



PREDICIR UN ACCIDENTE CEREBROVASCULAR

Lucas Aguilera, Claudio Bórquez, Josefa Fernández

Introducción

El problema que va abordar el proyecto es:

Predecir si es probable que un paciente sufra un accidente cerebrovascular en función de los parámetros de entrada como el sexo, la edad, diversas enfermedades y el tabaquismo. Cada fila de los datos proporciona información relevante sobre el paciente.

Optimiza proceso médico.

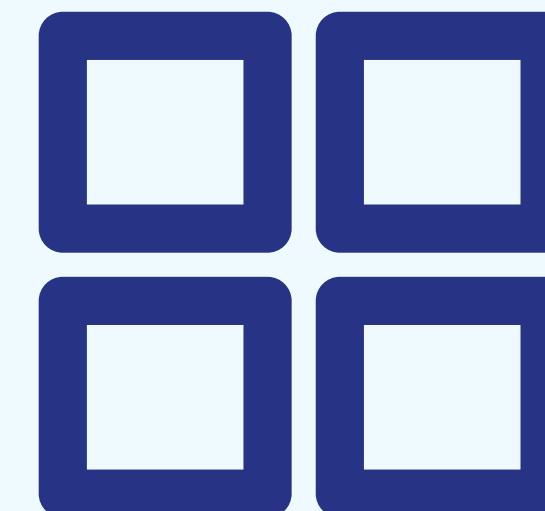


Datos a utilizar

Datos clínicos, centrados en accidentes cerebrovasculares



Datos abiertos,
extraídos de
Kaggle.
Volumen: (5111,
12)



Categoricos:
genre,
ever_married,
work_type,
residence_type,
smoking_status

1
2
3

Numericos:
Id, age,
hypertension,
heart_disease,
avg_glucose_level, bmi, stroke

Análisis de los datos



Análisis de los datos

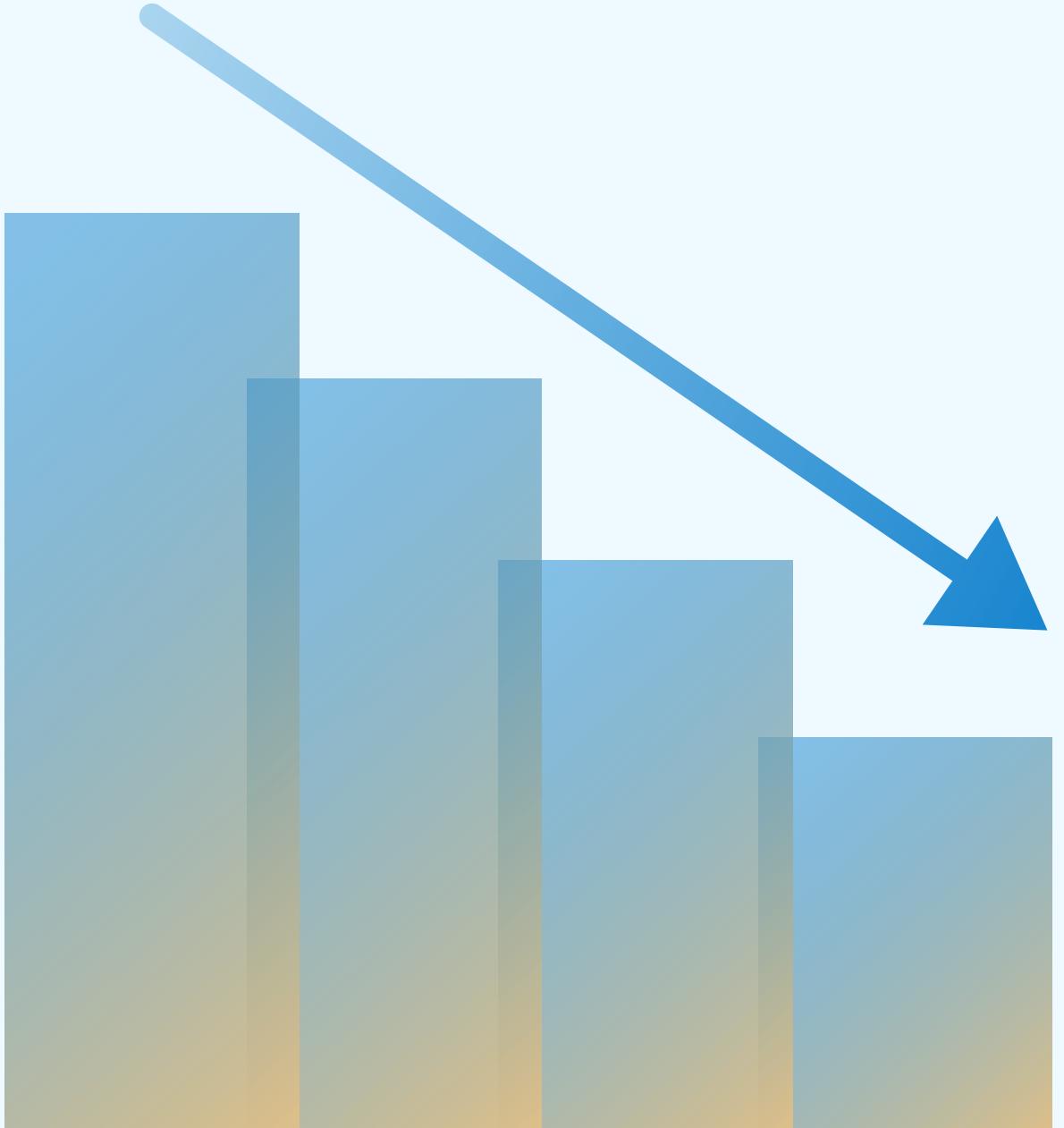
```
dframe.head(10)
```

	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
5	56669	Male	81.0	0	0	Yes	Private	Urban	186.21	29.0	formerly smoked	1
6	53882	Male	74.0	1	1	Yes	Private	Rural	70.09	27.4	never smoked	1
7	10434	Female	69.0	0	0	No	Private	Urban	94.39	22.8	never smoked	1
8	27419	Female	59.0	0	0	Yes	Private	Rural	76.15	NaN	Unknown	1
9	60491	Female	78.0	0	0	Yes	Private	Urban	58.57	24.2	Unknown	1

- 5110 datos y 12 columnas
- Valores nulos en columna BMI, valores numéricos y categóricos.



Parcial escasez de datos



Se investiga posibilidad de data augmentation, basado en parcial escasez de datos. Esto se verá desarrollando los modelos y su evaluación de resultados.

Pre- procesamiento

Se hace el preprocessamiento adecuado,
considerando la limpieza de datos y la
possible normalización.



Preprocesamiento

Remover columnas

```
dframe.drop(['id'], axis=1, inplace=True)
```

Eliminamos columna "id" pues no aporta información relevante para el modelo en predecir si tiene un accidente cerebrovascular o no.

Preprocesamiento

Manejo datos categóricos con One Hot Encoding

```
categorical_columns = ['work_type', 'smoking_status']
encoder = OneHotEncoder(sparse_output=False)

encoded_features = encoder.fit_transform(dframe[categorical_columns])

feature_names = encoder.get_feature_names_out(categorical_columns)

dframe_encoded = pd.DataFrame(encoded_features, columns=feature_names)

dframe = pd.concat([dframe_encoded, dframe.drop(categorical_columns, axis=1)], axis=1)
dframe.head()
```

Las columnas 'work_type', 'smoking_status' son categóricas con más de 2 valores posibles, por lo que usamos este método para convertirlas en variables numéricas.

Preprocesamiento

Manejo datos categóricos con One Hot Encoding

```
print(dframe[dframe['gender'] == 'Other'])
rows_to_delete = dframe[dframe['gender'] == 'other'].index
dframe.drop(rows_to_delete, inplace=True)
print(dframe[dframe['gender'] == 'Other'])
```

Investigando los datos, nos encontramos con que de las 5110 filas que tenemos, el valor de 'gender' en solo una de ellas es 'Other', por lo que eliminamos este valor con el fin de evitar problemas con los modelos a entrenar.

Preprocesamiento

Manejo datos categóricos con One Hot Encoding

```
gender_mapping = {'Male': 0, 'Female': 1}  
marry_mapping = {'Yes': 1, 'No': 0}  
residence_mapping = {'Urban': 1, 'Rural': 0}  
  
dframe['gender'] = dframe['gender'].map(gender_mapping)  
dframe['ever_married'] = dframe['ever_married'].map(marry_mapping)  
dframe['Residence_type'] = dframe['Residence_type'].map(residence_mapping)  
dframe.head()
```

Transformamos las variables categóricas binarias en numéricas

Preprocesamiento

Normalizar variables

```
numerical_features = ['age', 'bmi', 'avg_glucose_level']
scaler = MinMaxScaler()
dframe[numerical_features] = scaler.fit_transform(dframe[numerical_features])
```

Se normalizan las columnas: edad, índice de masa corporal, y promedio del nivel de glucosa.

Preprocesamiento

Manejo de valores nulos

Antes del One Hot Encoding

```
dframe.isnull().sum()  
  
id           0  
gender       0  
age          0  
hypertension 0  
heart_disease 0  
ever_married 0  
work_type    0  
Residence_type 0  
avg_glucose_level 0  
bmi          201  
smoking_status 0  
stroke        0  
dtype: int64
```

Se borran los valores nulos

Post One Hot Encoding

```
dframe.dropna(inplace=True)  
dframe.isnull().sum()  
  
work_type_Govt_job      0  
work_type_Never_worked  0  
work_type_Private        0  
work_type_Self-employed 0  
work_type_children       0  
smoking_status_Unknown  0  
smoking_status_formerly smoked 0  
smoking_status_never smoked 0  
smoking_status_smokes   0  
gender                   0  
age                      0  
hypertension              0  
heart_disease             0  
ever_married               0  
Residence_type             0  
avg_glucose_level          0  
bmi                      0  
stroke                    0  
dtype: int64
```

Inicio modelaje



Inicio modelaje

División de datos en train y test

```
X = dframe.drop(['stroke'], axis=1)
y = dframe['stroke']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=42)
X_new_train, X_val, y_new_train, y_val = train_test_split(X_train, y_train, test_size=0.2)
```

El código separa las características "X" y la variable objetivo "y" del DataFrame original. Luego divide los datos en conjuntos de entrenamiento, prueba y validación, lo que permite el entrenamiento, la evaluación y el ajuste del modelo. El cual, predice la variable "stroke".

Inicio modelaje - MLP

MLP

```
classes = np.unique(y_new_train)
print(classes)
print(x_train.shape[1])
```

```
[0 1]
17
```

```
inputs = Input(shape=(x_train.shape[1],)) # hacemos los inputs y las capas
dense1 = Dense(128, activation="relu")
dense2 = Dense(64, activation="relu")
dense3 = Dense(len(classes), activation="softmax")

# hacemos
x = dense1(inputs)
x = dense2(x)
outputs = dense3(x)
```

Se obtiene las diferentes clases presentes en los datos de entrenamiento. Luego, se define una red neuronal con 1 capa de entrada, 2 capas ocultas (uso ReLu) y 1 capa de salida (uso softmax y número de nodos es el número de clases en los datos de training).

Inicio modelaje - MLP

MLP

```
model = Model(inputs=inputs, outputs=outputs) # establecemos el modelo  
  
model.summary() # hacemos el resumen  
  
Model: "model"  
  
Layer (type) Output Shape Param #  
===== ====== ====== ======
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 17]	0
dense (Dense)	(None, 128)	2304
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 2)	130

```
Total params: 10,690  
Trainable params: 10,690  
Non-trainable params: 0
```

```
y_new_train_encoded = to_categorical(y_new_train)  
y_val_encoded = to_categorical(y_val)  
print(y_val_encoded)  
print("#####")  
print(y_new_train_encoded)  
  
[[1. 0.]  
 [1. 0.]  
 [1. 0.]  
 ...  
 [1. 0.]  
 [1. 0.]  
 [1. 0.]]  
#####
```

- Es el resumen del modelo de red neuronal, viendo las capas del modelo con su forma de salida y su número de parámetros entrenables.
- La codificación de las variables objetivo en una representación de one hot encoding para su uso en train y test del modelo de aprendizaje automático.

Inicio modelaje - MLP

MLP

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3) #establecemos el parámetro
```

- Se hace la compilación con el optimizer
- Se establece el callback, para establecer la patience

Inicio modelaje - RL

Regresión Logística

```
# Regresión Logística sin GMM  
clf = LogisticRegression()  
  
# Train the model on the training data  
clf = LogisticRegression(class_weight='balanced')
```

Definimos un modelo de Regresión Logística usando el parámetro 'class_weight' que va a considerar el desbalance de clases en el dataset



Evaluación del modelo

Entrenamiento de modelos y sus métricas
de evaluación.

Evaluación del modelo

MLP - Entrenar modelo

```
model.fit(x_new_train, y_new_train_encoded, batch_size=32, epochs=20, validation_data=(x_val, y_val_encoded), callbacks=[callback])
```

Hacemos model fit, para entrenar al modelo con los datos de entrenamiento, además usando la data de validación dicha antes

Evaluación del modelo

MLP - Métricas de evaluación

```
Y_preds = model.predict(X_test).argmax(axis=-1) #finalmente, hacemos la predicción

print("Test Accuracy : {}".format(accuracy_score(y_test, Y_preds)))
print("\nClassification Report : ")

print(np.unique(y_test), np.unique(Y_preds), np.unique(y_new_train), np.unique(y_val)) #
#veremos mas adelante
labels = ['0', '1'] #establecemos los labels de las clases
print(classification_report(y_test, Y_preds, zero_division=0, target_names=labels))#hacemos el reporte
```

```
47/47 [=====] - 0s 2ms/step
Test Accuracy : 0.9504412763068567

Classification Report :
[0 1] [0 1] [0 1] [0 1]
precision    recall   f1-score   support
          0       0.95      1.00      0.97     1401
          1       0.00      0.00      0.00      72

accuracy           0.95      0.95     1473
macro avg        0.48      0.50      0.49     1473
weighted avg     0.90      0.95      0.93     1473
```

Se hace la predicción utilizando el modelo entrenado y se muestra la precisión de la predicción y un informe de clasificación que nos da información del rendimiento del modelo en las diferentes clases.

Evaluación del modelo

Regresión Logística - Entrenar modelo

```
# Entrenamiento del modelo de regresión logística  
clf.fit(x_new_train, y_new_train)  
  
# Predicción del modelo de regresión logística  
y_pred = clf.predict(x_test)
```

Se entrena el modelo con los datos y hacemos la predicción usando el modelo de Regresión Logística

Evaluación del modelo

Regresión Logística - Métricas de evaluación

```
# Evaluación del modelo de regresión logística
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Accuracy: 0.745417515274949

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.99	0.74	0.85	1401
1	0.15	0.86	0.25	72

accuracy			0.75	1473
----------	--	--	------	------

macro avg	0.57	0.80	0.55	1473
-----------	------	------	------	------

weighted avg	0.95	0.75	0.82	1473
--------------	------	------	------	------

Evaluación del modelo RL viendo su accuracy y su reporte de clasificación

Comparación de modelos

Investigar comparación de modelos, o mezcla de ambos para lograr máxima accuracy en el futuro.



Comparación de modelos

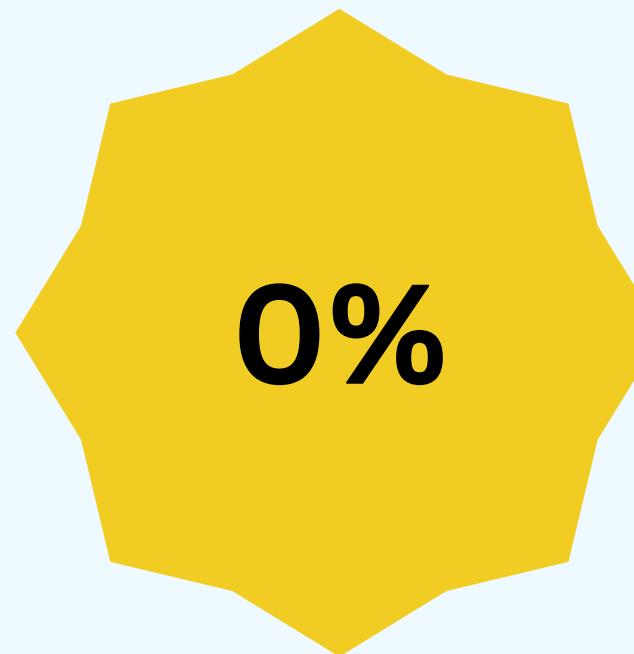
	Complejidad del modelo	Flexibilidad	Interpretabilidad	Tamaño conjunto de datos
MLP	+ complejo, pues es un modelo de aprendizaje profundo con múltiples capas	+ flexible, puede aprender relaciones no lineales	Difícil de interpretar, por su complejidad en su estructura	Requiere conjuntos de datos más grandes
Regresión Logística	+ simple, pues es un modelo lineal con función logística	- flexible, se limita a relaciones lineales	Interpretación directa de los coeficientes del modelo	Puede funcionar con conjuntos pequeños

Análisis de resultados



Análisis de resultados

Interpretación de resultados



MLP

Precisión para los datos 1 en
"stroke"



RL

Problemas con la clasificación, y los modelos no están logrando aprender los patrones.

Análisis de resultados

Por mejorar



- Investigar uso de data augmentation u otro método en el dataset.
- Priorizar algunas columnas por sobre otras (cambio de pesos).
- Investigar eliminación de columnas.
- Probar con otros modelos y/o cambiar hiperparámetros.

Declaración de avance: Próxima entrega final

01

Tomar en cuenta cualquier tipo de retroalimentación entregada en esta presentación

02

Realizar las mejoras sugeridas del análisis.



GRACIAS

Lucas Aguilera, Claudio Bórquez, Josefá Fernández