Puntaje Bancario: Aprobación de Crédito Mediante Redes Neuronales

En esta ocasión se busca desarrollar un protocolo de pruebas que permita encontrar la mejor arquitectura de red neuronal completamente conectada. En esta ocasión debe utilizar la librería SciKit-Learn (sklearn.neural_network) para diseñar cada red. Además, veremos algunos conceptos de *feature engineering* para analizar los datos a nuestra disposición.

Debe completar las celdas vacías y seguir las instrucciones anotadas en el cuaderno.

La fecha límite de entrega es el día **18 de octubre** y se realizará a través de Bloque Neón.

In [ ]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
```

Se tienen dos archivos:

`application_record.csv` : posee información general (puede observar los nombres de las columnas a continuación) sobre cada usuario, definido a partir de una ID única.

- ID: número de cliente
- CODE_GENDER: género
- FLAG_OWN_CAR: posee un automóvil
- FLAG_OWN_REALTY: posee un inmueble
- CNT_CHILDREN: cantidad de hijos
- AMT_INCOME_TOTAL: ingresos anuales
- NAME_INCOME_TYPE: categoría de ingresos
- NAME_EDUCATION_TYPE: nivel educativo
- NAME_FAMILY_STATUS estado civil
- NAME_HOUSING_TYPE: forma de vivienda (e.g. renta, apartamento propio, ...)
- DAYS_BIRTH: fecha de nacimiento, en días hacia atrás desde la actualidad, -1 significa ayer
- DAYS_EMPLOYED: tiempo de empleo, en días hacia atrás desde la actualidad, -1 significa ayer. Si es positivo, el usuario se encuentra desempleado.
- FLAG_MOBIL: teléfono móvil
- FLAG_WORK_PHONE: teléfono de trabajo
- FLAG_PHONE: teléfono
- FLAG_EMAIL: email
- OCCUPATION_TYPE: ocupación
- CNT_FAM_MEMBERS: tamaño de familia

`credit_record.csv` :

- ID: número de cliente
- MONTHS_BALANCE: mes de registro
- ESTADO:
  - 0: 1-29 días atrasados
  - 1: 30-59 días atrasados
  - 2: 60-89 días atrasados
  - 3: 90-119 días atrasados
  - 4: 120-149 días atrasados
  - 5: Atrasados o incobrables, cancelaciones durante más de 150 días
  - C: cancelado ese mes X: sin préstamo durante el mes

In [ ]:
```
data = pd.read_csv("creditCardScore/application_record.csv", encoding
= 'utf-8')
record = pd.read_csv("creditCardScore/credit_record.csv", encoding =
'utf-8')
```

In [ ]:
```
plt.rcParams['figure.facecolor'] = 'white'
```

In [ ]:
```
data.tail()
```

In [ ]:
```
record.head()
```

Preparación de los Datos

Etiquetas

Inicialmente, se concatenan ambas tablas mediante el tiempo de registro máximo (`MONTHS_BALANCE`) y la ID del cliente.

In [ ]:
```
# find all users' account open month.
begin_month=pd.DataFrame(record.groupby(["ID"])["MONTHS_BALANCE"].agg
(min))
begin_month=begin_month.rename(columns={'MONTHS_BALANCE':'begin_mont
h'})
new_data=pd.merge(data,begin_month,how="left",on="ID") #merge to reco
rd data
```

Los usuarios con mora durante más de 60 días se etiquerarán como `1` , de lo contrario, serán `0` .

In [ ]:
```
record['dep_value'] = None
record['dep_value'][record['STATUS'] =='2']='Yes'
record['dep_value'][record['STATUS'] =='3']='Yes'
record['dep_value'][record['STATUS'] =='4']='Yes'
record['dep_value'][record['STATUS'] =='5']='Yes'
```

In [ ]:
```
cpunt=record.groupby('ID').count()
cpunt['dep_value'][cpunt['dep_value'] > 0]='Yes'
cpunt['dep_value'][cpunt['dep_value'] == 0]='No'
cpunt = cpunt[['dep_value']]
new_data=pd.merge(new_data,cpunt,how='inner',on='ID')
new_data['target']=new_data['dep_value']
new_data.loc[new_data['target']=='Yes','target']=1
new_data.loc[new_data['target']=='No','target']=0
```

In [ ]:
```
print(cpunt['dep_value'].value_counts())
cpunt['dep_value'].value_counts(normalize=True)
```

Proporción de clases.

Descriptores

- Renombramiento de las Columnas

In [ ]:
```
new_data.rename(columns={'CODE_GENDER':'Gender','FLAG_OWN_CAR':'Car',
'FLAG_OWN_REALTY':'Reality',
                         'CNT_CHILDREN':'ChldNo','AMT_INCOME_TOTAL':
'inc',
                         'NAME_EDUCATION_TYPE':'edutp','NAME_FAMILY_S
TATUS':'famtp',
                        'NAME_HOUSING_TYPE':'houtp','FLAG_EMAIL':'ema
il',
                         'NAME_INCOME_TYPE':'inctp','FLAG_WORK_PHONE'
:'wkphone',
                         'FLAG_PHONE':'phone','CNT_FAM_MEMBERS':'fams
ize',
                        'OCCUPATION_TYPE':'occyp'
                        },inplace=True)
```

In [ ]:
```
new_data.dropna()
new_data = new_data.mask(new_data == 'NULL').dropna() # Retiramos los
valores NaN
```

In [ ]:
```
ivtable=pd.DataFrame(new_data.columns,columns=['variable'])
ivtable['IV']=None
namelist = ['FLAG_MOBIL','begin_month','dep_value','target','ID']

for i in namelist:
    ivtable.drop(ivtable[ivtable['variable'] == i].index, inplace=Tru
e)
```

Funciones Auxiliares

A continuación se crean algunas funciones que serán utilizadas más adelante.

Función `calc_iv` para obtener las variables IV (information value) y WoE (weight of evidence). Estas variables, de forma general, nos permiten conocer la importancia de cada feature disponible.

Puede encontrar más información en:

- https://www.kaggle.com/puremath86/iv-woe-starter-for-python (https://www.kaggle.com/puremath86/iv-woe-starter-for-python)
- https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html (https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html)

In [ ]:
```
# Cálculo de IV
def calc_iv(df, feature, target, pr=False):
    lst = []
    df[feature] = df[feature].fillna("NULL")

    for i in range(df[feature].nunique()):
        val = list(df[feature].unique())[i]
        lst.append([feature,
# Variable
                    val,
# Value
                    df[df[feature] == val].count()[feature],
# All
                    df[(df[feature] == val) & (df[target] == 0)].coun
t()[feature],  # Good (think: Fraud == 0)
                    df[(df[feature] == val) & (df[target] == 1)].coun
t()[feature]]) # Bad (think: Fraud == 1)

    data = pd.DataFrame(lst, columns=['Variable', 'Value', 'All', 'Go
od', 'Bad'])
    data['Share'] = data['All'] / data['All'].sum()
    data['Bad Rate'] = data['Bad'] / data['All']
    data['Distribution Good'] = (data['All'] - data['Bad']) / (data[
'All'].sum() - data['Bad'].sum())
    data['Distribution Bad'] = data['Bad'] / data['Bad'].sum()
    data['WoE'] = np.log(data['Distribution Good'] / data['Distributi
on Bad'])

    data = data.replace({'WoE': {np.inf: 0, -np.inf: 0}})

    data['IV'] = data['WoE'] * (data['Distribution Good'] - data['Dis
tribution Bad'])

    data = data.sort_values(by=['Variable', 'Value'], ascending=[True
, True])
    data.index = range(len(data.index))

    if pr:
        print(data)
        print('IV = ', data['IV'].sum())

    iv = data['IV'].sum()
    print('El IV de esta variable es:',iv)
    print(df[feature].value_counts())
    return iv, data
```

In [ ]:
```
# Codificación One-Hot
def convert_dummy(df, feature,rank=0):
    pos = pd.get_dummies(df[feature], prefix=feature)
    mode = df[feature].value_counts().index[rank]
    biggest = feature + '_' + str(mode)
    pos.drop([biggest],axis=1,inplace=True)
    df.drop([feature],axis=1,inplace=True)
    df=df.join(pos)
    return df
```

In [ ]:
```
def get_category(df, col, binsnum, labels, qcut = False):
    if qcut:
        localdf = pd.qcut(df[col], q = binsnum, labels = labels) # qu
antile cut
    else:
        localdf = pd.cut(df[col], bins = binsnum, labels = labels) #
 equal-length cut

    localdf = pd.DataFrame(localdf)
    name = 'gp' + '_' + col
    localdf[name] = localdf[col]
    df = df.join(localdf[name])
    df[name] = df[name].astype(object)
    return df
```

In [ ]:
```
# Matriz de Confusión
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[
1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Etiqueta')
    plt.xlabel('Predicción')
```

Descriptores Binarios

Se utilizará la función desarrollada anteriormente para realizar un análisis de cada uno de los descriptores binarios y **su influencia dentro de la predicción de cada clase**.

Género

In [ ]:
```
new_data['Gender'] = new_data['Gender'].replace(['F','M'],[0,1])
print(new_data['Gender'].value_counts())
iv, data = calc_iv(new_data,'Gender','target')
ivtable.loc[ivtable['variable']=='Gender','IV']=iv
data.head()
```

Posesión de un Automóvil

In [ ]:
```
new_data['Car'] = new_data['Car'].replace(['N','Y'],[0,1])
print(new_data['Car'].value_counts())
iv, data=calc_iv(new_data,'Car','target')
ivtable.loc[ivtable['variable']=='Car','IV']=iv
data.head()
```

Posesión de un Inmueble

In [ ]:
```
new_data['Reality'] = new_data['Reality'].replace(['N','Y'],[0,1])
print(new_data['Reality'].value_counts())
iv, data=calc_iv(new_data,'Reality','target')
ivtable.loc[ivtable['variable']=='Reality','IV']=iv
data.head()
```

Posesión de un Teléfono

In [ ]:
```
new_data['phone']=new_data['phone'].astype(str)
print(new_data['phone'].value_counts(normalize=True,sort=False))
new_data.drop(new_data[new_data['phone'] == 'nan' ].index, inplace=Tr
ue)
iv, data=calc_iv(new_data,'phone','target')
ivtable.loc[ivtable['variable']=='phone','IV']=iv
data.head()
```

Posesión de Correo Electrónico (Email)

In [ ]:
```
print(new_data['email'].value_counts(normalize=True,sort=False))
new_data['email']=new_data['email'].astype(str)
iv, data=calc_iv(new_data,'email','target')
ivtable.loc[ivtable['variable']=='email','IV']=iv
data.head()
```

Posesión de Teléfono para Trabajo

In [ ]:
```
new_data['wkphone']=new_data['wkphone'].astype(str)
iv, data = calc_iv(new_data,'wkphone','target')
new_data.drop(new_data[new_data['wkphone'] == 'nan' ].index, inplace=
True)
ivtable.loc[ivtable['variable']=='wkphone','IV']=iv
data.head()
```

Descriptores Continuos

Cantidad de Hijos

In [ ]:
```
# Separamos aquellos que: no tienen hijos, tienen 1 hijo, tienen 2 o
 más hijos.

new_data.loc[new_data['ChldNo'] >= 2,'ChldNo']='2More'
print(new_data['ChldNo'].value_counts(sort=False))
```

In [ ]:
```
iv, data=calc_iv(new_data,'ChldNo','target')
ivtable.loc[ivtable['variable']=='ChldNo','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'ChldNo') # Adicionamos una Codific
ación One-Hot
```

In [ ]:
```
new_data.head()
```

Ingresos Anuales

Gráfica de Histograma para observar la distribución.

In [ ]:
```
new_data['inc'] = new_data['inc'].astype(object)
new_data['inc'] = new_data['inc']/10000
print(new_data['inc'].value_counts(bins=10,sort=False))
```

In [ ]:
```
new_data['inc'].plot(kind='hist',bins=50,density=True)
```

In [ ]:
```
new_data = get_category(new_data,'inc', 3, ["low","medium", "high"],
qcut = True)
iv, data = calc_iv(new_data,'gp_inc','target')
ivtable.loc[ivtable['variable']=='inc','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'gp_inc')
```

Edad

In [ ]:
```
new_data['Age']=-(new_data['DAYS_BIRTH'])//365
print(new_data['Age'].value_counts(bins=10,normalize=True,sort=False
))
```

In [ ]:
```
new_data['Age'].plot(kind='hist',bins=20,density=True)
```

In [ ]:
```
new_data = get_category(new_data,'Age',5, ["lowest","low","medium","h
igh","highest"])
iv, data = calc_iv(new_data,'gp_Age','target')
ivtable.loc[ivtable['variable']=='DAYS_BIRTH','IV'] = iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'gp_Age')
```

Años de Trabajo

In [ ]:
```
new_data['worktm']=-(new_data['DAYS_EMPLOYED'])//365
new_data[new_data['worktm']<0] = np.nan # replace by na
new_data['DAYS_EMPLOYED']
new_data['worktm'].fillna(new_data['worktm'].mean(),inplace=True) #re
place na by mean
```

In [ ]:
```
new_data['worktm'].plot(kind='hist',bins=20,density=True)
```

In [ ]:
```
new_data = get_category(new_data,'worktm',5, ["lowest","low","medium"
,"high","highest"])
iv, data=calc_iv(new_data,'gp_worktm','target')
ivtable.loc[ivtable['variable']=='DAYS_EMPLOYED','IV']=iv
data.head()
```

In [ ]:
```
# Codificación One-Hot
new_data = convert_dummy(new_data,'gp_worktm')
```

Tamaño de Familia

In [ ]:
```
new_data['famsize'].value_counts(sort=False)
```

In [ ]:
```
new_data['famsize']=new_data['famsize'].astype(int)
new_data['famsizegp']=new_data['famsize']
new_data['famsizegp']=new_data['famsizegp'].astype(object)
new_data.loc[new_data['famsizegp']>=3,'famsizegp']='3more'
iv, data=calc_iv(new_data,'famsizegp','target')
ivtable.loc[ivtable['variable']=='famsize','IV']=iv
data.head()
```

In [ ]:
```
# Codificación One-Hot
new_data = convert_dummy(new_data,'famsizegp')
```

Descriptores Categóricos

Forma de Ingresos

In [ ]:
```
print(new_data['inctp'].value_counts(sort=False))
print(new_data['inctp'].value_counts(normalize=True,sort=False))
new_data.loc[new_data['inctp']=='Pensioner','inctp']='State servant'
new_data.loc[new_data['inctp']=='Student','inctp']='State servant'
iv, data=calc_iv(new_data,'inctp','target')
ivtable.loc[ivtable['variable']=='inctp','IV']=iv
data.head()
```

In [ ]:
```
# Codificación One-Hot
new_data = convert_dummy(new_data,'inctp')
```

Tipo de Ocupación

In [ ]:
```
new_data.loc[(new_data['occyp']=='Cleaning staff') | (new_data['occy
p']=='Cooking staff') | (new_data['occyp']=='Drivers') | (new_data['o
ccyp']=='Laborers') | (new_data['occyp']=='Low-skill Laborers') | (ne
w_data['occyp']=='Security staff') | (new_data['occyp']=='Waiters/bar
men staff'),'occyp']='Laborwk'
new_data.loc[(new_data['occyp']=='Accountants') | (new_data['occyp']=
='Core staff') | (new_data['occyp']=='HR staff') | (new_data['occyp']
=='Medicine staff') | (new_data['occyp']=='Private service staff') |
(new_data['occyp']=='Realty agents') | (new_data['occyp']=='Sales sta
ff') | (new_data['occyp']=='Secretaries'),'occyp']='officewk'
new_data.loc[(new_data['occyp']=='Managers') | (new_data['occyp']=='H
igh skill tech staff') | (new_data['occyp']=='IT staff'),'occyp']='hi
ghtecwk'
print(new_data['occyp'].value_counts())
iv, data=calc_iv(new_data,'occyp','target')
ivtable.loc[ivtable['variable']=='occyp','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'occyp')
```

In [ ]:
```
new_data.columns
```

Forma de Vivienda

In [ ]:
```
iv, data=calc_iv(new_data,'houtp','target')
ivtable.loc[ivtable['variable']=='houtp','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'houtp')
```

In [ ]:
```
new_data.columns
```

Educación

In [ ]:
```
new_data.loc[new_data['edutp']=='Academic degree','edutp']='Higher ed
ucation'
iv, data=calc_iv(new_data,'edutp','target')
ivtable.loc[ivtable['variable']=='edutp','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'edutp')
```

In [ ]:
```
new_data.columns
```

Estado Civil

In [ ]:
```
new_data['famtp'].value_counts(normalize=True,sort=False)
```

In [ ]:
```
iv, data=calc_iv(new_data,'famtp','target')
ivtable.loc[ivtable['variable']=='famtp','IV']=iv
data.head()
```

In [ ]:
```
new_data = convert_dummy(new_data,'famtp')
```

In [ ]:
```
new_data.columns
```

Utilidad de: IV y WoE

Puede leer el artículo a continuación para comprender un poco más sobre los conceptos de IV y WoE:

https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB_sfire-dsc_user-guide/GUID-07A78308-525A-406F-8221-9281F4E9D7CF.html (https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB_sfire-dsc_user-guide/GUID-07A78308-525A-406F-8221-9281F4E9D7CF.html)

La tabla a continuación fue tomada de la referencia indicada:

IV	Ability to predict
<0.02	Bajo poder predictivo
0.02~0.1	Poder predictivo débil
0.1~0.3	Poder predictivo moderado
0.3~0.5	Poder predictivo fuerte
>0.5	Sospechosamente alto, revisar esta variable

In [ ]:
```
ivtable=ivtable.sort_values(by='IV',ascending=False)
ivtable.loc[ivtable['variable']=='DAYS_BIRTH','variable']='agegp'
ivtable.loc[ivtable['variable']=='DAYS_EMPLOYED','variable']='worktmgp'
ivtable.loc[ivtable['variable']=='inc','variable']='incgp'
ivtable
```

Predicción de Buen/Mal Cliente Mediante Redes Neuronales

- Split Dataset

In [ ]:
```
new_data.columns
```

Se tomarán únicamente aquellas columnas preprocesadas y con un $IV > 0.001$

In [ ]:
```
Y = new_data['target']
X = new_data[['Gender','Reality','ChldNo_1', 'ChldNo_2More', 'gp_inc_
medium',  'gp_inc_high','wkphone',
              'gp_Age_high', 'gp_Age_highest', 'gp_Age_low',
              'gp_Age_lowest','gp_worktm_high', 'gp_worktm_highest',
              'gp_worktm_low', 'gp_worktm_medium','occyp_hightecwk',
              'occyp_officewk','famsizegp_1', 'famsizegp_3more',
              'houtp_Co-op apartment', 'houtp_Municipal apartment',
              'houtp_Office apartment', 'houtp_Rented apartment',
              'houtp_With parents','edutp_Higher education',
              'edutp_Incomplete higher', 'edutp_Lower secondary','fam
tp_Civil marriage',
              'famtp_Separated','famtp_Single / not married','famtp_W
idow']]
```

SMOTE

Concepto: Synthetic Minority Over-Sampling Technique(SMOTE) utilizado para lidiar con datos desbalanceados. Puede encontrar más información en:

- http://glemaitre.github.io/imbalanced-learn/generated/imblearn.over_sampling.SMOTE.html (http://glemaitre.github.io/imbalanced-learn/generated/imblearn.over_sampling.SMOTE.html)
- https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/)

In [ ]:
```
Y = Y.astype('int')
sm = SMOTE()
X_balance,Y_balance = sm.fit_resample(X,Y)
X_balance = pd.DataFrame(X_balance, columns = X.columns)
```

Separación de datos en conjuntos: entrenamiento y prueba.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_balance,Y_balan
ce,
                                                    stratify=Y_balanc
e, test_size=0.3,
                                                    random_state = 10
086)
```

Seleccione esta celda y luego la opción Run All Above

PARTE 1

Separación de Conjunto de Features

Teniendo en cuenta los resultados de IV obtenidos anteriormente, comprobaremos la capacidad predictiva de tres conjuntos de datos basados en la tabla anterior. Primero removeremos los últimos cuatro ('phone', 'inctp', 'email', 'Car'), y luego realizaremos la siguiente división:

- A. Primera mitad: 'agegp', 'famtp', 'worktmgp', 'Reality', 'Gender', 'edutp'
- B. Segunda mitad: 'houtp', 'famsize', 'occyp', 'incgp', 'wkphone', 'ChldNo'
- C. Todos los descriptores.

De acuerdo a estos nombres, utilice la siguiente lista para identificar aquellos solicitados en cada caso:

```
    'Gender','Reality','ChldNo_1', 'ChldNo_2More', 'gp_inc_medium',  'gp_inc
_high','wkphone',
    'gp_Age_high', 'gp_Age_highest', 'gp_Age_low',
    'gp_Age_lowest','gp_worktm_high', 'gp_worktm_highest',
    'gp_worktm_low', 'gp_worktm_medium','occyp_hightecwk',
    'occyp_officewk','famsizegp_1', 'famsizegp_3more',
    'houtp_Co-op apartment', 'houtp_Municipal apartment',
    'houtp_Office apartment', 'houtp_Rented apartment',
    'houtp_With parents','edutp_Higher education',
    'edutp_Incomplete higher', 'edutp_Lower secondary','famtp_Civil marriag
e',
    'famtp_Separated','famtp_Single / not married','famtp_Widow'
```

A. Top 6

In [ ]:
```
X_train_subA = X_train[[# 'descriptor_1', 'descriptor_2', ... #]]

X_test_subA = X_test[[# 'descriptor_1', 'descriptor_2', ... #]]

X_train_subA.head()
```

B. Últimos 6

In [ ]:
```
X_train_subB = X_train[[# 'descriptor_1', 'descriptor_2', ... #]]

X_test_subB = X_test[[# 'descriptor_1', 'descriptor_2', ... #]]

X_train_subB.head()
```

Caso C

Acá simplemente tomaremos, no es necesario crear nuevas variables:

```
X_train_subC = X_train
X_test_subC = X_test
```

PARTE 2

Implementación de pruebas en los conjuntos de descriptores. A continuación debe implementar, inicialmente tres modelos de regresión logística, y posteriormente tres redes neuronales (2 capas escondidas, 20 neuronas en cada una). Observe los resultados y analice lo sucedido. Concluya sobre qué modelo es deseable teniendo en cuenta la factibilidad de implementación práctica y la matriz de confusión correspondiente.

Regresión Logística

Inicialmente se probará un modelo de regresión logística para tener una referencia (también se conoce como *baseline*) y comprobar que un modelo de red neuronal permite obtener mejores resultados.

$$\log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_q x_q$$

Caso A

In [ ]:
```
model = LogisticRegression(C=0.8,
                           random_state=0,
                           solver='lbfgs')
model.fit(  ) # Ajuste el modelo con los datos del conjunto A #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto A #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes= class_names,
                      title='Matriz de Confusión Normalizada: Regresi
ón Logística')
```

Caso B

In [ ]:
```
model = LogisticRegression(C=0.8,
                           random_state=0,
                           solver='lbfgs')
model.fit(  ) # Ajuste el modelo con los datos del conjunto B #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto B #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes= class_names,
                      title='Matriz de Confusión Normalizada: Regresi
ón Logística')
```

Caso C

In [ ]:
```
model = LogisticRegression(C=0.8,
                           random_state=0,
                           solver='lbfgs')
model.fit(  ) # Ajuste el modelo con los datos del conjunto C #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto C #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

sns.set_style('white')
class_names = ['0','1']
plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes= class_names,
                      title='Matriz de Confusión Normalizada: Regresi
ón Logística')
```

Red Neuronal, Perceptrón Multicapa

Ahora utilice la función `MLPClassifier` de la librería SciKit-Learn para desarrollar una red neuronal que permita mejorar el rendimiento del clasificador *baseline* desarrollado.

Caso A

In [ ]:
```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier(   ) # Inicialice un modelo de red neuronal con
2 capas escondidas x 20 neuronas en cada capa y función de activación
ReLu #
model.fit(  ) # Ajuste el modelo con los datos del conjunto A #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto A #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes=class_names,
                      title='Matriz de Confusión Normalizada: Red Neu
ronal')
```

Caso B

In [ ]:
```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier(   ) # Inicialice un modelo de red neuronal con
2 capas escondidas x 20 neuronas en cada capa y función de activación
ReLu #
model.fit(  ) # Ajuste el modelo con los datos del conjunto B #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto B #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes=class_names,
                      title='Matriz de Confusión Normalizada: Red Neu
ronal')
```

Caso C

In [ ]:
```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier(   ) # Inicialice un modelo de red neuronal con
2 capas escondidas x 20 neuronas en cada capa y función de activación
ReLu #
model.fit(  ) # Ajuste el modelo con los datos del conjunto C #
y_predict = model.predict(  ) # Realice la predicción de etiquetas co
n los datos de prueba del conjunto Cmm #

print(f'Precisión {round(accuracy_score(y_test, y_predict),5)}')
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))

plot_confusion_matrix(confusion_matrix(y_test,y_predict),
                      classes=class_names,
                      title='Matriz de Confusión Normalizada: Red Neu
ronal')
```

Conclusiones Parte 2

Concluya con respecto a sus resultados.

PARTE 3

Los resultados obtenidos para las redes neuronales anteriores únicamente corresponden a una arquitectura. Un proceso necesario e importante en casos de estudio como este es la búsqueda de hiperparámetros, en este caso, el número de neuronas más adecuado (al menos dentro de cierto rango, este proceso también se conoce como GridSearch (https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e)).

Divida el conjunto de datos de prueba en dos mitades: datos de validación y datos de prueba. Utilice los datos de validación para realizar una evaluación preliminar de cada modelo. Utilice `train_test_split` para esta parte.

Utilice todos los descriptores para esta prueba y realice las siguientes búsquedas:

- Caso A: 1 capa escondida $\times$ {5, 10, 20, 50, 100} neuronas.
- Caso B: 2 capas escondidas $\times$ {5, 10, 20, 50, 100} neuronas.
- Caso C: 3 capas escondidas $\times$ {5, 10, 20, 50, 100} neuronas.

Utilice `matplotlib.pyplot` para graficar la precisión en los datos de validación, seleccione el mejor modelo, y obtenga una evaluación final para esta selección utilizando los datos de prueba.

Caso A

In [ ]:
```
# # # # Implemente un ciclo FOR para entrenar cada una de las configu
raciones solicitadas # # # #
#
#
#
#
#
#
# ...
```

Caso B

In [ ]:
```
# # # # Implemente un ciclo FOR para entrenar cada una de las configu
raciones solicitadas # # # #
#
#
#
#
#
#
# ...
```

Caso C

In [ ]:
```
# # # # Implemente un ciclo FOR para entrenar cada una de las configu
raciones solicitadas # # # #
#
#
#
#
#
#
# ...
```

Gráficas Evaluativas

Bono (1 punto)

Implemente el mejor modelo de red neuronal sin utilizar ningún tipo de librería que tenga funciones prestablecidas con objetivos de apoyo en el tema de Machine Learning. Puede utilizar Numpy, Pandas, etc. NO puede utilizar: SciKit-Learn, Tensorflow/Keras, PyTorch, etc.

In [ ]:
```

```