Universitat Oberta de Catalunya

uoc.edu

Algoritmo de aprendizaje incremental para agrupar y clasificar running trails.

Carlos Alfonso Cuervo Rodríguez
Máster universitario en ciencia de datos
Área 1

Anna Bosch Rué
Albert Solé Ribalta

junio de 2020

Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-SinObraDerivada
3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Algoritmo de aprendizaje incremental para agrupar y clasificar running trails*
Nombre del autor:	*Carlos Alfonso Cuervo Rodríguez*
Nombre del consultor/a:	*Anna Bosch Rué*
Nombre del PRA:	*Albert Solé Ribalta*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Máster universitario en ciencia de datos*
Área del Trabajo Final:	*Área 1*
Idioma del trabajo:	*Castellano*
Palabras clave	*Clustering, k-means, incremental learning, running trails*

Resumen del Trabajo:

Es habitual entre los aficionados al running, buscar rutas en alguna zona de interés. Hay herramientas que permiten hacer este tipo de búsquedas, filtrando por distancia, locación, etc. No obstante, una vez aplicados todos los filtros aparecen rutas duplicados o muy similares, creando así mucha entropía y dificultando el proceso de búsqueda.

En este trabajo se propone hacer un agrupamiento de rutas para identificar aquellas que son únicas en una zona determinada. Se propone usar un algoritmo basado en **k-means incremental** que agrupe las rutas disponibles, y posteriormente en un momento indeterminado en el tiempo, dada una nueva ruta ingresada al sistema, decide si esta debe ser asignada a un cluster ya creado debido a su similitud con las rutas que lo conformen, o se debe crear un nuevo cluster ya la ruta es lo suficientemente distante a las demás.

En este trabajo se estudiará la mejor manera de representar las variables de una ruta, considerando feature engineering y técnicas de reducción de dimensionalidad que facilite y mejore los resultados del algoritmo de agrupamiento. Así mismo, se estudiarán los efectos sobre el rendimiento y las medidas de error intra e Inter grupos al considerar diferentes valores de threshold.

Al final se espera entregar un aplicativo que permita visualizar las rutas únicas, filtrarlas, introducir nuevas rutas y ver información relacionada como: duración, elevación, distancia, velocidad promedio, entre otras.

Abstract:

It is common among running fans look for trails in some location of interest. There are tools that allow runners to do this kind of searches, filtering by distance, location, etc. However, once all the filters are applied, the result is duplicated trails or very similar ones, creating a lot of entropy and hindering the process.

In this work it is proposed to make a grouping of trails to identify those that are unique in a given area. It is proposed to use an algorithm based on incremental k-means that clusters the available running trails, and later at some moment in time, given a new trail entered into the system, decides whether it should be assigned to a cluster already created due to its similarity with the trails that make it up, or a new cluster must be created since the trail is distant enough from the others.

In this work it is studied the best way to represent the variables of a running trail, considering feature engineering and dimensionality reduction techniques that facilitate and improve the results of the clustering algorithm. Likewise, the effects on performance and intra and intergroup error measures will be studied when considering different threshold values.

In the end, it is expected to deploy an application that allows viewing the unique routes, filtering them, introducing new routes and seeing related information such as: duration, elevation, distance, average speed, among others.

Índice

1. Introducción ... 1
   1.1 Descripción y justificación del trabajo ... 1
   1.2 Motivación ... 2
2. Objetivos del Trabajo ... 2
   2.1 Objetivo general ... 2
   2.2 Objetivos específicos ... 2
   2.3 Preguntas de investigación ... 3
3. Metodología ... 3
   3.1 Metodología general ... 3
   3.2 Metodología específica ... 4
      3.2.1 Lenguaje y ambiente de programación ... 4
      3.2.2 Algoritmos ... 4
4. Planificación del trabajo ... 5
   4.1 Tareas y cargas de trabajo ... 5
   4.2 Diagrama de Gantt de tareas a realizar ... 9
5. Marco teórico y trabajos relacionados ... 10
   5.1 Explotación de datos GPS y rutas ... 10
   5.2 Algoritmos de Agrupamiento ... 11
      5.2.1 Agrupamiento Jerárquico ... 12
      5.2.2 Agrupamiento Particional ... 13
         5.2.2.1 Algoritmo k-means ... 14
         5.2.2.2 Algoritmo de k-medians ... 15
         5.2.2.3 Algoritmo de k-medoids ... 15
         5.2.2.4 Fuzzy C-means ... 15
         5.2.2.5 Métricas de validación y k óptimo ... 16
         5.2.2.6 Preclustering Canopy ... 18
         5.2.2.7 ISODATA ... 19
      5.2.3 Agrupamiento por densidad y cuadrícula ... 20
   5.3 Incremental learning ... 22
   5.4 Incremental clustering ... 24
   5.5 Reducción de dimensionalidad ... 27
   5.6 Estado del arte de herramientas usadas ... 28
      5.6.1 Buscadores de rutas ... 29
      5.6.2 Visualizadores de archivos GPX ... 31
      5.6.3 Librerías para ML con incremental learning ... 32
      5.6.4 Archivos GPX en Python ... 33
6. Desarrollo del algoritmo y experimentación ... 34
   6.1 Selección de variables y PCA ... 37
   6.2 Indicadores de rendimiento de clustering ... 43
   6.3 Selección de Threshold “óptimo” ... 46
   6.4 Usando métodos clásico y K-means ... 49
   6.5 Interfaz gráfica y producto final ... 50
7. Conclusiones ... 53
8. Trabajos futuros ... 54
9. Glosario ... 55
10. Referencias bibliográficas ... 56

Lista de figuras

Figura 1: Ejemplo de incremental learning. Fuente:[11]..... 1
Figura 2: Metodología CRISP-dm. Fuente:[31]..... 4
Figura 3: Diagrama Gantt en Trello. Fuente: Autor..... 9
Figura 4: Diagrama kanban en Trello. Fuente: Autor..... 9
Figura 5: Métodos de agrupamiento. Fuente: [4]..... 11
Figura 6: Pseudocódigo algoritmos AHC y DHC. Fuente: [4]..... 13
Figura 7: Pseudocódigo de k-means. Fuente: [4]..... 14
Figura 8: Resultado de partición suave restringida. Fuente: [4]..... 16
Figura 9: Ejemplo método silhouette para k=3. Fuente: Scikit-learn..... 18
Figura 10: Pseudocódigo del algoritmo Canopy. Fuente: [4]..... 19
Figura 11: Parámetros de DBSCAN. Fuente: [24]..... 21
Figura 12: Métodos de aprendizaje incremental. Fuente: [8]..... 23
Figura 13: Pseudocódigo de k-means incremental. Fuente: Autor basado en [12]..... 25
Figura 14: Ejemplo de PCA. Fuente: PCA in Python/v3..... 28
Figura 15: Ejemplo de deportes con rutas: Fuente: Wikiloc..... 29
Figura 16: Ejemplo buscador rutas de running. Fuente: Wikiloc..... 29
Figura 17: Opciones de búsqueda y creación de rutas. Fuente: Mapmyfitness..... 30
Figura 18: Ejemplo de visualización de ruta. Fuente: ridewithgps..... 30
Figura 19: Visualización de archivo GPX con googlemaps. Fuente Autor..... 31
Figura 20: Visualización de archivo GPX con gpxviewer. Fuente: Autor..... 32
Figura 21: Ejemplo de visualización de rutas con Python. Fuente: [28]..... 33
Figura 22: Modelo conceptual del software a desarrollar. Fuente: Autor..... 34
Figura 23: Lista de ficheros software..... 34
Figura 24: Gráfica de ruta y su centroide. Fuente: Autor..... 36
Figura 25: Componentes principales y valor de tuplas de centroides..... 40
Figura 26: Relación entre variables..... 41
Figura 27: Comparación de mapas de rutas..... 42
Figura 28: Clusters sintéticos generados..... 43
Figura 29: Cantidad de clusters versus Thr. Fuente: Autor..... 44
Figura 30: Indicador C&H y Hartigan versus el Thr..... 44
Figura 31: Indicares de C&H y Hartigan Vs. número de clústeres K..... 45
Figura 32: Indicares de C&H y Hartigan Vs. número de clústeres K (cerca)..... 45
Figura 33: Threshold versus K cluster en la agrupación de rutas..... 46
Figura 34: Indicar C&H versus K clusters en la agrupación de rutas..... 47
Figura 35: Indicar C&H versus K clusters en la agrupación de rutas(iteración 2)..... 47
Figura 36: Comparación de rutas de un mismo cluster con Thr=0.15..... 48
Figura 37: Método del codo y silhouette en kmeans..... 49
Figura 38: Agrupamiento con K=2 y K=3 en kmeans..... 49
Figura 39: Pantalla "Home" del aplicativo..... 50
Figura 40: Mensaje de notificación sobre bases vacías..... 50
Figura 41: Mensaje de error, no hay archivos .gpx..... 51
Figura 42: Pantalla 2. Buscador de rutas del aplicativo..... 51
Figura 43: Ejemplo de mapa de ruta en el navegador con Folium..... 52
Figura 44: Pantalla 3 del aplicativo. Creador de rutas..... 52

Lista de fragmentos de Código

Fragmento de Código 1: Implementación Incremental k-means. 37
Fragmento de Código 2: Algoritmo para tratar valores NaN. 38
Fragmento de Código 3: Cambio de coordenadas 3D a 2D. 39
Fragmento de Código 4: Estandarización de variables y PCA en Python. 39
Fragmento de Código 5: Segunda aproximación de PCA con variables elegidas..... 41
Fragmento de Código 6: Creación de clusters con data sintética. 43

1. Introducción

1.1 Descripción y justificación del trabajo

En los últimos años las personas han empezado a practicar diferentes deportes como alternativa para mantenerse en forma, no solo indoor como puede ser el crossfit, sino outdoor como lo son el running o el cycling. Es habitual ver que se promocionan maratones o medias maratones para apoyar alguna causa, o simplemente para competir a diferentes niveles.

Las personas en el caso del running, siempre están buscando nuevas rutas cerca a sus hogares o trabajo, para realizar entrenamiento que los prepare para las diferentes competencias. Sin embargo, no es fácil encontrarlas, así que para ello utilizan herramientas de búsqueda de rutas gratuitas, que permiten buscarlas filtrando por distancia, tipo de deporte, velocidad, locación etc. No obstante, una vez aplicados todos los filtros se obtienen rutas duplicados o muy similares, creando así mucha entropía y dificultando el proceso de búsqueda.

Con este trabajo se pretende atacar este problema, por medio de la aplicación de un algoritmo de clustering, que permita identificar las rutas que son similares entre ellas, crear un clúster y sólo mostrarle al usuario las rutas que son verdaderamente diferentes. Sin embargo, esto no es posible realizarse por medio de los métodos tradicionales de agrupamiento por dos razones: la primera es que no es posible saber a priori cuantos clusters se van a realizar, porque no se sabe cuántas rutas podrían existir en una zona determinada, o cuantas nuevas pudieran crearse a futuro, y la segunda (muy de la mano con la primera), es que es un proceso incremental, es decir, no se tienen todas las posibles rutas dispuestas en el tiempo t=0, porque a medida que un usuario vaya creando rutas, estas serán agrupadas, pero esto puede ocurrir en cualquier momento indeterminado en el tiempo, así que se debe utilizar un modelo que permita realizar diferentes procesos de agrupamiento, que permita conservar la información de los clusters ya creados y cuando aparezca una nueva ruta, decida si se asigna a un clúster existente o si se debe crear otro completamente nuevo. Para esto se hará uso del método de aprendizaje incremental, más específicamente del algoritmo **incremental K-means**.

En la siguiente figura se expone un ejemplo sencillo del resultado esperado de un algoritmo de aprendizaje incremental, aplicado a la clasificación de imágenes[11]:

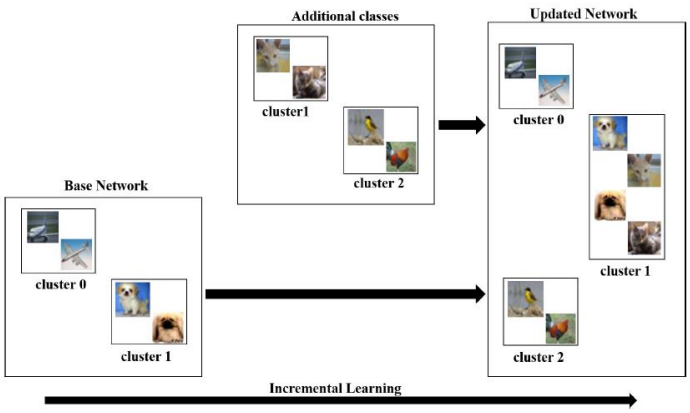

Figura 1: Ejemplo de incremental learning. Fuente:[11]

En la figura 1 se observa como partiendo de dos clusters, uno de perros y otro de aviones, se le pide al algoritmo que en un segundo paso agrupe nuevas imágenes, que son de gatos y gallos. El algoritmo decide que hay similitud entre perros y gatos, seguramente por ser mamíferos y mascotas, pero el gallo no es un mamífero y tampoco es un medio de transporte como el avión, así que se crea un nuevo clúster de gallos.

En este trabajo se estudiará la mejor manera de representar las variables de una ruta, considerando feature engineering y técnicas de reducción de dimensionalidad que facilite y mejore los resultados del algoritmo de agrupamiento. Así mismo, se estudiarán los efectos sobre el rendimiento y las medidas de error intra e Inter grupos al considerar diferentes valores de threshold.

Al final se espera entregar un aplicativo que permita visualizar las rutas únicas, filtrarlas, introducir nuevas rutas, graficar mapas y ver información relacionada como: duración, elevación, distancia, velocidad, entre otros.

1.2 Motivación

La ciencia de datos se piensa que sólo está para solucionar problemas en ámbitos de la ciencia o los negocios, pero la verdad es, que la ciencia de datos tiene aplicaciones en cualquier ámbito de la vida humana. Es por esto por lo que con este trabajo se quiere hacer una aproximación del uso de la ciencia de datos, para mejorar la experiencia de practicantes de deportes de exterior, pero a su vez avanzar un poco más en el estudio de modelos de machine learning no supervisados y de aprendizaje incremental que permita resolver muchos más problemas de la vida cotidiana.

2. Objetivos del Trabajo

En este apartado se detallan los objetivos que guían este trabajo de investigación. Desde el punto de vista más general, pasando por los pasos a seguir para alcanzar ese objetivo y finalmente planteando la hipótesis que con este trabajo se desea comprobar.

2.1 Objetivo general

Desarrollar un algoritmo de aprendizaje incremental que sea capaz de identificar de forma automática nuevas rutas cuándo estas son introducidas en el sistema y a la vez identificar todos los trails que son iguales o similares para que estos sean mostrados al usuario de forma única.

2.2 Objetivos específicos

- Encontrar la mejor manera de representar variables de los tracks para ser usados por el algoritmo de agrupamiento.
- Comparar las metodologías de búsqueda de threshold adecuado, y encontrar el mejor threshold para el caso específico.
- Realizar análisis de sensibilidad sobre el algoritmo desarrollado para comprobar su robustez.

- Desarrollar un aplicativo que permita: visualizar las rutas y su información completa, filtrar rutas, introducir rutas nuevas.

2.3 Preguntas de investigación

Establecidos los objetivos generales y específicos, se pretende resolver las siguientes preguntas para alcanzarlos:

- ¿Qué transformación o configuración de variables de las rutas se requieren para que un buen funcionamiento del algoritmo de agrupamiento?
- ¿Es posible encontrar un threshold "optimo" que maximice el resultado del algoritmo?
- ¿Cuál estrategia de selección de threshold resulta ser más apropiada para elegir dicho parámetro?

3. Metodología

En este apartado se detallan las metodologías a utilizar, desde un punto de vista general, hasta el más específico donde se tocan temas como: algoritmos, software, ambientes de desarrollo y herramientas.

3.1 Metodología general

Para la metodología general se piensa hacer uso de la metodología CRISP [32], considerando que este es un proyecto en el fondo de desarrollo de software, que permite cumplir los requerimientos de un cliente y necesidades de unos usuarios que en este caso son los usuarios de deportes de exterior.

Entendimiento del problema: La primera parte es la de entender a profundidad el problema a resolver, por lo que se requiere documentarse sobre diferentes temas como: manejo de archivos .gpx que son como están almacenados los datos GPS de las rutas, técnicas de agrupamiento, medidas de rendimiento de agrupamiento, ejemplos de algoritmos de aprendizaje incremental, software disponible para resolver problemas similares y herramientas de visualización.

Entendimiento de la data y preparación: como se mencionaba es una data en formato .gpx que debe ser explotada de cierta manera ya que es un formato semiestructurado. Así mismo se requiere entender las variables que se tienen de cada una de las rutas y estas como pueden ser representadas para alimentar el modelo.

Modelado: En esta fase se hará uso de diferentes prototipos que se irán desarrollando de manera incremental para poder corregir desviaciones tempranas y perder la menor cantidad de tiempo posible. Se evaluarán diferentes técnicas de programación y configuración de parámetros para saber cuál tiene mejor rendimiento. En este punto se empieza un proceso iterativo con el procesamiento de la data, ya que el tipo de variables y cantidad deben ir en sintonía con el modelo.

Evaluación: Se probarán diferentes indicadores de rendimiento para medir la eficiencia del modelo, entre ellos: tiempo de procesamiento, error entre los grupos, error dentro de los grupos, entre otros. Dependiendo de los resultados que se obtengan de las primeras evaluaciones se decidirá si se inicia o no una nueva iteración desde el principio, es decir desde el entendimiento del problema, pasando por el entendimiento y procesamiento de la data, hasta ajustes en el modelo. Este es precisamente el poder que tiene esta metodología de no esperar hasta que se tenga todo el producto terminado para hacer ajustes de manera rápida y ágil. Este ciclo es posible observarse en la figura 2.

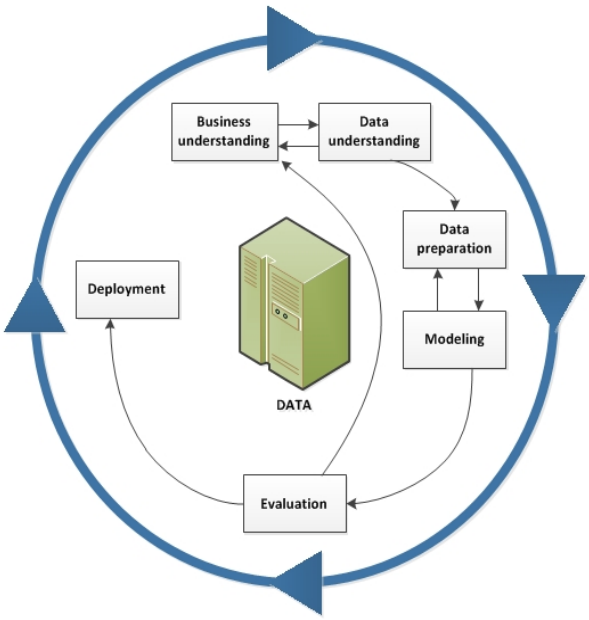

Figura 2: Metodología CRISP-dm. Fuente:[31]

Despliegue de solución: Finalmente cuando se ha comprobado que la formulación del algoritmo resuelve el problema planteado y responde a las preguntas de investigación, se procederá a desarrollar el artefacto empaquetado que podrá ser usado por el usuario. Este se espera que sea un software on-premise, desarrollado principalmente en Python, con librerías de creación de interfaces como por ejemplo tkinter.

3.2 Metodología específica

La metodología específica corresponde a la estrategia de desarrollo del artefacto final. Lo cual incluye algoritmos, lenguaje, estrategia de programación y ambiente de desarrollo.

3.2.1 Lenguaje y ambiente de programación

El lenguaje seleccionado es Python 3.7, con el cual se piensa desarrollar el algoritmo desde cero. Se evaluará si una programación orientada a objetos es la más adecuada o si se requiere una programación estructurada o una estrategia de programación modular. Para el IDLE se hará uso de Spyder, el cual hace parte de la plataforma integradora de desarrollo ANACONDA. Se escoge esta alternativa porque se considera que es la más profesional entre todas las demás y por su facilidad para visualizar los datos, variables y diagramas.

3.2.2 Algoritmos

El algoritmo de referencia será el expuesto en [12], donde se ilustra de manera clara como opera el algoritmo de k-means incremental. No se pretende hacer uso de librerías como puede ser scikitlearn, sino programarlo desde cero, de tal manera que permita la máxima flexibilidad posible en términos de: manipulación de variables,

control parámetros y persistencia de resultados. Sin embargo, se hará uso de otras librerías como numpy, pandas, math y las relacionadas con el manejo de datos .gpx.

Se pretende desarrollar el artefacto por medio de la programación de 5 prototipos de manera incremental, donde en cada uno de los releases sea posible reproducir lo del prototipo anterior, más el objetivo del prototipo actual. En la sección de planificación de programa se detalla cada uno de estos prototipos y su resultado.

4. Planificación del trabajo

El trabajo de fin de máster está dividido en 5 grandes entregas que brindan una estructura clara del camino a seguir, así como agrupan las actividades que cada una de ellas conlleva:

PEC1	Definición y planificación de trabajo final de máster
PEC2	Estado del arte o análisis del mercado del proyecto
PEC3	Diseño e implementación del trabajo
PEC4	Redacción de la memoria
PEC5	Presentación y defensa del proyecto

A continuación, se detalla cada una de las tareas a realizar en cada una de las entregas, el tiempo de trabajo que conlleva, así como el producto o los entregables que se harán a lo largo del plan de trabajo. Así mismo, se detallan aspectos del software a desarrollar.

4.1 Tareas y cargas de trabajo

PEC1:

Tarea:	Selección de línea de investigación		
Descripción: Dentro de las posibilidades de líneas de investigación de machine learning, se escoge la que vaya acorde a las asignaturas tomadas y a las motivaciones personales.			
Inicio	Fin	Horas trabajo	Tipo: Planificación
19/02/2020	19/20/2020	2	

Tarea:	Documentación general sobre el tema elegido		
Descripción: Después de elegir el tema, se requiere conocer un poco más sobre el problema específico, y sobre las técnicas posibles a usar para su resolución.			
Inicio	Fin	Horas trabajo	Tipo: Documentación
20/02/2020	23/02/2020	5	

Tarea:	Escritura del plan de trabajo		
Descripción: Se plasman las ideas de metodología para resolver el problema planteado en la línea de investigación tomando como base los formatos de TFM y lo encontrado en la documentación general.			
Inicio	Fin	Horas trabajo	Tipo: Planificación.
24/02/2020	01/03/2020	13	

PEC2:

Tarea:	Documentación específica		
Descripción: Se deben leer artículos sobre el estado del arte de: clustering, aprendizaje incremental, k-means, procesamiento de datos en streaming, reducción dimensionalidad, librerías para aprendizaje incremental.			
Inicio	Fin	Horas trabajo	Tipo: Documentación
02/03/2020	20/03/2020	40	

Tarea:	Investigación sobre herramientas		
Descripción: Los datos en formato .gpx son datos semi estructurados que deben ser explotados de la manera adecuada. Se explorarán las librerías, APIs y desarrollos disponibles para ser integrados con Python.			
Inicio	Fin	Horas trabajo	Tipo: Documentación
06/03/2020	08/03/2020	8	

Tarea:	Realización de pruebas preliminares		
Descripción: Con base en lo encontrado en las dos tareas pasadas, se procederá a hacer las pruebas del caso para comprobar el funcionamiento de las diferentes librerías o desarrollos encontrados que servirán como ejemplo para el despliegue del proyecto.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
09/03/2020	13/03/2020	2	

Tarea:	Escritura y entrega preliminar de PEC2		
Descripción: Se procederá a plasmar en documento lo encontrado en las tareas pasadas con el propósito de brindar un contexto mucho más amplio del problema a resolver y los enfoques que se han utilizado para resolver problemas similares. Se espera recibir feedback y hacer los ajustes requeridos.			
Inicio	Fin	Horas trabajo	Tipo: Documentación
14/03/2020	16/03/2020	10	

Tarea:	Ajustes de la entrega de la PEC2		
Descripción: Después del feedback recibido se ajustará el documento, añadiendo o removiendo las referencias que sean necesarias. Cumplir con el formato establecido para este fin.			
Inicio	Fin	Horas trabajo	Tipo: Documentación
17/03/2020	22/03/2020	5	

PEC3:

Tarea:	Procesamiento de data en formato .gpx		
Descripción: Se escribirá el código requerido para poder extraer de cada uno de los datasets de rutas: elevación, distancia, puntos del recorrido, tiempo, y que sea graficada.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
23/03/2020	29/03/2020	20	

Tarea:	Prototipo No.1 del modelo – agrupamiento sencillo		
Descripción: En esta primera versión se espera que el algoritmo pueda agrupar un conjunto de datos establecidos que pueden ser de ejemplo. No se espera que lo haga con nueva información, sino en forma de batch.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
30/03/2020	05/04/2020	20	

Tarea:	Prototipo No.2 del modelo – selección de variables		
Descripción: Se obtendrá la manera de representar las variables de las rutas, posiblemente con técnicas de reducción de dimensionalidad. Se probarán con el modelo desarrollado en el prototipo No.1			
Inicio	Fin	Horas trabajo	Tipo: Codificación
06/04/2020	12/04/2020	20	

Tarea:	Prototipo No.3 del modelo – persistencia de parámetros previos		
Descripción: Para esta fase no solo se querrá que agrupe en batch, sino que además agrupe nueva data. Por esto, se requerirá encontrar la manera de hacer que el modelo persista o almacene los parámetros de la última ejecución y así no empezar de cero cuando llega una nueva información.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
13/04/2020	19/04/2020	20	

Tarea:	Prototipo No.4 del modelo – threshold óptimo		
Descripción: Para este punto el modelo debe estar en capacidad de agrupar rutas existentes y nuevas rutas, pero ahora se deberá elegir el threshold “optimo”, por lo que se probará con: threshold constante, threshold incremental y threshold dinámico. Se hará uso de indicadores de rendimiento para establecer el mejor y se hará análisis de sensibilidad de los parámetros del algoritmo.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
20/04/2020	29/04/2020	30	

Tarea:	Video llamada con directora de tesis		
Descripción: En esta llamada se espera presentar en vivo el trabajo ya realizado, para ver el algoritmo operando y discutir sobre los puntos a mejorar y poder aclarar temas que hayan quedado durante el intercambio de correros que se dan periódicamente entre estudiante y directora.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
30/04/2020	30/04/2020	2	

Tarea:	Prototipo No.5 del modelo – Interfaz gráfica – Entrega preliminar		
Descripción: En esta fase lo que se busca es introducir el modelo de ML a una interfaz gráfica que le permita al usuario filtras, visualizar y agregar nuevas rutas. En esta fase habrá trabajo de diseño también.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
01/05/2020	16/05/2020	30	

Tarea:	Entrega definitiva PEC3		
Descripción: Con el feeedback recibido se espera hacer los ajustes del caso y completar la entrega establecida, considerando posibles cambios o modificación de código.			
Inicio	Fin	Horas trabajo	Tipo: Codificación
17/05/2020	23/05/2020	5	

PEC4:

Tarea:	Escritura y presentación preliminar de la memoria		
Descripción: En esta fase se busca recoger todo lo que se ha hecho en las demás entregas y ordenarlas de manera comprensible, siguiente un hilo conductor claro bajo el formato establecido.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
24/05/2020	04/06/2020	30	

Tarea:	Entrega definitiva de la PEC4		
Descripción: Se espera entregar la memoria completa, considerando el feedback de la directora de tesis, el formato establecido y las normas para citar.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
05/06/2020	10/06/2020	10	

PEC5:

Tarea:	Realización de la ppt		
Descripción: Se presentará en menos de 30 diapositivas el contenido del proyecto, desde el contexto, pasando por el marco teórico, algoritmo planteado, experimentación hasta llegar a los resultados. Se espera sea una presentación para todo tipo de público, técnico y no técnico.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
11/06/2020	15/06/2020	22	

Tarea:	Realización de video de presentación		
Descripción: Después de construida la pptx, esta será la base del guion a seguir en el video de presentación para la sustentación de tesis. Debe ser un video no mayor a 20 minutos. Es requerido que el estudiante aparezca en él.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
14/06/2020	16/06/2020	3	

Defensa pública:

Tarea:	Debate virtual		
Descripción: Durante una semana el jurado podrá formular preguntas sobre el proyecto, las cuales deben ser respondidas dentro de las 24 horas siguientes o antes de la fecha de finalización del debate.			
Inicio	Fin	Horas trabajo	Tipo: Presentación
17/06/2020	24/06/2020	10	

A continuación, se resumen el esfuerzo por cada tipo de tarea, considerando el número de actividades y las horas a emplear en ellas:

Tipo de tarea	Conteo de tareas	Horas totales	% del total
Planificación	2	15	5%
Documentación	5	68	22%
Codificación	8	147	48%
Presentación	6	77	25%

Totales:	21	307

El plan está diseñado para que el 70% del tiempo sea invertido en aspectos que estén directamente relacionados con el desarrollo del algoritmo, es decir: documentación más codificación. El 30% restante es usado para la metodología general y presentación de los resultados.

4.2 Diagrama de Gantt de tareas a realizar

Considerando que es un proyecto colaborativo donde la información debe estar disponible prácticamente en tiempo real entre director y estudiante, se decide hacer uso de una plataforma libre llamada Trello, que junto con goodgantt permite gestionar los proyectos de equipo. Se tiene la vista en Gantt como se ve a continuación:

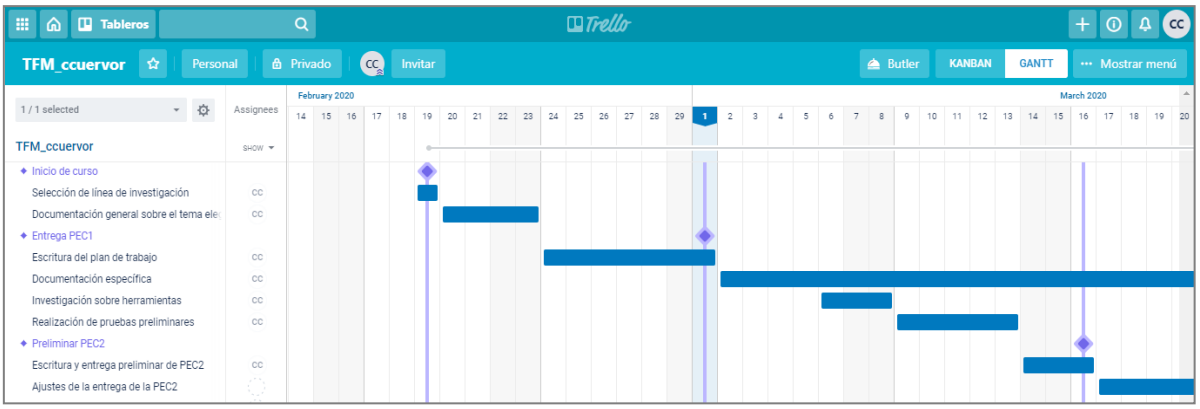

Figura 3: Diagrama Gantt en Trello. Fuente: Autor

Pero también es posible ver las tareas en formato Kanban, logrando saber qué hay en proceso, qué ya se ha ejecutado, qué está próximo a vencer o qué todavía está en lista de tareas sin arrancar:

Figura 4: Diagrama kanban en Trello. Fuente: Autor

El diagrama es público y se puede encontrar en el siguiente enlace: https://trello.com/b/NZIeB0TC/tfmccuervor Para el caso del kanban no se requiere nada adicional, para el caso del diagrama Gantt sólo es cuestión de agregar un complemento al navegador llamado Good Gantt.

5. Marco teórico y trabajos relacionados

En este apartado se abordarán los temas relacionados con el proyecto que comprenden tanto el estado actual del problema de búsqueda de rutas de deportes de exterior, así como los conceptos teóricos relacionados con agrupamiento, aprendizaje incremental, agrupamiento incremental, reducción de dimensionalidad y finalmente se hace revisión de las herramientas usadas para manipular y visualizar datos en formatos GPX con ayuda del lenguaje de programación Python.

5.1 Explotación de datos GPS y rutas

Gracias a que la humanidad hoy en día está mucho más conectada que hace unas décadas, es posible obtener registro de los lugares visitados por medio de la georreferenciación, de hecho, es posible trazar rutas habituales de las que una persona hace uso cada día. Esto se ha convertido en una muy rica fuente de información que abre posibilidades de hacer diferentes tipos de análisis en ámbitos como: deporte, logística, tránsito, políticas públicas, etc.

No se tienen muchos trabajos relacionados con la explotación de datos por medio de aprendizaje automático de deportes de exterior como es correr (running) y montar en bicicleta (cycling), pero se encuentran trabajos como [1], que expone las vulnerabilidades (identificar lugar de vivienda, rutina diaria, lugares frecuentes, etc) a las que una persona se expone al hacer uso de aplicaciones de rutas que no protegen la privacidad de su información como lo son Strava[1], Endomondo[2] o Movescount[3]. Así que el enfoque ha sido más hacia crear aplicativos que permitan llevar un registro seguro y privado de los movimientos de los usuarios. Es por esto, y otras razones que poder crear un aplicativo que agrupe las rutas sin necesidad de saber quien las creo, se vuelve un aporte significativo.

Por otra parte, el uso de datos GPS de rutas y aprendizaje automático es más extendido en otros ámbitos, por ejemplo, en [2] se hace uso de la regresión múltiple, K-NN(k-nearest neighbours),validación cruzada y análisis de componentes principales (PCA), para clasificar las rutas de una ciudad croata y así lograr tener un mejor entendimiento del tránsito y congestión vehicular en dicha ciudad. Se evidencia además la importancia de usar reducción de dimensionalidad para mejorar el rendimiento computacional y manejar registros con gran número de variables. Sin embargo, el uso de K-NN no es apropiado para manejar un problema incremental, ya que es un algoritmo de clasificación de tipo *lazy learning method* [4], es decir, que no crea un modelo en sí, sino que va actualizándose cada vez que llega una nueva instancia, y requiere que se almacene toda la data, que es lo que precisamente no se desea en este proyecto.

[1] Strava: https://www.strava.com/
[2] Endomundo: https://www.endomondo.com/
[3] Movescount: http://www.movescount.com/es/

Otro ejemplo es el de [3], donde se hizo uso del algoritmo de agrupamiento k-means que minimiza la suma de cuadrados del error (WCSS – within-cluster sum of squares), para tipificar y encontrar similitudes o diferencias entre las rutas de paratránsito en la ciudad de cabo, África del sur. Este algoritmo pudo identificar 3 tipos de rutas: troncales, intermedias y alimentadores, haciendo selección de variables como: distancia, velocidad, número de pasajeros, número de paradas, entre otras. De este trabajo es importante considerar qué variables pueden ser relevantes al momento de trabajar con rutas y, además, que al parecer PCA no siempre es concluyente al querer reducir la dimensionalidad de un conjunto de datos.

5.2 Algoritmos de Agrupamiento

Los algoritmos de agrupamiento son los que hacen parte del aprendizaje no supervisado del aprendizaje automático. Se caracterizan porque basan su entrenamiento en un juego de datos sin etiquetas o clases establecidas con antelación. Como se menciona en el primer capítulo de [4], su objetivo principal es encontrar clústeres o segmentos similares en los juegos de datos.

Estos algoritmos pueden ser de dos tipos:
Los métodos jerárquicos, que producen una organización jerárquica de las instancias que forman el conjunto de datos, haciendo posible diferentes agrupaciones dependiendo de lo que se observe en el dendrograma.
Los métodos particionales, que generan clústeres de instancias que no responden a ningún tipo de organización jerárquica.

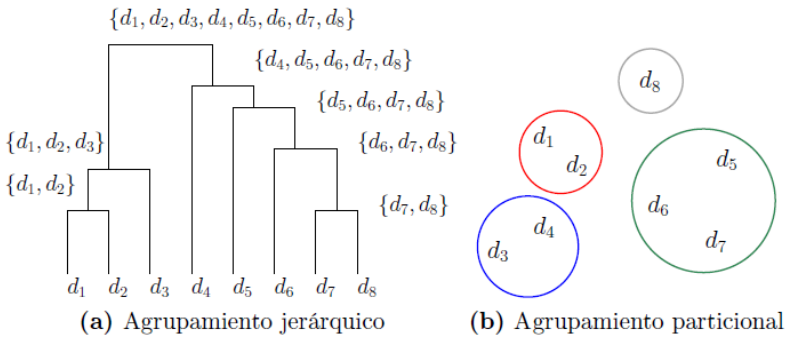

Figura 5: Métodos de agrupamiento. Fuente: [4]

En la figura 5, es posible ver gráficamente el ejemplo de estos dos tipos de agrupamiento. El primero en la figura 5.a, permite ir agrupando los elementos a medida que se va subiendo por el dendrograma, lo que posibilita la elección del número de clústeres. En la figura 5.b, se observa que no hay ningún tipo de jerarquía, sencillamente se agrupan los elementos dependiendo de criterios de cercanía o similitud y número deseado de clústeres (k).
Estos algoritmos permiten, una vez creados los clústeres, determinar a qué grupo pertenece una nueva instancia, lo que lo habilita para realizar análisis predictivos.

5.2.1 Agrupamiento Jerárquico

Como se menciona en el capítulo siete de [4], el agrupamiento jerárquico (hierarchical clustering o HC, en inglés) es un método de agrupamiento no supervisada que busca construir una jerarquía de clústeres en el conjunto de datos.

Hay dos tipos de algoritmos de agrupamiento jerárquico con enfoques inversos:
El algoritmo aglomerativo (Agglomerative Hirerchical Clustering, AHC): Este inicia desde una fragmentación completa de los datos, es decir, cada dato es un grupo y fusiona los clústeres progresivamente hasta que todos los datos pertenecen a un único grupo. Esta estrategia es conocida como bottom-up. Este resulta ser muchas veces más sencillo porque existe un único modo de unir clústeres, mientras que existen muchas maneras de separar un conjunto en más elementos.

El algoritmo divisivo (Divisive Hirerchical Clustering, DHC): Es inverso a AHC e inicia con un único grupo que contiene todas las instancias y lo va dividiendo de forma recursiva hasta obtener un grupo para cada dato. Esta estrategia se le conoce como top-down. Suele ser más eficiente que el aglomerativo, porque no se suele llegar al final del árbol, es decir al punto donde cada instancia es un grupo por si solo, así que se define un criterio de parada.

En la figura 6, es posible comparar los algoritmos y pseudocódigos de agrupamiento jerárquico aglomerativo (figura 6.a) y divisivo (figura 6.b). Como se observa cada uno se enfrenta a consideraciones particulares. Por ejemplo, el aglomerativo debe decidir:

Cómo calcular la distancia entre dos puntos: la cual suele ser la distancia euclídea, pero dependiendo el problema puede ser otra, ejemplo Manhattan.

Cómo determinar la distancia mínima entre dos grupos: se refiere a la manea de determinar si dos grupos se encuentran cercanos o no. Para esto se hace uso del concepto de linkage o enlace. Hay diferentes tipos como, por ejemplo:

a) Enlace simple: Se basa en la distancia mínima entre dos miembros de diferentes grupos.

$$d(A,B) = \min\{d\left(a_i, b_j\right) a_i \in A, b_j \in B\} \tag{1}$$

b) Enlace completo: Se basa en la distancia máxima entre dos miembros de diferentes grupos.

$$d(A,B) = \max\{d\left(a_i, b_j\right) a_i \in A, b_j \in B\} \tag{2}$$

c) Enlace medio: Se basa en tomar el promedio de todos los miembros de los grupos a los que se quiere determinar su distancia. Trata de corregir la sensibilidad a los outliers de los dos enlaces anteriores.

$$d(A,B) = \frac{1}{|A||B|} \sum_{a_i \in A} \sum_{b_j \in B} d(a_i, b_j) \tag{3}$$

Algoritmo 1 Pseudocódigo del agrupamiento jerárquico aglomerativo (AHC)

Entrada: D (conjunto de datos individuales)
  Inicializar $\ell = 0$
  Inicializar el conjunto de clústeres $C_\ell = D$
  mientras $|C_\ell| > 1$ **hacer**
    Encontrar $c_i, c_j \in C_\ell$ tal que la distancia $d(c_i, c_j)$ sea mínima
    Crear $c_* = c_i \cup c_j$ como padre de los clústeres c_i i c_j
    Actualizar $C_{\ell+1} = C_\ell - \{c_i, c_j\} \cup c_*$
    $\ell = \ell + 1$
  fin mientras
  devolver C

(a)

Algoritmo 2 Pseudocódigo del agrupamiento jerárquico divisivo (DHC)

Entrada: D (conjunto de datos individuales)
  Crear clúster inicial $C = \{D\}$ y marcarlo como «abierto»
  mientras (existe un clúster abierto) **hacer**
    Encontrar el clúster abierto $c^d \in C$
    si (c^d no cumple la condición de parada) **entonces**
      Dividir $c^d = c_1^d \cup c_2^d \cup \ldots \cup c_m^d$
      para $i = 1, 2, \ldots, m$ **hacer**
        Crear el clúster c_i^d como descendiente de c^d
      fin para
      Marcar c^d como «cerrado»
    si no
      Marcar c^d como «cerrado»
    fin si
  fin mientras

(b)

Figura 6: Pseudocódigo algoritmos AHC y DHC. Fuente: [4]

d) Enlace centroide: Será el cálculo de la distancia entre los dos centroides de los grupos a comparar. Es más eficiente computacionalmente porque no requiere iterar sobre todos los puntos. Este criterio es el que se usará en este proyecto para determinar la distancia entre dos grupos, ya que en algoritmos de partición de agrupamiento como k-means también se requiere determinar la distancia entre dos grupos.

Por su parte el algoritmo divisivo debe tener en consideración:

- La profundidad máxima del árbol resultante, ya que no siempre se querrá llegar al último nivel donde cada instancia es un grupo, ya que no aporta información útil.
- El número máximo de instancias que deberán de tener los grupos. Ya que se debe tener control sobre el balance entre los grupos y no dividir grupos que ya son muy pequeños.
- Un valor máximo para una medida de similitud. Es decir, los grupos que no superen el umbral establecido no serán divididas otra vez, ya que presentan un nivel de cohesión suficiente.

5.2.2 Agrupamiento Particional

Así como en el caso aglomerativo, es un algoritmo no supervisado que se basa en la partición de *n* datos o instancias en un número determinado de *k* clústeres, donde cada instancia pertenece al grupo cuya distancia es menor.
Con esta primera definición obtenida del capítulo ocho de [4], es claro que se pueden tener muchas variantes considerando diferencias en como se obtiene la medida de distancia, cómo se calculan los centroides o si se hace una partición dura o suave, así que el algoritmo más popular es el k-means, pero existen otros como es el k-medians, k-medoids o el fuzzy c-means, por nombrar algunos.

5.2.2.1 Algoritmo k-means

Cómo se mencionó es el algoritmo más popular, el cual se puede dividir en dos grandes fases. La primera es de inicialización, donde se eligen los centroides de los k clústeres que se desea partir (deben ser ingresados a priori). La segunda, es la fase iterativa donde, asignar cada instancia a un grupo, dependiendo de la distancia mínima entre esta y cada centroide, después se procede a actualizar el centroide después del agrupamiento. En la figura 7 es posible ver el pseudocódigo del algoritmo.

Algoritmo 3 Pseudocódigo del algoritmo *k-means*

Entrada: D (conjunto de datos) y k (número de clústeres)
  Inicializar los clústeres vacíos $P = \{p_1, p_2, \ldots, p_k\}$
  Seleccionar los *centroides* iniciales $\zeta_1, \zeta_2, \ldots, \zeta_k$
  mientras (Condición de parada no satisfecha) **hacer**
    para todo $d_i \in D$ **hacer**
      Asignar la instancia d_i al clúster $p_j \mid d(d_i, \zeta_j)$ es mínima
    fin para
    para todo $p_i \in P$ **hacer**
      Recalcular ζ_i según los valores $d_i \in p_i$
    fin para
  fin mientras
  devolver El conjunto de clústeres P

Figura 7: Pseudocódigo de k-means. Fuente: [4]

Después de analizar el algoritmo de la figura 7, es claro que quedan interrogantes relacionadas a como operarlo, ya que se pueden generar variantes. Por ejemplo, la inicialización de los centroides, estos pueden ser iguales a las coordenadas de alguna instancia o no. En caso de que no, deben estar dentro del rango. La manera más usual de inicializarlos es de manera aleatoria:

$$\zeta_i = rand(d_i) | d_i \in D, i = \{1,2, \ldots, \mathrm{m}\} \tag{4}$$

En este caso se elige una de las instancias dentro de las *m* que hay en el conjunto de datos de manera aleatoria. Esto provoca que la semilla del algoritmo cambie cada momento que se inicializa, pero se soluciona después de correrlo varias veces y determinar el centroide.

A pesar de que la inicialización aleatoria es buena en la mayoría de los casos, hay otras formas de inicializar k-means como se presenta en [7], donde se hace la comparación de cuatro métodos diferentes: Forgy, Aleatorio, MacQueen y Kaufman. Esto considerando que, como se mencionó anteriormente, el punto de partida puede alterar el resultado del algoritmo y en muchos casos ser determinante. En ese trabajo se concluye que el método Aleatorio y Kaufman presentan los mejores resultados al comparar la distribución de probabilidad del error cuadrático, además de ser menos dependientes del orden de inicio de las instancias.

Para el caso de este proyecto considerando que es un aprendizaje incremental y donde se tiene el supuesto de que no se tiene toda la data de entrenamiento a priori, no cabe la posibilidad de elegir un punto de inicio de los centroides, ya que estarán dados de la misma manera como vaya llegando la data.

Otro aspecto para tener en cuenta es el cálculo de la distancia, la cual suele ser el resultado del cálculo de la distancia euclídea, pero podría ser otra como la distancia cuadrada media [6], o como se menciona en el capítulo dos de [4], la distancia estadística o de Gauss, la distancia de Mahalanobis o la distancia de Hamming para atributos nominales o binarios.

El tercer aspecto para tener en cuenta es el recálculo de los centroides, el cual para el caso de k-means, corresponde a la media de los puntos del grupo, por eso su nombre. Sin embargo, como se verá más adelante la variación de esto genera diferentes aproximaciones al problema.

Por último, se tiene el criterio de parada, el cual como se vio en el algoritmo de ACH, debía ser establecido, pero para este caso, la convergencia del algoritmo se da de manera natural y último ya que se detiene en el momento en el que todas las instancias estén asignadas a algún grupo.

5.2.2.2 Algoritmo de k-medians

Es una variante del algoritmo de k-means, pero que en lugar de usar la media para calcular los centroides (eficiente y fácilmente interpretable, pero que presenta problemas cuando los datos no están distribuidos de manera simétrica), utiliza la mediana. Es más costoso computacionalmente, pero puede ser preferible en algunos contextos específicos como por ejemplo cuando la distribución es asimétrica o hay presencia de outliers. Suele usar la distancia de Manhattan para determinar la distancia de los puntos y los centroides.

5.2.2.3 Algoritmo de k-medoids

Este algoritmo propone el cálculo de los centroides basado en las coordenadas de las instancias que presentan menor disimilitud con las demás instancias del mismo grupo, conocido como medoid. Claramente tiene un costo computacional elevado porque deberá saber la distancia de cada punto con todas las instancias del grupo al que pertenece y elegir el que tenga la mínima distancia total. Sin embargo, es mucho más robusto ante el ruido y los outliers.

5.2.2.4 Fuzzy C-means

Es una generalización de los métodos anteriores. Construye una partición difusa o suave mientras que los otros métodos construyen una partición disjunta o dura. La idea se basa en que no todas las instancias tienen la misma relación de pertenencia con el grupo o segmento. Como resultado se tendrá la probabilidad de pertenecer a cada uno de los clústeres P = {p1, p2, p3,…,pk} dependiendo de la distancia con cada centroide. Así que será útil para cuando se quiera medir el grado de pertenencia de una instancia a cada grupo. Se considera una partición suave del conjunto de datos D, sí y sólo sí, se cumplen las siguientes dos restricciones expuestas en [5]:

$$\forall\, d_i \in D\,, \forall p_j \in P \qquad 0 \leq \mu p_j(d_i) \leq 1 \tag{5}$$

$$\forall\, d_i \in D\,, \exists p_j \in P \quad tal\ que\ \mu p_j(d_i) > 0 \tag{6}$$

Donde *µpj(di)* denota el grado en el que la instancia di pertenece al clúster *pj.*

$$\sum_{p_j \in P} \mu p_j(d_i) = 1 \quad \forall\, d_i \in D \tag{7}$$

Por otro lado, si la partición suave cumple la restricción de la ecuación (7), se dice que es una partición suave restringida, donde la suma de todos los grados de pertenencia de una instancia a los grupos debe ser igual a 1. Si esto ocurre se pueden presentar resultados como el que se evidencia en la figura 8, donde cada una de las 20 instancias tiene una suma de pertenencia igual a 1, repartida entre dos diferentes grupos.

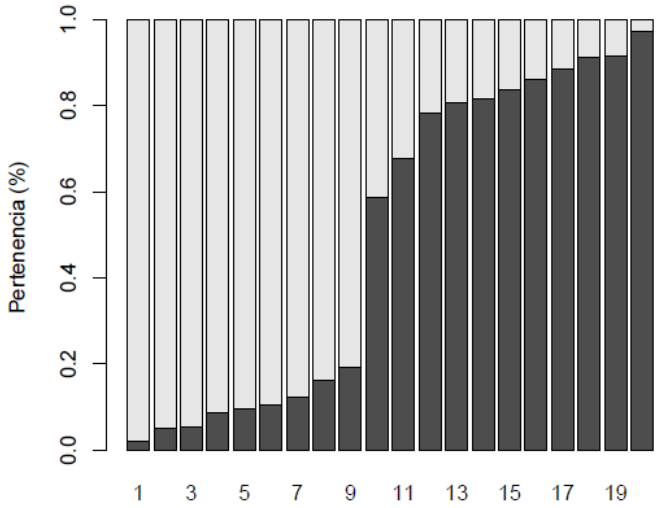

Figura 8: Resultado de partición suave restringida. Fuente: [4]

5.2.2.5 Métricas de validación y k óptimo

Los modelos no supervisados, a diferencia de los supervisados, no poseen etiquetas, por lo que no es posible hacer uso de matrices de confusión o análisis de curvas ROC y obtención de un valor de AUC. Sin embargo, como se ve en el capítulo cinco de [4], es posible medir el rendimiento de los algoritmos de clustering con métricas de validación interna, como de validación externa, estas últimas asumiendo que se tiene información previa sobre la clasificación de los datos.

La idea de las métricas internas es evaluar la cohesión (homogeneidad del clúster) y la separación (distancia entre los clústeres) de las particiones [22], y para esto se hace uso de varias métricas o índices. Dentro de los más tradicionales se encuentra el DUNN index [4], el cual es la proporción entre la distancia mínima entre puntos que no sea del mismo clúster, ecuación (8), y la distancia máxima entre clústeres, ecuación (9), como se observa en la ecuación (10):

$$diam(p) = max\{d(x_1, x_2)\} \mid x_1, x_2 \in p \tag{8}$$

Donde *x1* y *x2*, hacen referencia a dos puntos cualquiera de la partición *p.*

$$sep(p) = min\{d(x_1, x_2)\} \mid x_1 \in D^p, x_2 \in D - D^p \tag{9}$$

Donde *Dp* es el conjunto de instancias de la partición *p*, y *D-Dp*, es el conjunto de instancias restantes que no pertenecen a la partición *p*.

$$dunn = \frac{\min\ sep(p)}{\max diam(p)} \mid p \in P \tag{10}$$

Es de esperar que cuanto menor sea este indicador, mejor será el rendimiento del proceso de agrupamiento.

Sin embargo, hay otras métricas un poco más elaboradas como las que se mencionan en [21]. En este mismo trabajo se muestra como la suma al cuadrado del error total de los clústeres se puede desagregar entre la suma al cuadrado de los errores dentro-grupos (SSW) y la suma al cuadrado de los errores entre grupos (SSB), de manera análoga a como se hace en ANOVA.

$$SSW = \sum_{i=1}^{k} \sum_{x \in p_i} d^2(x, c_i) \tag{11}$$

Sea *k* el número de clústeres, *x* un punto del clúster *pi* y *ci* el centroide del clúster *pi*.

$$SSB = \sum_{i=1}^{k} n_i d^2(\mu, c_i) \tag{12}$$

Sea *k* el número de clústeres, *nj* el número de instancias en el clúster i, *ci* el centroide del clúster i y μ es la media de todo el data set.

Con los valores calculados en (11) y (12) que denotan cohesión y separación respectivamente, es posible calcular los siguientes índices:

$$Ball\ \&\ Hall = \frac{SSW}{k} \tag{13}$$

$$Calinski\ y\ Harabasz = \frac{SSB/(k-1)}{SSW/(n-k)} \tag{14}$$

Siendo k el número de clústeres y n el número total de datos del set de datos.

$$Hartigan = log\left(\frac{SSB}{SSW}\right) \tag{15}$$

$$Xu = d * log\left(\sqrt{\frac{SSW}{d * n^2}}\right) + \log(k) \tag{16}$$

Siendo d la dimensión de los datos y *n* el número total de datos del set de datos.
También es posible medir qué tan bien agrupado está un punto específico dentro de su grupo, por medio del coeficiente de Silhouette [22]:

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}} \tag{17}$$

Donde b(x) es la distancia promedio de x a todos los demás puntos en el clúster más cercano, mientras que a(x) es la distancia promedio de x a todos los demás puntos del mismo clúster.
S(x) puede variar entre -1 y 1, donde: –1 = mal agrupamiento; 0 = indiferente
1 = bueno. Ahora, si se desea saber el desempeño de todo el agrupamiento:

$$SC = \frac{1}{n}\sum_{i=1}^{n} s(x) \tag{18}$$

<u>Obtención de k óptimo:</u>
Calcular el k óptimo se reduce a ejecutar el algoritmo varias veces y en cada una de ellas, a diferentes valores de k, medir su rendimiento con algún índice de los anteriormente mencionados, y al final elegir la configuración que tenga el mejor rendimiento. Como se observa en [23] hay diferentes métodos visuales para hacer esta elección, entre los que se encuentra: a) el método del codo, que basa su métrica en la suma del error dentro-grupos que se vio líneas arriba, y hay que elegir el número de grupos tal, que al adicionar uno más, no mejore mucho la suma total del error; b) el promedio de silhouette, que se puede observar de manera gráfica implementado en Python[4] en la figura 9, c) Además, se tiene el método de Gap estadístico que se basa en encontrar un valor de k tal, que al comparar la suma de errores dentro-grupos del agrupamiento, sea mucho mayor al error dentro-grupos promedio, de las agrupaciones hechas a subconjuntos de datos seleccionados de manera aleatoria con una distribución uniforme del conjunto original.

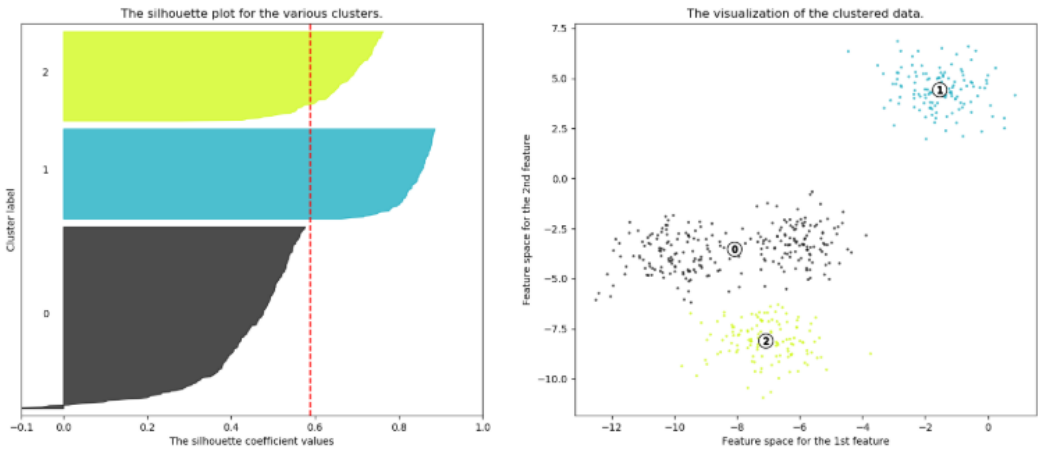

Figura 9: Ejemplo método silhouette para k=3. Fuente: Scikit-learn

5.2.2.6 Preclustering Canopy

Cómo se menciona en el capítulo 9 de [4], el preclustering es el conjunto de tareas que se realizan previo a un proceso de agrupamiento, con el objetivo de reducir el número

[4] Scikit-learn: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

de instancias a evaluar, y por ende el número de cálculos es mucho menor. Es especialmente utilizado para cuando el número de datos es demasiado grande.

Se divide principalmente en dos fases:

- **Fase 1:** Usar una métrica sencilla de distancia, para generar los canopies o subgrupos superpuestos de instancias. Cada instancia podrá pertenecer a más de un canopy, pero deberá pertenecer a al menos uno.

- **Fase 2:** Se utiliza un método de agrupamiento (ya sea jerárquico o particional), pero con la restricción de calcular únicamente distancias entre puntos que pertenezcan al mismo canopy.

En la figura 10 es posible observar con más detalle el pseudicódigo del algoritmo. En síntesis, lo que define es un par de umbrales, denominados T1 y T2 (T2 < T1). Si la distancia entre el punto elegido al azar y el núcleo del canopy que se esté evaluando es menor a T1, este punto es añadido al canopy, pero si además el punto es inferior a T2, no sólo se añade al canopy, sino que además se elimina del conjunto de datos a evaluar, ya que este punto no será elegido más adelante como núcleo de algún otro canopy. Esto se hace hasta que el conjunto de datos quede vacío.

Algoritmo 4 Pseudocódigo del algoritmo Canopy

Entrada: D (conjunto de datos), $T_1, T_2 \mid T_1 > T_2$ (distancias)
  mientras $(D <> \emptyset)$ **hacer**
    Elegir un d_i aleatoriamente e inicializar nuevo *canopy* $\omega_v = \{d_i\}$
    Eliminar d_i del conjunto de datos $D = D - \{d_i\}$
    para todo $d_j \in D$ **hacer**
      Calcular la distancia $\Delta_{i,j} = d(d_i, d_j)$ con la métrica sencilla
      si $(\Delta_{i,j} < T_1)$ **entonces**
        Añadir d_j al *canopy*, esto es $\omega_v = \omega_v \cup \{d_j\}$
      fin si
      si $(\Delta_{i,j} < T_2)$ **entonces**
        Eliminar d_j del conjunto de datos $D = D - \{d_j\}$
      fin si
    fin para
  fin mientras
  devolver El conjunto de *canopies*

Figura 10: Pseudocódigo del algoritmo Canopy. Fuente: [4]

Este tipo de técnicas no sólo ayudan a acelerar el proceso de agrupamiento en grandes volúmenes de datos, sino que, además, en algunos casos ha permitido mejorar el resultado final.

5.2.2.7 ISODATA

Cómo se ha podido observar, los algoritmos de agrupamiento vistos hasta ahora tienen la particularidad que se debe elegir el número de grupos, ya sea en un enfoque jerárquico (altura del dendrograma) o en uno particional (elección del número de ks).

Sin embargo, el algoritmo ISODATA (Iterative self- Data Analysis Technique, por sus siglas en inglés, más la adición de la última "A")[5], es una generalización de k-means, que permite separar, unir y eliminar grupos de manera dinámica [14], es uno de los métodos más usados en aplicaciones de procesamiento de imágenes en geociencias. Aunque no tiene un criterio específico de optimización, si se asemeja a k-means, en que busca minimizar la distancia cuadrada media entre los puntos y sus centroides más cercanos, a lo que se conoce como la distorsión promedio [6]. A pesar de ser dinámico requiere de una serie de parámetros para proceder, como son: número inicial de grupos, máximo número de grupos, máxima desviación entre los puntos de un grupo y su centro, la distancia mínima entre dos grupos, máximo número de iteraciones, entre otros más. Este algoritmo de manera iterativa, dadas esas reglas, separa y une grupos, de tal manera de encontrar la mejor distribución de puntos que minimice la distorsión. Sin embargo, según [14] es un algoritmo que puede llegar a la no convergencia.
De ISODATA se desprenden otros algoritmos como el que se presenta en [14], llamado ISOCLUS, pero con una particularidad y es que no procesa la distorsión por media de la suma de la distancia euclídea entre los puntos y su centroide, sino que hace uso de la raíz cuadrada de la distancia media cuadrada, obteniendo resultados similares, pero aumentando la velocidad de procesamiento.

5.2.3 Agrupamiento por densidad y cuadrícula

Los algoritmos de agrupamiento se pueden categorizar en cuatro grupos: los jerárquicos, los particionales, los basados en densidad y los de grilla [20]. Hasta ahora sólo se han presentado los jerárquicos y los particionales que suelen ser los más utilizados. Sin embargo, para los clústeres tienen formas irregulares, son de distinto tamaño, diferente densidad, o hay demasiado ruido (outliers), lo adecuado es utilizar algoritmos basados en densidad. Estos últimos son capaces de reconocer clústeres de formas arbitrarias, son robustos antes la presencia de ruido y escalables porque hacen un único recorrido a los datos [24].

Hay varios algoritmos de densidad entre los cuales se encuentran:

- DBSCAN: Density Based Spatial Clustering of Applications with Noise
- OPTICS: Ordering Points To Identify the Clustering Structure
- DENCLUE: DENsity-based CLUstEring
- CLIQUE: Clustering in QUEst
- SNN (Shared Nearest Neighbor)

DBSCAN resulta ser uno de los más utilizados. Tiene dos parámetros principales que son épsilon y minpts. Se denomina "core", al conjunto de datos interiores de un clúster con un número mínimo puntos minpts y un radio máximo de épsilon [24]. Todos los puntos que están por fuera de esta frontera son catalogados como "ruido". En la figura 11 se observan como participan estos parámetros en el algoritmo.

[5] ISODATA:http://leefoxadvremote.blogspot.com/2014/10/unsupervised-isodata-classification.html

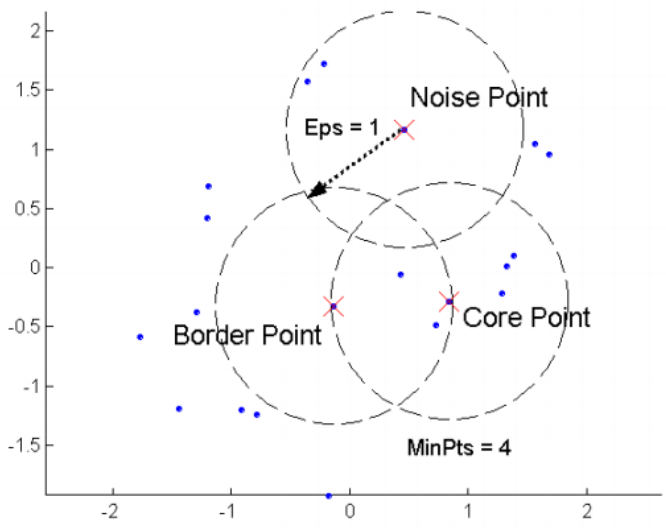

Figura 11: Parámetros de DBSCAN. Fuente: [24]

El algoritmo se basa principalmente en primero eliminar el ruido del conjunto de datos y luego agrupar a aquellos datos restantes.

Basados en cuadrícula:

Los métodos basados en cuadrículas cuantifican el espacio del punto u objeto en un número finito de celdas (hiper-rectángulos) y luego realizan las operaciones requeridas en el espacio cuantificado. La principal ventaja del método basado en grilla es su rápido tiempo de procesamiento que depende del número de celdas en cada dimensión en el espacio cuantificado [20]. Hay varios algoritmos de esta naturaleza, entre ellos CLIQUE (Clustering In QUEst), STING (Statistical Information Grid Approach to Spatial Data), Wave Clustering, entre otros.

Otras alternativas:

Para solucionar el problema de realizar agrupamiento de información espacial, los métodos basados en densidad o grilla no han sido los únicos desarrollados. Existe otro tipo de métodos que se basan en encontrar el perfil límite del clúster, más que en encontrar todos los puntos que lo contienen. El algoritmo BPIC (boundary-profile-based incremental clustering) en conjunto con BV-BPD (boundary-vector-based boundary point detection) son un ejemplo de esto, diseñados para encontrar clústeres con formas arbitrarias en conjuntos de datos que crecen dinámicamente [15], es decir, que hace uso de aprendizaje incremental, un concepto que se expondrá más adelante.

5.3 Incremental learning

El aprendizaje incremental en un paradigma del aprendizaje automático donde el proceso de aprendizaje se da en cualquier momento que una nueva instancia aparece y ajusta lo ya aprendido hasta el momento [8].
Una diferencia marcada con los modelos de aprendizaje automático tradicionales es que no asume la disponibilidad de un set de datos de entrenamiento suficiente a priori, sino que aparecen a lo largo del tiempo.

Adopta su concepto de la manera como los seres humanos aprenden, ya que este es incremental, donde un concepto es aprendido en un momento determinado y este es refinado a medida que la persona aprende u obtiene nueva información y hechos. Hay dos razones principales por las que el aprendizaje debe ser de esta manera para los humanos [8]:

- El flujo de información es secuencial
- La capacidad de memoria y de procesamiento es limitada.

Se puede decir que un algoritmo es de aprendizaje incremental si presenta las siguientes características:

1. Puede aprender y actualizarse con cada nuevo dato etiquetado o sin etiquetar.
2. Conserva el conocimiento previamente adquirido.
3. No requiere acceso a los datos originales.
4. Genera una nueva clase o clúster cuando es necesario.
5. Será de naturaleza dinámica con el entorno cambiante.

Hay principalmente dos metodologías para lograr esto. En la figura 12 se presentan estos dos enfoques. Por un lado, se tiene el método acumulativo de datos (figura 12.a), el cual, cuando recibe una nueva porción de datos *Dj*, descartará la hipótesis *hj-1* y desarrollará una nueva hipótesis *hj*, basado en todos los datos disponibles acumulados hasta el momento. Por el otro lado, se tiene el método de ensamble de hipótesis (figura 12.b), el cual, cuando recibe una nueva porción de datos *Dj*, se desarrolla una única hipótesis nueva o un conjunto de hipótesis nuevas basadas en los nuevos datos. Después se usará un mecanismo para combinar todas las decisiones (que puede ser por votación) de diferentes hipótesis para obtener la predicción final.

El enfoque de aprendizaje incremental se basa en solucionar dos principales problemas: 1) cómo pasar el conocimiento ya aprendido al nuevo bloque de datos, 2) cómo acumular la experiencia y el conocimiento ya adquirido.

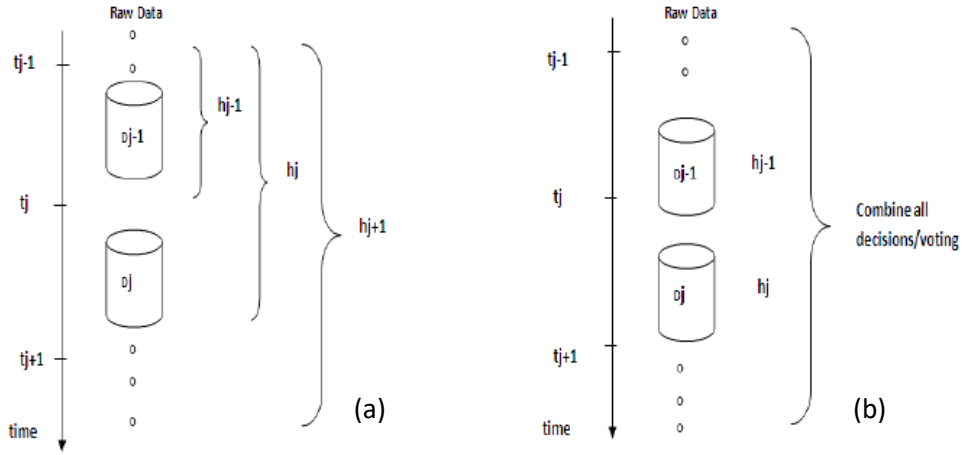

Figura 12: Métodos de aprendizaje incremental. Fuente: [8]

El aprendizaje incremental puede ser aplicado, tanto a modelos supervisados como no supervisados. Dentro de los supervisados se tiene ejemplos como [8]: Learn++ (algoritmo para redes neuronales supervisadas), Genetic Algotihms y SVM (support vector machines). En cuanto a los modelos no supervisados, se tiene: algoritmos CF1 (concept follower algorithm) y CF2 (concept follower with unlearning algorithm), los dos basados en el algoritmo tradicional de clasificación "Leader follower" [25]. Diseñados para lidiar con el "concept drift", es decir, con los cambios que se generan sobre el ambiente, además cuando los streams de datos no se encuentras etiquetados. Están inspirados en la manera como el sistema visual humano se adapta a los cambios del entorno o "concept drifts" y ajusta su conocimiento al recibir información sin necesidad de que tenga alguna etiqueta.
Otro algoritmo importante, es la estructura de redes neuronales llamada ARTMAP. En realidad, hay muchas versiones de esta misma arquitectura para diferentes tipos de problemas, entre ellos el clustering. Esta arquitectura se desarrolló para el aprendizaje supervisado incremental [26]. Dado su éxito, se crearon varios algoritmos modificados para diferentes situaciones, como el ARTMAP gaussiano para el aprendizaje incremental de datos ruidosos, la arquitectura híbrida del ARTMAP difuso y la red neuronal probabilística para el "on-line learning" y la "probability estimation", las estructuras celulares de aprendizaje permanente, el elipsoide ART y ARTMAP para la agrupación y clasificación incremental, entre otros.

Estos algoritmos son usados para el aprendizaje incremental. Sin embargo, se presentan problemas cuando se desea escalar a granes volúmenes de datos. Los modelos modernos de aprendizaje automático sufren de "catastrophy forgetting" (degradación considerable del rendimiento cuando la data pasada ya no está disponible), cuando aprenden nuevas clases de manera incremental y el rendimiento cae por datos perdidos de clases antiguas [9].
Para evitar esto se usan principalmente dos técnicas: "knowledge distilling" y mantener algunos ejemplos de las clases antiguas (exemplars), pero aun así tienen problemas escalando a un elevado número de clases. Esta data perdida genera dos problemas a resolver: a) mantener el rendimiento de clasificación y b) balancear entre clases nuevas y antiguas.

En [9] se propone el método BiaC (bias correction) para una CNN (convolutional neural network) en la clasificación de imágenes. Permite corregir el problema b) de desbalanceo entre clases nuevas y antiguas, y lo hace con un modelo lineal y un data set pequeño de validación, así que funciona mejor para grandes volúmenes de datos.

Para evitar el "catastrophy forgetting", se usan técnicas como el "knowledge distilling", o el almacenar datos, pero muchas veces cuando la data es masiva o la capacidad de almacenamiento es reducida, la segunda opción deja de serlo. En [10], se hace uso de la técnica de "knowledge distilling", para un algoritmo de reconocimiento de objetos, donde buscan transferir el conocimiento sin necesidad de data, además introduce un método de penalización de destilation, para conservar sólo la información más relevante.

Los algoritmos usados para la clasificación de data no estructurada en ambientes de grandes volúmenes de datos que se enfrentan al reto de la stream data, la cual viene de diferentes fuentes, gran tamaño a gran velocidad y no organizada (Big Data), encuentran en el aprendizaje incremental la manera de afrontar dicho reto. En [11] se propone la metodología CUIL (Classification of Unstructured data using Incremental Learning), que agrupa los metadatos, asigna una etiqueta a cada grupo y luego crea un modelo usando ELM (Extreme Learning Machine), una red neuronal de para cada lote de datos que va llegando. Esta metodología se divide en dos fases: la primera se encarga de agrupar los datos no estructurados usando uCLUST (algoritmo para clasificar data no estructurada como por ejemplo en formato JSON), los grupos creados se introducen en ELM para crear un modelo para cada subconjunto que llega en streaming. Cada modelo creado de forma incremental se almacena y, utilizando estos modelos, se clasifican los datos de prueba, así se evita el reentrenamiento de muestras y reduce la memoria.

5.4 Incremental clustering

El algoritmo de k-mean es sin duda el algoritmo más popular de agrupamiento particional. Sin embargo, presenta ciertas limitaciones como: requiere que se defina el número de clústeres con antelación, es sensible al punto de inicio de las semillas, sensible a outliers y tiene un número de iteraciones desconocido.
Dado todo esto, en [12] se propone un algoritmo incremental de clustering, que según sus autores es mucho más eficiente que el k-means clásico, ya que es más rápido, no requiere de un k previo, y el número de iteraciones estará dado por el número de instancias a agrupar, es decir, que recorrerá el conjunto de datos una sola vez.
Este algoritmo se basa en el establecimiento de un threshold (umbral), para determinar la similitud entre objetos.

A continuación, la descripción de los pasos del algoritmo:

a. El primer paso es establecer las n instancias a agrupar.
b. Asigne cualquier instancia de los datos al primer grupo.
c. Seleccionar la siguiente de manera aleatoria.

d. Determine la distancia entre la instancia seleccionado y los centroides de los grupos existentes.
e. Compare la distancia con el límite de umbral (threshold). Si la distancia es menor al umbral, agrupe el objeto en un grupo existente, pero si la distancia es mayor al umbral, forme un nuevo grupo con esa instancia.
f. Repetir los pasos del 3 al 5 hasta que se seleccionen todas las instancias.

En la figura 13, se presenta el pseaudocódigo del algoritmo de k-means incremental, adaptado de [12], considerando la notación del presente trabajo.

Algoritmo 5 Pseudocódigo del algoritmo de k-means incremental

Entrada: $D = \{d_1, d_2, \dots, d_n\}(conjunto\ de\ datos), T_{th}(Threshold)$
Inicializar la iteración con $k = 1$
Elegir d_i de manera aleatoria
Asignar la instancia al clúster $p_k = \{d_i\}$
Asignar el clúster creado al conjunto de clústeres $P = \{p_k\}$
Definir el centroide del clúster p_k como $\zeta_k = d_i$
Asignar un valor a T_{th}
para todo $d_i \in D | \forall\ i \geq 2$ **hacer**
calcular $\Delta_{ij} = \min dist(d_i, \zeta_k) | \forall\ p_k \in P$
si $(\Delta_{ij} \leq T_{th})$ **entonces**
Añadir la instancia al clúster de mínima distancia $p_j = p_j \cup d_i$
Recalcular la media del centroide de cluster ζ_j
de lo contrario
Aumentar el número de clusters $k = k + 1$
Asignar la instancia al clúster $p_k = \{d_i\}$
Asignar el clúster creado al conjunto de clústeres $P = \{p_k\}$
Definir el centroide del clúster p_k como $\zeta_k = d_i$
fin si
fin para
Devolver: $P = \{p_1, p_2, \dots, p_k\}(conjunto\ de\ clústeres)$,
$C = \{\zeta_1, \zeta_2, \dots, \zeta_k\}(conjunto\ de\ centroides)$

Figura 13: Pseudocódigo de k-means incremental. Fuente: Autor basado en [12]

Los resultados del experimento muestran que el agrupamiento se hace más rápido que el agrupamiento clásico. Así mismo se evidencio que a medida que el threshold es mayor, el número de clústeres se reduce, y como consecuencia el tiempo de procesamiento es menor, por la reducción de centroides a comparar.

Aunque no es explícito en [12], se entiende que es un algoritmo que se puede usar en los casos en los que no se tiene disponible toda la data o no se conoce el tamaño de esta con antelación (contrario al paso uno). Por otra parte, no se discute la manera de elegir el umbral óptimo que como se observa es determinante para el resultado el algoritmo.

Por su lado, en [13] se propone un algoritmo de clustering incremental, que no sólo permite agrupar data con atributos categóricos (ya que mide la frecuencia de aparición de atributos por clúster y mide la distancia por atributo), sino que además es adecuado para escenarios de grandes volúmenes de datos o en streaming porque (cómo en el

caso anterior) la data es recorrida una única vez. Además, a diferencia del algoritmo anterior, permite cambiar el número de clústeres de manera dinámica, es decir, que no solo aumente este número, sino que está la posibilidad de realizar fusiones y así reducir el número de clústeres. Esto último es controlado con el parámetro de máximo número de clústeres, cuando este es superado, une los dos más cercanos y el threshold cambia dinámicamente por la mínima distancia entre estos dos.

Cómo se mencionaba en [8], existen dos casos de aplicación de aprendizaje incremental, ya sea en línea o por batches o lotes. Los dos casos anteriores claramente son en línea (streaming), pero el que se presenta en [14] es un caso de clustering incremental por batches. Propone procedimientos estáticos y dinámicos para superar las limitaciones del k-means (expuestas al inicio de este apartado), y además enfrentar situaciones de grandes volúmenes o que la data venga dada en diferentes momentos del tiempo.
La idea consiste en agrupar porciones (chunks) manejables de datos y mantener los resultados para la siguiente porción de datos, hasta que no haya más datos para procesar. Pide que el bloque de datos sea lo más grande posible (según capacidades de memoria), para obtener mejores resultados.

Así como en [13], permite fusionar clústeres, pero esta vez no por el número máximo de clústeres establecido, sino por el número de elementos del clúster (cuando son muy pocos, el clúster se fusiona con otro), o cuando la cercanía es demasiada con otro, basado en un umbral de distancia, calculado según el MSE (mean squared error).

El experimento realizado fue corrido en MATLAB con la función MATLAN k-means y la evaluación fue basada en tres criterios: 1) la distorsión de la iteración, 2) el tiempo de ejecución, y 3) la calidad de la agrupación. Consideraciones de interés para los experimentos que se realizarán más adelante en este proyecto.

En el mundo de la Big Data, la preocupación de tener algoritmos eficientes y de rápido procesamiento es primordial. Hasta ahora se ha visto como enfoques de aprendizaje incremental han logrado mejoras en el tiempo de procesamiento, pero a veces no es suficiente. En el trabajo presentado en [16], se propone un algoritmo de agrupamiento jerárquico para Big Data, que en cada nivel del árbol hace uso de k-means incremental en paralelo. Es decir, que encontraron la manera de paralelizar las tareas del k-mean incremental y así poder aprovechar el procesamiento en GPU y CPU, para acelerar aún más el poder de procesamiento. Por otro lado, hace un análisis sobre la influencia de la elección del thershold (*thr*) sobre el rendimiento del algoritmo, y señala que considerar un incremento lineal de *thr'= [thr*constante]* puede conllevar a un valor demasiado grande o pequeño de thr, que agrupe todos los puntos en un solo nodo, o que le asigne un clúster a cada punto o instancia. Proponen entonces el uso de histograma de distancias para conocer la distribución de esta y obtener así un thr adecuado considerando una distancia euclídea, esto lo hace en cada nivel de jerarquía.
Está diseñado para grandes volúmenes de datos, pero así mismo para una gran dimensionalidad, porque realizan reducción de esta basados en la variabilidad aportada por cada una de ellas, en su experimento pasaron de 450 a 36 dimensiones de la data set denominado Aerosol.

5.5 Reducción de dimensionalidad

Cómo se ha visto hasta ahora, la reducción de dimensionalidad es de importancia cuando se tratan con problemas de culturización ya que la cantidad de datos y atributos puede afectar su rendimiento, además que permite una mejor interpretación visual, considerando que el cerebro humano sólo puede procesar cierto número de dimensiones [18]. Según el capítulo 11 de [17], los métodos de agrupación intentan encontrar grupos en subespacios del espacio de datos original, pero en algunas situaciones, es más efectivo construir un nuevo espacio en lugar de usar subespacios de los datos originales, y esta es la motivación detrás de la reducción de dimensionalidad, para agrupar datos de alta dimensión.

Cómo se indica en el capítulo 3 de [4], la reducción del número de atributos consiste en encontrar un subconjunto de atributos tal que permita obtener modelos de la misma calidad que los que se obtendrían utilizando todos los atributos, pero con una reducción en la complejidad temporal. Existen dos enfoques: 1) selección de atributos: técnicas para determinar qué atributos pueden ignorarse y cuales no, y 2) fusión y creación de nuevos atributos que son combinaciones lineales de pesos de los atributos {a1,a2,a3,…,an} y los pesos wj de contribución de cada atributo:

$$a_{n+1} = \sum_{i=1}^{n} a_j * w_j \tag{19}$$

El análisis de componentes principales (PCA) resulta ser uno de los métodos más utilizados de fusión y creación de nuevos atributos.

Según el capítulo 3 de [17], suponiendo que se tiene un set de datos de *n* dimensiones, el análisis de componentes principales (PCA, también llamado método de Karhunen-Loeve o K-L) buscará k vectores ortogonales n-dimensionales que se pueden usar mejor para representar los datos, donde k < n. Los datos originales se proyectan así en un espacio mucho más pequeño, lo que resulta en una reducción de la dimensionalidad. El procedimiento para usarlo es el siguiente:

1. Lo primero es normalizar los datos de entrada, ya que al estar dentro del mismo rango se garantiza que los atributos con unidades más grandes dominen los atributos con unidades más pequeñas.
2. PCA calcula k vectores ortonormales que proporcionan una base para los datos de entrada normalizados. Estos son vectores unitarios que cada punto apunta en una dirección perpendicular a los demás, esto son los que se les llama componentes principales.
3. Los componentes principales se ordenan de manera decreciente de "importancia" o fuerza. Los componentes principales sirven esencialmente como un nuevo conjunto de ejes para los datos que proporcionan información importante sobre la varianza. Es decir, los ejes ordenados son tales que el primer eje muestra la mayor variación entre los datos, el segundo eje muestra la siguiente mayor varianza, y así sucesivamente.

4. Debido a que los componentes se ordenan en orden decreciente de "importancia", el tamaño de los datos puede reducirse eliminando los componentes más débiles, es decir, aquellos con baja varianza.
5. Usando los componentes principales más fuertes, debería ser posible reconstruir una buena aproximación de los datos originales.

En la figura 14 se observa un ejemplo de selección de componentes principales en Python[6] , donde el resultado arroja que hay cuatro componentes, pero que el primero de ellos explica casi el 73% de toda la variabilidad, por lo que entre el pc1 y el pc2, son capaces de explicar más del 90% de toda la variabilidad, suficiente para capturar la información de los datos.

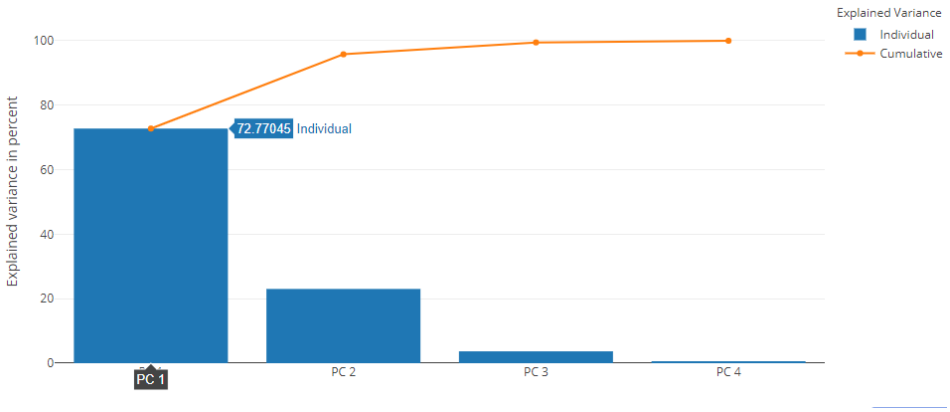

Figura 14: Ejemplo de PCA. Fuente: PCA in Python/v3

A pesar de que PCA es una técnica muy poderosa, tiene sus limitaciones. En [18] se puede encontrar una revisión de este método y otros similares (como t-SNE), para obtener la información de diferentes tipos de conjuntos de datos con alta dimensionalidad, y demostraron que no siempre PCA tiene un buen rendimiento, por lo que se debe ser cuidadoso al momento de elegir el método adecuando para reducir la dimensión de los datos.

Pero sin lugar a duda la reducción de dimensionalidad resulta ser muy útil. En [19] se hizo uso de variables sintéticas por medio de MFA (Multiple Factor Analysis), una extensión de PCA, que toma los atributos como si estuvieran estructurados por grupos, para determinar si ciertas variables (generales: peso, altura, género, etc.; huesos: L1BMD, etc.) relacionadas con los huesos, la masa muscular y la lipodistrofia para detectar anomalías corporales de personas con VIH+.

5.6 Estado del arte de herramientas usadas

Hasta ahora en este documento se han expuesto las bases teóricas del agrupamiento y el aprendizaje incremental, así como la reducción de dimensionalidad. Conocimiento necesario para encontrar la manera de agrupar rutas de deportes de exterior que es el fin último de este proyecto. Sin embargo, se debe echar un vistazo a las herramientas disponibles para: 1) buscar rutas: cómo se busca una ruta, qué variables son

[6] PCA in Python/v3: https://plotly.com/python/v3/ipython-notebooks/principal-component-analysis/

consideradas, qué ofrece al usuario; 2) visualizar archivos GPX: cómo se ven los datos, qué información ofrece; 3) Librerías de ML Incremental: qué alternativas hay, cómo operan; 4) manejo de archivos GPX en Python. Se pretende tener una idea más clara de lo que el producto final debe ofrecer al usuario.

5.6.1 Buscadores de rutas

Cómo se mencionaba en [1], las herramientas de búsqueda de rutas se han popularizado hoy en día, permitiendo encontrar rutas para correr o montar bicicleta, pero para muchos más deportes de exterior. Un ejemplo de esto es la plataforma Wikiloc[7], la cual permite buscar rutas de diversos deportes, como se ven en la figura 15, donde no sólo está corre o bicicleta, sino que incluye otros como cayac, moto Trail, todo terreno, senderismo, entre muchas otras más.

Figura 15: Ejemplo de deportes con rutas: Fuente: Wikiloc

Cuando nos adentramos en la herramienta, específicamente en el deporte de correr, se accede a una consola como la que se ve en la figura 16. En ella se observa que la información desplegada incluye: mapa indicando cada uno de los puntos y la ruta, metros de subida, bajada y planos; elevación en cada uno de los puntos, así como información de quien la "creó" y la dificultad que representa.

Figura 16: Ejemplo buscador rutas de running. Fuente: Wikiloc

Sin embargo, Wikiloc no permite filtrar por las características que se desean de una ruta, sino que despliega un listado de las rutas disponibles que haya de cada uno de los

[7] Wikiloc: https://es.wikiloc.com/

deportes o tipos de rutas, lo que puede conllevar a rutas iguales o similares, además de dificultad para encontrar la deseada.

Por su parte, **mapmyfitness**[8] permite filtrar las rutas por ciertas características como son: palabra clave, ciudad de la ruta, tipo de actividad y distancia recorrida. Así como en el caso anterior permite tanto buscar rutas como crear o subir rutas, como se ve en la figura 17.

Figura 17: Opciones de búsqueda y creación de rutas. Fuente: Mapmyfitness

Por otra parte, se encuentra la opción de **ridewithgps**[9]. Herramienta que permite filtrar rutas por: distancia, elevación, palabras clave, tipo de recreación y una segmentación sobre sobre eventos, rutas con hojas de referencia entre otras. El ejemplo de visualización se aprecia en la figura 18:

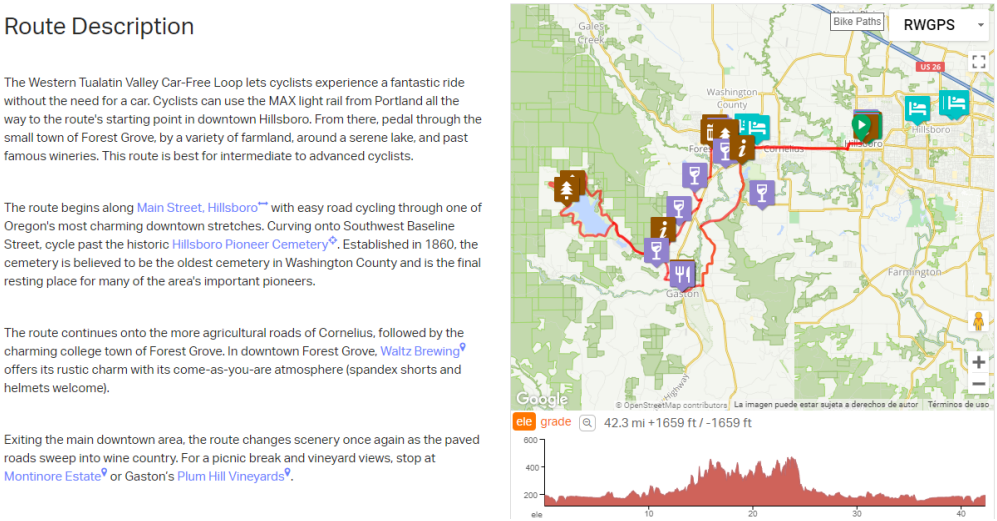

Figura 18: Ejemplo de visualización de ruta. Fuente: ridewithgps

En ridewithgps es posible tener una descripción de la ruta, el mapa con los puntos y ruta, así como indicaciones de los lugares para tomar fotos, comer o dormir, así como la elevación de cada uno de los puntos del track.

[8] Mapmyfitness: https://www.mapmyfitness.com/routes/
[9] Ridewithgps: https://ridewithgps.com/find

Se evidencia entonces que las alternativas de búsqueda son variadas, pero que aún así, ninguna ofrece una manera sencilla de buscar la ruta deseada y tampoco garantiza que dicha ruta no sea igual o similar a otra.

5.6.2 Visualizadores de archivos GPX

Muchas personas gracias a sus relojes inteligentes pueden registrar su actividad física mientras corren, lo que incluye las coordenadas de georreferenciación de los lugares que han estado durante un tiempo determinado y así crear un track. Así las personas pueden obtener archivos GPX con los datos de sus rutas y buscan una herramienta que les permita ver dichas rutas y obtener la información que les brinda.

Hay varias opciones y entre ellas se encuentra googlemaps[10]. Esta herramienta no sólo sirve para encontrar lugares o sugerir la ruta de un punto A, a un punto B, sino que además permite abrir archivos GPX como se observa en la figura 19.

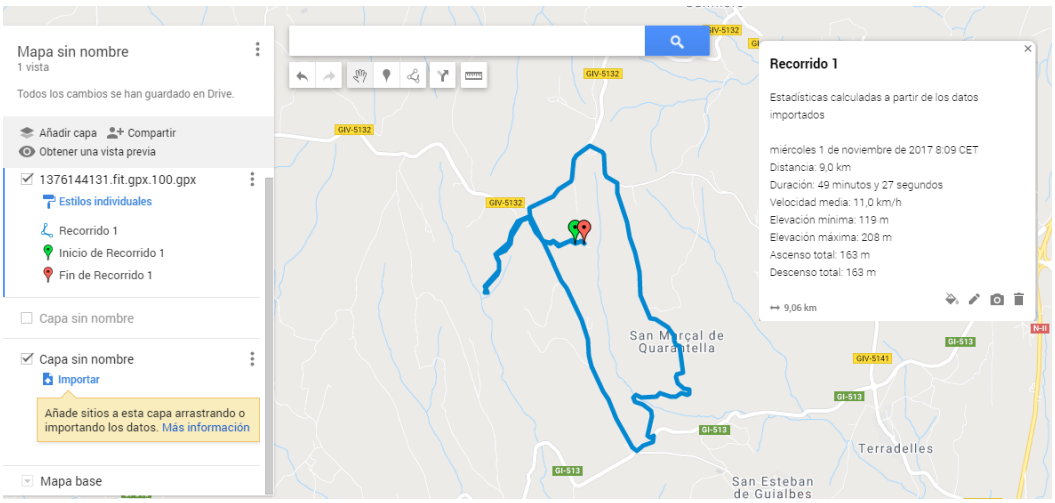

Figura 19: Visualización de archivo GPX con googlemaps. Fuente Autor

Con googlemaps, sólo basta con ir a *Tus sitios\mapas\crear mapa* para visualizar el archivo GPX disponible. Se desplegará la información que se ve en la figura 19 indicando punto de partida, llegada, ruta en información del recorrido como:

- Distancia: dada en kilómetros.
- Duración: dada en minutos con segundos.
- Velocidad media: km/h
- Elevación mínima: metros
- Elevación máxima: metros
- Ascenso total: metros
- Descenso total: metros

Variables que en definitiva serán de mucha utilidad para determinar si una ruta se parece a otra o no.
Otra alternativa de visualización es gpxviewer, el cual se puede apreciar en la figura 20, usando el mismo archivo GPX utilizado en el ejemplo de Googlemaps.

[10] Googlemaps: https://www.google.com/maps

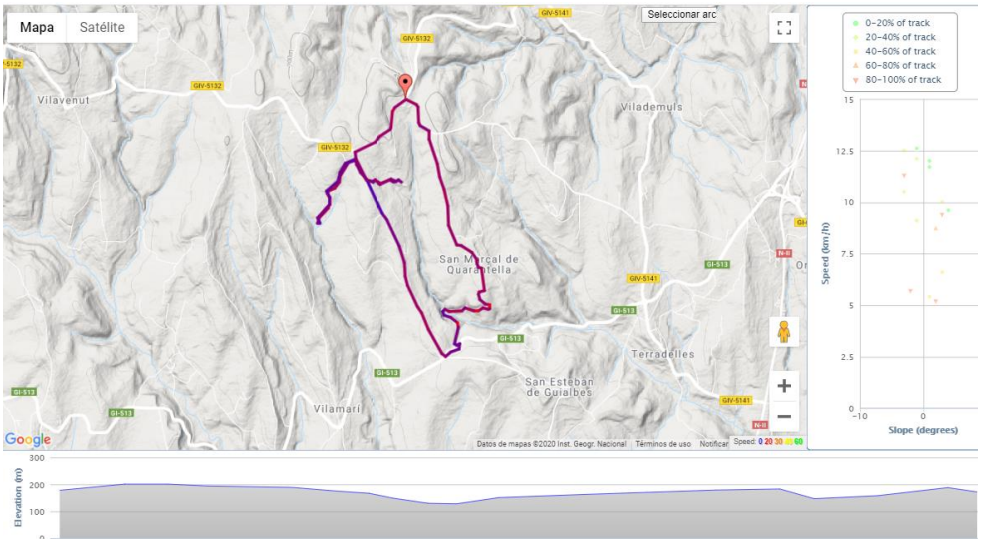

Figura 20: Visualización de archivo GPX con gpxviewer. Fuente: Autor

Esta opción permite observar la elevación de cada uno de los puntos de la ruta, pero no permite visualizar la información estadística de la ruta como distancia, elevación mínima, máxima, etc., que si se puede ver con googlemas. Es por esto por lo que lo ideal sería una herramienta que permitiera tener los mejor de estas dos herramientas para así aprovechar toda la información disponible en los archivos GPX.

5.6.3 Librerías para ML con incremental learning

Cómo se ha podido ver en la sección de aprendizaje incremental, la ventaja de esta forma de aprendizaje es que permite entrenar un modelo sin necesidad de usar toda la data de entrenamiento a la misma vez, de hecho, se puede hacer en forma de batch (lotes), o instancia por instancia (streaming), lo cual se vuelve idóneo para cuando se trabaja con set de datos demasiado grandes como en el caso del uso de imágenes.
Las redes neuronales son una forma de aprendizaje incremental [27], sin embargo, no significa que siempre se deban usar y que sean la mejor herramienta para todos los problemas, así el aprendizaje incremental permite hacer uso de muchas técnicas de ML diferentes a las redes neuronales de manera incremental y superando las limitaciones de memoria RAM.
En [27] se presenta un tutorial de como desarrollar un modelo de clasificación de imágenes de perros y gatos, haciendo uso de las librerías para Python como keras y Creme[11]. Esta última permite desarrollar modelos “online machine learning”, por lo que puede ser usada para data en streaming. Es una librería similar a scikitlearn, la cual también permite desarrollar modelos de esta manera, pero con implementaciones mucho más complejas y difíciles de desarrollar.
En este tutorial se observa como es posible usar un set de datos de 15.000 imágenes en un pc convencional y obteniendo un accuracy del 97% después de seis horas y media aproximadamente.

[11] Creme: https://creme-ml.github.io/

Esta es una herramienta idónea para realizar benchmarking con el algoritmo a desarrollar en este presenta proyecto.

5.6.4 Archivos GPX en Python

Por último, se toca el tema del manejo de datos en formato GPX en Python, ya que como el aplicativo será desarrollado en este lenguaje, se debe crear un procedimiento para leer y manipular toda la información que este tipo de formatos semiestructurados ofrece. En [28] se presenta una manera de plotear el track, la velocidad y la elevación de las rutas que sean cargadas. Para esto requiere de Python 3, y las librerías: matplotlib.pyplot, time, IPython.display, xml.dom. minidom y math. En la figura 21 se observa como las rutas y su información serían visualizadas.

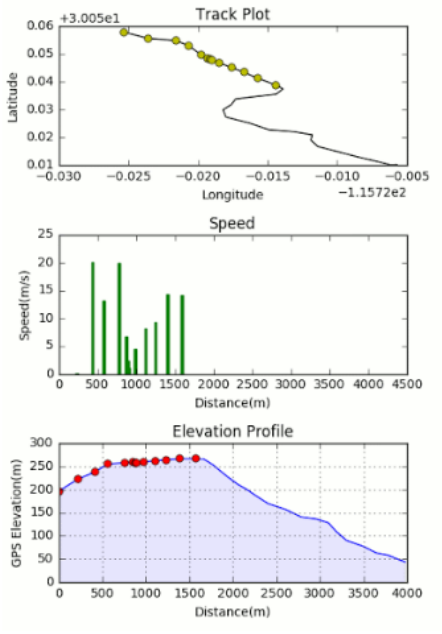

Figura 21: Ejemplo de visualización de rutas con Python. Fuente: [28]

Así mismo en [29] se presenta otro tutorial donde ilustra la manera de cargar datos GPX en python, parsearlos y guardarlos en una variable, a la cual se puede acceder fácilmente a su información. En este ejemplo se utilizan las librerías: gpxpy,*matplotlib*, *datetime*,*geopy*,*math*, n*umpy*,*pandas*,*haversine y Plotly* (esta última para líneas interactivas sobre mapas en 3d).

En [30] es posible ver un ejemplo de como graficar mapas de Googlemaps haciendo uso de una librería de Python gmplot. Esta es una interfaz similar a matplotlib que genera el HTML y JavaScript para representar todos los datos que el usuario quisiera sobre Google Maps .

Lo que se ha podido evidenciar es que las opciones son muchas y variadas para diferentes casos. Estas son las opciones base, pero a medida que se fue codificando el aplicativo algunas de estas fueron usadas, otras descartadas según la necesidad que se presentó en el camino.

6. Desarrollo del algoritmo y experimentación

A continuación, se detalla el proceso de desarrollo del algoritmo y los experimentos realizados para determinar su mejor configuración. Se sigue el proceso que se detalla en el apartado 4, planificación de trabajo.

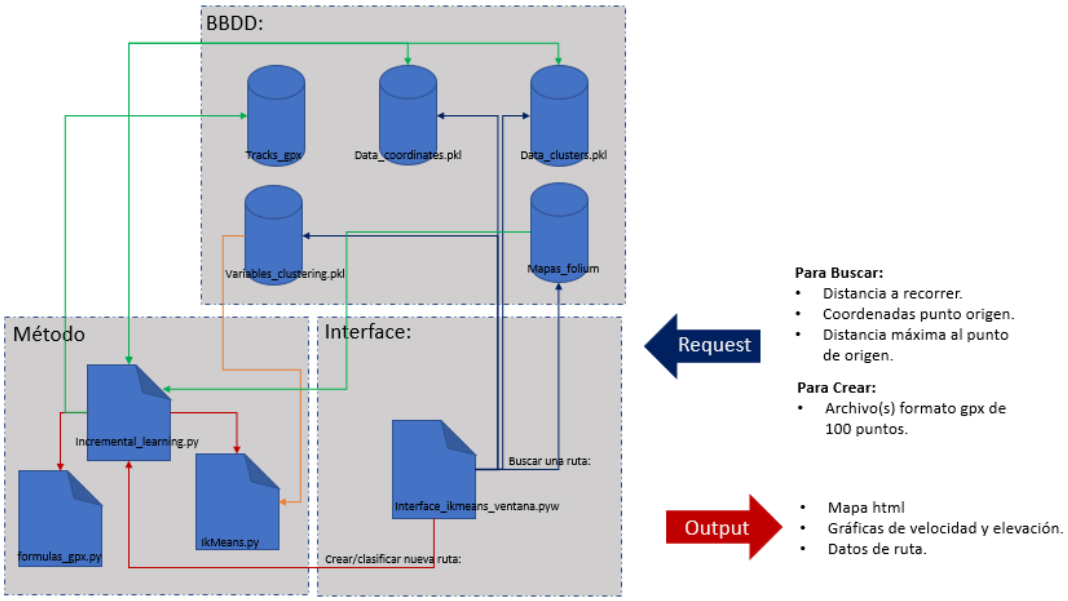

Figura 22: Modelo conceptual del software a desarrollar. Fuente: Autor

En la figura 22 se plantea el modelo conceptual de como está desarrollado el aplicativo. Se divide en 3 grandes segmentos: Interfaze, método y BBDD. En el primer segmento se encuentra el código necesario para capturar las solicitudes del usuario, consultar la información en las bases y solicitar al algoritmo de clustering que se ejecute. En el segmento de método, se encuentra todo lo requerido para que el agrupamiento de las rutas se realice, considerando además los procesos para manipular data semiestructurada en formato .gpx. En el segmento de BBDD se encuentran los archivos que son utilizados para guardar información relevante en el momento del agrupamiento, ya sea para realizar nuevos agrupamientos o para visualizar mapas de las rutas, o para graficar información que será desplegada en la interfaz. La estructura de ficheros es como la que se ve en la figura 23, donde la información se organiza en 3 carpetas y 4 archivos.
A continuación, se especifica la finalidad de cada fichero:

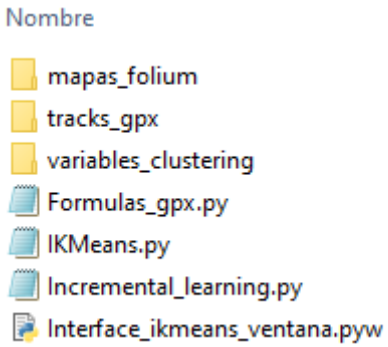

Figura 23: Lista de ficheros software

- **IKMeans.py:** Es el algoritmo propiamente de Incremental Clustering, el cual recibe como input un conjunto de puntos que son los centroides de las rutas, y el threshold denominado Thr, el cual establece qué tan cerca deben estar dos rutas para considerarse dentro del mismo clúster (en este caso 150 metros, 0.15). El pseudo código del algoritmo se encuentra en el apartado 5.4, en la figura 13.

- **Formulas_gpx.py:** Script que contiene las fórmulas requeridas para procesar los archivos en formato .gpx y tratar información incompleta, como, por ejemplo, el dato de elevación de algunas rutas.

- **Incremental_learning.py:** Es el script orquestador de todo el proceso. Es el que se encarga de capturar la información de las rutas de la carpeta "tracks_gpx", y procesarla por medio de Formulas_gpx.py, solicitar a IKMeans.py crear los clusters y crear los mapas para posteriormente guardarlos en formato .html en la carpeta denominada "mapas_folium".

- **Interface_ikmeans_ventana.pyw:** Este script contiene todo el código con el que fue creada la interfaz gráfica. Se encarga de permitirle al usuario interactuar con el algoritmo, los mapas, realizar búsquedas o crear/clasificar nuevas rutas.

Carpeta tracks_gpx: En ella se encuentran las rutas gpx que se desean agrupar en un momento determinado. Es de donde se alimenta el aplicativo.

Carpeta mapas_folium: En esta carpeta se almacenas todos los archivos formato .html que contiene el mapa de la ruta representante de cada cluster. Habrá un único mapa por cluster porque se espera que todas las rutas que lo componen sean similares.

Carpeta variables_clustering: Esta carpeta contiene 3 archivos .pkl que son el resultado de haber realizado el agrupamiento:

- **variables_clustering.pkl:** En este archivo contiene toda la información que se obtiene como resultado de aplicar el incremental clustering. Contiene las particiones, los centroides de cada cluster, la asignación de cada ruta, las métricas de validación, más específicamente las que se mencionan en las fórmulas (14) y (15).
- **data_cluster.pkl:** Así como hay un solo mapa de ruta para cada cluster, también se conserva únicamente un set de datos para cada cluster. Dicho set es el que se almacena en este archivo.
- **data_coordinates.pkl:** Este archivo conserva las coordenadas (lat, lng) promedio de cada cluster para que en procesos posteriores la librería folium sea capaz de ubicar la ruta en el mapa.

<u>Representación de una ruta:</u>
El primer inconveniente que se presenta al querer agrupar rutas es que estas son la consecución de varios puntos (100 puntos) que forman un trayecto. Cualquiera de los 100 puntos puede ser considerado inicio o fin de la trayectoria, por lo que comparar punto a puntos trayectorias no tienen sentido, porque a pesar de que fueran la misma ruta, si el punto inicial y final es diferente, cuando se quiera realizar la diferencia entre sus puntos, el resultado será que están muy alejados, porque la distancia claramente será en términos absolutos.

Así que para saber si una ruta está o no cerca a otra se decide hacer uso de sus centroides, o centros geométricos, ya que con ellos no importará la posición que ocupen sus puntos de inicio y fin, y además será posible representar cada ruta con un punto con dos coordenadas.

En la figura 24 se observa un ejemplo de una ruta y su centroide:

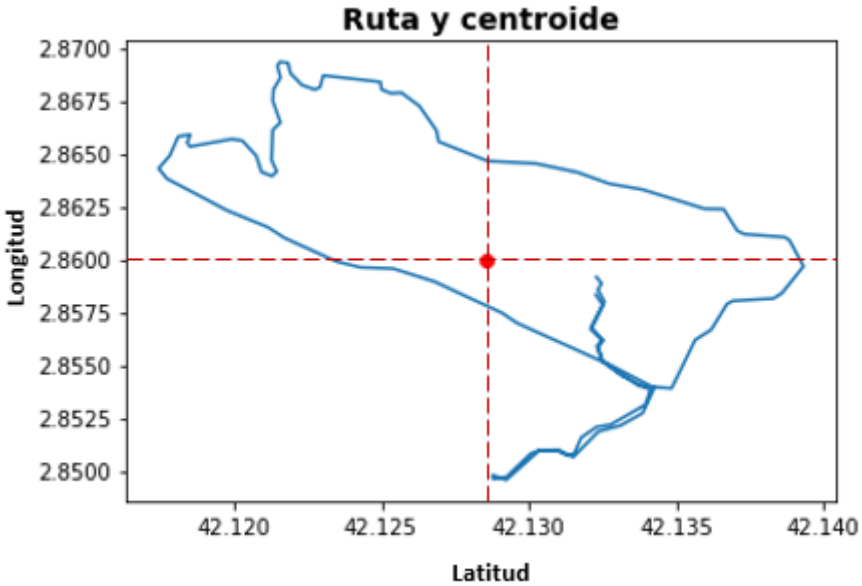

Figura 24: Gráfica de ruta y su centroide. Fuente: Autor.

El cálculo se realiza obteniendo el promedio de sus coordenadas de latitud y longitud. También se consideró la posibilidad de que dos rutas con centroides cercanos tuvieran una forma completamente diferente. Sin embargo, después de probar con muchas rutas, al parecer las condiciones topográficas de una zona, hace que las opciones de variación de las rutas no sean significativas, es decir, qué si una persona entrena en una zona determinada, lo más probable es que siga un sendero o ruta que la misma topografía le indica, como puede ser por ejemplo, un parque, donde la gente lo que haría es recorrer la periferia del mismo, en lugar de moverse entre extremos del parque. Esto también se sustenta considerando que precisamente el propósito de este trabajo es agrupar rutas, ya que como las personas realizan un mismo trayecto, o muy similar en un área, se requiere eliminar todos aquellos duplicados.

<u>Implementación del incremental k-means:</u>
En el fragmento de código 1, se detalla la implementación en Python 3.7 del pseudo código presentado anteriormente en la figura 13. Es importante anotar que el algoritmo recibe como insumo, una lista de coordenadas (los centroides) de cada ruta, así como el Thr (Threshold):

```
#Conjunto de particiones:
P=[[]]
#Conjunto de centroides:
C=[]
#Conjunto de indices de tracks de cada partición:
P_nombres=[]
#Indice de Calisnky&Harabasz
index_cyh=0
#Indice de Hartigan
index_hart=0
#Empezar con la agrupacion:
k=0
for k in range(len(D)):
    if k==0 and len(P)==1: #si no hay clusters, cree el primero:
        P[k].append(D[k])
        C.append(D[k])
        P_nombres.append((k,len(P)-1))
    else:
        dist_list=[] #calcular la distancia con todos los centroides disponibles:
        for i in range(len(C)):
            dist_list.append(calculardistancia(D[k],C[i]))
        #Elegir el centroide de minima distancia:
        delta = np.min(dist_list)
        index= dist_list.index(delta)

        if delta <= Thr: #si la distancia mínima es menor o igual al Thr agrupelo en el cluster de minima distancia
           P[index].append(D[k])
           #Recalcular el centroide de la particio i:
           C[index]=calcularcentroide(P[index])
           P_nombres.append((k,index))
        else:
           #Se crea un nuevo cluster o particion:
           P.append([D[k]])
           C.append(D[k])
           P_nombres.append((k,len(P)-1))

#Calculo de Indices:
index_cyh= np.nanmax([(ssb(P,C)/(len(P)-1))/(ssw(P,C)/(calcularn(P)-len(P))),0])
index_hart= np.log(ssb(P,C)/ssw(P,C))
#Guardar el diccionario a usar:
joblib.dump((P,C,index_cyh,index_hart,P_nombres), os.path.join(os.getcwd(),carpeta,'variables_clustering.pkl'))
return P,C,index_cyh,index_hart,P_nombres
```

Fragmento de Código 1: Implementación Incremental k-means.

6.1 Selección de variables y PCA

La selección de variables es un paso sumamente importante al momento de desarrollar modelos de machine learning, ya que dependiendo de la información que se le suministre y la relación que estas tengan con la variable de salida o el objetivo que se quiera alcanzar, el resultado será diferente.

Tratamiento de valores perdidos:

Uno de los primeros problemas al adquirir la data de los archivos .gpx, es que varios de sus datos se encontraban vacíos (NaN). Así que se desarrolló el siguiente método expuesto en el fragmento de código 2 para tratarlos. No se quiso hacer uso de otras técnicas como el promedio de todos los valores de la serie, o el uso de k-mean para valores perdidos, sino algo mucho más ajustado al problema específico. Se obtuvo el promedio, pero entre el valor anterior y posterior, cuando el vacío se encontraba en la mitad, o del valor anterior o posterior, cuando era posible. Así mismo, toma en consideración si la serie es float o es integer. Cuando es integer lo único que hace es aproximar el valor a un entero.

```
#metodo de imputacion de valores nulos:
def imputarnan(lista):
  import numpy as np
  lista_final=[]
  valor=0
  if np.mean(lista).is_integer() == False:
        for i in range(0,len(lista)):
             if np.isnan(lista[i])== True:
                if i <len(lista)-1:
                   if i >1:
                      if np.isnan(lista[i+1])== False:
                         valor = np.mean([lista_final[i-1],lista[i+1]])
                      else:
                         valor = np.mean([lista_final[i-1],lista_final[i-2]])
                   elif i>0:
                      if np.isnan(lista[i+1])== False:
                         valor = np.mean([lista_final[i-1],lista[i+1]])
                      else:
                          valor = lista_final[i-1]
                   else:
                        if np.isnan(lista[i+1])== False:
                           valor = lista[i+1]
                        else:
                            valor = np.mean(lista)
                else:
                     valor= lista_final[i-1]
             else:
                 valor = lista[i]

             lista_final.append(valor)

        return lista_final
  else: #Si el valor es un int
        for i in range(0,len(lista)):
             if np.isnan(lista[i])== True:
                if i <len(lista)-1:
                     if i >1:
```

Fragmento de Código 2: Algoritmo para tratar valores NaN.

<u>Pasando de variables 3D a variables 2D:</u>

Dado que se calcula la velocidad en este trabajo, era requerido encontrar la distancia precisa entre punto y punto, pero para ello se debía obtener unas coordenadas cartesianas que permitan hacer este cálculo, teniendo en consideración la curvatura de la tierra. Así que con base en el ejemplo de [28] se hizo uso de la conversión de coordenadas geodésicas a coordenadas “ECEF coordinates". Esto permite convertir coordenadas (lat,lng) a coordenadas "planas" (x,y) con las que ya se puede calcular distancias, y estas últimas quedan en metros, por lo que tienen un significado directo.
En el fragmento de código 3 se observa la implementación de dicha conversión.

```python
#Transformar las coordenadas de 3d a 2d para calcular la distancia cartesiana:
def coordenadas2d(list_lat,list_lng,list_alt):
    import math
    a=6378137 #WGS 84 Major axis
    b=6356752.3142 #WGS 84 Minor axis
    e2=1-(b**2/a**2)
    list_x=[]
    list_y=[]
    for i in range(0,len(list_lat)):
        n=float(a/math.sqrt(1-e2*(math.sin(math.radians(abs(list_lat[i])))**2)))
        x=(n+list_alt[i])*math.cos(math.radians(list_lat[i]))*math.cos(math.radians(list_lng[i]))
        y=(n+list_alt[i])*math.cos(math.radians(list_lat[i]))*math.sin(math.radians(list_lng[i]))

        list_x.append(x)
        list_y.append(y)

    return list_x, list_y
```

Fragmento de Código 3: Cambio de coordenadas 3D a 2D.

Primera aproximación con PCA:

Cómo se mencionaba en la sección de revisión bibliográfica y marco teórico, la reducción de dimensionalidad es un aspecto para explorar cuando se desean representar rutas. Esto porque en una ruta intervienen distintas variables, como: distancia, velocidad (que implica la distancia y tiempo entre punto y punto), latitud, longitud y altura.

El proceso de carga de información se hace de manera incremental, es decir, no se cargan todos los archivos .gpx en memoria de una sola vez, sino que se va cargando uno a uno, se obtiene la información que se requiere, y se carga otro, dejando al anterior libre. Esto porque el algoritmo está diseñado para que pueda ser utilizado en streaming o en entornos Big Data, donde cargar toda la información en RAM en el momento cero no sea posible, y se requiera procesarla de manera incremental.

Después de cargar el archivo .gpx, obtener su data, limpiarla, llenar los valores vaciós, se obtiene la información que se requiere, y para este caso son los dos componentes principales. En el fragmento 4 se observa la parte del proceso en la que se estandariza la información limpia de la ruta que está siendo procesada en ese momento, se obtienen sus dos componentes principales y finalmente se obtiene su centroide, para que sea representada por un punto. Se guarda esta información en la lista denominada "data_k" y se procede con la siguiente ruta.

```python
#Uso de PCA para reducir la dimensionalidad:
from sklearn.decomposition import PCA
from sklearn import preprocessing
data_pca= data[['lat','lng','x','y','distancia_acum','tiempo_acum','speed_prom']].copy()
data_pca_std = preprocessing.scale(data_pca) #estandarizacion de las variables
componts=PCA(n_components=2)
componts.fit(data_pca_std)
trans=componts.transform(data_pca_std) #Aplica la reducción de dimensionalidad
print (componts.explained_variance_ratio_) #Varianza explicada por los componentes

centroide_track = (np.mean(trans[:, 0]),np.mean(trans[:, 1]))
data_k.append(centroide_track)
```

Fragmento de Código 4: Estandarización de variables y PCA en Python.

En la gráfica 25 se observa un listado ejemplo de la varianza capturada por los dos componentes principales de cada ruta. Cómo se observa el valor es significativo, ya que, sumando los dos componentes de cada una, se obtiene un promedio que va desde 0.8 hasta 0.95.

```
[0.4904777  0.30278053]
[0.47833291 0.3082537 ]
[0.42378978 0.33406123]
[0.48833591 0.25999995]
[0.51914774 0.23702528]
[0.44562563 0.3409889 ]
[0.59059042 0.31078609]
[0.46805265 0.40207396]
[0.76115615 0.19835625]
[0.4309461  0.29481379]
[0.47366265 0.3116216 ]
[0.44945658 0.40962774]
[0.56997789 0.22756565]
[0.75372073 0.16483868]
[0.47618316 0.25778445]
[0.5619678 0.2869481]
```

Índice	Tipo	Tamaño	Valor
0	tuple	2	(-4.440892098500626e-18, -2.220446049250313e-18)
1	tuple	2	(-1.7763568394002505e-17, -6.88338275267597e-17)
2	tuple	2	(-7.549516567451065e-17, 1.7763568394002505e-17)
3	tuple	2	(3.1086244689504386e-17, 5.273559366969494e-18)
4	tuple	2	(-5.773159728050814e-17, -1.8873791418627663e-17)
5	tuple	2	(3.552713678800501e-17, -2.2204460492503132e-17)
6	tuple	2	(2.2204460492503132e-17, 8.881784197001253e-18)
7	tuple	2	(1.9984014443252817e-17, 2.6645352591003756e-17)
8	tuple	2	(1.7763568394002505e-17, -6.106226635438361e-18)
9	tuple	2	(-5.773159728050814e-17, 4.829470157119431e-17)
10	tuple	2	(-1.9984014443252817e-17, 3.552713678800501e-17)

Figura 25: Componentes principales y valor de tuplas de centroides.

Hasta este punto, todo iba bien, sin embargo, al momento de revisar los datos obtenidos de sacar el promedio de los dos componentes principales, se observa que sus valores son sumamente pequeños (muy cercanos a cero). En la figura 25 se presenta un ejemplo de las tuplas obtenidas. Esto hace que se pierda la noción de lo que es algo “grande”, “distante” o “pequeño”, lo cual es problemático, considerando que el modelo de clustering incremental precisa de un threshold como parámetro de entrada que determina si dos rutas están o no cerca una a la otra, y si no se tiene noción de las unidades en las que esa distancia está, será muy difícil determinar un Thr adecuado.

<u>Una consideración adicional:</u>
El lector podría preguntarse por qué en lugar de aplicar PCA a cada set de datos de cada ruta, por qué no obtiene primero los valores medios de cada variable de cada ruta, junta esta serie de datos, los estandariza y finalmente hace PCA obteniendo el valor del componente de cada ruta. La respuesta de por qué no se hace es porque cómo se mencionó líneas arriba, este es un algoritmo diseñado para ser procesado de manera incremental, lo que se traduce en que no se tendrá toda la información en el tiempo cero, por tal razón, la posibilidad de realizar una estandarización no es factible, ya que esto se realiza partiendo de la media del set de datos. Por otra parte, cabe la posibilidad de sólo querer clasificar una ruta, y PCA no se puede realizar con un set de datos con varias columnas y una sola fila.

<u>Solucionando el problema de unidades desconocidas:</u>
Por un lado, se desea obtener la mayor información posible de todas las variables que el set de datos puede ofrecer, y por el otro se desea reducir la dimensionalidad sólo a dos componentes, de tal manera que sea posible representar una ruta como un punto sobre un plano cartesiano.

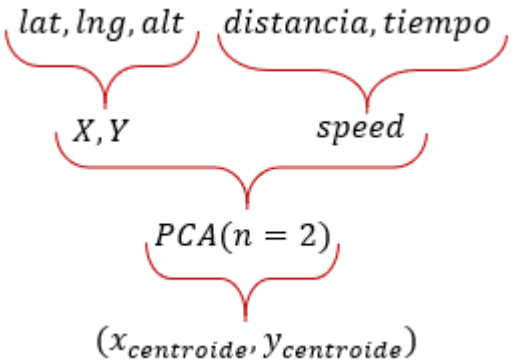

Figura 26: Relación entre variables.

En la figura 26 se ilustra la lógica detras de la selección de variables para resolver el problema de dimensionalidad y no perder información. A esta altura se conocía la latitud, longitud, distancia y tiempo de cada uno de los puntos de una ruta. Además, se tenía información de las coordenadas planas X,Y, que precisamente habían permitido calcular dicha distancia.

Así que se trató de inferir aquellas variables que ya contenían información de otras. Las variables X, Y habían sido calculadas con base a la latitud, longitud y altura de cada punto, por lo que ya contenían información de estas variables. Por su parte la información de distancia y tiempo ya estaba contenida en la variable speed. Así que se había pasado de tener 5 variables, a sólo 3 con la misma información, pero concentrada. Estas últimas 3 eran las candidatas para realizar PCA, no sólo porque contenían la mayor cantidad de información, sino que es importante anotar que todas están en las mismas dimensiones, es decir, en metros.

<u>Segunda aproximación con PCA:</u>

Con las variables anteriormente elegidas se procede a hacer una segunda aproximación con PCA que se expresa en el fragmento de código 5. Para solventar el problema de las unidades sumamente pequeñas causadas por la estandarización y el PCA, se decide multiplicar los centroides por una cifra con 17 ceros para que el resultado al menos este en el rango entre -5 y 5.

```
#Uso de PCA pero considerando X,Y,speed
data_pca= data[['x','y','speed_prom']].copy()
data_pca_std = preprocessing.scale(data_pca) #estandarizacion de las variables
componts=PCA(n_components=2)
componts.fit(data_pca_std)
trans=componts.transform(data_pca_std) #Aplica la reducción de dimensionalidad
print (componts.explained_variance_ratio_) #Varianza explicada por los componentes

centroide_track = (np.mean(trans[:, 0])*10**17,np.mean(trans[:, 1])*10**17)
data_k.append(centroide_track)

lista_nombres.append(list_archivos[p])
```

Fragmento de Código 5: Segunda aproximación de PCA con variables elegidas.

Se realizaron pruebas con **213 archivos .gpx válidos** (que tuvieran segmento y 100 puntos, así como que ninguna columna de latitud, longitud o altura estuviera vacía), corriendo el algoritmo con diferentes valores de Thr. Se probaron varios porque en este punto se sigue teniendo el problema de que no se sabe qué unidades manejar, además considerando que se multiplicaron los centroides por 10^17 para tener al menos cifras manejables.

Se observa que el algoritmo tiende a agrupar sólo aquellas rutas que presentan unos centroides exactamente iguales (archivos .gpx iguales con diferente nombre), pero no es capaz de agrupar rutas similares.
En una de las iteraciones con Thr=0.3 se logra obtener clústeres de rutas que no son exactamente iguales y que el algoritmo considere semejantes. En la figura 27 se observa la comparación de estas 3 rutas.

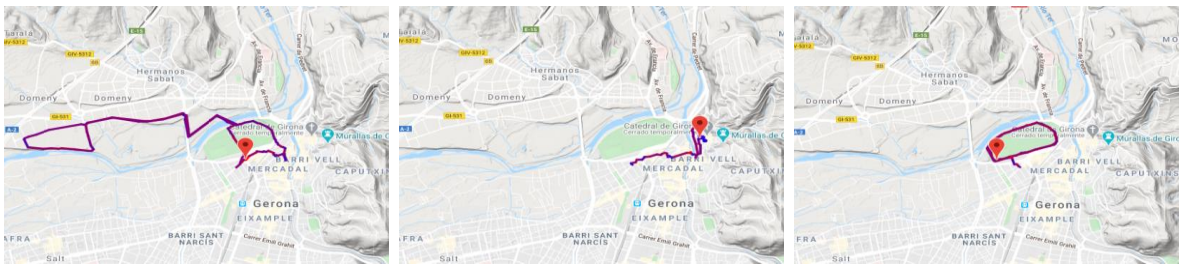

Figura 27: Comparación de mapas de rutas.

Claramente a pesar de que las rutas están cerca a la misma zona (parque de la Devesa), no son rutas similares. Así que el algoritmo con esta selección de variables no logra obtener un buen resultado, además que se vuelve muy subjetivo determinar si una ruta es similar a otra por medio del Threshold cuando no se conocen las unidades de las cifras, perdiendo así la autonomía del agrupamiento incremental.

<u>Selección final de variables:</u>
Posterior a lo anterior y sabiendo que PCA no es la solución, se prueba con diferentes combinaciones lineales para conseguir unas coordenadas de centroides que tengan sentido y contengan toda la información, pero se llega a conclusiones donde las rutas están distantes geográficamente, pero al tener unas condiciones de distancia, tiempo, velocidades similares, el algoritmo considera que son similares, pero en realidad no lo son. Esto daría razón a por qué en la segunda aproximación con PCA no funcionó, y es que en ese caso se le daba el mismo peso a la velocidad, que el que se le daba a las **variables X,Y**, haciendo que dos rutas se distanciaran a pesar de ser similares, o acercaran a pesar de ser diferentes, esto porque se pudo evidenciar, que rutas claramente similares, presentaban valores muy diferentes de velocidad, y altura. Esto se debe principalmente a que no todos los días el atleta recorre la ruta en el mismo tiempo, o son rutas recorridas por atletas diferentes con suficiente diferencia entre sus rendimientos. Es por esto por lo que considerar todas las variables en este caso, no resulta ser de utilidad y al contrario se genera mayor entropía.
Con base en lo anterior, se decide entonces hacer uso únicamente de dos variables: las coordenadas planas X,Y ignorando así el resto de las variables. Esto se hace considerando el supuesto que ya se discutió líneas arriba en el apartado 6, representación de una ruta, donde se plantea el supuesto que la trayectoria en una zona determinada esta dada en gran parte por la misma topografía del lugar, así que, si los centroides de un par de rutas son lo suficientemente cercanos, se puede deducir entonces que las rutas serán lo suficientemente similares.
Por otra parte, otra ventaja de hacer uso de las coordenadas X,Y es que las unidades si tienen sentido, porque la distancia entre centroides estará dada en metros, o en kilómetros si se multiplican por la cifra adecuada.

6.2 Indicadores de rendimiento de clustering

Los indicadores de rendimiento en problemas de clustering son sumamente importantes, ya que, al ser aprendizaje no supervisado, no hay etiquetas de comparación que permitan establecer el grado de exactitud, precisión o sensibilidad del modelo. En cambio, se tienen métricas que permiten medir qué tan lejos se encuentran los clústeres entre ellos, y qué tan cercanos se encuentran los elementos dentro de cada clúster. Para este caso se utilizaron dos indicadores de rendimiento, mencionados en la sección de marco teórico, llamados Calinski & Harabasz y Hartigan[21], representados en las fórmulas (14) y (15) respectivamente.

<u>Probando los indicadores de rendimiento con data sintética e incremental clustering:</u>
Para saber si los indicadores elegidos en realidad son adecuados para medir la fiabilidad de un agrupamiento, se realiza un experimento con data sintética (generada en Python), donde se conoce de antemano claramente cual es el número de clusters óptimo. Se aplica el algoritmo de incremental kmean a este set de datos, y se captura la información de estos dos indicadores, así como el valor de thr y número de k clústeres en cada iteración para graficar posteriormente su comportamiento.
Para crear los clústeres se hace uso de las librerías y código expuesto en el fragmento de código 6, donde se hace uso de 1500 puntos, agrupados en 7 clusters.

```
import random
from sklearn import datasets
#Crear nubes de puntos sintéticos:
N_SAMPLES = 1500
MIN_CLUSTERS = 6
MAX_CLUSTERS = 8
x_sintetica, y_sintetica = datasets.make_blobs(n_samples=N_SAMPLES, # Cuantos puntos
                           n_features=2,        # Cuantas dimensiones
                           centers=random.randint(MIN_CLUSTERS, MAX_CLUSTERS),
                           center_box=(-20, 20),
                           cluster_std=.6)
print('Hay {} puntos en {} dimensiones (repartidos en {} clusters)'.format(x_sintetica.shape[0],
      x_sintetica.shape[1], y_sintetica.max() + 1))
fig, ax = plt.subplots(1, 1, figsize=(4, 4))
ax.scatter(x_sintetica[:,0], x_sintetica[:,1], s=2,c=y_sintetica)
ax.axis('equal')
plt.tight_layout()
```

Fragmento de Código 6: Creación de clusters con data sintética.

```
Hay 1500 puntos en 2 dimensiones (repartidos en 7 clusters)
```

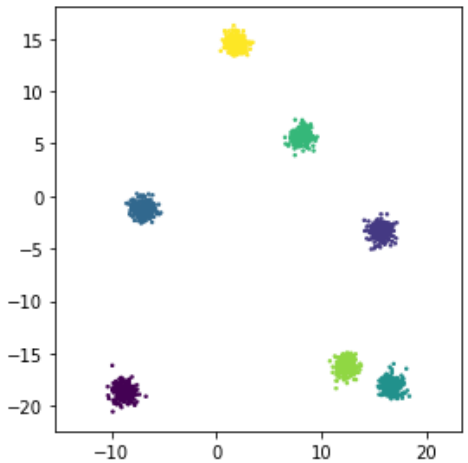

Figura 28: Clusters sintéticos generados.

En la figura 28 se observa cómo son creados los 7 clústeres y cada uno es claramente disjunto al resto.

Posteriormente se procede a iterar el algoritmo de incremental clustering, iniciando con un Thr= 0.7 durante 60 iteraciones, aumentando 0.2 en cada iteración. Los resultados son los que se presentan a continuación:

El primer análisis consiste en comparar el número de clústeres versus el valor del threshold, y cómo se observa en la figura 29, claramente a medida que el threshold es mayor, en número de k clústeres es inferior. Esto tiene todo el sentido considerando que a medida que el Thr se vuelve más grande, será mayor el número de puntos que estarán entre el centroide del clúster más cercano y él mismo.

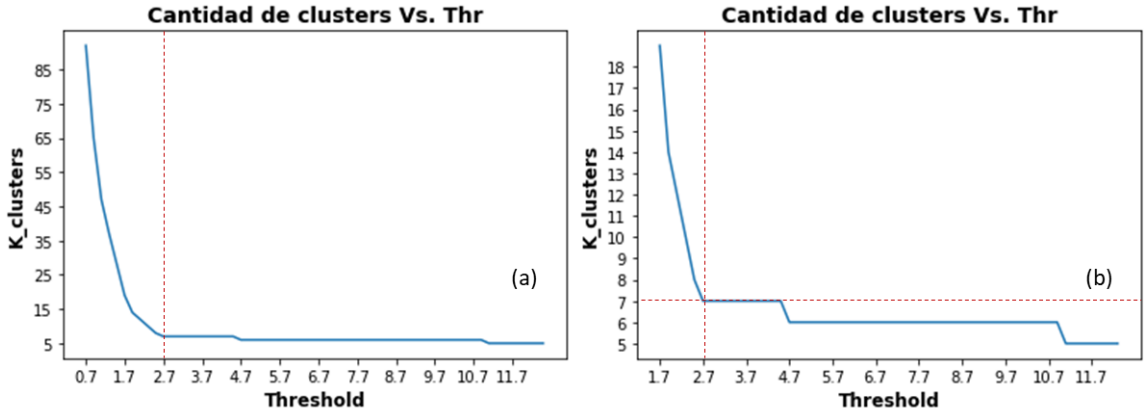

Figura 29: Cantidad de clusters versus Thr. Fuente: Autor.

Por otra parte, es interesante anotar, que al parecer se crea un comportamiento entre el k y Thr, que en este ejemplo se ve representado en un valor muy cercano a 2.7 de Thr, ya que en ese punto se genera todo un cambio de comportamiento de la curva, algo similar a lo que ocurre con el método del codo.

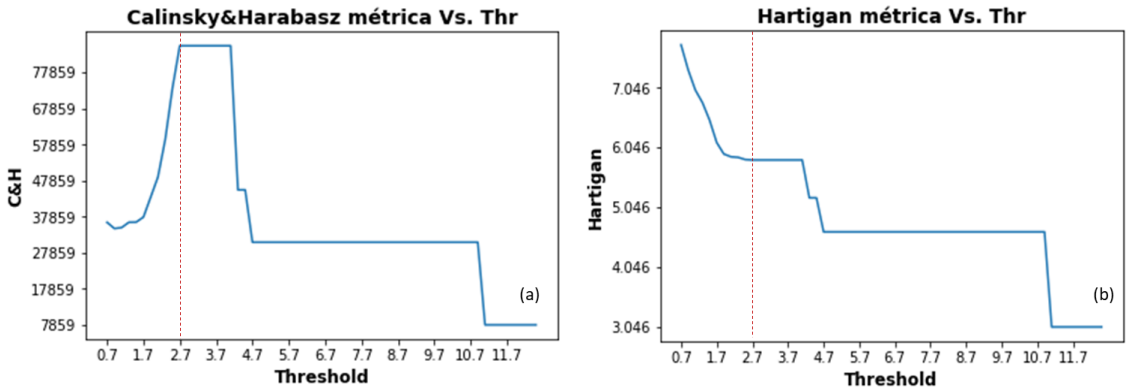

Figura 30: Indicador C&H y Hartigan versus el Thr.

En la figura 30, se presentan dos curvas, en la sección (a), se presenta el indicador de Calinsky &Harabasz versus el valor de Thr y en la sección (b), se presenta el indicador de Hartigan versus el Thr. Es claro que el primer indicador es un poco más claro, considerando que se podría pensar en que cuanto más alto es su valor, mejor será el resultado. Sin embargo, el segundo indicador no es tan claro, porque pareciera que sencillamente disminuye a medida que el Thr aumenta. Sin embargo, cuando se traza

una línea vertical que pasa por el Thr=2.7, se observa en los dos casos que algo particular ocurre. Para el primer indicador en ese valor, la métrica es máxima, y el segundo indicador, este es el punto donde por primera vez se estabiliza la métrica.
Dadas las 3 gráficas anteriores algo parece indicar que el Thr adecuado para este ejemplo sintético es Thr=2.7 . Sin embargo, aún falta analizar el comportamiento del número de clústeres.

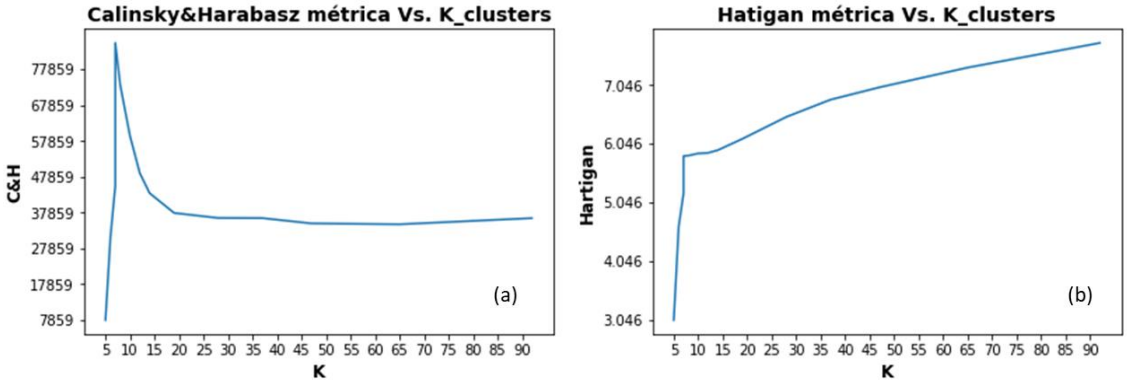

Figura 31: Indicares de C&H y Hartigan Vs. número de clústeres K.

En la figura 31, se comparar los dos indicadores versus el número de clusters. De nuevo, el indicador de C&H presenta un máximo y luego cae, mientras que el de Hartigan sigue con un comportamiento monótono, con excepción de un punto entre 5 y 10 del valor de K, que es donde presenta un punto de inflexión.

Para ver más claramente los valore que presentan punto de inflexión se reduce el rango de K, y se presenta la figura 32:

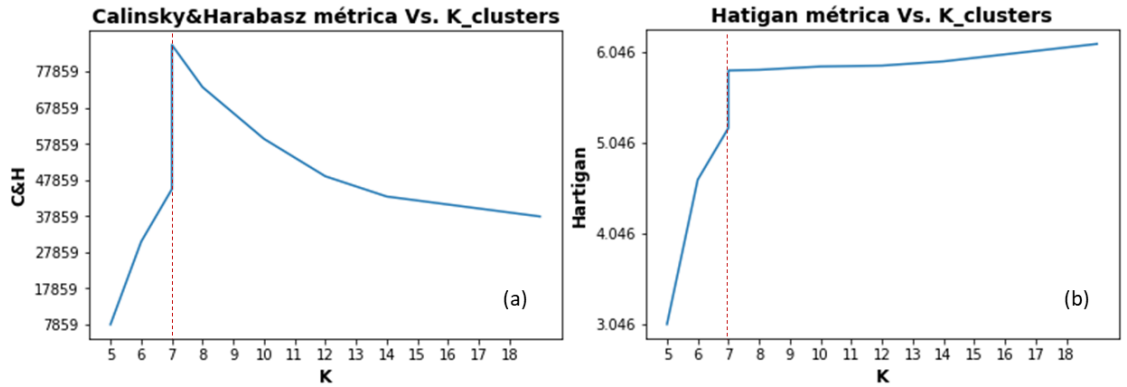

Figura 32: Indicares de C&H y Hartigan Vs. número de clústeres K (cerca).

De nuevo se obtiene un punto de inflexión en K= 7, en los dos indicadores. Además, si se recuerda la gráfica 30, donde se comparaba el número de clústeres con el valor de Thr, el punto de cambio de pendiente es justamente donde k=7 y Thr=2.7.

De esta manera se concluye que los indicadores de Calinsky&Halabasz y Hartigan son adecuados para establecer si un agrupamiento es correcto o no lo es, ya que se encuentra completa relación entre los valores de Thr, K, los errores dentro y entre grupos, que son los que representan precisamente los indicadores dentro de sus

métricas. Sin embargo, debido a su interpretación directa, se elige el indicador de Calinsky&Halabasz como base, para medir la calidad del agrupamiento de las rutas.

6.3 Selección de Threshold "óptimo"

Según lo discutido en el apartado anterior, para encontrar el valor óptimo de Thr, bastará con iterar el algoritmo tantas veces, hasta que el indicador de Calinsky&Harabasz sea máximo. Cabe anotar que este es un procedimiento que se puede hacer porque se tiene toda la data disponible en el tiempo cero. De no ser el caso se deberá establecer un Thr basado en otros criterios que dependerán del contexto al cual se aplique el algoritmo, u optar por desarrollar un mecanismo que haga que el Thr vaya aumentando o cambiando de manera dinámica si es el caso en el que no se tiene conocimiento muy claro del contexto, o si las unidades de medida de la distancia no tienen un significado real (como en el caso de usar PCA).

Sin embargo, es posible (como en este caso), que a priori se tenga una muestra de la data con la que se puede realizar una primera aproximación del Thr, y que, con el tiempo, de manera periódica, este parámetro se irá calibrando a medida que se obtenga más data.

<u>Probando los indicadores de rendimiento con data gpx e incremental clustering:</u>
En este caso se realiza la prueba con la data gpx disponible que consta de 213 rutas válidas. Se realizan 100 iteraciones, con un Thr inicial de 0.15 (es decir, 15 metros porque la unidad que se maneja son los kilómetros), y un incremento también de 0.015 por iteración. Los resultados se presentan a continuación:

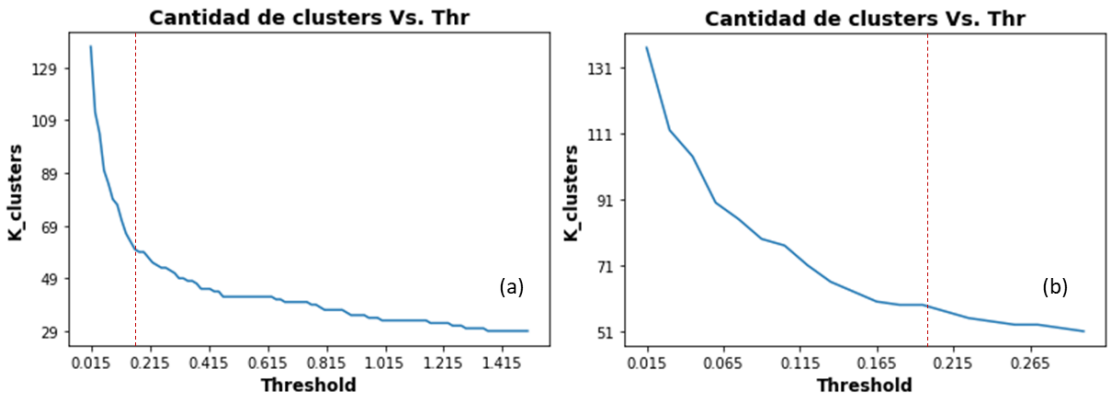

Figura 33: Threshold versus K cluster en la agrupación de rutas.

En la figura 33, se presenta la curva del Thr versus el número de clusters. En la sección (a) no es tan claro, pero pareciera que hay un cambio de pendiente en un punto cercano entre 0.11 y 0.21 kilómetros. Sin embargo, al acercarse como se ve en la sección (b), es menos claro ver este comportamiento. Aunque el corte no es completamente claro, pareciera que el Thr adecuado estuviera en ese rango. Se procede a visualizar el comportamiento de la métrica de evaluación, es decir, el índice de Carlinsky&Harabasz.

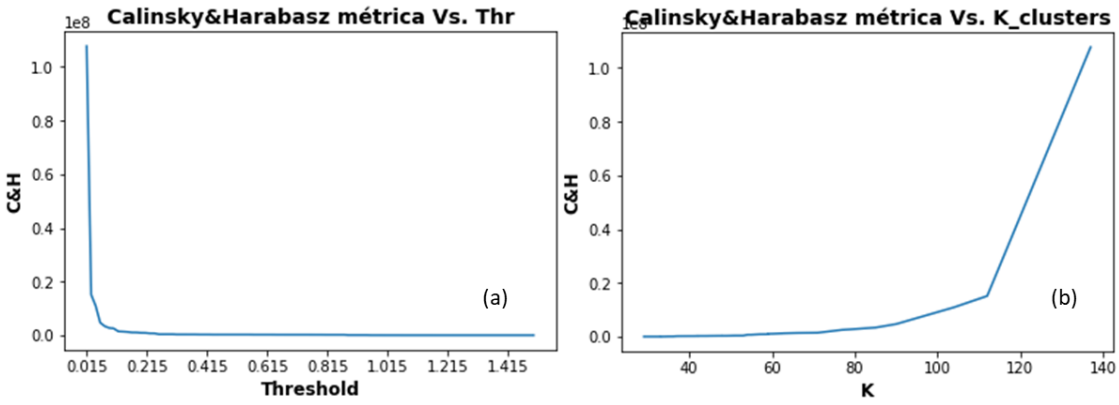

Figura 34: Indicar C&H versus K clusters en la agrupación de rutas.

En la figura 34.a, se presenta la curva de Thr versus C&H, y en la sección 34.b, se presenta el número de clusters versus C&H. El comportamiento es bastante particular e inesperado. Anteriormente se había visto que el indicador de C&H obtenía su valor máximo en aquel punto donde el Thr y por ende el K era óptimo, pero en este caso a medida que Thr aumenta (y por ende K disminuye) el indicador se hace más y más pequeño, hacia +infinito, porque no pareciera que fuera a converger. Sin embargo, es posible que a pesar de que se inició la iteración con un Thr=0.015, que equivale a 15 metros, este sea aún un valor demasiado grande para considerar dos rutas similares, así que se procede a hacer el ejercicio, pero ahora iniciando con un Thr=0.0015 (1.5 metros) con un incremento igual por iteración.

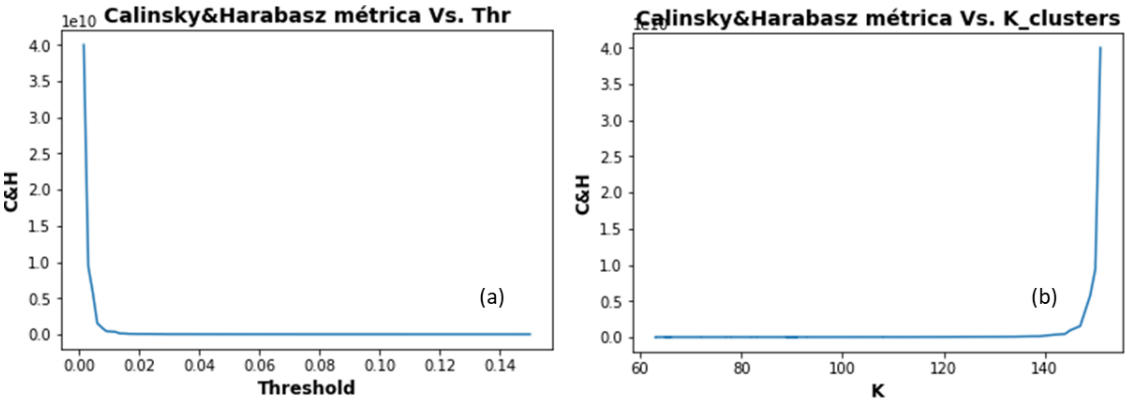

Figura 35: Indicar C&H versus K clusters en la agrupación de rutas(iteración 2).

Después de correr el modelo con los valores indicados, en la figura 35 se presentan de nuevo las curvas de rendimiento y se observa que efectivamente el indicador no converge. A medida que el Thr es más y más pequeño, el K es más grande y el C&H se dispara hacia más infinito. En este punto se descarta que la solución sea encontrar un Thr aún más pequeño, porque por conocimiento del problema, se sabe que si los centroides de dos rutas están a 1.5 metros, es prácticamente la misma posición, y claramente se pueden considerar iguales sin necesidad de realizar un algoritmo de agrupamiento. Es así como se concluye que, al parecer debido al número de puntos a agrupar, el número de clústeres a crear o la entropía del modelo en sí, el indicador no es determinante a la hora de decidir el Threshold óptimo.

El indicador falló, entonces ¿Cuál Threshold elegir?:

Según los resultados previos, el indicador C&H parece no converger en este caso en particular , al parecer por la sensibilidad de pequeños cambios de Thr y el número elevado de K. Sin embargo, se debe elegir un Thr para el modelo, así que se hará uso de la información que se tiene tanto del contexto del problema como de la información que se haya podido recolectar.

Por un lado, se sabe que, en el mundo real, si se habla de qué tan alejado se encuentran dos puntos geográficos, se considera que 100 metros es una distancia prudente y se puede considerara cercana, ahora, si se tiene en cuenta que se tienen rutas que pueden tener 8, 10 o más kilómetros, parece que una desviación de 100 metros no parece ser significativa. Por otra parte, no hay que olvidar el gráfico de K versus Thr, donde se ve claramente que hay un cambio de pendiente entre 0.11 y 0.21 kilómetros. Al revisar dato a dato, se observa que el punto de mayor inflexión es aquel donde el Thr es de 0.15 kilómetros. Es posible que sea un poco subjetivo este criterio, pero es la mejor aproximación que se puede tener con la información disponible, además que, con este Thr se generan 63 clústeres, los cuales, al realizar una verificación visual, demuestra que los clústeres creados contienen rutas similares entre ellas, ejemplo de esto se puede observar en la gráfica 36, donde un clúster con 8 rutas visualmente similares.

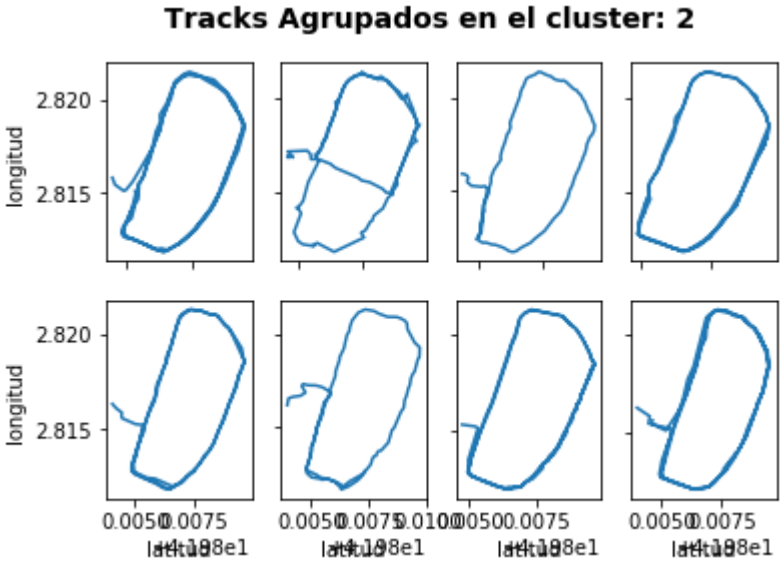

Figura 36: Comparación de rutas de un mismo cluster con Thr=0.15

¿por qué no un Threshold dinámico?:

Otra de las opciones que se encuentra en la bibliografía[13], es la posibilidad de un Thr dinámico que vaya cambiando a medida que el problema va evolucionando. Sin embargo, se considera que, para este caso, donde el Thr tiene una dimensión determinada, y en sí mismo puede considerarse grande o pequeño, no se hace necesario tratar de "encontrar" su valor por medio de un cambio dinámico, ya que, si por ejemplo en este caso se establece que 150 metros es una medida adecuada de similitud entre dos rutas, no hay razón para que este valor aumente. Por otra parte, aumentar o cambiar el Thr, si se considera un riesgo, ya que si por ejemplo en las

primeras etapas del agrupamiento, justo se agrupen dos rutas bastante disimiles entre ellas y si se establece que el Thr cambie en relación con su distancia entre centroides, las rutas que a futuro no tendrán un agrupamiento ajustado. Por esto se considera que lo mejor es encontrar un valor de Thr que tenga sentido en la vida real, permita establecer si dos puntos son cercanos y se mantenga constante al largo del tiempo, hasta que se realice una nueva calibración del modelo.

6.4 Usando métodos clásico y K-means

En el apartado anterior se estableció que el Thr adecuado era de 0.15 kilómetros, lo que daba como resultado un agrupamiento de 213 rutas en 63 clusters. Sin embargo, se veía que el criterio de decisión se basada en conocimiento propio del problema y contextualización, y el comportamiento que se observaba de k clusters versus Threshold. Así entonces se quiso realizar una comparación con el algoritmo clásico de K-means, junto con el método del codo y de silhouette , para ver si el resultado era similar o no al obtenido anteriormente.

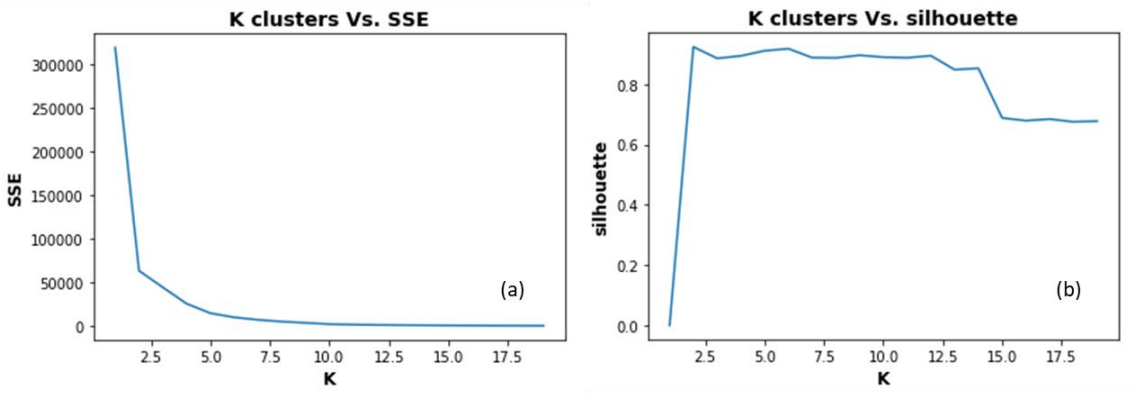

Figura 37: Método del codo y silhouette en kmeans.

El resultado de la gráfica 37, muestra claramente que el método del codo(a) y el método de silhouette(b) tampoco son una buena aproximación al problema, porque según estos indicadores, el número de clusters adecuado está cercano a los 2. Lo que a todas luces no es un buen agrupamiento para 213 rutas. Sin embargo, se quiso comprobar de gráficamente el comportamiento después de implementar k-mean, así que se realizó el agrupamiento para k=2 y k=3.

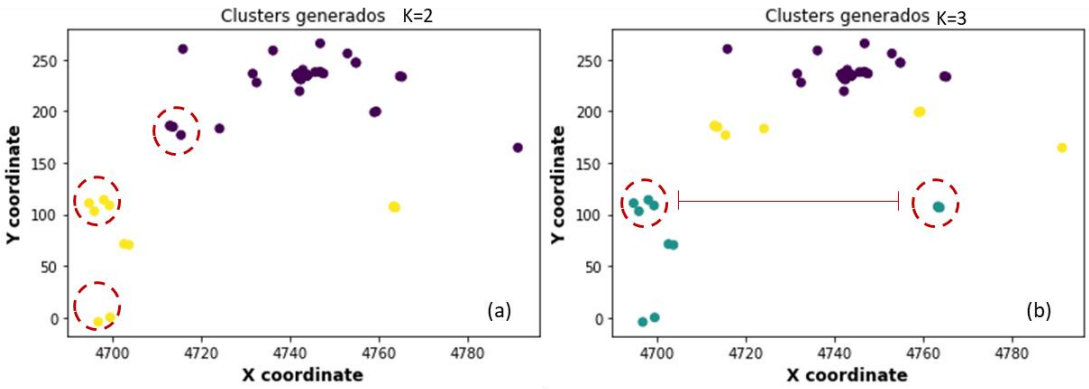

Figura 38: Agrupamiento con K=2 y K=3 en kmeans.

En la gráfica 38, se observa claramente que k-means no es capaz de identificar correctamente los clusters que deben formarse, ya que tanto para el caso de k=2, como para k=3, se evidencia que hay subclusters dentro de los formados. Son puntos que claramente se encuentran bastante cercanos entre ellos, y alejados de los demás. Por ejemplo, en la sección (b) se observa como un punto que pertenece al clúster 3 está mucho más alejados de los miembros de su mismo cliuster, que, a miembros de otro clúster, y es más, este debería ser considerado un clúster aparte, ya que se encuentra lo suficientemente alejado de otros puntos.

6.5 Interfaz gráfica y producto final

La interfaz le permite al usuario diferentes acciones, entre ellas: buscar una nueva ruta de su interés según diferentes criterios, crear una nueva ruta o rutas y observar si esta ha sido asignada a algún clúster existente o a creado un nuevo clúster, graficar las curvas de velocidad, elevación de diferentes clusters y finalmente desplegar el mapa de ruta de un clúster en el navegador.

En la figura 39 se observa el home del aplicativo:

Figura 39: Pantalla "Home" del aplicativo.

En el home se le da la bienvenida al usuario y le presenta las dos posibilidades: Encontrar rutes o crear una nueva. Sin embargo, el aplicativo lanza diferentes mensajes de advertencia según sea la ocasión.

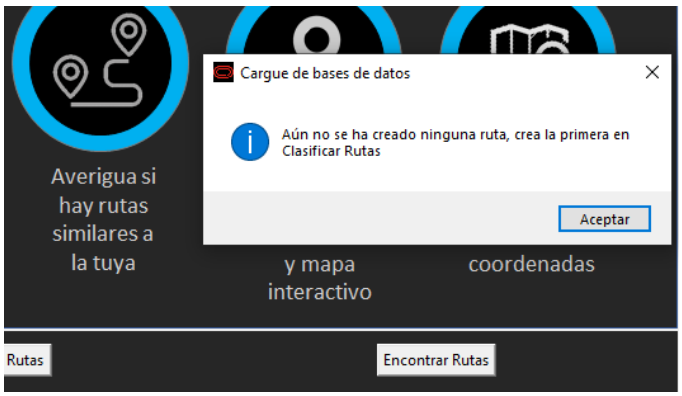

Figura 40: Mensaje de notificación sobre bases vacías.

Si se desea realizar una búsqueda, pero aún no hay información de clusters creados, se obtendrá el mensaje que se observa en la figura 40. Esto porque el archivo variables_clustering.pkl se encuentra vacío, o mejor, no ha sido creado.

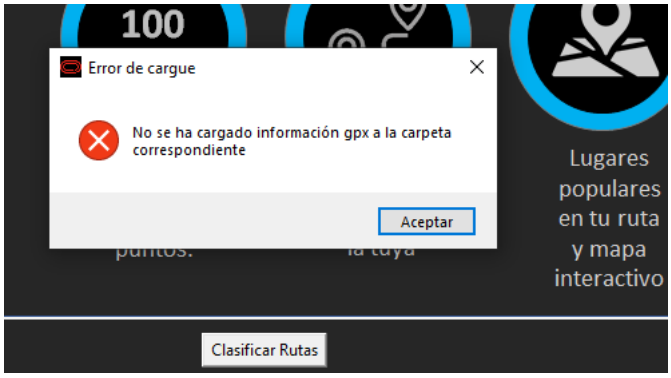

Figura 41: Mensaje de error, no hay archivos .gpx

Si se desea realizar una nueva clasificación o agrupamiento, pero no se han cargado las rutas .gpx en la carpeta correspondiente (tracks_gpx), el aplicativo lanzara el mensaje de error que se observa en la figura 41.

Acción: Buscar Rutas

En la figura 42 se observa la segunda pantalla del aplicativo, que aparece al realizar una búsqueda de rutas:

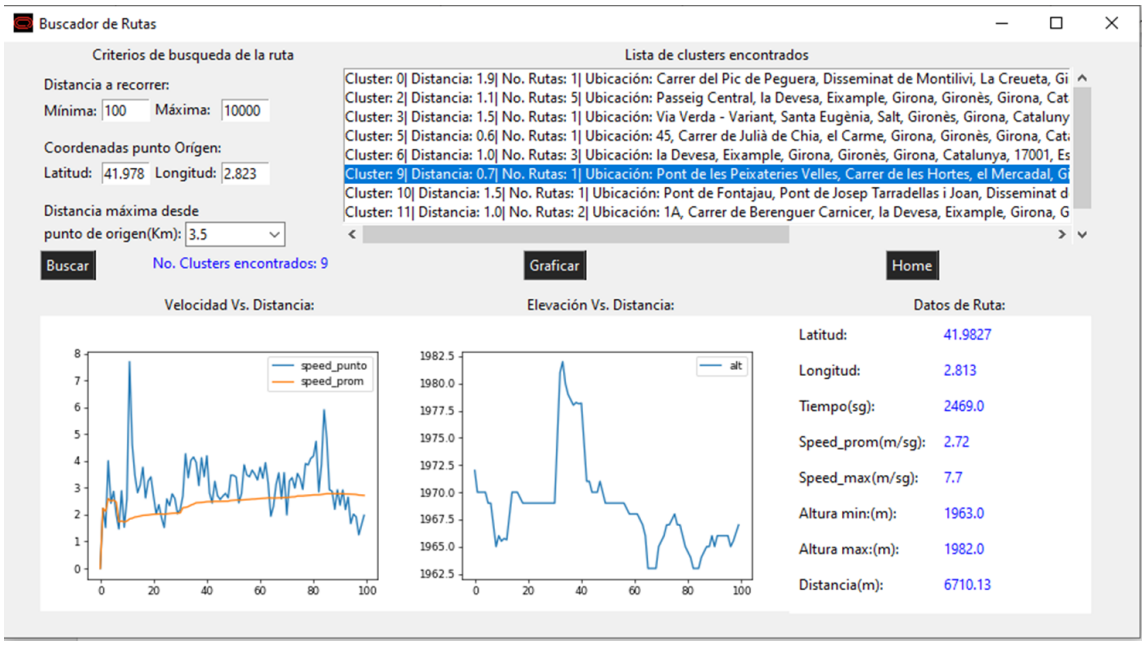

Figura 42: Pantalla 2. Buscador de rutas del aplicativo.

La pantalla de búsqueda se compone de 3 segmentos principales: el primero son los criterios de búsqueda, los cuales le permiten al usuario establecer las características de las rutas de su interés. Estas características son: la distancia mínima y máxima a recorrer, es decir que tan larga es el Trail; las coordenadas de origen y la distancia máxima a ese punto de origen, le permite al usuario establecer unas coordenadas de donde se encuentra, y el aplicativo buscará todas aquellas rutas que cuyo centroide esté a un máximo de distancia del establecido. El segundo segmento hace referencia al listado de clústeres que el aplicativo encontró con las características deseadas. Este listado tiene variedad de información como: nombre del clúster, distancia a la que se encuentra del punto de origen, número de rutas similares que componen el clúster y la ubicación, en la que se puede identificar la provincia, el país y hasta el código postal.

En el tercer segmento es posible ver 3 diferentes gráficas: la primea muestra la velocidad puntual y la velocidad promedio de toda la ruta, la segunda muestra los cambios de altitud del recorrido y la tercera muestra un listado con las estadísticas y datos importantes de la ruta, como, por ejemplo, coordenadas de origen, tiempo en promedio que toma recorrerse, velocidad máxima, velocidad mínima, altura, entre otros. Esta información aparece cada vez que se haga clic sobre cada uno de los clusters en la lista de clusters encontrados.
Esta pantalla tiene dos botones, uno para regresar al home (si no, no será posible salir del aplicativo) y otro para graficar el mapa del clúster seleccionado. En la figura 43 se ve un ejemplo de un mapa desplegado en el navegador:

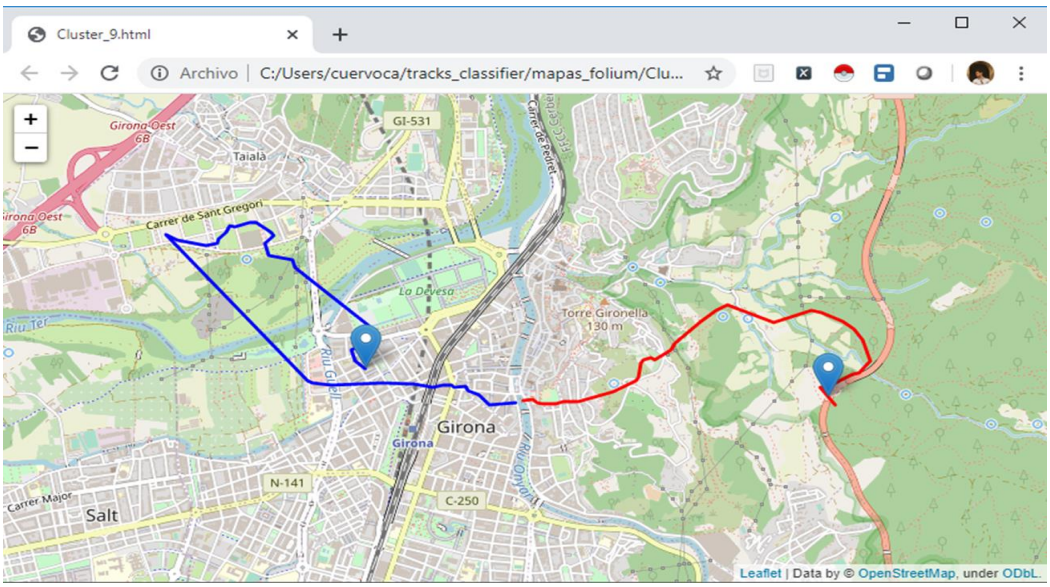

Figura 43: Ejemplo de mapa de ruta en el navegador con Folium.

<u>Acción: Crear Rutas</u>

En la tercera pantalla del aplicativo, es posible crear o clasificar nuevas rutas. En la figura 44 se observa esta pantalla:

Figura 44: Pantalla 3 del aplicativo. Creador de rutas.

Es una pantalla similar a la anterior, sin embargo, acá no hay criterios de búsqueda, en cambio se obtienen dos listados. El primero indica en qué clúster fue asignada cada ruta, y si tal clúster es nuevo o es una asignación a uno ya creado. En el segundo listado se encuentra la información del clúster, número de rutas y ubicación del mismo. En esta pantalla también existe la posibilidad de ver las gráficas de la ruta y graficar el mapa del clúster seleccionado, así como se ve en la figura 43.

7. Conclusiones

- En este trabajo se ha evidenciado que no siempre es conveniente hacer uso de la mayor cantidad de variables que se tenga disponibles, sino de aquellas que en realidad le pueden aportar al modelo. Se evidencia como al contemplar no solo las variables de geoposicionamiento, sino otras como tiempo o velocidad, el agrupamiento no tenía un buen desempeño, y esto se debe a que a pesar de ser una misma ruta, los valores de tiempo y velocidad varían fuertemente según el atleta o la condición en la que se encuentre en un momento determinado el mismo atleta, haciendo que unos días recorra 9 kilómetros en una hora, y otros recorra la misma distancia en hora y media. Esto hace que el modelo considere a todas las variables con el mismo peso, y así, agrupa rutas que pueden tener velocidades y tiempos similares, lo cual sería acertado si se quisieran agrupar rutas con esas características en común, pero no es el propósito de este trabajo, sino agrupar rutas similares según su posicionamiento geográfico.

- La segunda conclusión, muy relacionada con la primera es que la estandarización y el uso de PCA no siempre es el enfoque adecuado para resolver un problema. En este caso particular, se presentan dos dificultades: la primera es la que se comenta en la conclusión anterior sobre darle el mismo peso a todas las variables cuando no es deseado, y la segunda es que al usar PCA, se pierde por completo la información de las unidades, haciendo que no tengan ningún significado en la vida real las variables obtenidas y así, dificultando determinar la magnitud de distancia entre dos puntos.

- Existe la posibilidad de realizar agrupamiento con variables que tengan unidades con significado en la vida real, como es el caso de los centroides de las variables X,Y obtenidas al realizar la conversión de coordenadas geodésicas a cartesianas. Presentan la ventaja de que cualquier distancia calculada entre dos puntos con estas coordenadas estará en términos de metros, o en kilómetros si se multiplica por 0.01, mostrando así que la conversión de coordenadas se vuelve un paso importante a la hora de trabajar con información de geoposicionamiento.

- Se pudo comprobar que para casos en los que el número de grupos no es muy elevado, los indicadores de rendimiento como Calinsky&Harabasz y Hartigan, son determinantes al momento de elegir el Thr adecuado y por consiguiente el número de clústeres. Sin embargo, también se evidenció, que cuando el problema contempla un número de clústeres elevado, los indicadores no

convergen, haciendo que no sea posible determinar los parámetros adecuados del agrupamiento.

- Así mismo, se comprueba que los métodos clásicos como el del codo o silhouette, tampoco presentan un buen comportamiento en todos los casos. De hecho, la solución que planteaban estos dos métodos era bastante mala cuando se quiso agrupar 213 rutas, dando como resultado que formaban 2 grupos, los cuales gráficamente era muy sencillo de notar su baja calidad.

- Se considera que el método de incremental kmean no sólo es un algoritmo que se debe usar cuando el procesamiento es en streaming, o se trabaja con Big Data, sino que además se considera que es un algoritmo superior al clásico kmeans, ya que este segundo elige las semillas de los clusters al azar, generando diferentes resultados en cada iteración, mientras que el incremental learning va ajustando los centroides de cada clúster según su threshold, que como se pudo ver en este trabajo, cuando su valor tiene sentido en la vida real y va alienado con la naturaleza del problema, tales centroides también se ajustan a la naturaleza del problema, haciendo que el resultado sea estable sin importar mucho el orden de agrupación.

- Es importante anotar, que a pesar de que los indicadores de rendimiento, tanto los clásicos como los propuestos en este trabajo, no siempre funcionan, se debe realizar su respectivo análisis para poder descartar o no, el valor de un hiper-parámetro. Así mismo, hay que considerar que al momento de usar incremental learning, al parecer el plot entre Threshold y número de clusters K, puede dar una muy buena aproximación del valor a elegir.

8. Trabajos futuros

Cómo trabajo futuro sería interesante trabajar en diferentes frentes como:

- Encontrar la razón teórica y empírica de por qué los indicadores de rendimiento que tienen como base la varianza del error dentro y entre grupos, no convergen en este caso, y como se podría solucionar, o si existe otra aproximación de indicadores que sí permitan medir la calidad del agrupamiento.
- Sería interesante investigar sobre otras maneras de representar una ruta, no necesariamente haciendo uso únicamente de su centroide sino de otras variables, o tal vez las mismas, pero con un tratamiento diferente, por ejemplo, una especie de ponderación por cada uno de los puntos de la ruta.
- Poder realizar un agrupamiento con rutas de diferente cantidad de puntos, no con la restricción de que sean de 100.
- En la parte de producto, sería interesante desarrollar el aplicativo en una plataforma web, en la que sea mucho más sencilla su manipulación de información y visualización de mapas.

9. Glosario

Running trails: Hace referencia a los senderos o rutas al aire libre, que las personas utilizan habitualmente para correr o prepararse para una competencia de running.

Particiones: Son cada una de las segmentaciones o divisiones del conjunto de datos , obtenidas al aplicar el algoritmo de clustering. Se espera que los miembros que la componen sean similares entre ellos y muy diferentes a miembros de otros grupos.

Centroides: Es el punto tal, donde interceptan todos los hiperplanos permiten que el resultado de la división cada fragmento tenga el mismo tamaño. Es decir, permite establecer cuál es el centro de una figura geométrica.

Clustering: Método de aprendizaje no supervisado que permite agrupar en un mismo cluster a individuos que presentan similitud entre ellos, según algún criterio de distancia o similaridad.

Folium: Librería de Python que permite graficar coordenadas de latitud y longitud sobre un mapa del mundo real para ser desplegado en un navegador web.

Formato .gpx: Es la manera estandar de intercanviar y almacenar información de mapas en dispositivos GPS, teléfonos inteligentes y computadoras. La estructura del archivo es similar a la estructura de un archivo semiestructurada de XML

Threshold: Umbral que sirve como parámetro de entrada al algoritmo de incremental clustering y permite establecer el límite máximo de la distancia que debe existir entre un punto de un grupo y su centroide. Si esta distancia es mayor al Threshold, se deberá crear un nuevo cluster o grupo.

Paratránsito: Modo de transporte flexible y a veces informal de pasajeros, que no sigue un programa u horario fijo.

10. Referencias bibliográficas

[1]Agustí S. & Pérez C. (2016). *Sports Tracker Segur*. Universitat Oberta de Catalunya. Trabajo de fin de máster.

[2] Cávar I., Kavran Z. & Petrovic M. (2011). *Hybrid approach for urban roads classification based on GPS tracks and road subsegments data*. University of Zagreb. Faculty of transport and Traffic Sciences.

[3] Preez D., Zuidgeest M. & Behrens R. (2019). *A quantitative clustering analysis of paratransit route typology and operating attributes in Cape Town*. University of Cape Town, Centre for Transport Studies, Department of Civil Engineering. ElSEVIER https://doi.org/10.1016/j.jtrangeo.2019.102493

[4] Gironés J., Casas J. Minguillón J, et. al. (2017). *MINERÍA DE DATOS MODELOS Y ALGORITMOS*. Editorial UOC, Barcelona. ISBN: 978-84-9116-904-8

[5] Rojas J., Chavarro J. & Moreno R.(2008). TECNICAS DE LOGICA DIFUSA APLICADAS A LA MINERIA DE DATOS. Universidad tecnológica de Pereira. Revista revista Scientia et Technica Vol. 3, Núm. 40 (2008)

[6] Memarsadeghi N., , M. Mount D., S. Netanyahu N., & Le Moigne J. (2007). *A Fast Implementation of the ISODATA Algorithm*. International Journal of Computational Geometry and Applications, 17 (2007), 71-103.

[7] Peña J.M., Lozano J.A. & Larrañaga P.(1999). *An empirical comparison of four initialization methods for the K-Means algorithm*. University of the Basque Country. Department of Computer Science and Artifcial Intelligence, Intelligent Systems Group. Pattern Recognition Letters 20 (1999) 1027-1040

[8] Ade R. R., Deshmukh P. R. (2013). *METHODS FOR INCREMENTAL LEARNING: A SURVEY*. International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.3, No.4, July 2013

[9] Yue W., Yinpeng Ch., Lijuan W., et. al. (2019). Large Scale Incremental Learning. arXiv: 1905.13260v1

[10] Prithviraj D., Rajat V., Kuan-Chuan P., et. al. (2019) Learning without Memorizing. arXiv:1811.08051v2

[11] Sathya M., Suresh J.& Jayashree S. (2018). *Incremental Learning for Classification of Unstructured Data Using Extreme Learning Machine*. Algorithms 2018, 11, 158; doi:10.3390/a11100158 Journal (WCSIT) ISSN: 2221-0741 Vol. 3, No. 5, 97-99

[12] Nidhi Gupta & R.L.Ujjwal. An Efficient Incremental Clustering Algorithm.(2013). World of Computer Science and Information Technology J

[13] Xiaoke S., Yang L., Renxia W., et. alr.(2009) *A Fast Incremental Clustering Algorithm*. International Symposium on Information Processing (ISIP'09).

[14] Aaron B., Tamir D. E., Rishe N., et. al.(2014). *Dynamic Incremental K-means Clustering*. Intemational Conference on Computational Science and Computational Intelligcnce. DOI 00110.1109/CSC!.2014.60

[15] Bao J, Wang W, Yang T, Wu G (2018) *An incremental clustering method based on the boundary profile*. PLoS ONE 13(4): e0196108.https://doi.org/10.1371/journal.pone.0196108

[16] Wang J., Zelenyuk A. , Imre D. & Mueller k. (2017). *Big Data Management with Incremental K-Means Trees–GPU-Accelerated Construction and Visualization. Informatics*. 2017, 4, 24; doi:10.3390/informatics4030024

[17] Han J., Kamber M. & Pei J.(2011). *Data mining : concepts and techniques*. Jiawei Han, Micheline Kamber, Jian Pei. – 3rd ed.p. cm. ISBN 978-0-12-381479-1

[18] Stephen L. France, Ulas Akkucuk. (2020). *A review, framework, and R toolkit for exploring, evaluating, and comparing visualizationmethods*. Springer.doi.org/10.1007/s00371-020-01817-5

[19] Pelegrón C., Peréz N. & Ventura C. (2019). *Exploring dimensionality reduction and Machine learning methods for the prediction of body composition abnormalities among an HIV+ population*. Universitat Oberta de Catalunya. Trabajo de fin de máster. Master in Biostatistics and Bioinformatics.

[20] Neethu C.V. & Surendran S. (2013). *Review of Spatial Clustering Methods*. nternational Journal of Information Technology Infrastructure. ISSN 2320-2629. Disponible en: http://warse.org/pdfs/2013/ijiti01232013.pdf

[21] Qinpei Z., Mantao X., & Pasi F.(2009). *Sum-of-Squares Based Cluster Validity Index and Significance Analysis*. University of Joensuu. Department of Computer Science. Finland. Springer ICANNGA 2009, LNCS 5495, pp. 313–322, 2009

[22] Pardo M. *Clustering*. National Research Council (CNR), Italy & Max Planck Institute for Molecular Genetics, Berlin, Germany. Recuperado el 18 de marzo de 2020 del sitio web: http://lectures.molgen.mpg.de/algsysbio10/clustering.pdf

[23] Kassambara A. *Determining The Optimal Number Of Clusters: 3 Must Know Methods*. Data Novia. Recuperado el 25 de marzo de 2020 del sitio web: https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/

[24] Berzal F.Clustering basado en densidad. Universidad de Granada. Departamento de ciencias de la computación e I.A.. Recuperado el 22 de marzo de 2020 del sitio web: https://elvex.ugr.es/idbis/dm/slides/43%20Clustering%20-%20Density.pdf

[25] Hadas D. & Intrator N. (2011). *Unsupervised Learner for coping with gradual Concept Drift*. TEL AVIV UNIVERSITY, The Blavatnik School of Computer Science. Master's thesis in Computer Science.

[26] Haibo H., Sheng Ch., Kang L., & Xin X.,.(2011*). Incremental Learning from Stream Data*. IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 22, NO. 12, DECEMBER 2011. DOI 10.1109/TNN.2011.2171713

[27] Rosebrock A. (2019). *Online/Incremental Learning with Keras and Creme*. Pyimagesearch. Recuperado el 28 de marzo de 2020 del sitio web: https://www.pyimagesearch.com/2019/06/17/online-incremental-learning-with-keras-and-creme/

[28] Geomatics. (2018). *Creating GPX File Viewer in Python*. GEODOSE. Recuperado el 18 de marzo de 2020 del sitio web: https://www.geodose.com/2018/04/create-gpx-tracking-file-visualizer-python.html

[29] Van Dorpe S. (2018). *How tracking apps analyse your GPS data: a hands-on tutorial in Python*. Towards data science. Recuperado el 19 de marzo de 2020 del sitio web: https://towardsdatascience.com/how-tracking-apps-analyse-your-gps-data-a-hands-on-tutorial-in-python-756d4db6715d

[30] Rai A. *Plotting Google Map using gmplot Package*. GeeksforGeeks. Recuperado el 20 de marzo de 2020 del sitio web: https://www.geeksforgeeks.org/python-plotting-google-map-using-gmplot-package/

[31] Conceptos básicos de ayuda de CRISP-DM. IBM. Recuperado el 29 de febrero de 2020 del sitio web: https://www.ibm.com/support/knowledgecenter/es/SS3RA7_sub/modeler_crispdm_ddita/clementine/crisp_help/crisp_overview.html

[32] *Conceptos básicos de ayuda de CRISP-DM. IBM*. Recuperado el 29 de febrero de 2020 del sitio web: https://www.ibm.com/support/knowledgecenter/es/SS3RA7_sub/modeler_crispdm_ddita/clementine/crisp_help/crisp_overview.html