Instituto Tecnológico Autónomo de México

Matemática Computacional

Herramientas Matemáticas para el Análisis de Datos

Integrantes del equipo:

1. Tlatoani Real C.U: 181135
2. Daniela Gómez C.U: 172374
3. Paola San Martín C.U: 180328
4. Isaac Jair Jimenez C.U: 181724
5. Itzama Delgadillo García C.U: 181081

Noviembre 2020

Motivación

El rápido desarrollo de la informática ha impactado de muchas formas nuestro día a día: grandes empresas de electrónicos se ven desplazadas por los gigantes del *e-commerce*. Compramos en internet, consumimos series, películas y música a través de aplicaciones que no solo nos ofrecen este servicio, sino que con cada reproducción son capaces de ofrecer recomendaciones personalizadas a cada usuario. En suma, cada día estamos produciendo datos a través de las diversas aplicaciones que usamos, datos que, en las manos correctas, pueden transformarse en un enorme valor para una organización; no es coincidencia que las marcas estadounidenses más valiosas sean aquellas cuyo negocio se enfoca precisamente a la tecnología y a la informática. Pero la palabra *datos* no es sinónimo de *información*, los datos son un registro crudo de clicks, visitas y reproducciones que poco dicen por sí solos, es trabajo del analista procesar estos datos y convertirlos en información, en conocimiento que se traducirá en valor para su organización. Como en todo, las matemáticas juegan un papel fundamental en el análisis de datos, en este proyecto buscamos exponer dos algoritmos con aplicaciones tan diversas que abarcan desde el análisis del riesgo de crédito, hasta el diagnóstico del cáncer.

Modelo de Regresión Lineal

Consideremos m puntos en el plano, esto es $Q_1, Q_2, ..., Q_m \in \mathbb{R}^2$ en donde $Q_i = (x_i, y_i)\ \forall i \in \{1, 2, ..., m\}$. Buscamos encontrar una función, $h(x)$, que mejor se ajuste al conjunto de puntos dado.

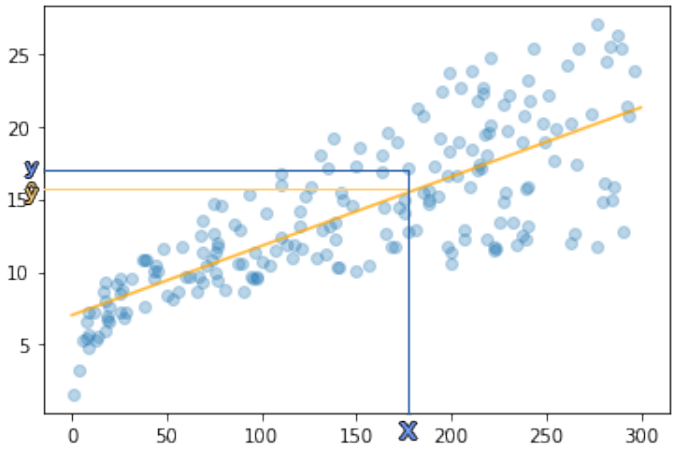


Figura 1: Regresión Lineal

Es importante notar que no buscamos que nuestra función h pase por cada uno de los m puntos Q, sino que refleje la tendencia general de todo el conjunto. Tomemos un punto $Q = (x, y)$ cualquiera, y llamemos $\hat{y}$ a la aproximación dada por nuestra función h en el punto x, esto es: $\hat{y} = h(x)$. Como es de esperarse, existe un error entre la aproximación $\hat{y}$, y y mismo, esto ocurre en general para cualquiera de nuestras Q. Aún no hemos dicho qué significa que nuestra función h se ajuste de "mejor forma" al conjunto de puntos. A continuación se define el error cuadrático medio.

Definición 1 (Error Cuadrático Medio)**.** *Sean $\hat{Y}, Y \in \mathbb{R}^m$ dos vectores tales que $\hat{Y}$ es una aproximación de Y,definimos el error cuadrático medio, denotado por ECM como: $ECM = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$.*

Para nuestro problema, la aproximación $\hat{y}_i$ está dada por nuestra función h evaluada en el punto x_i, esto es: $\hat{y}_i = h(x_i)$. De esta forma el error cuadrático medio está dado por la siguiente expresión: $ECM = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - h(x_i))^2$. Una vez que hemos definido esta medida de que tan buena es nuestra aproximación, podemos decir que nuestra función h es aquella que minimiza el error, el error cuadrático medio. Existe una variedad de formas que h puede tomar. Nosotros trataremos con el modelo de regresión lineal.

Regresión Lineal Simple

Una forma que h puede tomar es la de una función lineal, esto es $h(x) = ax + b$. En este caso, el i-ésimo término de la suma en el error cuadrático medio, está dado por: $(y_i - h(x_i))^2 = (y_i - (ax_i + b))^2$. Finalmente, el error cuadrático medio está dado por la siguiente expresión:

$$ECM = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - (ax_i + b))^2.$$

Bajo esta perspectiva, encontrar nuestra funcion $h(x) = ax + b$ significa encontrar los valores de (a, b) tales el ECM es mínimo.

Definamos $f : \mathbb{R}^2 \to \mathbb{R}$, como el error cuadrático medio en función de los parámetros a, b:

$$f(a, b) = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - (ax_i + b))^2.$$

Podemos encontrar los valores de (a,b) que minimizan el error cuadrático medio a través de las condiciones de primer orden de está función, esto es, encontrar los valores (a, b) tales que $\nabla f(a, b) = \bar{0}$.

Las derivadas parciales de f respecto a b y a son, respectivamente:

$$\partial_b f(a, b) = -\frac{2}{m} \cdot \sum_{i=1}^{m} (y_i - (ax_i + b)) \qquad \partial_a f(a, b) = -\frac{2}{m} \cdot \sum_{i=1}^{m} [(y_i - (ax_i + b)) \cdot x_i]$$

Igualando a cero la derivada parcial de f respecto a b:
$\partial_b f(a, b) = 0 \Leftrightarrow b \sum_{i=1}^{m} + a \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} y_i \Leftrightarrow b \cdot m + a \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} y_i$.

Igualando a cero la derivada parcial de f respecto a a:
$\partial_a f(a, b) = 0 \Leftrightarrow b \sum_{i=1}^{m} x_i + a \sum_{i=1}^{m} x_i^2 = \sum_{i=1}^{m} x_i y_i$.

Notamos que este es un sistema lineal, con forma matricial:

$$\begin{pmatrix} \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \\ m & \sum_{i=1}^{m} x_i \end{pmatrix} \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} x_i y_i \\ \sum_{i=1}^{m} y_i \end{pmatrix}.$$

Resolviendo este sistema lineal encontramos los valores de (a, b) tales que el error cuadrático medio es mínimo para el conjunto de puntos dados. De esta forma encontramos nuestra función h. Este razonamiento puede ser extendido a más dimensiones.

Regresión Lineal Múliple

Consideremos ahora m puntos en $\mathbb{R}^{n+1}$, esto es $Q_1, Q_2, ..., Q_m \in \mathbb{R}^{n+1}$ en donde $Q_i = (x_{i1}, x_{i2}, ..., x_{in}, y_i)$ $\forall i \in \{1, 2, ..., m\}$. Buscamos encontrar una función lineal, de la forma $L(x_1, ..., x_n) = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n$, que se aproxime de la mejor forma al conjunto de puntos dado. Para esto, consideramos el error cuadrático medio y la idea introducida para puntos en el plano:

$$ECM = \frac{1}{m} \cdot \sum_{i=1}^{m}(y_i - \hat{y_i})^2 = \frac{1}{m} \cdot \sum_{i=1}^{m}(y_i - L(x_{i1}, ..., x_{in}))^2 = \frac{1}{m} \cdot \sum_{i=1}^{m}(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j))^2.$$

Buscamos ahora los valores de $(w_0, w_1, ..., w_n)$ para los cuales el ECM es mínimo.

Definamos $f : \mathbb{R}^{n+1} \to \mathbb{R}$, como el error cuadrático medio, en función de los parámetros $w_0, w_1, ..., w_n$:

$$f(w_0, w_1, ..., w_n) = \sum_{i=1}^{m}(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j))^2.$$

Buscamos ahora los valores $(w_0, w_1, ..., w_n)$ tales que $\nabla f(w_0, w_1, ..., w_n) = \bar{0}$.
La derivada parcial de f respecto a w_0 es:

$$\partial_{w_0} f = -\frac{2}{m} \sum_{i=1}^{m}(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j)).$$

Ahora sea $k \in \{1, 2, ..., n\}$, entonces la derivada parcial respecto al k-ésimo parámetro es:

$$\partial_{w_k} f = -\frac{2}{m} \sum_{i=1}^{m}(y_i - (w_0 + \sum_{j=1}^{m} x_{ij}w_j)) \cdot \sum_{j=1}^{m} \partial_{w_k} x_{ij}w_j)) = -\frac{2}{m} \sum_{i=1}^{m}[(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j)) \cdot x_{ik}].$$

Igualando la derivada parcial de f respecto a w_0 a cero:

$$\begin{aligned}\partial_{w_0} f = -\frac{2}{m} \sum_{i=1}^{m}(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j)) = 0 &\Leftrightarrow w_0 \sum_{i=1}^{m} + \sum_{i=1}^{m}(\sum_{j=1}^{n} x_{ij}w_j) = \sum_{i=1}^{m} y_i \\ &\Leftrightarrow w_0 \sum_{i=1}^{m} + \sum_{j=1}^{n} w_j(\sum_{i=1}^{m} x_{ij}) = \sum_{i=1}^{m} y_i \\ &\Leftrightarrow w_0 m + w_1 \sum_{i=1}^{m} x_{i1} + w_2 \sum_{i=1}^{m} x_{i2} + ... + w_n \sum_{i=1}^{m} x_{in} = \sum_{i=1}^{m} y_i.\end{aligned}$$

Igualando la derivada parcial de f respecto a w_k a cero ($k \in \{1, 2, ..., n\}$):

$$\begin{aligned}\partial_{w_k} f = -\frac{2}{m} \sum_{i=1}^{m}[(y_i - (w_0 + \sum_{j=1}^{n} x_{ij}w_j)) \cdot x_{ik}] = 0 \\ \Leftrightarrow w_0 \sum_{i=1}^{m} x_{ik} + \sum_{i=1}^{m} x_{ik}(\sum_{j=1}^{n} x_{ij}w_j) = \sum_{i=1}^{m} x_{ik}y_i \\ \Leftrightarrow w_0 \sum_{i=1}^{m} x_{ik} + \sum_{j=1}^{n} w_j(\sum_{i=1}^{m} x_{ik}x_{ij}) = \sum_{i=1}^{m} x_{ik}y_i \\ \Leftrightarrow w_0 \sum_{i=1}^{m} x_{ik} + w_1 \sum_{i=1}^{m} x_{ik}x_{i1} + w_2 \sum_{i=1}^{m} x_{ik}x_{i2} + ... + w_n \sum_{i=1}^{m} x_{ik}x_{in} = \sum_{i=1}^{m} x_{ik}y_i.\end{aligned}$$

De esta forma encontramos el siguiente sistema de ecuaciones lineales:

1. $w_0 \sum_{i=1}^{m} x_{i1} + w_1 \sum_{i=1}^{m} x_{i1}x_{i1} + w_2 \sum_{i=1}^{m} x_{i1}x_{i2} + ... + w_n \sum_{i=1}^{m} x_{i1}x_{in} = \sum_{i=1}^{m} x_{i1}y_i$

2. $w_0 \sum_{i=1}^{m} x_{i2} + w_1 \sum_{i=1}^{m} x_{i2}x_{i1} + w_2 \sum_{i=1}^{m} x_{i2}x_{i2} + ... + w_n \sum_{i=1}^{m} x_{i2}x_{in} = \sum_{i=1}^{m} x_{i2}y_i$

.

.

.

n. $w_0 \sum_{i=1}^{m} x_{in} + w_1 \sum_{i=1}^{m} x_{in}x_{i1} + w_2 \sum_{i=1}^{m} x_{in}x_{i2} + ... + w_n \sum_{i=1}^{m} x_{in}x_{in} = \sum_{i=1}^{m} x_{in}y_i$

n+1. $w_0 m + w_1 \sum_{i=1}^{m} x_{i1} + w_2 \sum_{i=1}^{m} x_{i2} + ... + w_n \sum_{i=1}^{m} x_{in} = \sum_{i=1}^{m} y_i$

Recordemos que partimos de m puntos $Q_1, Q_2, ..., Qm \in \mathbb{R}^{n+1}$ en donde $Q_i = (x_{i1}, x_{i2}, ..., x_{in}, y_i)$ $\forall i \in \{1, 2, ..., m\}$. Podemos agrupar las primeras n componentes de cada punto en una matriz, donde en el renglón i, estarán los respectivos componentes de Q_i.

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

Entonces el sistema puede ser escrito en términos de suma de columnas y productos punto entre columnas de la matriz. Sea c_j la j-ésima columna de la matriz presentada arriba y S_j la suma de dicha columna.

1. $w_0 S_1 + w_1\langle c_1, c_1\rangle + w_2\langle c_1, c_2\rangle + ... + w_n\langle c_1, c_n\rangle = \langle c_1, y\rangle$

2. $w_0 S_2 + w_1\langle c_2, c_1\rangle + w_2\langle c_2, c_2\rangle + ... + w_n\langle c_2, c_n\rangle = \langle c_2, y\rangle$.
.
.

n. $w_0 S_n + w_1\langle c_n, c_1\rangle + w_2\langle c_n, c_2\rangle + ... + w_n\langle c_n, c_n\rangle = \langle c_n, y\rangle$

n+1. $w_0 m + w_1 S_1 + w_2 S_n + ... + w_n S_n = S_y$

Donde S_y es la suma del vector y. Finalmente, el sistema en su forma matricial es:

$$\begin{pmatrix} S_1 & \langle c_1, c_2\rangle & \langle c_1, c_2\rangle & \cdots & \langle c_1, c_n\rangle \\ S_2 & \langle c_2, c_1\rangle & \langle c_1, c_2\rangle & \cdots & \langle c_2, c_n\rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_n & \langle c_n, c_1\rangle & \langle c_n, c_2\rangle & \cdots & \langle c_n, c_n\rangle \\ m & S_1 & S_2 & \cdots & S_n \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} \langle c_1, y\rangle \\ \langle c_2, y\rangle \\ \vdots \\ \langle c_n, y\rangle \\ S_y \end{pmatrix} \tag{1}$$

Este sistema de ecuaciones puede ser resuelto empleando la descomposición LU de la matriz del sistema, junto a los algoritmos de sustitución hacia delante y hacia atrás.

Obtención de la matriz de coeficientes

Hemos dicho que resolveremos el sistema de ecuaciones, presentado arriba, usando los algoritmos de descomposición LU, sustitución hacia delante y hacia atrás, pero para hacer esto, primero debemos obtener la matriz de coeficientes del sistema, así como el vector con los términos constantes, para ello, presentamos el siguiente algoritmo escrito en *Python*:

```
def matriz_sistema(A,y):

    m = A.shape[0]
    n = A.shape[1]
    M = np.zeros((n+1,n+1))
    b=np.zeros(n+1)
    for i in range(n):
        b[i] = A[:,i].dot(y)
        M[i,0]=A[:,i].sum()
        M[n,i+1]=M[i,0]
        for j in range(i,n):
            M[i,j+1]=A[:,i].dot( A[:,j] )
            M[j,i+1]=M[i,j+1]
    b[n]=y.sum()
    M[n,0]=m

    return M,b
```

Donde A es la matriz de tamaño $m \times n$ el i-ésimo renglón contiene los componentes $x_1, x_2, ..., x_n$ del punto Q_i. La función entrega la matriz M y el vector b que determinan el sistema de ecuaciones (1), una vez obtenida la pareja (M, b), resolvemos el sistema para obtener los valores $(w_0, w_1, w_2, ..., w_n)$ de nuestra función $L(x_1, x_2, ..., x_n) = w_0 + \sum_{j=1}^{n} w_j x_j$.

Gasto en Publicidad

El siguiente conjunto de datos contiene 200 observaciones con información sobre el gasto en publicidad en distintas plataformas: televisión, radio y periódico, así como los ingresos por ventas en ese periodo.

```
In [2]: dataframe
```

Out[2]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

Figura 2: Gasto en Publicidad e Ingresos por Ventas

Con fines de poder realizar un gráfico en tres dimensiones, únicamente usaremos las columnas correspondientes al gasto en publicidad en televisión y radio, nuestra variable objetivo será el ingreso por ventas. Tomaremos 160 de las 200 observaciones, el 80 por ciento, para obtener, mediante una regresión lineal múltiple, una funcion lineal, $ventas(tv, rd) = \beta_0 + \beta_1 tv + \beta_2 rd$, posteriormente usaremos esta función para tratar de predecir el nivel de ventas para las 40 observaciones restantes. Llamaremos a las primeras 160 observaciones, el conjunto de entrenamiento; a las 40 restantes las llamaremos el conjunto de validación.

Primero importamos los paquetes de *Python*, *numpy*, para el manejo de matrices y vectores, *pandas* para trabajar con la tabla de datos y *matplotlib.pyplot* para realizar la gráfica.

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

Importamos el archivo en formato *.csv* donde se encuentran nuestros datos en publicidad.

```
In [10]: mainpath="C:\\Users\\itzam\\Desktop\\ITAM\\PythonML\\python-ml-course-master\\datasets"
         filepath="ads\\Advertising.csv"
         fullpath= os.path.join(mainpath,filepath)
         dataframe = pd.read_csv(fullpath)
```

Generamos una lista de indices con *numpy*.

```
In [11]: indices = np.arange(200)
         indices

Out[11]: array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
```

Con *shuffle* mezclamos la lista de indices, después tomamos los primeros 160 (que no estarán en orden) y de la tabla de datos original tomamos las observaciones cuyo índice este contenido en los primeros 160, los guardamos en el conjunto de entrenamiento, el que usaremos para obtener nuestra función. Tomamos el resto y los guardamos en el conjunto de validación, será este conjunto de datos el que usaremos para ver qué tan bien predice el modelo resultante.

```
In [14]: np.random.shuffle(indices)
         n_entrenamiento = int(160)
         data_entrenamiento=dataframe.iloc[indices[0:n_entrenamiento],:]
         data_validación=dataframe.iloc[indices[n_entrenamiento:],:]
```

Creamos una matriz, A, con 160 renglones y 2 columnas, correspondientes a las 160 observaciones con dos variables que hemos tomado, además en el vector y guardamos sus correspondientes niveles de ventas.

```
n [15]: A = np.zeros((160,2))
        A[:,0] = data_entrenamiento['TV']
        A[:,1] = data_entrenamiento['Radio']

        y = data_entrenamiento['Sales']
```

Obtenemos la matriz y el vector de constantes (M, b) del sistema de ecuaciones (1).

```
In [16]: M,b = matriz_sistema(A,y)
```

Realizamos la descomposición LU de la matriz M, esta funcion devuelve la matriz L y la matriz U en la primera y segunda posición (notar que los índices en*Python* comienzan en cero). Después mediante sustituciones hacia delante y hacia atrás obtenemos el vector de soluciones.

```
In [18]: LU = desc_LU(M)
         beta = sustitucion_delante(LU[0],b)
         beta = sustitucion_atras(LU[1],beta)
```

Una vez que tenemos el valor de los parámetros β, procedemos a graficar el plano que mejor se aproxima a nuestro conjunto de puntos.

Definimos nuestra función $ventas(tv, rd) = \beta_0 + \beta_1 tv + \beta_2 rd$. Además, creamos el dominio sobre el cual graficaremos el plano generado por nuestra función, y con una gráfica de dispersión colocaremos los puntos correspondientes a cada observación usada para la creación del modelo.

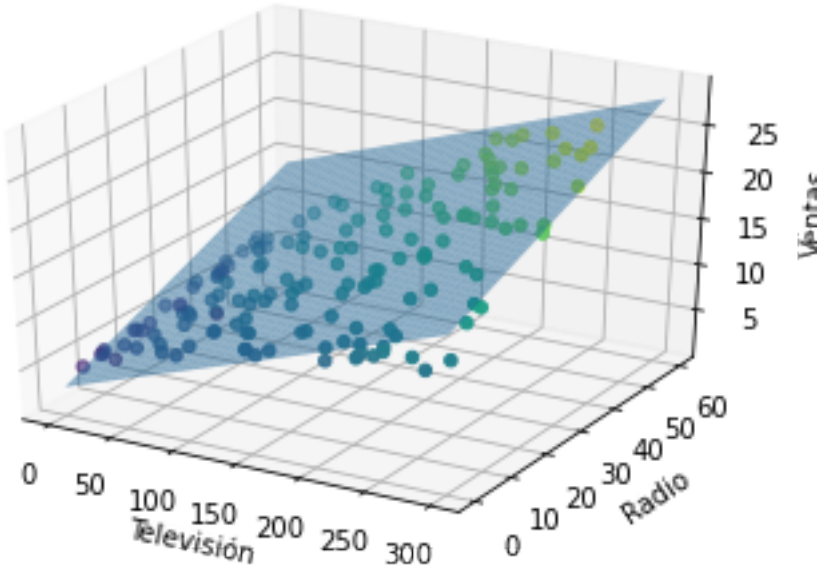
```
In [20]: def ventas(tv,rd):
             return beta[0]+beta[1]*tv+beta[2]*rd
```

```
In [23]: x_v = np.linspace(0,300,100)
         y_v = np.linspace(0,60,100)
         Xv,Yv = np.meshgrid(x_v,y_v)

         X1 = A[:,0]
         X2 = A[:,1]
         X3 = y

         fig = plt.figure()
         ax=fig.add_subplot(111,projection='3d',alpha=0.5)
         ax.scatter3D(X1,X2,X3,c=y)
         ax.plot_surface(Xv,Yv,ventas(Xv,Yv),alpha=0.5)
         plt.show()
```

A continuación, el gráfico correspondiente al plano generado por la función $ventas(tv, rd)$ en conjunto a las 160 observaciones:



Visualmente, parece que el plano se ajusta razonablemente bien al conjunto de datos. Sin embargo, como se mencionó al inicio de esta prueba, usaremos el resto de las observaciones para juzgar el desempeño de la regresión lineal.

A continuación se presentan algunas observaciones del conjunto de validación. La columna *Sales* corresponde al nivel de ventas real, la columna *ventas_m* corresponde al valor arrojado por la función $ventas(tv, rd)$. A simple vista podemos ver que hay valores para los cuales la predicción es buena, como la observación número 109, pero también observamos datos para los que parece un poco peor, como la 108.

Out[41]:

	TV	Radio	Newspaper	Sales	ventas_m
10	66.1	5.8	24.2	8.6	6.969377
16	67.8	36.6	114.0	12.5	12.926216
46	89.7	9.9	35.7	10.6	8.840525
179	165.6	10.0	17.6	12.6	12.360761
109	255.4	26.9	5.5	19.8	19.728593
113	209.6	20.6	10.7	15.9	16.413506
37	74.7	49.4	45.7	14.7	15.687482
108	13.1	0.4	25.6	5.3	3.493937
110	225.8	8.2	56.5	13.4	14.794150
171	164.5	20.9	47.4	14.5	14.390370
68	237.4	27.5	11.0	18.9	19.012796

Para una perspectiva más general calculemos el error relativo entre la aproximación dada por la función $ventas(tv, rd)$ y los datos reales de la columna. Para calcular el error relativo usaremos la norma uno del paquete *numpy*.

In [49]:
```
ventas = data_validación['Sales']
ventas_aprox = data_validación['ventas_m']

error = np.linalg.norm(ventas_aprox-ventas,1) / np.linalg.norm(ventas,1)
error*100
```

Out[49]: 7.937473010728501

Observamos que el error relativo de la aproximación es de casi el 8 por ciento, sin embargo, es importante mencionar que existen otros criterios importantes para juzgar qué tan bueno es un modelo de regresión lineal, esta teoría es estudiada en las asignaturas de estadística. Algo importante a resaltar es que solo usamos 2 de las 3 variables presentes en el conjunto de datos, la predicción podría mejorar, o no, si decidimos usar la tercera columna. Esto nos lleva a una pregunta interesante: ¿valdría la pena añadir la tercer variable? y no solo eso, ¿vale la pena usar las dos variables originales en la regresión lineal?, el gasto en publicidad en la radio podría no incidir tanto en el nivel de ventas como la publicidad en televisión. Estas preguntas son importantes, en primer lugar, por el costo computacional que supone añadir una columna más para un conjunto grande de observaciones, en segundo lugar porque en la práctica medir una variable más cuesta dinero y su importancia podría no compensar el costo de medirla periódicamente.

Clasificación de datos en k grupos

Una tarea frecuente en el análisis de datos consiste en agrupar un conjunto de datos en k grupos, donde k depende mucho del fin con el que se realiza esta división. Consideremos m puntos en $\mathbb{R}^n$, esto es $Q_1, Q_2, ..., Q_m \in \mathbb{R}^n$ en donde $Q_i = (x_{i1}, x_{i2}, ..., x_{in})$ $\forall i \in \{1, 2, ..., m\}$. Buscamos agrupar estos m puntos en k grupos, por su puesto, en una agrupación que tenga sentido.

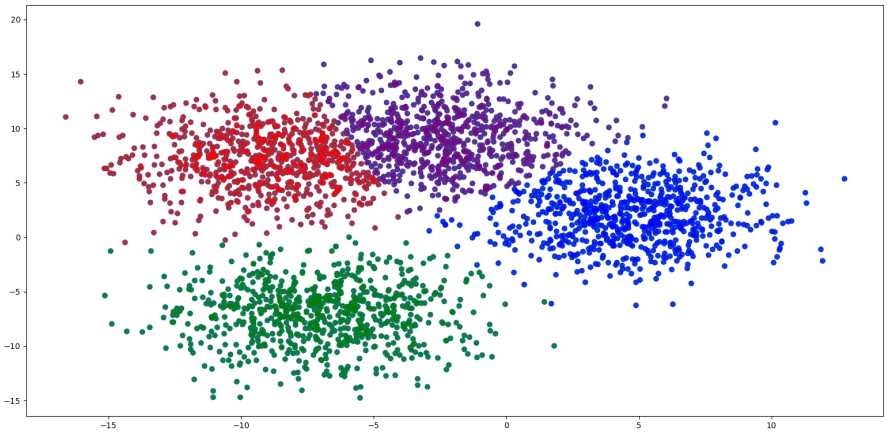


Figura 3: m=3000 , k=4

Dependiendo de la situación, en ocasiones es fácil observar el número k de grupos en los que puede divirse un conjunto de datos en dos dimensiones, sin embargo, al igual que la regresión lineal múltiple, este es un problema que puede (y suele) expandirse a más dimensiones. Las aplicaciones de esta segmentación son diversas, en general nos permtite dividir un conjunto de datos en grupos, en los cuales los integrantes de cada grupo comparten cierto grado de similitud: desde el reconocimiento de imágenes hasta el diagnóstico de cáncer.

Enumeramos los grupos con los índices $1, 2, ..., k$, especificamos a qué grupo pertenece cada uno de los m puntos Q a través de un vector $C \in \mathbb{R}^m$, donde cada entrada corresponde a un vector Q y el valor que tome esa entrada será el grupo al que pertenezca el vector correspondiente, esto es:

$$C = (c_1, ..., c_m),\ c_i = l \Leftrightarrow Q_i \text{ está en el grupo } l \text{ para } l \in \{1, 2, ..., k\},\ i \in \{1, 2, ..., m\}\ .$$

También definiremos k conjuntos correspondientes a cada grupo, cada conjunto contendrá los indices de los Q_i que estén en el respectivo grupo, es decir:

$$G_l = \{i : c_i = l\} \qquad \forall l \in \{1, 2, ..., k\}.$$

A cada grupo l se le asignará un vector representativo , llamaremos a estos vectores z, de esta forma z_l es el vector representativo del grupo l. Deseamos que, al ser vectores representativos, estos estén cerca de los miembros Q pertenecientes a ese grupo. Es decir, nos gustaría que las cantidades

$\|Q_i - z_{c_i}\|$ fueran pequeñas, por su puesto, esto depende de la propia dispersión de los datos y el número de grupos, k, en el que tratemos de dividirlos.

Al igual que en la regresión lineal, necesitamos un número que nos diga qué tan buena o mala es nuestra clasificación. Para este propósito, para cada Q_i calcularemos la distancia con el vector representativo del grupo al que pertenece, z_{c_i}, elevaremos al cuadrado y tomaremos el promedio de ellas, llamaremos a esa cantidad J_{clust}.

$$J_{clust} = \frac{1}{m} \sum_{i=1}^{m} \|Q_i - z_{c_i}\|^2.$$

Mientras más pequeña sea esta cantidad, mejor será nuestra elección de grupos. Lo ideal sería encontrar el vector $C = (c_1, ..., c_m)$ y los vectores $z_1, z_2, ..., z_n$ que minimizan J_{clust}, esto resulta complicado y nada común en la práctica. Sin embargo, existe un algoritmo iterativo, conocido como *k-means*, que nos proporciona una solución que en muchos casos resulta ser bastante buena.

Notamos que en J_{clust} la elección del grupo para Q_i, es decir c_i, afecta solo al término de la suma donde aparece Q_i, $\|Q_i - z_{c_i}\|^2$. Entonces, para minimizar J_{clust} simplemente tenemos que minimizar cada uno de estos términos, eligiendo correctamente c_i. Asignaremos el valor de $c_i = l$ de forma que $\|Q_i - z_{c_i}\|^2$ sea mínima, esto es:

$$\|Q_i - z_{c_i}\|^2 = \min \|Q_i - z_l\|^2 \quad l \in \{1, 2..., k\}.$$

Es importante notar que para este paso se ha supuesto que la elección de $z_1, z_2, ..., z_k$ está dada. Ahora veremos cómo escoger a los vectores representativos z cuando la asignación de grupos está dada.

Notamos que J_{clust} puede ser escrito de la siguiente manera: $J_{clust} = \sum_{l=1}^{k} J_l = J_1 + J_2 + ... + J_k$ donde J_l está dado por:

$$J_l = \frac{1}{m} \sum_{i \in G_l} \|Q_i - z_l\|^2.$$

Es decir J_l es la contribución del grupo l a la suma de J_{clust}. Ademásm, notamos que la elección de z_l solo afecta a J_l en la suma, entonces la elección de z_l para minimizar J_{clust} es la elección de z_l que minimiza J_l. Esto se soluciona escogiendo a z_l como el centro geométrico de los Q pertenecientes a ese grupo:

$$z_l = \frac{1}{|G_l|} \sum_{i \in G_l} Q_i$$

en donde $|G_l|$ es el número de elementos en el grupo l. Importante observar que para realizar este paso se ha supuesto que la asignación de grupos dado por C está dada.

El algoritmo k-means

En los parrafos anteriores se han dicho dos cosas: en primer lugar, dada la selección de vectores representativos $z_1, ..., z_k$, elegimos para cada $i \in \{1, 2, ..., m\}$ el grupo de Q_i de forma que $\|Q_i - z_{c_i}\|^2 = \min \|Q_i - z_l\|^2$, $l \in \{1, 2..., k\}$; en segundo lugar, una vez dada la configuración de grupos, indicada por las entradas de C, escogemos los vectores z representativos de cada grupo, tomando a z_l como el centro geométrico de los Q_i que conforman el grupo. Es fácil notar que estos dos pasos dependen uno del otro.

El algoritmo *k-means* consiste en realizar de forma iterativa estos dos pasos, en cada iteración la medida J_{clust} debe mejorar, a menos que en un paso la elección de los grupos o los vectores

representativos no cambien en nada, en la programación tendremos cuidado de este caso. Otra cuestión a tomar en cuenta es el estado inicial de la asignación, tomaremos al azar k vectores Q como los elementos representativos iniciales, después procederemos con el paso uno.

Algoritmo *k-means*

1. Escoger los grupos a los que pertenece cada uno de los Q_i, dada la elección de los vectores representativos.

2. Escoger los vectores representativos de cada grupo, z, dada la asignación de grupos.

Como se mencionó con anterioridad, el algoritmo *k-means* nos brinda una solución aproximada a la mejor clasificación en k grupos que puede existir, pero el resultado final del algoritmo también depende en cierta medida del estado inicial. Al igual que en la regresión lineal, más que el algoritmo numérico, existe otra serie de factores que inciden en la calidad del resultado final: el número k que depende de la propia necesidad de quien emplea el algoritmo, la elección del estado inicial, el número de iteraciones que el algoritmo realiza, entre otros.

Implementación

A continuación se presenta la implementación en *Python*. Para hacer más fácil el problema dividiremos el código en cinco funciones, una de ellas será la principal que mande a llamar al resto.

1. **Función distancia mínima**
   Esta función recibe como primer parámetro un vector x de dimensión n, como segundo parametro, una matriz Z donde el l-ésimo renglón será el vector representativo del grupo l, z_l. Durante su ejecución encuentra el vector z_l tal que su distancia a x es mínima y regresa el número l, este será el valor que próximamente tome la entrada c_i del vector C.

```
def dist_min(x,Z):
    k = Z.shape[0]
    dist_min = np.linalg.norm(x-Z[0],2)
    indice = 0
    for l in range(k):
        if(np.linalg.norm(Z[l]-x,2)<dist_min):
            dist_min = np.linalg.norm(Z[l]-x,2)
            indice = l
    return indice
```

2. **Función elige grupo**
   Esta función recibe como primer parámetro una matriz X de tamaño $m \times n$, donde el i-ésimo renglón contiene a l vector Q_i. Como segundo parámetro, recibe una matriz Z donde el l-ésimo renglón será el vector representativo del grupo l, z_l. Durante su ejecución crea el vector $C = (c_1, ..., c_n)$ de la siguiente manera: para cada Q_i, con ayuda de la función *distancia minima*, encuentra el valor de l tal que la distancia de Q_i al vector z_l es mínima, luego asigna este número al correspondiente c_i, es decir, $c_i = l$. Regresa como resultado el vector C.

```
def elige_grupo(X,Z):
    m = X.shape[0]
    C = np.zeros(m)
    for i in range(m):
        C[i] = dist_min(X[i],Z)
    return C
```

3. **Función elige *Z***
   Esta función recibe como primer parámetro una matriz X descrita como en el punto anterior, como segundo parámetro recibe el vector con las asignaciones de grupo de cada vector Q, así como el número de vectores representativos deseados. Durante su ejecución, para cada $l \in \{1, 2, ..., k\}$ toma de la matriz X aquellos vectores Q_i tales que pertenecen al grupo l y realiza una suma (vectorial). Posteriormente calcula el total de vectores en el grupo l y asigna a z_l el centro geométrico de los miembros del grupo. Regresa la matriz Z con los vectores representativos.

```
def elige_Z(X,C,k):

    n = X.shape[1]
    Z = np.zeros((k,n))

    for l in range(k):
        suma_vectores = X[C==l].sum(axis=0)
        cardinal_grupo = np.where(C==l,1,0).sum()
        Z[l] = (suma_vectores)/(cardinal_grupo)

    return Z
```

4. **Función inicializa**
   Esta función recibe una matriz X descrita como antes y el número de grupos, k. Durante su ejecución genera una lista de índices, despues los barajea y toma k vectores Q correspondientes a los primeros k índices después del barajeo. Devuelve la matriz Z con esta elección de vectores representativos.

```
def inicializa(X,k):

    m = X.shape[0]
    n = X.shape[1]
    Z = np.zeros((k,n))

    indices = np.arange(m)
    np.random.shuffle(indices)
    elegidos = indices[:k]

    for l in range(k):
        Z[l]=X[elegidos[l]]

    return Z
```

5. **Función K-means**
   Esta es la función principal. Recibe en la primera entrada la matriz X, descrita como antes, como segunda entrada recibe el número de grupos deseados, k, y, finalmente, el número de iteraciones que se desean realizar. Durante su ejecución manda a llamar primero a la función *inicializa* que se encarga de la elección inicial de los vectores z. En esta función también definimos tres asignaciones diferentes para C, esto con el propósito de tener memoria sobre las dos asignaciones anteriores y así poder terminar el ciclo si en algún momento la asignacion de C no cambia o comienza a alternar entre dos asignaciones siempre iguales. Como se mencionó antes, el propósito fundamental de esta función es iterar entre los dos pasos descritos del algoritmo *k-means*. La función regresa el vector de asignaciones, C.

```
def K_means(X,k,iteraciones):
    Z = inicializa(X,k)
    m = X.shape[0]
    c_ant = np.random.randint(0,100,m)
    c_act = np.random.randint(0,100,m)
    c_nuevo = np.random.randint(0,100,m)
    for i in range(iteraciones):
        if( not((c_nuevo==c_act).all()) and not((c_nuevo==c_ant).all()) ):
            c_ant = c_act
            c_act = c_nuevo
            c_nuevo = elige_grupo(X,Z)
            Z = elige_Z(X,c_nuevo,k)
    return c_nuevo.astype(dtype=int)
```

Prueba de la implementación

Además de los paquetes usuales importamos de *sklearn* la función *make blobs* que nos ayudará a generar un conjunto de datos para poner a prueba nuestra implementación.

```
In [9]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.datasets import make_blobs
```

Creamos una muestra de 5,000 datos, con 3 centros y además guardamos su asignación real para compararla con la asignación de nuestro algoritmo.

```
In [11]: datos, asignacion_real = make_blobs(
          n_samples=5000,
          centers=3,
          cluster_std=2.5,
          random_state=42
         )
```

A continuación la gráfica de dispersión:

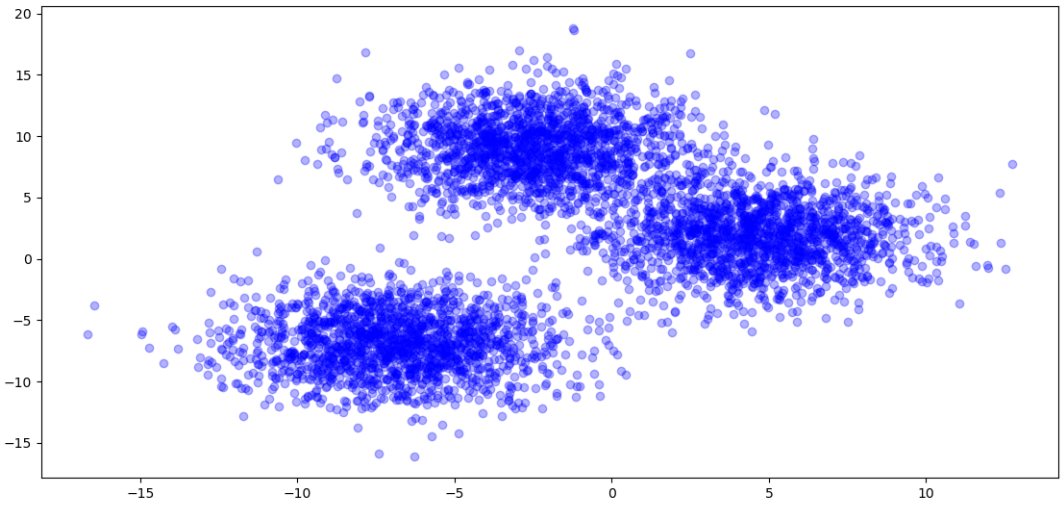


Figura 4: Gráfica de dispersión sin agrupación de datos

Pondremos en marcha nuestra implementación del algoritmo *k-means*:

```
In [20]: C= K_means(datos,3,500)
         grupoA = datos[C==0]
         grupoB = datos[C==1]
         grupoC = datos[C==2]
```

Figura 5: Llamada a nuestra función *k-means*

Realizamos una gráfica de dispersión, esta vez asignando un color a cada grupo.

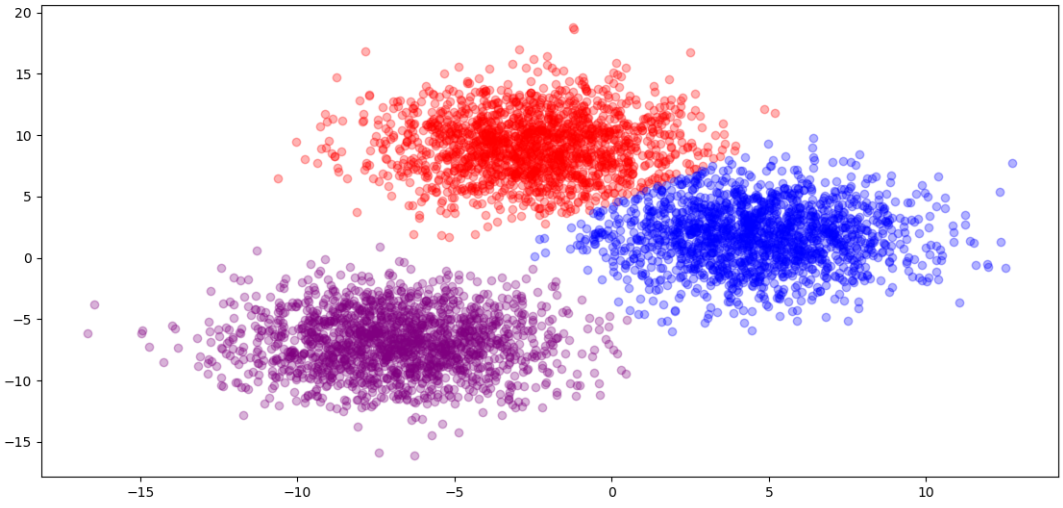


Figura 6: Gráfica de dispersión después de la agrupación

Comparamos visualmente esta gráfica con la gráfica de la clasificación real, dada al inicio del problema.

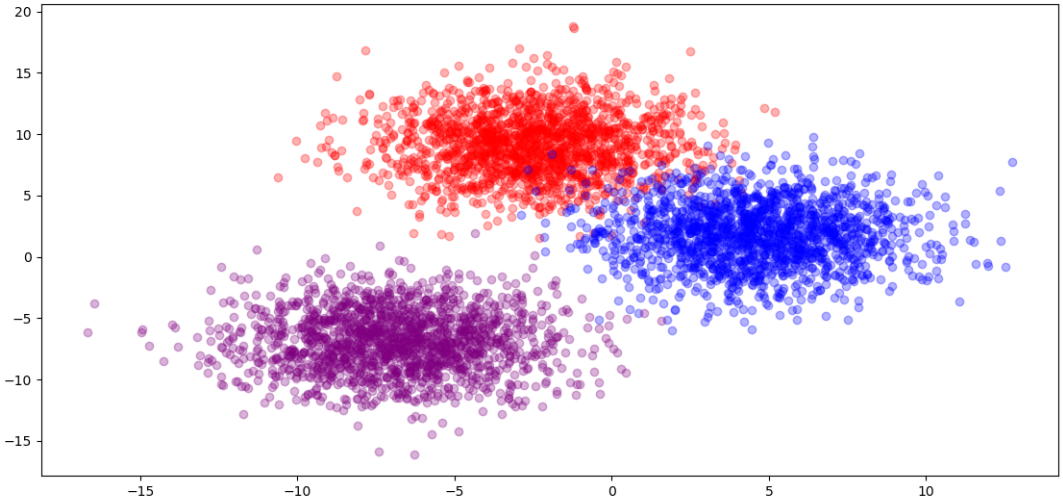


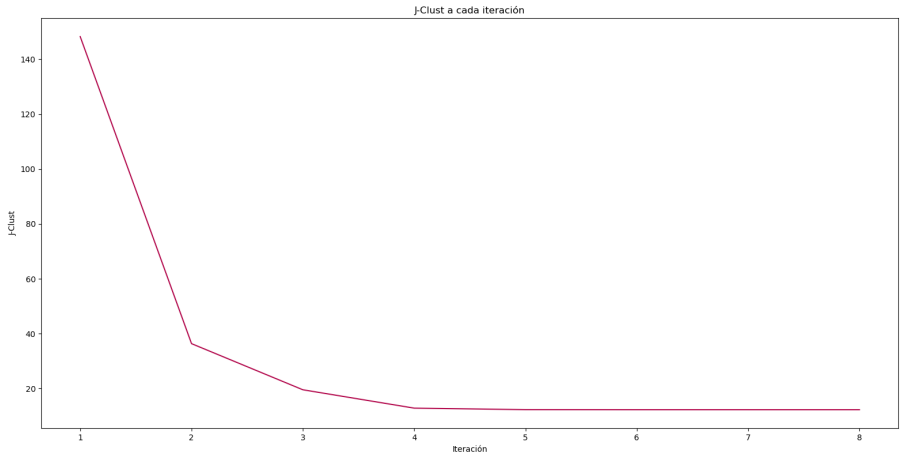
Figura 7: Gráfica de dispersión con la asignación real de grupos

La asignación, visualmente, parece ser razonable, sin embargo, podemos comparar esta asignación dada por el vector C con la asignación real obtenida al inicio.

```
In [49]: aciertos = (C==asignacion_real).sum()
         porcentaje = aciertos/5000
         porcentaje*=100
         porcentaje
Out[49]: 98.66
```

Como podemos ver, hemos acertado en el 98 por ciento de los casos, además, visualmente podemos observar que hemos fallado en una de las zonas más difíciles, donde se juntan el grupo azul y el grupo rojo. Un comentario importante acerca de la instrucción que acabamos de ejecutar es que la etiqueta de los grupos no es necesariamente la misma, es decir, podría ocurrir que el algoritmo haya realizado correctamente la clasificación, incluyendo a cada vector en el grupo que le corresponde, pero que su etiquetado sea diferente, esto sería fácil de observar si existiera una discrepancia en el patrón de colores de las dos gráficas anteriores.

Un detalle importante a notar es que hemos corrido el algoritmo con 500 iteraciones, pero en verdad ¿cuántas ha realizado? Con algunas modificaciones al código podemos guardar el número de iteraciones y el valor de J_{clust} a cada paso. El siguiente gráfico nos muestra el número de iteraciones y el valor de J_{clust} en cada uno de ellos.



En la gráfica observamos que el algoritmo apenas ha realizado 8 iteraciones en este experimento, por su puesto esto no siempre será así, además, el problema es relativamente sencillo en este caso. Sin embargo, sí podemos decir que, partiendo de un estado aleatorio, en 8 pasos ha terminado su ejecución porque llegó a un punto fijo, un punto donde su asignación ya no mejoraría en el siguiente paso. Como última prueba, realizaremos 10 veces el experimento y en cada uno de ellos tomamos el número de iteraciones al final de la ejecución.

```
In [46]: #Realizamos el experimento 10 veces para observar el número de observaciones que realiza.
         iteraciones = np.zeros(10)
         for i in range(10):
             prueba = K_means(datos,3,500)
             iteraciones[i] = max(prueba[1])

         iteraciones

Out[46]: array([ 9.,  6.,  5.,  7., 10., 21.,  7.,  6.,  7.,  7.])
```

Observamos que si bien el número de iteraciones varia, pues hay que considerar que en cada experimento partimos de un estado diferente, el número de pasos, para este conjunto de datos, no sobrepasa los 30.

Sin duda existen distintos métodos y mejores implementaciones para resolver los problemas aquí planteados. Los algoritmos aquí expuestos buscan ser una primera e intuitiva aproximación a dos herramientas muy útiles en el análisis de datos, área que en los últimos años representa una salida profesional natural para egresados de carreras como matemáticas y matemáticas aplicadas. Lenguajes como *R* y*Python* son frecuentemente utilizados para este tipo de tareas, pues además de una sintaxis sencilla, tienen el respaldo de una gran comunidad que crea y comparte implementaciones de toda clase de algoritmos para el análisis de datos, lo que lo hace ideal para personas que cuentan con un conocimiento técnico, pero no se encuentran tan inmersos en el área de la informática. Por último, quizá la lección más importante de este proyecto, es que además del conocimiento teórico de las matemáticas, es importante incorporar a nuestro repertorio de herramientas el uso de un lenguaje de programación que nos permitirá transformar nuestro conocimiento teórico en aplicaciones capaces de resolver una gran variedad de problemas.

Bibliografía

[1] Zabala Tejada, Ion, *"Ampliación de Métodos Numéricos"*, Capítulo 8 "El problema de mínimos cuadrados", septiembre 2018. En: http://www.ehu.eus/izaballa/Ana_Matr/Apuntes/lec8.pdf Consultado el 20 de noviembre del 2020.

[2] Boyd, Stephen; Vandenberghe, Lieven, *"Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares"*, Cambridge United Kingdom, Cambridge University Press, 2018. Consultado el 10 noviembre del 2020.

[3] Grabinsky, Guillermo , apuntes clases de Cálculo Diferencial e Integral III, Primavera 2020 ITAM, 12 de marzo del 2020. Consultado el 10 de noviembre del 2020.

[4] Castañeda, Pablo , pdf capítulo 4 y 6 libro de clase Matemática Computacional, Otoño 2020 ITAM. Consultado el 15 de noviembre del 2020.

[5] M. Vidal, Antonio, *"El Problema Lineal de Mínimos Cuadrados: Descomposición QR"*, Murcia Diciembre 2007. "El problema de mínimos cuadrados", septiembre 2018. En: http://http://dis.um.es/ domingo/apuntes/CAP/0708/QRPar.pdf Consultado el 20 de noviembre del 2020.