FIT5202 Data processing for big data¶

Activity: Machine Learning with Spark (Clustering using K-Means)¶

This week we are going to look into clustering using K-Means alogrithm. We will look into the case study where **we use machine learning to identify the involvement of the attackers**.

Table of Contents¶

- K-Means Clustering
- K-Means Clustering-DEMO
- Use Case : Problem Statement
    - Data Loading
    - Data Preparation
    - Feature Engineering
    - Clustering
    - Silhouette Score
- Optimal Number of Clusters
- Lab Tasks
    - Lab Task 1
    - Lab Task 2
    - Lab Task 3
    - Lab Task 4

K-Means Clustering ¶

Cluster analysis is an area of machine learning that focuses on finding patterns in unlabelled data. The main idea behind clustering is to group similar kinds of data together into clusters, in hopes of creating useful labels for these groups. This field is also known as `unsupervised learning`.

When supervised algorithms are used more for making predictions, unsupervised algorithms would be more useful for exploring data. `K-means` is one of the simplest unsupervised algorithms used today, and it can help partition a set of **n** observations into **k** clusters or groups.

K-Means Clustering DEMO ¶

The following example shows the implementation of K-Means clustering for a simple dataset with three columnns i.e. `email,income and gender.` The pipeline used here is very familier to what we have been doing since the last 2 labs.

1. Lab Task: Try to understand the steps shown in the example below. Briefly explain the pipeline implementation to your tutor to demonstrate your understanding about:

- 1. The use of 3 different transformers for feature engineering
- 2. How pipeline API is is used to organize the steps.

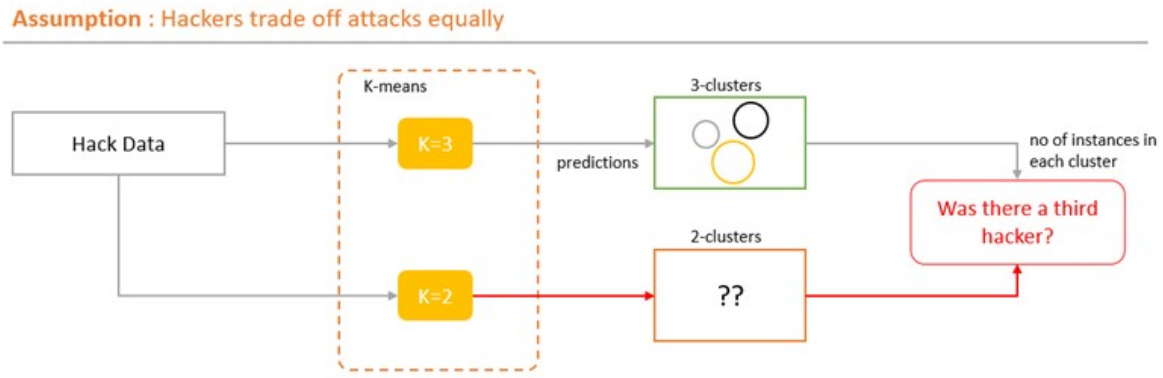
Also discuss about the **Predictions** and the **Silhouette score** that you see.

In [ ]:

```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
```

```
#Create Spark Session
spark = SparkSession.builder.appName('Clustering using K-Means').getOrCreate()

#Step 1 : Prepare the Data
df = spark.createDataFrame(
 [("a@email.com", 12000,"M"),
    ("b@email.com", 43000,"M"),
    ("c@email.com", 5000,"F"),
    ("d@email.com", 60000,"M"),
    ("e@email.com", 55000,"M"),
    ("f@email.com", 11000,"F")
 ],
 ["email",'income','gender'])

#Step 2 : Feature Engineering
indexer =
StringIndexer(inputCols=['email','gender'],outputCols=['email_index','output_index'])
encoder =
OneHotEncoder(inputCols=['email_index','output_index'],outputCols=['email_vec','outpu
t_vec'])
assembler =
VectorAssembler(inputCols=['email_vec','output_vec','income'],outputCol='features')
#Create a KMeans Model Estimator initialized with 2 clusters
k_means = KMeans(featuresCol='features',k=2)

#Step 3 : Pipeline API and ML Model
pipeline = Pipeline(stages = [indexer,encoder,assembler,k_means])
pipelineModel = pipeline.fit(df)

# Make predictions
predictions = pipelineModel.transform(df)
predictions.show()
# # Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
#HINT:You can use model.stages[-1] to access the model from the pipeline.
centers = pipelineModel.stages[-1].clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Problem Statement ¶



RhinoTech is a large technology firm that has been recently hacked. Luckily, the forensic engineers at the company have been able to grab metadata about each session used by the hackers to connect to RhinoTech servers. This data includes information such as session time, locations, words-per-minute typing speed, etc. You have been informed that there are **three potential hackers that perpetrated the attack. The RhinoTech forensic team are certain that the first two hackers were involved, but they want to know whether the third hacker was involved as well.**

One last key piece of information you've been given by the forensic engineers is that:

The Hackers trade off attacks equally!

For example, imagine there were 100 attack instances. If 2 hackers were involved, then each hacker would have about 50 attacks. but in the case of 3 hackers, each would have only 33 attacks.

To help you solve this problem, RhinoTech has provided you with a CSV file (Download from Moodle) that contains statistics about the attacks.

Step 1 : Loading dataset ¶

First, let's import the necessary Spark libraries we will use for data analysis, preparation, and clustering. Use `inferSchema=True` and display the schema of the dataset.

In [ ]:

```
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans

# Load csv file using spark session
data = spark.read.csv('hack_data.csv', header=True, inferSchema=True)
```

In [ ]:

```
# print the schema
data.printSchema()
```

Information on Dataset:¶

Our dataset contains 334 attack instances, with the following information for each one:

- **Session_Connection_Time** - How long the session lasted in minutes?
- **Bytes Transferred** - Megabytes transferred during session
- **Kali_Trace_Used** - Whether the hacker was using Kali Linux
- **Servers_Corrupted** - Number of server corrupted during the attack
- **Pages_Corrupted** - Number of pages illegally accessed
- **Location** - Location attack came from
- **WPM_Typing_Speed** - Estimated typing speed based on session logs

Step 2 : Data preparation ¶

`MLlib` library in Spark only accepts dataframes that have one column for clustering. This column should contain all the features in the form of vectors, with each vector corresponding to the features in that particular row.

IMPORTANT: The `Location` column will be useless to consider because the hackers probably used VPNs to hide their real locations during the attacks. Therefore, we do not have to include that.

In [ ]:
```
cols = ['Session_Connection_Time', 'Bytes Transferred', 'Kali_Trace_Used',
'Servers_Corrupted', 'Pages_Corrupted', 'WPM_Typing_Speed']
```

Step 3 : Feature Engineering ¶

Assemble using VectorAssembler¶

We can assemble our attributes into one column using Spark's **VectorAssembler**. When creating a **VectorAssembler** object, we must specify the input columns and the output column.

The input columns are a list of columns that we want to assemble, and the output column is just a name for the column created by the assembler.

In [ ]:
```
assembler = VectorAssembler(inputCols=cols, outputCol='features')
assembled_data = assembler.transform(data)
assembled_data.columns
```


Feature scaling : Standard Scaler¶

Next, we need to standardise our data. To accomplish this, Spark has its own **StandardScaler** which takes in two arguments — the name of the input column and the name of the output (scaled) column. [REF]

In [ ]:
```
scaler = StandardScaler(inputCol='features', outputCol='scaledFeatures')
```


In [ ]:
```
scaler_model = scaler.fit(assembled_data)
scaled_data = scaler_model.transform(assembled_data)
scaled_data.columns
```

Step 4: Clustering using K-Means ¶

In [ ]:
```
k_means = KMeans(featuresCol='scaledFeatures', k=3)
```

In [ ]:
```
model = k_means.fit(scaled_data)
```

In [ ]:
```
predictions = model.transform(scaled_data)
predictions.groupBy('prediction').count().show()
```

Was the third hacker involved?¶

Finally, it's time to find out how many hackers were involved with the attacks. Using `.transform()` on the clustering model will transform our dataset so that a new attribute called `predictions` will be created. This new column will contain integers that indicate the cluster to which each attack instance has been classified. Let's take a look at how many instances are grouped into each cluster in the case of three clusters.

2. Lab Task: Based on the above count you see for each group of attach, what do you think? Does it look like there were 3 hackers involved? Discuss this with your tutor?

Silhouette Score ¶

Silhouette Score represents the separation distance between the resulting clusters. Higher the

silhouette score, the better.

In [ ]:
```
from pyspark.ml.evaluation import ClusteringEvaluator
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

3. Lab Task: Recreate the above steps using the **Pipeline API**. In this case use **K = 2** in the K-Means Estimator. Also calculate the Silhouette Score for the pipeline model. **HINT:** You can use `model.stages[-1]` to access the model from the pipeline.

Step 1 : Load/Prepare the data¶

In this case you can use the same dataset from above.

Step 2 : Vector Assembler¶

In [ ]:
```
#Define the vector Assembler Here
```

Step 3 : Standard Scaler¶

In [ ]:
```
#Define the Standard Scaler here
```

Step 4 : K-Means (k=2)¶

In [ ]:
```
#Define the K-Means estimator with number of clusters =2
```


Step 5 : Pipeline¶

Plug all the steps above into a pipeline and generate the prediction count like before and analyze the number of instances in each cluster.

In [ ]:
```
#Define the pipeline
```


In [ ]:
```
#Fit and transform the pipeline to generate predictions
```

In [ ]:
```
#Display the number of instances grouped by cluster
```

Step 6: Silhouette Score¶

In [ ]:
```
#Calculate the Silhouette Score for the Pipeline Model
```

4. Lab Task: Discuss your observations about the following with your tutor:

1. What is the difference is Silhoutte Score with K=3 vs K=2? What do you think is the optimal number of clusters in this case?
2. Based on the number of instances in each cluster, does it look like there were 3 hackers involved in the attack?

Finding optimal number of clusters based on the Silhouette Score ¶

We already know, better Silhouette Score signifes a better separation of clusters. To find the optimal number of clusters, we can simply calculate the score for different cluster size and decide the optimal number of clusters based on the maximum score.

In the code below, we have calculated the Silhouette score for 2-10 cluster size.

In [ ]:
```
#Here we are taking the data transformed by the StandardScaler
silhouette_arr=[]
for k in range(2,10):
    k_means= KMeans(featuresCol='scaledFeatures', k=k)
    model = k_means.fit(scaled_data)
```

```
    predictions = model.transform(scaled_data)
    silhouette = evaluator.evaluate(predictions)
    silhouette_arr.append(silhouette)
    print('No of clusters:',k,'Silhouette Score:',silhouette)
```

In [ ]:

```
#Visualizing the silhouette scores in a plot
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1,1, figsize =(8,6))
ax.plot(range(2,10),silhouette_arr)
ax.set_xlabel('k')
ax.set_ylabel('cost')
```

In [ ]: