

✓ EECS 4415 - Task 3

High-Dimensional Data with Spark K-Means & PCA

✓ Setup

Let's set up Spark on your Colab environment. Run the cell below!

```
!pip install pyspark
!pip install -U -q PyDrive
!apt install openjdk-8-jdk-headless -qq
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

➞ Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.3)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from p
openjdk-8-jdk-headless is already the newest version (8u422-b05-1~22.04).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.

Now we import some of the libraries usually needed by our workload.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import pyspark
from pyspark.sql import *
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf
```

Let's initialize the Spark context.

```
# create the session
conf = SparkConf().set("spark.ui.port", "4050")

# create the context
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession.builder.getOrCreate()
```

✓ Data Preprocessing

In this Colab, rather than downloading a file from Google Drive, we will load a famous machine learning dataset, the [Breast Cancer Wisconsin dataset](#), using the `scikit-learn datasets` loader.

```
from sklearn.datasets import load_breast_cancer
breast_cancer = load_breast_cancer()
```

For convenience, we first

- construct a Pandas dataframe
- tune the schema
- and convert it into a Spark dataframe.

```
pd_df = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
df = spark.createDataFrame(pd_df)
```

```
def set_df_columns_nullable(spark, df, column_list, nullable=False):
    for struct_field in df.schema:
        if struct_field.name in column_list:
            struct_field.nullable = nullable
    df_mod = spark.createDataFrame(df.rdd, df.schema)
    return df_mod
```

```
df = set_df_columns_nullable(spark, df, df.columns)
df = df.withColumn('features', array(df.columns))
vectors = df.rdd.map(lambda row: Vectors.dense(row.features))
```

```
df.printSchema()
```

```
⇒ root
  |-- mean radius: double (nullable = false)
  |-- mean texture: double (nullable = false)
  |-- mean perimeter: double (nullable = false)
  |-- mean area: double (nullable = false)
  |-- mean smoothness: double (nullable = false)
  |-- mean compactness: double (nullable = false)
  |-- mean concavity: double (nullable = false)
  |-- mean concave points: double (nullable = false)
```

```

|-- mean symmetry: double (nullable = false)
|-- mean fractal dimension: double (nullable = false)
|-- radius error: double (nullable = false)
|-- texture error: double (nullable = false)
|-- perimeter error: double (nullable = false)
|-- area error: double (nullable = false)
|-- smoothness error: double (nullable = false)
|-- compactness error: double (nullable = false)
|-- concavity error: double (nullable = false)
|-- concave points error: double (nullable = false)
|-- symmetry error: double (nullable = false)
|-- fractal dimension error: double (nullable = false)
|-- worst radius: double (nullable = false)
|-- worst texture: double (nullable = false)
|-- worst perimeter: double (nullable = false)
|-- worst area: double (nullable = false)
|-- worst smoothness: double (nullable = false)
|-- worst compactness: double (nullable = false)
|-- worst concavity: double (nullable = false)
|-- worst concave points: double (nullable = false)
|-- worst symmetry: double (nullable = false)
|-- worst fractal dimension: double (nullable = false)
|-- features: array (nullable = false)
|   |-- element: double (containsNull = false)

```

With the next cell, we build the two data structures that we will be using throughout this Colab:

- `features`, a dataframe of Dense vectors, containing all the original features in the dataset;
- `labels`, a series of binary labels indicating if the corresponding set of features belongs to a subject with breast cancer, or not.

```

from pyspark.ml.linalg import Vectors
features = spark.createDataFrame(vectors.map(Row), ["features"])
labels = pd.Series(breast_cancer.target)

```

✓ Your task

If you run successfully the Setup and Data Preprocessing stages, you are now ready to cluster the data with the [K-means](#) algorithm included in MLlib (Spark's Machine Learning library). Set the `k` parameter to **2** and seed to **1**, fit the model, and then compute and print the [Silhouette score](#) (i.e., a measure of quality of the obtained clustering, here we use squared euclidean distance). Note: K-means in Spark's MLlib library is distributed by default.

IMPORTANT: use the MLlib implementation of the Silhouette score (via `ClusteringEvaluator`).

```
''' 8-9 lines of code in total expected but can differ based on your style.
The running time should be less than 1 minute'''
# YOUR CODE HERE
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

```
K = 2
SEED = 1
```

```
kmeans = KMeans().setK(K).setSeed(SEED)
model = kmeans.fit(features)
```

```
predictions = model.transform(features)
```

```
evaluator = ClusteringEvaluator()
```

```
silhouette = evaluator.evaluate(predictions)
print("Silhouette score = " + str(silhouette))
```

```
⇒ Silhouette score = 0.8342904262826145
```

Take the predictions produced by K-means, and compare them with the `labels` variable (i.e., the ground truth from our dataset).

Compute how many data points in the dataset have been clustered correctly (i.e., positive cases in one cluster, negative cases in the other), please use the best case scenario since the output cluster ids can be a permutation of labels.

HINT: you can use `np.count_nonzero(series_a == series_b)` to quickly compute the element-wise comparison of two series.

```
''' 4 lines of code in total expected but can differ based on your style.'''
# YOUR CODE HERE
#
labels = labels.reset_index(drop=True)

np.count_nonzero(predictions.select('prediction').toPandas()['prediction'] == labels)
```

```
⇒ 83
```

Now perform dimensionality reduction on the `features` using the [PCA](#) statistical procedure, available as well in MLlib.

Set the `k` parameter to **2**, effectively reducing the dataset size of a **15X** factor. Show top 20 rows.

''' 6 lines of code in total expected but can differ based on your style.

The running time should be less than 30 seconds.'''

YOUR CODE HERE

```
from pyspark.ml.feature import PCA
```

```
pca = PCA(k=K, inputCol="features", outputCol="pcaFeatures")
```

```
model = pca.fit(features)
```

```
pca_features = model.transform(features).select("pcaFeatures")
```

```
pca_features.show(20, truncate=False)
```

```

⇨ +-----+
  |pcaFeatures|
  +-----+
  |[-2260.013886292542,-187.96030122263687]|
  |[-2368.9937557820544,121.58742425815537]|
  |[-2095.6652015478608,145.11398565870124]|
  |[-692.6905100570509,38.57692259208171]|
  |[-2030.2124927427067,295.2979839927931]|
  |[-888.2800535760762,26.079796157025662]|
  |[-1921.0822124748454,58.807572473099455]|
  |[-1074.7813350047968,31.771227808469558]|
  |[-908.5784781618834,63.83075279044635]|
  |[-861.5784494075684,40.57073549705316]|
  |[-1404.5591306499475,88.23218257736251]|
  |[-1524.2324408687823,-3.2630573167779313]|
  |[-1734.385647746416,273.16267815114594]|
  |[-1162.914003223036,217.6348180834464]|
  |[-903.4301030498837,135.6151766608479]|
  |[-1155.8759954206853,76.8088938374218]|
  |[-1335.7294321308073,-2.4684005450354585]|
  |[-1547.2640922523092,3.805675972574516]|
  |[-2714.964765181216,-164.37610864258835]|
  |[-908.2502671870881,118.21642008223107]|
  +-----+
only showing top 20 rows

```

Now run K-means with the same parameters as above, but on the `pcaFeatures` produced by the PCA reduction you just executed.

Compute and print the Silhouette score, as well as the number of data points that have been clustered correctly.

''' 8-11 lines of code in total expected but can differ based on your style.'''

YOUR CODE HERE


```
kmeans = KMeans().setK(K).setSeed(SEED).setFeaturesCol("pcaFeatures")
```

```
model = kmeans.fit(pca_features)
```

```
pca_predictions = model.transform(pca_features)


evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(pca_predictions, {evaluator.featuresCol: 'pcaFeatures'})
print("Silhouette score = " + str(silhouette))
```

 Silhouette score = 0.8348610363444832

Report the number of data points that have been clustered correctly.

```
''' 1 line of code in total expected but can differ based on your style.'''
# YOUR CODE HERE
np.count_nonzero(pca_predictions.select('prediction').toPandas()['prediction'] == labels)
```

 83

Comment based on these scores on K-Means without and with PCA (2 sentences)

PCA yields slightly better silhouette score than without PCA. Although, the number of correct prediction remains the same.

Write a paragraph of conclusions below summarizing your insights.

It seems PCA itself impact little to the accuracy of the prediction. However, feature reduction by PCA makes the prediction faster compared to prediction with the original features. This difference in running time between PCA and non-PCA would be bigger as the original feature set has more features correlated to each other.

Once you obtained the desired results, **head over to eClass and submit your solution for this Colab!**