PROYECTO DE INTELIGENCIA ARTIFICIAL

Nombre: Edy German Perez Calcina

Materia: DAT-245/ Inteligencia Artificial

0. Descripción a detalle del Dataset:

El dataset utilizado para el proyecto será el dataset de Exoplanetas (Planetas fuera de nuestro sistema solar), posee 12 columnas y **4842 filas. B**ajado de kaggle con ruta: https://www.kaggle.com/datasets/diaaessam/exoplanets-planets-outside-our-galaxy; este dataset tiene las columnas de :

- Name: (el nombre oficial del exoplaneta)
- **Mass (MJ): La masa júpiter del exoplaneta, la Masa Júpiter equivale a 1.898×1027kg.**
- Radius(RJ): El radio Jupiter del exoplaneta, el Radio Júpiter equivale a **7.1492×107 m.**
- Period (days): Los días que tarda el exoplaneta en dar una vuelta alrededor de su estrella (Sol).
- Semi-major axis (AU): La mitad del diámetro más largo de una elipse, en **Unidades astronómicas (AU). Equivale a 1.496×1011 m**
- Temp. (K): Temperatura del exoplaneta en grados kelvin, **1 Kelvin** es equivalente a **-272.15 °C**.
- Discovery method: La forma en que el exoplaneta fue descubierto.
- Disc. Year: Representa el año de su descubrimiento.
- Distance (ly): Representa la distancia del exoplaneta a la tierra en años luz. Un año luz equivale a **9.461×10^{12} kilómetros**
- **Host star mass (M⊙): La masa de la estrella en la que orbita el exoplaneta en masa solar.**
- Host star temp. (K): La temperatura de la estrella que orbita en kelvins.
- Remarks: Detalles relevantes de conocer referidos al exoplaneta. No todas las filas poseen una remarcación.

0.5 Describa claramente del objetivo de investigación a partir del dataset elegido

Utilizando los datos encontrados en el dataset el objetivo es encontrar el exoplaneta más similar a la Tierra en base a sus características físicas y orbitales; así como las características de su estrella anfitriona.

Identificando qué características de los exoplanetas y sus estrellas anfitrionas son más relevantes para determinar la "similitud" con la Tierra. Algunas de las características clave podrían incluir:

-Radio: El radio del exoplaneta, ya que un planeta más pequeño o más grande podría tener diferentes condiciones geofísicas.
-Masa: La masa influye en la gravedad y otros aspectos físicos.
Temperatura de equilibrio: Relacionada con la distancia al sol y la capacidad para mantener agua en estado líquido.
-Periodo orbital: El tiempo que el exoplaneta tarda en orbitar su estrella.

- **Semi-major axis (AU)**: Esta medida indica la distancia promedio del exoplaneta a su estrella. Para que un exoplaneta sea similar a la Tierra, debería estar en la zona habitable de su estrella.

- **Temp. (K)**: La temperatura del exoplaneta, que se relaciona con la posibilidad de agua líquida, similar a la Tierra (alrededor de 273 K a 373 K para condiciones de habitabilidad).
- **Host star mass (M⊙)**: La masa de la estrella anfitriona ayuda a determinar la longevidad y estabilidad de la estrella. Las estrellas más similares al Sol (tipo G) son más propensas a albergar planetas similares a la Tierra.
  - Host star temp. (K): Relacionado con la luminosidad de la estrella, lo que influye en la zona habitable y la temperatura de su exoplaneta.

1. Preprocesamiento de los datos

- **Conversión de unidades**:
  - La masa de Júpiter (MJ) se convierte a la masa de la Tierra.
  - El radio de Júpiter (RJ) se convierte al radio de la Tierra (aproximadamente 6371 km).
- **Normalización de los datos**: Dado que los valores de las características tienen diferentes rangos, será útil normalizar los datos para que todas las características contribuyan de manera similar al modelo.

2. Cálculo de similitudes

Utilizando la distancia euclidiana entre las características del exoplaneta y las de la Tierra para calcular cuál es el más similar:

Distancia= $\sqrt{[}(Massplanet-MassTierra)^2+(Radiusplanet-RadiusTierra)^2+(Periodplanet-PeriodTierra)^2+(Semi-majoraxisplanet-Semi-majoraxisTierra)^2+(Tempplanet-TempTierra)^2+(Hoststarmassplanet-HoststarmassTierra)^2+(Hoststartempplanet-HoststartempTierra)^2]$

La Tierra tiene los siguientes valores aproximados para comparación:

- **Mass**: 1 (masa terrestre)
- **Radius**: 1 (radio terrestre)
- **Period**: 365.25 días (un año terrestre)
- **Semi-major axis**: 1 AU (distancia media de la Tierra al Sol)
- **Temp**: 288 K (temperatura promedio de la Tierra)
- **Host star mass**: 1 M⊙ (masa solar)
- **Host star temp**: 5778 K (temperatura de la estrella del Sol)

1. Preprocesamiento (al menos una valida, otros dos por ver los resultados, si no se aplica justifique porque), Balanceo de datos

R Para el preprocesamiento se transformó todas las columnas a tipo numérico para asegurar que los datos sean trabajables en la línea:

```
columnas_a_numerico = [
    "Mass (MJ)", "Radius (RJ)", "Period (days)",
    "Semi-major axis (AU)", "Temp. (K)", "Distance (ly)",
    "Host star mass (M⊙)", "Host star temp. (K)"
]
for col in columnas_a_numerico:
    datos[col] = pd.to_numeric(datos[col], errors='coerce')
```

En la que se introduce los nombres de cada columna y utilizando la función de for por columnas de las columnas mencionadas se determinaba el cambio de tipo de dato.
Posteriormente se utiliza el **simpleimputer** para cambiar los datos vacíos que se mostrarían como 'NaN', se los reemplazaría con la mediana de la columna. La aplicación se la encuentra en las líneas:

```
imputador = SimpleImputer(strategy="median")
datos[columnas_a_numerico] = imputador.fit_transform(datos[columnas_a_numerico])
```

Donde se define el imputador llamando a la librería de sklearn.impute, SimpleImputer; indicando que el método de reemplazo será la mediana y se le ingresa la lista de columnas ya definidas en "Columnas_a_numerico" para que realice la transformación sobre ellas.

Lo siguiente es cambiar los valores para Mass (MJ) y Radius RJ; utilizando una regla de tres simple para que en lugar de estar en valor de masa y radio jupiter esten en valor de masa y radio tierra, que serian multiplicarlos por 317.8 masas de la tierra y 11.2 radios de la tierra; respectivamente.

```
datos["Mass (Tierra)"] = datos["Mass (MJ)"] * 317.8
datos["Radius (Tierra)"] = datos["Radius (RJ)"] * 11.2
```

Para la normalización se utiliza el MinMaxScaler que utiliza la fórmula de x'= (x-minX)/(maxX-MinX), donde x son los datos originales, minX el valor mínimo del conjunto, max el valor máximo del conjunt, asi x' es el nuevo valor normalizado.

```
columnas_a_escalar = ["Mass (Tierra)", "Radius (Tierra)", "Period (days)", "Semi-major axis (AU)", "Temp. (K)"]
escalador = MinMaxScaler()
datos[columnas_a_escalar] = escalador.fit_transform(datos[columnas_a_escalar])
```

En la captura se puede establecer como primero se definen las columnas que serán normalizadas, posteriormente se establece el escalador de normalización invocando a MinMaxScaler y finalmente se establece una variable de datos[columnas_a_escalar] que será igual a la normalización de los datos seleccionados.
Posteriormente se cuenta cuántas veces aparece cada valor único en la columna 'Discovery method' del DataFrame 'datos'; se filtra esos conteos para conservar solo aquellos que tienen más de 2 ocurrencias y utilizando el ".index" se extrae los índices de este filtro, que son las clases válidas (valores únicos) que cumplen el criterio de tener más de 2 ocurrencias.
Se pasa a filtrar e l DataFrame creando una serie booleana que indica si cada fila en la columna 'Discovery method' está en la lista de 'clases_validas'. Además se filtra el DataFrame 'datos',

manteniendo solo las filas donde la condición anterior es verdadera. Esto elimina las filas que no tienen clases válidas. Posteriormente se separan características y etiqueta.

- X = datos[columnas_a_escalar]: Asigna a `X` las columnas especificadas en la lista `columnas_a_escalar`, que se utilizarán como características (inputs) para el modelo.
- y = datos['Discovery method']: Asigna a `y` la columna `'Discovery method'`, que se utilizará como la etiqueta (output) para el modelo.

```
clases_validas = datos['Discovery method'].value_counts()[datos['Discovery method'].value_counts() > 2].index
datos = datos[datos['Discovery method'].isin(clases_validas)]

X = datos[columnas_a_escalar]
y = datos['Discovery method']

X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
    X, y, test_size=0.3, random_state=45, stratify=y
)
```

Posteriormente se dividen los datos en conjuntos de entrenamiento y prueba.
Invocando la función train_test_split(...) que divide los datos en conjuntos de entrenamiento y prueba; sabiendo que X y "Y" son los datos de entrada y las etiquetas que se están dividiendo; se estable el test_size en 0.3 que indica que el 30% de los datos se utilizarán para la prueba, y el 70% para el entrenamiento mientras que se usa un random_state de 45, estableciendo una semilla para que la división sea reproducible; cada vez que se ejecute el código con esta semilla, se obtendrán los mismos conjuntos. Al final se usa el stratify=y para asegurar que la proporción de clases en el conjunto de prueba sea la misma que en el conjunto completo, lo que es importante para mantener la representación de las clases.
Durante el siguiente paso de preprocesamiento de los datos se utilizare el smote de imblearn.over_sampling que permite un control de sobre muestreo, que evitara que exista un desbalanceo entre clases y las clases minoritarias no se vean oprimidas por clases mas numerosas.

```
smote = SMOTE(random_state=45,k_neighbors=1)
X_entrenamiento, y_entrenamiento = smote.fit_resample(X_entrenamiento, y_entrenamiento)
```

Se inicializa un objeto de la clase SMOTE, con una semilla para la aleatoriedad, asegurando que los resultados sean reproducibles; a su vez que se establecen los k_neighbors en 1, que especifica que se utilizará un solo vecino más cercano para generar nuevos ejemplos sintéticos. Esto significa que el nuevo ejemplo se generará directamente entre el ejemplo de la clase minoritaria y su único vecino más cercano.
Se llama un fit_resample que ajusta SMOTE a los datos de entrenamiento y genera nuevos ejemplos sintéticos de la clase minoritaria, usando X_entrenamiento como el conjunto de características de entrenamiento y a y_entrenamiento como las etiquetas de entrenamiento. El resultado son conjuntos de datos X_entrenamiento y "y_entrenamiento" que ahora incluyen ejemplos sintéticos de la clase minoritaria, equilibrando así las clases.

```
clasificador = RandomForestClassifier(random_state=45, class_weight='balanced')
parametros = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(
    estimator=clasificador, param_grid=parametros, cv=3, scoring='f1_weighted', n_jobs=-1
)
grid_search.fit(X_entrenamiento, y_entrenamiento)


y_pred = grid_search.best_estimator_.predict(X_prueba)
print("Mejores parámetros encontrados:")
print(grid_search.best_params_)
print("\nReporte de clasificación:")
print(classification_report(y_prueba, y_pred, zero_division=0))
```

Para pasa a iniciar un clasificador basado en el algoritmo de Random Forest. Asegurando la reproducibilidad de los resultados en 45 y ajustando automáticamente los pesos de las clases inversamente proporcionales a sus frecuencias en el conjunto de datos, lo que ayuda a manejar el desbalanceo de clases.

Posteriormente se procede con la definición de parámetros para Grid Search a través de un diccionario llamado parámetros; donde n_estimators es el número de árboles en el bosque (50, 100, 200), max_depth hace referencia a lrofundidad máxima de cada árbol (10, 20, o sin límite) y el min_samples_split representa el número mínimo de muestras requeridas para dividir un nodo (2, 5, 10). Así podremos realizar la configuración de Grid Search, usando GridSearchCV para realizar una búsqueda exhaustiva sobre los hiperparámetros especificados. Mientras el estimator=clasificador se está optimizando, dentro de param_grid=parámetros los parámetros que se van a ajustar, seleccionando cv=3 como los números de particiones para la validación cruzada (3 pliegues). El scoring= f1_weighted es la métrica utilizada para evaluar el rendimiento del modelo, en este caso, el F1 score ponderado; mientras que n_jobs igual a utiliza todos los núcleos disponibles para realizar la búsqueda en paralelo, lo que acelera el proceso. Una vez se define, se entrena el modelo utilizando la búsqueda de cuadrícula sobre los datos de entrenamiento (`X_entrenamiento`, `y_entrenamiento`). Esto ajusta el modelo a los mejores hiperparámetros encontrados.

Tendremos así y_pred = grid_search.best_estimator_.predict(X_prueba) que obtiene el mejor modelo encontrado durante la búsqueda y realiza predicciones sobre el conjunto de prueba (`X_prueba`). Siendo para el control la impresión de los resultados. Tanto print(grid_search.best_params_) que mestra los mejores parámetros encontrados durante la búsqueda y classification_report(y_prueba, y_pred, zero_division=0) que genera un reporte que incluye métricas como precisión, recall y F1-score para cada clase, evaluando el rendimiento del modelo en el conjunto de prueba.

```
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
X_entrenamiento_pca, X_prueba_pca, y_entrenamiento_pca, y_prueba_pca = train_test_split(
    X_pca, y, test_size=0.3, random_state=45, stratify=y
)
clasificador.fit(X_entrenamiento_pca, y_entrenamiento_pca)

y_pred_pca = clasificador.predict(X_prueba_pca)
print("\nResultados con PCA:")
print(confusion_matrix(y_prueba_pca, y_pred_pca))
print(classification_report(y_prueba_pca, y_pred_pca))
```

Eventualmente llegamos a la aplicación de PCA (PCA permite el análisis de Componentes Principales). Se inicializa un objeto de la clase PCA (Análisis de Componentes Principales) con el parámetro n_components=3, que indica que se quieren reducir las dimensiones del conjunto de datos a 3 componentes principales, esto será almacenado en una variable llamada X_pca, que será la aplicación de pca a los datos de las columnas principales, pues con fit se ajusta el modelo PCA a los datos `X`, calculando las componentes principales y la varianza explicada por cada componente. Transformando los datos originales `X` a un nuevo espacio de características con 3 dimensiones, que son las componentes principales; así X_pca es el nuevo conjunto de datos con las características reducidas.
Se pasa posteriormente a la división del Conjunto de Datos, dividiendo el conjunto de datos en conjuntos de entrenamiento y prueba, usando el conjunto de datos transformado por PCA y las etiquetas correspondientes a los datos. Se indica que el 30% de los datos se utilizarán para el conjunto de prueba, mientras que el 70% se utilizarán para el entrenamiento y se establece una semilla para la aleatoriedad, asegurando que la división sea reproducible. Asegurando que la proporción de clases en el conjunto de entrenamiento y prueba sea la misma que en el conjunto original, manteniendo el balance de clases. Así se entrena el clasificador (que fue definido posterior a la declaración del smote) utilizando el conjunto de entrenamiento reducido (`X_entrenamiento_pca`) y sus etiquetas correspondientes (`y_entrenamiento_pca`).
Asi se utiliza el clasificador entrenado para hacer predicciones sobre el conjunto de prueba (`X_prueba_pca`), generando las predicciones en `y_pred_pca`. Se imprime un mensaje indicando que se mostrarán los resultados de la evaluación, se genera y muestra la matriz de confusión, que muestra el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos en las predicciones; así como se genera un informe que incluye métricas de rendimiento como precisión, recall y F1-score para cada clase, evaluando el rendimiento del clasificador en el conjunto de prueba.

```
kmeans = KMeans(n_clusters=3, random_state=45)
kmeans.fit(X)
datos['Cluster'] = kmeans.labels_
```

Se inicializará también el KMeans que agrupa conjuntos de datos similares conocidos como clusters, en el código se establece 3 conjuntos de datos similares y se inicializa una semilla de 45 para la repartición de datos ,posteriormente se le aplica el ajuste de subconjuntos de datos a los datos iniciales X y a trabes del kmeans.labels_ se le realiza el etiquetado a los datos similares.
Posteriormente en busca de ejecutar el objetivo del proyecto de identificar el planeta más similar a la tierra se establece los parámetros comparativos.

```
caracteristicas_tierra = {
    "Mass (Tierra)": 1,
    "Radius (Tierra)": 1,
    "Period (days)": 365.25,
    "Semi-major axis (AU)": 1,
    "Temp. (K)": 288
}
```

Utilizando datos cuasi reales de la tierra.

Posteriomente se crea una función para calcular la similitud utilizando la distancia euclidiana que saca la raíz a la resta de los valores de cada planeta menos los datos de la tierra, cada resta elevada al cuadrado para todos los datos relevantes.

```
def calcular_similitud(fila):
    return np.sqrt(sum((fila[col] - caracteristicas_tierra[col])**2 for col in caracteristicas_tierra))

datos['Similitud'] = datos[columnas_a_escalar].apply(calcular_similitud, axis=1)
mas_similar = datos.loc[datos['Similitud'].idxmin()]
print("\nExoplaneta más similar a la Tierra:")
print(mas_similar)
```

Se seleccionan las columnas del DataFrame 'datos' que se especifican en la lista `columnas_a_escalar`. Estas columnas son las que se utilizarán para calcular la similitud entre los exoplanetas. Se aplica una función a lo largo de un eje del DataFrame llamda calcular_similitud a cada fila y por eso se usa axis=1.
Se crea una nueva columna en el DataFrame `datos` llamada `'Similitud'`, que contiene los valores de similitud calculados para cada fila (cada exoplaneta), mientras se busca el índice de la fila que tiene el valor mínimo en la columna `'Similitud'`. El valor mínimo indica que ese exoplaneta es el más similar a la Tierra). Se Utiliza el índice encontrado para acceder a la fila correspondiente en el DataFrame 'datos'. Finalmete se imprime un mensaje en la consola para indicar que se mostrará el exoplaneta más similar a la Tierra y se imprime la fila del DataFrame 'mas_similar', que contiene toda la información sobre el exoplaneta que es más similar a la Tierra, incluyendo sus características y valores.

2. **Selección del clasificador:**

Para abordar el problema planteado en el código, se utilizaron dos algoritmos: **Random Forest** como clasificador supervisado y **KMeans** como método no supervisado. Estas estrategias fueron seleccionadas porque se complementan entre sí, ajustándose tanto a las características del conjunto de datos como a los objetivos del análisis.

Clasificador supervisado: Random Forest

El **Random Forest** fue elegido como el modelo principal para el análisis supervisado debido a sus ventajas técnicas y prácticas:

1. **Capacidad para manejar datos multiclase:** La variable objetivo, "Discovery method", incluye varias clases que representan diferentes enfoques para descubrir exoplanetas. Random Forest es particularmente eficaz para manejar estos escenarios multiclase porque utiliza un conjunto de árboles de decisión. Al combinar las predicciones de múltiples árboles, este modelo ofrece un análisis robusto y confiable.
2. **Resistencia a datos ruidosos:** Este algoritmo minimiza el impacto de valores atípicos o ruido en los datos, ya que promedia las decisiones de múltiples árboles independientes. Esto reduce significativamente el riesgo de que un solo árbol domine el resultado, haciéndolo más fiable para conjuntos de datos complejos y variables.
3. **Importancia de las características:** Una de las ventajas clave de Random Forest es su capacidad para identificar las variables que tienen un mayor impacto en las predicciones. Esto no solo permite mejorar el modelo al centrarse en las características más relevantes, sino que también ayuda a entender mejor las relaciones entre las variables y los métodos de descubrimiento.

Para maximizar el rendimiento del modelo, se empleó **GridSearchCV**, una técnica que explora combinaciones óptimas de hiperparámetros como el número de estimadores (n_estimators), la profundidad de los árboles (max_depth) y el mínimo de muestras requeridas para dividir un nodo (min_samples_split). Gracias a este proceso de optimización, el modelo alcanzó un rendimiento excepcional en el conjunto de prueba.

Algoritmo no supervisado: KMeans

En el análisis no supervisado, se utilizó **KMeans** para identificar patrones subyacentes en los datos, sin depender de una variable objetivo. Este algoritmo clasificó los exoplanetas en tres clusters, considerando características escaladas como masa, radio y temperatura.

La elección de KMeans se basa en varias razones:

1. **Simplicidad y eficiencia computacional:** KMeans es un algoritmo rápido y fácil de implementar, lo que lo convierte en una opción ideal para analizar datos de gran escala y alta dimensionalidad.
2. **Revelación de estructuras ocultas:** Aunque asume que los datos forman clusters esferoidales, su aplicación sigue siendo válida en este contexto, ya que ayuda a descubrir conexiones y patrones ocultos en los datos. Esto proporciona información valiosa para complementar el análisis supervisado.
3. **Complemento al enfoque supervisado:** Los resultados de KMeans no solo ofrecen una perspectiva alternativa sobre las estructuras del conjunto de datos, sino que también pueden servir como base para futuras iteraciones del modelo, ayudando a identificar áreas que requieren mayor exploración o refinamiento.

Justificación del Clasificador

La elección de **Random Forest** como el principal clasificador supervisado está fundamentada en las siguientes ventajas:

1. **Interpretabilidad:** Random Forest permite inspeccionar de manera directa la importancia de las características, facilitando la comprensión de los resultados tanto para audiencias técnicas como no técnicas. Esto lo hace más accesible en comparación con modelos más complejos como las redes neuronales.
2. **Flexibilidad:** Este algoritmo puede manejar datos heterogéneos, incluyendo variables categóricas y continuas. Por ejemplo, en este caso, procesó con éxito variables como "Mass (Tierra)" y "Temp. (K)", adaptándose a las necesidades específicas del conjunto de datos.
3. **Mitigación de sobreajuste:** Random Forest reduce el riesgo de sobreajustar los datos de entrenamiento al combinar múltiples árboles de decisión. Esto asegura que el modelo generalice bien a nuevos datos, mejorando su rendimiento en escenarios del mundo real.

Referencias técnicas

- **Fuente ISBN:** 978-1-59327-868-4. Este libro describe en detalle la teoría y práctica de Random Forest, con ejemplos aplicados a problemas similares.
- **Fuente DOI:** 10.1007/978-1-4471-7448-4_6. Este capítulo destaca las ventajas y limitaciones de Random Forest en el aprendizaje supervisado.

Preprocesamiento y su relación con el clasificador

El éxito de los clasificadores en este análisis depende en gran medida de las técnicas de preprocesamiento aplicadas, que garantizan que los datos estén listos para su uso:

1. **Imputación de valores faltantes:** La imputación con la mediana asegura que no se pierda información valiosa debido a datos incompletos. Esto permite que Random Forest y KMeans trabajen con conjuntos de datos completos y confiables.
2. **Escalado de características:** La normalización mediante MinMaxScaler asegura que todas las características tengan la misma escala, evitando que una variable domine el modelo. Esto es especialmente crucial para KMeans, pero también beneficia a Random Forest al equilibrar la contribución de cada característica.

3. **Balanceo de datos:** Dado que la variable objetivo puede estar desbalanceada, el uso de SMOTE crea un conjunto de datos balanceado, mejorando significativamente el rendimiento del clasificador supervisado.

Estas técnicas no solo mejoran la eficiencia de los clasificadores, sino que también aseguran que las conclusiones obtenidas sean más confiables y precisas.

La combinación de **Random Forest** y **KMeans** representa un enfoque robusto y complementario para analizar datos complejos como los exoplanetas. Random Forest sobresale en tareas supervisadas, ofreciendo predicciones precisas y explicaciones claras, mientras que KMeans aporta valor al identificar patrones implícitos en los datos. Además, un proceso de preprocesamiento bien diseñado asegura que ambos algoritmos trabajen al máximo de su potencial. Este análisis no solo mejora nuestra comprensión del conjunto de datos, sino que también establece una base sólida para futuras investigaciones en el campo.

Para estructurar el código en secciones bien definidas, puedes organizarlo en los subtítulos que mencionaste. Aquí está una guía sobre cómo dividir el código en piezas según los requerimientos, junto con las partes específicas que ya están presentes o que se necesitan agregar.

1. Primera ejecución: Confiabilidad, matriz de confusión

- **Código relacionado:**

```
y_pred = grid_search.best_estimator_.predict(X_prueba)
print("Mejores parámetros encontrados:")
print(grid_search.best_params_)
print("\nReporte de clasificación:")
print(classification_report(y_prueba, y_pred, zero_division=0))
```

- **Acciones necesarias:**
  - Explicar la métrica de confiabilidad basada en el reporte de clasificación.
  - Mostrar la matriz de confusión:
  - from sklearn.metrics import confusion_matrix

```
4
5   y_pred_pca = clasificador.predict(X_prueba_pca)
6   print("\nResultados con PCA:")
7   print(confusion_matrix(y_prueba_pca, y_pred_pca))
8   print(classification_report(y_prueba_pca, y_pred_pca))
```

Resultados:

```
                           precision    recall  f1-score   support

                astrometry       0.00      0.00      0.00         1
                   imaging       0.92      0.85      0.88        13
             microlensing        0.16      0.89      0.28        37
orbital brightness modulation    0.00      0.00      0.00         1
              radial vel.        0.86      0.66      0.75       182
                   timing        0.15      0.20      0.17        10
                  transit        0.98      0.82      0.89       727

                 accuracy                           0.78       971
                macro avg       0.44      0.49      0.42       971
             weighted avg       0.92      0.78      0.83       971
```

```
Mejores parámetros encontrados:
{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}

Reporte de clasificación:
                              precision    recall  f1-score   support

                  astrometry       0.00      0.00      0.00         1
                     imaging       0.94      0.79      0.86        19
                 microlensing      0.85      0.31      0.45        55
orbital brightness modulation      0.00      0.50      0.01         2
                 radial vel.       0.89      0.65      0.75       274
                      timing       0.05      0.07      0.06        15
                     transit       0.98      0.81      0.89      1091

                    accuracy                           0.75      1457
                   macro avg       0.53      0.45      0.43      1457
                weighted avg       0.95      0.75      0.83      1457
```

```
Matriz de Confusión:
[[  0   0   1   0   0   0   0]
 [  0  11   2   0   0   0   0]
 [  0   0  33   0   4   0   0]
 [  0   0   0   0   0   0   1]
 [  1   1  43   0 121   8   8]
 [  0   0   3   0   4   2   1]
 [  0   0 119   0  12   3 593]]
```

2. Splits: Al menos 100 asignaciones, la mediana de la confiabilidad

- **Código relacionado:**

```
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
    X, y, test_size=0.3, random_state=45, stratify=y
)
```

- **Acciones necesarias:**
  - Verificar que el dataset tiene al menos 100 asignaciones por clase.
    Calcular la mediana de la confiabilidad para diferentes ejecuciones:
    from statistics import median
    from sklearn.metrics import f1_score

```
scores = []
for _ in range(5):
    X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
        X, y, test_size=0.3, random_state=45, stratify=y
    )
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    y_pred = clasificador.predict(X_prueba)
    scores.append(f1_score(y_prueba, y_pred, average='weighted'))
print("Mediana de la confiabilidad:", median(scores))
```

```
Mediana de la confiabilidad: 0.8382950605838515
```

La mediana de la confiabilidad es una medida estadística que resume el rendimiento de un modelo de aprendizaje automático en términos de un puntaje de evaluación, como el F1-score ponderado en este caso.
Su fórmula es:

$$F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recuperación}}{\text{Precisión} + \text{Recuperación}}$$

$$\text{Precisión} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

$$\text{Recuperación} = \frac{\text{VP}}{\text{VP} + \text{FN}}$$

Académico (Primera ejecución: 80/20)

```
scores = []

for _ in range(5):
    X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
        X, y, test_size=0.2, random_state=45, stratify=y
    )
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    y_pred = clasificador.predict(X_prueba)
    scores.append(f1_score(y_prueba, y_pred, average='weighted'))

print("Mediana de la confiabilidad:", median(scores))
```

```
Mediana de la confiabilidad: 0.839471088076958
```

Investigación (Segunda ejecución: 50/50)

```
scores = []

for _ in range(5):
    X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(
        X, y, test_size=0.5, random_state=45, stratify=y
    )
    clasificador.fit(X_entrenamiento, y_entrenamiento)
    y_pred = clasificador.predict(X_prueba)
    scores.append(f1_score(y_prueba, y_pred, average='weighted'))

print("Mediana de la confiabilidad:", median(scores))
```

- **Código relacionado:**
- # División Académica: 80/20

```
Mediana de la confiabilidad: 0.8329017598399433
```

- # División Investigación: 50/50

```
Mediana de la confiabilidad: 0.8329017598399433
```

- Comparar resultados entre ambas configuraciones.
  Tanto para 80/20 y para 50/50 se mantiene un estándar en la mediana de confiablidad, esto puede deberse ya sea a que el conjunto de datos es lo suficientemente grande, incluso con diferentes divisiones, el modelo puede seguir generalizando bien, lo que resulta en métricas similares. Pero también puede deberse a que el código para procesar y clasificar datos de exoplanetas, se observa que algunos modelos, como el RandomForestClassifier, que son robustos a cambios en la cantidad de datos de entrenamiento. Esto significa que, si el modelo tiene un buen rendimiento en general, puede que no muestre variaciones significativas en la confiabilidad, incluso al modificar la proporción de entrenamiento y prueba. Además, si el conjunto de datos es balanceado y se aplica correctamente el estratificado, la distribución de clases en ambas divisiones puede resultar en resultados similares. La mediana de confiabilidad calculada al final del script puede no cambiar drásticamente con pequeñas variaciones en los datos, especialmente si el rendimiento del modelo es consistente a lo largo de múltiples ejecuciones. Por otro lado, un conjunto de prueba más grande (como en una división 50/50) puede proporcionar una estimación más estable de la confiabilidad, mientras que un conjunto más pequeño puede ser más sensible a variaciones, lo que refuerza la idea de que la robustez del modelo y la calidad de la división de datos son fundamentales para obtener resultados confiables.
  Después de 10 iteraciones se mantenía la mediana entre 0.830 a 0.839

4. Primer Código: Github, Kaggle, Codelab

https://github.com/EdyPerez03/proyectoIA/tree/main/cod1

5. Aplicar Componentes Principales (PCA)

- **Código relacionado:**

```
pca = PCA(n_components=3)
X_entrenamiento_pca = pca.fit_transform(X_entrenamiento)
X_prueba_pca = pca.transform(X_prueba)

clasificador.fit(X_entrenamiento_pca, y_entrenamiento)
y_pred_pca = clasificador.predict(X_prueba_pca)

print("\nResultados con PCA:")
print(confusion_matrix(y_prueba, y_pred_pca))
print(classification_report(y_prueba, y_pred_pca))
```

El uso de PCA en el código se enfoca en reducción de dimensionalidad: Transformar los datos de entrada a un menor número de dimensiones (3 en este caso) para simplificar el modelo y evitar el sobreajuste. Mejora de la visualización y rendimiento: A veces, la reducción de dimensiones puede mejorar la capacidad de generalización del modelo y facilitar la visualización de los datos. PCA se aplica después de la división del conjunto de datos y el balanceo con SMOTE. Esto ayuda a trabajar con un espacio de características más manejable y, potencialmente, más informativo, manteniendo la capacidad de clasificación de manera eficiente.

6. Explicar cómo funciona PCA (Álgebra lineal)

El Análisis de Componentes Principales (PCA) es una técnica de reducción de dimensionalidad ampliamente utilizada en estadística y aprendizaje automático. El objetivo principal de PCA es transformar un conjunto de datos de alta dimensión en un espacio de menor dimensión, preservando la mayor cantidad posible de varianza en los datos. Para lograr esto, PCA identifica las componentes principales, que son las direcciones en las que los datos varían más. Para describir como se calcula el PCA utilizando la descomposición en valores singulares (SVD) se considera lo siguiente:

1. Conceptos fundamentales de SVD:

La descomposición en valores singulares es un método matemático que descompone una matriz A (de tamaño m×n) en tres matrices:

A=UΣV^T

- U: Una matriz ortogonal de tamaño m×m, cuyas columnas son los vectores singulares izquierdos.
- Σ\Sigma: Una matriz diagonal de tamaño m×n que contiene los valores singulares de A en orden descendente. Estos valores representan la magnitud de la varianza explicada por cada componente.
- V^T: La transpuesta de una matriz ortogonal de tamaño n×n, cuyas filas son los vectores singulares derechos.

2. Cálculo de PCA usando SVD:

PCA utiliza SVD para encontrar los vectores principales de un conjunto de datos.

a. Preparación de los datos:

- Supongamos que tenemos un conjunto de datos X con n observaciones (filas) y p características (columnas). Antes de aplicar PCA, es común centrar los datos, restando la media de cada columna para que los datos tengan media cero.

Xcentrado=X−media(X)

b. Aplicación de SVD:

- Se realiza la descomposición en valores singulares de la matriz de datos centrada

Xcentrado=UΣV^T

Los vectores de V (las filas de V^T) son los vectores singulares derechos, que corresponden a las direcciones principales en el espacio de características. Estos vectores representan las componentes principales de los datos.

c. Selección de componentes principales:

- Los valores singulares en la matriz Σ\Sigma indican la importancia de cada componente principal. Los primeros componentes principales son los que explican la mayor parte de la varianza de los datos. Se seleccionan los primeros k componentes principales (vectores de V) para reducir la dimensionalidad de los datos, donde k es menor que p.
- Los datos proyectados en el espacio de las k componentes principales se obtienen multiplicando la matriz de datos centrada por los primeros k vectores de V.

Xproyectado=Xcentrado * V sub k

donde Vk es una submatriz que contiene los primeros k vectores singulares derechos.

7. Aprendizaje no supervisado sin considerar "y" o la clase

Utilizado el KMeans para conseguir que el programa por su cuenta agrupe conjuntos similares de datos dentro de sí, para aplicarlo se tiene:

- **Código relacionado:**

```
kmeans = KMeans(n_clusters=3, random_state=45)
kmeans.fit(X)
datos['Cluster'] = kmeans.labels_
```

Se utiliza el algoritmo KMeans para agrupar los datos en 3 clusters (n_clusters=3). kmeans.fit(X) ajusta el modelo de clustering a los datos, dividiéndolos en grupos según las similitudes en las características de las observaciones. La columna Cluster se agrega al DataFrame datos para indicar a qué grupo pertenece cada observación.

Utilizando un conjunto de características que representan a la Tierra. Se utiliza la función calcular_similitud para medir la similitud de cada fila en el DataFrame con este conjunto de características, utilizando la distancia euclidiana.

```
caracteristicas_tierra = {
    "Mass (Tierra)": 1,
    "Radius (Tierra)": 1,
    "Period (days)": 365.25,
    "Semi-major axis (AU)": 1,
    "Temp. (K)": 288
}

def calcular_similitud(fila):
    return np.sqrt(sum((fila[col] - caracteristicas_tierra[col])**2 for col in caracteristicas_tierra))

datos['Similitud'] = datos[columnas_a_escalar].apply(calcular_similitud, axis=1)
```

La utilización de KMeans en este contexto posibilita la agrupación autónoma de los datos en grupos de observaciones similares, sin guiarse por una etiqueta de clase particular. Este procedimiento resulta beneficioso para segmentar datos en función de sus particularidades, y puede ser empleado en tareas como la identificación de patrones, el análisis de clusters y la búsqueda de objetos similares sin contar con un conocimiento previo acerca de las clases.

8. CORRIDA FINAL:

```
PS C:\Users\59171\Documents\Programacion\proyectoIA> & C:/Users/59171/anaconda3/python.exe c:/Users/59171/Documents/Programacion/proyectoIA/cod2/segundocodigo.py
Primeras filas del DataFrame:
              Name Mass (MJ) Radius (RJ) Period (days)  ... Distance (ly) Host star mass (M☉) Host star temp. (K)                                  Remarks
0        16 Cygni Bb      2.38        NaN        799.5  ...         68.99              1.04              5750                                      NaN
1        23 Librae b      1.61        NaN       258.18  ...         85.46              1.07              5736                                      NaN
2  47 Ursae Majoris b      2.53        NaN         1078  ...         45.02              1.08              5892                   Proper name Taphao Thong
3        51 Pegasi b      0.46        NaN     4.230785  ...         50.45              1.12              5793  Proper name Dimidium; previously informally na...
4        55 Cancri b    0.8306        NaN     14.65152  ...         41.06             0.905              5196                       Proper name Galileo

[5 rows x 12 columns]

Información del DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4855 entries, 0 to 4854
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Name                 4855 non-null   object
 1   Mass (MJ)            2295 non-null   object
 2   Radius (RJ)          3700 non-null   object
 3   Period (days)        4637 non-null   object
 4   Semi-major axis (AU) 3148 non-null   object
 5   Temp. (K)            1094 non-null   object
 6   Discovery method     4855 non-null   object
 7   Disc. Year           411 non-null    float64
 8   Distance (ly)        4706 non-null   object
 9   Host star mass (M☉)  4475 non-null   object
 10  Host star temp. (K)  4513 non-null   object
 11  Remarks              4502 non-null   object
dtypes: float64(1), object(11)
memory usage: 455.3+ KB
```

```
Evaluando con 5 componentes principales:
Mejores parámetros encontrados: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}

Reporte de clasificación:
                            precision    recall  f1-score   support

                astrometry       0.00      0.00      0.00         1
                   imaging       0.64      0.69      0.67        13
             microlensing        0.13      0.73      0.22        37
orbital brightness modulation       0.00      0.00      0.00         1
              radial vel.       0.83      0.58      0.68       182
                   timing       0.12      0.20      0.15        10
                  transit       0.98      0.81      0.88       727

                 accuracy                           0.75       971
                macro avg       0.39      0.43      0.37       971
             weighted avg       0.90      0.75      0.81       971
```


```
Evaluando con 5 componentes principales:
Mejores parámetros encontrados: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}

Reporte de clasificación:
                            precision    recall  f1-score   support

                astrometry       0.00      0.00      0.00         1
                   imaging       0.64      0.69      0.67        13
             microlensing        0.13      0.73      0.22        37
orbital brightness modulation       0.00      0.00      0.00         1
              radial vel.       0.83      0.58      0.68       182
                   timing       0.12      0.20      0.15        10
                  transit       0.98      0.81      0.88       727

                 accuracy                           0.75       971
                macro avg       0.39      0.43      0.37       971
             weighted avg       0.90      0.75      0.81       971
```


```
Evaluando con 3 componentes principales:
Mejores parámetros encontrados: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}

Reporte de clasificación:
                            precision    recall  f1-score   support

                astrometry       0.00      0.00      0.00         1
                   imaging       0.50      0.62      0.55        13
             microlensing        0.13      0.76      0.23        37
orbital brightness modulation       0.00      0.00      0.00         1
              radial vel.       0.83      0.55      0.66       182
                   timing       0.10      0.20      0.13        10
                  transit       0.97      0.80      0.88       727

                 accuracy                           0.74       971
                macro avg       0.36      0.42      0.35       971
             weighted avg       0.90      0.74      0.80       971


F1 scores por cantidad de componentes:
Componentes 5: F1 score 0.8093
Componentes 3: F1 score 0.7996

Número óptimo de componentes principales: 5
```

```
Mediana de la confiabilidad con 100 ejecuciones (50/50 split): 0.8342566444593045

Mediana de la confiabilidad con 100 ejecuciones (20/80 split): 0.8308892819167913
```

9. Código 2:

https://github.com/EdyPerez03/proyectoIA/tree/main/cod2