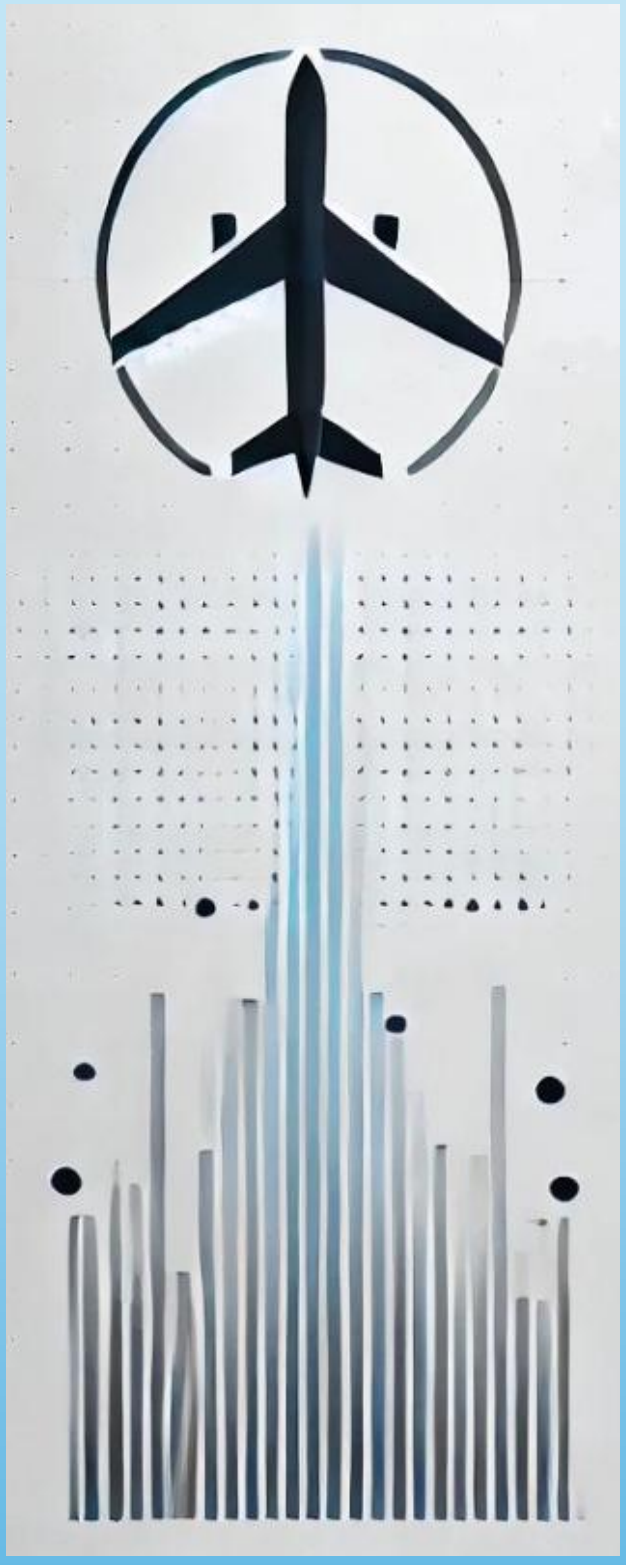


Asignatura: Aprendizaje Automatico

Modelo Predictivo

David Morán Gorgojo
Miguel Sánchez Rodríguez
Pablo Ruíz Morán

Índice

1. Limpieza de datos:

Trabajar con datos reales a menudo implica enfrentarse a problemas como valores faltantes, errores o inconsistencias que pueden afectar negativamente el rendimiento de los modelos. En este proyecto, detectamos valores nulos en varias columnas clave, lo que nos llevó a explorar diferentes estrategias para manejarlos.

A través de este apartado, describimos las técnicas utilizadas para tratar los valores faltantes, evaluamos sus impactos en los datos y justificamos la selección del método final.

Métodos probados para manejar valores nulos

Evaluamos las siguientes técnicas:

1. **Rellenar con la media**
   En este método, los valores nulos se reemplazan por la media de la columna correspondiente. Aunque es fácil de implementar, puede distorsionar la distribución de los datos y no considera las relaciones entre características.

   ```
   df['Arrival Delay in Minutes'].fillna(mean_delay, inplace=True)
   ```

2. **Rellenar con valores promedio de otra columna relacionada**
   Probamos reemplazar los valores nulos en columnas como Arrival Delay in Minutes usando el promedio de Total Delay. Este enfoque aprovecha relaciones lógicas entre columnas, pero su eficacia depende de que estas relaciones sean claras y consistentes, por lo que hace necesario un buen análisis de las relaciones y si este no es correcto se puede distorsionar todo.

   ```
   data.fillna(data['Total Delay'].mean())
   ```

3. **Eliminar filas con valores nulos**
   Optamos temporalmente por eliminar todas las filas con valores faltantes. Si bien este método garantiza que los datos restantes sean completos, puede resultar en una pérdida significativa de información, especialmente en datasets más pequeños o en los que falten muchos datos.

   ```
   df.dropna()
   ```

4. **Usar un modelo de predicción basado en vecinos cercanos (KNNImputer)**
   Finalmente, seleccionamos esta técnica, que utiliza los valores de características similares (según los vecinos más cercanos) para estimar los valores faltantes. Este método no solo considera las relaciones entre variables, sino que también es más robusto y adaptable a diferentes tipos de datos.

```
#1.4 Usando KNNImputer
imputer = KNNImputer(n_neighbors=5)
#buscar que caracteristicas son nulas y rellenamos con el KNNimputer
for col in df_preprocessed_standard.columns:
    if df_preprocessed_standard[col].isnull().sum() > 0:
        df_preprocessed_standard[col] = imputer.fit_transform(df_preprocessed_standard[[col]])
```

Razones para elegir KNNImputer

La decisión final de usar KNNImputer se basó en las siguientes razones:

- Este método aprovecha la información de registros similares para estimar valores, preservando patrones y correlaciones inherentes en los datos.
- Minimiza la introducción de sesgos comparado con métodos más simples como rellenar con la media.
- Es especialmente adecuado en datasets donde hay relaciones claras entre características, como en nuestro caso.
- En nuestras pruebas preliminares, este método mostró un impacto positivo en la calidad del preprocesamiento y los resultados del modelo.

2. Ingeniería de características:

La ingeniería de características es una etapa donde se crean o transforman variables existentes para mejorar la capacidad predictiva del modelo. Este proceso nos permite obtener información más relevante y aprovechar al máximo los datos disponibles. En este proyecto, implementamos diversos métodos para generar nuevas características a partir de las existentes, con el objetivo de enriquecer la representación de los datos.

Nuevas características generadas

Durante la etapa de preprocesamiento, identificamos la posibilidad crear nuevas características relevantes a partir de algunas existentes:

1. **Retrasos totales (Total Delay)**
   Combinamos los retrasos en la salida (Departure Delay in Minutes) y en la llegada (Arrival Delay in Minutes) en una nueva columna llamada Total Delay. Esto lo hicimos al considerar su gran relación entre ambos, ya que si existe un retraso de salida este por lo tanto se va a extender a ser un retraso de llegada.

```
#1. Total Delay
data['Total Delay'] = data['Departure Delay in Minutes'] + data['Arrival Delay in Minutes']
data.drop(columns=['Departure Delay in Minutes', 'Arrival Delay in Minutes'], inplace=True)
```

2. **Grupos de edad (Age Groups)**
   Discretizamos la variable continua Age en categorías que representan rangos de edad (Joven, Adulto Joven, Adulto, Adulto Mayor). Esta segmentación ayuda a identificar patrones específicos en la satisfacción según el rango de edad y así comprender mejor sus relaciones.

```
#2. Age en rangos
ranges = [0, 18, 35, 60, 100]
labels = ['Joven', 'Adulto Joven', 'Adulto', 'Adulto Mayor']

data['Age Groups'] = pd.cut(data['Age'], bins=ranges, labels=labels)
data.drop(columns='Age', inplace=True)
```

3. **Rangos de distancia (Distance Range)**
   Clasificamos la distancia del vuelo (Flight Distance) en categorías (Short, Medium, Long, Very Long, Ultra Long) usando la misma lógica de discretización que en Age Groups. Esto facilita la detección de tendencias relacionadas con trayectos cortos o largos en relación con la satisfacción del cliente.

```
ranges = [0, 500, 1500, 3000, 5000, 10000]
labels = ['Short', 'Medium', 'Long', 'Very Long', 'Ultra Long']

data['Distance Range'] = pd.cut(data['Flight Distance'], bins=ranges, labels=labels)
data.drop(columns='Flight Distance', inplace=True)
```

4. **Índice de confort total (Comfort Total)**
   Creamos una métrica que promedia las puntuaciones de todas las columnas relacionadas con las puntuaciones de los clientes, como Seat comfort, Inflight entertainment, y Leg room service, entre otras. Esta nueva

característica permite evaluar de manera global el nivel de confort ofrecido a los pasajeros, simplificando el análisis y haciendo el dataset mucho más pequeño y manejable.

```
#4. Comfort Total
data['Comfort Total'] = (data['Seat comfort'] + data['Inflight entertainment'] + data['Inflight service'] + data['Leg room service'] +
                         data['On-board service'] + data['Cleanliness'] + data['Food and drink'] + data['Baggage handling'] + data['Checkin service']
                         + data['Inflight wifi service'] + data['Ease of Online booking'] + data['Departure/Arrival time convenient']
                         + data['Gate location'])/13
data.drop(columns= ['Seat comfort', 'Inflight entertainment', 'Inflight service', 'Leg room service', 'On-board service', 'Cleanliness',
                    'Food and drink', 'Baggage handling', 'Checkin service', 'Inflight wifi service', 'Ease of Online booking',
                    'Departure/Arrival time convenient', 'Gate location'], inplace=True)

df_preprocessed_standard = data.copy() #minmax
df_preprocessed_min_max = data.copy() #standard
```

Conclusión

Tras hacer pruebas pudimos ver que la velocidad de ejecución disminuía en gran cantidad, pero por otro lado influían de manera negativa en los resultados, sobre todo la característica Confort Total, que al unir tantas características en una sola hacían que se perdiera detalle e información. Por lo tanto decidimos no contar con ella.

3. Selección de características:

En un conjunto de datos con múltiples variables, no todas contribuyen de manera significativa al objetivo del modelo. Algunas pueden ser irrelevantes, redundantes o incluso generar ruido, afectando negativamente el rendimiento. Por ello, realizamos un análisis para identificar y seleccionar las características más relevantes para el problema de predicción de la satisfacción.

En este apartado, exploramos diferentes técnicas, tanto manuales como automáticas, para reducir la dimensionalidad del conjunto de datos y mejorar la eficiencia del modelo. A continuación, describimos los métodos empleados y los resultados obtenidos, destacando las características clave que tuvieron mayor impacto en las predicciones.

Estrategias empleadas

1. **Análisis exploratorio y correlación**
   En una etapa inicial, calculamos las correlaciones entre las características numéricas para identificar posibles redundancias o relaciones débiles con las variables. Aquellas características con baja correlación con satisfaction o con alta redundancia entre sí fueron candidatas para eliminación.

Durante el análisis, eliminamos características que no aportaban valor al modelo:

- **Columnas redundantes:** Agrupamos variables relacionadas en nuevas características como Comfort Total y eliminamos las originales.

- **Identificadores únicos:** Columnas como id fueron descartadas, ya que no contribuían al análisis.

2. **Técnicas de selección automática** Utilizamos diversos métodos para evaluar y seleccionar las características más relevantes:

    - **SelectKBest**
      Este método selecciona las k mejores características según una métrica estadística. En nuestro análisis, probamos diferentes valores de k para evaluar el impacto en el rendimiento del modelo.

```python
def select_kbest(X:pd.DataFrame, y:np.array, ks:int, sc_func=chi2) -> pd.DataFrame:
    """Realiza la selección de las k mejores características, para un
    dataframe de caracterísitcas y un array de etiqutas."""
    selector = SelectKBest(score_func=sc_func, k=ks)
    x_best = selector.fit_transform(X, y)
    best_columns = X.columns[selector.get_support()]

    return pd.DataFrame(x_best, columns=best_columns)
```

    - **Recursive Feature Elimination (RFE)**
      Usamos RFE con un modelo de regresión logística para seleccionar características mediante un proceso iterativo que elimina las menos importantes. Esto ayudó a identificar combinaciones óptimas de características relevantes.

```python
def select_rfe(X:pd.DataFrame, y:np.array, n_features:int) -> pd.DataFrame:
    """Realiza la selección de las n mejores características usando
    el Recursive Feature Elimination"""
    selector = RFE(estimator=LogisticRegression(), n_features_to_select=n_features)
    x_best = selector.fit_transform(X, y)
    best_columns = X.columns[selector.get_support()]

    return pd.DataFrame(x_best, columns=best_columns)
```

    - **Análisis de Componentes Principales (PCA)**
      Utilizamos PCA para reducir la dimensionalidad transformando las características originales en combinaciones lineales. Esto fue útil para evaluar el impacto de reducir la complejidad sin perder información significativa.

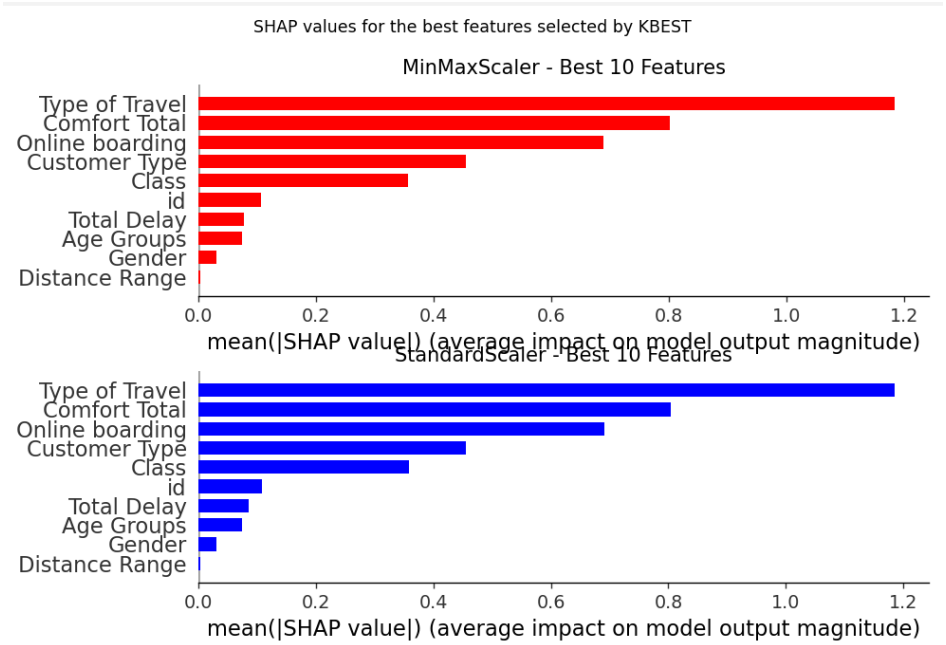
```python
def select_pca(X:pd.DataFrame, n_comp:int) -> pd.DataFrame:
    """Realiza la selección de las n mejores características
    usando PCA"""
    selector = PCA(n_components=n_comp)
    x_best = selector.fit_transform(X)

    return pd.DataFrame(x_best, columns=["PCA{i}" for i in range(1, n_comp+1)])
```

3. **Interpretación con SHAP**
   Para comprender la importancia de las características seleccionadas por los métodos anteriores, utilizamos valores SHAP, que muestran el impacto de cada variable en las predicciones del modelo y así poder ver el número de características más relevantes para poder realizar el modelo.

Resultados

Probamos cada una de estas técnicas con los datos escalados tanto por **Min-Max** como por **StandardScaler**, y evaluamos el impacto en la precisión de un modelo de regresión logística. Los resultados mostraron que:

- **SelectKBest** y **RFE** seleccionaron características más relevantes y mejoraron la interpretabilidad del modelo.
- **PCA** fue útil para reducir dimensionalidad, pero la interpretabilidad de las nuevas componentes lineales fue más limitada.
- Los valores SHAP respaldaron la relevancia de las características seleccionadas, destacando el impacto de variables como Total Delay, Comfort Total, y Age Groups.



4. Transformación de datos:

La transformación de datos es un paso fundamental en el preprocesamiento, especialmente cuando se trabaja con algoritmos que son sensibles a la escala o no pueden manejar directamente variables categóricas. En este proyecto, aplicamos técnicas de escalado y codificación para preparar los datos adecuadamente.

Escalado y normalización

Los modelos de aprendizaje automático, como los basados en distancias (por ejemplo, KNN) o redes neuronales, requieren que las características numéricas tengan una escala similar para evitar que las variables con valores más grandes dominen el modelo. En este proyecto utilizamos dos técnicas:

1. **Escalado Min-Max**
   Este método transforma las características numéricas a un rango entre [0, 1], manteniendo las proporciones originales. Es útil cuando los datos tienen límites conocidos y se desea mantener relaciones relativas entre características.

```python
def transform_data_min_max(df: pd.DataFrame, labels_encoded: dict, scaler:MinMaxScaler) -> pd.DataFrame:
    """
    Transforma las columnas categóricas y numéricas de un DataFrame con un LabelEncoder y los
    escala mediante el escalado MinMax.
    """
    for col, le in labels_encoded.items():
        df[col] = le.transform(df[col]) if df[col].iloc[0] in le.classes_ else -1  # Asignar -1 si es desconocido

    numeric_columns = [col for col in df.columns if col in scaler.feature_names_in_]
    df[numeric_columns] = scaler.transform(df[numeric_columns])

    return df
```

2. **Escalado estándar (StandardScaler)**
   Este método ajusta las características para que tengan media 0 y desviación estándar 1, siendo adecuado para algoritmos que asumen distribuciones gaussianas o trabajan mejor con datos centrados.

```python
def transform_data_standard(df: pd.DataFrame, labels_encoded: dict, scaler:StandardScaler) -> pd.DataFrame:
    """
    Transforma las columnas categóricas y numéricas de un DataFrame con un LabelEncoder y
    realiza el escalado standard.
    """
    for col, le in labels_encoded.items():
        df[col] = le.transform(df[col]) if df[col].iloc[0] in le.classes_ else -1  # Obtengo el valor de la columna y lo transformo

    numeric_columns = [col for col in df.columns if col in scaler.feature_names_in_]
    df[numeric_columns] = scaler.transform(df[numeric_columns])

    return df
```

Pruebas con los distintos tipos de normalizado y técnicas de selección:

```
Standard                                MinMax
Accuracy: 0.8592945479043357            Accuracy: 0.8588133391078389
Train accuracy: 0.854131828735729       Train accuracy: 0.8540957376418079
Test accuracy: 0.8592945479043357       Test accuracy: 0.8588133391078389
Normalization: StandardScaler           Normalization: MinMaxScaler
Selector: SelectKBest(10)               Selector: SelectKBest(10)

Accuracy: 0.8588133391078389            Accuracy: 0.8572734709590492
Train accuracy: 0.8540837072771675      Train accuracy: 0.8529648833656148
Test accuracy: 0.8588133391078389       Test accuracy: 0.8572734709590492
Normalization: StandardScaler           Normalization: MinMaxScaler
Selector: SelectKBest(9)                Selector: SelectKBest(9)

Accuracy: 0.8592945479043357            Accuracy: 0.8588133391078389
Train accuracy: 0.854131828735729       Train accuracy: 0.8540957376418079
Test accuracy: 0.8592945479043357       Test accuracy: 0.8588133391078389
Normalization: StandardScaler           Normalization: MinMaxScaler
Selector: RFE(10)                       Selector: RFE(10)

Accuracy: 0.8590539435060873            Accuracy: 0.858957701746788
Train accuracy: 0.8540837072771675      Train accuracy: 0.8541197983710886
Test accuracy: 0.8590539435060873       Test accuracy: 0.858957701746788
Normalization: StandardScaler           Normalization: MinMaxScaler
Selector: RFE(9)                        Selector: RFE(9)

Accuracy: 0.8592945479043357            Accuracy: 0.858957701746788
Train accuracy: 0.854131828735729       Train accuracy: 0.8540716769125272
Test accuracy: 0.8592945479043357       Test accuracy: 0.858957701746788
Normalization: StandardScaler           Normalization: MinMaxScaler
Selector: PCA(10)                       Selector: PCA(10)
```

Ambos métodos se probaron en los datos transformados. La decisión sobre cuál usar dependió del rendimiento de los modelos, aunque el escalado estándar fue el elegido para los algoritmos finales.

Codificación de variables categóricas

En el conjunto de datos original, muchas columnas contenían variables categóricas (como Gender, Customer Type, Class, etc.). Estas variables no pueden ser procesadas directamente por la mayoría de los algoritmos, por lo que se codificaron en un formato numérico:

1. **Label Encoding**
   Se utilizó un LabelEncoder para transformar las categorías en valores enteros únicos.

```python
#Codificación de variables categóricas
category_columns = [col for col in df_preprocessed_standard.columns
                    if df_preprocessed_standard[col].dtype == 'object' or
                    df_preprocessed_standard[col].dtype == 'category']

for col in category_columns: #Ajustamos el LabelEncoder con todas las categorías posibles en los datos de entrenamiento
    le = LabelEncoder()
    df_preprocessed_standard[col] = le.fit_transform(df_preprocessed_standard[col])
    df_preprocessed_min_max[col] = le.fit_transform(df_preprocessed_min_max[col])
```

2. **Manejo de categorías desconocidas**
   Durante la codificación, nos aseguramos de que cualquier categoría desconocida o que no estuviera presente en los datos de entrenamiento

fuera manejada de manera explícita asignándole un valor distintivo (por ejemplo, -1). Esto reduce el riesgo de errores durante la inferencia.

```
for col, le in labels_encoded.items():
    df[col] = le.transform(df[col]) if df[col].iloc[0] in le.classes_ else -1  # Asignar -1 si es desconocido
```

5. División de datos:

División de Datos

La división de datos es un paso crítico en el flujo de trabajo de aprendizaje automático, ya que permite evaluar la capacidad de generalización del modelo. En este proyecto, empleamos varias estrategias de división para separar los datos en conjuntos de entrenamiento, validación y prueba.

División inicial del conjunto de datos

El conjunto de datos se dividió en **70 % para entrenamiento** y **30 % para prueba**, manteniendo la proporción original de las clases mediante una división estratificada. Esta estrategia garantiza que ambas clases estén representadas de manera equitativa en ambos subconjuntos.

```
data = pd.read_csv('train_students.csv')
train_csv = data.sample(frac=0.7, random_state=42)
data = data.drop(train_csv.index)
test_csv = data.copy()
train_csv.to_csv('train_students2.csv', index=False)
test_csv.to_csv('test_students2.csv', index=False)
```

El **conjunto de entrenamiento** se utilizó para ajustar los modelos, mientras que el **conjunto de prueba** se reservó exclusivamente para evaluar el rendimiento final.

Validación con K-Fold

Para maximizar el uso de los datos de entrenamiento y obtener métricas más representativas, empleamos validación cruzada con K-Fold (5 particiones). Esta técnica divide el conjunto de entrenamiento en cinco subconjuntos:

1. En cada iteración, uno de los subconjuntos se utiliza como conjunto de validación, y los otros cuatro como entrenamiento.
2. Al final, se calculan métricas promedio sobre todas las iteraciones, proporcionando una estimación más robusta del rendimiento del modelo.

```python
def kfold(df:pd.DataFrame, folds:int) -> list:
    """Está función realiza la validación con k-folds,
    dando una lista de tuplas con los dataframes de entrenamiento y test
    para cada fold"""
    df_rand = df.sample(frac=1).reset_index(drop=True)

    return create_folds(df_rand, folds)
```

Conclusión

El enfoque adoptado de división estratificada y validación cruzada permitió obtener modelos bien ajustados y evaluar su rendimiento de manera rigurosa. Esto asegura que los resultados obtenidos sean representativos y generalizables a nuevos datos.

6. Selección de modelo:

La selección del modelo es una etapa crucial para garantizar que el sistema sea capaz de aprender patrones significativos en los datos y realizar predicciones precisas.

Dado que la variable objetivo (satisfaction) es binario, era necesario el uso de modelos de clasificación. Evaluamos cuatro modelos diferentes inicialmente por sus características, seleccionando el que mejor rendimiento que encontramos tras realizar varias pruebas con cada uno.

Modelos probados

1. **Regresión Logística (Logistic Regression)**
    - **Descripción:** Modelo lineal probabilístico que estima la probabilidad de una clase en función de características independientes.
    - **Resultados:**
        - Ventajas: Simplicidad, rapidez en el entrenamiento y facilidad de interpretación.
        - Conclusión: Fue seleccionado como modelo base debido a su buen rendimiento con bajo costo computacional.
2. **Árboles de Decisión (Decision Tree)**
    - **Descripción:** Modelo basado en reglas de decisión que divide los datos iterativamente según criterios óptimos.

- **Resultados:**
    - Ventajas: Fácil de interpretar y manejar datos mixtos.
    - Conclusión: No fue seleccionado debido a su menor rendimiento, analizando demasiado sus datos de entrenamiento dándonos un caso de overfiting.

3. **K-Nearest Neighbors (KNN)**
    - **Descripción:** Algoritmo basado en la proximidad, que clasifica observaciones según las clases de sus vecinos más cercanos.
    - **Resultados:**
        - Ventajas: Simplicidad y buena capacidad para pequeños datasets.
        - Conclusión: Aunque su rendimiento no fue tan malo como el decision tree, su rendimiento fue inferior al de MLP, además de ser un modelo que se puede ver muy afectado por datos erróneos.
4. **Redes Neuronales Multi-Capa (MLP)**
    - **Descripción:** Modelo no lineal que utiliza múltiples capas ocultas para aprender relaciones complejas en los datos.
    - **Resultado:**
        - Conclusión: Elegido como el modelo principal debido a su capacidad para capturar patrones complejos, logrando así el mejor resultado de los 4.

7. Entrenamiento del modelo:

El objetivo principal del entrenamiento es permitir que los modelos aprendan patrones presentes en el conjunto de datos de entrenamiento. Los dos modelos seleccionados, **Redes Neuronales Multi-Capa (MLP)** y **K-Nearest Neighbors (KNN)**, fueron entrenados utilizando las características procesadas y escaladas, asegurando que los datos fueran representativos y preparados para la fase de validación y prueba.

1. Redes Neuronales Multi-Capa (MLP)
Entrenamos una red neuronal diseñada para capturar relaciones complejas en los datos. La configuración utilizada inicialmente fue:

- **Arquitectura:** Tres capas ocultas con tamaños (5, 5, 5).

- **Funciones:** Activación tanh y optimización con Adam (learning_rate=0.001).
- **Control del entrenamiento:** Tolerancia de convergencia 1e-5 y un máximo de 1000 iteraciones.

2. K-Nearest Neighbors (KNN)
KNN clasifica una observación basándose en sus vecinos más cercanos. No realiza un entrenamiento tradicional, sino que almacena el conjunto de entrenamiento y calcula distancias durante la predicción. La configuración utilizada inicialmente fue:

- **Número de vecinos (k):** Selección inicial de k=5
- **Métrica de distancia:** Distancia euclidiana aplicada a datos normalizados.

Validación cruzada durante el entrenamiento

Ambos modelos fueron entrenados utilizando **validación cruzada con K-Fold (k=4)**. Este enfoque permitió:

- Evaluar el rendimiento intermedio del modelo en diferentes particiones del conjunto de entrenamiento.
- Reducir la posibilidad de sobreajuste antes de pasar a la validación con el conjunto de prueba.

8. Validación y ajuste de hiperparámetros:

La validación y ajuste de hiperparámetros son etapas para garantizar que los modelos seleccionados generalicen bien a datos nuevos. Durante esta fase, evaluamos el rendimiento de los modelos **MLP** y **KNN** en el conjunto de prueba y optimizamos sus hiperparámetros clave para mejorar su precisión.

1. Validación del modelo

Utilizamos el conjunto de prueba para evaluar el rendimiento inicial de los modelos entrenados. Este proceso permitió estimar su capacidad de generalización con los hiperparámetros seleccionados inicialmente:

- **Redes Neuronales Multi-Capa (MLP):**
  - Métricas iniciales:
    - Precisión: 85 %.
    - Recall ponderado: 84 %.

        - F1-score ponderado: 84 %.
        - Tasa de error: 15 %.

- **K-Nearest Neighbors (KNN):**
    - Métricas iniciales:
        - Precisión: 83 %.
        - Recall ponderado: 82 %.
        - F1-score ponderado: 82 %.
        - Tasa de error: 17 %.

2. Ajuste de hiperparámetros

Tras comprobar la fiabilidad funcionamiento y valided de los modelos procedimos a maximizar sus resultados configurando sus parámetros.

MLP: Ajustamos los siguientes parámetros:

- **Tamaño de capas ocultas:** partimos de una capa inicial de (5,5,5), tras comprobar su correcto funcionamiento decidimos probar con máximos para ver si mejoraba su funcionamiento o no. Por ello probamos con (20,20,20), (100,100,100),(1000,1000,1000) de forma simultánea. Pudimos observar que cuanto mas grandes eran las capas mas lento era la ejecución y los resultados mejoraban con la de 20 pero con las otras dos empeoraban y nos daban mucho overfiting. Por ello optamos por buscar el más optimo probando o por encima y por debajo de la de 20, con (15,15,15) y (25,25,25). Pudimos observar que la de 25 era la mejor opción por ello probamos con 30 y 26, y estas empeoraron. Una vez vimos que 25 era la mejor probamos a alternar un de 24 en alguna de ellas, llegando a la conclusión que la mejor opción era la de (25,25,24)
- **Funciones de activación:** relu, tanh, logistic, identity. Tras probar las 4 opciones vimos que la mejor de todos con diferencia era la tanh
- **Tasa de aprendizaje:** partimos de un valor inicial de 0,01 y lo fuimos disminuyendo hasta comprobar que la de 0,001 era la más optima.

Resultados del ajuste:

- Tamaño de capas óptimo: (25, 25, 24).
- Activación óptima: tanh.
- Tasa de aprendizaje óptima: 0.001.

KNN: Exploramos distintos valores de k y métricas de distancia:

- **Número de vecinos (k):** 1,3, 5, 7, 9,11.
- **Métricas de distancia:** Euclidiana y Manhattan.

Resultados del ajuste:

- k óptimo: 9.
- Métrica de distancia óptima: Euclidiana.

9. Evaluación de los modelos:

La evaluación de los modelos se centró en analizar su rendimiento en el conjunto de pruebas realizadas en el aprtado anterior. Estas métricas reflejan la eficacia de cada modelo en clasificar correctamente las observaciones y en manejar desequilibrios entre clases.

Métricas de evaluación

- **Precisión:** Proporción de predicciones correctas sobre el total de predicciones realizadas.
- **Recall:** Proporción de elementos positivos correctamente clasificados entre todos los elementos positivos reales.
- **F1-score:** Media armónica entre precisión y recall, útil para datos desequilibrados.
- **Tasa de error:** Proporción de predicciones incorrectas sobre el total de predicciones realizadas.

Comparación de resultados de MLP y KNN

Métrica	MLP	KNN
Precisión	95,87 %	88,22 %
Recall	95,83 %	88,24 %
F1-score	95,82 %	88,18 %
Tasa de error	4,1 %	11,75 %

10. Elección del modelo final

Tras evaluar los modelos entrenados y validados, seleccionamos el **Modelo Final** basado en su rendimiento general en el conjunto de prueba, considerando las métricas precisión, recall, F1-score y tasa de error.

Decisión final

Tras todas las pruebas realizadas, mas algunas mas que no se registraron debido a que por el mal resultado no la almacenábamos, hemos terminado con los resultados vistos en el apartado anterior. Con el que podemos la superioridad del modelo MLP respecto al KNN, consiguiendo un resultado bastante mayor. Por ellos hemos decidió tener al MLP como modelo definitivo respecto al resto.