Universidad Nacional Autónoma de México

Facultad de Ingeniería

PRÁCTICA 6
MÉTRICAS DE DISTANCIA

Minería de Datos

Profesor:

Dr. Molero Castillo Guillermo Gilberto

Grupo 1

Alumna:

Monroy Velázquez Alejandra Sarahí

No. Cuenta: 314000417

OBJETIVO

Obtener las matrices de distancia (Euclidiana, Chebyshev, Manhattan, Minkowski) a partir de una matriz de datos.

DESARROLLO

El conjunto de datos corresponde a diversas variables relacionada a las hipotecas. Este conjunto de datos incluye:

- ingresos: son ingresos mensuales de 1 o 2 personas, si están casados.
- gastos_comunes: son gastos mensuales de 1 o 2 personas, si están casados.
- pago_coche
- gastos_otros
- ahorros
- vivienda: valor de la vivienda.
- estado_civil: 0-soltero, 1-casado, 2-divorciado
- hijos: cantidad de hijos menores (no trabajan).
- trabajo: 0-sin trabajo, 1-autonomo, 2-asalariado, 3-empresario, 4-autonomos, 5-asalariados, 6-autonomo y asalariado, 7-empresario y autonomo, 8-empresarios o empresario y autónomo
- comprar: 0-alquilar, 1-comprar casa a través de crédito hipotecario con tasa fija a 30 años.

Primero comenzamos la importación de bibliotecas correspondientes que nos ayudarán para la realización del código, las cuales son *pandas* para la manipulación y análisis de datos, *numpy* para crear vectores y matrices, *matplotlib* para la generación de gráficas, así como *seaborn* para la visualización de datos. En esta práctica agregamos una biblioteca más, la cual es *scipy* que nos ayudara para el cálculo de distancias. Por último, la biblioteca files para subir el archivo *csv*.

Una vez importadas, el *dataframe* se lee y se despliega en pantalla:

```
import pandas as pd                          # Para la manipulación y análisis de datos
import numpy as np                           # Para crear vectores y matrices n dimensionales
import matplotlib.pyplot as plt              # Para generar gráficas a partir de los datos
from scipy.spatial.distance import cdist     # Para el cálculo de distancias
from scipy.spatial import distance
```

```
from google.colab import files
files.upload()
```

```
Elegir archivos  Hipoteca.csv
• Hipoteca.csv(application/vnd.ms-excel) - 8014 bytes, last modified: 30/9/2021 - 100% done
Saving Hipoteca.csv to Hipoteca.csv
{'Hipoteca.csv': b'ingresos,gastos_comunes,pago_coche,gastos_otros,ahorros,vivienda,estado_civil,hi
```

```
Hipoteca = pd.read_csv("Hipoteca.csv")
Hipoteca
```

	ingresos	gastos_comunes	pago_coche	gastos_otros	ahorros	vivienda	estado_civil	hijos	trabajo	comprar
0	6000	1000	0	600	50000	400000	0	2	2	1
1	6745	944	123	429	43240	636897	1	3	6	0
2	6455	1033	98	795	57463	321779	2	1	8	1
3	7098	1278	15	254	54506	660933	0	0	3	0
4	6167	863	223	520	41512	348932	0	0	3	1
...	...	...	...	...	...	...	...	...	...	...
197	3831	690	352	488	10723	363120	0	0	2	0
198	3961	1030	270	475	21880	280421	2	3	8	0
199	3184	955	276	684	35565	388025	1	3	8	0
200	3334	867	369	652	19985	376892	1	2	5	0
201	3988	1157	105	382	11980	257580	0	0	4	0

202 rows × 10 columns

Ahora que el *dataframe* está cargado, comenzamos con los pasos vistos en clase:

Para observar si existen datos faltantes utilizamos *info()*, por lo que observamos que todas las variables se encuentran integras.

```
[4]  Hipoteca.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202 entries, 0 to 201
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ingresos        202 non-null    int64
 1   gastos_comunes  202 non-null    int64
 2   pago_coche      202 non-null    int64
 3   gastos_otros    202 non-null    int64
 4   ahorros         202 non-null    int64
 5   vivienda        202 non-null    int64
 6   estado_civil    202 non-null    int64
 7   hijos           202 non-null    int64
 8   trabajo         202 non-null    int64
 9   comprar         202 non-null    int64
dtypes: int64(10)
memory usage: 15.9 KB
```

Estandarización de los datos

En este paso estandarizamos o normalizamos el rango de las variables iniciales, para que cada una de ellas contribuya por igual al análisis. Para ello instanciamos el objeto StandardScaker, calculamos media y

desviación estándar de cada variable y escalamos los datos. Guardamos en una nueva variable llamada *MEstandarizada*.

```
[5] from sklearn.preprocessing import StandardScaler, MinMaxScaler
    estandarizar = StandardScaler()                    # Se instancia el objeto StandardScaler o MinMaxScaler
    MEstandarizada = estandarizar.fit_transform(Hipoteca)     # Se calcula la media y desviación, y se escalan los datos
```

```
[6] pd.DataFrame(MEstandarizada)
```

	0	1	2	3	4	5	6	7	8	9
0	0.620129	0.104689	-1.698954	0.504359	0.649475	0.195910	-1.227088	0.562374	-0.984420	1.419481
1	1.063927	-0.101625	-0.712042	-0.515401	0.259224	1.937370	-0.029640	1.295273	0.596915	-0.704483
2	0.891173	0.226266	-0.912634	1.667244	1.080309	-0.379102	1.167809	-0.170526	1.387582	1.419481
3	1.274209	1.128886	-1.578599	-1.559015	0.909604	2.114062	-1.227088	-0.903426	-0.589086	-0.704483
4	0.719611	-0.400042	0.090326	0.027279	0.159468	-0.179497	-1.227088	-0.903426	-0.589086	1.419481
...	...	...	...	...	...	...	...	...	...	...
197	-0.671949	-1.037402	1.125381	-0.163554	-1.617963	-0.075199	-1.227088	-0.903426	-0.984420	-0.704483
198	-0.594508	0.215214	0.467439	-0.241079	-0.973876	-0.683130	1.167809	1.295273	1.387582	-0.704483
199	-1.057368	-0.061099	0.515581	1.005294	-0.183849	0.107880	-0.029640	1.295273	1.387582	-0.704483
200	-0.968013	-0.385305	1.261783	0.814462	-1.083273	0.026040	-0.029640	0.562374	0.201581	-0.704483
201	-0.578424	0.683102	-0.856468	-0.795686	-1.545397	-0.851037	-1.227088	-0.903426	-0.193753	-0.704483

Después de ello estamos listos para calcular las matrices de distancia.

Matrices de distancia

EUCLIDIANA

La distancia euclidiana es una de las métricas más utilizadas para calcular la distancia entre dos puntos, conocida también como espacio euclidiano. Sus bases se encuentran en la aplicación del Teorema de Pitágoras, donde la distancia viene a ser la longitud de la hipotenusa. Calculamos la matriz de distancia euclidiana:

```
[7] DstEuclidiana = cdist(MEstandarizada, MEstandarizada, metric='euclidean')
    MEuclidiana = pd.DataFrame(DstEuclidiana)
```

```
[8] print(MEuclidiana)
    #MEuclidiana
```

```
           0         1         2    ...       199       200       201
0    0.000000  3.797535  3.804493  ...  4.561513  4.691204  4.167673
1    3.797535  0.000000  4.438534  ...  3.534064  4.002834  4.619277
2    3.804493  4.438534  0.000000  ...  4.035627  4.692546  5.370211
3    4.038794  3.395203  5.722746  ...  5.762757  5.793142  4.439623
4    2.525802  4.221725  3.896767  ...  4.392829  3.891753  3.529423
..        ...       ...       ...  ...       ...       ...       ...
197  4.824041  4.871541  5.943808  ...  4.101772  2.602537  2.920367
198  5.153131  3.842371  4.327305  ...  2.128398  2.477457  3.962922
199  4.561513  3.534064  4.035627  ...  0.000000  1.861647  4.177486
200  4.691204  4.002834  4.692546  ...  1.861647  0.000000  3.618619
201  4.167673  4.619277  5.370211  ...  4.177486  3.618619  0.000000

[202 rows x 202 columns]
```

Podemos utilizar la función *round()* para redondear los decimales, en este caso los redondeamos a tres decimales después del punto:

```
[9]  print(MEuclidiana.round(3))
```

```
        0      1      2      3      4    ...    197    198    199    200    201
0    0.000  3.798  3.804  4.039  2.526  ...  4.824  5.153  4.562  4.691  4.168
1    3.798  0.000  4.439  3.395  4.222  ...  4.872  3.842  3.534  4.003  4.619
2    3.804  4.439  0.000  5.723  3.897  ...  5.944  4.327  4.036  4.693  5.370
3    4.039  3.395  5.723  0.000  4.276  ...  5.392  6.008  5.763  5.793  4.440
4    2.526  4.222  3.897  4.276  0.000  ...  3.360  4.780  4.393  3.892  3.529
..     ...    ...    ...    ...    ...  ...    ...    ...    ...    ...    ...
197  4.824  4.872  5.944  5.392  3.360  ...  0.000  4.358  4.102  2.603  2.920
198  5.153  3.842  4.327  6.008  4.780  ...  4.358  0.000  2.128  2.477  3.963
199  4.562  3.534  4.036  5.763  4.393  ...  4.102  2.128  0.000  1.862  4.177
200  4.691  4.003  4.693  5.793  3.892  ...  2.603  2.477  1.862  0.000  3.619
201  4.168  4.619  5.370  4.440  3.529  ...  2.920  3.963  4.177  3.619  0.000

[202 rows x 202 columns]
```

Ahora calculamos la matriz de distancias de una parte del total de objetos:

```
[10] DstEuclidiana = cdist(MEstandarizada[0:10], MEstandarizada[0:10], metric='euclidean')
     MEuclidiana = pd.DataFrame(DstEuclidiana)
     print(MEuclidiana)
```

```
          0         1         2  ...         7         8         9
0  0.000000  3.797535  3.804493  ...  3.266670  2.580767  4.539208
1  3.797535  0.000000  4.438534  ...  3.990354  4.623600  4.146656
2  3.804493  4.438534  0.000000  ...  4.800687  3.691077  3.765019
3  4.038794  3.395203  5.722746  ...  3.421725  4.268658  4.867024
4  2.525802  4.221725  3.896767  ...  3.823082  1.609706  3.729517
5  2.812974  3.334202  4.080909  ...  5.323531  4.155028  3.676100
6  3.947750  3.816188  4.145313  ...  5.100542  2.732431  1.876685
7  3.266670  3.990354  4.800687  ...  0.000000  3.841435  5.966857
8  2.580767  4.623600  3.691077  ...  3.841435  0.000000  3.640088
9  4.539208  4.146656  3.765019  ...  5.966857  3.640088  0.000000

[10 rows x 10 columns]
```

Y calculamos la distancia entre dos objetos, en este caso entre las primeras dos filas de la matriz *MEstandarizada* que es nuestro conjunto de datos estandarizado:

```
[11] Objeto1 = MEstandarizada[0]
     Objeto2 = MEstandarizada[1]
     dstEuclidiana = distance.euclidean(Objeto1,Objeto2)
     dstEuclidiana
```

```
3.797535116785011
```

CHEBYSHEV

La distancia de Chebyshev es el valor máximo absoluto de las diferencias entre las coordenadas de un par de elementos. Otro nombre para la distancia de Chebyshev es métrica máxima. Calculamos la matriz de distancia de Chebyshev:

```
[12] DstChebyshev = cdist(MEstandarizada, MEstandarizada, metric='chebyshev')
     MChebyshev = pd.DataFrame(DstChebyshev)
```

```
[13] print(MChebyshev)
```

```
            0         1         2    ...       199       200       201
0    0.000000  2.123964  2.394897  ...  2.372002  2.960737  2.194872
1    2.123964  0.000000  2.316473  ...  2.121295  2.031939  2.788408
2    2.394897  2.316473  0.000000  ...  2.123964  2.174417  2.625706
3    2.123964  2.198699  3.226259  ...  2.564310  2.840382  2.965099
4    1.789280  2.198699  2.394897  ...  2.198699  2.123964  2.123964
..        ...       ...       ...  ...       ...       ...       ...
197  2.824335  2.198699  2.698272  ...  2.372002  1.465800  1.981849
198  2.394897  2.620501  2.123964  ...  1.246374  1.197448  2.394897
199  2.372002  2.121295  2.123964  ...  0.000000  1.186001  2.198699
200  2.960737  2.031939  2.174417  ...  1.186001  0.000000  2.118251
201  2.194872  2.788408  2.625706  ...  2.198699  2.118251  0.000000

[202 rows x 202 columns]
```

Ahora calculamos la matriz de distancias de una parte del total de objetos:

```
[14] DstChebyshev = cdist(MEstandarizada[0:10], MEstandarizada[0:10], metric='chebyshev')
     MChebyshev = pd.DataFrame(DstChebyshev)
     print(MChebyshev)
```

```
          0         1         2  ...         7         8         9
0  0.000000  2.123964  2.394897  ...  2.123964  1.676949  2.394897
1  2.123964  0.000000  2.316473  ...  2.198699  2.542659  2.299425
2  2.394897  2.316473  0.000000  ...  2.767335  2.394897  3.279931
3  2.123964  2.198699  3.226259  ...  3.148734  2.719351  2.476117
4  1.789280  2.198699  2.394897  ...  2.123964  1.425772  2.394897
5  1.639965  2.123964  2.802850  ...  2.931599  2.931599  2.198699
6  2.768169  2.123964  2.898266  ...  2.820741  1.562440  1.197448
7  2.123964  2.198699  2.767335  ...  0.000000  2.317671  3.202405
8  1.676949  2.542659  2.394897  ...  2.317671  0.000000  2.394897
9  2.394897  2.299425  3.279931  ...  3.202405  2.394897  0.000000

[10 rows x 10 columns]
```

Y calculamos la distancia entre dos objetos, en este caso entre las primeras dos filas de nuestro conjunto de datos:

```
Objeto1 = MEstandarizada[0]
Objeto2 = MEstandarizada[1]
dstChebyshev = distance.chebyshev(Objeto1,Objeto2)
dstChebyshev
```

```
2.1239636695175896
```

MANHATTAN

La distancia euclidiana es una buena métrica. Sin embargo, en la vida real, por ejemplo, en una ciudad, es imposible moverse de un punto a otro de manera recta. Se utiliza la distancia de Manhattan si se necesita calcular la distancia entre dos puntos en una ruta similar a una cuadrícula (información geoespacial). Calculamos la matriz de distancia de Manhattan:

```
[16] DstManhattan = cdist(MEstandarizada, MEstandarizada, metric='cityblock')
     MManhattan = pd.DataFrame(DstManhattan)
```

```
[17] print(MManhattan)
```

```
            0          1          2    ...        199        200        201
0     0.000000  10.424141   8.847472  ...  11.906421  11.759006  11.541748
1    10.424141   0.000000  11.599318  ...   7.973369  10.001368  11.631632
2     8.847472  11.599318   0.000000  ...  10.864423  13.306994  14.376265
3    10.025386   8.760185  15.070180  ...  16.652502  16.505087   9.599311
4     5.597122  10.910516   9.665707  ...  11.646667  10.687158   9.046640
..         ...        ...        ...  ...        ...        ...        ...
197  12.054727  13.728162  17.321540  ...  10.525713   6.547757   6.067082
198  14.363146   9.270791  10.733106  ...   4.812175   6.758826   9.276845
199  11.906421   7.973369  10.864423  ...   0.000000   4.250761  11.694123
200  11.759006  10.001368  13.306994  ...   4.250761   0.000000   9.584179
201  11.541748  11.631632  14.376265  ...  11.694123   9.584179   0.000000

[202 rows x 202 columns]
```

Ahora calculamos la matriz de distancias de una parte del total de objetos:

```
[18] DstManhattan = cdist(MEstandarizada[0:10], MEstandarizada[0:10], metric='cityblock')
     MManhattan = pd.DataFrame(DstManhattan)
     print(MManhattan)
```

```
           0          1          2  ...         7          8          9
0   0.000000  10.424141   8.847472  ...   7.738225   5.611823  11.881937
1  10.424141   0.000000  11.599318  ...   9.695689  12.626546  11.067069
2   8.847472  11.599318   0.000000  ...  10.679192   8.879890   7.046665
3  10.025386   8.760185  15.070180  ...   5.972322   9.129148  12.206451
4   5.597122  10.910516   9.665707  ...   9.578937   2.837118   8.760076
5   6.427056   7.989831   9.654680  ...  13.887734  10.092330   9.510240
6   9.618351   9.562377   9.875240  ...  14.008775   7.132801   4.489027
7   7.738225   9.695689  10.679192  ...   0.000000   9.177593  16.710664
8   5.611823  12.626546   8.879890  ...   9.177593   0.000000   8.280457
9  11.881937  11.067069   7.046665  ...  16.710664   8.280457   0.000000

[10 rows x 10 columns]
```

Y calculamos la distancia entre dos objetos, en este caso entre las primeras dos filas de nuestro conjunto de datos:

```
[19] Objeto1 = MEstandarizada[0]
     Objeto2 = MEstandarizada[1]
     dstManhattan = distance.cityblock(Objeto1,Objeto2)
     dstManhattan
```

```
10.424141070900212
```

MINKOWSKI

La distancia de Minkowski es una distancia entre dos puntos en un espacio n-dimensional. Es una métrica de distancia generalizada: Euclidiana, Manhattan y Chebyshev. Calculamos la matriz de distancia de Minkowski:

```
[20] DstMinkowski = cdist(MEstandarizada, MEstandarizada, metric='minkowski', p=1.5)
     MMinkowski = pd.DataFrame(DstMinkowski)
```

```
[21] print(MMinkowski)
```

```
            0         1         2    ...       199       200       201
0    0.000000  5.230769  4.898143  ...  6.180060  6.259283  5.782133
1    5.230769  0.000000  6.008218  ...  4.581278  5.365428  6.178405
2    4.898143  6.008218  0.000000  ...  5.546062  6.552322  7.359566
3    5.380078  4.577987  7.767533  ...  8.164713  8.153774  5.613723
4    3.194333  5.684360  5.170442  ...  5.976566  5.392557  4.762823
..        ...       ...       ...  ...       ...       ...       ...
197  6.456871  6.837178  8.422537  ...  5.518374  3.484710  3.653702
198  7.165865  5.054451  5.822982  ...  2.757183  3.428694  5.157947
199  6.180060  4.581278  5.546062  ...  0.000000  2.407156  5.832106
200  6.259283  5.365428  6.552322  ...  2.407156  0.000000  4.928902
201  5.782133  6.178405  7.359566  ...  5.832106  4.928902  0.000000

[202 rows x 202 columns]
```

Ahora calculamos la matriz de distancias de una parte del total de objetos:

```
22] DstMinkowski = cdist(MEstandarizada[0:10], MEstandarizada[0:10], metric='minkowski', p=1.5)
    MMinkowski = pd.DataFrame(DstMinkowski)
    print(MMinkowski)
```

```
          0         1         2  ...         7         8         9
0  0.000000  5.230769  4.898143  ...  4.251992  3.276410  6.152703
1  5.230769  0.000000  6.008218  ...  5.260852  6.347660  5.663025
2  4.898143  6.008218  0.000000  ...  6.191502  4.830481  4.472055
3  5.380078  4.577987  7.767533  ...  3.924650  5.421550  6.534552
4  3.194333  5.684360  5.170442  ...  5.127385  1.876635  4.862207
5  3.636612  4.337667  5.280479  ...  7.195193  5.463708  4.949609
6  5.183831  5.116112  5.411805  ...  7.025383  3.708546  2.467307
7  4.251992  5.260852  6.191502  ...  0.000000  5.046566  8.299101
8  3.276410  6.347660  4.830481  ...  5.046566  0.000000  4.667053
9  6.152703  5.663025  4.472055  ...  8.299101  4.667053  0.000000

[10 rows x 10 columns]
```

Y calculamos la distancia entre dos objetos, en este caso entre las primeras dos filas de nuestro conjunto de datos:

```
[23] Objeto1 = MEstandarizada[0]
     Objeto2 = MEstandarizada[1]
     dstMinkowski = distance.minkowski(Objeto1,Objeto2, p=1.5)
     dstMinkowski
```

```
5.230769189641202
```

HAMMING

La distancia de Hamming se denomina así por su inventor Richard Hamming, quien introdujo el término para establecer una métrica capaz de contar el número de desvíos en cadenas de igual longitud y estimar el error (distancia de señal). A continuación, varios ejercicios:

```
[24] # Sean dos cadenas
     cadena_1 = 'euclidean'
     cadena_2 = 'manhattan'
```

```
[25] dstHamming = distance.hamming(list('euclidean'), list('manhattan'))*len('euclidean')
     dstHamming
```

```
     7.0
```

```
[26] # Cadenas
     cadena_1 = 'Distancias'
     cadena_2 = 'Distorsión'
```

```
[27] dstHamming = distance.hamming(list(cadena_1), list(cadena_2))*len(cadena_1)
     dstHamming
```

```
     5.0
```

```
[28] # Cadenas
     cadena_1 = 'La División de Ingeniería Eléctrica tiene siete Departamentos Académicos '
     cadena_2 = 'La División de Ingeniería Mecánica e Industrial tiene cinco Departamentos'
```

```
[29] dstHamming = distance.hamming(list(cadena_1), list(cadena_2))*len(cadena_1)
     dstHamming
```

```
     45.0
```

CONCLUSIÓN

La métrica de distancias es una herramienta muy importante dentro de la minería de datos, ya que nos ayuda a mejorar en gran manera el rendimiento de los procesos, ya sean los de clasificación, clusterización, recuperación de información entre otras aplicaciones. Como en todos los casos, tenemos a nuestra disposición diversas formas de medir distancias, por lo que es cuestión de analizar cual es la mejor para aplicarla en los datos con los que trabajemos.