Informe SDYST 1: Self Organizing Maps (SOM)/ Hierarchical Clustering (HC) en MNIST

Cristóbal Lecaros, Felipe Miranda

6/26/2019

Contents

Introducción

Self Organizing Maps (SOM) es un algoritmo de clasificación no supervisado inventado por Teuvo Kohonen en la década de los ochenta que está basado en la idea de que el cerebro procesa estímulos sensoriales que tienen un espacio de entrada multidimensional que se reduce a ciertas regiones topógraficas de la corteza neuronal (Kohonen 1990). Sus aplicaciones son variadas y entre ellas están el reconocimiento de patrones estadísticos en discursos, el control de brazos robóticos, compresión de imágenes, clasificación de cantos de apareamiento de insectos, etc. Es un método interesante porque permite visualizar en dos dimensiones las relaciones que tienen datos compuestos por variables de muchas dimensiones y que no son obvias a la inspección humana.

El objetivo de este trabajo es replicar la metodología utilizada en (Palamara, Piglione, and Piccinini 2011), utilizando la base MNIST (LeCun, Cortes, and Burges 2019) para comparar el resultado de clusterización de dos métodologías:
- SOM + K-means
- Clustering Jerárquico

Utilizando las etiquetas de cada imagen, se analizará el performance de clusterización con una métrica de accuracy del tipo "Caso Correcto / Casos totales".

Métodos

1. Base de datos. MNIST

La base de datos MNIST está compuesta por digitos escritos a mano. Tiene un conjunto de entrenamiento de 60,000 ejemplos, y un conjunto de prueba de 10,000 ejemplos. Es un subconjunto de un set mayor proveniente de NIST. Los digitos fueron normalizados por tamaño y centrados en una tamaño fijo de imagen. Los 60,000 patrones del conjunto de entramiento contienen ejemplos de aproximadamente 250 escritores.

Descarga y lectura de datos

El análisis fue realizado en `R version 3.5.2`. Para cargar los datos de MNIST, utilizamos el código de Dalpiaz (2019) disponible aquí.

2. Algoritmo SOM

2.1. Conceptos y definiciones

Self-Organizing Maps es una red neuronal competitiva que proyecta vectores que provienen de un espacio de altas dimensiones a un mapa de dos dimensiones. El algoritmo de proyección está diseñado para preservar las relaciones de distancia entre los datos de entrada de la manera más fidedigna posible. En nuestro caso, SOM proyecta las *muestras de 28x28-dimensiones del data set* hacia las unidades, también llamadas *vectores prototipos*, de una red cuadrada.

La red de SOM está compuesta por $M = m_1 \times m_2$ neuronas, o *unidades*. Cada unidad está asociada con dos tipos de información: (1) la posición de la unidad en la red; (2) un vector N-dimensional $w_i (i = 1, ..., M)$, llamado *vector prototipo*, donde N es la dimensión del espacio de los vectores de entrada. Por lo tanto, en este caso el término unidad se refiere tanto a los elementos de la red como a los vectores prototipos asociados. El conjunto de vectores prototipo también es llamado *codebook* del SOM.

Mapear datos para entrenar un SOM significa calcular la distancia de los nuevos puntos de datos o muestras a los vectores prototipos o codebook, y asignar cada objeto a la unidad con el vector prototipo más similar (la *unidad ganadora* o "best matching unit").

Para realizar el análisis nosotros utilizamos el paquete `kohonen` que se encuentra ampliamente documentado (Wehrens and Kruisselbrink 2018).

2.2 Funcion vecindario

Lo diferente que tiene SOM con otros algoritmos adaptivos es que utiliza una *función vecindario* gaussiana que se define como:

$$h_{c(i)}(t) = e^{\frac{-\left|r_{c(i)} - r_i\right|^2}{2\sigma(t)^2}}$$

donde $r_c(i)$ y r_i son las posiciones en el mapa de la unidad ganadora y la unidad genérica i, respectivamente; $\sigma(t)$ es el radio a la iteración t y corresponde a la amplitud de la función vecindario en el tiempo t.

2.3 Algoritmo de actualización: Batch

El algoritmo de entrenamiento usado fue el usado de manera standard y se conoce como *batch*. Consiste en comparar los objetos de entrenamiento con el conjunto de vectores prototipo. El vector prototipo de la unidad ganadora, al igual que las unidades del vecindario se modifican y se vuelven más similares al objeto que mapearon. Durante el entrenamiento, el radio decrece lentamente; al final del proceso solo las unidades ganadoras son actualizadas. La regla de aprendizaje y actualización se define como:

$$w_i(t+1) = \frac{\sum_{i=1}^{S} h_{c(i)}(t) \cdot x_j}{\sum_{i=1}^{S} h_{c(i)}(t)}$$

donde $w_i(t+1)$ es el peso del vector actualizado; x_j es la muestra de entrada; $h_{c(i)}(t)$ es la función vecindario descrita anteriormente; S es el número de muestras de entrada.

2.4 Distancia entre vectores prototipo

En el trabajo de (Palamara, Piglione, and Piccinini 2011), utilizaron datos catégoricos que definían diferentes causas o mecanismos de accidentes laborales. En este caso, todas las variables de entrada vienen del mismo espacio vectorial que tiene valores númericos que van desde 0 a 255, por lo que no fue necesario usar la distancia de Hamming y este espacio fue tratado de manera continua. Usamos la distancia *suma de cuadrados*. La distancia *suma de cuadrados* utilizada en un mapa con una única capa es equivalente a una distancia Euclidiana, pero más rápida de computar porque no requiere calcular la raíz cuadrada. Esta distancia se define matemáticamente como:

$$d^2(\mathbf{p}, \mathbf{q}) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2$$

2.5 Tamaño del mapa

No existen métodos teóricos para determinar el tamaño de un mapa. Una regla general (Palamara, Piglione, and Piccinini 2011) sugiere usar la fórmula $5 * \sqrt{S}$, donde S es el número de muestras para entrenar. Nosotros probamos con esta regla, pero los mejores resultados dieron ocupando la máxima cantidad de unidades posibles permitidas por la implementación del algoritmo.

3. Algoritmo K-means

Una vez realizado el entrenamiento de SOM, el mapa fue divido en un número de areas finito. Este segundo nivel de clustering fue realizado utilizado el algoritmo *k-means*, visto en clases.

3.1 Índice Davies & Bouldin

Para saber cuántos cluster debian hacerse en la base de datos de accidentes laborales, (Palamara, Piglione, and Piccinini 2011) cuantificaron el rendimiento de los cluster usando el *Índice Davies & Bouldin*. Este índice está definido matemáticamente como:

$$DB = \frac{1}{n}\sum_{i=1}^{n} \max_{i \neq j} \left\{ \frac{S_n\left(Q_i\right) + S_n\left(Q_j\right)}{S_n\left(Q_i, Q_j\right)} \right\}$$

donde n es el número de clusters, Q_i es el cluster i; $S(Q_i)$ es la distancia promedio de todos los elementos del cluser con respecto a su centro; $S(Q_i, Q_j)$ es la distancia entre los centros de los cluster. En nuestro caso calculamos el índice para los primeros 10 cluster, ya que sabemos a priori que los datos pertenecen a 10 clases.

Resultados

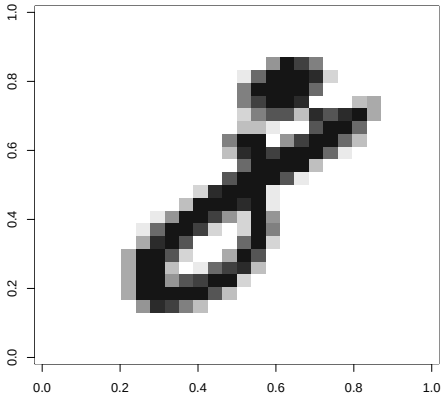
Para realizar nuestros análisis, utilizamos una submuestra del dataset de muestras de entrenamiento. Utilizamos 1,000 muestras provenientes del data set de entrenamiento.

```
set.seed(1)
n_muestra <- 1000
muestra <- train[sample(1:nrow(train), n_muestra,
                        replace=FALSE),]

X <- as.matrix(muestra[,-785])
Y <- muestra[,785]
```

Una vez cargados los datos y realizada la submuestra, revisamos que los datos estuvieran correctamente etiquetados y que las imagenes correspondieran a su clase. Un ejemplo de esto puede verse en la imagen a continuación, donde se observa que la imagen se corresponde con su valor en la columna de etiqueta.

```
show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784[-785]), nrow = 28)[, 28:1], col = col, ...)
}

show_digit(muestra[50, ])
muestra[50, 785]
```

```
## [1] 8
## Levels: 0 1 2 3 4 5 6 7 8 9
```



SOM

Función `som`

La función SOM utilizando el paquete `kohonen` tiene la siguiente forma:

```
som_grid <- somgrid(xdim = val_grid, ydim= val_grid, topo="hexagonal",
                    neighbourhood.fct = "gaussian")

som_model_batch <- som(X,
                       grid = som_grid,
                       mode ="pbatch")
```

donde, como se observa, primero es necesario definir los parámetros de la red: `xdim` e `ydim` corresponden a las dimensiones, `topo`es la topología y `neighbourhood.fct` a la función vecindario. El modelo toma los datos `X`, y realiza el entrenamiento con el algoritmo batch, definido como `pbatch`, que significa parallel batch. Puede leerse un resumen de las características del modelo a continuación

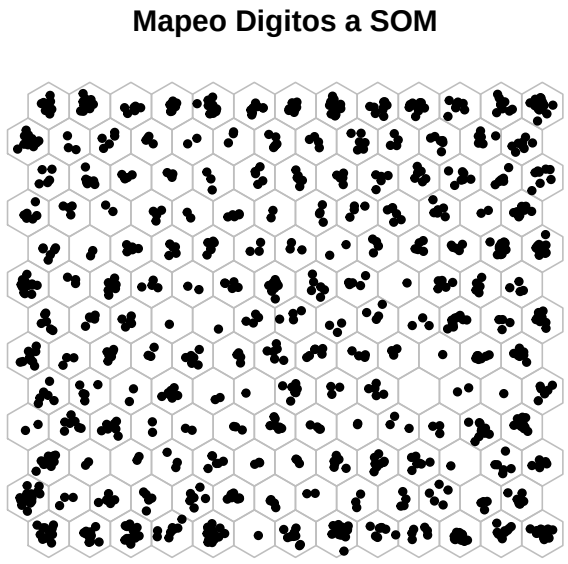
```
summary(modelo1)
```

```
## SOM of size 13x13 with a hexagonal topology and a gaussian neighbourhood function.
## The number of data layers is 1.
## Distance measure(s) used: sumofsquares.
## Training data included: 1000 objects.
## Mean distance to the closest unit in the map: 1409074.
```

Ajuste del modelo

Podemos observar que la distancia de los vectores prototipo con los valores de la muestra caen y se ajustan a medida que se entrena el agoritmo.

Una vez ajustado el modelo, podemos revisar cual es la disposición general de cada muestra en la red, mediante el siguiente gráfico

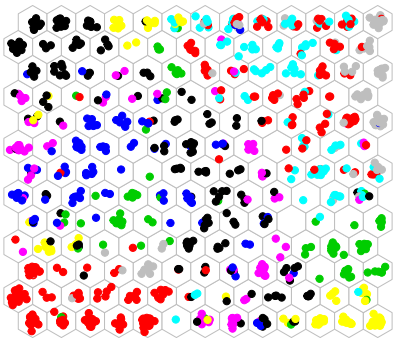


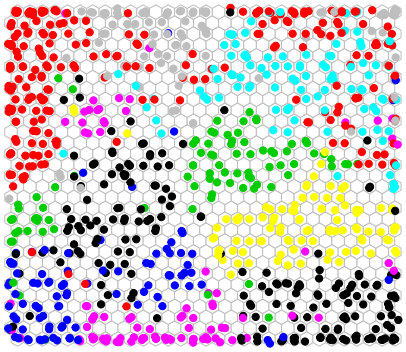
Número óptimo de unidades

Como se comentó en la sección métodos, no existe una forma teórica de determinar la mejor cantidad de unidades de la red, por lo que se ocupa una fórmula general como se expresa a continuación.

```
val_grid <- sqrt(n_muestra)*5
val_grid <- round(sqrt(val_grid))
```

Sin embargo, al analizar la red aumentando el número de unidades, se observa que la capacidad de discriminar entre los digitos aumenta, como puede verse comparando las siguientes imágenes.



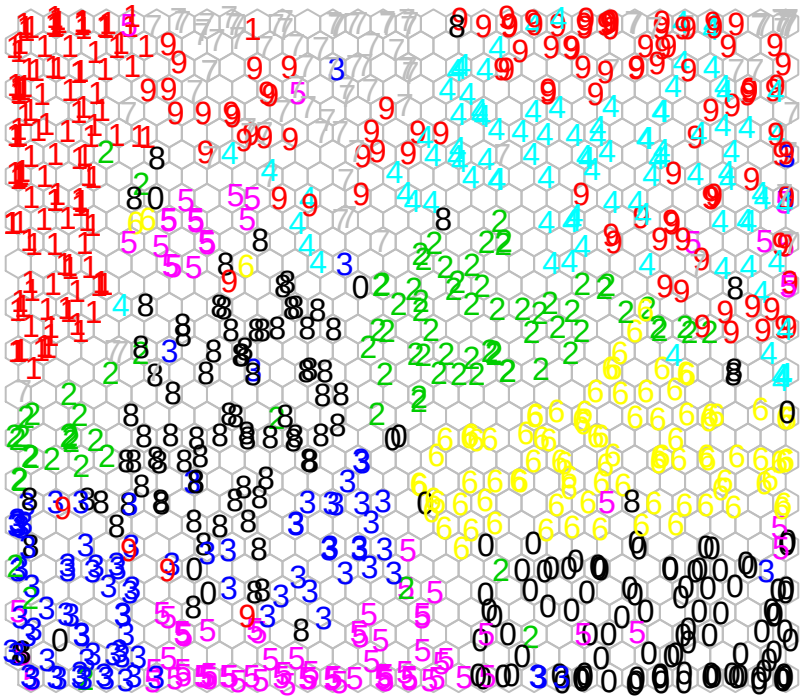


Es por esta razón que trabajamos finalmente con una red de 31x31 unidades.

Análisis exploratorio y evaluación de distancia

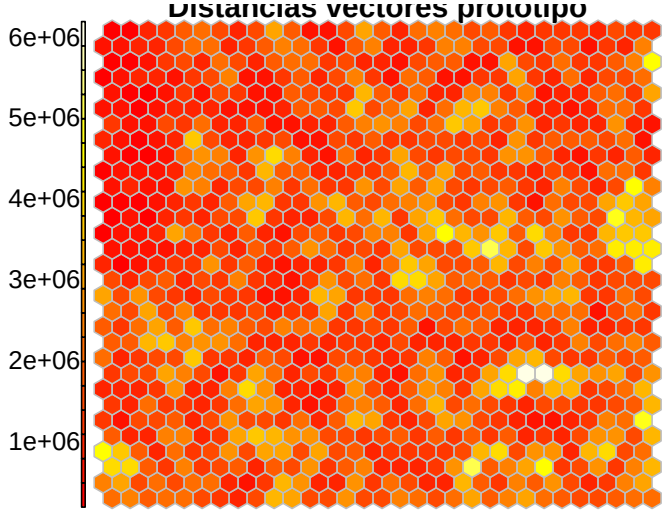
Al hacer un análisis exploratorio de la red, podemos observar que hay algunos digitos que son bien identificados, mientras otros por su semejanza entre ellos caen en regiones del mapa similares. Es llamativo también constatar que existe una vecindad morfológica más general aún, y como es esperable, los números que se parecen entre sí están más cerca (a pesar de estar en regiones separables) que los que tienen forma diferente. Por ejemplo, el digito 1 está cerca del digito 7 en la región superior; el digito 4 (celeste) está cerca de la región donde se encuentra el digito 9; y la región de digito 8 (negro) colinda con la región de digito 6 (amarillo) y esta a su vez con la región de digito 0 (negro).

```
plot(modelo2, type = "mapping",
     labels = Y, col = as.integer(Y),
     border = "gray", shape = "straight" , main = "")
```



En el trabajo de (Palamara, Piglione, and Piccinini 2011), ellos utilizaron el mapa de distancias para distinguir regiones independientes. En nuestro caso, el mapa de distancia entre los vectores prototipos no permite distinguir con claridad clusters.

```
plot(modelo2, type="dist.neighbours", main = "Distancias vectores prototipo",
     shape="straight",
     border ="gray")
```



Clusters

Índice DB

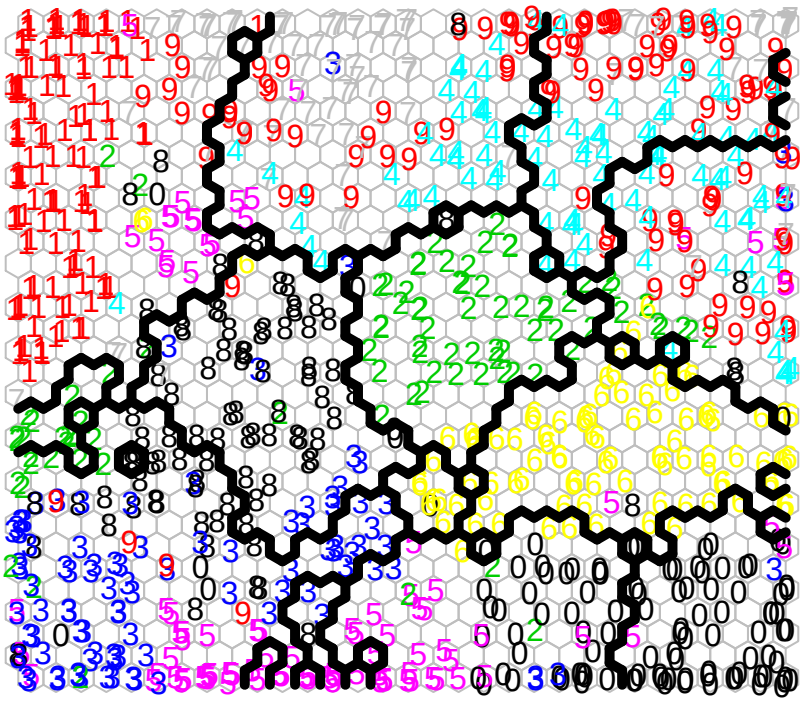
El paso siguiente es realizar una segunda clusterización sobre los vectores prototipo del SOM. Para determinar la cantidad de cluster óptima, se utilizó el índice DB, como se mencionó en los métodos. La siguiente tabla muestra los valores para los cluster, siendo el mejor valor 9. Dado que conocemos que son 10 digitos, trabajamos con 10 cluster de todas formas.

Table 1: Índice DB según K-cluster

Clusters	Indice DB
2	3.450507
3	3.190180
4	2.802869
5	2.659203
6	2.497815
7	2.425721
8	2.483274
9	2.343781
10	2.488783

SOM con K-Means

El siguiente gráfico presenta la red SOM con los cluster realizados por K-Means sobre los vectores prototipo.



Como puede observarse, en general la clusterización es de mala calidad, con excepción para los dígitos 8 y 6. Para cuantificar esto, se realizó el calculo de *Accuracy* según la métrica propuesta por Palamara et al., como fue descrito en la introducción del trabajo. Consideramos clasificación correcta del grupo a la de la clase mayoritaria del cluster, y el denominador fueron los casos totales. La matriz de confusión para el algoritmo se presenta en la siguiente tabla.

	0	1	2	3	4	5	6	7	8	9
Cluster 1	2	79	5	1	1	13	12	16	6	10
Cluster 2	0	0	2	13	0	5	1	1	59	1
Cluster 3	61	0	0	1	0	2	2	0	0	0
Cluster 4	0	0	7	2	58	6	1	25	3	61
Cluster 5	36	0	3	6	0	32	4	0	0	0
Cluster 6	0	46	14	0	1	0	0	3	2	0
Cluster 7	6	0	6	73	0	30	0	0	14	5
Cluster 8	1	0	51	0	1	0	5	0	2	0
Cluster 9	1	0	1	0	0	1	69	0	2	0
Cluster 10	0	0	0	1	40	4	0	45	1	38

Accuracy SOM + K-Means

Los valores de accuracy para cada cluster son presentados a continuación. El promedio de accuracy, ponderado por el numero de elementos en cada cluster, es 58.0%.

Cluster	Acc
Cluster 1	0.5448276
Cluster 2	0.7195122
Cluster 3	0.9242424
Cluster 4	0.3742331
Cluster 5	0.4444444
Cluster 6	0.6969697
Cluster 7	0.5447761
Cluster 8	0.8500000
Cluster 9	0.9324324
Cluster 10	0.3488372

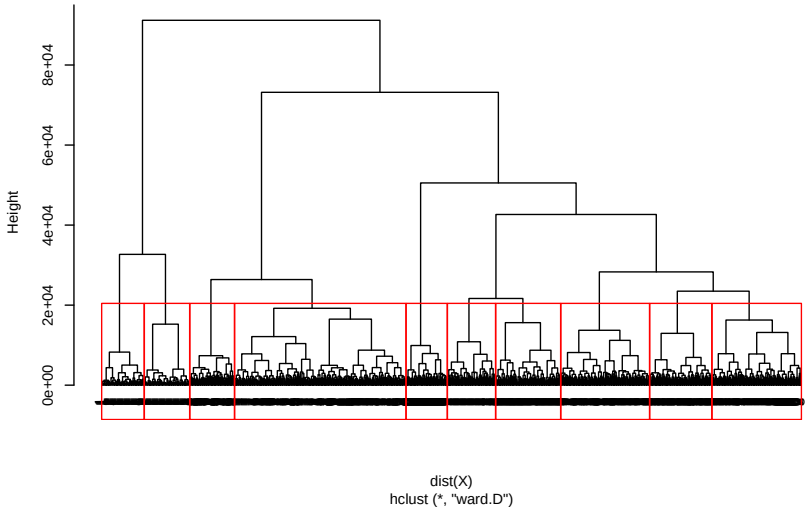
Clustering Jerárquico

Construcción de Dendograma

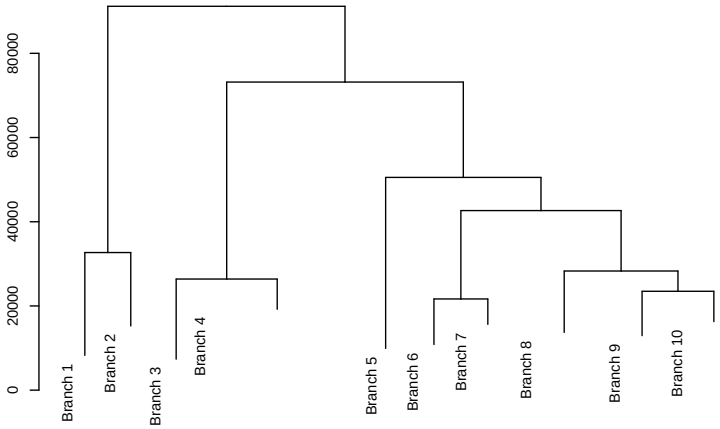
En el paper se hace una comparación entre la clusterización con SOM+Kmeans y la clusterización con Clustering Jerárquico (HC). Para el data set utilizado en este trabajo, se aplicó HC utilizando la librería `caret`. Se utilizó el criterio de linkage de Ward, tal como se hizo en el paper. El árbol fue separado en 10 clusters.

```
h_clusters <- hclust(dist(X), method = "ward.D")

par(mfrow=c(3,1))

plot(h_clusters, hang=-1, labels = Y, main = "Dendograma cortado en 10 branches")
groups <- cutree(h_clusters, 10)
rect.hclust(h_clusters,k=10)

plot(cut(as.dendrogram(h_clusters), h=20000)$upper, main = "Arbol cortado en 10 branches")

plot(cut(as.dendrogram(h_clusters), h=20000)$lower[[10]], main = "Ultimo branch")
```

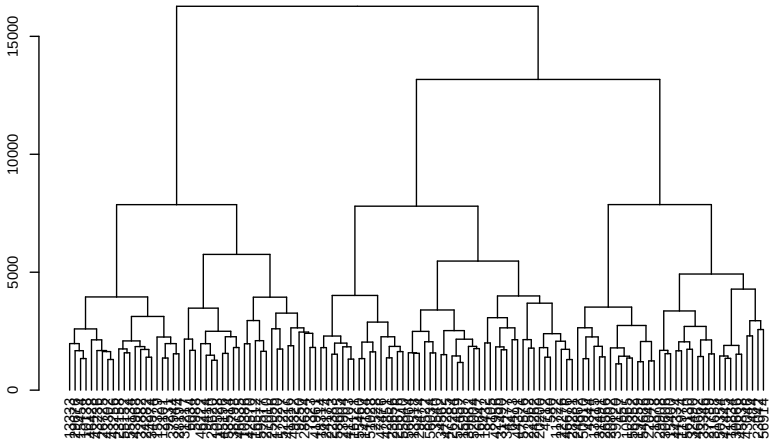
Dendograma cortado en 10 branches



Arbol cortado en 10 branches



Ultimo branch



El último dendograma corresponde al branch 10 del árbol original. Aunque las etiquetas de cada elemento no permiten visualizar los números en ese cluster (ya que corresponden al ID de cada elemento), la siguiente tabla enumera cada ocurrencia. A partir de los resultados, se deduce que el branch 10 corresponde al cluster del número 2.

Var1	Freq
0	0
1	0
2	66
3	0
4	0
5	0
6	0
7	0
8	2
9	1

A continuación se observan los resultados de clusterización para cada branch. El mejor resultado se observa en los clusters 3, 6, 7, 8 y 10. Curiosamente, los clusters 3 y 8 contienen las imágenes de 1s. Se observa en la tabla que los numeros mejor clusterizados fueron 0, 1, 2 y 6.

	0	1	2	3	4	5	6	7	8	9
Cluster 1	8	0	9	51	0	35	5	4	10	5
Cluster 2	0	1	5	1	64	4	0	84	5	81
Cluster 3	0	65	0	0	0	0	0	0	0	0
Cluster 4	35	0	4	35	0	50	1	1	2	0
Cluster 5	1	0	1	0	34	1	0	1	1	25
Cluster 6	59	0	0	0	0	0	0	0	0	0
Cluster 7	2	0	0	0	0	3	88	0	0	0
Cluster 8	0	59	1	0	0	0	0	0	0	1
Cluster 9	2	0	3	10	3	0	0	0	69	2
Cluster 10	0	0	66	0	0	0	0	0	2	1

Accuracy Clustering Jerárquico

Los valores de accuracy para cada branch del HC son presentados a continuación. El promedio de accuracy ponderado por el número de elementos en cada branch es 62.5%. Este valor es superior al obtenido con el método de SOM + Kmeans.

Cluster	Acc
Cluster 1	0.4015748
Cluster 2	0.3428571
Cluster 3	1.0000000
Cluster 4	0.3906250
Cluster 5	0.5312500
Cluster 6	1.0000000
Cluster 7	0.9462366
Cluster 8	0.9672131
Cluster 9	0.7752809
Cluster 10	0.9565217

Conclusiones

En el caso de (Palamara, Piglione, and Piccinini 2011), el método de clusterización SOM + Kmeans generó mejores resultados que el método de Clustering Jerárquico. En este trabajo, sin embargo, los resultados obtenidos sugieren lo contrario. Esto puede deberse a las diferencias que existen entre los data sets de cada trabajo. El data set de accidentes laborales está descrito por 48 variables binarias, y los autores señalan que aplicar SOM antes de clusterizar puede ser útil para variables categóricas pues permite capturar relaciones entre elementos en un espacio continuo. En el caso de MNIST, las variables son numéricas, por lo que esa ventaja no aplicaría.

Otro factor que puede influir en los resultados es el número de variables de cada elemento. Los modelos fueron alimentados directamente con el valor de cada pixel de las imágenes para un total de 784 variables (28x28px), a diferencia de las 48 variables de la base de accidentes. Adicionalmente, se ha visto en clases que los modelos de clasificación y/o clusterización de imágenes pueden ser muy sensibles a la posición de los objetos de interés dentro de la misma imagen. Aunque esto debería afectar ambas metodologías, es posible que al aplicar Kmeans sobre los vectores prototipos de SOM se esté construyendo sobre el error del método inicial. El error de SOM también puede ser mayor por el hecho de que cada vector prototipo arrastra a sus vecinos en cada iteración.

Como trabajo futuro, se propone pre-procesar las imágenes antes de incorporarlas a los modelos de clusterización. El objetivo es reducir el error asociado al uso de los pixeles crudos. Se plantean como alternativas la extracción de componentes principales, o la convolución de imágenes para identificar patrones relevantes.

Bibliografía

Dalpiaz, David. 2019. "Load the MNIST Handwritten Digits Dataset into R as a Tidy Data Frame. Gist." Accessed June 27. https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706.

Kohonen, T. 1990. "The Self-Organizing Map." *Proceedings of the IEEE* 78 (9): 1464–80. doi:10.1109/5.58325.

LeCun, Yann, Corinna Cortes, and Christopher Burges. 2019. "MNIST Handwritten Digit Database." Accessed June 27. http://yann.lecun.com/exdb/mnist/.

Palamara, Federica, Federico Piglione, and Norberto Piccinini. 2011. "Self-Organizing Map and Clustering Algorithms for the Analysis of Occupational Accident Databases." *Safety Science* 49 (8): 1215–30. doi:10.1016/j.ssci.2011.04.003.

Wehrens, Ron, and Johannes Kruisselbrink. 2018. "Flexible Self-Organizing Maps in **Kohonen** 3.0." *Journal of Statistical Software* 87 (7). doi:10.18637/jss.v087.i07.