PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# MANEJO DE DATOS ESPACIALES

**ERICK JHONATAN MAULEN TORO**
**LUCAS TOMAS AGULLO VIVEROS**
**MATIAS ALONSO ESPINOZA ARANGUIZ**
**BENJAMIN JESUS ROJAS DELPINO**

ESTRUCTURAS DE DATOS AVANZADAS Y ALGORITMOS

Octubre, 2020

# Índice

## Lista de Figuras

# 1. Modelado

Para el modelado de este problema se trabajó mediante una entrada en formato csv, la cual se cargará de la siguiente forma.

```
datos = pd.read_csv('Desafio3.csv',engine='python',index_col=0)
datos = datos.replace(np.nan,"0")
```

Figura 1: Carga de datos

Para hacer mas sencillo el trabajo con csv se ocupo Pandas, el cual consiste en un paquete de Python que proporciona estructuras de datos similares a los dataframes de R, lo cual facilita la manipulación de gran volumen de datos. Los datos serán insertados en un KD-Tree, debido a su sencillez y parecido a un árbol binario. Posteriormente se obtendrán los puntos, para poder agregar solo lo importante al KD-Tree, sin embargo es necesario aplicar One Hot Encodig en los Géneros, debido a que es necesario definir una representación vectorial numérica para ellos y así poder comparar. La estrategia que implementa es crear una columna para cada valor distinto que exista en la característica que estamos codificando y, para cada registro, marcar con un 1 la columna a la que pertenezca dicho registro y dejar las demás con 0.

```
primegenre_dummy = pd.get_dummies(self.data['prime_genre'], prefix="prime_genre")
datos = pd.concat([self.data, primegenre_dummy], axis = 1)
```

Figura 2: One hot Encoding

Los nodos , donde se almacenará la información tienen la siguiente estructura

```
def __init__(self,data,point: np.ndarray,hijoDerecho=None, hijoIzquierdo=None):
    self.node_right = hijoDerecho
    self.node_left = hijoIzquierdo
    self.data = data
    self.point = point
```

Figura 3: Estructura nodo

Donde data es toda la información de cada aplicación y point es el punto en el espacio que representa ese conjunto de datos.

## 2. Estrategia

Para dar solución a este desafío como grupo decidimos utilizar un kd tree debido a su capacidad para trabajar con k dimensiones de datos, lo que nos beneficia un montón al momento de tener que encontrar los vecinos cercanos basándonos en todos los criterios que teníamos, en este caso 28 criterios, además de esto el kd tree tiene una estructura muy similar a un árbol binario, estructura de datos con la cual estamos muy familiarizados.

## 3. Análisis

El análisis se hará basándose principalmente en comparar la estructura escogida para resolver el problema, KDTree y una Lista de Python, compararemos las tres funciones utilizadas en el problema: Inserción, búsqueda de un objeto y búsqueda de vecinos cercanos a un objeto.

Inserción:
Debido a la naturaleza del KDTree (que es un árbol binario), la inserción comparará cada dato para ir viendo hacia donde ramificar y añadir el nuevo dato, dando como resultado una inserción de complejidad temporal de o(log n). En cambio, la lista es más rápido de insertar, sólo se deberá concatenar el valor al final, es decir o(1), ya que es inmediato, y se sabe cuál es el último valor (por la implementación de python). Otra diferencia, es el hecho de que las listas de Python están implementadas en arreglos dinámicos, lo que significa que en algún momento cuando este llegue al tope, se tendrá que reasignar el tamaño al arreglo.

Búsqueda de un Objeto:
En el caso de la búsqueda, el KDTree gana por mucho. Este tendrá una complejidad de o(log n) mientras que el de la lista dependerá de la cantidad de objetos dentro de esta. Es decir, o(n).

Búsqueda de vecinos cercanos:
La búsqueda de vecinos cercanos, al menos en nuestra implementación, en la lista será de dos iteraciones, una que buscará la máxima distancia y la otra que comparará y encontrará las 10 menores distancias encontradas, dando como resultado o(2n) ->o(n). En el caso del KDTree, será de complejidad o(mlog n), donde m será la cantidad de vecinos y n la cantidad de nodos en el árbol.

Conclusiones:
Hecha esta comparación podemos concluir que la utilización de estructuras de búsqueda espacial para problemas como este es sumamente esencial. Claro, para problemas en donde se desee buscar entre pocos valores la diferencia quizá no sea tan clara, pero para grandes datos, como lo fue en este desafío, es sumamente importante tener una manera de almacenar los datos para poder realizar operaciones de una manera óptima, en poco tiempo y que no cueste tanto.