Tarea 3: SVM y Random Forests (parte práctica)

Fecha de entrega: 11 de octubre, 23:59

Profesor: Pablo Estévez V.
Auxiliar: Ignacio Reyes J.
Semestre: Primavera 2019

Parte experimental Two Moons

1.

2.

3. **Sol:**

   En esto experimento se obtuvo resultados con 2 tipos de kernels.

   Probamos con diferentes parametros C para lo kernel lineal.

   Para el Gaussiano, mantenendo C=1, se varió γ entre $\{0.1 , 1 , 10, 100\}$ y después, con γ=1, se vario C.

   Para el kernel lineal, comparando los resultados de AUC en validación se escojeu C=10 porque tenia mayor porcentage de area bajo de la curva ROC, 94.27 %.

   Para el kernel Gaussiano, mantenendo C, se escojeu γ=1, porque tenia mayor porcentage de AUC de la curva ROC, 98.3 %.

   Mantenendo γ, se escojeu C=1 para el mejor parametro por la misma razon, AUC= 98.4 %.

   Entonces llegamos a las siguientes conclusiones:

   C menor significa un modelo sencillo, mayor error en el entrenamiento,suavidad en la frontera de decisión.

   C mayor significa un modelo complejo, poca suavidad de la frontera dedecisión, riesgo desobreajuste.

   γ menor significa mayor solape entre Gaussianas, suavidad en la frontera de decisión.

   γ mayor significa todos los puntos tienden a ser ortogonales unos a otros, sobreajuste.

   Para el kernel lineal se hace una separación por hyperplanos rectos, mientras el Gauss intenta ajustar mejor a los datos, es más flexible.

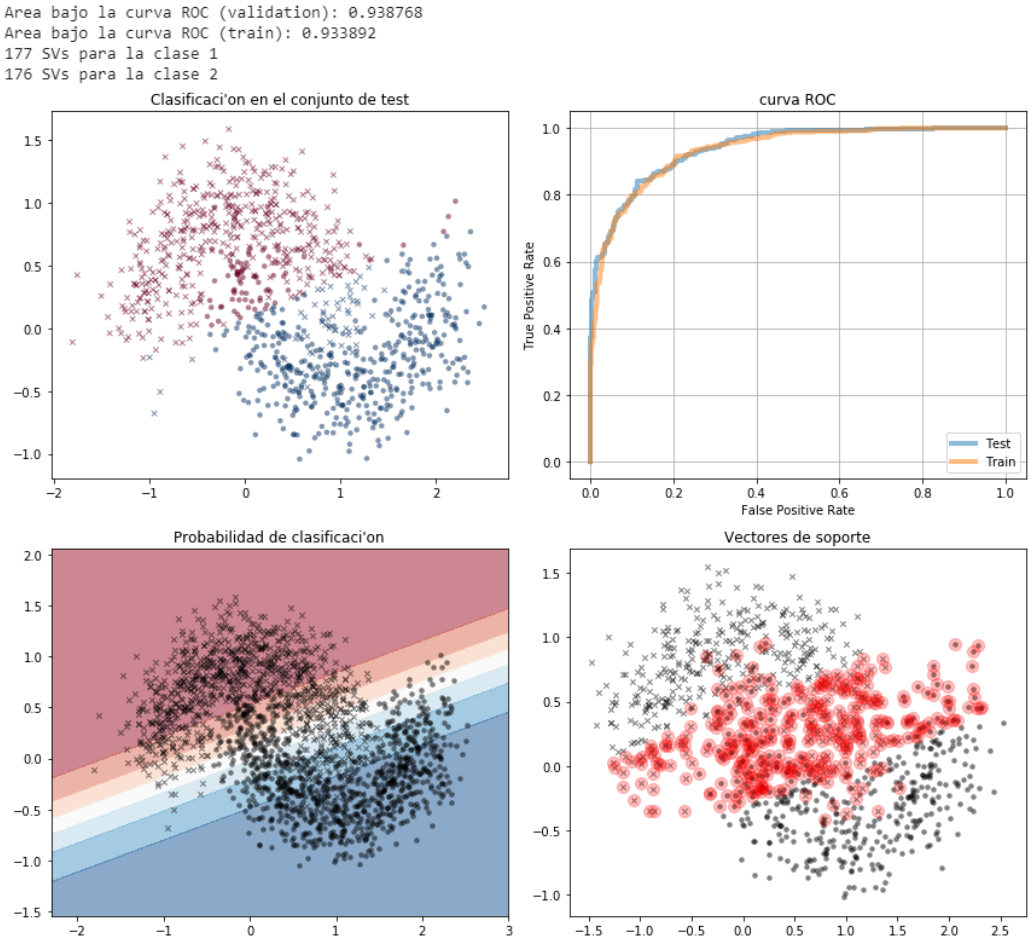


Figura 1: Resultados para C=1 y γ=1 para kernel lineal

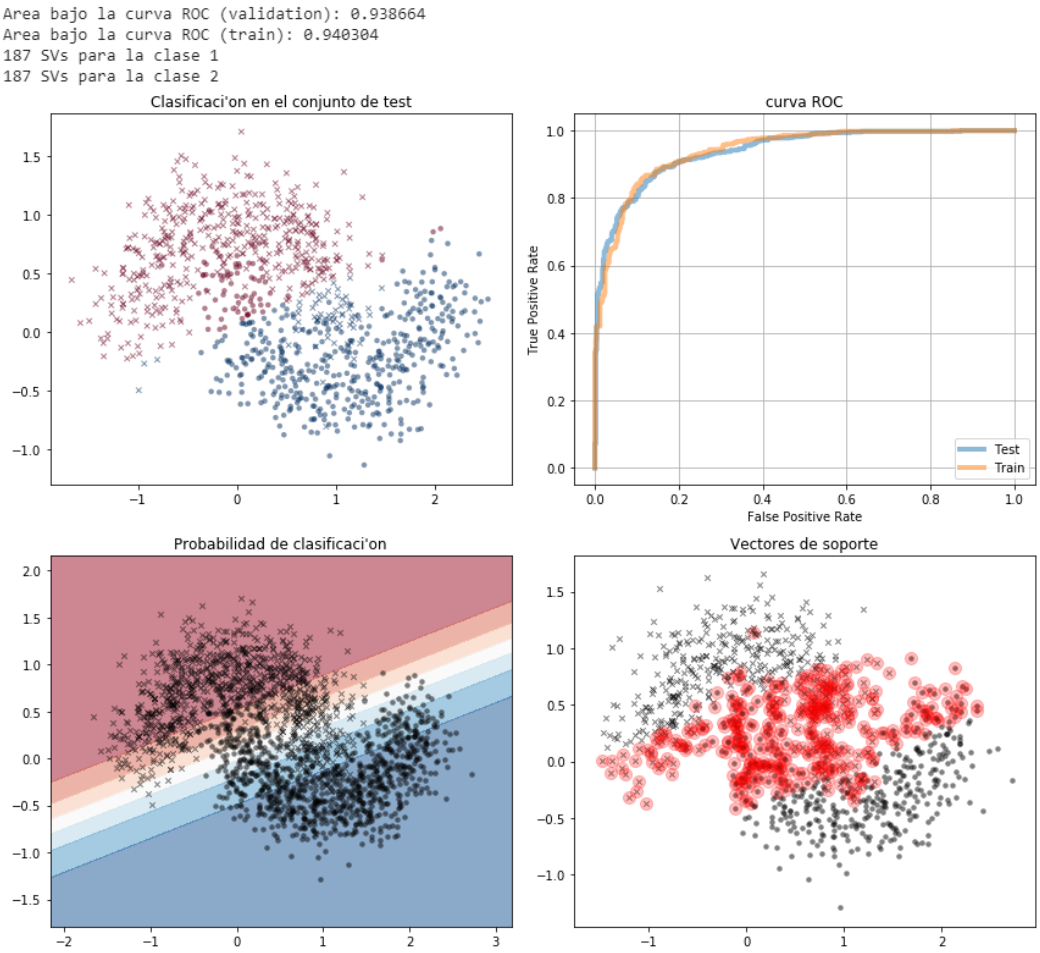


Figura 2: Resultados para C=0.1 y γ=1 para kernel lineal

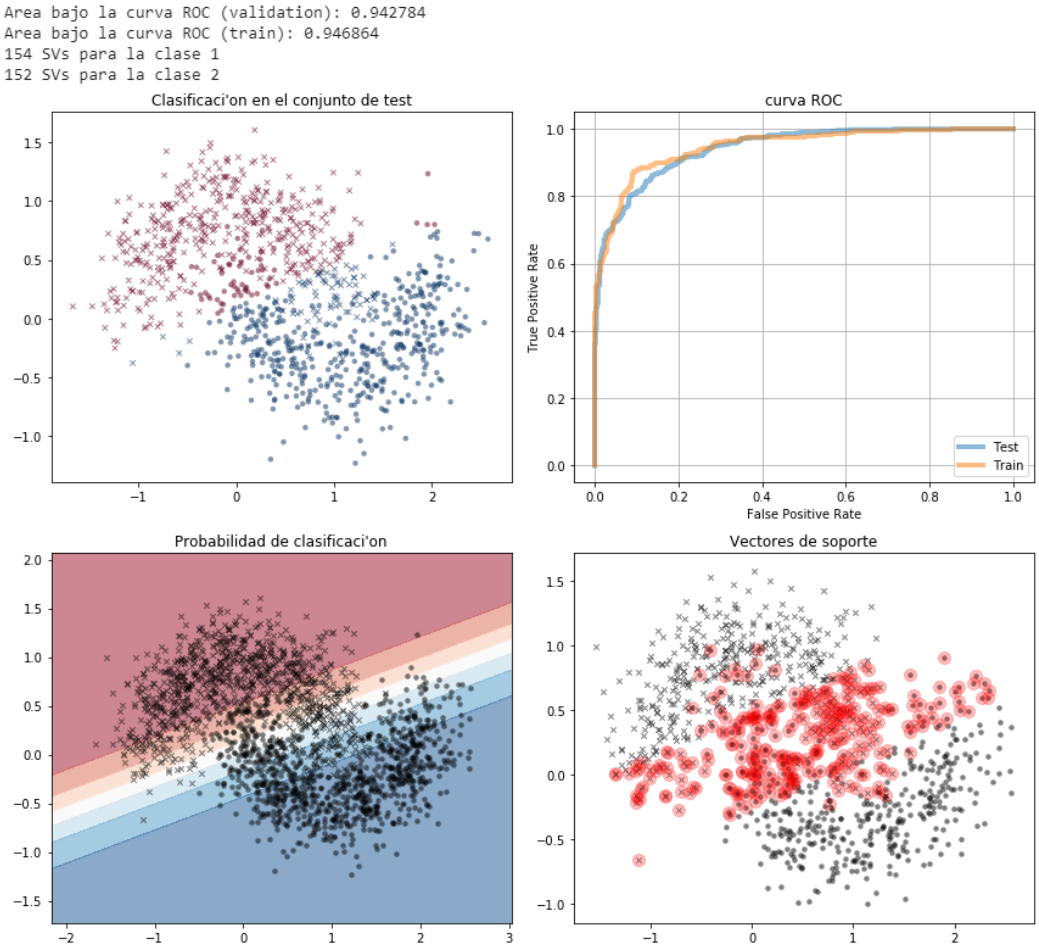


Figura 3: Resultados para C=10 y γ=1 para kernel lineal

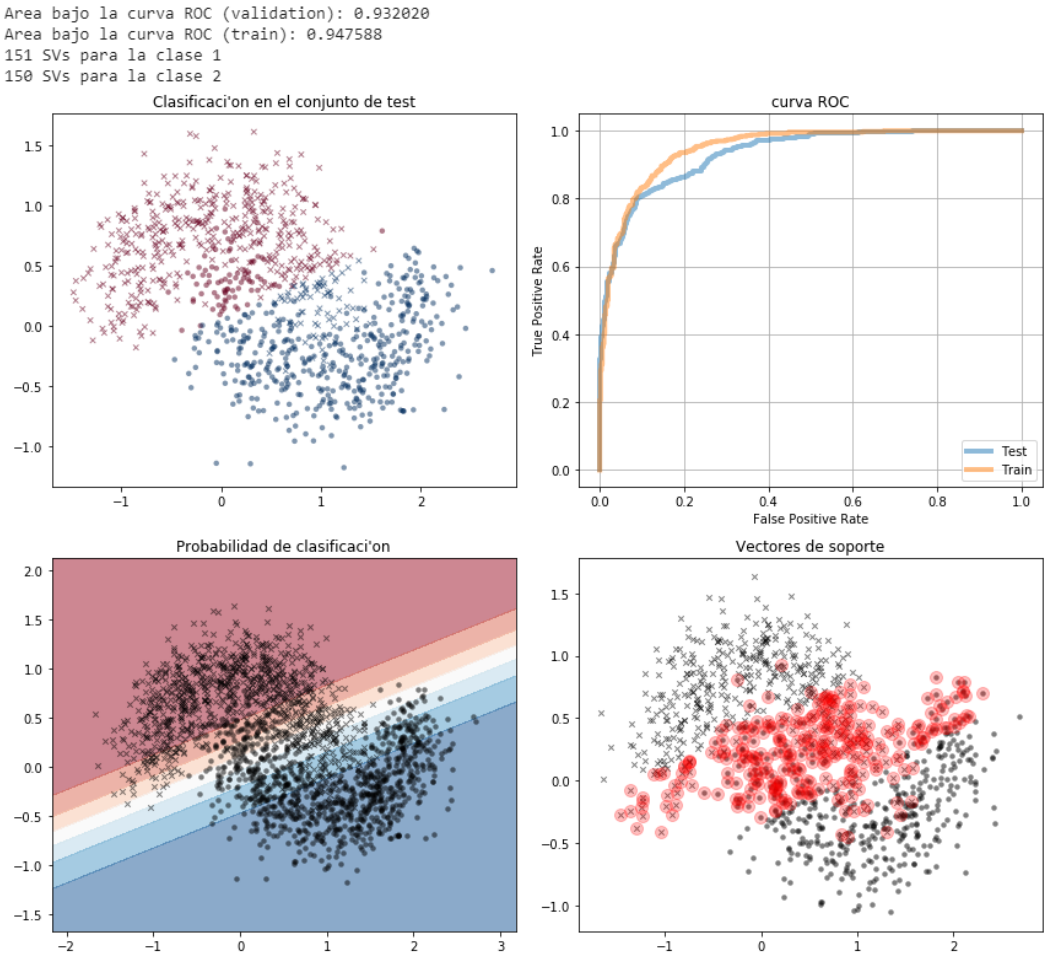


Figura 4: Resultados para C=100 y γ=1 para kernel lineal

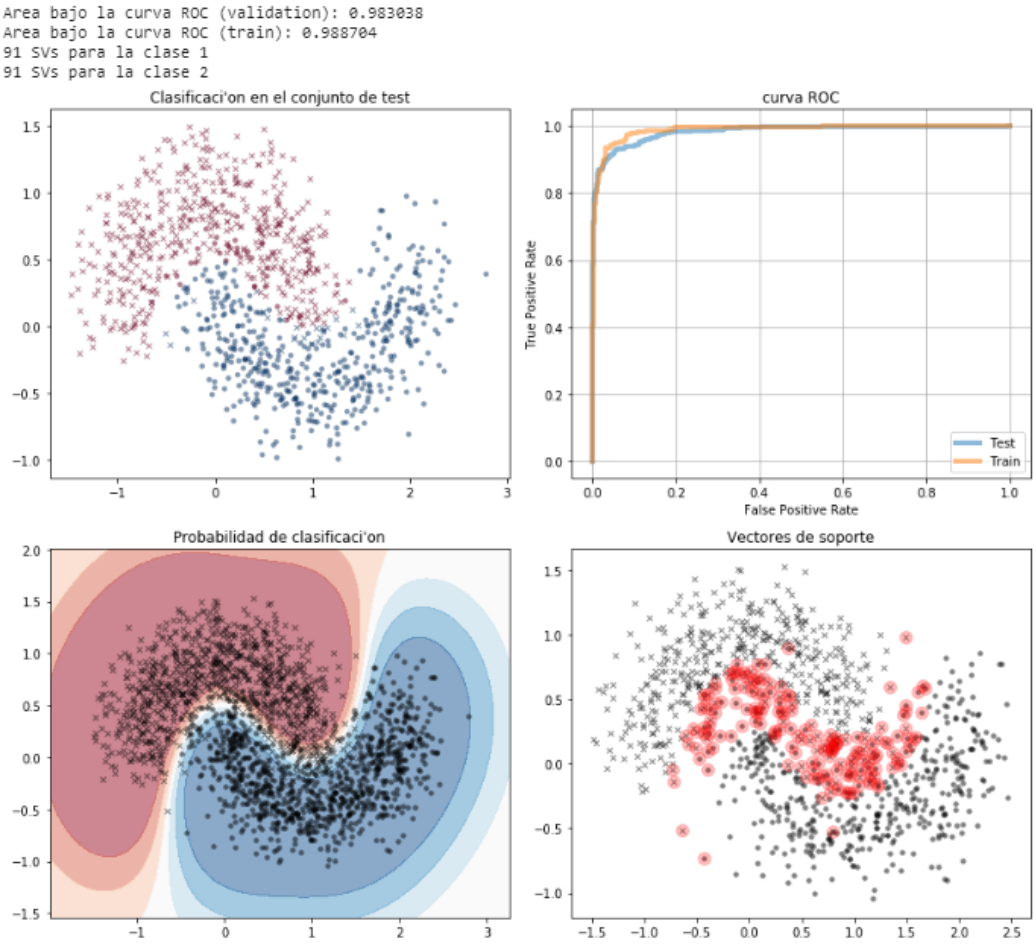


Figura 5: Resultados para C=1 y γ=1 para kernel Gaussiano

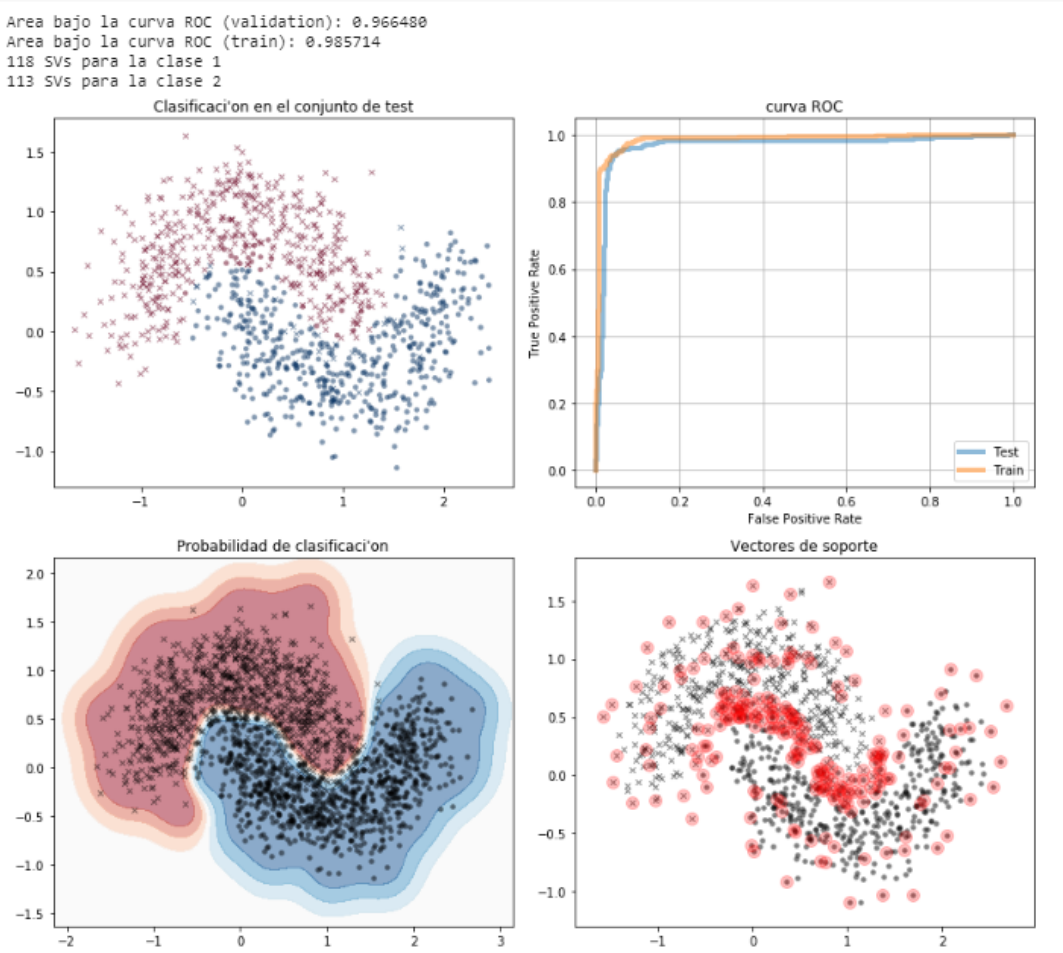


Figura 6: Resultados para C=1 y γ=10 para kernel Gaussiano

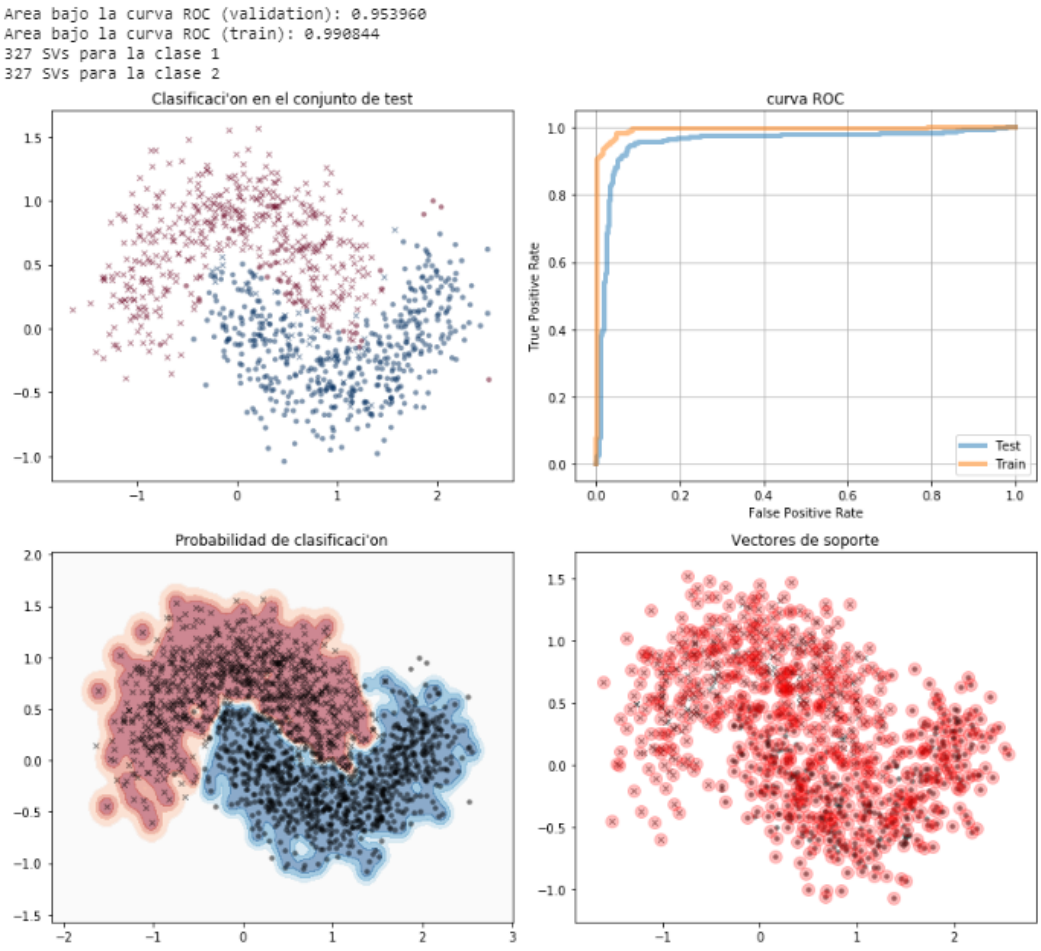


Figura 7: Resultados para C=1 y γ=100 para kernel Gaussiano

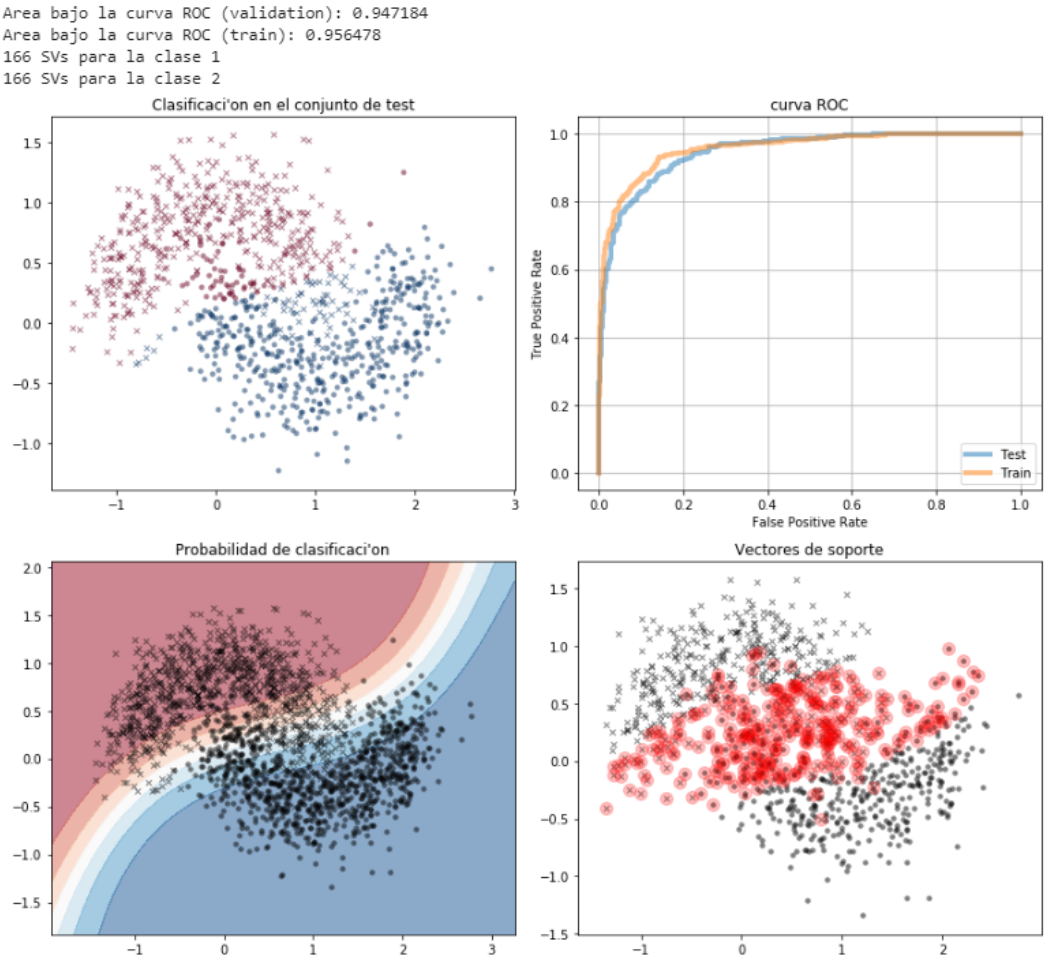


Figura 8: Resultados para C=1 y γ=0.1 para kernel Gaussiano

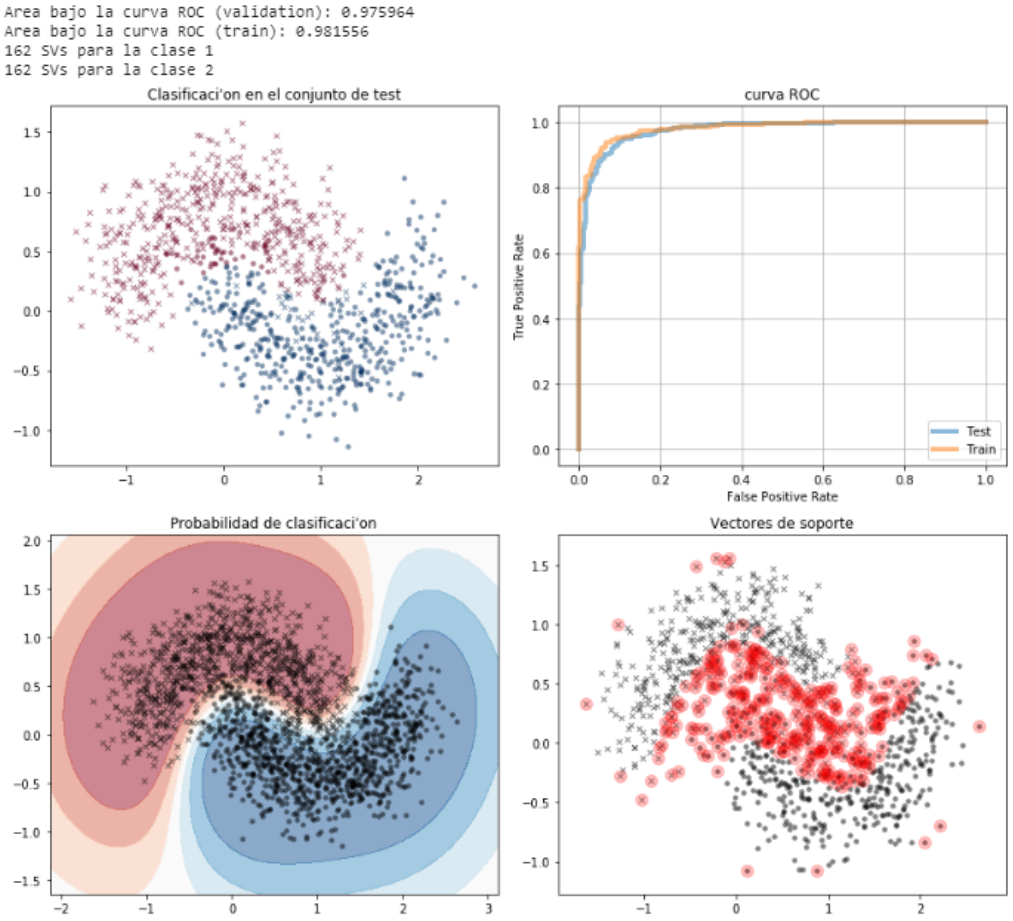


Figura 9: Resultados para C=0.1 y γ=1 para kernel Gaussiano

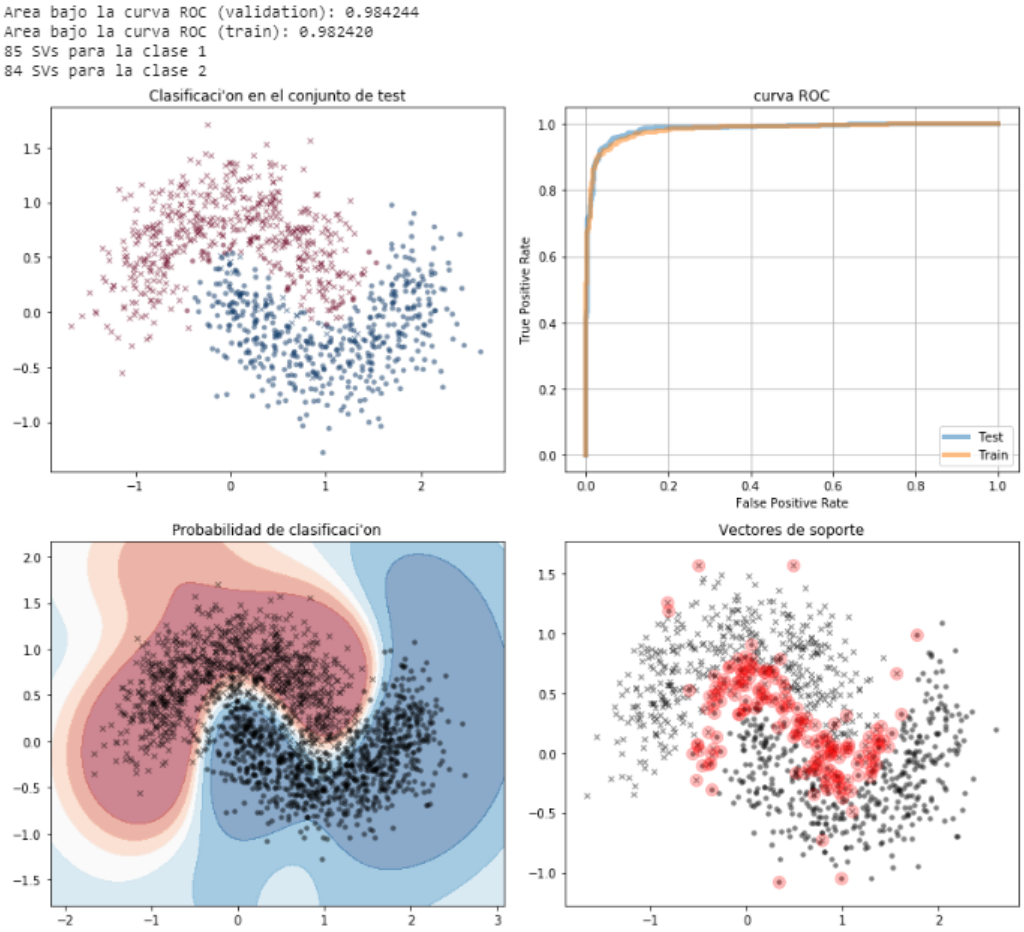


Figura 10: Resultados para C=10 y γ=1 para kernel Gaussiano

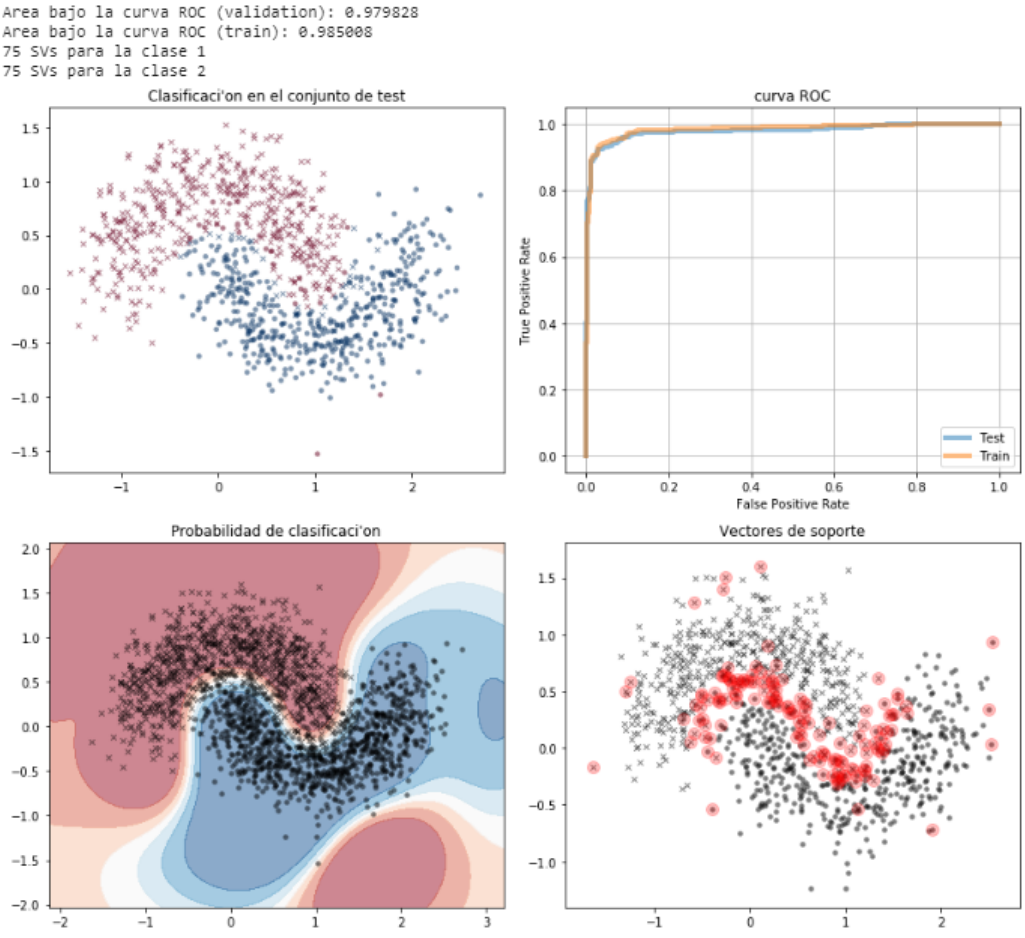


Figura 11: Resultados para C=100 y γ=1 para kernel Gaussiano

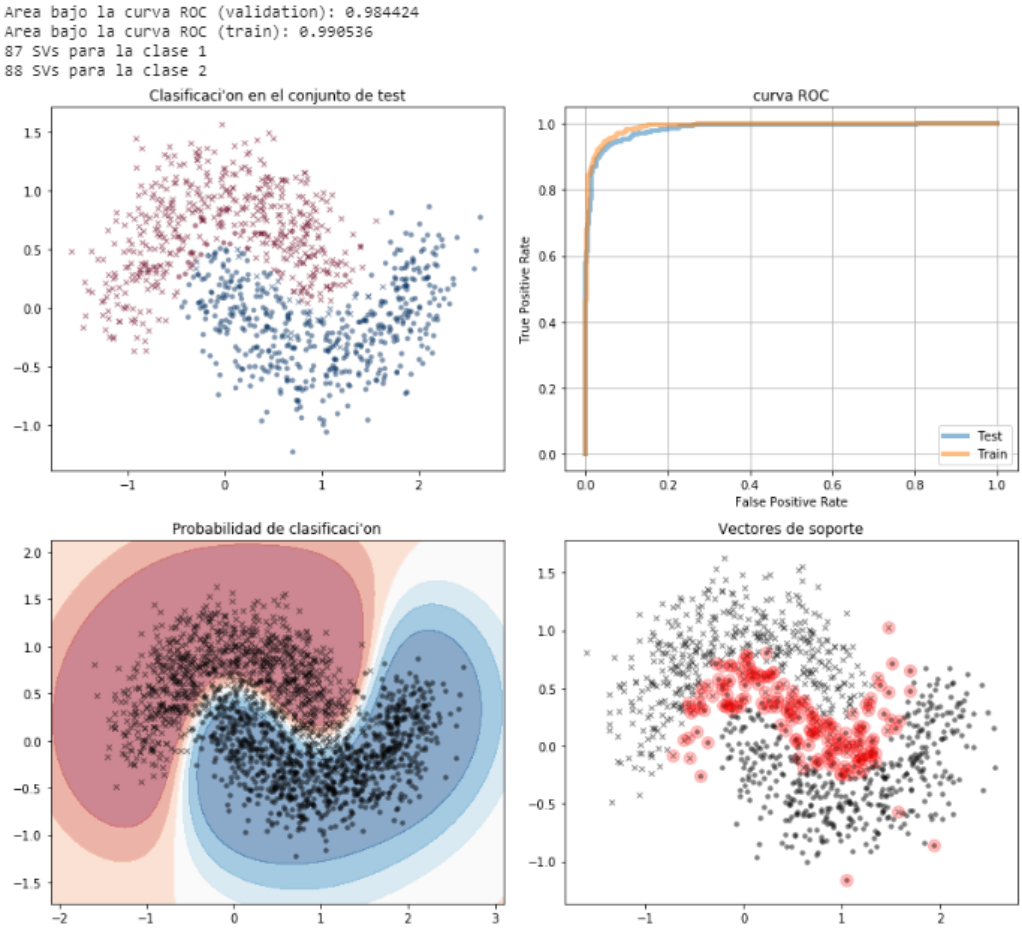


Figura 12: Resultados para C=1 y γ=1 para kernel Gaussiano

4. **Sol:** Al construir un árbol pequeño se obtendrá un modelo con baja varianza y alto sesgo. Normalmente, al incrementar la complejidad del modelo, se verá una reducción en el error de predicción debido a un sesgo más bajo en el modelo. En un punto el modelo será muy complejo y se producirá un sobre-ajuste del modelo el cual empezará a sufrir de varianza alta.

   El modelo óptimo debería mantener un balance entre estos dos tipos de errores. A esto se le conoce como "trade-off" (equilibrio) entre errores de sesgo y varianza.

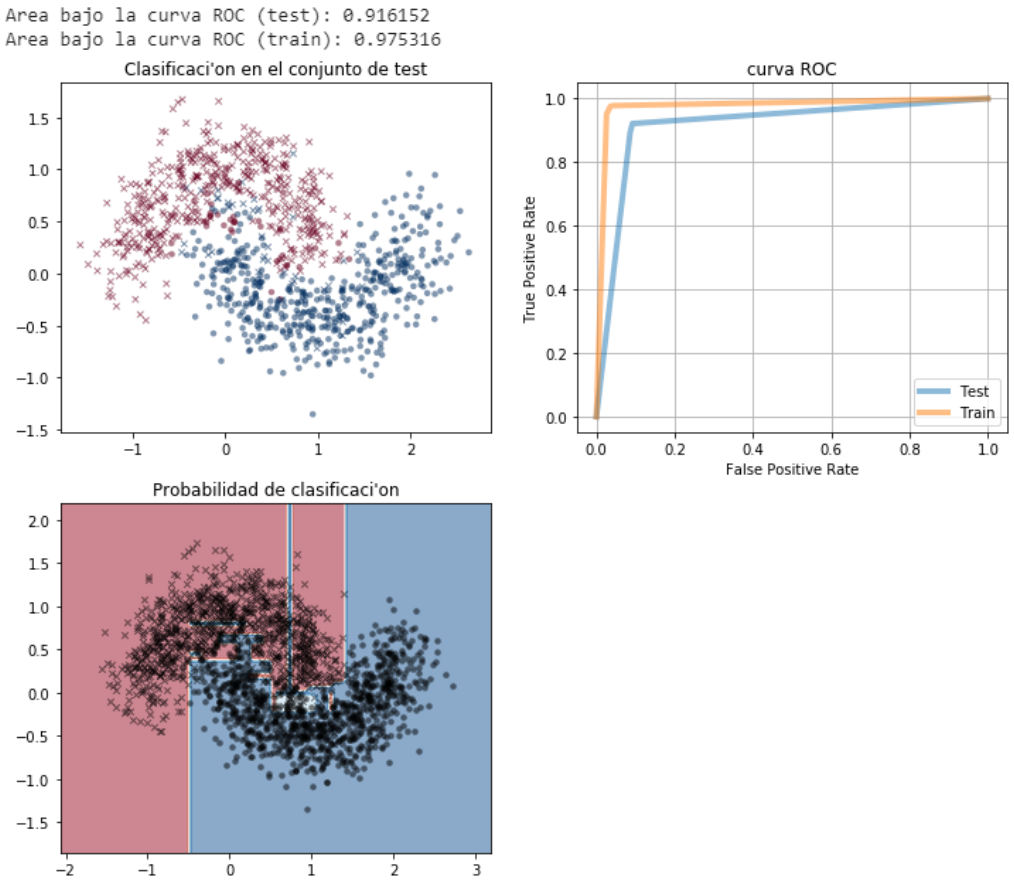


Figura 13: Resultados para nº estimadores =1 profundidad máxima 10

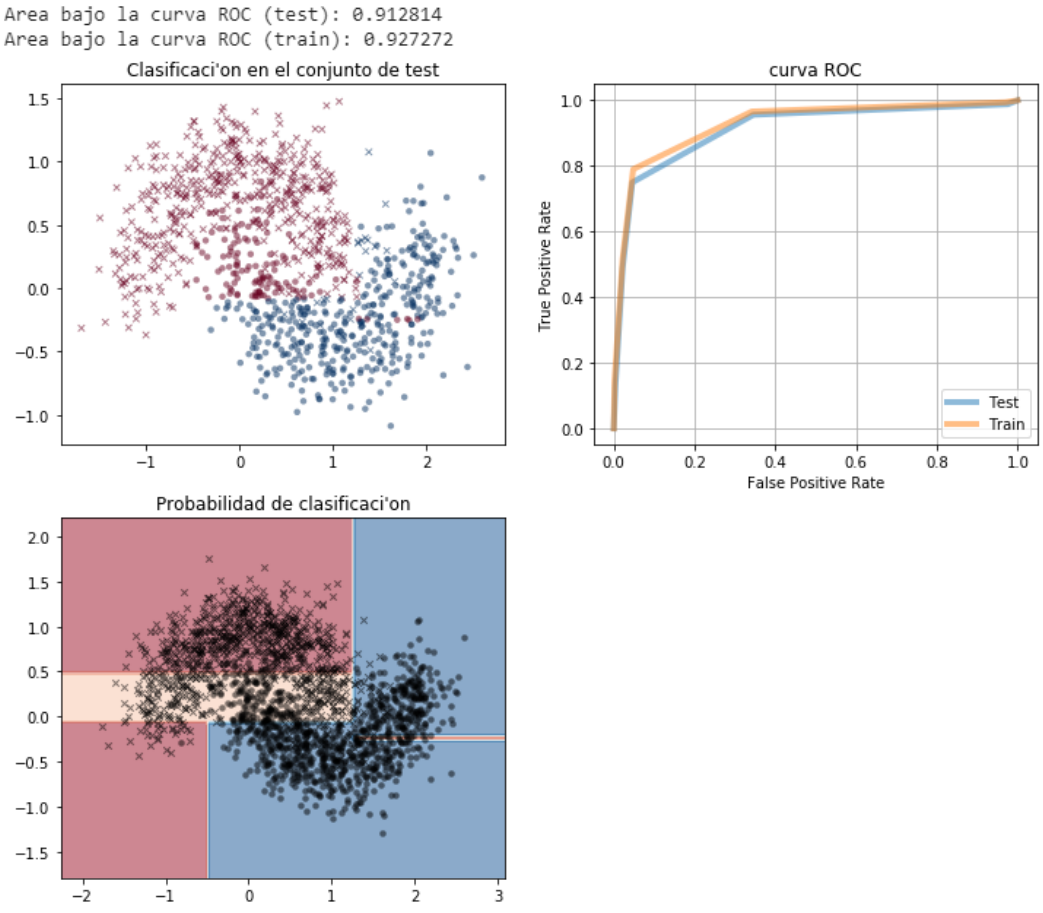


Figura 14: Resultados para nº estimadores =1 profundidad máxima 3

Sol: Observando los resultados con profundidad máxima 7, es posible afirmar que existe un overfitting, debido a la forma de los gráficos y por comparación de la diferencia entre la AUC en entrenamiento y test.

Por otro lado, con profundidad máxima 3, se nota underfitting debido a la forma de la curva ROC y respectivos gráficos con AUC de entrenamiento y test bajos en comparación.

5. **Sol:** Entre todos los valores y resultados, el mejor modelo es 50 árboles con una profundidad máxima de 10, ya que tiene el AUC más alto en los resultados de test.

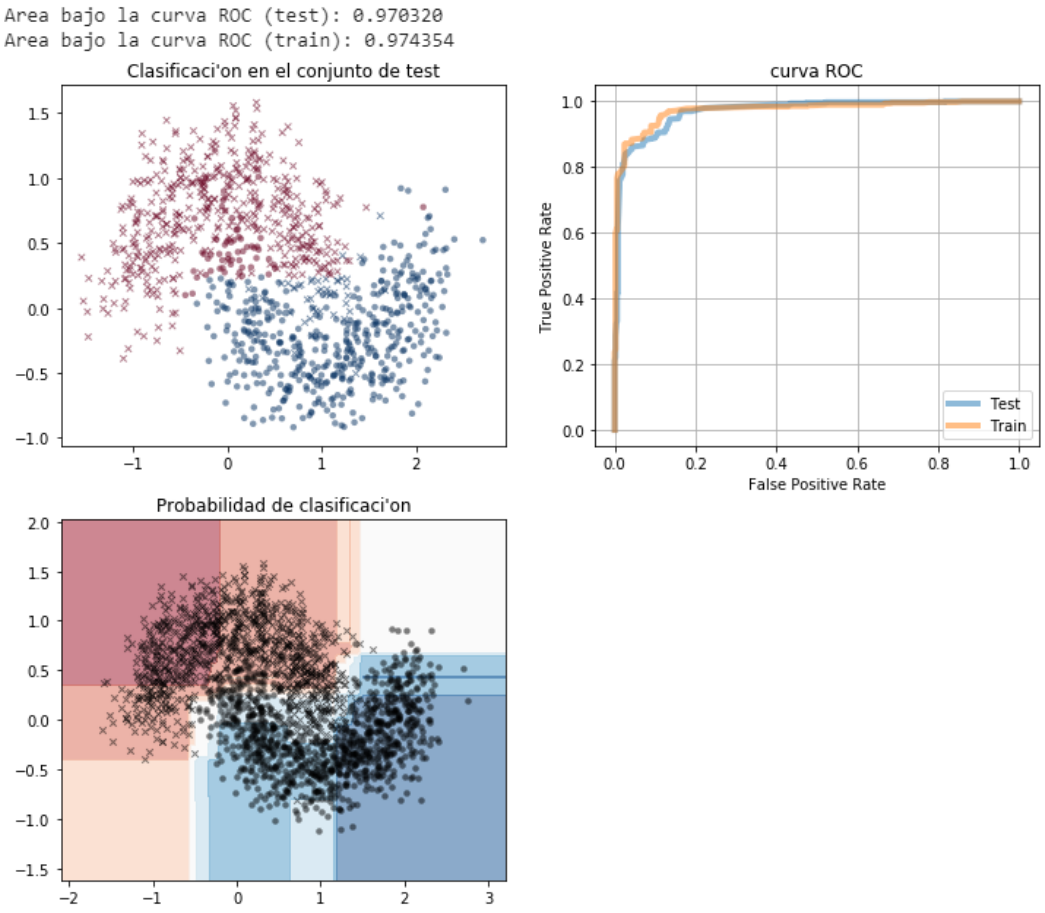


Figura 15: Resultados para nº arboles =7 profundidad máxima 3

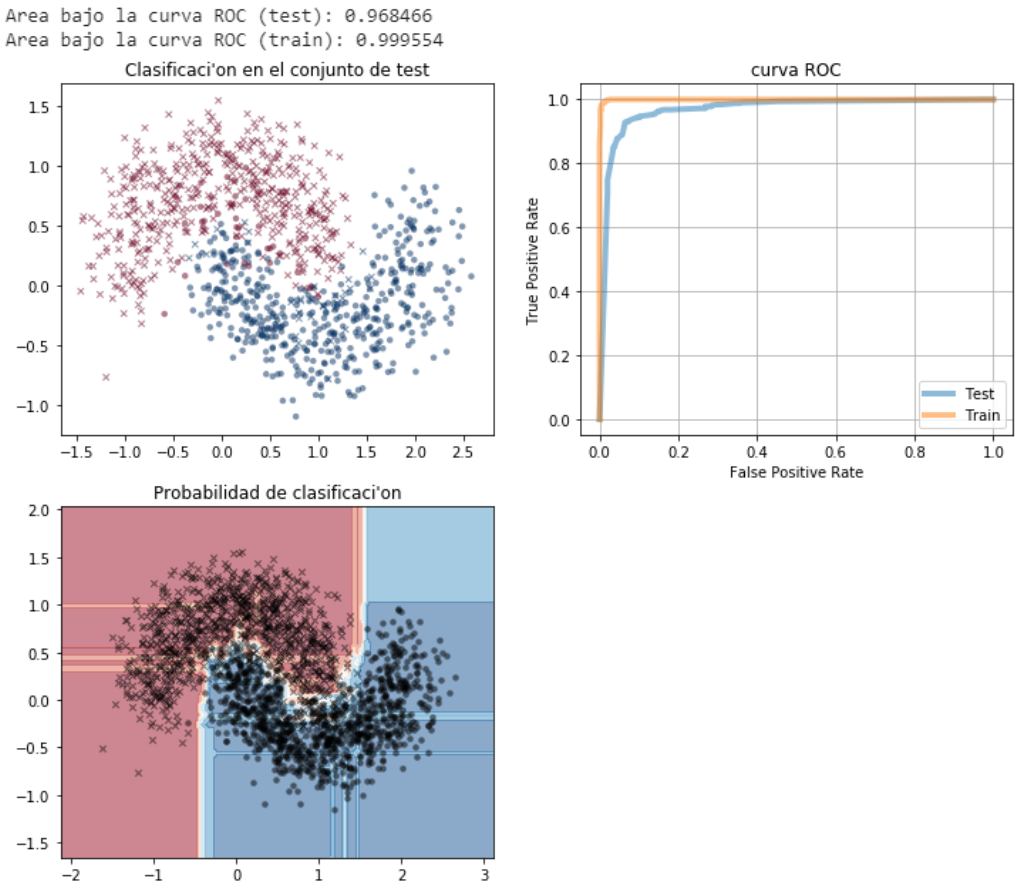


Figura 16: Resultados para nº arboles =7 profundidad máxima 10

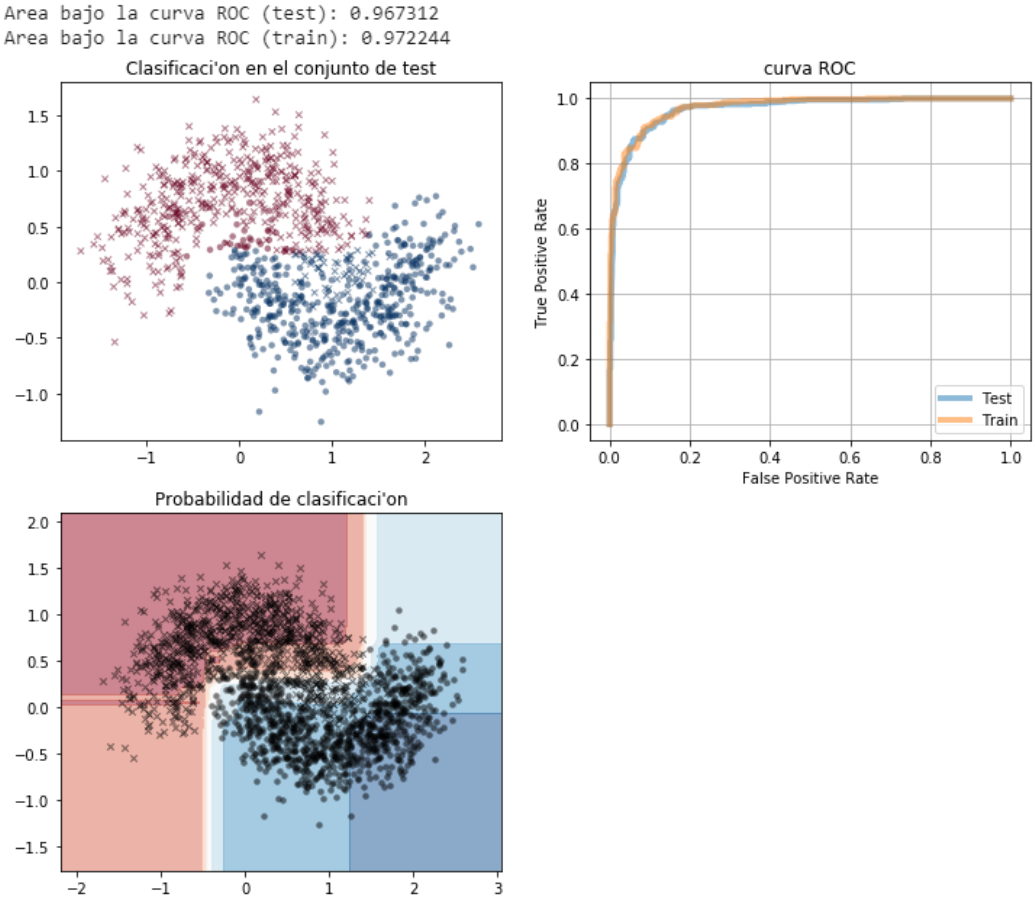


Figura 17: Resultados para nº arboles =50 profundidad máxima 3

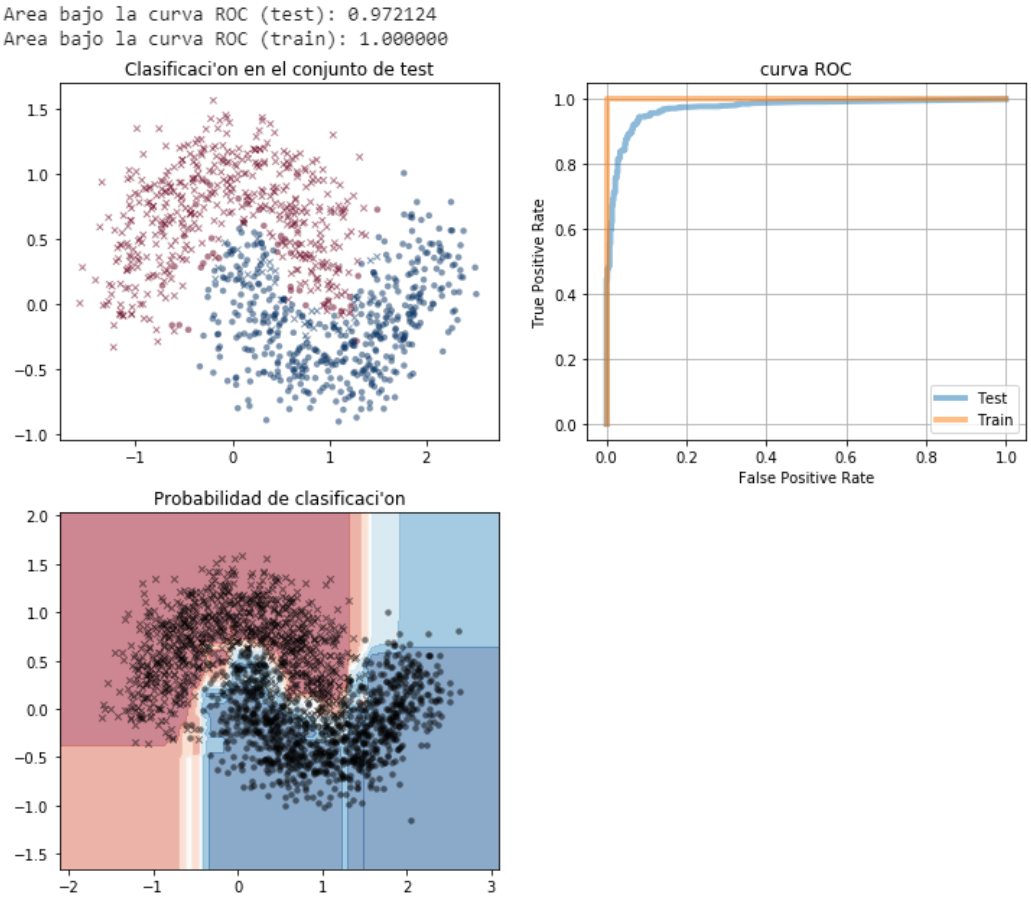


Figura 18: Resultados para nº arboles =50 profundidad máxima 10

Covertype dataset

1. **Sol:** La base de datos debe estar balanceada, es decir, cada clase debe tener la misma cantidad de ejemplos. Observando los resultados del numero de ejemplos por clase, se verifica que todos son distintos.

```
Ejemplos por clase (entrenamiento)
[148288. 198310.  25028.   1923.   6645.  12157.  14357.]
Ejemplos por clase (validacion)
[63552. 84991. 10726.   824.  2848.  5210.  6153.]
```

Figura 19: Resultados Desbalance

2. **Sol:** La cantidad de ejemplos clasificados correctamente de la base de datos balanceada disminuye en comparación con la desbalanceada. Per aumenta los aciertos en la clase minoritaria. Le recomendaria la base de datos balanceada para que no favoresca solo las clases mayoritarias y aumente la predicción sobre todas las clases. Esto se puede comprobar por el valor de recall promedio por clase que aumentó substancialmente.

```
Promedio recall por clase (validacion) 0.499
Promedio recall por clase (training) 0.508
Numero de ejemplos en validacion clasificados correctamente: 131568
```

Figura 20: Resultados Base de datos balanceada

```
Promedio recall por clase (validacion) 0.805
Promedio recall por clase (training) 0.817
Numero de ejemplos en validacion clasificados correctamente: 114309
```

Figura 21: Resultados Base de datos desbalanceada

3. **Sol:** El mejor es el modelo con profundidad máxima de 30 porque tiene mayor recall, 88.9 %.

```
Promedio recall por clase (validacion) 0.499
Promedio recall por clase (training) 0.508
Numero de ejemplos en validacion clasificados correctamente: 131568
```

Figura 22: Resultados Recall para modelo con profundidad máxima 10

```
Promedio recall por clase (validacion) 0.889
Promedio recall por clase (training) 0.992
Numero de ejemplos en validacion clasificados correctamente: 164571
```

Figura 23: Resultados Recall para modelo con profundidad máxima 30

```
Promedio recall por clase (validacion) 0.889
Promedio recall por clase (training) 0.992
Numero de ejemplos en validacion clasificados correctamente: 164571
```

Figura 24: Resultados Recall para modelo con profundidad máxima 50

4. **Sol:** Las características más importantes son las 3 primeras porque tienen un mayor valor porcentual de importancia. Elevation con 25.8 %, Horizontal distance to roadways con 10.5 % y Horizontal distance to fire points con 8.7 % de importancia. Sumando los valores porcentuales hasta el 80 % de importancia, necesitamos 11 características, 20.37 % das características totais.

```
Caracteristicas ordenadas por importancia (RF)
        0.248 Elevation
        0.116 Horizontal distance to roadways
        0.107 Horizontal distance to fire points
        0.059 Horizontal distance to hydrology
        0.055 Vertical distance to hydrology
        0.047 Aspect
        0.042 Hillshade noon
        0.040 Hillshade 9am
        0.039 Hillshade 3pm
        0.031 Slope
        0.031 Cache la Poudre Wilderness Area
        0.016 Soil type 22
        0.014 Soil type 10
        0.013 Soil type 4
        0.012 Comanche Peak Wilderness Area
        0.012 Rawah Wilderness Area
        0.011 Soil type 39
        0.011 Soil type 38
        0.011 Soil type 23
        0.010 Soil type 2
        0.009 Neota Wilderness Area
        0.008 Soil type 12
        0.006 Soil type 29
        0.006 Soil type 40
        0.005 Soil type 32
        0.004 Soil type 24
        0.004 Soil type 33
        0.004 Soil type 13
        0.004 Soil type 30
        0.004 Soil type 31
        0.003 Soil type 11
        0.003 Soil type 3
        0.002 Soil type 6
        0.002 Soil type 20
        0.002 Soil type 17
        0.002 Soil type 35
        0.001 Soil type 19
        0.001 Soil type 1
        0.001 Soil type 16
        0.001 Soil type 21
        0.001 Soil type 27
        0.001 Soil type 5
        0.001 Soil type 34
        0.001 Soil type 37
        0.000 Soil type 14
        0.000 Soil type 18
        0.000 Soil type 26
        0.000 Soil type 28
        0.000 Soil type 25
        0.000 Soil type 9
        0.000 Soil type 36
        0.000 Soil type 8
        0.000 Soil type 7
        0.000 Soil type 15
```

Figura 25: Resultados características con sus respectivas porcentages de importancia

Programación

1. Construya un subconjunto del dataset Covertype con 1000 muestras por cada clase tomadas al azar.

```
# ================ P3.1 ============================
def process_dataset(images, labels, selected_class):

    shuffled_indexes = np.random.permutation(len(labels))
    images = images[shuffled_indexes]
    labels = labels[shuffled_indexes]

    indexes = np.where(labels == selected_class)[0]
    selected_size = len(indexes)

    np.random.shuffle(indexes)
    images_subset = images[indexes,:]
    labels_subset = labels[indexes]

    return images_subset, labels_subset


DD = dataset.data
TT = dataset.target

print('')

X = np.zeros((1, DD.shape[1]))
Y = np.zeros(1)
print(Y)
print('')

for selected_class in range(1,8):
  examples, labels = process_dataset(DD, TT, selected_class)
  examples_subset = examples[:1000,:]
  labels_subset = labels[:1000]

  X = np.concatenate( (X, examples_subset), axis=0)
  Y = np.concatenate( (Y, labels_subset), axis=0)


X_balanced = np.delete(X, 0, 0)
Y_balanced = np.delete(Y, 0, 0)
```

Figura 26: Subconjunto del dataset Covertype con 1000 muestras por cada clase tomadas al azar

2. Particione dicho subconjunto en un conjunto de entrenamiento y validación, con proporciones 70 % y 30 % respectivamente. Utilice la función train test split de sklearn con el parámetro stratify.

```
# ================ P3.2 ============================
X = X_balanced
Y = Y_balanced
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7, stratify=Y)
print('done')
```

Figura 27: subconjunto en un conjunto de entrenamiento y validación, con proporciones 70 % y 30 % respectivamente

3. Normalice cada característica restando la media y dividiendo por la desviación estándar. Puede utilizar el StandardScaler de sklearn. Recuerde computar la media y desviación estándar utilizando sólo datos del conjunto de entrenamiento.

```
# ================ P3.3 ============================


X_train_ini = np.zeros((X_train.shape[0], 1))


X_test_ini = np.zeros((X_test.shape[0], 1))



for i in range(X_train.shape[1]):
  feature = X_train[:,i]
  m = feature.mean()
  ds = feature.std()

  if(ds == 0):
     ds = ds_reserva
  ds_reserva = ds

  feature_norm = (feature - m) / ds

  e = np.expand_dims(feature_norm, axis=0)
  feature_norm_T = np.transpose(e)


  X_train_ini = np.concatenate( (X_train_ini, feature_norm_T), axis=1)

  feature_test = X_test[:,i]
  feature_test_norm = (feature_test - m) / ds
  e = np.expand_dims(feature_test_norm, axis=0)
  feature_test_norm_T = np.transpose(e)
  X_test_ini = np.concatenate( (X_test_ini, feature_test_norm_T), axis=1)


X_train_norm = np.delete(X_train_ini, 0, 1)
X_test_norm = np.delete(X_test_ini, 0, 1)



print(X_train_norm)

print(X_test_norm)


print('\n\nfinished\n')
```

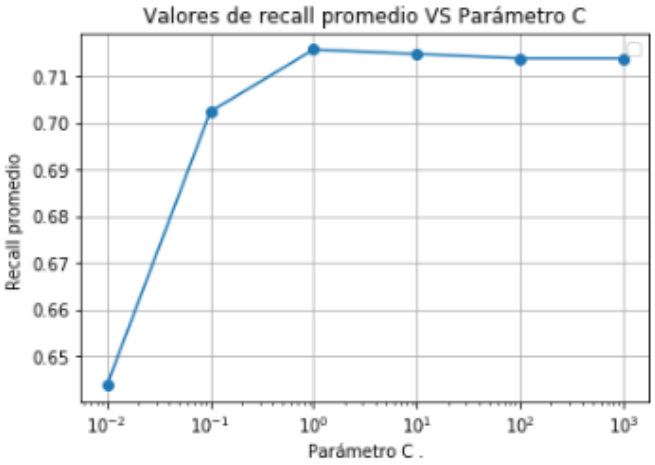
Figura 28: Normalización

4. Normalice cada característica restando la media y dividiendo por la desviación estándar. Puede utilizar el StandardScaler de sklearn. Recuerde computar la media y desviación estándar utilizando sólo datos del conjunto de entrenamiento.

```
# ================ P3.4 Lineal ============================
C_range = np.logspace(-2, 10, 13)

C_range = C_range[:6]
gamma_range = C_range

target_names = [
    'Spruce-Fir',
    'Lodgepole Pine',
    'Ponderosa Pine',
    'Cottonwood/Willow',
    'Aspen',
    'Douglas-fir',
    'Krummholz'
]

X = []
Y = []

for i in range(6):
  c = C_range[i]
  print('Valor de C: ',c)
  classifier = SVC(C=c, kernel='linear', gamma=1.0, probability=True, decision_function_shape='ovr')
  classifier.fit(X_train_norm, Y_train)
  Y_pred = classifier.predict(X_test_norm)

  cm = confusion_matrix(Y_test, Y_pred)
  recall_val = mean_recall(cm)
  X.append(c)
  Y.append(recall_val)

print('Valores de X: ',X)
print('Valores de Y: ',Y)

fig, ax = plt.subplots()
line2, = ax.semilogx(X, Y, marker='o')
ax.legend()
plt.title('recall promedio VS C')
plt.xlabel('Parámetro C .')
ax.grid()
plt.ylabel('Recall promedio ')
plt.show()

plot_confusion_matrix(cm, target_names, normalized=False)
```

Figura 29: Normalización

5. .




```
/usr/local/lib/python3.6/dist-packages/matplotlib/cbook/__init__.py:424: M
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; us
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a
```

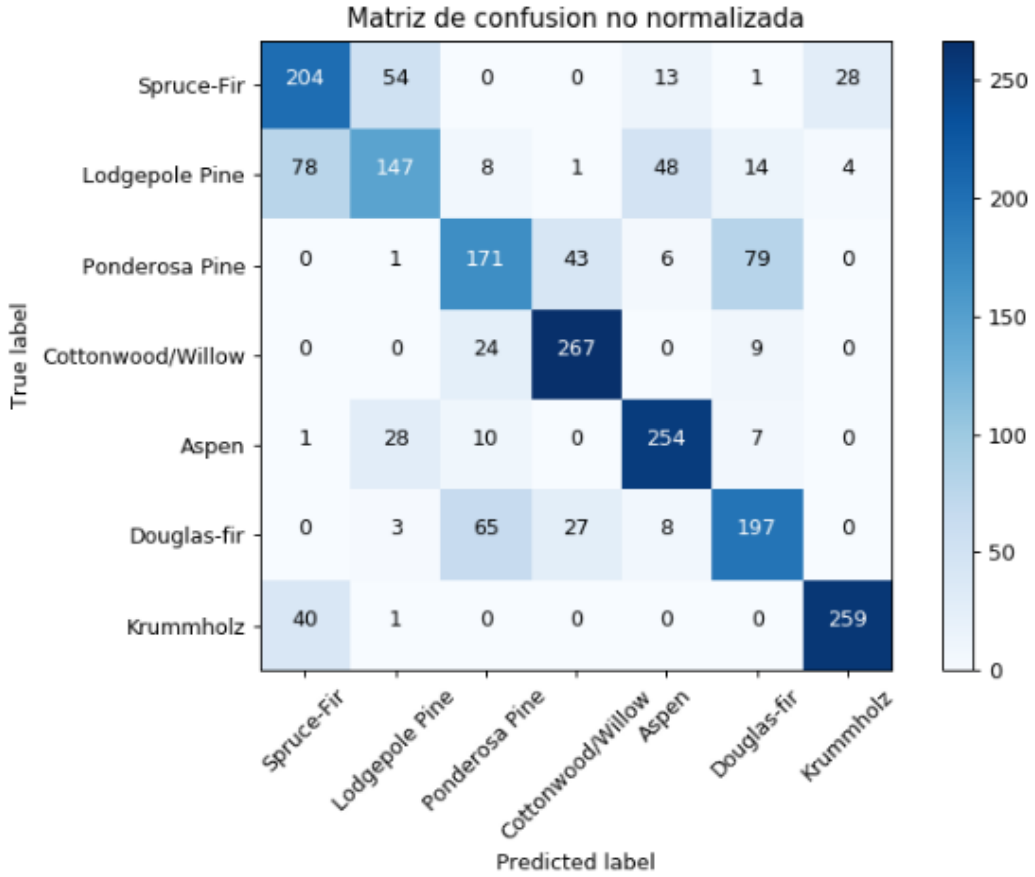


Figura 30: Normalización