Universidad Autónoma de Querétaro
división de investigación y posgrado

Facultad de Ingeniería
Maestría en Ciencias en Inteligencia Artificial

Machine Learning

Evaluación 1: Titanic y k-Nearest Neighbor

Estudiante:
Alejandro Daniel
MATÍAS PACHECO

Profesor:
Dr. Marco Antonio
ACEVES FERNÁNDEZ

Querétaro, Qro. 20 de octubre de 2023

Índice

Índice de figuras

1. Introducción

El aprendizaje máquina o machine learning ha progresado gracias a la variedad de algoritmos que se desarrollan constantemente, cada uno con sus características y ventajas únicas. En este contexto, el algoritmo de k-Nearest Neighbor, comúnmente conocido como KNN, representa una herramienta simple y fundamental con una aplicabilidad generalizada. Su importancia en el ámbito de machine learning es de gran alcance, convirtiéndolo en un componente básico del área.

El principio fundamental de KNN gira en torno al concepto del aprendizaje basado en la proximidad. En su base, el algoritmo asume que los puntos de datos que comparten similitudes son más propensos a poseer atributos comunes. Esta noción se basa en la observación de que la proximidad en el espacio de datos denota similitud. Esta simplicidad ha demostrado constantemente su utilidad en diversos campos, desde sistemas de recomendación hasta el reconocimiento de imágenes y una amplia variedad más.

Basado en el concepto intuitivo de proximidad, kNN a día de hoy sigue ofreciendo un enfoque universal para abordar las tareas de clasificación o regresión del mundo real, por lo que es una herramienta sólida y versátil para cualquier científico de datos, razón por la cuál, en este trabajo se presenta el desarrollo de una implementación de kNN, con el fin de realizar clasificación empleando el dataset Titanic, cuyo objetivo es determinar si cierto pasajero sobrevive o no a dicha tragedia, así mismo, se ponen en práctica conceptos abordados en trabajos anteriores de la optativa Machine Learning.

2. Justificación

La importancia de comprender y dominar los algoritmos de machine learning es vital en el mundo actual. Los algoritmos de clasificación, en particular, desempeñan un papel crucial en la toma de decisiones automatizada en una amplia gama de campos, desde la medicina hasta la seguridad y el análisis de datos. Entre estos algoritmos, K-Nearest Neighbors se presenta como un método fundamental y versátil.

En este trabajo, se aborda la necesidad de obtener una comprensión completa de KNN, un algoritmo que ha demostrado su utilidad en numerosas aplicaciones del mundo real. Se abordará este objetivo utilizando la base de datos Titanic, siendo esta una base popular en el aprendizaje máquina que involucra la clasificación de pasajeros como "sobrevivientes.º "no sobrevivientes".

El conjunto de datos del Titanic, con su combinación de atributos categóricos y numéricos, presenta desafíos típicos que los profesionales de la ciencia de datos pueden encontrar en su trabajo diario. A través de este trabajo, no solo se comprenderá cómo funciona KNN, sino también se abordarán cuestiones de preprocesamiento de datos y selección de características. Además, se obtendrá una apreciación más profunda de la importancia de la elección del hiperparámetro "kƶ cómo afecta al rendimiento del modelo.

En última instancia, este trabajo tiene como objetivo abordar las herramientas necesarias para comprender y aplicar KNN en situaciones del mundo real.

3. Desarrollo

A continuación se presenta una descripción de kNN, así como del algoritmo de validación cruzada o crossvalidation, empleada para determinar el número óptimo de k (vecinos), así mismo, se describe la base de datos y se presenta el procesamiento de los datos previo a la clasificación, mostrando el código implementado en una notebook de python.

3.1. K-Nearest Neighbor

A continuación, se presenta una breve explicación de los algoritmos kNN y crossvalidation, así como su implementación en python.

3.1.1. K-Nearest Neighbors

K-Nearest Neighbors (KNN) es un algoritmo de clasificación supervisada que se encuentra en la base de muchos problemas de aprendizaje automático. La premisa fundamental de KNN es que los elementos que comparten características similares tienden a pertenecer a la misma clase. En otras palabras, si observamos a qué clases pertenecen los "vecinos más cercanos"de un punto de datos, podemos predecir la clase de ese punto de datos [1].

El funcionamiento del algoritmo KNN se puede resumir en los siguientes pasos:

- ✓ Selección del valor de k: KNN utiliza un parámetro "k"que representa el número de vecinos más cercanos a considerar. La elección de "k."es crucial y afectará la precisión del algoritmo. En general, un "k"pequeño puede llevar a un modelo más sensible al ruido, mientras que un "k"grande puede suavizar la decisión y llevar a un sesgo bajo.

- ✓ Cálculo de la distancia: KNN utiliza una función de distancia (como la distancia euclidiana o Manhattan) para medir cuán cerca están los puntos de datos entre sí. Cuanto más cercanos estén los puntos, más "similares"serán considerados. Las distancias más utilizadas son las siguientes:

  Distancia Euclidiana:

$$Dist(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

  Distancia Manhattan:

$$Dist(A, B) = \sum_{i=1}^{n} |A_i - B_i|$$

- ✓ Identificación de los k vecinos más cercanos: Una vez que se ha calculado la distancia entre el punto de datos que se va a clasificar y todos los demás puntos en el conjunto de datos, se seleccionan los "k"vecinos más cercanos.

- ✓ Clasificación por votación: KNN realiza una clasificación por mayoría. Los "k"vecinos más cercanos votan por su clase respectiva, y la clase más común se asigna al punto de datos que se clasifica.
- ✓ Predicción: El punto de datos se asigna a la clase con más votos.

A continuación, se presenta el código implementado en python para kNN (Figura 1).

```python
class kNN:
    def __init__(self, k=5, distance='euclidean'):
        self.k = k
        self.distance = distance

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def euclidean_distance(self, x1, x2):
        return np.sqrt(np.sum((x1 - x2) ** 2))

    def manhattan_distance(self, x1, x2):
        return np.sum(np.abs(x1 - x2))

    def minkowski_distance(self, x1, x2, p):
        return np.power(np.sum(np.abs(x1 - x2) ** p), 1/p)

    def predict(self, X_test):
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)

    def _predict(self, x):
        if self.distance == 'euclidean':
            distances = [self.euclidean_distance(x, x_train) for x_train in self.X_train]
        elif self.distance == 'manhattan':
            distances = [self.manhattan_distance(x, x_train) for x_train in self.X_train]
        elif self.distance == 'minkowski':
            distances = [self.minkowski_distance(x, x_train, p=3) for x_train in self.X_train]
        else:
            raise ValueError("Error al ingresar distancia")

        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        assigned_class = np.bincount(k_nearest_labels).argmax()

        return assigned_class
```

Figura 1: Clase KNN para clasificar datos.

El parámetro 'k' es crítico en el algoritmo KNN. Un valor de 'k' bajo (por ejemplo, 1) puede llevar a un modelo propenso al ruido y a valores atípicos, mientras que un valor de 'k' alto puede suavizar la decisión y ocultar patrones importantes.

3.1.2. Crossvalidation para encontrar k óptimo

Una parte crucial en la implementación de KNN es determinar el valor óptimo de 'k'. Para lograrlo, se utiliza la técnica de validación cruzada o crossvalidation, que permite evaluar el rendimiento del modelo para diferentes valores de 'k'. En concreto, se utiliza la validación cruzada k-fold, donde se divide el conjunto de datos en k subconjuntos (folds) y se realiza el proceso de entrenamiento y evaluación k veces, utilizando un subconjunto diferente como conjunto de prueba en cada iteración.

La métrica que se evalúa en cada iteración es la exactitud (accuracy) del modelo en el conjunto de prueba. Luego, se calcula el promedio de estas métricas para cada valor de 'k'. El valor de 'k' que produce la exactitud promedio más alta se considera el valor óptimo. La exactitud (accuracy) se calcula de la siguiente manera: Exactitud = Número de predicciones correctas / Número total de predicciones.

A continuación, se presenta el código implementado en python para crossvalidation (Figura 2).

```python
def get_optimal_k(X, y, k_max, num_folds, distance):

    # Prueba valores de 1 a k_max
    k_vals=list(range(1, k_max))

    # Lista para almacenar los resultados de la validación cruzada
    cv_mean_acc = []

    # Realizar validación cruzada para cada valor de k
    for k in k_vals:
        # Crear un objeto k-NN para k vecinos
        knn_cv = kNN(k=k, distance=distance)

        # Realizar validación cruzada para k-folds
        num_folds = 5
        fold_size = len(X) // num_folds

        accuracies = []

        for i in range(num_folds):

            # Dividir los datos en subconjuntos de entrenamiento y prueba
            start = i * fold_size
            end = (i + 1) * fold_size
            X_train = pd.concat([X.iloc[:start], X.iloc[end:]])
            y_train = pd.concat([y.iloc[:start], y.iloc[end:]])
            X_test = X.iloc[start:end]
            y_test = y.iloc[start:end]

            # Entrenar el modelo (almacenar train_dataset)
            knn_cv.fit(X_train.to_numpy(), y_train.to_numpy())

            # Predecir
            y_pred = knn_cv.predict(X_test.to_numpy())

            # Calcular la precisión de la predicción para el k-fold
            accuracy = np.sum(y_pred == y_test) / len(y_test)
            accuracies.append(accuracy)

        # Calcular la media de las precisiones de la validación cruzada para k actual
        cv_mean_acc.append(np.mean(accuracies))

    plt.figure(figsize=(10, 6))
    plt.plot(k_vals, cv_mean_acc, marker='o', linestyle='-')
    plt.title(f'Crossvalidation para determinar k - Distancia: {distance}')
    plt.xlabel('Valor de k')
    plt.ylabel('Precisión promedio')
    plt.grid(True)
    plt.show()

    # Obtener el valor de k que maximiza la precisión media
    optimal_k = k_vals[cv_mean_acc.index(max(cv_mean_acc))]
    return optimal_k
```

Figura 2: Función de crossvalidation implementada en python.

3.1.3. Métricas

A continuación, se presenta la implementación de distintas métricas en python para la evaluación posterior del rendimiento de KNN (Figura 3).

```python
def metricas(y_pred, y_test):

    accuracy = np.mean(y_pred == y_test)

    # Calcular TP, FP, TN y FN
    TP = np.sum((y_pred == 1) & (y_test == 1))
    FP = np.sum((y_pred == 1) & (y_test == 0))
    TN = np.sum((y_pred == 0) & (y_test == 0))
    FN = np.sum((y_pred == 0) & (y_test == 1))

    # Calcular recall, precisión y puntuación F1
    recall = TP / (TP + FN)
    precision = TP / (TP + FP)
    f1_score = 2 * (precision * recall) / (precision + recall)

    return {
        'Accuracy': accuracy,
        'Recall': recall,
        'Precision': precision,
        'F1-Score': f1_score
    }
```

Figura 3: Métricas de evaluación.

3.2. Dataset: Titanic

El conjunto de datos que utilizado para este trabajo es el Titanic, un recurso popular en el ámbito del aprendizaje automático. Este conjunto de datos contiene información sobre los pasajeros a bordo del RMS Titanic y puede descargarse en https://www.kaggle.com/competitions/titanic/data.

A continuación, se presenta una descripción de los atributos del conjunto de datos: Una muestra del dataset se muestra en la Figura 4.

- ✓ **Survival (Supervivencia)**: Representa si un pasajero sobrevivió (1) o no (0).
- ✓ **Pclass (Clase del Boleto)**: Indica la clase del boleto del pasajero, que puede ser 1ra, 2da o 3ra clase.
- ✓ **Sex (Género)**: El género del pasajero.
- ✓ **Age (Edad)**: La edad del pasajero en años.
- ✓ **SibSp (Número de Hermanos/Cónyuges a Bordo)**: La cantidad de hermanos o cónyuges que el pasajero tenía a bordo del Titanic.
- ✓ **Parch (Número de Padres/Hijos a Bordo)**: La cantidad de padres o hijos que el pasajero tenía a bordo.
- ✓ **Ticket (Número de Boleto)**: El número de boleto del pasajero.

- ✓ **Fare (Tarifa del Pasajero)**: La tarifa que el pasajero pagó por el viaje.
- ✓ **Cabin (Número de Cabina)**: El número de cabina del pasajero.
- ✓ **Embarked (Puerto de Embarque)**: El puerto de embarque del pasajero, que puede ser Cherbourg (C), Queenstown (Q) o Southampton (S).

```
df = pd.read_csv('Titanic.csv')
print("   (Instancias, Atributos): ", df.shape)
df
```

```
   (Instancias, Atributos):  (1309, 12)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	S	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925	S	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	S	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236	8.05	S	NaN	NaN
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9	C105	C

Figura 4: Muestra del dataset Titanic.

3.2.1. Análisis de datos

Como primer paso, se averiguó el tipo de dato de cada atributo en el dataset, encontrando que se dividían en categóricos (8 atributos) y numéricos (todos los demás), por lo que se procedió a analizar la base de datos, con el fin de facilitar el desarrollo posterior y evitar incompatibilidades con las funciones a emplear. En la Figura 5 se presenta el tipo de dato en python. Durante el análisis de los atributos se identificó que el atributo de decisión es 'Survived'.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     1309 non-null   int64
 2   Pclass       1309 non-null   int64
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   object
 5   Age          1309 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   object
 8   Ticket       1309 non-null   object
 9   Fare         1309 non-null   object
 10  Cabin        1068 non-null   object
 11  Embarked     270 non-null    object
dtypes: float64(1), int64(4), object(7)
memory usage: 122.8+ KB
```

Figura 5: Tipo de datos del dataset.

Posteriormente se obtuvo el número de observaciones por atributo empleando pandas, el número de valores únicos por atributo se presentan en la Figura 6.

```
# Observaciones por atributo

obs_vals = {col: df[col].nunique() for col in df_cols}
obs_df = (pd.DataFrame(obs_vals.items(), columns=['Atributo', 'Obs'])).T
obs_df
```

	0	1	2	3	4	5	6	7	8	9	10	11
Atributo	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
Obs	1309	2	3	1307	2	99	8	230	818	282	177	3

Figura 6: Número de observaciones por atributo.

Siguiendo las formulas definidas en trabajos previos, se procedió a realizar el análisis de la base de datos, revisando el número de datos faltantes (Figura 7) y las instancias graficadas por atributo (Figura 8).

```
# Valores faltantes
nan_count = df.isna().sum()
nan_count = pd.DataFrame(nan_count.items(), columns=['Atributo', 'NaN']).T
nan_count
```

	0	1	2	3	4	5	6	7	8	9	10	11
Atributo	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
NaN	0	0	0	0	0	0	0	0	0	0	241	1039

Figura 7: Datos faltantes.

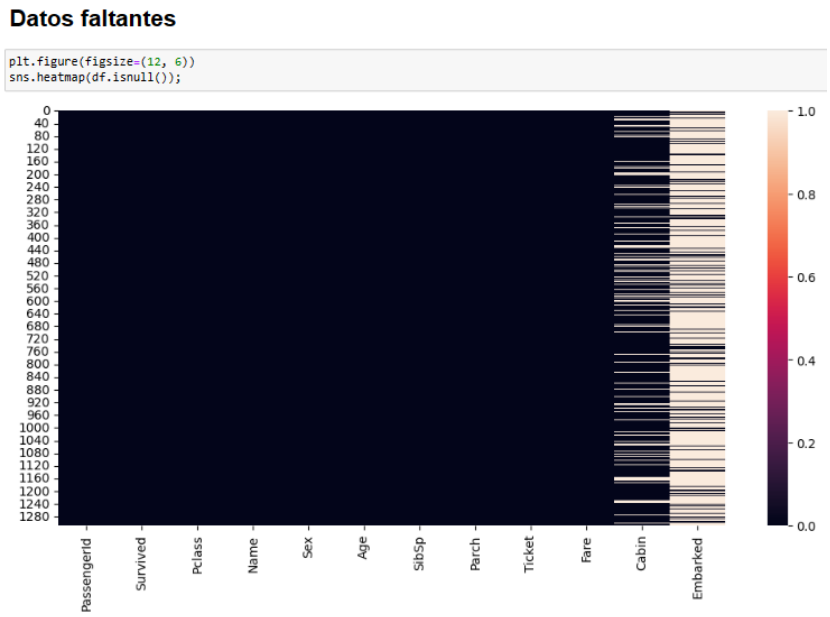

Figura 8: Gráfica de las instancias por atributo.

3.2.2. Pre-procesamiento de datos

Observando el dataset, se identifica que es necesario procesarlo antes de pasar a la estadística descriptiva, ya que presenta problemas con la calidad de datos, por ejemplo, valores atípicos no válidos y errores de cardinalidad.

Analizando los datos, se observa que el atributo "Parch" presenta outliers y outliers no esperados, al igual que "Ticket," "Fare," "Cabin" y "Embarked." Filtrando únicamente los outliers en "Parch," se identifica que dichos valores están presentes en "Ticket," y las instancias en donde "Parch" presenta outliers son las mismas en donde los atributos antes mencionados presentan también outliers. Observando esto, se observa que los outliers en "Parch" son en realidad valores de "Ticket," los outliers en "Ticket" son valores de "Fare," los outliers de "Fare" son valores de "Cabin" y los outliers de "Cabin" son valores de "Embarked." Por lo tanto, se pasarán dichos valores al atributo correcto. Dicho proceso se muestra en las Figuras 9 a 15.

```
df_parch_out = df[(df['Parch'] != '0') & (df['Parch'] != '1')
                & (df['Parch'] != '2') & (df['Parch'] != '3')
                & (df['Parch'] != '4') & (df['Parch'] != '5')
                & (df['Parch'] != '6')]
df_parch_out
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
5	6	0	3	Moran, Mr. James	male	0.0	0	330877	8.4583	Q	NaN	NaN
17	18	1	2	Williams, Mr. Charles Eugene	male	0.0	0	244373	13	S	NaN	NaN
19	20	1	3	Masselmani, Mrs. Fatima	female	0.0	0	2649	7.225	C	NaN	NaN
26	27	0	3	Emir, Mr. Farred Chehab	male	0.0	0	2631	7.225	C	NaN	NaN
28	29	1	3	O'Dwyer, Miss. Ellen "Nellie"	female	0.0	0	330959	7.8792	Q	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
1299	1300	1	3	Riordan, Miss. Johanna Hannah""	female	0.0	0	334915	7.7208	Q	NaN	NaN
1301	1302	1	3	Naughton, Miss. Hannah	female	0.0	0	365237	7.75	Q	NaN	NaN
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236	8.05	S	NaN	NaN
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	359309	8.05	S	NaN	NaN
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	2668	22.3583	C	NaN	NaN

Figura 9: Pre-procesamiento de datos, paso 1.

```
# Copiar dataset original
df_p = df.copy()
```

```
# Pasando datos de "Cabin" a "Embarked
df_p.loc[((df_p['Cabin'] == 'C') | (df_p['Cabin'] == 'S') | (df_p['Cabin'] == 'Q')), 'Embarked'] = df_p['Cabin']
df_p.loc[((df_p['Cabin'] == 'C') | (df_p['Cabin'] == 'S') | (df_p['Cabin'] == 'Q')), 'Cabin'] = ''
df_p
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236	8.05	S	NaN	NaN
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.25		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	359309	8.05	S	NaN	NaN
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	2668	22.3583	C	NaN	NaN

Figura 10: Pre-procesamiento de datos, paso 2.

```
# Pasando datos de "Fare" a "Embarked"
df_p.loc[((df_p['Fare'] == 'C') | (df_p['Fare'] == 'S') | (df_p['Fare'] == 'Q')), 'Embarked'] = df_p['Fare']
df_p.loc[((df_p['Fare'] == 'C') | (df_p['Fare'] == 'S') | (df_p['Fare'] == 'Q')), 'Fare'] = ''
df_p
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236		8.05		NaN	S
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.25		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	359309		8.05		NaN	S
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	2668		22.3583		NaN	C

Figura 11: Pre-procesamiento de datos, paso 3.

```
# Pasando datos de "Fare" a "Cabin"
df_p.loc[(df_p['Fare'].str.contains('[a-zA-Z]')), 'Cabin'] = df_p['Fare']
df_p.loc[(df_p['Fare'].str.contains('[a-zA-Z]')), 'Fare'] = ''
df_p
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236		8.05		NaN	S
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.25		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	359309		8.05		NaN	S
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	2668		22.3583		NaN	C

Figura 12: Pre-procesamiento de datos, paso 4.

```
# Verificar que Fare esté limpio
contiene_letra = df_p[df_p['Fare'].str.contains('[a-zA-Z]')]
contiene_letra
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

```
# Máximo de "Fare"
df_p['Fare'] = (pd.to_numeric(df_p['Fare'], errors='coerce'))
df_p['Fare'].max()
```

```
512.3292
```

Figura 13: Pre-procesamiento de datos, paso 5.

En seguida, se procedió a realizar nuevamente el análisis de la base de datos, revisando el número de datos faltantes (Figura 16) y las instancias graficadas por atributo (Figura 17).

```
# Definir función para pasar datos de Ticket a Fare,
# en el rango 1 a 513 (basándose en el máximo de Fare)

def value_is_fare(val):
    try:
        num_val = float(val)
        # Verificar que el valor esté en el rango de 1 a 513
        if(1 <= num_val <= 513):
            return True
        else:
            return False
    except ValueError:
        # Si no se puede convertir a float, no es un valor válido
        return False

df_p.loc[((df_p['Ticket'].apply(value_is_fare)) & (df_p['Fare'].isna())), 'Fare'] = df_p['Ticket']
df_p.loc[(df_p['Ticket'].apply(value_is_fare)), 'Ticket'] = ''
df_p
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	A.5. 3236		8.05	NaN	S
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.25		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	359309		8.05	NaN	S
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	2668		22.3583	NaN	C

Figura 14: Pre-procesamiento de datos, paso 6.

```
# Definir función para pasar datos de Parch a Ticket,
# para cualquier valor distinto de 0,1,2,3,4,5,6

def value_is_ticket(val):
    try:
        num_val = float(val)
        # Verificar que el valor esté en el rango de 0 a 6
        if(0 <= num_val <= 6):
            return False
        else:
            return True
    except ValueError:
        # Si no se puede convertir a float, es un valor válido
        return True

df_p.loc[((df_p['Parch'].apply(value_is_ticket)) & (df_p['Ticket'].isna())), 'Ticket'] = df_p['Parch']
df_p.loc[(df_p['Parch'].apply(value_is_ticket)), 'Parch'] = ''
df_p
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0			8.0500	NaN	S
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0			8.0500	NaN	S
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1			22.3583	NaN	C

Figura 15: Pre-procesamiento de datos, paso 7.

```
# Valores faltantes
nan_count_p = df_p.isna().sum()
nan_count_p = pd.DataFrame(nan_count_p.items(), columns=['Atributo', 'NaN']).T
nan_count_p
```

	0	1	2	3	4	5	6	7	8	9	10	11
Atributo	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
NaN	0	0	0	0	0	0	0	283	0	10	241	2

Figura 16: Datos faltantes.

```
# Graficar valores faltantes
plt.figure(figsize=(12, 6))
sns.heatmap(df_p.isnull());
```

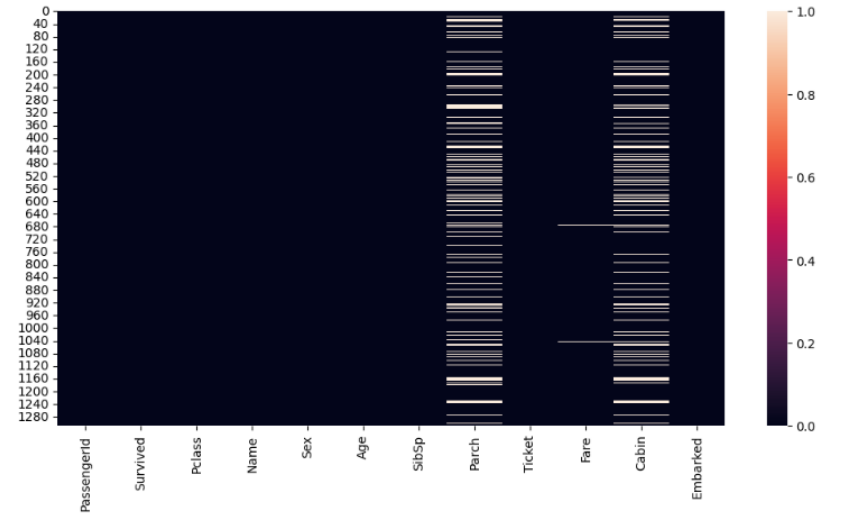

Figura 17: Gráfica de las instancias por atributo.

3.2.3. Estadística descriptiva

Para el subconjunto de datos categóricos, se obtuvo la moda para Sex y Embarked, esto debido a que los otros atributos presentan valores prácticamente únicos o difíciles de representar por medio de una gráfica de moda. Las gráficas se presentan en las Figuras 18 y 19.

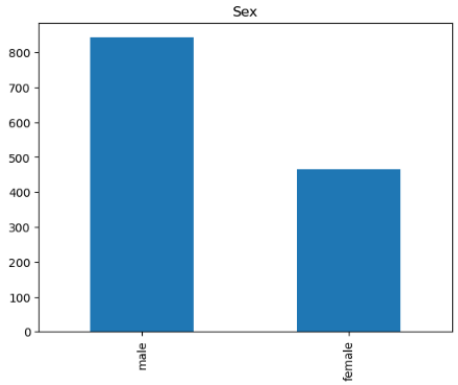

Figura 18: Moda del atributo Sex.

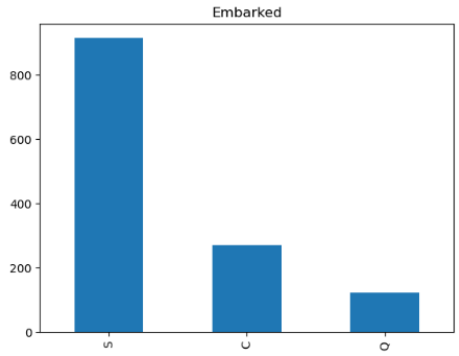

Figura 19: Moda del atributo Embarked.

Usando las funciones de pandas min, max, mean y std en el subconjunto numérico, se calcularon el valor mínimo, máximo, media y desviación estándar por atributo; los resultados se muestran en las Figuras 20, 21, 22 y 23.

```
# Mínimo
min_vals = {col: df_p[col].min() for col in num_cols}
min_df = pd.DataFrame(min_vals.items(), columns=['Atributo', 'Min']).T
min_df
```

	0	1	2	3
Atributo	Age	SibSp	Parch	Fare
Min	0.0	0.0	0.0	0.0

Figura 20: Valores mínimos por atributo.

```
# Máximo
max_vals = {col: df_p[col].max() for col in num_cols}
max_df = pd.DataFrame(max_vals.items(), columns=['Atributo', 'Max']).T
max_df
```

	0	1	2	3
Atributo	Age	SibSp	Parch	Fare
Max	80.0	9.0	6.0	512.3292

Figura 21: Valores máximos por atributo.

```
# Media
mean_vals = {col: df_p[col].mean() for col in num_cols}
mean_df = (pd.DataFrame(mean_vals.items(), columns=['Atributo', 'Mean']).round(2)).T
mean_df
```

	0	1	2	3
Atributo	Age	SibSp	Parch	Fare
Mean	23.97	0.45	0.42	33.53

Figura 22: Media de los datos por atributo.

```
# Desviación estandar
std_vals = {col: df_p[col].std() for col in num_cols}
std_df = (pd.DataFrame(std_vals.items(), columns=['Atributo', 'Std']).round(2)).T
std_df
```

	0	1	2	3
Atributo	Age	SibSp	Parch	Fare
Std	17.47	0.93	0.84	51.86

Figura 23: Desviación estándar de los datos por atributo.

En seguida se realizaron las gráficas de distribución de los datos numéricos, empleando la función histograma de matplotlib y la función norm.pdf de scypy. Los resultados se muestran en las Figuras 24 y 25.

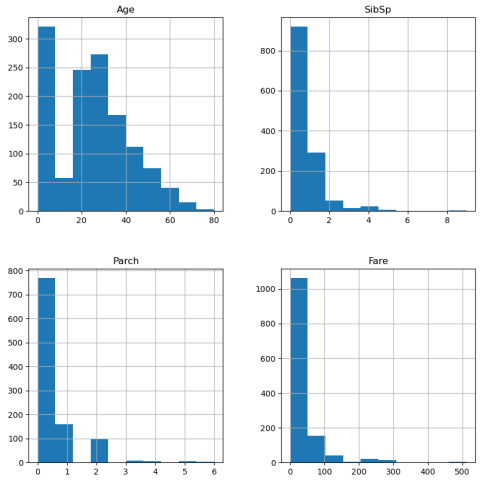

Figura 24: Histograma por atributo.

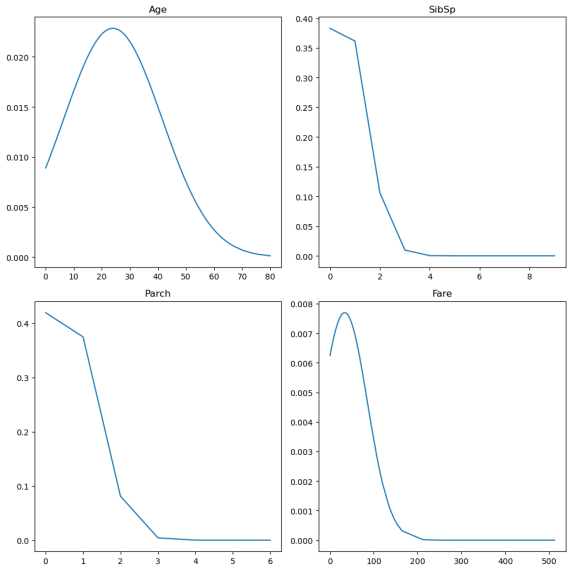

Figura 25: Función de distribución de probabilidad por atributo.

A continuación, empleando matplotlib, se graficó la box plot (26), con el fin de identificar valores atípicos y comparar de mejor manera la distribución de los datos. Es posible observar que los atributos presentan mayor variabilidad en los datos, y que cada uno presenta valores atípicos, por lo que con el fin de conocer el número exacto de éstos, se empleó la función mostrada en la Figura 27, los resultados se muestran en la misma, notando que son un porcentaje pequeño respecto al total.

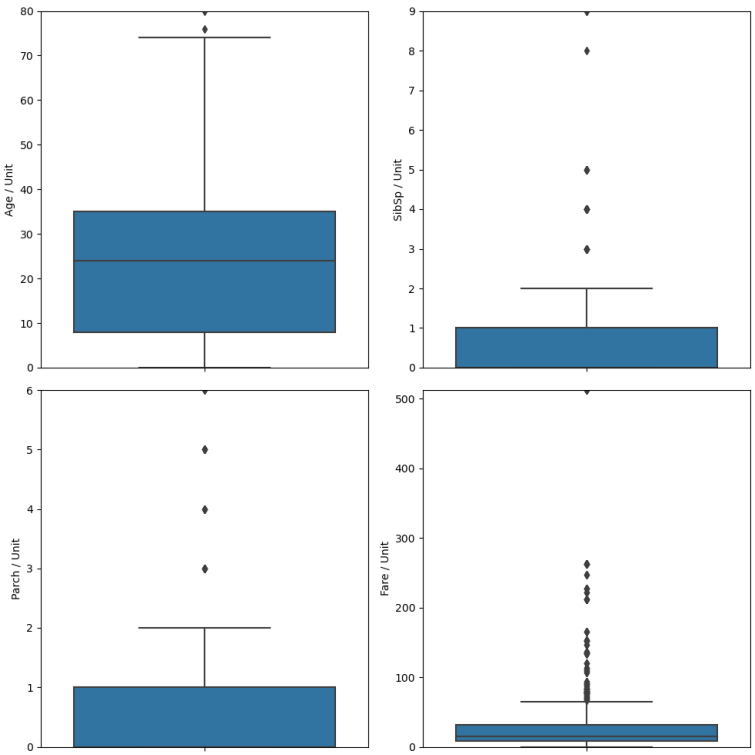

Figura 26: Box plot por atributo.

```
Q1 = df_2.quantile(0.25)
Q3 = df_2.quantile(0.75)
IQR = Q3 - Q1
IQR
```

```
Age      27.0000
SibSp     1.0000
Parch     1.0000
Fare     23.3792
dtype: float64
```

```
outliers = ((df_2 < (Q1 - 1.5 * IQR)) | (df_2 > (Q3 + 1.5 * IQR))).sum()
out_count = pd.DataFrame(outliers.items(), columns=['Atributo', 'Outliers']).T
out_count
```

	0	1	2	3
Atributo	Age	SibSp	Parch	Fare
Outliers	2	48	21	171

Figura 27: Número de valores atípicos por atributo.

3.2.4. Imputación, codificación y normalización

Siguiendo la definición presentada en trabajos anteriores, se desarrolló el algoritmo para imputación por media por clase para el atributo Fare, pues de acuerdo con el análisis de datos faltantes, presenta algunas instancias vacías, los resultados se muestran en la Figura 28 y la gráfica de distribuciones se presenta en la sección de resultados.

```
df_total = df_p.copy()
```

```
# Imputación de media por clase ('Pclass') usando una función Lambda
df_total['Fare'] = df_total.groupby('Pclass')['Fare'].transform(lambda x: x.fillna(x.mean()))
df_total
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0.0	A/5 21171	7.2500		S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0.0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0.0	STON/O2. 3101282	7.9250		S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0.0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0.0	373450	8.0500		S
...	...	...	...	...	...	...	...	...	...	...	...	...
1304	1305	0	3	Spector, Mr. Woolf	male	0.0	0	NaN		8.0500	NaN	S
1305	1306	1	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0.0	PC 17758	108.9000	C105	C
1306	1307	0	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0.0	SOTON/O.Q. 3101262	7.2500		S
1307	1308	0	3	Ware, Mr. Frederick	male	0.0	0	NaN		8.0500	NaN	S
1308	1309	0	3	Peter, Master. Michael J	male	1.0	1	NaN		22.3583	NaN	C

Figura 28: Algoritmo y dataset resultante de imputación por media por clase.

Con el fin de facilitar la clasificación, SibSp y Parch se codificarán como categóricos, pues lo importante es si llevaban o no algún familiar con ellos, es decir, se implementará One-Hot basándose en la definición de trabajos previos, los resultados se muestran en las Figuras 29, 30.

```
# Con el fin de facilitar la clasificación, SibSp y Parch
# se codificarán como categóricos, pues lo importante es
# si llevaban o no algún familiar con ellos
num_att = ['Age', 'Fare']
cat_att = ['Pclass', 'Sex', 'SibSp', 'Parch']
label = ['Survived']
```

```
data = df_total[num_att + cat_att + label]
data
```

	Age	Fare	Pclass	Sex	SibSp	Parch	Survived
0	22.0	7.2500	3	male	1	0.0	0
1	38.0	71.2833	1	female	1	0.0	1
2	26.0	7.9250	3	female	0	0.0	1
3	35.0	53.1000	1	female	1	0.0	1
4	35.0	8.0500	3	male	0	0.0	0
...	...	...	...	...	...	...	...
1304	0.0	8.0500	3	male	0	NaN	0
1305	39.0	108.9000	1	female	0	0.0	1
1306	38.5	7.2500	3	male	0	0.0	0
1307	0.0	8.0500	3	male	0	NaN	0
1308	1.0	22.3583	3	male	1	NaN	0

Figura 29: Normalización categórica: uno-a-n.

Finalmente, para los atributos Fare y Sex, se realiza la normalización en el rango 0 a 1, con el fin de evitar que al clasificar los datos éstos atributos tomen una mayor importancia, los resultados se muestran en la Figura 31.

	Age	Fare	Survived	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_True	SibSp_False	SibSp_Unknown	Parch_True	Parch_False	Par
0	22.0	7.2500	0	0	0	1	0	1	1	0	0	1	1	
1	38.0	71.2833	1	1	0	0	1	0	1	0	0	1	1	
2	26.0	7.9250	1	0	0	1	1	0	1	1	0	1	1	
3	35.0	53.1000	1	1	0	0	1	0	1	0	0	1	1	
4	35.0	8.0500	0	0	0	1	0	1	1	1	0	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	0.0	8.0500	0	0	0	1	0	1	1	1	0	0	0	
1305	39.0	108.9000	1	1	0	0	1	0	1	1	0	1	1	
1306	38.5	7.2500	0	0	0	1	0	1	1	1	0	1	1	
1307	0.0	8.0500	0	0	0	1	0	1	1	1	0	0	0	
1308	1.0	22.3583	0	0	0	1	0	1	1	0	0	0	0	

Figura 30: Dataset resultante de la normalización categórica: uno-a-n.

```
# Normalizar datos numéricos
data['Age'] = min_max_normalization(data['Age'])
data['Fare'] = min_max_normalization(data['Fare'])
data
```

	Age	Fare	Survived	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_True	SibSp_False	SibSp_Unknown	Parch_True	Parch_False
0	0.27500	0.014151	0	0	0	1	0	1	1	0	0	1	1
1	0.47500	0.139136	1	1	0	0	1	0	1	0	0	1	1
2	0.32500	0.015469	1	0	0	1	1	0	1	1	0	1	1
3	0.43750	0.103644	1	1	0	0	1	0	1	0	0	1	1
4	0.43750	0.015713	0	0	0	1	0	1	1	1	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	0.00000	0.015713	0	0	0	1	0	1	1	1	0	0	0
1305	0.48750	0.212559	1	1	0	0	1	0	1	1	0	1	1
1306	0.48125	0.014151	0	0	0	1	0	1	1	1	0	1	1
1307	0.00000	0.015713	0	0	0	1	0	1	1	1	0	0	0
1308	0.01250	0.043640	0	0	0	1	0	1	1	0	0	0	0

Figura 31: Normalización min-max.

3.2.5. Preparación de datos: Train y Test set

En este punto, el dataset está listo para pasar a la etapa de clasificación con kNN. En primer lugar se separan los atributos a clasificar del atributo de desición y a la vez, éstos se separan en subconjuntos de prueba y entrenamiento, esto con apoyo de la librería train_test_split() de sklearn. El proceso se muestra en las Figuras 32, 33 y 34.

```
# Separar las características (X) y las etiquetas (y)
X = data.drop('Survived', axis=1)
X
```

	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_True	SibSp_False	SibSp_Unknown	Parch_True	Parch_False	Parch_Unk
0	0.27500	0.014151	0	0	1	0	1	1	0	0	1	1	
1	0.47500	0.139136	1	0	0	1	0	1	0	0	1	1	
2	0.32500	0.015469	0	0	1	1	0	1	1	0	1	1	
3	0.43750	0.103644	1	0	0	1	0	1	0	0	1	1	
4	0.43750	0.015713	0	0	1	0	1	1	1	0	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	0.00000	0.015713	0	0	1	0	1	1	1	0	0	0	
1305	0.48750	0.212559	1	0	0	1	0	1	1	0	1	1	
1306	0.48125	0.014151	0	0	1	0	1	1	1	0	1	1	
1307	0.00000	0.015713	0	0	1	0	1	1	1	0	0	0	
1308	0.01250	0.043640	0	0	1	0	1	1	0	0	0	0	

Figura 32: Separación de las características (X) y las etiquetas (y).

```
# Separar las características (X) y las etiquetas (y)
y = data['Survived']
y
```

```
0       0
1       1
2       1
3       1
4       0
       ..
1304    0
1305    1
1306    0
1307    0
1308    0
Name: Survived, Length: 1309, dtype: int64
```

Figura 33: Separación de las características (X) y las etiquetas (y).

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

Figura 34: División en subconjuntos de entrenamiento y prueba.

En seguida se realiza la elección de k empleando crossvalidation y posteriormente se realiza la clasificación empleando la clase kNN previamente definida, dicho proceso se muestra en la sección de Resultados.

4. Resultados

En las Figuras 35, 36 se presentan las gráficas obtenidas al evaluar distintos valores de k con crossvalidation, usando las distancias Manhattan y Euclidiana, en ambos casos el valor de k óptimo es 7.

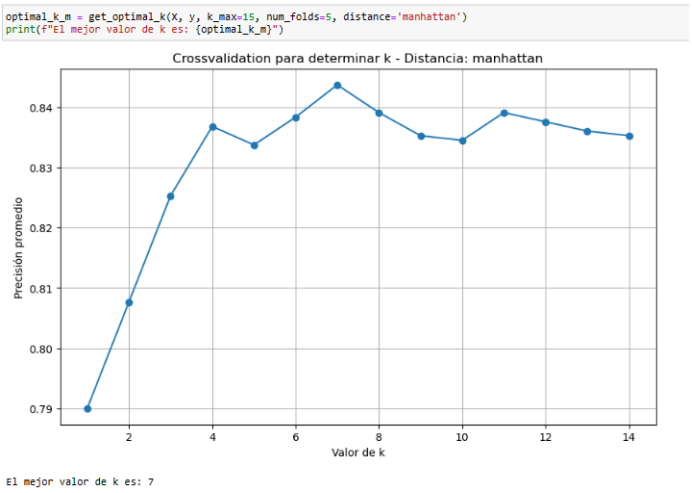

Figura 35: Gráfica obtenida al evaluar k con el algoritmo de crossvalidation para distancia Manhattan.

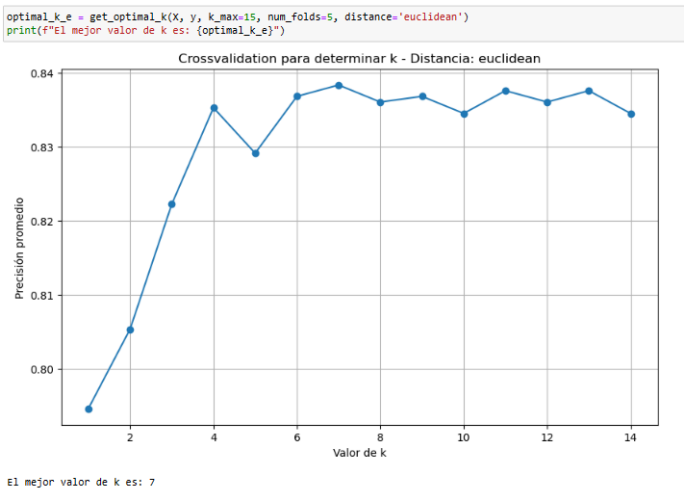

Figura 36: Gráfica obtenida al evaluar k con el algoritmo de crossvalidation para distancia Euclidiana.

En seguida, se crean objetos de tipo kNN para entrenar y clasificar los datos usando los subconjuntos de prueba y entrenamiento, empleando la distancia Manhattan y Euclidiana respectivamente. Los resultados se muestran en las Figuras 37 a 42.

```
# Crear objeto y entrenar modelo k-NN
knn_model_m = kNN(k=optimal_k_m, distance='manhattan')
knn_model_m.fit(X_train.to_numpy(), y_train.to_numpy())
```

```
# Realizar predicciones en el conjunto de prueba
y_pred_m = knn_model_m.predict(X_test.to_numpy())
y_pred_m
```

```
array([0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1],
      dtype=int64)
```

Figura 37: Entrenamiento y predicción del modelo de kNN empleando distancia Manhattan.

```
# Calcular la exactitud del modelo
accuracy_m = np.sum(y_pred_m == y_test) / len(y_test)
print(f'Exactitud del modelo kNN: {accuracy}')
```

```
Exactitud del modelo kNN: 0.9348659003831418
```

```
eval_m = metricas(y_pred_m, y_test)
print(eval_m)
```

```
{'Accuracy': 0.8625954198473282, 'Recall': 0.7651006711409396, 'Precision': 0.8571428571428571, 'F1-Score': 0.8085106382978723}
```

Figura 38: Evaluación del modelo de distancia Manhattan usando las métricas definidas en Desarrollo.

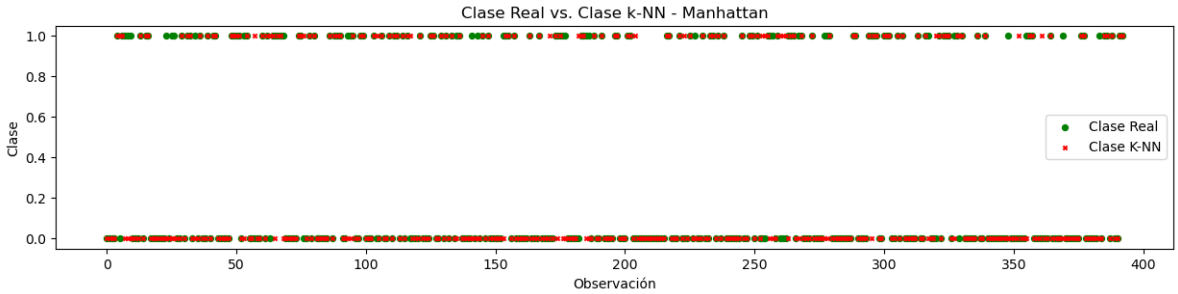

Figura 39: Comparación de la clase real vs la clase predicha por kNN-distancia Manhattan.

Se observa que el valor óptimo de k, utilizando tanto la distancia Manhattan como la distancia Euclidiana, resultó ser 7. Esto implica que considerar los siete vecinos más cercanos es eficaz en ambas métricas de distancia. La elección de k es fundamental en el algoritmo k-NN, ya que influye directamente en el equilibrio entre sesgo y varianza del modelo.

En cuanto al rendimiento del clasificador k-NN, se aprecia una diferencia significativa entre el uso de la distancia Manhattan y la distancia Euclidiana. La distancia Manhattan superó en rendimiento a la distancia Euclidiana, logrando una exactitud del 93 %, mientras que la distancia Euclidiana alcanzó una exactitud del 86 %. Esta disparidad puede deberse a

```
# Crear objeto y entrenar modelo k-NN
knn_model_e = kNN(k=optimal_k_e, distance='euclidean')
knn_model_e.fit(X_train.to_numpy(), y_train.to_numpy())
```

```
# Realizar predicciones en el conjunto de prueba
y_pred_e = knn_model_e.predict(X_test.to_numpy())
y_pred_e
```

```
array([0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1],
      dtype=int64)
```

Figura 40: Entrenamiento y predicción del modelo de kNN empleando distancia Euclidiana.

```
# Calcular la exactitud del modelo
accuracy_e = np.sum(y_pred_e == y_test) / len(y_test)
print(f'Exactitud del modelo kNN: {accuracy_e}')
```

```
Exactitud del modelo kNN: 0.8600508905852418
```

```
eval_e = metricas(y_pred_e, y_test)
print(eval_e)
```

```
{'Accuracy': 0.8600508905852418, 'Recall': 0.7651006711409396, 'Precision': 0.8507462686567164, 'F1-Score': 0.8056537102473499}
```

Figura 41: Evaluación del modelo de distancia Euclidiana usando las métricas definidas en Desarrollo.

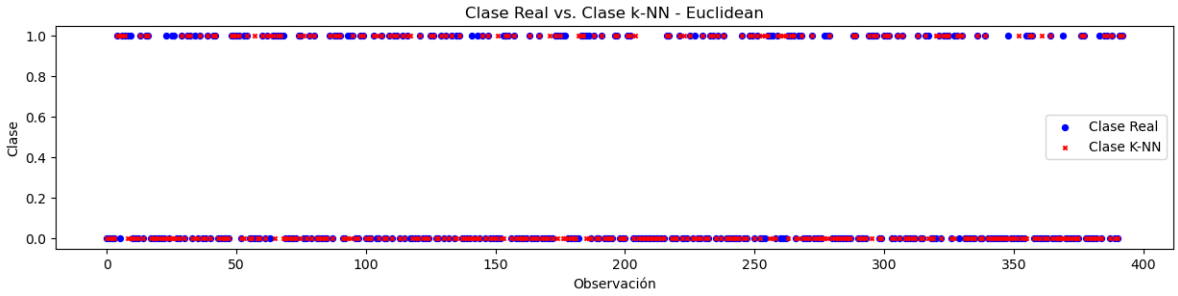

Figura 42: Comparación de la clase real vs la clase predicha por kNN-distancia Euclidiana.

las particularidades de los datos y a cómo cada métrica de distancia los interpreta.

Una posible explicación para esta diferencia podría relacionarse con la naturaleza de los datos y la forma en que las medidas de distancia calculan la similitud entre observaciones. La distancia Manhattan tiende a ser más resistente a los datos atípicos y es sensible a las diferencias en los valores de atributos individuales. Por otro lado, la distancia Euclidiana considera todas las dimensiones de los atributos y es más sensible a la magnitud de las diferencias en todas las dimensiones. En este conjunto de datos, la distancia Manhattan pudo haber sido más adecuada debido a su capacidad para manejar atributos codificados en one-hot, como

Pclass, Sex, SibSp y Parch.

La normalización de los atributos Sex y Fare entre 0 y 1 puede haber contribuido al mejor rendimiento del modelo empleando Manhattan. La normalización asegura que todas las características tengan el mismo rango, evitando que una característica con una escala mayor domine la contribución a la distancia, estando a la vez en el rango de valores obytenidos al codificar con one-hot.

5. Conclusión

La elección de la medida de distancia y el valor de k óptimo son elementos críticos en el éxito de un clasificador k-NN. En este trabajo, se encontró que la distancia Manhattan con un valor de k igual a 7 proporciona el mejor rendimiento, con una precisión del 93 %. La normalización de los atributos y la codificación one-hot de características categóricas también influyeron en el rendimiento del modelo. Estos resultados subrayan la importancia de comprender las características específicas de los datos y seleccionar cuidadosamente los parámetros del algoritmo para lograr un rendimiento óptimo.

Referencias

[1] M.A.A. Fernández. *Inteligencia artificial para programadores con prisa*. UNIVERSO DE LETRAS. Universo de Letras, 2022. ISBN: 9788418856723. URL: https://books.google.com.mx/books?id=ieFYEAAAQBAJ.