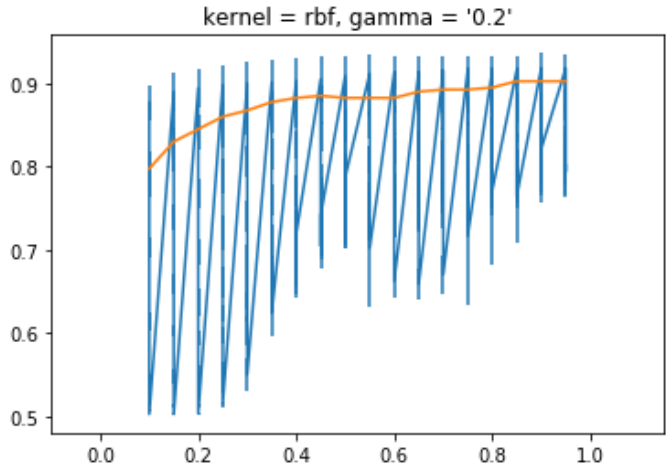
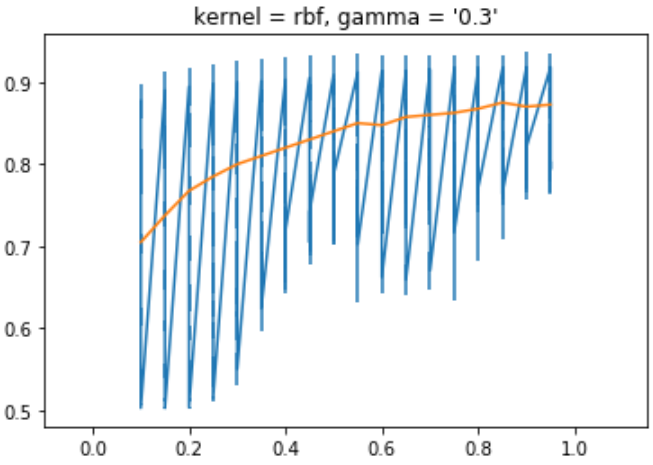
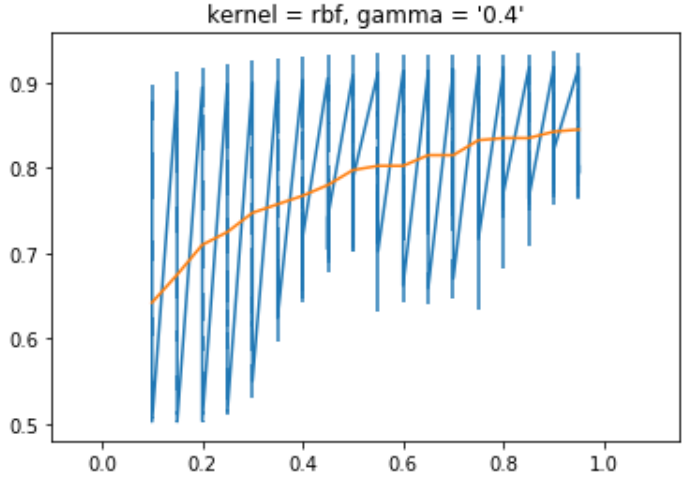


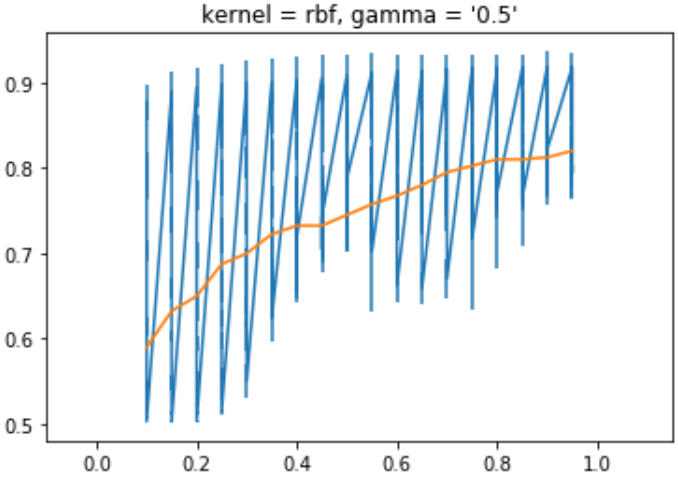
Ilustración 24. Precisión del modelo SVM para la clasificación de la toxicidad con kernel polinomial y gamma=1. La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el testeo.

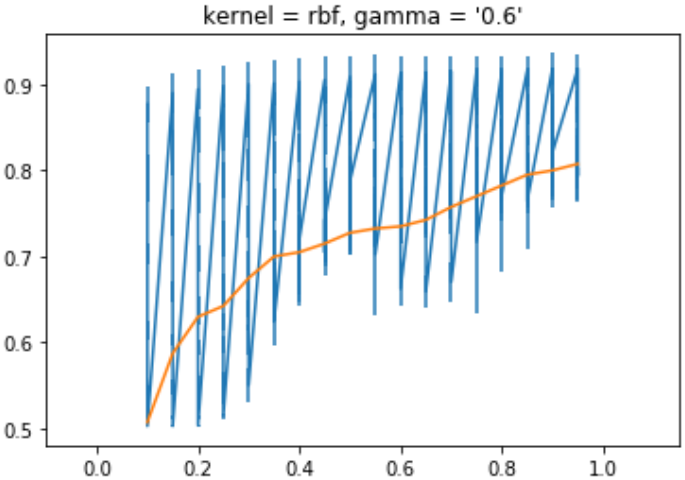


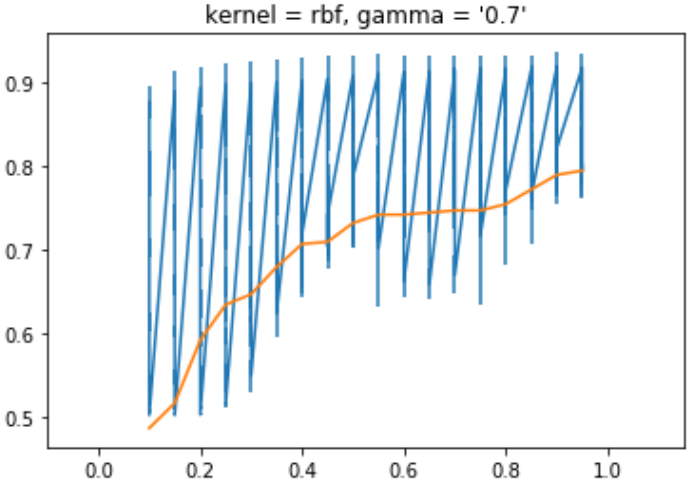

kernel = rbf, gamma = '0.2'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


kernel = rbf, gamma = '0.3'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


kernel = rbf, gamma = '0.4'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


kernel = rbf, gamma = '0.5'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


kernel = rbf, gamma = '0.6'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


kernel = rbf, gamma = '0.7'
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0

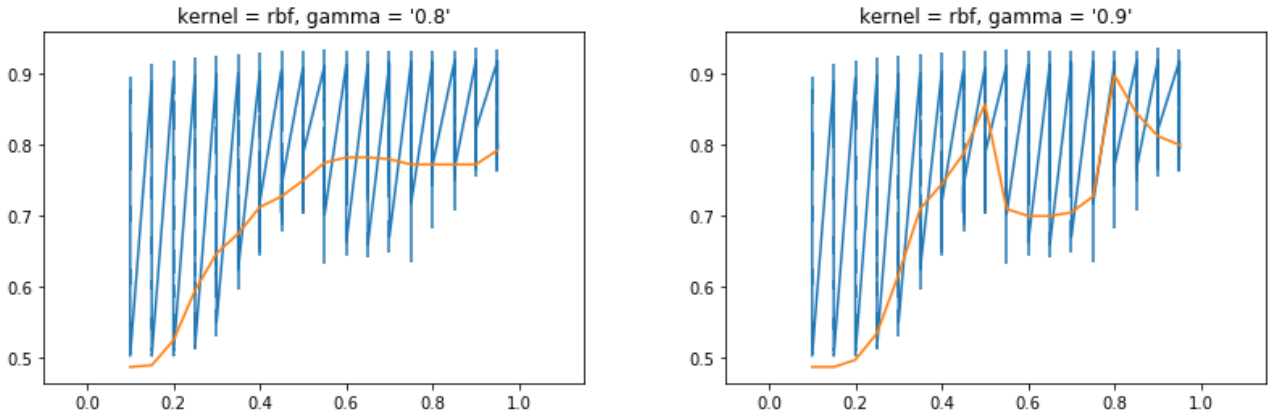


Ilustración 25. *Las diez imágenes anteriores muestran la precisión del modelo SVM para la clasificación de la actividad antimicrobiana con kernel rbf y gamma=0.1 – 0.9 (cada imagen con un aumento de 0.1 con respecto a la anterior). La línea azul marca los valores durante el entrenamiento, las líneas verticales muestras las barras de error; y la línea naranja es la precisión durante el testeo de modelo.*

En este caso el modelo tiene buenos resultados en casi todos los casos, superando en casi todas las gráficas el 85%. Analizando el parámetro C vemos que en *kernel* lineal el aumento del valor genera mejores resultados en el entrenamiento, pero durante el testeo, a partir de 0.6, se mantiene una precisión estable. En el caso polinomial el modelo mantiene los valores del porcentaje, aunque se aumente la exigencia del modelo la precisión sigue siendo del mismo valor. Esto ocurre tanto durante el entrenamiento y el testeo.

Es destacable que, según se reduce valor de gamma en el *kernel rbf*, la precisión a valores bajos de C es muy inferior a los que se obtienen con los valores más altos, casi doblándose. En otras palabras, según se va aumentando gamma la precisión del modelo se reduce, sin embargo, cuando se aumenta *C* para un valor de gamma fijo se consigue mejorar estos porcentajes. También destaca de este *kernel* que, durante el entrenamiento, los valores tienen una variabilidad muy alta, producida por el tipo de función que se aplica para el *rbf*.

El mejor modelo se consigue con el *kernel rbf* con *gamma* = 1 y *C* = 0.1, que son los hiperparámetros óptimos especificados por el programa. En la gráfica que tiene las características similares a las mencionadas la precisión es de casi del 93%, un resultado muy favorable para un clasificador. Sin embargo, también es destacable el *kernel* polinomial ya que, independientemente de los parámetros que se usen, el valor de la precisión (~87%). Esto puede ser más importante que el porcentaje de la precisión, ya que al no disminuir pese a aumentar la exigencia, más adelante puede abrir la puerta a trabajar con un grado de requisito mayor sin afectar al modelo.

Estos dos modelos fueron guardados con el programa **Save_Train_Model.py** y se utilizaron para predecir una muestra de proteínas que solo contenía un tipo de clase. Como se conocen los resultados que deberían obtener de antemano, se sabe qué valores debería clasificar el modelo y, por lo tanto, se puede comprobar de nuevo su eficacia. También se genera una matriz de confusión con la que se puede analizar los errores que comete, pero que salen con el programa de guardado.

Matriz de confusión:

Como análisis de fiabilidad de los resultados del testeo del modelo, se analiza una muestra que tiene solo péptidos con actividad microbiana, es decir, que todos pertenecen a la misma categoría que obtenemos de las bases de datos originales. Los resultados esperados se dan en forma de matriz, con los números de aciertos y errores para cada categoría.

En este caso se utilizó la función *confusión_matrix*. Además, se representan estos valores de forma gráfica con un *heatmap* del módulo *seaborn*. De esta forma se puede visualizar de forma fácil los resultados que obtenemos para cada uno de los modelos.
La suma de las filas y las columnas dan el número total de muestras clasificadas.

Kernel rbf: $\begin{pmatrix} 454 & 40 \\ 27 & 479 \end{pmatrix}$ **Kernel polinomial:** $\begin{pmatrix} 452 & 42 \\ 43 & 463 \end{pmatrix}$

Se puede obtener la precisión si sumamos las casillas 1 y 4 y se dividen entre el número total de muestras. Al multiplicarlo por 100 se consigue en porcentaje la precisión, el número de veces que se clasifica bien la categoría a la que pertenece la secuencia.

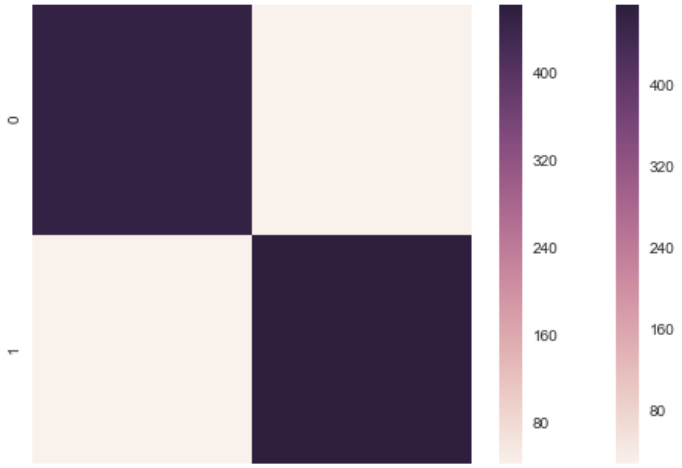


Ilustración 26. *Representación de la matriz confusión como un* heatmap del kernel rbf, que es igual al del kernel polinomial. Los valores de los ejes van *de 0 a 1 y la leyenda de colores representa los valores absolutos.*

También se puede saber si el modelo tiene alguna tendencia a la hora clasificar, por ejemplo, si catalogará bien una variable muchas veces y no lo hará con la otra.

En ninguno de los modelos se observa una tendencia clara, aunque el *kernel rbf* se ve que en una variable falla casi el doble que en la otra. Pese a todo, ambas matrices arrojan resultados muy positivos, tanto en precisión como con la tendencia.

3. Conclusiones

Los siguientes pasos a seguir pueden estar enfocados a mejorar la herramienta, por ejemplo, empleando nuevas formas de clasificar, más datos para poder mejorar la precisión… También se pueden buscar otras categorías en las cuales las mismas variables permitan una buena precisión y, de esta forma, ver qué actividades tienen una relación.

En cualquiera de los casos, una conclusión que parece obvia es que la bioinformática, en concreto la minería de datos y el *machine learning*, permiten trabajar para inferir conocimiento biológico de una forma relativamente sencilla. Permite aprovechar la enorme cantidad de datos que se generan en los últimos años explotando toda la información que de ellos se puede desprender.

Principalmente, en este trabajo se ha conseguido un modelo que predice la actividad antimicrobiana con un porcentaje superior al 90% con un *kernel rbf* y otro con un 87% con un *kernel poly*. Ambos son válidos para el volumen datos utilizados, aunque es probable que, si se quiere escalar y emplearlos para cantidades de datos más grandes, es posible que el modelo polinomial sea más robusto y mantenga valores de precisión altos pese a asumir una mayor exigencia.

Por otro lado, el modelo de toxicidad no funciona de forma eficiente. Hay varias posibilidades, pero como se sabe que han hecho herramientas con la misma técnica y resultados superiores al 90%, deberían replantearse las variables predictivas y utilizar otro tipo de información para conseguir anticipar esta característica de una secuencia *aminoacídica.*

4. Glosario

- *Antimicrobial Peptide (*AMP*).*
- *Antimicrobial Peptides Database (*APD3*).*
- *Cationic Antimicrobial Peptides (CAMP)*.
- Quantitative Matrix (*QM)*.
- *Organización Mundial de la Salud (*OMS*).*
- *Resistencia a los Antimicrobianos (*RAM*).*
- *Support Vector Machine (*SVM*).*
- *Trabajo Fin de Máster (*TFM*).*

5. Bibliografía

Castañeda-casimiro, J., Ortega-roque, J. A., Marcela, A., Aquino-andrade, A., Serafín-lópez, J., Estrada-parra, S., & Estrada, I. (2009). Péptidos antimicrobianos: péptidos con múltiples funciones. *Alergia, Asma E Inmunología*, *18*, 16–29.

Chioro, A., Coll-Seck, A. M., Hoie, B., Moeloek, N., Motsoaledi, A., Rajatanavin, R., & Touraine, M. (2015). Antimicrobial resistance: a priority for global health action. *Bulletin of the World Health Organization*, *93*(7), 439. https://doi.org/10.2471/BLT.15.158998

Clark, W. T., & Radivojac, P. (2011). Analysis of protein function and its prediction from amino acid sequence. *Proteins: Structure, Function and Bioinformatics*, *79*(7), 2086–2096. https://doi.org/10.1002/prot.23029

De Ferrari, L., & Mitchell, J. B. O. (2014). From sequence to enzyme mechanism using multi-label machine learning. *BMC Bioinformatics*, *15*(1), 150. https://doi.org/10.1186/1471-2105-15-150

Dubois, D., Prasadarao, N. V., Mittal, R., Bret, L., Roujou-Gris, M., & Bonnet, R. (2009). CTX-M β-lactamase production and virulence of Escherichia coli K1. *Emerging Infectious Diseases*, *15*(12), 1988–1990. https://doi.org/10.3201/eid1512.090928

Gupta, S., Kapoor, P., Chaudhary, K., Gautam, A., Kumar, R., & Raghava, G. P. S. (2013). In Silico Approach for Predicting Toxicity of Peptides and Proteins. *PLoS ONE*, *8*(9). https://doi.org/10.1371/journal.pone.0073957

Lata, S., Sharma, B., & Raghava, G. (2007). Analysis and prediction of antibacterial peptides. *BMC Bioinformatics*, *8*(1), 263. https://doi.org/10.1186/1471-2105-8-263

Lira, F., Perez, P. S., Baranauskas, J. A., & Nozawa, S. R. (2013). Prediction of antimicrobial activity of synthetic peptides by a decision tree model. *Applied and Environmental Microbiology*, *79*(10), 3156–3159. https://doi.org/10.1128/AEM.02804-12

Nordberg, P., Monnet, D. L., Cars, O., Lodato, B. E. M., & Kaplan, W. (2013). Priority Medicines for Europe and the World “A Public Health Approach to Innovation” Background Paper 6.1 Antimicrobial resistance, (April).

Omardien, S., Brul, S., & Zaat, S. A. J. (2016). Antimicrobial Activity of Cationic Antimicrobial Peptides against Gram-Positives: Current Progress Made in Understanding the Mode of Action and the Response of Bacteria. *Frontiers in Cell and Developmental Biology*, *4*(October), 1–16. https://doi.org/10.3389/fcell.2016.00111

Rajagopalan, B., & Krovi, R. (2002). Data Mining Algorithms. *Data warehousing and web engineering*, *77*.

Robert, C., & Casella, G. (2010). Introducing Monte Carlo Methods with R. https://doi.org/10.1007/978-1-4419-1576-4

Rohl, C. A. (2005). Protein structure estimation from minimal restraints using Rosetta. *Methods in Enzymology*, *394*, 244–260. https://doi.org/10.1016/S0076-6879(05)94009-3

Salud, O. M. de la. (2011). OMS | Resistencia a los antimicrobianos. *Who Media Centre*.

Téllez, G. A., & Castaño, J. C. (2010). Péptidos antimicrobianos. *Grupo de Inmunología Molecular*, *14*(1), 55–67. https://doi.org/10.1016/S0123-9392(10)70093-X

Torrent, M., Di Tommaso, P., Pulido, D., Nogués, M. V., Notredame, C., Boix, E., & Andreu, D. (2012). AMPA: An automated web server for prediction of protein antimicrobial regions. *Bioinformatics*, *28*(1), 130–131. https://doi.org/10.1093/bioinformatics/btr604

Xiong, Y. Q., Mukhopadhyay, K., Yeaman, M. R., Adler-moore, J., & Bayer, A. S. (2005). Functional Interrelationships between Cell Membrane and Cell Wall in Antimicrobial Peptide-Mediated Killing of Staphylococcus aureus Functional Interrelationships between Cell Membrane and Cell Wall in Antimicrobial Peptide-Mediated Killing of Staphylococ. *Antimicrobial Agents and Chemotherapy*, *49*(8), 3114–3121. https://doi.org/10.1128/AAC.49.8.3114

Zhou, P., & Huang, J. (2015). Computational peptidology. *Computational Peptidology*, (402), 1–333. https://doi.org/10.1007/978-1-4939-2285-7

Zvarich, V. I., Stasevich, M. V., Stan'ko, O. V., Komarovskaya-Porokhnyavets, E. Z., Poroikov, V. V., Rudik, A. V., … Novikov, V. P. (2014). Computerized Prediction, Synthesis, and Antimicrobial Activity of New Amino-Acid Derivatives of 2-Chloro-N-(9,10-Dioxo-9,10-Dihydroanthracen-1-Yl)Acetamide. *Pharmaceutical Chemistry Journal*, *48*(9), 582–586. https://doi.org/10.1007/s11094-014-1154-z

Recursos web:

- Base de datos AMPs: http://aps.unmc.edu/AP/main.php
- Biopython: http://biopython.org/
- Biopython/Github: https://github.com/biopython/biopython
- Google Scholar: https://scholar.google.es/
- Organización Mundial de la Salud: http://www.who.int/es/
- Paquete numpy: http://www.numpy.org/
- Paquete pandas: https://pypi.python.org/pypi/pandas/
- Paquete scikit-learn: http://scikit-learn.org/stable/
- PubMed: http://www.ncbi.nlm.nih.gov/pubmed
- Python: https://www.python.org/
- Quora: https://es.quora.com/
- Research Gate: https://www.researchgate.net/
- Stackover Flow: https://stackoverflow.com/
- UniProt: http://www.uniprot.org/
- Wikipedia: https://www.wikipedia.org/

6. Anexo

Programa: **Database.py**

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 18 17:55:07 2017

@author: Alvaro Mariño Solís
"""

from sklearn import preprocessing, cross_validation, svm
from sklearn.grid_search import GridSearchCV
from sklearn.preprocessing import Imputer
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.externals import joblib
from sklearn.metrics import confusion_matrix

datasetName = 'C:/TFM/SwissProt/FinalAntimicrobialSample.csv'

raw_data = open(datasetName, 'rt')
reader = csv.reader(raw_data, delimiter = ',', quoting=csv.QUOTE_NONE)
next(reader, None)
x = list(reader)

# Load the CSV file as a numpy matrix
dataset = np.array(x).astype('float')

# separate the data from the target attributes
xRaw = dataset[:,0:dataset.shape[1]-1]
yRaw = dataset[:,dataset.shape[1]-1]

# Set labels for wighting
le = preprocessing.LabelEncoder()
y = le.fit_transform(yRaw)

# missing values
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
xPrep = imp.fit_transform(xRaw)
#Standardize data
scaler = preprocessing.StandardScaler().fit(xPrep)
x=scaler.transform(xPrep)

#Divide in training and test, shuffling the examples and keeping the proportion of examples of each class
xTrain, xTest, yTrain, yTest = cross_validation.train_test_split(x, y, test_size=0.2, random_state=1)
#print('xTrain', xTrain, "\nxTest", xTest, "\nyTrain", yTrain, "\nyTest", yTest)

#Generate grid search
kernels = ['linear', 'poly', 'rbf']

for k in kernels:
    errList, devList, cValues, gValues = [], [], [], []
    gamma = [1]
    cList = list(np.arange(0.1, 1, 0.05))
    if k == 'rbf':
        gamma = list(np.arange(0.1, 1, 0.1))
    hyperParams = {'kernel': [k], 'C': cList, 'gamma': gamma}

    #Create an instance of Tree Classifier and fit the data for the grid parameters
    modelCV = GridSearchCV(svm.SVC(), param_grid=hyperParams, cv=5)
    modelCV.fit(xTrain, yTrain)
    print("Best hyperparameters", modelCV.best_params_)
    errList2, devList2 = [], []
    for hyperP, mean_score, scores in modelCV.grid_scores_:
        print("%0.3f (+/-%0.03f) for %r"
              % (mean_score, scores.std(), hyperP))
        cValues.append(hyperP['C'])
        gValues.append(hyperP['gamma'])
        errList.append(mean_score)
        devList.append(scores.std())
    #Create an instance of Tree Classifier with the best hyperparameters and weight: auto and the full training set
    model = svm.SVC(kernel = modelCV.best_params_['kernel'],
        C = modelCV.best_params_['C'],
        gamma = modelCV.best_params_['gamma'])
    model.fit(xTrain, yTrain)


```

```
    #Test
    for g in gamma:
      testError = []
      for c in cList:
          model = svm.SVC(kernel = k, C = c, gamma = g)
          model.fit(xTrain, yTrain)
          testError.append(model.score(xTest, yTest))

      cValuesForGamma = [x for index, x in enumerate(cValues) if gValues[index] == g]

      print("Best hyperparameters", modelCV.best_params_)
      plt.title("kernel = %s, gamma = '%s'" % (k, g))

      model.predict(xTest)

      plt.errorbar(cValues, errList)
      plt.errorbar(cValuesForGamma, testError)
      plt.xlim(cValues[0]-0.2, cValues[len(cValues)-1]+0.2)
      plt.show()

```

Programa: **Antimicrobial.py**

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 17 18:36:46 2017

@author: Ramon Mariño Solís
"""

import pandas as pd
from Bio.SeqUtils.ProtParam import ProteinAnalysis

file = 'C:/TFM/SwissProt/AntimicrobialDatabase.xlsx'
file2 = 'C:/TFM/SwissProt/NotAntimicrobialDatabase.xlsx'

Antimicrobial_Database = pd.read_excel(file)
NotAntimicrobial_Database = pd.read_excel(file2)
Antimicrobial_Analysis = Antimicrobial_Database['Sequence'].apply(ProteinAnalysis)
NotAntimicrobial_Analysis = NotAntimicrobial_Database['Sequence'].apply(ProteinAnalysis)

aminoacids = list()
isoelectricpoint = list()
aromaticity = list()
seq = range(0, len(Antimicrobial_Analysis))

for i in list(seq):
   aminoacids.append(Antimicrobial_Analysis[i].get_amino_acids_percent())
   isoelectricpoint.append(Antimicrobial_Analysis[i].isoelectric_point())
   aromaticity.append(Antimicrobial_Analysis[i].aromaticity())

aminoacidosFinalDict = dict([(k, [x[''+k+''] \
            for x in aminoacids]) for k in aminoacids[0]])

AntimicrobialPeptides = pd.DataFrame()
AntimicrobialPeptides['Mass'] = Antimicrobial_Database['Mass']
AntimicrobialPeptides['Length'] = Antimicrobial_Database['Length']
AntimicrobialPeptides['Aromaticity'] = aromaticity
AntimicrobialPeptides['IsoelectricPoint'] = isoelectricpoint

for k,value in aminoacidosFinalDict.items():
    AntimicrobialPeptides[k] = value
```

```
for k,value in aminoacidosFinalDict.items():
    AntimicrobialPeptides[k] = value

AntimicrobialPeptides['AntimicrobialActivity']=[1 for x in seq]

aminoacids = list()
isoelectricpoint = list()
aromaticity = list()
seq2 = range(0, len(NotAntimicrobial_Analysis))

amino2 = dict()

for i in list(seq2):
   aminoacids.append(NotAntimicrobial_Analysis[i].get_amino_acids_percent())
   isoelectricpoint.append(NotAntimicrobial_Analysis[i].isoelectric_point())
   aromaticity.append(NotAntimicrobial_Analysis[i].aromaticity())

aminoacidosFinalDict = dict([(k, [x[''+k+''] \
    for x in aminoacids]) for k in aminoacids[0]])

NotAntimicrobialPeptides = pd.DataFrame()
NotAntimicrobialPeptides['Mass'] = NotAntimicrobial_Database['Mass']
NotAntimicrobialPeptides['Length'] = NotAntimicrobial_Database['Length']
NotAntimicrobialPeptides['Aromaticity'] = aromaticity
NotAntimicrobialPeptides['IsoelectricPoint'] = isoelectricpoint

for k,value in aminoacidosFinalDict.items():
    NotAntimicrobialPeptides[k] = value

NotAntimicrobialPeptides['AntimicrobialActivity']=[0 for x in seq2]

AntimicrobialSample=pd.DataFrame(AntimicrobialPeptides.sample(n=1000, random_state=6668452))
NotAntimicrobialSample = pd.DataFrame(NotAntimicrobialPeptides.sample(n=1000, random_state=6668452))
FinalAntimicrobialSample = pd.concat([AntimicrobialSample, NotAntimicrobialSample])

BigAntimicrobialSample=pd.DataFrame(AntimicrobialPeptides.sample(n=2500, random_state=6668452))
BigNotAntimicrobialSample = pd.DataFrame(NotAntimicrobialPeptides.sample(n=2500, random_state=6668452))
BigFinalAntimicrobialSample = pd.concat([BigAntimicrobialSample, BigNotAntimicrobialSample])
BigFinalAntimicrobialSample.to_csv('C:/TFM/SwissProt/BigFinalAntimicrobialSample.csv', index=False)

FinalAntimicrobialSample.to_csv('C:/TFM/SwissProt/FinalAntimicrobialSample.csv', index=False)
NotAntimicrobialSample.to_csv('C:/TFM/SwissProt/NotAntimicrobialSample.csv', index=False)
AntimicrobialSample.to_csv('C:/TFM/SwissProt/AntimicrobialSample.csv', index=False)

FinalAntimicrobialDataBase=pd.concat([AntimicrobialPeptides, NotAntimicrobialPeptides])
FinalAntimicrobialDataBase.to_csv('C:/TFM/SwissProt/FinalAntimicrobialDatabase.csv', index=False)
```

Programa: **svmClassificacion1.py**

```
from sklearn import preprocessing, cross_validation, svm
from sklearn.grid_search import GridSearchCV
from sklearn.preprocessing import Imputer
import csv
import numpy as np
import matplotlib.pyplot as plt

datasetName = 'C:/TFM/SwissProt/FinalSample.csv'

raw_data = open(datasetName, 'rt')
reader = csv.reader(raw_data, delimiter = ',', quoting=csv.QUOTE_NONE)
next(reader, None)
x = list(reader)

# Load the CSV file as a numpy matrix
dataset = np.array(x).astype('float')

# separate the data from the target attributes
xRaw = dataset[:,0:dataset.shape[1]-1]
yRaw = dataset[:,dataset.shape[1]-1]

# Set labels for wighting
le = preprocessing.LabelEncoder()
y = le.fit_transform(yRaw)

# missing values
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
xPrep = imp.fit_transform(xRaw)
#Standardize data
scaler = preprocessing.StandardScaler().fit(xPrep)
x=scaler.transform(xPrep)

#Divide in training and test, shuffling the examples and keeping the proportion of examples of each class
xTrain, xTest, yTrain, yTest = cross_validation.train_test_split(x, y, test_size=0.2, random_state=1)
#print('xTrain', xTrain, "\nxTest", xTest, "\nyTrain", yTrain, "\nyTest", yTest)

#Generate grid search
kernels = ['linear', 'poly', 'rbf']

for k in kernels:
    errList, devList, cValues, gValues = [], [], [], []
    gamma = [1]
    cList = list(np.arange(0.1, 1, 0.05))
    if k == 'rbf':
        gamma = list(np.arange(0.1, 1, 0.1))
    hyperParams = {'kernel': [k], 'C': cList, 'gamma': gamma}

    #Create an instance of Tree Classifier and fit the data for the grid parameters
    modelCV = GridSearchCV(svm.SVC(), param_grid=hyperParams, cv=5)
    modelCV.fit(xTrain, yTrain)
    print("Best hyperparameters", modelCV.best_params_)
    errList2, devList2 = [], []
    for hyperP, mean_score, scores in modelCV.grid_scores_:
        print("%0.3f (+/-%0.03f) for %r"
              % (mean_score, scores.std(), hyperP))
        cValues.append(hyperP['C'])
        gValues.append(hyperP['gamma'])
        errList.append(mean_score)
        devList.append(scores.std())
    #Create an instance of Tree Classifier with the best hyperparameters and weight: auto and the full training set
    model = svm.SVC(kernel = modelCV.best_params_['kernel'],
        C = modelCV.best_params_['C'],
        gamma = modelCV.best_params_['gamma'])
    model.fit(xTrain, yTrain)


    #Test
    for g in gamma:
      testError = []
      for c in cList:
          model = svm.SVC(kernel = k, C = c, gamma = g)
          model.fit(xTrain, yTrain)
          testError.append(model.score(xTest, yTest))

      cValuesForGamma = [x for index, x in enumerate(cValues) if gValues[index] == g]

      print("Best hyperparameters", modelCV.best_params_)
      plt.title("kernel = %s, gamma = '%s'" % (k, g))

      model.predict(xTest)

      plt.errorbar(cValues, errList, yerr = devList)
      plt.errorbar(cValuesForGamma, testError)
      plt.xlim(cValues[0]-0.2, cValues[len(cValues)-1]+0.2)
      plt.show()

```

Programa: **Save_Train_Model.py**

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 18 19:38:51 2017

@author: Alvaro Mariño Solís
"""

from sklearn import preprocessing, cross_validation, svm
from sklearn.preprocessing import Imputer
import csv
import numpy as np
from sklearn.externals import joblib
from sklearn.metrics import confusion_matrix

datasetName = 'C:/TFM/SwissProt/BigFinalAntimicrobialSample.csv'

raw_data = open(datasetName, 'rt')
reader = csv.reader(raw_data, delimiter = ',', quoting=csv.QUOTE_NONE)
next(reader, None)
x = list(reader)
print(len(x))

# load the CSV file as a numpy matrix
dataset = np.array(x).astype('float')

# separate the data from the target attributes
xRaw = dataset[:,0:dataset.shape[1]-1]
yRaw = dataset[:,dataset.shape[1]-1]

# Set labels for wighting
le = preprocessing.LabelEncoder()
y = le.fit_transform(yRaw)

# missing values
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
xPrep = imp.fit_transform(xRaw)
#Standardize data
scaler = preprocessing.StandardScaler().fit(xPrep)
x=scaler.transform(xPrep)

#Divide in training and test, shuffling the examples and keeping the proportion of examples of each clas
xTrain, xTest, yTrain, yTest = cross_validation.train_test_split(x, y, test_size=0.2, random_state=1)
#Save best model

def saveModel(kernel, c, g, name):
    model = svm.SVC(kernel = kernel, C = c, gamma = g)
    model.fit(xTrain, yTrain)

    yPred=model.predict(xTest)
    cnf_matrix = confusion_matrix(yTest, yPred)
    print(cnf_matrix)

    joblib.dump(model, 'C:/TFM/'+ name +'.pkl')

saveModel('rbf',0.9, 0.1, 'ampModelRbf')
saveModel('poly',0.1, 1, 'ampModelPoly')


```

Programa: **test.py**

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 18 19:12:39 2017

@author: Ramon Mariño Solís
"""
from sklearn import preprocessing
from sklearn.preprocessing import Imputer
import csv
import numpy as np
from sklearn.externals import joblib

model=joblib.load('C:/TFM/ampModel.pkl')
dataset = 'C:/TFM/SwissProt/NotAntimicrobialSample.csv'
dataset2 = 'C:/TFM/SwissProt/AntimicrobialSample.csv'

def preprocessData(dataset):
    raw_data = open(dataset, 'rt')
    reader = csv.reader(raw_data, delimiter = ',', quoting=csv.QUOTE_NONE)
    next(reader, None)
    x = list(reader)

    # load the CSV file as a numpy matrix
    dataset = np.array(x).astype('float')

    # separate the data from the target attributes
    xRaw = dataset[:,0:dataset.shape[1]-1]

    # missing values
    imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
    xPrep = imp.fit_transform(xRaw)
    #Standardize data
    scaler = preprocessing.StandardScaler().fit(xPrep)
    x=scaler.transform(xPrep)
    return x

def getResults(x):
    prediction = model.predict(x)
    predictionAccuracy = len([x for x in prediction if x == 1]) / len(prediction)

    print(len([x for x in prediction if x == 1]))
    print(len(prediction))
    print(predictionAccuracy)

x = preprocessData(dataset)
x2 = preprocessData(dataset2)

getResults(x)
getResults(x2)
```