(https://www.bigdatauniversity.com)

K-Medias Clustering

Introduction

Existen muchos modelos para **clustering**. En este lab, estaremos presentando el modelo que es considerado el más simple de todos. Aunque simple, el **K-medias** se utiliza mucho para clustering en varias aplicaciones de ciencia de datos, especialmente si se necesita de forma rápida descubrir nuevas conclusiones a partir de datos **no etiquetados**. En este lab, aprenderá a utilizar k-Medias para segmentar clientes.

Algunas aplicaciones de k-medias del mundo real:

- Segmentacion del cliente
- Entender lo que los visitantes de un sitio web intentan realizar
- Reconocimiento de patrones
- Machine learning
- Compresión de datos

En este lab practicamos clustering k-medias con 2 ejemplos:

- k-means en un dataset generado al azar
- Usando k-medias para la segmentación del cliente

Importar librerias

Importemos primero las librerías que se necesitan. Ejecutemos **%matplotlib inline** para realizar los trazados en esta sección.

In [1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

k-Media en un dataset generado aleatoriamente

¡Creemos nuestro propio dataseet para este lab!

Necesitamos primero configurar una semilla aleatoria(random seed). Utilizaremos la función **numpy's random.seed()**, donde la semilla se establecerá con el valor **0**

In [5]:
```
np.random.seed(0)
```

Luego, haremos *clusters aleatorios* de puntos usando la clase **make_blobs** . La clase **make_blobs** puede aceptar varias entradas, pero estaremos usando concretamente estas.

Entrada

- **n_samples**: El número total de puntos equitativamente divididos entre los clusters.
  - El valor será: 5000
- **centers**: El número de centros a generar.
  - El valor será: [[4, 4], [-2, -1], [2, -3],[1,1]]
- **cluster_std**: El desvío estándar de los clusters.
  - El valor será: 0.9

Salida

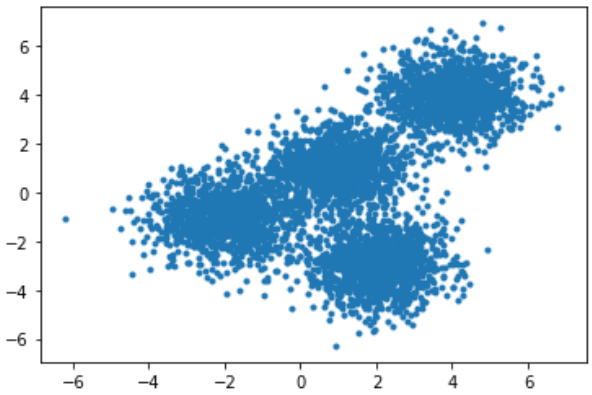
- **X**: Arreglo de la forma [n_samples, n_features]. (Matríz de Distancia)
  - Muestras generadas.
- **y**: Arreglo de la forma [n_samples]. (Response Vector)
  - Las etiquetas de números enteros para la pertenencia de cluster en cada muestra.

In [10]:
```
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1,
1]], cluster_std=0.9)
```

Mostrar los puntos de los datos generados al azar.

In [11]:
```
plt.scatter(X[:, 0], X[:, 1], marker='.')
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x7fd86c45cb70>
```



Estableciendo K-Medias

Ahora que tenemos nuestros datos aleatorios, configuremos nuestro Clustering K-Medias.

La clase KMedias tiene muchos parámetros que se pueden utilizar, pero estaremos usando estos tres:

- **init**: Método de inicialización de los centroides.
  - El valor será: "k-means++"
  - k-means++: Elije centros de clusters iniciales eficientes para el clustering k-media de forma tal de acelerar la convergencia.
- **n_clusters**: El número de clusters a formar y la cantidad de centroides a generar.
  - El valor será: 4 (tenemos 4 centros)
- **n_init**: Cantidad de veces que el algoritmo k-medias se ejecutará con diferentes semillas centroides. El resultado final será la mejor salida de consecutivas ejecuciones de n_init en términos de inercia.
  - Value will be: 12

Inicializar KMedias con estos parámetros, donde el parámetro de salida se llama **k_means**.

In [12]:
```
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```

Ahora, unamos el modelo KMedias con la matriz de distancia que creamos anteriormente, **X**

In [13]:
```
k_means.fit(X)
```

Out[13]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=12, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

Ahora juntemos las etiquetas de cada punto en el modelo usando el atributo de KMedias **.labels_** y lo guardamos como **k_means_labels**

In [14]:
```
k_means_labels = k_means.labels_
k_means_labels
```

Out[14]:
```
array([3, 3, 0, ..., 3, 0, 0], dtype=int32)
```

También obtendremos las coordenadas de los centros del cluster usando KMedias **.cluster_centers_** y guardémoslo en **k_means_cluster_centers**

In [15]:
```
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```

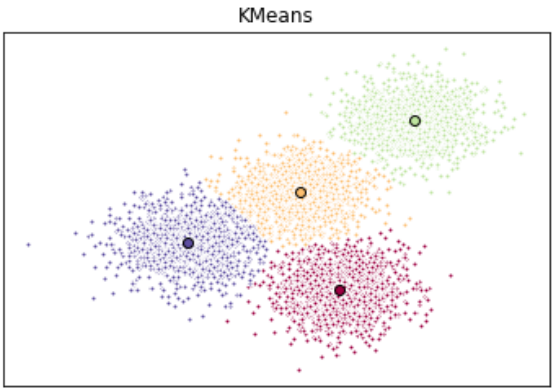
Out[15]:
```
array([[ 2.01712402, -2.98474324],
       [ 0.97468147,  1.02839633],
       [ 4.00130768,  3.9898029 ],
       [-1.99025056, -1.06295457]])
```

Creando la Trama

En este momento, tenemos los datos generados al azar y el modelo KMedias inicializado. Podemos dibujarlos y ver de qué se trata!

Favor leer todo el código y los comentarios para entender cómo dibujar el modelo.

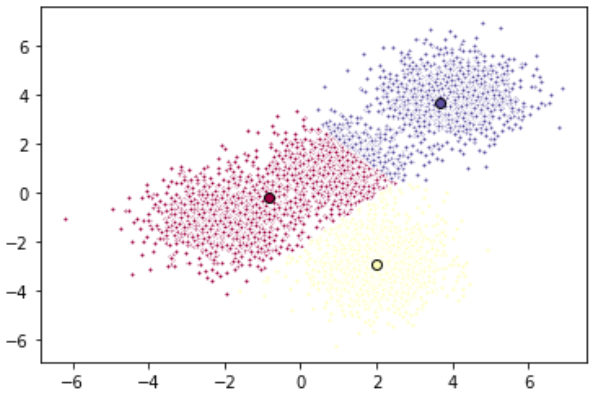
In [16]:
```
# Inicializar el dibujo con las dimensiones especificadas.
fig = plt.figure(figsize=(6, 4))

# Los colores usan un mapa de color, dónde produciremos un arreglo de colore
s basados en
# el número de etiquetas que hay. Usaremos set(k_means_labels) para obtener
# etiquetas unívocas.
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels))))

# Crear un dibujo
ax = fig.add_subplot(1, 1, 1)

# Loop For que dibuja los puntos de datos y los centroides.
# k tomará valores entre 0-3, los cuales coincidirán con los clusters posibl
es en el
# que está cada punto.
for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):

    # Crear una lista de todos los puntos, donde aquellos que están
    # en el cluster (ej. cluster 0) están etiquetados como verdadero, o en s
u defecto
    # estarán etiquetados como falso.
    my_members = (k_means_labels == k)

    # Definir el centroide o centro del cluster.
    cluster_center = k_means_cluster_centers[k]

    # Dibjuar los puntos de datos con color col.
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, ma
rker='.')

    # Dibujo de los centroides con un color específico pero una linea más os
cura
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
markeredgecolor='k', markersize=6)

# Título del dibujo
ax.set_title('KMeans')

# Eliminar los ticks del eje x
ax.set_xticks(())

# Eliminar los ticks del eje y
ax.set_yticks(())

# Mostrar el dibujo
plt.show()
```



Práctica

Probar de agrupar el set de datos en 3 clusters
Nota: no vuelvas a generar los datos utiliza el mismo dataset de ántes.

In [28]:
```
# escribe tu código aquí
import matplotlib.pyplot as plt
# Initializing the K means algorithm with 3 clusters
k_means_3_clusters = KMeans(init = "k-means++", n_clusters = 3, n_init = 12)
# Fitting the data to the algorithm
k_means_3_clusters.fit(X)
# Labels of each point
k_means_3_clusters_labels = k_means_3_clusters.labels_
# Coordinates of cluster centers
k_means_3_clusters_cluster_centers = k_means_3_clusters.cluster_centers_

# Plot
fig = plt.figure(figsize=(6, 4))
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_3_clusters_label
s))))
ax = fig.add_subplot(1, 1, 1)
for k, col in zip(range(len(k_means_3_clusters_cluster_centers)), colors):
    my_members = (k_means_3_clusters_labels == k)
    cluster_center = k_means_3_clusters_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, ma
rker='.')
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
markeredgecolor='k', markersize=6)
plt.show()
```



Haz doble click **aquí** para la solución.

Segmentación de Cliente con K-Medias

Imagine que tiene un conjunto de datos de clientes y tiene que aplicar la segmentación de clientes en estos datos históricos. La segmentación de clientes es la práctica de particionar una base de clientes en particiones de individuos que tengan características similares. Es una estrategia significativa ya que un negocio puede poner el foco en estos grupos específicos de clientes y ubicar los recursos de marketing lo más eficientemente posible. Por ejemplo, un grupo podría tener clientes con alto ingreso y bajo riesgo, lo que quiere decir que es muy probable adquieran productos, o se suscriban para un servicio. Una de las tareas del negocio es retener este tipo de clientes. Otro grupo podría incluir clientes de organizacion sin fines de lucro y asi seguiríamos con más ejemplos.

Descarguemos el set de datos. Para descargarlo, utilizaremos **`!wget`** de IBM Object Storage.
¿Sabías? Cuando se trata de Machine Learning, seguro trabajarás con grandes datasets (juego de datos). Entonces, ¿dónde podrás guardar esos datos? IBM ofrece una oportunidad única para las empresas, con 10 Tb de IBM Cloud Object Storage: Registrate ahora gratuitamente (http://cocl.us/ML0101EN-IBM-Offer-CC)

In [ ]:
```
!wget -O Cust_Segmentation.csv https://s3-api.us-geo.objectstorage.softlaye
r.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/Cust_Segmentation.csv
```

Cargar los Datos Desde un Archivo CSV

Antes de trabajar con los datos, se deberá usar la URL para obtener el archivo Cust_Segmentation.csv.

In [ ]:
```
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

Pre-procesamiento

Como podrás ver, **Address** en este set de datos es una variable categórica. El algoritmo k-medias no está directamente aplicado a variables categóricas porque la función de la distancia Euclediana no tiene sentido para variables discretas. Por lo que descartaremos esta característica y seguiremos adelante para correr el clustering.

In [ ]:
```
df = cust_df.drop('Address', axis=1)
df.head()
```

Normalizando el desvío estándar

Ahora normalicemos el set de datos. Pero, ¿por que necesitamos normalizar? La normalización es un método estadístico que ayuda a los algoritmos basados en matemática interpretar características con distintas magnitudes y distribuciones de manera igual. Usamos **tandardScaler()** para normalizar nuestros set de datos.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Modeling

En nuestro ejemplo (en caso no hayamos tenido acceso al algoritmo de k-medias), sería lo mismo que suponer que cada grupo de cliente tendría cierta edad, educación, etc., con muchas pruebas y experimentos. Sin embargo, usando k-medias clustering, podemos hacer todo este proceso mucho más facilmente.

Apliquemos k-medias en nuestro set de datos y miremos las etiquetas del cluster.

In [ ]:
```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

Descubrimientos

Asignamos las etiquetas a cada fila dentro del marco de datos.

In [ ]:
```
df["Clus_km"] = labels
df.head(5)
```

Podemos revisar fácilmente los valores centroides sacando el promedio de las características de cada cluster.

In [ ]:
```
df.groupby('Clus_km').mean()
```

Ahora, miremos la distribuición de los clientes basados en su edad e ingreso:

In [ ]:
```
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)

plt.show()
```

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
# plt.ylabel('Age', fontsize=18)
# plt.xlabel('Income', fontsize=16)
# plt.zlabel('Education', fontsize=16)
ax.set_xlabel('Education')
ax.set_ylabel('Age')
ax.set_zlabel('Income')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

k-medias particionará los clientes en grupos mutuamente excluyentes, por ejemplo, en 3 clusters. Los clientes de cada cluster son parecidos unos a otros en el aspecto demográfico. Ahora, podemos crear un perfil para cada grupo, teniendo en cuenta las características en común de cada cluster. Por ejemplo, los 3 clusters podrían ser:

- AFLUENTE, EDUCADO Y TERCERA EDAD
- EDAD MEDIA E INGRESO PROMEDIO
- JOVEN E INGRESO BAJO

¿Querés aprender más?

IBM SPSS Modeler es una plataforma para analytics que contiene varios algoritmos de machine learning. Fue diseñada para acercar inteligencia predictiva a las decisiones hechas por individuos, grupos, sistemas, toda la empresa. Un free trial está disponible a través de este curso en: SPSS Modeler (http://cocl.us/ML0101EN-SPSSModeler).

Asi mismo, puedes utilizar Watson Studio para ejecutar estos notebooks más rápido y con datasets más grandes. Watson Studio es una solución en la nube lider de IBM's para científicos de datos, construída por científicos de datos. Con Jupyter notebooks, RStudio, Apache Spark y librerías conocidas pre instaladas en la nube, Watson Studio posibilita a los científicos de datos colaborar en sus proyectos sin tener que instalar nada. Sumate a la comunidad de usuarios Watson Studio hoy mismo por medio de una cuenta gratuita en Watson Studio (https://cocl.us/ML0101EN_DSX)

¡Gracias por completar esta lección!

Laboratorio creado por: Saeed Aghabozorgi (https://ca.linkedin.com/in/saeedaghabozorgi)
