

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“IMPLEMENTACIÓN DE UNA LIBRERÍA EN PYTHON PARA GENERACIÓN AUTOMÁTICA DE MAPAS URBANOS”

AXEL IVÁN REYES ORELLANA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Marcelo Mendoza Rocha
Profesor Correferente: Claudio Lobos

Agosto - 2023

DEDICATORIA

Quisiera dedicar este trabajo a toda la gente que me acompañó. A mi madre y padre que siempre me han apoyado, a mis hermanos y familiares con los que he compartido buenos momentos, y a mis amigos que fui conociendo a lo largo del tiempo. Especialmente al Closet que recorrimos juntos este camino. Y un saludo para HK Panas que este último año hemos compartido varias cosas.

AGRADECIMIENTOS

Quisiera agradecer a mi profesor Marcelo Mendoza que me otorgó las facilidades para trabajar en esta memoria al incorporarme en el Instituto Milenio Fundamentos de los Datos (IMFD). También agradecer el apoyo que recibí del grupo de investigación Ciudades del Proyecto Emblemático 1 del IMFD.

RESUMEN

Resumen— Con el motivo de conocer qué sectores urbanos poseen indicadores de interés con valores homogéneos, de manera rápida y automática, se propone la implementación de una librería en Python con 5 algoritmos de clustering urbano distintos: Deep Modularity Networks (DMoN), Gaussian Mixture Model (GMM), K-Nearest Neighbours (KNN), Self Organized Map (SOM) y un método basado en la Teoría de la Información (TDI). También se ofrecen herramientas de comparación de mapas urbanos: Adjusted Rand Index (ARI) y Adjusted Mutual Information (AMI), y por último una documentación. Se desarrolla un caso de estudio con datos de la Región Metropolitana de Chile, se mide el uso de recursos de los algoritmos y se valida el correcto funcionamiento de estos al generar mapas urbanos de manera automática. También se valida el funcionamiento de las herramientas de comparación.

Palabras Clave— Clustering; Mapas; Urbanismo; Segregación.

ABSTRACT

Abstract— With the purpose of identifying urban sectors that exhibit homogeneous values of interest indicators quickly and automatically, the implementation of a Python library is proposed. This library includes 5 different urban clustering algorithms: Deep Modularity Networks (DMoN), Gaussian Mixture Model (GMM), K-Nearest Neighbours (KNN), Self Organized Map (SOM) and a method based on Information Theory (TDI). Also, urban map comparison tools are provided: Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI), and a documentation. A case study is conducted using data from the Metropolitan Region of Chile to measure the resource usage of these algorithms and validate their proper functioning in generating urban maps automatically. The functionality of the comparison tools is also validated.

Keywords— Clustering; Maps; Urbanism; Segregation.

GLOSARIO

BMU: Best Matched Unit

DI: Departamento de Informática.

DMoN: Deep Modularity Networks.

GMM: Gaussian Mixture Model

KNN: K-nearest neighbours

GCN: Graph Convolutional Network

GNN: Graph Neural Network

SOM: Self Organized Maps.

UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	VIII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Contextualización	2
1.2 Objetivos	3
1.2.1 Objetivo general:	3
1.2.2 Objetivos específicos:	3
CAPÍTULO 2: MARCO CONCEPTUAL	4
2.1 Definiciones previas	4
2.2 Trabajos relacionados	4
2.3 Métodos de interés	5
2.3.1 Deep Modularity Networks (DMoN)	5
2.3.2 Modelo de Mezcla de Gaussianas (GMM)	7
2.3.3 Vecinos Cercanos (KNN)	8
2.3.4 Self Organized Map (SOM)	11
2.3.5 Método basado en Teoría de la Información (TDI)	12
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	14
3.1 Preprocesamiento	14
3.1.1 Dar formato a tabla	14
3.1.2 Matriz de adyacencia	15
3.1.3 Anillos	16
3.2 Métodos de Clustering	18
3.2.1 DMoN	18
3.2.2 GMM	19
3.2.3 KNN	20
3.2.4 SOM	22
3.2.5 TDI	23
3.3 Visualización de Clustering	24
3.3.1 Graficar Mapa completo	24
3.3.2 Graficar Muestreo aleatorio	24

3.4 Evaluación de Clustering	25
3.4.1 Adjusted Rand Index (ARI)	25
3.4.2 Adjusted Mutual Information (AMI)	26
3.5 Dataset Región Metropolitana	27
3.5.1 Origen	27
3.5.2 Características Visuales	28
3.5.3 Características Sociales	28
3.5.4 Características de Suelo	29
3.5.5 Uso	30
3.6 Documentación	30
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	32
4.1 Uso de recursos	32
4.2 Generación de mapas	32
4.3 Herramientas de comparación de mapas	33
CAPÍTULO 5: CONCLUSIONES	34
REFERENCIAS BIBLIOGRÁFICAS	38

ÍNDICE DE FIGURAS

1 Ejemplo gráfico de la confección de vecindarios para KNN.	10
2 Esquema de la topología de la red SOM.	11
3 Documentación de SpatialCluster	31
4 Tiempo de ejecución.	33
5 Uso de Ram.	34
6 Uso de CPU.	35
7 Mapas generados por los métodos con SpatialCluster	36
8 Matriz de los puntajes de ARI para cada par de mapas.	37
9 Matriz de los puntajes de AMI para cada par de mapas.	38

ÍNDICE DE TABLAS

INTRODUCCIÓN

La tarea de realizar clustering urbano corresponde a detectar unidades geográficas que son internamente homogéneas y distintas de su entorno. Tiene aplicaciones en la planificación urbana y en estudios de segregación. Sin embargo, existen diversos métodos de clustering y una variedad de datos disponibles que no está claro cómo o cuáles usar para cumplir con la tarea de manera óptima.

En este trabajo se propone la implementación de una librería en Python para generación automática de mapas urbanos, llamada SpatialCluster. Esta librería incluye diversas herramientas de preprocesamiento (Matriz de Adyacencia, Anillos Concéntricos), evaluación, comparación y visualización de mapas (Adjusted Rand Index y Adjusted Mutual Information), y 5 algoritmos de interés que permiten realizar la tarea de clustering urbano. También disponibiliza un conjunto de datos de la Región Metropolitana de Chile tanto para poder probar las herramientas que provee SpatialCluster como para estudiar la región. Además, se desarrolla una documentación de la librería que abarca los 5 módulos principales de esta y explica cómo usarlos.

Los algoritmos considerados corresponden a Deep Modularity Networks (DMoN), Gaussian Mixture Model (GMM), K-Nearest Neighbours (KNN), Self Organized Map (SOM) y un método basado en la Teoría de la Información (TDI). Son incluidos en SpatialCluster debido a sus diferentes cualidades y formas de abordar el problema. Por otra parte, son algoritmos que no se encuentran disponibles en un entorno en común, lo cual hace más difícil poder experimentar y usarlos.

Una vez implementada la librería, se desarrolla un caso de estudio con datos de la Región Metropolitana de Chile para validar el correcto funcionamiento de esta. Se mide el uso de recursos para los 5 métodos de clustering, se hace una demostración de los mapas que genera cada uno de estos y finalmente se comparan los resultados de los clusterings con el uso de las herramientas de evaluación y comparación de SpatialCluster.

El documento consta con la siguiente estructura: Definición del problema, Marco conceptual, Propuesta de solución, Validación de la solución, Conclusiones.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contextualización

La tarea de realizar clustering urbano es un desafío que consiste en identificar zonas dentro de una ciudad que sean homogéneas internamente y diferentes de sus alrededores [Romano *et al.*, 2016]. Esta tarea permite estudiar propiedades de zonas urbanas y la planificación de políticas aplicadas a estas. Estos métodos de *clustering* han resultado útiles para medir y analizar la segregación residencial [Roberto, 2018], así como la influencia de espacios públicos en la concentración territorial basada en grupos étnicos [Fowler *et al.*, 2016].

La planificación urbana corresponde a analizar y evaluar las condiciones de funcionamiento de las ciudades para así llevar a cabo propuestas de diseño y proyectos urbanos que regulan condiciones del desarrollo económico, social y espacial. Esto requiere de una programación, seguimiento y control bien definido [Ornés, 2009].

Por otro lado, los indicadores son medidas cuantificadas recurrentes las cuales pueden ser analizadas a lo largo del tiempo e incluso ver cómo se ven afectadas frente a algún fenómeno en particular. El uso de indicadores es muy requerido en la administración y servicios públicos, ya que sirven para monitorear y evaluar varios aspectos urbanos tal como la calidad de vida o la situación socioeconómica [Kitchin *et al.*, 2015].

Con la gran cantidad de datos que se recolectan de los ciudadanos, dispositivos y también de las fuentes tradicionales, como censos y encuestas, las zonas urbanas tienen la oportunidad de monitorear y manejar la infraestructura y planificación urbana rápidamente. Esto permite que se pueda manejar y dar forma al futuro de las ciudades [Engin *et al.*, 2020]. El análisis de esta masiva cantidad de datos está adquiriendo un rol significativo en detectar características claves de las ciudades, permitiendo eficiencia en operaciones y funciones, uso prudente de recursos naturales, manejo inteligente de instalaciones e infraestructuras, mejora de movilidad y accesibilidad, y por sobre todo mejora en la calidad de vida y bienestar de los ciudadanos [Bibri, 2019].

Las tareas automatizadas son necesarias para realizar una eficaz gestión de los procesos urbanos [Font-Casaseca, 2021], en el caso del análisis de indicadores, se utilizan mapas de atributos urbanos. Estos corresponden a puntos que están localizados en el plano y que tienen mediciones para un conjunto de indicadores de interés. El mapa urbano genera una agrupación (*clustering*) de zonas urbanas donde estos atributos son homogéneos.

Los indicadores cambian a lo largo del tiempo, haciendo necesario analizar el cambio en la distribución de estos a nivel espacial. Es por esto que surge la necesidad de conocer qué sectores poseen indicadores de interés con valores homogéneos de manera rápida y auto-

mática.

La representación de la información desempeña un papel crucial al realizar clustering en datos urbanos, dado que estos datos suelen estar compuestos por múltiples y diversos indicadores. No existe consenso sobre la mejor manera de combinar estos datos o cómo representarlos de forma óptima.

Actualmente existen diferentes métodos de *clustering* y regionalización. Sin embargo, no es claro cuál método entrega mayor información relevante para resolver el problema. También, es necesario tener estos métodos en un entorno coherente para poder utilizarlos y evaluarlos cómodamente.

PySAL es a una librería que permite utilizar algunos métodos de regionalización, sin embargo no permite realizar *clustering*, tarea la cual aporta información diferente e incluso complementaria. En la sección 2.1 de este trabajo se profundizará más en las diferencias entre estas dos tareas.

1.2. Objetivos

1.2.1. Objetivo general:

Implementar un framework de comparación de técnicas generativas de mapas urbanos.

1.2.2. Objetivos específicos:

- Crear una librería en Python para la generación automática de mapas urbanos.
- Desarrollar herramientas de validación y comparación de mapas en la librería.
- Diseñar una documentación para la librería.
- Validar el correcto funcionamiento de la librería por medio de caso de estudio.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Definiciones previas

Un algoritmo de agrupamiento (*clustering*) trabaja sobre un conjunto de puntos los cuales tienen ciertas características, ya sean sus coordenadas u otros atributos de interés. Estas características son utilizadas para determinar similitud entre puntos, de tal manera que puntos más similares se considerarán como parte del mismo grupo y puntos más diferentes tenderán a estar en distintos grupos. Estos grupos resultantes son los llamados *clusters*, mientras que al proceso de asignar los puntos a estos grupos se le llama *clustering*.

Otra tarea que es relevante definir es la regionalización. Esta corresponde a un método de particionamiento del espacio en zonas homogéneas respecto a los atributos o indicadores a estudiar, y que requiere que estos sean geográficamente consistentes [Sánchez Undurraga, 2021], siendo esta última propiedad la mayor diferencia respecto al *clustering*. Dos candidatos pueden ser considerados como parte del mismo grupo sólo si existe un camino desde un miembro al otro de tal manera que este nunca salga de la región. Si bien la regionalización suele ser más coherente geográficamente, esto suele provocar que se ajuste peor a las características, por lo que existe un problema de compensación entre estas dos propiedades.

Debido a que las técnicas de regionalización y agrupamiento entregan información diferente según sea el caso, estas pueden utilizarse de forma que sus resultados se complementen, ya que reducen datos con varios atributos a representaciones más simples y permiten que se entienda de mejor manera la distribución en el espacio que tienen al utilizar estas dos tareas con enfoques distintos.

2.2. Trabajos relacionados

Trabajos previos han realizado estudios de análisis de datos urbanos y generación de mapas. En [Barrai et al., 2012] se estudió la población de Chile a través de la distribución de apellidos que existen, aprovechando la práctica hispana utilizada (en la mayoría de los casos) de heredar el primer apellido del padre y el segundo apellido de la madre.

En [Bro y Mendoza, 2021], se estudió la distribución de apellidos de Santiago, Chile. Se generaron dos redes basadas en los apellidos, la primera formada por los pares de apellidos paternos y maternos, y la segunda formada de la distancia medida a través de la isonimia entre vecindarios de la ciudad, revelando la distribución espacial de etnias. También haciendo cruce con un índice socioeconómico, se encontró que apellidos de alto estatus tienden

a agruparse entre sí. Se sugiere la existencia de un alto grado de segregación urbana en la ciudad de Santiago.

En [Mendoza y Bro, 2021], se continuó trabajando con la distribución de apellidos en Santiago, Chile. A partir de registros administrativos se creó una red de afinidad de apellidos que codifican datos socioeconómicos. A partir de esta red se logra realizar predicción de enlaces entre nodos originalmente desconectados de manera bastante certera. Finalmente, se discuten los resultados obtenidos de manera que explique los altos niveles de endogamia en Santiago.

2.3. Métodos de interés

A continuación, se revisarán algunos métodos de *clustering* junto a sus cualidades y ventajas para el contexto del problema.

2.3.1. Deep Modularity Networks (DMoN)

Las Redes Neuronales basadas en Grafos (GNN) han sido bastante utilizadas en tareas supervisadas y semi supervisadas, es por esto que se propuso DMoN, el cual realiza detección de *clusters* en grafos entregando interesantes resultados [Tsitsulin et al., 2020].

DMoN corresponde a un modelo capaz de realizar *clustering* no supervisado que aprovecha también de la topología intrínseca que pueden tener los datos, ya que trabaja sobre el grafo que representa las conexiones entre los puntos a agrupar utilizando Redes Neuronales basadas en Grafos (GNN), permitiendo una optimización a la asignación de *clusters* [Tsitsulin et al., 2020].

La detección de *clusters* urbanos es una importante tarea con aplicaciones en políticas urbanas. Debido a esto, varios métodos han sido propuestos para resolver el problema, pero no es claro qué tan estables son los resultados que estos entregan respecto a los hiperparámetros necesarios para usarlos. DMoN en cambio, muestra ser robusto a variaciones de estos.

Según un estudio realizado en [Vera et al., 2022], en el cual se comparaban mapas de *clustering* urbano realizados por DMoN y *Gaussian Mixture Models* (GMM), se detectó una tendencia a que la gente prefería los mapas creados por DMoN. Debido a su capacidad de poder manejar de mejor manera vectores de datos de mayor dimensionalidad, DMoN presentó una performance mejor que la de GMM.

A continuación, se muestra la propuesta del modelo planteado en [Tsitsulin et al., 2020].

Sea $X^0 \in \mathbb{R}^{n \times s}$ donde n corresponde a la cantidad de nodos y s a la cantidad de atributos

por nodo, y sea $\tilde{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ la matriz de adyacencia normalizada, la salida para la capa t -ésima X^{t+1} corresponde a

$$X^{t+1} = SeLU(\tilde{A}X^tW + X^0W_{skip})$$

Donde $W, W_{skip} \in \mathbb{R}^{sxs}$ corresponden a parámetros entrenables y $SeLU$ corresponde a la siguiente función

$$SeLU(x) = \lambda \begin{cases} x & \text{si } x > 0 \\ \alpha e^x - \alpha & \text{si } x \leq 0 \end{cases}$$

Con λ y α hiperparámetros del modelo.

La tarea consta de definir una matriz de asignación $C \in 0, 1^{n \times k}$, donde k corresponde a la cantidad de particiones a encontrar. Esta matriz determina a qué *cluster* pertenece cada nodo y se obtiene de la siguiente forma

$$C = softmax(GCN(\tilde{A}, X))$$

Donde $GCN(\tilde{A}, X)$ corresponde a pasar el grafo por el modelo convolucional multi-capas, el cual opera sobre los atributos de los datos y la matriz de adyacencia normalizada, y $softmax(\cdot)$ corresponde a la siguiente función aplicada a cada fila de la matriz X^t

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^s e^{x_j}}$$

Para entrenar el modelo, se trabaja con la siguiente matriz

$$B = A - \frac{dd^T}{2m}$$

Donde d corresponde al vector de grados del grafo y m a la cantidad de aristas de este. Luego, se define la siguiente función de pérdida

$$\mathcal{L} = -\frac{1}{2m}Tr(C^TBC) + \frac{\sqrt{k}}{n} \left\| \sum_i C_i^T \right\|_F - 1$$

Donde $Tr(\cdot)$ es la traza de la matriz y $\|\cdot\|_F$ la norma de Frobenius.

Cabe destacar que el primer término se asocia al cálculo de la modularidad $\mathcal{Q}$, la cual es una medida para evaluar la calidad del *clustering* realizado, y el segundo término se asocia a una regularización que asegure *clusters* informativos.

2.3.2. Modelo de Mezcla de Gaussianas (GMM)

Los modelos de mezcla gaussianas (Gaussian Mixture Models o GMM) son el método que comúnmente se utiliza para realizar *clustering* urbano. Corresponde a una técnica de aprendizaje automático no supervisado utilizada para modelar y describir distribuciones de datos que pueden estar compuestas por múltiples componentes o subpoblaciones. Estos modelos se basan en la suposición de que los datos provienen de una mezcla de varias distribuciones gaussianas.

La idea principal detrás de GMM es representar la distribución de datos como una combinación lineal de varias distribuciones gaussianas, donde cada componente gaussiano representa una subpoblación específica dentro de los datos. A continuación, se explican las bases del método en el contexto de clustering [Reynolds, 2009].

La combinación lineal de gaussianas, considerando $x \in \mathbb{R}^D$, se denota de la siguiente forma

$$p(x|\lambda_k) = \sum_{k=1}^m w_k g(x|\mu_k, \Sigma_k)$$

Donde λ_k corresponde al conjunto de parámetros a estimar de la combinación lineal de gaussianas (w_k, μ_k, Σ_k) , m corresponde a la cantidad total de clusters y w_k corresponde al peso asociado al cluster k . Cabe destacar que $\sum_{k=1}^m w_k = 1$.

Luego, la probabilidad $g(x|\mu_k, \Sigma_k)$ de cada gaussiana se define como

$$g(x|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right\}$$

Ahora, se inicializan los parámetros w_k, μ_k, Σ_k y se aplica Expectation-Maximization (EM), el cual corresponde a un algoritmo de dos pasos iterativos: el paso E (Expectation) y el paso M (Maximization). A continuación, se detallan estos pasos:

1. Paso E (Expectation):

En este paso, se calculan las probabilidades de pertenencia de cada punto x_t a cada cluster gaussiano con los valores de los parámetros actuales.

$$p(k|x_t, \lambda_k) = \frac{w_k g(x_t|\mu_k, \Sigma_k)}{\sum_{k=1}^m w_k g(x_t|\mu_k, \Sigma_k)}$$

2. Paso M (Maximization):

En este paso, se actualizan los parámetros del modelo GMM utilizando las probabilidades calculadas en el paso E.

La actualización se realiza mediante la maximización de la función de verosimilitud, que es una medida de cuánto se ajusta el modelo a los datos observados.

$$L(X|\lambda) = \prod_{t=1}^T p(x_t|\lambda)$$

Donde T corresponde a la cantidad total de datos. Las fórmulas de actualización son las siguientes

$$\begin{aligned}\bar{w}_k &= \frac{1}{T} \sum_{t=1}^T p(k|x_t, \lambda_k) \\ \bar{\mu}_k &= \frac{\sum_{t=1}^T p(k|x_t, \lambda_k) x_t}{\sum_{t=1}^T p(k|x_t, \lambda_k)} \\ \bar{\Sigma}_k &= \frac{\sum_{t=1}^T p(k|x_t, \lambda_k) (x_t - \mu_k)(x_t - \mu_k)^T}{\sum_{t=1}^T p(k|x_t, \lambda_k)}\end{aligned}$$

Estos dos pasos se repiten iterativamente hasta que se alcanza la convergencia. En cada iteración, se recalculan las probabilidades $p(k|x_t, \lambda_k)$ y se actualizan los parámetros del modelo GMM.

La convergencia se puede verificar mediante diferentes criterios, como la diferencia en el valor de la función de verosimilitud entre iteraciones consecutivas o la estabilidad de los parámetros.

Finalmente, una vez que se alcanza la convergencia, para asignar un dato x_t a un cluster, basta con escoger el cluster k que maximice $p(k|x_t, \lambda_k)$.

2.3.3. Vecinos Cercanos (KNN)

Para análisis de datos con áreas y delimitaciones, surge el problema de que si se mantiene una escala fija para la agregación, se cae en el riesgo de obtener resultados solo válidos para esa configuración. Este percance se denomina como el problema de la unidad de área modificable (MAUP). Para solventarlo, se utiliza la razón o diferencia entre el valor para la

unidad de área y un valor de un área de referencia más grande. Sin embargo, también se puede caer en un resultado arbitrario si no se analiza que este sea válido para más de un área de delimitación específica. Para abordar este problema, se utiliza el método de Vecinos Cercanos, el cual permite una agregación multiescalar de los k vecinos más cercanos y utiliza un área de referencia más grande que sea consistente con los vecinos utilizados [Hennerdal y Nielsen, 2017].

A continuación, se muestra la propuesta del modelo planteado en [Hennerdal y Nielsen, 2017].

Se define $t(x_i)$ como el número de individuos en la ubicación x_i y $f(x_i)$ como el número de individuos con la propiedad f en la ubicación x_i . Luego, según la ecuación 1, se crea un conjunto de ubicaciones $N_{j,k}$ por cada ubicación x_j y valor de k con el menor valor posible de $r_{j,k}$, de manera que el criterio de la ecuación 2 se cumpla.

$$N_{j,k} = \{x_i \in \mathbb{R} : |x_i - x_j| \leq r_{j,k}\} \quad (1)$$

$$\sum_{x_i \in N_{j,k}} t(x_i) \geq k \quad (2)$$

De esta manera, el conjunto $N_{j,k}$ consiste en las ubicaciones más cercanas a x_j , los cuales suman un total de al menos k individuos. Se puede ver un ejemplo de esta definición en la figura 1.

Luego, la suma de todos los individuos de las ubicaciones que son parte del conjunto $N_{j,k}$ se denota por $T_{j,k}$ y la suma de todos los individuos que cumplen con la propiedad f y pertenecen a las ubicaciones incluidas en el conjunto $N_{j,k}$ se denotan por $F_{j,k}$. Esto se puede ver reflejado en las ecuaciones 3 y 4.

$$T_{j,k} = \sum_{x_i \in N_{j,k}} t(x_i) \quad (3)$$

$$F_{j,k} = \sum_{x_i \in N_{j,k}} f(x_i) \quad (4)$$

Para verificar que el conjunto de ubicaciones encontradas en realidad entregan información y no es un conjunto aleatorio, se calcula la probabilidad de que si los datos fuesen escogidos aleatoriamente del conjunto más grande $N_{j,K}$ ($K > k$), al menos $F_{j,k}$ individuos con la propiedad f se encuentren en el conjunto $N_{j,k}$. Esta probabilidad se calcula según la ecuación 5, usando la probabilidad acumulada de la distribución hipergeométrica y adoptando la probabilidad de sacar individuos sin reposición de un conjunto como hipótesis nula.

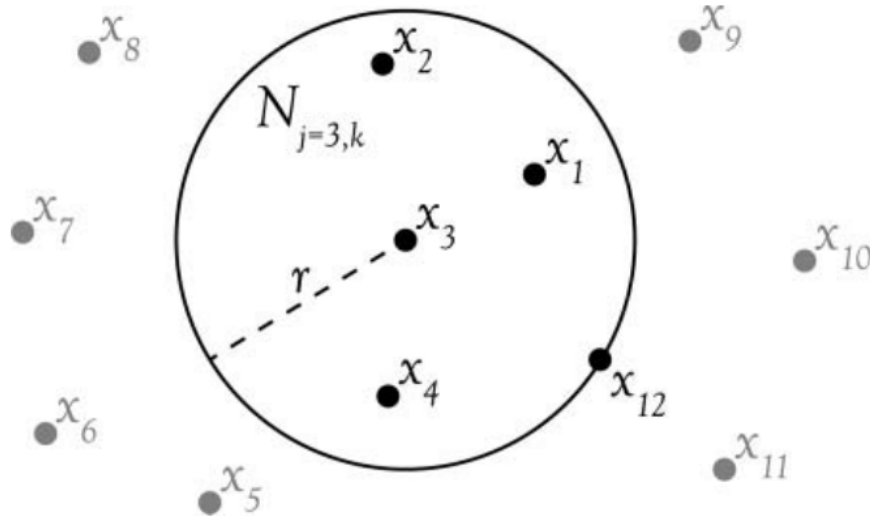


Figura 1: Ejemplo gráfico de la confección de vecindarios para KNN.

Si se considera $j = 3$, el conjunto $N_{j=3,k}$ correspondería a las ubicaciones encerradas por la circunferencia de radio r ($x_1, x_2, x_3, x_4, x_{12}$), donde la suma de individuos debe cumplir la restricción de ser al menos k .

Fuente: [Hennerdal y Nielsen, 2017].

$$p_{j,kK}^{hot} = \sum_{z=F_{j,k}}^{T_{j,k}} \frac{\binom{F_{j,K}}{z} \binom{T_{j,K}-F_{j,K}}{T_{j,k}-z}}{\binom{T_{j,K}}{T_{j,k}}} \quad (5)$$

Dada una ubicación x_j , cuando su probabilidad $p_{j,kK}^{hot}$ sea menor que un α , será considerado como un punto caliente (*hot spot*) con una proporción mayor estadísticamente significativa de la propiedad f entre los k vecinos más cercanos en comparación a la proporción entre los K vecinos más cercanos.

Análogamente, para una ubicación x_j se puede calcular la probabilidad $p_{j,kK}^{cold}$ como se muestra en la ecuación 6. Cuando esta sea menor que un α , será considerado como un punto frío (*cold spot*) con una proporción menor estadísticamente significativa de la propiedad f entre los k vecinos más cercanos en comparación a la proporción entre los K vecinos más cercanos.

$$p_{j,kK}^{cold} = \sum_{z=0}^{F_{j,k}} \frac{\binom{F_{j,K}}{z} \binom{T_{j,K}-F_{j,K}}{T_{j,k}-z}}{\binom{T_{j,K}}{T_{j,k}}} \quad (6)$$

Finalmente, para realizar *clustering* multiescalar, los cálculos de $p_{j,kK}^{hot}$ y $p_{j,kK}^{cold}$ pueden obtenerse para distintas combinaciones de k y K , de forma que ambas probabilidades sean menor que α para cada ubicación.

2.3.4. Self Organized Map (SOM)

Agrupar puntos espaciales en *clusters* significativos y relevantes es una tarea difícil, más aún si se toma en cuenta que se debe lidiar con el problema de la unidad de área modificable (MAUP), la auto correlación espacial y la multicolinealidad de atributos. Para esto se propone trabajar con un algoritmo de regionalización jerárquica que es sensible a variaciones de escala de correlaciones espaciales multivariadas, de esta manera, se pueden revelar significativas interacciones entre diversos fenómenos espaciales que con otros métodos se tratan simplemente como ruido [Sánchez Undurraga, 2021].

El algoritmo de regionalización utilizado corresponde a un *Self Organized Map* (SOM), el cual incorpora restricciones espaciales a una red neuronal y arroja como resultado una grilla de neuronas. La red se entrena considerando primero las proximidades geográficas y luego los relaciona con los patrones de los atributos de los puntos. Iterativamente los datos son llevados al espacio de salida, donde las observaciones son pesadas según su proximidad a cada neurona, resultando en una representación topológica plana de puntos inicialmente multidimensionales [Sánchez Undurraga, 2021].

El método de *Self Organized Map* también se ha utilizado para realizar *clustering*. En [Olteanu et al., 2020], se utilizó la librería de R llamada *SOMbrero*, la cual ejecuta SOM combinado con *clustering* jerárquico aglomerativo de forma estocástica.

A continuación, se muestra el proceso de entrenamiento y *clustering* de SOM [Almi'ani et al., 2018].

Para el entrenamiento de SOM, se tienen J neuronas las cuales tienen una posición topológica específica en la grilla. Los datos de entrada D se componen de N registros de conexiones. Cada registro es representado por un vector $\vec{R} = [r_1, r_2, \dots, r_n]$ con n la cantidad de características o atributos. Cada neurona es representada como un vector $\vec{W} = [w_1, w_2, \dots, w_n]$.

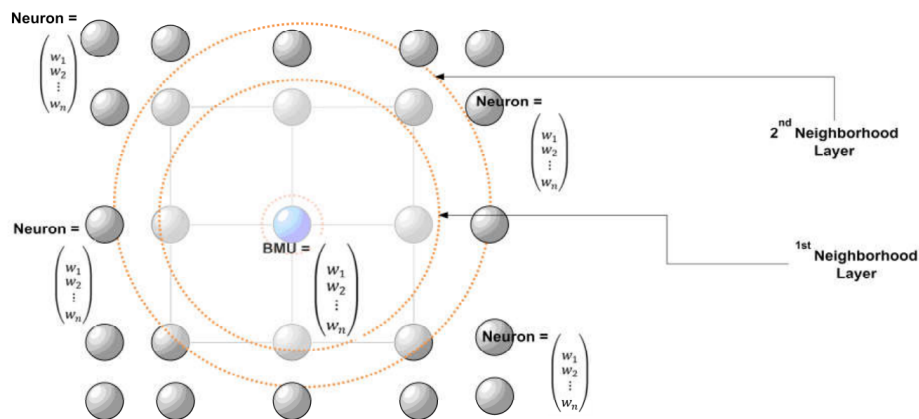


Figura 2: Esquema de la topología de la red SOM.

Fuente: [Almi'ani et al., 2018].

En la figura 2 se puede ver una ilustración de la topología de la red SOM.

Por cada iteración de entrenamiento t , se aplican los siguientes pasos hasta cumplir algún criterio de parada establecido:

Primero se inicializan los vectores que representan las neuronas. Luego de esto, se calcula la distancia entre cada j -ésima neurona $\vec{W}_j$ y el vector de entrada $\vec{R}$ utilizando la ecuación 7.

$$d_j = \left\| \vec{R} - \vec{W}_j \right\| \quad (7)$$

Después de calcular la distancia anterior para todos los vectores de neuronas $\vec{W}_j$ respecto al vector de entrada $\vec{R}$, la neurona con menor distancia se activa y se le llama *Best Matched Unit* (BMU). Cabe destacar que de esto surge la propiedad de la ecuación 8.

$$\left\| \vec{R} - \vec{W}_{bmu} \right\| \leq \left\| \vec{R} - \vec{W}_j \right\| \quad (8)$$

El siguiente paso corresponde a la adaptación de las neuronas, donde a los vectores de la neurona BMU y sus vecinas se les suma una cantidad $\Delta \vec{R}_j$, la cual se calcula según la ecuación 9.

$$\Delta \vec{R}_j = \eta(t) \cdot h(bmu, j, t) \cdot (\vec{R} - \vec{W}_{bmu}) \quad (9)$$

Donde $\eta(t)$ corresponde a la tasa de aprendizaje y $h(bmu, j, t)$ corresponde a la función de topología. Existen varias funciones que pueden ser utilizadas para h , tales como *cone function*, *bubble function*, *exponential function*, *ricker wavelet function*, *cylinder function* o *gaussian bell function*.

El entrenamiento termina según el criterio de parada establecido, ya sea alcanzar un cierto numero de iteraciones T o que el error sea menor a algún umbral.

Finalmente, para cada dato se puede obtener su respectivo BMU para determinar a qué cluster se asignará. También se puede aplicar algún algoritmo de *clustering* sobre las neuronas resultantes del entrenamiento de SOM.

2.3.5. Método basado en Teoría de la Información (TDI)

Modelar la estructura y dinámica de la segregación es un problema para la planificación urbana y también para la sociología. Aunque existen métodos propuestos como solución, estos tienen limitaciones. Es por esto que se propone un método para el estudio de la estructura de la segregación basado en la teoría de la información [Chodrow, 2017].

Este método se desarrolló inicialmente como una extensión de métodos de regionalización existentes y luego se formuló una medida local de escala espacial que permite el análisis tanto dentro de una ciudad como entre ciudades. El resultado de este método es que distingue detallada información de la dinámica y la estructura de la segregación urbana, la cual no es trivial de identificar o cuantificar. Además, el método es computacionalmente eficiente y aplicable a distintos estudios [Chodrow, 2017].

El enfoque propuesto implica el cálculo de una matriz de pesos utilizando la matriz de adyacencia que representa la topología de los datos. Para esto, se utiliza la Divergencia de Jensen-Shannon como medida de distancia. Considerando P y Q dos distribuciones de probabilidad, esta divergencia se define como

$$JSD(P||Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M)$$

Donde $M = \frac{1}{2}(P + Q)$ y $D(\cdot)$ corresponde a la Divergencia Kullback-Leibler

$$D(P||Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

Luego, se aplica Spectral Clustering, algoritmo que consiste en lo siguiente:

Primero, se calcula el Laplaciano del grafo de la siguiente forma

$$L = D - A$$

Donde D corresponde a la matriz de grados del grafo y A corresponde a la matriz de adyacencia (en este caso, la matriz de pesos).

Luego, se calculan los primeros k vectores propios, los cuales corresponden a los vectores asociados a los k valores propios más pequeños.

Estos vectores propios serán las columnas de una nueva matriz, donde la i -ésima fila de esta corresponderá a los atributos del i -ésimo nodo. Con estos nuevos atributos, finalmente se debe aplicar algún método de clustering, por ejemplo k-means.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

La solución propuesta corresponde al desarrollo de *SpatialCluster*, una librería en Python para generación automática de mapas urbanos. Esta se encuentra alojada en [PyPI](#) y el código fuente se encuentra en [GitHub](#).

Esta librería se construyó en base a 5 módulos principales: Preprocesamiento de datos, Métodos de *clustering*, Visualización de mapas, Evaluación de *clustering* y Dataset de la Región Metropolitana de Chile.

3.1. Preprocesamiento

Este módulo ofrece diversas funciones para el preprocesamiento de los datos, de manera que el uso de los métodos sea provechoso. A continuación se mencionan y explican las funciones que ofrece el módulo:

3.1.1. Dar formato a tabla

Para obtener los datos separados en variables, con el formato que utilizan los métodos de la librería, se ofrecen dos funciones distintas.

La primera corresponde a *attributes_format* que entrega los datos separados en posición (longitud, latitud) y atributos.

Parámetros

- **df** : (*Pandas DataFrame*) Contiene los datos a utilizar.

Retorno

- **features_position**: (*Pandas DataFrame*) Contiene la longitud y latitud de los datos.
- **features_X**: (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud).

```
1 from SpatialCluster.preprocess.data_format import  
   attributes_format  
2 features_position, features_X = attributes_format(df)
```

La segunda corresponde a *attributes_with_zone_format* que entrega los datos separados en posición (longitud, latitud); y atributos, con una columna extra que indica a qué zona pertenece cada punto (en el dataset de ejemplo corresponde a la comuna).

Parámetros

- **df** : (*Pandas DataFrame*) Contiene los datos a utilizar.
- **zona**: (*string*) Nombre de la columna que contiene la zona a la que pertenece cada punto.

Retorno

- **features_position**: (*Pandas DataFrame*) Contiene la longitud y latitud de los datos.
- **features_X**: (*Pandas DataFrame*) Contiene los atributos de los datos incluyendo la columna de zona (sin longitud ni latitud).

```
1 from SpatialCluster.preprocess.data_format import
   attributes_with_zone_format
2 features_position, features_X =
   attributes_with_zone_format(df, zona = "comuna")
```

3.1.2. Matriz de adyacencia

La función *adjacencyMatrix* crea una matriz de adyacencia en la que cada punto es relacionado con sus vecinos más cercanos siguiendo un criterio en específico.

Para crear la matriz de adyacencia se pueden usar los siguientes criterios para definir una vecindad:

- Por *k* vecinos más cercanos (criterio '*k*').
- Por vecinos dentro de un radio *r* (criterio '*r*').
- Por vecinos dentro de un radio *r*, con un mínimo de *k_min* vecinos. En caso de que no hayan suficientes puntos para superar ese umbral, se usarán *k* vecinos más cercanos (criterio '*rk*').

Parámetros

- **features_X**: (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud).

- **r:** (*float*) Distancia máxima en metros a la que se considerará a un punto como vecino (radio del vecindario para cada punto). Por defecto: 300.0
- **k:** (*int*) Cantidad de vecinos máxima que tendrá el vecindario para cada punto. Por defecto: 5
- **min_k:** (*int*) Cantidad mínima de vecinos que debe tener el vecindario en caso de usar el criterio 'rk'. Por defecto: 2
- **criteria:** (*string*) Criterio que se usará para determinar los vecindarios (*k*, *r*, *rk*). Por defecto: 'k'
- **leafsize:** (*int*) Número de puntos en los que el algoritmo de KDTree cambia a fuerza bruta ([Documentación de KDTree](#)). Por defecto: 10

Retorno

- **A:** (*Numpy Matrix*) Matriz de adyacencia

```
1 from SpatialCluster.preprocess.adjacency import
   adjacencyMatrix
2 A = adjacencyMatrix(features_position, r = 300.0, k = 5,
   min_k = 2, criteria = "k", directed = True, leafsize =
   10)
```

3.1.3. Anillos

La función *rings* utiliza la ponderación de los datos de los vecinos de cada punto, creando una nueva columna con estos datos. Esto se repite para cada columna original del dataset, permitiendo así suavizar la diferencia de los atributos entre puntos cercanos. Para cada columna original se crearán tantas columnas como cantidad de parámetros que se hayan ingresado en *max_radios* o *max_neighbours*, considerando para cada uno la nueva definición de vecindario correspondiente.

Por ejemplo: Si *max_radios* corresponde a (300, 400, 500), por cada columna de *features_X* se creará una columna que pondere definiendo vecindarios de 300 metros, luego otra columna que utilice vecindarios de 400 metros y finalmente otra columna que utilice vecindarios de 500 metros.

Para crear los anillos se pueden usar los siguientes criterios para definir un vecindario:

- Por *k* vecinos más cercanos (criterio 'k').

- Por vecinos dentro de un radio r (criterio 'r').
- Por vecinos dentro de un radio r , con un mínimo de k_{min} vecinos. En caso de que no hayan suficientes puntos para superar ese umbral, se usarán k vecinos más cercanos (criterio 'rk').

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud).
- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **criteria:** (*string*) Criterio que se usará para determinar los vecindarios (k , r , rk). Por defecto: 'k'
- **max_radios:** (*Lista de floats*) Lista de radios en metros que se utilizarán para definir los vecindarios. Por defecto: [200.0, 300.0, 400.0]
- **max_neighbours:** (*Lista de ints*) Lista de cantidad máxima de puntos que tendrán los vecindarios. Por defecto: [200, 500, 1000]
- **weight_mode:** (*string*) Criterio que se utilizará para la ponderación ('Simple' o 'Distance Inverse'). Por defecto: 'Simple'
- **keep_original_value:** (*bool*) Determinará si se conservan las columnas originales o no. Por defecto: True
- **smoothing:** (*float*) Parámetro que se utiliza para suavizar las distancias al momento de ponderar los datos (útil en caso de ponderar con el inverso de la distancia, en caso de que estas sean muy cercanas a 0). Por defecto: 1e-08
- **normalize:** (*bool*) Determina si se normalizan las distancias, lo cual evita que al ponderar por el inverso de la distancia algunos datos se inflen más de lo deseado. Por defecto: True
- **leafsize:** (*int*) Número de puntos en los que el algoritmo de KDTree cambia a fuerza bruta ([Documentación de KDTree](#)). Por defecto: 10

Retorno

- **features_rings:** (*Pandas DataFrame*) Contiene las nuevas columnas creadas.

```
1 from SpatialCluster.preprocess.rings import rings
2 features_rings = rings(features_X, features_position,
    criteria="k", max_radios=[200.0, 300.0, 400.0],
    max_neighbours=[200, 500, 1000], weight_mode="Simple",
    keep_original_value=True, smoothing=1e-08, normalize=
    True, leafsize=10)
```

3.2. Métodos de Clustering

3.2.1. DMoN

Deep Modular Neural network es una red neuronal basada en grafos (GNN) usada en este caso para realizar clustering no supervisado, se enfoca en optimizar la modularidad, la cual corresponde a una medida usada para determinar la calidad del clustering.

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud y latitud). DMoN no trabaja con strings y se recomienda que los datos sean tipo float.
- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **A:** (*Numpy matrix*) Matriz de adyacencia utilizada, en caso de que no se entregue una se utilizará el método de *adjacencyMatrix* de *SpatialCluster* para crear una. Por defecto: None
- **criteria:** (*string*) Criterio que se usará para crear la matriz de adyacencia (*k*, *r*, *rk*). Por defecto: 'k'
- **r_max:** (*float*) Radio máximo en metros que se utilizará para la creación de la matriz de adyacencia. Por defecto: 300.0
- **n_clusters:** (*int*) Cantidad de clusters que considerará el método para agrupar los datos. Por defecto: 4
- **reg:** (*float*) Parámetro dentro del rango [0,1] que pondera la regularización de la función de pérdida para así evitar soluciones triviales como agrupar todos los datos en el mismo cluster. Por defecto: 1.0
- **dropout:** (*float*) Parámetro del rango [0,1] para evitar el sobreajuste a los datos (overfitting). Valores altos permitirán evitar más el sobreajuste, pero afectará el entrenamiento haciendo necesario más datos o más épocas (epochs). Por defecto: 0.0

- **num_epochs:** (*int*) Indica cuánto durará el entrenamiento. Valores muy altos podrían provocar sobreajuste. Por defecto: 500
- **learning_rate:** (*float*) Parámetro dentro del rango [0,1] que indica la tasa de ajustes que irá realizando la red en cada época. Valores muy altos permitirán un cambio más agresivo, lo cual puede provocar que no logre encontrar el punto óptimo. Valores bajos implicará que el aprendizaje será más lento por lo cual necesitará más épocas y más datos para entrenar.

Retorno

- **DMoN_areas_to_points:** (*dict*) Para cada cluster guarda los puntos que pertenecen a este.
- **DMoN_clusters:** (*Numpy Array*) Contiene los puntos etiquetados según el cluster al que fueron asignado.

```
1 from SpatialCluster.methods.DMoN import DMoN_Clustering
2 DMoN_areas_to_points, DMoN_clusters = DMoN_Clustering(
    features_X, features_position, A = None, criteria = "k",
    r_max = 300.0, n_clusters = 4, reg = 1.0, dropout =
    0.0, num_epochs = 500, learning_rate = 0.001)
```

3.2.2. GMM

Este método permite utilizar una distribución de probabilidad del modelo de mezcla gaussiana para realizar clustering. Para la inclusión de este algoritmo en *SpatialCluster* se hizo uso del método *GaussianMixture* de *scikit-learn*.

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud). *GMM* no trabaja con strings y se recomienda que los datos sean tipo float.
- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **n_clusters:** (*int*) Indica la cantidad de clusters que considerará el método para agrupar los datos. Por defecto: 4
- **covariance_type:** (*string*) Describe el tipo de parámetros de covarianza. Debe ser uno de los siguientes:

- 'full': Cada componente tiene su propia matriz de covarianza general.
- 'tied': Todos los componentes comparten la misma matriz de covarianza general.
- 'diag': Cada componente tiene su propia covarianza de matriz diagonal.
- 'spherical': Cada componente tiene su propia varianza individual.

Por defecto: 'full'

- **tol:** (*float*) Umbral de tolerancia. Las iteraciones del proceso Expectation-Maximization (EM) pararán cuando la ganancia promedio de la cota inferior esté por debajo de este umbral. Por defecto: 1e-3
- **reg_covar:** (*float*) Parámetro no negativo para la regularización añadida a la diagonal de la matriz de covarianza. Permite asegurar que las matrices de covarianza sean positivas. Por defecto: 1e-6

Retorno

- **GMM_areas_to_points:** (*dict*) Para cada cluster guarda los puntos que pertenecen a este.
- **GMM_clusters:** (*Numpy Array*) Contiene los puntos etiquetados según el cluster al que fueron asignado.

```
1 from SpatialCluster.methods.GMM import GMM_Clustering
2 GMM_areas_to_points, GMM_clusters = GMM_Clustering(
    features_X, features_position, n_clusters = 4,
    covariance_type = "full", tol=1e-3, reg_covar=1e-6)
```

3.2.3. KNN

K-Nearest Neighbours es un enfoque que evita el problema de la unidad areal modificable (MAUP). Este método se basa en la agregación multiescalar de los k vecinos más cercanos de una localización en una comparación estadística con un área más grande de referencia en el que se utilizan K vecinos más cercanos (siendo $K > k$). Este método distingue entre dos tipos de clusters, los *hot spots* y los *cold spots*.

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud).

- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **attribute:** (*string*) Indica qué columna de *features_X* se utilizará para caracterizar a los datos.
- **threshold:** (*float/int/bool*) Umbral que determina si un dato cumple con el criterio según su columna *attribute*.
- **location:** (*string*) Nombre de la columna de *features_X* se utilizará para determinar la localización a la que pertenecen los datos (Por ejemplo: 'comuna').
- **condition:** (*string*) Operador que se utilizará ('>', '<', '>=', '<=', '==') para ver si el dato cumple con la condición o no. Por defecto: '<'
- **k:** (*int*) Cantidad mínima de vecinos cercanos se utilizarán para la unidad de área más pequeña. Por defecto: 1500
- **K:** (*int*) Cantidad mínima de vecinos cercanos se utilizarán para la unidad de área más grande. Por defecto: 5000
- **alfa:** (*float*) Umbral para evaluar si el dato corresponde a un *cold spot* o *hot spot* con una significancia estadística. Por defecto: 0.01
- **leafsize:** (*int*) Número de puntos en los que el algoritmo de KDTree cambia a fuerza bruta ([Documentación de KDTree](#)). Por defecto: 10

Retorno

- **KNN_areas_to_points:** (*dict*) Para cada cluster guarda los puntos que pertenecen a este.
- **KNN_clusters:** (*Numpy Array*) Contiene los puntos etiquetados según el cluster al que fueron asignado.

```
1 from SpatialCluster.methods.KNN import KNN_Clustering
2 KNN_areas_to_points, KNN_clusters = KNN_Clustering(
    features_X, features_position, attribute, threshold,
    location, condition="<", k=1500, K=5000, alfa = 0.01,
    leafsize=10)
```

3.2.4. SOM

Self Organized Map es un tipo de red neuronal artificial capaz de convertir relaciones estadísticas complejas y no lineales entre elementos de datos de alta dimensión en relaciones geométricas simples en una pantalla de baja dimensión. Este método aprovecha esta propiedad de SOM para realizar clustering. Para la inclusión de este algoritmo en *SpatialCluster* se hizo uso de la librería *MiniSOM*.

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud). *SOM* no trabaja con strings y se recomienda que los datos sean tipo float.
- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **som_shape:** (*tupla de ints*) Determina la topología de la red neuronal que usará *SOM* de la forma (cantidad_filas, cantidad_columnas). Por defecto: (2,2)
- **sigma:** (*float*) Dispersión de la función de vecindad, debe ser adecuada a la dimensión de *SOM* utilizada. Por defecto: 0.5
- **learning_rate:** (*float*) Parámetro dentro del rango [0,1] que corresponde a la cantidad de información inicial que se compartirá entre neuronas en cada iteración del proceso de entrenamiento de la red. Por defecto: 0.5
- **num_iterations:** (*int*) Cantidad de iteraciones que se realizarán en el proceso de entrenamiento. Por defecto: 100

Retorno

- **SOM_areas_to_points:** (*dict*) Para cada cluster guarda los puntos que pertenecen a este.
- **SOM_clusters:** (*Numpy Array*) Contiene los puntos etiquetados según el cluster al que fueron asignado.

```
1 from SpatialCluster.methods.SOM import SOM_Clustering
2 SOM_areas_to_points, SOM_clusters = SOM_Clustering(
    features_X, features_position, som_shape = (2,2),
    sigma=0.5, learning_rate=0.5, num_iterations = 100)
```

3.2.5. TDI

TDI corresponde a un método basado en Teoría de la Información, la cual utiliza una matriz de pesos utilizando la divergencia de Shannon como distancia entre los puntos. Luego se aplica Spectral Clustering sobre esta matriz. Este método permite curvas de perfil con escala espacial no constante y análisis de descomposición con unidades de área no arbitrarias.

Parámetros

- **features_X:** (*Pandas DataFrame*) Contiene los atributos de los datos (sin longitud ni latitud). TDI no trabaja con strings y se recomienda que los datos sean tipo float.
- **features_position:** (*Pandas DataFrame*) Contiene longitud y latitud de los datos.
- **n_clusters:** (*int*) Cantidad de clusters que considerará el método para agrupar los datos. Por defecto: 4
- **A:** (*Numpy matrix*) Matriz de adyacencia que representará la topología del grafo que se utilizará para realizar *Spectral clustering*. Este debe ser conexo y simétrico para asegurar resultados coherentes. En caso de que no se entregue ninguna matriz, se utilizará *adjacencyMatrix* para crear una. Por defecto: None
- **k:** (*int*) Cantidad de vecinos máxima que tendrá el vecindario para cada punto en la construcción de la matriz de adyacencia en caso de no entregar una. Si la matriz resultante no es conexa, se debe usar un *k* más grande. Por defecto: 20
- **leafsize:** Corresponde al número de puntos en los que el algoritmo de KDTree cambia a fuerza bruta ([Documentación de KDTree](#)). Por defecto: 10

Retorno

- **TDI_areas_to_points:** (*dict*) Para cada cluster guarda los puntos que pertenecen a este.
- **TDI_clusters:** (*Numpy Array*) Contiene los puntos etiquetados según el cluster al que fueron asignado.

```
1 from SpatialCluster.methods.TDI import TDI_Clustering
2 TDI_areas_to_points, TDI_clusters = TDI_Clustering(
    features_X, features_position, n_clusters=4, A=None, k
    =20, leafsize=10)
```

3.3. Visualización de Clustering

Dado que los datos a clusterizar corresponden a indicadores pertenecientes al ámbito urbano, es natural querer ver y localizar los resultados en el mapa de la ciudad correspondiente. Este módulo ofrece dos métodos para esto.

3.3.1. Graficar Mapa completo

El siguiente método genera el mapa con todos los puntos, coloreados según el cluster asignado.

Parámetros

- **gdf:** (*GeoPandas DataFrame*) Contiene los clusters a los que pertenece cada punto y la geometría para localizarlos geográficamente en el mapa. En caso de que la columna 'geometry' no exista, el método la generará a partir de las columnas 'lon' y 'lat'.
- **markersize:** (*int*) Tamaño de los puntos en el mapa. Por defecto: 30
- **figsize:** (*Tupla de ints*) Tamaño de la figura que contendrá el mapa. Por defecto: (12,8)
- **path:** (*string*) Indica la ruta y el nombre que se usará para guardar el archivo (ej: '/mapas/mapa.png'). Por defecto: None

Retorno

- No retorna nada, dibuja el mapa generado.

```
1 from SpatialCluster.visualization.plotters import  
   plot_map  
2 plot_map(gdf, markersize=30, figsize=(12,8), path=None)
```

3.3.2. Graficar Muestreo aleatorio

El siguiente método genera un mapa interactivo con un muestreo aleatorio de los puntos, coloreados según el cluster asignado.

Parámetros

- **areas_to_points:** (*dict*) Diccionario que para cada cluster guarda los puntos que pertenecen a este.
- **min_supp:** (*int*) Cantidad mínima de puntos que debe tener un cluster para aparecer.
- **max_samples_per_clusters:** (*int*) Cantidad máxima de puntos que se mostrarán por cada clúster.
- **location:** (*Tupla de ints*) Longitud y latitud de dónde estará posicionado el centro del mapa. Por defecto: (-33.45, -70.65)
- **radius:** (*int*) Tamaño del círculo que representará a cada punto. Por defecto: 10
- **path:** (*string*) Indica la ruta y el nombre que se usará para guardar el archivo (ej: '/mapas/mapa.html'). Por defecto: None

Retorno

- **hmap:** (*Folium Map*) Mapa interactivo generado.

```
1 from SpatialCluster.visualization.plotters import
   plot_map_sample
2 hmap = plot_map_sample(areas_to_points, min_supp=min_supp,
   , max_samples_per_clusters=max_samples_per_clusters,
   location=(-33.45, -70.65), radius=10, path=None)
```

3.4. Evaluación de Clustering

A continuación se ofrecen dos mediciones para comparar clusterings. Ambas son usadas comúnmente de forma conjunta ya que aún cuando su finalidad es la misma, tienen diferente base teórica. El valor es 0 cuando las particiones comparadas son aleatorias e independientes, y es 1 cuando son idénticas.

3.4.1. Adjusted Rand Index (ARI)

Para comparar dos clusterings de mapas de atributos se puede usar la función ARI, la cual retorna un número real entre 0 y 1 que indica el grado de similaridad entre ambas soluciones.

Parámetros

- **clustering_1:** (*Numpy Array*) Contiene los clusters a los que pertenece cada punto para el clustering 1.
- **clustering_2:** (*Numpy Array*) Contiene los clusters a los que pertenece cada punto para el clustering 2.

Retorno

- **ARI:** (*float*) ARI score.

```
1 from SpatialCluster.metrics.ARI import ARI
2 ari = ARI(clustering_1, clustering_2)
```

En caso de que se quieran comparar más de un par de mapas a la vez, se puede generar una matriz que muestra la relación ARI para cada par de clusterings.

Parámetros

- **clusterings:** (*Pandas DataFrame*) Contiene los clusters a los que pertenece cada punto para los distintos clusterings a comparar.
- **plot:** (*bool*) Indica si se desea mostrar la matriz gráficamente.

Retorno

- **ARI_matr:** (*Pandas DataFrame*) Contiene el ARI score para cada par de clusterings.

```
1 from SpatialCluster.metrics.ARI import ARI_matrix
2 ari_matr = ARI_matrix(clusterings, plot=True)
```

3.4.2. Adjusted Mutual Information (AMI)

Para comparar dos clusterings de mapas de atributos se puede usar la función AMI, la cual retorna un número real entre 0 y 1 que indica el grado de similaridad entre ambas soluciones.

Parámetros

- **clustering_1:** (*Numpy Array*) Contiene los clusters a los que pertenece cada punto para el clustering 1.

- **clustering_2:** (*Numpy Array*) Contiene los clusters a los que pertenece cada punto para el clustering 2.

Retorno

- **AMI:** (*float*) AMI score.

```
1 from SpatialCluster.metrics.AMI import AMI
2 ami = AMI(clustering_1, clustering_2)
```

En caso de que se quieran comparar más de un par de mapas a la vez, se puede generar una matriz que muestra la relación AMI para cada par de clusterings.

Parámetros

- **clusterings:** (*Pandas DataFrame*) Contiene los clusters a los que pertenece cada punto para los distintos clusterings a comparar.
- **plot:** (*bool*) Indica si se desea mostrar la matriz gráficamente.

Retorno

- **AMI_matr:** (*Pandas DataFrame*) Contiene el AMI score para cada par de clusterings.

```
1 from SpatialCluster.metrics.AMI import AMI_matrix
2 ami_matr = AMI_matrix(clusterings, plot=True)
```

3.5. Dataset Región Metropolitana

Este módulo permite acceder a un conjunto de 40962 datos de distintos indicadores reunidos sobre la Región Metropolitana de Chile utilizados en [Vera et al., 2022]. A continuación se explica en más detalle a qué corresponden estos datos, cómo se procesaron y cómo se pueden utilizar.

3.5.1. Origen

El dataset originalmente corresponde a datos de unidades administrativas del censo de Santiago de Chile a nivel de manzana, las cuales contienen distintas características que fueron agrupadas en 3 categorías: Visuales, Sociales, Suelo.

3.5.2. Características Visuales

Estas características fueron obtenidas a partir de fotografías del lugar físico correspondiente, las cuales fueron procesadas por una red neuronal de las cuales se consideraron las siguientes 6 características.

- **beautiful:** (*float*) Indica qué tanta belleza se percibe en el lugar de la imagen.
- **boring:** (*float*) Indica qué tan monótono se ve el lugar de la imagen.
- **depressing:** (*float*) Indica qué tan triste se percibe el lugar de la imagen.
- **lively:** (*float*) Indica qué tan vivo o emocionante se ve el lugar de la imagen.
- **safe:** (*float*) Indica qué tan seguro se ve el lugar de la imagen.
- **wealth:** (*float*) Indica qué tan lujoso se ve el lugar de la imagen.

Se aplicó Principal Components Analysis (PCA) para reducir el número de características. Se escogió el número de dimensiones de PCA que capturarán al menos 80 % de la varianza, por lo que para las características visuales bastó con solo dos ('visual_0', 'visual_1').

3.5.3. Características Sociales

Estas características fueron obtenidas a partir del nivel socioeconómico del Censo 2012, la proporción de habitantes inmigrantes del Censo 2017, el plebiscito constitucional 2020 y la primera ronda de las elecciones presidenciales 2021 estimadas a nivel de manzana, y por último se obtuvo datos de la distribución de apellidos basados en el índice alfa de [\[Bro y Mendoza, 2021\]](#):

- **prom_nse:** (*float*) Nivel socioeconómico promedio.
- **edad_prom:** (*float*) Edad promedio.
- **porcentaje_inmigrantes:** (*float*) Proporción de habitantes inmigrantes.
- **prop_apruebo_promedio:** (*float*) Proporción de habitantes que votaron apruebo en el Plebiscito Constitucional 2020.
- **mapuche_prom:** (*float*) Proporción de habitantes promedio con apellido mapuche.
- **elite_prom:** (*float*) Proporción de habitantes promedio con apellido de clase alta.

Se aplicó Principal Components Analysis (PCA) para reducir el número de características. Se escogió el número de dimensiones de PCA que capturarán al menos 80 % de la varianza, por

lo que para las características sociales bastó con solo dos ('social_0', 'social_1').

3.5.4. Características de Suelo

Estas características corresponden a cómo el Estado clasifica las diferentes áreas de la ciudad con el propósito para la declaración de impuestos. De este se obtuvieron las siguientes características:

- **prop_uso_A:** (*float*) Proporción de uso de suelo destinado a Armamento.
- **prop_uso_C:** (*float*) Proporción de uso de suelo destinado a Comercio.
- **prop_uso_D:** (*float*) Proporción de uso de suelo destinado a Deporte.
- **prop_uso_E:** (*float*) Proporción de uso de suelo destinado a Educación.
- **prop_uso_F:** (*float*) Proporción de uso de suelo destinado a Forestal.
- **prop_uso_G:** (*float*) Proporción de uso de suelo destinado a Hotelería.
- **prop_uso_H:** (*float*) Proporción de uso de suelo destinado a Vivienda.
- **prop_uso_I:** (*float*) Proporción de uso de suelo destinado a Industria.
- **prop_uso_K:** (*float*) Proporción de uso de suelo destinado a No codificado.
- **prop_uso_L:** (*float*) Proporción de uso de suelo destinado a Almacenamiento.
- **prop_uso_M:** (*float*) Proporción de uso de suelo destinado a Minería.
- **prop_uso_O:** (*float*) Proporción de uso de suelo destinado a Negocio.
- **prop_uso_P:** (*float*) Proporción de uso de suelo destinado a Gobierno.
- **prop_uso_Q:** (*float*) Proporción de uso de suelo destinado a Culto.
- **prop_uso_S:** (*float*) Proporción de uso de suelo destinado a Salud.
- **prop_uso_T:** (*float*) Proporción de uso de suelo destinado a Transporte.
- **prop_uso_V:** (*float*) Proporción de uso de suelo destinado a Otro.
- **prop_uso_W:** (*float*) Proporción de uso de suelo destinado a Baldío.
- **prop_uso_Z:** (*float*) Proporción de uso de suelo destinado a Estacionamiento.
- **total_m2_manzana:** (*float*) Total de metros cuadrados que utiliza la manzana.

Se aplicó Principal Components Analysis (PCA) para reducir el número de características. Se escogió el número de dimensiones de PCA que capturarán al menos 80 % de la varianza, por lo que para las características de suelo se necesitaron cuatro ('suelo_0', 'suelo_1', 'suelo_2', 'suelo_3').

3.5.5. Uso

Para tener acceso al dataset basta con usar la siguiente función:

```
1 from SpatialCluster.datasets import load_manzana_data
2 df = load_manzana_data()
```

Luego dependiendo del método que se quiera usar, se puede dar el formato correspondiente con las siguientes funciones:

Si se desea usar el método *KNN*, se debe utilizar esta función.

```
1 from SpatialCluster.preprocess.data_format import
   attributes_with_zone_format
2 features_position, features_X =
   attributes_with_zone_format(df)
```

Si se desea usar cualquier otro método que ofrece *SpatialCluster*, se debe utilizar esta función.

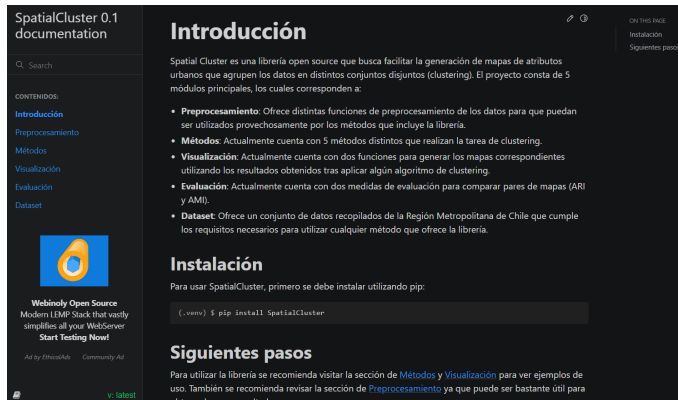
```
1 from SpatialCluster.preprocess.data_format import
   attributes_format
2 features_position, features_X = attributes_format(df)
```

3.6. Documentación

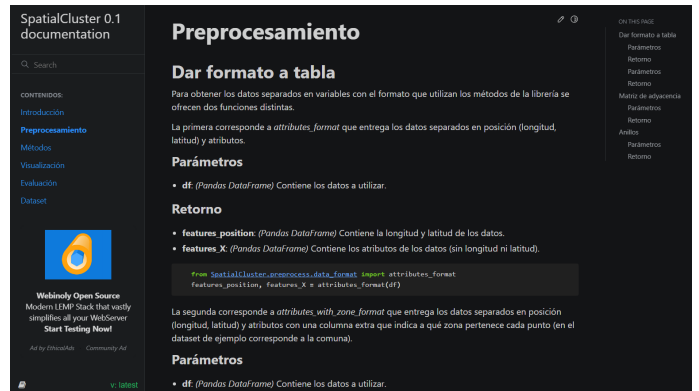
Finalmente, se presenta la documentación de *SpatialCluster*. Esta se encuentra constituida por una breve introducción e instrucciones de instalación, seguido de la explicación de cómo utilizar los módulos que componen la librería.

La documentación completa en español se puede encontrar aquí: [SpatialCluster Docs](#).

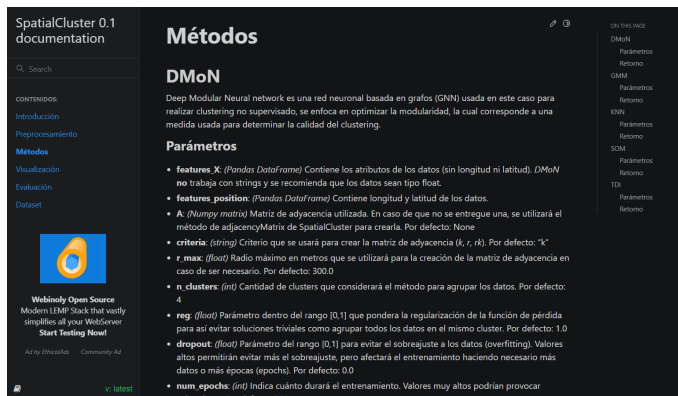
La versión en inglés se puede encontrar aquí: [SpatialCluster English Docs](#).



(a) Introducción



(b) Preprocesamiento



(c) Métodos de clustering



(d) Métodos de visualización



(e) Métricas de evaluación de clustering



(f) Uso del dataset

Figura 3: Documentación de SpatialCluster

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

Para la validación, se desarrolló un caso de estudio con datos de la Región Metropolitana de Chile. En este, se medirá el uso de recursos para los 5 métodos de clustering que ofrece SpatialCluster, luego se generará un mapa por cada método y finalmente se compararán utilizando las matrices de ARI y AMI.

4.1. Uso de recursos

Para analizar el uso de recursos de los algoritmos, se realizaron una serie de experimentos en los que se utilizaron los 5 métodos ofrecidos por la librería. Se monitoreó el tiempo de ejecución, uso de RAM, y uso de CPU. Para esto se utilizaron 5 distintos tamaños de muestra del dataset disponible en la librería (10, 25, 50, 75 y 100 % de los datos).

Cada uno de estos experimentos se repitieron 10 veces y se calculó el promedio de cada medición realizada. El procesador utilizado corresponde a un Intel Core i5-12600KF. Los resultados se muestran en los gráficos de las figuras 4, 5 y 6.

En el gráfico de la figura 4, se puede apreciar que TDI es el método, con gran diferencia, que más tiempo demora en ejecutarse, seguido por DMoN. KNN, GMM y SOM son los métodos que menos tiempo demoran en ejecutarse.

En el gráfico de la figura 5, se observa que DMoN es el método que considerablemente utiliza más RAM, seguido por TDI. SOM, KNN y GMM son los métodos que menos RAM utilizan.

Finalmente, en el gráfico de la figura 6, se muestra que GMM es el método que más utiliza CPU. Luego le siguen DMoN y TDI con un uso de CPU bastante menor, y por último están SOM y KNN con un uso de CPU mínimo.

También cabe destacar que para los 3 indicadores utilizados, los métodos tienden a aumentar el uso de estos recursos a medida que el tamaño de la muestra del dataset aumenta, a excepción de TDI en el uso de RAM, en el cual se puede apreciar un comportamiento más errático.

4.2. Generación de mapas

En la figura 7 se muestran 5 mapas, cada uno generado por uno de los métodos de SpatialCluster. En estos se puede apreciar tanto las diferencias como las similitudes que pueden presentar los resultados de cada algoritmo. Por ejemplo, para los datos que son agrupados

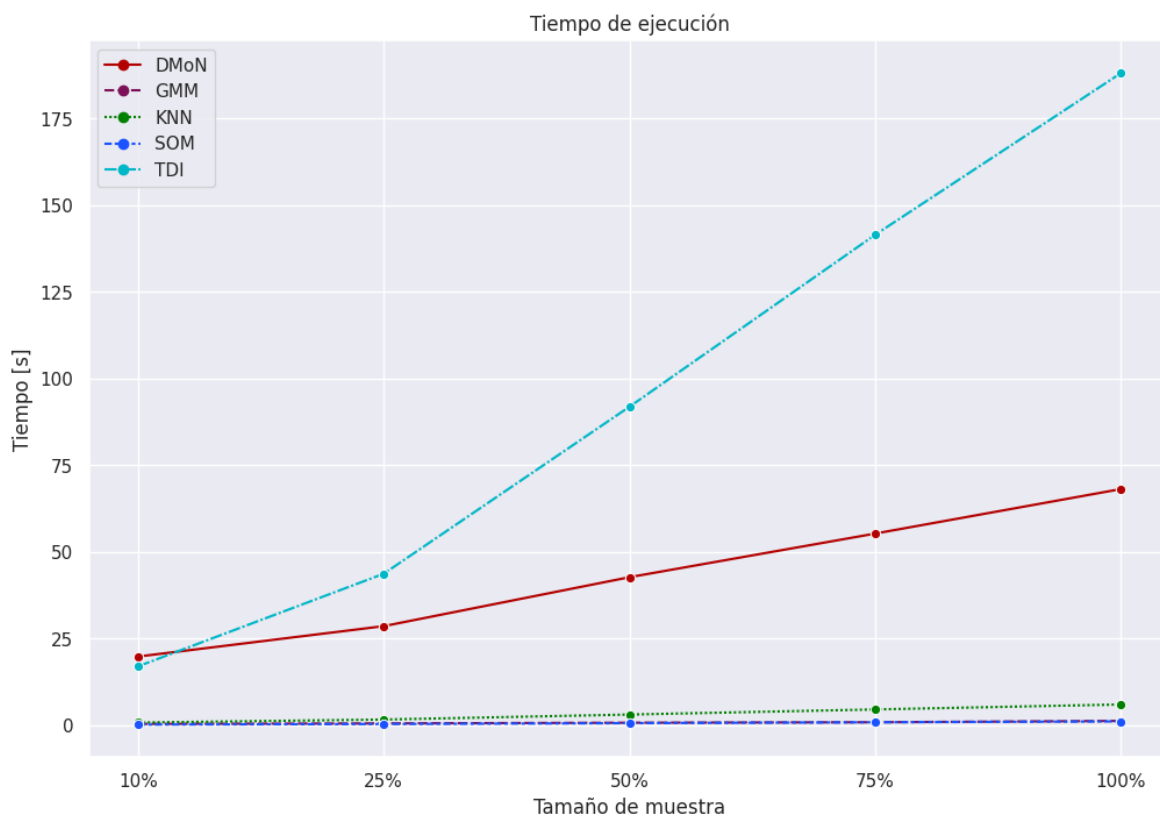


Figura 4: Tiempo de ejecución.

por el color rosado en el mapa de GMM, todos los demás algoritmos decidieron considerarlos en el mismo cluster.

4.3. Herramientas de comparación de mapas

En las figuras 8 y 9 se muestran dos matrices que comparan estos 5 mapas entre sí, la primera corresponde a la medición de ARI y la segunda corresponde a la medición de AMI.

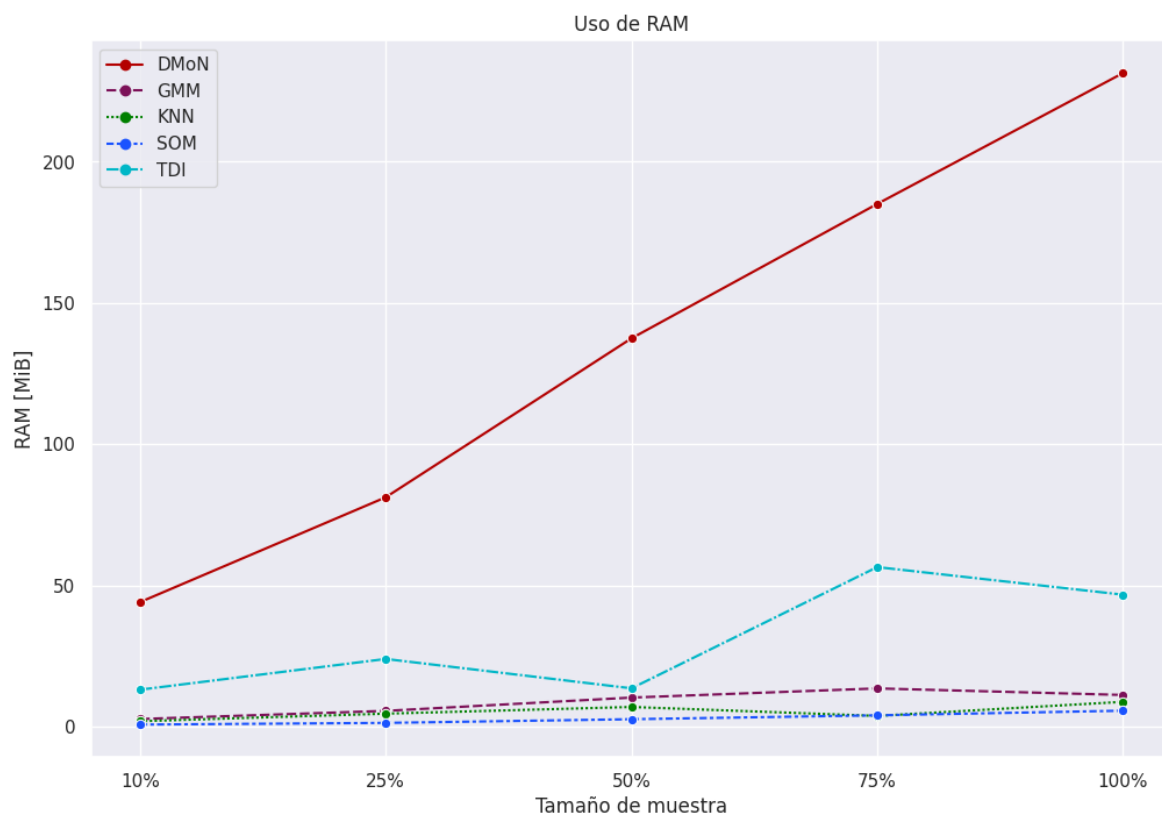


Figura 5: Uso de Ram.

CAPÍTULO 5

CONCLUSIONES

La tarea de clustering urbano corresponde a un desafío y área de estudio que tiene mucho por recorrer. La falta de respuestas correctas con las que comparar y la incerteza de qué método o qué datos son los óptimos para poder cumplir con el objetivo son problemas que se deben abordar. Sin embargo, se pueden amenizar con la disponibilidad de herramientas y datos para el estudio del área, tal como ofrece SpatialCluster.

En cuanto a los resultados del uso de recursos de los métodos, KNN y SOM corresponden a los métodos que menos recursos utilizan al momento de ejecutarse.

Los métodos TDI y DMoN resultaron ser los métodos que más tiempo de ejecución requieren. Esto se puede deber a que ambos métodos necesitan una matriz de adyacencia (la cual es creada en caso de no existir por defecto). Esta se crea a partir de los datos para representar un grafo con la topología correspondiente. Además, lo que más tiempo demora para el método de TDI es aplicar *Spectral Clustering* sobre una matriz con una cantidad considerable de datos. Esto se ve reflejado en el caso en que se usa solo el 10 % de los datos, DMoN y TDI

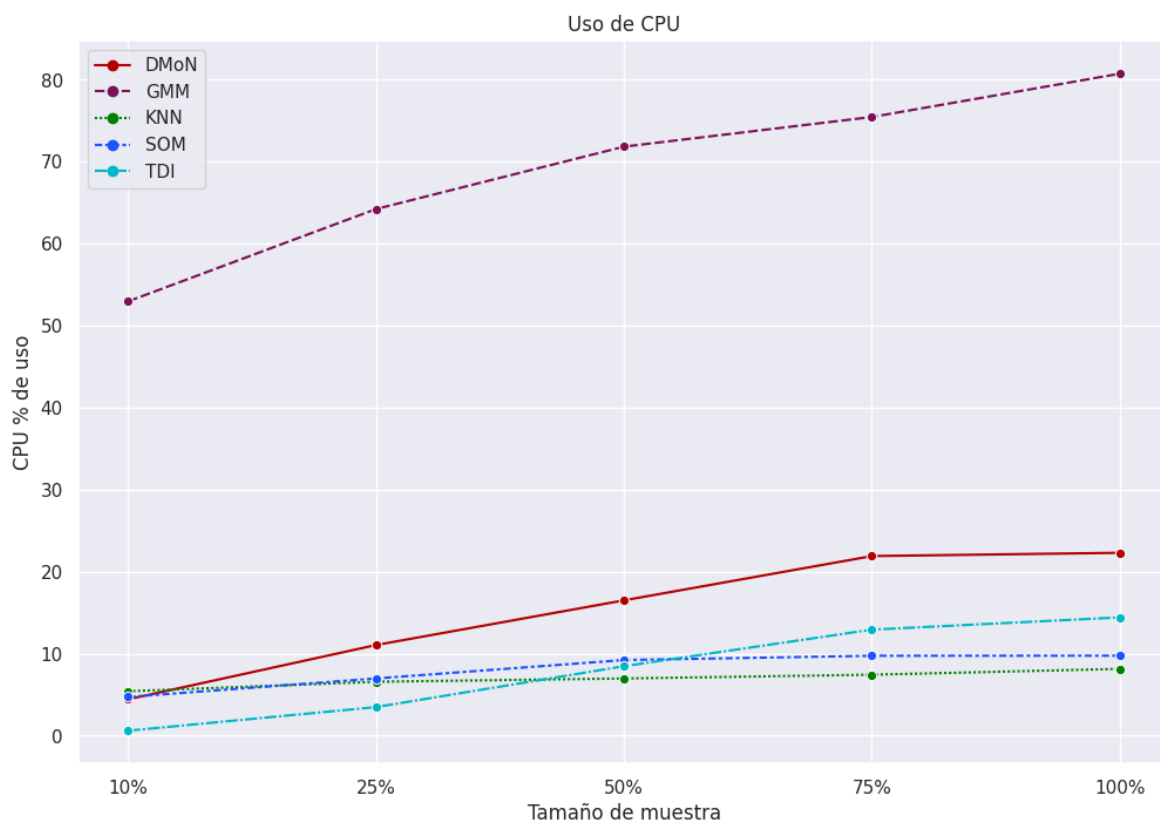


Figura 6: Uso de CPU.

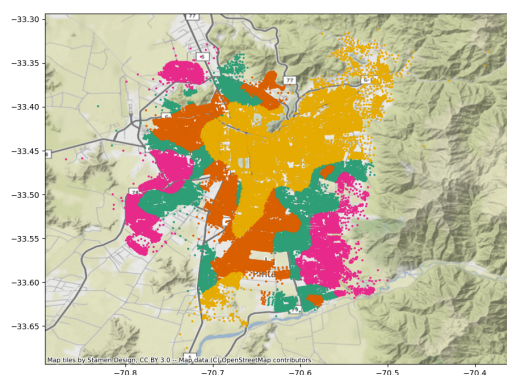
utilizan prácticamente la misma cantidad de tiempo para ejecutarse, pero a medida que la cantidad de datos aumenta, el tiempo de ejecución de TDI se dispara.

En cuanto al uso de RAM, DMoN es el método que más espacio utiliza. Esto se debe a que corresponde a una red neuronal basada en grafos, la cual requiere de guardar los pesos y otros parámetros que componen el modelo correspondiente. También, el uso de RAM se justifica debido a la necesidad de almacenar la matriz de adyacencia a utilizar, al igual que TDI.

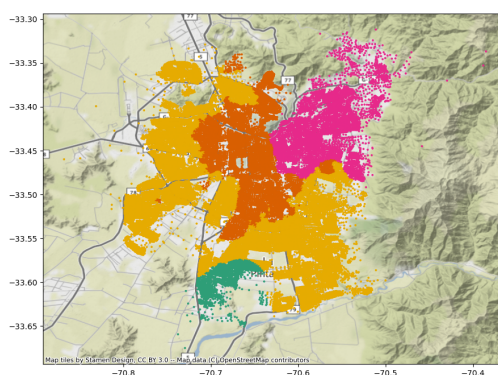
Para el uso de CPU, GMM resulta ser el método que más unidad de cómputo ocupa al momento de ejecutarse.

En cuanto a los mapas generados, si se observa la matriz de la figura 8, los valores de ARI para cada par de clusterings no alcanza un valor considerable para concluir que sean similares. El valor más alto corresponde a un ARI de 0.40, entre los mapas generados por GMM y SOM.

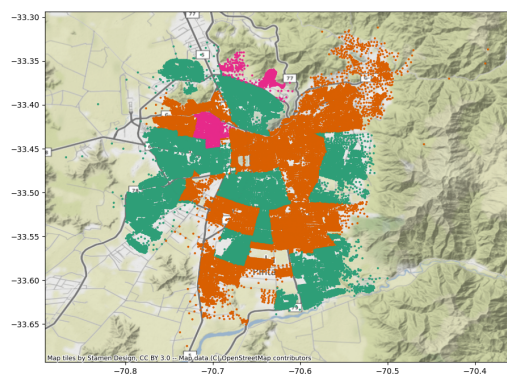
Un escenario similar se observa en la matriz de la figura 9, los valores de AMI para cada par de clusterings no alcanza un valor considerable para concluir que sean similares. El valor más alto corresponde a un ARI de 0.36, entre los mapas generados por DMoN y GMM.



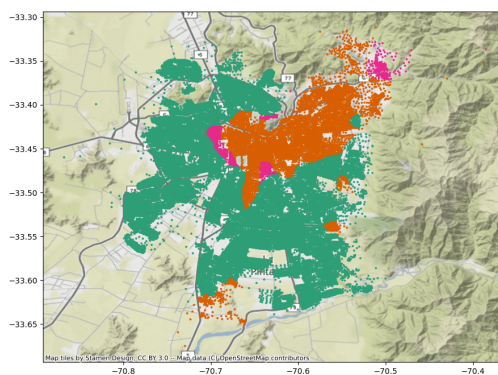
(a) DMon



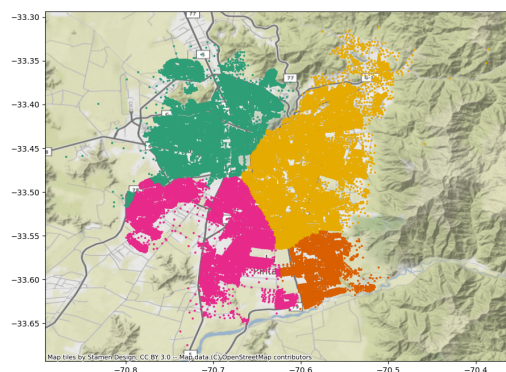
(b) GMM



(c) KNN



(d) SOM



(e) TDI

Figura 7: Mapas generados por los métodos con SpatialCluster

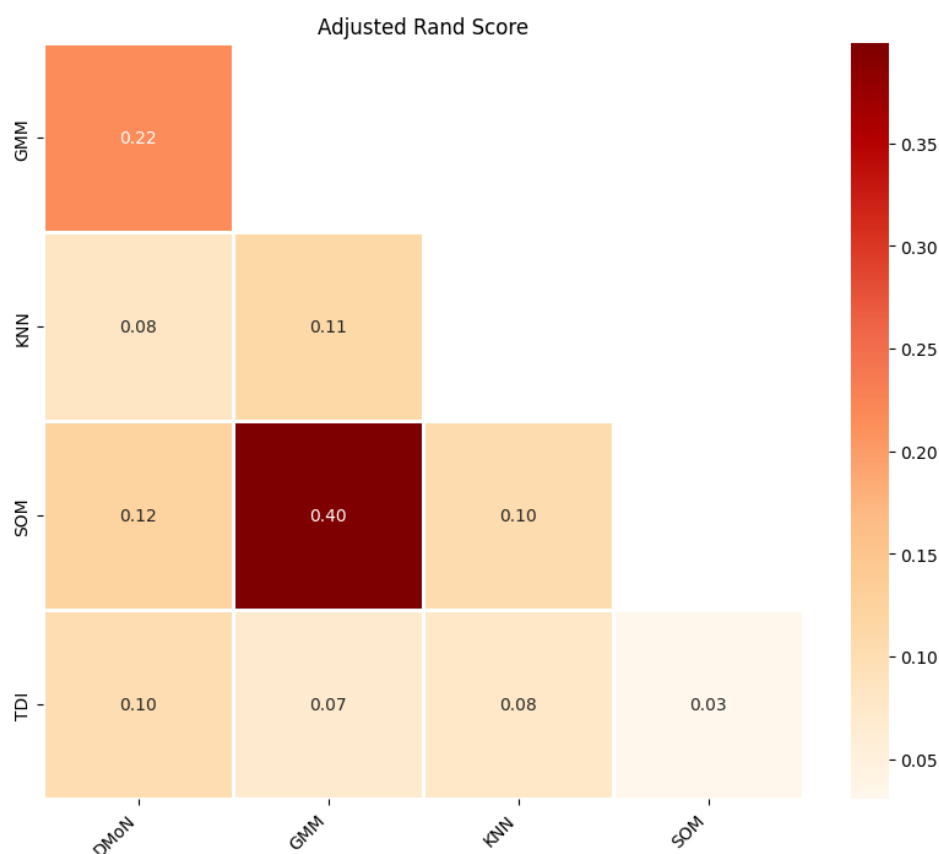


Figura 8: Matriz de los puntajes de ARI para cada par de mapas.

Se puede observar que aunque los 5 métodos tienen la misma tarea, generan resultados bastante diferentes, respaldando así la utilidad de tener estos algoritmos disponibles en la misma librería para poder experimentar y tomar una mejor decisión respecto a los resultados de los clusterings. También cabe destacar la observación de que los 5 mapas encierran la zona conformada por barrios de clase alta en el mismo cluster (sector nororiente que se puede ver coloreado de rosado en el mapa de GMM), lo cual remarca la segregación existente, dado que aún con distintos métodos se logra distinguir ese sector. Además, valida el trabajo de clustering urbano que realizan los algoritmos al poder discernir ese fenómeno.

Se logró demostrar el correcto funcionamiento de la generación automática de mapas urbanos, a través de 5 distintos algoritmos, y también se mostró el uso de las herramientas de validación y comparación que ofrece SpatialCluster a través del caso de estudio con los datos de la Región Metropolitana de Chile.

Para trabajo futuro, sería ideal agregar más algoritmos de clustering que se puedan utilizar en el área de urbanismo, de esta manera los usuarios tienen más herramientas a la hora de explorar sus datos.

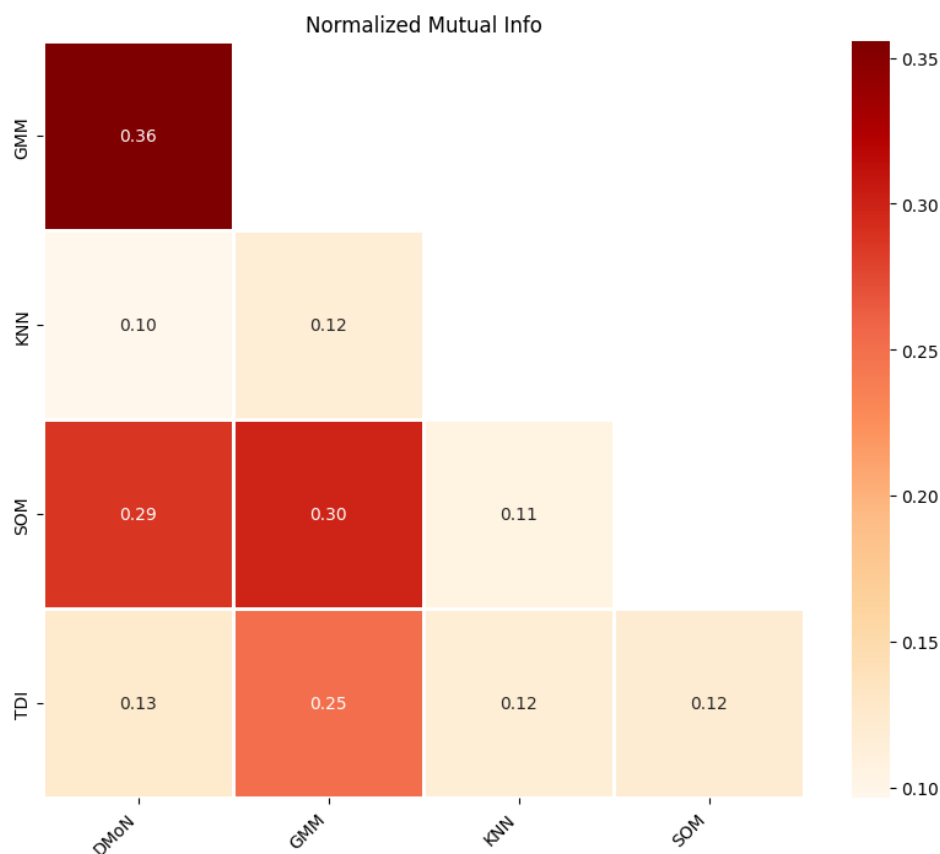


Figura 9: Matriz de los puntajes de AMI para cada par de mapas.

REFERENCIAS BIBLIOGRÁFICAS

- [Almi'ani *et al.*, 2018] Almi'ani, M., Ghazleh, A. A., Al-Rahayfeh, A., y Razaque, A. (2018). Intelligent intrusion detection system using clustered self organized map. En *2018 Fifth international conference on software defined systems (SDS)*, pp. 138–144. IEEE.
- [Barrai *et al.*, 2012] Barrai, I., Rodriguez-Larralde, A., Dipierri, J., Alfaro, E., Acevedo, N., Mamolini, E., Sandri, M., Carrieri, A., y Scapoli, C. (2012). Surnames in chile: a study of the population of chile through isonymy. *American journal of physical anthropology*, 147(3):380–388.
- [Bibri, 2019] Bibri, S. E. (2019). The anatomy of the data-driven smart sustainable city: instrumentation, datafication, computerization and related applications. *Journal of Big Data*, 6(1):1–43.
- [Bro y Mendoza, 2021] Bro, N. y Mendoza, M. (2021). Surname affinity in santiago, chile: A network-based approach that uncovers urban segregation. *PloS one*, 16(1):e0244372.

- [Chodrow, 2017] Chodrow, P. S. (2017). Structure and information in spatial segregation. *Proceedings of the National Academy of Sciences*, 114(44):11591–11596.
- [Engin et al., 2020] Engin, Z., van Dijk, J., Lan, T., Longley, P. A., Treleaven, P., Batty, M., y Penn, A. (2020). Data-driven urban management: Mapping the landscape. *Journal of Urban Management*, 9(2):140–150.
- [Font-Casaseca, 2021] Font-Casaseca, N. (2021). Los mapas y la transición digital:¿ una oportunidad para la geografía urbana crítica? *Scripta Nova. Revista Electrónica de Geografía y Ciencias Sociales*, 25(3).
- [Fowler et al., 2016] Fowler, C. S., Lee, B. A., y Matthews, S. A. (2016). The contributions of places to metropolitan ethnoracial diversity and segregation: Decomposing change across space and time. *Demography*, 53(6):1955–1977.
- [Hennerdal y Nielsen, 2017] Hennerdal, P. y Nielsen, M. M. (2017). A multiscalar approach for identifying clusters and segregation patterns that avoids the modifiable areal unit problem. *Annals of the American Association of Geographers*, 107(3):555–574.
- [Kitchin et al., 2015] Kitchin, R., Lauriault, T. P., y McArdle, G. (2015). Knowing and governing cities through urban indicators, city benchmarking and real-time dashboards. *Regional Studies, Regional Science*, 2(1):6–28.
- [Mendoza y Bro, 2021] Mendoza, M. y Bro, N. (2021). Predicting affinity ties in a surname network. *Plos one*, 16(9):e0256603.
- [Olteanu et al., 2020] Olteanu, M., Hazan, A., Cottrell, M., y Randon-Furling, J. (2020). Multidimensional urban segregation: toward a neural network measure. *Neural Computing and Applications*, 32(24):18179–18191.
- [Ornés, 2009] Ornés, S. (2009). El urbanismo, la planificación urbana y el ordenamiento territorial desde la perspectiva del derecho urbanístico venezolano. *Politeia*, 32(42):197–225.
- [Reynolds, 2009] Reynolds, Douglas, e. a. (2009). Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663).
- [Roberto, 2018] Roberto, E. (2018). The spatial proximity and connectivity method for measuring and analyzing residential segregation. *Sociological Methodology*, 48(1):182–224.
- [Romano et al., 2016] Romano, S., Vinh, N. X., Bailey, J., y Verspoor, K. (2016). Adjusting for chance clustering comparison measures. *The Journal of Machine Learning Research*, 17(1):4635–4666.
- [Sánchez Undurraga, 2021] Sánchez Undurraga, R. (2021). Spatial self-organization in santiago: methods and applications.
- [Tsitsulin et al., 2020] Tsitsulin, A., Palowitch, J., Perozzi, B., y Müller, E. (2020). Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*.

[Vera *et al.*, 2022] Vera, Camila and Lucchini, Francesca and Bro, Naim and Mendoza, Marcelo and Löbel, Hans and Gutiérrez, Felipe and Dimter, Jan and Cuchacovic, Gabriel and Reyes, Axel and Valdivieso, Hernán and others (2022). Learning to cluster urban areas: two competitive approaches and an empirical validation. *EPJ Data Science*, 11(1):62.