Tarea 3 EL4106 SVM y Random Forests

Joaquín Zepeda

November 13, 2021

1 Parte Teórica

1. Imagine que está entrenando un modelo Random Forests. Dada una característica elegida para dividir el espacio ¿qué criterio se utiliza para colocar el umbral de clasificación? ¿Cómo se decide qué característica se utilizaría para dividir el conjunto de muestras en un nodo?

Uno de los criterios es tomar un umbral de clasificación binario, es decir dividir en 2 el espacio en cada nodo, con el fin de tener un arbol balanceado y un espacio bien particionado, a pesar de esto, muchas veces lo mejor es utilizar un umbral aleatorio, aunque esto no siempre es recomendable pues puede variar mucho los tamaños de los árboles.

2. En los modelos Random Forests cada árbol de decisión clasifica utilizando un subconjunto de las características disponibles. Explique por qué se hace esto.

La razón de esto es distribuir las muestras de forma de hacer un modelo más robusto y que cada uno de los arboles sea más aleatorio (permitiendo obtener un mejor modelo), es decir, que no cambien tanto los resultados cada vez que se ejecuta el algoritmo, este método da como resultado una amplia diversidad que permite que el modelo sea mejor que los arboles de decisión.

3. ¿Qué son los métodos de kernel? ¿Qué ventaja tiene llevar los puntos a un espacio distinto? ¿Es necesario conocer la función que mapea los puntos al nuevo espacio para aplicar los métodos de kernel?

Los métodos de kernel son una familia de algoritmos que llevan los puntos a un nuevo espacio distinto del original, permitiendo así encontrar la solución en este nuevo plano (en el cual generalmente es más simple encontrar la solución) y luego devolvernos al espacio original para encontrar la solución en el espacio original. Mo es necesario conocer esta función que mapea, pero si es necesario conocer el kernel (el cual puede ser lineal, gaussiano, sigmoide,etc). La ventaja de llevar los puntos a este nuevo espacio es que en este espacio determinar una solución al problema es más simple que en el espacio original, esto gracias a las propiedades de este plano, permite que el hiperplano solución sea lineal lo que permite que sea más sencillo determinar una solución.

2 Parte Práctica

2.1 Two Moons

A lo largo de la primera parte de esta tarea, se irán variando los valores de C y gamma, para determinar los que permiten obtener un mayor valor de AUC en validación, por lo que es importante entender que representan estos valores.

- Valor AUC: corresponde al área bajo la curva de la curva ROC, la función auc permite calcular esta área.
- Valor de C: parámetro de la función SVM, funciona como regularizador, permite controlar el error.
- Valor de gamma: parámetro de la función SVM, determina hasta dónde puede llegar la influencia de un solo ejemplo en el algoritmo.

A continuación se presenta el Dataset a utilizar para las clasificaciones, los cuales corresponden a un problema de clasificación binaria, este se presenta en la figura 1.

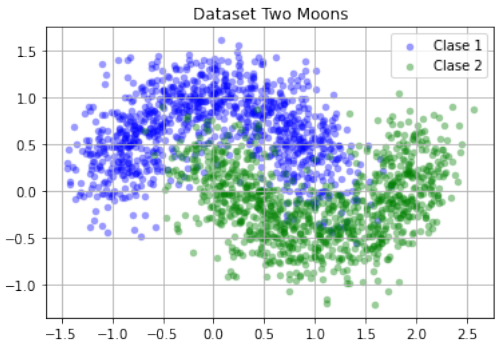


Figure 1: Conjunto de datos binario Two moons

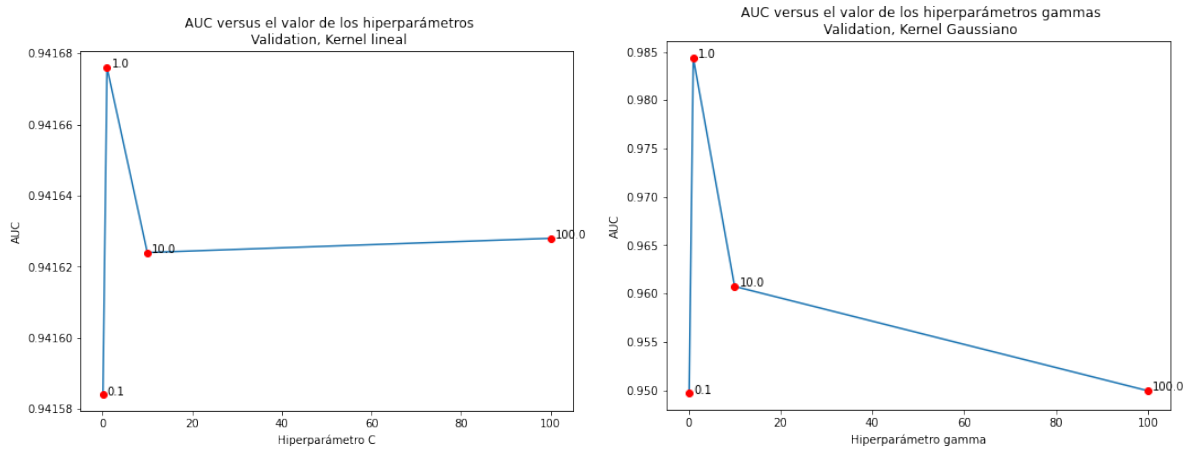
2.2 Clasificación Two Moons utilizando SVM (Support Vector Machines)

2.2.1 Determinando el valor de C que maximiza el valor de AUC en la validación para el kernel lineal

A partir de la figura 2a, se determina que el valor de C que maximiza el valor de AUC de validación corresponde a C= 1.0.

2.2.2 Determinando el valor de Gamma y C que maximiza el valor de AUC en la validación para el kernel gaussiano

Primero para determinar el valor de Gamma óptimo se realizan pruebas utilizando C = 1.0.



(a) AUC para cada valor C usando kernel lineal

(b) AUC para cada valor C usando kernel gaussiano

Figure 2

A partir de la figura 2b, se determina que el valor de gamma que maximiza el valor de AUC de valicación corresponde a $gamma = 1.0$. Utilizando este valor de gamma, se procede a determinar el valor óptimo de C.
A continuación, se prueban los diferentes valores de C los cuales se pueden observar en la figura 3, a partir de esta se puede determinar que el valor del hiperparametro C que maximiza el valor de AUC corresponde a $C = 1.0$.

La elección del kernel tiene un efecto directo en la forma los hiperplanos separadores, como lo dice su nombre, cuando se elije un kernel lineal, los hiperplanos separadores corresponden a rectas, esto se puede observar en la figura 4. EL hiperparametro C tiene un efecto de regularización de los errores, este hiperparametro regula el nivel de error que acepta el algoritmo, a menor C menor es el error y viceversa. Por otro lado el parámetro gamma decide cuanta curvatura se le desea dar a los hiperplanos, a mayor gamma mayor curvatura y viceversa, la evolución de estos se puede observar en la figura 5, en general los extremos de ambos parámetros no son tan convenientes (valores muy altos o muy pequeños), pero esto va variando dependiendo del problema a resolver. Juntando el valor de C y de gamma que maximizan el valor de AUC en validación, se obtiene como resultado los presentes en la tabla 1.

SVM	Area bajo la curva ROC TRAIN	Area bajo la curva ROC Validación
C = 1.0 Gamma = 1.0	0.985388	0.981276

Table 1: Resultados de la clasificación utilizando SVM

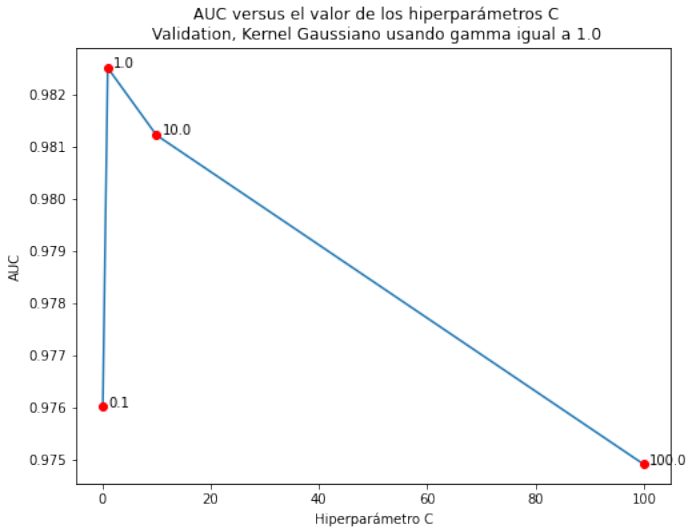
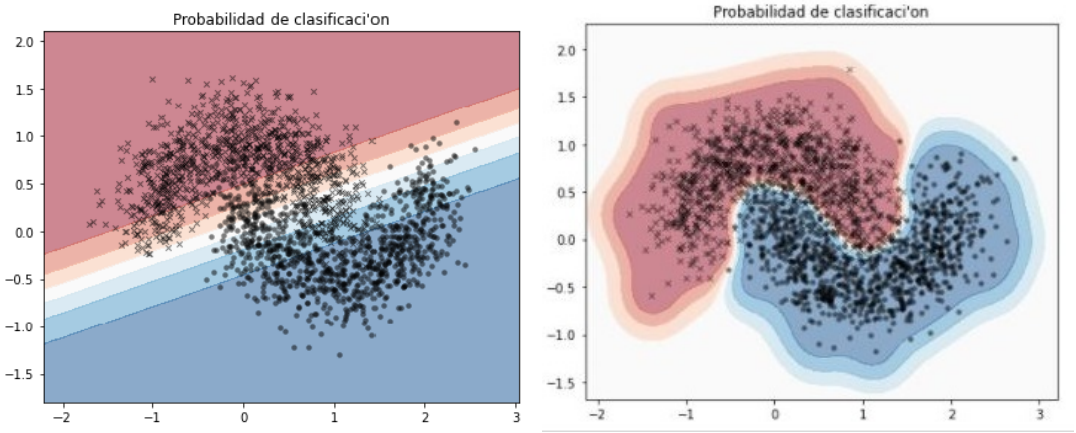


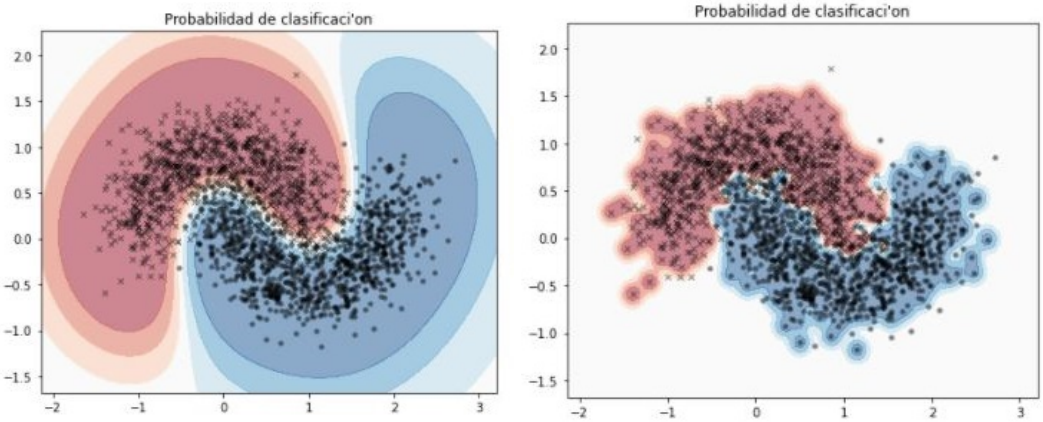
Figure 3: AUC para cada valor C ([0.1, 1.0, 10.0, 100.0]) usando kernel gaussiano



(a) Ejemplo hiperplanos separadores usando un kernel lineal

(b) Ejemplo hiperplanos separadores usando un kernel gaussiano

Figure 4: Efectos de los tipos de kernel para la forma de los hiperplano



(a) Ejemplo hiperplanos separadores usando gamma igual a 1

(b) Ejemplo hiperplanos separadores usando gamma igual a 100

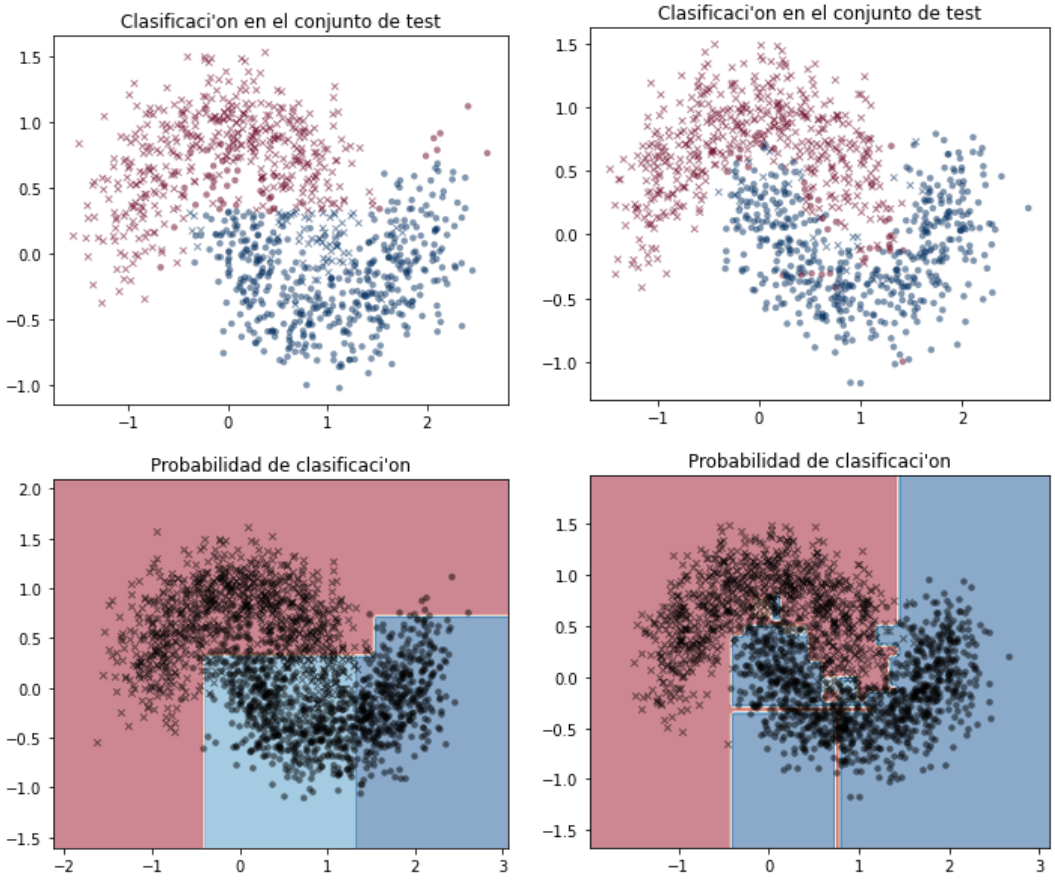
Figure 5: Efecto del parámetro gamma para kernel gaussiano

2.3 Clasificación Two Moons utilizando el algoritmo Random Forest (RF)

Como se observa en la figura 7 el modelo es más simple cuando se utiliza una máxima profundidad de 3, pero este divide el espacio de forma muy general, a pesar de esto se obtienen mejores resultados en test pues el modelo de RF con máxima profundidad 10 **se tiende a sobreajustar**, esto se puede observar en la tabla 2 en la cual se muestra que este modelo obtiene mejores resultados en train, pero en test disminuye su rendimiento. Esto es una muestra que no siempre los modelos más complejos son mejores, como se ve en la tabla mencionada anteriormente, se obtuvieron mejores resultados en test utilizando una máxima profundidad de 3, es decir, un modelo más simple. Para obtener mejores resultados, se debería aumentar la complejidad del modelo pero no tan al extremo de sobreajustar el modelo, por lo que probablemente una máxima profundidad entre $[4, 9]$ debería presentar mejores resultados en test.

Máxima profundidad del modelo	Area bajo la curva ROC TRAIN	Area bajo la curva ROC TEST
3	0.940086	0.923216
10	0.963304	0.909020

Table 2: Áreas bajo la curva para cada experimento de clasificacion random forest



(a) Modelo RF con máxima profundidad de 3

(b) Modelo RF con máxima profundidad de 10

Figure 6: Clasificación utilizando random forest con número de árboles del modelo en 1

La mejor combinación corresponde a 50 árboles y una máxima profundidad de 10, con esto es posible obtener un valor de AUC en TEST de 0.973304.

Valor AUC TEST		Máxima profundidad	
		3	10
Número de arboles	7	0.951376	0.968936
	50	0.957702	0.973304

Table 3: Valores de AUC en los datos de TEST para las 4 combinaciones de número de arboles y máxima profundidad

2.4 Covertype dataset

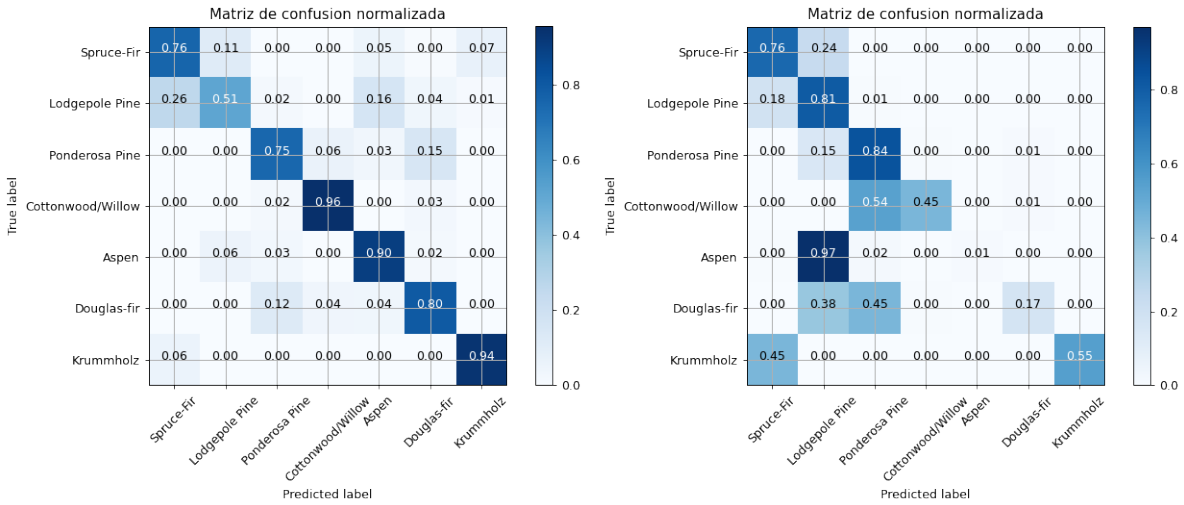
Se puede observar que las clases presentan un gran desbalance, las clases minoritarias son aproximadamente 70 veces menos que algunas de las clases mayoritarias.
Como se puede observar en la figura 7, se obtiene un mejor resultado para todas las clases utilizando el flag "class weight" en "balanced",además de esto, los promedios de Recall en validación son mayores cuando se tiene este flag activado, por lo que recomiendo utilizar este configuración.

	Ejemplos por clase
Entrenamiento	148288. - 198310. - 25028. - 1923 - 6645 - 12157 - 14357
Validación	63552. - 84991. - 10726. - 824 - 2848 - 5210 - 6153

Table 4: Datos generados Covertype dataset, el número de datos por clase se separan por "-"

Flag "class weight"	Recall por clase entrenamiento	Recall por clase validación	Numero de ejemplos en validacion clasificados correctamente
'balanced'	0.815	0.804	113406
None	0.524	0.512	130661

Table 5: Resultados de las tasas de Recall con el flag "class weight" en 'balanced' y None respectivamente.



(a) Modelo con el flag "class weight" en "balanced"

(b) Modelo con el flag "class weight" en None

Figure 7: Clasificación utilizando random forest usando 30 árboles con profundidad máxima 10.

El mejor modelo de Random Forest corresponde a el modelo usando el flag "class weight" en balanced y utilizando una profundidad máxima de 30, usando 30 árboles.

Profundidad	Promedio recall por clase (validación)
10	0.803
30	0.904
50	0.896

Table 6: Modelo Random Forest con corrección por desbalance

Las características más importantes se presentan en la figura 7, están capturan más del 80% de la importancia del problema. A partir de estas se puede concluir que se necesitan 13 características para capturar el 80% de la importancia, esto corresponde a aproximadamente un 24% del total de las características (54).

Características	Importancia
Elevation	0.237
Horizontal distance to roadways	0.102
Horizontal distance to fire points	0.087
Horizontal distance to hydrology	0.063
Vertical distance to hydrology	0.051
Cache la Poudre Wilderness Area	0.047
Aspect	0.044
Hillshade 9am	0.041
Hillshade noon	0.037
Hillshade 3pm	0.037
Slope	0.029
Soil type 10	0.022
Soil type 3	0.022

Table 7: Top 13 características con más importancia en la clasificación

2.5 Programación

Se programa un C-SVM multiclase del tipo one-versus-all, usando kernel Gaussiano, para el cual se probaron 5 combinaciones de los parámetros C y gamma con el fin de determinar luna buena combinación de parámetros, los resultados de esto se pueden observar en la figura 8.

C	Gamma	Tasa de acierto validación	Recall validación
1.0	1.0	0.7347	0.7350
1.0	10.0	0.4438	0.4440
10.0	1.0	0.73047	0.73050
10.0	10.0	0.4723	0.4723
100.0	1.0	0.7319	0.7320

Table 8: Resultados de la clasficiacion usando SVM para distintas combinaciones de parametros

En la figura 8 se presenta la matriz de confusión del mejor modelo encontrado experimentalmente, esta corresponde al modelo utilizando C = 1.0 y Gamma=1.0 .

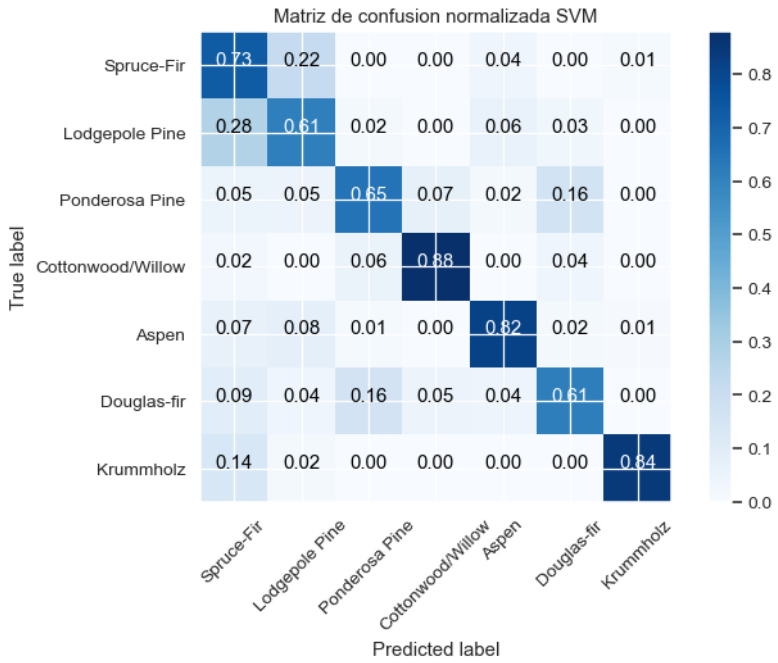


Figure 8: Matriz de confusión normalizada de la clasificación multiclase del dataset covertype utilizando el algoritmo SVM

2.5.1 Comparación con mejor modelo Random Forest

Las matrices de confusión de ambos modelos, las cuales corresponden a 7a y 8 permiten observar un mejor rendimiento en el modelo Random forest, además al analizar los resultados del promedio de recall en la validación el modelo de Random Forest es **bastante mayor** al promedio de recall en la validación del modelo C-SVM, esto queda ilustrado en la tabla 9. Por esto, es posible concluir que al menos para este dataset la clasificación utilizando Random Forest es mucho mejor que la clasificación utilizando C-SVM, esto utilizando el mejor modelo de Random Forest el cual corresponde a un modelo con 30 arboles y una profundidad máxima de 30.

Modelo	Recall promedio validación
Random Forest	0.904
C-SVM	0.7350

Table 9: Comparación resultados de los modelos Random Forest y C-SVM para clasificar el dataset Covertype

```
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_covtype
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix
from sklearn.svm import SVC
import matplotlib.pylab as plt

dataset = fetch_covtype(shuffle=True,as_frame=True) # flag de shuffle activado
para  que nos entregue los datos ya mezclados
dataframe = dataset.frame

df1 = dataframe[dataframe['Cover_Type']==1].sample(n=1000)
df2 = dataframe[dataframe['Cover_Type']==2].sample(n=1000)
df3 = dataframe[dataframe['Cover_Type']==3].sample(n=1000)
df4 = dataframe[dataframe['Cover_Type']==4].sample(n=1000)
df5 = dataframe[dataframe['Cover_Type']==5].sample(n=1000)
df6 = dataframe[dataframe['Cover_Type']==6].sample(n=1000)
df7 = dataframe[dataframe['Cover_Type']==7].sample(n=1000)

df = pd.concat([df1,df2,df3,df4,df5,df6,df7])
X = df.iloc[:,0:54].to_numpy()
Y = df['Cover_Type'].to_numpy()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7,
stratify=Y)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# transform data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform (X_test)

C_value=1.0
gamma=1.0

classifier = SVC(C=C_value, kernel='rbf', gamma=gamma,
probability=True,decision_function_shape='ovr')
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)
print(Y_pred)
```