Modelos - ROSE-UNDER

February 2, 2022

1 MODELOS DE MACHINE LEARNING

Autor: Jenny Marisol Tenisaca Moposita

Importar librerias

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.1 Conjunto con Datos balanceados método ROSE - BOTH

Importar el data de entrenamiento (balanceados) y data test

Cargar conjuntos de entrenamiento balanceados (4 métodos) y conjunto de test. (los conjuntos de entrenamiento y test fueron dividimos en 80% y 20%)

[2]:
```
#data entrenamiento balanceado con ROSE-BOTH
data_train_bal = pd.read_csv('Data_train_Rose_Under.csv',␣
↪encoding='latin-1',sep=';')
# 80% balanceado
```

[4]:
```
#data test
data_test = pd.read_csv('Data_test.csv',sep=';')
# 20% de la data completa
```

[5]:
```
X_train=data_train_bal.iloc[:,1:23].values
y_train=data_train_bal.iloc[:,0].values
X_test=data_test.iloc[:,1:23].values
y_test=data_test.iloc[:,0].values
```

1.2 Ajustar el clasificador Random Forest en el Conjunto de Entrenamiento

[11]:
```
#Validación cruzada (datos)
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
kf =KFold(n_splits=5, shuffle=True, random_state=42)
```

```
score = cross_val_score(RandomForestClassifier(n_estimators = 100, n_jobs=2,␣
↪criterion = "entropy", random_state = 123), X_train, y_train, cv= kf,␣
↪scoring="accuracy")
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.4f}".format(score.mean())}')
```

```
Scores for each fold are: [0.99539411 0.9952203  0.99591553 0.99565482
0.99556792]
Average score: 0.9956
```

[12]:

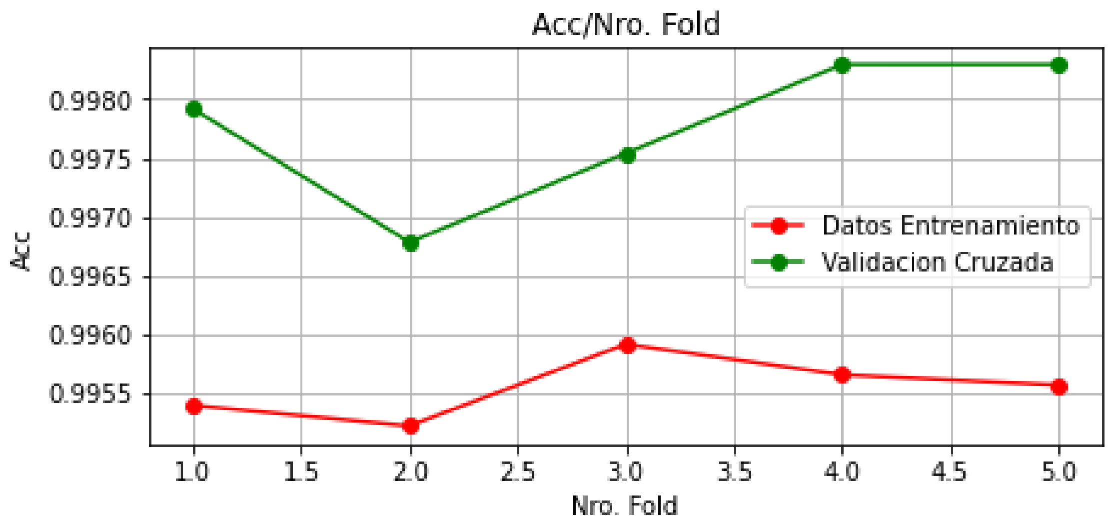
```
#Validación cruzada (gráfico y datos)
def graficar_Accu_scores(estimator, X_train,␣
↪y_train,X_test,y_test,nparts=5,jobs=None):
    kf = KFold(n_splits=nparts,shuffle=True, random_state=42)
    fig,axes = plt.subplots(figsize=(7, 3))
    axes.set_title("Acc/Nro. Fold")
    axes.set_xlabel("Nro. Fold")
    axes.set_ylabel("Acc")
    train_scores = cross_val_score(estimator, X_train, y_train, cv = kf,␣
↪n_jobs=jobs,scoring="accuracy")
    test_scores = cross_val_score(estimator, X_test,y_test, cv = kf,␣
↪n_jobs=jobs,scoring="accuracy")
    train_sizes = range(1,nparts+1,1)
    axes.grid()
    axes.plot(train_sizes, train_scores, 'o-', color="r",label="Datos␣
↪Entrenamiento")
    axes.plot(train_sizes, test_scores, 'o-', color="g",label="Validacion␣
↪Cruzada")
    axes.legend(loc="best")
    return train_scores
```

[13]:

```
#Validación cruzada (gráfico y datos)
graficar_Accu_scores(clas_rndforest,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[13]:

```
array([0.99539411, 0.9952203 , 0.99591553, 0.99565482, 0.99556792])
```



[6]:
```
from sklearn.ensemble import RandomForestClassifier
clas_rndforest = RandomForestClassifier(n_estimators = 100, n_jobs=2, criterion␣
↪= "entropy", random_state = 123)
clas_rndforest.fit(X_train, y_train)
```

[6]:
```
RandomForestClassifier(criterion='entropy', n_jobs=2, random_state=123)
```

1.2.1 Predicción resultados

[8]:
```
y_pred  = clas_rndforest.predict(X_test)
```

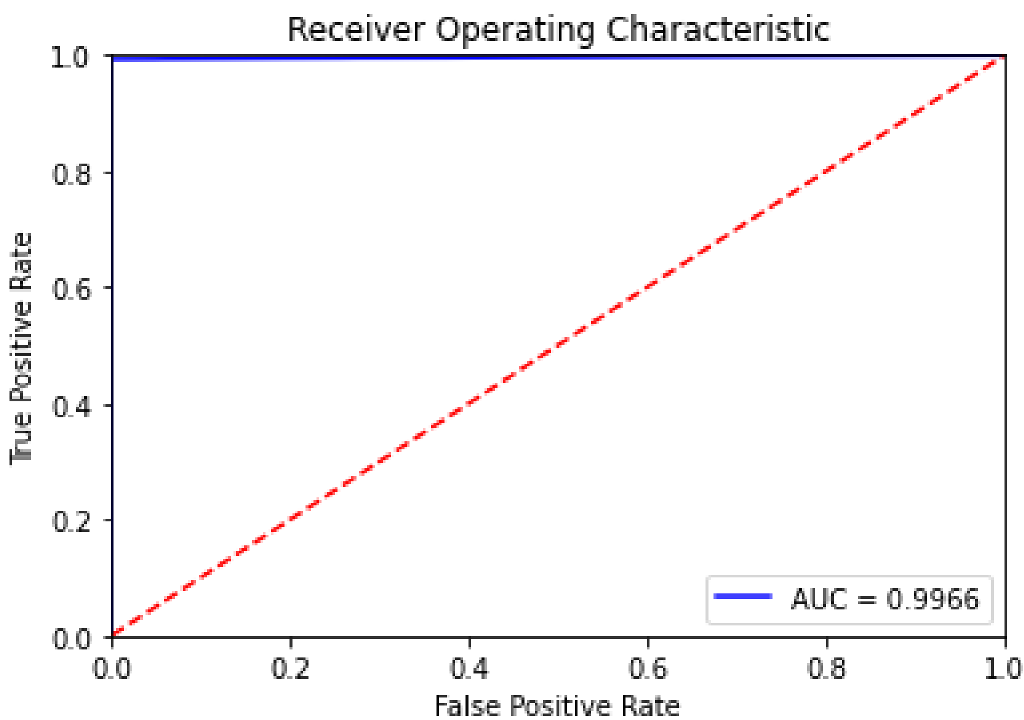
[10]:
```
##matriz de confusión
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[10]:
```
array([[19249,     0],
       [   52,  7123]], dtype=int64)
```

[14]:
```
#Curvas ROC
import sklearn.metrics as metrics
# calcular fpr y tpr para todos los thresholds de la clasificación
probs = clas_rndforest.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



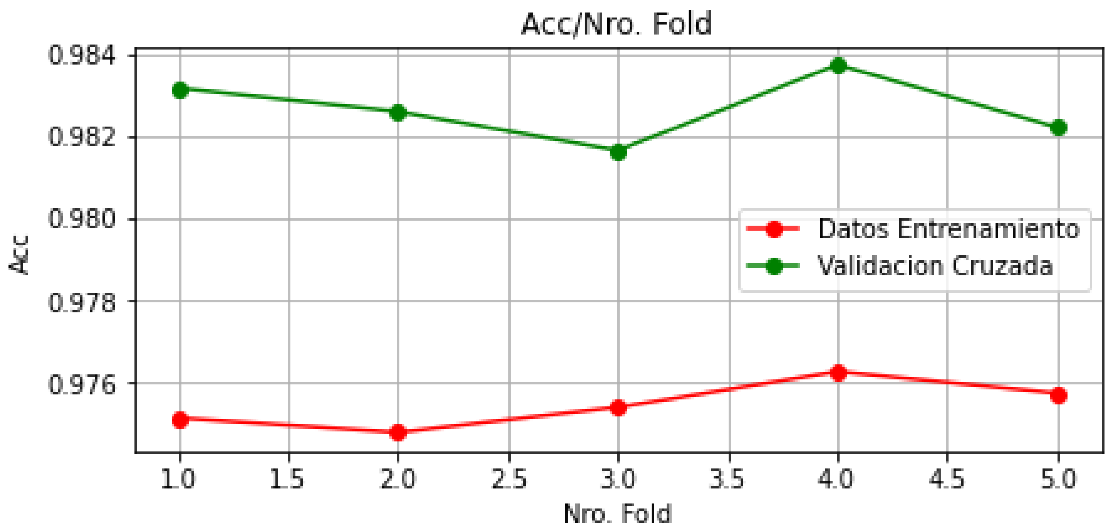
1.3 Ajustar el clasificador SVM en el Conjunto de Entrenamiento

[18]:
```
#validación cruzada (datos)
from sklearn.model_selection import KFold, cross_val_score
from sklearn.svm import SVC
kf =KFold(n_splits=5, shuffle=True, random_state=42)
score = cross_val_score(SVC(kernel='rbf',random_state=123), X_train, y_train,␣
↪cv= kf, scoring="accuracy")
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.4f}".format(score.mean())}')
```

```
Scores for each fold are: [0.97514556 0.97479795 0.97540627 0.97627531
0.97575389]
Average score: 0.9755
```

[19]:
```python
#Validación cruzada (gráfico y datos)
graficar_Accu_scores(class_svm,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[19]:
```
array([0.97514556, 0.97479795, 0.97540627, 0.97627531, 0.97575389])
```



[15]:
```python
# Fitting SVM to the Training set using Kernel as rbf.
from sklearn.svm import SVC
class_svm = SVC(kernel='rbf',probability=True,random_state=123)
class_svm.fit(X_train, y_train)
```

[15]:
```
SVC(probability=True, random_state=123)
```

[16]:
```python
# Predecir los resultados
y_pred_svm = class_svm.predict(X_test)
```

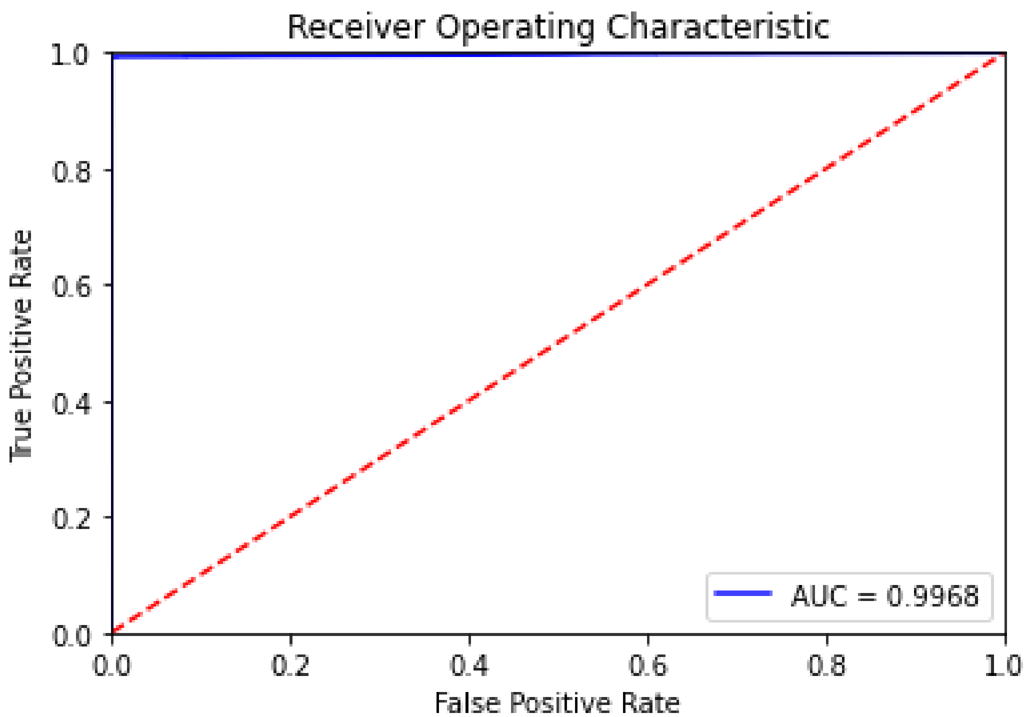
[17]:
```python
# Matriz de confusión
from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_svm # display
```

[17]:
```
array([[19249,     0],
       [  317,  6858]], dtype=int64)
```

[20]:
```python
#Curvas ROC
import sklearn.metrics as metrics
# calcular fpr y tpr para todos los thresholds de la clasificación
probs = class_svm.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
```

```python
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



1.4 Ajustar el clasificador NAIVE BAYES en el Conjunto de Entrenamiento

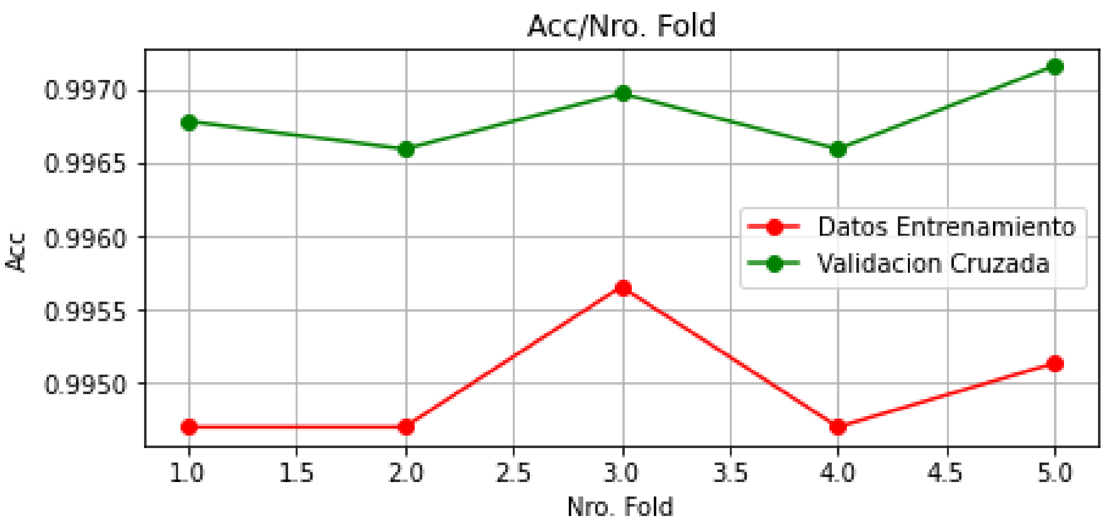
[24]:
```python
#validación cruzada (datos)
from sklearn.model_selection import KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB
kf =KFold(n_splits=5, shuffle=True, random_state=42)
score = cross_val_score(GaussianNB(), X_train, y_train, cv= kf,␣
 ↪scoring="accuracy")
print(f'Scores for each fold are: {score}')
```

```
print(f'Average score: {"{:.4f}".format(score.mean())}')
```

```
Scores for each fold are: [0.99469888 0.99469888 0.99565482 0.99469888 0.9951334
]
Average score: 0.9950
```

[25]:
```
#Validación cruzada
graficar_Accu_scores(class_nb,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[25]:
```
array([0.99469888, 0.99469888, 0.99565482, 0.99469888, 0.9951334 ])
```



[21]:
```
from sklearn.naive_bayes import GaussianNB
class_nb = GaussianNB()
class_nb.fit(X_train, y_train)
```

[21]:
```
GaussianNB()
```

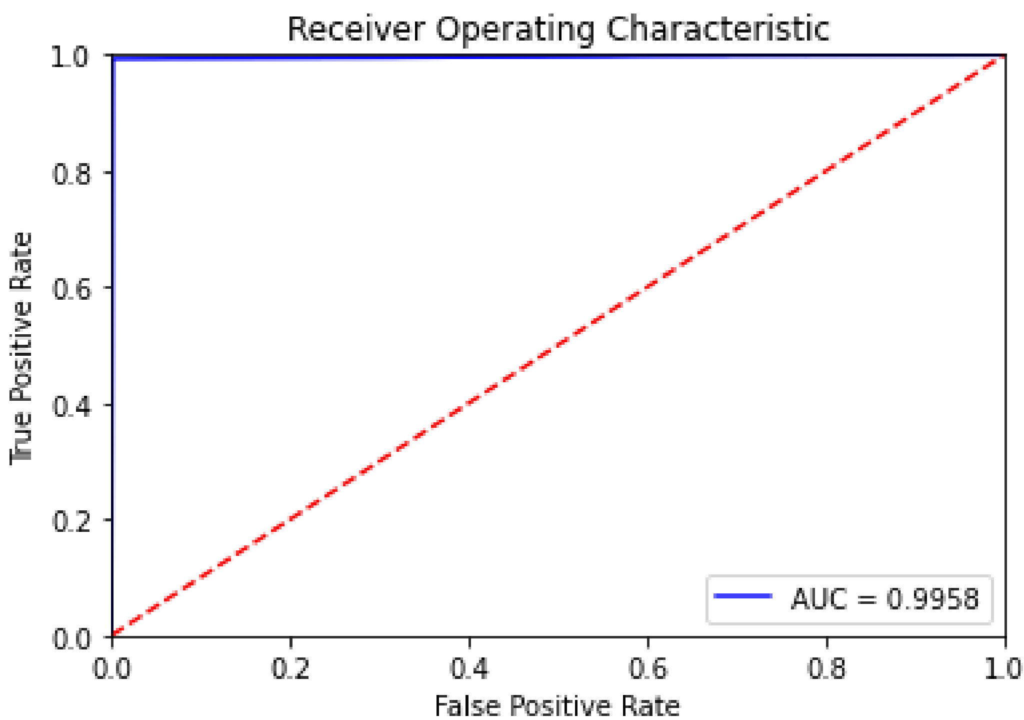
[22]:
```
# Predecir los resultados
y_pred_nb  = class_nb.predict(X_test)
```

[23]:
```
# Matriz de confusión
from sklearn.metrics import confusion_matrix
cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_nb # display
```

[23]:
```
array([[  142, 19107],
       [    0,  7175]], dtype=int64)
```

[26]:
```
#Curvas ROC
import sklearn.metrics as metrics
# calcular fpr y tpr para todos los thresholds de la clasificación
probs = class_nb.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc1 = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.4f' % roc_auc1)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



1.5 Ajustar el clasificador REDES NEURONALES en el Conjunto de Entrenamiento

[33]:
```
#validación cruzada (datos)
import keras
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import KFold, cross_val_score

def built_class_RN():
    #Inicializar la RNA
    class_RN = Sequential()
    #Añadir las capas de entrada y primera capa oculta
    class_RN.add(Dense(units = 12, kernel_initializer = "uniform", activation =␣
 ↪"relu", input_dim = 23))
    #Añadir la segunda capa oculta
    class_RN.add(Dense(units = 10, kernel_initializer = "uniform",  activation␣
 ↪= "relu"))
    #Añadir la capa de salida
    class_RN.add(Dense(units = 1, kernel_initializer = "uniform",  activation =␣
 ↪"sigmoid"))
    #Compilar la RNA
    class_RN.compile(optimizer = "adam", loss = "binary_crossentropy", metrics␣
 ↪= ["accuracy"])
    return class_RN

#Ajustar la RNA al Conjunto de Entrenamiento
class_RN = KerasClassifier(build_fn=built_class_RN, batch_size = 10, epochs =␣
 ↪100)
kf =KFold(n_splits=5, shuffle=True, random_state=42)
Accuracy = cross_val_score(class_RN, X_train, y_train, cv= kf, n_jobs=-1)
```

```
C:\Users\jenny\AppData\Local\Temp/ipykernel_15192/2278733859.py:22:
DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead.
  class_RN = KerasClassifier(build_fn=built_class_RN, batch_size = 10, epochs =
100)
```

[34]:
```
print(f'Scores for each fold are: {Accuracy}')
print(f'Average score: {"{:.4f}".format(Accuracy.mean())}')
```

```
Scores for each fold are: [0.98905015 0.98592162 0.98739898 0.99061441
0.98826802]
Average score: 0.9883
```

[3]:
```
import keras
from keras.models import Sequential
```

```
from keras.layers import Dense
```

[9]:
```
#Inicializar la RNA
class_RN = Sequential()
#Añadir las capas de entrada y primera capa oculta
class_RN.add(Dense(units = 12, kernel_initializer = "uniform",
                      activation = "relu", input_dim = 23))
#Añadir la segunda capa oculta
class_RN.add(Dense(units = 10, kernel_initializer = "uniform",  activation =␣
 ↪"relu"))
#Añadir la capa de salida
class_RN.add(Dense(units = 1, kernel_initializer = "uniform",  activation =␣
 ↪"sigmoid"))
#Compilar la RNA
class_RN.compile(optimizer = "adam", loss = "binary_crossentropy", metrics =␣
 ↪["accuracy"])
#Ajustar la RNA al Conjunto de Entrenamiento
class_RN.fit(X_train, y_train,  batch_size = 10, epochs = 100)
```

```
Epoch 1/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.1260 -
accuracy: 0.9612
Epoch 2/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0768 -
accuracy: 0.9798
Epoch 3/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0716 -
accuracy: 0.9822
Epoch 4/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0686 -
accuracy: 0.9830
Epoch 5/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0676 -
accuracy: 0.9836
Epoch 6/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0660 -
accuracy: 0.9839: 0s - loss: 0.0659
Epoch 7/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0647 -
accuracy: 0.9844
Epoch 8/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0642 -
accuracy: 0.9842
Epoch 9/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0651 -
accuracy: 0.9842
Epoch 10/100
```

```
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0630 -
accuracy: 0.9848
Epoch 11/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0626 -
accuracy: 0.9846
Epoch 12/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0628 -
accuracy: 0.9848
Epoch 13/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0621 -
accuracy: 0.9851
Epoch 14/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0620 -
accuracy: 0.9853
Epoch 15/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0611 -
accuracy: 0.9854
Epoch 16/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0609 -
accuracy: 0.9852
Epoch 17/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0613 -
accuracy: 0.9854
Epoch 18/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0606 -
accuracy: 0.9854
Epoch 19/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0599 -
accuracy: 0.9859
Epoch 20/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0602 -
accuracy: 0.9856
Epoch 21/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0604 -
accuracy: 0.9856
Epoch 22/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0596 -
accuracy: 0.9856
Epoch 23/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0586 -
accuracy: 0.9860
Epoch 24/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0587 -
accuracy: 0.9860
Epoch 25/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0585 -
accuracy: 0.9860
Epoch 26/100
```

```
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0591 -
accuracy: 0.9861
Epoch 27/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0589 -
accuracy: 0.9859
Epoch 28/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0583 -
accuracy: 0.9860
Epoch 29/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0574 -
accuracy: 0.9863
Epoch 30/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0583 -
accuracy: 0.9859
Epoch 31/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0576 -
accuracy: 0.9862
Epoch 32/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0567 -
accuracy: 0.9864: 0s - loss: 0.056
Epoch 33/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0574 -
accuracy: 0.9863
Epoch 34/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0572 -
accuracy: 0.9863
Epoch 35/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0563 -
accuracy: 0.9864
Epoch 36/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0563 -
accuracy: 0.9865
Epoch 37/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0557 -
accuracy: 0.9867
Epoch 38/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0564 -
accuracy: 0.9866
Epoch 39/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0556 -
accuracy: 0.9865
Epoch 40/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0553 -
accuracy: 0.9867
Epoch 41/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0553 -
accuracy: 0.9868
Epoch 42/100
```

```
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0555 -
accuracy: 0.9868: 0s - loss: 0.0554 -
Epoch 43/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0543 -
accuracy: 0.9869
Epoch 44/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0535 -
accuracy: 0.9872
Epoch 45/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0545 -
accuracy: 0.9871
Epoch 46/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0546 -
accuracy: 0.9872
Epoch 47/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0536 -
accuracy: 0.9873
Epoch 48/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0543 -
accuracy: 0.9871
Epoch 49/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0531 -
accuracy: 0.9874
Epoch 50/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0548 -
accuracy: 0.9869
Epoch 51/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0537 -
accuracy: 0.9875
Epoch 52/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0534 -
accuracy: 0.9876
Epoch 53/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0532 -
accuracy: 0.9873
Epoch 54/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0524 -
accuracy: 0.9874
Epoch 55/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0525 -
accuracy: 0.9874
Epoch 56/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0520 -
accuracy: 0.9879
Epoch 57/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0528 -
accuracy: 0.9875
Epoch 58/100
```

```
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0539 -
accuracy: 0.9871
Epoch 59/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0522 -
accuracy: 0.9877
Epoch 60/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0529 -
accuracy: 0.9870
Epoch 61/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0507 -
accuracy: 0.9883
Epoch 62/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0519 -
accuracy: 0.9876
Epoch 63/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0524 -
accuracy: 0.9875
Epoch 64/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0515 -
accuracy: 0.9877
Epoch 65/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0529 -
accuracy: 0.9873
Epoch 66/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0514 -
accuracy: 0.9878
Epoch 67/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0509 -
accuracy: 0.9880
Epoch 68/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0507 -
accuracy: 0.9881
Epoch 69/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0505 -
accuracy: 0.9880
Epoch 70/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0513 -
accuracy: 0.9877
Epoch 71/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0503 -
accuracy: 0.9880
Epoch 72/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0504 -
accuracy: 0.9880
Epoch 73/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0506 -
accuracy: 0.9884
Epoch 74/100
```

```
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0500 -
accuracy: 0.9881
Epoch 75/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0503 -
accuracy: 0.9880
Epoch 76/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0502 -
accuracy: 0.9881
Epoch 77/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0506 -
accuracy: 0.9880
Epoch 78/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0494 -
accuracy: 0.9883
Epoch 79/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0501 -
accuracy: 0.9880
Epoch 80/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0493 -
accuracy: 0.9886: 0s - loss: 0.0493 - accura
Epoch 81/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0501 -
accuracy: 0.9880
Epoch 82/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0494 -
accuracy: 0.9883
Epoch 83/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0490 -
accuracy: 0.9885
Epoch 84/100
5754/5754 [==============================] - 8s 1ms/step - loss: 0.0487 -
accuracy: 0.9885
Epoch 85/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0493 -
accuracy: 0.9884
Epoch 86/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0493 -
accuracy: 0.9883
Epoch 87/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0479 -
accuracy: 0.9887
Epoch 88/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0486 -
accuracy: 0.9886
Epoch 89/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0487 -
accuracy: 0.9884
Epoch 90/100
```

```
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0474 -
accuracy: 0.9888
Epoch 91/100
5754/5754 [==============================] - 6s 1ms/step - loss: 0.0489 -
accuracy: 0.9885
Epoch 92/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0501 -
accuracy: 0.9881
Epoch 93/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0495 -
accuracy: 0.9885
Epoch 94/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0478 -
accuracy: 0.9887
Epoch 95/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0482 -
accuracy: 0.9887
Epoch 96/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0487 -
accuracy: 0.9887
Epoch 97/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0477 -
accuracy: 0.9886
Epoch 98/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0482 -
accuracy: 0.9887
Epoch 99/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0467 -
accuracy: 0.9892
Epoch 100/100
5754/5754 [==============================] - 7s 1ms/step - loss: 0.0473 -
accuracy: 0.9889
```

[9]:
```
<keras.callbacks.History at 0x1e544d87bb0>
```

[10]:
```python
test_loss, test_acc = class_RN.evaluate(X_test,  y_test, verbose=2)
print('\nTest Accuracy:', test_acc)
```

```
826/826 - 1s - loss: 0.0391 - accuracy: 0.9936 - 676ms/epoch - 819us/step

Test Accuracy: 0.9936421513557434
```

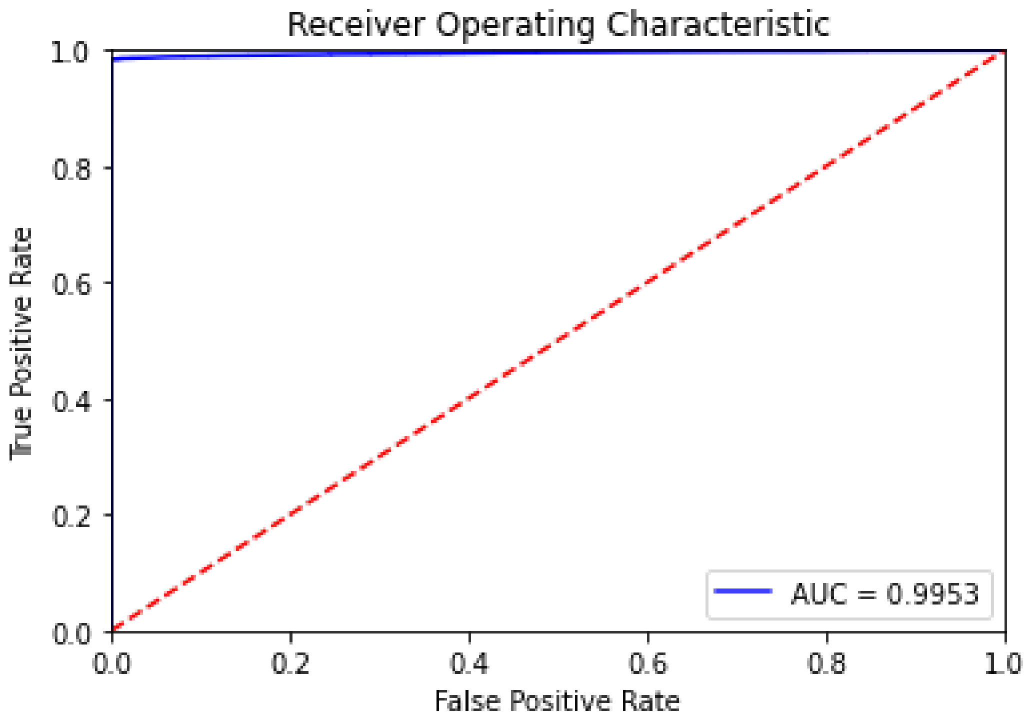
[11]:
```python
# Evaluar el modelo y calcular predicciones finales
# Predicción de los resultados con el Conjunto de Testing
y_pred_rn  = class_RN.predict(X_test)
y_pred_rn = (y_pred_rn>0.5)
```

[12]:
```python
#Elaborar una matriz de confusión
from sklearn.metrics import confusion_matrix
cm_rn = confusion_matrix(y_test, y_pred_rn)
cm_rn
```

[12]:
```
array([[19196,    53],
       [  115,  7060]], dtype=int64)
```

[14]:
```python
#Curva ROC
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
y_pred_rn_curv = class_RN.predict(X_test).ravel()

nn_fpr_keras, nn_tpr_keras, nn_thresholds_keras = roc_curve(y_test,␣
 ↪y_pred_rn_curv)
auc_keras = auc(nn_fpr_keras, nn_tpr_keras)
# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(nn_fpr_keras, nn_tpr_keras, 'b', label = 'AUC = %0.4f' % auc_keras)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.9953
0.0
0.2
0.4
0.6
0.8
1.0