Algoritmos de Aprendizaje Automático en Ciencias de la Vida: Una Explicación Detallada

1. Introducción:

El aprendizaje automático (ML), una rama de la inteligencia artificial (IA), está experimentando un crecimiento exponencial en su aplicación dentro del campo de las ciencias de la vida, transformando la biomedicina en áreas tan diversas como el diagnóstico clínico, los tratamientos de precisión y la monitorización de la salud [1]. La capacidad de la IA y el ML para analizar vastas cantidades de datos permite a los científicos realizar predicciones precisas y abordar desafíos médicos complejos [2]. El mercado global de IA en ciencias de la vida se valoró en $1.3 mil millones en 2020, con una tasa de crecimiento anual compuesta (CAGR) prevista del 41.2% entre 2021 y 2028. Este aumento se atribuye a la creciente adopción de la IA en el descubrimiento de fármacos, el diagnóstico de enfermedades y la atención médica personalizada [2]. En el ámbito de la biotecnología y la medicina, el ML se ha convertido en un componente crítico para numerosas aplicaciones, abordando el problema de la sobrecarga de datos [3].

El ML se aplica en diversas áreas de la investigación biomédica, incluyendo el desarrollo de pruebas de diagnóstico, modelos de pronóstico y sistemas de apoyo a la toma de decisiones terapéuticas [3]. La IA y el ML también se utilizan en ensayos clínicos para acelerar el proceso y mejorar la toma de decisiones. Por ejemplo, los investigadores analizan grandes conjuntos de muestras tumorales para identificar proteínas que pueden ser importantes para diferenciar entre células cancerosas y no cancerosas, utilizando algoritmos de ML para buscar patrones en las proteínas expresadas que son comunes en múltiples muestras de cáncer [3]. Las aplicaciones del ML en biotecnología y medicina son extensas, abarcando la imagenología médica (aprendizaje profundo aplicado a la resonancia magnética), la biología molecular (ML supervisado para predecir interacciones fármaco-diana), la detección de objetos (redes neuronales para diagnosticar células anormales en pruebas de Papanicolaou y analizar mamografías), el cribado de imágenes, el modelado predictivo (opciones de tratamiento basadas en datos genómicos) y los dispositivos de salud portátiles (predicción de electrocardiogramas) [3].

El ML también permite personalizar la prevención y el tratamiento mediante la identificación de las características que exponen a las personas a una enfermedad particular y la caracterización de las principales vías biológicas que causan el trastorno [4]. Una de las tendencias actuales es la administración de algoritmos de ML en la medicina de precisión para evaluar diversos datos de pacientes, como datos

clínicos, genómicos, metabolómicos, de imagen, de reclamaciones, experimentales, de nutrición y de estilo de vida [4]. El ML contribuye a la observación de pacientes enfermos, al análisis de patrones de enfermedades, al diagnóstico y a la prescripción de fármacos, a la prestación de atención centrada en el paciente, a la reducción de errores clínicos, a la puntuación predictiva, a la toma de decisiones terapéuticas, a la detección de sepsis y a las emergencias de alto riesgo en pacientes [4]. Los primeros sistemas de IA en ciencias de la vida, como Dendral (1965) y MYCIN (1976), demostraron el potencial de estas tecnologías [5]. El ML y el aprendizaje profundo se utilizan en genómica para comprender patrones ocultos en grandes conjuntos de datos, como la identificación del tipo primario de cáncer a partir de biopsias líquidas y la detección de trastornos genéticos mediante el análisis facial [5]. La IA desempeña un papel fundamental en el proceso de descubrimiento y desarrollo de fármacos, identificando patrones en conjuntos de datos, confirmando dianas farmacológicas y refinando el diseño de moléculas de fármacos [5]. La capacidad de la IA para procesar y analizar grandes cantidades de datos a una velocidad mucho mayor que los profesionales de la salud e investigadores acelera el proceso de investigación [5]. La IA también puede identificar signos tempranos de enfermedades y predecir los fármacos o terapias más apropiados mediante el análisis de datos de pacientes [5]. Los primeros usos del ML mostraron resultados prometedores en el diagnóstico de cáncer de mama, el descubrimiento de nuevos antibióticos, la predicción de la diabetes gestacional y la identificación de agrupaciones de datos [6]. El ML se aplica cada vez más en medicina para ayudar a pacientes y médicos, aprendiendo de conjuntos de datos cada vez mayores [7]. Los algoritmos de aprendizaje profundo se utilizan para la clasificación de imágenes celulares, el análisis del genoma, el descubrimiento de fármacos y la vinculación de datos de imagen y genoma con registros médicos electrónicos [8]. Las aplicaciones que utilizan ML pueden ayudar a generar confianza en el sistema y facilitar una comprensión más profunda del mecanismo biológico subyacente de la enfermedad al explicar las predicciones [1]. El ML permite modelar los datos de forma extremadamente precisa sin utilizar supuestos sólidos sobre el sistema modelado, proporcionando tanto soluciones de ingeniería como un punto de referencia esencial en biología [9]. En genómica, el ML se utiliza para el análisis de secuencias, la genómica reguladora (aprendizaje de motivos de unión de factores de transcripción) y el análisis de datos unicelulares (reducción de dimensionalidad, corrección de lotes, imputación) [10].

Este informe se centrará en cinco algoritmos clave de aprendizaje automático supervisado que se utilizan ampliamente en ciencias de la vida: K-Nearest Neighbors (KNN), árboles de decisión, máquinas de soporte vectorial (SVM), bosques aleatorios y máquinas de aumento de gradiente (Gradient Boosting Machines, GBM). Cada

algoritmo se analizará en detalle, cubriendo sus fundamentos teóricos, el papel del análisis exploratorio de datos (EDA) en su aplicación y ejemplos prácticos de implementación en Python con ejemplos de ciencias de la vida. Estos algoritmos han sido seleccionados por su versatilidad y utilidad probada en la resolución de diversos problemas en ciencias de la vida, incluyendo la clasificación (por ejemplo, diagnóstico de enfermedades, identificación de biomarcadores) y la regresión (por ejemplo, predicción de la eficacia de los fármacos, pronóstico de pacientes).

El análisis exploratorio de datos (EDA) es un paso inicial fundamental en cualquier proyecto de ciencia de datos. Implica la observación y visualización de los datos para comprender sus características principales, encontrar patrones y descubrir cómo se conectan las diferentes partes de los datos [11]. El EDA ayuda a identificar datos inusuales o valores atípicos y generalmente se realiza antes de comenzar un análisis estadístico más detallado o la construcción de modelos [11]. Ayuda a comprender la estructura del conjunto de datos (número de características, tipos de datos, distribución), lo cual es esencial para elegir los métodos de análisis adecuados [11]. El EDA puede revelar patrones y relaciones ocultas entre los puntos de datos, lo que informa la construcción de modelos [11]. La identificación de errores o valores atípicos a través del EDA puede evitar que afecten negativamente el rendimiento del modelo [11]. Los conocimientos obtenidos del EDA ayudan a seleccionar las características más importantes y a prepararlas para mejorar la precisión del modelo [11]. El EDA implica el análisis univariante (estudio de una variable), el análisis bivariante (relación entre dos variables) y el análisis multivariante (relaciones entre múltiples variables) [11]. Los pasos del EDA incluyen la comprensión del problema y los datos, la importación e inspección de los datos, el manejo de los datos faltantes, la exploración de las características de los datos, la realización de la transformación de los datos, la visualización de las relaciones de los datos y el manejo de los valores atípicos [11]. Para KNN, el EDA es esencial para comprender las relaciones entre las características y la distribución de los datos, como se ilustra con diagramas de pares e histogramas [12]. La importancia del EDA radica en la necesidad de comprender y preparar los datos de manera adecuada para obtener resultados fiables y significativos con el aprendizaje automático en ciencias de la vida. Dada la complejidad y heterogeneidad de los datos biológicos, un EDA exhaustivo es primordial.

2. K-Nearest Neighbors (KNN) en Ciencias de la Vida:

El análisis exploratorio de datos (EDA) juega un papel crucial en la aplicación efectiva del algoritmo K-Nearest Neighbors (KNN) en las ciencias de la vida. Las técnicas de visualización del EDA, como los diagramas de dispersión, pueden ayudar a identificar la relevancia de las características al mostrar la relación entre diferentes variables y la

variable objetivo [11]. Por ejemplo, en la clasificación de enfermedades, la visualización de las características de los pacientes en relación con su estado de enfermedad puede revelar qué características son más informativas para el algoritmo KNN. Además, KNN se basa en cálculos de distancia, por lo que el EDA es fundamental para comprender la escala de las diferentes características y la presencia de valores atípicos [11]. Las características con escalas más grandes pueden influir desproporcionadamente en los cálculos de distancia, mientras que los valores atípicos pueden sesgar la vecindad de un punto de datos. Por lo tanto, el EDA informa sobre la necesidad de escalado de características (por ejemplo, utilizando StandardScaler) y técnicas de manejo de valores atípicos antes de aplicar KNN. La sensibilidad de KNN al escalado de características y a los valores atípicos, revelada a través del EDA, subraya la importancia del preprocesamiento de datos para lograr un rendimiento óptimo. Sin un escalado adecuado, las características con magnitudes mayores podrían dominar los cálculos de distancia, lo que llevaría a resultados sesgados. Los valores atípicos también podrían tergiversar la verdadera vecindad de un punto de datos. El EDA también puede guiar la selección del valor óptimo de 'k' (número de vecinos). Esto puede implicar la visualización del rendimiento del modelo (por ejemplo, precisión, tasa de error) para diferentes valores de 'k' [11]. Técnicas como la validación cruzada, a menudo visualizadas a través de gráficos, pueden ayudar a determinar el 'k' óptimo que equilibra el sesgo y la varianza para el conjunto de datos dado. El método del codo, donde se grafica la tasa de error contra 'k', es otro enfoque impulsado por el EDA [15].

El algoritmo KNN clasifica o predice un nuevo punto de datos en función de la clase mayoritaria o el valor promedio de sus 'k' vecinos más cercanos en el espacio de características [14]. La noción de "más cercano" se define mediante una métrica de distancia. Las métricas comunes incluyen la distancia euclidiana, que es la distancia en línea recta entre dos puntos [15]. La fórmula es $\Sigma i=1n(xi-yi)2$. La distancia de Manhattan es la suma de las diferencias absolutas de sus valores de coordenadas [15]. Su fórmula es $\Sigma i=1n|xi-yi|$. La distancia de Minkowski es una generalización de las distancias euclidiana y de Manhattan, con un parámetro 'p' (p=2 para euclidiana, p=1 para Manhattan) [15]. La fórmula es $(\Sigma i=1n|xi-yi|p)1/p$. La distancia de Hamming es el número de posiciones en las que los símbolos correspondientes son diferentes (utilizada para datos categóricos o vectores binarios) [21]. La similitud del coseno mide el coseno del ángulo entre dos vectores no nulos, útil para datos de alta dimensión como la expresión génica [21]. El valor 'k' en KNN representa el número de vecinos más cercanos a considerar [14]. Para la clasificación, el nuevo punto de datos se asigna a la clase que tiene la mayoría entre sus 'k' vecinos (votación mayoritaria) [13]. Para la regresión, el valor predicho suele ser el promedio o la mediana de los valores objetivo

de los 'k' vecinos [14]. La elección de 'k' es crucial; un 'k' pequeño puede llevar al sobreajuste (sensible al ruido), mientras que un 'k' grande puede llevar al subajuste (suavizado excesivo) [12].

KNN ofrece varias ventajas en el contexto de los datos de ciencias de la vida. Es simple de entender e implementar [14] y se puede utilizar tanto para tareas de clasificación como de regresión [14]. Al ser un algoritmo no paramétrico, no hace suposiciones sobre la distribución subyacente de los datos [18] y puede capturar relaciones complejas y no lineales. Sin embargo, KNN también tiene limitaciones. Es computacionalmente costoso para grandes conjuntos de datos, ya que necesita calcular las distancias a todos los puntos de entrenamiento para cada predicción (aprendizaje perezoso) [17]. Es sensible a las características irrelevantes y a la maldición de la dimensionalidad (el rendimiento se degrada en espacios de alta dimensión) [17]. El rendimiento depende en gran medida de la elección de 'k' y de la métrica de distancia [14] y puede ser propenso al sobreajuste con valores pequeños de 'k' [12]. La simplicidad y la naturaleza no paramétrica de KNN lo convierten en un buen punto de partida para muchos problemas de ciencias de la vida, pero su costo computacional y su sensibilidad a las características de los datos requieren una consideración y un preprocesamiento cuidadosos. La alta dimensionalidad que a menudo se encuentra en los datos genómicos o proteómicos puede desafiar particularmente la efectividad de KNN.

A continuación, se presenta un ejemplo de implementación de KNN en Python utilizando la biblioteca scikit-learn para una aplicación de ciencias de la vida, específicamente la clasificación de enfermedades basada en las características del paciente. El primer paso es la preparación de los datos, que implica cargar un conjunto de datos de ciencias de la vida adecuado para la clasificación (por ejemplo, un conjunto de datos de características de pacientes y su estado de enfermedad). Se pueden encontrar ejemplos de conjuntos de datos relevantes en repositorios como Healthdata.gov, Data.gov o el repositorio de aprendizaje automático de la UCI [28]. Por ejemplo, se podría utilizar el conjunto de datos "Diabetes" disponible en scikit-learn o en repositorios en línea [24]. Una vez cargados los datos, se realiza un análisis exploratorio de datos (EDA) para comprender la distribución de los datos, identificar los valores faltantes, detectar los valores atípicos y visualizar las relaciones entre las características y la variable objetivo (estado de la enfermedad) [11]. Se pueden utilizar técnicas como histogramas, diagramas de dispersión y diagramas de caja. Después del EDA, se procede al preprocesamiento de los datos, que incluye dividir los datos en conjuntos de entrenamiento y prueba (por ejemplo, 80% entrenamiento, 20% prueba) [12] y escalar las características utilizando StandardScaler de scikit-learn para

garantizar que todas las características contribuyan por igual a los cálculos de distancia [12]. También se deben manejar los valores faltantes o atípicos identificados durante el EDA. A continuación, se entrena el modelo KNN. Se importa la clase KNeighborsClassifier de sklearn.neighbors, se instancia el clasificador con un valor elegido de 'k' (por ejemplo, knn = KNeighborsClassifier(n_neighbors=5)) y se entrena el modelo utilizando los datos de entrenamiento: knn.fit(X_train_scaled, y_train) [13]. Una vez entrenado el modelo, se realizan predicciones en el conjunto de prueba: y_pred = knn.predict(X_test_scaled). El rendimiento del modelo se evalúa utilizando métricas como la precisión, la exactitud, la sensibilidad y la puntuación F1 de sklearn.metrics [13]. Una matriz de confusión también puede proporcionar información sobre los tipos de errores que comete el modelo [12]. Finalmente, se pueden utilizar técnicas como la validación cruzada con GridSearchCV o RandomizedSearchCV de sklearn.model_selection para encontrar el valor óptimo de 'k' que maximice el rendimiento del modelo en el conjunto de validación [12].

Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos (reemplazar con la carga de datos real)
# Ejemplo utilizando un conjunto de datos de ejemplo
data = {'feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'feature2': [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
        'disease_status': }
df = pd.DataFrame(data)
X = df[['feature1', 'feature2']]
y = df['disease_status']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Escalar las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Inicializar y entrenar el clasificador KNN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)

# Hacer predicciones
y_pred = knn.predict(X_test_scaled)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión: {accuracy}")
print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de Confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()
```

El fragmento de código Python proporcionado demuestra los pasos prácticos involucrados en la aplicación de KNN a un problema de clasificación de enfermedades en ciencias de la vida, enfatizando la importancia de la división de datos, el escalado de características, el entrenamiento del modelo y la evaluación utilizando las herramientas estándar de scikit-learn. Esto permite a los investigadores de ciencias de la vida implementar y experimentar fácilmente con KNN en sus propios conjuntos de datos.

3. Árboles de Decisión en Ciencias de la Vida:

El análisis exploratorio de datos (EDA) es fundamental para la aplicación de árboles de decisión en ciencias de la vida. El EDA ayuda a visualizar la distribución de las características individuales y su relación con la variable objetivo, lo que informa sobre los posibles puntos de división para los árboles de decisión [11]. Por ejemplo, la visualización de la distribución del nivel de un biomarcador específico en pacientes

con y sin una enfermedad puede sugerir un umbral potencial para dividir los datos. El EDA también revela los tipos de características presentes en el conjunto de datos (categóricas, numéricas). Los árboles de decisión pueden manejar ambos tipos, pero el EDA ayuda a comprender la naturaleza de las variables categóricas (número de categorías únicas) y las variables numéricas (continuas o discretas) [11]. Esta información es crucial para posibles pasos de ingeniería o codificación de características si es necesario. Si bien los árboles de decisión pueden capturar inherentemente relaciones e interacciones no lineales, el EDA a través de técnicas como diagramas de pares o matrices de correlación puede proporcionar indicios iniciales sobre tales relaciones, lo que podría guiar la complejidad del árbol (por ejemplo, la profundidad máxima) [11].

Los árboles de decisión funcionan particionando recursivamente los datos en función de los valores de las características. La elección de qué característica utilizar para la división en cada nodo se determina mediante un criterio de división que tiene como objetivo maximizar la separación de clases (para clasificación) o reducir la varianza (para regresión) [4]. Los criterios comunes incluyen la entropía y la ganancia de información (para clasificación), donde la entropía mide la impureza de un nodo y la ganancia de información mide la reducción de la entropía después de la división. La característica con la mayor ganancia de información se elige para la división. La impureza de Gini es otra medida de la impureza del nodo para la clasificación. El error cuadrático medio se utiliza para la regresión, midiendo la diferencia cuadrática promedio entre los valores predichos y reales. El árbol se construye dividiendo recursivamente los datos hasta que se cumple un criterio de detención (por ejemplo, profundidad máxima, número mínimo de muestras por hoja). Esto puede llevar al sobreajuste, donde el árbol se vuelve demasiado complejo y aprende el ruido en los datos de entrenamiento. Las técnicas de poda se utilizan para reducir el tamaño del árbol y mejorar su capacidad de generalización. Los árboles de decisión son altamente interpretables porque el proceso de toma de decisiones se puede visualizar fácilmente como una serie de reglas if-else. La ruta desde la raíz hasta un nodo hoja representa una regla de clasificación. Además, los árboles de decisión pueden proporcionar una medida de la importancia de las características, lo que indica qué características contribuyen más a la predicción.

A continuación, se presenta un ejemplo de implementación de árboles de decisión en Python utilizando scikit-learn para predecir factores de riesgo de enfermedades. El primer paso es obtener un conjunto de datos de ciencias de la vida donde el objetivo es predecir el riesgo de una enfermedad en función de diversos factores (por ejemplo, marcadores genéticos, estilo de vida, historial médico) [4]. Se podrían utilizar

conjuntos de datos relacionados con el riesgo de enfermedades cardiovasculares, la predisposición al cáncer o el riesgo de diabetes. Se lleva a cabo un análisis exploratorio de datos (EDA) para comprender la distribución de los factores de riesgo y la prevalencia de la enfermedad. Se visualiza la relación entre los factores de riesgo individuales y el resultado de la enfermedad utilizando técnicas como gráficos de barras, histogramas y diagramas de caja. Los datos se dividen en conjuntos de entrenamiento y prueba. Se manejan los valores faltantes o las características categóricas. Para los árboles de decisión, las características categóricas a menudo se pueden utilizar directamente, pero las características numéricas podrían beneficiarse del escalado o la discretización según el conjunto de datos específico y los objetivos del análisis. Se importa la clase DecisionTreeClassifier (para clasificación) o DecisionTreeRegressor (para regresión) de sklearn.tree. Se instancia el modelo con los hiperparámetros deseados (por ejemplo, max_depth, min_samples_leaf). Se entrena el modelo utilizando los datos de entrenamiento: dtree.fit(X_train, y_train). Se realizan predicciones en el conjunto de prueba. El rendimiento del modelo se evalúa utilizando las métricas apropiadas (precisión, exactitud, sensibilidad, puntuación F1 para clasificación; R-cuadrado, error cuadrático medio para regresión). El árbol de decisión se visualiza utilizando plot_tree de sklearn.tree. Se accede a las puntuaciones de importancia de las características utilizando el atributo dtree.feature_importances_. Estas puntuaciones se visualizan para identificar los factores de riesgo más influyentes.

Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos (reemplazar con la carga de datos real)
# Ejemplo utilizando un conjunto de datos de ejemplo
data = {'age': ,
        'smoker': ,
        'high_cholesterol': ,
        'disease_risk': }
```

```
df = pd.DataFrame(data)
X = df[['age', 'smoker', 'high_cholesterol']]
y = df['disease_risk']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Inicializar y entrenar el clasificador de árbol de decisión
dtree = DecisionTreeClassifier(max_depth=3)
dtree.fit(X_train, y_train)

# Hacer predicciones
y_pred = dtree.predict(X_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión: {accuracy}")
print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de Confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()

# Visualizar el árbol de decisión
plt.figure(figsize=(12, 8))
plot_tree(dtree, feature_names=X.columns, class_names=, filled=True)
plt.show()

# Importancia de las características
importance = dtree.feature_importances_
print("\nImportancia de las Características:")
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[i]:.4f}")
```

El código Python demuestra cómo construir y visualizar un árbol de decisión para

predecir el riesgo de enfermedad, destacando la interpretabilidad del modelo a través de la función plot_tree y la capacidad de evaluar la importancia de los diferentes factores de riesgo. Esta interpretabilidad es una ventaja significativa de los árboles de decisión en ciencias de la vida, donde la comprensión de los factores subyacentes a menudo es tan importante como realizar predicciones precisas.

4. Máquinas de Soporte Vectorial (SVM) en Ciencias de la Vida:

El análisis exploratorio de datos (EDA) desempeña un papel fundamental en la preparación de datos para las Máquinas de Soporte Vectorial (SVM). El EDA puede ayudar a visualizar los datos en dimensiones inferiores (por ejemplo, utilizando técnicas de reducción de dimensionalidad como PCA después del EDA) para obtener una idea inicial de si las clases son linealmente separables. SVM funciona mejor cuando las clases están bien separadas. El EDA también puede revelar la dimensionalidad de los datos, lo que puede afectar el rendimiento de SVM y la elección del kernel. SVM tiene como objetivo encontrar el hiperplano que maximiza el margen entre diferentes clases. Las características con escalas más grandes pueden dominar los cálculos de distancia y afectar el hiperplano óptimo. El EDA ayuda a identificar la necesidad de escalado de características (por ejemplo, utilizando StandardScaler) para garantizar que todas las características contribuyan por igual al cálculo del margen [4]. Los valores atípicos pueden afectar significativamente la posición del hiperplano óptimo en SVM. El EDA ayuda a detectar valores atípicos que podrían necesitar ser abordados antes de entrenar el modelo SVM.

El objetivo de SVM para la clasificación es encontrar el hiperplano óptimo que separa los puntos de datos de diferentes clases con el mayor margen posible (la distancia entre el hiperplano y el punto de datos más cercano de cualquier clase). Para datos linealmente separables, este es un proceso sencillo. Los vectores de soporte son los puntos de datos que se encuentran más cerca del hiperplano e influyen en su posición y orientación. Solo estos puntos son cruciales para definir el límite de decisión; otros puntos de entrenamiento se pueden eliminar sin afectar el modelo. Cuando los datos no son linealmente separables, SVM utiliza funciones kernel para mapear los datos a un espacio de mayor dimensión donde se vuelven linealmente separables. Las funciones kernel comunes incluyen el kernel lineal, adecuado para datos linealmente separables; el kernel polinómico, que puede capturar relaciones no lineales considerando combinaciones polinómicas de las características originales; el kernel de función de base radial (RBF), un kernel popular que puede manejar límites de decisión complejos y no lineales; y el kernel sigmoide, similar a la función de activación de una red neuronal.

A continuación, se presenta un ejemplo de implementación de SVM en Python utilizando scikit-learn para la clasificación de muestras biológicas. El primer paso es obtener un conjunto de datos de ciencias de la vida donde la tarea es clasificar muestras biológicas en diferentes categorías en función de sus características (por ejemplo, clasificación de tipos de células en función de perfiles de expresión génica, clasificación de muestras de tejido como cancerosas o benignas en función de datos de imágenes) [4]. Se realiza un análisis exploratorio de datos (EDA) para visualizar la distribución de las características y la separación entre las diferentes clases. Se considera el uso de técnicas de reducción de dimensionalidad como PCA para la visualización en dimensiones inferiores. Los datos se dividen en conjuntos de entrenamiento y prueba. Las características se escalan utilizando StandardScaler. Se manejan los valores faltantes o atípicos. Se importa la clase SVC (Support Vector Classifier) de sklearn.svm. Se instancia el modelo con un kernel elegido (por ejemplo, svm_model = SVC(kernel='rbf')). Se entrena el modelo utilizando los datos de entrenamiento: svm_model.fit(X_train_scaled, y_train). Se realizan predicciones en el conjunto de prueba. El rendimiento del modelo se evalúa utilizando métricas como la precisión, la exactitud, la sensibilidad y la puntuación F1. Se utiliza la validación cruzada con GridSearchCV o RandomizedSearchCV para encontrar los hiperparámetros óptimos para el kernel elegido (por ejemplo, el parámetro de regularización 'C' y el coeficiente del kernel 'gamma' para el kernel RBF).

Python

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos (reemplazar con la carga de datos real)
# Ejemplo utilizando un conjunto de datos de ejemplo
data = {'feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'feature2': [11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
        'sample_type': }
df = pd.DataFrame(data)
```

```
X = df[['feature1', 'feature2']]
y = df['sample_type']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Escalar las características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Inicializar el clasificador SVM
svm_model = SVC(kernel='rbf', random_state=42)

# Definir la cuadrícula de parámetros para GridSearchCV
param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale', 0.1, 0.01]}

# Realizar GridSearchCV para el ajuste de hiperparámetros
grid_search = GridSearchCV(svm_model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Obtener los mejores parámetros y el mejor modelo
best_params = grid_search.best_params_
best_svm_model = grid_search.best_estimator_

print(f"Mejores Parámetros: {best_params}")

# Hacer predicciones utilizando el mejor modelo
y_pred = best_svm_model.predict(X_test_scaled)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión: {accuracy}")
print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de Confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
```

```
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()
```

El código Python ilustra cómo implementar SVM para la clasificación de muestras biológicas, enfatizando el uso del kernel RBF para datos no lineales y el paso crucial del ajuste de hiperparámetros utilizando GridSearchCV para optimizar el rendimiento del modelo. Esto destaca la flexibilidad de SVM para manejar conjuntos de datos biológicos complejos, pero también la necesidad de una selección cuidadosa de parámetros.

5. Bosques Aleatorios en Ciencias de la Vida:

El análisis exploratorio de datos (EDA) puede proporcionar información inicial sobre qué características podrían ser importantes para la predicción. Los Bosques Aleatorios inherentemente proporcionan una medida de la importancia de las características, pero el EDA puede ayudar a validar estos hallazgos o revelar posibles interacciones entre características que podrían necesitar una investigación más profunda con modelos más complejos [11]. El EDA también podría implicar la comparación del rendimiento de los árboles de decisión individuales (que forman la base de un bosque aleatorio) con el rendimiento del conjunto para ilustrar los beneficios de combinar múltiples modelos [11]. La visualización de los límites de decisión de los árboles individuales frente al bosque aleatorio también puede ser informativa.

El Bosque Aleatorio es un método de aprendizaje conjunto que combina múltiples árboles de decisión para realizar predicciones. Utiliza una técnica llamada bagging (bootstrap aggregating), donde se crean múltiples subconjuntos de los datos de entrenamiento mediante muestreo con reemplazo. Se entrena un árbol de decisión en cada uno de estos subconjuntos. Además del bagging, los Bosques Aleatorios introducen aleatoriedad en el proceso de selección de características. Al construir cada árbol, solo se considera un subconjunto aleatorio de características para la división en cada nodo. Esto ayuda a descorrelacionar los árboles del bosque y reduce el riesgo de sobreajuste. La combinación de bagging y selección aleatoria de características hace que los Bosques Aleatorios sean robustos al sobreajuste y generalmente conduce a un buen rendimiento de generalización en datos no vistos. El rendimiento del bosque se puede evaluar observando el error out-of-bag (OOB), que es el error de predicción promedio en cada muestra de entrenamiento, utilizando solo

los árboles que no tuvieron esa muestra en su conjunto de datos bootstrap.

A continuación, se presenta un ejemplo de implementación de Bosques Aleatorios en Python utilizando scikit-learn para predecir la actividad de fármacos en función de descriptores moleculares. El primer paso es obtener un conjunto de datos de moléculas con sus correspondientes descriptores moleculares (características numéricas que representan diversas propiedades de las moléculas) y su actividad biológica contra una diana específica (por ejemplo, activa o inactiva) [3]. Se podrían utilizar conjuntos de datos de plataformas de descubrimiento de fármacos o bases de datos públicas como PubChem. Se lleva a cabo un análisis exploratorio de datos (EDA) para comprender la distribución de los descriptores moleculares y el equilibrio entre compuestos activos e inactivos. Se visualiza la relación entre diferentes descriptores y la actividad. Los datos se dividen en conjuntos de entrenamiento y prueba. Se manejan los valores faltantes. El escalado de características podría ser beneficioso para algunos descriptores moleculares, pero no es estrictamente necesario para modelos basados en árboles como los Bosques Aleatorios. Se importa la clase RandomForestClassifier (para la clasificación de la actividad) o RandomForestRegressor (para la predicción de valores de actividad continua) de sklearn.ensemble. Se instancia el modelo con los hiperparámetros deseados (por ejemplo, n_estimators - número de árboles, max_depth). Se entrena el modelo utilizando los datos de entrenamiento: rf_model.fit(X_train, y_train). Se realizan predicciones en el conjunto de prueba. El rendimiento del modelo se evalúa utilizando las métricas apropiadas. Se exploran las puntuaciones de importancia de las características obtenidas de rf_model.feature_importances_ para identificar qué descriptores moleculares son más predictivos de la actividad del fármaco.

Python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos (reemplazar con la carga de datos real)
# Ejemplo utilizando un conjunto de datos de ejemplo
```

```
data = {'descriptor1': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
        'descriptor2': [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0],
        'activity': }
df = pd.DataFrame(data)
X = df[['descriptor1', 'descriptor2']]
y = df['activity']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Inicializar y entrenar el clasificador de bosque aleatorio
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Hacer predicciones
y_pred = rf_model.predict(X_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión: {accuracy}")
print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de Confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()

# Importancia de las características
importance = rf_model.feature_importances_
print("\nImportancia de las Características:")
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[i]:.4f}")
```

El código Python demuestra la aplicación de Bosques Aleatorios para predecir la actividad de fármacos en función de descriptores moleculares. El ejemplo destaca la facilidad de uso de RandomForestClassifier en scikit-learn y la capacidad de extraer

puntuaciones de importancia de características, que son valiosas para identificar las propiedades moleculares clave que influyen en la actividad del fármaco.

6. Máquinas de Aumento de Gradiente (Gradient Boosting Machines) en Ciencias de la Vida:

El análisis exploratorio de datos (EDA) es crucial para comprender la complejidad de la relación entre las características y la variable objetivo. Las Máquinas de Aumento de Gradiente (GBM) pueden modelar relaciones no lineales complejas, y el EDA puede ayudar a determinar si existe tal complejidad en los datos [11]. Las visualizaciones como diagramas de dispersión o mapas de calor pueden revelar patrones o interacciones no lineales. Si bien el EDA no visualiza directamente el proceso de aumento, la comprensión de las características de los datos a través del EDA (por ejemplo, la presencia de valores atípicos, el desequilibrio de clases) puede informar la elección de los hiperparámetros en GBM que controlan la tasa de aprendizaje y el número de etapas de aumento.

GBM es otra técnica de aprendizaje conjunto que construye múltiples modelos secuencialmente. Cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores. Esto se hace ajustando el nuevo modelo a los residuos (las diferencias entre los valores reales y las predicciones) del modelo anterior. GBM utiliza una función de pérdida para medir el error del modelo. El objetivo de cada nuevo modelo es minimizar esta función de pérdida moviéndose en la dirección del gradiente negativo de la pérdida con respecto a las predicciones actuales. Se pueden utilizar diferentes funciones de pérdida según la tarea (por ejemplo, error cuadrático para regresión, pérdida logística para clasificación). GBM es propenso al sobreajuste si el número de etapas de aumento es demasiado alto o si los árboles individuales son demasiado complejos. Se utilizan técnicas de regularización, como limitar la profundidad del árbol, utilizar un factor de contracción (tasa de aprendizaje) y submuestrear los datos o las características, para mejorar la robustez y la capacidad de generalización del modelo.

A continuación, se presenta un ejemplo de implementación de Máquinas de Aumento de Gradiente en Python utilizando scikit-learn para predecir la respuesta del paciente a la terapia. El primer paso es obtener un conjunto de datos de ciencias de la vida donde el objetivo es predecir cómo responderán los pacientes a una terapia en particular en función de sus características (por ejemplo, marcadores genéticos, historial médico, estilo de vida). Esto podría implicar predecir si un paciente responderá positiva o negativamente (clasificación) o predecir el grado de su respuesta (regresión) [4]. Se lleva a cabo un análisis exploratorio de datos (EDA) para

comprender la distribución de las características del paciente y la variabilidad en la respuesta a la terapia. Se visualiza la relación entre diferentes características y la variable de respuesta. Los datos se dividen en conjuntos de entrenamiento y prueba. Se manejan los valores faltantes y las características categóricas (que podrían necesitar ser codificadas para GBM). El escalado de características podría ser beneficioso dependiendo de la implementación específica de GBM utilizada. Se importa la clase GradientBoostingClassifier (para clasificación) o GradientBoostingRegressor (para regresión) de sklearn.ensemble. Se instancia el modelo con los hiperparámetros deseados (por ejemplo, n_estimators, learning_rate, max_depth). Se entrena el modelo utilizando los datos de entrenamiento: gbm_model.fit(X_train, y_train). Se realizan predicciones en el conjunto de prueba. El rendimiento del modelo se evalúa utilizando las métricas apropiadas. Se explora la importancia de las características del modelo GBM para comprender qué características del paciente son más predictivas de la respuesta a la terapia. Se utiliza la validación cruzada con GridSearchCV o RandomizedSearchCV para encontrar los hiperparámetros óptimos para el modelo GBM.

Python

```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el conjunto de datos (reemplazar con la carga de datos real)
# Ejemplo utilizando un conjunto de datos de ejemplo
data = {'marker1': [0.5, 0.6, 0.7, 0.8, 0.9, 0.4, 0.3, 0.2, 0.1, 1.0],
        'marker2': [2.0, 1.9, 1.8, 1.7, 1.6, 2.1, 2.2, 2.3, 2.4, 1.5],
        'therapy_response': }
df = pd.DataFrame(data)
X = df[['marker1', 'marker2']]
y = df['therapy_response']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Inicializar el clasificador de aumento de gradiente
gbm_model = GradientBoostingClassifier(random_state=42)

# Definir la cuadrícula de parámetros para GridSearchCV
param_grid = {'n_estimators': ,
              'learning_rate': [0.01, 0.1, 0.2],
              'max_depth': [3, 4, 5]}

# Realizar GridSearchCV para el ajuste de hiperparámetros
grid_search = GridSearchCV(gbm_model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Obtener los mejores parámetros y el mejor modelo
best_params = grid_search.best_params_
best_gbm_model = grid_search.best_estimator_

print(f"Mejores Parámetros: {best_params}")

# Hacer predicciones utilizando el mejor modelo
y_pred = best_gbm_model.predict(X_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Precisión: {accuracy}")
print("\nInforme de Clasificación:")
print(classification_report(y_test, y_pred))

# Matriz de Confusión
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()

# Importancia de las características
importance = best_gbm_model.feature_importances_
print("\nImportancia de las Características:")
for i, feature in enumerate(X.columns):
```

```
print(f"{feature}: {importance[i]:.4f}")
```

El código Python demuestra el uso de Máquinas de Aumento de Gradiente para predecir la respuesta del paciente a la terapia, incluyendo el ajuste de hiperparámetros con GridSearchCV para encontrar la combinación óptima de n_estimators, learning_rate y max_depth. Esto destaca el poder de GBM para capturar relaciones complejas en datos médicos y la importancia de una cuidadosa optimización de hiperparámetros para lograr el mejor rendimiento predictivo.

7. Conclusión:

Los cinco algoritmos de aprendizaje automático supervisado discutidos en este informe (KNN, árboles de decisión, SVM, bosques aleatorios y máquinas de aumento de gradiente) ofrecen diferentes enfoques para abordar problemas de clasificación y regresión en ciencias de la vida. Cada algoritmo tiene sus propias fortalezas y debilidades, lo que los hace más o menos adecuados para tareas específicas y tipos de datos. La siguiente tabla resume las características clave, ventajas, desventajas y aplicaciones típicas en ciencias de la vida de cada algoritmo.

Tabla 1: Comparación de Algoritmos de Aprendizaje Automático en Ciencias de la Vida

Algoritmo	Base Teórica	Papel del EDA	Ventajas	Desventajas	Aplicaciones Típicas en Ciencias de la Vida
KNN	Aprendizaje basado en la distancia, votación mayoritaria/promedio	Relevancia de las características, necesidades de escalado, detección de valores atípicos, selección óptima de 'k'	Simple, versátil (clasificación y regresión), no paramétrico, captura relaciones no lineales	Computacionalmente costoso, sensible a características irrelevantes/dimensionalidad, el rendimiento depende de	Clasificación de enfermedades, predicción de pronóstico, análisis de similitud de pacientes, identificació

				'k' y la distancia	n de biomarcadores
Árboles de Decisión	Partición recursiva basada en criterios de división	Distribuciones de características, manejo de tipos de datos, interacciones potenciales	Interpretable, maneja datos categóricos y numéricos, importancia de las características	Propenso al sobreajuste, puede ser inestable (sensible a pequeños cambios en los datos)	Predicción de riesgo de enfermedades, generación de reglas de diagnóstico, identificación de factores clave que influyen en los resultados
Máquinas de Soporte Vectorial	Maximización del hiperplano, funciones kernel	Separabilidad de datos, dimensionalidad, necesidades de escalado, detección de valores atípicos	Eficaz en espacios de alta dimensión, versátil con diferentes kernels, robusto a valores atípicos (con ajuste cuidadoso)	Puede ser computacionalmente costoso, la elección del kernel es crítica, menos interpretable	Clasificación de muestras biológicas (por ejemplo, cancerosas vs. benignas), descubrimiento de biomarcadores, predicción de actividad de fármacos
Bosques Aleatorios	Aprendizaje conjunto (bagging), selección aleatoria de características	Información sobre la importancia de las características, comprensión de los beneficios del conjunto	Robusto al sobreajuste, buena generalización, proporciona importancia de las características, maneja alta dimensionali	Menos interpretable que los árboles de decisión individuales, puede ser computacionalmente intensivo para bosques	Predicción de actividad de fármacos, clasificación de enfermedades, identificación de características importantes

			dad	muy grandes	a partir de grandes conjuntos de datos, predicción de resultados de pacientes
Máquinas de Aumento de Gradiente	Aprendizaje secuencial (boosting), optimización de gradiente	Identificación de relaciones complejas, información sobre la elección de hiperparámetros	Alta precisión predictiva, maneja no linealidades complejas, importancia de las características	Propenso al sobreajuste (requiere un ajuste cuidadoso), puede ser computacionalmente costoso, menos interpretable	Predicción de la respuesta del paciente a la terapia, predicción de la progresión de la enfermedad, predicción de la eficacia de los fármacos

El análisis exploratorio de datos (EDA) y la selección de algoritmos de aprendizaje automático tienen una relación sinérgica. Los conocimientos obtenidos del EDA sobre la estructura, la calidad y los patrones subyacentes de los datos pueden guiar la selección de algoritmos apropiados y el ajuste de sus hiperparámetros. Por ejemplo, si el EDA revela una fuerte separabilidad lineal entre las clases, un algoritmo más simple como la regresión logística o una SVM lineal podría ser suficiente. Si las relaciones son altamente no lineales, algoritmos como los Bosques Aleatorios o las Máquinas de Aumento de Gradiente podrían ser más apropiados.

En el futuro, se espera que la aplicación de estas técnicas en ciencias de la vida siga evolucionando. Las tendencias emergentes incluyen el uso creciente del aprendizaje profundo para tareas complejas como el análisis de imágenes y el procesamiento del lenguaje natural de registros médicos, los desafíos de lidiar con datos de alta dimensión y multiómicos, la necesidad de IA explicable (XAI) en aplicaciones clínicas y la importancia de la privacidad de los datos y las consideraciones éticas.

Fuentes citadas

1. How Machine Learning will Transform Biomedicine - PMC, acceso: marzo 26, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC7141410/

2. The Role of Artificial Intelligence in Life Sciences MRL - MRL Consulting Group, acceso: marzo 26, 2025, https://www.mrlcg.com/resources/blog/ai-in-life-sciences/
3. Machine Learning for Healthcare and Life Sciences (HCLS) | Bitstrapped Blog, acceso: marzo 26, 2025, https://www.bitstrapped.com/blog/machine-learning-role-biotech-medical-industry
4. Artificial intelligence and machine learning in precision and genomic medicine - PMC, acceso: marzo 26, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9198206/
5. How AI Is Transforming Life Sciences, acceso: marzo 26, 2025, https://www.sapiosciences.com/blog/how-ai-is-transforming-life-sciences/
6. pmc.ncbi.nlm.nih.gov, acceso: marzo 26, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC7141410/#:~:text=Early%20uses%20of%20machine%20learning,%2C%202020)%2C%20and%20identify%20clusters
7. (PDF) Machine Learning in Biology and Medicine - ResearchGate, acceso: marzo 26, 2025, https://www.researchgate.net/publication/336962967_Machine_Learning_in_Biology_and_Medicine
8. The Applications of Machine Learning in Biology - The Kolabtree Blog, acceso: marzo 26, 2025, https://www.kolabtree.com/blog/applications-of-machine-learning-in-biology/
9. The Roles of Machine Learning in Biomedical Science - Frontiers of Engineering - NCBI, acceso: marzo 26, 2025, https://www.ncbi.nlm.nih.gov/books/NBK481619/
10. Applications of machine learning in bio/bioinformatics - Reddit, acceso: marzo 26, 2025, https://www.reddit.com/r/bioinformatics/comments/t3qiwz/applications_of_machine_learning_in/
11. What is Exploratory Data Analysis? - GeeksforGeeks, acceso: marzo 26, 2025, https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/
12. k-nearest neighbors (KNN) with Examples in Python - Domino Data Lab, acceso: marzo 26, 2025, https://domino.ai/blog/knn-with-examples-in-python
13. K-Nearest Neighbors (KNN) Classification with scikit-learn - DataCamp, acceso: marzo 26, 2025, https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
14. Guide to K-Nearest Neighbors Algorithm in Machine Learning - Analytics Vidhya, acceso: marzo 26, 2025, https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/
15. K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks, acceso: marzo 26, 2025, https://www.geeksforgeeks.org/k-nearest-neighbours/
16. KNN Classifier in Python: Implementation, Features and Application - Analytics Vidhya, acceso: marzo 26, 2025, https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/

17. What is the k-nearest neighbors algorithm? - IBM, acceso: marzo 26, 2025, https://www.ibm.com/think/topics/knn
18. K Nearest Neighbors (KNN) | Statistical Software for Excel - XLSTAT, acceso: marzo 26, 2025, https://www.xlstat.com/en/solutions/features/k-nearest-neighbors-knn
19. K-Nearest Neighbor: The Theory Behind the Algorithm | by Brunno Márcio | Medium, acceso: marzo 26, 2025, https://medium.com/@brunno.marcio/k-nearest-neighbor-the-theory-behind-the-algorithm-84b97c60d1ac
20. K-Nearest Neighbors (KNN) Algorithm for Machine Learning | by Madison Schott | Capital One Tech | Medium, acceso: marzo 26, 2025, https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26
21. K-Nearest Neighbor (KNN) Explained - Pinecone, acceso: marzo 26, 2025, https://www.pinecone.io/learn/k-nearest-neighbor/
22. K-Nearest Neighbors (KNN) in Python - Tutorial - centron GmbH, acceso: marzo 26, 2025, https://www.centron.de/en/tutorial/k-nearest-neighbors-knn-in-python-tutorial/
23. k-nearest neighbor algorithm in Python - GeeksforGeeks, acceso: marzo 26, 2025, https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/
24. K-Nearest Neighbors (KNN) Regression with Scikit-Learn - GeeksforGeeks, acceso: marzo 26, 2025, https://www.geeksforgeeks.org/k-nearest-neighbors-knn-regression-with-scikit-learn/
25. High-Level K-Nearest Neighbors (HLKNN): A Supervised Machine Learning Model for Classification Analysis - MDPI, acceso: marzo 26, 2025, https://www.mdpi.com/2079-9292/12/18/3828
26. Using k-Nearest Neighbors (k-NN) in Production - Domino Data Lab, acceso: marzo 26, 2025, https://domino.ai/blog/summary-using-k-nn-production
27. Exploring K-Nearest Neighbour Classification - Towards Data Science, acceso: marzo 26, 2025, https://towardsdatascience.com/exploring-k-nearest-neighbour-classification-514d09175248/
28. Life Sciences Datasets: Home - Research Guides - Central Michigan University, acceso: marzo 26, 2025, https://libguides.cmich.edu/lifesciencedata
29. Data Collections - Biology - Library Guides at Brown University, acceso: marzo 26, 2025, https://libguides.brown.edu/c.php?g=545426&p=3741199
30. Healthcare Data Sets: 10 Great Data Sources, acceso: marzo 26, 2025, https://kms-healthcare.com/blog/healthcare-data-sets/
31. End-to-End K-Nearest Neighbour Modelling | by Data Science Wizards | Medium, acceso: marzo 26, 2025, https://medium.com/@datasciencewizards/end-to-end-k-nearest-neighbour-modelling-66ac6971365b
32. How To Classify Data In Python using Scikit-learn - ActiveState, acceso: marzo 26, 2025,

https://www.activestate.com/resources/quick-reads/how-to-classify-data-in-python/
33. AI and machine learning: revolutionising drug discovery and transforming patient care - Roche, acceso: marzo 26, 2025, https://www.roche.com/stories/ai-revolutionising-drug-discovery-and-transforming-patient-care
34. Machine Learning for Drug Discovery - Manning Publications, acceso: marzo 26, 2025, https://www.manning.com/books/machine-learning-for-drug-discovery
35. Machine Learning for Drug Development - Zitnik Lab - Harvard University, acceso: marzo 26, 2025, https://zitniklab.hms.harvard.edu/drugml/
36. Machine Learning for Drug Discovery | ACS In Focus, acceso: marzo 26, 2025, https://pubs.acs.org/doi/book/10.1021/acsinfocus.7e5017
37. Artificial Intelligence in Health Care: Benefits and Challenges of Machine Learning in Drug Development [Reissued with revisions on Jan. 31, 2020.] - GAO, acceso: marzo 26, 2025, https://www.gao.gov/products/gao-20-215sp