Práctica 3
Clasificadores k-NN y regresión logística

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

Jesus Abraham Haros Madrid
A01252642@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

ABSTRACT

Esta práctica tiene el propósito de demostrar dos metodologías de predicción para la clasificación de datos, una utilizando regresión logística y la otra utilizando el algoritmo de los *k* vecinos más cercanos. El objetivo de esta es entrenar un modelo utilizando ambos algoritmos para poder compararlos en términos de precisión y poder llegar a conclusiones en cuanto a su uso; este trabajo fue realizado utilizando la librería *scikit-learn* para la creación de los modelos de regresión.

1 INTRODUCCIÓN

Hoy en día existen múltiples métodos para analizar datos y en base a ese análisis hacer aprendizaje automático; en la práctica pasada pudimos ver un ejemplo de predicción de datos utilizando regresión logística y gradiente descendente para predecir en qué grupo estarían ubicados los datos. En esta práctica se elaborará algo parecido, reemplazando lo que fue el gradiente descendente por el algoritmo de clasificación de los *k* vecinos más cercanos, de esta manera expandimos nuestro conocimiento en las diferentes metodologías que existen. Se utilizará *scikit-learn* para la creación de los modelos de predicción, el procedimiento será el mismo que en la práctica anterior, se partirán los datasets de manera aleatoria en una proporción 80% - 20% para generar el training set y el test set, respectivamente. En todos los casos se hará uso de la validación simple y se ejecutará el entrenamiento utilizando los mismos datos para ambos modelos.

2 CONCEPTOS PREVIOS

- Programación básica en los lenguajes R y Python
- Conocimiento de las librerías *scikit-learn*, *pandas* y *numpy*
- Conocimientos de estadística y de regresión logística
- Algoritmo *k* vecinos más cercanos
- Espacio ROC

3 METODOLOGÍA

3.1 Datasets

Para comparar ambas implementaciones utilizamos dos datasets, expuestos a continuación:

3.1.1 Dataset DEFAULT.

Este dataset está compuesto por 10,000 filas, cada una representa un cliente de un banco que puede o no cumplir con los pagos de su tarjeta de crédito (columna ”default”). De cada cliente tenemos la siguiente información:

- Columna ”default”: Tiene los valores ”Yes”/”No”, representa si la persona realizó el pago mínimo a su tarjeta de crédito.
- Columna ”student”: Valores ”Yes”/”No”, representa si el cliente es un estudiante en ese momento.
- Columna ”balance”: Número decimal positivo que representa el balance de la tarjeta de crédito del cliente. Promedio de 835.4, números en el rango [0, 2654.3].
- Columna ”income”: Número decimal positivo que representa los ingresos que tiene el cliente. Promedio de 33517, números en el rango [772, 73554]

De este dataset, nuestro objetivo es predecir la columna ”default” a partir de los otros tres parámetros, y como preparación cambiamos los valores de ”student” (”Yes”/”No”) a valores 1 y 0 respectivamente.

3.1.2 Dataset GENERO.

Este dataset representa las mediciones de peso y altura de 10,000 personas, en conjunto con el género de la persona a la que se realizaron las medidas. Las columnas son:

- ”Gender”: Valores ”Male”/”Female”, el género de la persona a la que le corresponde esta fila de mediciones.
- ”Height”: La altura de la persona en pulgadas, promedio de 66.37, en el rango [54.26 y 79.00].
- ”Weight”: El peso de la persona en libras, promedio de 161.4, en el rango [64.7, 270.0].

La columna objetivo seleccionada de este dataset fue el género ya que tiene dos clasificaciones.

3.2 Clasificación con k-NN

Para esta etapa de la práctica utilizaremos el algoritmo que se encuentra implementado en la librería *scikit-learn*, específicamente la clase *sklearn.neighbors.KNeighborsRegressor*.

Lo primero que se hizo fue declarar un arreglo *k_neighbors* que contenía los valores 1, 2, 3, 4, 10, 15, 20, 50, 75, 100, los cuales son el número de vecinos que utilizará el algoritmo para clasificar. Establecemos brute como el valor de algorithm para la clase *NearestNeighbors*.

Después se corrió un ciclo iterando sobre el arreglo *k_neighbors* para entrenar el modelo variando los *k* vecinos, para cada ciclo se calculó la precisión de los modelos (obtener la tasa de precisión) y la matriz de confusión.

Con base a los valores obtenidos de las tasas de precisión de cada uno se generó una gráfica donde se muestran los *k* vecinos contra la tasa de precisión obtenida por cada dataset.

3.2.1 *Dataset DEFAULT.*

Para este dataset primero leemos el archivo CSV a un Dataframe de pandas, transformamos las columnas "default" y "student" para que contengan valores booleanos y enteros respectivamente, seleccionamos las columnas que nos servirán como variables independientes (columnas "student", "balance" e "income") y variable dependiente (columna "default"). Después partimos las filas del dataset en una porción del 80% que usaremos para entrenar el modelo, y otra porción del 20% para probarlo. Instanciamos KNN de sklearn, lo entrenamos con los datos y después predecimos la variable dependiente con el modelo para así compararlo con los datos reales de prueba, usando una medida de tasa de precisión y la matriz de confusión.

El código fuente de este ejemplo se encuentra en el apéndice A.

3.2.2 *Dataset GENERO.*

Para este dataset realizamos un procedimiento similar: leer el dataset para crear un Dataframe, seleccionar las columnas de variables independientes ("Height" y "Weight") y la dependiente ("Gender"), dividir el dataset en 80%/20%, entrenar el modelo, y predecir la variable dependiente para los datos de prueba, para así comparar estos con los datos reales.

El código fuente puede ser encontrado en el apéndice B.

3.3 Regresión logística

Para la primera parte de la práctica, en la que utilizamos la implementación de *scikit-learn*, seleccionamos la clase *sklearn.linear_model.LogisticRegression*[1] para nuestros scripts.

3.3.1 *Dataset DEFAULT.*

Para este dataset primero leemos el archivo CSV a un *Dataframe* de *pandas*, transformamos las columnas "default" y "student" para que contengan valores booleanos y enteros respectivamente, seleccionamos las columnas que nos servirán como variables independientes (columnas "student", "balance" e "income") y variable dependiente (columna "default"). Después partimos las filas del dataset en una porción del 80% que usaremos para entrenar el modelo, y otra porción del 20% para probarlo.

Instanciamos el modelo de regresión de *sklearn*, lo entrenamos con los datos y después predecimos la variable dependiente con el modelo para así compararlo con los datos reales de prueba, usando una medida de tasa de precisión y la matriz de confusión.

El código de este ejemplo puede se encuentra en el apéndice A.

3.3.2 *Dataset GENERO.*

Para este dataset realizamos un procedimiento similar: leer el dataset para crear un *Dataframe*, seleccionar las columnas de variables independientes ("Height" y "Weight") y la dependiente ("Gender"), dividir el dataset en 80%/20%, entrenar el modelo, y predecir la variable dependiente para los datos de prueba, para así comparar estos con los datos reales.

El código para esta sección se encuentra en el apéndice B.

4 RESULTADOS

4.1 Clasificación con k-NN

4.1.1 *Dataset DEFAULT.*

Al iterar sobre la cantidad de vecinos para el algoritmo k-NN pudimos observar que cuando se selecciona un sólo vecino se tiene la precisión más baja, mientras que al seleccionar dos vecinos para hacer la clasificación se presenta la precisión más alta de 0.966, y después, con una mayor cantidad de vecinos, la precisión baja y se mantiene constante en un valor de 0.963.

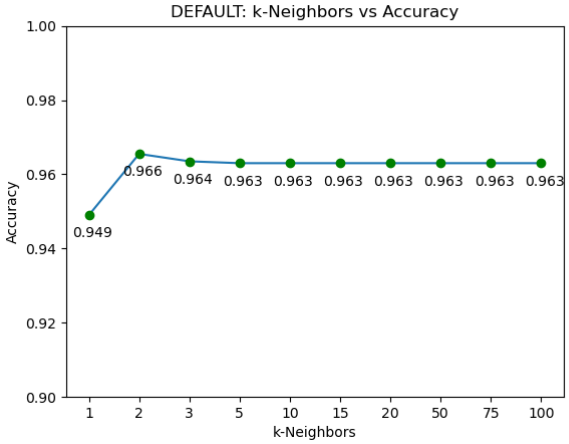

Figure 1: Variación del número de vecinos elegidos contra la precisión en el dataset DEFAULT

La matriz de confusión obtenida para el dataset default utilizando k-NN fue la siguiente:

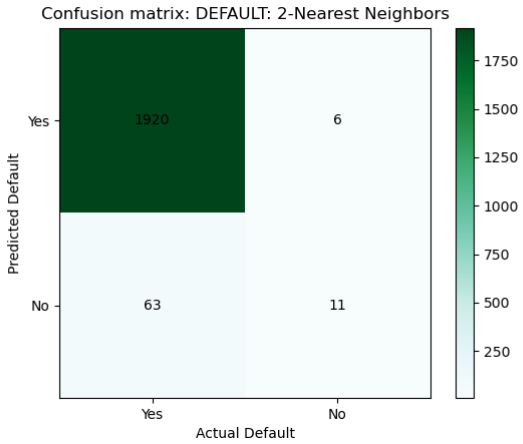

Figure 2: Matriz de confusión del dataset DEFAULT con 2-Nearest Neighbors

4.1.2 *Dataset GENERO.*

Al iterar sobre la cantidad de vecinos para el algoritmo k-NN pudimos observar como al tener un sólo vecino se tiene la precisión más alta y conforme se aumenta el número de vecinos la precisión de k-NN va bajando, siendo la mayor precisión con un vecino de 0.884 y la peor precisión con 100 vecinos de 0.677.

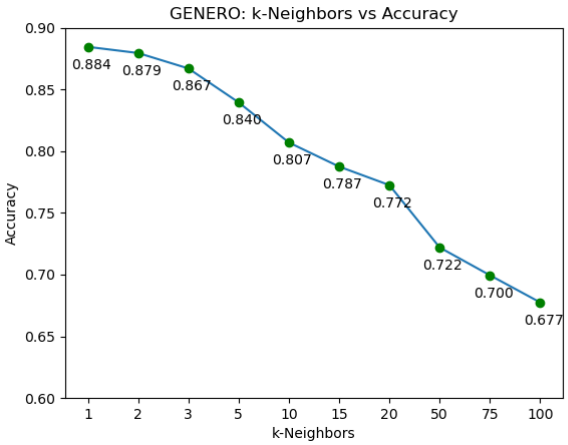

Figure 3: Variación del número de vecinos elegidos contra la precisión en el dataset GENERO

La matriz de confusión obtenida para el dataset default utilizando k-NN fue la siguiente:

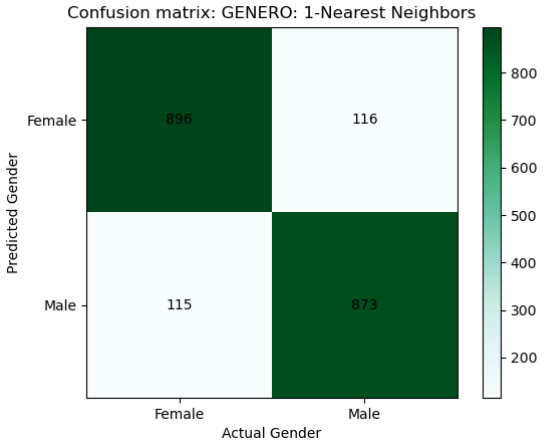

Figure 4: Matriz de confusión del dataset GENERO con 1-Nearest Neighbors

También se creó un diagrama de dispersión utilizando como eje X la altura de la persona, y en el eje Y el peso. Los puntos rojos representan personas del género femenino y los azules personas del género masculino.

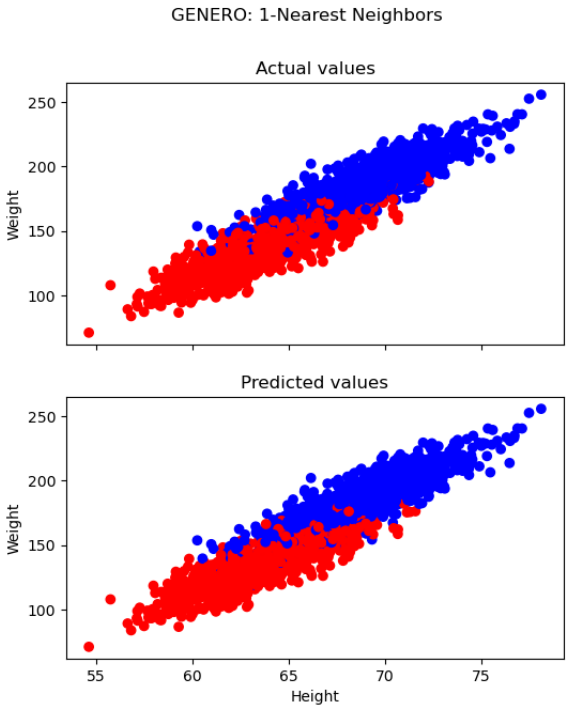

Figure 5: Gráfica de dispersión del dataset GENERO con 1-Nearest Neighbors

4.2 Regresión logística

4.2.1 *Dataset DEFAULT.*

El modelo de regresión logística tuvo una tasa de precisión de 0.965, con una gráfica de la matriz de confusión así:

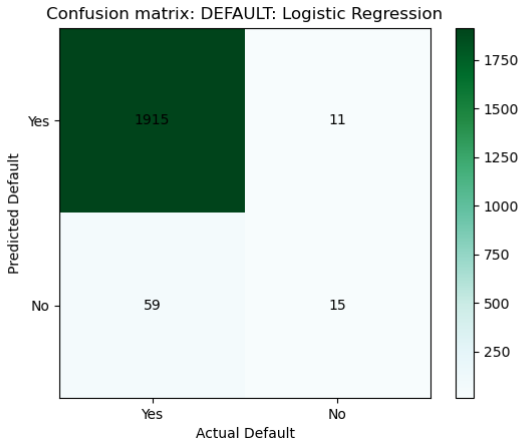

Figure 6: Matriz de confusión del dataset DEFAULT con regresión logística

Se puede apreciar como para el dataset DEFAULT la regresión logística tuvo peor precisión que k-NN pero la diferencia fue casi despreciable, siendo de 0.966 a 0.965.

4.2.2 *Dataset GENERO.*

Para este dataset el modelo de regresión logística tuvo una tasa de precisión de 0.9225 con la siguiente gráfica de la matriz de confusión.

Figure 7: Matriz de confusión del dataset GENERO con regresión logística

Se creó un diagrama de dispersión utilizando como eje X la altura de la persona, y en el eje Y el peso. Los puntos rojos representan personas del género femenino y los azules personas del género masculino.

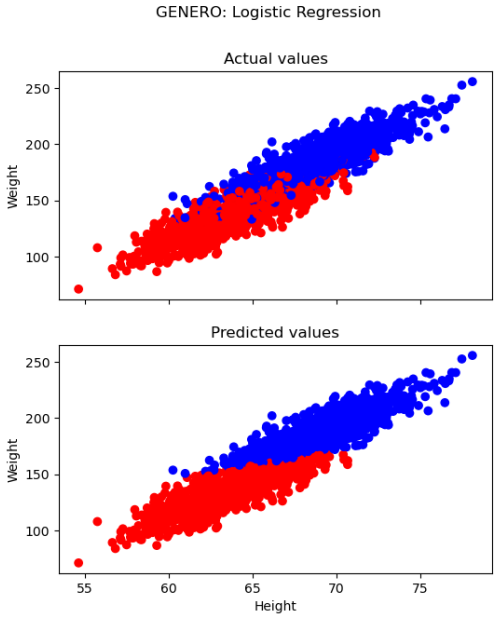

Figure 8: Gráfica de dispersión del dataset GENERO con regresión logística

4.3 Comparación de clasificadores

En ambos casos los resultados de la matriz de confusión fueron muy similares; si comparamos la figura 4 con la figura 7 los valores en cada cuadrante tienen muy poca variación respecto a la otra.

Ahora si comparamos las gráficas de dispersión elaboradas con los resultados de los clasificadores la historia es muy parecida, ambas se ven casi idénticas con diferencias mínimas que cambian en cada corrida del algoritmo, ambas son muy similares.

El espacio ROC nos sirve para poder identificar las ventajas y desventajas de cada clasificador graficando el promedio de los falsos positivos y verdaderos positivos en un plano. Lo ideal para un modelo es que este su TPR esté lo más cercano al 1 y su FPR lo más cercano al 0, esa sería la precisión ideal de predicción.

A continuación utilizaremos las matrices de confusión creadas por ambos clasificadores para el dataset GENERO y calcularemos su True positive rate y False positive rate para poder graficarlos en el espacio ROC y poder determinar cuál es el mejor clasificador.

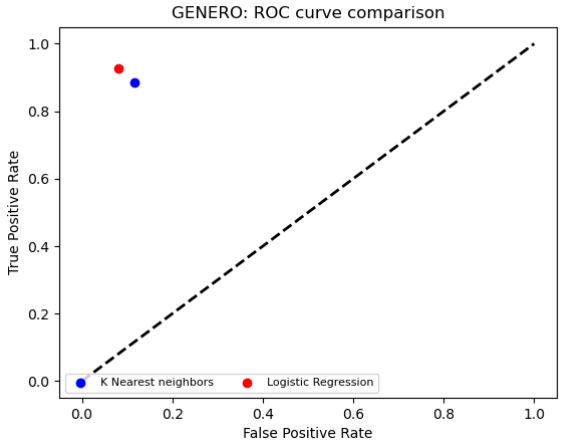

Figure 9: Gráfica del espacio ROC comparando ambos modelos de clasificación

Se puede observar como la regresión logística es mejor método que el k-NN pero por muy poco. Al tener un TPR mayor y menor FPR que k-NN se puede llegar a la conclusión de que es mejor la regresión logística, el caso interesante hubiera sido que uno hubiera tenido mayor TPR que el otro pero también mayor FPR, ahí pudieramos tomar una comparativa y analizar a profundidad sus desventajas y ventajas para tomar una decisión de que modelo utilizar.

5 CONCLUSIONES Y REFLEXIONES

En la práctica anterior nos dimos cuenta de que existen múltiples formas de clasificar los datos de un dataset para poder hacer predicciones de sus variables, en esta práctica pudimos aprender una nueva manera, K vecinos más cercanos (K-NN). Fue muy interesante ver el comportamiento de un nuevo método y ver como era un procedimiento completamente distinto a la regresión logística pero su resultado muy similar a ella.

Pudimos predecir el sexo de una persona dada su altura y su peso y tambien predecir la columna ¨default¨ del dataset default.

Al igual que en todas las prácticas fue muy útil utilizar las funciones, clases y métodos incluidas en scikit learn para elaborar esta práctica.

Al final se pudieron observar resultados muy similares entre ambos métodos, algo que cambió de la práctica anterior a esta fue que tuvimos que utilizar los mismos conjuntos de datos para entrenar ambos modelos, en prácticas pasadas se pudo observar más variación en los resultados debido a que se generaban de nuevo los conjuntos de entrenamiento y prueba por cada método de clasificación, esta vez al ser solo un set de datos los resultados fueron más similares entre cada método.

Por último conocer sobre el espacio ROC y el comportamiento de predicción ideal de los datasets fue muy interesante y a la vez muy útil para poder comparar las precisiones de diferentes métodos de clasificación, al tomar en cuenta su FPR y su TPR se pueden ver también las ventajas y desventajas de cada modelo.

5.1 Refrexión de Abraham

Esta práctica me permitió ver mi avance personal en esta materia, esta vez se me hizo sencillo implementar un modelo nuevo, también me sentí más seguro de lo que estaba pasando ya que se tomó como método la regresión logística que ya había trabajado anteriormente, me pareció muy bueno ver como diferentes métodos se acercan al mismo resultado y poder comparar la precisión de los métodos en el espacio ROC para poder seleccionar el mejor.

Al implementar K-NN observé que tener más número de vecinos no da mejores resultados, dependiendo del caso de uso y del dataset podemos ver como K vecinos fluctúa, siento que fue muy útil comparar los K vecinos y tomar el mejor, sólo me queda la duda de qué rango es el mejor para comparar (de 0 a 10, de 0 a 100, de 0 a 1000).

Prácticas como esta hacen que lo aprendido en clase se convierta en conocimiento aplicable.

5.2 Reflexión de Mario

En mi opinión, esta práctica fue muy útil para visualizar cómo funciona el algoritmo de los k vecinos más cercanos, especialmente con la gráfica de dispersión del dataset GENERO, ya que como curiosidad generé las gráficas para los demás valores de k y pude observar cómo se mantenía la precisión en los alrededores de la gráfica, pero en el centro realmente predominaban las clasificaciones de los puntos con el género femenino, problema que fue resuelto en los modelos que utilizaban un valor de k menor (entre 1 y 5).

También considero que esta práctica fue mucho menos pesada en cuestión de trabajo ya que ya habíamos trabajado con estos dos datasets, y ya habíamos generado modelos de regresión logística en la práctica anterior.

REFERENCES

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

A CÓDIGO DE CLASIFICACIÓN K-NN Y REGRESIÓN LOGÍSTICA DEL DATASET DEFAULT

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from graphs import graphConfusionMatrix, graphNeighborsAccuracy

print(" ~ Reading default.txt and generating train and test sets")
data = pd.read_csv('default.txt', sep=" ")
# Transform 'default' and 'student' columns from Yes/No to integers (1/0)
data["default"] = (data["default"] == "Yes").astype(int)
data["student"] = (data["student"] == "Yes").astype(int)
x = data.iloc[:, 1:4].values.reshape(-1, 3)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(
    x, y, test_size=0.2, random_state=0)


# Nearest neighbors
print(" ~ Creating and testing the k-NN models for different values")
print(" →   neighbors   accuracy       confusion matrix")
k_neighbors_vals = [1, 2, 3, 5, 10, 15, 20, 50, 75, 100]
accuracy_vals = []
best_k_val = (-1, 0, [])  # k_neighbors, accuracy, conf_matrix
for val in k_neighbors_vals:
    nn = KNeighborsRegressor(n_neighbors=val, algorithm="brute")
    nn.fit(xTrain, yTrain)
    yPredicted = nn.predict(xTest).astype(int)
    accuracy = accuracy_score(yTest, yPredicted)
    accuracy_vals.append(accuracy)
    cm = confusion_matrix(yTest, yPredicted)
    print(" →   {:^9}   {:^8}   {}".format(val, accuracy, cm.tolist()))
    if best_k_val[1] < accuracy:
        best_k_val = (val, accuracy, cm)

graphNeighborsAccuracy(k_neighbors_vals, accuracy_vals,
                       "DEFAULT", "default_neighbors_acc.png")
graphConfusionMatrix(best_k_val[2], ["Yes", "No"], "Default",
                     "DEFAULT: {}-Nearest Neighbors".format(best_k_val[0]),
                     "default_neighbors_cm.png")


# Logistic regression
print("\n\n ~ Creating the logistic regression model")
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())
print(" ~ Testing the logistic regression model")
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

graphConfusionMatrix(cm, ["Yes", "No"], "Default",
                     "DEFAULT: Logistic Regression", "default_regression_cm.png")
```

B CÓDIGO DE CLASIFICACIÓN K-NN Y REGRESIÓN LOGÍSTICA DEL DATASET GENERO

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from graphs import graphConfusionMatrix, graphNeighborsAccuracy, graphGENERO, graphROC

print(" ~ Reading genero.txt and generating train and test sets")
data = pd.read_csv('genero.txt')
x = data.iloc[:, 1:3].values.reshape(-1, 2)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(
    x, y, test_size=0.2, random_state=0)


# Nearest neighbors

print(" ~ Creating and testing the k-NN models for different values")
yIntToStr = np.vectorize(lambda x: ["Female", "Male"][int(x)])
print(" →   neighbors   accuracy       confusion matrix")
k_neighbors_vals = [1, 2, 3, 5, 10, 15, 20, 50, 75, 100]
accuracy_vals = []
best_k_val = (-1, 0, [], [])  # k_neighbors, accuracy, conf_matrix, yPredicted
for val in k_neighbors_vals:
    nn = KNeighborsRegressor(n_neighbors=val, algorithm="brute")
    # Convert String Male/Female to integers 1/0
    yTrainInt = (yTrain.ravel() == "Male").astype(int)
    nn.fit(xTrain, yTrainInt)
    yPredicted = nn.predict(xTest)
    yPredicted = yIntToStr(yPredicted)
    accuracy = accuracy_score(yTest, yPredicted)
    accuracy_vals.append(accuracy)
    cm = confusion_matrix(yTest, yPredicted)
    print(" →   {:^9}   {:^8}   {}".format(val, accuracy, cm.tolist()))
    if best_k_val[1] < accuracy:
        best_k_val = (val, accuracy, cm, yPredicted)
print("\n →   Confusion matrix:", best_k_val[2].tolist())

# Calculate true positive rate and false positive rate to compare models
knn_tpr = best_k_val[2][0][0] / best_k_val[2][0].sum()  # TPR = TP / (TP + FN)
knn_fpr = best_k_val[2][1][0] / best_k_val[2][1].sum()  # FPR = FP / (FP + TN)
graphNeighborsAccuracy(k_neighbors_vals, accuracy_vals,
                       "GENERO", "genero_neighbors_acc.png")
graphGENERO(xTest, yTest, best_k_val[3], "{}-Nearest Neighbors".format(best_k_val[0]),
            "genero_neighbors_pred.png")
graphConfusionMatrix(best_k_val[2], ["Female", "Male"], "Gender",
                     "GENERO: {}-Nearest Neighbors".format(best_k_val[0]),
                     "genero_neighbors_cm.png")
```

```
→→→

# Logistic regression

print("\n\n ~ Creating the logistic regression model")
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())

print(" ~ Testing the logistic regression model")
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

# Calculate true positive rate and false positive rate to compare models
lr_tpr = cm[0][0] / cm[0].sum()  # TPR = TP / (TP + FN)
lr_fpr = cm[1][0] / cm[1].sum()  # FPR = FP / (FP + TN)
graphGENERO(xTest, yTest, yPredicted, "Logistic Regression",
            "genero_regression_pred.png")
graphConfusionMatrix(cm, ["Male", "Female"], "Gender",
                     "GENERO: Logistic Regression", "genero_regression_cm.png")
graphROC(knn_tpr, knn_fpr, lr_tpr, lr_fpr,
         "GENERO", "genero_roc_curve.png")
```

C CÓDIGO DE GENERACIÓN DE GRÁFICAS

```
import math
import sys
import matplotlib.pyplot as plt
import numpy as np

shouldDisplay = "--display-graphs" in sys.argv
shouldSave = "--save-graphs" in sys.argv

mapColor = np.vectorize(lambda x: "b" if x == "Male" else "r")


def graphGENERO(xys, actualVals, predictedVals, title, filename):
    fig, (ax1, ax2) = plt.subplots(2, sharex=True)
    fig.set_size_inches(6, 7)
    fig.suptitle("GENERO: " + title)

    ax1.set_title("Actual values")
    ax1.set_ylabel("Weight")
    actualColors = mapColor(actualVals).flatten()
    ax1.scatter(xys[:, 0], xys[:, 1], c=actualColors)

    ax2.set_title("Predicted values")
    ax2.set_xlabel("Height")
    ax2.set_ylabel("Weight")
    predictedColors = mapColor(predictedVals).flatten()
    ax2.scatter(xys[:, 0], xys[:, 1], c=predictedColors)

    if shouldSave:
        fig.savefig(filename)
    if shouldDisplay:
        plt.show()


def graphConfusionMatrix(cm, labels, variableName, title, filename):
    plt.figure()
    plt.imshow(cm, interpolation="nearest", cmap="BuGn")
    plt.title("Confusion matrix: " + title)
    plt.colorbar()
    plt.xticks(np.arange(2), labels, size=10)
    plt.yticks(np.arange(2), labels, size=10)
    plt.xlabel("Actual " + variableName)
    plt.ylabel("Predicted " + variableName)

    for x in range(2):
        for y in range(2):
            plt.annotate(cm[x][y], xy=(y, x),
                         horizontalalignment="center",
                         verticalalignment="center")

    if shouldSave:
        plt.savefig(filename)
    if shouldDisplay:
        plt.show()
```

```
56 def graphNeighborsAccuracy(k_neighbors, accuracy, title, filename):
57     x = list(map(str, k_neighbors))
58 
59     plt.figure()
60     plt.title(title + ": k-Neighbors vs Accuracy")
61     plt.plot(x, accuracy)
62     plt.plot(x, accuracy, 'og')
63 
64     for i in range(len(x)):
65         xy = (x[i], accuracy[i])
66         plt.annotate("{:.3f}".format(accuracy[i]), xy, xycoords='data',
67                      xytext=(-12, -16), textcoords="offset points")
68 
69     plt.xlabel("k-Neighbors")
70     plt.ylabel("Accuracy")
71     bottom_ylim = math.floor(10 * min(accuracy)) / 10
72     top_ylim = math.ceil(10 * max(accuracy)) / 10
73     plt.ylim((bottom_ylim, top_ylim))
74 
75     if shouldSave:
76         plt.savefig(filename)
77     if shouldDisplay:
78         plt.show()
79 
80 
81 def graphROC(knn_tpr, knn_fpr, lr_tpr, lr_fpr, title, filename):
82     plt.figure()
83     plt.title(title + ': ROC curve comparison')
84     plt.plot([0, 1], [0, 1], 'k--', lw=2)
85 
86     knn = plt.scatter(knn_fpr, knn_tpr, c='b')
87     lr = plt.scatter(lr_fpr, lr_tpr, c='r')
88 
89     plt.legend((knn, lr), ('K Nearest neighbors', 'Logistic Regression'),
90                scatterpoints=1, loc='lower left', ncol=3, fontsize=8)
91 
92     plt.xlabel("False Positive Rate")
93     plt.ylabel("True Positive Rate")
94 
95     if shouldSave:
96         plt.savefig(filename)
97     if shouldDisplay:
98         plt.show()
```