Modelos - ROSE BOTH

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%)

[27]:
```
#data entrenamiento balanceado con ROSE-BOTH
data_train_bal = pd.read_csv('Data_train_Rose_Both.csv',␣
 ↪encoding='latin-1',sep=';')
# 80% balanceado
```

[28]:
```
#data test
data_test = pd.read_csv('Data_test.csv',sep=';')
# 20% de la data completa
```

[29]:
```
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

[78]:
```
#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)
```

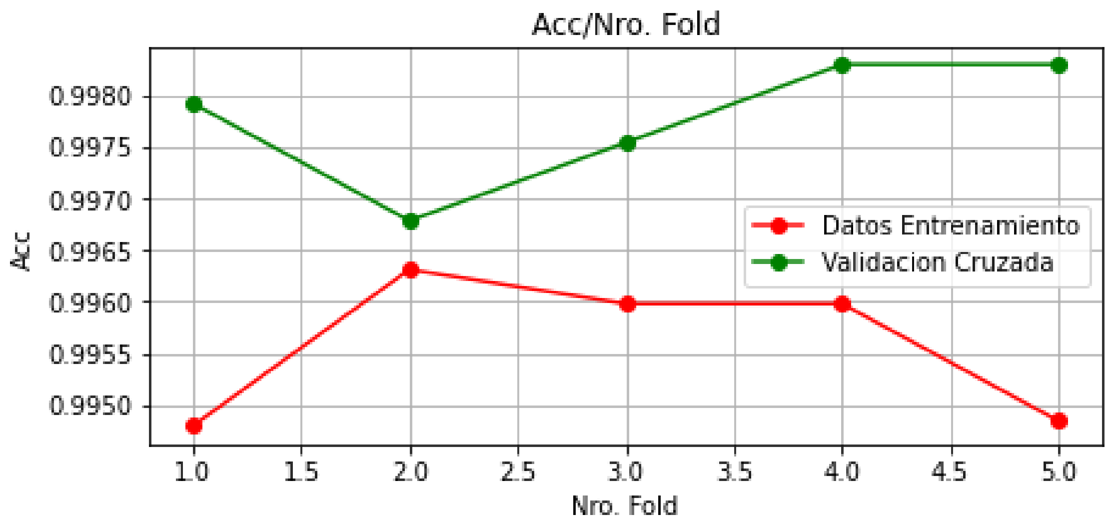
```
score = cross_val_score(RandomForestClassifier(n_estimators = 100, n_jobs=2,␣
↪criterion = "entropy", random_state = 123), X_train, y_train, cv= kf,␣
↪scoring="accuracy")
print(f'Scores for each fold are: {score}')
print(f'Average score: {"{:.4f}".format(score.mean())}')
```

```
Scores for each fold are: [0.99479659 0.99631031 0.99597919 0.995979
0.99484365]
Average score: 0.9956
```

[42]:
```
#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
```

[43]:
```
#Validación cruzada (gráfico y datos)
graficar_Accu_scores(clas_rndforest,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[43]:
```
array([0.99479659, 0.99631031, 0.99597919, 0.995979  , 0.99484365])
```



[30]:
```python
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)
```

[30]:
```
RandomForestClassifier(criterion='entropy', n_jobs=2, random_state=123)
```

1.2.1 Predicción resultados

[31]:
```python
y_pred  = clas_rndforest.predict(X_test)
```

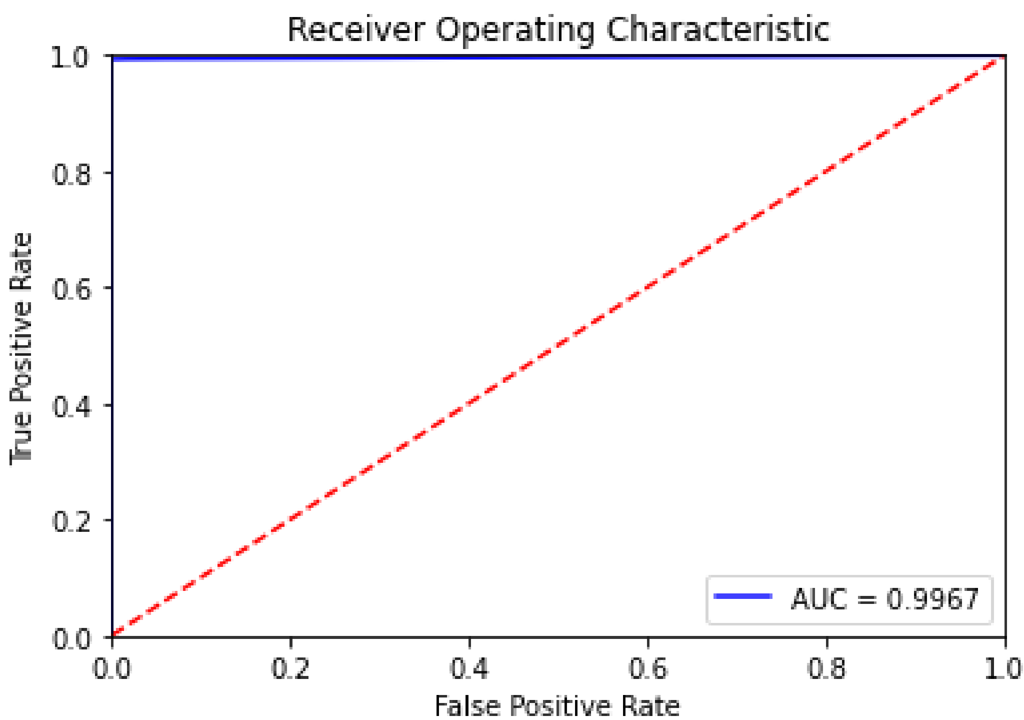
[33]:
```python
##matriz de confusión
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[33]:
```
array([[19237,    12],
       [   52,  7123]], dtype=int64)
```

[77]:
```python
#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()
```



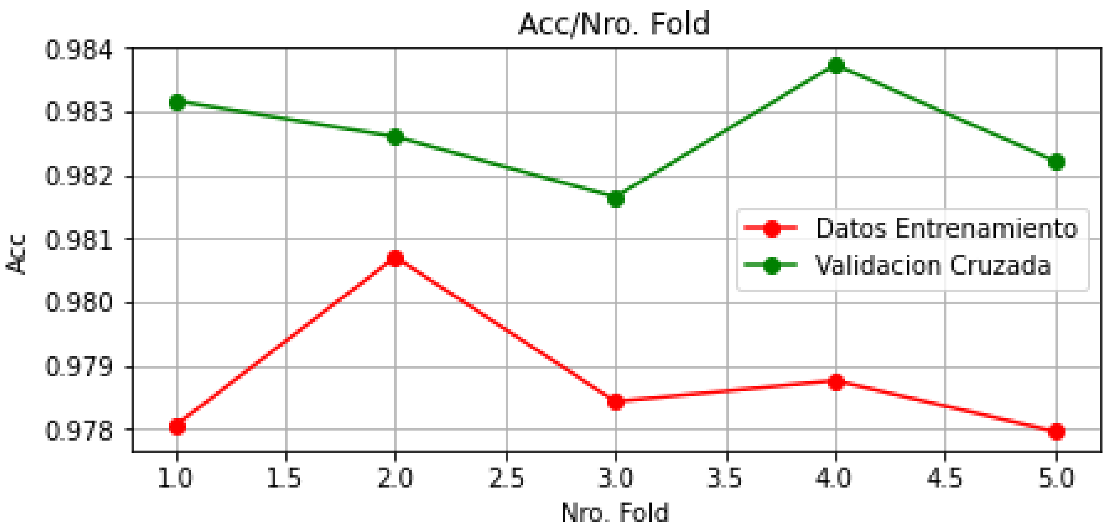
1.3 Ajustar el clasificador SVM en el Conjunto de Entrenamiento

[45]:
```
#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: [0.97805109 0.98070009 0.97842952 0.97875964
0.97795544]
Average score: 0.98
```

[44]:
```python
#Validación cruzada (gráfico y datos)
graficar_Accu_scores(class_svm,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[44]:
```
array([0.97805109, 0.98070009, 0.97842952, 0.97875964, 0.97795544])
```



[98]:
```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)
```

[98]:
```
SVC(probability=True, random_state=123)
```

[35]:
```python
# Predecir los resultados
y_pred_svm = class_svm.predict(X_test)
```

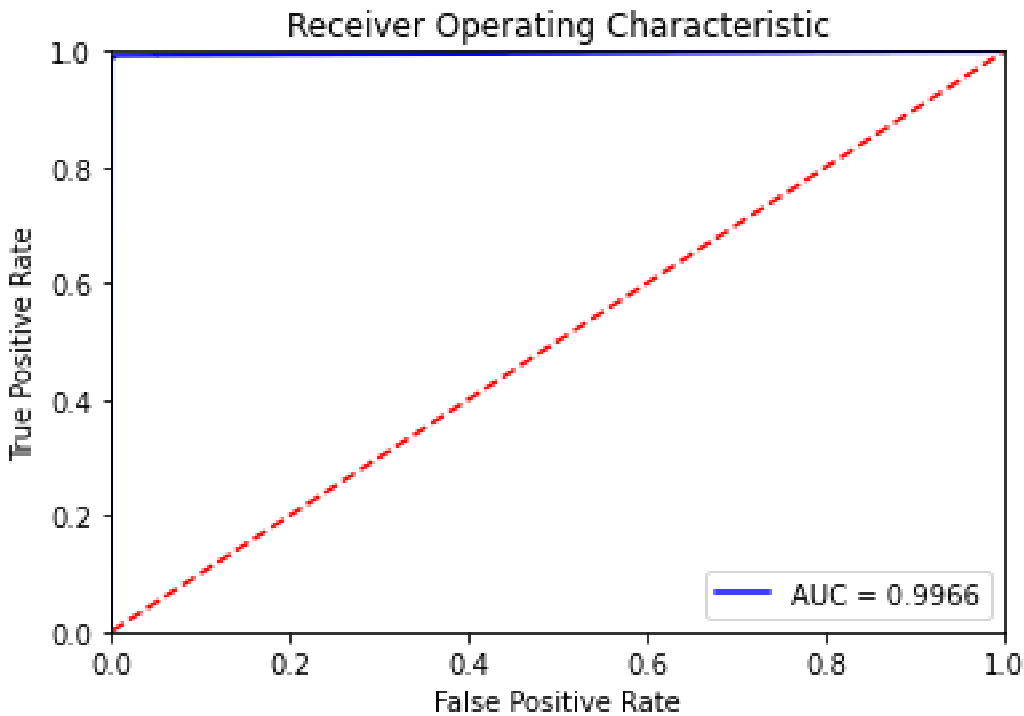
[37]:
```python
# Matriz de confusión
from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_svm # display
```

[37]:
```
array([[19249,     0],
       [  269,  6906]], dtype=int64)
```

[100]:
```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)
```

```
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

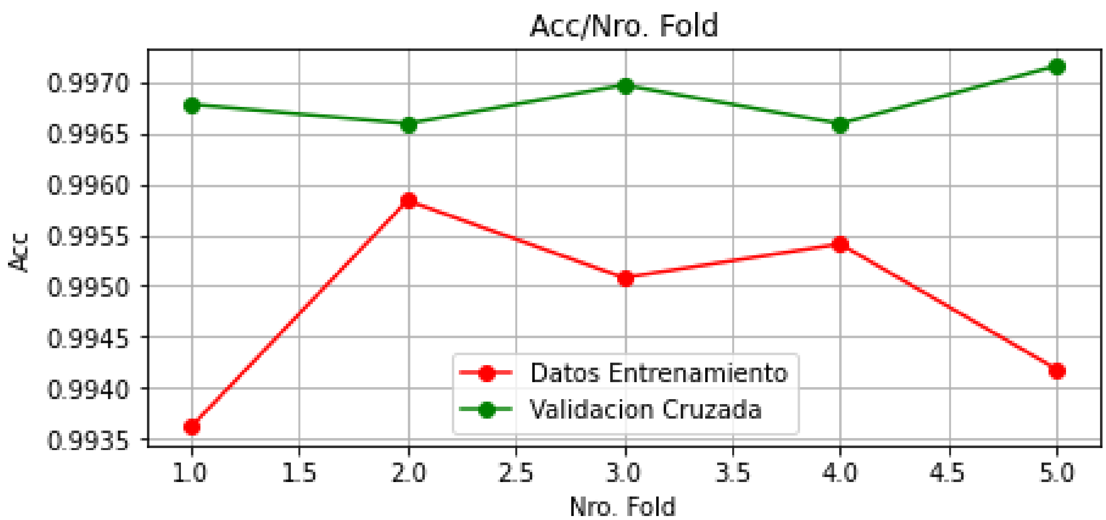
[79]:
```
#validación cruzada (datos)
from sklearn.model_selection import KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB
kf =KFold(n_splits=5, shuffle=True, random_state=42)
score = cross_val_score(GaussianNB(), X_train, y_train, cv= kf,␣
 ↪scoring="accuracy")
print(f'Scores for each fold are: {score}')
```

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

```
Scores for each fold are: [0.993614   0.99583728 0.99508042 0.99541133
0.99418137]
Average score: 0.9948
```

[66]:
```
#Validación cruzada
graficar_Accu_scores(class_nb,X_train,y_train,X_test,y_test,nparts=5,jobs=2)
```

[66]:
```
array([0.993614  , 0.99583728, 0.99508042, 0.99541133, 0.99418137])
```



[59]:
```
from sklearn.naive_bayes import GaussianNB
class_nb = GaussianNB()
class_nb.fit(X_train, y_train)
```

[59]:
```
GaussianNB()
```

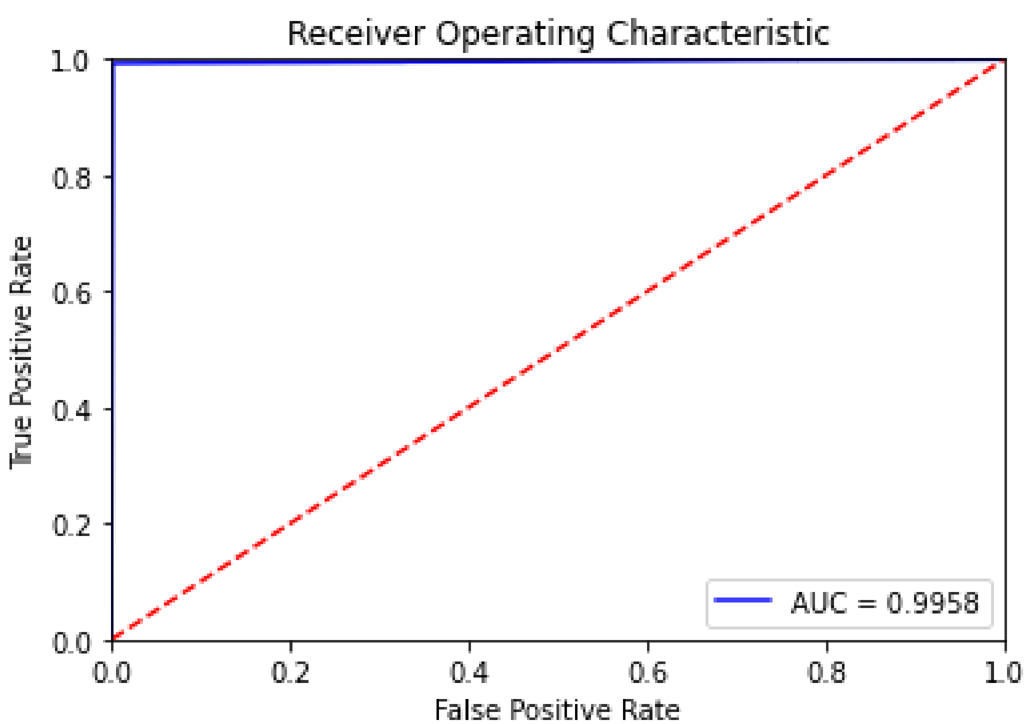
[60]:
```
# Predecir los resultados
y_pred_nb  = class_nb.predict(X_test)
```

[61]:
```
# Matriz de confusión
from sklearn.metrics import confusion_matrix
cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_nb # display
```

[61]:
```
array([[  150, 19099],
       [    0,  7175]], dtype=int64)
```

[76]:
```
#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

[96]:
```
#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_10964/2640115640.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)
```

[97]:
```
print(f'Scores for each fold are: {Accuracy}')
print(f'Average score: {"{:.4f}".format(Accuracy.mean())}')
```

```
Scores for each fold are: [0.9975875  1.         0.98907286 0.97738779
0.98259139]
Average score: 0.9893
```

[51]:
```
import keras
from keras.models import Sequential
```

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

[52]:
```python
#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
10570/10570 [==============================] - 13s 1ms/step - loss: 0.1031 -
accuracy: 0.9705
Epoch 2/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0718 -
accuracy: 0.9822
Epoch 3/100
10570/10570 [==============================] - 15s 1ms/step - loss: 0.0678 -
accuracy: 0.9837
Epoch 4/100
10570/10570 [==============================] - 15s 1ms/step - loss: 0.0651 -
accuracy: 0.9844
Epoch 5/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0644 -
accuracy: 0.9846
Epoch 6/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0636 -
accuracy: 0.9847
Epoch 7/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0626 -
accuracy: 0.9850
Epoch 8/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0625 -
accuracy: 0.9850
Epoch 9/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0625 -
accuracy: 0.9850
Epoch 10/100
```

```
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0607 -
accuracy: 0.9854
Epoch 11/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0597 -
accuracy: 0.9857
Epoch 12/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0586 -
accuracy: 0.9858
Epoch 13/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0592 -
accuracy: 0.9859
Epoch 14/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0581 -
accuracy: 0.9861
Epoch 15/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0575 -
accuracy: 0.9864
Epoch 16/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0570 -
accuracy: 0.9865
Epoch 17/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0568 -
accuracy: 0.9863
Epoch 18/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0568 -
accuracy: 0.9865
Epoch 19/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0561 -
accuracy: 0.9865
Epoch 20/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0567 -
accuracy: 0.9865
Epoch 21/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0551 -
accuracy: 0.9867 0s - l
Epoch 22/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0549 -
accuracy: 0.9869
Epoch 23/100
10570/10570 [==============================] - 14s 1ms/step - loss: 0.0535 -
accuracy: 0.9871
Epoch 24/100
10570/10570 [==============================] - 16s 2ms/step - loss: 0.0544 -
accuracy: 0.9870
Epoch 25/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0545 -
accuracy: 0.9870
Epoch 26/100
```

```
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0537 -
accuracy: 0.9872
Epoch 27/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0539 -
accuracy: 0.9871
Epoch 28/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0537 -
accuracy: 0.9872
Epoch 29/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0536 -
accuracy: 0.9872
Epoch 30/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0529 -
accuracy: 0.9873
Epoch 31/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0525 -
accuracy: 0.9874
Epoch 32/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0519 -
accuracy: 0.9876
Epoch 33/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0516 -
accuracy: 0.9877
Epoch 34/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0508 -
accuracy: 0.9879
Epoch 35/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0514 -
accuracy: 0.9877
Epoch 36/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0513 -
accuracy: 0.9877 0s - loss:
Epoch 37/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0509 -
accuracy: 0.9879
Epoch 38/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0508 -
accuracy: 0.9879
Epoch 39/100
10570/10570 [==============================] - 16s 2ms/step - loss: 0.0502 -
accuracy: 0.9881
Epoch 40/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0503 -
accuracy: 0.9881
Epoch 41/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0501 -
accuracy: 0.9882
Epoch 42/100
```

```
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0494 -
accuracy: 0.9882
Epoch 43/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0500 -
accuracy: 0.9881
Epoch 44/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0495 -
accuracy: 0.9882
Epoch 45/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0491 -
accuracy: 0.9885
Epoch 46/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0492 -
accuracy: 0.9883
Epoch 47/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0489 -
accuracy: 0.9884
Epoch 48/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0478 -
accuracy: 0.9886 0s -
Epoch 49/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0481 -
accuracy: 0.9887
Epoch 50/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0478 -
accuracy: 0.9884
Epoch 51/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0486 -
accuracy: 0.9883
Epoch 52/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0479 -
accuracy: 0.9886
Epoch 53/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0479 -
accuracy: 0.9886
Epoch 54/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0470 -
accuracy: 0.9888
Epoch 55/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0472 -
accuracy: 0.9886
Epoch 56/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0474 -
accuracy: 0.9888
Epoch 57/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0465 -
accuracy: 0.9893
Epoch 58/100
```

```
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0467 -
accuracy: 0.9889
Epoch 59/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0463 -
accuracy: 0.9890
Epoch 60/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0465 -
accuracy: 0.9890
Epoch 61/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0459 -
accuracy: 0.9889
Epoch 62/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0458 -
accuracy: 0.9890
Epoch 63/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0458 -
accuracy: 0.9891
Epoch 64/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0455 -
accuracy: 0.9894
Epoch 65/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0459 -
accuracy: 0.9892
Epoch 66/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0451 -
accuracy: 0.9891
Epoch 67/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0454 -
accuracy: 0.9891
Epoch 68/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0448 -
accuracy: 0.9893
Epoch 69/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0449 -
accuracy: 0.9894
Epoch 70/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0448 -
accuracy: 0.9894 0s - loss: 0.0
Epoch 71/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0456 -
accuracy: 0.9892
Epoch 72/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0451 -
accuracy: 0.9893
Epoch 73/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0453 -
accuracy: 0.9891
Epoch 74/100
```

```
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0442 -
accuracy: 0.9896
Epoch 75/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0447 -
accuracy: 0.9893
Epoch 76/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0449 -
accuracy: 0.9894
Epoch 77/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0449 -
accuracy: 0.9892
Epoch 78/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0447 -
accuracy: 0.9893
Epoch 79/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0443 -
accuracy: 0.9894
Epoch 80/100
10570/10570 [==============================] - 15s 1ms/step - loss: 0.0454 -
accuracy: 0.9891
Epoch 81/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0439 -
accuracy: 0.9895
Epoch 82/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0443 -
accuracy: 0.9894
Epoch 83/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0445 -
accuracy: 0.9894
Epoch 84/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0433 -
accuracy: 0.9897
Epoch 85/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0435 -
accuracy: 0.9896
Epoch 86/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0426 -
accuracy: 0.9897
Epoch 87/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0436 -
accuracy: 0.9897
Epoch 88/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0439 -
accuracy: 0.9896
Epoch 89/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0433 -
accuracy: 0.9896
Epoch 90/100
```

```
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0431 -
accuracy: 0.9897
Epoch 91/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0424 -
accuracy: 0.9899
Epoch 92/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0433 -
accuracy: 0.9898
Epoch 93/100
10570/10570 [==============================] - 11s 1ms/step - loss: 0.0431 -
accuracy: 0.9899
Epoch 94/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0423 -
accuracy: 0.9900
Epoch 95/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0426 -
accuracy: 0.9899
Epoch 96/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0425 -
accuracy: 0.9898
Epoch 97/100
10570/10570 [==============================] - 13s 1ms/step - loss: 0.0425 -
accuracy: 0.9900
Epoch 98/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0424 -
accuracy: 0.9900
Epoch 99/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0429 -
accuracy: 0.9898
Epoch 100/100
10570/10570 [==============================] - 12s 1ms/step - loss: 0.0423 -
accuracy: 0.9900
```

```
[52]: <keras.callbacks.History at 0x27c3380bc40>
```

[54]:

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

```
826/826 - 1s - loss: 0.0303 - accuracy: 0.9943 - 700ms/epoch - 847us/step

Test Accuracy: 0.9943233132362366
```

[55]:

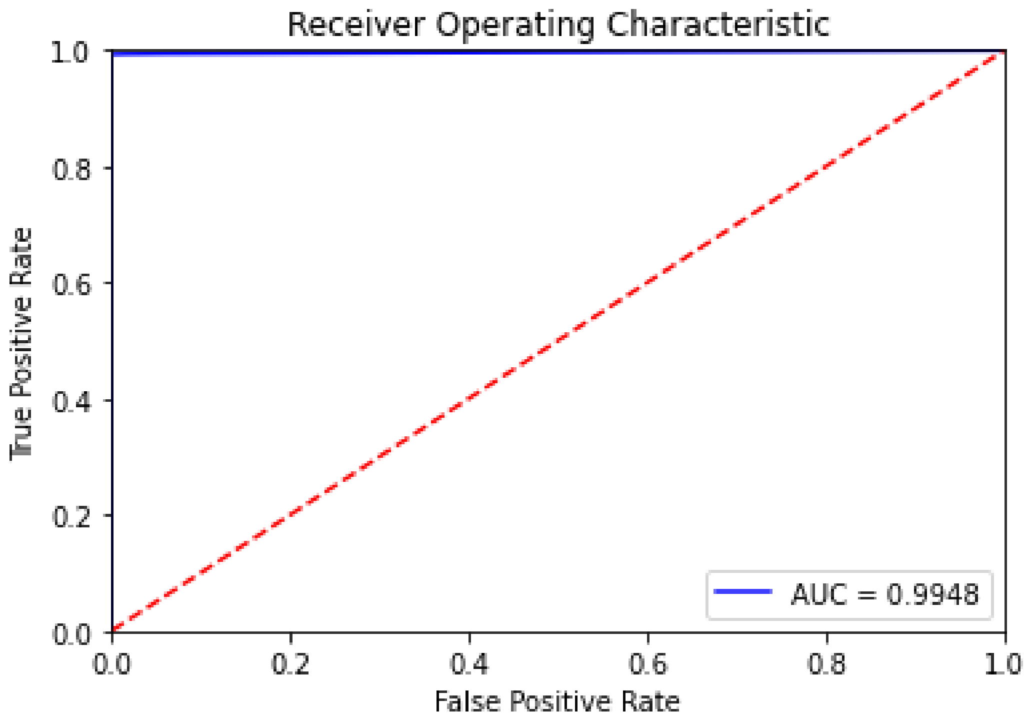
```python
# Evaluar el modelo y calcular predicciones finales
# Predicción de los resultados con el Conjunto de Testing
y_pred_rn  = class_RN.predict(X_test)
y_pred_rn = (y_pred_rn>0.5)
```

[58]:
```python
#Elaborar una matriz de confusión
from sklearn.metrics import confusion_matrix
cm_rn = confusion_matrix(y_test, y_pred_rn)
cm_rn
```

[58]:
```
array([[19249,     0],
       [  150,  7025]], dtype=int64)
```

[80]:
```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(fpr, tpr, '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 = 0.9948
0.0
0.2
0.4
0.6
0.8
1.0