UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

UNIDAD DIDÁCTICA VII

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 VII – sklearn – No Supervisado.

Presentación:

En el aprendizaje no supervisado los datos de entrenamiento consisten en un conjunto de vectores de entrada x sin ningún valor objetivo correspondiente. En este tipo de casos el objetivo suele ser descubrir grupos similares dentro de los datos (agrupamientos), o determinar la distribución de datos dentro del espacio de entrada (estimación de densidad), intentando proyectar los datos en 2 o tres dimensiones para tratar de visualizarlos.

En esta unidad nos introduciremos en el aprendizaje no supervisado utilizando sklearn.

Objetivos:

Que los participantes:

Aprendan a trabajar con modelos no supervisados.

Aprendan a trabajar con pre-procesamiento de datos.

Bloques temáticos:

1.- Agrupación de datos sin etiquetas (Clustering).

2.- Pre-preprocesamiento de datos.

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. Agrupación de datos sin etiquetar (Clustering)

Podemos utilizar sklearn para trabajar con agrupaciones de datos sin etiquetar, es decir que no necesitamos las etiquetas como en el caso de los métodos supervisados, sino solo sus características y esperaríamos que el método encuentre algún patrón para realizar la clasificación. Cada algoritmo de agrupación viene en dos variantes:

- una clase, que implementa el método de ajuste para aprender los agrupamientos en datos.
- una función, que, dados los datos, retorna una serie de etiquetas de números enteros correspondientes a los diferentes agrupamientos.

Para la clase, las etiquetas sobre los datos de entrenamiento se pueden encontrar con el atributo **labels**.

Los algoritmos implementados en este módulo pueden tomar diferentes tipos de matrices como entrada. Todos los métodos aceptan matrices de datos estándar de forma **[n_muestras, n_caracteristicas]**. Estos se pueden obtener de las clases en el módulo **sklearn.feature_extraction**. Para **AffinityPropagation, SpectralClustering** y **DBSCAN** también se pueden ingresar matrices de similitud de formas **[n_muestras, n_muestras].** Estos se pueden obtener de las funciones en el módulo **sklearn.metrics.pairwise**.

K-means

El algoritmo KMeans agrupa los datos al tratar de separar muestras en n grupos de igual varianza, minimizando un criterio conocido como la inercia o la suma de cuadrados dentro del clúster. Este algoritmo requiere que se especifique la cantidad de grupos. Se adapta bien a un gran número de muestras y se ha utilizado en una amplia gama de áreas de aplicación en muchos campos diferentes.

El algoritmo k-means divide un conjunto **X** de **N** muestras en **K** grupos separados **C**, cada una descrita por la media μ_j de las muestras en el grupo. Las medias son comúnmente

llamadas los "**centroides**", los cuales no son en general puntos de X aún cuando conviven en el mismo espacio. El algoritmo K-means tiene como objetivo elegir los **centroides** que minimizan **la inercia**, o el criterio de **suma de cuadrados** dentro del grupo: $\sum_{i=0}^{n} min\left(\|x_i - \mu_j\|^2\right)$ con $\mu_j \in C$

La **inercia** se puede tomar como una medida de cuan coherentes son los grupos internamente. Tiene varios inconvenientes:

- La inercia asume que los conglomerados son convexos e isotrópicos, lo que no siempre es así. Responde mal a los grupos alargados, o múltiples con formas irregulares.
- La inercia no es una métrica normalizada: solo sabemos que los valores más bajos son mejores y que el cero es óptimo. Pero en espacios de muy alta dimensión, las distancias euclidianas tienden a inflarse (esto es un ejemplo de la llamada "maldición de la dimensionalidad"). La ejecución de un algoritmo de reducción de dimensionalidad como el análisis de componentes principales (PCA) antes de la agrupación de k-means puede aliviar este problema y acelerar los cálculos.

K-means se conoce a menudo como el algoritmo de **Lloyd**. En términos básicos, el algoritmo tiene tres pasos.

1. El primer paso elige los centroides iniciales, con el método más básico para elegir muestras del conjunto de datos
2. Después de la inicialización, K-means consiste en hacer un bucle entre los otros dos pasos.
    a. El primer paso asigna cada muestra a su centroide más cercano.
    b. El segundo paso crea nuevos centroides tomando el valor medio de todas las muestras asignadas a cada centroide anterior.

    La diferencia entre los centroides antiguos y los nuevos se calcula y el algoritmo repite estos dos últimos pasos hasta que este valor es menor que un determinado umbral. En otras palabras, se repite hasta que los centroides no se mueven significativamente.

Dado el tiempo suficiente, K-medias siempre convergerá, sin embargo, esto puede ser a un mínimo local. Esto depende en gran medida de la inicialización de los centroides. Como resultado, el cálculo a menudo se realiza varias veces, con diferentes inicializaciones de los centroides. Un método para ayudar a resolver este problema es el esquema de inicialización k-means ++, que se ha implementado en scikit-learn (use el parámetro init = 'k-means ++'). Esto inicializa los centroides para que estén (generalmente) distantes entre sí, lo que lleva a resultados probablemente mejores.

El algoritmo admite ponderaciones de muestra, que se pueden dar mediante un parámetro sample_weight. Esto permite asignar más peso a algunas muestras al calcular centros de clúster y valores de inercia.

Veamos cómo este método clasifica los puntos de nuestra librería de datos iris, partamos de importar las librerías y el conjunto de datos con sus etiquetas:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn import metrics

iris = datasets.load_iris()
datos = iris.data
etiquetas = iris.target
```

Al utilizar el método KMeans podemos estimar:

- La cantidad de centroides, que corresponde a la cantidad de grupos que esperamos (etiquetas) (n_clusters)
- El número máximo de iteraciones (max_iter), es decir cuántas veces se mueve el centroide de posición para realizar el ajuste.

```
k_means = KMeans(n_clusters=3, max_iter=2000)
```

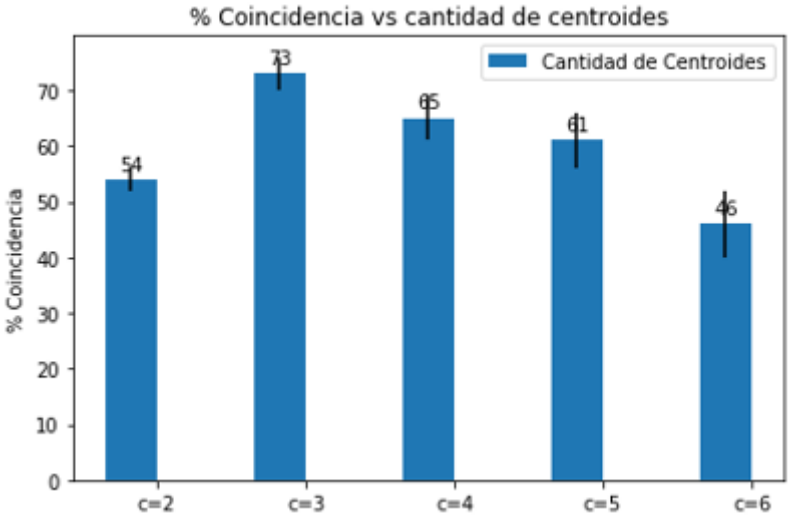
Luego llenamos el modelo con los datos y obtenemos las predicciones realizadas por el modelo:

```
k_means.fit(datos)
predicciones=k_means.predict(datos)
```

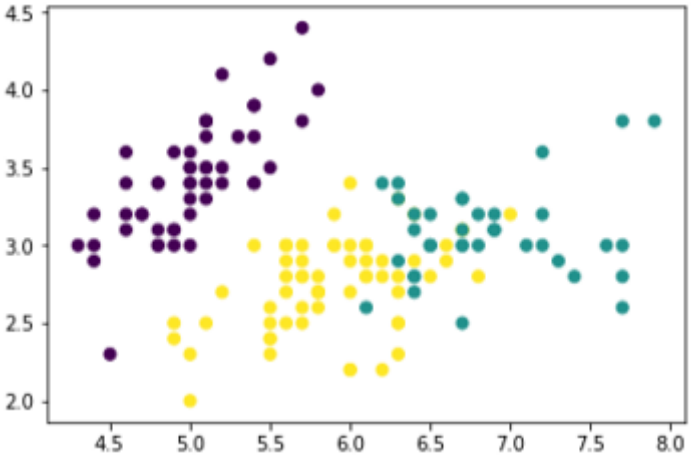
Dado que para iris tenemos ya la clasificación, podemos evaluar cuan preciso ha sido el método de clasificación.

score=metrics.adjusted_rand_score(etiquetas, predicciones)

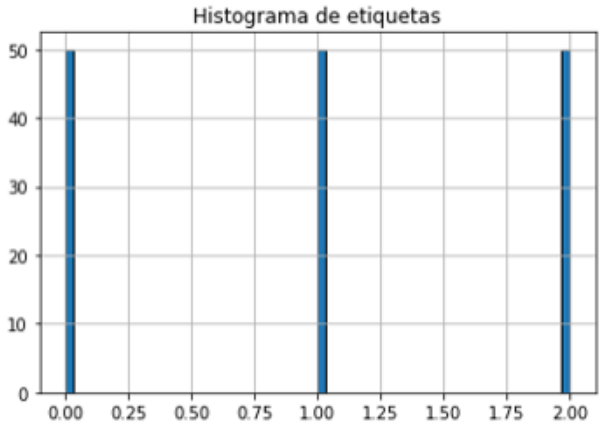
En este caso nos ha dado para 3 centroides como esperábamos un porcentaje de coincidencia del 73%. Si graficamos los porcentajes de coincidencia en función de la cantidad de centroides obtenemos el siguiente histograma:

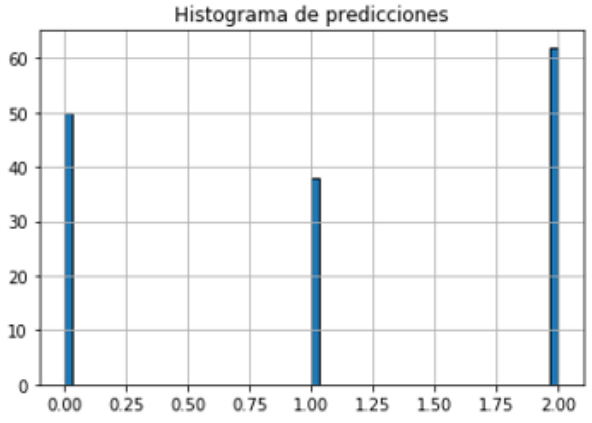


Observamos como la clasificación obtiene un mayor valor de aciertos al considerar tres centroides, podemos ver la representación de la clasificación realizada al graficar una de las columnas de datos en función de la predicción para el caso de tres centroides.



Al realizar un histograma de la cantidad de valores obtenidos para cada etiqueta según la predicción, vemos que al modelo lo ha costado diferenciar mas a los elementos del grupo uno y dos.





Existen casos en los cuales el modelo K-means puede producir clusters no intuitivos y posiblemente inesperados, podemos encontrar un análisis de estos casos en la siguiente referencia de sklearn:

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py

MiniBatchKMeans

El algoritmo K-mean posee una variante llamada MiniBatchKMeans el cual utiliza mini lotes para reducir el tiempo de cálculo, mientras intenta optimizar la misma función objetivo. Los mini lotes son subconjuntos de los datos de entrada, muestreados aleatoriamente en cada iteración de entrenamiento. Estos mini lotes reducen drásticamente la cantidad de cálculos necesarios para converger a una solución local. En contraste con otros algoritmos que reducen el tiempo de convergencia de k-means, los mini-batch-k-means producen resultados que son generalmente solo un poco peores que el algoritmo estándar. MiniBatchKMeans converge más rápido que KMeans, pero la calidad de los resultados se reduce, en la práctica, esta diferencia en la calidad puede ser bastante pequeña, de hecho si en nuestro caso anterior ejecutamos este método obtenemos para tres centroides un porcentaje ligeramente superior del 75,8% contra el 73% que habíamos obtenido.

Propagación por afinidad - Affinity Propagation.

La propagación por afinidad crea clústeres enviando mensajes entre pares de muestras hasta la convergencia, luego se describe un conjunto de datos utilizando un pequeño número de ejemplares, que se identifican como los más representativos de otras muestras. Los mensajes enviados entre pares representan la idoneidad para que una muestra sea el ejemplar de la otra, que se actualiza en respuesta a los valores de otros pares. Esta actualización ocurre de manera iterativa hasta la convergencia, momento en el que se eligen los ejemplares finales y, por lo tanto, se proporciona la agrupación final.

La propagación de afinidad puede retorna la cantidad de clústeres en función de los datos proporcionados, para este propósito, los dos parámetros importantes son:

- **preference**, el cual controla cuántos ejemplares se utilizan
- **damping (factor de amortiguamiento)** evita oscilaciones numéricas.

Podemos modificar un poco nuestro código anterior para utilizarlo con este método y comparar los resultados obtenidos:

```
import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.cluster import AffinityPropagation
from sklearn import datasets
from sklearn import metrics
iris = datasets.load_iris()
datos = iris.data
etiquetas = iris.target

afinidad_por_propagacion = AffinityPropagation(preference=-50 , damping=0.5)

af = AffinityPropagation(preference=-50).fit(datos)
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_

n_clusters_ = len(cluster_centers_indices)

print('Estimated number of clusters: %d' % n_clusters_)

af.fit(datos)
predicciones=af.predict(datos)
score=metrics.adjusted_rand_score(etiquetas, predicciones)
print(score)
plt.scatter(datos[:, 0], datos[:, 1], c=predicciones)
```

```
plt.show()
```

A partir de los datos de iris, el modelo predice tres clusters con un porcentaje de acierto del 80%.

2. Pre-procesamiento de datos.

Re escalado de datos - MinMaxScaler().

Para representar mejor los datos es conveniente llevarlos a una escala común en el rango de cero a uno, para ello utilizaremos el método MinMaxScaler(), el cual permite comparar variables que pertenecen a diferentes distribuciones. Veamos un ejemplo muy simple antes de aplicarlo a nuestro trabajo. Partamos como siempre de importar las librerías necesarias:

```
import numpy as np

from sklearn import preprocessing
```

Y consideremos el siguiente array:

```
x = np.array([[1, 2,  3], [4,  5, 6], [7, 8, 9]])
```

Si ahora aplicamos el indicándole el rango de cero a uno y transformamos los datos del array obtenemos:

```
minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))

minmax.fit(x).transform(x)
```

Resultado:

```
array([[0. , 0. , 0. ],
       [0.5, 0.5, 0.5],
       [1. , 1. , 1. ]])
```

Internamente se está realizando el siguiente proceso de escalado:

$$x_{escalado} = \frac{(x - x_{min})}{(x_{max} - x_{min})}$$

Lo cual es como realizar:

```
#Para primera columna
x_std_0 = (1-1)/(7-1)
print(x_std_0)
x_std_4 = (4-1)/(7-1)
print(x_std_4)
x_std_7 = (7-1)/(7-1)
print(x_std_7)
```

Resultado:

```
0.0
0.5
1.0
```

Estandarización o eliminación de la media.

La estandarización o eliminación de la media es una técnica que simplemente centra los datos eliminando el valor promedio de cada característica, y luego la escala dividiendo las características no constantes por su desviación estándar. Por lo general, es beneficioso eliminar la media de cada función para que se centre en cero. Esto nos ayuda a eliminar el sesgo de las características. La fórmula utilizada para lograrlo es la siguiente:

$$x_{escalado} = \frac{x - media}{desviación\ standard}$$

En sklearn podemos utilizar el método StandardScaler() para obtener el array resultante:

```
import numpy as np

from sklearn import preprocessing

x = np.array([[1, 2,  3], [4,  5, 6], [7, 8, 9]])


standar = preprocessing.StandardScaler().fit(x)

standar.transform(x)
```

Resultado

```
array([[-1.22474487, -1.22474487, -1.22474487],
       [ 0.        ,  0.        ,  0.        ],
       [ 1.22474487,  1.22474487,  1.22474487]])
```

Si lo hiciéramos a mano lo podríamos realizar de la siguiente manera:

```
x_std_0 = (1-4)/np.std(x[:,0])

print(x_std_0)

x_std_4 = (4-4)/np.std(x[:,0])

print(x_std_4)

x_std_7 = (7-4)/np.std(x[:,0])

print(x_std_7)
```

Obteniendo los resultados:

-1.2247448713915892

0.0

1.2247448713915892

Normalización

La normalización de datos es otra forma de escalar los datos, en donde se ajustan los valores de un vector de características para que sumen 1. La norma es una función que asigna una longitud positiva a cada vector que pertenece a un espacio vectorial, excepto 0.

Para normalizar los datos en sklearn, se utiliza la función **normalize()**. Esta función escala los vectores de entrada individualmente a un vector unidad. Se proporcionan tres tipos de normas, l1, l2 o max. Si x es el vector de covariables de longitud n, el vector normalizado es y = x / z, donde z se define de la siguiente manera:

$$l1: z = \sum_{i}^{n} \left| x_i \right|$$

$$l2: z = \sqrt{\sum_{i}^{n} x_i^2}$$

$$max: z = max(x_i)$$

```
datos_n = preprocessing.normalize(x, norm='l1', axis=0)

datos_n
```

Resultado:

```
array([[0.08333333, 0.13333333, 0.16666667],
       [0.33333333, 0.33333333, 0.33333333],
       [0.58333333, 0.53333333, 0.5       ]])
```

Binarización

La binarización se utiliza cuando se desea convertir un vector de características numéricas en un vector booleano, es muy utilizado en el campo del procesamiento de imágenes digitales. La binarización de imágenes es el proceso mediante el cual una imagen en color o en escala de grises se transforma en una imagen binaria, es decir, una imagen con solo dos colores (generalmente, blanco y negro).

Al tomar nuestro array x, nos quedaría:

```
datosb = preprocessing.Binarizer(threshold=6).transform(x)

datosb
```

Resultado:

```
array([[0, 0, 0],
       [0, 0, 0],
       [1, 1, 1]])
```

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/

Lo que vimos

En esta unidad hemos trabajado con modelos de aprendizaje no supervisado.

Lo que viene:

En la siguiente unidad seguiremos trabajando con sklearn.