UNIVERSIDAD ANDRÉS BELLO

FACULTAD DE INGENIERÍA

INGENIERÍA CIVIL INFORMÁTICA

Trabajo 3 – Aprendizaje No-Supervisado (K-Means/DBSCAN)

ANDRÉS EDUARDO VALENZUELA GONZÁLEZ

SANTIAGO - CHILE

OCTUBRE, 2017

Contenido

1. Introducción

Para comprender el contexto del siguiente informe, se utilizaron los dos mil (2000) perfiles vectorizados en la tarea anterior para hacer uso del algoritmo *K-Means* provisto por el programa proporcionado por el sitio *http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/yakmo/* .

Antes de comenzar la ejecución, es necesario entender la estructura de los vectores recibidos por el programa (más detalles en https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Informe%20II.pdf , punto 2):

Para realizar el aprendizaje no-supervisado se debió modelar el problema mediante vectores. Si asumimos que cada una de las palabras es un atributo, entonces podríamos crear una representación vectorial que indique la presencia o ausencia de esta palabra en cada una de las descripciones del usuario etiquetadas en la primera tarea. Para la etiqueta se utilizará un valor desde 1 hasta 4:

1 = *Undetermined* 2 = *Non-USA* 3 = *World* 4 = *USA only*.

Cada etiqueta debe estar asociada con un único valor numérico. *Extracto del enunciado Tarea 2, Espacio Vectorial, 28 de agosto 2017 [consulta: 11 octubre 2017, 12:32 hrs]. Disponible en:* https://gitlab.com/Choapinus/SistemasInteligentes/blob/master/Tarea%202/enunciado_informe/Tarea%202.pdf

2. Ejecución

Para explicar el proceso de aprendizaje, se considera usar el siguiente comando con el programa *yakmo*:

./yakmo –k 4 vectores.txt - - -O 1

Donde:

- El archivo *vectores.txt* contiene nuestros vectores creados con anterioridad.
- K representa el numero de vecinos.
- *-O 1* indica que la salida seran los datos dentro de cluster.

2.1. Errores

Este metodo de ejecucion condujo a errores ya que los datos dentro del archivo *vectores.txt* estaban de cierta manera ordenados. Tal orden afectó en gran medida la salida del programa *yakmo*, dando origen a los primeros tres centroides solo con un dato y el ultimo con el resto de ellos como muestra la siguiente imagen.

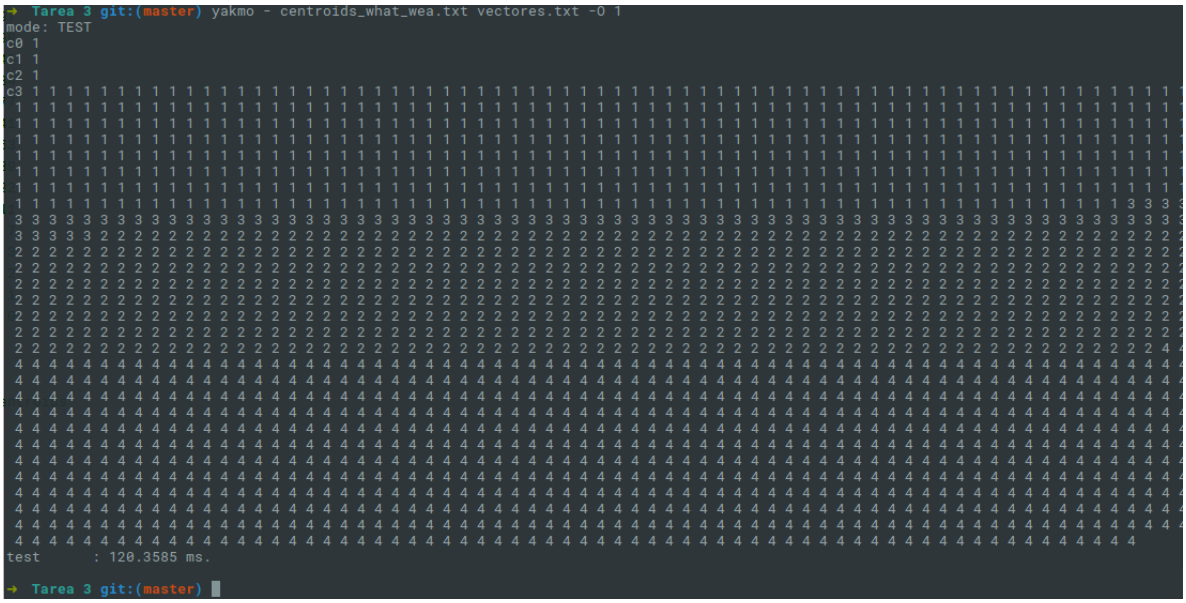


Ilustración 1 - Centroides

En la ilustración 1 se puede apreciar que se evalua el modelo generado por la aplicacion *yakmo* para recrear los centroides con base en el archivo *vectores.txt.* De esto se puede inferir que la gran mayoria de los datos cayo dentro de la clasificacion del centroide 4 (c3) y los tres datos restantes fueron ***outliers***, o bien la gran mayoria de datos fueron ***outliers*** que cayeron dentro del centroide 4 (c3) y los bien clasificados fueron los primeros tres datos (tiene mas sentido la primera conclusión ya que todos los ejemplos que siguieron al cuarto perfil estaban mas cerca de este).

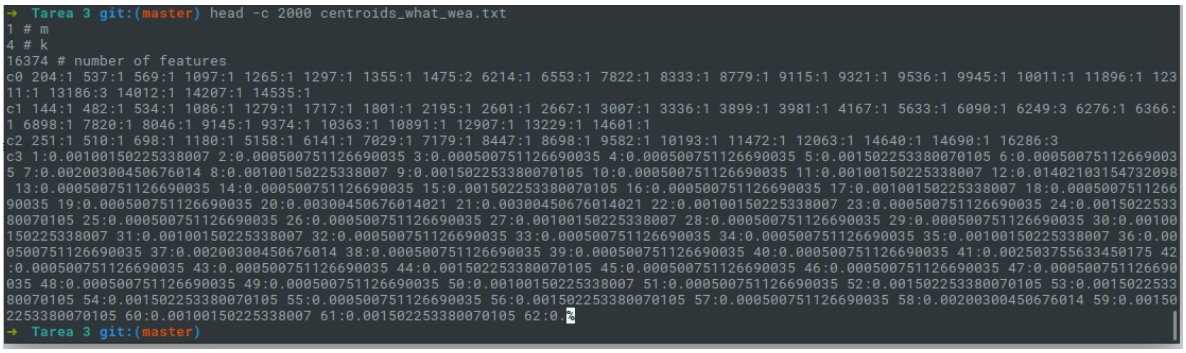


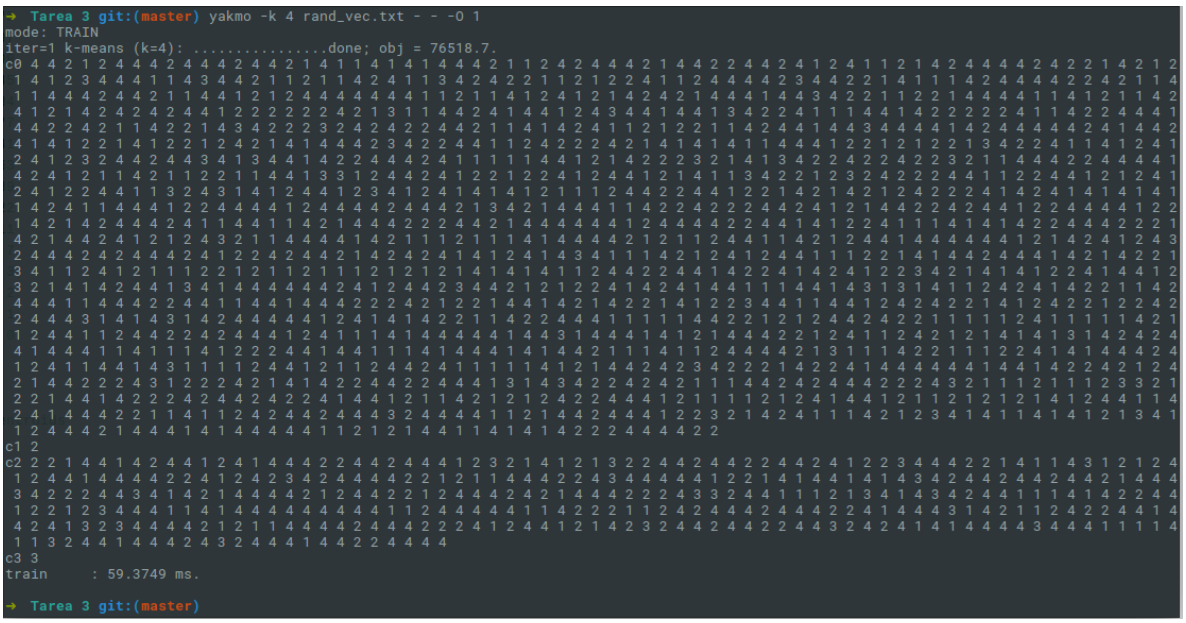
Ilustración 2 - Primer Modelo Generado

2.2. Solución propuesta

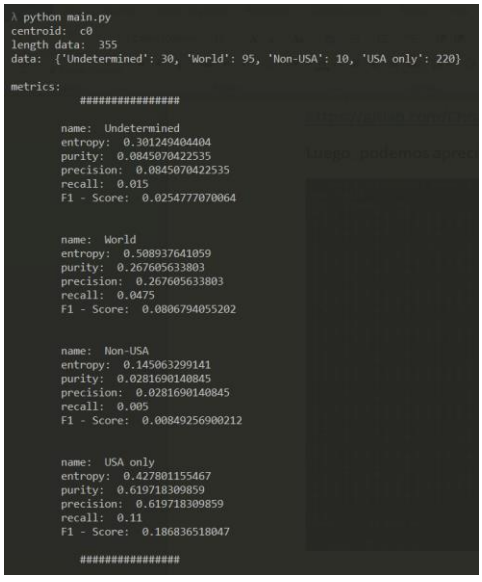
Como solución se propuso generar un *random input* a partir de los vectores originales. Para realizar esto, se creó un programa en *Python* el cual toma los vectores originales, randomiza su orden, ejecuta el programa *yakmo* y verifica que los clusters tengan un mínimo de 150 elementos y un máximo de 1300. De no ser así, se repite el proceso de randomizacion.

La implementación de este programa puede ser encontrada en
https://gitlab.com/Choapinus/SistemasInteligentes/tree/master/Tarea%203 .

Luego, podemos apreciar los nuevos centroides generados con una entrada de datos aleatoria:



Además, el programa es capaz de recrear los clusters de una manera mas interpretable y con todas sus metricas de cada clase contenida:



3. Valores mas representativos

Dentro de cada cluster se pueden ubicar los 10 valores más representativos (10 primeros):

1. **Cluster 1** (c0): [**4**, **4**, 3, **4**, **4**, **4**, **4**, **4**, 2, **4**] -> etiqueta 4 con más repeticiones
2. **Cluster 2** (c1): [1, **4**, 1, 1, **4**, 2, 3, **4**, **4**, 2] -> etiqueta 4 con más repeticiones
3. **Cluster 3** (c2): [**1**, 4, 2, **1**, 4, **1**, **1**, 4, **1**, 2] -> etiqueta 1 con más repeticiones
4. **Cluster 4** (c3): [**4**, 2, **4**, 1, 2, 1, **4**, **4**, **4**, 2] -> etiqueta 4 con más repeticiones

Como se puede observar, la etiqueta 4 fue la que más repeticiones obtuvo en 3 de 4 clusters. Esto pudo haberse dado por distintos motivos, ya sean estos el orden de llegada de los vectores, la implementación del algoritmo, la métrica de distancia utilizada, etc.

4. Cálculo y análisis de resultados

Una vez interpretados los clusters y sus datos contenidos, se pueden realizar diversos cálculos tales como la precisión, el recall, F1-score, entropías y puritys respectivos.

1. Para calcular la entropía, se utilizara la siguiente fórmula propuesta por las ponencias del curso *Sistemas Inteligentes*:
   Sea | D | el total de datos del dataset, | Di | la cantidad de datos contenidos por cluster y cj la cantidad de datos por clase

$$entropy(D_i) = -\sum_{j=1}^{k} \Pr_i(c_j) \log_2 \Pr_i(c_j)$$

Ilustración 3 - Entropía por clase

$$entropy_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times entropy(D_i)$$

Ilustración 4 - Entropía total

2. Para calcular la pureza *(purity)* se utilizaron las siguientes formulas:

$$purity(D_i) = \max_j (\Pr_i(c_j))$$

Ilustración 5 - Purity cluster

$$purity_{total}(D) = \sum_{i=1}^{k} \frac{|D_i|}{|D|} \times purity(D_i)$$

Ilustración 6 - Purity total

3. Para calcular la precisión, el recall y el F1 – Score de cada cluster se tiene:

$$Recall = \frac{\max_j(c_j)}{\sum_{i=1}^{k} c_i}$$

$$Precision = \frac{\max_j(c_j)}{total\ etiquetas}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

4.1. Cluster 1 (c0):

	Undetermined	World	Non-USA	USA only
Precisión	*0.0845070422535*	*0.267605633803*	*0.0281690140845*	*0.619718309859*
Recall	*0.015*	*0.0475*	*0.005*	*0.11*
F1-score	*0.0254777070064*	*0.0806794055202*	*0.00849256900212*	*0.186836518047*
Entropía	*0.301249404404*	*0.508937641059*	*0.145063299141*	*0.427801155467*
Purity	*0.0845070422535*	*0.267605633803*	*0.0281690140845*	*0.619718309859*
Total datos	30	95	10	220

4.2. Cluster 2 (c1):

	Undetermined	World	Non-USA	USA only
Precisión	*0.195402298851*	*0.256704980843*	*0.0766283524904*	*0.471264367816*
Recall	*0.0255*	*0.0335*	*0.01*	*0.0615*
F1-score	*0.0451127819549*	*0.0592658115878*	*0.0176912870411*	*0.108801415303*
Entropía	*0.460266334807*	*0.50360814563*	*0.283982980972*	*0.511506334948*
Purity	*0.195402298851*	*0.256704980843*	*0.0766283524904*	*0.471264367816*
Total datos	51	67	20	123

4.3. Cluster 3 (c2):

	Undetermined	World	Non-USA	USA only
Precisión	*0.269230769231*	*0.263736263736*	*0.0274725274725*	*0.43956043956*
Recall	*0.0245*	*0.024*	*0.0025*	*0.04*
F1-score	*0.044912923923*	*0.0439963336389*	*0.00458295142071*	*0.0733272227314*
Entropía	*0.509676675869*	*0.507120564258*	*0.142468861135*	*0.521260019917*
Purity	*0.269230769231*	*0.263736263736*	*0.0274725274725*	*0.43956043956*
Total datos	49	48	5	80

4.4. Cluster 4 (c3):

	Undetermined	World	Non-USA	USA only
Precisión	*0.349417637271*	*0.278702163062*	*0.0357737104825*	*0.336106489185*
Recall	*0.21*	*0.1675*	*0.0215*	*0.202*
F1-score	*0.262336039975*	*0.209244222361*	*0.0268582136165*	*0.252342286071*
Entropía	*0.530058051979*	*0.513704912602*	*0.171891120065*	*0.528698767035*
Purity	*0.349417637271*	*0.278702163062*	*0.0357737104825*	*0.336106489185*
Total datos	420	335	43	404

Entropía total: 1.04835606386

Purity total: 0.601

4.5. Análisis de resultados

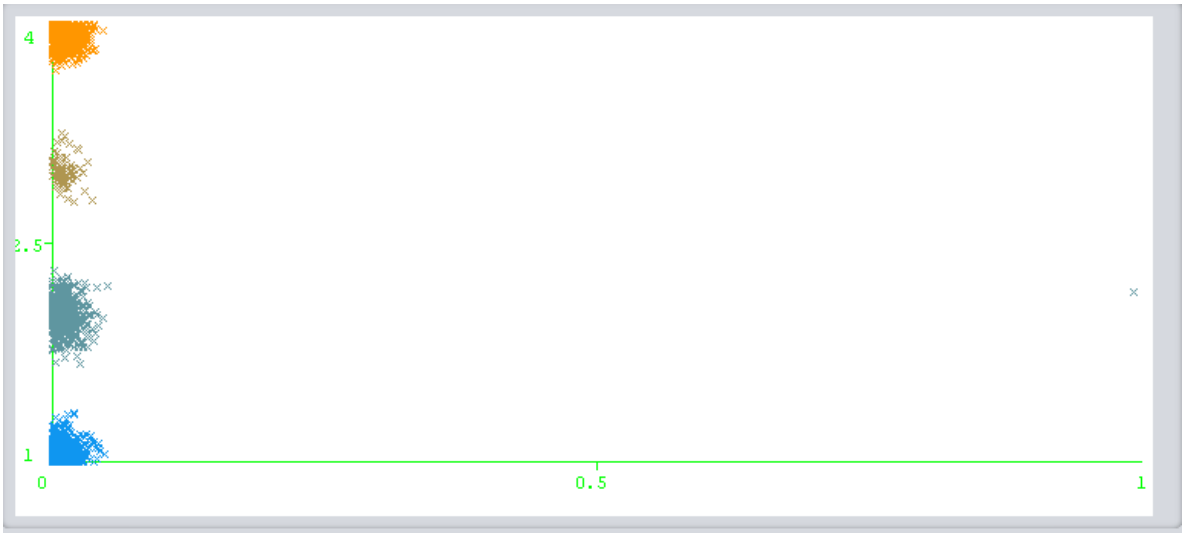
Sorpresivamente la mayoría de los datos fue anexada al cuarto cluster. Se puede observar además que todos los clusters contienen datos de las cuatro clases evaluadas, de esto se puede inferir que, y por teoría, todos los datos dentro de un cluster cumplen con alguna característica en común que los difiere de los demás clusters (en este caso, la distancia vectorial mínima cercana al *seed*, se estima que es euclidiana).

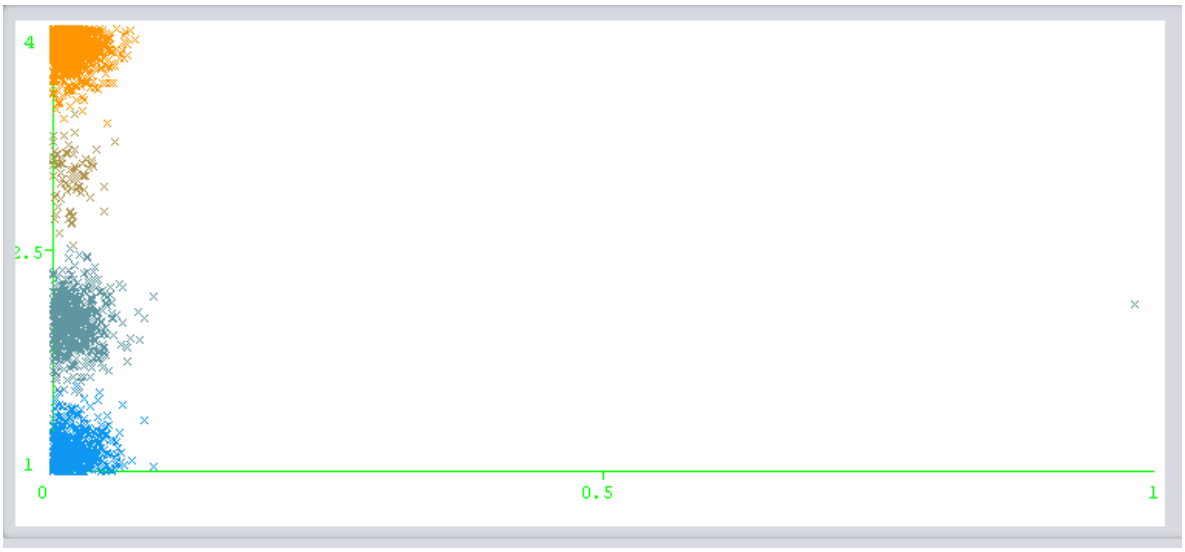
Por otro lado, los resultados varían en gran medida debido al orden de los datos de entrada. Se entiende que el *seed* escogido es distinto por cada entrada y por lo tanto los clusters difieren en gran medida, por lo tanto es improbable que se repitan exactamente las mismas métricas para cada cluster con cada iteración con entrada randomizada.

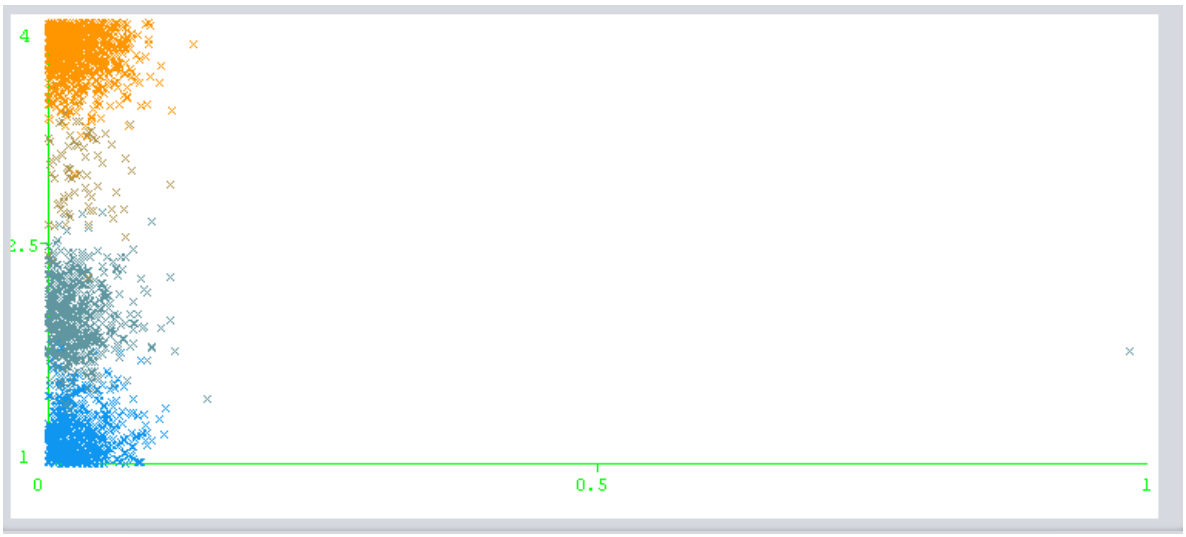
5. DBSCAN

Para esta sección se utilizó una implementación de DBSCAN distribuido por *Weka*, teniendo como entrada los perfiles vectorizados en *SVMLight Format* con orden randomizado. Lamentablemente no se pudo implementar el código sugerido desde el enunciado ya que se producía errores de referencias y tampoco se pudo obtener un resultado realista desde *Weka* ya que solo producía un cluster para todos los datos (incluso cambiando la función de distancia [euclidiana por defecto] y ajustando el épsilon a 6 unidades para obtener menos perfiles clasificados como “ruido”).

De todas maneras, resultaron gráficos de interés:







En estos gráficos se pueden apreciar ciertos *outliers*, que tan dispersos están los datos dentro de sus mismos conjuntos e incluso como pueden caer dentro del área otros clusters (cafés con naranjas y verdes con azules). Las clases se ordenan de abajo hacia arriba 1 = *Undetermined,* 2 = *Non-USA,* 3 = *World,* 4 = *USA only*.

6. Conclusiones

¿Qué observa?

En el programa *yakmo* se puede apreciar un algoritmo sensible a la entrada e impredecible en cuanto a salida, pero se obtiene un resultado deseable e interpretable bajo ciertos estándares (nivel de verbosidad solicitado).

La implementación del DBSCAN no se pudo concretar con éxito en su totalidad y los resultados no fueron realistas ni interpretables (excepto por los gráficos, pero estos solo interpretan los datos ingresados y no una salida por clusters). De todas formas el ruido da bastante para discutir, es decir, se puede destacar el bajo nivel de agrupamiento (o la gran magnitud de distancia) entre los datos, la similitud entre datos de distintas clases e incluso los posibles ***outliers***.

¿Hay relación entre los clústeres generado por K-means y DBSCAN?

Existen diversas características que hacen la distinción entre los clústeres generados por K-means y DBSCAN, por ejemplo la densidad utilizada por DBSCAN, la *épsilon*, los *minPoints* y la clasificación de datos como ruido. K-means resulta más simple dado a que posiciona los vectores solo por distancia desde cada cluster.

¿Qué algoritmo es mejor?

DBSCAN es mejor que K-means debido a que no es necesario que se le especifique una cantidad de *clusters* a priori, puede o no tomar en cuenta los datos ruidosos, hace la distinción entre el ruido de los datos reales basándose en medidas que uno le proporciona (épsilon) y su calidad de clusterización se verá incrementada (o reducida) dependiendo de la métrica de distancia utilizada debido a la “maldición de la dimensionalidad”.

En cambio, K-mean solo asigna datos a clusters basándose en las distancias de cada vector hacia cada centroide (por ser simple no significa que sea el mejor, además, el programa *yakmo* solo disponía de una métrica de distancia).