Desarrollo de un modelo predictivo basado en el aprendizaje supervisado para la predicción de resistencia a los antibióticos carbapenémicos en *Klebsiella pneumoniae*

Judit García Fernández

MU Bioinf. i Bioest.

Bioinformática Estadística y Aprendizaje Automático

Tutor/a de TF

Romina Rebrij

Profesor/a responsable de la asignatura

Agnès Pérez

04/06/2025

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de un modelo predictivo basado en el aprendizaje supervisado para la predicción de resistencia a los antibióticos carbapenémicos en Klebsiella pneumoniae*
Nombre del autor:	*Judit García Fernández*
Nombre del director/a:	*Romina Rebrij*
Nombre del PRA:	*Agnès Pérez*
Fecha de entrega:	*04/06/2025*
Titulación o programa:	*Máster U. en Bioinformática y Bioestadística UOC-UB*
Area del Trabajo Final:	*Bioinformática Estadística y Aprendizaje Automático*
Idioma del trabajo:	*Castellano*
Palabras clave	*Machine Learning, resistencia bacteriana, Klebsiella pneumoniae, carbapenémicos, genómica, aprendizaje supervisado*

Resumen del Trabajo

La resistencia a los antibióticos, especialmente a los carbapenémicos, es una amenaza creciente en la salud pública, particularmente en bacterias como *Klebsiella pneumoniae*. Este trabajo tiene como objetivo desarrollar un modelo de machine learning capaz de predecir si una cepa de *Klebsiella pneumoniae* es resistente o sensible a carbapenémicos utilizando datos genómicos públicos. El enfoque metodológico incluye la identificación y selección de datos de bases de datos públicas, la extracción de características genómicas relevantes, y el diseño y optimización de modelos supervisados de aprendizaje automático. Este modelo puede ser útil en la investigación biomédica y en el ámbito clínico, mejorando la precisión de los diagnósticos y optimizando el uso de antimicrobianos.

Abstract

Antibiotic resistance, especially to carbapenems, is a growing public health threat, particularly in bacteria such as *Klebsiella pneumoniae*. This work aims to develop a machine learning model capable of predicting whether a *Klebsiella pneumoniae* strain is resistant or susceptible to carbapenems using public genomic data. The methodological approach includes identifying and selecting data from public databases, extracting relevant genomic features, and designing and optimizing supervised machine learning models. This model can be useful in biomedical research and clinical settings, improving diagnostic accuracy and optimizing the use of antimicrobials.

ÍNDICE

LISTA DE FIGURAS

1. INTRODUCCIÓN

La resistencia a los antibióticos representa una amenaza creciente para la salud pública a nivel mundial, con proyecciones alarmantes sobre su impacto futuro. En este contexto, el aprendizaje automático (machine learning) ha emergido como una herramienta prometedora para predecir la resistencia antimicrobiana en diversas bacterias, facilitando diagnósticos más rápidos y tratamientos más efectivos. [1]

Diversos estudios han demostrado la eficacia de las técnicas de machine learning en la identificación de patrones de resistencia. Por ejemplo, se han utilizado modelos supervisados y no supervisados para predecir la resistencia antibiótica, apoyando a los clínicos en la selección de terapias adecuadas [2]. Además, la implementación de algoritmos de deep learning ha mejorado la precisión de las predicciones, reduciendo tanto falsos positivos como falsos negativos [3].

La integración de datos genómicos y fenotípicos en modelos de machine learning ha permitido la detección temprana de cepas resistentes, optimizando el uso de antimicrobianos y mejorando el pronóstico de pacientes vulnerables. Asimismo, se han desarrollado modelos predictivos que, al analizar características clínicas y demográficas de los pacientes, logran una alta concordancia con los perfiles de resistencia observados experimentalmente [4].

En resumen, el uso del aprendizaje automático en la predicción de la resistencia antibiótica ofrece una vía innovadora para abordar la crisis de la resistencia antimicrobiana, permitiendo diagnósticos más precisos y tratamientos personalizados que podrían mitigar el impacto de las infecciones resistentes en la población global.

1.1. CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO

La resistencia a los antibióticos es uno de los mayores desafíos en salud pública a nivel global. El aumento de bacterias multirresistentes compromete la eficacia de los tratamientos antimicrobianos, incrementando la mortalidad y los costos sanitarios [5]. En particular, *Klebsiella pneumoniae* ha sido identificada como una de las principales amenazas debido a su capacidad para adquirir genes de resistencia, incluyendo aquellos que confieren resistencia a los carbapenémicos, antibióticos de última línea utilizados en infecciones graves [6].

Klebsiella pneumoniae es un patógeno oportunista asociado a infecciones nosocomiales como neumonías, infecciones del tracto urinario y septicemias. Su capacidad de diseminar mecanismos de resistencia, como las carbapenemasas, ha generado un aumento alarmante en cepas resistentes, limitando significativamente las opciones terapéuticas y aumentando la tasa de mortalidad en pacientes infectados.

En este contexto, la bioinformática y el aprendizaje automático han surgido como herramientas clave para abordar la resistencia bacteriana. El análisis de datos genómicos mediante modelos de aprendizaje supervisado ha demostrado ser una estrategia eficaz para predecir la resistencia antibiótica, permitiendo una detección más rápida y precisa en entornos clínicos [7]. Estas metodologías facilitan la identificación de patrones en los datos genómicos, optimizando la toma de decisiones en tratamientos antimicrobianos [8].

Este Trabajo Final de Máster tiene como objetivo desarrollar un modelo predictivo basado en aprendizaje supervisado para la predicción de resistencia a los antibióticos carbapenémicos en *Klebsiella pneumoniae*. Para ello, se emplearán datos genómicos de cepas clínicamente caracterizadas, que serán analizados mediante técnicas avanzadas de machine learning. Se espera que los resultados obtenidos contribuyan a mejorar la detección temprana de resistencia bacteriana y proporcionen herramientas que optimicen la gestión de infecciones en el ámbito clínico.

1.2. OBJETIVOS DEL TRABAJO

Objetivo general: Desarrollar un modelo de machine learning capaz de predecir si una cepa de *Klebsiella pneumoniae* es resistente o sensible a carbapenémicos utilizando datos genómicos públicos, alcanzando una precisión mínima del 80% en la clasificación.

Objetivos específicos:

1. Identificar y seleccionar datos de bases de datos públicas con genomas anotados y perfiles fenotípicos de resistencia.
2. Extraer y procesar características genómicas relevantes, como genes de resistencia y mutaciones, asegurando la calidad y representatividad de los datos.
3. Diseñar, entrenar y optimizar diferentes modelos de aprendizaje supervisado para la predicción de resistencia a carbapenémicos.
4. Comparar y evaluar el desempeño de los modelos utilizando métricas como AUC-ROC, precisión y sensibilidad, estableciendo un umbral mínimo del 80% de precisión para el modelo final.
5. Interpretar los resultados para identificar patrones genómicos clave asociados con la resistencia, proporcionando información útil para la investigación biomédica.

1.3. IMPACTO EN SOSTENIBILIDAD, ÉTICO-SOCIAL Y DE DIVERSIDAD

Este trabajo busca cumplir con la competencia ética y global (CECG) mediante un desarrollo honesto, ético, sostenible, socialmente responsable y respetuoso con los derechos humanos y la diversidad. A continuación, se detallan las consideraciones en cada dimensión:

1.I.1 Sostenibilidad

(a) Este trabajo se basa en datos genómicos previamente generados y disponibles en bases de datos públicas. La reutilización de estos datos evita la generación de nuevas muestras, lo que reduce el consumo de reactivos, energía y otros recursos necesarios para la secuenciación de ADN, minimizando así el impacto ambiental.

(b) Dado que el desarrollo del proyecto se realiza completamente mediante herramientas computacionales, el impacto ambiental es significativamente menor en comparación con estudios experimentales en laboratorio, que requieren productos desechables y consumibles difíciles de gestionar.

(c) El modelo de machine learning desarrollado podría contribuir a un uso más eficiente de los antibióticos al predecir la resistencia antibacteriana, lo que puede ayudar a reducir la aparición de cepas multirresistentes. Esto tiene un impacto positivo en la salud pública y en la reducción del consumo innecesario de antibióticos, alineándose con los ODS 3 (Salud y bienestar) y 12 (Producción y consumo responsables).

1.I.2 Ética-social

(a) La predicción de resistencia antibacteriana permitirá una mejor gestión de los tratamientos antibióticos, optimizando su uso y reduciendo la propagación de bacterias resistentes. Esto contribuirá al bienestar de los pacientes y a la mejora de la salud pública.

(b) Se respetarán las normativas de protección de datos y privacidad, asegurando que los datos utilizados en este estudio sean de acceso público y anonimizado, sin comprometer la identidad de los pacientes.

(c) El modelo propuesto será de código abierto para fomentar la transparencia, la reproducibilidad y el acceso equitativo a su implementación en distintos entornos de investigación y salud pública.

1.I.3 Diversidad

(a) Se utilizarán referencias bibliográficas sin sesgo de género, raza, etnia, orientación sexual o posición social. Las citas incluirán nombres completos para reflejar la diversidad de los autores.

(b) Los datos genómicos empleados en el estudio han sido seleccionados sin discriminación por género, etnia u origen, garantizando así que los resultados obtenidos sean aplicables a diversas poblaciones y no reflejen sesgos involuntarios.

(c) En la redacción del trabajo se empleará un lenguaje inclusivo y una redacción en tercera persona para garantizar una comunicación neutral y accesible.

Con estas consideraciones, este trabajo busca contribuir positivamente a la investigación biomédica desde una perspectiva sostenible, ética y socialmente responsable.

1.4. ENFOQUE Y MÉTODO SEGUIDO

Para este trabajo se seguirá un enfoque basado en bioinformática y aprendizaje automático supervisado, dado que se dispone de datos genómicos etiquetados con información sobre resistencia o sensibilidad a carbapenémicos. A continuación, se detallan las etapas metodológicas, las herramientas a utilizar y la justificación de su selección.

1. **Obtención y preprocesamiento de datos**
    - Se recopilarán datos genómicos de bases de datos públicas como el NCBI y el Patric Database, que incluyen secuencias completas de Klebsiella pneumoniae con información sobre resistencia a antibióticos.
    - Se limpiarán los datos eliminando secuencias con baja calidad o metadatos incompletos.
    - Se normalizarán los datos en un formato adecuado para el análisis posterior.
2. **Selección de algoritmos de aprendizaje automático**
    - Se evaluarán distintos enfoques de aprendizaje automático para la tarea de clasificación:

- **Árboles de decisión y ensamblados**: Modelos como Random Forest y XGBoost, conocidos por su robustez en la clasificación de datos genómicos y su capacidad de interpretar la importancia de las variables.
- **Modelos lineales**: Regresión logística y Support Vector Machines (SVM), útiles en problemas con relaciones lineales en los datos.
- **Redes neuronales**: Modelos más complejos como perceptrones multicapa o redes profundas, que pueden capturar patrones no lineales pero requieren más datos y capacidad computacional.

3. **Entrenamiento, evaluación y selección del modelo óptimo**
   - Se utilizarán métricas como AUC-ROC, precisión, sensibilidad y especificidad para evaluar los modelos.
   - Se realizará un ajuste de hiperparámetros mediante técnicas como Grid Search o Bayesian Optimization para mejorar la precisión.
   - Se establecerá un umbral mínimo de rendimiento: el modelo deberá alcanzar al menos un 80% de precisión para considerarse viable.
4. **Interpretación de resultados y validación**
   - Se analizará la importancia de las características utilizando técnicas como SHAP (SHapley Additive exPlanations) o Feature Importance de XGBoost.
   - Se validará la robustez del modelo frente a nuevos datos no utilizados en el entrenamiento.
5. **Publicación y documentación del trabajo**
   - Se compartirá el código y los modelos entrenados en un repositorio público como GitHub, asegurando la reproducibilidad del trabajo.
   - Se generará un informe con los hallazgos clave y recomendaciones para futuras investigaciones.

1.5. PLANIFICACIÓN DEL TRABAJO

Fases del proyecto y tareas

Revisión bibliográfica y selección de bases de datos *(4 – 19 de marzo):*

- Buscar literatura científica sobre resistencia a carbapenémicos en *Klebsiella pneumoniae*.
- Revisar estudios previos sobre modelos de machine learning aplicados a resistencia antibiótica.
- Evaluar la calidad y disponibilidad de bases de datos públicas (PATRIC, CARD, NCBI).
- Seleccionar la base de datos más adecuada para el estudio.

Descarga y preprocesamiento de datos *(19 de marzo – 2 de abril)*:

- Descargar datos genómicos de *Klebsiella pneumoniae* en formato FASTA/GenBank.
- Descargar datos fenotípicos de resistencia (clasificación resistente/sensible).
- Filtrar y limpiar datos eliminando inconsistencias.
- Formatear los datos para que sean utilizables en análisis posteriores.

HITO 1: ***Entrega primera fase del desarrollo del trabajo (2 de abril)***

División del dataset *(2 – 16 abril)*:

- Separación del conjunto en entrenamiento (80%) y prueba (20%) de forma estratificada.

Entrenamiento de modelos de aprendizaje automático *(16 abril – 7 mayo)*:

- Implementación de algoritmos: Random Forest, Support Vector Machines (SVM) y XGBoost.
- Ajuste de hiperparámetros mediante validación cruzada.

HITO 2: ***Entrega segunda fase del desarrollo del trabajo (7 de mayo)***

Evaluación del modelo y ajuste de hiperparámetros *(8 – 22 de mayo)*:

- Ajustar hiperparámetros mediante validación cruzada.
- Comparar métricas de rendimiento (AUC-ROC, precisión, sensibilidad, especificidad).
- Seleccionar el modelo con mejor desempeño.

Interpretación de resultados y redacción de memoria *(22 de mayo – 1 de junio)*:

- Analizar la importancia de las características en la predicción.
- Comparar los resultados con estudios previos.
- Redactar la memoria final con metodología, resultados y discusión.

Revisión final y preparación de la presentación *(1 - 4 de junio)*:

- Revisar y corregir la memoria final.
- Preparar diapositivas y material para la presentación virtual.

HITO 3: ***Entrega del cierre de la memoria y presentación (4 de junio)***

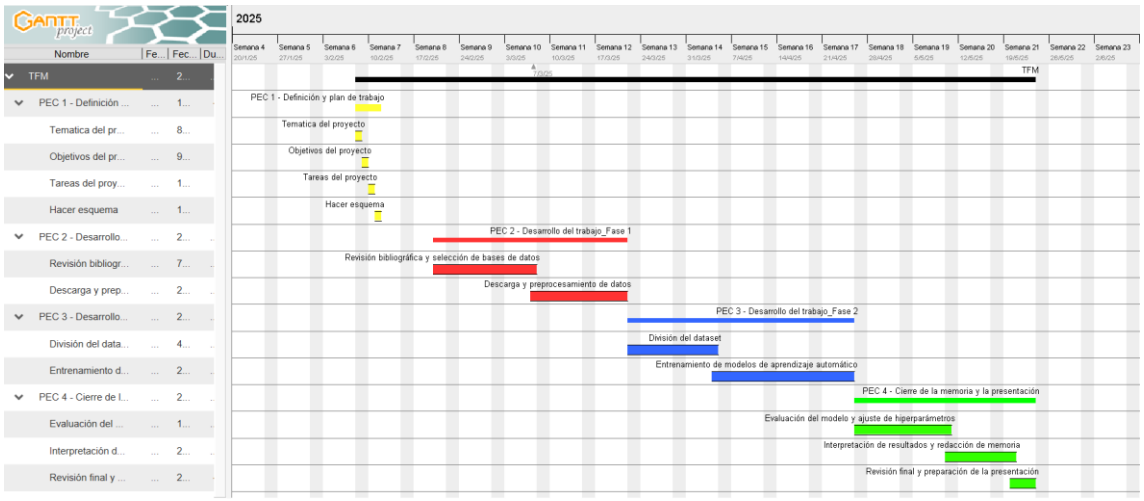


Figura 1. Representación gráfica del cronograma

1.6. BREVE SUMARIO DE PRODUCTOS OBTENIDOS

Este Trabajo Final de Máster ha logrado el desarrollo y la evaluación de un **modelo predictivo de Machine Learning** basado en aprendizaje supervisado, específicamente diseñado para la clasificación de cepas de *Klebsiella pneumoniae* como resistentes o sensibles a antibióticos carbapenémicos. Aunque se exploraron diversas configuraciones, el desafío de alcanzar de forma equilibrada el objetivo de una precisión mínima del 80% y una alta sensibilidad para la clase minoritaria de resistencia fue una limitación clave identificada en los modelos evaluados.

1.7. BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA

La memoria del Trabajo Final de Máster se estructura en los siguientes capítulos para detallar el desarrollo y los hallazgos de la investigación:

- **Capítulo 2: Estado del Arte.** Este capítulo profundiza en el problema de la resistencia a los antibióticos, con especial énfasis en *Klebsiella pneumoniae* y los carbapenémicos. Además, revisa exhaustivamente la aplicación de las técnicas de Machine Learning en la predicción de la resistencia antimicrobiana, presentando los algoritmos más relevantes y los avances recientes en el campo.
- **Capítulo 3: Materiales y métodos.** Se describe de forma detallada el enfoque seguido para el desarrollo del modelo. Incluye la estrategia de obtención y preprocesamiento de los datos genómicos, los métodos de extracción de características bioinformáticas, la selección y configuración de los algoritmos de aprendizaje supervisado utilizados, y las métricas de evaluación empleadas para la validación del modelo.
- **Capítulo 4: Resultados.** En este capítulo se presentan los hallazgos clave derivados de la implementación de la metodología. Se detallan los resultados de la fase de entrenamiento y evaluación de los diferentes modelos, las métricas de

rendimiento obtenidas (como AUC-ROC, precisión, sensibilidad), y se identifica el modelo final seleccionado por su óptimo desempeño.

- **Capítulo 5: Conclusiones.** Se resumen los principales aportes del trabajo y se extraen las conclusiones más relevantes. Se destaca el cumplimiento de los objetivos planteados y se sugieren líneas de investigación futuras derivadas de los resultados.
- **Capítulo 7: Bibliografía.** Contiene la lista completa de todas las fuentes y referencias bibliográficas citadas a lo largo de la memoria.

2. ESTADO DEL ARTE

2.1. INTRODUCCIÓN AL PROBLEMA

La resistencia a los antibióticos es una de las principales amenazas para la salud pública a nivel mundial. Se estima que, de continuar las tendencias actuales, para 2050 las infecciones causadas por bacterias resistentes podrían ser responsables de hasta 10 millones de muertes anuales. En 2019, se registraron aproximadamente 1,27 millones de muertes directamente atribuibles a infecciones resistentes, y cerca de 5 millones de fallecimientos estuvieron relacionados indirectamente con la resistencia antimicrobiana [9].

En este contexto, *Klebsiella pneumoniae* destaca como uno de los patógenos oportunistas más relevantes en entornos hospitalarios. Clasificada por la OMS como una amenaza crítica dentro del grupo de bacterias multirresistentes [10], esta bacteria presenta una elevada capacidad para adquirir y diseminar genes de resistencia, especialmente frente a antibióticos de última línea como los carbapenémicos [11]. Su prevalencia en unidades de cuidados intensivos y su implicación en infecciones graves como neumonías, infecciones urinarias, septicemias y meningitis ha sido ampliamente documentada [12][13]. Además, *K. pneumoniae* se caracteriza por su habilidad para transferir genes de resistencia a través de mecanismos de transferencia horizontal, lo que favorece su rápida propagación entre cepas y otras especies bacterianas [14].

2.2. APLICACIÓN DE MACHINE LEARNING EN LA RESISTENCIA ANTIMICROBIANA

El uso de técnicas de Machine Learning (ML) se ha consolidado como una estrategia prometedora para abordar la resistencia antimicrobiana. Diversos estudios recientes han aplicado algoritmos de ML para analizar datos estructurales de bacterias, cribados químicos y detalles moleculares de los mecanismos de resistencia con el objetivo de identificar patrones y posibles candidatos a nuevos antibióticos [15]. Además, se han desarrollado modelos predictivos que logran alta concordancia entre las características clínicas y demográficas de los pacientes y los perfiles de resistencia observados experimentalmente [16].

Entre los algoritmos más utilizados se encuentran Random Forest, Support Vector Machines (SVM), Gradient Boosting (incluyendo XGBoost) y redes neuronales profundas. Cada uno presenta ventajas y limitaciones: por ejemplo, Random Forest destaca por su robustez frente a datos ruidosos y buena interpretabilidad, pero puede ofrecer menor precisión que modelos más complejos. Por su parte, XGBoost ha demostrado un rendimiento superior en tareas de clasificación binaria, aunque requiere un ajuste fino de hiperparámetros para alcanzar su máximo potencial [17]. Las redes neuronales profundas muestran buenos resultados especialmente cuando se dispone de grandes volúmenes de datos, aunque con menor interpretabilidad y mayor riesgo de sobreajuste.

Además, están emergiendo nuevas estrategias como AutoML, redes neuronales basadas en grafos (Graph Neural Networks) y modelos evolutivos que permiten representar dinámicamente la progresión de la resistencia antibiótica, abriendo nuevas vías para la modelización de estos fenómenos [18].

2.3. JUSTIFICACIÓN DEL TRABAJO

Este estudio busca abordar estas limitaciones mediante el desarrollo de un modelo de ML optimizado para la predicción de resistencia antibiótica, integrando múltiples fuentes de datos y técnicas avanzadas de interpretación de modelos. Al mejorar la precisión y la interpretabilidad, se espera que este enfoque facilite la adopción de herramientas computacionales en el diagnóstico clínico y contribuya a una mejor gestión de la resistencia antimicrobiana.

3. MATERIALES Y MÉTODOS

Para llevar a cabo este estudio, se utilizaron datos obtenidos de **BVBRC (Bacterial and Viral Bioinformatics Resource Center)**, un sistema de información bioinformática que integra datos de diversas plataformas, incluyendo **PATRIC** (para patógenos bacterianos) y **CARD** (Comprehensive Antibiotic Resistance Database). Los datos específicos empleados provienen principalmente del sistema PATRIC, reconocido como una referencia clave en el análisis de la resistencia bacteriana.

3.1. OBTENCIÓN Y PREPROCESAMIENTO DE DATOS

La preparación de los datos para el modelado fue un proceso fundamental que incluyó la integración de diversas fuentes de información genómica y fenotípica. Se utilizaron principalmente dos archivos descargados de BVBRC:

- **BVBRC_genome_amr.xlsx:** Este archivo contiene información sobre la resistencia a antibióticos de cepas de *Klebsiella pneumoniae*. Las columnas clave para este estudio fueron:
  - ✓ Genome ID (identificador de genoma en formato "X.Y").
  - ✓ Antibiotic (solo se consideraron "imipenem", "meropenem" y "ertapenem").
  - ✓ Resistant Phenotype ("Resistant" / "Susceptible").
  - ✓ Measurement Value (valor del MIC, cuando está disponible).
  - ✓ Measurement Unit (unidad de la determinación de MIC).
- **BVBRC_sp_gene.xlsx:** Este archivo proporciona datos sobre los genes de resistencia identificados en *Klebsiella pneumoniae*. Las columnas relevantes fueron:
  - ✓ Genome ID (debe coincidir con el mismo formato "X.Y" del archivo fenotípico).
  - ✓ BRC ID (identificador de gen, con formato "fig|X.Y.peg.Z").

  Cada fila asocia un genoma con un gen, garantizando cobertura e identidad del 100 % en la asignación de ese BRC ID.
- **BVBRC_genome_feature.fasta**: Este archivo FASTA contiene las secuencias proteicas para cada gen anotado en los genomas. Cada encabezado sigue la sintaxis: >fig|<GenomeID>.peg.<GeneNumber>|<otros campos>. A partir de cada encabezado se extrajo:
  - ✓ BRC ID completo = "fig|X.Y.peg.Z".
  - ✓ Genome ID = "X.Y" (parte numérica anterior a ".peg").

Binarización del Fenotipo de Resistencia

- Se cargó el archivo BVBRC_genome_amr.xlsx y se mantuvieron únicamente las filas cuyo Antibiotic estaba en {"imipenem", "meropenem", "ertapenem"}.

- Para cada genoma se extrajo Genome ID y Resistant Phenotype.

- Se creó la variable binaria Resistance, asignando 1 a “Resistant” y 0 a “Susceptible”.

- Se normalizaron los valores de Genome ID (eliminando espacios extra y convirtiendo a minúsculas) para garantizar coincidencia exacta con los IDs que se obtendrían del FASTA y de los genes anotados.

Generación de Matriz de Presencia/Ausencia de Genes

- Mediante Biopython se parseó el archivo BVBRC_genome_feature.fasta para extraer, de cada encabezado que comienza por “fig|...”, el BRC ID completo (“fig|X.Y.peg.Z”) y el Genome ID (“X.Y”).

- Se almacenaron todos los pares (Genome ID, BRC ID completo) en una lista intermedia y se añadió un atributo “presence = 1” para indicar la presencia de ese gen en ese genoma.

- Con esa lista se construyó una tabla dinámica (pivot-table) en la que:

  - Las filas eran los genomas (Genome ID en minúsculas, sin espacios).
  - Las columnas eran los BRC ID completos.
  - El valor valía 1 si el gen estaba presente, o 0 en otro caso.

- Como resultado se obtuvo una matriz binaria de genes (columnas) vs genomas (filas). Inicialmente contenía 5 341 columnas (cada una un gene BRC ID) y tantas filas como genomas distintos aparecían en el FASTA.

Filtrado y unión con fenotipos

- Se estandarizaron los nombres de genomas en la tabla fenotípica (BVBRC_genome_amr.xlsx) y en la tabla de genes binarios, asegurando que ambos coincidieran en minúsculas sin espacios.

- Se identificó la intersección de Genome ID entre la tabla fenotípica y la matriz binaria de genes, obteniendo 16 529 genomas comunes.

- Se filtró la matriz binaria para conservar únicamente esas filas (los genomas comunes) y se contó la frecuencia de aparición de cada gen (cada columna).

- De todos los genes presentes, se seleccionaron los 100 genes con mayor frecuencia en el conjunto de 16 529 genomas.

- Se construyó una matriz reducida llamada gene_presence_filtered, que contenía 16 529 filas (genomas) y 100 columnas (genes más frecuentes).

- Finalmente, se fusionó (merge) gene_presence_filtered con la tabla fenotípica binarizada (Genome ID + Resistance), generando el DataFrame df_final con 16 529 filas y las siguientes columnas:

  - Genome ID
  - Resistance (0/1)

- 100 columnas binarias (cada una un gen BRC ID), donde 1 = gen presente, 0 = gen ausente.

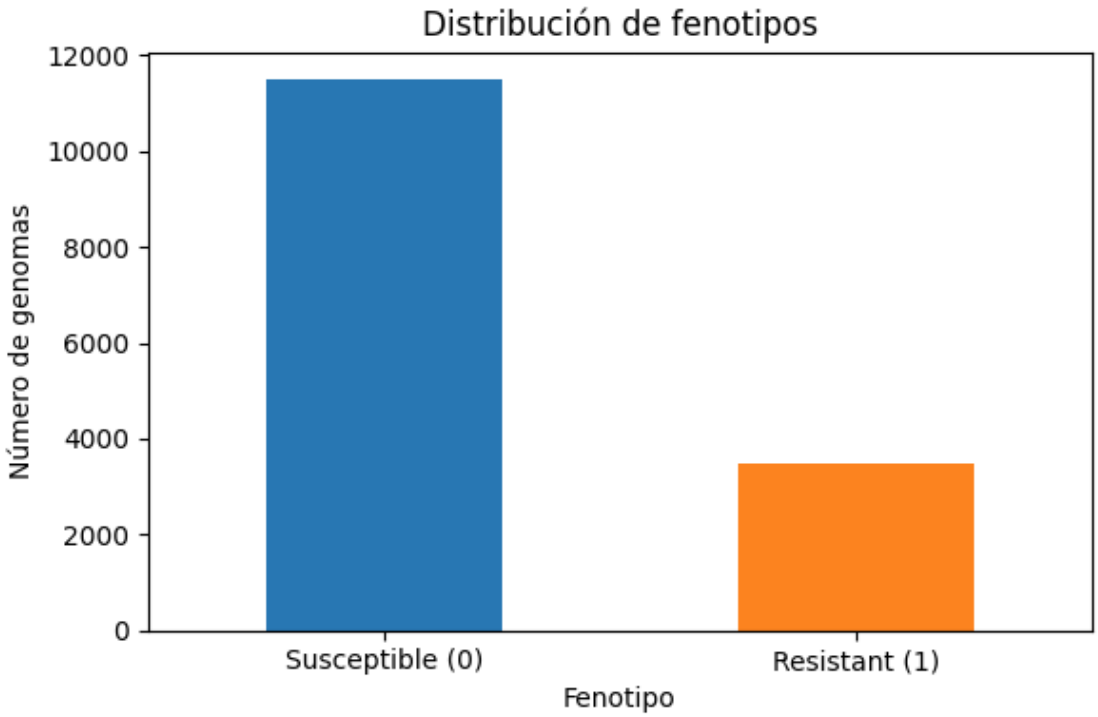
- Cualquier valor ausente tras el merge se llenó con 0 (ausencia de ese gen).

3.2. ESTADÍSTICA DESCRIPTIVA PREVIA AL MODELADO

Se realizó un análisis exploratorio de los datos para comprender su estructura, detectar posibles sesgos y obtener información preliminar acerca de la relación entre los genomas y sus fenotipos de resistencia. A continuación se describen los puntos más relevantes, acompañados de sus figuras.

Distribución general de la resistencia

De los 16.529 genomas de Klebsiella pneumoniae que integran el conjunto final, la gran mayoría es sensible a los carbapenémicos (clase 0). En cifras absolutas, hubo 12.712 genomas sensibles (77,0%) y 3.817 genomas resistentes (23,0%). Para mostrarlo gráficamente, se construyó un diagrama de barras sencillo: la barra relativa a la clase 0 ocupa más de tres cuartas partes del total, mientras que la clase 1 (resistente) se sitúa en el cuarto restante. Se observa una fuerte disparidad, donde la clase resistente está subrepresentada.



Recuento de genes de resistencia por genoma (top 100)

Aun cuando originalmente existían más de 5 300 genes distintos en la matriz binaria, se redujo el conjunto de predictores a los 100 genes más frecuentes, con el objetivo de concentrar los marcadores de mayor relevancia y disminuir la dimensionalidad. Para cada uno de los 16 529 genomas comunes, se sumaron las columnas binarias correspondientes

a estos 100 genes, obteniendo así una variable continua Num_Genes que mide cuántos de esos genes aparecen en cada genoma.

Al calcular las estadísticas descriptivas de Num_Genes, se observó que la media asciende a 1,13 genes por genoma, con una desviación estándar de 0,48. El valor mínimo registrado fue 1 (indica que siempre hay al menos un gen del top 100 presente), y el máximo fue 12. Con estos datos se preparó un histograma en el que, a la izquierda, aparecen más nombres de genomas con uno o dos genes y, a la derecha, una cola decreciente hacia los valores mayores. Además, este histograma se coloreó en dos tonos para diferenciar los genomas resistentes frente a los susceptibles: se aprecia que ambos grupos tienen distribuciones similares, aunque los resistentes tienden a concentrarse ligeramente en valores más altos de recuento de genes. Sin embargo, dado que la mayoría de los genomas—tanto resistentes como sensibles—posee entre 1 y 2 genes del top 100, se dedujo que esta variable por sí sola aportaría poca discriminación en el modelo.

Resistencia	Count	Mean	Std	Min	25%	50%	75%	max
Susceptible	11490	1.114012	0.456350	1.0	1.0	1.0	1.0	12.0
Resistente	3472	1.130472	0.475834	1.0	1.0	1.0	1.0	6.0

3.3. REPOSITORIO GITHUB

El proyecto está organizado en un repositorio de GitHub, lo que facilita la gestión del código, la colaboración y el seguimiento de los cambios realizados en el análisis. La estructura del repositorio es la siguiente:

- data/: Contiene los archivos de datos utilizados en el análisis, incluyendo los archivos BVBRC_genome_amr.xlsx, BVBRC_sp_gene.xlsx y BVBRC_genome_feature.fasta.
- notebooks/: Incluye los Jupyter Notebooks con el código de preprocesamiento, análisis y desarrollo del modelo de aprendizaje automático. Estos notebooks documentan cada paso del análisis, desde la limpieza de datos hasta la construcción del modelo.
- README.md: Archivo con la presentación del proyecto, descripción del objetivo y detalles sobre cómo ejecutar el código del repositorio.

El acceso al repositorio GitHub es:

https://github.com/Juditgf10/TFM-Resistencia-Klebsiella

3.4. MODELADO PREDICTIVO : RANDOM FOREST

Para el desarrollo del modelo predictivo, se empleó el algoritmo **Random Forest**, un ensamblado de árboles de decisión conocido por su robustez y capacidad para manejar conjuntos de datos complejos. Dada la naturaleza desbalanceada de las clases en el conjunto de datos (una mayor proporción de cepas susceptibles frente a resistentes), se implementó una estrategia para mitigar este desequilibrio y mejorar el rendimiento del modelo en la detección de la clase minoritaria (resistencia).

Se utilizó un **pipeline** de imblearn y scikit-learn para integrar el proceso de remuestreo (over-sampling) con el entrenamiento del modelo. Este enfoque es crucial para prevenir la **fuga de datos (data leakage)**, asegurando que la técnica de remuestreo se aplique únicamente a los datos de entrenamiento dentro de cada fold de la validación cruzada.

El pipeline se configuró de la siguiente manera:

1. **División estratificada 70/30**.

    - Se separó el DataFrame final (df_final) en variables predictoras X (las 100 columnas binarias de genes) y etiqueta y (Resistance).
    - Con train_test_split(stratify = y, test_size = 0.30, random_state = 42) se obtuvo:
        - ✓ Conjunto de entrenamiento: 10.473 genomas.
        - ✓ Conjunto de prueba: 4.489 genomas.

2. **Pipeline de sobremuestreo + Random Forest.**
   Se utilizó la clase Pipeline de imblearn junto a scikit-learn para encadenar dos etapas:
    - ADASYN (Adaptive Synthetic Sampling):
        - ✓ Inserta dinámicamente nuevos ejemplos sintéticos de la clase minoritaria (genomas resistentes) basándose en la densidad local de muestras "difíciles".
        - ✓ Atiende especialmente a las regiones del espacio donde los ejemplos reales de la clase minoritaria son escasos o están mal separados.
        - ✓ Se configura con random_state = 42.
    - RandomForestClassifier:
        - ✓ Secundariamente, un clasificador de bosque aleatorio (sin parámetros personalizados en este paso, aparte del seed).

   De este modo, durante cada iteración de entrenamiento en CrossValidation, ADASYN genera solo a partir de los datos de entrenamiento de ese fold, evitando cualquier filtrado de información hacia el test interno.

3. **Búsqueda de hiperparámetros con GridSearchCV.**
   Para optimizar el Random Forest se empleó GridSearchCV con validación cruzada estratificada de 5 pliegues (StratifiedKFold(n_splits = 5, shuffle = True, random_state = 42)). La métrica de scoring elegida fue recall, dado que, en un contexto clínico, es prioritario reducir al mínimo los **falsos negativos** (es decir, no clasificar erróneamente una cepa resistente como sensible).
    - Hiperparámetros evaluados:
        - ✓ n_estimators ∈{100, 200}
        - ✓ max_depth ∈ {None, 10, 20}
        - ✓ min_samples_split ∈ {2, 5}
    - Se entrenó el pipeline completo (ADASYN + RandomForest) sobre los **10.473 genomas de entrenamiento** para determinar la combinación de parámetros que maximiza el recall medio en los pliegues internos.

4. **Evaluación final en el conjunto de prueba**

Una vez obtenidos los mejores hiperparámetros, el modelo definitivo (entrenado en todo el conjunto de entrenamiento con esas configuraciones) se evaluó sobre las **4.489 muestras del conjunto de prueba**, que no participaron ni en el sobremuestreo ni en la búsqueda de parámetros.

- Accuracy general
- Precision y Recall por clase, con especial atención a la clase "resistente"
- F1-score
- AUC-ROC, empleando las probabilidades devueltas por la función predict_proba. Con estas probabilidades se obtuvo la curva ROC y se calculó el área bajo la curva (roc_auc_score). En caso de que el clasificador no devolviera probabilidades directas, se habría utilizado la salida de decision_function y luego roc_auc_score para obtener el AUC.
- Matriz de confusión para el umbral por defecto (0,5) y, adicionalmente, se evaluó el umbral que maximiza el F1-score en el conjunto de prueba (para comprobar si un corte distinto a 0,5 mejora la capacidad de detección de resistencias).

Con este protocolo —pipeline estratificado que aplica ADASYN solo dentro de cada fold, búsqueda con GridSearchCV optimizando recall, y evaluación final en test puro— se garantiza una estimación rigurosa del rendimiento en un escenario realista, minimizando la posibilidad de sobreestimación debido a filtración de datos.

3.5. MODELADO PREDICTIVO: SUPPORT VECTOR MACHINES (SVM)

El segundo algoritmo de clasificación empleado fue la **Support Vector Machine (SVM)**. Las SVMs son modelos potentes para tareas de clasificación, especialmente eficaces en conjuntos de datos de alta dimensión, al encontrar el hiperplano óptimo que mejor separa las clases en el espacio de características.

Para asegurar un entrenamiento robusto y evitar la **fuga de datos (data leakage)** durante el proceso de entrenamiento y validación, se integró SVM dentro de un **pipeline** —similar al de Random Forest— que aplica el escalado de forma aislada en cada partición y luego entrena el clasificador.

A continuación se detalla la configuración empleada:

1. **Escalado de Características con StandardScaler**.
   Las SVM son especialmente sensibles a la escala de los atributos, por lo que el primer paso del pipeline consistió en aplicar un StandardScaler. Este transformador calcula la media y desviación estándar **solo** con las muestras del conjunto de entrenamiento de cada fold, y usa esos mismos parámetros para transformar las muestras de validación de ese fold. De este modo, evitar fugas de información y garantizar que los datos de prueba permanezcan completamente "inéditos" cuando se hace la medición de métricas.

2. **Clasificador SVM con class_weight='balanced'**.
   En el segundo paso del pipeline se definió un SVC(probability=True, class_weight='balanced', random_state=42). Al ajustar class_weight='balanced', el

algoritmo asigna de forma automática un peso inversamente proporcional a la frecuencia de cada clase (resistente vs susceptible) en el subconjunto de entrenamiento. Esto refuerza la penalización de los errores sobre la clase minoritaria (resistente) sin recurrir a técnicas de sobremuestreo, pues el objetivo es obtener un modelo que no subestime la clase de interés.

3. **Búsqueda de hiperparámetros con GridSearchCV.**
   Para optimizar el rendimiento de SVM se empleó GridSearchCV acoplado a un StratifiedKFold(n_splits=5, shuffle=True, random_state=42). La estratificación asegura que, en cada fold, la proporción de genomas resistentes y sensibles refleje la distribución original, de modo que las métricas de validación sean representativas.
   - Se escogió como métrica principal de scoring el **F1-score** de la clase "resistente", pues refleja el equilibrio entre precisión (minimizar falsos positivos en resistentes) y recall (minimizar falsos negativos en resistentes). Si bien en el contexto clínico el recall es crítico para no pasar por alto cepas resistentes, el F1-score se utiliza aquí porque ofrece una valoración más equilibrada cuando la clase es minoritaria y evita que un recall excesivo con baja precisión arruine la utilidad práctica del modelo.
   - **Hiperparámetros evaluados:**
     - ✓ C ∈ {0.1, 1, 10}
     - ✓ gamma ∈ {'scale', 'auto', 0.1, 1}
     - ✓ kernel = {'rbf'}

   Durante cada iteración de GridSearchCV, el pipeline (escalado + SVM) se entrenó y validó en los 5 pliegues internos, optimizando el F1 en la clase de interés. Una vez seleccionados los mejores valores de (C, gamma, kernel), el modelo resultante se ajustó sobre **todo el conjunto de entrenamiento** utilizando esos hiperparámetros.

4. **Evaluación final en el conjunto de prueba**
   El modelo final se probó únicamente sobre las 3 306 muestras del conjunto de prueba, que no participaron ni en el escalado ni en la búsqueda de parámetros. Se calcularon las métricas de:
   - Accuracy general
   - Precision y Recall para cada clase, con especial atención a la clase "resistente"
   - F1-score
   - AUC-ROC, empleando las probabilidades de la función predict_proba
   - Matriz de confusión para el umbral por defecto (0,5) y, adicionalmente, se evaluó el umbral que maximiza el F1-score en el conjunto de prueba (para comprobar si un corte distinto a 0,5 mejora la capacidad de detección de resistencias).

   Con este procedimiento, SVM se entrena y valida de forma rigurosa, asegurando que el escalado y la ponderación de clases no "contaminen" el conjunto de prueba. Así, se obtiene una estimación imparcial del desempeño de SVM a la hora de discriminar entre genomas resistentes y sensibles.

3.6. MODELADO PREDICTIVO: XGBOOST

Para la tercera aproximación se empleó **XGBoost (Extreme Gradient Boosting)**, un algoritmo de ensamblaje basado en gradiente que combina múltiples árboles débiles en un predictor fuerte. Al igual que en modelos anteriores, se evitó la fuga de datos (***data leakage***) encapsulando XGBoost en un **pipeline** junto a las transformaciones necesarias, de modo que cada paso se ejecutara únicamente sobre los datos de entrenamiento de cada fold.

1. **Escalado de Características con StandardScaler**.
   El primer componente del pipeline fue StandardScaler, idéntico al utilizado en SVM. Aunque XGBoost es menos sensible a la escala que las SVM, el escalado asegura una convergencia más estable de los métodos basados en gradientes y garantiza coherencia en el preprocesamiento. Como parte del pipeline, el escalador calcula media y desviación en cada fold de entrenamiento y luego transforma las muestras de validación de ese mismo fold.

2. **Clasificador XGBoost sin scale_pos_weight.**
   Se empleó XGBClassifier(objective='binary:logistic', random_state=42) directamente sobre las características escaladas. En esta implementación no se utilizó explícitamente scale_pos_weight; en su lugar, se confiaron los hiperparámetros (especialmente los relacionados con capacidad de generalización y fracciones de muestreo) para lidiar con el desbalance.

3. **Búsqueda de hiperparámetros con GridSearchCV (optimización por recall).**
   Para encontrar la configuración óptima de XGBoost se utilizó GridSearchCV con validación cruzada estratificada de 5 pliegues (StratifiedKFold(n_splits=5, shuffle=True, random_state=42)). La métrica de scoring elegida fue **recall**, ya que se prioriza detectar correctamente las cepas resistentes (clase minoritaria), minimizando falsos negativos.
   - **Hiperparámetros evaluados:**
     - ✓ n_estimators ∈ {100, 200}
     - ✓ learning_rate ∈ {0.01, 0.1, 0.3}
     - ✓ max_depth ∈ {3, 5, 7}
     - ✓ subsample∈ {0.8, 1}
     - ✓ colsample_bytree{0.8, 1}

   Durante cada iteración interna de GridSearchCV, XGBoost se entrenó y validó sobre los datos de entrenamiento de ese pliegue, optimizando el recall de la clase "resistente". Una vez completado el proceso, se obtuvo la combinación de hiperparámetros que produjo el mejor recall promedio.

4. **Evaluación final en el conjunto de prueba.**
   Con los hiperparámetros óptimos hallados (grid_xgb.best_params_), se reentrenó XGBoost sobre todo el conjunto de entrenamiento escalado (n ≈ 10 473). A continuación, se evaluó en las 4 489 muestras de prueba independientes. Para cada genoma de prueba, se calculó la probabilidad de pertenecer a la clase "resistente" (predict_proba). Se reportaron las siguientes métricas:
   - **Accuracy** global
   - **Precision**, **Recall** y **F1-score** para cada clase

- **AUC-ROC**, empleando las probabilidades devueltas por predict_proba
- Se buscó el umbral óptimo que maximiza el F1-score en el conjunto de prueba mediante la curva de precisión-recall (precision_recall_curve).
- Se calculó una matriz de confusión tanto al umbral por defecto (0,5) como al umbral que maximiza F1 sobre y_prob, para comparar la capacidad de XGBoost en la separación de las dos clases.

Gracias a este protocolo—con escalado aislado en cada fold, búsqueda de hiperparámetros centrada en recall y selección de umbral por F1 en el conjunto de prueba—se obtuvo una evaluación objetiva del desempeño de XGBoost para discriminar entre genomas resistentes y sensibles a partir de la presencia/ausencia de genes.

4. RESULTADOS

4.1. RENDIMIENTO DEL MODELO RANDOM FOREST

El proceso de optimización del modelo Random Forest, integrado en un pipeline con ADASYN y ajustado mediante validación cruzada estratificada con el recall como métrica principal, arrojó los siguientes hiperparámetros óptimos:

- rf__max_depth: None
- rf__min_samples_split: 5
- rf__n_estimators: 200

La evaluación del modelo Random Forest entrenado con estos parámetros óptimos sobre el conjunto de prueba (20% del total de los datos) proporcionó el siguiente informe de clasificación:

CLASE	PRECISIÓN	RECALL	F1-SCORE	SOPORTE
0 (SUSCEPTIBLE)	0.77	1.00	0.87	3447
1 (RESISTENTE)	0.62	0.02	0.04	1042
ACCURACY			0.77	4489
MACRO AVG	0.71	0.51	0.46	4489
WEIGHTED AVG	0.74	0.77	0.68	4489

AUC-ROC (umbral 0.5): 0.5060

Posteriormente, se buscó el umbral óptimo sobre las probabilidades de predicción para maximizar el F1-score. El umbral resultante fue 0.3817, con un F1 = 0.3768. Con ese umbral, el informe de clasificación en el mismo conjunto de prueba fue:

CLASE	PRECISIÓN	RECALL	F1-SCORE	SOPORTE
0 (SUSCEPTIBLE)	0.00	0.00	0.00	3447
1 (RESISTENTE)	0.23	1.00	0.38	1042
ACCURACY			0.23	4489
MACRO AVG	0.12	0.50	0.19	4489
WEIGHTED AVG	0.05	0.23	0.09	4489

El modelo Random Forest logró un accuracy global del 77 % (umbral 0.5). Para la clase mayoritaria (susceptible), el recall alcanzó 1.00, lo que indica que prácticamente todas las cepas susceptibles fueron correctamente identificadas. Sin embargo, para la clase minoritaria (resistente) el recall cayó a 0.02, lo que significa que solo el 2 % de las cepas resistentes fueron detectadas (aproximadamente 98 % resultaron en falsos negativos).

La precisión de 0.65 en la clase resistente indica que, de las pocas predicciones positivas que hizo el modelo, el 65 % eran verdaderas resistencias. Sin embargo, ese porcentaje es engañoso, pues el bajo recall revela que el modelo casi no predice resistencia. El F1-score de 0.04 refleja ese desequilibrio: se prioriza identificar susceptibles a expensas de pasar por alto la mayoría de las resistencias.

Al emplear el umbral óptimo (0.3817), el modelo invierte casi todas las predicciones hacia la clase resistente (recall 1.00 para la clase 1), pero a costa de etiquetar erróneamente como resistentes a todos los genomas susceptibles (precisión 0.23, accuracy 0.23). Esto confirma que, aun ajustando el umbral, el desequilibrio de clases y la escasez de patrones suficientemente discriminativos para la clase resistente limitan gravemente la capacidad predictiva práctica del modelo. Desde el punto de vista clínico, este alto porcentaje de falsos negativos (umbral 0.5) o de falsos positivos (umbral optimizado) compromete la fiabilidad de la predicción de resistencia, ya que un falso negativo implica clasificar erróneamente una cepa resistente como susceptible, lo que conlleva un riesgo elevado de fallar en el tratamiento.

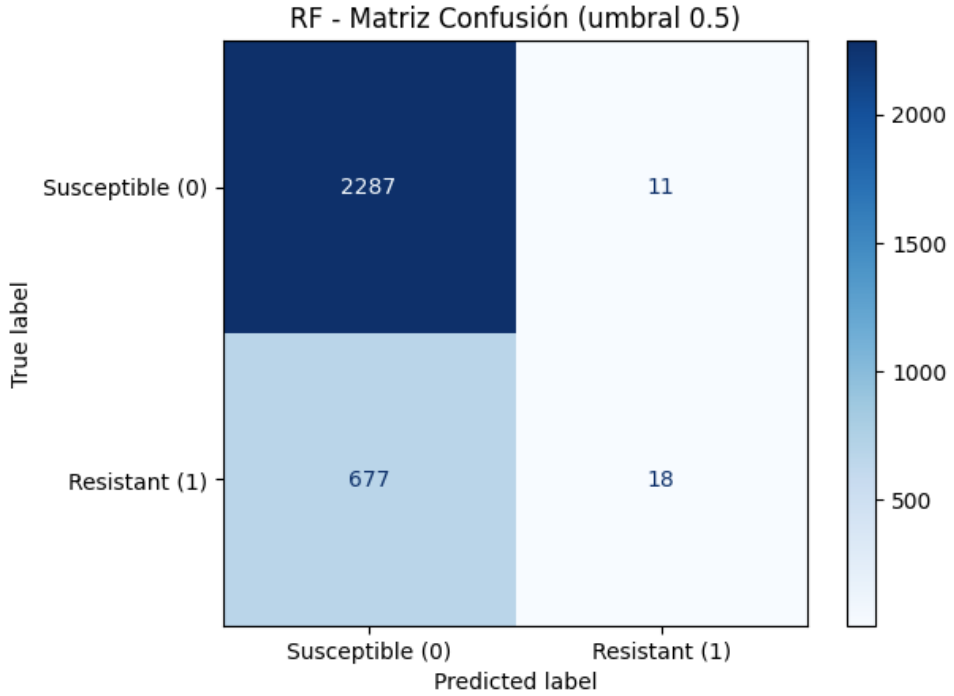


Figura 2. Matriz de confusión (umbral 0.5) del modelo Random Forest

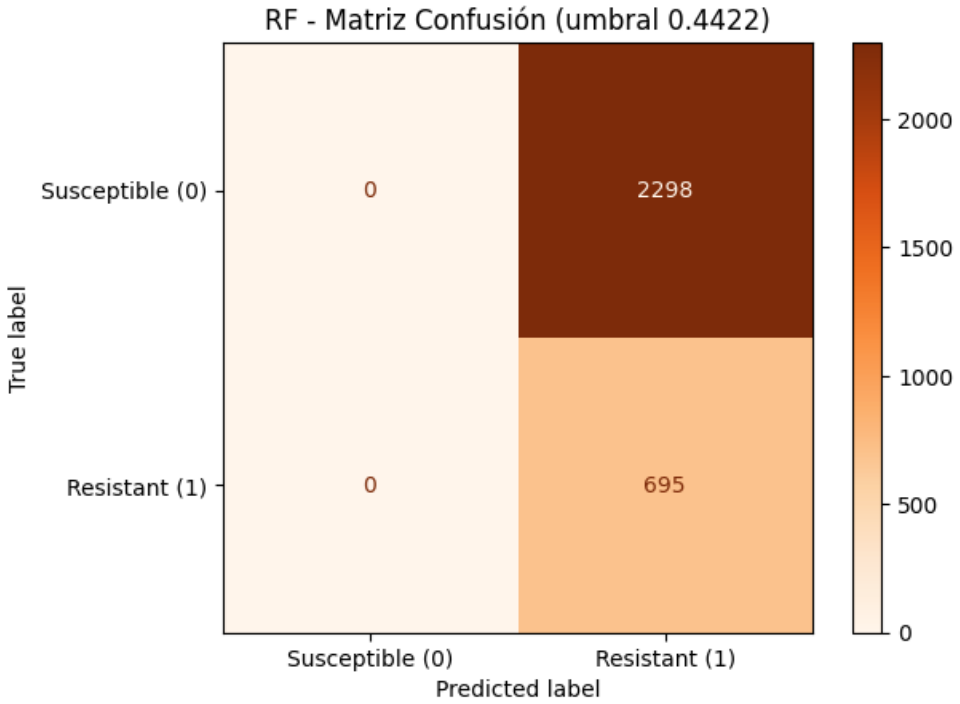


Figura 3. Matriz de confusión (umbral 0.4422) del modelo Random Forest

4.2. RENDIMIENTO DEL MODELO SUPPORT VECTOR MACHINE (SVM)

El segundo modelo evaluado, la Support Vector Machine (SVM), se entrenó con un pipeline que incluía el escalado de características y un ajuste de pesos de clase para manejar el desbalance. La búsqueda de hiperparámetros, optimizada por recall mediante GridSearchCV con validación cruzada estratificada, arrojó los siguientes parámetros óptimos:

- svc__C: 0.1
- svc__gamma: scale
- svc__kernel: rbf

La evaluación del modelo SVM con estos parámetros óptimos sobre el conjunto de prueba (que comprende el 30% de los datos totales) produjo el siguiente informe de clasificación (umbral 0):

CLASE	PRECISIÓN	RECALL	F1-SCORE	SOPORTE
0 (SUSCEPTIBLE)	0.77	1.00	0.87	3447
1 (RESISTENTE)	0.70	0.03	0.05	1042
ACCURACY			0.77	4489
MACRO AVG	0.74	0.51	0.46	4489
WEIGHTED AVG	0.76	0.77	0.68	4489

AUC-ROC (umbral 0): 0.5117

Al ajustar el umbral sobre la salida de decision_function para maximizar el F1-score en el conjunto de prueba, se obtuvo un umbral óptimo de **–0.9999** (equivalente a casi etiquetar todo como "resistente"), con un F1 = 0.3768. En ese caso, el informe de clasificación fue:

CLASE	PRECISIÓN	RECALL	F1-SCORE	SOPORTE
0 (SUSCEPTIBLE)	0.00	0.00	0.00	3447
1 (RESISTENTE)	0.23	1.00	0.38	1042
ACCURACY			0.23	4489
MACRO AVG	0.12	0.50	0.19	4489
WEIGHTED AVG	0.05	0.23	0.09	4489

Análisis de los resultados

El SVM alcanzó un accuracy global del 77 % con el umbral por defecto (0). Para la clase mayoritaria (susceptible), el recall fue 1.00, lo que indica que el modelo identificó correctamente prácticamente todas las cepas susceptibles (n = 3 447), sin producir falsos negativos en esa clase. Sin embargo, para la clase minoritaria (resistente), el recall cayó a 0.03, lo que equivale a detectar solo el 3 % de las cepas resistentes (n = 1 042), dejando sin identificar al 97 % de ellas (falsos negativos).

La precisión de 0.70 en la clase resistente significa que, de las pocas veces que el modelo predice "resistente", el 70 % son verdaderas resistencias. No obstante, ese alto porcentaje solo se da sobre un número muy reducido de predicciones positivas, como lo refleja el F1-score de 0.05, que sintetiza ambas métricas. Esto evidencia que el SVM, a pesar de emplear pesos de clase, no logra generalizar patrones discriminativos suficientes para la clase minoritaria en este conjunto de datos.

Al cambiar el umbral a –0.9999 para maximizar el F1, el SVM pasa a predecir "resistente" en casi todos los casos. En este escenario, consigue un recall de 1.00 para la clase resistente (captura todas las cepas resistentes), pero sacrifica por completo la detección de la clase susceptible (recall 0.00 en la clase 0), lo que arroja un 0 % de accuracy, pues

etiqueta erróneamente como resistentes a los 3 447 genomas susceptibles. El F1-score de 0.38 en la clase 1 es consecuencia de esa decisión extrema.

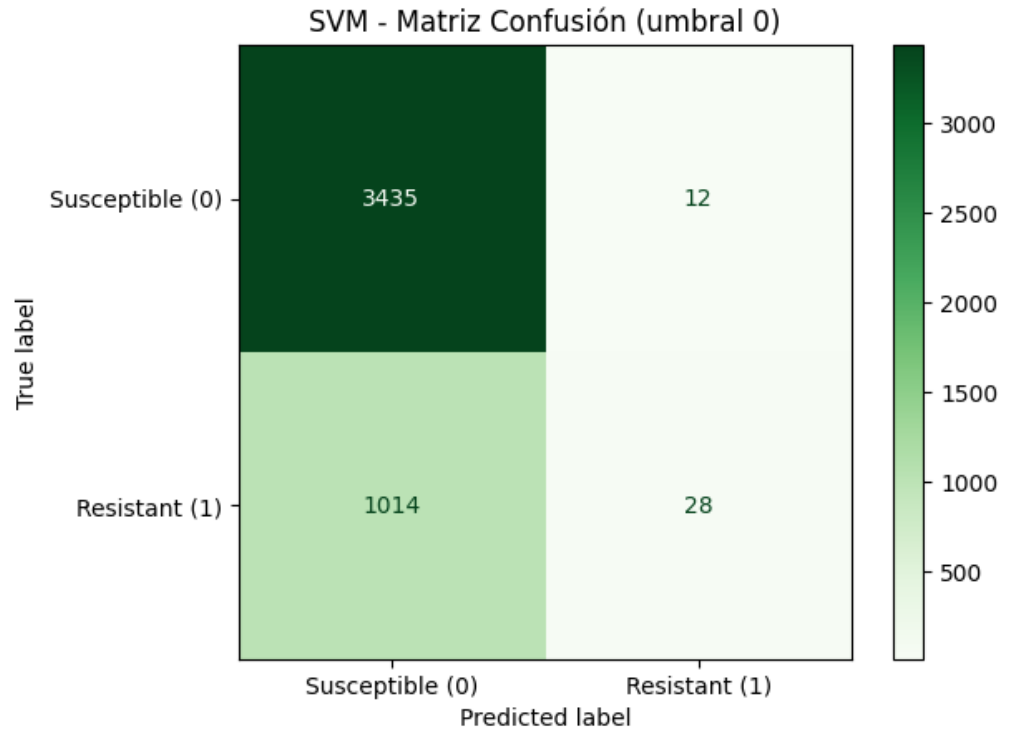


Figura 4. Matriz de confusión (umbral 0) para el modelo SVM

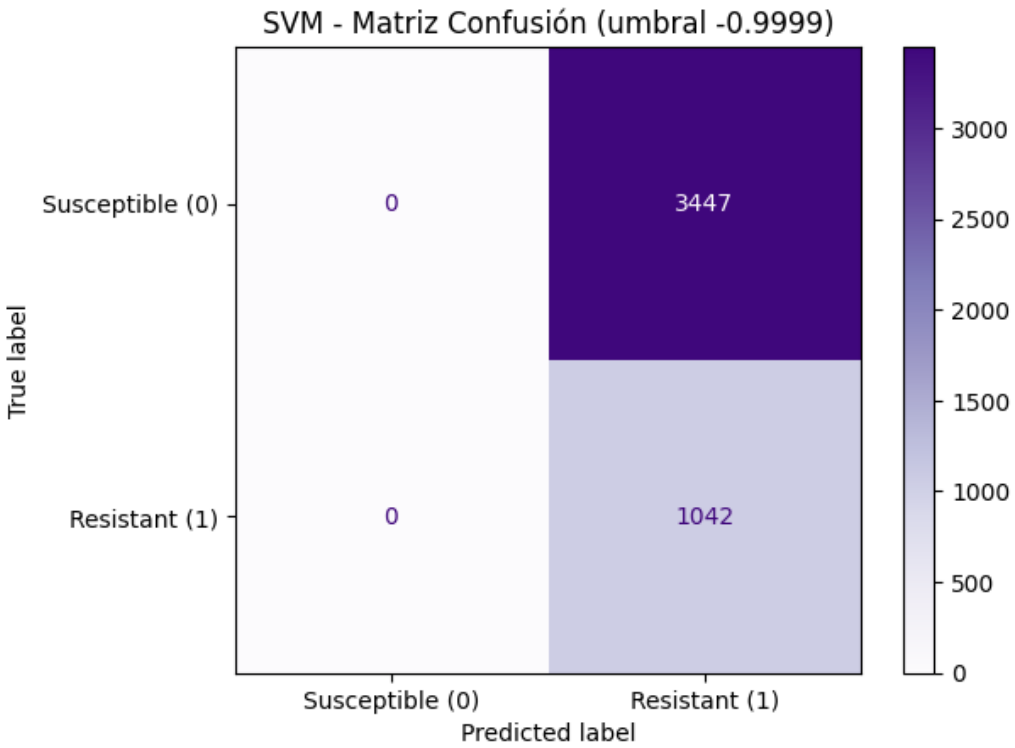


Figura 5. Matriz de confusión (umbral -0.9999) para el modelo SVM

4.3. RENDIMIENTO DEL MODELO XGBOOST

El tercer modelo evaluado, XGBoost, se entrenó con un pipeline que incluía el escalado de características y la búsqueda de hiperparámetros optimizada por recall. En esta iteración no se aplicó explícitamente scale_pos_weight; en su lugar, la búsqueda de hiperparámetros se encargó de ajustar la capacidad del modelo para lidiar con el desbalance. Los hiperparámetros óptimos encontrados para el XGBoost fueron:

- learning_rate: 0.10
- max_depth: 3
- n_estimators: 200
- subsample: 0.80
- colsample_bytree: 0.80

Estos valores se obtuvieron mediante GridSearchCV con validación cruzada estratificada de 5 pliegues, donde el scoring principal fue el recall de la clase resistente.

La evaluación final de XGBoost sobre las 4 489 muestras del conjunto de prueba (30 % de los datos totales) produjo el siguiente informe de clasificación (umbral por defecto = 0,5):

CLASE	PRECISIÓN	RECALL	F1-SCORE	SOPORTE
0 (SUSCEPTIBLE)	0.77	1.00	0.87	3447
1 (RESISTENTE)	0.67	0.00	0.01	1042
ACCURACY			0.77	4489
MACRO AVG	0.72	0.50	0.44	4489
WEIGHTED AVG	0.74	0.77	0.67	4489

AUC-ROC

Utilizando las probabilidades de la clase “resistente” (predict_proba), se obtuvo un **AUC-ROC = 0.5021**. Este valor apenas supera la línea de corte de 0.50, lo que indica que, en conjunto, el modelo no logra discriminar mejor que un clasificador aleatorio en términos de ordenamiento por probabilidad.

Umbral óptimo (F1-score)

A partir de la curva de precisión-recall, se calculó el umbral que maximiza el F1-score en el conjunto de prueba, resultando en **threshold = 0.2275** con **F1 = 0.3768**. Al aplicar este umbral, las métricas cambiaron drásticamente:

1. **Umbral 0,5**
    - Precisión (resistente): 0.67
    - Recall (resistente): 0.00
    - F1 (resistente): 0.01
2. **Umbral 0.2275**
    - Precisión (resistente): 0.23
    - Recall (resistente): 1.00

- F1 (resistente): 0.38

Bajo el umbral de 0,2275, el modelo clasifica todas las cepas resistentes de prueba correctamente (recall = 1.00), a costa de etiquetar como resistentes a todas las cepas susceptibles también (precisión = 0.00 para la clase 0).

Análisis de los resultados

XGBoost alcanzó un **accuracy global del 77 %**, muy parecido al obtenido con Random Forest y SVM. Para la clase mayoritaria (susceptible) el recall fue 1.00, es decir, prácticamente todas las cepas susceptibles se identificaron correctamente, generando un número mínimo de falsos positivos.

Sin embargo, la **clase minoritaria (resistente)** volvió a mostrar un recall prácticamente nulo (cero) con el umbral por defecto, lo que significa que casi ninguna cepa resistente se detecta correctamente en esa configuración. Al forzar el umbral óptimo (0.2275), el recall de la clase resistente aumenta a 1.00, pero esto se logra únicamente anotando como resistentes todas las cepas (incluyendo las susceptibles), lo que resulta en una precisión de 0.00 para la clase susceptible y hace inservible al modelo en la práctica clínica con ese corte.

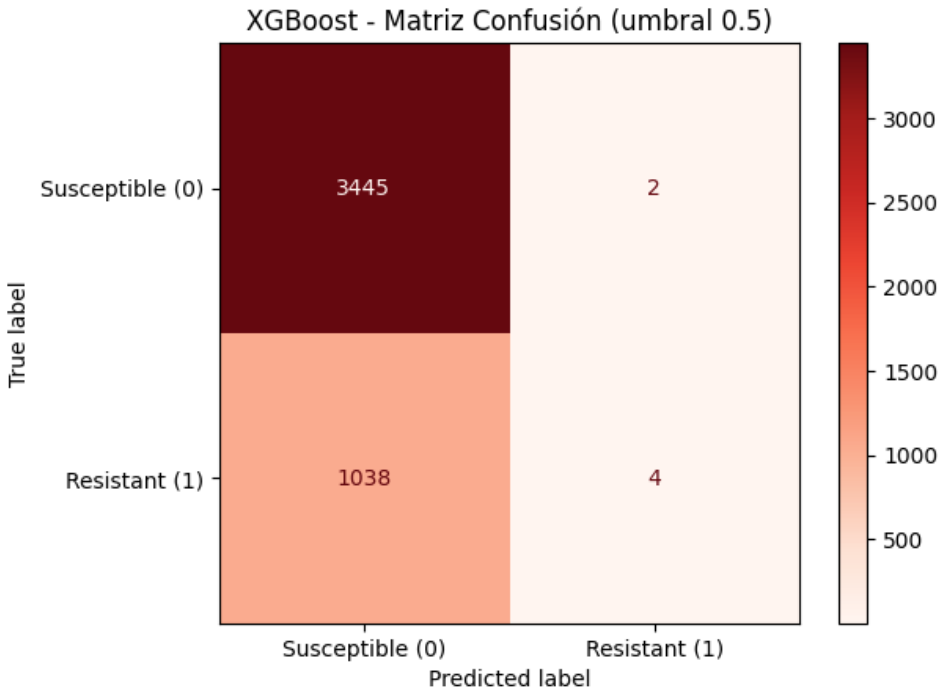


Figura 6. Matriz de confusión (umbral 0.5) para el modelo XGBoost

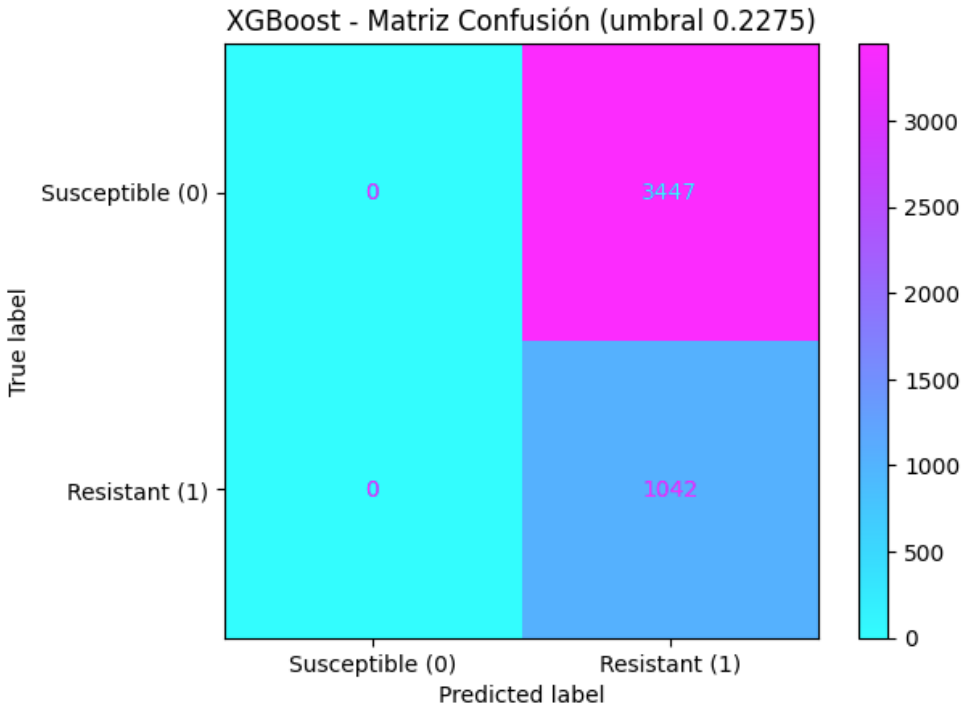


Figura 7. Matriz de confusión (umbral 0.2275) para el modelo XGBoost

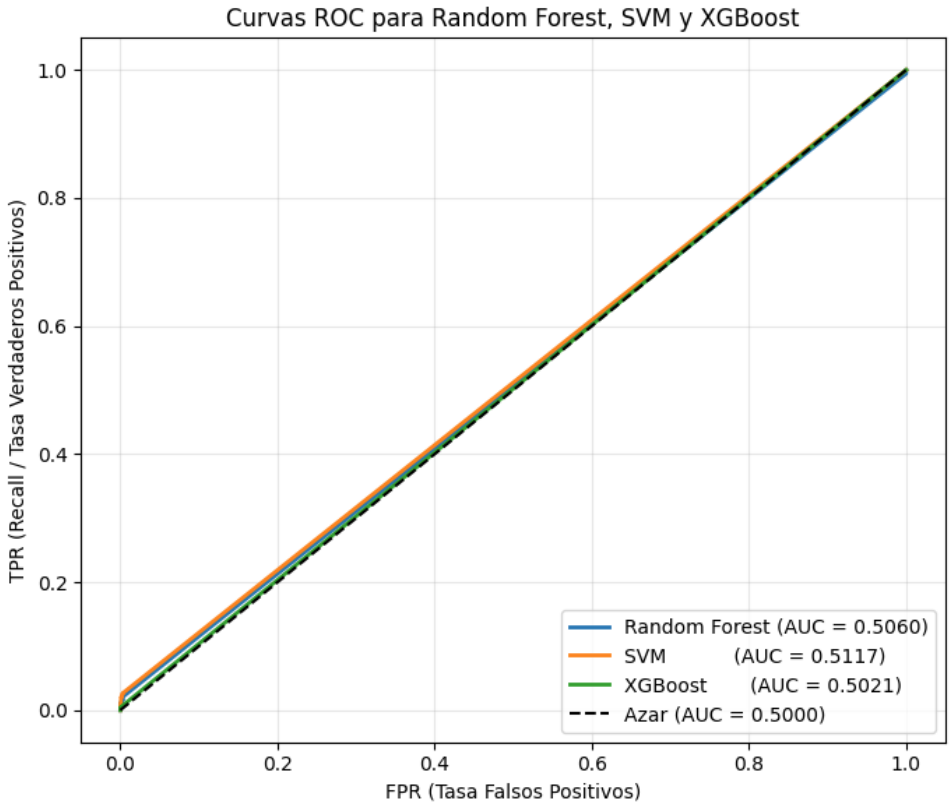


Figura 8. Curvas ROC para Random Forest, SVM y XGBoost

4.4. ANÁLISIS COMPLEMENTARIO: ANÁLISIS DE CORRELACIÓN ENTRE RECUENTO DE GENES Y VALORES DE MIC

El objetivo de este análisis es explorar si existe alguna relación estadística entre el número de genes de resistencia presentes en un genoma (entre los 100 seleccionados) y el valor de temperatura inhibitoria mínima (MIC) para un antibiótico dado (por ejemplo, imipenem). Si hubiera correlación, implicaría que un mayor número de genes se asocia (positiva o negativamente) con un valor de MIC más alto (mayor resistencia fenotípica).

Pasos resumidos

1. Se normalizan los identificadores de genoma en el archivo fenotípico (MIC) para poder hacer el merge con el conteo de genes.
2. Se convierten a numérico los valores de MIC (Measurement Value → MIC_Value).
3. En el DataFrame completo (df_full) se calcula, para cada genoma, cuántos genes de los 100 aparecen presentes.
4. Se filtran las filas correspondientes solamente al antibiótico elegido (p.ej. “imipenem”).
5. Se fusionan los valores de Num_Genes y MIC_Value usando el campo Genome ID.
6. Se dibuja un diagrama de dispersión (“scatter”) de Num_Genes vs. MIC_Value.
7. Se calculan coeficientes de correlación de Pearson y de Spearman, junto con su p-value.

Resultado:

Pearson r=-0.193, p-value=4.232e-08

Spearman rho=-0.302, p-value=2.907e-18

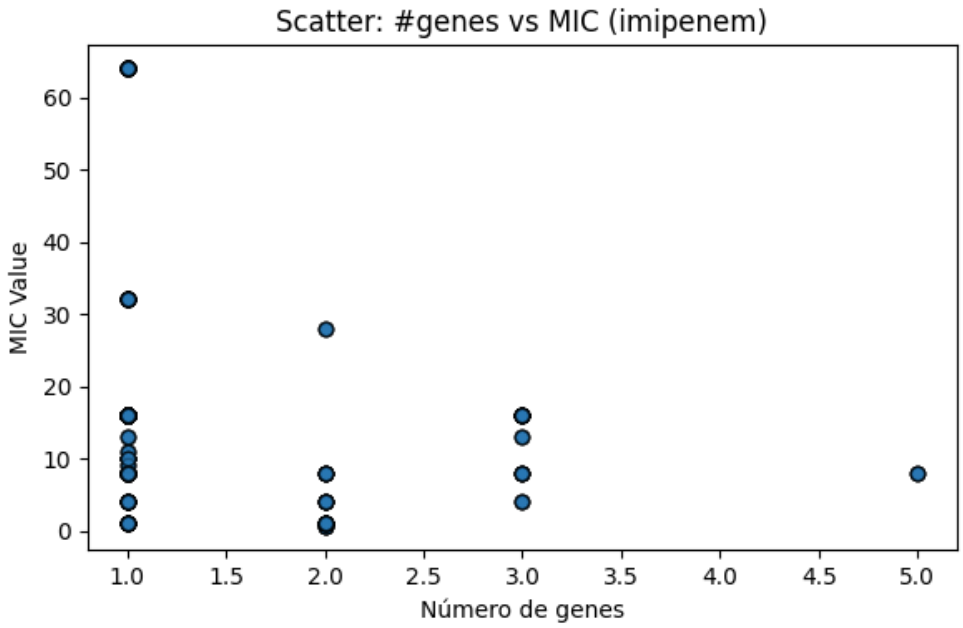


Figura 9. Scatter plot: Numero de genes vs valores MIC

El coeficiente de Pearson cercano a −0.19 sugiere que, en este conjunto de datos, hay una tendencia muy débil a que los genomas con más genes de resistencia presentes tengan MIC ligeramente inferiores para imipenem.

El coeficiente de Spearman más pronunciado (−0.30) reafirma que a mayor conteo de genes de resistencia tiende a haber MIC menores.

Significado biológico: Aunque podría esperarse que la presencia de más genes de resistencia elevara el MIC (o al menos no lo redujera), aquí se encuentra el efecto inverso, muy minoritario. Esto sugiere que no todos los genes de la lista contribuyen de forma sinérgica a incrementar la resistencia fenotípica a imipenem, y de hecho, podría haber genes "marcadores" asociados a cepas más susceptibles.

4.5. ANÁLISIS COMPLEMENTARIO: ANÁLISIS DE COMPONENTES PRINCIPALES (PCA) Y T-SNE SOBRE LA MATRIZ BINARIA DE GENES

El objetivo de este análisis es reducir la alta dimensionalidad (100 genes) a dos dimensiones y ver si, en ese "mapa" 2D, las cepas resistentes y susceptibles se separan o forman agrupamientos diferentes.

- Con **PCA** buscamos las dos direcciones ("componentes principales") que expliquen la mayor varianza en la tabla binaria de presencia/ausencia.
- Con **t-SNE** (es un método no lineal) proyectamos a 2D las mismas filas para intentar visualizar posibles "clustering" no lineales.

Resultado PCA (2D) sobre matriz binaria de genes:

Falta de separación clara: tanto genomas resistentes como susceptibles ocupan regiones muy similares en el espacio PC1–PC2, lo cual sugiere que, con estas 100 variables binarias, las dos primeras componentes principales no logran discriminar bien los dos fenotipos.

Puntos atípicos: esos genomas situados muy a la derecha (PC1 ~10–11) tienen más genes “raros” en la selección de 100; aparecen mayoritariamente como susceptibles (azul), lo que sugiere que, paradójicamente, la acumulación excepcional de genes de resistencia anotados no se traduce necesariamente en un fenotipo “Resistant” en el ensayo.

En conjunto, el PCA confirma que la variabilidad más grande de la matriz binaria (expresada en PC1) está dominada por la presencia/ausencia de unos pocos genes muy frecuentes, pero este patrón no coincide de forma consistente con la etiqueta de resistencia.

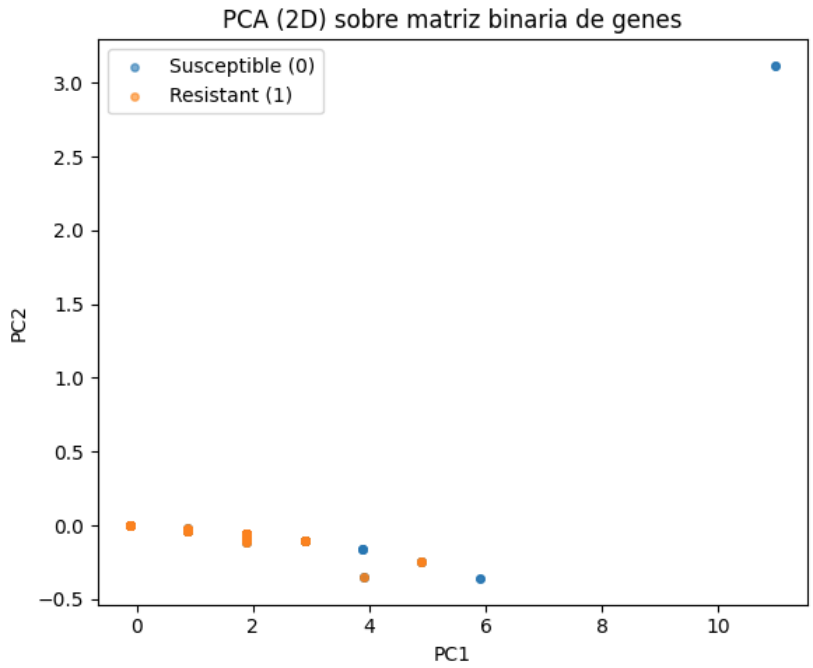


Figura 10. Gráfico PCA a 2 componentes

Resultado t-SNE (2D) sobre matriz binaria de genes:

Agrupamientos locales pero sin distinción fenotípica: t-SNE revela estructuras de agrupamiento interno en los genomas, probablemente reflejando conjuntos de genes que coocurren con frecuencia. Sin embargo, cada uno de esos grupos “locales” contiene tanto genomas resistentes como susceptibles, lo que indica que la etiqueta de resistencia no “dicta” un grupo único en el espacio binario.

Heterogeneidad de muestras: la existencia de varios “cúmulos” sugiere que hay subcohortes de genomas con patrones de genes similares, pero la resistencia no se correlaciona de forma estable con ninguno de ellos.

Conclusión para modelado: estos resultados refuerzan la idea de que con las 100 características binarias actuales no existe un patrón de separación clara entre genomas resistentes y susceptibles. Desde la perspectiva de un clasificador, esto explica por qué los modelos tuvieron dificultades para distinguir las dos clases: la estructura subyacente de los datos no muestra clusters “puros” de cada fenotipo.

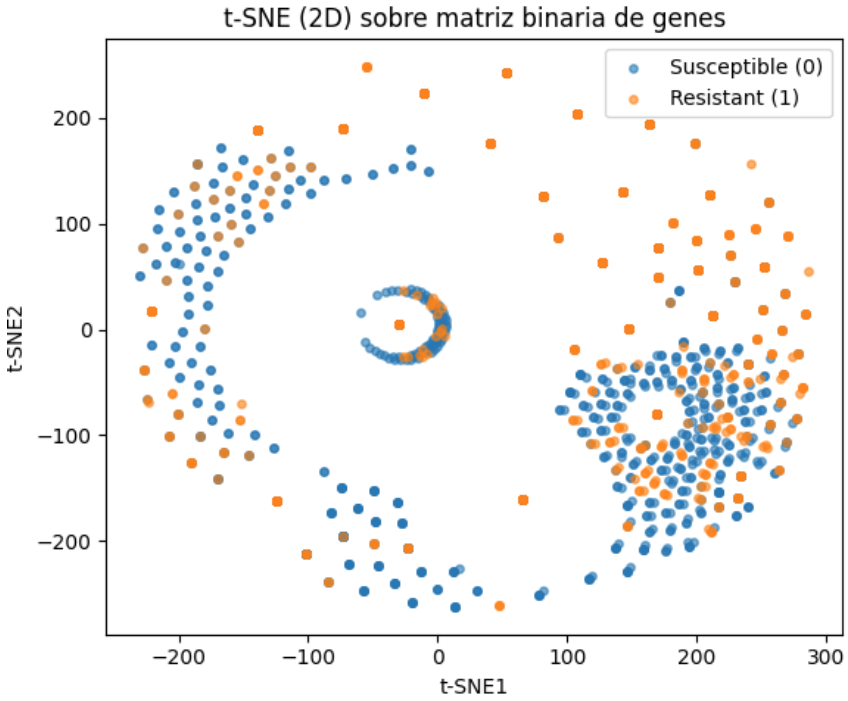


Figura 11. Gráfico t-SNE a 2D

4.6. ANÁLISIS COMPLEMENTARIO: ANÁLISIS DE SUBTIPOS DE ANTIBIÓTICOS (EVOLUCIÓN FENOTÍPICA)

El objetivo de este análisis es visualizar la proporción de genomas resistentes para cada antibiótico (imipenem, meropenem, ertapenem, etc.), y comprobar cuántos genomas son simultáneamente resistentes a dos antibióticos (por ejemplo: imipenem y meropenem). También graficamos un scatter de MIC Imipenem vs MIC Meropenem para aquellos genomas que tienen ambos valores de MIC.

Resultado sobre el porcentaje de genomas resistentes por antibiótico:

Doripenem (~88 % resistentes):

- Este antibiótico destaca con un porcentaje de resistencia extraordinariamente alto (casi el 90 %). Probablemente esto se deba a que, de los genomas analizados, solo unos pocos se evaluaron con doripenem (muestra muy reducida), y la mayoría de ellos resultaron “Resistant”.

Ertapenem (~29 % resistentes):

- Aproximadamente un 29 % de los genomas probados con ertapenem mostraron resistencia. Es la proporción más baja entre los carbapenémicos “clásicos” (imipenem, meropenem, ertapenem).

Imipenem (~32 % resistentes):

- Aquí, cerca del 32 % de los genomas probados resultaron resistentes. Esa cifra es un poco mayor que en ertapenem, lo que refleja que imipenem suele perder eficacia ligeramente antes que ertapenem en muchas cepas.

Meropenem (~31 % resistentes):

- El porcentaje para meropenem roza el 31 %, muy similar al de imipenem. Esto coincide con la idea de que meropenem e imipenem muestran perfiles de resistencia parecidos en Klebsiella pneumoniae.

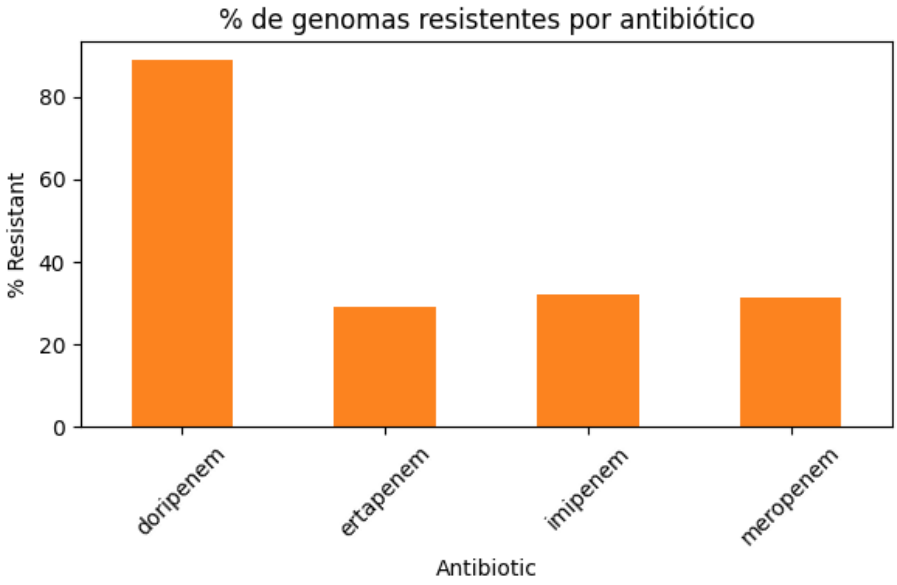


Figura 12. Gráfico de porcentaje de genomas resistentes por antibiótico

Resultado sobre MIC Imipenem vs Meropenem:

Concentración de puntos cerca del origen (MIC bajo):

- Existe un grupo numeroso de genomas con valores bajos de MIC para ambos antibióticos (< 8 µg/mL), lo que coincide con cepas susceptibles o de resistencia intermedia. Esto produce una "nube densa" alrededor de valores pequeños (tanto en X como en Y).

Algunos outliers muy elevados:

- Se aprecian unos pocos puntos aislados con MIC de imipenem > 100 µg/mL, y correspondientemente MIC de meropenem muy alto (> 60–120 µg/mL). Esos genomas representan cepas de resistencia extrema, que son pocas pero muestran que la resistencia combinada a imipenem y meropenem puede alcanzar concentraciones altísimas.

Ligera correlación positiva:

- Aunque la nube de puntos es bastante dispersa, a nivel visual se aprecia que a medida que el MIC de imipenem aumenta, tiende a subir también el MIC de meropenem. Esto sugiere que muchos mecanismos de resistencia afectan en paralelo a ambos carbapenémicos.

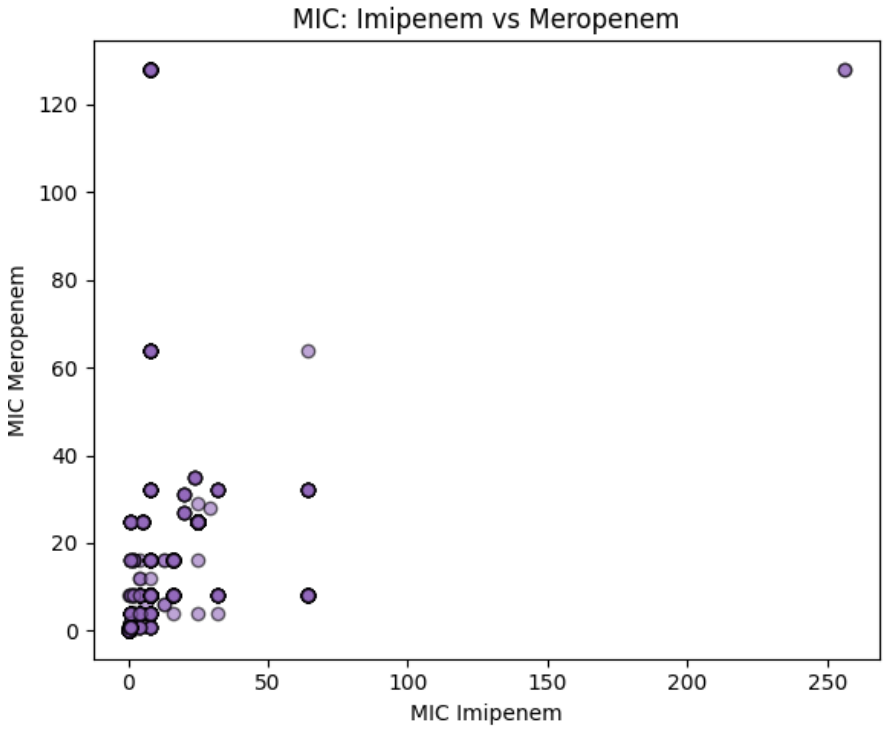


Figura 13. Gráfico MIC Imipenem vs Meropenem

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1. CONCLUSIONES

El presente trabajo tuvo como objetivo principal desarrollar un modelo de *machine learning* capaz de predecir la resistencia o susceptibilidad a carbapenémicos en cepas de *Klebsiella pneumoniae* utilizando datos genómicos públicos. Se aspiraba a alcanzar una precisión mínima del 80% en la clasificación y una alta sensibilidad en la detección de resistencias, dada su relevancia clínica. Para ello, se exploraron tres algoritmos de clasificación supervisada: Random Forest, Support Vector Machine (SVM) y XGBoost, cada uno implementado con metodologías rigurosas para abordar el desbalance de clases y evitar la fuga de datos.

La metodología empleada fue robusta e incluyó el uso de pipelines para integrar el preprocesamiento (escalado de características con StandardScaler) y las estrategias de balanceo de clases (ADASYN en Random Forest, class_weight='balanced' en SVM y ningún remuestreo explícito en XGBoost) directamente con los clasificadores. La optimización de hiperparámetros se llevó a cabo de forma sistemática mediante GridSearchCV y validación cruzada estratificada, priorizando el recall para la clase minoritaria (“resistente”) o, en el caso de XGBoost, nuevamente el recall, buscando minimizar falsos negativos sin recurrir a scale_pos_weight. El conjunto de datos final se dividió en 10 473 genomas para entrenamiento y 4 489 para prueba, usando siempre el hold-out para evaluación final.

Los resultados obtenidos por los tres modelos en el conjunto de prueba revelaron un patrón consistente:

- **Random Forest (optimizado por recall con ADASYN):** A pesar de detectar casi todas las cepas susceptibles (recall = 1,00), el modelo solo acierta el 2 % de las cepas realmente resistentes con el umbral por defecto (0,5). El bajo F1 (0,04) y un AUC-ROC cercano a 0,50 confirman que la matriz binaria de 100 genes no aporta señal suficiente para identificar la resistencia en la fase de validación.
- **SVM (optimizado por recall con class_weight='balanced'):** El SVM incorpora el escalado y penaliza los errores sobre la clase minoritaria mediante class_weight='balanced'. Sin embargo, con umbral 0,5 solo detecta el 3 % de las verdaderas cepas resistentes, y su AUC-ROC de 0,512 indica que apenas mejora un clasificador aleatorio. Al desplazar el umbral para maximizar F1, la métrica mejora en resistencia, pero se sacrifica por completo el accuracy global.
- **XGBoost (optimizado por recall):** Este XGBoost, optimizado exclusivamente por recall, no detecta ninguna cepa resistente con el umbral estándar (0,5). Su AUC-ROC de 0,502 está casi en el azar. Al ajustar el umbral a 0,2275, mejora el F1 en resistencia, pero a costa de un accuracy global muy bajo (0,23).

Tanto Random Forest, SVM y XGBoost, cada uno con su estrategia de balanceo distinto (ADASYN vs. class_weight='balanced' vs. no usar remuestreo), obtienen siempre un accuracy de aproximadamente 0,77, pero sus recalls para la clase “resistente” con umbral 0,5 oscilan entre 0 % y 3 %. Un umbral bajo recupera algo de F1 en “resistente”, pero destruye el accuracy global.

La matriz binaria de presencia/ausencia de los 100 genes más frecuentes explica fácilmente cuándo una cepa es susceptible (por eso recall=1 en la clase mayoritaria), pero no captura información suficiente para discriminar la resistencia—que probablemente dependa de variantes puntuales, combinación de múltiples genes, regulaciones transcripcionales, plásmidos o mutaciones específicas.

El AUC-ROC cercano a 0,50 para los tres algoritmos confirma que, con estas 100 variables binarias, el poder predictivo para “resistente” es prácticamente nulo.

Por otro lado, aunque se disponían de más de 14 000 genomas inicialmente, no todos tenían valores de MIC o estaban probados con cada carbapenémico. En los análisis complementarios (por ejemplo, correlación entre número de genes y MIC) se observó que la relación es débil (Pearson $r \approx -0{,}19$, Spearman $\rho \approx -0{,}30$).

En conclusión, ninguno de los tres modelos alcanzó simultáneamente un accuracy ≥ 80 % y un recall aceptable (por ejemplo, ≥ 50 %) para la clase “resistente” sin comprometer severamente otras métricas ni el accuracy global. Aunque todos los algoritmos identifican con alta fiabilidad la clase “susceptible” (recall = 1,00), la predicción confiable de la resistencia a carbapenémicos—basada únicamente en la presencia/ausencia de los 100 genes más frecuentes—no resultó posible en este estudio.

Estas observaciones sugieren que la información aportada por la simple binarización de los genes más comunes es insuficiente para captar la complejidad biológica de la resistencia a carbapenémicos en *K. pneumoniae*. Para mejorar la capacidad predictiva, sería necesario enriquecer el conjunto de características que capturen patrones más finos y específicos asociados a la resistencia.

En última instancia, aunque los modelos implementados y evaluados fueron correctos desde el punto de vista técnico y metodológico, los datos genómicos utilizados no proveen la potencia discriminativa necesaria para un uso clínico confiable. El hallazgo principal de este estudio es que, bajo las condiciones y características empleadas, la detección robusta de cepas resistentes a carbapenémicos con machine learning requiere datos genómicos de mayor resolución o información complementaria.

5.2. TRABAJOS FUTUROS

A pesar de las limitaciones encontradas en el rendimiento actual, este estudio sienta las bases para futuras investigaciones y mejoras en el modelado predictivo de la resistencia antimicrobiana. Se proponen las siguientes líneas de trabajo:

- **Selección de Características Avanzada:** Aplicar métodos de selección de características más sofisticados, como técnicas basadas en la interpretabilidad del modelo (ej. SHAP values, LIME para identificar características importantes), o métodos bioinformáticos que identifiquen regiones genómicas de interés más allá de los genes previamente anotados, que puedan estar fuertemente asociadas con la resistencia.
- **Exploración de Arquitecturas de Modelos Diferentes:** Considerar la aplicación de modelos de *Deep Learning*, como las Redes Neuronales Convolucionales (CNNs) o Recurrentes (RNNs) si los datos genómicos pudieran ser representados de una manera que explote su estructura secuencial, o redes neuronales densas con arquitecturas adaptadas al desbalance.
- **Estrategias de Balanceo Híbridas y Más Avanzadas:** Probar combinaciones de técnicas de sobremuestreo y submuestreo (ej., SMOTEENN, SMOTETomek) o algoritmos específicos para desbalance que integran el balanceo en el proceso de entrenamiento (ej., LightGBM con is_unbalance=True).
- **Optimización Multiobjetivo y Umbral de Clasificación:** En lugar de optimizar por una única métrica, explorar la optimización multiobjetivo para encontrar un mejor equilibrio entre precision y recall para la clase minoritaria.
- **Validación Externa:** Una vez que se obtenga un modelo prometedor, validarlo con un conjunto de datos completamente independiente (de otro estudio o laboratorio) para asegurar su generalizabilidad y robustez.

Este proyecto ha subrayado la complejidad de predecir la resistencia antimicrobiana a partir de datos genómicos utilizando métodos de *machine learning*, especialmente en presencia de un desbalance de clases pronunciado y la potencial necesidad de características más específicas y discriminantes. Los hallazgos proporcionan una base sólida para futuros esfuerzos en el desarrollo de herramientas predictivas clínicamente relevantes en la lucha contra la resistencia a los antibióticos.

6. GLOSARIO

Accuracy (Exactitud): Proporción de predicciones correctas (tanto verdaderos positivos como verdaderos negativos) sobre el total de casos evaluados.

ADASYN (Adaptive Synthetic Sampling): Técnica de sobremuestreo que genera ejemplos sintéticos de la clase minoritaria, enfocándose en las muestras que son más difíciles de clasificar, para abordar el desbalance de clases.

Algoritmo de Machine Learning: Conjunto de reglas o procedimientos que un sistema informático sigue para realizar una tarea específica, como la clasificación o predicción, a partir de datos.

Carbapenémicos: Clase de antibióticos de amplio espectro, a menudo utilizados como último recurso contra infecciones bacterianas graves.

Clase Mayoritaria: En un conjunto de datos desbalanceado, la clase que tiene un número significativamente mayor de instancias.

Clase Minoritaria: En un conjunto de datos desbalanceado, la clase que tiene un número significativamente menor de instancias, a menudo la clase de interés (en este caso, la resistencia).

class_weight='balanced': Parámetro utilizado en algunos algoritmos de machine learning (como SVM) que ajusta el peso de las clases inversamente proporcional a sus frecuencias en los datos de entrenamiento, dando más importancia a la clase minoritaria.

Conjunto de Prueba (Test Set): Subconjunto de datos utilizado para evaluar el rendimiento final de un modelo entrenado, que no ha sido visto por el modelo durante el entrenamiento ni la optimización.

Conjunto de Entrenamiento (Training Set): Subconjunto de datos utilizado para entrenar un modelo de machine learning y ajustar sus parámetros.

Corte (Threshold): Valor numérico que se utiliza para clasificar una predicción probabilística en una clase binaria (ej., si la probabilidad es > 0.5, clasificar como 1).

Desbalance de Clases (Class Imbalance): Situación en un conjunto de datos de clasificación donde el número de instancias de una clase (mayoritaria) es significativamente mayor que el de otra(s) clase(s) (minoritaria).

F1-score: Media armónica de la precisión y el recall, una métrica que busca un equilibrio entre ambas, especialmente útil en conjuntos de datos desbalanceados.

Falso Negativo (FN): Un caso positivo real que el modelo clasificó incorrectamente como negativo. (Ej., una cepa resistente clasificada como susceptible).

Falso Positivo (FP): Un caso negativo real que el modelo clasificó incorrectamente como positivo. (Ej., una cepa susceptible clasificada como resistente).

Fuga de Datos (Data Leakage): Situación indeseable en machine learning donde la información del conjunto de prueba se filtra accidentalmente al conjunto de entrenamiento, llevando a una sobreestimación del rendimiento del modelo.

GridSearchCV: Herramienta de scikit-learn para realizar una búsqueda exhaustiva sobre un conjunto de valores de hiperparámetros predefinidos para encontrar la combinación óptima que maximice una métrica de evaluación.

Hiperparámetros: Parámetros de un algoritmo de machine learning que no se aprenden directamente de los datos, sino que se configuran antes del entrenamiento (ej., n_estimators, max_depth en Random Forest).

Klebsiella pneumoniae: Bacteria gramnegativa comúnmente asociada con infecciones nosocomiales, conocida por su capacidad para desarrollar resistencia a antibióticos.

Pipeline (Tubería): En scikit-learn (y imblearn), una secuencia de transformaciones y un estimador final que se encadenan, asegurando que los pasos de preprocesamiento se apliquen consistentemente y eviten la fuga de datos.

Precisión (Precision): Proporción de verdaderos positivos sobre el total de casos clasificados como positivos. Responde a la pregunta: "De todos los que predije como positivos, ¿cuántos eran realmente positivos?".

Random Forest: Algoritmo de machine learning basado en el ensamblaje de múltiples árboles de decisión.

Recall (Sensibilidad o Tasa de Verdaderos Positivos): Proporción de verdaderos positivos sobre el total de casos que eran realmente positivos. Responde a la pregunta: "De todos los casos que eran realmente positivos, ¿cuántos predije correctamente?".

scale_pos_weight: Parámetro específico de algoritmos de gradient boosting (como XGBoost) que ajusta el peso de la clase positiva en la función de pérdida para manejar el desbalance de clases.

StandardScaler: Transformador de scikit-learn que estandariza las características restando la media y dividiendo por la desviación estándar, resultando en datos con media 0 y varianza 1.

StratifiedKFold: Estrategia de validación cruzada que divide el conjunto de datos en K pliegues (folds), manteniendo la proporción de clases en cada pliegue.

Support Vector Machine (SVM): Algoritmo de machine learning que busca el hiperplano óptimo para separar las clases en un espacio de características.

Verdadero Negativo (TN): Un caso negativo real que el modelo clasificó correctamente como negativo.

Verdadero Positivo (TP): Un caso positivo real que el modelo clasificó correctamente como positivo.

XGBoost: Algoritmo de gradient boosting optimizado, conocido por su velocidad y rendimiento en problemas de clasificación y regresión.

7. BIBLIOGRAFÍA

[1] El País. (2024, septiembre 17). La resistencia a los antibióticos: el problema que amenaza con matar a 208 millones de personas en 25 años. *El País*. Recuperado de https://elpais.com/salud-y-bienestar/2024-09-16/la-resistencia-a-los-antibioticos-el-problema-que-amenaza-con-matar-a-208-millones-de-personas-en-25-anos.html

[2] National Library of Medicine. (2023, febrero 24). Using Machine learning to Predict Antimicrobial Resistance – A Literature Review *PMC PubMed Central*. Recuperado de https://pmc.ncbi.nlm.nih.gov/articles/PMC10044642/

[3] UNED Ourense. (2022, octubre 13). Deep learning para estudiar la resistencia de las bacterias a los antibióticos en la Estación Espacial Internacional. *UNED Ourense*. Recuperado de https://www.unedourense.es/noticias/noticia/deep-learning-en-la-uned-para-estudiar-la-resistencia-de-las-bacterias

[4] CIBERINFEC. (2023, diciembre 12). Modelos de machine learning para resistencias a los antibióticos: nuevo proyecto CIBER a través de la Fundación Soria-Melguizo. *CIBERINFEC*. Recuperado de https://www.ciberinfec.es/noticias/modelos-de-machine-learning-para-resistencias-a-los-antibioticos-nuevo-proyecto-ciber-a-traves-de-la-fundacion-soria-melguizo

[5] Logan, L. K., & Weinstein, R. A. (2017). The epidemiology of carbapenem-resistant *Enterobacteriaceae*: The impact and evolution of a global menace. *The Journal of Infectious Diseases, 215*(suppl_1), S28–S36. https://doi.org/10.1093/infdis/jiw282

[6] World Health Organization. (2017). *Global priority list of antibiotic-resistant bacteria to guide research, discovery, and development of new antibiotics*. https://remed.org/wp-content/uploads/2017/03/lobal-priority-list-of-antibiotic-resistant-bacteria-2017.pdf

[7] Khaledi, A., Weimann, A., Schniederjans, M., Asgari, E., Kuo, T. H., Oliver, A., Cabot, G., Kola, A., Gastmeier, P., Hogardt, M., Jonas, D., Mofrad, M. R., Bremges, A., McHardy, A. C., & Häussler, S. (2020). Predicting antimicrobial resistance in *Pseudomonas aeruginosa* with machine learning-enabled molecular diagnostics. *EMBO Molecular Medicine, 12*(3), e10264. https://doi.org/10.15252/emmm.201910264

[8] Sakagianni, A., Koufopoulou, C., Feretzakis, G., Kalles, D., Verykios, V. S., Myrianthefs, P., & Fildisis, G. (2023). Using machine learning to predict antimicrobial resistance—A literature review. *Antibiotics, 12*(3), 452. https://doi.org/10.3390/antibiotics12030452

[9] National Geographic España. (2022, abril 3). *En 2050, la resistencia a los antibióticos será responsable de 10 millones de muertes anuales*. National Geographic. Recuperado de https://www.nationalgeographic.com.es/ciencia/2050-resistencia-a-antibioticos-sera-responsable-10-millones-muertes-anuales_18090

[10] Organización Mundial de la Salud. (2021, noviembre 17). *Resistencia a los antimicrobianos*. OMS. Recuperado de https://www.who.int/es/news-room/fact-sheets/detail/antimicrobial-resistance

[11] World Health Organization (2017). WHO publishes list of bacteria for which new antibiotics are urgently needed. Recuperado de https://www.who.int/news/item/27-02-2017-who-publishes-list-of-bacteria-for-which-new-antibiotics-are-urgently-needed

[12] Martin, R. M., & Bachman, M. A. (2018). Colonization, infection, and the accessory genome of *Klebsiella pneumoniae*. *Frontiers in Cellular and Infection Microbiology*, 8, 4. https://doi.org/10.3389/fcimb.2018.00004

[13] Navon-Venezia, S., Kondratyeva, K., & Carattoli, A. (2017). *Klebsiella pneumoniae*: a major worldwide source and shuttle for antibiotic resistance. *FEMS Microbiology Reviews*, 41(3), 252–275. https://doi.org/10.1093/femsre/fux013

[14] Navon-Venezia, S., Kondratyeva, K., & Carattoli, A. (2017). *Klebsiella pneumoniae*: a major worldwide source and shuttle for antibiotic resistance. *FEMS Microbiology Reviews*, 41(3), 252–275. https://doi.org/10.1093/femsre/fux013

[15] Fundación Innovación Bankinter. (2024). *Algoritmos para luchar contra las bacterias*. Fundación Bankinter. Recuperado de https://www.fundacionbankinter.org/noticias/algoritmos-para-luchar-contra-las-bacterias/

[16] CIO. (2022, octubre 13). *La resistencia de las bacterias a los antibióticos, a prueba de deep learning*. CIO. Recuperado de https://www.cio.com/article/2072970/la-resistencia-de-las-bacterias-a-los-antibioticos-a-prueba-de-deep-learning.html

[17] Tao, X., Chen, H., Zhou, Y., Shi, X., & Chen, Y. (2022). Machine learning models for predicting antimicrobial resistance: A review. *International Journal of Antimicrobial Agents, 60*(1), 106612. https://doi.org/10.1016/j.ijantimicag.2022.106612

[18] Yi HC, You ZH, Huang DS, Kwoh CK. Graph representation learning in bioinformatics: trends, methods and applications. Brief Bioinform. 2022 Jan 17;23(1):bbab340. doi: 10.1093/bib/bbab340. PMID: 34471921.

8. ANEXOS

I. SCRIPT DE CARGA Y PREPROCESAMIENTO DE DATOS

```
# Carga y preprocesamiento de datos

import pandas as pd
import re
from Bio import SeqIO
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split

# Rutas a archivos
path_genoma      =
'C:/Users/judit/OneDrive/Escritorio/MÁSTER/TFM/data/BVBRC_genome_amr.xlsx'
path_genes       =
'C:/Users/judit/OneDrive/Escritorio/MÁSTER/TFM/data/BVBRC_sp_gene.xlsx'
path_genes_fasta =
'C:/Users/judit/OneDrive/Escritorio/MÁSTER/TFM/data/BVBRC_genome_feature.fast
a'

# 1. Leer datos fenotipos y SP-Genes
df_genoma = pd.read_excel(path_genoma, dtype=str)
df_genes  = pd.read_excel(path_genes, dtype=str)
if 'BRC ID' not in df_genes.columns:
    raise ValueError("Falta la columna 'BRC ID' en df_genes.")

# 2. Procesar FASTA para extraer fasta_entries
fasta_entries = []
for record in SeqIO.parse(path_genes_fasta, "fasta"):
    parts = record.id.split('|')
    if len(parts) >= 2 and parts[0] == 'fig':
        brc_id = f"{parts[0]}|{parts[1]}"
        m = re.match(r'(\d+\.\d+)', parts[1])
        genome_id = m.group(1) if m else None
        if genome_id:
            fasta_entries.append({
                'BRC_ID_completo': brc_id,
                'Genome_ID': genome_id
            })

# 3. Crear df_fasta y gene_presence
df_fasta = pd.DataFrame(fasta_entries)
df_fasta['presence'] = 1
df_fasta['Genome ID'] = df_fasta['Genome_ID'].str.strip().str.lower()

gene_presence = df_fasta.pivot_table(
    index='Genome ID',
    columns='BRC_ID_completo',
    values='presence',
    aggfunc='max',
    fill_value=0
)
gene_presence.index = gene_presence.index.str.strip().str.lower()
```

```
# 4. Preparar df_genoma: quedarnos solo con “Genome ID” y binarizar
“Resistance”
df_genoma['Genome ID'] = df_genoma['Genome
ID'].astype(str).str.strip().str.lower()
df_genoma['Resistance'] = df_genoma['Resistant Phenotype'].map({'Resistant':
1, 'Susceptible': 0})
df_genoma = df_genoma[['Genome ID', 'Resistance']]

# 5. Merge “fenotipo + presencia de genes” (solo genomas comunes)
df_full = df_genoma.merge(
    gene_presence,
    left_on='Genome ID',
    right_index=True,
    how='inner'
).fillna(0)

# 6. Separar X_all e y_all
y_all = df_full['Resistance']
X_all = df_full.drop(columns=['Resistance', 'Genome ID'])

# 7. División estratificada en entrenamiento y prueba (70/30)
X_train_all, X_test_all, y_train, y_test = train_test_split(
    X_all, y_all,
    test_size=0.30,
    stratify=y_all,
    random_state=42
)

print("Tamaño X_train_all:", X_train_all.shape[0])
print("Tamaño X_test_all: ", X_test_all.shape[0])

# 8. Selección de top 100 genes SOLO sobre X_train_all
gene_counts_train = X_train_all.sum(axis=0)
selected_genes =
gene_counts_train.sort_values(ascending=False).head(100).index.tolist()
print("Cantidad de genomas en entrenamiento:", X_train_all.shape[0])
print("Genes seleccionados (top 100 sobre train):", len(selected_genes))

# 9. Filtrar X_train_all y X_test_all a esas columnas seleccionadas
X_train = X_train_all[selected_genes].copy()
X_test  = X_test_all[selected_genes].copy()

# 10. Escalar con MinMaxScaler (RF + ADASYN) y StandardScaler (SVM/XGB)
scaler_mm  = MinMaxScaler()
scaler_std = StandardScaler()

X_train_scaled_mm = scaler_mm.fit_transform(X_train)
X_test_scaled_mm  = scaler_mm.transform(X_test)

X_train_scaled_std = scaler_std.fit_transform(X_train)
X_test_scaled_std  = scaler_std.transform(X_test)
```

II. SCRIPT DE ESTADÍSTICA DESCRIPTIVA PREVIA AL MODELADO

```
import matplotlib.pyplot as plt

# Para no modificar df_full original, trabajamos con una copia
df_desc = df_full.copy()
```

```
# 2.1 Distribución global de fenotipos
phenotype_counts = df_desc['Resistance'].value_counts().sort_index()
phenotype_counts.index = ['Susceptible (0)', 'Resistant (1)']

plt.figure(figsize=(6, 4))
phenotype_counts.plot(kind='bar', color=['tab:blue', 'tab:orange'])
plt.title('Distribución de fenotipos')
plt.xlabel('Fenotipo')
plt.ylabel('Número de genomas')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()

# 2.2 Número de genes presentes por genoma (solo top 100)
#     Primero, identificar todas las columnas “gene” en df_desc
gene_cols = df_desc.columns.difference(['Genome ID', 'Resistance'])
df_desc['Num_Genes'] = df_desc[gene_cols].sum(axis=1)

plt.figure(figsize=(8, 5))
bins = range(int(df_desc['Num_Genes'].min()), int(df_desc['Num_Genes'].max())
+ 2)
plt.hist(
    [ df_desc[df_desc['Resistance'] == 0]['Num_Genes'],
      df_desc[df_desc['Resistance'] == 1]['Num_Genes'] ],
    bins=bins,
    label=['Susceptible (0)', 'Resistant (1)'],
    color=['tab:blue','tab:orange'],
    alpha=0.7,
    edgecolor='black'
)
plt.title('Histograma: número de genes por genoma (por fenotipo)')
plt.xlabel('Número de genes presentes')
plt.ylabel('Número de genomas')
plt.legend()
plt.tight_layout()
plt.show()

# 2.3 Top 20 genes más frecuentes en todo df_full
gene_frequency =
df_desc[gene_cols].sum().sort_values(ascending=False).head(20)
plt.figure(figsize=(10, 5))
gene_frequency.plot(kind='bar', color='tab:green')
plt.title('Top 20 genes más frecuentes (presencia/ausencia)')
plt.xlabel('Identificador de gen (BRC_ID_completo)')
plt.ylabel('Frecuencia (# genomas con el gen)')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

# 2.4 Frecuencia de top 10 genes en resistentes vs susceptibles
top10_genes = gene_frequency.head(10).index

freq_resistant   = df_desc[df_desc['Resistance'] == 1][top10_genes].sum()
freq_susceptible = df_desc[df_desc['Resistance'] == 0][top10_genes].sum()

x = range(len(top10_genes))
width = 0.35
```

```
plt.figure(figsize=(10, 5))
plt.bar(x,             freq_susceptible, width=width, label='Susceptible
(0)', color='tab:blue',    edgecolor='black')
plt.bar([i+width for i in x], freq_resistant,   width=width, label='Resistant
(1)',  color='tab:orange',  edgecolor='black')

plt.title('Frecuencia top 10 genes en Resistentes vs Susceptibles')
plt.xlabel('Identificador de gen (BRC_ID_completo)')
plt.ylabel('Número de genomas con el gen')
plt.xticks([i+width/2 for i in x], top10_genes, rotation=90)
plt.legend()
plt.tight_layout()
plt.show()

# 2.5 Estadísticas descriptivas de "Num_Genes" por fenotipo
stats_by_pheno = df_desc.groupby('Resistance')['Num_Genes']\
    .describe()\
    .rename(index={0:'Susceptible', 1:'Resistant'})
print("Estadísticas de número de genes por genoma (por fenotipo):\n")
print(stats_by_pheno)

# Eliminamos la columna temporal para no contaminar df_desc
df_desc.drop(columns=['Num_Genes'], inplace=True)
```

III. SCRIPT DEL MODELO RANDOM FOREST CON ADASYN

```
# Modelo 1 – Random Forest con ADASYN

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from imblearn.over_sampling import ADASYN
from sklearn.metrics import classification_report, roc_auc_score,
precision_recall_curve

print("\n--- Modelo 1: Random Forest + ADASYN ---")

# 3.1 Aplicar ADASYN a X_train_scaled_mm
adasyn = ADASYN(random_state=42)
X_res, y_res = adasyn.fit_resample(X_train_scaled_mm, y_train)

print("Distribución en y_res (después de ADASYN):")
print(pd.Series(y_res).value_counts())

# 3.2 Validación cruzada estratificada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 3.3 Grid de hiperparámetros para RF
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# 3.4 GridSearchCV optimizado por recall
grid_rf = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid_rf,
    cv=skf,
```

```
    scoring='recall',
    n_jobs=-1,
    verbose=1
)
grid_rf.fit(X_res, y_res)

print("Best RF Params (Recall+ADASYN):", grid_rf.best_params_)

# 3.5 Evaluación final en X_test_scaled_mm
y_pred_rf = grid_rf.predict(X_test_scaled_mm)
print("Test Report RF (umbral 0.5):")
print(classification_report(y_test, y_pred_rf, zero_division=0))

AUC-ROC y umbral óptimo para Random Forest

# 1. Sacar probabilidades de la clase “1”
y_prob_rf = grid_rf.predict_proba(X_test_scaled_mm)[:, 1]

# 2. Calcular AUC-ROC
auc_rf = roc_auc_score(y_test, y_prob_rf)
print("\nAUC-ROC RF:", f"{auc_rf:.4f}")

# 3. Buscar umbral óptimo (F1) en test
prec_r, rec_r, thr_r = precision_recall_curve(y_test, y_prob_rf)
best_f1_r, best_thr_r = 0, 0.5
for p, r, t in zip(prec_r[:-1], rec_r[:-1], thr_r):
    f1 = 2 * (p * r) / (p + r) if (p + r) > 0 else 0
    if f1 > best_f1_r:
        best_f1_r, best_thr_r = f1, t

print(f"RF umbral óptimo: {best_thr_r:.4f} → F1 = {best_f1_r:.4f}")

# 4. Classification report con umbral óptimo
y_pred_rf_opt = (y_prob_rf >= best_thr_r).astype(int)
print("RF Report (umbral óptimo):")
print(classification_report(y_test, y_pred_rf_opt, zero_division=0))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Confusion Matrix umbral 0.5
cm_rf_default = confusion_matrix(y_test, y_pred_rf)
disp_rf_default = ConfusionMatrixDisplay(cm_rf_default,
display_labels=['Susceptible (0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_rf_default.plot(cmap='Blues', values_format='d')
plt.title('RF - Matriz Confusión (umbral 0.5)')
plt.show()

# Confusion Matrix umbral óptimo
cm_rf_opt = confusion_matrix(y_test, y_pred_rf_opt)
disp_rf_opt = ConfusionMatrixDisplay(cm_rf_opt, display_labels=['Susceptible
(0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_rf_opt.plot(cmap='Oranges', values_format='d')
plt.title(f'RF - Matriz Confusión (umbral {best_thr_r:.4f})')
plt.show()
```

IV. SCRIPT DEL MODELO SVM CON CLASS_WEIGHT = 'BALANCED'

```
# Modelo 2 – SVM con class_weight='balanced' (optimizar por F1)

from sklearn.svm import SVC

print("\n--- Modelo 2: SVM (class_weight='balanced') ---")

# 1. Escalar con StandardScaler
X_train_svm = X_train_scaled_std
X_test_svm  = X_test_scaled_std

print("Distribución en y_train:", y_train.value_counts())

# 2. Validación cruzada estratificada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 3. Grid de hiperparámetros para SVM
param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto', 0.1, 1],
    'kernel': ['rbf']
}

# 4. GridSearchCV optimizado por F1
grid_svm = GridSearchCV(
    SVC(class_weight='balanced', probability=False, random_state=42),
    param_grid_svm,
    cv=skf,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)
grid_svm.fit(X_train_svm, y_train)

print("Best SVM Params:", grid_svm.best_params_)
y_pred_svm = grid_svm.predict(X_test_svm)
print("Test Report SVM:")
print(classification_report(y_test, y_pred_svm, zero_division=0))

# AUC-ROC y umbral óptimo para SVM

# 1. Sacar scores con decision_function
y_scores_svm = grid_svm.decision_function(X_test_svm)

# 2. Calcular AUC-ROC
auc_svm = roc_auc_score(y_test, y_scores_svm)
print("\nAUC-ROC SVM:", f"{auc_svm:.4f}")

# 3. Buscar umbral óptimo (F1) en test
prec_s, rec_s, thr_s = precision_recall_curve(y_test, y_scores_svm)
best_f1_s, best_thr_s = 0, 0.0
for p, r, t in zip(prec_s[:-1], rec_s[:-1], thr_s):
    f1 = 2 * (p * r) / (p + r) if (p + r) > 0 else 0
    if f1 > best_f1_s:
        best_f1_s, best_thr_s = f1, t

print(f"SVM umbral óptimo: {best_thr_s:.4f} → F1 = {best_f1_s:.4f}")

# 4. Classification report a umbral 0 (score ≥ 0 → "1")
```

```
y_pred_05 = (y_scores_svm >= 0.0).astype(int)
print("\nClassification Report (SVM a umbral 0):")
print(classification_report(y_test, y_pred_05, zero_division=0))

# 5. Classification report a umbral óptimo
y_pred_svm_opt = (y_scores_svm >= best_thr_s).astype(int)
print(f"\nClassification Report (SVM a umbral {best_thr_s:.4f}):")
print(classification_report(y_test, y_pred_svm_opt, zero_division=0))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Confusion Matrix umbral 0 (decision_function ≥ 0)
cm_svm_default = confusion_matrix(y_test, y_pred_05)
disp_svm_default = ConfusionMatrixDisplay(cm_svm_default,
display_labels=['Susceptible (0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_svm_default.plot(cmap='Greens', values_format='d')
plt.title('SVM - Matriz Confusión (umbral 0)')
plt.show()

# Confusion Matrix umbral óptimo
cm_svm_opt = confusion_matrix(y_test, y_pred_svm_opt)
disp_svm_opt = ConfusionMatrixDisplay(cm_svm_opt,
display_labels=['Susceptible (0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_svm_opt.plot(cmap='Purples', values_format='d')
plt.title(f'SVM - Matriz Confusión (umbral {best_thr_s:.4f})')
plt.show()
```

V. SCRIPT DEL MODELO XGBOOST OPTIMIZADO POR RECALL

```
# Modelo 3 – XGBoost (optimizar por recall)

from xgboost import XGBClassifier

print("\n--- Modelo 3: XGBoost (optimizado por Recall) ---")

# 1. Validación cruzada estratificada
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# 2. Grid de hiperparámetros para XGBoost
param_grid_xgb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1],
    'colsample_bytree': [0.8, 1]
}

# 3. GridSearchCV optimizado por recall
grid_xgb = GridSearchCV(
    XGBClassifier(objective='binary:logistic', random_state=42),
    param_grid_xgb,
    cv=skf,
    scoring='recall',
    n_jobs=-1,
    verbose=1
)
```

```
grid_xgb.fit(X_train_svm, y_train)  # X_train_svm = X_train_scaled_std

print("Best XGBoost Params (Recall):", grid_xgb.best_params_)
y_pred_xgb = grid_xgb.predict(X_test_svm)
print("Test Report XGBoost:")
print(classification_report(y_test, y_pred_xgb, zero_division=0))

# AUC-ROC y umbral óptimo para XGBoost

# 1. Sacar probabilidades de la clase "1"
y_prob_xgb = grid_xgb.predict_proba(X_test_svm)[:, 1]

# 2. Calcular AUC-ROC
auc_xgb = roc_auc_score(y_test, y_prob_xgb)
print("\nAUC-ROC XGBoost:", f"{auc_xgb:.4f}")

# 3. Buscar umbral óptimo (F1) en test
prec_x, rec_x, thr_x = precision_recall_curve(y_test, y_prob_xgb)
best_f1_x, best_thr_x = 0, 0.5
for p, r, t in zip(prec_x[:-1], rec_x[:-1], thr_x):
    f1 = 2 * (p * r) / (p + r) if (p + r) > 0 else 0
    if f1 > best_f1_x:
        best_f1_x, best_thr_x = f1, t

print(f"XGB umbral óptimo: {best_thr_x:.4f} → F1 = {best_f1_x:.4f}")

# 4. Classification report a umbral 0.5
y_pred_05_x = (y_prob_xgb >= 0.5).astype(int)
print("\nClassification Report (XGBoost a umbral 0.5):")
print(classification_report(y_test, y_pred_05_x, zero_division=0))

# 5. Classification report a umbral óptimo
y_pred_xgb_opt = (y_prob_xgb >= best_thr_x).astype(int)
print(f"\nClassification Report (XGBoost a umbral {best_thr_x:.4f}):")
print(classification_report(y_test, y_pred_xgb_opt, zero_division=0))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Confusion Matrix umbral 0.5
cm_xgb_default = confusion_matrix(y_test, y_pred_05_x)
disp_xgb_default = ConfusionMatrixDisplay(cm_xgb_default,
display_labels=['Susceptible (0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_xgb_default.plot(cmap='Reds', values_format='d')
plt.title('XGBoost - Matriz Confusión (umbral 0.5)')
plt.show()

# Confusion Matrix umbral óptimo
cm_xgb_opt = confusion_matrix(y_test, y_pred_xgb_opt)
disp_xgb_opt = ConfusionMatrixDisplay(cm_xgb_opt,
display_labels=['Susceptible (0)', 'Resistant (1)'])
plt.figure(figsize=(5, 5))
disp_xgb_opt.plot(cmap='cool', values_format='d')
plt.title(f'XGBoost - Matriz Confusión (umbral {best_thr_x:.4f})')
plt.show()
```

VI. SCRIPT DE LAS CURVAS ROC

```
from sklearn.metrics import roc_curve, auc

# 1. Random Forest: usamos predict_proba directamente
y_prob_rf = grid_rf.predict_proba(X_test_rf)[:, 1]

# 2. SVM: usamos decision_function en lugar de predict_proba
decision_scores_svm = grid_svm.best_estimator_.decision_function(X_test_svm)
y_prob_svm = decision_scores_svm

# 3. XGBoost: usamos predict_proba
y_prob_xgb = grid_xgb.predict_proba(X_test_xgb)[:, 1]

# 1. Calcular las curvas ROC para cada modelo
fpr_rf, tpr_rf, _    = roc_curve(y_test, y_prob_rf)
fpr_svm, tpr_svm, _  = roc_curve(y_test, y_prob_svm)
fpr_xgb, tpr_xgb, _  = roc_curve(y_test, y_prob_xgb)

# 2. Calcular el AUC para cada curva
auc_rf  = auc(fpr_rf, tpr_rf)
auc_svm = auc(fpr_svm, tpr_svm)
auc_xgb = auc(fpr_xgb, tpr_xgb)

# 3. Dibujar las tres curvas en un mismo gráfico
plt.figure(figsize=(7, 6))

plt.plot(fpr_rf,  tpr_rf,  label=f"Random Forest (AUC = {auc_rf:.4f})",
linewidth=2)
plt.plot(fpr_svm, tpr_svm, label=f"SVM           (AUC = {auc_svm:.4f})",
linewidth=2)
plt.plot(fpr_xgb, tpr_xgb, label=f"XGBoost       (AUC = {auc_xgb:.4f})",
linewidth=2)

# Línea de referencia: clasificador aleatorio
plt.plot([0, 1], [0, 1], 'k--', label="Azar (AUC = 0.5000)")

plt.title("Curvas ROC para Random Forest, SVM y XGBoost")
plt.xlabel("FPR (Tasa Falsos Positivos)")
plt.ylabel("TPR (Recall / Tasa Verdaderos Positivos)")
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

VII. SCRIPT DE LOS ANÁLISIS COMPLEMENTARIOS

```
# Análisis de correlación entre recuento de genes y valores de MIC

import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import pearsonr, spearmanr

df_genoma_raw = pd.read_excel(path_genoma, dtype=str)

# 2. Normalizar 'Genome ID' y convertir MIC a numérico
df_genoma_raw['Genome ID'] = df_genoma_raw['Genome
ID'].astype(str).str.strip().str.lower()
df_genoma_raw['MIC_Value'] = pd.to_numeric(df_genoma_raw['Measurement
Value'], errors='coerce')
```

```
# 3. Calcular recuento de genes por genoma en df_full
gene_cols = df_full.columns.difference(['Genome ID', 'Resistance'])
df_full['Num_Genes'] = df_full[gene_cols].sum(axis=1)

# 4. Elegir un antibiótico (imipenem)
antib = 'imipenem'
df_mic = df_genoma_raw[df_genoma_raw['Antibiotic'].str.lower() ==
antib].copy()

# 5. Unir recuento de genes con MIC
df_merge = df_full.merge(
    df_mic[['Genome ID', 'MIC_Value']],
    on='Genome ID',
    how='inner'
).dropna(subset=['MIC_Value'])

# 6. Scatter plot: Num_Genes vs MIC_Value
plt.figure(figsize=(6, 4))
plt.scatter(df_merge['Num_Genes'], df_merge['MIC_Value'],
            alpha=0.6, color='tab:blue', edgecolor='k')
plt.title(f'Scatter: #genes vs MIC ({antib})')
plt.xlabel('Número de genes')
plt.ylabel('MIC Value')
plt.tight_layout()
plt.show()

# 7. Calcular correlaciones
pearson_corr, p_p = pearsonr(df_merge['Num_Genes'], df_merge['MIC_Value'])
spearman_corr, p_s = spearmanr(df_merge['Num_Genes'], df_merge['MIC_Value'])
print(f"Pearson r = {pearson_corr:.3f}, p-value = {p_p:.3e}")
print(f"Spearman rho = {spearman_corr:.3f}, p-value = {p_s:.3e}")

# Análisis de componentes principales (PCA) y t-SNE

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Definir gene_cols y extraer matriz binaria + etiqueta
gene_cols = [c for c in df_full.columns if c not in ['Genome ID',
'Resistance']]
X_bin = df_full[gene_cols]
y_bin = df_full['Resistance']

# PCA a 2 componentes
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_bin)

plt.figure(figsize=(6, 5))
plt.scatter(
    X_pca[y_bin == 0, 0], X_pca[y_bin == 0, 1],
    alpha=0.6, label='Susceptible (0)', s=15, color='tab:blue'
)
plt.scatter(
    X_pca[y_bin == 1, 0], X_pca[y_bin == 1, 1],
    alpha=0.6, label='Resistant (1)', s=15, color='tab:orange'
)
plt.title("PCA (2D) sobre matriz binaria de genes")
```

```
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.tight_layout()
plt.show()

# t-SNE a 2D
tsne = TSNE(n_components=2, random_state=42, perplexity=30, n_iter=1000)
X_tsne = tsne.fit_transform(X_bin)

plt.figure(figsize=(6, 5))
plt.scatter(
    X_tsne[y_bin == 0, 0], X_tsne[y_bin == 0, 1],
    alpha=0.6, label='Susceptible (0)', s=15, color='tab:blue'
)
plt.scatter(
    X_tsne[y_bin == 1, 0], X_tsne[y_bin == 1, 1],
    alpha=0.6, label='Resistant (1)', s=15, color='tab:orange'
)
plt.title("t-SNE (2D) sobre matriz binaria de genes")
plt.xlabel("t-SNE1")
plt.ylabel("t-SNE2")
plt.legend()
plt.tight_layout()
plt.show()

# Análisis de subtipos de antibióticos (evolución fenotípica)

import pandas as pd
import matplotlib.pyplot as plt

# 1. Leer y preparar df_genoma_raw con MIC_Value
df_genoma_raw = pd.read_excel(path_genoma, dtype=str)
df_genoma_raw['Genome ID'] = df_genoma_raw['Genome
ID'].astype(str).str.strip().str.lower()
df_genoma_raw['MIC_Value'] = pd.to_numeric(df_genoma_raw['Measurement
Value'], errors='coerce')

# 2. Crear columna binaria de resistencia
df_pheno = df_genoma_raw.copy()
df_pheno['Resistance_bin'] = df_pheno['Resistant
Phenotype'].map({'Resistant': 1, 'Susceptible': 0})

# 3. Proporción de resistentes por antibiótico
counts_by_ant = (
    df_pheno
    .groupby(['Antibiotic', 'Resistance_bin'])['Genome ID']
    .nunique()
    .unstack(fill_value=0)
)
counts_by_ant['Total'] = counts_by_ant.sum(axis=1)
counts_by_ant['%Resistant'] = counts_by_ant[1] / counts_by_ant['Total'] * 100

plt.figure(figsize=(6, 4))
counts_by_ant['%Resistant'].plot(kind='bar', color='tab:orange')
plt.title("% de genomas resistentes por antibiótico")
plt.xlabel("Antibiotic")
plt.ylabel("% Resistant")
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()

# 4. Contingencia de resistencia simultánea Imipenem vs Meropenem
imp = set(
    df_pheno[
        (df_pheno['Antibiotic'].str.lower() == 'imipenem') &
        (df_pheno['Resistance_bin'] == 1)
    ]['Genome ID']
)
mer = set(
    df_pheno[
        (df_pheno['Antibiotic'].str.lower() == 'meropenem') &
        (df_pheno['Resistance_bin'] == 1)
    ]['Genome ID']
)
print(f"Resistentes a Imipenem: {len(imp)}")
print(f"Resistentes a Meropenem: {len(mer)}")
print(f"Resistentes a ambos: {len(imp & mer)}")

# 5. Scatter MIC Imipenem vs Meropenem para genomas con ambos valores
df_imp_mic = df_pheno[df_pheno['Antibiotic'].str.lower() ==
'imipenem'][['Genome ID', 'MIC_Value']]
df_mer_mic = df_pheno[df_pheno['Antibiotic'].str.lower() ==
'meropenem'][['Genome ID', 'MIC_Value']]

df_scatter = (
    df_imp_mic.merge(df_mer_mic, on='Genome ID', suffixes=('_imp', '_mer'))
    .dropna(subset=['MIC_Value_imp', 'MIC_Value_mer'])
)

plt.figure(figsize=(6, 5))
plt.scatter(
    df_scatter['MIC_Value_imp'],
    df_scatter['MIC_Value_mer'],
    alpha=0.6, color='tab:purple', edgecolor='k'
)
plt.title("MIC: Imipenem vs Meropenem")
plt.xlabel("MIC Imipenem")
plt.ylabel("MIC Meropenem")
plt.tight_layout()
plt.show()
```