# Ejercicio de Redes Neuronales

## Descripción del Problema

Se tiene un DataSet de clientes inscritos en un banco y el banco desea saber porque los clientes están abandonando el banco y porque la gente se está yendo. Todos los datos están guardados en el DataSet (.csv). Este problema ayuda si una persona es o no es (positivo o negativo, 1 o 0). Problema de Clasificación

## Pre procesamiento de Datos

## Importacion del DataSet

Exportamos las librerias importantes para iniciar el programa.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Conexion a mi carpeta de Google Drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Conexion al archivo que deseo ocupar (Datos del Banco).

```
dataset = pd.read_csv('/content/drive/MyDrive/Curso de Deep Learning de la A a la Z/datase
# print(dataset)
```

Extraccion de las filas y columnas.

```
# Matriz de Caracteristicas.
X = dataset.iloc[:, 3:13].values
# Vector de la variable dependiente.
y = dataset.iloc[:, 13].values
```

## Codificar datos categóricos

Importamos la libreria para el maching learning.

```
from sklearn.preprocessing import LabelEncoder
```

Codificamos las variables categoricas que no son numeros y se hacen variables dummy (0 o 1).

```
# Codificamos en variable Damit el apartado de "Geografia", lo pasa a numeros (0, 1, 2, et
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

# Codificamos en variable Damit el apartado de "Genero" (0 o 1 dependiendo el genero).
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
```

Importamos las librerias de Maching Learning para transformar todas las variables a unas mismas y no tener categoricas y numericas.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

Creamos en la columna de paises que se separen en mas filas dependiendo el pais.

```
# Una lista de transformaciones.
transformer = ColumnTransformer(
    transformers=[
        ("Churn_Modelling",        # Un nombre de la transformación
         OneHotEncoder(categories='auto'), # La clase a la que transformar
         [1]            # Especifica que se transformará la segunda columna (índice 1) del
         )
    ], remainder='passthrough' # Las columnas que no se especificaron en la lista de trans
)

X = transformer.fit_transform(X)
X = X[:, 1:]
```

## Implementamos la tecnica de k-Foald Cross Validation

Importamos libreria para dividir los conjuntos de entrenamiento y test.

```
from sklearn.model_selection import train_test_split
```

Definimos las variables de entrenamiento y las variables que se ocuparan para hacer el test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

## Escalado de Variables.

Importamos la libreria para el escalado.

```
from sklearn.preprocessing import StandardScaler
```

Se realiza un escalado de variables cuando las escalas de los numeros son diferentes (En este caso tenemos muchas variables en rango de 0 o 1, numericas, precios, etc.). Aqui puede que tengamos variables negativas y que ya no entendamos muy bien (Variables Normalizadas).

```
# Declaramos el metodo para el escalado
sc_X = StandardScaler()

# Ponemos el escalado a las variables de entrenamiento X. Calculamos el cambio de escala y
X_train = sc_X.fit_transform(X_train)
# Ponemos el escalado a las variables de test X. Aqui solo se hace el transform ya que en
X_test = sc_X.transform(X_test)
```

# Parte 2 - Construir la RNA

## Importamos las librerias para trabajar nuestra Red Neuronal

Importacion de Keras y librerías adicionales.

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

Creamos y inicializamos el modelo de clasificacion (Red Neuronal Artificial).

```
# Inicializacion del Modelo de la RNA (con Sequential)
classifier = Sequential()
```

Creamos la Capas de manera sucesiva con una secuencia de capas. Y añadimos las clases que tendra esta.

- units: especifica el número de neuronas en la capa (en este caso, 6).
- kernel_initializer: especifica cómo se inicializan los pesos de las conexiones entre las neuronas de la capa (en este caso, de forma uniforme).
- activation: especifica la función de activación a utilizar en las neuronas de la capa (en este caso, ReLU).
- input_dim: especifica el número de variables de entrada en la capa (en este caso, 11).

```
# Añadir las capas de entrada y primera capa oculta
classifier.add(Dense(units = 6, kernel_initializer = "uniform",
                     activation = "relu", input_dim = 11))

# Añadir la segunda capa oculta
classifier.add(Dense(units = 6, kernel_initializer = "uniform",
                     activation = "relu"))

# Añadir la capa de salida
classifier.add(Dense(units = 1, kernel_initializer = "uniform",
                     activation = "sigmoid"))
```

Compilamos y configuramos el Modelo de la Red Neuronal

- optimizer: especifica el algoritmo de optimización a utilizar para minimizar la función de pérdida del modelo durante el entrenamiento. En este caso, se utiliza el algoritmo de optimización Adam ("adam").
- loss: especifica la función de pérdida que se utilizará para evaluar la precisión del modelo durante el entrenamiento. En este caso, se utiliza la función de pérdida binaria cruzada ("binary_crossentropy") porque se está trabajando en un problema de clasificación binaria.
- metrics: especifica la métrica a utilizar para evaluar el desempeño del modelo durante el entrenamiento. En este caso, se utiliza la métrica de precisión ("accuracy"), que mide la fracción de predicciones correctas del modelo.

```
 # Compilar la RNA
classifier.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"
```

Entrenamos la Red Neuronal.

```
# Ajustamos la RNA al Conjunto de Entrenamiento
classifier.fit(X_train, y_train,  batch_size = 10, epochs = 100)
```

```
Epoch 1/100
800/800 [==============================] - 2s 1ms/step - loss: 0.4857 - accuracy:
Epoch 2/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4181 - accuracy:
Epoch 3/100
```

```
800/800 [==============================] - 1s 1ms/step - loss: 0.4016 - accuracy:
Epoch 4/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3833 - accuracy:
Epoch 5/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3719 - accuracy:
Epoch 6/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3649 - accuracy:
Epoch 7/100
800/800 [==============================] - 1s 2ms/step - loss: 0.3604 - accuracy:
Epoch 8/100
800/800 [==============================] - 1s 2ms/step - loss: 0.3579 - accuracy:
Epoch 9/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3565 - accuracy:
Epoch 10/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3545 - accuracy:
Epoch 11/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3537 - accuracy:
Epoch 12/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3516 - accuracy:
Epoch 13/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3516 - accuracy:
Epoch 14/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3505 - accuracy:
Epoch 15/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3493 - accuracy:
Epoch 16/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3489 - accuracy:
Epoch 17/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3483 - accuracy:
Epoch 18/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3474 - accuracy:
Epoch 19/100
800/800 [==============================] - 1s 2ms/step - loss: 0.3480 - accuracy:
Epoch 20/100
800/800 [==============================] - 2s 3ms/step - loss: 0.3462 - accuracy:
Epoch 21/100
800/800 [==============================] - 2s 2ms/step - loss: 0.3466 - accuracy:
Epoch 22/100
800/800 [==============================] - 3s 3ms/step - loss: 0.3461 - accuracy:
Epoch 23/100
800/800 [==============================] - 2s 2ms/step - loss: 0.3463 - accuracy:
Epoch 24/100
800/800 [==============================] - 2s 2ms/step - loss: 0.3455 - accuracy:
Epoch 25/100
800/800 [==============================] - 1s 1ms/step - loss: 0.3450 - accuracy:
Epoch 26/100
800/800 [==============================] - 1s 2ms/step - loss: 0.3444 - accuracy:
Epoch 27/100
800/800 [==============================] - 2s 3ms/step - loss: 0.3446 - accuracy:
Epoch 28/100
```

## Parte 3 - Evaluar el modelo y calcular predicciones finales

[ ] ↳ *8 celdas ocultas*

## Reto

[ ] ↳ *4 celdas ocultas*

# Parte 4-Evaluar, mejorar y ajustar la RNA

## Evaluar la RNA

Importacion de las librerias para k-Foald Croos Validation

```
# Funciones de alto nivel.
from keras.wrappers.scikit_learn import KerasClassifier
# Funcion del metodo.
from sklearn.model_selection import cross_val_score
```

Creamos una funcion para la implementacion de el K-Foald Cross Validation.

```
# Funcion para crear el clasificador.
def build_classifier():
  # Inicializacion.
  classifier = Sequential()

  # Definicion de las capas de la Red Neuronal
  classifier.add(Dense(units = 6, kernel_initializer = "uniform",
                       activation = "relu", input_dim = 11))
  classifier.add(Dense(units = 6, kernel_initializer = "uniform",
                       activation = "relu"))
  classifier.add(Dense(units = 1, kernel_initializer = "uniform",
                       activation = "sigmoid"))

  # Fase de Compilacion del modelo.
  classifier.compile(optimizer = "adam", loss = "binary_crossentropy",
                     metrics = ["accuracy"])
  # Retorno del clasificador.
  return classifier
```

Ajustamos la RNA para entrenar la RNA o los pliegues (Vector de precision).

- el número de pliegues de validación cruzada (cv), el número de trabajos en paralelo (n_jobs) y el nivel de verbosidad (verbose).

```
classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, nb_epoch = 100)
# Realizar la validación cruzada del modelo.
```

```
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10, n_
```

```
<ipython-input-16-144c095514f9>:1: DeprecationWarning: KerasClassifier is deprecated,
  classifier = KerasClassifier(build_fn = build_classifier, batch_size = 10, nb_epoch
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   29.8s finished
```

Opservamos la precision optenida.

```
print(accuracies)
```

```
[0.78625    0.79000002 0.80000001 0.78250003 0.81625003 0.81
 0.78750002 0.79374999 0.79874998 0.79500002]
```

Sacamos la Media y Varianza de nuestros

```
# Calculamos la Media.
mean = accuracies.mean()
# Calculamos la desviación estándar
variance = accuracies.std()
```

Obsevamos la Media y la Varianza (Desviacion Estandar).

```
print(mean)
print(variance)
```

```
0.7960000097751617
0.010105690527913859
```

✓ 0 s se ejecutó 12:26