K

k-Anonymity

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona,
Catalonia

Synonyms

p-Sensitive *k*-Anonymity

Definition

A protected dataset is said to satisfy *k*-anonymity for $k > 1$ if, for each combination of key attribute values (e.g., address, age, gender, etc.), at least *k* records exist in the dataset sharing that combination [2,3].

Key Points

If, for a given *k*, *k*-anonymity is assumed to be sufficient protection, one can concentrate on minimizing information loss with the only constraint that *k*-anonymity should be satisfied. This is a clean way of solving the tension between data protection and data utility. Since *k*-anonymity is usually achieved via generalization (equivalent to global recoding, as said above) and local suppression, minimizing information loss usually translates to reducing the number and/or the magnitude of suppressions.

k-Anonymity bears some resemblance to the underlying principle of microaggregation and is a useful concept because quasi-identifiers are usually categorical or can be categorized, i.e., they take values in a finite (and ideally reduced) range. However, re-identification is not necessarily based on categorical key attributes: sometimes, numerical outcome attributes (which are continuous and often cannot be categorized) give enough clues for re-identification. Microaggregation was suggested as a possible way to achieve *k*-anonymity for numerical, ordinal and nominal attributes [1].

p-Sensitive *k*-anonymity is a stronger property whereby it is required that a dataset is *k*-anonymous and additionally that there are at least *p* distinct values for each confidential attribute within a group of records sharing a combination of key attributes [4].

Cross-references

▶ Global Recoding
▶ Inference Control in Statistical Databases
▶ Local Suppression
▶ Microaggregation
▶ Microdata

Recommended Reading

1. Domingo-Ferrer J. and Torra V. Ordinal, continuous and heterogenerous *k*-anonymity through microaggregation. Data Mining Knowl. Discov., 11(2):195–212, 2005.
2. Samarati P. Protecting respondents' identities in microdata release. IEEE Trans. Knowl. Data Eng., 13(6):1010–1027, 2001.
3. Samarati P. and Sweeney L. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International, 1998.
4. Truta T.M. and Vinay B. Privacy protection: *p*-sensitive *k*-anonymity property. In Proc. 2nd Int. Workshop on Privacy Data Management, 2006, p. 94.

k-Closest Pair Join

▶ Closest-Pair Query

k-Closest Pair Query

▶ Closest-Pair Query

KDD Pipeline

Hans-Peter Kriegel, Matthias Schubert
Ludwig-Maximilians-University, Munich, Germany

Synonyms

KDD process; Data mining process; Data mining pipeline

Definition

The KDD pipeline describes the complete process of knowledge discovery in databases (KDD), i.e., the process of deriving useful, valid and non-trivial patterns from a large amount of data. The pipeline consists of five consecutive steps:

Selection

The selection step identifies the goal of the current application and selects a data set that is likely to contain relevant patterns.

Preprocessing

The preprocessing step increases the quality of the data set by supplementing missing attributes, removing duplicate instances and resolving data inconsistencies.

Transformation

The transformation step deletes correlated and irrelevant attributes and derives new more meaningful attributes from the current data description.

Data Mining

This step selects a data mining algorithm with respect to the goal which was identified in the selection step and derives patterns or learns functions that are valid for the current data set.

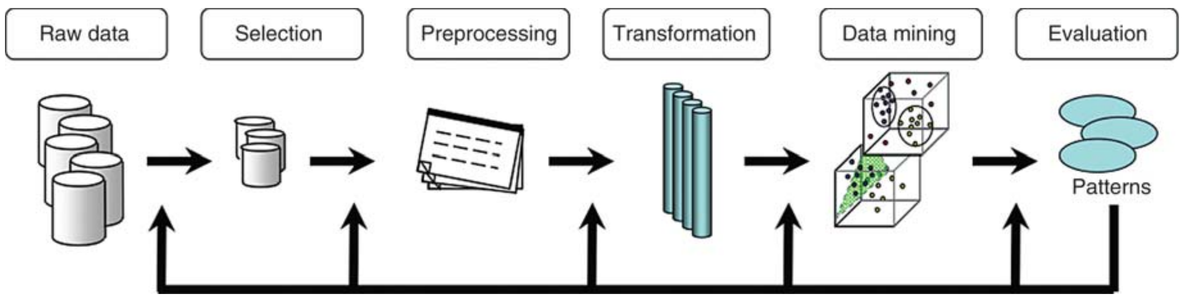
Evaluation and Interpretation

In the last step, the found patterns are checked with respect to their validity. Furthermore, the user examines the usefulness of the found knowledge for the given application.

The quality of the found patterns depends on the methods being employed in each of these steps. Thus, the pipeline is usually repeated after adjusting the parametrization or even exchanging the methods in any of these steps until the quality of the results is sufficient. Figure 1 illustrates the complete KDD pipeline.

Key Points

The KDD pipeline was originally introduced as KDD process [1–3]. It describes a unifying framework containing all necessary and optional steps when deriving patterns using data mining algorithms. An important aspect of this view is that data mining is just one step in the complete KDD process, which emphasizes the importance of a meaningful and consistent data representation. The KDD pipeline is considered to be an interactive and adjustable framework rather than a strict work flow. The necessity for this flexibility arises from the large variety of methods and parameter selections that can be applied in each step. In the majority of cases, it is necessary to adjust parameters or even exchange the applied method in one of the steps if the final patterns do not display satisfactory quality in the evaluation step. Furthermore, the borders of each step cannot be outlined in a strict manner because the quality of results strongly depends on a well selected combination of the methods applied in each step. Additionally, there exist methods fulfilling the tasks of two consecutive steps, e.g., transformation and data mining.



KDD Pipeline. Figure 1. Schema of the KDD pipeline.

Cross-references

▶ Data Mining
▶ Knowledge Discovery
▶ OLAP

Recommended Reading

1. Brachman R. and Anand T. The process of knowledge discovery in databases: a human centered approach. In Proc. 10th National Conf. on AI, 1996, pp. 37–38.
2. Fayyad U., Piatetsky-Shapiro G., and Smyth P. From data mining to knowledge discovery in databases. In Proc. 10th National Conf. on AI, 1996, pp. 1–30.
3. Fayyad U., Piatetsky-Shapiro G., and Smyth P. Knowledge discovery and data mining: towards a unifying framework. In Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, 1996, pp. 82–88.

KDD Process

▶ KDD Pipeline

k-Distance Join

▶ Closest-Pair Query

Key

David W. Embley
Brigham Young University, Provo, UT, USA

Synonyms

Uniqueness constraint

Definition

In the relational model, a *key* for a relational schema is a set of attributes whose value(s) uniquely identify a tuple in a valid instance of the relation. Said another way, key value(s) appear at most once in a relation. Often the set of attributes constituting a key is a set with a single attribute. For example, in the relation

Customer

(CustomerID	Name	Address)
11111	Pat	12 Maple
22222	Tracy	44 Elm

the singleton set consisting of just *CustomerID* is a key and so is the set consisting of the pair of attributes (*Name*, *Address*). The values of *CustomerID* (here, 11111 and 22222) each uniquely identify a tuple. In any valid instance for this relation, no *CustomerID*-value may appear twice. Similarly, the pair (*Name*, *Address*) is a key. Pairs of (*Name*, *Address*) values (e.g., <*Pat*, *12 Maple* >) uniquely identify a tuple, and no value pair may appear twice.

Designating a set of attributes as a key is a property of a relational schema, not a property of a relation instance. It may just happen that for some valid relation instances a value uniquely identifies a tuple, but a key must always uniquely identify a tuple for *all* valid relations. In this example, *Pat* and *Tracy* uniquely identify tuples, but it would also be valid for another *Tracy*, say, one living at *905 Lincoln Ave.*, to be a customer. Although *Name* uniquely identifies tuples in this *Customer* table instance, *Name* does not uniquely identify tuples in *all* valid *Customer* table instances and thus is not a key for the *Customer* relational schema.

Key Points

The term *key* is often ambiguous and may mean *minimal key*, *superkey*, *primary key*, or *candidate key*. A *minimal key* is a key that does not have superfluous attributes. Clearly, in the sample relational schema above, *CustomerID* together with *Name* uniquely identify a customer because *CustomerID* alone is enough. *Name* is superfluous, and thus the set consisting of the pair of attributes (*CustomerID*, *Name*) is not a minimal key. The set consisting of the pair of attributes (*Name*, *Address*), however, is a minimal key. Both are necessary – in the table above, for example, both <*Tracy*, *905 Lincoln Ave.* > and <*Lynn*, *12 Maple* > could appear so that neither *Name* alone nor *Address* alone would uniquely identify a tuple. Usually only minimal keys are of interest. Keys that may not be minimal are called *superkeys*. A *superkey* is any set of attributes, minimal or not, that uniquely identifies a tuple in any valid relation instance. A relational schema may have several minimal keys. The sample *Customer* relational schema above, for example, has two minimal keys – *CustomerID* and the pair (*Name*, *Address*). When there are several minimal keys, one is chosen as a *primary key*. When there is only one, it is also called a *primary key*. Because there can be a choice among minimal keys, they are often each referred to as *candidate keys*.

Keys and functional dependencies are strongly related. Indeed, keys can be defined in terms of functional dependencies. Let U be a set of attributes, and let F be a set of functional dependencies over U. Let R, a subset of U, be a relational schema. A subset K of R is a *superkey* of R if $K \rightarrow R \in F^+$, where F^+ includes F and any functional dependency implied by F. A subset K of R is a *candidate key* of R (also called a *minimal key* of R) if K is a superkey and there does not exist a proper subset K' of K such that $K' \rightarrow R \in F^+$.

Database normalization is largely about aligning keys and functional dependencies. A relational schema R is in Boyce-Codd Normal Form (BCNF) if for every non-trivial functional dependency $X \rightarrow Y$ (given or implied) that applies to R (satisfies $XY \subseteq R$), X is a superkey of R. More informally, the idea of aligning keys and functional dependencies to yield BCNF is intuitively captured by making every attribute in the relational schema depend functionally on a minimal key, the whole minimal key, and nothing but the whole minimal key.

Cross-references

- ▶ Functional Dependency
- ▶ Normal Forms and Normalization
- ▶ Relational Model

Key Range Locking

▶ B-Tree Locking

Key Value Locking

▶ B-Tree Locking

KL-ONE Style Languages

▶ Description Logics

K-Means and K-Medoids

Xue Li
The University of Queensland, Brisbane, QLD, Australia

Synonyms

K-means partition; PAM (Partitioning Around Medoids); CLARA (Clustering LARge Applications); CLARANS (Clustering large applications based upon randomized search)

Definitions

K-means

Given an integer k and a set of objects $S = \{p_1, p_2,...,p_n\}$ in Euclidian space, the problem of k-means clustering is to find a set of centre points (means) P = $\{c_1, c_2,..., c_k\}$, $|P| = k$ in the space, such that S can be partitioned into k corresponding clusters $C_1, C_2,...,C_k$, by assigning each object in S to the closest centre c_i. The sum of square error criterion (SEC) measure, defined as $\sum_{i=1}^{k} \sum_{p \in C_i} |p - c_i|^2$, is minimized.

K-medoids

Given an integer k and a set of objects $S = \{p_1, p_2,..., p_n\}$ in Euclidian space, the problem of k-medoids clustering is to find a set of objects as medoids P = $\{o_1, o_2,...,o_k\}$, $|P| = k$ in the space, such that S can be partitioned into k corresponding clusters $C_1, C_2,..., C_k$, by assigning each object in S to the closest medoid o_i. The sum of square error criterion (SEC) measure, defined as $\sum_{i=1}^{k} \sum_{p \in C_j} |p - o_j|^2$, is minimized.

Key Points

K-means

The k-means algorithm starts with a random selection of k objects as the centres or the means of k clusters. The challenge is to place these centres in a clever way because each different location of centres may result in different clustering. Hence, a reasonable initial choice is to locate them as far away from each other as possible. Then an iteration process is used to assign remaining objects to their nearest centres. After the membership of all clusters is considered according to the given initial centres, the cluster centres will all be reconsidered. Hence the new cluster centres are calculated a set of k means $P = \{c_1, c_2,\ldots,c_k\}$. For example, for the mean of a set of single-valued objects in cluster i with m points is defined as m_i:

$$m_i = \frac{1}{m}\sum_{j=1}^{m} p_{i,\,j}$$

The means of high-dimensional objects are calculated in a same by using this formula for each of the dimensions. When cluster centres are updated, the membership computations will be executed again to reallocate the objects to the nearest centers. The main idea of updating the cluster centres and the memberships is based on the computation of Euclidian distances such that the objects in a cluster have the minimum distances and the objects between clusters have the maximum distances. The process will continue until there are no more updates that could reallocate the centres and the memberships of clusters. Even though it is easy to prove that the procedure always terminates, the k-means algorithm does not necessarily find the global optimal solutions, corresponding to the global minimum of the objective function. Also, it is easy to demonstrate that this algorithm is significantly sensitive to the initial randomly selected centres. To mitigate this situation the k-means algorithm can be executed multiple times in order to identify a better global solution. The complexity of the algorithm is *O(nkt)*, for *n* objects, *k* clusters and *t* iterations.

The k-means algorithm is only applicable to the objects with mean defined. The user must specify an initial value k. The k-means approach is only suitable for finding convex shapes with all clusters having similar sizes. Moreover, when there is noise and outliers in the dataset, the means of clusters may appear at locations where the majority objects are far away.

K-medoids

K-medoids algorithm avoids calculating means of clusters in which extremely large values may affect the membership computations substantially. K-medoids can handle outliers well by selecting the most centrally located object in a cluster as a reference point, namely, medoid. The difference between k-means and k-medoids is analogous to the difference between mean and median: where mean indicates the average value of all data items collected, while median indicates the value around that which all data items are evenly distributed around it. The basic idea of k-medoids is that it first arbitrarily finds *k* objects amongst *n* objects in the dataset as the initial medoids. Then the remaining objects are partitioned into *k* clusters by computing the minimum Euclidian distances that can be maintained for the members in each of the clusters. An iterative process then starts to consider objects p_i, $i = 1,\ldots,n$, if a medoid o_j, $j = 1,\ldots,k$, can be replaced by a candidate object o_c, $c = 1,\ldots,n$, $c \neq i$. There are four situations to be considered in this process: (i) *Shift-out membership*: an object p_i may need to be shifted from currently considered cluster of o_j to another cluster; (ii) *Update the current medoid*: a new medoid o_c is found to replace the current medoid o_j; (iii) *No change*: objects in the current cluster result have the same or even smaller SEC for all the possible redistributions considered; (iv) *Shift-in membership*: an outside object p_i is assigned to the current cluster with the new (replaced) medoid o_c.

K-medoids algorithm needs to test if any existing medoids can be replaced by any other objects. By looking at all of these possible replacements, if the overall SEC is improved then the given medoid will be replaced. The computational cost of k-medoid is much higher than the k-means. For each iteration, computational cost is $k(n\text{-}k)^2$ for $k(n\text{-}k)$ pairs of objects to be considered with each $(n\text{-}k)$ evaluations of the SEC. K-meoids algorithm then is considered with a sampling method for the improvement on the computational complexity. In this way, several samples are taken from the dataset, and then the k-medoids algorithm is applied to each of the samples. The convergence of medoids is achieved by choosing the sample that performs the best.

Cross-references

► Density Based Clustering
► Hierarchical Clustering

Recommended Reading

1. Kaufman L. and Rousseeuw P.J. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley, NY, 1990.
2. MacQueen J. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symposium on Mathematics, Statistics and Probabilities, vol. 1, 1967, pp. 281–297.
3. Ng R.T. and Han J. Efficient and effective clustering methods for spatial data mining. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 144–155.

K-Means Partition

▶ K-Means and K-Medoids

k-Nearest Neighbor Classification

▶ Nearest Neighbor Classification

k-NN Classification

▶ Nearest Neighbor Classification

kNN Query

▶ Nearest Neighbor Query

Knowledge Creation

▶ Ontology Elicitation

Knowledge Discovery from Biological Resources

▶ Text Mining of Biological Resources

Knowledge Discovery from Data

▶ Data Mining

Knowledge Discovery in Streams

▶ Classification on Streams

Knowledge Discovery in Text (KDT)

▶ Text Mining

Knowledge Management

▶ Ontologies and Life Science Data Management

Knowledge Organization Systems

▶ Gazetteers

Knowledge-based Systems

▶ Executable Knowledge

Koch Snowflake

▶ Fractal