Cervantes Rubí Brandon

Inteligencia Artificial 2022-2, G.3, 02/03/2022
Practica No.2 Reglas de Asociación (Algoritmo Apriori)

Objetivo.

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

Características.

La fuente de datos tiene las siguientes características.

- 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.

Euclidiana.

$$\boldsymbol{dist}\,(\boldsymbol{a},\boldsymbol{b}) = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{p_i} - \boldsymbol{q_i})^2}$$

Chebyshev.

$$\boldsymbol{dist}\,(\boldsymbol{a},\boldsymbol{b}) = \boldsymbol{max}\,|\boldsymbol{p_i} - \boldsymbol{q_i}|$$

Manhattan.

$$\boldsymbol{dist}\,(\boldsymbol{a},\boldsymbol{b}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} |\boldsymbol{p_i} - \boldsymbol{q_i}|$$

Minkowsky.

$$\boldsymbol{dist}\,(\boldsymbol{a},\boldsymbol{b}) = (\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{q_i} - \boldsymbol{p_i})^{\boldsymbol{\lambda}})^{\frac{\boldsymbol{1}}{\boldsymbol{\lambda}}}$$

Desarrollo.

Durante esta práctica se trabajaron 4 matrices de distancias para calcular el nivel de similitud entre los registros de clientes con información acerca de sus ingresos y sus gastos, dichas matrices de distancia fueron calculadas con las métricas de distancia Euclidiana, Chebyshev, Manhattan y Minkowsky donde cada uno de estos métodos arroja índices de distancias diferentes para utilizarlas en diferentes algoritmos dependiendo de la situación. A continuación, se muestran los registros de datos abiertos utilizados.

	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

Para obtener estas matrices, el primer paso es importar los módulos necesarios para manejar los datos que se lean (**`Pandas`**), para trabajar con matrices numéricas más cómodamente (**`numpy`**), **`matplotlib`** para realizar las gráficas de las distribuciones de los datos y después vienen dos nuevos módulos para trabajar con estas distancias los cuales son **`cdist`** para obtener una matriz completa de distancias a partir de la matriz de registros y **`distance`** para obtener la distancia entre dos registros de la matriz completa de datos.

```
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
```

Una vez que se leen los datos se tiene que generar un **`DataFrame`** para dejar todos los datos en una sola estructura con la que podamos trabajar, se utilizó el módulo **`cdist`** para sacar la matriz de distancias con cada una de las métricas de distancia. Este módulo recibe 3 parámetros, los primeros 2 corresponden a los **`dataframes`** sobre los cuales se van a estar sacando las distancias (Para este caso los dos son los mismos porque las distancias se sacan entre los mismos registros de la matriz) y el ultimo parámetro es la métrica de distancia con la que se van a sacar las distancias, para esta práctica las métricas que se utilizaron fueron.

- euclidian, para distancias euclidianas.
- chebyshev, para distancias con chebyshev.
- cityblock, para distancias con Manhattan.
- minkowsky, para distancias con Minkowsky.

Para calcular la distancia Euclidiana se utilizó el módulo **`cdist`** como ya se mencionó de la siguiente manera.

```
DstEuclidiana = cdist(Hipoteca, Hipoteca, metric='euclidean')
MEuclidiana = pd.DataFrame(DstEuclidiana)
```

La cual no arroja la siguiente matriz de distancias.

	0	1	2	3	\
0	0.000000	236994.701964	78577.840350	260974.591407	
1	236994.701964	0.000000	315439.176808	26550.527773	
2	78577.840350	315439.176808	0.000000	339168.030097	
3	260974.591407	26550.527773	339168.030097	0.000000	
4	51769.581416	287970.807817	31494.808048	312273.311450	
..	...	...	...	...	
197	53923.596347	275716.907131	62456.926862	301032.758046	
198	122858.123985	357126.266127	54616.719848	381921.267013	
199	18967.999420	249015.957900	69848.439181	273593.155481	
200	37975.571227	261065.405879	66722.600009	286156.617026	
201	147421.532182	380612.957023	78717.767975	405600.560294	

Como se puede apreciar, los resultados de las distancias se calculan con muchos decimales por lo que es conveniente redondear esta matriz de datos con el método **`round`**, sin embargo, el módulo **`cdist`** regresa un arreglo de **`numpy`** por lo cual primero es conveniente pasar este arreglo a un **`dataframe`** y después usar el round para redondear a tantos números decimales.

```
print(MEuclidiana.round(3))
```

Además, es común que no siempre se necesite calcular la matriz completa sino, solamente la distancia entre un grupo de objetos o registros en la matriz completa de datos. Para ello podemos utilizar el método **`iloc`** de los **`dataframes`** para particionar la matriz original en una matriz más pequeña y señalando entre corchetes los índices de inicio y fin, donde en el fin se pone un numero después de donde queremos que termine de tal manera que si queremos los primeros 10 registros los corchetes irán [0:10] y la sub-matriz que se entregara será con los registros del 0 al 9.

```
DstEuclidiana = cdist(Hipoteca.iloc[0:10], Hipoteca.iloc[0:10], metric='euclidean')
#iloc para seleccionar filas y columnas según su posición
MEuclidiana = pd.DataFrame(DstEuclidiana)
print(MEuclidiana.round(3))
```

Y la matriz de distancias euclidianas para los primeros 5 que entregaría será la siguiente.

	0	1	2	3	4
0	0.000	236994.702	78577.840	260974.591	51769.581
1	236994.702	0.000	315439.177	26550.528	287970.808
2	78577.840	315439.177	0.000	339168.030	31494.808
3	260974.591	26550.528	339168.030	0.000	312273.311
4	51769.581	287970.808	31494.808	312273.311	0.000

Para calcular la distancia entre un par de registros de la matriz de datos utilizamos el módulo **`distance`**, el cual toma como parámetros los 2 arreglos de datos de 1 x N donde N es la cantidad de atributos o características y entrega solamente el índice de distancia entre esos dos objetos que representa que tan diferentes o similares son los objetos.

```
Objeto1 = Hipoteca.iloc[0]
Objeto2 = Hipoteca.iloc[1]
dstEuclidiana = distance.euclidean(Objeto1,Objeto2)
dstEuclidiana
```

Lo cual tiene como resultado el índice 236994.70196398906 que corresponde al índice de la distancia entre el objeto 0 y el objeto 1 de la matriz anterior.

Las próximas distancias para calcular se hicieron de la misma manera, solamente se cambió el parámetro de la métrica por `'chebyshev'`, `'cityblock'` y `'minkowsky'` donde para esta última además se definió el parámetro lambda (p) con 1.5 recordando que con este parámetro se consigue una combinación entre la distancia euclidiana y manhattan. A continuación, se muestran los resultados de cada una.

Chebishev. (No es necesario redondear porque son números enteros)

	0	1	2	3	4
0	0.0	236897.0	78221.0	260933.0	51068.0
1	236897.0	0.0	315118.0	24036.0	287965.0
2	78221.0	315118.0	0.0	339154.0	27153.0
3	260933.0	24036.0	339154.0	0.0	312001.0
4	51068.0	287965.0	27153.0	312001.0	0.0

Manhattan. (No es necesario redondear porque son números enteros)

	0	1	2	3	4
0	0.0	244759.0	86474.0	267180.0	60166.0
1	244759.0	0.0	330117.0	36279.0	290551.0
2	86474.0	330117.0	0.0	343632.0	43970.0
3	267180.0	36279.0	343632.0	0.0	326816.0
4	60166.0	290551.0	43970.0	326816.0	0.0

Minkowsky. (Es recomendable redondear debido a que contiene números flotantes)

	0	1	2	3	4
0	0.000	237690.996	79782.467	261389.574	53372.216
1	237690.996	0.000	317144.542	28999.550	288074.734
2	79782.467	317144.542	0.000	339376.379	34836.105
3	261389.574	28999.550	339376.379	0.000	313818.957
4	53372.216	288074.734	34836.105	313818.957	0.000

Es posible además obtener las distancias entre registros individuales, como, por ejemplo, el registro con id 0 con respecto al registro con id 1. Para ello se hace uso del módulo `distance` que se importó inicialmente, solo que a diferencia de utilizar el método `euclidian` que es para distancias euclidianas se utilizan los siguientes módulos.

- *distance.chebishev*
- *distance.cityblock*
- *distance.minkowsky*

Como podemos observar en las matrices obtenidas, realmente cada una de ellas si bien nos ayudan a representar la similitud que tienen los objetos con un índice numérico cada una de ellas lo hace de diferente manera. Por ejemplo para la distancia *Euclidiana* con respecto a *Chebyshev* observamos que la primera de ellas nos representa las distancias con una mayor exactitud ya que se consideran todas y cada una de las variables en el conjunto de datos para obtener la raíz de la suma de los cuadrados mientras que con *Chebyshev* solamente se va a considerar la variable que tenga una mayor distancia es por eso que pareciera que el número se truncara o estuviese muy cercano con respecto a la *Euclidiana*, porque no se consideran todas las variables para el análisis.

Un fenómeno similar pasa si comparamos la matriz de *Chebyshev* con respecto a la de *Manhattan* pues en ésta última también se considera la diferencia con respecto a todas las variables mientras que para *Chebyshev* solamente la distancia mayor por lo cual *Manhattan* tiende a tener distancias más altas que *Chebyshev* e inclusive que la *Euclidiana* en algunas ocasiones debido a que para *Manhattan* no es necesario obtener alguna raíz de la suma de las distancias como se hace en distancias *Euclidianas* lo cual puede ocasionar que se reduzca el índice de distancia.

O por ultimo la distancia de *Minkowsky* comparada con la *Euclidiana* y la de *Manhattan* observamos que el índice de distancia que nos entrega *Minkowsky* siempre cae prácticamente a un valor intermedio entre la distancia *Euclidiana* y la de *Manhattan* debido al índice lambda que se define produciendo un equilibrio entre ambas. Debemos tener en cuenta que cada una de estas matrices están enfocadas a servir como la entrada diferentes algoritmos donde cada uno de ellos tiene como propósito resolver problemáticas diferentes.

Un detalle importante por resaltar es el hecho de que los índices de distancias que se nos entregan con estos algoritmos son números muy grandes, esto debido a la naturaleza de los datos, estamos hablando de ingresos, ahorros, gastos, entre otros los cuales no suelen ser números pequeños. Esto hace sentido en la vida real, sin embargo, no es practico para usarlos como entrada a nuestros algoritmos de inteligencia artificial y por ello tenemos técnicas como la estandarización y la normalización de los datos las cuales nos van a ayudar para optimización y legibilidad en nuestros algoritmos.

La normalización consiste en reescalar los valores a un numero entre 0 y 1, esto lo podemos implementar en Python con el módulo **`MinMaxScaler`** de **`sklearn`**, mientras que la estandarización consiste en transformar los valores de modo que se ajusten a una media de 0 y a una desviación estándar de 1, en Python es practico utilizar el módulo **`StandardScaler`** para llevar a nuestra matriz inicial de datos a cumplir con estas condiciones.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
estandarizar = StandardScaler()
MEstandarizada = estandarizar.fit_transform(Hipoteca)
```

Como podemos ver el procedimiento que se sigue es importar primero las bibliotecas ya mencionadas para después crear un objeto con dichos módulos, en este caso se está siguiendo un proceso de estandarización lo que quiere decir que ahora los datos tendrán una media de 0 y una desviación estándar de 1, también implica que se va a reducir su escala llegando a tener inclusive números negativos. A continuación, se muestra la matriz inicial de datos estandarizada en comparación con la matriz original de datos.

Original.

	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

Estandarizada.

	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

202 rows × 10 columns

Lo primero que podemos notar es que todos los valores en la matriz han sido reescalados a números con una menor magnitud llegando a tener inclusive números negativos, pero si observamos con atención veremos que mantienen una relación y un sentido con respecto a los valores anteriores. Por ejemplo, en la primera columna un 6000 se vuelve 0.62 mientras que un 6745 se hace 1.06 y un 3831 se va a -0.67.

Con estos datos ahora el siguiente paso es obtener las matrices de distancias que ya se vinieron trabajando en la parte anterior de la misma manera. A continuación, se muestran los resultados.

Euclidiana.

	0	1	2	3	4
0	0.000	3.798	3.804	4.039	2.526
1	3.798	0.000	4.439	3.395	4.222
2	3.804	4.439	0.000	5.723	3.897
3	4.039	3.395	5.723	0.000	4.276
4	2.526	4.222	3.897	4.276	0.000

Chebyshev

	0	1	2	3	4
0	0.000	2.124	2.395	2.124	1.789
1	2.124	0.000	2.316	2.199	2.199
2	2.395	2.316	0.000	3.226	2.395
3	2.124	2.199	3.226	0.000	2.294
4	1.789	2.199	2.395	2.294	0.000

Manhattan.

	0	1	2	3	4
0	0.000	10.424	8.847	10.025	5.597
1	10.424	0.000	11.599	8.760	10.911
2	8.847	11.599	0.000	15.070	9.666
3	10.025	8.760	15.070	0.000	10.506
4	5.597	10.911	9.666	10.506	0.000

Minkowsky.

	0	1	2	3	4
0	0.000	5.231	4.898	5.380	3.194
1	5.231	0.000	6.008	4.578	5.684
2	4.898	6.008	0.000	7.768	5.170
3	5.380	4.578	7.768	0.000	5.719
4	3.194	5.684	5.170	5.719	0.000

Además de la escala, otra de las diferencias que hay en utilizar matrices con datos estandarizados es que ahora los valores son mas comparables, por ejemplo la distancia *Euclidiana* entre los objetos 0 y 1 sin este procedimiento es de 236994.702 comparándola con la de 0 y 2 que es de 78577.840 mientras que con ayuda de la estandarización estos valores se vuelven 3.798 y 3.804 los cuales ya son más fáciles de comparar a simple vista y computacionalmente menos costosos representando el mismo grado de similitud entre los vectores de datos.

Además de que se puede ver un poco mejor la diferencia entre la distancia *Manhattan* y *Euclidiana* debido a que como son números mas pequeños al aplicar una raíz cuadrada en distancias *Euclidianas* se notara de mejor manera como se alejan sus valores con respecto a *Manhattan*. De igual manera conservamos la propiedad con *Minkowsky* de ser un equilibrio entre la distancia *Euclidiana* y *Manhattan*. Podemos apreciar que *Chebishev* tiende a tomar valores mas pequeños que las otras debido a que para esta distancia solo se considera la mayor de las diferencias entre las variables de los vectores de datos.

Un detalle a resaltar es que ahora todas las matrices se componen por números flotantes a diferencia de las anteriores donde también había matrices constituidas solamente por números enteros, esto se debe a que la estandariacion por lo general regresa números flotantes y como todas las distancias toman como base esta matriz es normal que sus resultados también estén expresados con números flotantes por lo cual es aconsejable redondear las matrices a los decimales que se consideren pertinentes, para este caso se consideraron 3 números decimales.

Conclusiones.

En conclusión, las métricas de distancia nos ayudaran a obtener índices que miden el nivel de similitud entre los objetos los cuales pueden ser, por ejemplo, usuarios, rutas, productos, etc. las cuales luego serán la entrada para nuestros algoritmos de inteligencia artificial. Debemos de tener en cuenta que para poder obtener estas distancias debemos de filtrar las características para que sean numéricas. Además de que cada métrica de distancia nos servirá para propósitos o soluciones diferentes, por ejemplo, Chebyshev nos puede ser útil para procesos industriales, mientras que Manhattan para medir distancias un poco más apegadas a la realidad donde no es tan factible irse por la distancia mas corta la cual sería una línea diagonal como la hipotenusa de las distancias Euclidianas, sin embargo, en la actualidad se buscan distancias más factibles las cuales describen una distancia más generalizada o flexible como Minkowsky que define un parámetro lambda para obtener este efecto la cual por ejemplo con valor de 1.5 es un punto medio entre Euclidiana y Manhattan.

Es importante estandarizar o normalizar nuestros datos, pues con este proceso conseguiremos bajar la escala de los mismos haciendo que computacionalmente sea menos costoso optimizando nuestros algoritmos, además de que visualmente son más comparables al obtener nuestros resultados.

Fuentes.

Feature Scaling | Standardization Vs Normalization. (2021, 26 agosto). Analytics Vidhya. Recuperado 2022, de https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Standardization Vs Normalization- Feature Scaling. (2019, 7 noviembre). [Vídeo]. YouTube. https://www.youtube.com/watch?v=mnKm3YP56PY&t=559s&ab_channel=KrishNaik