Centro de Investigación en Matemáticas (CIMAT). Unidad Monterrey

Tarea 3

Enrique Santibáñez Cortés

23 de marzo de 2021

1. Problema 1.

En este ejercicio es sobre clustering y mezclas de Gaussianas. Considera un modelo de mezclas de k distribuciones

$$f(x) = \sum_{k=1}^{K} w_k f_k(x),$$

donde $w_k > 0$ y $\sum_k w_k = 1$. En este caso, supondremos que $f_k = N(\mu_k,_k)$. Supón que tienes datos $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \sim f(x)$, y queremos ajustar el modelo de mezclas de Gaussinas (MMG) para usarlo como un soft−clustering.

1.1. Log−verosimilitud de los datos y los estimadores de máxima verosimilitud para los parámetros del modelo.

Dado que tenemos $\mathbf{x}_i, \cdots, \mathbf{x}_n$ in $\mathbf{R}^d$, nuestro objetivo es determinar nuestros parámetos $\theta = \{w_1, \cdots, w_k, \mu_1, \cdots, \mu_k, \Sigma_1, \cdots, \Sigma_k\}$, tenemos que la log−verosimilitud de los datos es:

$$\mathbf{L}(\mathbf{x}_1, \cdots, \mathbf{x}_n; \theta) = \log \prod_{i=1}^{n} P(x_i; \theta) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} p_k f_k(x_i) \Bigg]$$

No existe una solución de forma cerrada para encontrar el conjunto de parámetros θ que maximice esta probabilidad. El algoritmo EM es un algoritmo iterativo que encuentra una solución localmente óptima θ al problema de maximización de la probabilidad de GMM.

Paso E. El Paso E del algoritmo implica encontrar la probabilidad posterior $\gamma_i^k = P(cluster\ k|x_i); \theta)$ de que el punto x_i fue generado por el cluster k, para cada $i = 1, \cdots, n$ y $k = 1, \cdots, K$. Este paso asume el conocimiento del conjunto de parámetros θ. Encontramos el posterior usando la regla de

Bayes:

$$\gamma_i^k = \mathbb{P}(C(i) = k|X = x_i) = \frac{w_k f_k(x_i;\theta)}{\sum_k w_k f_l(x_i;\theta}$$

Paso M. El paso M del algoritmo maximiza la función de probabilidad logarítmica esperada $\bar{\mathbf{L}}(x_i, \cdots, x_n;\theta)$, que es un límite inferior en la probabilidad logarítmica. Por tanto, el algoritmo empuja iterativamente hacia arriba la probabilidad de datos.
La función de verosimilitud logarítmica esperada $\bar{\mathbf{L}}(x_i, \cdots, x_n;\theta)$ es

$$\begin{aligned}
\bar{\mathbf{L}}(x_i, \cdots, x_n;\theta) &= \sum_{i=1}^{n}\left[\sum_{k=1}^{K}\gamma_i^k \log\left(\frac{P(x_i, cluster\ k;\theta)}{\gamma_i^k}\right)\right] \\
&= \sum_{i=1}^{n}\left[\sum_{k=1}^{K}\gamma_i^k \log\left(\frac{w_k f_k(x_i;\theta)}{\gamma_i^k}\right)\right] \\
&= \sum_{i=1}^{n}\left[\sum_{k=1}^{K}\gamma_i^k \log\left(\frac{w_k \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}_i-\mu_j)\right)}{\gamma_i^k}\right)\right]
\end{aligned}$$

Ahora calculemos los estimadores de maxima verosimilitud, para ello maximicemos respecto a μ_l:

$$\begin{aligned}
\frac{\partial\bar{\mathbf{L}}(x_i, \cdots, x_n;\theta)}{\partial\mu_l} &= \frac{\partial}{\partial\mu_l}\sum_{i=1}^{n}\left[\sum_{k=1}^{K}\gamma_i^k \log\left(\frac{w_k \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}}\exp\left(-\frac{1}{2}(\mathbf{x}_i-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}_i-\mu_j)\right)}{\gamma_i^k}\right)\right] \\
&= -\frac{\partial}{\partial\mu_l}\sum_{i=1}^{n}\sum_{k=1}^{K}\gamma_i^k\frac{1}{2}(\mathbf{x}_i-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}_i-\mu_j) \\
&= \frac{1}{2}\sum_{i=1}^{n}\gamma_i^k\frac{\partial}{\partial\mu_l}2\mu_l^T\Sigma_l^{-1}x_i - \mu_{\tilde{n}^T\Sigma_l^{-1}\mu_l} \\
&= \sum_{i=1}^{n}\gamma_l^j\left(\Sigma_l^{-1}x_i - \Sigma_l^{-1}\right).
\end{aligned}$$

Ahora igualamos a cero y resolvemos para μ_l tenemos que

$$\hat{\mu_\mathbf{l}} = \frac{\sum_{\mathbf{i=1}}^{\mathbf{n}}\gamma_\mathbf{i}^\mathbf{k}\mathbf{x_i}}{\sum_{\mathbf{i=1}}^{\mathbf{n}}\gamma_\mathbf{i}^\mathbf{k}.}$$

Ahora calculemos el estimador de máxima verosimilitud para w_l. Agrupando los términos que depende de w_l, encontramos que necesitamos maximizar

$$\sum_{i=1}^{n}\sum_{k=1}^{K}\gamma_i^k \log w_l.$$

Sin embargo, sabemos que $\sum_{i=1}^{n} w_i = 1$. Entonces tenemos un problema de optimización con restricción, para ello construimos el Lagrangiano

$$\mathbb{L}(w) = \sum_{i=1}^{n}\sum_{k=1}^{K}\gamma_i^k \log w_j + \beta\left(\sum_{j=1}^{n} w_j - 1\right),$$

donde β es el mutiliplicador de lagrange. Derivando, tenemos que

$$\frac{\partial \mathbb{L}(w)}{\partial w_j} = \sum_{i=1}^{n} \frac{\gamma_i^k}{w_j} + \beta.$$

Igualando a cero y resolviento tenemos que,

$$\hat{w}_j = \frac{\sum_{i=1}^{n} \gamma_i^k}{-\beta}.$$

Ahora, usando que sabemos que $\sum_{i=1}^{n} w_i = 1$, podemos despejar y obtener que $-\beta = \sum_{i=1}^{n} \sum_{k=1}^{K} w_i^k = \sum_{i=1}^{n} 1 = n$. Por lo tanto, el estimador de máxima verosimilitud es

$$\hat{w}_j = \frac{1}{n} \sum_{i=1}^{n} \gamma_i^k.$$

De la misma forma que los demás estimadores, tenemos que el estimador de máxima verosimilitud para Σ es

$$\Sigma_k = \frac{\sum_{i=1}^{n} (x_i - \mu_c)^2 \gamma_i^k}{\sum_{i=1}^{n} \gamma_i^k}.$$

1.2. Comparación de resultados

Para comprobar la eficiencia del modelo de mezclas de Gaussianas usemos un conjunto de datos de sklearn. En este conjunto de datos se observan claramente tres cluster. Si observamos la forma en que están en el espacio, podemos notar que no sería posible utilizar algún método líneal para particionar los tres cluster, por lo que se espera que el algoritmo fuzzy k−means tenga una menor eficiencia en este conjunto de datos.

Si observamos la Figura 1.1, la cual compara el algoritmo fuzzy k−means contra el modelo GMM. Podemos notar claramente la eficiencia que tiene el modelo GMM en comparación del fuzzy k−means.

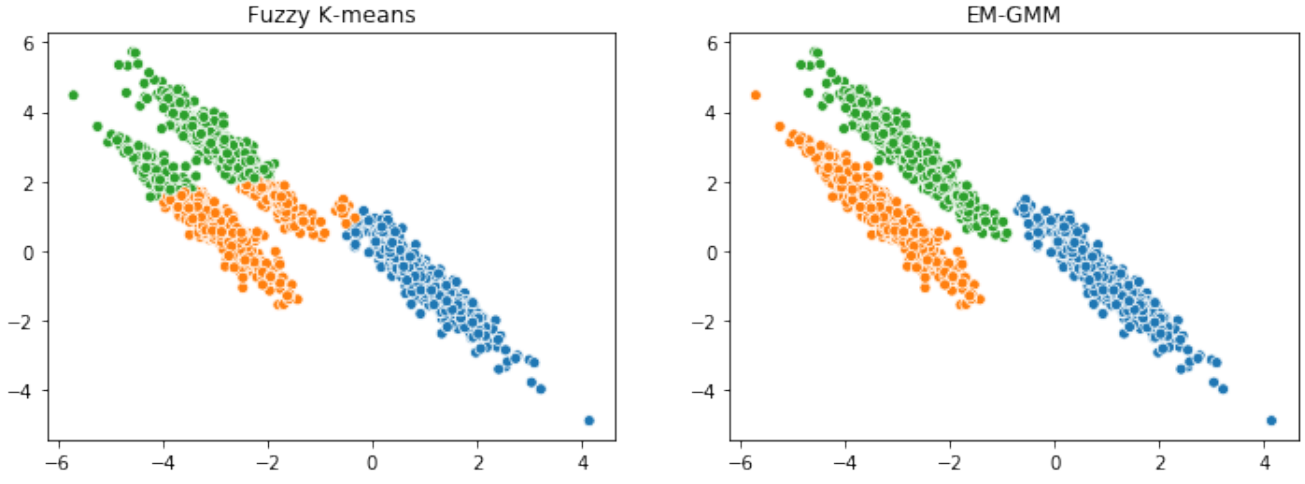

Figura 1.1: Data set: Blob con distintas medias

En la siguiente sección 2.1 realizamos más comparaciones con distintos modelos de clustering y observamos las desventajas que tiene GMM en algunas ocaciones.

1.3. $\sigma^2 \to 0$

Para mostrar que cuando $= \sigma^2\mathbf{I}$ y si $\sigma^2 \to 0$ entonces el modelo EM$-$GMM se parece al modelo k$-$means, utilizemos el mismo conjunto de datos solo que cambiemos $= \sigma^2\mathbf{I}$. Es decir, estamos considerando $\Sigma = \begin{pmatrix} 0{,}04 & 0 & 0 \\ 0 & 0{,}0150 & 0 \\ 0 & 0 & 0{,}09 \end{pmatrix}$ para generar los datos. Observando la figura 1.2 podemos notar que los clusters construidos por EM$-$GMM son iguales a los de k$-$means. Por lo que si se cumple. Se probaron distintas matriz de varianzas y covarianzas y se sigue cumpliendo para valores pequeños de σ^2.

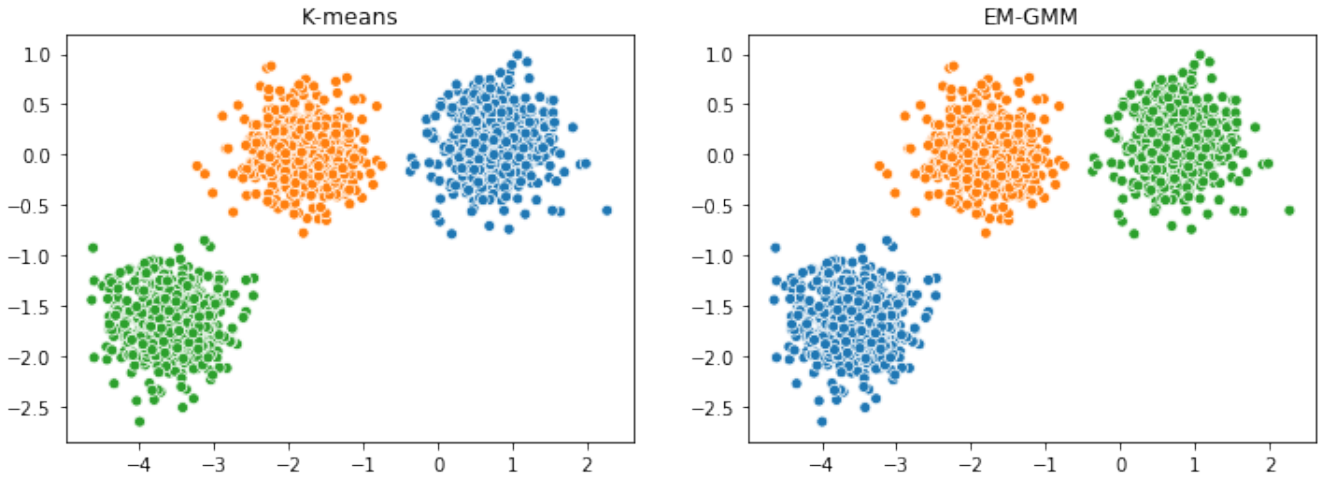

Figura 1.2: Data set: Blob con distintas medias y varianza constante

2. Kernel K$-$means

Consideremos que tenemos un conjunto de datos $\mathbf{x}_1, \cdots, \mathbf{x}_n$ el argoritmo $k-$means trata de encontar $\pi_1, \cdots, \pi_k$ que minicen la función objetivo

$$\mathbf{D}\left(\{\pi_c\}_{c=1}^k\right) = \sum_{c=1}^{k} \sim_{x_i \in \pi_c} ||x_i - m_c||^2, \text{ donde } m_c = \frac{\sum_{x_i \in \pi_c} x_i}{|\pi_c|}.$$

El kernel $k-$ means usa una función para mapear puntos a un espacio de características de mayor dimensión. Cuando se aplica $k-$means en este espacio de características, los separadores lineales en el espacio de características correponden a separadores no lineales en el espacio de entrada. El objetivo del kernel $k-$means se puede escribir como la minización de

$$\mathbf{D}\left(\{\pi_c\}_{c=1}^k\right) = \sum_{c=1}^{k} \sim_{x_i \in \pi_c} ||\phi(x_i) - m_c||^2, \text{ donde } m_c = \frac{\sum_{\phi(x_i) \in \pi_c} x_i}{|\pi_c|}.$$

Expendiendo la distancia $||\phi(x_i) - m_c||^2$ en la función anterior obtenemos la siguiente expresión

$$||\phi(x_i) - m_c||^2 = \phi(x_i) \cdot \phi(x_i) - \frac{2\sum_{x_i \in \pi_c} \phi(x_i) \cdot \phi(x_j)}{|\pi_c|} + \frac{\sum_{x_j, x_l \in \pi_c} \phi(x_j) \cdot \phi(x_l)}{|\pi_c|^2}$$

Entonces, Por lo tanto, solo se utilizan productos internos en el cálculo de la distancia euclidiana entre un punto y un centroide. Como resultado, si se nos da una matriz de núcleo K, donde $K_{ij} = \phi(a_i) \cdot \phi(x_j)$, podemos calcular distancias entre puntos y centroides sin conocer representaciones explícitas de $\phi(x_i)$ y $\phi(x_j)$ [1].

El algoritmo que se utilizo para construir la función kernel $k-$means es el que se describe en **Algoritmo:** 1, solo que con se vectorizar para mejorar la velocidad de ejecución. Es decir, en cada iteración nosotros calculamos todas las distancias del conjunto de datos al cluster c. Lo anterior permitió probar con distintos parámetros en las funciones.

Algoritmo: 1. ***Entradas:*** *X : matriz de datos, K: matriz kernel, k: número de clusters, t: número de iteración, t_max: número de iteraciones máximas, ${\pi_c^{(t)}}_{c=1}^{k}$: vector de la clasificación de los cluster.*
Ouput: *${\pi_c^{(t)}}_{c=1}^{k}$:vector de la clasificación de los cluster en el paso $t+1$.*

1. *Para cada punto x_i y para cada cluster c, calcular*

$$d(_i, m_c) = K_{ii} - \frac{2\sum_{x_i \in \pi_c} K_{ij}}{|\pi_c|} + \frac{\sum_{x_j, x_l \in \pi_c} K_{jl}}{|\pi_c|^2}$$

2. *Actualizamos los cluster como*

$$\pi_c^{(t+1)} = \{x : c^*(x_i) = c\}.$$

3. *Si no converge (es decir, comparamos si $\pi_c^{(t+1)} = \pi_c^{(t)}$) o $t\_max > t$ entonces actualizamos $t = t+1$ y regresamos al paso 2. En otro caso, terminamos y retornamos $\{\pi_c^{(t+1)}\}_{c=1}^{k}$.*

2.1. Comparación de Resultados

Utilizamos tres conjuntos de datos diferentes para probar la eficiencia del método kernel $k-$means, estos conjuntos de datos son de la liberaría de python sklearn[2]. Para cada conjunto de datos se utilizo kernel k−means, k−means, fuzzy k−means y EM GMM (del inciso anterior), se entrenaron distintos modelos y aquí se muestran los que se consideran como mejores.

El primer conjunto de datos (Ver figura 2.1) se pueden observar que estos no tienen ningún patrón, por lo que no existe como una partición de ellos. Observamos que utilizando kernel k−means parte los datos en dos segmentos al igual que EM−GMM. K−means y fuzzy k−means parten el conjunto de datos en tres. Para este caso, a todos los modelos se considero el número de cluster igual a 3 ($k = 3$).

El siguiente conjunto de datos (Ver figura 2.2) representa a dos cluster de no lineales (dos círculos diferentes). Se considera como mejor partición de los datos como los dos círculos, por lo que el mejor ajuste se obtiene cuando se usa kernel k−means. En este caso se ocupo un kernel polinomial de grado 2, que si recordamos lo visto en clase se obtiene lo mismo.

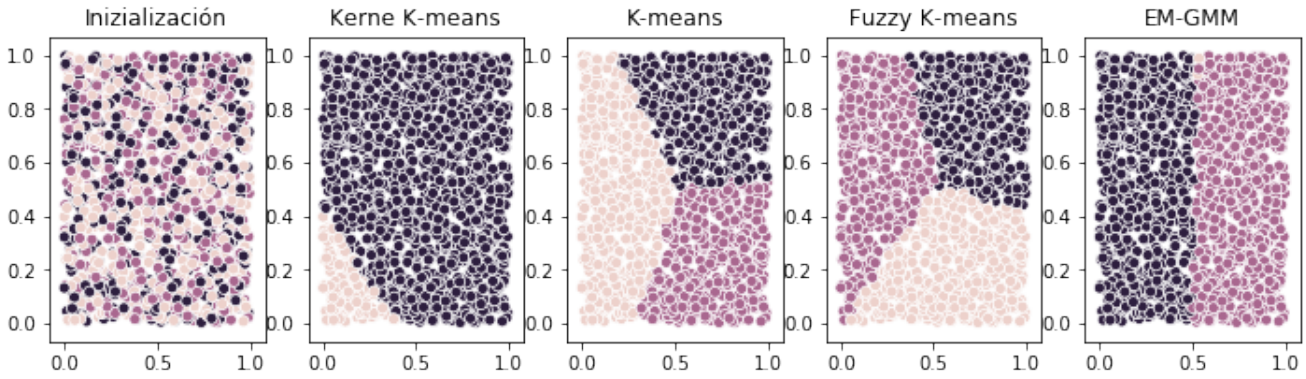

Figura 2.1: Data set: Sin estructura

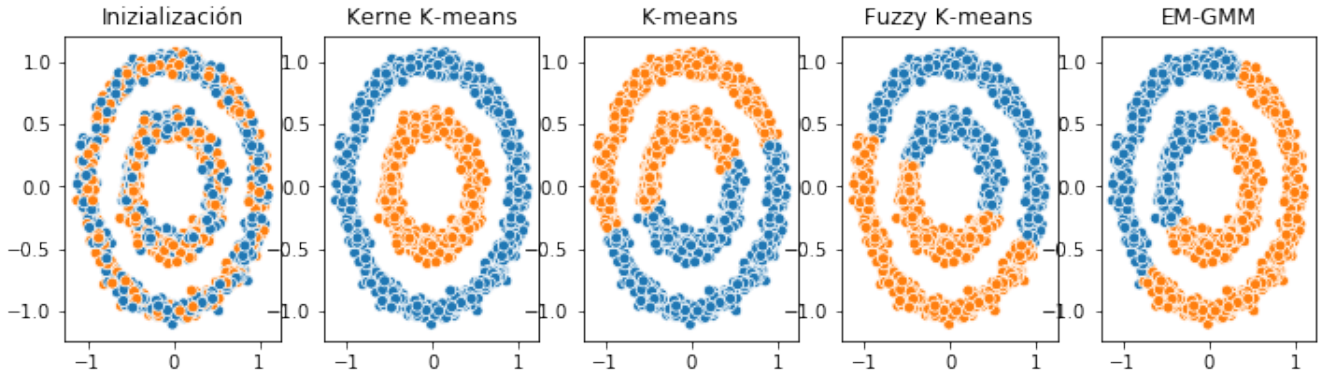

Figura 2.2: Data set: circulo.

Ahora consideramos un conjunto de datos en donde claramente se observan tres poblaciones distintas, es decir, visualmente se puede determinar de forma lineal los cluster (Ver figura 2.3). En este conjunto de datos, los cuatro algoritmos de clustering logran conseguir los tres cluster. Y por último, el conjunto de medias lunas (Ver figura 2.4) observamos que ningún método de clustering puede clasificar bien este conjunto de datos. Sería recomendable profundizar más con algún otro algoritmo de clasificación (DBSCAN).

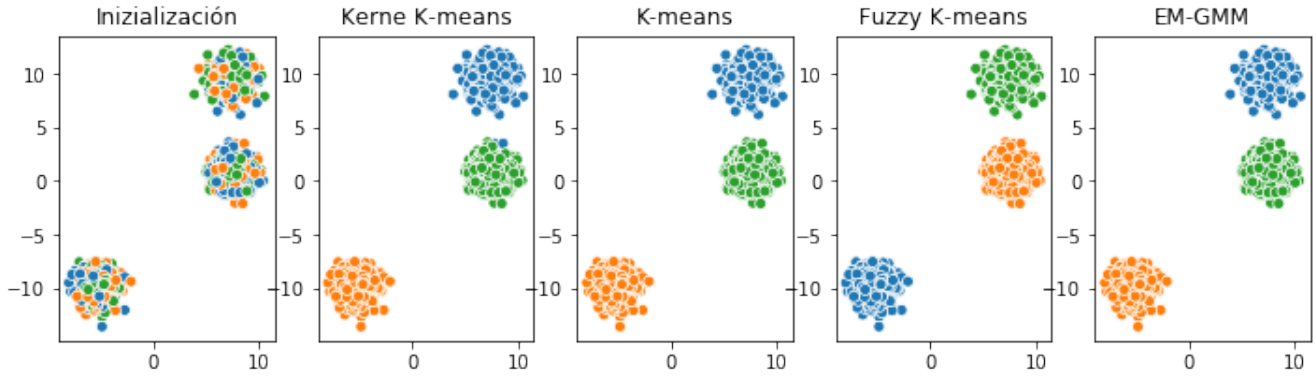

Figura 2.3: Data set: Blob

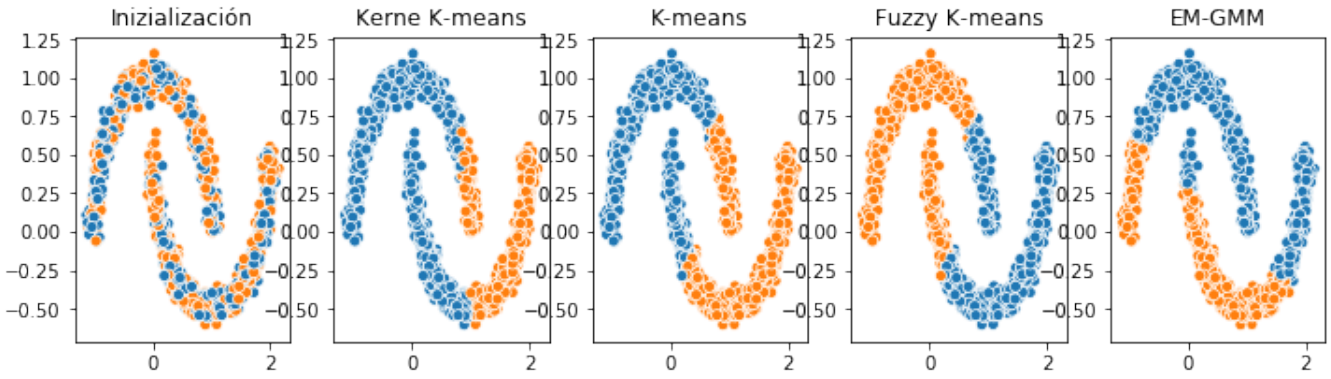

Figura 2.4: Data set: Lunas aleatorias

2.2. Conclusiones

En este ejercicio se mostró la eficiencia del kernel k−means, podríamos decir que usar este método en ocasiones cuando existan clusters no lineales aunque esto no es garantía de clasificar bien. Pero si existen una gran ventaja en esta situación que si utilizáramos k−means o fuzzy k−means, pues recordemos que estos son clasificadores lineales lo que sería imposible para ellos clasificar bien cuando existan patrones no lineas.

3. Frutas.

3.1. Introducción

Los datos en el archivo DATAFRUITSTAREA.ZIP contienen imágenes preprocesadas de 100×100 pixeles, que corresponde a diferentes tipos de frutas, tomadas en diferentes orientaciones y con diferentes características de forma y maduración. En este ejercicio, tratarás de identificar las frutas obteniendo algunas representaciones a partir de las imágenes (Figura 1, por ejemplo).

3.2. RGM

Utilizando la representación RGM de las imágenes obtenemos la media de cada color, es decir, teníamos un conjunto de datos de $1300 \times 100 \times 100$ y lo reducimos a 1300×3. En la figura 3.1 podemos observar que las cherry (café) y huckleberry (rosa) se pueden clasificar de manera sencilla utilizando la media del color rojo y azul, estas dos frutas casi no tiene presentes el color rojo por lo que eso explica esa separación entre las demás frutas. Ahora si nos centramos en el color verde (el otro rosa), podemos observar que los aguacates tiene una distribución muy estrecha en comparación de las demás frutas por lo que se podría separar utilizando esta variable.

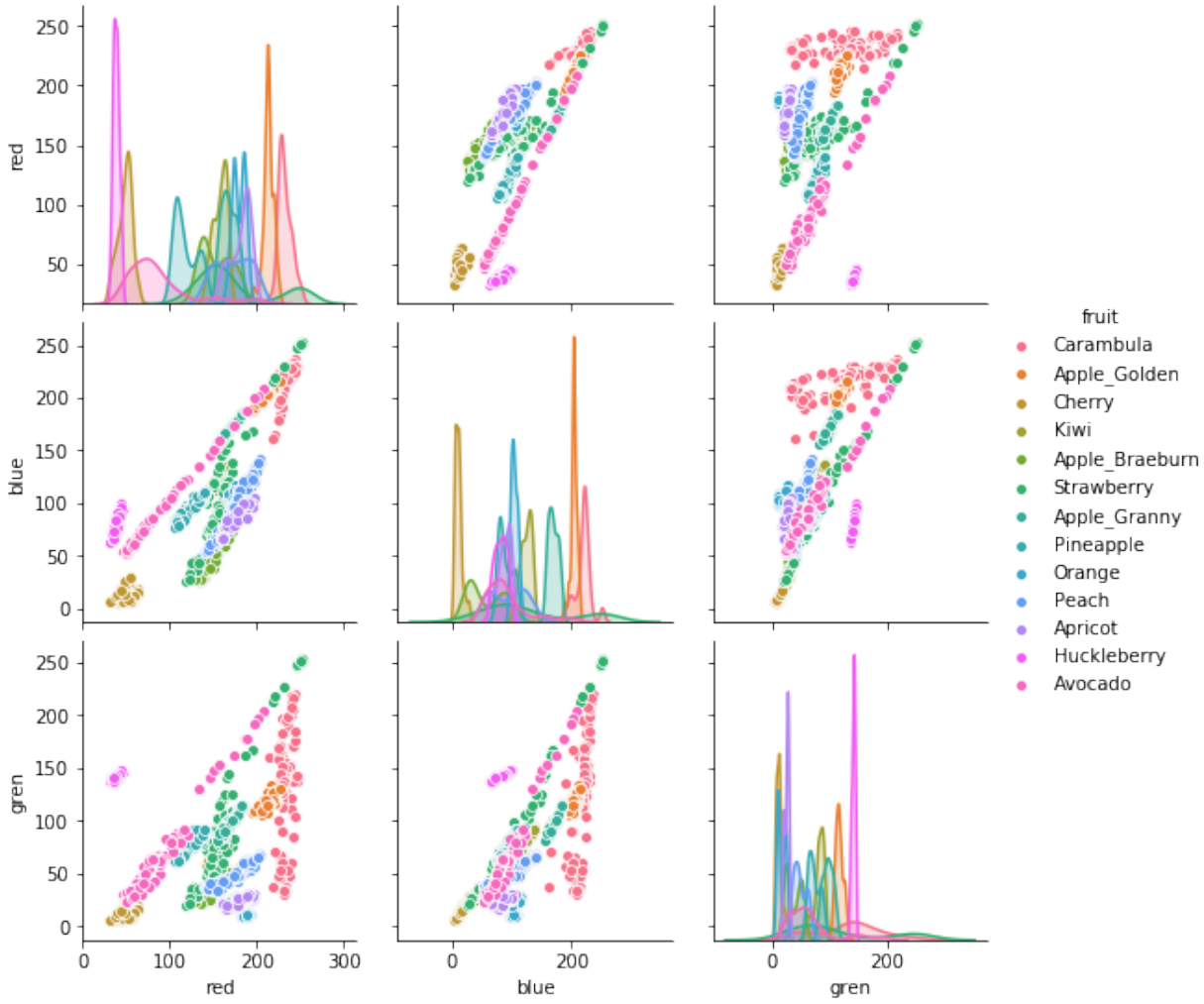

Figura 3.1: Gráfica en pares de las medias RGM

3.3. PCA y Kernel PCA

Ahora utilizaremos PCA y Kernel PCA (se considero $\alpha = 0{,}2$ en la mayoría de los casos) para reducir la dimensión todavía más, esto para poder vizualizar los datos de una manera sencilla sin perder demasiada información. Graficando los primeros dos componentes principales de ambos métodos, podemos observar nuevamente que los cherries, huckeberry y avocado se pueden clasificar facilmente. En el proyección utilizando PCA podemos ver que la apple golden se puede separar fácilmente. Qué a diferencia de las demás frutas no es tán intuitivo formar los cluster para las 13 frutas. Observemos que las naranjas y los duraznos están muy cercanos lo que tiene mucho sentido por los colores de las frutas.

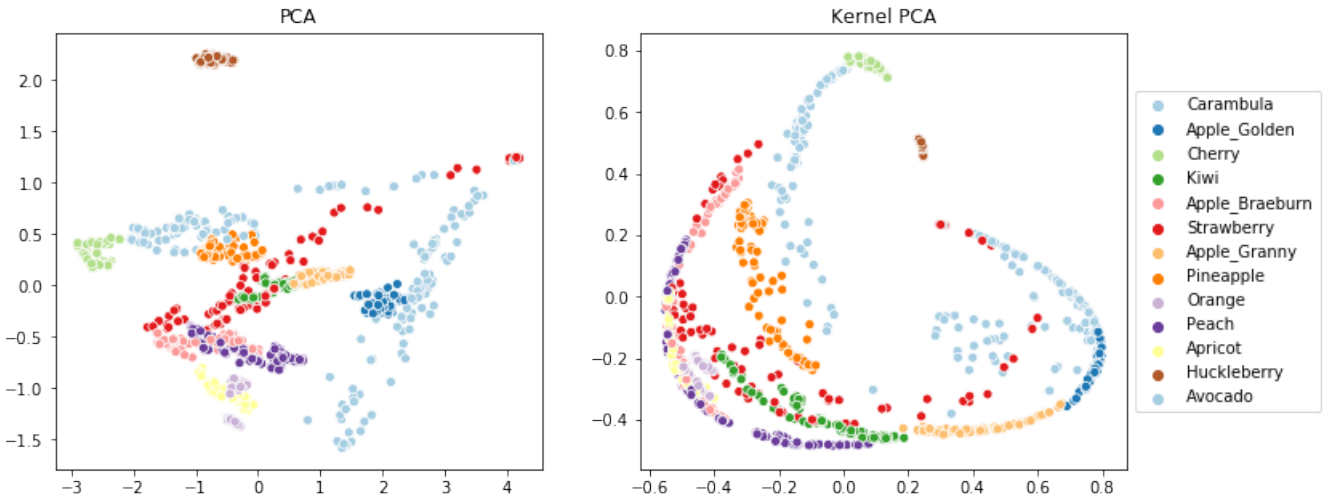

Figura 3.2: PCA y Kernel PCA

Consideró que ni PCA como kernel PCA son adecuados para la reducción de la dimensionalidad para este ejercicio, me hubiera gustado probar como TNE o MDS pero por cuestiones de tiempo no pude.

3.4. K−means y kernel K−means

Aplica Kmeans y Kernel Kmeans. Verifica si puedes identificar los diferentes grupos de frutas. Considerando las proyecciones de la sección anterior, aplicamos k−means y kernel k−means para intentar clasificar las 13 frutas diferentes. La mejor visualización de k−means fue utilizando la proyección PCA, lo cuál tiene sentido ya que realizar una proyección no lineal hace que el algoritmo k−means (lineal) no tenga una buena eficiencia. Se comprueba que clasifica bien las frutas mencionadas anteriormente, (Ver figura 3.3) es decir, las cherries, huckleberry las clasifican bien. Los duraznos y naranjas los junta en un mismo cluster.

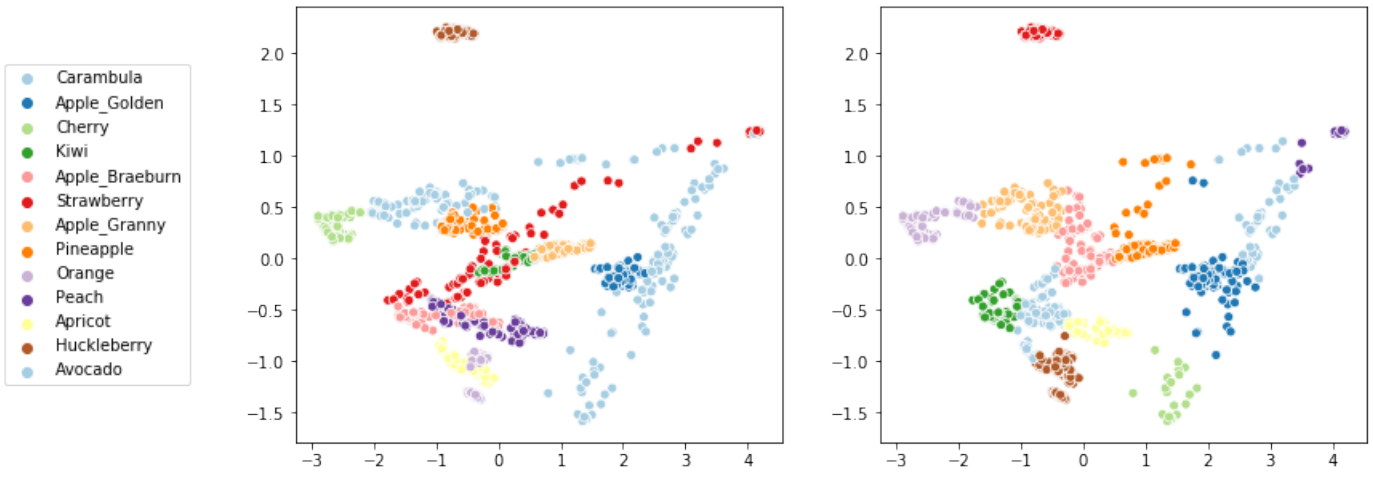

Figura 3.3: Kmeans en las proyecciones PCA

Ahora, si consideramos Kernel k−means en las proyecciones Kernel PCA algo interesante a observar

es que puede clasificar muy bien a las frutas apple brauburn, apricot y apple grammy. Esto se pueda deber a que estas frutas en las proyecciones kernel PCA son no lineales lo que hace posible separarlos utilizando kerner k−means.

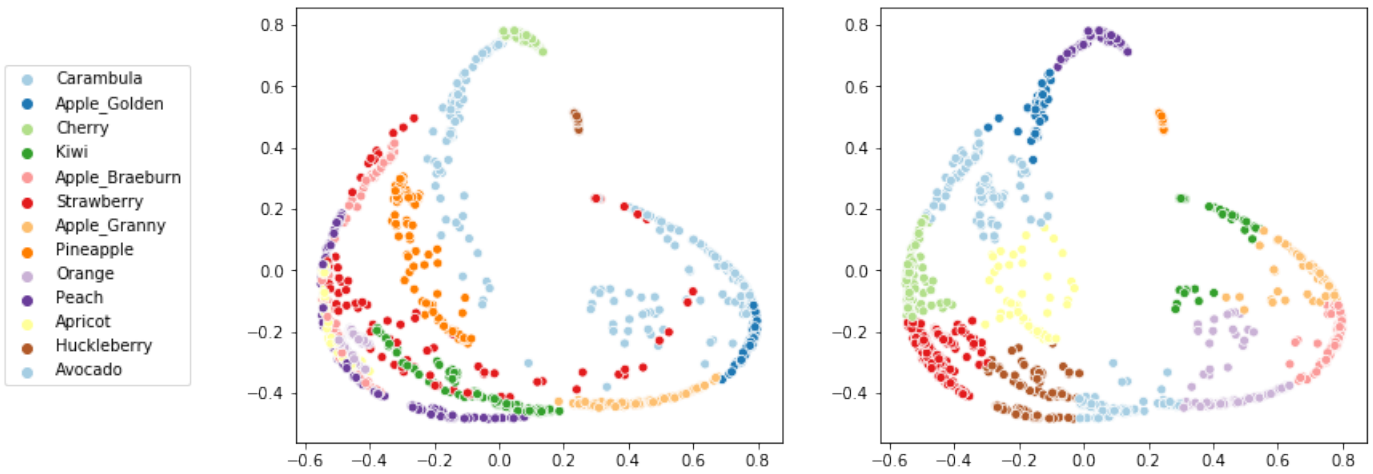

Figura 3.4: Kernel k−means en las proyecciones Kernel PCA

3.5. Representación usando HSV

Ahora consideraremos el espacio HSV de las imágenes. En lugar de utilizar solo la mediana como variable, ahora consideraremos los tres cuartiles centrales para cada uno de las dimensiones. Este cambio se espera tener una mayor eficiencia a la hora de clasificar las frutas.

Realizamos las proyecciones utilizando PCA y Kernel PCA (Ver figura 3.5). Para las proyecciones usando PCA podemos observar que huckleberry, pineapple y strawberry son las frutas que posiblemente se puedan separa, que a diferencia de las proyecciones KPCA solo huckleberry y pineable.

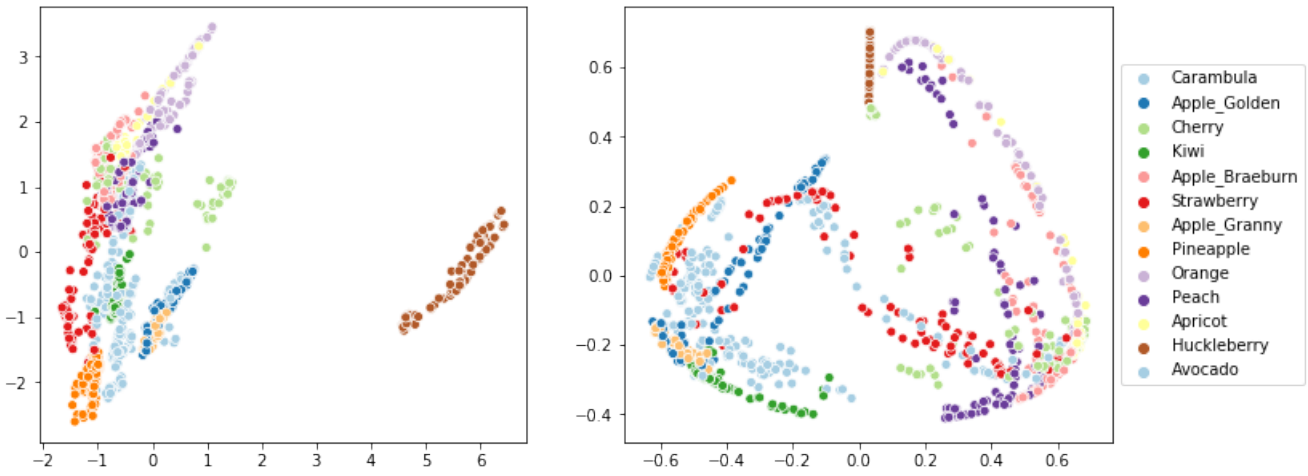

Figura 3.5: Proyecciones usando PCA y KPCA en HVS

Realizando todas las posibles combinaciones de k−means, kernel k−menas en las proyecciones PCA y KPCA, presentamos mejores resultados. Usando k−means usando las proyecciones PCA (Ver

Figura 3.6) si comparamos los resultados usando las imágenes en RGB estos no son buenos. Por lo que sería recomendado simplemente utilizar los proyecciones del inciso anterior.

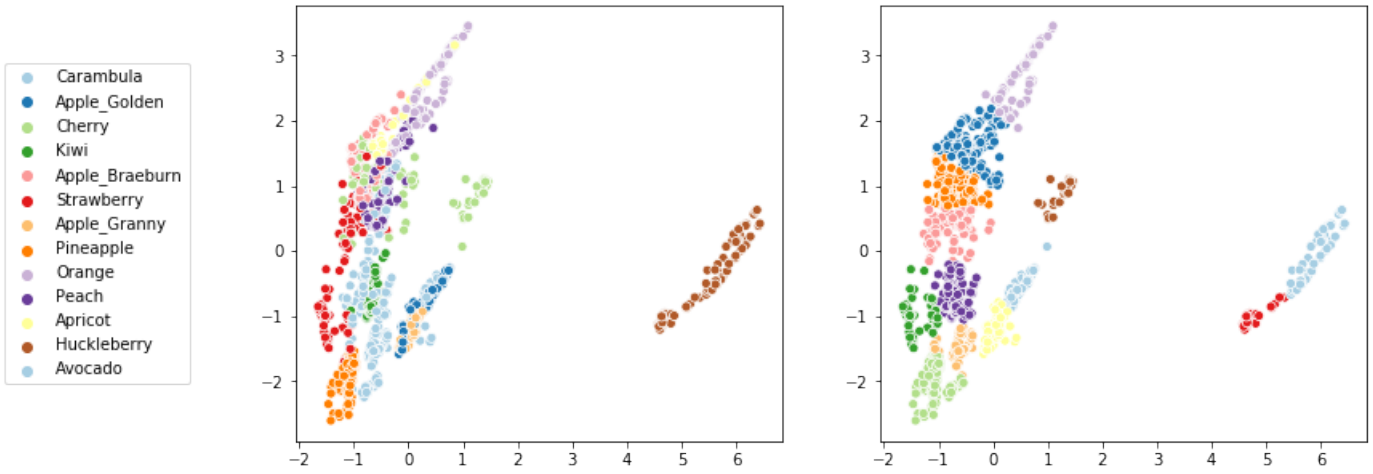

Figura 3.6: K−means usando las proyeciones PCA de HVS

Y ahora, si consideramos Kernel k−means usando las proyecciones KPCA podemos clasificar algunas frutas (orange, apple golden) pero la clasificación no sigue siendo muy buena.

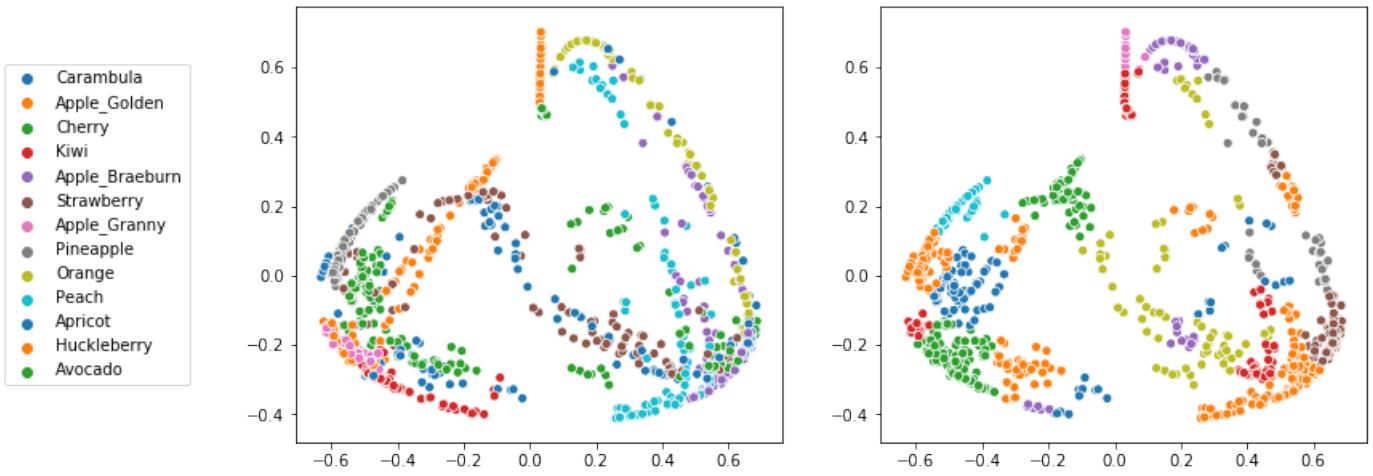

Figura 3.7: Kernel K−means usando las proyeciones KPCA de HVS

3.6. Conclusión

No se pudo clasificar de manera adecuada todas las frutas, esto puede deberse a la variedad de colores de cada fruta. Como el conjunto de datos tiene tanto frutas maduras como inmaduras, esto hace que sea sencillo confundirse. Por ejemplo, si comparamos el color de una naranja cuando esta madura es muy diferente que cuando no lo esta.

Otra razón puede deberse a que los hyperparametros de algoritmos afectan demasiado las clasificaciones. Por hyperparametros me refiero a los valores iniciales del Kernel k−means, así como el

parámetro que se utiliza al calcular la matriz de distancias. Esto hace que las clasificaciones varíen y no convengan a un mismo mínimo.

4. Anexos

4.1. Códigos

Todos los códigos utilizados para estos resultados se pueden encontrar en mi página personal de Gitgub: Enriquesec. En el repositorio Ciencia_de_Datos/Tareas/Tareas3/. El archivo EM.py contiene el algoritmo EM para ajustar el modelo de mezclas de Gaussinas. El archivo kernels.py contiene la implementación del kernel k−means. Y el notebook homework3.ipnyb contiene las gráficas de este reporte.

Referencias

[1] I. Dhillon, Yuqiang Guan y B. Kulis. "A Unified View of Kernel k-means , Spectral Clustering and Graph Cuts". En: (2004).

[2] F. Pedregosa y col. "Scikit-learn: Machine Learning in Python". En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830.