Scalable Density Clustering for Spark

THOMAS TRIPLET, PH.D., ENG.

MARCH 9TH 2016
TECHNOLOGIES BIG-DATA

• Hadoop Core
  – **HDFS**: Système de fichiers distribué
  – **YARN**: Gestion des ressources CPU et planification
  – **MapReduce**: Traitement en lot (batch) des données à grande échelle

• Écosystème Hadoop
• Popular distributed in-memory computing framework
• 10-100x faster than Hadoop MapReduce and low latency
• Linear horizontal scalability
• Fault tolerant (RDDs)
• Applications range from long-running batch jobs to stream processing
• High-level Scala, Java, Python and R APIs
AGENDA

• Clustering algorithms (unsupervised learning)
  – Distance-based (k-means)
  – Density-based (DBSCAN)

• PatchWork
  – Algorithm
  – Results
  – Performance

• Conclusion

• Future Work
INTRODUCTION: MACHINE LEARNING

Supervised Learning

- Class labels are known and predefined
- Training and testing datasets are (manually) labeled with same classes
- Goal is to learn function/rule that can classify new data points
- Examples: SVMs, Neural nets, Bayesian classifiers, Decision trees...

Unsupervised Learning (clustering)

- Class labels of the data are unknown
- Group/cluster similar data points without prior knowledge
- Goal is to discover structure or pattern in the data
- Examples: k-means, EM, DBScan, HCA...
INTRODUCTION: MACHINE LEARNING

Supervised Learning

- Class labels are known and predefined
- Training and testing datasets are (manually) labeled with same classes
- Goal is to learn function/rule that can classify new data points
- Examples: SVMs, Neural nets, Bayesian classifiers, Decision trees...

Unsupervised Learning (clustering)

- Class labels of the data are unknown
- Group/cluster similar data points without prior knowledge
- Goal is to discover structure or pattern in the data
- Examples: k-means, EM, DBScan, HCA...
### INTRODUCTION: CLUSTERING

#### Distance-based
- Popular algorithm: k-means (implemented in MLLib)
- Relies on distance function between data points
- Easy to implement
- Linear complexity (big-data)
- Easy to distribute
- Discovers spherical clusters of similar sizes only
- Sensitive to noise and local optima
- Prior knowledge of \( k \).

#### Density-based
- Popular algorithm: DBScan (not in MLLib)
- Relies on the density of data points in feature space
- Natural protection against noise and outliers
- Discovers clusters of arbitrary shape and size
- No prior knowledge of \( k \)
- Discovers clusters of similar densities only
- Quadratic complexity: not scalable
**INTRODUCTION: CLUSTERING**

### Distance-based
- Popular algorithm: *k*-means (implemented in MLLib)
- Relies on distance function between data points
- Easy to implement
- Linear complexity
- Easy to distribute
- Discovers spherical clusters of similar sizes only
- Sensitive to noise and local optima
- Prior knowledge of $k$

### Density-based
- Popular algorithm: DBScan (not in MLLib)
- Relies on the density of data points in feature space
- Natural protection against noise and outliers
- Discovers clusters of arbitrary shape and size
- No prior knowledge of $k$
- Discovers clusters of similar densities only
- Quadratic complexity: not scalable
2 main steps:

1. `createCells(dataPoints) → cells → RDD[(string, int)]`

2. `createClusters(cells) → clusters`
STEP 1: CELL CREATION
STEP 1: CELL CREATION

Parameter:
Grid size: $\varepsilon \in \mathbb{R}^D$
STEP 1: CELL CREATION

Coordinates:

- (-1,2 ; 4)
- (-1,3 ; 4)
- (-2,2 ; 4)
- (-3,4 ; 1)
- (2,3 ; 4)
- (2,4 ; 3)
- (3,3 ; 3)
- (3,4 ; 3)
STEP 1: CELL CREATION

\[
\text{setOfCells} = \text{dataPoints}
\]
\[
.\map(P \mapsto (\text{cellID}(P), 1))
\]
\[
.\reduceByKey(_ + _)
\]

\text{cellID}(P)

- Runs in constant time \(O(1)\)
- Depends on \(P\) and \(\epsilon\) only (no pre-processing)
STEP 2: CLUSTER CREATION

*Cluster creation using $m_\varepsilon$ cells (with $m_\varepsilon \ll n$ in practice)*

1. Filter out noise (optional) .filter(…)

2. Sort $setOfCells$ by decreasing density .sortBy(…)

3. For all cells, repeat 4–5 .foreach(

4. Create cluster with first cell (seed)

5. Expand cluster with adjacent cells (while density is close enough to seed) )

6. Filter out spatially small clusters (optional) .filter(…)


EXPERIMENTAL SETUP

- 6 servers, each with:
  - Intel Xeon E5-2650 8 cores @2.6GHz
  - 192GB memory
  - 30TB storage

- Cloudera CDH 5.4.0
- Apache Spark 1.3
DATASETS

**Aggregation**

**Compound**

**Jain**

**Spiral**
RESULTS (JAIN DATASET)

- **K-means**
- **DBScan**
- **PatchWork**
RESULTS (SPIRAL DATASET)

K-means

DBScan

PatchWork
RESULTS (AGGREGATION DATASET)

K-means

DBScan

PatchWork
RESULTS (COMPOUND DATASET)

- K-means
- DBScan
- PatchWork
PERFORMANCE: SCALABILITY

Normalized execution-time

Number of servers

MLLib k-means||

PatchWork
CONCLUSION

• **Benefits**
  – High-performance
    ▪ Linear computational complexity (DBScan has quadratic complexity)
    ▪ Significantly faster than Spark MLLib k-means on large datasets
  – Linear horizontal scalability
  – Can discover clusters of arbitrary shapes
  – Can discover *spatially large* clusters in addition to *dense* clusters
  – Natural protection against noise/outliers
  – No prior knowledge of number of clusters

• **Limitations**
  – High-dimensional datasets (computing neighborhood of cell is exponential with $D$)
  – Choice of parameter $\varepsilon$ can greatly impact performance
FUTURE WORK

• Tests against new clustering algorithms available in Spark 1.6

• Better distribution of step 2

• Indexing for region query using R-trees

• Streaming version
Q & A

Contact: thomas.triplet@crim.ca

Availability: https://github.com/crim-ca/patchwork (MIT Licence)