Instituto Politécnico Nacional
Escuela Superior de Cómputo


**Integrante:**

Torres Abonce Luis Miguel
Salazar Carreón Jeshua Jonatan

**Grupo:**

6CV1

**Materia:**

Machine Learning

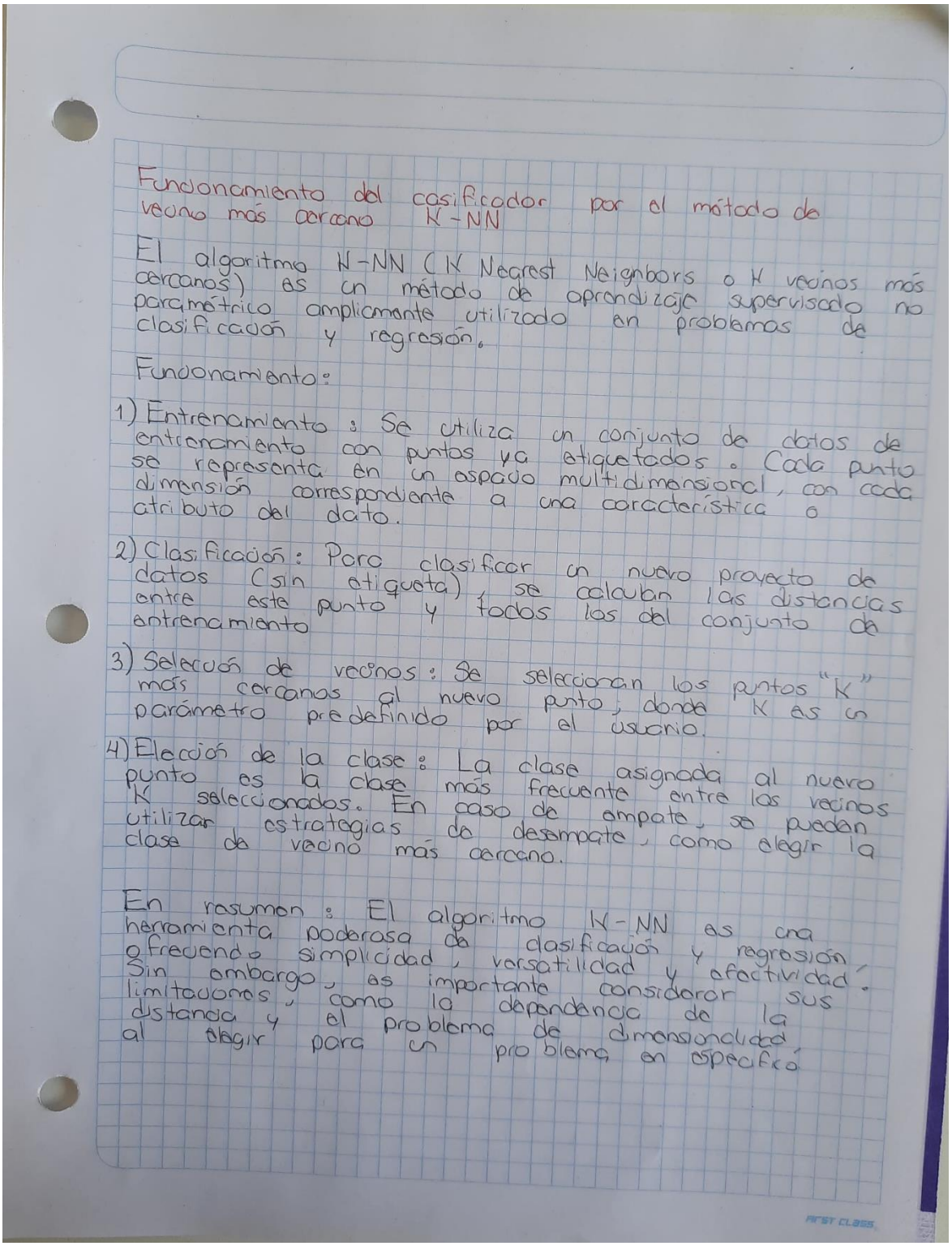
**Profesor:**

Camacho Vazquez Vanessa Alejandra

**Practica 3**

Clasificador por el método del vecino más cercano KNN

Marco teórico.



Funcionamiento del casificador por el método de vecino más cercano K-NN

El algoritmo K-NN (K Nearest Neighbors o K vecinos más cercanos) es un método de aprendizaje supervisado no paramétrico ampliamente utilizado en problemas de clasificación y regresión.

Funcionamiento:

1) Entrenamiento: Se utiliza un conjunto de datos de entrenamiento con puntos ya etiquetados. Cada punto se representa en un espacio multidimensional, con cada dimensión correspondiente a una característica o atributo del dato.

2) Clasificación: Para clasificar un nuevo proyecto de datos (sin etiqueta), se calculan las distancias entre este punto y todos los del conjunto de entrenamiento

3) Selección de vecinos: Se seleccionan los puntos "K" más cercanos al nuevo punto, donde K es un parámetro predefinido por el usuario.

4) Elección de la clase: La clase asignada al nuevo punto es la clase más frecuente entre los vecinos K seleccionados. En caso de empate, se pueden utilizar estrategias de desempate, como elegir la clase de vecino más cercano.

En resumen: El algoritmo K-NN es una herramienta poderosa de clasificación y regresión, ofreciendo simplicidad, versatilidad y efectividad. Sin embargo, es importante considerar sus limitaciones, como la dependencia de la distancia y el problema de dimensionalidad al elegir para un problema en específico

## Ventajas del clasificador KNN (Vecino Más Cercano)

1) Simplicidad: El algoritmo KNN es conceptualmente simple y fácil de entender, lo que lo hace accesible para principiantes.

2) Versatilidad: Puede utilizarse para tareas de clasificación y regresión sin necesidad de modificar el algoritmo.

3) Robustez: No se ve afectado por valores atípicos o outliers en los datos de entrenamiento.

4) Adaptabilidad: No realiza suposiciones a priori sobre la distribución de los datos, lo que permite adaptarse a una amplia variedad de conjunto de datos.

5) Eficiencia: Para conjuntos de datos pequeños o medianos, KNN puede ser computacionalmente eficiente cuando se implementa con técnicas de aproximación.

## Desventajas del clasificador KNN (Vecino más cercano)

1) Alta dimensionalidad: El rendimiento puede degradarse significativamente en conjuntos de datos con alta dimensionalidad, debido al aumento exponencial entre puntos.

2) Sensibilidad a la escala: Las características con mayor rango de valores pueden dominar las distancia, afectando la clasificación de puntos en regiones con valores atípicos.

3) Costo computacional: Para conjunto de datos grandes, la búsqueda de los vecinos más cercano puede ser costoso, especialmente en implementaciones sin aproximación.

4) Memorización: KNN no genera un modelo explícito, sino que memoriza los datos de entrenamiento lo que limita su capacidad de generalización a nuevos datos.

Aplicación reciente KNN: Detección de cáncer de mama

Estudio: "Clasificación del cáncer de mama mediante algoritmos de minería de datos".

Metodología: Se emplearon dos métodos KNN:

- KNN clásico: Se utilizó la distancia euclídea para calcular la similitud entre las imágenes y se asignó la clase de la mayoría de los vecinos cercanos.
- KNN ponderado: Se asignaron pesos a cada vecino en función de su distancia a la imagen objetivo, dando mayor importancia a los vecinos más cercanos.

Resultados: Ambos métodos de KNN obtuvieron resultados prometedores, con una precisión superior al 90% en la clasificación. El KNN ponderado mostró un mejor rendimiento en general, especialmente en la detección de casos de cáncer de mama de bajo grado.

Procedimiento

El conjunto de datos de entrada que se creó está relacionado con la música y consta de cuatro características: 'Duración de la Canción', 'Género Musical', 'Número de Artistas' y 'Número de Reproducciones'. Estos datos se generaron de manera sintética y se guardaron en un archivo CSV llamado "datos_musica.csv". El 'Género Musical' se representa como un número entero de 0 a 3, cada uno representando un género musical diferente. La 'Duración de la Canción' se mide en segundos, el 'Número de Artistas' es un conteo de cuántos artistas contribuyeron a la canción y el 'Número de Reproducciones' es un conteo de cuántas veces se ha reproducido la canción.

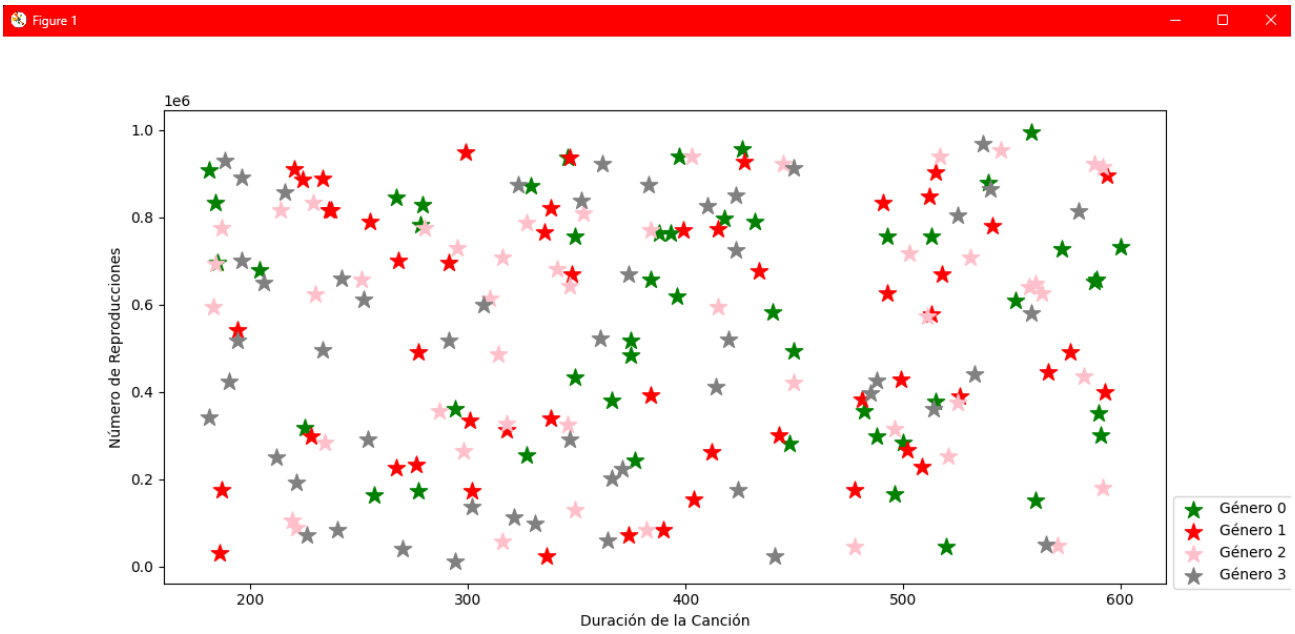
El clasificador K-Nearest Neighbors (KNN) que se utilizará tiene la tarea de clasificar una canción en uno de los cuatro géneros musicales basándose en sus características. El clasificador considerará la 'Duración de la Canción' y el 'Número de Reproducciones' para hacer esta clasificación. Se entrenará al clasificador con el conjunto de datos y luego se utilizará la validación cruzada para evaluar su rendimiento. Se calcularán varias métricas, incluyendo la precisión, el recall y el F-score, para tener una comprensión completa de cómo el clasificador está funcionando.

```python
import numpy as np

# Generar datos sintéticos para 'Duración de la canción', 'Género musical', 'Número de artistas' y 'Número de reproducciones'
duracionCancion = np.random.randint(180, 600 + 1, 200) # Duración de la canción de 180 segundos a 600 segundos
generoMusical = np.random.randint(0, 4, 200) # Género musical, 0 a 4 representando 5 géneros diferentes
numeroArtistas = np.random.randint(1, 5 + 1, 200) # Número de artistas, de 1 a 5
numeroReproducciones = np.random.randint(1000, 1000000 + 1, 200) # Número de reproducciones, de 1000 a 1000000

# Guardar los datos en un archivo de texto
with open('datos_musica.csv', 'w') as archivo:
    archivo.write("Duración de la Canción,Género Musical,Número de Artistas,Número de Reproducciones\n")
    for i in range(200):
        archivo.write(f"{duracionCancion[i]},{generoMusical[i]},{numeroArtistas[i]},{numeroReproducciones[i]}\n")
```

Estos son los datos que se generan:


| Duraci◆n de la | G◆nero Musica | N◆mero de Ar | N◆mero de Re |
|---|---|---|---|
| 371 | 3 | 3 | 224260 |
| 440 | 0 | 3 | 582765 |
| 233 | 1 | 5 | 888361 |
| 513 | 0 | 1 | 756095 |
| 301 | 1 | 4 | 335351 |
| 349 | 2 | 4 | 130483 |
| 418 | 0 | 3 | 796165 |
| 450 | 0 | 1 | 493237 |
| 327 | 0 | 4 | 254720 |
| 277 | 1 | 4 | 490188 |
| 594 | 1 | 5 | 896355 |

Después se generó una gráfica de dispersión de 2 dimensiones, también contamos la cantidad de objetos en cada clase:




```python
for genero in range(4):
    cantidad = len(musica[musica["Género Musical"]==genero])
    print(f"Cantidad de canciones en el género {genero}: {cantidad}")
```

```
Cantidad de canciones en el género 0: 49
Cantidad de canciones en el género 1: 52
Cantidad de canciones en el género 2: 48
Cantidad de canciones en el género 3: 51
```

Posteriormente realizamos el escalado de datos, los datos escalados:

```
[[0.45346062 0.21661979]
 [0.61813842 0.5807643 ]
 [0.12410501 0.89116751]
 [0.79236277 0.75682089]
 [0.28639618 0.32945832]
 [0.40095465 0.12136762]
 [0.56563246 0.79752121]
 [0.64200477 0.48982797]
 [0.34844869 0.24755895]
 [0.22911695 0.486731  ]
 [0.98568019 0.89928726]
 [0.92124105 0.44177375]
 [0.7398568  0.83593005]
```

Posteriormente realizamos la construcción de un clasificador por el método del vecino más cercano K-NN con valor K=3 introducimos los datos escalados, y lo evaluamos utilizando la técnica de validación cruzada técnica de validación cruzada técnica de validación cruzada técnica de validación cruzada de 10 iteraciones y las medidas accuracy, recall y F1/F-score.

Código:

```
# 10 Iteraciones
# Definir las métricas que deseas calcular
scoring = ['accuracy', 'precision_macro','recall_macro', 'f1_macro']
scores10 = cross_validate(clasificador, datos, clase, cv=10, scoring=scoring)
# k = 3
print('k = 3')
print('------Metricas-----')
print("Accuracy:", scores10['test_accuracy'].mean())
print("Precision:", scores10['test_precision_macro'].mean())
print("Recall:", scores10['test_recall_macro'].mean())
print("F-score:", scores10['test_f1_macro'].mean())
print('-----Scores10-----')
print(scores10.keys())
print(scores10)
```

Consola:

```
k = 3
------Metricas-----
Accuracy: 0.255
Precision: 0.22810876623376625
Recall: 0.2541666666666667
F-score: 0.21979140742298636
-----Scores10-----
dict_keys(['fit_time', 'score_time', 'test_accuracy', 'test_precision_macro', 'test_recall_macro', 'test_f1_macro'])
{'fit_time': array([0.00099993, 0.        , 0.0010016 , 0.        , 0.00050759,
       0.00663972, 0.        , 0.00103498, 0.00144076, 0.00132871]), 'score_time': array([0.00550652, 0.00600982, 0.00473857, 0.00703025, 0.0007441 ,
       0.00150347, 0.01109505, 0.00577378, 0.00573945, 0.00524831]), 'test_accuracy': array([0.25, 0.2 , 0.25, 0.15, 0.3 , 0.3 , 0.3 , 0.3 , 0.3 , 0.2 ]), 'test_precision_macro': array([0.21964286,
0.18888889, 0.17424242, 0.16071429, 0.275     ,
       0.18253968, 0.41964286, 0.17708333, 0.33333333, 0.15      ]), 'test_recall_macro': array([0.25      , 0.19166667, 0.25      , 0.15      , 0.3       ,
       0.3       , 0.3       , 0.3       , 0.3       , 0.2       ]), 'test_f1_macro': array([0.23344017, 0.17938312, 0.1875    , 0.12406015, 0.28636364,
       0.22619048, 0.28525641, 0.22252747, 0.28652597, 0.16666667])}
```

Posteriormente construimos otro clasificador por el método del vecino más cercano K-NN con valor K=15 (raíz cuadra de 200 objetos/instancias) introduciendo los datos escalados, y lo evaluamos utilizando la técnica de validación cruzada de 10 iteraciones y las medidas accuracy, recall y F1/F-score.

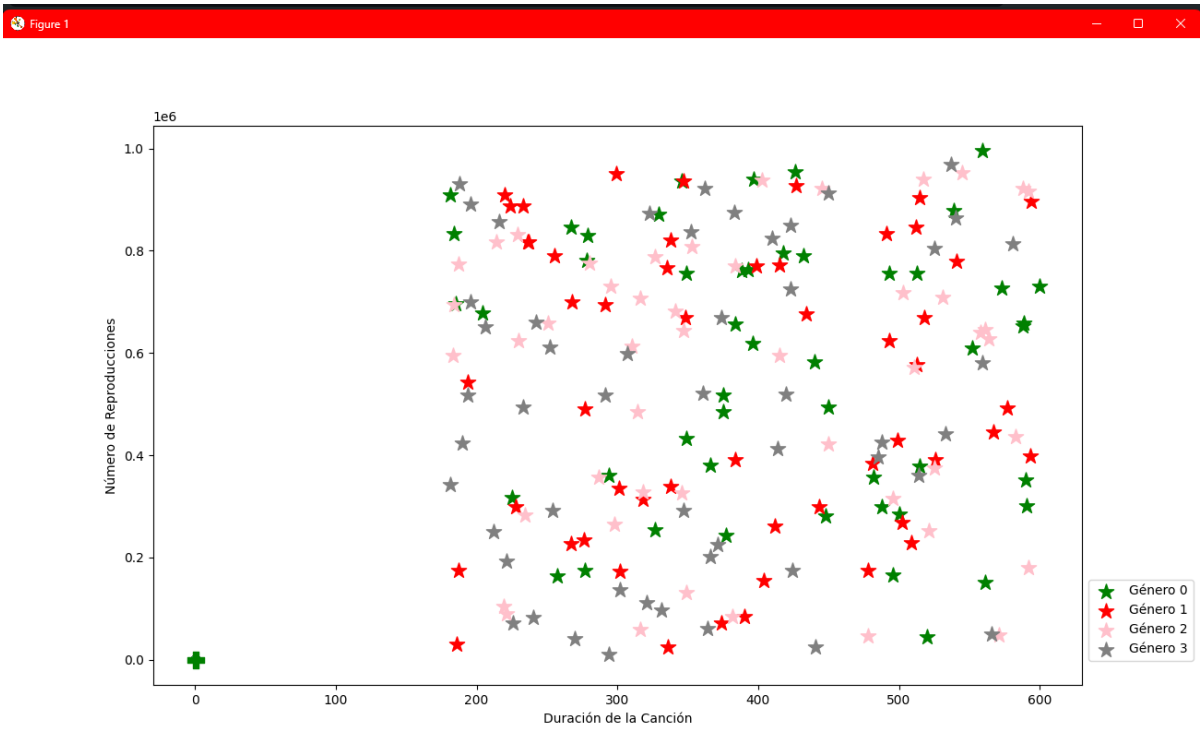
```python
# 10 Iteraciones
# Definir las métricas que deseas calcular
scoring = ['accuracy', 'precision_macro','recall_macro', 'f1_macro']
scores10 = cross_validate(clasificador15, datos, clase, cv=10, scoring=scoring)
# k = 15
print('k = 15')
print('------Metricas-----')
print("Accuracy:", scores10['test_accuracy'].mean())
print("Precision:", scores10['test_precision_macro'].mean())
print("Recall:", scores10['test_recall_macro'].mean())
print("F-score:", scores10['test_f1_macro'].mean())
print('-----Scores10-----')
print(scores10.keys())
print(scores10)
```

```
k = 15
------Metricas-----
Accuracy: 0.20500000000000002
Precision: 0.23176587301587298
Recall: 0.20458333333333334
F-score: 0.2039431401931402
-----Scores10-----
dict_keys(['fit_time', 'score_time', 'test_accuracy', 'test_precision_macro', 'test_recall_macro', 'test_f1_macro'])
{'fit_time': array([0.00099802, 0.0009973 , 0.00099993, 0.00100064, 0.0010004 ,
       0.00100183, 0.        , 0.00100017, 0.00100279, 0.00100684]), 'score_time': array([0.00551319, 0.00551033, 0.00502729, 0.00499797, 0.00502419,
       0.00550437, 0.00600529, 0.00501704, 0.00700712, 0.00499868]), 'test_accuracy': array([0.15, 0.1 , 0.25, 0.4 , 0.25, 0.15, 0.1 , 0.3 , 0.15, 0.2 ]), 'test_precision_macro': array([0.1875    ,
0.125     , 0.26041667, 0.43541667, 0.2375    ,
       0.15625   , 0.16666667, 0.4375    , 0.14236111, 0.16904762]), 'test_recall_macro': array([0.14583333, 0.1       , 0.25      , 0.4       , 0.25      ,
       0.15      , 0.1       , 0.3       , 0.15      , 0.2       ]), 'test_f1_macro': array([0.16309524, 0.11111111, 0.24038462, 0.40149573, 0.23803419,
       0.14957265, 0.125     , 0.29148629, 0.13667582, 0.18257576])}
```

Posteriormente para cada clasificador, realizamos una predicción de un nuevo objeto escalando los datos e imprimiendo tanto la Clase asignada como las probabilidades por Clase obtenidas. También graficamos sus 200 objetos y los nuevos objetos usados para las predicciones (remarcados con color verde***)

Para el primer clasificador:

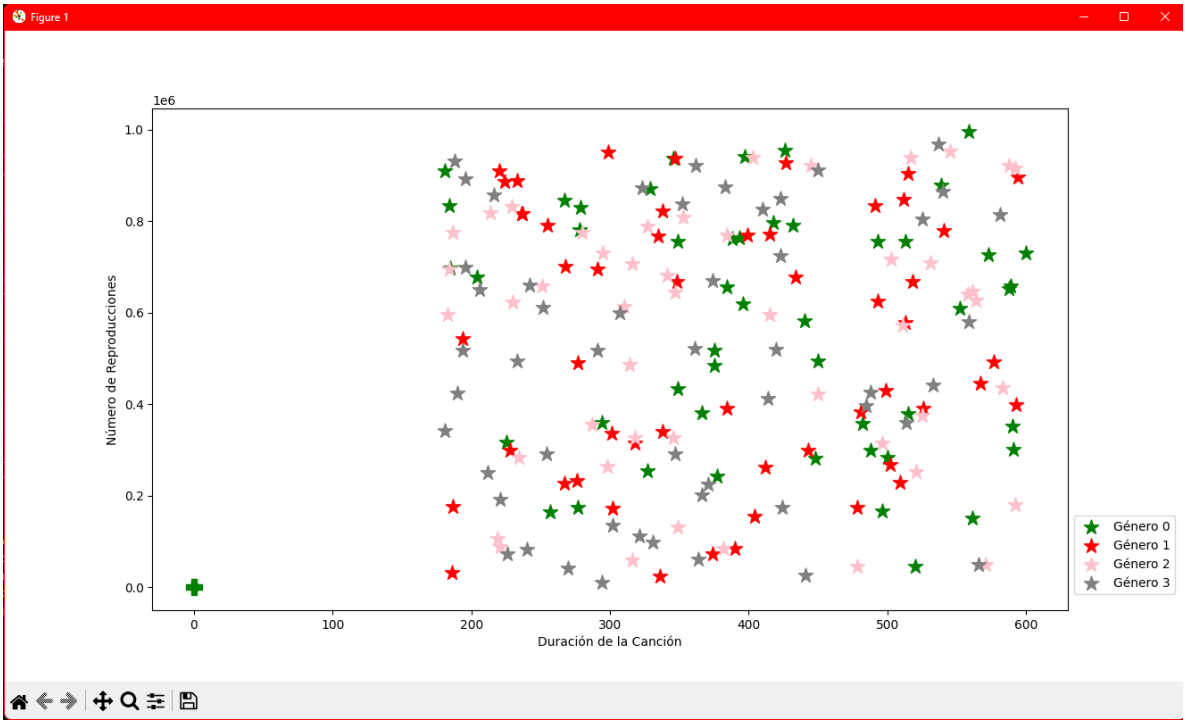
```python
# Crear un nuevo objeto
nuevo_objeto = [[300, 50000]]  # Duración de la Canción: 300, Número de Reproducciones: 50000

# Escalar los datos del nuevo objeto
nuevo_objeto_escalado = escalador.transform(nuevo_objeto)

# Realizar una predicción para el nuevo objeto
prediccion = clasificador.predict(nuevo_objeto_escalado)
probabilidades = clasificador.predict_proba(nuevo_objeto_escalado)

# Imprimir la clase asignada y las probabilidades por clase
print(f"Clase asignada para el nuevo objeto: {prediccion[0]}")
print(f"Probabilidades por clase: {probabilidades[0]}")

# Graficar los 200 objetos y el nuevo objeto
plt.scatter(datos[:, 0], datos[:, 1], c=clase)
plt.scatter(nuevo_objeto_escalado[0, 0], nuevo_objeto_escalado[0, 1], color="green", marker="P", s=150,label="Nuevo Objeto")
plt.ylabel("Número de Reproducciones")
plt.xlabel("Duración de la Canción")
plt.show()
```




```
Clase asignada para el nuevo objeto: 3
Probabilidades por clase: [0.         0.         0.33333333 0.66666667]
```

Para el segundo clasificador:

```python
# Crear un nuevo objeto
nuevo_objeto = [[300, 50000]]  # Duración de la Canción: 300, Número de Reproducciones: 50000

# Escalar los datos del nuevo objeto
nuevo_objeto_escalado = escalador.transform(nuevo_objeto)

# Realizar una predicción para el nuevo objeto
prediccion = clasificador.predict(nuevo_objeto_escalado)
probabilidades = clasificador15.predict_proba(nuevo_objeto_escalado)

# Imprimir la clase asignada y las probabilidades por clase
print(f"Clase asignada para el nuevo objeto: {prediccion[0]}")
print(f"Probabilidades por clase: {probabilidades[0]}")

# Graficar los 200 objetos y el nuevo objeto
plt.scatter(datos[:, 0], datos[:, 1], c=clase)
plt.scatter(nuevo_objeto_escalado[0, 0], nuevo_objeto_escalado[0, 1], color="green", marker="P", s=150,label="Nuevo Objeto")
plt.ylabel("Número de Reproducciones")
plt.xlabel("Duración de la Canción")
plt.show()
```




```
Clase asignada para el nuevo objeto: 3
Probabilidades por clase: [0.13333333 0.2        0.13333333 0.53333333]
```

Por último, se llevó a cabo, una gráfica por regiones de las clases (en total, 4 clases) entonces usamos 4 colores diferentes para iluminar cada región. Para visualizar las regiones de decisión de un clasificador en un espacio bidimensional, usamos una malla de puntos para cubrir el espacio de entrada y luego colorear cada punto de la malla según la clase que el clasificador asigna a ese punto.

```
x_min, x_max = datos[:, 0].min() - 1, datos[:, 0].max() + 1
y_min, y_max = datos[:, 1].min() - 1, datos[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# Usar el clasificador para predecir la clase de cada punto en la malla
Z = clasificador.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Dibujar la malla usando colores para indicar la clase de cada punto
plt.contourf(xx, yy, Z, alpha=0.8)

# Dibujar los puntos de datos
plt.scatter(datos[:, 0], datos[:, 1], c=clase, edgecolor='k')

# Dibujar el nuevo objeto
plt.scatter(nuevo_objeto_escalado[0, 0], nuevo_objeto_escalado[0, 1], color="green", marker="P", s=150,label="Nuevo Objeto")

plt.ylabel("Número de Reproducciones")
plt.xlabel("Duración de la Canción")
plt.show()
```