Proyecto 3

1st Barrios, Alonso
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
alonso.barrios@utec.edu.pe

2nd Rodriguez, Mauricio
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
mauricio.rodriguez@utec.edu.pe

3rd Tenazoa, Renzo
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
renzo.tenazoa@utec.edu.pe

***Abstract*—Este proyecto está desarrollado para poner a prueba lo aprendido acerca de los modelos de Clustering. El objetivo del proyecto es agrupar distintos puntos de un dataset, para predecir su clasificacón haciendo uso de distintos modelos de clusterting y analizar las graficas de error para definir cual es el que mejor resultado aporta.**

***Index Terms*—clustering, predicción, error, underfiting, overfiting**

I. Introducción

Con el objetivo de poner a prueba los modelos de *clustering* se utilizó el *dataset* datasettissue.txt, en cual consta de 22215 filas y 189 columnas que representan las distintas características de diferentes tipos de tejidos. Se buscó predecir el tipo de tejido al cual pertenece cada uno de los puntos, mediante agrupación haciendo uso de modelos de *clustering* tales como *GMM, KMeans, DBScan y Agglomerative Hierarchical Clustering*. Posteriormente, se comparó el resultado obtenido de los modelos con los presentados en el *dataset* para así lograr presentar las matrices de confusión y analizar cual de los modelos de *clustering* nos brinda mejores resultados.

II. Explicación

A. KMeans

Para este método se usaron las siguientes funciones:

- *distance()*: Obtiene la distancia euclidiana entre dos elementos del *dataset*.
- *initialize_centroids()*: Escoge elementos del *dataset* de forma aleatoria y los utiliza para inicializar los centroides.
- *_create_clusters()*: Asocia en clusters todos los indices de los datos que tengan el mismo centroide más cercano
- *_get_closest_centroid(row)*: funcion de ayuda utilizada en create_clusters(). Dado un elemento del *dataset* devuelve el centroide más cercano.
- *_update_centroids()*: Se encarga de calcular el error de cada fase de entrenamiento.
- *_update_centroids()*: Actualiza el centroide a la media de todos los elementos del cluster relacionados a este centroide.
- *kmeans()*: Es la función principal, inicializa los centroides con la función *initialize_centroids*. Luego de ello, genera los clusters asociados a los centroides actualiza estos últimos para ubicarse en la media del cluster. Este proceso se da con las funciones *_create_clusters()*, *_update_centroids()*, y se realiza hasta que se llegue un numero determinado de iteraciones o hasta que los centroides convergan. Finalmente retorna los clusters.

B. Sklearn

Para el resto de métodos de *clustering* se usó la librería *sklearn*.

III. Funciones Generales

- *reduction()*: Se encarga de reducir la dimensionalidad del *dataset* a la dimensión deseada.
- *normalize()*: Se encarga de normalizar los datos.
- *toCSV()*: Se encarga de pasar los datos de los archivos txt a formato csv para hacer uso eficiente de la lectura por disco.
- *read()*: Se encarga de los archivo csv proporcionados por la función *toCSV*.

IV. Experimentos

Para la parte de experimentación se contó con datos de 22215 atributos, por lo que se usó un método de reducción de dimensión llamado *PCA (principal component analysis)*. Luego de diversas pruebas se logró reducir la cantidad de atributos a 20 sin perder demasiado el cubrimiento total de los datos. Posteriormente, se comparó los *labels* obtenidos por cada modelo de *clustering* y se comparó con los del *dataset* para así analizar las gráficas de errores de cada modelo.

A. GMM

En este modelo se usaron los siguientes parámetros:

- *n_components*: Como 7, ya que este parámetro nos indica la cantidad de *clusters* que queremos tener. Ya que la cantidad de nombres de los tejidos es 7, creamos la misma cantidad de *clusters*.
- *random_state*: Como 0, este parámetro indica la semilla que queremos que tenga el modelo, como se desea replicar los resultados en diferentes computadoras se colocó un número estándar.

Los resultados obtenidos fueron:

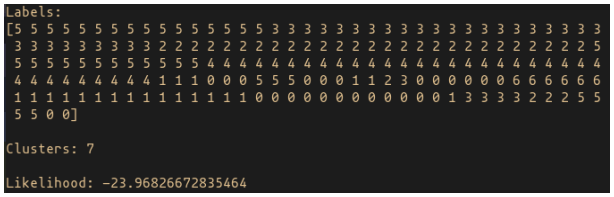

Fig. 1. GMM

Donde podemos observar que el *log likelihood* nos dio un valor negativo, lo cual no confirmaría que el método cumple con lo esperado.

B. KMeans

En este modelo se usaron los siguientes parámetros:

- *data*: Este parámetro simplemente recibe los elementos "X" de la *data*. En este caso recibe la data normalizada y posteriormente reducida mediante análisis de componentes principales(PCA)
- *k*: Como 7, ya que este parámetro nos indica la cantidad de *clusters* que queremos tener. Ya que la cantidad de nombres de los tejidos es 7, creamos la misma cantidad de *clusters*.
- *iterations*: Como 100 de forma estándar, este parámetro determina el número máximo de iteraciones que realizará el algoritmo. Es decir el número de actualizaciones que tendrán los centroides, y por ende, los clusters.

Los resultados obtenidos fueron:

Fig. 2. Impresión de clusters obtenidos

C. DBScan

En este modelo se usaron los siguientes parámetros:

- *eps*: Como 3, este parámetro nos indica el radio que queremos usar para capturar a los vecinos más cercanos.
- *min_samples*: Como 2, este parámetro indica el mínimo número de elementos que tienen que tener los *clusters*.

Los resultados obtenidos fueron: Lo más rescatable de los

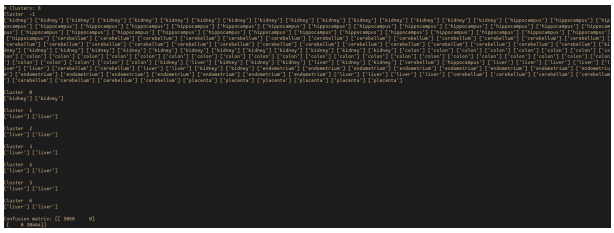

Fig. 3. DBScan

resultados es que la mayoría de datos nos pertenecen a ningún *cluster* (Cluster -1). Sin embargo sí se llegaron a crear 7 *clusters*.

Conclusiones

- Analizandos los distintos resultados que nos otorga cada modelo de *clustering*, el que nos da mayor *accuracy* es el modelo de
- Depende de los hiperparámetros que le pasemos a cada modelo para aumentar su precisión de clasificación.
- Es necesario la reducción de datos ya que si no los modelos serían muy pesados computacionalmente.
- Al no usar todos los atributos es que no se genera un mayor *accuracy* en los modelos.
- Al disminuir el *alpha* el algoritmo necesitará mayor cantidad de *epoch* para poder hallar un error mínimo óptimo.
- Para obtener un buen resultado con el modelo DBScan es un prueba y error de los hiperparámetros que le podríamos pasar.

Repositorio

Enlace al repositorio de Github:
https://github.com/mauricio-rodriguez/ai-project-3

References

[1] Montero, R. (2016). *Modelo de regresión lineal múltiple*. Universidad de Granada. España.

[2] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Cambridge CB3 0FB, U.K.

[3] Jain, S. (2020). *Some Notes on DBSCAN Algorithm*. Recovery from: https://medium.com/analytics-vidhya/some-notes-on-dbscan-algorithm-61a2e9acce29