Tecnológico de Monterrey

Inteligencia artificial avanzada para la ciencia de datos I

Uso de framework o biblioteca de aprendizaje máquina para la implementación de una solución

Luis Eduardo Aguilar Moreno

A01367545

Profesores:

Adrián Rodríguez Rocha

Momento de Retroalimentación: Módulo 2

Septiembre de 2023

Introducción

En el presente informe, se llevará a cabo un análisis exhaustivo del rendimiento de diversos modelos de **aprendizaje de maquina (Machine Learning)** en la tarea de evaluar y predecir con precisión los tipos de medicamentos consumidos por individuos en función de ciertos parámetros y características recopiladas. Estos datos se obtuvieron como parte de un conjunto de datos recopilados por Pratham Tripathi, quien a su vez recopiló información de una compañía farmacéutica con el propósito de identificar los medicamentos y los factores que los afectan, con el objetivo de predecir su recomendación para diferentes tipos de pacientes.

Este conjunto de datos se adquirió de Kaggle y cuenta con una licencia de dominio público otorgada por Creative Commons, lo que permite su copia, modificación, distribución y uso sin necesidad de permisos adicionales, haciéndolo adecuado para nuestro propósito.

El archivo **"drug200.csv"** contiene 200 registros que incluyen información sobre cinco parámetros necesarios para identificar el tipo de medicamento suministrado. Estos parámetros son la **edad (Age)**, el **sexo (Sex)**, el **nivel de presión arterial (BP)**, **los niveles de colesterol (Cholesterol Levels)** y la **proporción de sodio a potasio (Na to Potassium Ratio)**. Estos parámetros se utilizan para determinar qué medicamento se administró entre cinco posibles opciones: el **fármaco A (DrugA)**, el **fármaco B (DrugB)**, el **fármaco C (DrugC)**, el **fármaco X (DrugX)** y el **fármaco Y (DrugY)**.

En cuanto a la variable de edad, se observa que la edad promedio es de aproximadamente 44 años, con un rango que varía desde un mínimo de 15 años hasta un máximo de 74 años. En

cuanto al género, los registros están bastante equilibrados, aunque hay una ligera mayoría de hombres.

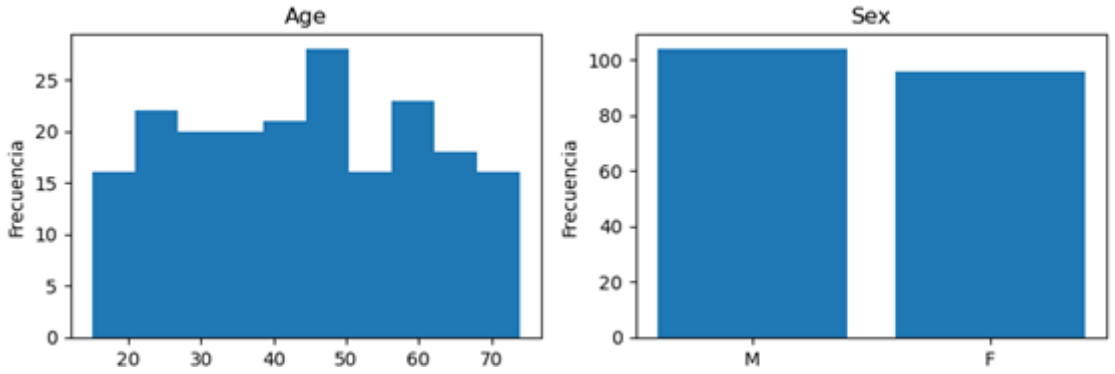

FIGURA 1: Gráficas de edad y sexo en el conjunto de datos "drug200.csv"

La variable del nivel de presión arterial es de naturaleza categórica, aunque podría haberse registrado de forma numérica. Se divide en categorías de presión baja, normal o alta. No hay grandes diferencias en la distribución de los registros, pero se observa una mayor cantidad de registros con presión alta, seguidos de presión baja y, por último, presión normal. De manera similar, la variable de niveles de colesterol se clasifica como alto o normal, con proporciones bastante similares de registros, siendo los niveles altos los más predominantes.

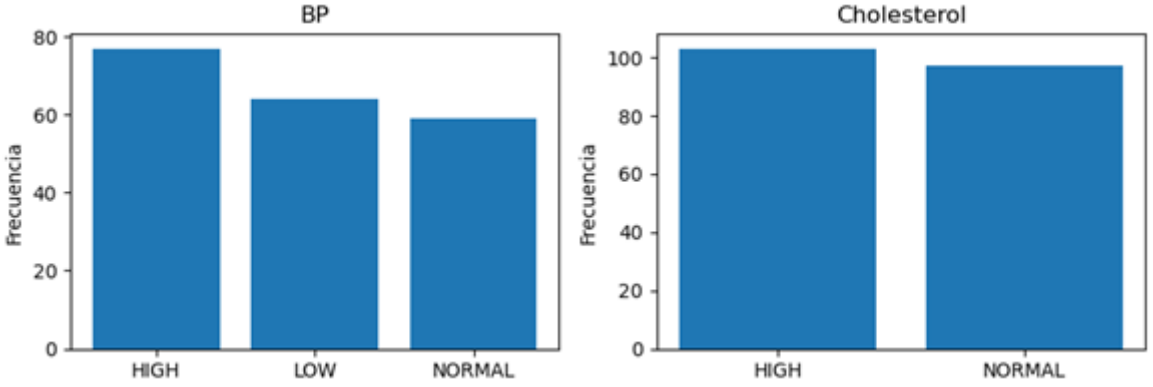

FIGURA 2: Gráficas de BP y Colesterol en el conjunto de datos "drug200.csv"

La proporción de sodio a potasio tiene una media de 16.08, con valores que oscilan entre 6.26 y 38.24. Finalmente, la variable objetivo, es decir, el tipo de fármaco suministrado, es de naturaleza categórica, con las categorías mencionadas previamente. Se observa un claro desequilibrio de clases, con una mayor cantidad de registros en los que se administró el fármaco

Y (más de 80 registros), seguido del fármaco X (más de 50 registros), y luego los fármacos A, B y C.

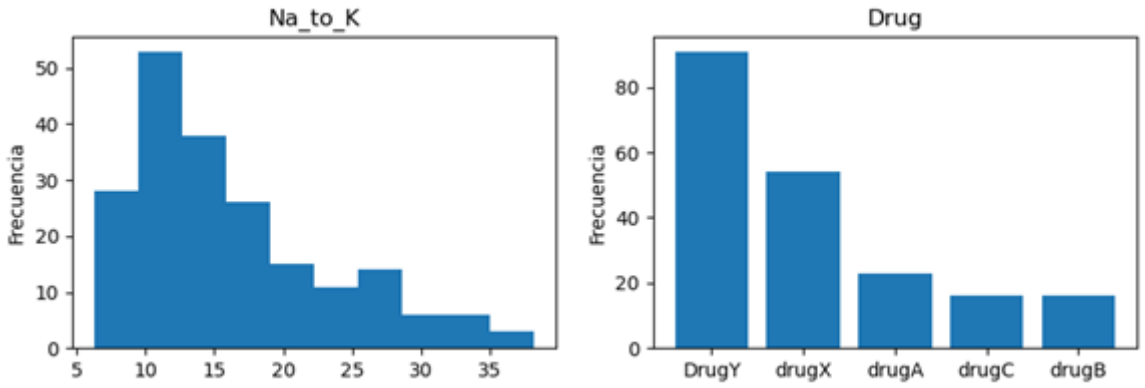

FIGURA 3: Gráficas de la proporción Na-K y fármaco suministrado en el conjunto de datos "drug200.csv"

Es importante destacar que no se encontraron datos faltantes en ninguno de los registros del conjunto de datos, lo que facilita su manejo, ya que no se requiere un proceso extenso de extracción, transformación y carga (ETL) de datos. Sin embargo, debido a la naturaleza de los algoritmos de aprendizaje automático que se utilizarán para construir un modelo adecuado, se ha decidido aplicar la técnica de codificación "One-hot encoding"a las variables categóricas del conjunto de datos. Esto tiene como objetivo simplificar el procesamiento de datos, aunque los modelos son capaces de trabajar con variables categóricas, esta técnica suele mejorar el rendimiento y la interpretación de los modelos.

Finalmente, se dividió el conjunto de datos en dos subconjuntos: el **conjunto de entrenamiento** y el **conjunto de evaluación**. Para la creación del conjunto de entrenamiento, se utilizó el 80% de la cantidad total de datos del conjunto original, mientras que el conjunto de evaluación consistió en el 20% restante del total. La selección de estos datos se realizó de manera aleatoria mediante la función **".train_test_split"** de la biblioteca sklearn.model_selection. Se especificó el parámetro "random_state=42"con el objetivo de garantizar la reproducibilidad en futuras divisiones del conjunto de datos. Como resultado, se obtuvo un conjunto de entrenamiento con 160 registros, compuesto por 5 variables independientes y una variable objetivo, mientras que el conjunto de evaluación consta de 40 registros con las mismas características.

Como parte de la implementación de los modelos de aprendizaje automático, se emplearon

cuatro métodos diferentes de **clasificación multiclase**: **regresión logística (Logistic Regression)**, **máquinas de vectores de soporte (Support Vector Machines)**, **bosques aleatorios (Random Forest)** y **vecinos más cercanos (K Nearest Neighbors)**. Estos algoritmos se evaluaron utilizando diversas métricas, destacando la **exactitud (Accuracy)** y la **matriz de confusión**.

Además, se calcularon los valores de **sesgo** y **varianza** para identificar posibles problemas en los hiperparámetros que pudieran afectar el entrenamiento de los modelos, como el **subajuste o sobreajuste**, según corresponda. Por último, se determinaron los hiperparámetros óptimos para cada uno de los modelos utilizando un algoritmo de **búsqueda aleatoria (Random Search)** junto con **validación cruzada (Cross-Validation)**. Con estos hiperparámetros optimizados, se entrenaron los modelos finales y se realizaron las predicciones correspondientes.

Regresión Logística multiclase

La regresión logística es un algoritmo de aprendizaje de maquina reconocido y empleado en tareas de clasificación. Su principal objetivo radica en predecir la probabilidad de un resultado específico. La regresión logística logra predecir correctamente, mediante el ajuste los datos a una función logística, lo que resulta en la estimación de la probabilidad de que ocurra un evento particular.

La regresión logística multiclase, es un tipo de clasificador de clasificación no binaria. Este algoritmo posee la capacidad de categorizar datos en más de dos clases distintas, lo cual permite enfrentarse a problemas de clasificación que involucran tres o más categorías. Lo que hace que la regresión logística multiclase trace automáticamente fronteras de decisión que delimitan con precisión cada categoría.

El funcionamiento de la regresión logística multiclase se basa en la asignación de ponderaciones a cada característica presente en el conjunto de datos. Estas ponderaciones se combinan para calcular la probabilidad de que un punto de datos en particular pertenezca a una clase específica. Luego, se compara esta probabilidad con las de otras clases para determinar a cuál de ellas se asigna el punto de datos. Este proceso se repite para cada clase en el problema de clasificación.

Este algoritmo tiene una amplia variedad de aplicaciones prácticas, que abarcan desde la clasificación de correos electrónicos como spam o no spam, hasta la categorización de temas en documentos (como deportes, tecnología o política), pasando por la clasificación de productos en diversas categorías o etiquetas, entre otras muchas áreas.

Para la implementación del modelo de regresión logística, se empleó la clase **LogisticRegression()** de la biblioteca **sklearn.linear_model**. Este modelo ofrece la capacidad de ajustar varios hiperparámetros, como los siguientes:

- **Penalty:** Controla el parámetro de regularización en el modelo.
- **C:** Controla la fuerza de regularización. Un valor bajo de C ayuda a prevenir el sobreajuste durante el entrenamiento.
- **Solver:** Especifica el algoritmo de optimización, lo que afecta el rendimiento y la velocidad de convergencia del modelo.
- **Max_iter:** Controla el número máximo de iteraciones para la convergencia.
- **Multi_class:** Define cómo se maneja la clasificación multiclase.
- **Class_weight:** Permite asignar pesos diferentes a las clases, según sea necesario, entre otros.

En el primer modelo de regresión logística, se configuraron los siguientes hiperparámetros: **max_iter=1000**, **multi_class="ovr"**, y **solver="saga"**. Con este modelo, se obtuvo una precisión del 80%. Sin embargo, se observó en la matriz de confusión (4) que no clasificaba completamente correctamente, ya que tendía a clasificar la mayoría de los datos de evaluación como "DrugY".

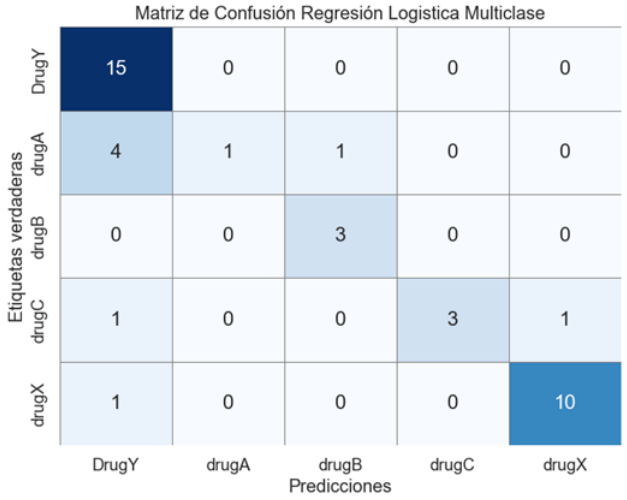

FIGURA 4: Matriz de confusión de LogisticRegression(max_iter=1000, multi_class='ovr',solver="saga")

En el segundo modelo de regresión logística, se establecieron los hiperparámetros como sigue: **C=1.0**, **solver="lbfgs"**, y **multi_class="multinomial"**. Con este modelo, se logró una precisión del 97.5%. Se calculó un sesgo de 0.03, una varianza de 0.00 y un error de 0.03. Al observar la curva de aprendizaje (5), se notó que la precisión en la validación cruzada comenzó en 0.70, mientras que en el entrenamiento casi alcanzaba 1.0. Esto sugiere que el modelo estaba sobreajustado, lo que podría atribuirse a su alta complejidad. Con el tiempo, la brecha entre la precisión del entrenamiento y la validación se redujo, lo que indica un equilibrio gradual entre la varianza y el sesgo, finalizando en la precisión mencionada anteriormente.

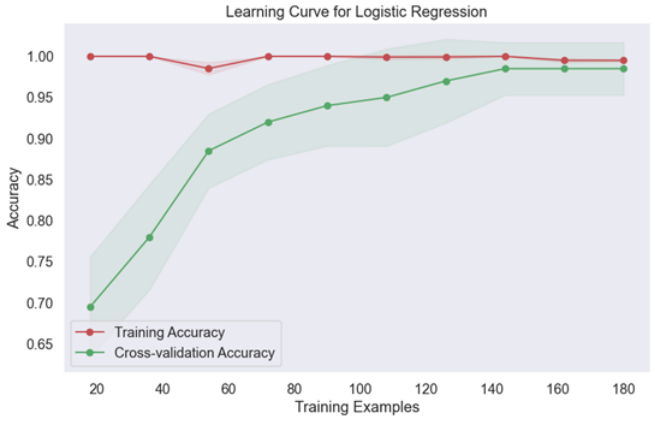

FIGURA 5: Curva de aprendizaje de LogisticRegression(C=1.0,solver='lbfgs',multi_class="multinomial",max_iter=10000)

Para analizar cómo la elección de hiperparámetros afecta al sesgo y la varianza, se realizó una comparación al variar el **hiperparámetro C en un rango de 0.1 a 3.0, junto con el solver utilizado ('lbfgs', 'newton-cg', 'sag')**. La gráfica de sesgo reveló que el valor óptimo de C para minimizar el sesgo estaba entre 1.0 y 1.5, con los solvers "lbfgs"y "newton-cg"ofreciendo los sesgos más bajos. La varianza mostró un comportamiento similar al sesgo. Además, se observó que los solvers "lbfgs"y "newton-cg"lograron una alta precisión, acercándose al 100% con C=1.2.

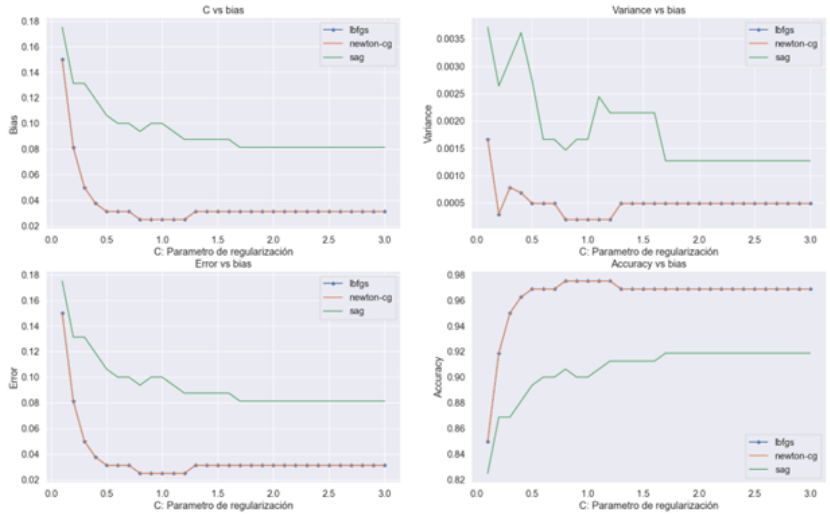

FIGURA 6: Gráficas de sesgo, varianza, error y exactitud de los modelos de regresión logística variando el hiperparámetro C y el hiperparámetro solver

Finalmente, mediante el algoritmo de búsqueda aleatoria y validación cruzada **RandomizedSearchCV** de la biblioteca **sklearn.model_selection**, se identificó la mejor combinación de hiperparámetros: **solver="newton-cg", C=10, y multi_class="ovr"**. Con este modelo, se logró una **precisión del 100%** en el conjunto de datos de evaluación, como se evidenció en la matriz de confusión (7). Las predicciones en el conjunto de entrenamiento también alcanzaron una precisión del 100%, indicando un modelo en un punto de equilibrio óptimo. La curva de aprendizaje (8) mostró que a partir del entrenamiento 50, se logró una precisión del 90%, mejorando gradualmente hasta alcanzar la mejor precisión posible.

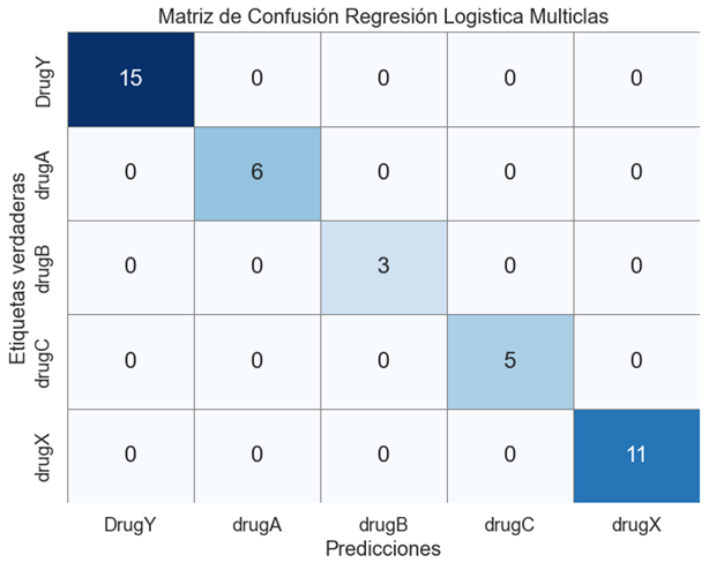

FIGURA 7: Matriz de confusión de LogisticRegression(solver="newton-cg",multi_class='ovr',C=10.0)

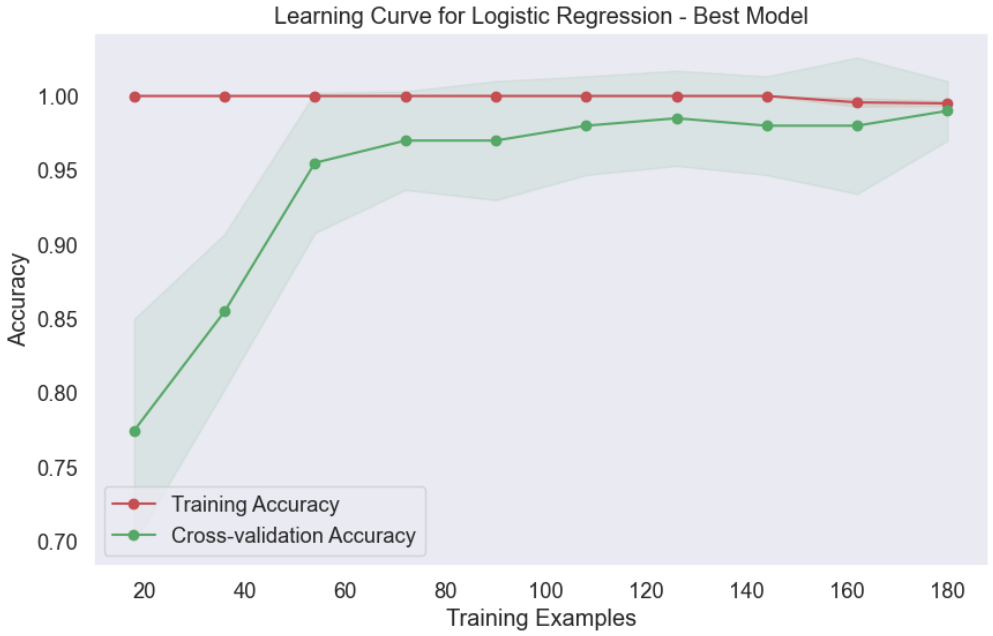

FIGURA 8: Curva de aprendizaje de LogisticRegression(solver="newton-cg",multi_class='ovr',C=10.0)

Máquinas de vectores de soporte (SVM)

Las Máquinas de Vectores de Soporte (SVM), conocidas en inglés como Support Vector Machines, representan un destacado algoritmo de aprendizaje supervisado ampliamente empleado en una diversidad de problemas de clasificación y regresión. El propósito fundamental de SVM radica en la identificación de un hiperplano óptimo capaz de separar de manera eficaz dos o más clases distintas de datos. Este "margen"se define como la máxima distancia entre el hiperplano y cualquier punto de datos contenido en su proximidad.

El término "vectores de soporte"hace referencia a un conjunto selecto de observaciones del conjunto de entrenamiento que desempeñan un papel crucial en la determinación de la ubicación del hiperplano de separación. Aunque el enfoque estándar de SVM se formula para la resolución de problemas de clasificación binaria, la mayoría de los problemas multiclase se reducen eficientemente a una serie de problemas binarios.

Las máquinas de vectores de soporte se utiliza en una variedad de aplicaciones, abarcando desde la gestión de relaciones con los clientes hasta la bioinformática y el reconocimiento de voz.

Para la implementación del modelo de regresión logística, se empleó la clase **SVC()** de la biblioteca **sklearn.svm**. Este modelo ofrece la capacidad de ajustar varios hiperparámetros, como los siguientes:

- **C**: Controla el parámetro de regularización. Un valor más alto de C permite que el modelo sea más complejo y ajuste mejor los datos de entrenamiento.
- **kernel:** Define el tipo de kernel a utilizar en el SVM y tienen un gran impacto en el rendimiento del modelo.
- **gamma:** Parámetro que afecta la forma de la función de base radial, un valor más alto de gamma puede hacer que el modelo sea más sensible a los puntos de datos cercano.
- **decision_function_shape:** Determina cómo se lleva a cabo la clasificación en problemas de clasificación multiclase.
- **class_weight:** Se utiliza para dar más peso a las clases minoritarias

En el primer modelo de SVM, se configuraron los siguientes hiperparámetros: **kernel='rbf'**, **C=1.0**, y **decision_function_shape='ovr'**. Con este modelo, se obtuvo una precisión del 62.5%. Se observa que en la matriz de confusión (9) no clasificaba correctamente, ya que tendía a clasificar únicamente los datos en dos clases, la mayoría de los datos de evaluación se clasificaron como "DrugX"y los restantes como "DrugY".

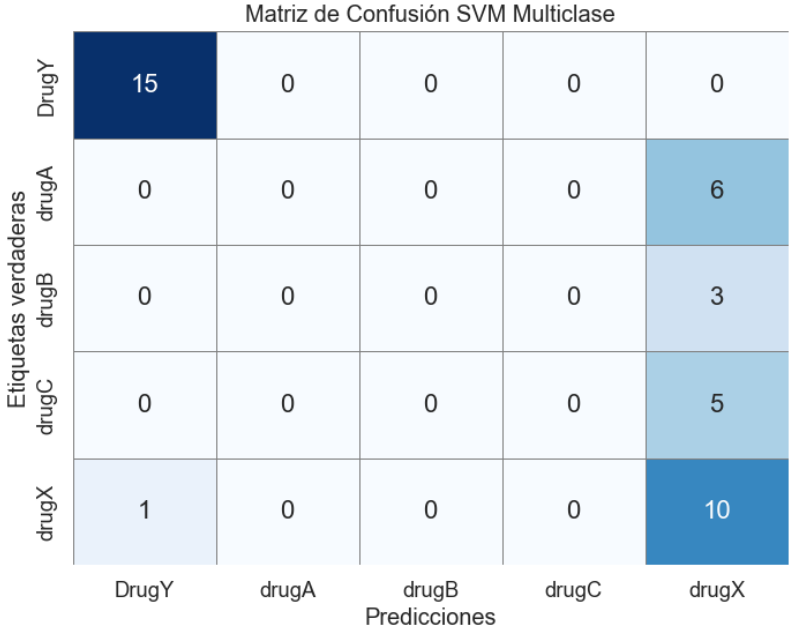

FIGURA 9: Matriz de confusión de SVC(kernel='rbf', C=1.0, decision_function_shape='ovr')

En el segundo modelo de SVM, se establecieron los hiperparámetros como sigue: **C=1.0** y **solver='poly'**. Con este modelo, se logró una precisión del 66.25%. Se calculó un sesgo de 0.34, una varianza de 0.00 y un error de 0.34. Al observar la curva de aprendizaje (10), se notó que la precisión en la validación cruzada comenzó en 0.45, valor similar al del entrenamiento que alcanzaba 0.52. Esto sugiere que el modelo estaba sub-ajustado, lo que podría atribuirse a su baja complejidad. Con el tiempo, la brecha entre la precisión del entrenamiento y la validación se redujo, sin embargo este llego a una exactitud máxima de 0.75 en el conjunto de entrenamiento y 0.65 para el conjunto de validación, sin embargo a partir del entrenamiento 60 este baja hasta alcanzar la exactitud anteriormente mencionada.

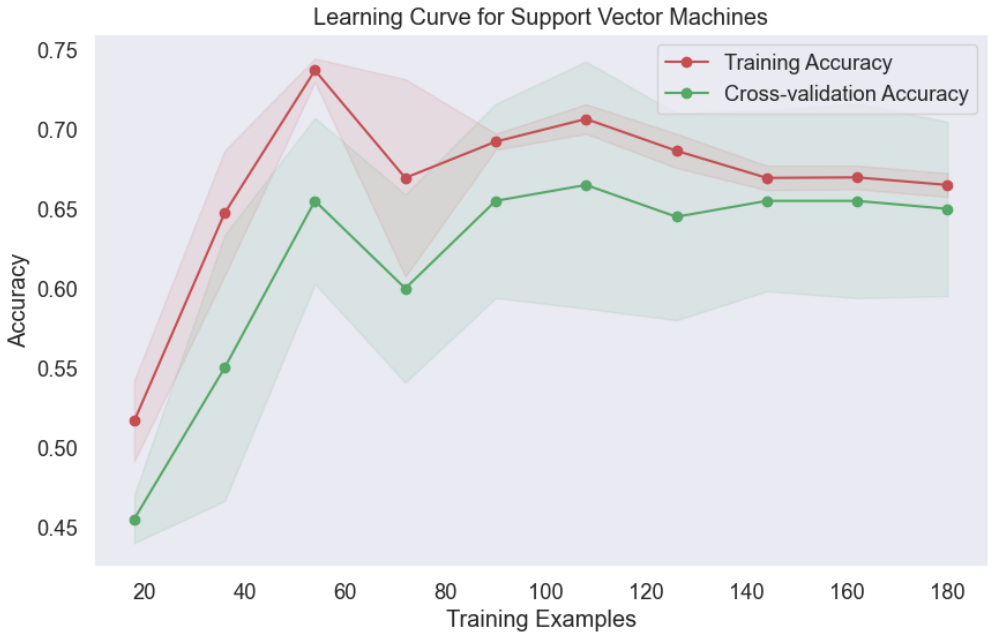

FIGURA 10: Curva de aprendizaje de SVC(kernel='poly', C=1.0, random_state=42, max_iter=10000)

Para analizar cómo la elección de hiperparámetros afecta al sesgo y la varianza, se realizó una comparación al variar el **hiperparámetro C en un rango de 0.1 a 3.0, junto con el solver utilizado ('linear', 'rbf', 'sigmoid')**. La gráfica de sesgo reveló que el valor óptimo de C para minimizar el sesgo estaba 0.5, con el solver "linear"y para los solvers 'rbf', 'sigmoid' el sesgo es mucho mas alto con valor promedio de 0.45. La varianza mostró un comportamiento similar al sesgo, donde destaca el solver "linear", que se mantiene constante con un hiperparámetro C¿0.5, mientras que para los otros solvers este varia sin ningún patrón. Además, se observó

que el solver "linear"logra una alta precisión, acercándose al 100% con C=0.5, mientras que los demás solver nos alcanzan una exactitud mayor a 75%.

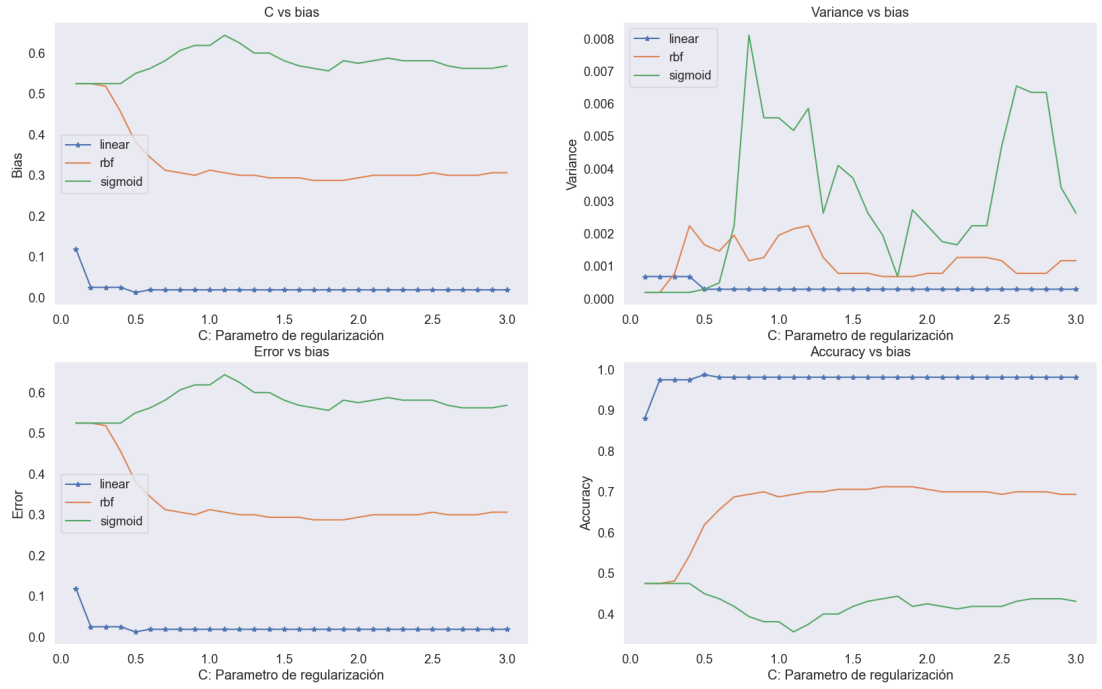

FIGURA 11: Gráficas de sesgo, varianza, error y exactitud de los modelos de SVM variando el hiperparámetro C y el hiperparámetro solver

Finalmente, mediante el algoritmo de búsqueda aleatoria y validación cruzada **RandomizedSearchCV** de la biblioteca **sklearn.model_selection**, se identificó la mejor combinación de hiperparámetros: **kernel="linear", C=1000, y gamma=0.01**. Con este modelo, se logró una **precisión del 100%** en el conjunto de datos de evaluación, como se evidenció en la matriz de confusión (12). Las predicciones en el conjunto de entrenamiento también alcanzaron una precisión del 100%, indicando un modelo en un punto de equilibrio óptimo. La curva de aprendizaje (13) mostró que a partir del entrenamiento 70, se logró una precisión del 90%, mejorando gradualmente hasta alcanzar la mejor precisión posible.

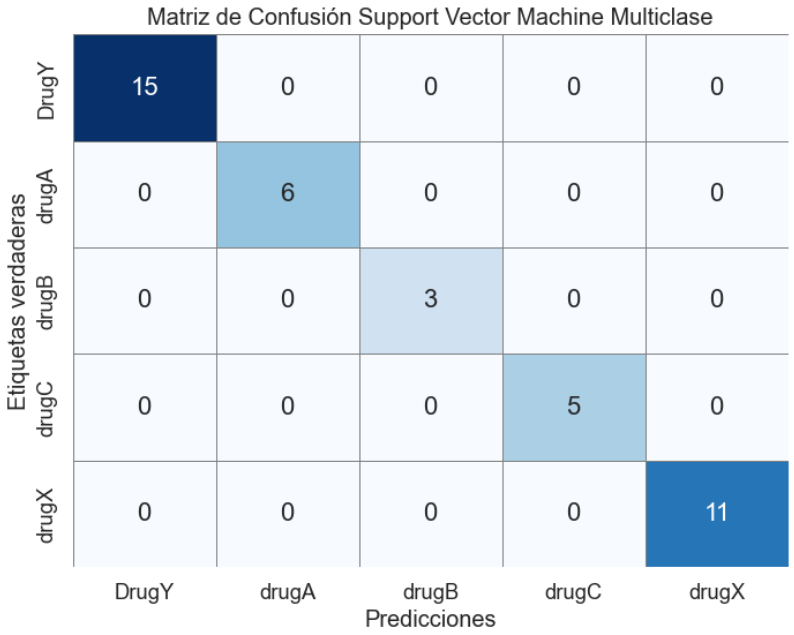

FIGURA 12: Matriz de confusión de SVC(kernel="linear",gamma=1000,C=1000)

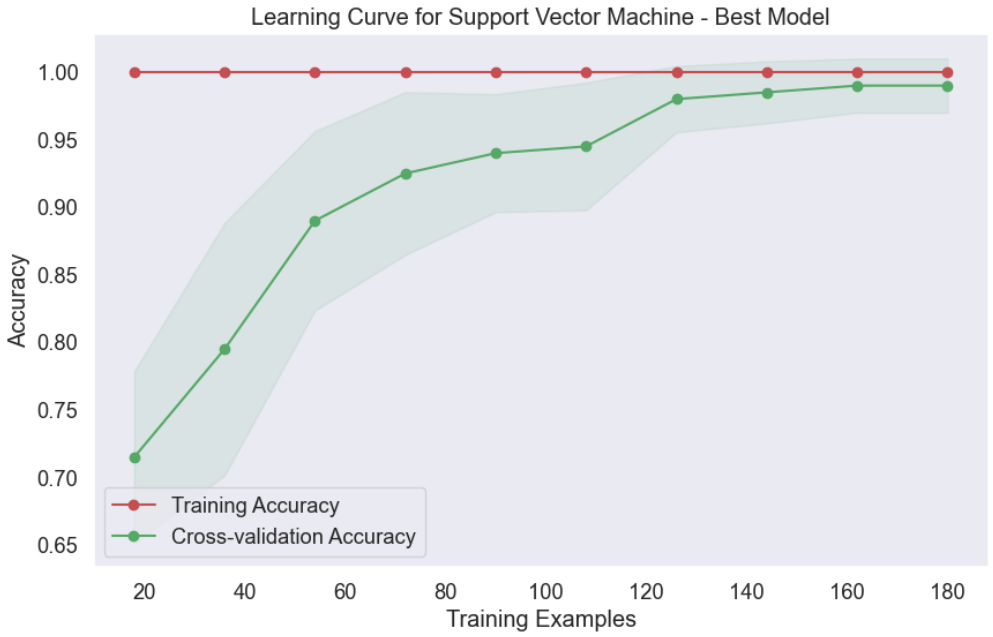

FIGURA 13: Curva de aprendizaje de SVC(kernel="linear",gamma=1000,C=1000)

Bosques aleatorios

El Bosque Aleatorio, conocido en inglés como Random Forest, es un algoritmo de aprendizaje supervisado ampliamente utilizado en clasificación y regresión. Este algoritmo, desarrollado por Leo Breiman, combina las salidas de múltiples Árboles de Decisión para producir un resultado único, lo que lo convierte en un método de conjunto.

Un Árbol de Decisión es un algoritmo de aprendizaje automático que se emplea en el modelado predictivo no paramétrico, aplicable tanto a tareas de clasificación como de regresión. Se caracteriza por su estructura jerárquica en forma de árbol, que incluye un nodo raíz, ramificaciones, nodos internos y nodos hoja.

Los Bosques Aleatorios encuentran aplicaciones en diversos campos, como la segmentación de clientes, detección de fraudes, pronóstico de ventas, autenticación de clientes y análisis de comportamiento de mercados, entre otros.

Para la implementación del modelo de regresión logística, se empleó la clase **RandomForestClassifier()** de la biblioteca **sklearn.ensemble**. Este modelo ofrece la capacidad de ajustar varios hiperparámetros, como los siguientes:

- **n_estimators:** Este parámetro controla el número de árboles de decisión en el bosque. Cuantos más árboles, más robusto será el modelo.
- **max_depth:** Controla la profundidad máxima de cada árbol en el bosque. Limitar la profundidad puede ayudar a prevenir el sobreajuste.
- **min_samples_leaf:** El número mínimo de muestras requerido para ser una hoja

- **max_features:** El número máximo de características a considerar al buscar la mejor división en un nodo.
- **bootstrap:** Indica si se deben realizar muestreos con reemplazo al construir cada árbol en el bosque.
- **class_weight:** Lista que asigna pesos a las clases

En el primer modelo de bosque aleatorio, se configuraron los siguientes hiperparámetros: **n_estimators=1** y **random_state=42**. Con este modelo, se obtuvo una precisión del 85%. Se observa que en la matriz de confusión (14) no clasificaba correctamente, ya que tendía a clasificar incorrectamente algunas de los datos en el conjunto de evaluación, no existe un patrón de calsificación sin emabargo si logra clasificar de manera correcta "DrugX"y "DrugY".

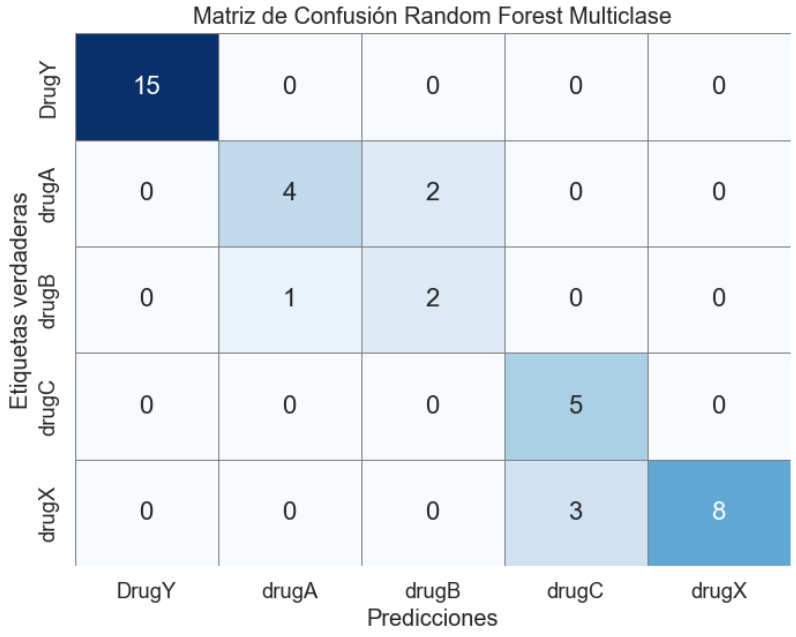

FIGURA 14: Matriz de confusión de RandomForestClassifier(n_estimators=1, random_state=42)

En el segundo modelo de bosques aleatorios, se establecieron los hiperparámetros como sigue: **n_estimators=1** y **max_depth=3**. Con este modelo, se logró una precisión del 89.37%. Se calculó un sesgo de 0.11, una varianza de 0.00 y un error de 0.11. Al observar la curva de aprendizaje (15), se notó que la precisión en la validación cruzada comenzó en 0.45, mientras que el valor de exactitud del conjunto entrenamiento que alcanzaba 0.67. Esto sugiere que el

modelo estaba sub-ajustado, lo que podría atribuirse a su baja complejidad. Con el tiempo, la brecha entre la precisión del entrenamiento y la validación se redujo, llegando a una exactitud máxima de casi el 90% en ambas al final del entrenamiento.

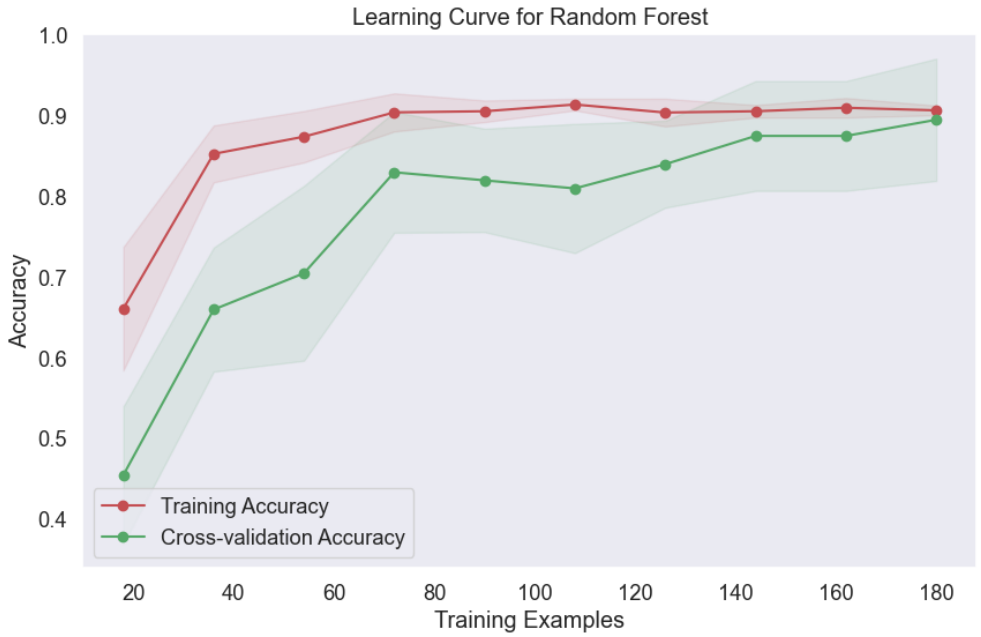

FIGURA 15: Curva de aprendizaje de RandomForestClassifier(n_estimators=1, max_depth=3,random_state=42)

Para analizar cómo la elección de hiperparámetros afecta al sesgo y la varianza, se realizó una comparación al variar el **hiperparámetro n_estimators con valores de 1,5,10 y 15, junto con max_depth en un rango de 1 a 10**. La gráfica de sesgo reveló que el valor óptimo de profundidad para minimizar el sesgo estaba 3, para la mayoría de los estimadores a excepción del que solo tiene un estimador. La varianza mostró un comportamiento aleatorio, donde destaca el modelo con estimadores igual a 10, aunque el comportamiento dependiendo de la profundidad es errático. Además, se observó que los solvers con una profundidad mayor a 4 y estimadores mayores a 5 logran una alta precisión, acercándose al 100%, mientras que el que tiene un estimador alcanza una exactitud máxima a 90%.

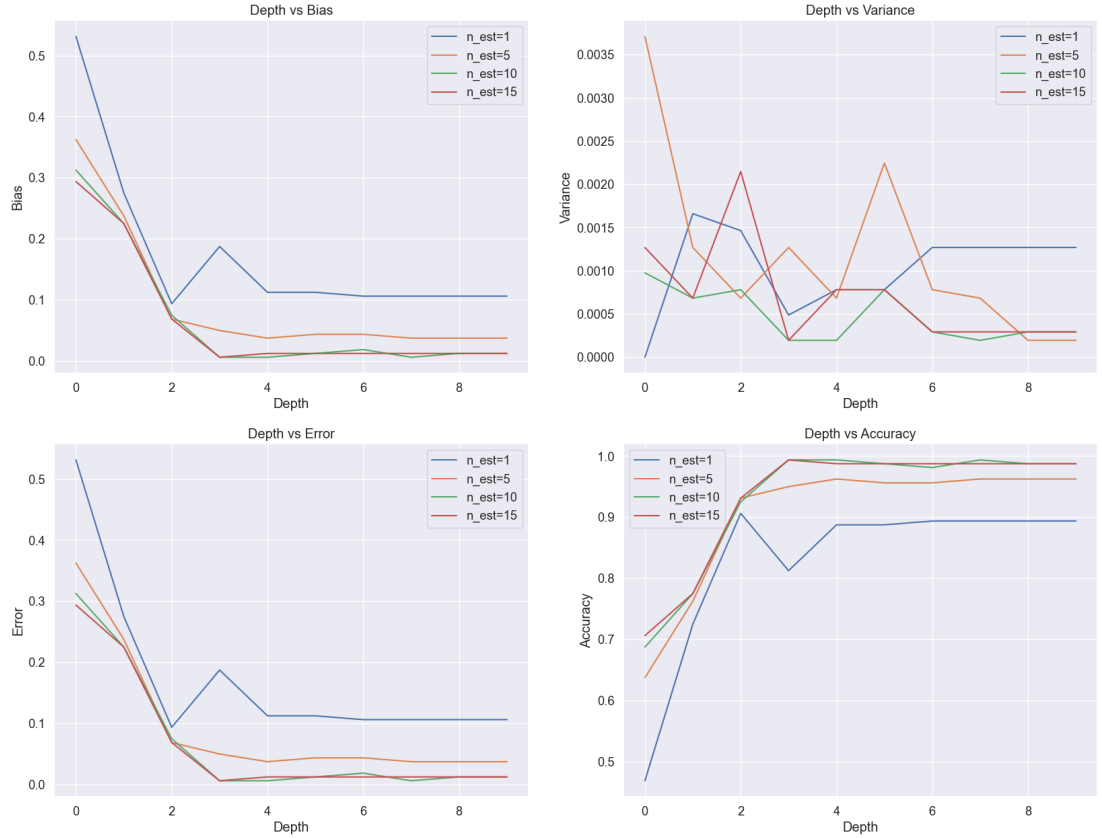

FIGURA 16: Gráficas de sesgo, varianza, error y exactitud de los modelos de Random Forest variando el hiperparámetro C y el hiperparámetro solver

Finalmente, mediante el algoritmo de búsqueda aleatoria y validación cruzada **RandomizedSearchCV** de la biblioteca **sklearn.model_selection**, se identificó la mejor combinación de hiperparámetros: **n_estimators=15, min_samples_split=6, min_samples_leaf=3, max_features="sqrt", max_depth=70, bootstrap=True, random_state=42**. Con este modelo, se logró una **precisión del 100%** en el conjunto de datos de evaluación, como se evidenció en la matriz de confusión (17). Las predicciones en el conjunto de entrenamiento también alcanzaron una precisión del 100%, indicando un modelo en un punto de equilibrio óptimo. La curva de aprendizaje (18) mostró que a partir del entrenamiento 70, se logró una precisión del 90%, mejorando gradualmente hasta alcanzar la mejor precisión posible.

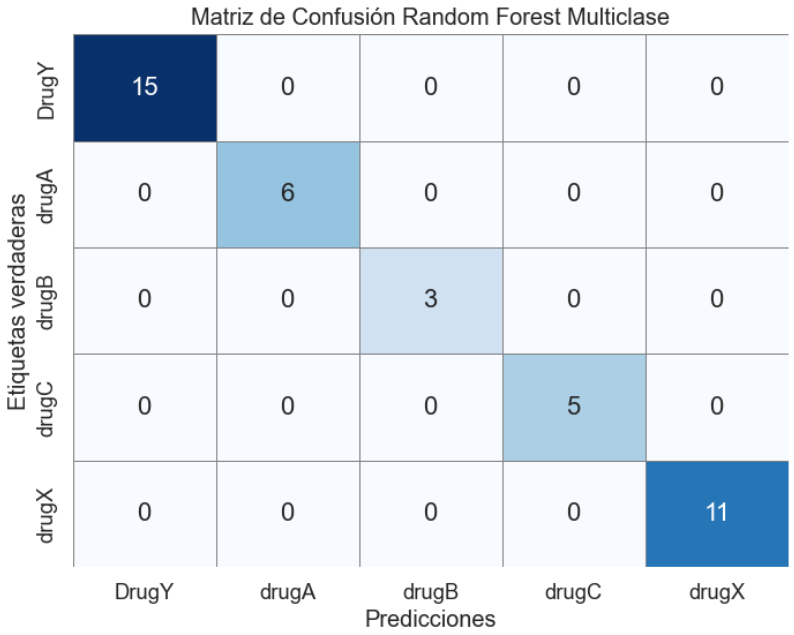

FIGURA 17: Matriz de confusión de RandomForestClassifier(random_state=42, n_estimators=15, min_samples_split=6, min_samples_leaf=3, max_features="sqrt", max_depth=70, bootstrap=True)

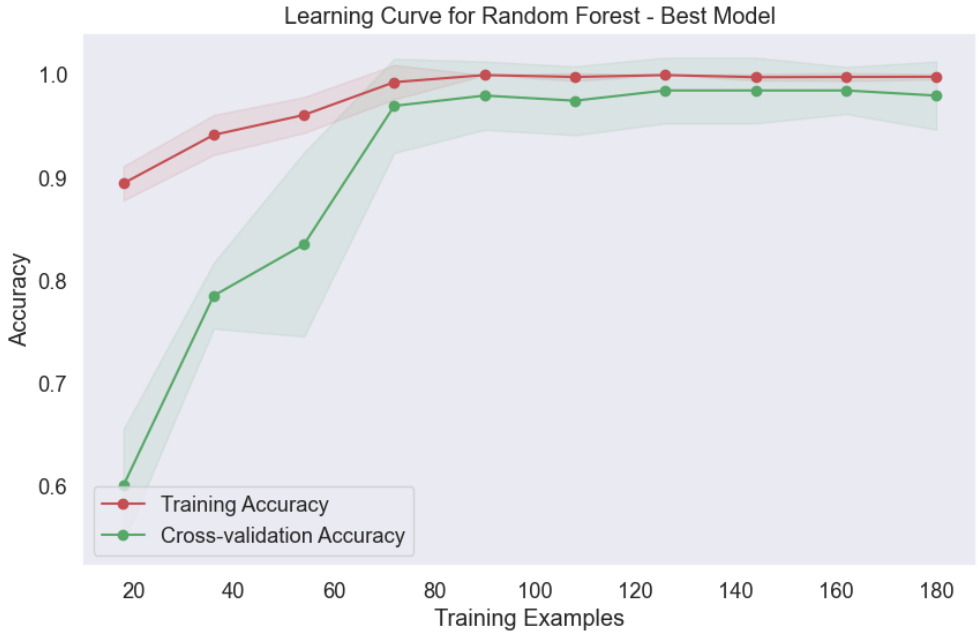

FIGURA 18: Curva de aprendizaje de RandomForestClassifier(random_state=42, n_estimators=15, min_samples_split=6, min_samples_leaf=3, max_features="sqrt", max_depth=70, bootstrap=True)

K vecinos cercanos

K-Vecinos Cercanos, más conocido en inglés como K-Nearest Neighbors (K-Vecinos más Cercanos), es un algoritmo de aprendizaje supervisado utilizado tanto para clasificar muestras con valores discretos como para predecir valores continuos.

Dado un conjunto de datos de entrenamiento con etiquetas conocidas, cuando se presenta una nueva muestra sin etiquetar, el algoritmo busca los "k"ejemplos más cercanos en función de alguna métrica de distancia, como la distancia euclidiana. Luego, asigna a la muestra no etiquetada la etiqueta más común entre sus "k"vecinos cercanos si es un problema de clasificación, o promedia los valores de las muestras vecinas si es un problema de regresión.

K-Nearest Neighbors se utiliza en diversos campos como en la clasificación de documentos, ayuda a categorizar automáticamente textos en diferentes temas. En la recomendación de productos, plataformas de comercio electrónico sugieren productos a los clientes basándose en sus historiales de compra y preferencias. En el ámbito médico, se utiliza para identificar enfermedades o condiciones médicas al comparar síntomas con casos similares en una base de datos.

Para la implementación del modelo de KNN, se empleó la clase **KNeighborsClassifier()** de la biblioteca **sklearn.neighbors**. Este modelo ofrece la capacidad de ajustar varios hiperparámetros, como los siguientes:

- **n_neighbors:** Controla el número de vecinos más cercanos que se utilizarán para tomar una decisión.
- **weights:** Determina cómo se ponderan los vecinos cercanos

- **algorithm:** Controla el algoritmo utilizado para calcular los vecinos más cercanos.
- **p:** Se utiliza junto con la métrica "minkowski"para definir el exponente de la distancia.
- **outlier_label:** El valor que se asigna a los puntos que se consideran valores atípicos o que no tienen vecinos cercanos dentro de n_neighbors.

En el primer modelo de KNN, se configuraron los siguientes hiperparámetros: **n_neighbors=3**. Con este modelo, se obtuvo una precisión del 75%. Se observa que en la matriz de confusión (19) no clasificaba correctamente, ya que tendía a clasificar incorrectamente algunas de los datos en el conjunto de evaluación, no existe un patrón de calsificación sin emabargo si logra clasificar de manera correcta "DrugC".

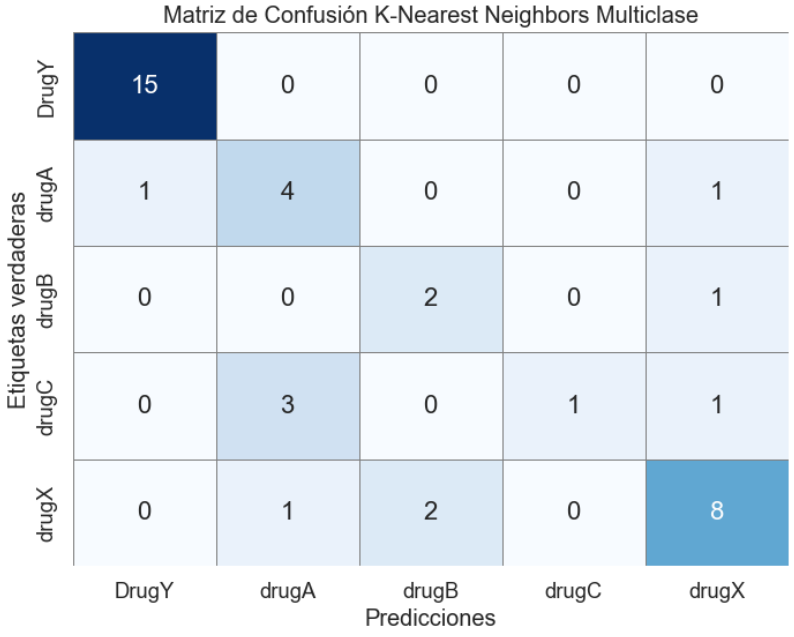

FIGURA 19: Matriz de confusión de KNeighborsClassifier($n_neighbors = 3$)

En el segundo modelo de KNN, se establecieron los hiperparámetros como sigue: **n_neighbors=2**. Con este modelo, se logró una precisión del 65%. Se calculó un sesgo de 0.35, una varianza de 0.01 y un error de 0.35. Al observar la curva de aprendizaje (20), se notó que la precisión en la validación cruzada comenzó en 0.55, mientras que el valor de exactitud del conjunto entrenamiento que alcanzaba 0.78. Esto sugiere que el modelo estaba sub-ajustado, lo que podría atribuirse a su baja complejidad. Con el tiempo, la brecha entre la precisión del entrenamiento

y la validación no se redujo, llegando a una exactitud máxima de casi el 65% en el conjunto de evaluación al final del entrenamiento.

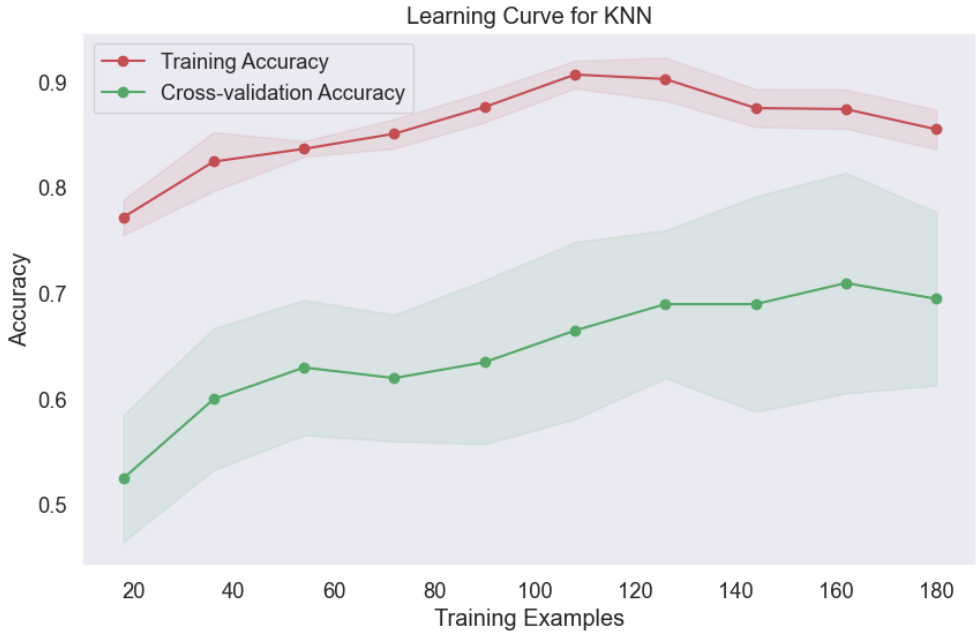

FIGURA 20: Curva de aprendizaje de KNeighborsClassifier($n_neighbors = 2$)

Para analizar cómo la elección de hiperparámetros afecta al sesgo y la varianza, se realizó una comparación al variar el **hiperparámetro n_neighbors con un rango de valores 1 a 19**. La gráfica de sesgo reveló que el valor óptimo de vecinos para minimizar el sesgo estaba 14. La varianza mostró un comportamiento aleatorio, donde se minimiza la varianza con un numero de vecinos igual a 3 y 9. Además, se observó que el modelo con vecinos igual a 14 logran la precisión mas alta, aunque no supera al 70%.

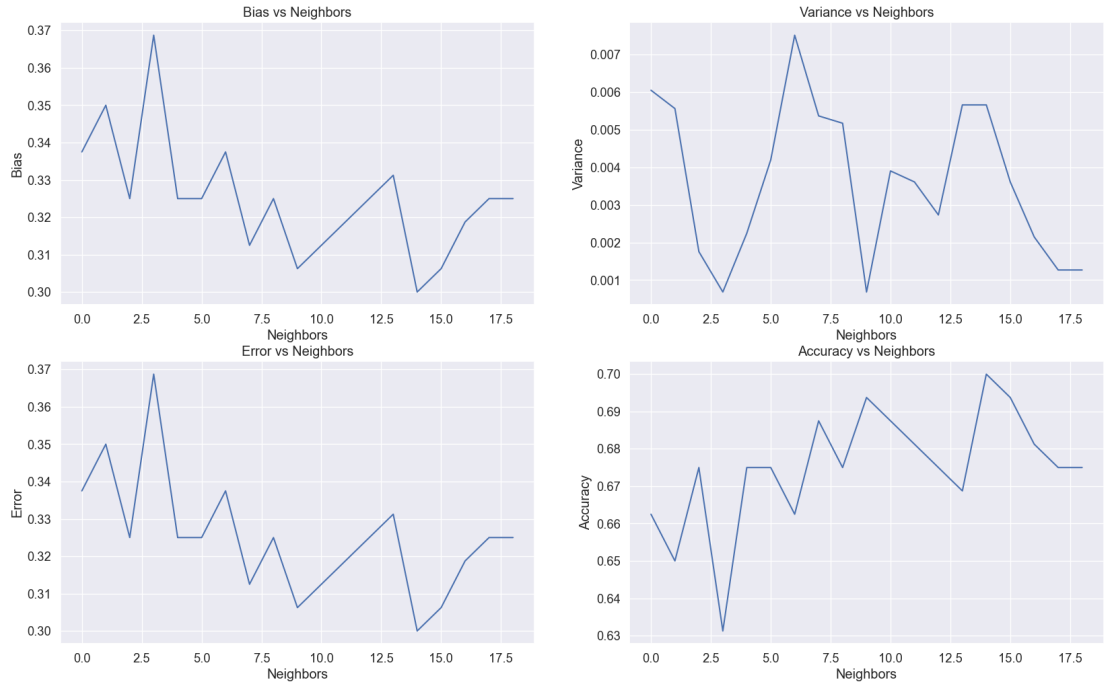

FIGURA 21: Gráficas de sesgo, varianza, error y exactitud de los modelos de KNN variando el hiperparámetro de vecinos

Finalmente, mediante el algoritmo de búsqueda aleatoria y validación cruzada **RandomizedSearchCV** de la biblioteca **sklearn.model_selection**, se identificó la mejor combinación de hiperparámetros: **weights="distance", p=1, n_neighbors=10**. Con este modelo, se logró una **precisión del 82%** en el conjunto de datos de evaluación, como se evidenció en la matriz de confusión (22) donde no se clasificaron correctamente todos los datos de evaluación. Las predicciones en el conjunto de entrenamiento si alcanzaron una precisión del 100%, indicando un modelo en un punto de sobre ajuste donde la complejidad del modelo no permite un ajuste óptico. La curva de aprendizaje (23) mostró que a la brecha de la exactitud entre el conjunto de validación y el conjunto de entrenamiento nunca se cerro, aunque se logró una precisión del 80%, de exactitud como máximo.

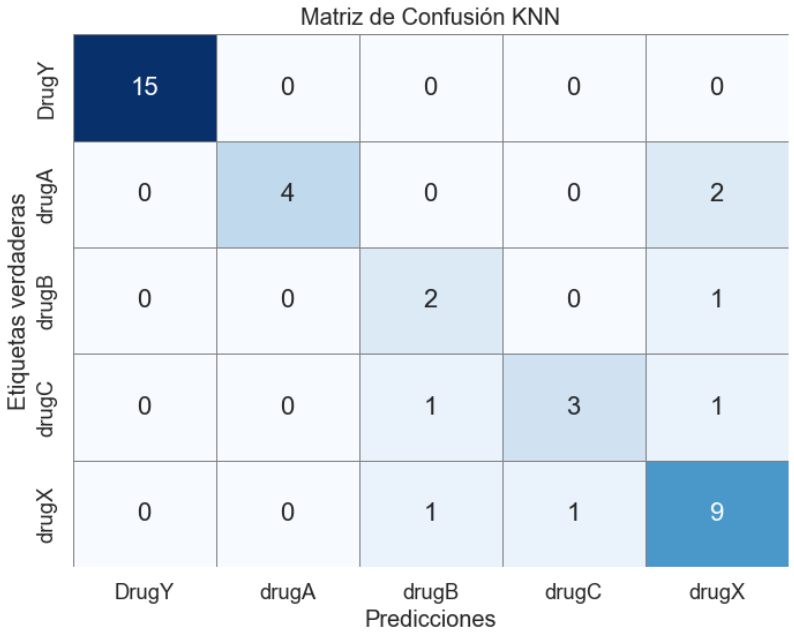

FIGURA 22: Matriz de confusión de KNeighborsClassifier(weights="distance",p=1,n_neighbors=6)

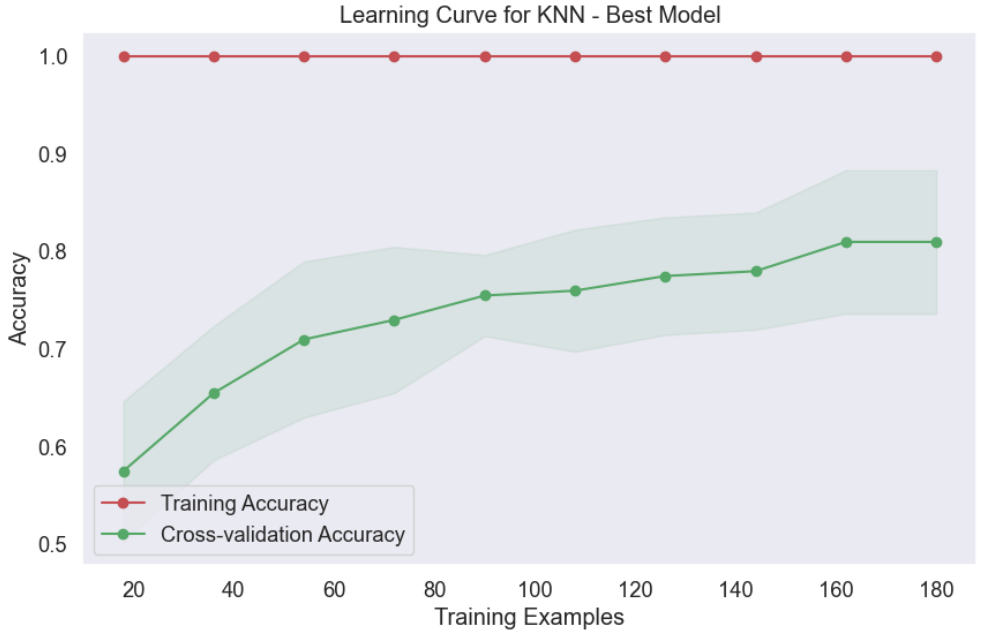

FIGURA 23: Curva de aprendizaje de KNeighborsClassifier(weights="distance",p=1,n_neighbors=6)

Conclusiones

Se puede concluir que los algoritmos de aprendizaje automático supervisado ofrecen una gama diversa de modelos capaces de abordar problemas de clasificación y regresión, que a menudo resultan ser desafiantes para el ser humano debido a la multiplicidad de parámetros y variables que pueden influir en la determinación de la variable objetivo.

Estas herramientas algorítmicas son valiosas, aunque es esencial reconocer que no todos están diseñados para todo tipo de tareas y su rendimiento varía. Por lo tanto, adquirir un profundo entendimiento del problema a resolver es fundamental para optimizar su uso y obtener los mejores resultados posibles.

A pesar de una meticulosa elección del algoritmo, surge el desafío de proporcionar las características óptimas y relevantes para lograr una máxima eficiencia. Comprender cómo funcionan los hiperparámetros resulta crucial para garantizar el éxito del modelo, así como el análisis de las métricas asociadas a los resultados, ya que brindan información valiosa sobre el funcionamiento interno del algoritmo y orientan hacia posibles ajustes necesarios para una clasificación exitosa.

En particular, en relación a los modelos evaluados en nuestro conjunto de datos, se puede afirmar que la selección adecuada de los hiperparámetros fue determinante para lograr predicciones precisas en el suministro de fármacos. La combinación de una búsqueda aleatoria de hiperparámetros junto con la técnica de validación cruzada simplificó significativamente este proceso.

De los cuatro algoritmos de clasificación utilizados, todos demostraron un excelente desempeño en términos de precisión tanto en los conjuntos de entrenamiento como en los de evaluación, a

excepción del algoritmo de K vecinos cercanos. Aunque su rendimiento no fue tan deficiente, las métricas de precisión y la matriz de confusión no lograron demostrar que las predicciones fueran correctas en su totalidad. Esto destaca la importancia de seleccionar cuidadosamente el algoritmo más apropiado para el problema en cuestión y ajustar sus hiperparámetros de manera precisa para obtener resultados satisfactorios.