Clustering in KDD

Javier Béjar

CS - MIA

AMLT - 2016/2017
1 Introduction
   - Motivations

2 Scalability Strategies

3 Scalable Algorithms
1 Introduction

2 Scalability Strategies

3 Scalable Algorithms
Clustering in KDD

- One of the main tasks in the KDD process is the analysis of data when we do not know its structure.

- This task is very different from the task of prediction in which we know the goal function and we try to approximate it.

- A great part of the KDD tasks are non-supervised problems (KDNuggets poll, 2-3 most frequent task).

- Problems: Scalability, arbitrary cluster shapes, limited types of data, finding the correct parameters, ...

- There are some new algorithms that deal with these kind of problems.
1 Introduction

2 Scalability Strategies

3 Scalable Algorithms
Strategies for cluster scalability

- **One-pass**
  - Process data as a stream

- **Summarization/Data compression**
  - Compress examples to fit more data in memory

- **Sampling/Batch algorithms**
  - Process a subset of the data and maintain/compute a global model

- **Approximation**
  - Avoid expensive computations by approximate estimation

- **Paralelization/Distribution**
  - Divide the task in several parts and merge models
Scalability Strategies

One pass

- This strategy is based on incremental clustering algorithms
- They are cheap but order of processing affects greatly their quality
- Although they can be used as a preprocessing step

Two steps algorithms

1. A large number of clusters is generated using the one-pass algorithm
2. A more accurate algorithm clusters the preprocessed data
Data Compression/Summarization

- Not all the data is necessary to discover the clusters
- Discard sets of examples and summarize by:
  - Sufficient statistics
  - Density approximations
- Discard data irrelevant for the model (do not affect the result)
Approximation

- Not using all the information available to make decisions
  - Using K-neighbours (data structures for computing k-neighbours)
- Preprocessing the data using a cheaper algorithm
  - Generate batches using approximate distances (eg: canopy clustering)
- Use approximate data structures
  - Use of hashing or approximate counts for distances and frequency computation
Scalability Strategies

Batches/Sampling

- Process only data that fits in memory
- Obtain from the data set:
  - Samples (process only a subset of the dataset)
    - Determine the size of the sample so all the clusters are represented
  - Batches (process all the dataset)
Paralelization/Distribution/Divide&Conquer

- Paralelization of clustering usually depends on the specific algorithm
- Some are difficult to parallelize (e.g., hierarchical clustering)
- Some have specific parts that can be solved in parallel or by Divide&Conquer
  - Distance computations in k-means
  - Parameter estimation in EM algorithms
  - Grid density estimations
  - Space partitioning
- Batches and sampling are more general approaches
  - The problem is how to merge all the different partitions
1 Introduction

2 Scalability Strategies

3 Scalable Algorithms
Patra, Nandi, Viswanath, **Distance based clustering method for arbitrary shaped clusters in large datasets** Pattern Recognition, 2011, 44, 2862-2870

- **Strategy:** One pass + Summarization

- The leader algorithm is used as a one pass summarization using Leader algorithm (many clusters)

- Single link is used to cluster the summaries

- Theoretical results guarantee the equivalence to SL at top levels

- Summarization makes the algorithm independent of the dataset size (depends on the radius used on the leader algorithm and the volume of the data)

- Complexity $O(c^2)$
One pass + Single Link

1st Phase
Leader Algorithm

2nd Phase
Hierarchical Clustering
Zhang, Ramakrishnan, Livny *BIRCH: An Efficient Data Clustering Method for Very Large Databases* (1996)

- Strategy: One-pass + Summarization
- Hierarchical clustering with limited memory
- Incremental algorithm
- Based on probabilistic prototypes and distances
- We need two pass from the database
- Based on an specialized data structure named CF-tree (Clustering Feature Tree)
BIRCH (CF-tree)

- It is a balanced n-ary tree that contains groups represented by probabilistic prototypes.
- Leaves can contain as much as $L$ prototypes and its radius can not be more than $T$.
- Each non terminal node has a fixed branching factor ($B$), each element is a prototype that summarizes its subtree.
- The choice of the parameters is crucial because we could fill all available space during the process.
- This can be solved by changing the parameters values (basically $T$) and recompressing the tree. In fact $T$ determines the granularity of the final groups.
BIRCH - Insertion algorithm

1. Traverse the tree until a leave is found and choose the nearest prototype
2. On this leave we could introduce the instance in an existing group or create a new prototype depending on if the distance is greater than parameter $T$ or not
3. If the current leave has not space for the new prototype we proceed to create a new terminal node and we distribute the prototypes among the current node and the new node
4. The distribution is performed choosing the two most different prototypes and dividing the rest using their proximity to these two prototypes
5. This division will create a new node in the ascendant node, if the new node exceeds the capacity of the father we will split it and we will continue the process until the root of the tree is reached if necessary
6. Additionally we could perform merge operations to compact the tree and reduce space
BIRCH - Insertion Algorithm
Phase 1: Construction of the CF-tree, we obtain a hierarchy that summarizes the database as a set of groups which granularity is defined by $T$

Phase 2: Optionally we modify the CF-tree in order to reduce its size by merging near groups and deleting outliers

Phase 3: We use the prototypes inside the leaves of the trees as new instances and we run a clustering algorithm with them (for instance K-means)

Phase 4: We refine the groups assigning the instances from the original database to the prototypes obtained in the previous phase
One pass + CFTREE (BIRCH)

1st Phase - CFTree

2nd Phase - Kmeans

Scalable Algorithms
Scalable K-means clustering

Bradley, Fayyad, Reina, *Scaling Clustering Algorithms to Large Databases*, Knowledge Discovery and Data Mining (1998)

- **Strategy**: Sampling + Summarization
- Clustering algorithms need to have all data in main memory to perform their task
- We try to obtain scalability looking for an algorithm that:
  - Only look at the data one time
  - To be anytime (always a result is available)
  - To be incremental (to cluster more data we do not need to start from scratch)
  - To be suspendable (we can restart the process from the actual state)
  - To use limited memory
Scalable K-means clustering (Algorithm)

- Obtain a sample that fits in memory
- Update the actual model
- Classify new instances as:
  - Necessary
  - Discardable (We keep their information as sufficient statistics)
  - Summarizable using data compression
- Decide if the model is stable or we keep clustering more data
Canopy Clustering

McCallum, Nigam, Ungar *Efficient clustering of high-dimensional data sets with application to reference matching* (2002)

- Strategy: Divide&Conquer + Approximation
- The approach is based on a two stages clustering
- The first stage can be seen as a preprocess determining the neighborhood of the densities of examples and reducing the number of distances that have to be computed on the second stage
- This first stage is the called *canopy clustering*, relies on a cheap distance and two parameters $T_1 > T_2$
- This parameters are used as two centered spheres that determine how to classify the examples.
Canopy Clustering - Algorithm

Algorithm:
1. One example is picked at random and the cheap distance from this example to the rest is computed
2. All the examples that are at less than T2 are deleted and included in the canopy
3. The points at less than T1 are added to the canopy of this examples without deleting them
4. The procedure is repeated until the example list is empty
5. Canopies can share examples

After this procedure the examples can be clustered using different algorithms

For agglomerative clustering only the distances among the examples in the canopies have to be computed
Canopy Clustering - Algorithm

1st Canopy

2nd Canopy

3rd Canopy
Canopy Clustering - Algorithm

For EM and K-means there are different methods:

- **Method 1:** Prototypes are associated with canopies, only distances inside a canopy are computed and only the examples inside a canopy can influence a prototype. The number of prototypes per canopy have to be decided. The EM or K-means only use for each prototypes the examples of the associated canopies.

- **Method 2:** Use a set of prototypes covering the dataset and then distribute them in the canopies. We allow data in other canopies to influence the prototypes but only using the mean of their canopies.

- **Method 3:** We allow to create or merge prototypes inside a canopy avoiding to have prototypes that cover examples that belong to more than one canopy.
Mini-batch K-means

Sculley, **Web-scale k-means clustering** Proceedings of the 19th international conference on World wide web, 2010, 1177-1178

- Strategy: Sampling
- Apply K-means to a sequence of bootstrap samples of the data
- Each iteration the samples are assigned to prototypes and the prototypes are updated with the new sample
- Each iteration the weight of the samples is reduced (learning rate)
- The quality of the results depends on the size of the batches
- Convergence is detected when prototypes are stable
Mini-batch K-means (algorithm)

**Given:** k, mini-batch size b, iterations t, data set X

Initialize each \( c \in C \) with an \( x \) picked randomly from \( X \)

\( v \leftarrow 0 \)

for \( i \leftarrow 1 \) to \( t \) do

\( M \leftarrow b \) examples picked randomly from \( X \)

for \( x \in M \) do

\( d[x] \leftarrow f(C,x) \)

for \( x \in M \) do

\( c \leftarrow d[x] \)

\( v[c] \leftarrow v[c] + 1 \)

\( \eta \leftarrow \frac{1}{v[c]} \)

\( c \leftarrow (1-\eta)c + \eta x \)
CURE


- Strategy: Sampling + Divide & Conquer
- It uses hierarchical agglomerative clustering
- It is scalable and is capable of treating outliers
- Scalability is obtained using sampling techniques and partitioning the dataset
- It uses a set of representatives \((c)\) for cluster instead of centroids, this allows to find non spherical groups
- The distance is computed as the nearest pair of representatives among groups
- The clustering algorithm is agglomerative and merges pairs of groups until \(k\) groups are obtained
CURE - Algorithm

1. Draws a random sample from the dataset
2. Partitions the sample in \( p \) groups
3. Executes the clustering algorithm on each partition
4. Deletes outliers
5. Runs the clustering algorithm on the union of all groups until it obtains \( k \) groups
6. Label the data accordingly to the similarity to the \( k \) groups
CURE - Algorithm

DATA

Sampling+Partition

Clustering partition 1

Clustering partition 2

Join partitions

Labelling data
Viswanath, Babu, **Rough-DBSCAN: A fast hybrid density based clustering method for large data sets** Pattern Recognition Letters, 2009, 30, 1477 - 1488

- Strategy; One-pass + Summarization
- Two stages algorithm:
  1. Preprocess using the leader algorithm
     - Determine the instances that belong to the higher densities and their neighbours
  2. Apply DBSCAN algorithm
     - Determine the densities for the selected instances
     - Approximate the values of the densities from their distances and the sizes of the neighbor
     - Assign the neighbors accordingly to the found densities
Python Notebooks

This Python Notebook has examples comparing K-means algorithm with two scalable algorithms Mini Batch K-means and BIRCH

- Clustering DM Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)