Introduction to Algorithms for Data Mining and Machine Learning

Chapter 6: Data Mining Techniques

Xin-She Yang

For details, please read the book:

Xin-She Yang, **Introduction to Algorithms for Data Mining and Machine Learning**, Academic Press/Elsevier, (2019).

Data Type

Data mining includes many different techniques such as clustering and classification, feature selection and machine learning techniques.

Data

- Structured data: With a fixed structure (e.g., images, databases).
- Unstructured data: Can be any form without a specific structure (e.g., social media, web pages, business transactions).
- Mixed: Some data can be the mixture of the previous two types (though this can also be considered unstructured data in some literature).

Big Data: 5 Vs

- Volume: The volume of data has dramatically increased, driven by social media/Internet.
- Velocity: The high rate of data accumulation. More than 20 trillion GB per year.
- Variety: The diverse variety of data from different sources and media.
- Value: The purpose of analysis is to extra useful features and gain insight. That is, to make sense of the data.
- Veracity: The quality and accuracy of the solutions, subject to dynamic changes or uncertainties.

Distance Metric

Proper distance metrics should be defined in a D-dimensional space between two data points $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{x} = (x_1, x_2, ..., x_D)^T, \quad \boldsymbol{y} = (y_1, y_2, ..., y_D)^T.$$

Cartesian Distance

The general distance $d(\boldsymbol{x}, \boldsymbol{y})$ can be defined as L_p-norm

$$d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_p = \Big(\sum_i^D |x_i - y_i|^p \Big)^{1/p}.$$

In case of $p = 2$, it becomes the standard Euclidean or Cartesian distance.

Manhattan Distance

The Manhattan distance is the L_1-norm, defined by

$$D_m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^D |x_i - y_i|.$$

Other Distance Metrics

- Jaccard' similarity: Jaccard's similarity index of two sets U and V is defined as

$$J(U,V) = |U \cap V|/|U \cup V|,$$

which leads to $0 \leq J(U,V) \leq 1$, and the Jaccard distance is defined as

$$d_J(U,V) = 1 - J(U,V).$$

- Edit distance: The edit distance between two strings U and V is the smallest number of insertions and deletions of single characters that will convert U to V. For example, U='abcde' and V='ackdeg', the edit distance is d(U,V)=3. By deleting b, inserting k after c, and then inserting g after e, string U can be converted to V.
- Hamming distance: The Hamming distance is the number of components which two vectors/strings differ. For example, the Hamming distance between 10101 and 11110 is 3.

There are some distance metrics such as the fuzzy distance and fuzzy relationship. Depending on the context and applications, 'time' can also be considered 'distance'.

Even for the same data set and same algorithm, different distance metrics used for analysis may lead to (very) different results.

Clustering

For a given set of n observations, the aim is to divide them into some clusters [say, k different clusters] so as to minimize certain clustering measures/objectives.

Hierarchy clustering usually works well for small datasets. It starts with every point in its own cluster (that is, $k = n$ for n data points), followed by a simple iterative procedure.

Algorithm 1: Hierachy clustering algorithm

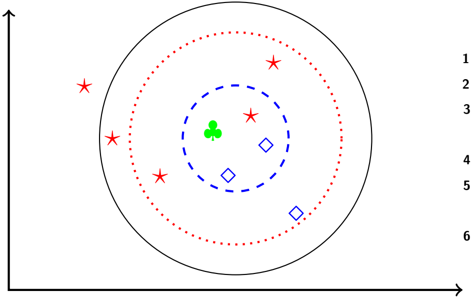
```
Each point belongs to its own cluster $k = n$;
for (all data points) do
    Choose two nearest clusters to merge into one cluster;
    Update $k \leftarrow k - 1$;
    Repeat until the metric measure goes up;
```

Here, k is a hyper-parameter (usually unknown, may be not unique), which needs some tuning and parametric studies in practice.

The complexity of this algorithm is $O(n^3)$ where n is the number of points. For $n = 10^9$ (a billion data points), this can lead to $O(10^{27})$ floating-point operations, which is quite computationally expensive.

kNN

The k-nearest neighbour (kNN) algorithm is a voting algorithm as it uses k sample points in terms of a given distance measure to determine the class of the data point under consideration. The main steps of the kNN can be summarized as the pseudocode.



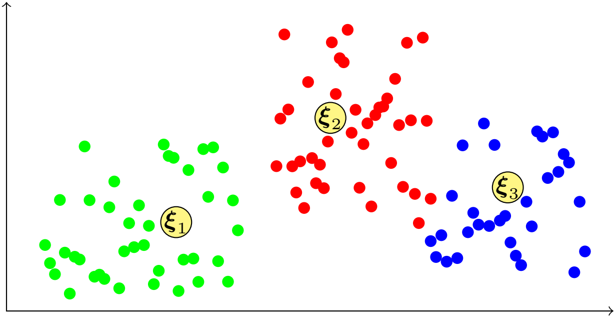
Algorithm 2: kNN Algorithm

```
Define $k$;
while (stopping criterion is not met) do
    Compute distances from other data points to point $i$;
    Sort the computed distances;
    Select the $k$ points with smallest distances;
    Assign the test point to the class by the simple majority;
```

The main idea of a simple classification problem is shown in the figure with two classes: stars ($\star$) and diamonds ($\diamond$). The task is to determine what class ♣ should belong?

If we use $k = 3$ (the smaller dashed circle), then the unknown ♣ should belong to $\diamond$. If we use $k = 7$ (the bigger solid circle), then this ♣ should be classified as $\star$. However, if we use $k = 6$ (the dotted circle), it would be difficult to classify ♣ because it is a tie. This highlights the importance of choosing the right parameter k and its sensitivity. In general, k should be odd, typically $k < \sqrt{n}$ (number of points).

k-means clustering



The k-means clustering method is to divide a set of n observations into k different clusters $(S_1, S_2, ..., S_k)$ such that each point $\boldsymbol{x}_i$ belongs to the nearest cluster with the shortest distance to its corresponding cluster mean/centroid $\boldsymbol{\xi}_j$ $(j = 1, 2, ..., k)$. That is

$$\text{minimize} \quad \sum_{j=1, \boldsymbol{x}_i \in S_i}^{k} ||\boldsymbol{x}_i - \boldsymbol{\xi}_j||^2,$$

where $1 < k \leq n$ and typically $k \ll n$.

This k-means method for dividing n points into k clusters can be summarized schematically as the pseudocode.

Algorithm 3: K-means Algorithm.

```
Define $k$;
Choose randomly $k$ points as the initial centroids of the $k$ clusters;
for (each remaining point $i$) do
    Assign $i$ to the cluster with the closest centroid;
    Update the centroid of that cluster (containing $i$);
```

Some issues concerning k-means clustering

- Initialization of k clusters may not be efficient. In some case, all initial points might belong to the same cluster. This issue may be avoidable by using other methods, including choosing the initial points that are far from each other as possible.
- If the results are not good, random restart may be needed to avoid too much dependence on the initial points.
- The complexity of this method can be $O(n^{kd+1}\log(n))$ where d is the dimension of the data. For $n = 10^6$, $k = 3$ and $d = 2$, this becomes $O(10^{43})$ (extremely computationally expensive). Thus, the k-means method is not suitable for big data.

All major software packages for data mining have implemented kNN and k-means methods.

Exercise: `Try the kNN and k-means in R or Python with some random data.`

Decision Trees

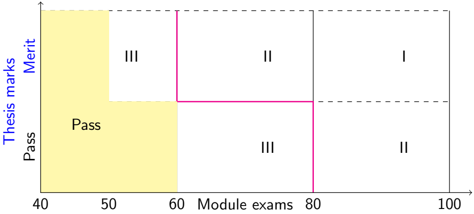
- The decision tree algorithm is a widely used algorithm for classification, which uses attribute values to partition the decision space into smaller subspaces iteratively.
- It is a divide-and-conquer approach, starting with a root node and gradually grow to a final classification (or leaf).
- The decision processes can be represented graphically as a tree, though this tree is usually drawn upside down. These trees are directed trees, but they are not unique, depending on which attribute(s) to use first.

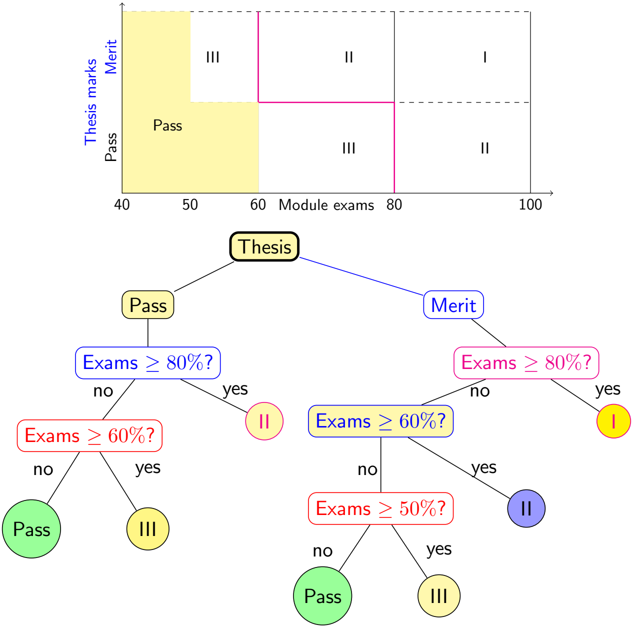
Example

This example is about a hypothetical classification system for a university degree.

Suppose a student has to take different modules and a major dissertation or thesis for a degree, and each module subject has an exam. The minimum pass marks are 40%. For each thesis, three grades (merit, pass, fail) are given. A naïve system is used for this hypothetical degree programme, as shown in the figure (next slide), to put equal weights on the exams and thesis.

If a student get all the modules with a mark above 80% with a merit of his or her thesis, a first class degree (I) will be award to this student. The final classes are first class (I), second class (II), third class (III) or pass.


Thesis marks
Merit
Pass
III
II
I
Pass
III
II
40
50
60
Module exams
80
100



Decision tree for the hypothetical degree classification.

ID3 Algorithm

In the decision tree method, different algorithms/implementations may choose attributes using different criteria, and thus decision trees are not unique.

The popular algorithm, ID3 (Iterative Dichotomiser 3), uses an information gain (IG) as a rule to select the attribute that can explain the training examples and maximize the IG.

For a two-class classification (1 or 0) problem with n training examples, if p_1 represents the fraction of class 1 among the whole set, and $p_0 = 1 - p_1$ represents the fraction of class 0 among the whole set, then the entropy S can be defined as

$$S = -p_1 \log_2 p_1 - p_0 \log_2 p_0,$$

where base 2 logarithm is used. Conventionally, we set $0 \log_2 0 = 0$.

In case of multiple K classes, this entropy can be extended to

$$S = -\sum_{i=1}^{K} p_i \log_2 p_i,$$

where p_i is the fraction or probability of samples belonging to Class i. Another related concept is the Gini index or Gini impurity

$$G_I = \sum_{i=1}^{K} p_i (1 - p_i).$$

Information Gain

The information gain $IG(A)$, of an attribute A with possible categorical values of x_j where $j = 1, 2, ..., K$ are the categories, is defined as

$$IG(A) = S - \sum_{x_j \in \Omega(A)} f_j S_{x_j},$$

where $\Omega(A)$ is the set of categorical values of attribute A, and f_j is the fraction of value $\boldsymbol{x}_j \in A$, which can be calculated by the ratio of the number $|A_{x_j}|$ of $x_j \in A$ to the total number of values or cardinality $|A|$. The S_{x_j} is the entropy, with a similar formula to entropy S, though its fractions are based on the categorical values of that attribute.

Algorithm 4: The ID3 Algorithm

```
Load the data and calculate the sample entropy $S$;
for (all attributes) do
    Find the attribute with the maximum information gain $IG(A)$;
    Split the set into subsets by values of that best attribute;
    Create a decision tree node for that attribute with $IG(A)_{\max}$;
    Iterate over the rest of unused/unsplit attributes;
```

The ID3 is simple and can work well, however, it is a greedy method and there is no guarantee of optimality. In addition, it cannot handle continuous data set.

Previous Example (Revisited)

In the hypothetical degree classification example, we used the decision tree method. However, the attribute we used for making the first decision seems an arbitrary choice.

Now the ID3 algorithm provides a criterion to select the main attribute. For a degree programme, there are 30 students. The numbers of students of different classes (I, II, III and pass) are 12, 9, 6, 3, respectively.

So we have $K = 4$ classes, and fractions of each class are

$$p_1 = \frac{12}{30} = 0.4, \quad p_2 = \frac{9}{30} = 0.3,$$

$$p_3 = \frac{6}{30} = 0.2, \quad p_4 \text{ (pass)} = \frac{3}{30} = 0.1.$$

So their entropy is

$$S = -\sum_{i=1}^{4} p_i \log_2 p_i$$

$$= -\Big[0.4 \log_2 4 + 0.3 \log_2 0.3 + 0.2 \log_2 0.2 + 0.1 \log_2 0.1\Big] = 1.8464.$$

Among all the four classes, suppose there are 20 theses that are merit, and 10 theses that are just pass. For merits, their corresponding students' classes are 16, 2, 1, 1 for I, II, III, and pass, respectively, while for theses with a pass, they are 2, 2, 4, 2 for I, II, III, and pass, respectively. So the attribute 'thesis' has two values 'merit' and 'pass' (all 30 students got a degree). So the fractions for merit and pass are 20/30 and 10/30, respectively. That is, $f_1 = 2/3$ and $f_2 = 1/3$.
For the theses with a merit, their fractions or probabilities of different classes are $p = [16, 2, 1, 1]/20 = [0.8, 0.1, 0.05, 0.05]$, whose entropy is

$$S_{x_1} = -[0.8 \log_2 0.8 + 0.1 \log_2 0.1 + 0.05 \log_2 0.05 + 0.05 \log_2 0.05] = 1.0219.$$

For the theses with a pass, their fractions are $p = [2, 2, 4, 2]/10 = [0.2, 0.2, 0.4, 0.2]$ and their entropy is

$$S_{x_2} = -[0.2 \log_2 0.2 + 0.2 \log_2 0.2 + 0.4 \log_2 0.4 + 0.2 \log_2 0.2] = 1.9219.$$

Finally, the information gain is

$$IG(\text{thesis}) = S - \sum_{x_j \in [\text{merit, pass}]} f_j S_{x_j} = 1.8464 - \Big[\frac{2}{3} \times 1.0219 + \frac{1}{3} \times 1.9219\Big] = 0.5245.$$

Thus, the information gain by using thesis marks as a decision criterion is 0.5245 bit.

Exercise: `Use R or Python to implement this example.`

C4.5 Algorithm

A significant improvement over ID3 is the C4.5 method that can deal with continuous data and missing data. Pruning is used in C4.5 so as to reduce the depth of the trees and remove unnecessary or insignificant branches.

Split Information

For K different classes/categories, the split information (SI) of an attribute is the entropy

$$SI(A) = -\sum_{i=1}^{K} f_i \log_2 f_i,$$

where f_i is the fraction or probability of the data in category i among the whole data set. The information gain ratio (IGR) is the ratio of information gain to the split information. That is

$$\mathrm{IGR}(A) = \frac{IG(A)}{SI(A)},$$

which is essentially the information gain, normalized by the split information.

Pruning

The C4.5 algorithm uses the so-called reduced-error pruning to avoid overfitting. Pruning is to remove the subtree or child nodes of a decision node and turn that decision node into a leaf node by giving it with the most common class among the training data

The main steps of the well-known C4.5 algorithm are summarized as follows:

Algorithm 5: C4.5 Algorithm

```
Load data and initialize;
for (each attribute $A$) do
    Calculate its normalized information gain ratio IGR(A);
    Find the attribute with the maximum IGR;
    Create a decision node by splitting that attribute;
    Iterate over the rest of the unsplit attributes;
```

Continuous Attributes

The way of dealing with continuous attributes is relatively straightforward. For example, if an attribute A is continuous, it can simply be converted into a boolean attribute at a critical value x_* by setting true (or 1) if $A \geq x_*$ and false (or 0) if $A < x_*$. The critical value x_* can be any decision boundary that be appropriate for that attribute and decision-making.

Exercise: Explore ID3 and C4.5 algorithms and their implementations in R or Python (and other packages such as Matlab/Octave).

Though C4.5 algorithm and its variants work well, the tree for each test instance constructed is a single tree. For large datasets, a natural extension is to use multiple trees simultaneously. Multiple random trees lead to the random forest algorithm.

Random Forest

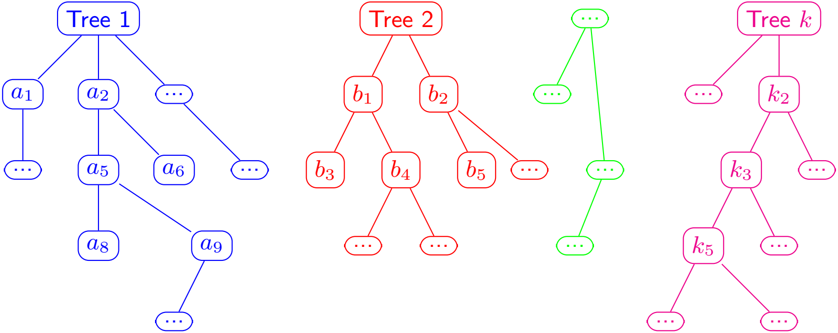
The random forest classifier constructs multiple decision trees randomly using sampling with replacement, and final decisions are based on a voting system of the tree ensemble.

Random Forest

- For a set of D features (or dimensionality of the training dataset), we often use m features for splitting the data at each decision split, though $m = \lceil \sqrt{D} \rceil$ or $m = \lceil \log_2 D \rceil$ (rounded to the nearest integer) is usually used to determine m.
- For each subset of m features, the training samples are randomly selected from the original data set with replacement. Then, each subset is fed into a decision tree classifier, and classification is based on the majority vote of the ensemble. Each tree can cast a single vote for the most popular class for a given input feature vector.

Algorithm 6: Random Forest Algorithm

```
Load data and initialize;
for (a given number of trees) do
    Select $m$ features randomly from $D$ features;
    Sample data to create $T$ different trees;
    Create each tree by iteratively splitting/forming decision nodes;
    Predict any test feature by running the multiple trees;
    Classify using majority voting;
```



Random forest and tree ensemble.

Notes

- Computational efforts tend to be high due to multiple trees (typically a few dozens to a few hundred trees).
- Random forests usually do not lead to overfitting, but overfitting can still occur for noisy data.
- There are many different variants of random forests, such as the kernel random forest and boosting methods (e.g., Adaboost).

Bayesian Classifiers

Probabilistic reasoning and inferencing can be considered approaches based on the assumptions that decision variables obey some probability distributions and data are drawn from certain probability distributions.

Naïve Bayesian Classifier

A Bayesian classifier is to estimate the probabilities of all alternative models or hypotheses, given data as evidence, so as to find the most probable classification to be assigned to each new input. Its core foundation is the Bayes' theorem

$$p(H|S) = \frac{p(S|H)p(H)}{p(S)},$$

which estimate the *posterior* probability $p(H|S)$ for a hypothesis or model, given the samples or training data set S. Here, $p(S|H)$ is the likelihood probability of the data samples, given H is true. $p(H)$ is the *prior* probability of hypothesis H, which somehow incorporates any background knowledge about H [If no prior knowledge, uniform distributions can be used as a prior].

Depending on the emphasis/formulations, we can seek a maximum a posterior (MAP) hypothesis

$$\text{Maximize}\, p(H|S) \propto \text{Maximize}\, P(S|H)P(H),$$

or a maximum likelihood (ML)

$$\text{Maximize}\, p(S|H),$$

which is equivalent to the previous MAP if both $p(H)$ and $p(S)$ are constant.

For a classification problem with a focus on an attribute $A = x$ with K different attribute values $[x_1, x_2, ..., x_K]$, its model function (the model hypothesis) $y = f(x)$ forms a set of finite discrete values $y_j \in \Omega$.

The aim of a Bayesian classifier is to estimate the probability of y, given data x_i, so as to assign the class probability $\max p(y_j|x_i) = p(y_j|x_1, x_2, ..., x_K), \quad y_j \in \Omega$, which is equivalent to

$$\max \frac{p(x_1, x_2, ..., x_K|y_j)p(y_j)}{p(x_1, x_2, ..., x_K)} \propto \max p(x_1, x_2, ..., x_K|y_j)p(y_j).$$

It is usually difficult (sometimes impossible) to calculate the probability $p(x_1, ..., x_K|y_j)$.

Naïve Bayesian

A naive assumption is that all the data sample values are conditionally independent from each other, so the joint probability becomes a product of individual probability. That is

$$p(x_1, x_2, ..., x_K|y_j) = \prod_{i=1}^{K} p(x_i|y_j).$$

The classification problem becomes an optimization problem

$$\max \quad p(y_j) \prod_{i=1}^{K} p(x_i|y_j).$$

A classifier using this to assign probabilities becomes a naïve Bayesian classifier.

Bayesian Network

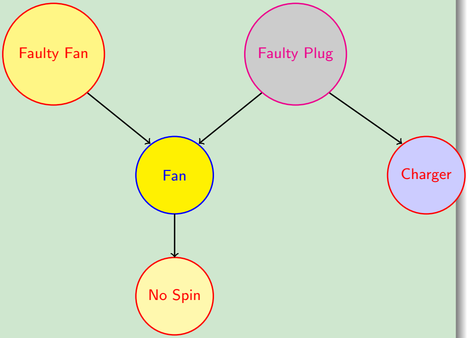
A Bayesian network (BN), also called Bayes net or belief network, is a probabilistic graphical model for representing knowledge about an uncertain domain where each node corresponds to a random variable and each edge represents the conditional probability for the corresponding random variables. Typically, a BN can be represented as a directed acyclic graph (DAG) without loop or self connection.

Example

In my office, there is an electric fan that I use only in summer. Imagine a scenario that I try to switch on the fan, it does not spin. The fan is plugged into an extension socket or plug, there is a possibility of a plug failure. How do we figure out what the possible causes are?

The fan has a probability of 0.02 for failure, while the plug is very old, which has failure probability 0.2. I also have a mobile phone charger connected to the same plug. I found the charger works well, now what is the probability of the problem is caused by a faulty fan?

We can represent the above scenario as a simple Bayesian network on the right.



A simple Bayesian network example.

Bayesian belief network

Bayesian belief networks (BBN) use a set of conditionally independent probabilities, but not imposing all variable values. In a BBN, nodes represent variables (continuous or discrete), and arcs represent causality relationships in terms of conditional probabilities.

For a given structure of a BBN, not every variable is observable. Unobservable variables are called hidden or latent variables.

BBN

The BBN model has both observable random variables X and hidden random variables Z, which means that likelihood function $p(S|H)$ is a function of X and Z, that is $L(X, Z)$. The maximization of likelihood $L(X, Z)$ is to maximize the expectation of the logarithmic likelihood (though it can be difficult to write explicitly). Thus, the method becomes the expectation-maximization (EM) method with an E step and an M step.

- In the E step, we define an expectation $Q_L = \mathbb{E}\Big[\log L(X, Z)\Big]$, which starts with an arbitrary values initially and repeatedly estimate their expectation.
- The aim in the M step is to use an optimizer to solve an optimization problem

$$\text{Maximize}\, Q_L = \mathbb{E}\Big[\log L(X, Z)\Big].$$

The optimizer can be a gradient-based search algorithm.

Big Data

As we mentioned before, big data has five Vs: volume, velocity, variety, value and veracity. Big data can pose significant extra challenges, compared to small data sets.

Challenging issues

- Data samples may not be statistically independent, thus the traditional data mining techniques based on independent sampling assumptions may not work well.
- The data sets can be extremely large, much larger than the physical memory of a computer. Thus, some kind of resampling or segment-by-segment techniques are often used to deal with such challenges.
- Data can be dynamical and noisy (e.g., streaming data via unstable networks). Data can also be incomplete with missing data. Thus, specialized techniques are needed to cope with non-stationarity and uncertainties.
- High dimensionality and high volumes may require high computational costs, which may be a major constraint in many applications in practice.

Recent developments show that new methods may be more suitable for large data sets. For example, the Bradley-Fayyad-Reina (BFR) algorithm and Clustering Using REpresentative (CURE) algorithm have shown good results.

BFR Algorithm

The BFR algorithm is an extension of k-means to high-dimensional data. It assumes that data are distributed around centroids according to Gaussian distributions in Euclidean spaces. The essence of BFR algorithm can be outlined here.

BFR

1. Choose a small subset (by resampling) of the big data using either k-means or hierarchy methods to find the initial k clusters.
2. Take each chunk data (a subset) from the big dataset, do the following:
    - Assign the data points and summarize the clusters (see details below), then discard the data;
    - Compress and merge points that are close to one another (forming compressed sets);
    - Retain the data points that are not assigned or not close to one another (forming retained sets).

For any given N points (in a subset), calculate the dimension-wise SUM$_i$ and SUMSQ$_i$ where SUM$_i$ is the vector sum of data in the i-th dimension, and SUMSQ$_i$ is the sum of the squares of all the data points in the i-th dimension. Then, the centroid can be updated at SUM$_i$/N in the i-th dimension, and its variance in the i-th dimension can be estimated as SUMSQ$_i$-(SUM$_i$/N)^2.

BFR

Assigning a new point

When deciding if a new point $\boldsymbol{x}_i$ is close enough to one of the k clusters, we can use the following rules: (1) add point $\boldsymbol{x}_i$ to a cluster if it has the centroid closest to $\boldsymbol{x}_i$, or (2) assign $\boldsymbol{x}_i$ to a cluster with the least Mahalanobis distance.

The Mahalanobis distance between $\boldsymbol{x} = (x_1, \ x_2, \ ..., x_n)$ to a cluster centroid $c = (c_1, \ c_2, \ ..., c_n)$ as

$$d_m = \sqrt{\sum_{i=1}^{n} \Big(\frac{x_i - c_i}{\sigma_i} \Big)^2},$$

where σ_i is the standard deviation of the cluster in the i-th dimension. Therefore, this distance is a variance-based, scaled distance.

Update

- For points that are assigned to a cluster (including any mini-cluster), update the centroid and other metrics (then discard such points).
- Finally, (after going through all the subset of the data or loading the last chunk of the data), post-process the retained sets and compressed sets by either assigning each point to the cluster of the nearest centroid, or discarding them as outliers.

CURE Algorithm

Though the BFR algorithm is efficient, it is mainly for data that are symmetric around clusters, thus it cannot deal with S-shapes or rings effectively. For such complicated datasets, we can use the CURE (Clustering Using REpresentives) algorithm.

CURE Algorithm

1. Choose a small sample (a small subset) of the data so as to be clustered in the main memory using any good methods such as k-means and hierarchy methods, which should give the initial clusters.
2. Select a small set of points from each cluster to be representative points (points should be as far from one another as possible).
3. Move each of the representative points a fixed fraction (typically 20%) of the distance between its location to the centroid of its cluster.
4. Merge two clusters if they have a pair of representation points (one from each cluster) that are sufficiently close.
5. Repeat the above steps and merge until no more close cluster to merge.
6. Carry out point assignment for all the rest of points in the big data.

This algorithm can be sufficiently efficient in practice, though no guarantee for the global optimality. For large datasets, it is impractical to reach the true global optimality.

Thank you :)

References

- Xin-She Yang, **Introduction to Algorithms for Data Mining and Machine Learning**, Academic Press/Elsevier, (2019).
- Xin-She Yang, **Optimization Techniques and Applications with Examples**, John Wiley & Sons, (2018).

Notes on Software

Almost all major software packages in statistics, data mining and machine learning have implemented the algorithms we introduced here in this chapter.

- R has `kmeans`, `kNN`, decision trees and random forest packages. In addition, R can be paired with big databases such as Amazon Redshift and Google BigQuery.
- Python has `kmeans` as well as decision-tree classifiers. There are also Python libraries such as Pandas, NumPy, SciPy, and machine learning libraries such as Mlpy, Theano, Scikit-learn and others.
- Matlab have all the functionalities such as `kmeans`, `fitcknn`, `fitctree` and `fitrensemble`. More computer codes in Matlab can be found from Mathworks file exchanges, though not all files are well tested on the file exchange web site.
- RapidMiner has all these classifiers implemented, including logistic regression, decision trees, naive Bayesian classifier. It is worth mentioning that these algorithms including random forests have been implemented in other programming languages such as C++.