Welcome to exercise twp of week three of “Apache Spark for Scalable Machine Learning on BigData”. In this exercise we’ll work on clustering.

Let’s create our DataFrame again:

```
In [ ]: # delete files from previous runs
        !rm -f hmp.parquet*

        # download the file containing the data in PARQUET format
        !wget https://github.com/IBM/coursera/raw/master/hmp.parquet

        # create a dataframe out of it
        df = spark.read.parquet('hmp.parquet')

        # register a corresponding query table
        df.createOrReplaceTempView('df')
```

Let’s reuse our feature engineering pipeline.

```
In [ ]: from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAss
        embler, Normalizer
        from pyspark.ml.linalg import Vectors
        from pyspark.ml import Pipeline

        indexer = StringIndexer(inputCol="class", outputCol="classIndex")
        encoder = OneHotEncoder(inputCol="classIndex", outputCol="categoryVec"
        )
        vectorAssembler = VectorAssembler(inputCols=["x","y","z"],
                                          outputCol="features")
        normalizer = Normalizer(inputCol="features", outputCol="features_norm"
        , p=1.0)

        pipeline = Pipeline(stages=[indexer, encoder, vectorAssembler, normali
        zer])
        model = pipeline.fit(df)
        prediction = model.transform(df)
        prediction.show()
```

Now let’s create a new pipeline for kmeans.

```
In [ ]: from pyspark.ml.clustering import KMeans
        from pyspark.ml.evaluation import ClusteringEvaluator

        kmeans = KMeans(featuresCol="features").setK(14).setSeed(1)
        pipeline = Pipeline(stages=[vectorAssembler, kmeans])
        model = pipeline.fit(df)
        predictions = model.transform(df)

        evaluator = ClusteringEvaluator()

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

We have 14 different movement patterns in the dataset, so setting K of KMeans to 14 is a good idea. But please experiment with different values for K, do you find a sweet spot? The closer Silhouette gets to 1, the better.

https://en.wikipedia.org/wiki/Silhouette_(clustering)

In [ ]:
```
# please change the pipeline the check performance for different K, fe
el free to use a loop
```

Now please extend the pipeline to work on the normalized features. You need to tell KMeans to use the normalized feature column and change the pipeline in order to contain the normalizer stage as well.

In [ ]:
```
kmeans = KMeans($$).setK(14).setSeed(1)
pipeline = $$
model = pipeline.fit(df)

predictions = model.transform(df)

evaluator = ClusteringEvaluator()

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

Sometimes, inflating the dataset helps, here we multiply x by 10, let’s see if the performance inceases.

In [ ]:
```
from pyspark.sql.functions import col
df_denormalized = df.select([col('*'),(col('x')*10)]).drop('x').withCo
lumnRenamed('(x * 10)','x')
```

In [ ]:
```
kmeans = KMeans(featuresCol="features").setK(14).setSeed(1)
pipeline = Pipeline(stages=[vectorAssembler, kmeans])
model = pipeline.fit(df_denormalized)
predictions = model.transform(df_denormalized)

evaluator = ClusteringEvaluator()

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

Apache SparkML can be used to try many different algorithms and parametrizations using the same pipeline. Please change the code below to use GaussianMixture over KMeans. Please use the following link for your reference.

https://spark.apache.org/docs/latest/ml-clustering.html#gaussian-mixture-model-gmm

In [ ]:
```
from pyspark.ml.clustering import GaussianMixture

gmm = $$
pipeline = $$

model = pipeline.fit(df)

predictions = model.transform(df)

evaluator = ClusteringEvaluator()

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