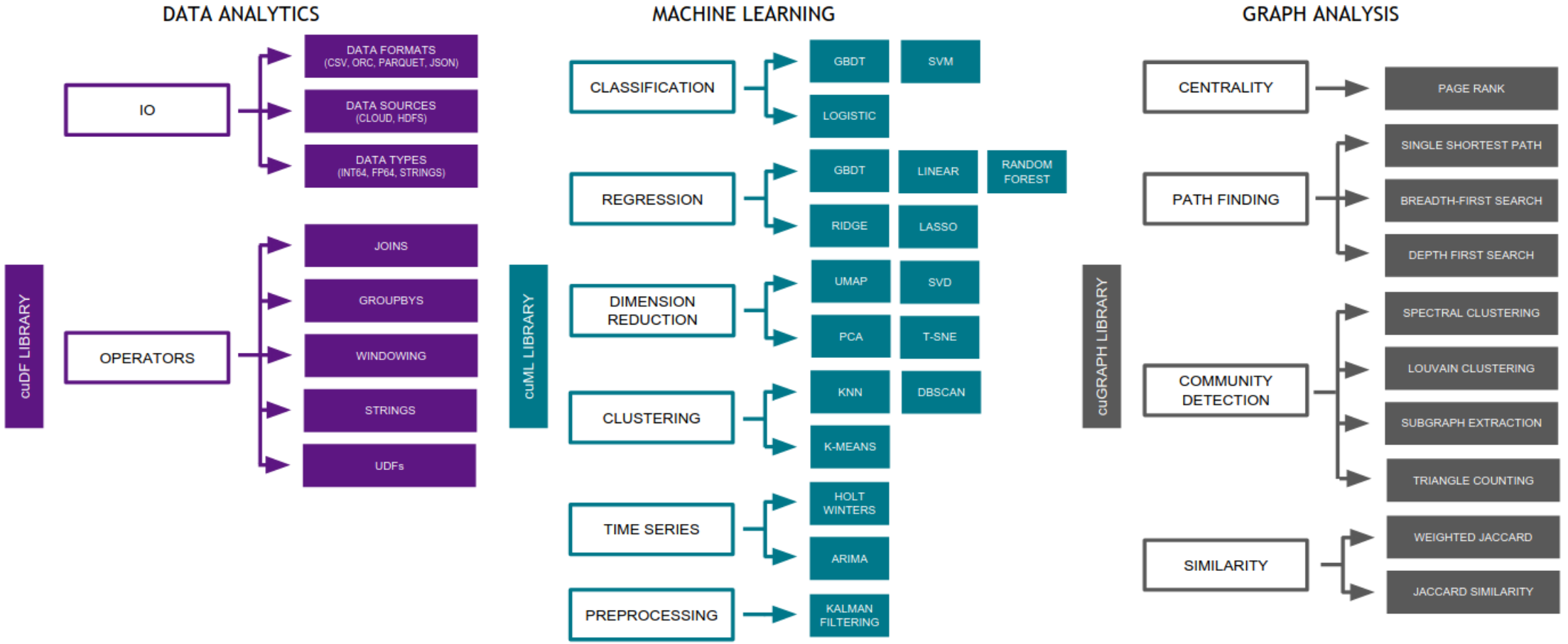
HPC for ML
Magurele Summer School

GPU Accelerated Distributed Data Science 2

Marco Celoria – CINECA

Magurele 9 July 2025

RAPIDS


DATA ANALYTICS
cuDF LIBRARY
IO
DATA FORMATS
(CSV, ORC, PARQUET, JSON)
DATA SOURCES
(CLOUD, HDFS)
DATA TYPES
(INT64, FP64, STRINGS)
OPERATORS
JOINS
GROUPBYS
WINDOWING
STRINGS
UDFs
MACHINE LEARNING
cuML LIBRARY
CLASSIFICATION
GBDT
SVM
LOGISTIC
REGRESSION
GBDT
LINEAR
RANDOM FOREST
RIDGE
LASSO
DIMENSION REDUCTION
UMAP
SVD
PCA
T-SNE
CLUSTERING
KNN
DBSCAN
K-MEANS
TIME SERIES
HOLT WINTERS
ARIMA
PREPROCESSING
KALMAN FILTERING
GRAPH ANALYSIS
cuGRAPH LIBRARY
CENTRALITY
PAGE RANK
PATH FINDING
SINGLE SHORTEST PATH
BREADTH-FIRST SEARCH
DEPTH FIRST SEARCH
COMMUNITY DETECTION
SPECTRAL CLUSTERING
LOUVAIN CLUSTERING
SUBGRAPH EXTRACTION
TRIANGLE COUNTING
SIMILARITY
WEIGHTED JACCARD
JACCARD SIMILARITY

Linear Regression

- In regression.py, we consider a linear regression on a synthetic dataset.
- The dataset is $\{Y_j, X_{j1}, \ldots, X_{jM}\}$ with $j=1,\ldots,N$ for N samples and M features, generated from a linear model

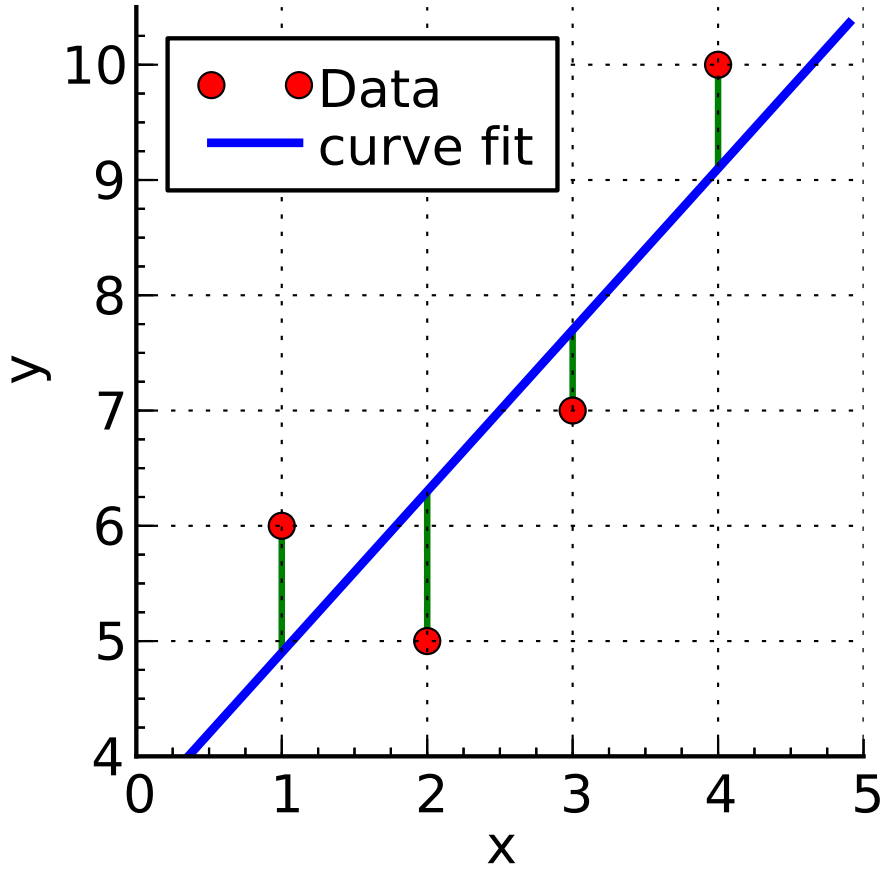
$$Y_i = \sum_{f=1}^{M} X_{if} c_f + b$$

- where c_f with $f=1,\ldots,M$ is a vector of m coefficients and b is the bias.
- Among the M features, we suppose that only K features are informative, that is only $c_1, \ldots, c_K$ are different from zero.
- Given the dataset X, y generated from the true underlying coefficients c_M and the true underlying bias b, the goal is to learn the coefficients C_M and the bias B from data, by minimizing the Mean Squared Error.

$$\hat{Y}_i = \sum_{f=1}^{M} X_{if} C_f + B$$

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2$$

Linear Regression



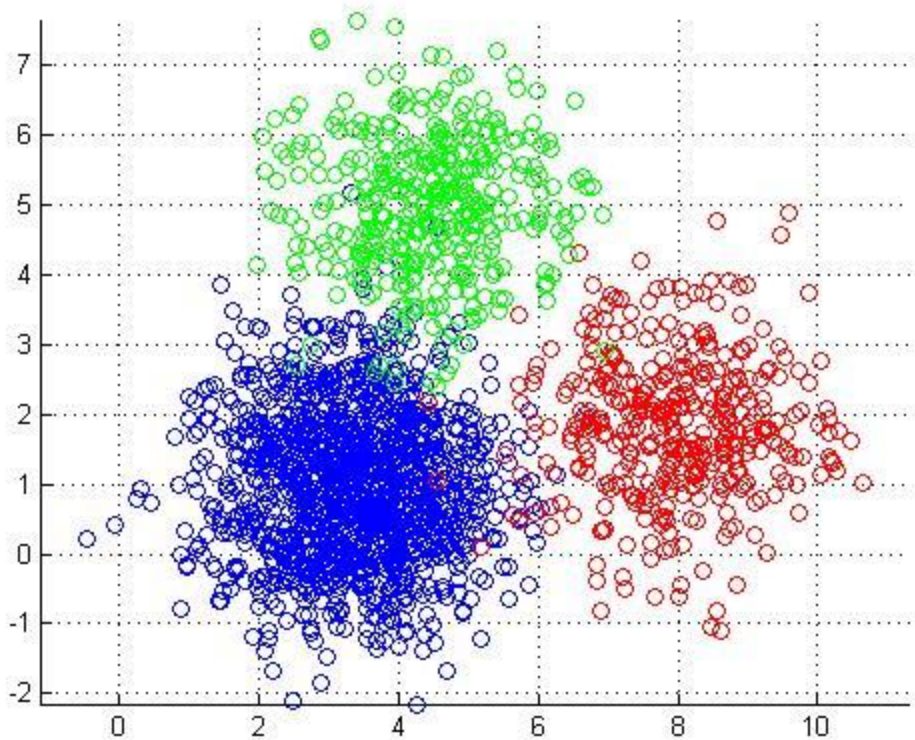
K-Means

- In clustering.py, we consider a clustering on a synthetic dataset
- Partitioning blobs of points into clusters such that points within the same cluster exhibit greater similarity to one another than to those in other clusters
- Given a set of observations (X_1, X_2, …, X_n), where each observation is a D-dimensional real vector, k-means clustering aims to partition the n observations into k (k< n) sets S = {S_1, S_2, …, S_k} so as to minimize the within-cluster sum of squares (variance)

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

- Where μ_i is the mean also called centroid of points in S_i $\boldsymbol{\mu_i} = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x},$

K-Means

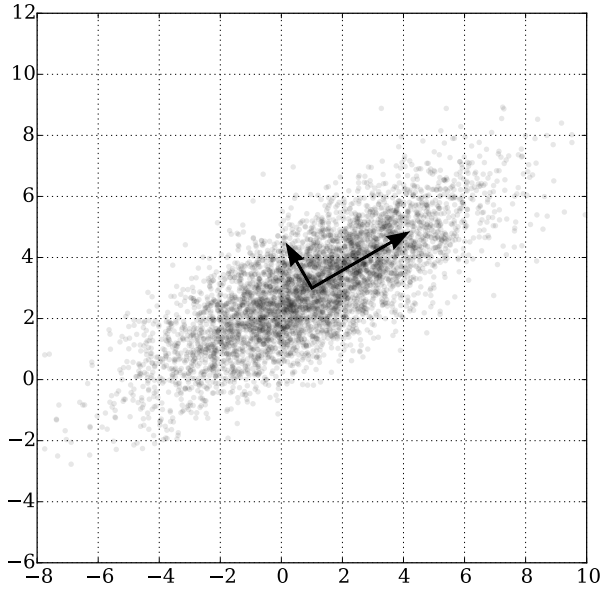


PCA

- In `pca.py`, we apply the Principal component analysis (PCA) on a synthetic dataset.
- PCA is a linear dimensionality reduction method used in exploratory data analysis and processing
- The data is linearly transformed onto a new coordinate system such that the directions (principal components) capturing the largest variation in the data can be easily identified.
- PCA is a statistical procedure that reduces the number of dimensions in large datasets to principal components that retain most of the original information,
  thus summarizing the information content in large data tables.
- Suppose we have a tabular data, i.e. large matrix **X**, where the N rows represent N samples, and each of the M columns is a particular kind of feature.
- The covariance matrix of C is given by $C = \frac{1}{N-1}(X - \bar{X})^T(X - \bar{X})$ $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_{ij}$

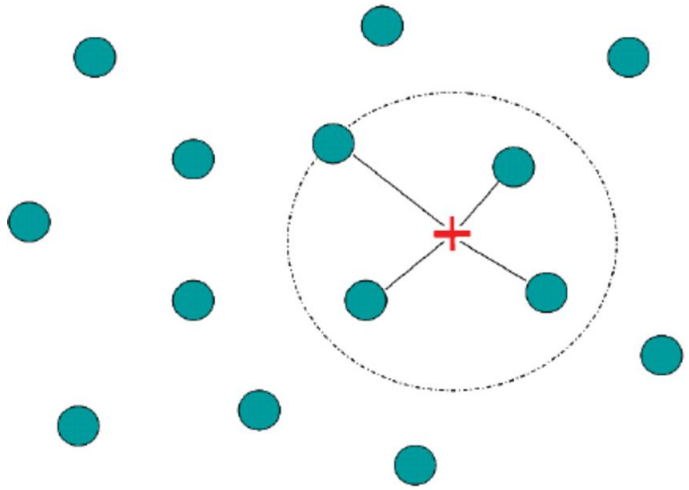
PCA

- The covariance matrix C is symmetric and positive semi-definite, with non-negative real eigenvalues
- Each entry Cij quantifies the correlation of the i and j features across all experiments
- The principal components are the eigenvectors of C
- CV = VD
- They define a change of coordinates in which C is diagonal
- The columns of the eigenvector matrix V are the principal components
- The elements of the diagonal matrix D are the variances of the data along these directions



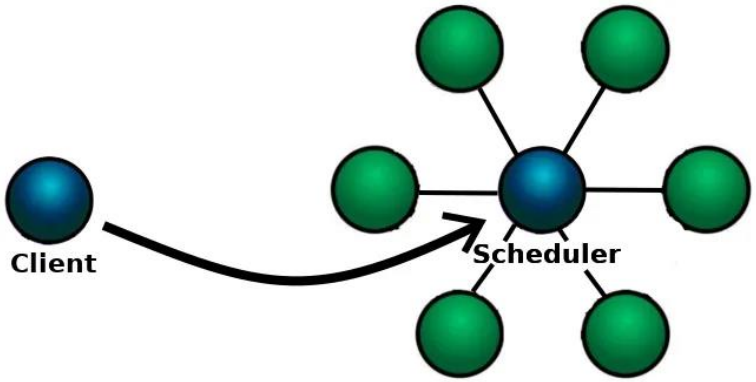
k-Nearest Neighbors (kNN)

- When trying to predict outcomes of a particular observation in a dataset, one of the most natural and intuitive options is to consider similar observations and check their outcomes
- kNN is used for classification, regression, or search algorithms
- For each new observation, we find the k most similar (defined in terms of distance metric) observations with known outcomes
- Labelled observations are gathered into a data structure known as *index*
- New unlabelled observations upon which search is performed are the *query*



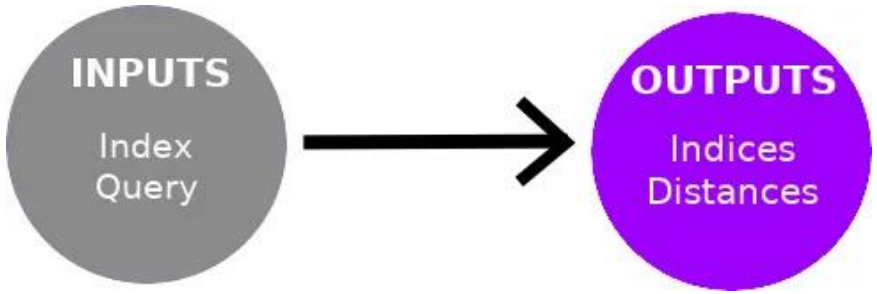
How does distributed kNN work?

- Based on https://medium.com/rapids-ai/scaling-knn-to-new-heights-using-rapids-cuml-and-dask-63410983acfe
- For large datasets, we scale the algorithm by splitting the workload to multiple GPUs, using the multi-node, multi-GPU kNN algorithm
- Workers, each attached to a single-GPU, directly interact with each other
- A scheduler first lists the workers available in the cluster
- A client initiate the loading of the data (partitioned across workers)
- Then it will trigger the distributed operations to perform the search
- All the workers run the same set of instructions on their local data partitions



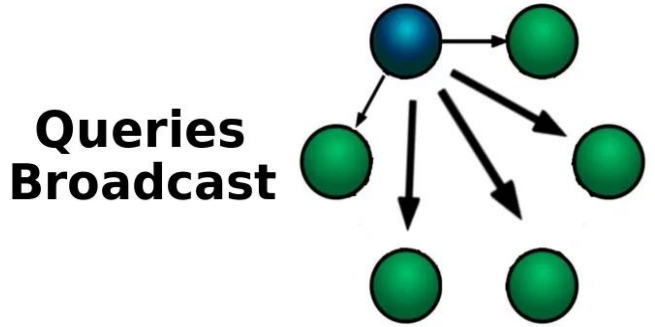
Partitions

- Data is stored in partitions shared among the workers, a distributed array
- **Input index:** it contains reference observations **(n_references x n_features)**
- **Input query:** it contains observations for which to find the nearest neighbors, size **(n_queries x n_features)**
- **Output indices:** it will contain the indices of the nearest neighbors in the index for each of the query points, size **(n_queries x n_neighbors)**
- **Output distances:** it will contain the distances of the nearest neighbors toward their respective query points, size **(n_queries x n_neighbors)**

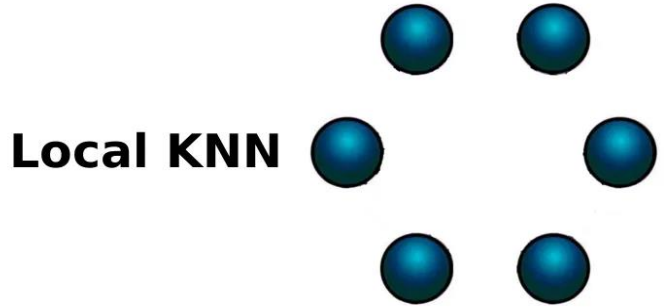


Distributed Operations

- Queries are processed sequentially as a series of batches. For each batch:
- **Queries broadcasting:**
  The unique owner of the currently processed batch of queries broadcasts it to all the other workers

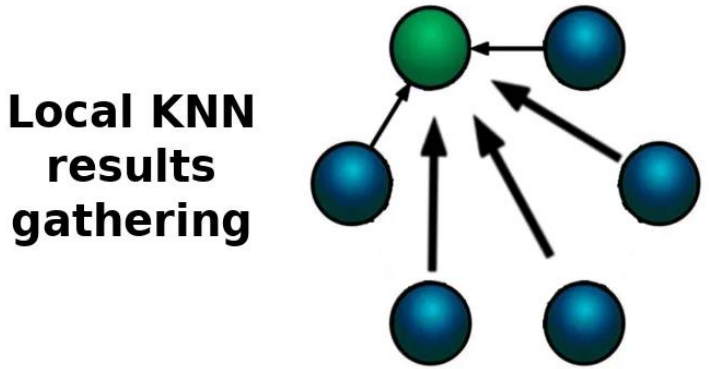


- **Local kNN:** All workers run a local kNN.
  The local kNN searches for the nearest neighbors of the currently processed batch of queries in its locally stored index partitions.
  The produced indices and distances correspond to the local nearest neighbors for these query points.



Distributed Operations

- **Local results gathering:** Workers send the result of their local kNN back.
  The owner of the currently processed batch of queries collects all these data.
  Workers then start processing the next batch



- **Reduce:** The owner performs a reduce operation on the received data (merging the results of the local kNN).
  This produces a set of global nearest neighbors' indices for each of the queries.
  Along with the corresponding distances between query and newly found nearest neighbors.