"Año del Fortalecimiento de la Soberanía Nacional"

UNIVERSIDAD DE PIURA

FACULTAD DE INGENIERÍA

MACHINE LEARNIG Y DEEP LEARNING CON PYTHON

GRUPO 08

"Aplicación de modelos de clasificación para la predicción de la posibilidad de sufrir un ataque cerebrovascular"

DOCENTE: Ing. Pedro Rotta Saavedra

INTEGRANTES:

- Flores Guerrero, Abdebel
- García Chinguel, Emersson Aldhair
- Jaramillo Baca, Jenner Jeampierre
- More Sernaque, Jhon Darwin
- Pacherres Chávez, Eliane Ethel
- Vilca Aguilar, Luis Angel

INFORME FINAL

Piura, 6 de febrero de 2022

CONTENIDO

1 INTRODUCCIÓN 3
2 PLANTEAMIENTO DEL PROBLEMA 3
3 ANÁLISIS DEL PROBLEMA 4
4 ANÁLISIS DE RESULTADO 4
4.1 Análisis estadístico 4
4.2 Gráficas estadísticas 7
4.3 Gráficas analíticas 9
4.4 Entrenamiento del modelo 9
4.4.1 Regresión Logística 9
4.4.2 K_Nearest Neighbors (KNN) 10
4.4.3 SVC 11
4.4.4 Random Forest 12
4.5 Programa principal 13
5 CONCLUSIONES 15

1 INTRODUCCIÓN

Un accidente cerebrovascular ocurre cuando el suministro de sangre a una parte del cerebro se interrumpe o se reduce, lo que impide que el tejido cerebral reciba oxígeno y nutrientes. Las células cerebrales comienzan a morir en minutos. Se le considera una emergencia médica, y el tratamiento oportuno es crucial. La acción temprana puede reducir un daño crítico y otras complicaciones.

Para evaluar el riesgo de contraer esta anormalidad se consideran los siguientes factores: la hipertensión, niveles de colesterol, diabetes, dolores de cabeza o dolores musculares. Así mismo, se recomienda que la persona no fume, mantenga una dieta saludable, realice ejercicio y no consumir alcohol.

El presente proyecto busca predecir si la persona es propensa a sufrir un accidente cerebrovascular a partir de una base de datos de pacientes. Esta data se tomó de “Kaggle” y presenta 5110 datos con 12 columnas. ID, como identificador del paciente, género, edad, estado civil, tipo de trabajo y tipo de residencia como datos personales, hipertensión, dolor de cabeza, niveles de glucosa, índice de masa corporal y si es un fumador.

2 PLANTEAMIENTO DEL PROBLEMA

Este trabajo usará 4 modelos de Machine Learning de clasificación. Pedirá al usuario los parámetros suficientes para que dicho modelo prediga si el paciente es propenso a sufrir este tipo de accidente.

Como se dijo antes, la base de datos con la que se ha trabajado ha sido tomada de Kaggle (https://www.kaggle.com/datasets)

Asimismo, se ha considerado: análisis estadístico con seaborn, pandas, matplotlib o plotly, gráficas estadísticas y analíticas con matplotlib o Plotly y un programa en terminal que dé el valor de predicción para cualquier dato.

3 ANÁLISIS DEL PROBLEMA

Se busca realizar un modelo predictivo competente y beneficioso para las personas por lo que el análisis de las variables y la data es fundamental.

Las 2 primeras interrogantes que nos planteamos es: ¿Tenemos datos faltantes? ¿Se requiere un Label Enconding o Hot Enconding? Para lo cual, se ha respondido que sí y se ha procedido a realizar un remplazo de datos por la media y un Label Enconding o Hot Enconding donde se ha considerado necesario.

Seguidamente, ¿Están estandarizados los datos? ¿Se requiere escalar? En este caso se requerían ambos procedimientos y después de realizados pudimos concluir que se podía iniciar con la creación de modelos, así como la generación de gráficas estadísticas para el análisis de las variables y sus relaciones.

4 ANÁLISIS DE RESULTADO

4.1 ANÁLISIS ESTADÍSTICO

Se observo que el dataset de *stroke predictions* presentaba data faltante y datos sin estandarizar y escalar, por lo cual se decidió realizar un análisis de preprocesamiento para poder tener una data equilibrada.

Obteniendo el dataset

```
[ ] df_strokes = pd.read_csv('https://raw.githubusercontent.com/CoolfaceJerkCity/Grupo_N_UDEP_Python/main/healthcare-dataset-stroke-data.csv')
    df_strokes.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Figura 1 Procesamiento - Obtención de la data (Elaboración propia)

En el caso de la data faltante, optamos por eliminar las filas del dataframe, para evitar errores futuros, y lo realizamos utilizando dropna() de la librería pandas.

Eliminando missing values

```
df_strokes = df_strokes.dropna() # Borrando datos nulos
df_strokes.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1

Figura 2 Procesamiento - Eliminación de los valores faltantes (Fuente: Elaboración propia)

Encontramos también óptimo el aplicar *hot encoding* al *dataframe* a través de *labeling*, sin antes redimensionar la data de 4 dimensiones a un valor binario, con condicionales y funciones de la librería pandas.

HotEncoding / Labeling / Downgrading

```
[ ] #Convirtiendo smoking status y gender a un valor binario
    df_strokes['smoking_status'] = df_strokes['smoking_status'].replace({'formerly smoked' or 'smokes':'smoked','never smoked' or 'Unknown':'non_smoking'})
    df_strokes['smoking_status'] = [1 if i.strip() == 'smoked' else 0 for i in df_strokes.smoking_status]
    df_strokes['gender'] = [1 if i.strip() == 'Male' else 0 for i in df_strokes.gender]
    df_strokes = df_strokes.drop(columns=['id', 'work_type'])
    df_strokes.head()
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	1	67.0	0	1	Yes	Urban	228.69	36.6	1	1
2	1	80.0	0	1	Yes	Rural	105.92	32.5	0	1
3	0	49.0	0	0	Yes	Urban	171.23	34.4	0	1
4	0	79.0	1	0	Yes	Rural	174.12	24.0	0	1
5	1	81.0	0	0	Yes	Urban	186.21	29.0	1	1

Figura 3 Procesamiento - Codificación binaria parte I (Fuente: Elaboración propia)

Acá establecimos una función para el *labeling*, el cual toma como parámetros el *dataframe* y las columnas a aplicar.

```
[ ] from sklearn.preprocessing import LabelEncoder

    def labelingData(df, columns):
      try:
        for column_name in columns:
          X_feat_data = df[column_name]#Recibiendo datos de la columna
          X_feat_data_array = np.array(X_feat_data)
          print(X_feat_data_array) #Se imprime los datos sin normalizar

          encoder = LabelEncoder()#Aplicando labeling
          encoder.fit(X_feat_data_array)
          X_encoder = encoder.fit_transform(X_feat_data_array)
          print("="*40)
          print(X_encoder)#Se imprime los datos ya normalizados

          df[column_name] = X_encoder
          print("="*40)
          print(f'La columna {column_name}_lbl fue agregada al dataframe correctamente')

        return df

      except:
        print("No se actualizo el dataframe debido a un error inesperado =(")
```

Figura 4 Procesamiento - Codificación binaria parte II (Fuente: Elaboración propia)

Una vez aplicado el labeling en todos los casos posibles, realizamos estandarización con la función *StandarScaler* de *Sklearn* para el dataframe completo.

Estandarización de datos

```
from sklearn.preprocessing import StandardScaler
#Estandarizando la data
scaler = StandardScaler().fit(X)
X_standarized = scaler.transform(X)

print(X[0, :])
print('='*50)
print(X_standarized[0, :])
```

Figura 5 Procesamiento - Estandarización (Fuente: Elaboración propia)

Luego aplicamos escalamiento con la función *MinMaxScaler* de Sklearn a la data ya estandarizada.

Escalamiento de datos

```
[ ]  from sklearn.preprocessing import MinMaxScaler as MMS
     scaler2 = MMS(feature_range = (0, 1))
     X_scaled = scaler2.fit_transform(X_standarized)

     print(X_standarized[0, :])
     print('='*50)
     print(X_scaled[0, :])
```

```
[ 1.20044746  1.07013796 -0.31806673  4.38196829  0.72948428  0.98563987
  2.77769839  0.98134488  2.20567315]
==================================================
[1.         0.81689453 0.         1.         1.         1.
 0.80126489 0.30126002 1.        ]
```

Figura 6 Procesamiento - Escalamiento (Fuente: Elaboración propia)

Una vez estandarizado y escalado observamos la matriz de dispersión de nuestro *dataframe*. En ella, se aprecia que de todas las variables que componen el "feature" solo la edad y el "estado de casado" tienen un grado significativo de correlación. Por tanto, se podría quitar uno de los parámetros sin perder precisión en el modelo, aquí se optará por mantener la edad.

Observando la matriz de correlacion estandarizada/escalada

```
corr_data = new_df.corr()
sns.clustermap(corr_data,annot= True,fmt = '.2f')

plt.title('Correlacion entre features')
plt.show();
```

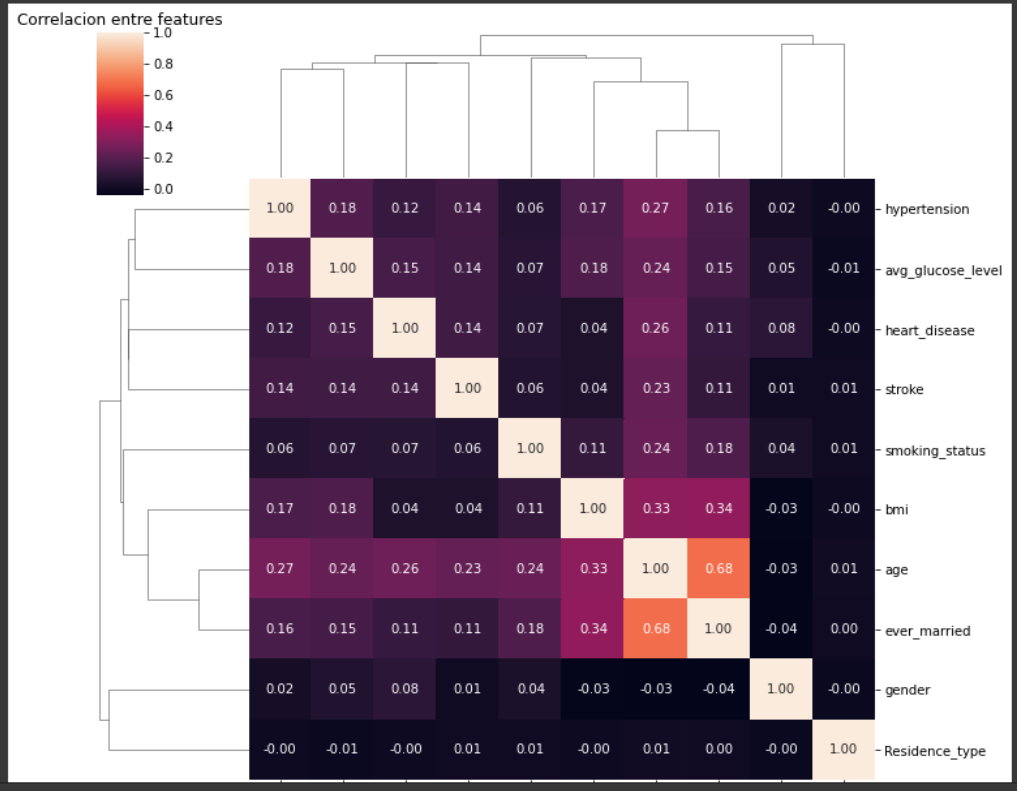

Figura 7 Matriz de la multicolinealidad (o correlación) entre las variables categóricas (Fuente: Elaboración propia)

Por último, se exporto a un nuevo **csv** para su próximo entrenamiento y modelamiento.

Exportando el dataframe estandarizado a un nuevo CSV

```
[ ]  new_df.to_csv('df_strokes_standarized.csv', index=False)
```

Figura 8 Data final para el análisis de clasificación (Fuente: Elaboración propia)

4.2 GRÁFICAS ESTADÍSTICAS

En el presente apartado se proporciona 3 tipos de gráficas: de barras e histogramas Para la primera, se usó la librería *seaborn* y, sobre todo, en las variables discretas de la data tal como se muestra en la siguiente figura.

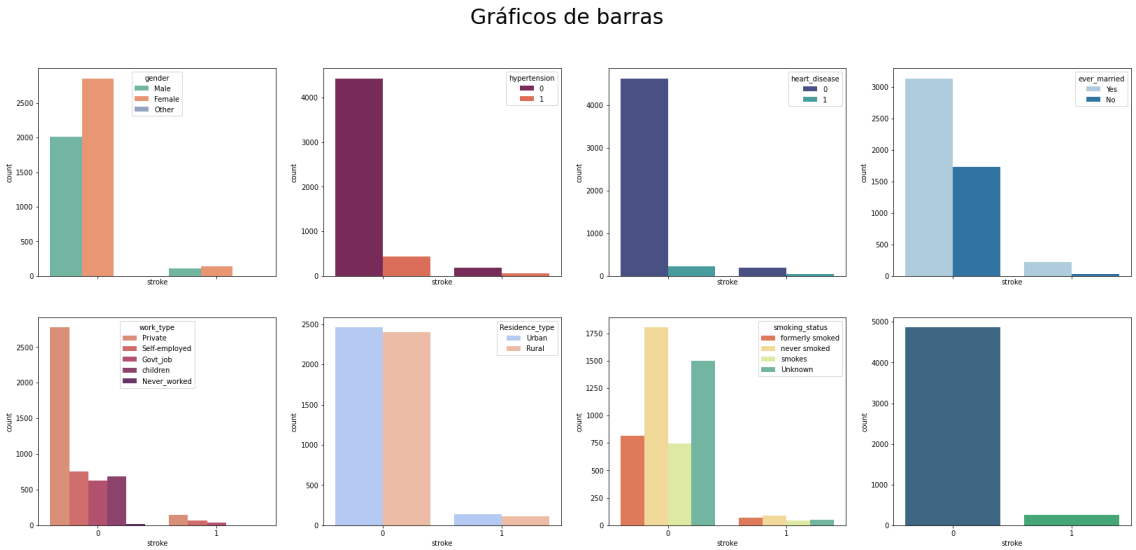

Figura 9: Gráficas de barras para las variables categóricas discretas (Fuente: Elaboración propia)

De la figura es importante mencionar que el valor "1" del eje de las abscisas de cada gráfica representa a las personas que han sufrido un accidente cardiovascular, o derrame cerebral, mientras que el valor "0" indica lo contrario. Cada gráfico de barras en la figura muestra la distribución de los datos según variables categóricas como género, hipertensión, etc.

Los histogramas se emplearon en las variables de tipo continuo y se usó tanto la librería anterior como la *"plotly",* de la cual se obtuvieron las siguientes figuras.

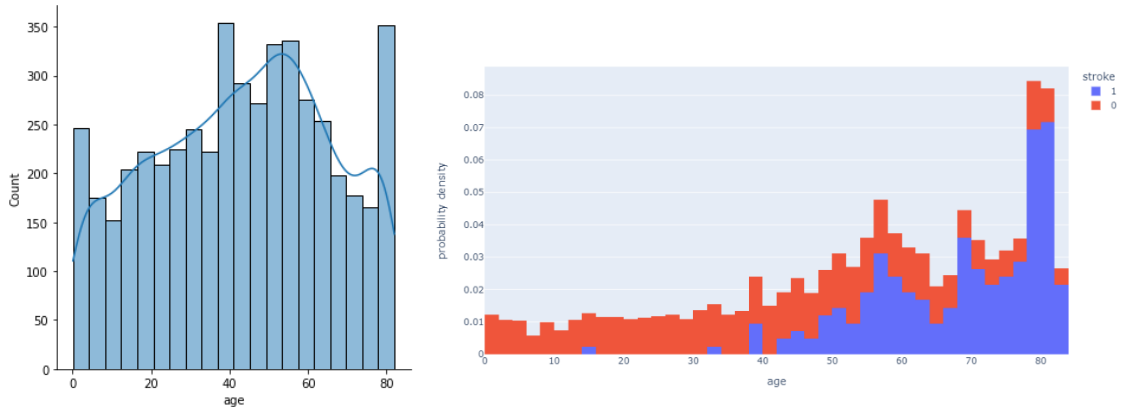

Figura 10: Histograma de los derrames cerebrales según la edad (Fuente: Elaboración propia)

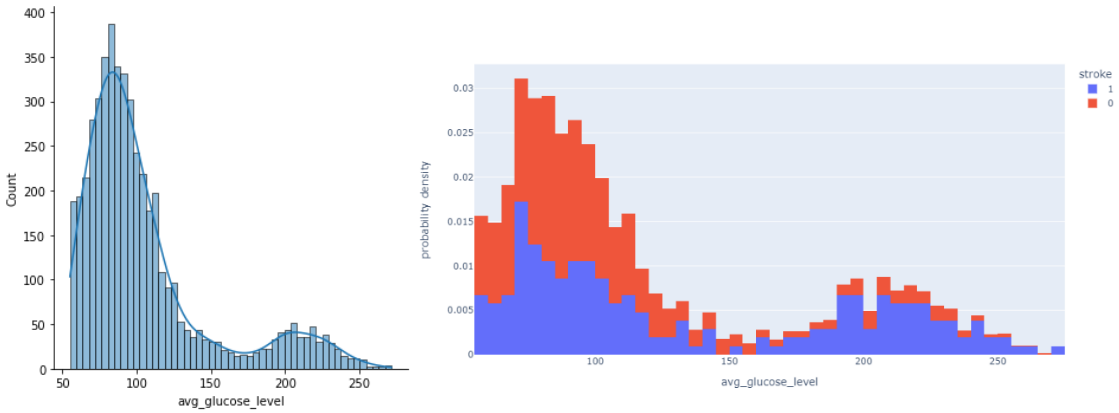

Figura 11: Histograma de los derrames cerebrales según el nivel de glucosa (Fuente: Elaboración propia)

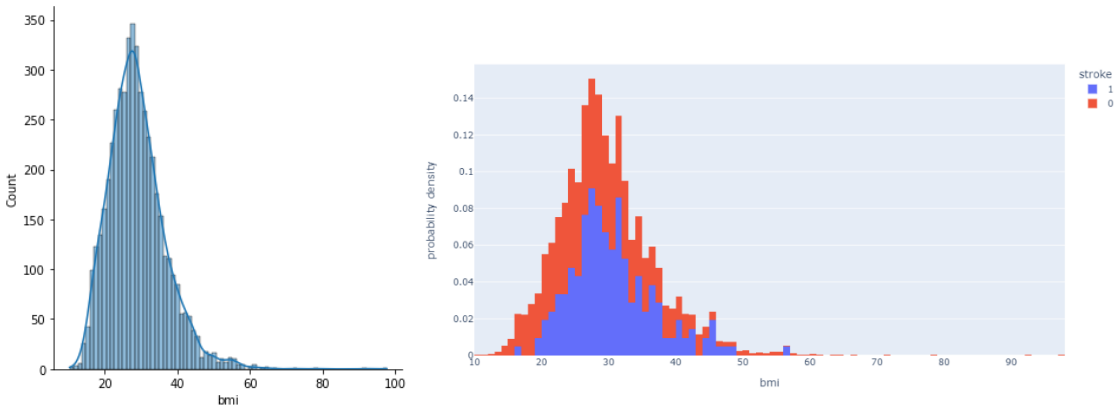

Figura 12 : Histograma de los derrames cerebrales según el índice de masa corporal (Fuente: Elaboración propia)

En las gráficas se puede apreciar que la probabilidad de padecer de un derrame cerebral gira en torno a determinadores valores para cada categoría, esto se puede constatar con las predicciones que se harán en los apartados posteriores.

4.3 GRÁFICAS ANALÍTICAS

Se ha iniciado con el método del Codo de Jambu y de acuerdo con la gráfica, deductivamente, se ha identificado el mejor K con el valor de 2. Es por lo que, se ha realizado una gráfica únicamente con dicho valor, observando unos datos alineados con pendiente negativa.

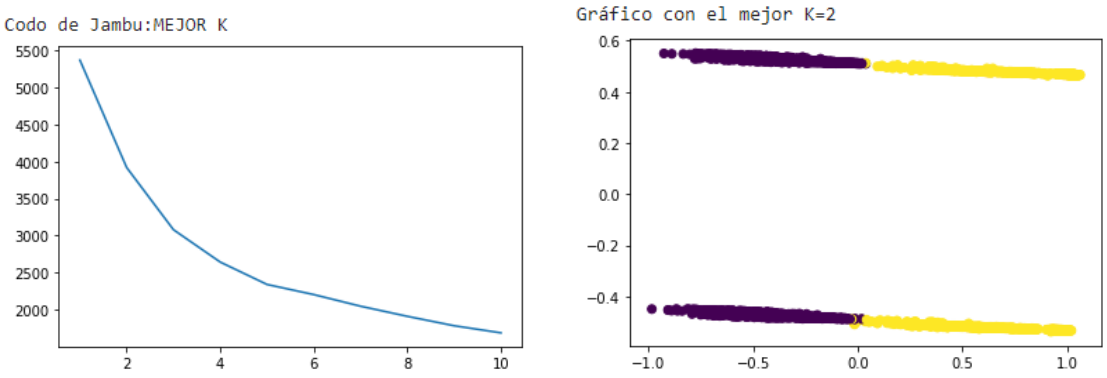

Figura 13: Codo de Jambu (izquierda) y agrupamiento correspondiente [derecha] (Fuente: Elaboración propia)

4.4 ENTRENAMIENTO DEL MODELO

Como siguiente paso del algoritmo de Machine Learning es crear los modelos de regresión. Debido a la data escogida, el tipo de regresión es de clasificación. Para ello se han considerado 4 modelos, los cuales se explicarán a continuación, junto al código.

4.4.1 Regresión Logística

El primer modelo para considerar es la Regresión Logística cuyo código se muestra en la figura 14. Se ha creado el modelo y calculado las métricas según lo estudiado.

Los resultados que se obtuvieron se muestran en la misma figura. Según sus métricas, el modelo se ajusta de forma casi exacta a los datos, pues son mayores al 90% y no existe diferencia notable entre el entrenamiento y la validación. Además, en la matriz de confusión los valores dados lo confirman.

Logistic Regression

```
[138] modeloLR = LogisticRegression(solver = "liblinear")
      modeloLR.fit(Xtrain, ytrain)

      LogisticRegression(solver='liblinear')

[139] testscore_lr = accuracy_score(ytest, modeloLR.predict(Xtest))
      accuracy_score(ytest, modeloLR.predict(Xtest))

      0.7822222222222223

[140] crosscore_lr =cross_val_score(modeloLR, Xtest, ytest, cv = 10).mean()
      cross_val_score(modeloLR, Xtest, ytest, cv = 10).mean()

      0.7683794466403162
```

Figura 14 Código y resultados del modelo Regresión logística (Fuente: Elaboración propia)

4.4.1.1 Gráficas analíticas para Regresión Logística

Se puede observar que la recta presenta forma de 'v', sin embargo, los datos si están cerca de la recta, permitiendo afirmar que los datos se ajustan adecuadamente.

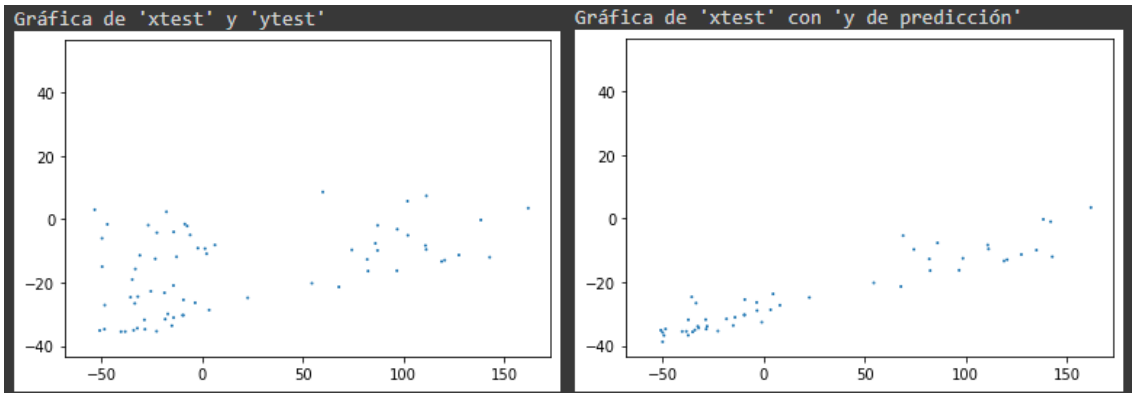

Figura 15 Gráficas de dispersión para los datos de validación y de predicción (Fuente: Elaboración propia)

4.4.2 K_Nearest Neighbors (KNN)

También conocido como los k_vecinos más cercanos y el código se muestra en la figura 16. Como el modelo anterior, también se han calculado sus métricas. Cabe recalcar que el parámetro k de este modelo se ha considerado igual a 14, pues mostraba mejores resultados que los demás. Se puede apreciar unas métricas excelentes, pues prácticamente no varían en el entrenamiento ni en la validación. Lo confirma la matriz de confusión cuyos valores verdaderos positivos y verdaderos negativos son la mayoría respecto de los falsos positivos y falsos negativos.

KNN

```
[142] knn = KNeighborsClassifier(n_neighbors=14)
      knn_tuned = knn.fit(Xtrain, ytrain)
      knn_tuned.score(Xtest, ytest)

      0.7022222222222222

[143] y_pred = knn_tuned.predict(Xtest)
      testscore_knn =accuracy_score(ytest, y_pred)

[144] y_pred = knn_tuned.predict(Xtest)
      crosscore_knn=accuracy_score(ytest, y_pred)
      accuracy_score(ytest, y_pred)

      0.7022222222222222
```

Figura 16 Código y resultados del modelo KNN (Fuente: Elaboración propia)

4.4.2.1 Gráficas analíticas para KNN

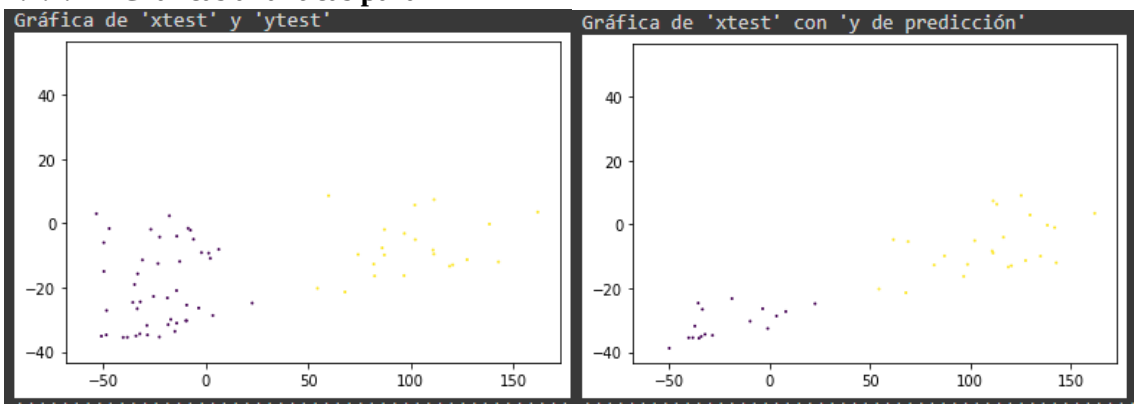

Figura 17 Gráficas de dispersión para los datos de validación y de predicción (Fuente: Elaboración propia)

Se ha realizado un bucle para poder identificar el mejor K y aunque todos se encuentran en valores mayor al 90%, con los valores 1 y 3 se obtiene un mejor resultado. Asimismo, se ha realizado una gráfica de validación con los resultados obtenidos.

4.4.3 SVC

Ahora se analiza el modelo de SVC (Support Vector Classifier) y para ello se parte de la repartición de los datos para entrenamiento y testeo usados en los modelos anteriores, el código usado es el siguiente.

SVC

```
[146] svc_tuned = SVC(kernel = "linear", C = 1).fit(Xtrain, ytrain)
      y_pred = svc_tuned.predict(Xtest)
      crosscore_svm = accuracy_score(ytest, y_pred)

[147] testscore_svm=accuracy_score(ytest, y_pred)
      accuracy_score(ytest, y_pred) #test acc

      0.7733333333333333
```

Figura 18 Código y resultados del modelo SVC (Fuente: Elaboración propia)

4.4.3.1 Gráficas analíticas para SCV

La gráfica realizada para este modelo es una conformada por una recta, que elige el límite de decisión que maximiza la distancia desde los puntos de datos más cercanos de todas las clases. Los puntos más cercanos al límite de decisión se denominan vectores de soporte y las líneas punteadas son una división para separar los límites de los puntos en los que se incluyen los vectores de soporte.

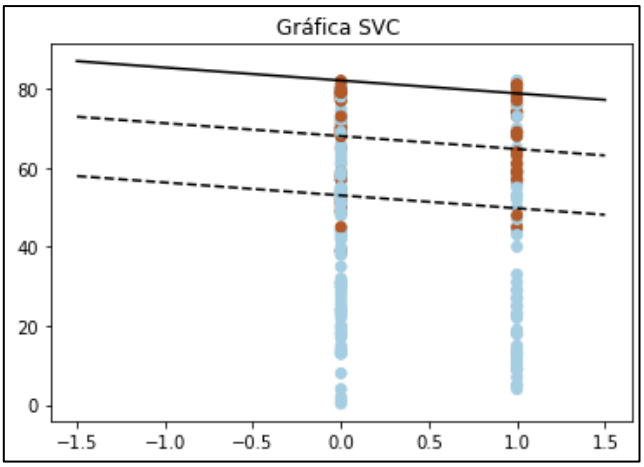

Figura 19 Gráfica del modelo SVC (Fuente: Elaboración propia)

4.4.4 Random Forest

Finalmente se tiene el análisis realizado para el algoritmo de Random Forest y para lo cual el código se tiene a continuación:

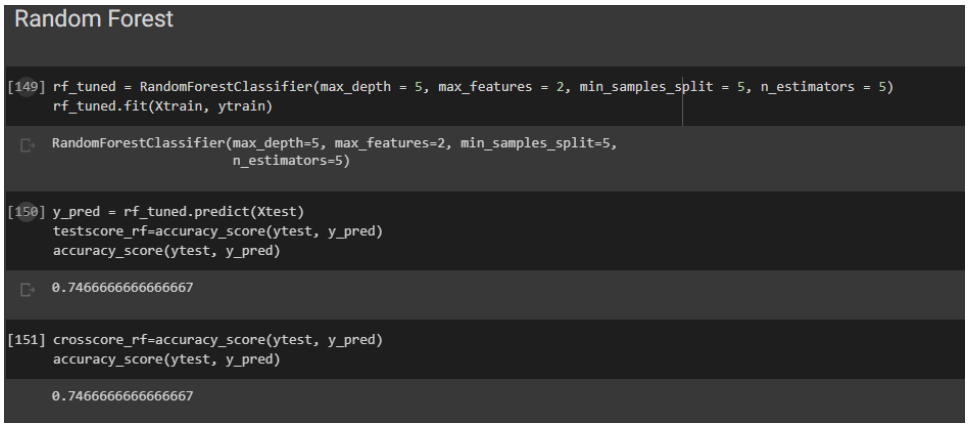

Figura 20 Código y resultados del modelo Random Forest (Fuente: Elaboración propia)

4.4.4.1 Gráficas analíticas para Random Forest

Para Random Forest, se ha realizado un histograma para observar la importancia de los árboles en la creación del modelo, observando una distribución log normal.

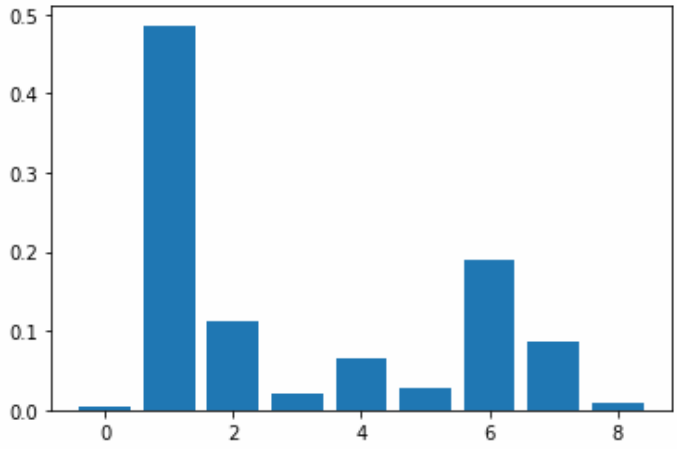

Figura 21 Histograma según el número de árboles (Fuente: Elaboración propia)

4.5 PROGRAMA PRINCIPAL

Después de entrenar y validar los modelos de *machine learning*, se ha realizado un programa en el cual se le ingresen las características del paciente y luego mostrarle si es probable que sufra un derrame cerebral o no. A continuación, en la figura 22 se presenta el programa principal que cuenta con dos funciones que se explicarán en detalle.

```
def main():
  print('Bienvenido al programa')
  print('Ingrese los valores de las siguientes columnas: ')
  paciente = llenardatos()
  seleccionamodelo(paciente)

main()
```

Figura 22 Main del programa (Fuente: Elaboración propia)

La función de llenado de datos pide la información al usuario que se requiere por cada columna del *dataframe stroke* y lo ingresa en una matriz que posteriormente se convertirá en un diccionario al finalizar el llenado de información, esto se puede apreciar en la figura 23. Respecto al programa principal, el diccionario se almacena en la variable paciente y se utiliza en la siguiente función.

```
#Función para llenar datos
def llenardatos():
  info = []
  global df_stroke
  y = df_stroke["stroke"]
  X = df_stroke.drop(["stroke"], axis=1)
  columnas = list(X.columns.values)
  for i in range(0,len(columnas)):

      if columnas[i]== 'gender':
        print('Ingrese 1 si es homnre o 0 si es mujer')
        info.append([lee_floatbin()])

      elif columnas[i]== 'age':
        print('Ingrese su edad')
        info.append([lee_float()])

      elif columnas[i]== 'hypertension':
        print('Ingrese 1 si sufre de hipertensión o 0 si no es el caso')
        info.append([lee_floatbin()])

      elif columnas[i]== 'heart_disease':
        print('Ingrese 1 si sufre de enfermedades cardíacas o 0 si no es el caso')
        info.append([lee_floatbin()])

      elif columnas[i]== 'ever_married':
        print('Ingrese 1 si se ha casado o 0 si no es el caso')
        info.append([lee_floatbin()])

      elif columnas[i]== 'Residence_type':
        print('Ingrese 1 si vive en la ciudad o 0 si no es el caso')
        info.append([lee_floatbin()])

      elif columnas[i]== 'avg_glucose_level':
        print('Ingrese su nivel de glucosa')
        info.append([lee_float()])

      elif columnas[i]== 'bmi':
        print('Ingrese su índice de masa corporal')
        info.append([lee_float()])

      elif columnas[i]== 'smoking_status':
        print('Ingrese 1 si fuma o si no es el caso')
        info.append([lee_float()])

      print(separador)

  paciente = dict(zip(columnas,info))
  print(paciente)
  return paciente
```

Figura 23 Código para ingresar los datos del paciente (Fuente: Elaboración propia)

La última función que se utiliza es la de selección de modelo, la cual permite elegir entre los 4 modelos de machine learning entrenados y permite predecir si el paciente tiene un alto o bajo riesgo de sufrir un derrame dependiendo de los datos que el usuario ha ingresado en la función anterior.

```
def seleccionamodelo(paciente):
  #Panel de opciones
  print('Ingrese el número de modelo que desea emplear')
  print('1 para el modelo Logistic Regresion')
  print('2 para el modelo KNN')
  print('3 para modelo SVC')
  print('4 para modelo Random Forest')

  #Opciones a ingresar

  contador = 0
  opciones = []
  pregunta = 's'

  while pregunta == 'S' or pregunta == 's':

    if contador == 4:
      break

    opciones.append(lee_float2())
    pregunta = input('S para elegir un modelo adicional u otra tecla para salir: ')
    contador +=1

  print(opciones)

  #Ejecutar los modelos

  for i in range(0,len(opciones)):
    datos_paciente = pd.DataFrame(paciente)

    #Modelo Logistic regresion
    if opciones[i] == 1:
      prediction = modeloLR.predict(datos_paciente)

    #Modelo KNN
    elif opciones[i] == 2:
      prediction = knn_tuned.predict(datos_paciente)

    #Modelo SVC
    elif opciones[i] == 3:
      prediction = svc_tuned.predict(datos_paciente)

    #Modelo RF
    elif opciones[i] == 4:
      prediction = rf_tuned.predict(datos_paciente)

    #Predicción
    print(prediction)

    if prediction == 0:
          print('Tiene bajo riesgo de sufrir un derrame')
    else:
          print('Tiene un riesgo elevado de sufrir un derrame')
```

Figura 24 Código para la elección del modelo de predicción (Fuente: Elaboración propia)

5 CONCLUSIONES

El *Machine Learning* (ML)hoy en día es una herramienta tecnológica de gran importancia e impacto en la sociedad. Pues, como en este caso, se puede emplear en la medicina y otros campos en la que se haga uso de una base de datos amplia.

Para el análisis de datos con ML es importante recurrir a varios modelos de regresión y de clasificación. De esta manera se podrá elegir a aquella que mejor se ajuste y menos error tenga en la predicción (métricas).

Para una mejora del procesamiento de los datos en los modelos es útil emplear técnicas de preprocesamiento, como, por ejemplo; normalizaciones, estandarizaciones, escalamiento, etc., tal que con ellas las métricas de los modelos mejoran sustancialmente.

De las métricas obtenidas por cada modelo se llega a la conclusión que el modelo SVC produce las predicciones más precisas de todos, pues se ajusta mejor a la data.

Para este dataset se ha visto conveniente trabajar con solo 750 datos del total. Esto a causa de un comportamiento anormal de los modelos (se sobreajustan), pues en el testeo no predice de manera correcta.

Finalmente, se puede agregar que se ha logrado aplicar los conocimientos y criterios adquiridos en clase para el correcto análisis de los datos de un caso real como el ejemplificado en el presente trabajo.