Tarea 7: Aplicación perceptrón multicapa

Alumno: Ricardo De León

En la tabla adjunta se muestran los datos de 1000 clientes que solicitaron créditos a un banco dado. La última columna muestra la información de los clientes que cayeron en mora en algún momento del período del crédito. El monto máximo de crédito que puede asignarse son $300,000 y la antigüedad laboral máxima que se toma en cuenta para asignar el crédito es de 15 años (es decir, antigüedades mayores ya no generan más posibilidad de ser aprobado).

Se busca una relación entre la información presentada (que se obtiene al contratar el crédito) y la posibilidad de que el cliente caiga en mora en algún momento del plazo.

Entrene un perceptrón multicapa para encontrar una relación tomando como entradas el monto solicitado (normalizado), la carga que implica al salario el pago de la mensualidad y la antigüedad laboral al contratar (normalizada).

Utilice 70-30 de relación entrenamiento-prueba y calcule el ***accuracy***.

El número de neuronas ocultas es a su criterio.

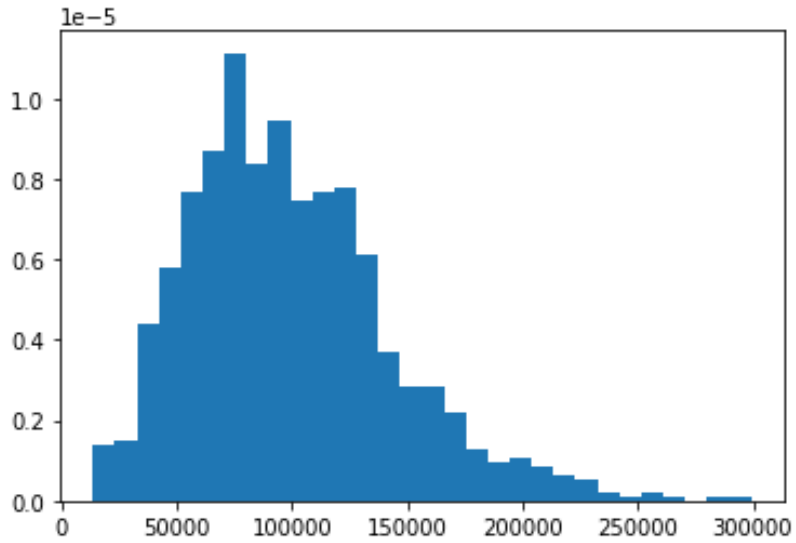
Desarrollo:

Identificamos las columnas requeridas en el problema y analizamos su distribución

Para Monto:

 # Observamos la gráfica de densidad de la columna monto

plt.hist(df['Monto'],density=True, bins=30)



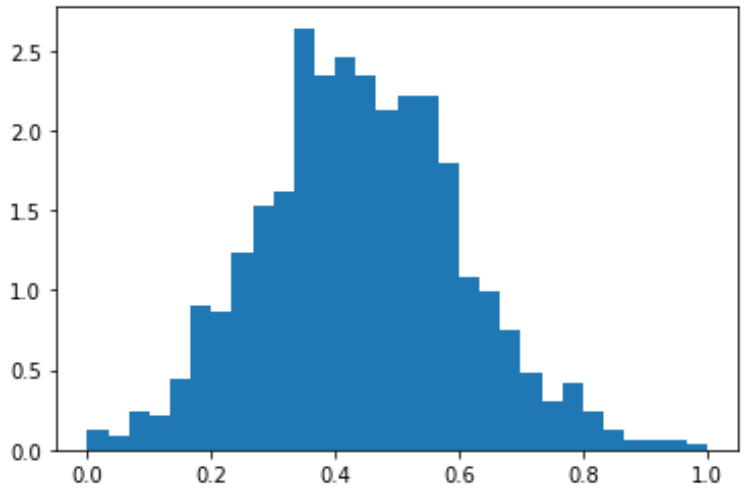
Tarea 7: Aplicación perceptrón multicapa

 # Tratamos de hacer la curva de más parecida a una desviacion normal

```
monto = np.sqrt(np.asarray(df['Monto']))
```

 # Normalizamos lo datos con min y max

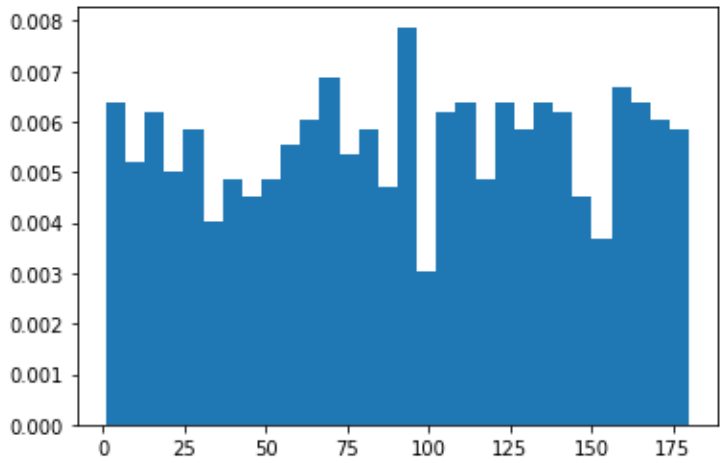
```
def normalizar(data):
    return (data - data.min())/ (data.max() - data.min())

monto_norm = normalizar(monto)
plt.hist(monto_norm,density=True, bins=30)
```



Para Edad Laboral

 # Observamos la gráfica de densidad de la columna Edad Laboral

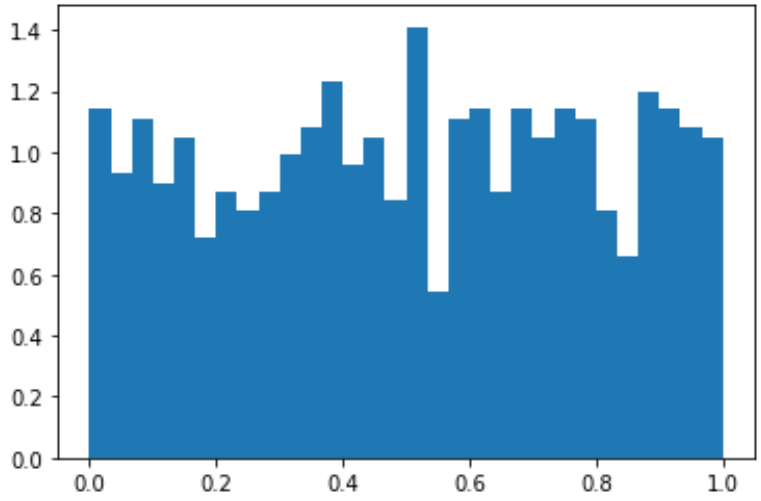
```
edad_laboral = np.asarray(df.iloc[:,6])
plt.hist(edad_laboral,density=True, bins=30)
```



Tarea 7: Aplicación perceptrón multicapa

 # Normalizamos lo datos con min y max

edad_laboral_norm = normalizar(edad_laboral)

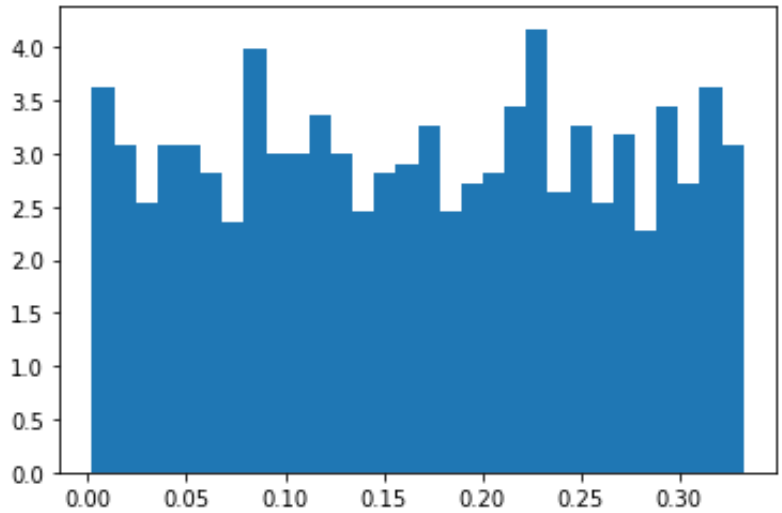


Observamos que la gráfica es casi idéntica, pero en una escala diferente ya que la distribución es más uniforme.

Para Salario

 # Observamos la gráfica de densidad de la columna Edad Laboral

carga_salario = np.asarray(df['Mensualidad']/df.iloc[:,5])
plt.hist(carga_salario,density=True, bins=30)

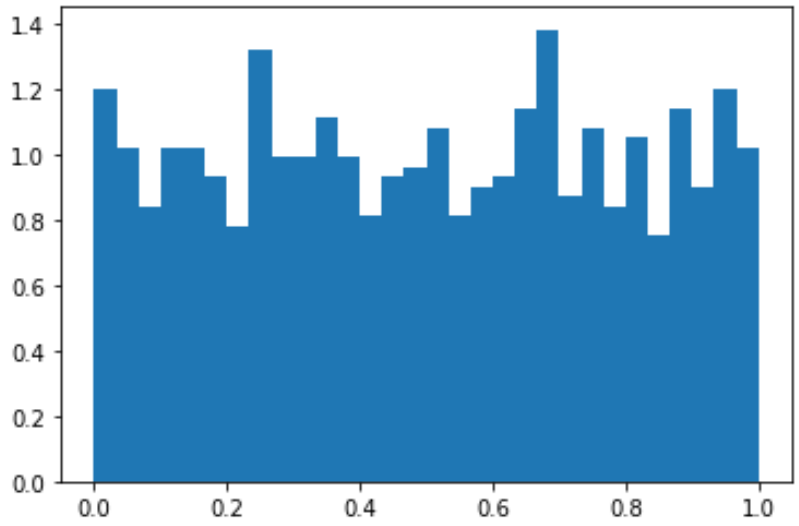


Tarea 7: Aplicación perceptrón multicapa

 # Normalizamos lo datos con min y max

carga_salario_norm = normalizar(carga_salario)

plt.hist(carga_salario_norm,density=True, bins=30)



Observamos que la gráfica es casi idéntica, pero en una escala diferente ya que la distribución es más uniforme.

- Una vez homogeneizados nuestros datos comenzamos a unir nuestro dataset final

```
mora = np.asarray(df['Mora'].map({'SI': 1, 'NO': 0}))

df_norm = pd.DataFrame({'Monto' : monto_norm,
                        'Salario' : carga_salario_norm,
                        'eLaboral' : edad_laboral_norm,
                        'Mora': mora})
```

- Hacemos la estratificación de la muestra de un 70-30 definiendo dataset de sus proporciones train (70) y test (30).

 # Estratificando muestreo

```
df_train = df_norm.groupby("Mora").sample(frac=0.7, random_state=1) # usar andom_state=1 para tomar las mismas muestras siempre
df_test = df_norm[~df_norm.index.isin(df_train.index)]
```

Tarea 7: Aplicación perceptrón multicapa

- Por último ejecutamos nuestro train y test y observamos nuestro acurracy

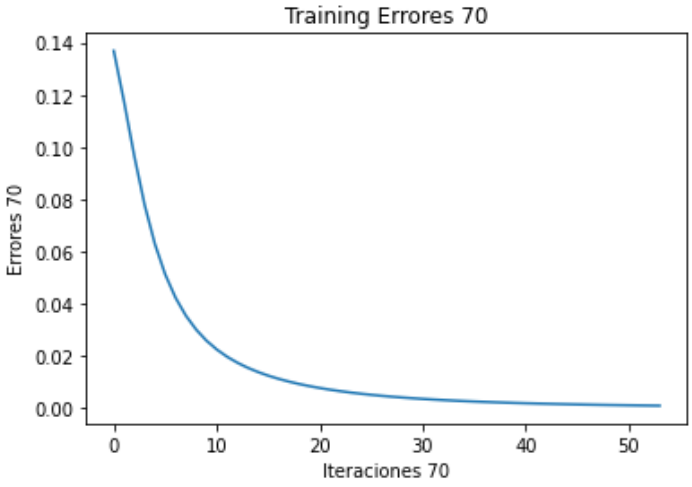
```
X_70,D_70 = get_X_D(df_train)
errors_70, wh_70, wo_70 = perceptron_m(X_70, D_70)

print('Comprobación de datos entrenados para el 70 de muestreo')
result_70 = []
for i in range(X_70.shape[0]):
   _, y = forward(X_70, wo_70, wh_70, i)
   result_70.append(round(y[0]))

accuracy_70 = np.mean(D_70 == result_70)

print(f'accuracy 70% sample {round(accuracy_70 * 100, 2)}%')

plt.plot(errors_70)
plt.xlabel("Iteraciones 70")
plt.ylabel("Errores 70")
plt.title("Training Errores 70")
plt.show()

X_30,D_30 = get_X_D(df_test)
errors_30, wh_30, wo_30 = perceptron_m(X_70, D_70)

print('Comprobación de datos entrenados para el 30 de muestreo')
result_30 = []
for i in range(X_30.shape[0]):
   _, y = forward(X_30, wo_30, wh_30, i)
   result_30.append(round(y[0]))

accuracy_30 = np.mean(D_30 == result_30)

print(f'accuracy 30% sample {round(accuracy_30 * 100, 2)}%')

plt.plot(errors_30)
plt.xlabel("Iteraciones 30")
plt.ylabel("Errores 30")
plt.title("Training Errores 30")
plt.show()
```

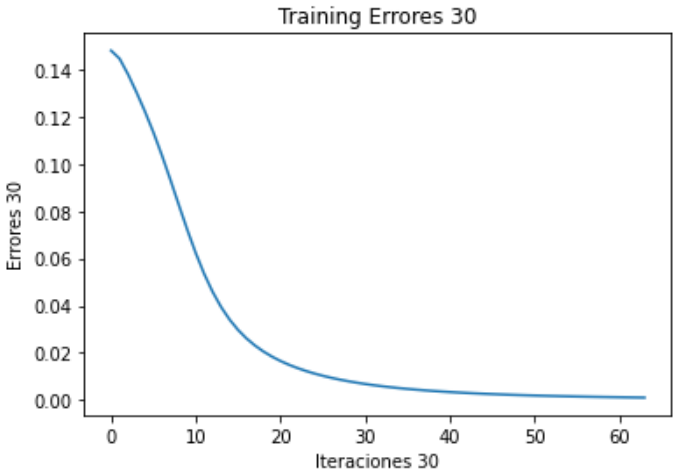
Tarea 7: Aplicación perceptrón multicapa

- Resultados

```
Limite de error alcanzado en iter 41300
Error : 9.6693e-04
Comprobación de datos entrenados para el 70 de muestreo
accuracy 70% sample 96.29%

Limite de error alcanzado en iter 39200
Error : 9.6999e-04
Comprobación de datos entrenados para el 30 de muestreo
accuracy 30% sample 94.0%
```

Gráficas de errores





Tarea 7: Aplicación perceptrón multicapa

- Código

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 10 16:52:58 2023

@author: Ricado De Leon
"""

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


df = pd.read_excel('PercMultAplicado.xlsx')

def normalizar(data):
    return (data - data.min())/ (data.max() - data.min())

# Observamos la grafica de densidad de la columna monto
# plt.hist(df['Monto'],density=True, bins=30)

# Tratamos de hacer la curva de más parecida a una desviacion normal
monto = np.sqrt(np.asarray(df['Monto']))
# plt.hist(monto,density=True, bins=30)
# Normalizamos lo datos con min y max
monto_norm = normalizar(monto)
# plt.hist(monto_norm,density=True, bins=30)

carga_salario = np.asarray(df['Mensualidad']/df.iloc[:,5])
# plt.hist(carga_salario,density=True, bins=30)
# Normalizamomos lo datos con min y max
carga_salario_norm = normalizar(carga_salario)
# plt.hist(carga_salario_norm,density=True, bins=30)

edad_laboral = np.asarray(df.iloc[:,6])
# plt.hist(edad_laboral,density=True, bins=30)
# Normalizamos lo datos con min y max
edad_laboral_norm = normalizar(edad_laboral)
# plt.hist(edad_laboral_norm,density=True, bins=30)

mora = np.asarray(df['Mora'].map({'SI': 1, 'NO': 0}))

df_norm = pd.DataFrame({'Monto' : monto_norm,
                        'Salario' : carga_salario_norm,
                        'eLaboral' : edad_laboral_norm,
                        'Mora': mora})

# Estratificando sampleo
df_train = df_norm.groupby("Mora").sample(frac=0.7, random_state=1) # usar andom_state=1 para tomar las
mismas muestras siempre
df_test = df_norm[~df_norm.index.isin(df_train.index)]
```

Tarea 7: Aplicación perceptrón multicapa

```python
def get_X_D(df):
    X = np.array(df.iloc[:,:-1])
    D = np.array(df.iloc[:,-1])
    return X,D

def forward(X, wo, wh, iteracion):
    neth = wh @ X[iteracion].T
    yh = 1 / (1 + np.exp(-neth))
    neto = wo @ yh
    y = 1 / (1 + np.exp(-neto))
    return yh, y

def perceptron_m(X,D):
    # Definir los parámetros del modelo en donde L es el numero de neuronas
    N = X.shape[1]
    M = 1
    Q = X.shape[0]
    L = 5
    alpha = 0.1
    iteraciones = 100000
    E = 1e-3

    # Definir los pesos para la capa oculta wo D capa de salida wo
    wo = np.random.uniform(low=-1, high=1, size=(M, L)) - 0.5
    wh = np.random.uniform(low=-1, high=1, size=(L, N)) - 0.5

    def backward(D, yh, y, iteracion):
        del_wh = np.zeros_like(wh)
        del_wo = np.zeros_like(wo)
        do = (D[iteracion].T - y) * (y * (1 - y))
        dh = yh * ( 1 - yh) * (wo.T @ do)
        del_wo += np.reshape(alpha*do,(M,1)) @ np.reshape(yh.T,(1,L))
        del_wh += np.reshape(alpha*dh,(L,1)) @ np.reshape(X[iteracion],(1,N))
        return dh, do, del_wo, del_wh

    errors = []
    iter = 0
    for i in range(iteraciones):
        for j in range(Q):
            yh, y = forward(X, wo, wh, j)
            _, do, del_wo, del_wh = backward(D, yh, y, j)
            error=np.linalg.norm(do)
            wo+=del_wo
            wh+=del_wh
            iter += 1
        if error < E:
            print(f'\nLimite de error alcanzado en iter {iter}')
            print(f'Error : {"{:.4e}".format(error)}')
            break
        errors.append(error)
    return errors, wh, wo

X_70,D_70 = get_X_D(df_train)
errors_70, wh_70, wo_70 = perceptron_m(X_70, D_70)

print('Comprobación de datos entrenados para el 70 de muestreo')
result_70 = []
for i in range(X_70.shape[0]):
    _, y = forward(X_70, wo_70, wh_70, i)
    result_70.append(round(y[0]))

accuracy_70 = np.mean(D_70 == result_70)

print(f'accuracy 70% sample {round(accuracy_70 * 100, 2)}%')

plt.plot(errors_70)
plt.xlabel("Iteraciones 70")
plt.ylabel("Errores 70")
plt.title("Training Errores 70")
plt.show()

X_30,D_30 = get_X_D(df_test)
errors_30, wh_30, wo_30 = perceptron_m(X_70, D_70)

print('Comprobación de datos entrenados para el 30 de muestreo')
result_30 = []
for i in range(X_30.shape[0]):
    _, y = forward(X_30, wo_30, wh_30, i)
    result_30.append(round(y[0]))

accuracy_30 = np.mean(D_30 == result_30)

print(f'accuracy 30% sample {round(accuracy_30 * 100, 2)}%')

plt.plot(errors_30)
plt.xlabel("Iteraciones 30")
plt.ylabel("Errores 30")
plt.title("Training Errores 30")
plt.show()
```