



29 DE DICIEMBRE DE 2019


SISTEMAS INTELIGENTES I

MAPA AUTOORGANIZADO

Universidad de Las Palmas de Gran Canaria

Benearo Semidan Páez Martín
Juan Sebastián Ramírez Artilles
Alberto Ramos Sánchez
Iru Nervev Navarro Alejo

benearo.paez101@alu.ulpgc.es
juan.ramirez107@alu.ulpgc.es
alberto.ramos104@alu.ulpgc.es
iru.navarro101@alu.ulpgc.es



Contenido

Introducción	2
Algoritmo del mapa autoorganizado.	2
Limpieza de datos.	5
Cuestiones planteadas y solución.....	5
Cuestión 1: ¿Existe alguna relación entre el coste total y la distancia y tipo de tarifa?	5
Cuestión 2: ¿Existe alguna relación entre la propina y el número de pasajeros, la distancia del recorrido y el método de pago?	6
Cuestión 3: ¿Influye el lugar de llegada y la distancia recorrida con la tarifa?	7
Cuestión 4: ¿Influye el lugar de salida y la distancia recorrida con la tarifa elegida?.....	8
Referencias	10
Tabla de figuras.....	10
Tabla de ecuaciones.....	10

Introducción

En esta práctica hemos implementado un mapa autoorganizado con el que hemos resuelto las cuestiones que planteamos sobre el dataset *Demanda de Movilidad Individual en la ciudad de New York (01/01/2016)*.

Algoritmo del mapa autoorganizado.

El *SOM* utilizado está formado por una malla —de forma toroidal— de neuronas conectadas de forma octogonal con sus vecinas, como vemos en la Ilustración 1. Cada neurona tiene un número de entradas dependiente del tamaño del vector de entrada a la red, a las cuales se les asigna un peso aleatorio según una distribución normal con media 0 y varianza 1.

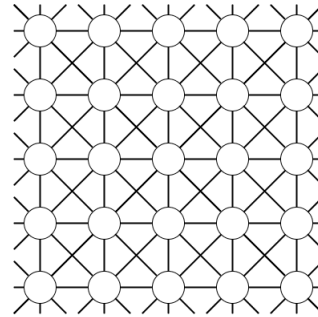
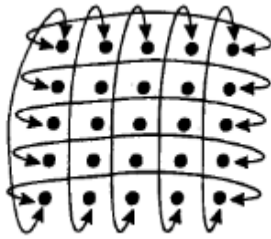


Ilustración 1: Mapa autoorganizado

La implementación del *SOM* se encuentra en el fichero *som.py* en la clase *Som*. Para representar el comportamiento del mapa SOM, tenemos una matriz de pesos —llamada *W*— de dimensión el número de neuronas ($n_neurons_x * n_neurons_y$) por el número de entradas de cada neurona (*number_inputs_each_neuron*).

Para realizar el algoritmo de entrenamiento —en el método *fit*—, nos hemos basado en el algoritmo descrito por Fernando Sancho Caparrini en su publicación *Mapas Autoorganizados* (Caparrini, 2019), modificándolo para ejecutarlo con *TensorFlow* de forma matricial.

El funcionamiento de este algoritmo consiste, en primer lugar, en encontrar la neurona cuyos pesos son más cercanos al vector de entrada —mediante el cálculo de la distancia euleriana— y, posteriormente, actualizar los pesos de dicha neurona y los vecinos que se encuentran en un radio determinado —definido inicialmente y que va disminuyendo progresivamente— para que estos se aproximen.

La localización de la neurona más cercana (*BMU* — *best matching unit*) se realiza de la siguiente forma. A partir de una matriz *X* —generada replicando el vector de entrada *x_input* por el número de neuronas de la red—, hallamos la neurona *BMU* mediante el cálculo de la distancia euleriana:

$$distancias_{number_neurons} = \sum_{j=0}^{number_inputs_each_neuron-1} (X - W)^2 \text{ (potencia elemento a elemento)}$$

Ecuación 1: Distancia euleriana

Una vez tenemos todas las distancias, localizamos la unidad con la distancia menor —con la función [argmin](#)—, obteniendo un índice *index*, del cual calculamos la posición en la matriz del mapa SOM del siguiente modo:

Fila $i = index \bmod x$ donde x es el número de neuronas en el eje x del mapa.

Columna $j = \left\lfloor \frac{index}{x} \right\rfloor$ donde x es el número de neuronas en el eje x del mapa.

Ecuación 2: Posición de la BMU

Al actualizar los pesos de esta neurona y sus vecinas, realmente lo que hacemos es actualizar los pesos de todas las neuronas, dando más significancia a las neuronas más cercanas. La ecuación de actualización de pesos para una iteración t es:

$$W(t + 1) = W(t) - dw$$

Ecuación 3: Actualización de pesos

donde W es la matriz de pesos de cada neurona de la red, y dw una matriz que representa la actualización de pesos en la iteración t .

*$dw = lr(t) * v(i, j) * (X - W)$ donde $v(i, j)$ es la función vecino.*

Ecuación 4: Derivada. Actualización de pesos.

El resultado de la función vecino es un vector con un factor para cada neurona, con el que seleccionamos en cuales vamos a modificar sus pesos. Estas neuronas se eligen siguiendo una función gaussiana.

$$v(i, j) = e^{-D^2 / 2 * r(t)^2} \text{ donde } r(t) = r_0 e^{-t/\lambda}$$

en la que $\lambda = \frac{\text{iteraciones}}{\ln(r_0)}$ es constante, D un vector con las distancias entre BMU y cada neurona.

Ecuación 5: Función vecino

Como nuestro mapa SOM sigue una topología toroidal y lo representamos como si fuera una matriz, debemos modificar el cálculo de distancias, como se explica en el *paper* Selecting the toroidal self-organizing feature maps best organized to object recognition (Andreu, Crespo, & Valiente González, 1997).

La distancia entre una neurona en la posición (i, j) de la neurona BMU en (i_{BMU}, j_{BMU}) tiene la siguiente expresión:

$$D = DT_{row}^2 + DT_{col}^2$$

$$DT_{row} = \min\{(i - i_{BMU}), f - \text{abs}(i - i_{BMU})\}$$

$$DT_{col} = \min\{(j - j_{BMU}), c - \text{abs}(j - j_{BMU})\}$$

Ecuación 6: Distancia toroidal

donde f y c son el número de filas y columnas, respectivamente.

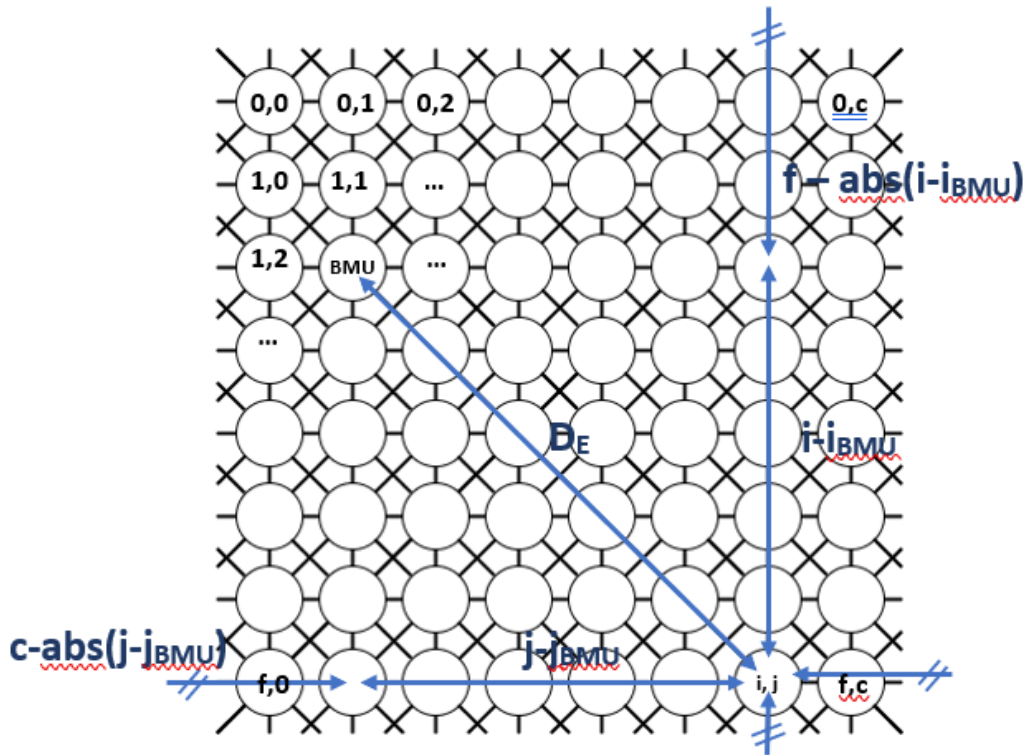


Ilustración 2: Distancia toroidal

Este cálculo lo ampliamos a forma matricial —teniendo como resultado el vector D de distancias entre BMU y cada neurona del mapa— con el que calculamos el resultado de la función vecino $v(i, j)$.

El *learning rate* también es adaptativo, disminuyendo de forma exponencial en cada iteración:

$$lr(t) = lr_0 * e^{-t/\lambda}$$

Ecuación 7: Learning rate adaptativo

Como hemos dicho, la implementación del *SOM* en *Tensorflow* se encuentra en la clase *Som* en el fichero *som.py* —para la cual nos basamos en el capítulo 5 *Automatically clustering data* del libro *Maching Learning with Tensorflow Meap Edition* (Shukla)—, y tiene la siguiente estructura:

- *init*: constructor donde inicializamos el grafo computacional *graph*, el radio inicial *initial_radius*, el *learning rate* inicial *lr₀* y el tensor de pesos *w* de tamaño número de neuronas por número de entradas por neurona.
- *neuron_indexes*: que devuelve un array con las posiciones en el mapa de cada neurona de la red.
- *get_bmu*: que calcula la posición de la neurona *BMU* más cercana al vector de entrada *x*.
- *_get_distance_torus*: que devuelve un tensor con la distancia de cada neurona con la *BMU*.
- *neighbours_function*: que devuelve un tensor con la función vecina de cada neurona.
- *fit*: que construye el grafo computacional para una iteración del algoritmo.

- *train*: que recibe el dataset de entrada y crea una sesión en la que se ejecuta el grafo computacional tantas veces como iteraciones se indiquen y por tantas entradas que tenga el dataset.
- *classify*: que clasifica un vector de entrada, devolviendo la posición de la neurona más cercana *BMU*.

Limpieza de datos.

En cuanto a la limpieza del dataset, en este apartado comentaremos el tratamiento realizado común a todas las preguntas. Debido a que cada una necesita de distintos formatos de datos, las acciones realizadas fueron:

- Localizar y nombrar las distintas columnas
- Localizar y eliminar columnas vacías (*pickup_location_id*, *dropoff_location_id*)
- Localizar y eliminar las filas cuyo campo *total_amount* y *tip_amount* sea negativo
- Localizar y eliminar las filas cuyo campo *passenger_count* sea 0 o menor
- Localizar y eliminar filas cuyas coordenadas de localización o *ratecode* no sean válidos o correctos (por ejemplo, coordenadas [0, 0])
- Cambiar la notación de 'N' e 'Y' a 0, 1 en el campo *store_and_fwd*
- Cambiar la notación de 'datetime' de las fechas de recogida y llegada a segundos desde el 1 de enero de 1970
- Creación de la columna *length_time*, que indica la duración del trayecto de cada viaje

Con esto tenemos un conjunto de datos listo para las preguntas individuales.

Cuestiones planteadas y solución.

Cuestión 1: ¿Existe alguna relación entre el coste total y la distancia y tipo de tarifa?

Para obtener una respuesta a esta pregunta hemos representado en el mapa la distancia del recorrido (*trip_distance*) y el tipo de tarifa (*ratecode_id*), y lo hemos coloreado con los valores del campo (*total_amount*).

El campo del tipo de tarifa es categórico, así que los hemos transformado en una representación one-hot antes de pasarlo como valor de entrada al SOM.

Hemos hecho un estudio previo de los datos con el objetivo de establecer unos rangos de coste final que resulten coherentes con los valores del dataset. Estos rangos serán utilizados para categorizar los valores continuos de coste y de este modo poder asignar un color diferente a cada rango.

Además, hemos tomado muestras de 1000 registros de cada rango, de modo que cada rango tenga una representación equitativa de valores en el cálculo del mapa. En total, la muestra que utilizamos para esta pregunta contiene 6000 registros distribuidos en 6 rangos.

En las pruebas que hemos hecho, hemos alcanzado la convergencia del mapa con un mínimo de 10 iteraciones y con una ratio de aprendizaje inicial de 0.1.

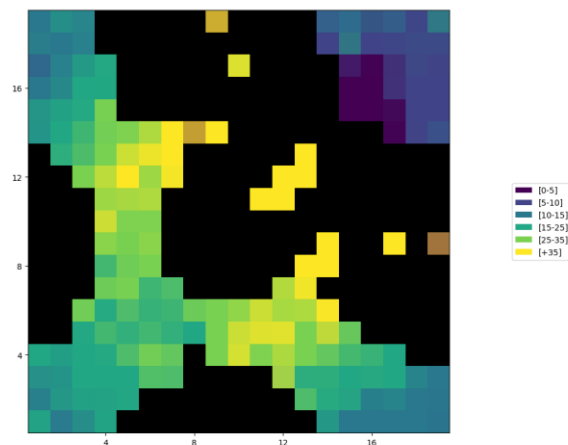


Ilustración 3: 10 iteraciones

Como se puede observar en la ilustración 3, podemos concluir que en efecto existe una relación directa entre la distancia del recorrido, el tipo de tarifa y el coste total del viaje. Sobre todo, destaca el agrupamiento en el mapa del rango de menor coste.

Cuestión 2: ¿Existe alguna relación entre la propina y el número de pasajeros, la distancia del recorrido y el método de pago?

Para esta pregunta hemos utilizado como campos de entrada: la distancia del trayecto (`trip_distance`), el número de pasajeros (`passenger_count`), y el tipo de pago (`payment_type`). Como campo de coloreado del mapa utilizamos el valor de la propina (`tip_amount`).

Una vez más hemos transformado el campo categórico tipo de pago a su representación one-hot de modo que pueda ser utilizado eficientemente por el algoritmo del cálculo del SOM.

Tal como hicimos para la pregunta 1 hemos dividido los valores de la propina en rangos representativos de los valores reales del dataset con el objetivo de categorizar los valores continuos y utilizarlos como colores del mapa.

Como en el caso anterior hemos tomado 1000 registros aleatorios del dataset para cada rango de propina. En total la pregunta maneja una muestra de 5000 registros divididos en 5 rangos.

Al igual que en la pregunta anterior se consigue una convergencia aceptable con un mínimo de 10 iteraciones y una ratio de aprendizaje inicial del 0.1.

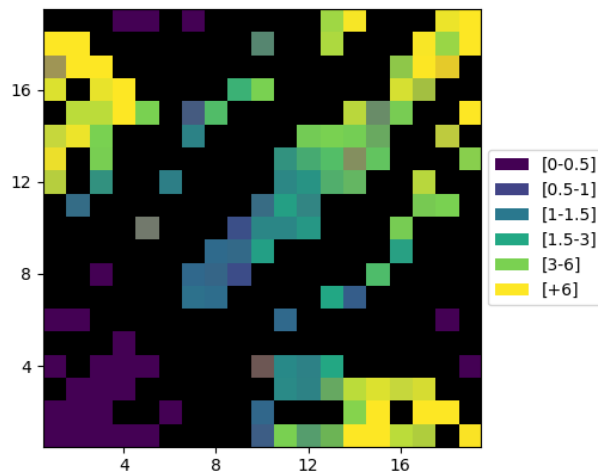


Ilustración 4: 10 iteraciones

Como se puede observar en la ilustración 4, podemos encontrar un alto grado de agrupamiento para propinas superiores a 6\$. Además, se puede apreciar un desplazamiento gradual desde el color amarillo de los 6\$ al verde de 3-6\$ y a los distintos tonos de azul.

Vemos que las propinas en el rango [0-0.5] es el que está más cohesionado y separado de los demás, por lo que podemos deducir que existe alguna causa relacionada al número de pasajeros, el método de pago o distancia recorrida que provoca que los pasajeros den poca propina.

Cuestión 3: ¿Influye el lugar de llegada y la distancia recorrida con la tarifa?

Para esta pregunta hemos utilizado como campos de entrada la posición de bajada del taxi (dropoff_longitude, 'dropoff_latitude'), y la distancia del trayecto (trip_distance). Como campo utilizado para colorear el mapa hemos escogido el tipo de tarifa (ratecode_id). El objetivo de esta pregunta es descubrir si la tarifa aplicada depende del lugar de llegada y la distancia recorrida.

Como en las preguntas anteriores hemos tomado una muestra de alrededor de 6000 registros. Así mismo, hemos intentado tomar una cantidad equitativa de registros de cada tipo de tarifa.

Esta vez hemos utilizado un mapa más grande que el anterior. Anteriormente era de 20x20 y el de esta y la siguiente pregunta es de 30x30. Es debido a esto que la convergencia mínima se consigue con al menos 20 iteraciones. La ratio inicial de aprendizaje la hemos dejado como en los anteriores casos en 0.1.

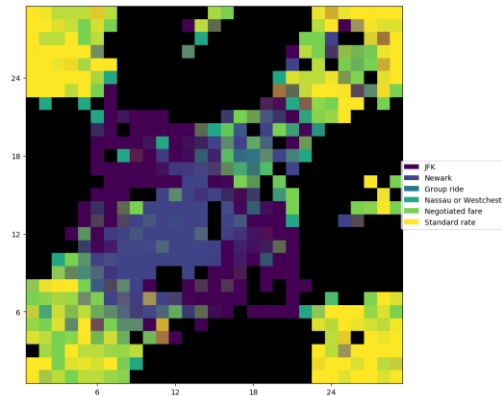


Ilustración 5 - 20 Iteraciones

Como se puede observar existe un claro orden en el mapa, sobre todo en lo concerniente al tipo de tarifa *Standard Rate*. Por lo que podemos concluir que, en efecto, existe algún tipo de relación entre la tarifa aplicada y el lugar de bajada y la distancia.

Cuestión 4: ¿Influye el lugar de salida y la distancia recorrida con la tarifa elegida?

Esta pregunta es complementaria a la anterior, solo que ahora tratamos de descubrir si la tarifa aplicada queda determinada por el lugar salida y por la distancia.

Los campos de entrada que utilizamos son los mismos que para el caso anterior con la diferencia de que la longitud y latitud que tomamos representa el lugar de partida de la ruta. Los campos de entrada son: el punto de subida al taxi (pickup_longitude, pickup_latitude), y la distancia del trayecto (trip_distance). El tratamiento que se realiza a los datos es similar al de la pregunta anterior.

Al igual que en el caso anterior, el número mínimo de iteraciones para alcanzar la convergencia es de 20 iteraciones. La ratio inicial de aprendizaje es como en los anteriores casos de 0.1.

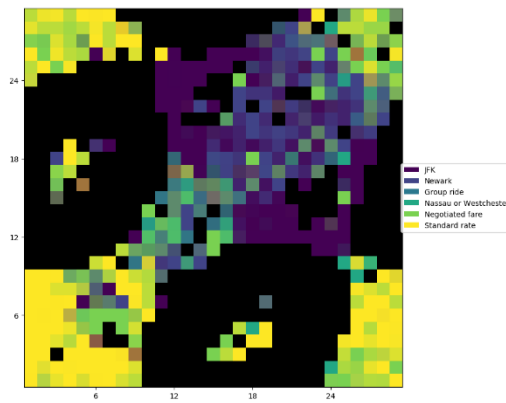


Ilustración 6 - 20 Iteraciones

Como en el caso anterior se puede observar que la tarifa estándar está bastante relacionada con el punto de salida de la ruta. No podemos concluir si la tarifa se aplica por el punto de subida, el punto de bajada, o por el de subida y el de bajada. Lo que sí podemos concluir es que el trayecto tiene que ver con el tipo de tarifa que se aplica.

Referencias

- Andreu, G., Crespo, A., & Valiente González, J. M. (Julio de 1997). *Selecting the toroidal self-organizing feature maps (TSOFM) best organized to object recognition*. Obtenido de Departamento de Ingeniería de Sistemas, Computadores y Automática. Universidad Politécnica de Valencia, SPAIN.:
https://www.researchgate.net/publication/3705520_Selecting_the_toroidal_self-organizing_feature_maps_TSOFM_best_organized_to_object_recognition
- Caparrini, F. S. (2019). *Mapas Auto-Organizados*. Obtenido de Fernando Sancho Caparrini:
<http://www.cs.us.es/~fsancho/?e=76>
- Shukla, N. (s.f.). Automatically clustering data. En N. Shukla, *Maching Learning with Tensorflow Meap Edition* (págs. 109-115). Manning.

Tabla de figuras

Ilustración 1: Mapa autoorganizado	2
Ilustración 2: Distancia toroidal	4
Ilustración 3: 10 iteraciones	6
Ilustración 4: 10 iteraciones	7
Ilustración 5 - 20 Iteraciones	8
Ilustración 6 - 20 Iteraciones	9

Tabla de ecuaciones

Ecuación 1: Distancia euleriana	2
Ecuación 2: Posición de la BMU	3
Ecuación 3: Actualización de pesos	3
Ecuación 4: Derivada. Actualización de pesos	3
Ecuación 5: Función vecino	3
Ecuación 6: Distancia toroidal	3
Ecuación 7: Learning rate adaptativo	4