TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

NOMBRE DE LOS ALUMNOS:

RAYMUNDO HIRALES LAZARENO (N. CONTROL: 17212339)

PAULA ANDREA RAMOS VERDIN (N. CONTROL: 18210721)

Carrera: Ingeniería Informática

MATERIA: Datos Masivos

PROFESOR: JOSE CHRISTIAN ROMERO HERNANDEZ

Practica Evaluatoria U3

FECHA: 01/06/22

Introduction

Este proyecto es una forma de poder manipular el marco de datos, así los datos y poder obtener resultados y hacer comparaciones y poder hacer probabilidad en un futuro, podemos ver la información de manera ordenada tan importante para poder ver mejor los resultados y la manera fácil.

La primera parte es importar la biblioteca que necesita, en este caso es Soark Session para la sesión Spark como vemos antes, después de iniciar sesión para hacer más pequeños los errores e importar la biblioteca para kmeans para el modelo, lo último es cargar el conjunto de datos.

```
// 1-. Importar una simple sesión Spark.
import org.apache.spark.sql.SparkSession

// 2-. Utilice las lineas de código para minimizar errores
import org.apache.log4j.
Logger.getLogger("org").setLevel(Level.ERROR)

// 3-. Cree una instancia de la sesión Spark
val spark = SparkSession.builder().getOrCreate()

// 4-. Importar la librería de Kmeans para el algoritmo de agrupamiento.
import org.apache.spark.ml.clustering.KMeans

// 5-. Carga el dataset de Wholesale Customers Data
val dataset =
spark.read.option("header","true").option("inferSchema","true").format("csv").load
("Wholesale customers data.csv")
```

```
scala> val dataset = spark.read.option("header","true").option("inferSchema","true").format("csv").load("Wholesale customers data.csv")
dataset: org.apache.spark.sql.DataFrame = [Channel: int, Region: int ... 6 more fields]
```

Para estas prácticas necesitamos seleccionar algunas columnas en específico

```
// 6-. Seleccione las siguientes columnas: Fresh, Milk, Grocery, Frozen,
Detergents_Paper, Delicassen y llamar a este conjunto feature_data
val feature_data = (dataset.select($"Fresh", $"Milk", $"Grocery", $"Frozen",
$"Detergents_Paper", $"Delicassen"))
```

```
scala> val feature_data = (dataset.select($"Fresh", $"Milk", $"Grocery", $"Frozen", $"Detergents_Paper", $"Delicassen"))
feature_data: org.apache.spark.sql.DataFrame = [Fresh: int, Milk: int ... 4 more fields]
```

Bueno, aquí solo importe la biblioteca para manipular vectores y ensamblar, y los usé

```
// 7-. Importar Vector Assembler y Vector
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.linalg.Vectors

//8-.Crea un nuevo objeto Vector Assembler para las columnas de caracteristicas
como un conjunto de entrada, recordando que no hay etiquetas
val assembler = new
```

```
VectorAssembler().setInputCols(Array("Fresh","Milk","Grocery","Frozen","Detergents
_Paper","Delicassen")).setOutputCol("features")
```

```
scala> val assembler = new VectorAssembler().setInputCols(Array("Fresh","Milk","Grocery","Frozen","Detergents_Paper","Delicassen")).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = vecAssembler_4f58f94a6714
```

Con el objeto create vamos a transform, usamos transform y save en las caracteristicas, lo otro se usa el modelo kmeans

```
//9-.Utilice el objeto assembler para transformar feature_data
val  features = assembler.transform(feature_data)

//10-.Crear un modelo Kmeans con K=3
val kmeans = new KMeans().setK(3).setSeed(1L)
val model = kmeans.fit(features)
```

```
model: org.apache.spark.ml.clustering.KMeansModel = kmeans_94dca89c2728
```

Aquí solo falta evaluar el modelo e imprimir los resultados

```
//11-.Evalúe los grupos utilizando Within Set Sum of Squared Errors WSSSE e
imprima los centroides.
val WSSSE = model.computeCost(features)
println(s"Within set sum of Squared Errors = $WSSSE")
```

```
scala> println(s"Within set sum of Squared Errors = $WSSSE")
Within set sum of Squared Errors = 8.095172370767671E10
```

Y lo último es solo imprimir los centros

```
println("Cluster Centers: ")
model.clusterCenters.foreach(println)
```

```
scala> println("Cluster Centers: ")
Cluster Centers:

scala> model.clusterCenters.foreach(println)
[7993.574780058651,4196.803519061584,5837.4926686217,2546.624633431085,2016.2873900293255,1151.4193548387098]
[9928.18918918919,21513.081081081084,30993.486486486487,2960.4324324324325,13996.594594594595,3772.3243243243246]
[35273.854838709674,5213.919354838709,5826.096774193548,6027.6612903225805,1006.9193548387096,2237.6290322580644]
```

Conclusion

Después de crear el modelo y trabajar con los datos, es claro las cosas que a simple vista no podemos ver toda la información más clara con las cosas que realmente le importan, y hacer comparaciones con los otros resultados.

Cuando haces este tipo de análisis puedes aprender más sobre los datos, puedes ver la información, porque son diferentes los datos y el resultado, y si hiciste este modelo es más fácil leer la información importante.