Universidad Autónoma de Querétaro

Facultad de Ingeniería
Maestría en Inteligencia Artificial
Machine Learning
Sheila Leyva López
Cecilia Gabriela Rodríguez Flores

Clasificación con KNN + validación cruzada KFOLD.

09 de enero del 2023

Objetivo

Implementar el algoritmo de KNN, el cual será probado con la base de datos de enfermedad cardíaca, haciendo uso de la validación de k-fold y la validación en línea 60-20-20.

Introducción

El algoritmo de k vecinos más cercanos, también conocido como KNN o k-NN, es un clasificador de aprendizaje supervisado no paramétrico, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual. Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.

En este trabajo se aplica dicha técnica de clasificación a la base de datos *Indicadores personales clave de enfermedad cardíaca*. Para ello se desarrollan funciones, en lenguaje de Python, necesarias para la predicción de clases de cualquier base de datos, sin embargo, toma como referencia la base de datos antes mencionada.

Marco Teórico

Es importante definir conceptos básicos implementados en este trabajo a fin brindar un mejor entendimiento del contexto en el que se trabaja.

K-vecinos más cercanos

El algoritmo K-vecinos más cercanos (KNN) es un tipo de algoritmo de aprendizaje automático supervisado que se utiliza para la clasificación, la regresión y la detección de valores atípicos. Es extremadamente fácil de implementar en su forma más básica, pero puede realizar tareas bastante complejas. Sin embargo, es un algoritmo de aprendizaje perezoso ya que no tiene una fase de entrenamiento especializada. Más bien, utiliza todos los datos para el entrenamiento mientras clasifica (o retrocede) un nuevo punto de datos o instancia.

Así bien, KNN es un algoritmo de aprendizaje no paramétrico, lo que significa que no asume nada sobre los datos subyacentes. Esta es una característica extremadamente útil ya que la mayoría de los datos del mundo real no siguen ningún supuesto teórico, por ejemplo, separabilidad lineal, distribución uniforme, etc.

Por otra parte, KNN ofrece múltiples beneficios, pero también adolece de ciertas deficiencias.

Ventajas

- Alta precisión
- Insensible a valores atípicos
- Sencillo y fácil de entender

Desventajas

- Alta complejidad temporal
- Alta complejidad espacial

División del conjunto de datos

Se evalúa un modelo para estimar la calidad de su generalización de patrones para los datos en los que dicho modelo no ha sido entrenado. Sin embargo, dado que las instancias futuras tienen valores de destino desconocidos y no se puede comprobar ahora mismo la precisión de las predicciones para las instancias del futuro, se deben de utilizar algunos de los datos para los que ya conocemos la respuesta para los datos futuros. Lamentablemente, evaluar un modelo con los mismos datos que se han utilizado para el entrenamiento no es útil, ya que termina perjudicando a los modelos que pueden "recordar" los datos de entrenamiento en lugar de generalizar.

Una estrategia común consiste en tomar todos los datos etiquetados y dividirlos en subconjuntos de entrenamiento y evaluación, normalmente con una proporción del 70 al 80 % para entrenamiento y un 20 al 30 % para evaluación. Así bien, existe la posibilidad de agregar un conjunto de validación con lo cual los porcentajes se suelen distribuir a fin de mantener el porcentaje más alto para el conjunto de entrenamiento, el cual se encuentra alrededor del 60 al 80 %; mientras que los porcentajes de los conjuntos de prueba y validación suelen ser muy similares.

Validación cruzada K-FOLD

Existen situaciones en las que no se cuenta con un gran número de datos para entrenamiento y prueba. La validación cruzada permite dividir el conjunto de datos, de tal forma que, los datos se pueden utilizar en el entrenamiento y en prueba. Específicamente en la validación cruzada de k iteraciones (kfold cross validation) se divide el conjunto de datos original en subconjuntos, de modo que, durante el entrenamiento se toma cada k subconjunto como un conjunto de prueba total, mientras que los demás subconjuntos se toman como entrenamiento, esto se repetirá k veces, y cada vez el conjunto de prueba será diferente y el resto de los datos se utilizará para

entrenamiento [12]. En cada iteración se busca el mayor valor de precisión y se elige la iteración que haya tenido el mejor desempeño.

Ilustración 1. Validación de kfold.

Métricas

Exactitud

La exactitud es la relación entre los simples correctamente clasificados y el número total de simples en el conjunto de datos de evaluación. Esta métrica es conocida por ser engañosa en el caso de diferentes proporciones de clases, ya que asignar simplemente todos los simples a la clase predominante es una forma fácil de lograr una alta precisión [1].

$$ACC = \frac{\#\ correctly\ classified\ samples}{\#\ all\ samples} = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

Precisión

La precisión es la capacidad de un modelo para identificar sólo objetos relevantes, es decir, es el porcentaje de predicciones positivas correctas entre todas las verdades básicas dadas [1].

$$P_r = \frac{\sum_{n=1}^{S} TP_n}{\sum_{n=1}^{S} TP_n + \sum_{n=1}^{N-S} FP_n} = \frac{\sum_{n=1}^{S} TP_n}{all\ detection} \quad (2)$$

Sensitividad

La sensitividad o recall se define como el porcentaje de predicciones correctas tomadas de la clase de predicciones correctas sin tomar en cuenta los falsos positivos [1].

$$REC = \frac{\#\ true\ positive\ samples}{\#\ samples\ classified\ positive} = \frac{TP}{TP + FN} \quad (3)$$

F1 Score

La F1 Score calcula la media armónica de la precisión y recall de forma que se enfatiza al valor más bajo entre ambas [1].

$$F1 = 2\,x\,\frac{precision\ x\ recall}{precision + recall} = \frac{2\,x\,TP}{2\,x\,TP + FP + FN} \quad (4)$$

Materiales y métodos

Herramientas utilizadas

- Google Collaboratory
- Conjunto de datos: *Indicadores personales clave de enfermedad cardíaca*
- AMD Ryzen 9 5900HS with Radeon Graphics 3.30 GHz
- RAM 16.0 GB
- 64-bit operating system, x64-based processor

Conjunto de datos

En este trabajo, se utilizará el conjunto de datos: *Indicadores personales clave de enfermedad cardíaca*, datos provenientes de 400,000 adultos, obtenidos durante la encuesta anual 2020 de los Centros para el Control y prevención de Enfermedades (CDC, por sus siglas en inglés) pertenecientes al departamento de salud y servicios humanos en los Estados Unidos. Originalmente el conjunto de datos contenía alrededor de 300 atributos, sin embargo, se redujo a solo 18 variables, los cuales son los que se encuentran disponibles públicamente en la plataforma Kaggle [2].

Casi la mitad de los estadounidenses (47%), incluyendo afroamericanos, indios americanos, nativos de Alaska y blancos; tienen al menos de 1 a 3 factores de riesgo de padecer alguna enfermedad cardíaca. A continuación, se agrega una breve descripción de los atributos incluidos en este conjunto de datos:

- HeartDisease: (atributo de decisión): personas encuestadas que informaron alguna vez haber padecido alguna enfermedad coronaria (CHD, por sus siglas en inglés) o infarto al miocardio(IM, por sus siglas en inglés).
- BMI: Índice de Masa Corporal.
- Smoking: personas encuestadas que han fumado al menos 100 cigarros en su vida entera.
- AlcoholDrinking: corresponde a hombres adultos que beben más de 14 tragos por semana y mujeres adultas que beben más de 7 tragos por semana.
- Stroke: responde a la pregunta: ¿alguna vez le dijeron o usted tuvo un derrame cerebral?
- PhysicalHealth: incluyendo enfermedades y lesiones físicas, responde a la pregunta: ¿durante cuántos días en los últimos 30 días su salud física no fue buena? (de 0 a 30 días)
- MentalHealth: ¿durante cuántos días en los últimos 30 días su salud mental no fue buena? (de 0 a 30 días).
- DiffWalking: responde a ¿tiene serias dificultades para caminar o subir escaleras?
- Sex: hombre o mujer.
- AgeCategory: 14 rangos de edad.
- Race: valor de raza / etnicidad imputada.
- Diabetic: responde a ¿alguna vez ha sido diagnosticada con diabetes?
- PhysiclActivity: adultos que informaron haber realizado actividad física o ejercicio en los últimos 30 días, no incluyendo su trabajo habitual.
- GenHealth: responde a ¿cómo calificarías tu salud en general?
- SleepTime: responde a un promedio de horas que duerme, en un periodo de 24 horas, la persona encuestada.
- Asthma: responde a ¿alguna vez ha sido diagnosticado con asma?

Métodos

Para la realización de esta tarea, fue necesario importar las librerías de Pandas, numpy, etc., así como el montaje de Google Drive en el entorno de ejecución a fin de poder hacer uso de la base de datos propuesta.

Posteriormente, se optó por crear diversas funciones a fin de distribuir las tareas de una manera más eficiente como se muestra a continuación:

- Función valores_faltantes: obtiene el porcentaje total de los valores faltantes, así como el porcentaje de valores faltantes para ca uno de los atributos que comprende la base de datos en cuestión. Esta función requiere como datos de entrada únicamente la base de datos.

```
def valores_faltantes(dataset):
    missing_values_count = dataset.isnull().sum()
    total_missing = missing_values_count.sum()
    #Porcentaje de datos faltantes
    total_missing_porcent = total_missing/(np.product(dataset.shape))*100
    print('Porcentaje total de valores faltantes:',total_missing_porcent,'%')
    print('')
    print('Porcentaje de valores faltantes de cada atributo:')
    for col in dataset.columns:
        VP_missing = np.mean(dataset[col].isnull())
        print('{} - {}%'.format(col,round(VP_missing*100)))
```

Ilustración 2. Función para calcular los valores faltantes.

- Función norm_min_max: realiza la normalización de la base de datos. Esta función requiere como datos de entrada la base de datos en cuestión.

```
def norm_min_max(datos):
  lim_sup = []
  lim_inf = []
  rangoDatos =[]
  maxNorm = 1
  minNorm = 0
  rango = maxNorm - minNorm
  for i in range (0,datos.columns.size):
    lim_sup.append(datos.iloc[:,i].max())
    lim_inf.append(datos.iloc[:,i].min())
    rangoDatos.append(lim_sup[i] - lim_inf[i])
  nombres = datos.columns.values.tolist()
  datosNorm = pd.DataFrame(columns = nombres)

  for j in range(len(datos.columns)):
    varNorm = []
    var = datos.iloc[:,j]
    for i in range(len(datos)):
      D = var[i] - lim_inf[j]
      DPct = D/rangoDatos[j]
      dNorm = rango*DPct
      varNorm.append(minNorm+dNorm)
    datosNorm.iloc[:,j] = varNorm
  datos = datosNorm
  return datos
```

Ilustración 3. Función para normalizar los datos.

- Función matriz_cov: se encarga de obtener la matriz de covarianza de los elementos de la base de datos en cuestión.

```
def matriz_cov(data):
    atributos = data.columns
    n = len(atributos)
    m = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            X = data[atributos[i]]
            Y = data[atributos[j]]
            m[i][j] = (((X-X.mean())*(Y-Y.mean())).sum())/(len(X)-1)
    return m
```

Ilustración 4. Función para calcular la matriz de covarianza.

- Función PCA: brinda los porcentajes de cada uno de los atributos de acuerdo con su relevancia dentro de la base de datos, a fin de discernir los atributos con mayor peso.

```
def PCA(datos,col_decision):
    datos1 = datos.drop([col_decision],axis=1) #Eliminando el atributo de decisión
    #Ajustar los datos restando la media a cada atributo
    datos_A = pd.DataFrame(columns=datos1.columns,index=range(len(datos1)))
    for i in datos_A.columns:
        datos_A[i] = datos1[i] - datos1[i].mean()
    #datos_A
    matrix = matriz_cov(datos_A)
    #sns.heatmap(matrix)
    L,V =  np.linalg.eig(matrix)
    #Obtener el porcentaje de covarianza de cada uno de los atributos
    total = L.sum()
    p = (L/total)*100
    pca =[]
    columnas1 = datos_A.columns.values
    for index, row in enumerate(p):
        print(columnas1[index] + ':',row)
```

Ilustración 5. Función de PCA.

- Función euclidean_distance: se encarga de obtener la distancia entre cada uno de los datos. Esta función requiere como entrada los atributos.

```
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):                    #for k in range(23)
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)
```

Ilustración 6. Función para calcular la distancia euclidiana.

- Función get_neighbors: realiza la extracción de los n vecinos más cercanos, de acuerdo con el valor de k propuesto. Esta función requiere como entradas el número de n vecinos, la lista de distancias ordenadas.

```
def get_neighbors(k,distances_ord):
  neighbors = list()
  indices = list()
  for i in range(k):
    neighbors.append(distances_ord[i][0])
    indices.append(distances_ord[i][1])
  return neighbors,indices
```

Ilustración 7. Función propuesta para obtener los vecinos cercanos.

- Función most_common: realiza el conteo de la cantidad de ceros y unos. Esta función requiere los valores de la columna de etiquetas.

```
def most_common(output_values):
    return max(set(output_values), key=output_values.count)
```

Ilustración 8. Función propuesta most_common.

- Función predict_classification: realiza el método de knn. Esta función requiere como entradas los datos, las etiquetas y el número de n vecinos.

```
def predict_classification(x,y, k):
    predict = []
    for j in range(0,len(x)):
        # Inicializacion de las distancias.
        distances = []
        x = np.array(x)
        for i ,example in enumerate(x):
            distance = euclidean_distance(example,x[j])# Calculate the Euclidean distance between two vectors
            distances.append((distance, i))
        distances.pop(j)
        distances_ord = sorted(distances)
        neighbors,indices = get_neighbors(k,distances_ord)
        output_values = y.iloc[indices]
        output_values = output_values.to_numpy().tolist()
       # print(type(output_values))
        #print(output_values)
        predict.append(most_common(output_values))
    return predict
```

Ilustración 9. Función propuesta para el método de knn.

- Función entrenamiento: se enfoca en obtener la mejor k y el mejor valor de k. Esta función requiere como entradas los conjuntos de entrenamiento y n.

```
def entrenamiento(x_,y_,n):
  error_rate = []
  mejor_k = []

  for k in range(2,n+1):
    pred_i = predict_classification(x_train.values.tolist(),y_train,k)
    mejor_k.append(k)
    error_rate.append(np.mean(pred_i != y_train))
  e = pd.DataFrame(error_rate)
  e_min = e.min()[0]
  idx_k = error_rate.index(e_min)
  best_k_value = mejor_k[idx_k]
  print(mejor_k)

  plt.figure(figsize=(10,6))
  plt.plot(mejor_k,error_rate,color='blue', linestyle='dashed', marker='o',
  markerfacecolor='red', markersize=10)
  plt.title('Error Rate vs. K Value')
  plt.xlabel('K')
  plt.ylabel('Error Rate')
  return best_k_value, mejor_k
```

Ilustración 10 1.Función propuesta para la obtención del nodo final.

- Función kfold: realiza la separación del conjunto de datos de acuerdo al número de k. Esta función requiere como entradas los conjuntos de datos, etiquetas, k y i.

```
def kfold(x,y,k,i):
    section = int(len(x)/k)
    x_test = x.iloc[i * section : (i+1) * section, :]
    x_train = x_.drop([i * section,((i+1) * section)-1 , 1],axis=0)

    y_test = y.iloc[i * section : (i+1) * section]
    y_train = y_.drop([i * section,((i+1) * section)-1 , 1],axis=0)

    return x_test, y_test, x_train, y_train
```

Ilustración 11 2.Función propuesta para la obtención del nodo final.

- Función method_602020: realiza la separación del conjunto de datos de acuerdo al porcentaje de 60% para el conjunto de entrenamiento, 20% para el conjunto de validación y 20% para el conjunto de prueba. Esta función requiere como entradas los datos y las etiquetas.

```
def method_602020(x,y):
    train_x = x[0 : int(len(x)*0.6)]
    train_y = y[0 : int(len(y)*0.6)]
    val_x = x[int(len(x)*0.6) : int(len(x)*0.8)]
    val_y = y[int(len(y)*0.6) : int(len(y)*0.8)]
    test_x = x[int(len(x)*0.8) : ]
    test_y = y[int(len(y)*0.8) : ]
    return train_x, train_y, val_x, val_y, test_x, test_y
```

Ilustración 12 3.Función propuesta para la obtención del nodo final.

- Función métricas: realiza la evaluación del modelo por medio de diversas métricas. Esta función requiere como entradas los datos predichos y los datos reales.

```
def metricas(claseP,true_train):
    TP = 0
    FP = 0
    TN = 0
    FN = 0
    #Determinar los TP,TN,FP y FN para la matriz de confusión del entrenamiento
    for i in range(len(claseP)):
        if (true_train[i] == claseP[i]) and true_train[i] == 1:
            TP += 1
        elif (true_train[i] == claseP[i]) and true_train[i] == 0:
            TN += 1
        elif (true_train[i] != claseP[i]) and true_train[i] == 0:
            FP += 1
        else:
            FN += 1

    accuracy = ((TP+TN)/(TP+TN+FP+FN))

    if TP + FP != 0:
        precision = TP/(TP+FP)
    else:
        precision = 0
    if TP + FN != 0:
        sensitividad = TP/(TP+FN)
    else:
        sensitividad = 0
    if precision != 0 and sensitividad != 0:
        f1 = (2*TP)/(2*TP+FP+FN)
    else:
        f1 = 0
    return [accuracy, precision, sensitividad, f1]
```

Ilustración 13. Función propuesta para la evaluación del modelo.

Diagrama de metodología

Para el análisis de cualquier base de datos se deben seguir los siguientes pasos:

1. Recolección: Definir y cargar la base de datos a utilizar.
2. Preparar: Comprobar la no existencia de datos faltantes, identificación de outliers, así como la normalización de los datos.
3. Analizar: Conocer la distribución de la información por cada atributo, reducción de dimensionalidad, división de los conjuntos de entrenamiento y prueba del modelo por medio de un kfolds 5 y un k-fold 10.
4. Etapa de entrenamiento y prueba: Implementación del modelo de k-nn. Así bien, se calcula la métrica de exactitud, precisión, sensitividad y F1 Score para conocer el desempeño obtenido.

A continuación, se muestra el diagrama de flujo que representa el proceso de desarrollo, el cual fue implementado en un programa basado en lenguaje Python.

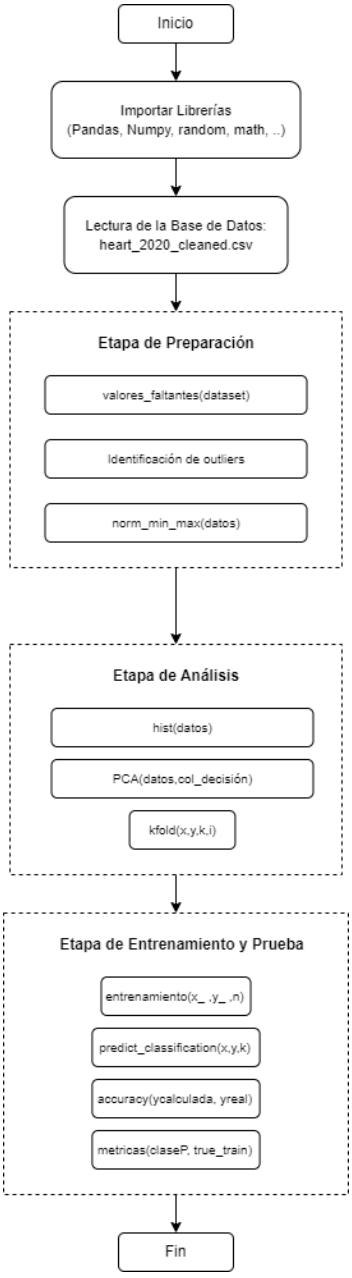

Ilustración 14. Diagrama de flujo para el análisis del conjunto de datos.

Resultados y discusión

A continuación, se presentan las distribuciones de los atributos de la base de datos en cuestión, después de haberle realizado un subsampling del 5%, al cual se le efectuó una preparación, la cual comprende los siguientes puntos: la conversión de características lingüísticas a valores categóricos, la comprobación de la no existencia de datos faltantes, la identificación de outliers y la normalización de los datos.

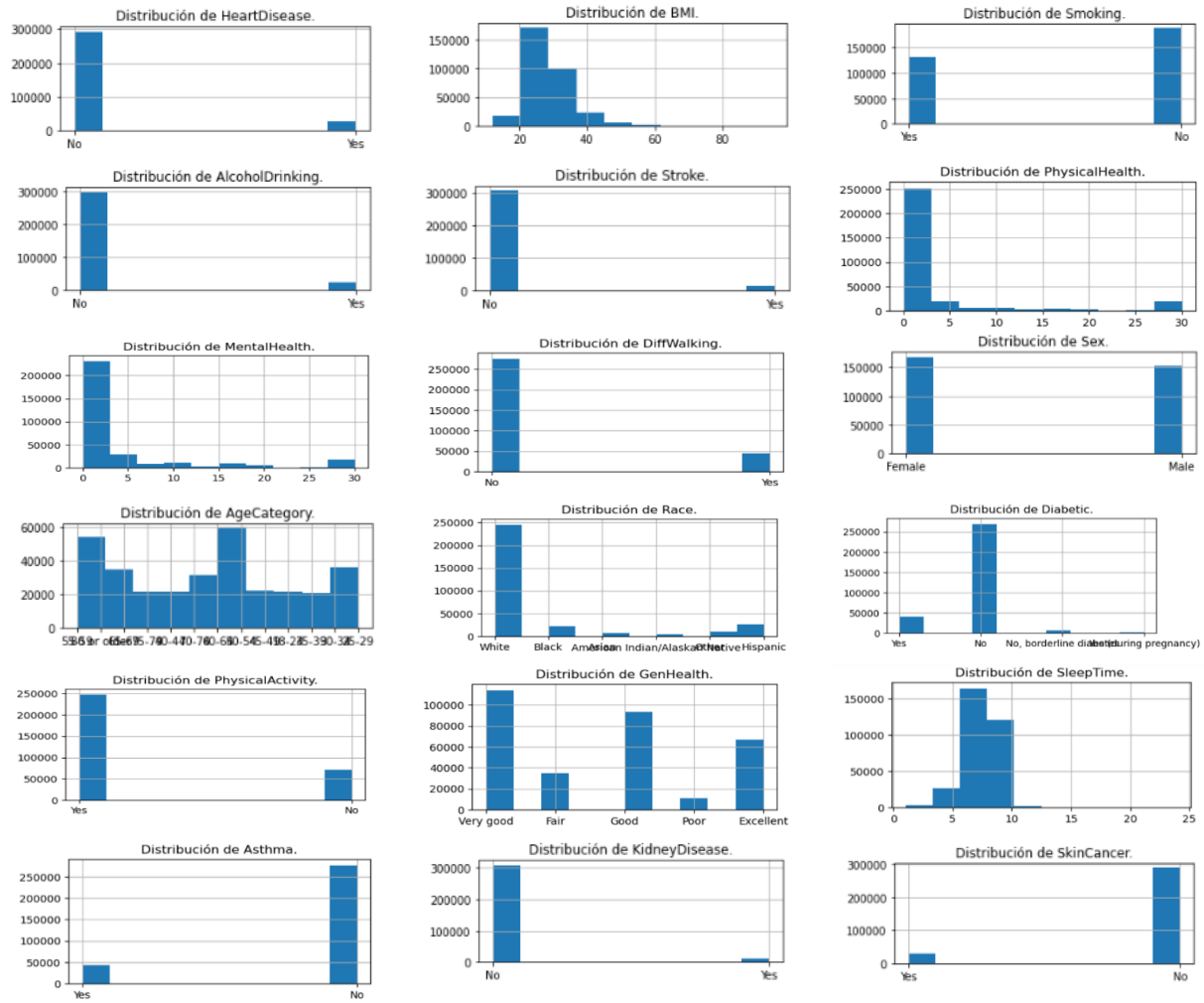

Ilustración 15. Distribución de los atributos de la base de datos.

A partir de haber implementado la técnica de análisis de componentes principales (PCA) en la base de datos en cuestión, se logró reducir la cantidad de atributos de 18 a 12, entre los atributos que contenían la mayor información se encuentran: BMI, PhysicalHealth, MentalHealth, AgeCategory, Race, Diabetic, Smoking, AlcoholDrinking, Stroke, DiffWalking y PhysicalActivity. Así bien, para los siguientes pasos se incluye a estos atributos seleccionados el atributo de decisión HeartDisease.

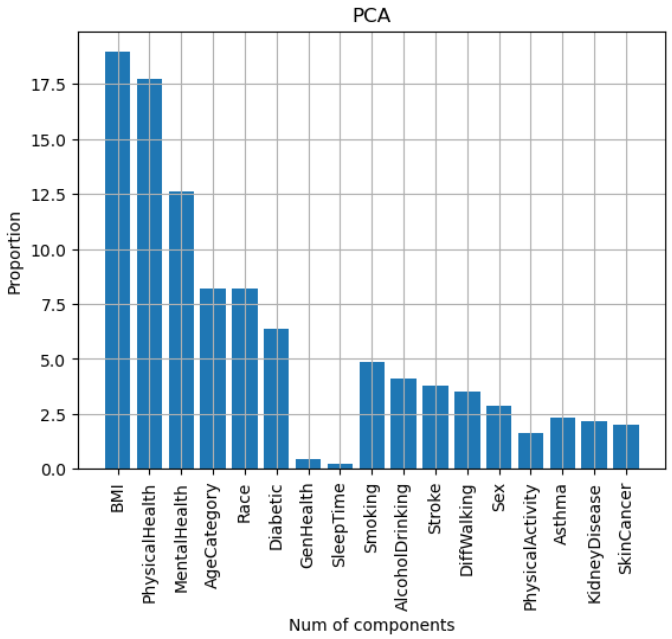

Ilustración 16. Atributos destacados del análisis de componentes principales (PCA).

Antes de proseguir con la implementación del algoritmo de clasificación será necesario dividir el conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba. En las siguientes tablas se presentan los totales de los datos para un cada uno de los conjuntos, en donde se utilizó un k-fold de 5, un k-fold de 10 y una validación en línea 60x20x20.

Tabla 1. Conjunto de Entrenamiento y Prueba para un k-fold 5.

	Conjunto de Entrenamiento	Conjunto de Prueba
Total de datos	15987	3198

Tabla 2. Conjunto de Entrenamiento y Prueba para un k-fold 10.

	Conjunto de Entrenamiento	Conjunto de Prueba
Total de datos	15987	1599

Tabla 3. División 60x20x20 para los conjuntos de Entrenamiento, Validación y Prueba.

	Conjunto de Entrenamiento	Conjunto de Validación	Conjunto de Prueba
Total de datos	9594	3198	3198

KNN

Como resultado de un barrido de valores de k para encontrar el valor, tal que, disminuya la tasa de error en la clasificación del algoritmo KNN después de un análisis de componentes principales, se obtuvo la siguiente gráfica, en donde se puede observar el incremento del error cuando aumenta el número de k vecinos.

Para un K-fold 5

En la Ilustración, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución del kfold 5.

```
100%|██████████| 5/5 [9:29:54<00:00, 6838.94s/it]
```

Ilustración 17. Tiempo utilizado en kfold 5.

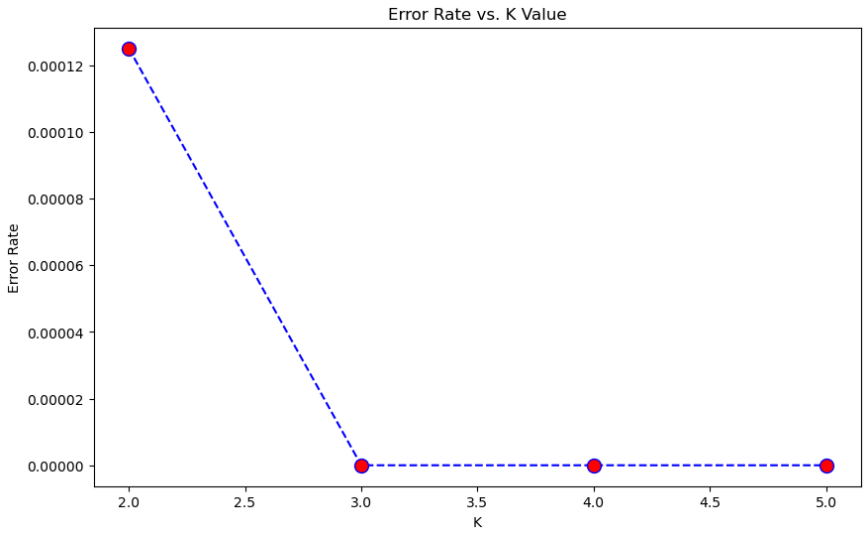

Ilustración 18. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 4.

Tabla 4. Resultados obtenidos en la clasificación en la etapa de prueba.

	Exactitud kfold-cross	Precisión kfold-cross	Sensitividad kfold-cross	F1 Score kfold-cross
Prueba	0.99987	1.0	0.99845	0.99922

Para un K-fold 10

En la Ilustración, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución del kfold 10.

```
100%|██████████| 10/10 [19:55:05<00:00, 7170.58s/it]
```

Ilustración 19. Tiempo utilizado en kfold 10.

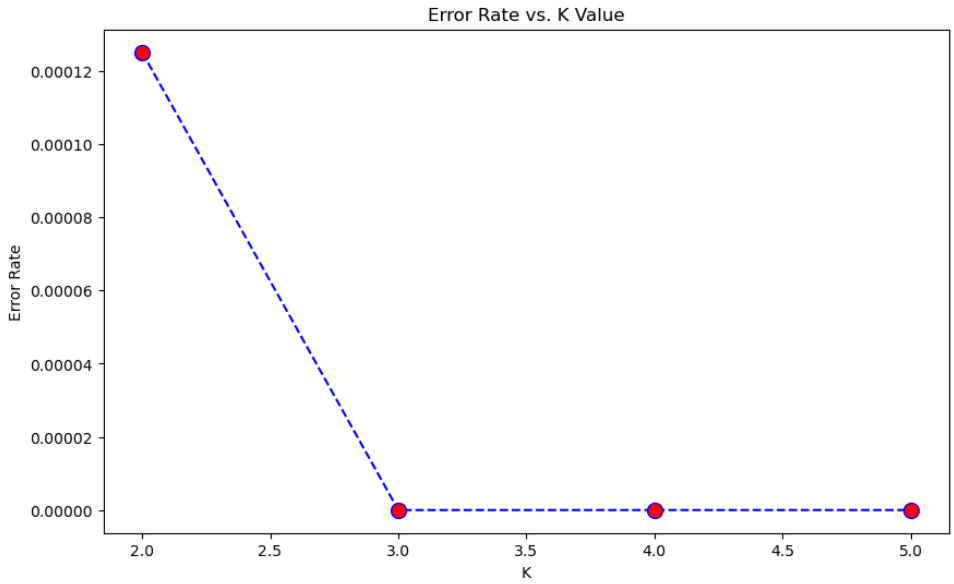

Ilustración 20. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de prueba del modelo, obteniendo los resultados mostrados en la Tabla 5.

Tabla 5. Resultados obtenidos en la clasificación en la etapa de prueba.

	Exactitud kfold-cross	Precisión kfold-cross	Sensitividad kfold-cross	F1 Score kfold-cross
Prueba	0.99949	1.0	0.99387	0.99692

Para una validación 60x20x20

En la Ilustración, se puede observar el tiempo que tardo el modelo en la etapa de entrenamiento para la ejecución de la validación 60x20x20.

```
100%|██████████| 4/4 [39:08<00:00, 587.22s/it]
```

Ilustración 21. Tiempo utilizado en validación 60x20x20.

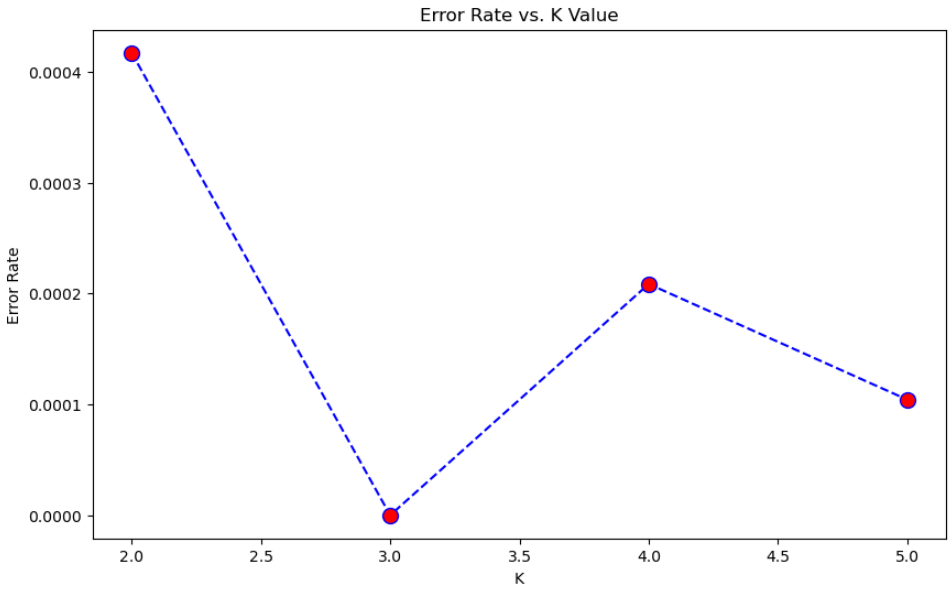

Ilustración 22. Visualización k mejor.

Una vez seleccionado el valor de k = 3, se continuó con la etapa de validación y prueba del modelo, obteniendo los resultados mostrados en la Tabla 6.

Tabla 6. Resultados obtenidos en la clasificación en la etapa de validación y prueba.

	Conjunto de Validación	Conjunto de Prueba
Exactitud	1.0	0.99968
Precisión	1.0	1.0
Sensitividad	1.0	0.99632
F1 Score	1.0	0.99815

En la siguiente tabla, se puede realizar una comparación entre las diferentes pruebas realizadas con respecto a los resultados obtenidos en cada una de las métricas, en donde se puede observar que los resultados fueron superiores al 98%.

Tabla 7. Comparación de los resultados obtenidos en kfold 5, kfold 10 y validación 60x20x20.

	Kfold 5	Kfold 10	Validación 60x20x20
Exactitud	0.99987	0.99949	0.99968
Precisión	1.0	1.0	1.0
Sensitividad	0.99845	0.99387	0.99632
F1 Score	0.99922	0.99692	0.99815

Sin embargo, en cuestión al tiempo de entrenamiento y prueba entre las diferentes pruebas realizadas se observó un tiempo computacional muy notable por parte del kfold 10 en comparación del kfold 5 y la validación 60x20x20.

Tabla 8. Comparación del tiempo de ejecución en kfold 5, kfold 10 y validación 60x20x20.

	Kfold 5	Kfold 10	Validación 60x20x20
Tiempo de ejecución	9:29:54	19:55:05	00:39:02

Conclusiones

En este programa generalizado inicialmente se generaron funciones que permitieran conocer el comportamiento de los datos mediante la implementación del algoritmo KNN, así como funciones que permiten obtener métricas estadísticas de los datos. Así bien, en comparación con otros algoritmos de machine learning, este requiere de demasiados recursos computaciones, como se pudo comprobar anteriormente ya que fue necesario la utilización de un subsampling del conjunto de datos para poder llevar a cabo esta tarea aun así el tiempo de ejecución fue considerablemente tardado si se compara con otros algoritmos.

En conclusión, como se pudo observar en el presente trabajo con respecto a las formas para evaluar el rendimiento de un modelo de Machine Learning, se probó primeramente con la estrategia de validación de k-fold, la cual permite garantizar que todas las observaciones de la

serie de datos original tengan la oportunidad de aparecer en la serie de entrenamiento y prueba; posteriormente se implementó el enfoque de validación 60x20x20, la cual es eficaz. Sin embargo, puede presentarse el caso de faltar alguna información contenida en los datos que no se utilizan para el entrenamiento y, por tanto, los resultados pueden tener un gran sesgo.

Referencias

[1] M. Antonio and A. Fernández, *Inteligencia artificial para programadores con prisa by Marco Antonio Aceves Fernández - Books on Google Play*. Universo de Letras. [Online]. Available: https://play.google.com/store/books/details/Inteligencia_artificial_para_programadores_con_pri?id=ieFYEAAAQBAJ&hl=en_US&gl=US

[2] "Personal Key Indicators of Heart Disease | Kaggle." https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease/code (accessed Aug. 25, 2022). https://learn.microsoft.com/es-es/azure/machine-learning/component-reference/smote