APRENDIZAJE AUTOMÁTICO

2° AÑO

Clase Nº 7: Aprendizaje No Supervisado

Contenido: Agrupación de datos sin etiquetas (Clustering).

En la clase de hoy trabajaremos los siguientes temas:

- Agrupación de datos sin etiquetas (Clustering):
  - K-means
  - MiniBatchKMeans
  - Affinity Propagation.

1. Presentación:

Bienvenidos a la clase N° 7 de Aprendizaje Automático.

En esta emocionante jornada de aprendizaje, exploraremos uno de los pilares fundamentales del análisis de datos. Nos sumergiremos en el mundo de la Agrupación de Datos sin Etiquetas, donde descubriremos cómo algoritmos como K-means, MiniBatchKMeans y Affinity Propagation pueden revelar estructuras ocultas en nuestros datos. Imagina aplicar K-means para segmentar clientes en grupos con patrones de compra similares, MiniBatchKMeans para acelerar el procesamiento en conjuntos de datos masivos o Affinity Propagation para identificar automáticamente comunidades en redes sociales. ¡Prepárense para una clase repleta de conocimientos prácticos y aplicaciones emocionantes!

1. Desarrollo y Actividades:

Agrupación de datos sin etiquetas (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:

$$\sum_{i=0}^{n} min\left(\|x_i - \mu_j\|^2\right) \quad \text{con } \mu_j \in C$$

La *inercia* se puede tomar como una medida de cuán 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_init='auto', n_clusters=3, max_iter=2000)
```

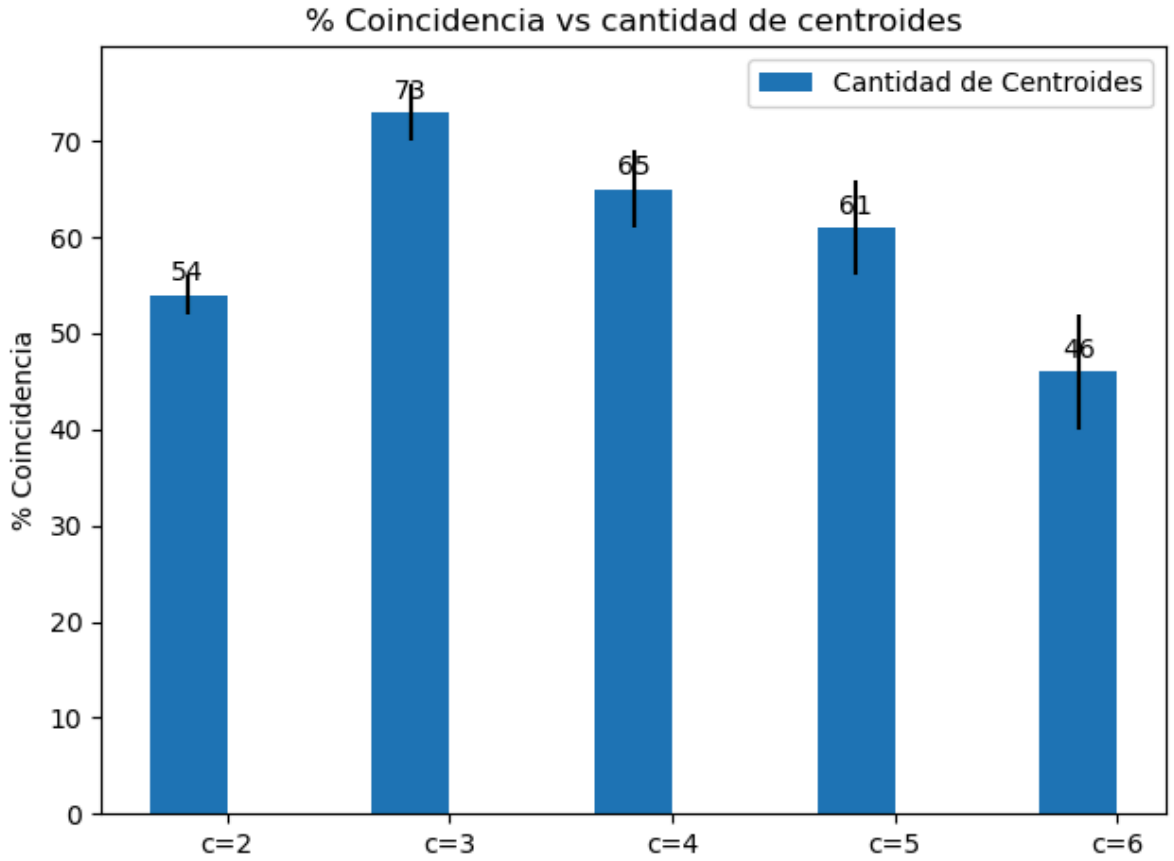
Luego llenamos el modelo con los datos y obtenemos las predicciones realizadas por el modelo:

```
k_means.fit(datos)
#predicciones del grupo al que cree que pertenece
predicciones=k_means.predict(datos)
```

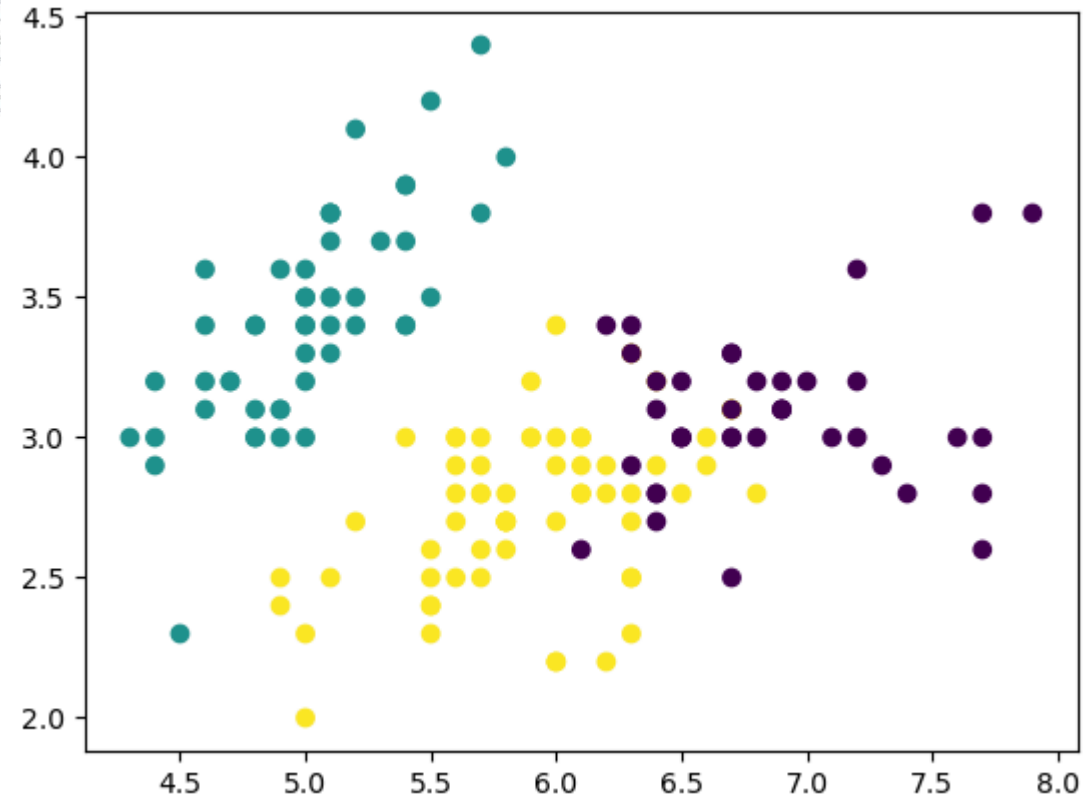
Dado que para iris tenemos ya la clasificación, podemos evaluar cuán preciso ha sido el método de clasificación.

```
score=metrics.adjusted_rand_score(etiquetas, predicciones)
```

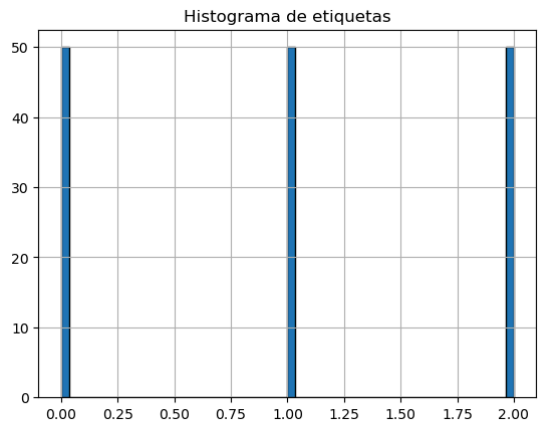
En este caso nos ha dado para 3 centroides como esperábamos un porcentaje de coincidencia del 73%. Si graficamos los porcentajes de coincidencia en función de la cantidad de centroides obtenemos el siguiente histograma:

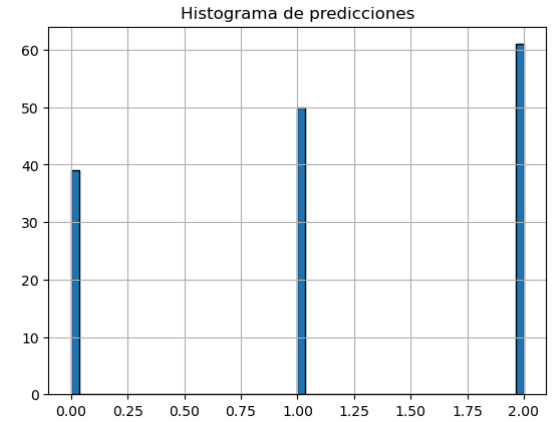


Observamos como la clasificación obtiene un mayor valor de aciertos al considerar tres centroides, podemos ver la representación de la clasificación realizada al graficar una de las columnas de datos en función de la predicción para el caso de tres centroides.



Al realizar un histograma de la cantidad de valores obtenidos para cada etiqueta según la predicción, vemos que al modelo le ha costado diferenciar más 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.

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, random_state=None)

af = afinidad_por_propagacion.fit(datos)
cluster_centers_indices = af.cluster_centers_indices_
labels = af.labels_

n_clusters_ = len(cluster_centers_indices)

print('Número estimado de 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%.

Instancia de Autoevaluación:

En esta instancia de autoevaluación te pedimos que puedas poner en juego lo que has aprendido hasta este momento.Recordá que siempre podrás volver a la teoría presentada en esta clase o hacer las preguntas pertinentes en los encuentros sincrónicos o las tutorías presenciales.

Es importante tener en cuenta que para aprobar esta instancia deberás:

- Entregar en tiempo y forma
- Realizar un 60% de la actividad correctamente.
- Cumplir con las consignas
- Cumplir con el formato requerido.

2. Actividad Integradora de Cierre:

En esta actividad integradora de cierre vamos a poner en juego todos los conocimientos que hemos recorrido juntos en esta clase. Tendremos en cuenta los tipos de ejercicios que realizamos la teoría que leímos y los ejemplos que vimos en los diferentes recursos.

Vamos a realizar una *Agrupar usuarios Twitter de acuerdo a su personalidad con K-means*. En el ejercicio utilizaremos de entradas un conjunto de datos que se encuentra disponible en un proyecto de Kaggle, en el que se analizan rasgos de la personalidad de usuarios de Twitter. Se han filtrado a 140 “famosos” del mundo en diferentes áreas: deporte, cantantes, actores, etc. Basado en una metodología de psicología conocida como “Ocean: The Big Five” tendemos como características de entrada:

- usuario (el nombre en Twitter)
- “op” = Openness to experience – grado de apertura mental a nuevas experiencias, curiosidad, arte
- “co” =Conscientiousness – grado de orden, prolijidad, organización
- “ex” = Extraversion – grado de timidez, solitario o participación ante el grupo social
- “ag” = Agreeableness – grado de empatía con los demás, temperamento
- “ne” = Neuroticism, – grado de neuroticismo, nervioso, irritabilidad, seguridad en sí mismo.
- Wordcount – Cantidad promedio de palabras usadas en sus tweets

- Categoría – Actividad laboral del usuario (actor, cantante, etc.)

Utilizaremos el algoritmo K-means para que agrupe estos usuarios -no por su actividad laboral- si no, por sus similitudes en la personalidad. Una de las hipótesis que podríamos tener es: "Todos los cantantes tendrán personalidad parecida" (y así con cada rubro laboral). Pues veremos si lo probamos, o por el contrario, los grupos no están relacionados necesariamente con la actividad de estas Celebridades. Expresa tus conclusiones en el Foro.

Para eso accederemos al proyecto pulsa en el siguiente link: Agrupar usuarios en Twitter de acuerdo a su personalidad con k-means

3. Cierre:

En el cierre de esta clase enriquecedora, hemos explorado las profundidades del Aprendizaje Automático y el Preprocesamiento de Datos, adquiriendo un conjunto valioso de herramientas y conocimientos. Hemos descubierto que la Agrupación de Datos sin Etiquetas, a través de algoritmos como K-means, MiniBatchKMeans y Affinity Propagation, nos permite desentrañar patrones subyacentes en nuestros datos, revelando insights ocultos que pueden impulsar la toma de decisiones en campos tan diversos como la segmentación de clientes, la identificación de comunidades en redes sociales o la detección de anomalías en sistemas. Este conocimiento es fundamental en un mundo donde los datos son la moneda más valiosa. En última instancia, comprender y aplicar estas técnicas no solo nos capacita para enfrentar desafíos complejos en el análisis de datos, sino que también nos empodera para tomar decisiones informadas que impulsan la innovación y el éxito en un amplio espectro de aplicaciones en la era de la información.

Te invitamos a que nos dejes tus impresiones sobre lo que aprendimos hoy en el foro. También podés compartir links de artículos o videos que te hayan parecido interesantes mientras estudiabas este tema.

Te esperamos en los próximos encuentros. Recordá que es importante participar de los foros, de los encuentros sincrónicos y en lo posible de las tutorías presenciales.

Para finalizar te dejamos estos recursos que pueden ayudarte a entender el tema un poco más.

Sitios

"scikit-learn." *scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation*, https://scikit-learn.org/stable/. Accedido el 23 Junio del 2023.

4. Bibliografía Obligatoria:

- HAN, R. (2019): *Matemáticas del Aprendizaje Automático*. *Introducción a la analítica de datos e Inteligencia Artificial*.
- JONES, H. (2019): *Aprendizaje Automático. El Aprendizaje Automático para principiantes que desean comprender aplicaciones, Inteligencia Artificial, Minería de Datos, Big Data y más*. Editorial Edición Kindle.
- RUSSELL, R. (2018): *Machine Learning. Guia Paso a Paso Para Implementar Algoritmos De Machine Learning Con Python*
- VIDALES, A. (2019) *Machine Learning Con Matlab*. Editorial Edición Kindle.
- CANE, A. (2019): *Programación Con Python. Guía completa para principiantes.*
- HERNÁNDEZ, V. (2019): *Aprendizaje Automático. Implementación en Inteligencia Artificial.* Editorial Edición Kindle.