Universidad De Guadalajara

Centro Universitario De Ciencias Exactas E Ingenierías.

Semanario de Inteligencia Artificial II

Actividad 04: Comparación de clasificadores no lineales.

Nombre:

Luis Alfredo Cruz Reyes | 214767231

Sección: D04

Introducción.

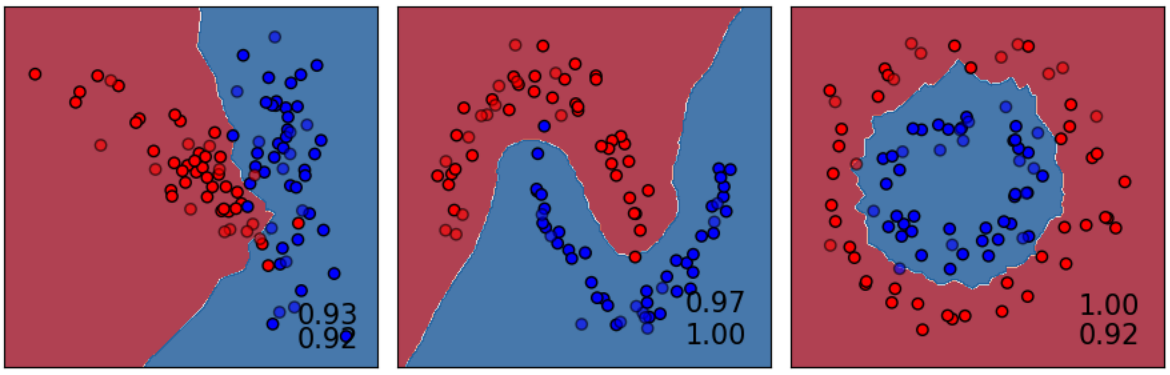
Aquí veremos algunos clasificadores no lineales, todos ellos estarán probados con sus parámetros por defecto, que están configurados para obtener los mejores resultados en la mayoría de casos. Veremos la línea de como importar su librería y los resultados que se obtienen teniendo en cuenta que datos generados ideales para ver cómo trabajan estos.

Clasificación por vecinos cercanos: KNN.

El clasificador K-Nearest Neighbors (KNN) funciona asignando una clase a un punto de datos desconocido basándose en la mayoría de las clases de sus K vecinos más cercanos en el espacio de atributos, donde la cercanía se mide mediante una métrica de distancia. KNN es un algoritmo de aprendizaje supervisado que se basa en la similitud entre puntos de datos para tomar decisiones de clasificación, y su elección de K y la métrica de distancia son aspectos clave en su rendimiento.

```
10  from sklearn.neighbors import KNeighborsClassifier
11  #from sklearn.tree import DecisionTreeClassifier
12  #from sklearn.neural_network import MLPClassifier
13  #from sklearnex.svm import SVC
14  #from sklearn.gaussian_process  import GaussianProcessClassifier
15  #from sklearn.gaussian_process.kernels import RBF
16  #from sklearn.naive_bayes import GaussianNB
```

Resultados:

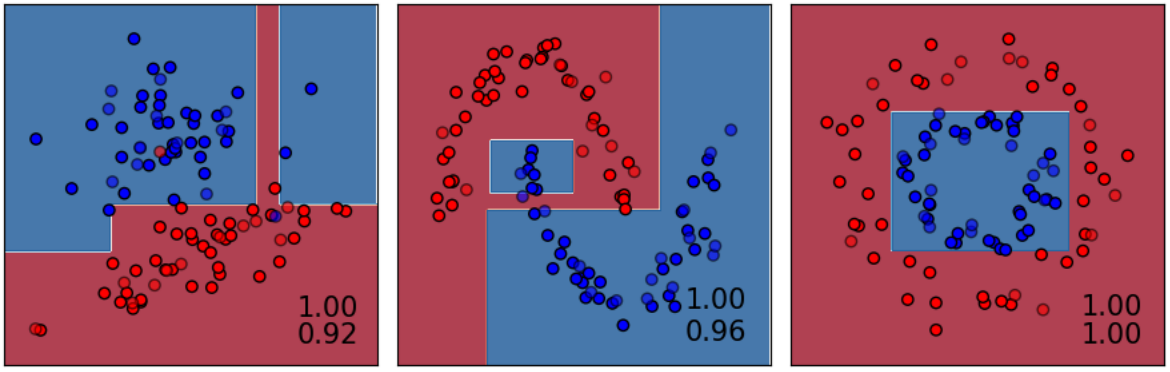


Clasificador de Árbol de decisión.

El clasificador de árbol de decisión funciona dividiendo repetidamente un conjunto de datos en subconjuntos más pequeños basándose en características específicas de los datos, de manera que cada subdivisión separe las clases de manera más efectiva. Estas divisiones se representan como nodos y ramas en un árbol, donde los nodos internos representan decisiones basadas en características y las hojas del árbol representan las clases finales.

```
10  #from sklearn.neighbors import KNeighborsClassifier
11  from sklearn.tree import DecisionTreeClassifier
12  #from sklearn.neural_network import MLPClassifier
13  #from sklearnex.svm import SVC
14  #from sklearn.gaussian_process  import GaussianProcessClassifier
15  #from sklearn.gaussian_process.kernels import RBF
16  #from sklearn.naive_bayes import GaussianNB
```

Resultados:

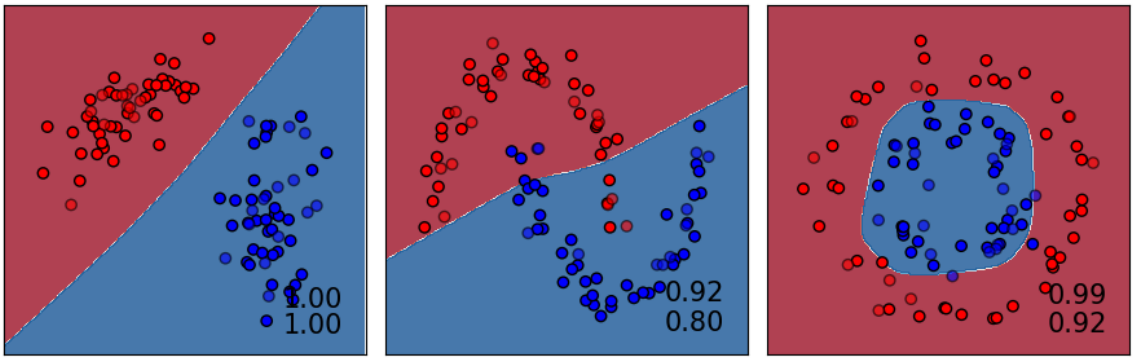


Clasificador Perceptrón multicapa.

Un Perceptrón Multicapa (MLP, por sus siglas en inglés) es una red neuronal artificial compuesta por múltiples capas de neuronas interconectadas. Cada neurona en una capa está conectada con todas las neuronas en la capa siguiente, y cada conexión tiene un peso asociado. Durante el entrenamiento, los pesos se ajustan iterativamente para minimizar la diferencia entre las salidas de la red y las etiquetas de entrenamiento.

```
10  #from sklearn.neighbors import KNeighborsClassifier
11  #from sklearn.tree import DecisionTreeClassifier
12  from sklearn.neural_network import MLPClassifier
13  #from sklearnex.svm import SVC
14  #from sklearn.gaussian_process  import GaussianProcessClassifier
15  #from sklearn.gaussian_process.kernels import RBF
16  #from sklearn.naive_bayes import GaussianNB
```

Resultados:

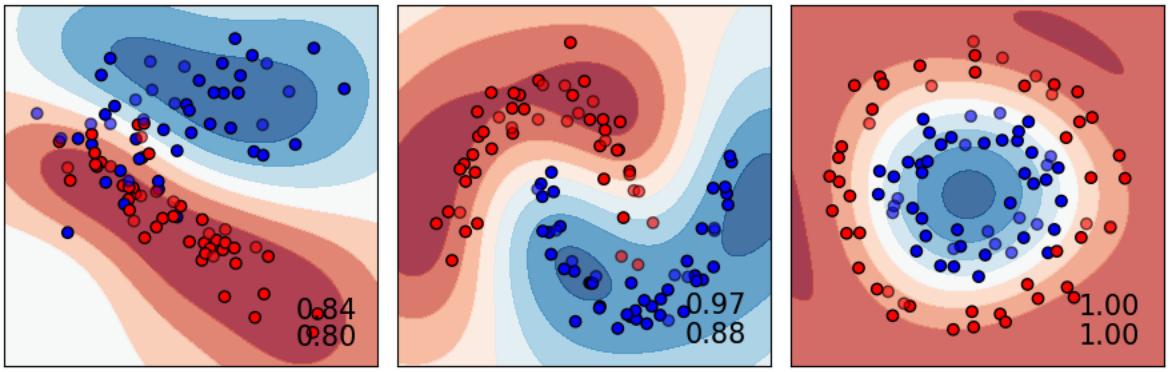


Clasificador de soporte por máquina de vector.

El clasificador Support Vector Machine (SVM, por sus siglas en inglés) funciona buscando una frontera de decisión óptima que separe eficazmente dos clases en un espacio de características. Esta frontera se denomina hiperplano y está posicionada de manera que maximice la distancia entre los puntos de datos más cercanos de ambas clases, llamados vectores de soporte.

```
10  #from sklearn.neighbors import KNeighborsClassifier
11  #from sklearn.tree import DecisionTreeClassifier
12  #from sklearn.neural_network import MLPClassifier
13  from sklearnex.svm import SVC
14  #from sklearn.gaussian_process  import GaussianProcessClassifier
15  #from sklearn.gaussian_process.kernels import RBF
16  #from sklearn.naive_bayes import GaussianNB
```

Resultados:



Conclusión.

Como podemos ver viendo solo algunos de los clasificadores con tres distintas datos de entradas, en los parámetros por defecto vemos que hay algunos que son mejores según el dataset, esto lo podemos mejorar evidentemente moviendo los parámetros, pero por lo general son bastante efectivos, incluso viendo como en la propia salida visual podemos ver que estos son diferentes.

Código:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles
from sklearn.datasets import make_classification

#from sklearn.neighbors import KNeighborsClassifier
#from sklearn.tree import DecisionTreeClassifier
#from sklearn.neural_network import MLPClassifier
from sklearnex.svm import SVC
#from sklearn.gaussian_process  import GaussianProcessClassifier
#from sklearn.gaussian_process.kernels import RBF
#from sklearn.naive_bayes import GaussianNB

# Crear datos sinteticos
x, y = make_classification(n_features=2, n_redundant= 0, n_informative=2,
n_clusters_per_class=1)
rng = np.random.RandomState(2)
x += 1 * rng.uniform(size = x.shape)
linearly_separable = (x, y)

datasets = [linearly_separable, make_moons(noise = 0.1), make_circles(noise
= 0.1, factor = 0.5)]

cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000','#0000FF'])

fig = plt.figure(figsize=(9,3))
h = 0.02        # step

for i,ds in enumerate(datasets):
    x, y = ds
    x = StandardScaler().fit_transform(x)
    xtrain, xtest, ytrain, ytest = train_test_split(x, y)

    model = SVC()
    model.fit(xtrain, ytrain)
    score_train = model.score(xtrain, ytrain)
    score_test = model.score(xtest, ytest)

    # Dibujo
```

```
    ax = plt.subplot(1,3,i+1)
    xmin, xmax = x[:,0].min() - 0.5, x[:,0].max() + 0.5
    ymin, ymax = x[:,1].min() - 0.5, x[:,1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))

    if hasattr(model, 'decision_function'):
        zz = model.decision_function(np.c_[xx.ravel(), yy.ravel()])
    else:
        zz = model.predict(np.c_[xx.ravel(), yy.ravel()])

    zz = zz.reshape(xx.shape)
    ax.contourf(xx, yy, zz, cmap = cm, alpha = 0.8)

    ax.scatter(xtrain[:,0],xtrain[:,1], c = ytrain, cmap = cm_bright,
edgecolors ='k')
    ax.scatter(xtest[:,0],xtest[:,1], c = ytest, cmap = cm_bright,
edgecolors ='k', alpha = 0.6)

    ax.set_xlim(xmin, xmax)
    ax.set_ylim(ymin, ymax)
    ax.set_xticks([])
    ax.set_yticks([])

    ax.text(xmax - 0.3, ymin+0.7,'%.2f'%score_train, size = 15,
horizontalalignment = 'right')
    ax.text(xmax - 0.3, ymin+0.3,'%.2f'%score_test, size = 15,
horizontalalignment = 'right')

plt.tight_layout()
plt.show()
```