Investigación Operativa

Equidad en el algoritmo k-means

Realizado por:

- Javier Amores Gálvez
- Carlos Perea Vega
- Juan Robles Ríos
- Jorge Rodríguez Domínguez
- Jacqueline Santos García

Índice general

Capítulo 1

Introducción

1.1. Fairness (Equidad)

En aprendizaje automático, un algoritmo es justo, o tiene equidad si sus resultados son independientes de un cierto conjunto de variables que consideramos sensibles y no relacionadas con él (por ejemplo: género, etnia, orientación sexual, etc.).

Las investigaciones sobre equidad en algoritmos de aprendizaje automático son bastante recientes. Algunos de los hechos más destacados en este ámbito han sido los siguientes:

- En 2018 IBM introduce AI Fairness 360 [4], un conjunto de herramientas extensible de código abierto que ayuda a examinar, informar y mitigar la discriminación y el sesgo en los modelos de aprendizaje automático a lo largo del ciclo de vida de la aplicación de IA.
- Facebook afirmó en 2018 que hace uso de Fairness Flow [3], un conjunto de herramientas que ayudan a comparar como de justo es un modelo e identificar que etiquetas funcionan para grupos de usuarios específicos.
- En 2019, Google [2] publica un conjunto de herramientas en Github para estudiar los efectos de la equidad a largo plazo.

A pesar de que se siguen perfeccionando los algoritmos utilizados, los principales avances vienen de la concienciación por parte de algunas grandes empresas de la importancia que va a tener en la sociedad la reducción del sesgo en los algoritmos de aprendizaje automático en un futuro.

1.2. Algoritmo k-means

Los algoritmos de aprendizaje no supervisados se usan para obtener patrones de un conjunto de puntos (datos) sin utilizar resultados conocidos, por lo que pueden usarse para descubrir la estructura que tienen nuestros datos.

El algortimo k-means es un algoritmo de aprendizaje no supervisado que agrupa en k grupos. Este agrupamiento se realiza minimizando la suma de las distancias entre cada punto de nuestro conjunto de datos y el centroide de su grupo, es decir,

$$\min_{c_i} \sum_{i=1}^{k} \sum_{j \in U_i} \|x_{ij} - c_i\|^2,$$

donde c_i es el centroide de la región U_i y x_{ij} son los puntos de nuestro conjunto de datos.

A priori no conocemos las regiones, y por ello reformulamos el problema de optimización, introduciendo variables binarias, y_{ij}, dadas por

$$y_{ij} = \begin{cases} 1 & \text{si el punto } x_j \in U_i \\ 0 & \text{otro caso} \end{cases},$$

así, un problema equivalente al anterior sería

$$\begin{aligned} &\min_{c_i, y_{ij}} \sum_{i=1}^{k} \sum_{j=1}^{J} y_{ij} \|x_j - c_i\|^2 \\ &\text{s.a} \quad \sum_{i=1}^{k} y_{ij} = 1, \ j = 1, ..., J \\ &\qquad y_{ij} \in \{0, 1\}, \ c_i \in \mathbb{R}^d. \end{aligned}$$

El problema de optimización que acabamos de formular es MINLP (Mixed-Integer Nonlinear Problem), y en particular es cúbico y no convexo.

Capítulo 2

Fair k-means

En el capítulo anterior, ya hemos introducido el concepto de equidad (fairness) en el aprendizaje automático, y también el algoritmo k-means. El objetivo en este capítulo, es implementar este concepto de equidad en el k-means.

Antes de definir nuestro algoritmo, vamos a ver un ejemplo ilustrativo en el que se puede apreciar la diferencia del algoritmo k-means original, con el fair k-means:

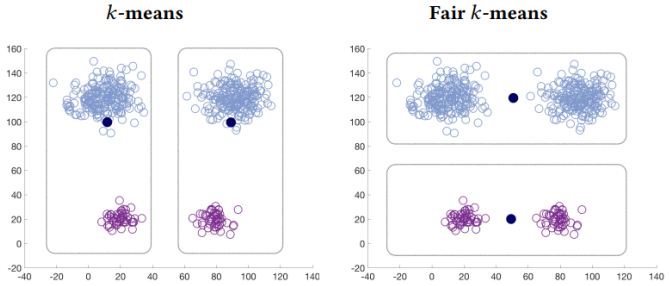


Figura 2.1: Dos grupos demográficos, representados con los colores azul y morado, a los que se le ha aplicado los algoritmos k-means y fair k-means (con $k = 2$).

En el ejemplo de la figura 2.1, al aplicar el k-means, logramos el objetivo de minimizar la distancia total de los puntos al centroide de su partición, sin embargo, para ello vemos que los puntos representados en color morado se han quedado bastante alejados del centroide de su partición y los puntos representados en color azul se han quedado bastante cerca. Aplicando el algoritmo fair k-means, esto se evita (aunque no sea la solución óptima del problema que planteamos en el algoritmo k-means original), ya que el algoritmo trata de minimizar esas distancias pero "sin que ningún grupo se vea perjudicado". Formalicemos esta idea [1].

2.1. Fair k-means: Objetivo y Algoritmo

Para introducir el objetivo del fair k-means, vamos a definir algo más general: el coste del k-means de un conjunto de puntos U con respecto a un conjunto de centros $C = \{c_1, ..., c_k\}$ y una partición $\mathcal{U} = \{U_1, ..., U_k\}$ como

$$\Delta(C, \mathcal{U}) := \sum_{i=1}^{k} \sum_{j \in U_i} \|x_{ij} - c_i\|^2.$$

Recordemos que el k-means original minimiza esta función. En el fair k-means introducimos un nuevo concepto, que son los grupos demográficos. Nosotros nos centraremos en el fair k-means de dos grupos demográficos (aunque se puede generalizar para m grupos demográficos). Consideremos dos grupos A y B y definamos

$$\Phi(C,\mathcal{U}) = \text{máx}\left\{\frac{\Delta(C,\mathcal{U}\cap A)}{|A|}, \frac{\Delta(C,\mathcal{U}\cap B)}{|B|}\right\},$$

donde $\mathcal{U}\cap A = \{U_1\cap A, ..., U_k\cap A\}$. El objetivo del fair k-means es minimizar $\Phi(C,\mathcal{U})$, lo que proporciona un conjunto de centros con la misma distancia media de ambos grupos demográficos.

2.2. Algoritmo Fair-Lloyd

Algoritmo 1 Fair-Lloyd

Require: Un conjunto de puntos $U = A\cup B$ y $k\in\mathbb{N}$
  repeat
    Asignar a cada punto el centro más cercano en C para formar la partición $\mathcal{U} = \{U_1, ..., U_k\}$.
    Escoger un conjunto de centros C que minimice $\Phi(C,\mathcal{U})$.
      $C \leftarrow$ Line Search$(U,\mathcal{U})$. ▷ Lo veremos a continuación
  until convergencia;
  return $C = \{c_1, .., c_k\}$

En cuanto a la calidad de la solución que se encuentra, el Algoritmo Fair-Lloyd converge a una solución óptima local. Se puede probar que bajo ciertas hipótesis sobre la existencia de una solución lo suficientemente buena, que este algortimo encuentra una solución óptima cercana a la solución del k-means.

El algoritmo Fair-Lloyd es un algoritmo de dos pasos, donde en el segundo paso tratamos de buscar un conjunto justo de centros respecto a la partición. Un conjunto de centros C^* es justo respecto a la partición $\mathcal{U}$ si $C^* = \arg\min_C \Phi(C,\mathcal{U})$.

2.3. Estructura y cálculo de centros justos via Line Search

En este apartado mostraremos como un algoritmo Line Search puede ser usado para encontrar C^* eficientemente.

Para un conjunto de puntos $U = A\cup B$ y una partición $\mathcal{U} = \{U_1, ..., U_k\}$ de U, definimos μ_i^A y μ_i^B las medias de $A\cap U_i$ y $B\cap U_i$ respectivamente para $i\in[k]$.

Lema 1. *Sea $U = A\cup B$ y $\mathcal{U} = \{U_1, ..., U_k\}$ una partición de U. Sea $C = \{c_1, ..., c_k\}$ un conjunto de centros justos con respecto a $\mathcal{U}$. Entonces c_i está en el segmento que une μ_i^A y μ_i^B.*

Definición 1. *Dado $U = A\cup B$ y una partición $\mathcal{U} = \{U_1, ..., U_k\}$ de U, para $i = 1, ..., k$, denotamos*

$$\alpha_i = \frac{|A\cap U_i|}{|A|}, \quad \beta_i = \frac{|B\cap U_i|}{|B|}, \quad y \quad l_i = \|\mu_i^A - \mu_i^B\|.$$

También, definimos $M^A = \{\mu_1^A, ..., \mu_k^A\}$ y $M^B = \{\mu_1^B, ..., \mu_k^B\}$.

Lema 2. *Sea $\mathcal{U} = \{U_1, ..., U_k\}$ una partición de $U = A\cup B$. Entonces C es un conjunto de centros justos con respecto a $\mathcal{U}$ si $c_i = \dfrac{(l_i - x_i^*)\mu_i^A + x_i^*\mu_i^B}{l_i}$ donde $(x_i^*, ..., x_k^*, \theta^*)$ es una solución óptima*

del siguiente problema:

$$\begin{aligned} &\text{mín}\,\theta \\ s.a\ \ &\frac{\Delta(M^A,\mathcal{U}\cap A)}{|A|}+\sum_{i\in[k]}\alpha_i x_i^2\le\theta \\ &\frac{\Delta(M^B,\mathcal{U}\cap B)}{|B|}+\sum_{i\in[k]}\beta_i(l_i-x_i)^2\le\theta \\ &0\le x_i\le l_i,\quad i\in[k]. \end{aligned} \tag{1}$$

Ahora necesitamos repasar algunos resultados sobre subgradientes. Dada una función convexa f, decimos que un vector u es un subgradiente de f en un punto x si $f(y)\ge f(x)+u^T(y-x)$ para cualquier y. Denotamos al conjunto de subgradientes de f en x como $\partial f(x)$.

Proposición 1. *Sea f una función convexa. Entonces un punto x^* es un mínimo de f si y solo si $\overrightarrow{0}\in\partial f(x^*)$.*

Proposición 2. *Sean $f_1,...,f_m$ funciones regulares y sea*

$$F(x)=\max_{j\in[m]} f_j(x).$$

Sea $S_x=\{j\in[m]: f_j(x)=F(x)\}$. Entonces el conjunto de subgradientes de F en x es la envolvente convexa de los subgradientes de las f_j en x para $j\in S_x$.

Definamos ahora

$$\begin{aligned} f_A(x)&:=\frac{\Delta(M^A,\mathcal{U}\cap A)}{|A|}+\sum_{i\in[k]}\alpha_i x_i^2, \\ f_B(x)&:=\frac{\Delta(M^B,\mathcal{U}\cap B)}{|B|}+\sum_{i\in[k]}\beta_i(l_i-x_i)^2. \end{aligned}$$

Así, podemos ver el problema (1) como minimizar la siguiente función

$$\begin{aligned} f(x)&:=\max\{f_A(x),f_B(x)\} \\ &\text{s.a } 0\le x_i\le l_i,\ \forall i\in[k] \end{aligned} \tag{2}$$

Nótese que f es convexa, pues es el máximo de dos funciones convexas. Debido a la Proposición 1, nuestro objetivo será encontrar un punto x^* tal que $\overrightarrow{0}\in\partial f(x^*)$. Observamos además que f_A y f_B son funciones diferenciables. De ahí, por la Proposición 2, solo necesitamos buscar entre los puntos x para los cuales existe una combinación convexa de $\bigtriangledown f_A(x)$ y $\bigtriangledown f_B(x)$ que sea igual a $\overrightarrow{0}$. Como veremos, este conjunto de puntos es solo una curva unidimensional en $[0,l_1]\times...\times[0,l_k]$. Cuando $f_A(x)>f_B(x)$, $f(x)$ tiene un único gradiente y es igual a $\bigtriangledown f_A(x)$. De igual forma, cuando $f_A(x)<f_B(x)$, tenemos $\bigtriangledown f(x)=\bigtriangledown f_B(x)$. En el caso en que $f_A(x)=f_B(x)$ para cualquier $\gamma\in[0,1]$,

$$u(\gamma,x):=\gamma\bigtriangledown f_A(x)+(1-\gamma)\bigtriangledown f_B(x)$$

es un subgradiente de $f(x)$, y estos son los únicos subgradientes de f en x. Ahora, consideremos el conjunto

$$Z=\{x\in[0,l_1]\times...\times[0,l_k]:\exists\gamma\in[0,1], u(\gamma,x)=\overrightarrow{0}\}.$$

Si encontramos $x^*\in Z$ tal que $f_A(x^*)=f_B(x^*)$, entonces $\overrightarrow{0}\in\partial f(x^*)$ y por lo tanto, x^* es una solución óptima. Primero describamos Z y veamos que existe una solución óptima en Z.

Lema 3. *Sea $\gamma\in[0,1]$ y $u(\gamma,x)=\overrightarrow{0}$. Entonces*

$$x_i=\frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i+(1-\gamma)\beta_i}.$$

El lema anterior nos da una completa descripción del conjunto Z. Un ejemplo del conjunto Z se muestra en la Figura 2.2 para el caso $k = 2$. El siguiente lema es un resultado inmediato del Lema 3.

Lema 4.

$$Z = \left\{ x : x_i = \frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i + (1-\gamma)\beta_i},\ \gamma \in [0,1] \right\}.$$

Nótese que cuando $\gamma = 1$, este recupera el vector de ceros y cuando $\gamma = 0$, $x = (l_1, ..., l_k)$. Por tanto estos puntos extremos están también en Z. Como ya hemos mencionado si existe un $x^* \in Z$ tal que $f_A(x^*) = f_B(x^*)$, entonces x^* es la solución óptima. Por tanto supongamos que dicha x^* no existe. Uno puede ver que

$$\frac{d}{d\gamma}\left(\frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i + (1-\gamma)\beta_i}\right) = \frac{-\alpha_i\beta_i l_i}{(\gamma\alpha_i + (1-\gamma)\beta_i)^2}.$$

Por tanto, para $1 \leq i \leq k$, x_i es decreciente en γ. También se puede comprobar que f_A es creciente en x_i y f_B es decreciente en x_i. Por tanto f_A es decreciente en γ y f_B es creciente en γ. La Figura 2.2 es un ejemplo que muestra el cambio de f_A y f_B respecto de γ. Esto implica que si no existe ningún $x^* \in Z$ tal que $f_A(x^*) = f_B(x^*)$, entonces o $f_A(\overrightarrow{0}) > f_B(\overrightarrow{0})$ o $f_B(\ell) > f_A(\ell)$ donde $\ell = (l_1, ..., l_k)$.

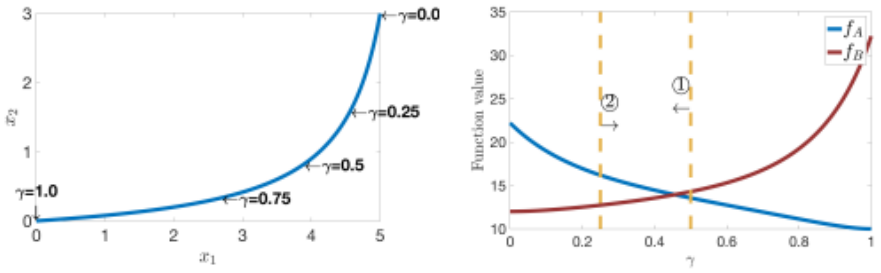


Figura 2.2: En la izquierda tenemos un ejemplo de una curva unidimensional para $k = 2$. En la derecha tenemos las funciones f_A y f_B respecto a γ y 2 iteraciones del algoritmo Line Search. Podemos usar el Line Search para encontrar el valor óptimo de γ y una solución óptima de (2)

Algoritmo 2 Line Search(U,$\mathcal{U}$)

Require: Un conjunto de puntos $U = A \cup B$ y una partición $\mathcal{U} = \{U_1, ..., U_k\}$ de U.
  Calcular $\alpha_i, \beta_i, \mu_i^A, \mu_i^B, l_i, M^A, M^B$ ▷ Ver Definición 1
  $\gamma \leftarrow 0{,}5$
  for t=1,...,T **do**
    $x_i \leftarrow \dfrac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i + (1-\gamma\beta_i)}$, para $i = 1, ..., k$
    Calcular $f_A(x)$ y $f_B(x)$
    if $f_A(x) > f_B(x)$ **then**
      $\gamma \leftarrow \gamma + (1/2)^{-(t+1)}$
    else if $f_A(x) < f_B(x)$ **then**
      $\gamma \leftarrow \gamma - (1/2)^{-(t+1)}$
    else
      break

  $c_i \leftarrow \frac{(l_i - x_i)\mu_i^A + x_i\mu_i^B}{l_i}$, para todo $i = 1, ..., k$
  return $C = \{c_1, ..., c_k\}$

2.4. Generalización a $m > 2$ grupos

En esta sección se introducirá cuál sería el problema que queremos optimizar usando el fair k-means, generalizando el problema de 2 grupos en el que nos hemos centrado anteriormente.

Sea $U = A_1 \cup ... \cup A_m$ el conjunto de puntos dividido en m grupos demográficos. De igual forma, definimos $C = \{c_1, ..., c_k\}$ al conjunto de los centroides y $\mathcal{U} = \{U_1, ..., U_k\}$ al conjunto de las regiones. Entonces, el objetivo del fair k-means con m grupos es minimizar la siguiente función:

$$\Phi(C,\mathcal{U}) = \text{máx}\left\{\frac{\Delta(C,\mathcal{U}\cap A_1)}{|A_1|}, ..., \frac{\Delta(C,\mathcal{U}\cap A_m)}{|A_m|}\right\},$$

donde $\mathcal{U} \cap A_i = \{U_1 \cap A_i, ..., U_k \cap A_i\}\ \forall i = 1, ..., m$.

Sea $\mathcal{U} = \{U_1, ..., U_k\}$ una partición de U, y sean μ_i^j la media de los puntos de $U_i \cap A_j$. Por un argumento similar al Lema 1, podemos concluir que para un conjunto de centros justos $C = \{c_1, ..., c_k\}$ con respecto a $\mathcal{U}$, c_i está en la envolvente convexa $\{\mu_i^1, ..., \mu_i^m\}$. Entonces podemos generalizar el problema convexo en la Ecuación 1 para m grupos demográficos de la siguiente manera:

$$\begin{aligned} &\text{mín}\ \theta \\ &\text{s.a}\ \ \frac{\Delta(M^j,\mathcal{U}\cap A_j)}{|A_j|} + \sum_{i\in[k]} \alpha_i^k \|c_i - \mu_i^j\|^2 \leq \theta, j \in [m] \\ &\qquad c_i \in Conv(\mu_i^1, ..., \mu_i^m) \end{aligned} \tag{3}$$

donde $\alpha_i^j = \frac{|U_i \cap A_j|}{A_j}$ y $M^j = \{\mu_1^j, ..., \mu_k^k\}$. El conjunto de puntos que encontremos resolviendo este problema, será el conjunto de centros juntos con respecto a $\mathcal{U}$.

Capítulo 3

k-means vs fair k-means

Pongamos en práctica lo ya visto en la sección anterior. Nuestra implementación del algoritmo Fair-Lloyd con dos grupos demográficos puede encontrarse en el Apéndice *A* o aquí. En el enlace también se hallan las entradas y las salidas de las pruebas de esta sección.

Vamos a trabajar con datos simulados. Consideremos el siguiente conjunto de datos, donde hay 500 individuos del grupo A y tan solo 50 del grupo B:

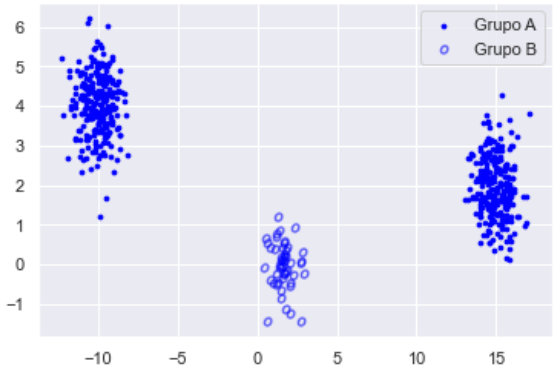


Figura 3.1

Intentemos separar la muestra en 2 clases. Si lo hacemos con el método k-means clásico, el resultado es el siguiente.

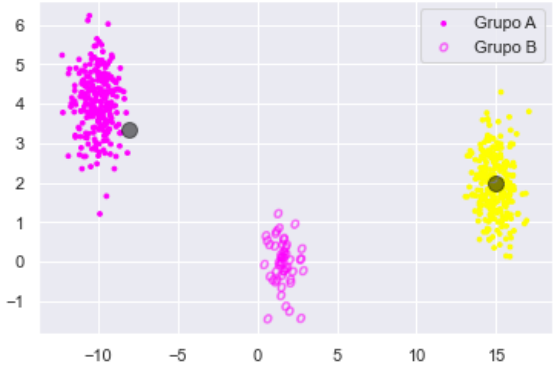


Figura 3.2

Podemos observar como los invididuos del grupo B se encuentran más lejos de su centro que aquellos que pertenecen al grupo A. Sin embargo, al usar el algoritmo en su versión "justa", pese a que en esta ocasión ningún individuo haya cambiado de clase, el centro correspondiente a la clase morada está más cerca de los individuos del grupo B.

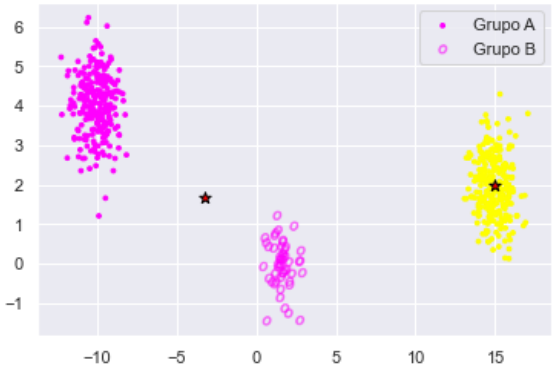


Figura 3.3

A continuación tenemos otro ejemplo donde se observa la misma tendencia:

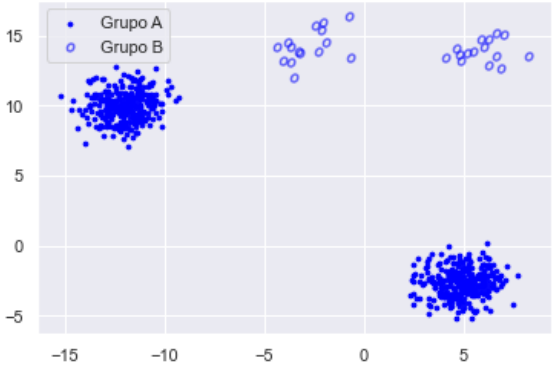
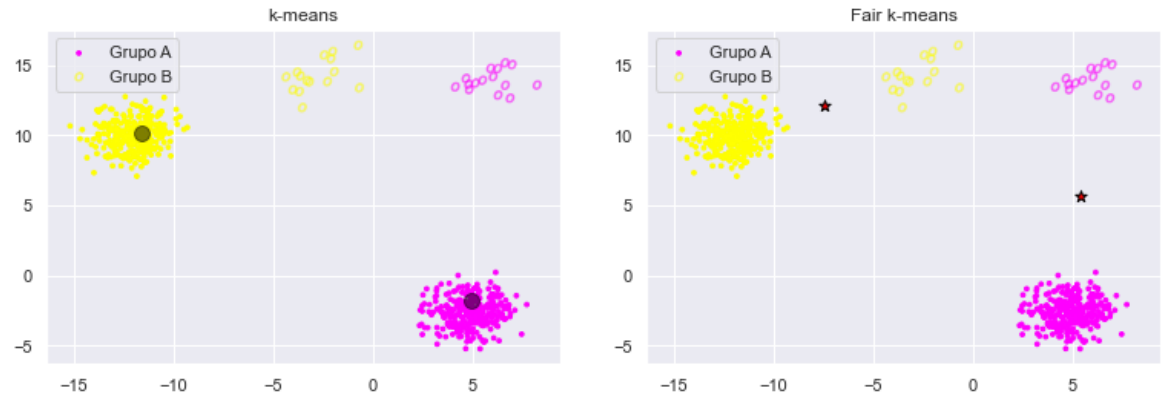


Figura 3.4



Capítulo 4

Conclusiones

La equidad se tiene cada vez más en consideración para el Machine Learning, incluyendo la clasificación y el clustering. Un método que se utiliza en el aprendizaje no supervisado es el k-means, pudiendo hacer de este un algoritmo "más justo" teniendo en cuenta la equidad.

Para ello se usa el algoritmo Fair Lloyd que asegura una solución óptima cercana a la solución del k-means, con mínimo aumento en el tiempo de ejecución manteniendo su sencillez, generalidad y estabilidad, basado en buscar un conjunto justo de centros.

En la búsqueda de centros justos se utiliza el Line Search, que como sugiere la Figura 2.2, estos criterios conducen a diferentes soluciones.

Se puede observar en los ejemplos vistos (véase Figura 3.3 y Figura 3.4) como, aplicando fair k-means en lugar de k-means, se obtienen las mismas clases pero con unos centros "más justos".

Ambas perspectivas son importantes, y la elección de qué agrupamiento usar dependerá del contexto y la aplicación.

Apéndice A

Código de Python

[1]:
```
#Librerias para cargar

import matplotlib.pyplot as plt
import seaborn as sns; sns.set()  # for plot styling
import numpy as np
from sklearn.datasets import make_blobs
import random
import math
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
import pandas as pd
```

[2]:
```
class FairKMeans:
    def __init__(self, k_clusters,tolerancia = 10e-10,max_iteraciones =␣
↪500,semilla = None,T_max = 64):
        self.k_clusters = k_clusters
        self.tolerancia = tolerancia
        self.max_iteraciones = max_iteraciones
        self.semilla = semilla
        self.T_max = T_max

    def clusterizar(self,clasificacion,cluster):
        indices = np.where(clasificacion == cluster)[0]
        return indices

    def asigna(self,U,tamaño_U,centros):
        clasificacion_puntos = np.array([-1 for a in range(tamaño_U)])
        distancia_a_centros = np.array([-1 for a in range(self.
↪k_clusters)],dtype = float)
        indice_punto = -1
        for punto in U:
            indice_punto +=1
            for i in range(self.k_clusters):
                distancia_a_centros[i] = np.linalg.norm(punto-centros[i,:])
            distancia_minima = min(distancia_a_centros)
            clase_seleccionada =  random.choice(np.where(distancia_a_centros␣
↪== distancia_minima)[0])
            clasificacion_puntos[indice_punto] = clase_seleccionada
```

```
        return clasificacion_puntos

    def centers(self,Group_A,Group_B):
        #Necesario para el algoritmo line search
        tamaño_A = Group_A.shape[0]
        tamaño_B = Group_B.shape[0]
        n_var_explicativas = Group_A.shape[1]

        #Juntamos los dos grupos en uno solo
        U = np.concatenate((Group_A,Group_B),axis=0)
        tamaño_U = tamaño_A+tamaño_B

        #Elegimos unos centros aleatorios antes de comenzar el algoritmo
        random.seed(self.semilla)
        random_indexes = sorted(random.sample(range(tamaño_U), self.
↪k_clusters))
        centros = U[random_indexes,:]

        #Bucle que realiza el algoritmo
        iteracion = 0
        error = self.tolerancia
        while error>=self.tolerancia  and iteracion< self.max_iteraciones:
            iteracion += 1
            #Asignamos a cada punto el centro más cercano para formar una␣
↪partición.
            clasificacion_puntos = self.asigna(U,tamaño_U,centros)

            #Algoritmo Line Search para encontrar los nuevos centros
            clasificacion_puntos_A = clasificacion_puntos[:tamaño_A]
            clasificacion_puntos_B = clasificacion_puntos[tamaño_A:]
            vector_alfas = np.array([-1 for a in range(self.k_clusters)],dtype␣
↪= float)
            vector_betas = np.array([-1 for a in range(self.k_clusters)],dtype␣
↪= float)
            mu_i_A = np.empty((self.k_clusters,n_var_explicativas),dtype=float)
            mu_i_B = np.empty((self.k_clusters,n_var_explicativas),dtype=float)
            vector_l = np.array([-1 for a in range(self.
↪k_clusters)],dtype=float)
            f_A = 0
            f_B = 0
            for i in range(self.k_clusters):
                A_U_i = self.clusterizar(clasificacion_puntos_A,i)
                B_U_i = self.clusterizar(clasificacion_puntos_B,i)
                tamaño_A_U_i = A_U_i.size
                tamaño_B_U_i = B_U_i.size
                vector_alfas[i] = tamaño_A_U_i/float(tamaño_A)
                vector_betas[i] = tamaño_B_U_i/float(tamaño_B)
                elementos_A_U_i = Group_A[A_U_i]
                elementos_B_U_i = Group_B[B_U_i]
                if tamaño_A_U_i == 0:
                    mu_i_A[i,:] = np.array([0 for a in␣
↪range(n_var_explicativas)],dtype = float)
                else:
                    mu_i_A[i,:] = elementos_A_U_i.mean(axis=0)
                if tamaño_B_U_i == 0:
```

```
                    mu_i_B[i,:] = np.array([0 for a in␣
↪range(n_var_explicativas)],dtype = float)
                else:
                    mu_i_B[i,:] = elementos_B_U_i.mean(axis=0)
                vector_l[i] = np.linalg.norm(mu_i_A[i,:]-mu_i_B[i,:])
                f_A_i = np.array([0 for a in range(tamaño_A_U_i)],dtype =␣
↪float)
                f_B_i = np.array([0 for a in range(tamaño_B_U_i)],dtype =␣
↪float)
                for j in range(tamaño_A_U_i):
                    f_A_i[j] = np.linalg.norm(elementos_A_U_i[j,:]-mu_i_A[i,:
↪])**2
                for j in range(tamaño_B_U_i):
                    f_B_i[j] = np.linalg.norm(elementos_B_U_i[j,:]-mu_i_B[i,:
↪])**2
                f_A += sum(f_A_i)
                f_B += sum(f_B_i)

            gamma = 0.5
            gammal = 1.
            gammah = 0.
            vector_x = np.array([0 for a in range(self.
↪k_clusters)],dtype=float)
            for t in range(1,self.T_max+1):
                gamma_conjugado = 1-gamma
                vector_x[:] = (gamma_conjugado*vector_betas[:]*vector_l[:])/
↪(gamma*vector_alfas[:]+gamma_conjugado*vector_betas[:])
                vectorlmx = vector_l-vector_x
                vectorlmx2 = vectorlmx**2
                f_A_x = (f_A/tamaño_A) + sum(vector_alfas*(vector_x**2))
                f_B_x = (f_B/tamaño_B) + sum(vector_betas*vectorlmx2)
                if abs(f_A_x-f_B_x)<self.tolerancia:
                    break
                elif f_A_x > f_B_x:
                    gammah = gamma
                    gamma = (gamma + gammal)/2.
                else:
                    gammal = gamma
                    gamma = (gamma + gammah)/2.
            centros_iteracion_anterior = np.copy(centros)
            for i in range(self.k_clusters):
                centros[i,:] = (vectorlmx[i]*mu_i_A[i,:]+vector_x[i]*mu_i_B[i,:
↪])/vector_l[i]

            error = np.linalg.norm(centros_iteracion_anterior-centros)
        if gamma<10e-10 or gamma == 1.0:
            print("Advertencia: No existe solución optima.")
        print("Valor de gamma = {}".format(gamma))
        return centros

    def predecir(self,Group_A,Group_B):
        centros = self.centers(Group_A,Group_B)
        U = np.concatenate((Group_A,Group_B),axis=0)
        tamaño_U = U.shape[0]
```

```
        return self.asigna(U,tamaño_U,centros)

    def grafica(self,Group_A,Group_B,s=30,labelA = 'Grupo A',labelB='Grupo␣
↪B',cmap='spring',marker_A = '.',marker_B = "$0$"):
        centros = self.centers(Group_A,Group_B)
        U = np.concatenate((Group_A,Group_B),axis=0)
        tamaño_U = U.shape[0]
        variables_U = U.shape[1]
        if variables_U != 2:
              raise Exception("Error")
        prediccion = self.asigna(U,tamaño_U,centros)
        fig, ax = plt.subplots()
        ax.scatter(Group_A[:, 0], Group_A[:, 1], c=prediccion[:Group_A.
↪shape[0]], s=s,label=labelA ,cmap=cmap,marker = marker_A)
        ax.scatter(Group_B[:, 0], Group_B[:, 1], c=prediccion[Group_A.shape[0]:
↪], s=s,label=labelB,alpha=0.4, cmap=cmap,marker = marker_B)
        ax.scatter(centros[:, 0], centros[:, 1], c='red', s=2*s,␣
↪edgecolors='black', marker = '*')
        ax.legend()
```

Bibliografía

[1] Mehrdad Ghadiri, Samira Samadi y Santosh Vempala. *Socially Fair k-Means Clustering*. 2020. URL: https://arxiv.org/abs/2006.10085.

[2] ML fairness gym project. *What is ML-fairness-gym?* 2019. URL: https://github.com/google/ml-fairness-gym.

[3] Facebook team. *How we're using Fairness Flow to help build AI that works better for everyone.* 2021. URL: https://www.ibm.com/blogs/research/2018/09/ai-fairness-360.

[4] Kush R. Varshney. *Introducing AI Fairness 360*. 2018. URL: https://www.ibm.com/blogs/research/2018/09/ai-fairness-360.