Homework 2 - Distributed Data Managment

Install findspark, pyspark in case it is not installed - if running on Colab.

You can copy the whole notebook to your Google account and work on it on Colab via:

File -> Save a copy in Drive -> Open the copied notebook

```
# * When using the Docker workspace do not run this step *
IM_RUNNNING_ON_COLAB = True

if IM_RUNNNING_ON_COLAB:

  !pip install --force-reinstall pyspark==3.2
  !pip install findspark
```

Uplaod the data from Moodle, it's a zip file so simply unzip it

```
!unzip /content/random_data.parquet.zip
```

SparkSession is created outside your function

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
import pyspark
from time import time

def init_spark(app_name: str):
  spark = SparkSession.builder.appName(app_name).getOrCreate()
  sc = spark.sparkContext
```

```
  return spark, sc

spark, sc = init_spark('hw2_kmeans')
```

Load samples points

```
data_df = spark.read.parquet("random_data.parquet")
data_df.show(5)

# You can load the small sample for quick testing and reproducing results:

sample_df = spark.read.option("header",True) \
                    .option('inferSchema', True)\
                    .csv('sample_data_84.csv')
sample_df.show(5)
```

Create initials centroids

```
init_centroids = \
  spark.createDataFrame([[6.693, 7.782, 5.63],
                         [3.744, 4.341, 7.225],
                         [9.01, 7.8, 8.03],
                         [2.134, 1.59, 1.93]])
init_centroids.show()
```

Place your kmeans_fit function here

Don't forget to also add it in a seperate .py file named HW2_WET_[ID1]_[ID1]

```
# All of your imports should go here
# You cannot use any premade k-means nor import sklearn...

def kmeans_fit(data: pyspark.sql.DataFrame,
               init: pyspark.sql.DataFrame,
               k: int = 4
```

```
             k: int = 4,
             max_iter: int = 10):
    # imports
    from pyspark.sql import SparkSession
    from pyspark.ml.feature import VectorAssembler
    import pyspark.sql.functions as F
    spark = SparkSession.builder.getOrCreate()

    def check_convergence(prev_centroids, centroids):
      for i in range(len(centroids)):
        if list(prev_centroids[i]) != list(centroids[i]):
          return False
      return True

    # initialization:
    columns = data.columns
    converged = False
    data_points = data.select("*").persist()
    # add a column with number 2 to speed things up late
    data_points = data_points.withColumn("pow", F.lit(2))
    # create a list of centroids
    new_centroids = [list(centroid) for centroid in init.collect()]
    # print(centroids)
    for iteration in range(max_iter):
        if converged:
            break
        else:
          prev_centroids = new_centroids
          distances = []
          for i,centroid in enumerate(prev_centroids):
              distances.append(f'dist_from_c{i}')
              # for each centroid calculate sigma{(x_j-c_j)^2} by creating (x_j-c_j)^2 column for each j
              for j in range(len(centroid)):
                  data_points = data_points.withColumn(f"x_{j}_minus_c{j}_pow", F.expr(f"{columns[j]} - {centroid[j]}"))\
                                              .withColumn(f"x_{j}_minus_c{j}_pow", F.pow(f"x_{j}_minus_c{j}_pow", "pow"))
              # now sum (x_j-c_j)^2 columns to get euclidian distance from centroid i
              # no need for sqrt since its monotonicly increasing and distances are strictly positive
              data_points = data_points.withColumn(f'dist_from_c{i}', sum([F.col(f"x_{k}_minus_c{k}_pow") for k in range(len(centroid))]))

          # create a column with the assigned centroid for each point
          cond = F.expr("CASE " + " ".join([f"WHEN {c} = minimum THEN '{i}'" for i,c in enumerate(distances)]) + " END")
```

```
        data_points = data_points.withColumn('minimum', F.least(*distances)).withColumn("centroid_id", cond)

        # compute new centroids by averaging the points per centroid
        new_centroids = data_points.withColumn("centroid_id",F.col("centroid_id").cast("int"))\
                            .groupBy("centroid_id").avg(*columns).orderBy("centroid_id")
        #
        new_centroids = [list(centroid[1:]) for centroid in new_centroids.collect()]
        # check convergence
        if iteration > 1:
            converged = check_convergence(prev_centroids, new_centroids)

    centroids = spark.createDataFrame(new_centroids)
    vecAssembler = VectorAssembler(inputCols=centroids.columns,outputCol="centroids")
    return vecAssembler.transform(centroids).select("centroids")
```

Test your function output and run time

```
start_time = time()
out = kmeans_fit(data_df, init_centroids)
end_time = time()

print('Final results:')
out.show(truncate=False)
print(f'Total runtime: {end_time-start_time:.3f} seconds')
```

```
Final results:
+-----------------------------------------------------------+
|centroids                                                  |
+-----------------------------------------------------------+
|[6.500257701999981,6.499862152000027,6.500300249000017]  |
|[4.500187320000003,4.500350787999989,4.500139439999997]  |
|[8.499745406999969,8.50008152299997,8.49965887499999]    |
|[1.5000080700000005,1.499990830999997,1.500135027500003]|
+-----------------------------------------------------------+

Total runtime: 4.384 seconds
```

Expected results

For the given intialization centroids and the sample_df_84 you got on Moodle, the expected results is:

```
#  +--------------------------------------------------------+
#  |centroid                                                |
#  +--------------------------------------------------------+
#  |[4.496670602125162,4.495811688311686,4.50180401416766]   |
#  |[1.4891561106155216,1.5075798394290807,1.4981257805530763]|
#  |[8.501673279603231,8.490239925604461,8.505297582145058]  |
#  |[6.496611142694714,6.508118249005111,6.510762933484945]  |
#  +--------------------------------------------------------+
```

```
%%shell
jupyter nbconvert --to HTML DDM_HW2_STARTER_318155843_206567067.ipynb
```

```
[NbConvertApp] Converting notebook DDM_HW2_STARTER_318155843_206567067.ipynb to HTML
[NbConvertApp] Writing 606590 bytes to DDM_HW2_STARTER_318155843_206567067.html
```

Notice that the algorithm is deterministic, and we expect your results to be the same for at least 3 decimal numbers after the point.

The ordering of the centroids in the Dataframe and the title may be different.

Don't forget to run your function on the WHOLE data and show the results in the notebook's PDF and HTML :)

Also don't forget to also add your function in a seperate .py file named HW2_WET_[ID1]_[ID1]

✓ 1s completed at 7:28 PM