



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

**Centro de
e-Learning**

UNIDAD DIDÁCTICA I
MACHINE LEARNING

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

Módulo II – Herramientas avanzadas y casos de estudio

Unidad I – sklearn – Supervisado II.



Presentación:

En ocasiones, cuando el caso que debemos analizar se encuentra relacionado con el número de muestras, es posible que los modelos paramétricos no nos sirvan para analizar nuestros casos particulares y necesitemos algún tipo de modelo más flexible. La exactitud de este tipo de modelos se ve afectará en gran medida por la cantidad de muestras, es decir que a mayor cantidad de muestras el modelo se vuelve más exacto.

Los modelos no paramétricos no aprenden parámetros, aprenden características o atributos sobre los datos, pero no parámetros en el sentido formal, por lo cual no vamos a obtener del modelo un vector de coeficientes como puede darse en una regresión lineal.

En esta unidad vamos a abordar dos tipos de modelos no paramétricos:

- Árboles de decisión
- KNN.



Objetivos:

Que los participantes:

Aprendan a trabajar con modelos no paramétricos de clasificación.

Afiancen el proceso de creación de modelo con sklearn y la visualización de datos.

Centro de e-Learning SCEU UTN - BA.

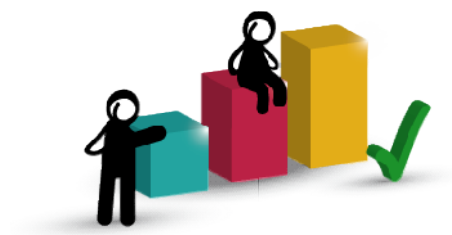
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



Bloques temáticos:

- 1.- K-Nearest Neighbors (K-NN)
- 2.- K-NN Clasificación.



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. K-Nearest Neighbors (K-NN)

El algoritmo (k-NN) es una forma de aprendizaje automático supervisado que se utiliza para predecir categorías, **sklearn.neighbors** proporciona funcionalidad para los métodos de aprendizaje basados en vecinos supervisados y sin supervisión. El aprendizaje supervisado basado en vecinos se presenta en dos tipos:

- Clasificación para datos con etiquetas discretas
- Regresión para datos con etiquetas continuas.

El principio detrás de los métodos del vecino más cercano es encontrar un número predefinido de ejemplos de entrenamiento más cercanos a la distancia al nuevo punto, y predecir la etiqueta a partir de estos. El número de muestras puede ser una constante definida por el usuario, o variar según la densidad local de puntos. La distancia puede ser, en general, cualquier medida métrica, los métodos basados en vecinos se conocen como métodos de aprendizaje automático no generalizados, ya que simplemente "recuerdan" todos sus datos de entrenamiento.

A pesar de su simplicidad, los vecinos más cercanos han tenido éxito en una gran cantidad de problemas de clasificación y regresión, en áreas de salud, análisis de imágenes satelitales, finanzas, ciencias políticas, reconocimiento de vides y más. Al ser un método no paramétrico, a menudo es exitoso en situaciones de clasificación donde el límite de decisión es muy irregular. Podría ser aplicado con mucho éxito en ambientes políticos complejos para clasificar votantes potenciales de un determinado candidato.

KNN es un algoritmo de aprendizaje no **paramétrico** y **perezoso**.

- **No paramétrico** significa que no hay suposiciones para la distribución de datos subyacentes, por lo que es muy útil en la práctica donde la mayoría de los conjuntos de datos del mundo real no siguen supuestos teóricos matemáticos.
- **Algoritmo perezoso** significa que no necesita datos de entrenamiento para la generación del modelo. Todos los datos de entrenamiento son utilizados en la fase de prueba. Esto hace que la capacitación sea más rápida y la fase de prueba más lenta y costosa. La fase de prueba costosa significa más tiempo y memoria. En el peor de los casos, KNN necesita más tiempo para escanear todos los puntos de datos y escanear todos los puntos de datos requerirá más memoria para almacenar datos de entrenamiento.

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

K-NN Clasificación

La clasificación basada en vecinos es un tipo de aprendizaje basado en instancias o no generalizado, no intenta construir un modelo interno general, sino que simplemente almacena instancias de los datos de entrenamiento. La clasificación se calcula a partir de una mayoría simple de votos de los vecinos más cercanos de cada punto, a un punto de consulta se le asigna la clase de datos que tiene más representantes dentro de los vecinos más cercanos de dicho punto. scikit-learn implementa dos clasificadores de vecinos más cercanos:

- **KNeighborsClassifier** implementa el aprendizaje basado en los k vecinos más cercanos de cada punto de consulta, donde k es un valor entero especificado por el usuario.
- **RadiusNeighborsClassifier** implementa el aprendizaje basado en el número de vecinos dentro de un radio r fijo de cada punto de entrenamiento, donde r es un valor de punto flotante especificado por el usuario.

La clasificación de vecinos en KNeighborsClassifier es la técnica más utilizada, la elección óptima del valor k depende en gran medida de los datos, en general, un valor más alto de k suprime los efectos del ruido, pero hace que los límites de clasificación sean menos definidos. En los casos en que los datos no se muestrean de manera uniforme, la clasificación de vecinos basada en el radio puede ser una mejor opción. El usuario especifica un radio fijo r , de modo que los puntos en vecindarios más dispersos usan menos vecinos cercanos para la clasificación. Para espacios de parámetros de alta dimensión, este método se vuelve menos efectivo debido a la llamada "maldición de la dimensionalidad".

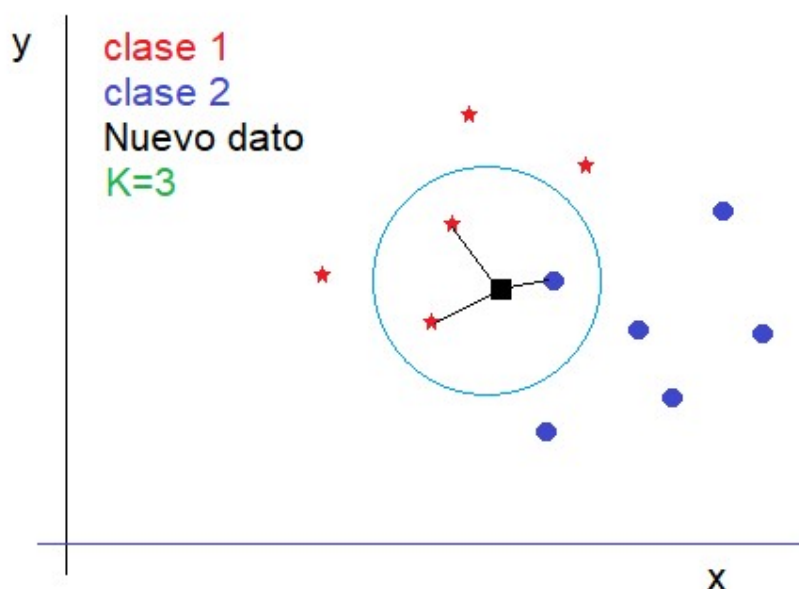
La clasificación básica de vecinos más cercanos utiliza ponderaciones uniformes: es decir, el valor asignado a un punto de consulta se calcula a partir de una mayoría simple de votos de los vecinos más cercanos. En algunas circunstancias, es mejor ponderar a los vecinos de modo que los vecinos más cercanos contribuyan más al ajuste. Esto se puede lograr a través de la palabra clave **weights**:

- **weights = 'uniform'**, asigna pesos uniformes a cada vecino.

- **weights = 'distance'** asigna pesos proporcionales al inverso de la distancia desde el punto de consulta. Alternativamente, se puede suministrar una función de la distancia definida por el usuario para calcular los pesos.

Cómo funciona

Supongamos que tenemos una muestra clasificada en dos clases (clase 1 y clase 2) según se muestra a continuación, y tomemos el valor de $k = 3$.






En este caso dado un nuevo punto a clasificar, se realizan los siguientes pasos:

1. Se calcula la distancia a los datos.
2. Se encuentran los tres más cercanos.
3. Se clasifica según categoría de la mayoría.

Ejemplo de clasificación mediante knn

Veamos un ejemplo que podemos obtener de la página de scikit-learn, en el cual se utiliza los datos de tres variantes de la flor “Iris” para realizar una **clasificación**.



Iris Setosa	Iris Versicolor	Iris Virginica
		
Referencia autor: CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=170298	Referencia autor: CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=248095	Referencia autor: De Frank Mayfield - originally posted to Flickr as Iris virginica shrevei BLUE FLAG, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=9805580

La clasificación se podría realizar en función de:

- El largo del sépalo.
- El ancho del sépalo.
- El largo del pétalo.
- El ancho del pétalo.

En esta oportunidad realizaremos la clasificación por:

- El largo del sépalo.
- El ancho del sépalo.

Los datos fueron recopilados por Edgar Anderson, los cuales fueron introducidos por Ronald Fisher en 1936 y se conocen como **Iris conjunto de datos de Fisher**. Este conjunto de datos se convirtió en un caso de prueba típico para muchas técnicas de clasificaciones estadísticas en aprendizaje automático

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



Paso 1

Primero debemos importar las librerías a utilizar:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
```

Paso 2

De datasets importamos el conjunto de datos de iris() y establecemos el número de vecino más cercano en 15

```
iris = datasets.load_iris()
n_neighbors = 15
```

Paso 3

La librería Iris se encuentra separada en:

- **data** que contiene todas las características.
- **target** que contiene las clases asociadas a esas características.

Los valores de target no se encuentran como string sino que tienen asociado para cada string un valor numérico de forma de poder analizar los datos. Como dijimos antes, del conjunto de datos posibles vamos a tomar solo las dos primeras características por lo que podemos representar los datos de la siguiente manera.

```
X = iris.data[:, :2]
y = iris.target
```

En este caso en **data** estamos indicando [**filas, columnas**], con dos puntos (:) estamos indicando todas las filas y con :2 (o 0:2) indicamos las primeras dos columnas. También podríamos indicar otras columnas especificando el rango como 1:3 en donde podríamos tomar de la segunda columna a la tercera.



Paso 4

Para analizar los datos visualmente crearemos un mapa de colores mediante **ListedColormap** a partir de una lista de colores, y utilizaremos un mallado con un paso de 0.2

```
h = .02
```

```
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
```

```
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

Paso 5

Ahora realizaremos dos clasificaciones, una para un peso uniforme y otra para un peso en función al inverso de la distancia.

```
for weights in ['uniform', 'distance']:
```

```
    # Creamos una instancia del clasificador de vecinos más cercanos y le pasamos los  
    # datos mediante fit().
```

```
    # El primer parámetro de KNeighborsClassifier es con cuantos vecinos quiero clasificar  
    # y el segundo el tipo de peso a utilizar.
```

```
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)  
    clf.fit(X, y)
```

```
    # Establecemos los límites del gráfico y asignamos un color a cada punto de malla.
```

```
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
```

```
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
```

```
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
    # Agregamos el resultado al gráfico
```

```
    Z = Z.reshape(xx.shape)
```

```
    plt.figure()
```

```
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
    # Ploteo los datos de entrenamiento
```

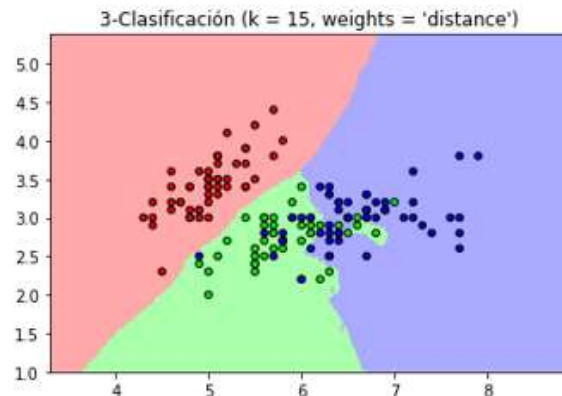
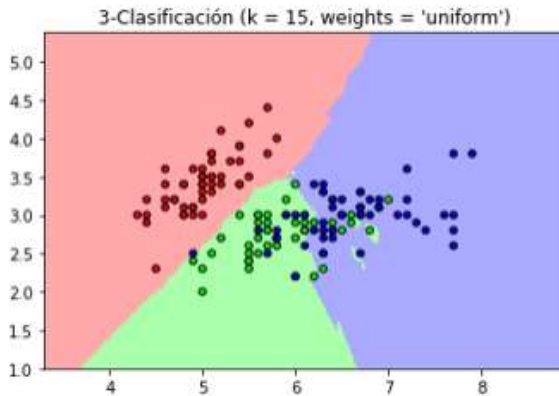
```
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
```

```
    plt.xlim(xx.min(), xx.max())
```

```
    plt.ylim(yy.min(), yy.max())
```



```
plt.title("3-Clasificación (k = %i, weights = '%s')" % (n_neighbors, weights))  
plt.show()
```



Si ahora creamos un conjunto de datos X_n a testear y lo incluimos dentro del bucle for podemos obtener la predicción mediante ambos pesos, según se muestra a continuación:

```
#Ploteo un nuevo dato  
Xn = np.array([[7.3,3], [5.1,2.9], [6.4,3.2]])  
Yn = clf.predict(Xn)  
print(Yn)  
plt.show()
```

Resultado:

weight = uniform	weight = distance
[2 0 2]	[2 0 1]

K-NN Regresión

La regresión basada en vecinos se puede utilizar en los casos en que las etiquetas de datos son variables continuas en lugar de discretas. La etiqueta asignada a un punto de consulta se calcula en función de la media de las etiquetas de sus vecinos más cercanos. scikit-learn implementa dos tipos de regresiones diferentes:

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

KNeighborsRegressor implementa el aprendizaje basado en los vecinos más cercanos de cada punto de consulta, donde r es un valor entero especificado por el usuario.

RadiusNeighborsRegressor implementa el aprendizaje basado en los vecinos dentro de un radio r fijo del punto de consulta, donde r es un valor de punto flotante especificado por el usuario.

La regresión básica de vecinos más cercanos utiliza pesos uniformes, es decir, cada punto en el vecindario local contribuye de manera uniforme a la clasificación de un punto de consulta. Bajo algunas circunstancias, puede ser ventajoso ponderar los puntos de manera tal que los puntos cercanos contribuyan más a la regresión que los puntos lejanos. Esto se puede lograr a través de la palabra clave **weights** (pesos). El valor predeterminado, **weights = 'uniform'**, asigna pesos iguales a todos los puntos mientras que **weights = 'distance'** asigna pesos proporcionales al inverso de la distancia desde el punto de consulta. Alternativamente, se puede suministrar una función de la distancia definida por el usuario, que se usará para calcular los pesos.

2.- Árboles de decisiones

Un árbol de decisiones se asemeja a las raíces de un árbol, en donde partimos de un conjunto de datos con determinadas características, que llamaremos raíz principal y que iremos descomponiendo por atributos, en ramas a partir de una determinada clasificación. Cada descomposición lleva asociada una condición que puede resultar verdadera o falsa y que se encuentra relacionada a una caracterización específica.

Podríamos tener por ejemplo el atributo “**tipo de vehículo**” con valores:

- Camionetas
- Autos

Y el atributo “**tracción**”, con valores:

- Cuatro ruedas
- Dos ruedas

En base a estos atributos podríamos crear un árbol en el cual la primera división se realice por “tipo de vehículo” y luego por “tracción” o al revés. Esta división la realizaremos a partir de un algoritmo que optimice la forma en la cual se lleva a cabo la división en base a un análisis probabilístico.

Cuanto más profundo es el árbol, más complejas son las reglas de decisión y más se ajusta el modelo

Algunas ventajas de los árboles de decisión son:

- Sencillos de entender e interpretar. Los árboles se pueden visualizar.
- Requiere poca preparación de datos.
- Permite trabajar con datos tanto numéricos como categóricos.
- Permite manejar problemas de salida múltiple.
- Posible validar un modelo utilizando pruebas estadísticas. Eso permite dar cuenta de la fiabilidad del modelo.

Algunas desventajas de los árboles de decisión incluyen:

- Los aprendices de árboles de decisión pueden crear árboles demasiado complejos que no generalizan bien los datos. Esto se llama sobreajuste.
- Los árboles de decisión pueden ser inestables porque las pequeñas variaciones en los datos pueden dar lugar a que se genere un árbol completamente diferente.
- Los aprendices de árboles de decisión crean árboles sesgados si algunas clases dominan. Por lo tanto, se recomienda equilibrar el conjunto de datos antes de ajustar con el árbol de decisión.

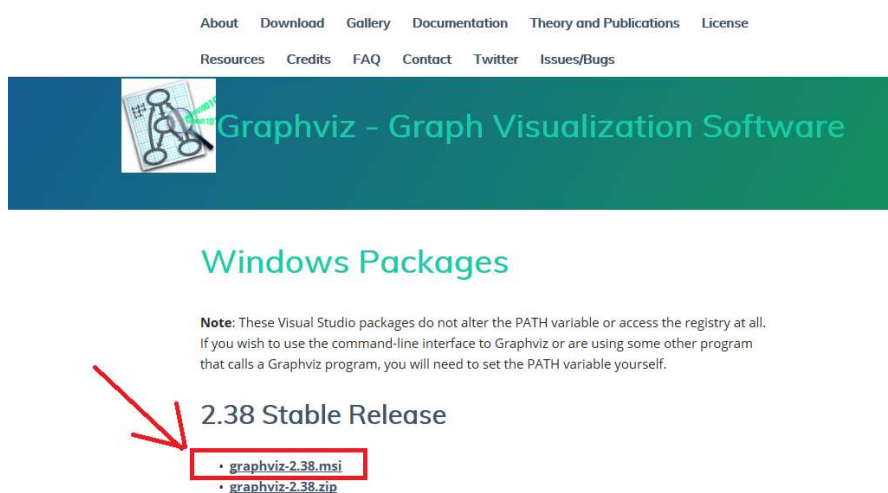
Instalaciones adicionales

Para poder trabajar cómodamente con árboles de decisiones, vamos a instalar un par de librerías, que nos van a ayudar a visualizar los datos, para ello seguiremos los siguientes pasos:

PASO 1 - Instalamos Graphviz

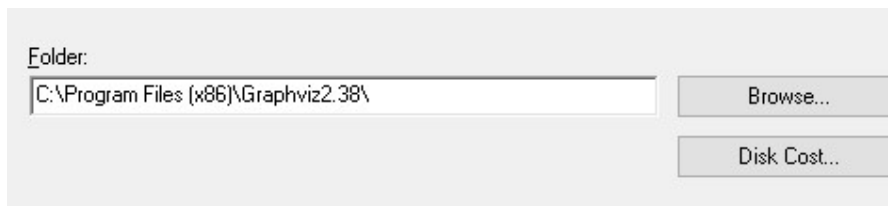
Vamos a la página de graphviz y descargamos el ejecutable:

https://graphviz.gitlab.io/pages/Download/Download_windows.html



Una vez descargado simplemente damos siguiente hasta terminar la instalación y nos fijamos en donde es que se va a instalar, en mi caso lo realiza en:

C:\Program Files (x86)\Graphviz2.38



PASO 2

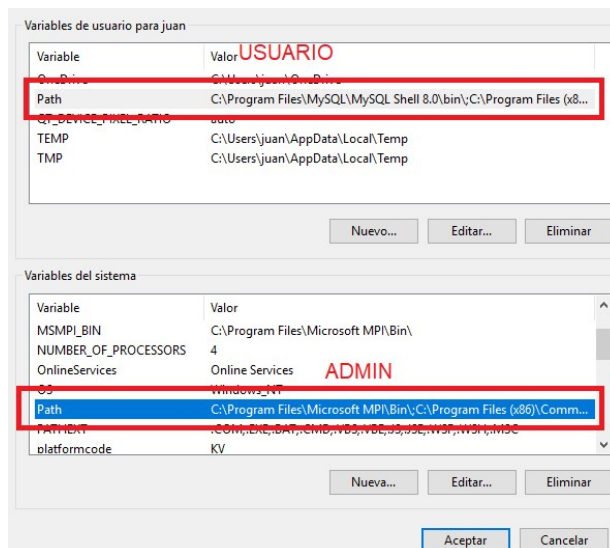
Si estoy trabajando en Windows, copio la ruta a donde se guardó el directorio “bin”, el cual se encuentra dentro de la instalación, en mi caso es:

C:\Program Files (x86)\Graphviz2.38\bin

Y al archivo dot.exe que está dentro de bin:

C:\Program Files (x86)\Graphviz2.38\bin\dot.exe

Y agrego la primera ruta al path del usuario y la segunda al path de admin en el panel de variables de entorno.



PASO 3

En el prompt de anaconda ejecutamos:

```
conda install python-graphviz
```

E instalamos pydotplus

```
pip install pydotplus
```

Ejemplo de árbol de clasificación utilizando iris

Vamos ahora a analizar el ejemplo realizado con k-nn pero mediante la creación de un árbol de clasificación, para ello en primer lugar importamos las librerías a utilizar, en donde tenemos el clasificador de árbol de decisiones **DecisionTreeClassifier**, y la librería **iris**:

```
import numpy as np

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import export_graphviz

import graphviz

import matplotlib.pyplot as plt
```

Cargamos la librería en la variable iris.

```
iris=load_iris()
```

Utilizamos **train_test_split** para crear las variables de entrenamiento y test en base a la librería **iris**:

```
X_entrenamiento, X_test, y_entrenamiento, y_test=train_test_split(iris.data, iris.target)
```

Creamos un objeto de clasificación y le indicamos la profundidad del árbol, es decir cuántos niveles bifurcaciones pretendemos. Si no indicamos un nivel de profundidad, el árbol podría generar un sobre ajuste de los datos, que si bien nos retornaría un 100% de acierto para los datos de entrenamiento, sería demasiado específico :

```
arbol=DecisionTreeClassifier(max_depth=3)
```



Y le pasamos los datos de entrenamiento:

```
arbol.fit(X_entrenamiento, y_entrenamiento)
```

En base al modelo ahora realizamos una predicción mediante `score()` para los datos de prueba:

```
arbol.score(X_test, y_test)
```

El modelo nos retorna un 97.3% de aciertos correctos: 0.9736842105263158

Si lo probamos con los datos de prueba:

```
arbol.score(X_entrenamiento, y_entrenamiento)
```

Obtenemos que el modelo nos retorna un 97,3% de aciertos correctos: 0.9732142857142857

Ahora podemos graficar la clasificación que el modelo ha realizado, antes podemos ver que alternativamente a agregar la librería Graphviz en las variables de entorno, podríamos lograr lo mismo de la siguiente forma:

```
import os  
os.environ["PATH"] += os.pathsep + 'C:\Program Files (x86)\Graphviz2.38\bin'
```

Ahora nos toca recuperar los datos que vamos a volcar dentro del esquema de árbol en un archivo mediante **`export_graphviz()`**

```
export_graphviz(arbol, out_file='arbol1.dot', class_names=iris.target_names,  
feature_names=iris.feature_names, impurity=False, filled=True)
```

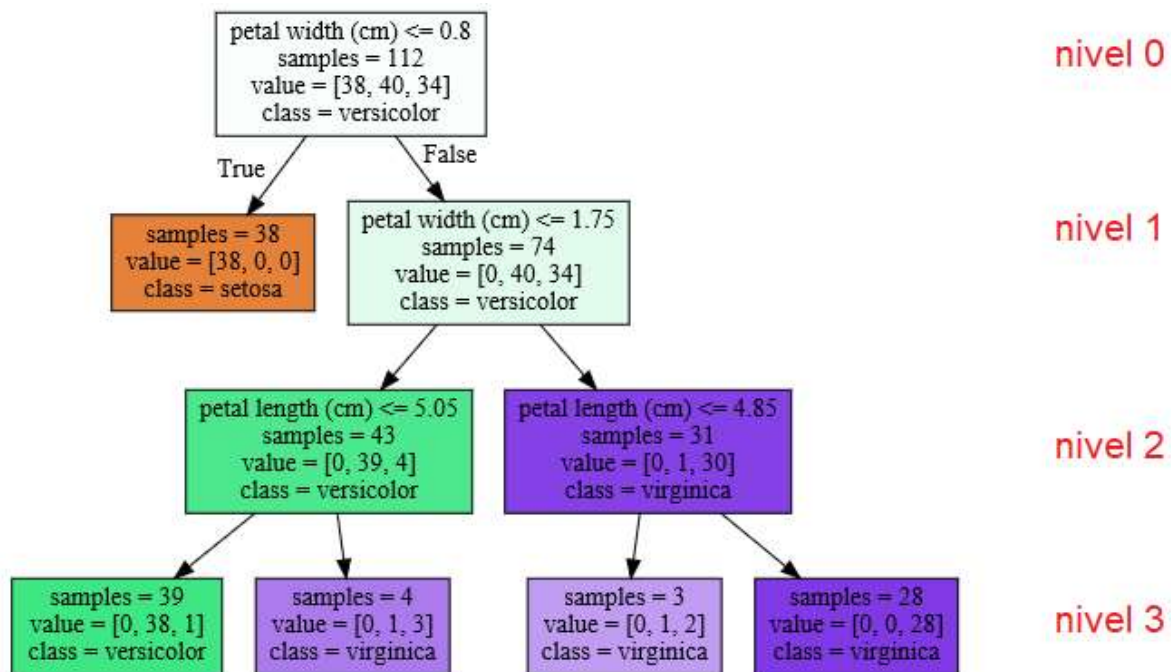


Luego recorremos el archivo y lo graficamos:

```
with open('arbol1.dot') as f:
```

```
    dot_graph=f.read()
```

```
graphviz.Source(dot_graph)
```



Dado el árbol obtenido, podemos analizar mediante `feature_importances_` la importancia asignada a cada característica y ver el resultado en un gráfico, de característica vs importancia asignada:

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



```
caracteristica=iris.data.shape[1]
```

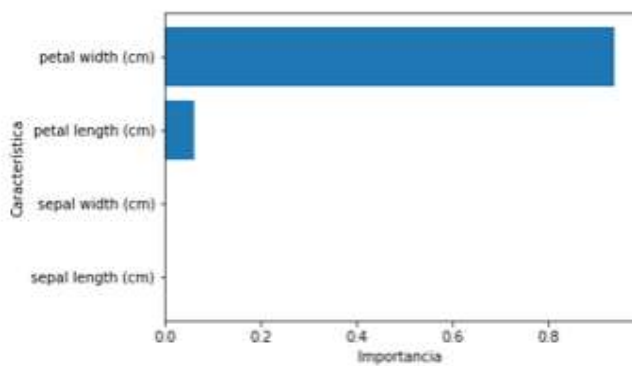
```
plt.barh(range(caracteristica),arbol.feature_importances_)
```

```
plt.yticks(np.arange(caracteristica),iris.feature_names)
```

```
plt.xlabel('Importancia')
```

```
plt.ylabel('Característica')
```

```
plt.show()
```



Del análisis sale que el modelo le ha asignado mucha importancia al ancho de los pétalos.



Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Mastering Exploratory Analysis with pandas - By Harish Garg - September 2018

Manual online

(2019, 01). Obtenido 01, 2019, de <https://pandas.pydata.org/>

(2019, 01). Obtenido 01, 2019, de <https://matplotlib.org/>

(2019, 01). Obtenido 01, 2019, de <https://scikit-learn.org/stable/>

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



Lo que vimos

En esta unidad hemos trabajado con modelos no paramétricos.



Lo que viene:

En la siguiente unidad seguiremos trabajando con sklearn.