D

Data Mining On Text

►Text Mining

Data Preparation

Geoffrey I. Webb
Monash University, Victoria, Australia

Synonyms

Data preprocessing; Feature construction

Definition

Before data can be analyzed, they must be organized into an appropriate form. Data preparation is the process of manipulating and organizing data prior to analysis.

Motivation and Background

Data are collected for many purposes, not necessarily with machine learning in mind. Consequently, there is often a need to identify and extract relevant data for the given analytic purpose. Every learning system has specific requirements about how data must be presented for analysis and hence, data must be transformed to fulfill those requirements. Further, the selection of the specific data to be analyzed can greatly affect the models that are learned. For these reasons, data preparation is a critical part of any machine learning exercise. Data preparation is often the most time-consuming part of any nontrivial machine learning project.

Processes and Techniques

The manner in which data are prepared varies greatly depending upon the analytic objectives for which they are required and the specific learning techniques and software by which they are to be analyzed. The following are a number of key processes and techniques.

Sourcing, Selecting, and Auditing Appropriate Data

It is necessary to review the data that are already available, assess their suitability to the task at hand, and investigate the feasibility of sourcing new data collected specifically for the desired task.

Much of the theory on which learning systems are based assumes that the training data are a random sample of the population about which the user wishes to learn a model. However, much historical data represent biased samples, for example, data that have been easy to collect or that have been considered interesting for some other purpose. It is desirable to consider whether the available data are sufficiently representative of the future data to which a learned model is to be applied.

It is important to assess whether there is sufficient data to realistically obtain the desired machine learning outcomes.

Data quality should be investigated. Much data is of low quality. Those responsible for manual data collection may have little commitment to assuring data accuracy and may take shortcuts in data entry. For example, when default values are provided by a system, these tend to be substantially overrepresented in the collected data. Automated data collection processes might be faulty, resulting in inaccurate or incorrect data. The precision of a measuring instrument may be lower than desirable. Data may be out of date and no longer correct.

Where the data contain ►noise, it may be desirable to identify and remove outliers and other suspect data points or take other remedial action.

Existing data may be augmented through data enrichment. This commonly involves sourcing of additional information about the data points on which data are already held. For example, customer data might be enriched by purchasing socioeconomic data about individual customers.

Transforming Representation

It may be necessary to frequently transform data from one representation to another. Reasons for doing so include highlighting relevant distinctions and formatting data to satisfy the requirements of a specific learner.

►Discretization is a process whereby quantitative data are transformed into qualitative.

Some systems cannot process multi-valued categorical variables. This limitation can be circumvented by converting a multi-valued categorical variable into multiple binary variables, one new variable to represent the presence or absence of each value of the original variable. Conversely, multiple mutually exclusive binary variables might be converted into a single multi-valued categorical variable.

Some systems require the input to be numeric. Categorical variables must be converted into numeric form. Multi-valued categorical variables should usually be converted into multiple binary variables before conversion to numbers, as projecting unordered values onto a linear scale can greatly distort analytic outcomes.

It is important to select appropriate levels of granularity for analysis. For example, when distinguishing products, should a gallon of low fat milk be described as a diary product, and hence not distinguished from any other type of dairy product, be described as low fat milk, and hence not distinguished from other brands and quantities, or uniquely distinguished from all other products. Analysis at the lowest level of granularity makes possible identification of potentially valuable fine-detail regularities in the data, but may make it more difficult to identify high-level relationships.

It is often desirable to create derived values. For example, the available data might contain fields for purchase price, costs, and sale price. The relevant quantity for analysis might be profit, which must be computed from the raw data. The creation of new features is called *feature construction*.

As many learning systems have difficulty with high dimension data, it may be desirable to project the data onto a lower dimensional space. Popular approaches to doing so include ►Principal Components Analysis and ►Kernel Methods.

Another approach to reducing dimensionality is to select only a subset of the available features (see ►Feature Selection).

It is important to determine whether the data have ►Missing Values and, if so, to ensure that appropriate measures are taken to allow the learning system to handle this situation.

►Propositionalization. Some data sets contain information expressed in a relational form, i.e., describing relationships between objects in the world. While some learning systems can accept relations directly, most operate only on attribute-value representations. Therefore, a relational representation must be reexpressed in attribute-value form. In other words, a representation equivalent to first-order logic must be converted to a representation equivalent only to propositional logic.

Cross References

►Data Set
►Discretization
►Entity Resolution
►Evolutionary Feature Selection and Construction
►Feature Construction in Text Mining
►Feature Selection
►Feature Selection in Text Mining
►Kernel Methods
►Measurement Scales
►Missing Values
►Noise
►Principal Component Analysis
►Propositionalization

Recommended Reading

Pyle, D. (1999). *Data preparation for data mining*. San Francisco, Morgan Kaufmann.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco, Morgan Kaufmann.

Data Preprocessing

►Data Preparation

Data Set

A data set is a collection of data used for some specific machine learning purpose. A ▶training set is a data set that is used as input to a ▶learning system, which analyzes it to learn a ▶model. A ▶test set or ▶evaluation set is a data set containing data that are used to ▶evaluate the model learned by a learning system. A training set may be divided further into a ▶growing set and a ▶pruning set. Where the training set and the test set contain disjoint sets of data, the test set is known as a ▶holdout set.

DBN

Dynamic Bayesian Network. See ▶Learning Graphical Models

Decision Epoch

In a ▶Markov decision process, *decision epochs* are sequences of times at which the decision-maker is required to make a decision. In a discrete time Markov decision process, decision epochs occur at regular, fixed intervals, whereas in a continuous time Markov decision process (or semi-Markov decision process), they may occur at randomly distributed intervals.

Decision List

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Synonyms

Ordered rule set

Definition

A decision list (also called an ordered rule set) is a collection of individual Classification Rules that collectively form a Classifier. In contrast to an unordered Rule Set, decision lists have an inherent order, which makes classification quite straightforward. For classifying a new instance, the rules are tried in order, and the class of the first rule that covers the instance is predicted. If no induced rule fires, a *default rule* is invoked, which typically predicts the majority class.

Typically, decision lists are learned with a ▶Covering Algorithm, which learns one rule at a time, appends it to the list, and removes all covered examples before learning the next one. Decision lists are popular in ▶Inductive Logic Programming, because PROLOG programs may be considered to be simple decision lists, where all rules predict the same concept.

A formal definition of decision lists, a comparison of their expressiveness to decision trees and rule sets in disjunctive and conjunctive normal form, as well as theoretical results on the learnability of decision lists can be found in Rivest (1987).

Cross References

▶Classification Rule
▶Disjunctive Normal Form
▶Rule Learning

Recommended Reading

Rivest, R.L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Decision Lists and Decision Trees

Johannes Fürnkranz
Fachbereich Informatik, Darmstadt, Germany

Definition

▶Decision Trees and ▶Decision Lists are two popular ▶Hypothesis Languages, which share quite a few similarities. The key difference is that decision trees may be viewed as unordered Rule Sets, where each leaf of the tree corresponds to a single rule with a condition part consisting of the conjunction of all edge labels on the path from the root to this leaf. The hierarchical structure of the tree ensures that the rules in the set are nonoverlapping, that is, each example can only be covered by a single rule. This additional constraint makes classification easier (no conflicts from multiple rules), but may result in more complex rules. For example, it has been shown that decision lists (ordered rule sets) with at most k conditions per rule are strictly *more expressive* than decision trees of depth k (Rivest, 1987). A similar result has been proved in Boström (1995).

Moreover, the restriction of decision tree learning algorithms to nonoverlapping rules imposes strong constraints on learnable rules. One problem resulting from this constraint is the *replicated subtree problem* (Pagallo and Haussler 1990); it often happens that identical subtrees have to be learned at various places in a decision tree, because of the fragmentation of the example space imposed by the restriction to nonoverlapping rules. Rule learners do not make such a restriction and are thus less susceptible to this problem. An extreme example for this problem has been provided by Cendrowska (1987), who showed that the minimal decision tree for the concept x defined as

```
IF A = 3 AND B = 3 THEN Class = x
IF C = 3 AND D = 3 THEN Class = x
```

has 10 interior nodes and 21 leafs assuming that each attribute `A` ... `D` can be instantiated with three different values.

On the other hand, a key advantage of decision tree learning is that not only a single rule is optimized, but that conditions are selected in a way that simultaneously optimizes the example distribution in all successors of a node. Attempts to adopt this property for rule learning have given rise to several hybrid systems, the best known being PART (Frank & Witten, 1998), which learns a decision list of rules, each one being the single best rule of a separate decision tree. This rule can be efficiently found without learning the full tree, by repeated expansion of its most promising branch. Similarly, pruning algorithms can be used to convert decision trees into sets of nonoverlapping rules (Quinlan, 1987).

Cross References

►Covering Algorithm
►Decision Trees
►Divide-and-Conquer Learning
►Rule Learning

Recommended Reading

Boström, H. (1995). Covering vs. divide-and-conquer for top-down induction of logic programs. In *Proceedings of the 14th international joint conference on artificial intelligence (IJCAI-95)*, (pp. 1194–1200). Montreal, Canada.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies, 27*, 349–370.

Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. In J. Shavlik (Ed.), *Proceedings of the 15th international conference on machine learning (ICML-98), Madison, Wisconsin* (pp. 144–151). San Francisco, CA. Morgan Kaufmann.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning, 5*, 71–99.

Quinlan, J. R. (1987). Generating production rules from decision trees. In *Proceedings of the tenth international joint conference on artificial intelligence (IJCAI-87)*, (pp. 304–307). Morgan Kaufmann, Milan, Italy.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.

Decision Rule

A decision rule is an element (piece) of knowledge, usually in the form of a "if-then statement":

if < Condition > then < Action >

If its Condition is satisfied (i.e., matches a fact in the corresponding database of a given problem) then its Action (e.g., classification or decision making) is performed. See also ►Markovian Decision Rule.

Decision Stump

Definition

A *decision stump* is a ►Decision Tree, which uses only a single attribute for splitting. For discrete attributes, this typically means that the tree consists only of a single interior node (i.e., the root has only leaves as successor nodes). If the attribute is numerical, the tree may be more complex.

Decision stumps perform surprisingly well on some commonly used benchmark datasets from the ►UCI repository (Holte, 1993), which illustrates that learners with a high ►Bias and low ►Variance may perform well because they are less prone to ►Overfitting. Decision stumps are also often used as weak learners in ►Ensemble Methods such as boosting (Freund & Schapire, 1996).

Cross References

►Bias and Variance
►Decision Tree
►Overfitting

Recommended Reading

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Proceedings of the 13th international conference on machine learning; Bari, Italy* (pp. 148–156). San Francisco: Morgan Kaufmann.
Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11*, 63–91.

Decision Threshold

The decision threshold of a binary classifier that outputs scores, such as ►decision trees or ►naive Bayes, is the value above which scores are interpreted as positive classifications. Decision thresholds can be either fixed if the classifier outputs calibrated scores on a known scale (e.g., 0.5 for a probabilistic classifier), or learned from data if the scores are uncalibrated. See ►ROC Analysis.

Decision Tree

Johannes Fürnkranz
Fachbereich Informatik
Darmstadt
Germany

Synonyms

C4.5; CART; Classification tree

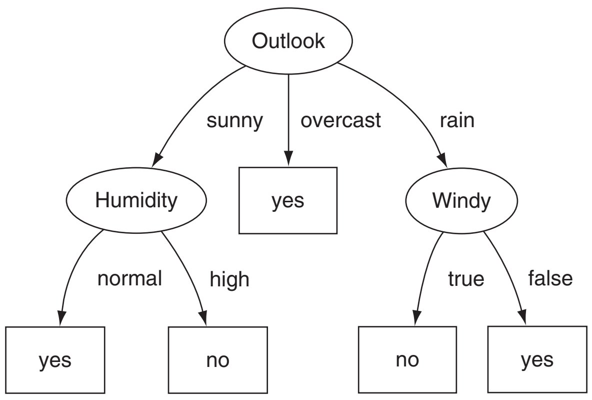
Definition

A *decision tree* is a tree-structured ►classification ►model, which is easy to understand, even by nonexpert users, and can be efficiently induced from data. The induction of decision trees is one of the oldest and most popular techniques for learning discriminatory models, which has been developed independently in the statistical (Breiman, Friedman, Olshen, & Stone, 1984; Kass, 1980) and machine learning (Hunt, Marin, & Stone, 1966; Quinlan, 1983, 1986) communities. An extensive survey of decision tree learning can be found in Murthy (1998).

Representation

Figure 1 shows a sample decision tree for a well-known sample dataset, in which examples are descriptions of weather conditions (*Outlook, Humidity, Windy, Temperature*), and the target concept is whether these conditions are suitable for playing golf or not (Quinlan, 1986). Classification of a new example starts at the top node—the *root*—and the value of the attribute that corresponds to this node is considered (*Outlook* in the example). The example is then moved down the branch that corresponds to a particular value of this attribute, arriving at a new node with a new attribute. This process is repeated until one arrives at a terminal node—a so-called *leaf*—which is not labeled with an attribute but with a value of the target attribute (*PlayGolf?*). For all examples that arrive at the same leaf, the same target value will be predicted. Figure 1 shows leaves as rectangular boxes.

Note that some of the attributes may not occur at all in the tree. For example, the tree in Fig. 1 does not contain a test on *Temperature* because the training data can be classified without making a reference to this variable. More generally, one can say that the attributes in



Decision Tree. Figure 1. A decision tree describing the Golf dataset (Quinlan, 1986)

the upper parts of the tree (near the root) tend to have a stronger influence on the value of the target variable than the nodes in the lower parts of the tree (e.g., *Outlook* will always be tested, whereas *Humidity* and *Windy* will only be tested under certain conditions).

Learning Algorithm

Decision trees are learned in a top-down fashion, with an algorithm known as *Top-Down Induction of Decision Trees (TDIDT), recursive partitioning*, or *divide-and-conquer* learning. The algorithm selects the best attribute for the root of the tree, splits the set of examples into disjoint sets, and adds corresponding nodes and branches to the tree. The simplest splitting criterion is for discrete attributes, where each test has the form $t \leftarrow (A = v)$ where v is one possible value of the chosen attribute A. The corresponding set S_t contains all training examples for which the attribute A has the value v. This can be easily adapted to numerical attributes, where one typically uses binary splits of the form $t \leftarrow (A < v_t)$, which indicate whether the attribute's value is above or below a certain threshold value v_t. Alternatively, one can transform the data before-hand using a ►Discretization algorithm.

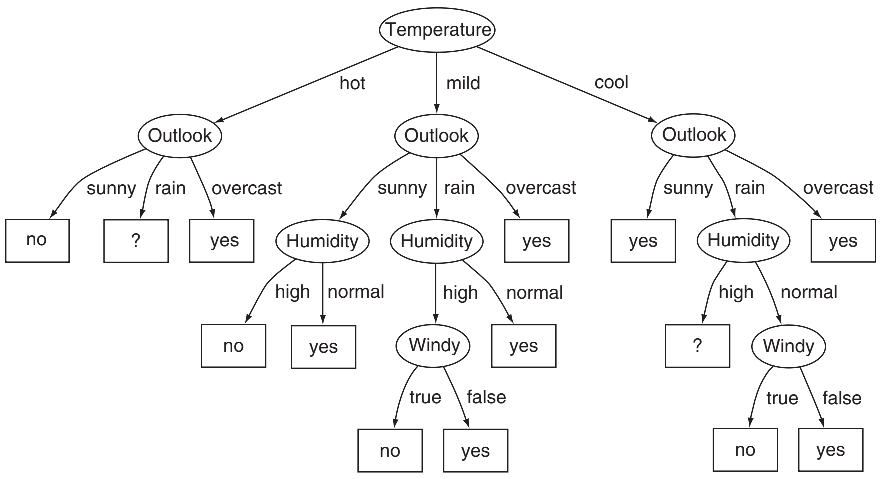
```
function TDIDT($S$)

Input: $S$, a set of labeled examples.

$Tree$ = new empty node
if all examples have the same class $c$
   or no further splitting is possible
then    // new leaf
        LABEL($Tree$) = $c$
else    // new decision node
        $(A, T)$ = FINDBESTSPLIT($S$)
        for each test $t \in T$ do
            $S_t$ = all examples that satisfy $t$
            $Node_t$ = TDIDT($S_t$)
            ADDEDGE($Tree \xrightarrow{t} Node_t$)
        endfor
endif
return $Tree$
```

After splitting the dataset according to the selected attribute, the procedure is recursively applied to each of the resulting datasets. If a set contains only examples from the same class, or if no further splitting is possible (e.g., because all possible splits have already been exhausted or all remaining splits will have the same outcome for all examples), the corresponding node is turned into a leaf node and labeled with the respective class. For all other sets, an interior node is added and associated with the best splitting attribute for the corresponding set as described above. Hence, the dataset is successively partitioned into nonoverlapping, smaller datasets until each set only contains examples of the same class (a so-called *pure* node). Eventually, a pure node can always be found via successive partitions unless the training data contains two identical but contradictory examples, that is, examples with the same feature values but different class values.

Attribute Selection

The crucial step in decision tree induction is the choice of an adequate attribute. In the sample tree of Fig. 2, which has been generated from the same 14 training examples as the tree of Fig. 1, most leaves contain only a single training example, that is, with the selected splitting criteria, the termination criterion (all examples of a node have to be of the same class) could, in many cases, only trivially be satisfied (only one example remained in the node). Although both trees classify the training data correctly, the former appears to be more trustworthy, and in practice, one can often observe that simpler trees are more accurate than more complex trees. A possible explanation could be that labels that are based on a higher number of training examples tend to be more reliable. However, this preference for simple models is a heuristic criterion known as ►Occam's Razor, which appears to work fairly well in practice, but is still the subject of ardent debates within the machine learning community.

Typical attribute selection criteria use a function that measures the *impurity* of a node, that is, the degree to which the node contains only examples of a single class. Two well-known impurity measures are the information-theoretic entropy (Quinlan, 1986), and the



Decision Tree. Figure 2. A needlessly complex decision tree describing the same dataset

Gini index (Breiman et al., 1984) which are defined as

$$\text{Entropy}(S) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \cdot \log_2\left(\frac{|S_i|}{|S|}\right)$$

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c}\left(\frac{|S_i|}{|S|}\right)^2$$

where S is a set of training examples, and S_i is the set of training examples that belong to class c_i. Both functions have their maximum at the point where the classes are equally distributed (i.e., where all S_i have the same size, maximum impurity), and their minimum at the point where one S_i contains all examples ($S_i = S$) and all other $S_j, j \neq i$ are empty (minimum impurity).

A good attribute divides the dataset into subsets that are as pure as possible, ideally into sets so that each one only contains examples from the same class. Thus, one wants to select the attribute that provides the highest decrease in average impurity, the so-called *gain*:

$$Gain(S, A) = Impurity(S) - \sum_t \frac{|S_t|}{|S|} \cdot Impurity(S_t)$$

where t is one of the tests on attribute A which partitions the set S is into nonoverlapping disjoint subsets S_t, and *Impurity* can be any impurity measure. As the first term, $Impurity(S)$, is constant for all attributes, one can also omit it and directly minimize the average impurity (which is typically done when *Gini* is used as an impurity measure).

A common problem is that attributes with many values have a higher chance of resulting in pure successor nodes and are, therefore, often preferred over attributes with fewer values. To counter this, the so-called *gain ratio* normalizes the gained entropy with the intrinsic entropy of the split:

$$GainRatio(S, A) = \frac{Gain(S, A)}{\sum_t \frac{|S_t|}{|S|} \cdot \log_2\left(\frac{|S_t|}{|S|}\right)}$$

A similar phenomenon can be observed for numerical attributes, where the number of possible threshold values determines the number of possible binary splits for this attribute. Numerical attributes with many possible binary splits are often preferred over numerical attributes with fewer splits because they have a higher chance that one of their possible splits fit the data. A discussion of this problem and a proposal for a solution can be found in Quinlan (1996).

Other attribute selection measures, which do not conform to the gain framework laid out above, are also possible, such as CHAID's evaluation with a χ^2 test statistic (Kass, 1980). Experimental comparison of different measures can be found in Buntine and Niblett (1992) and Mingers (1989a).

Thus, the final tree is constructed by a sequence of local choices that each consider only those examples that end up at the node that is currently split. Of course, such a procedure can only find local optima for each node, but cannot guarantee convergence to a global optimum (the smallest tree). One of the key advantages of this divide-and-conquer approach is its efficiency, which results from the exponential decrease in the quantity of data to be processed at successive depths in the tree.

Overfitting Avoidance

In principle, a decision tree model can be fit to any training set that does not contain contradictions (i.e., there are no examples with identical attributes but different class values). This may lead to ▶Overfitting in the form of overly complex trees.

For this reason, state-of-the-art decision tree induction techniques employ various ▶Pruning techniques for restricting the complexity of the found trees. For example, C4.5 has a ▶pre-pruning parameter *m* that is used to prevent further splitting unless at least two successor nodes have at least *m* examples. The *cost-complexity pruning* method used in CART may be viewed as a simple ▶Regularization method, where a good choice for the regularization parameter, which trades off the fit of the data with the complexity of the tree, is determined via ▶Cross-validation.

More typically, ▶post-pruning is used for removing branches and nodes from the learned tree. More precisely, this procedure replaces some of the interior nodes of the tree with a new leaf, thereby removing the subtree that was rooted at this node. An empirical comparison of different decision-tree pruning techniques can be found in Mingers (1989b).

It is important to note that the leaf nodes of the new tree are no longer pure nodes, that is, they no longer need to contain training examples that all belong to the same class. Typically, this is simply resolved by predicting the most frequent class at a leaf. The class distribution of the training examples within the leaf may be used as a reliability criterion for this prediction.

Well-known Decision Tree Learning Algorithms

The probably best-known decision tree learning algorithm is C4.5 (Quinlan, 1993) which is based upon ID3 (Quinlan, 1983), which, in turn, has been derived from an earlier concept learning system (Hunt et al., 1966). ID3 realized the basic recursive partitioning algorithm for an arbitrary number of classes and for discrete attribute values. C4.5 (Quinlan, 1993) incorporates several key improvements that were necessary for tackling real-world problems, including handling of numeric and ▶missing attribute values, ▶overfitting avoidance, and improved scalability. A C-implementation of C4.5 is freely available from its author. A re-implementation is available under the name J4.8 in the Weka data mining library. C5.0 is a commercial successor of C4.5, distributed by RuleQuest Research. CART (Breiman et al., 1984) is the best-known system in the statistical learning community. It is integrated into various statistical software packages, such as R or S.

Decision trees are also often used as components in ▶Ensemble Methods such as random forests (Breiman, 2001) or AdaBoost (Freund & Schapire, 1996). They can also be modified for predicting numerical target variables, in which case they are known as ▶Regression Trees. One can also put more complex prediction models into the leaves of a tree, resulting in ▶Model Trees.

Cross References

▶Decision List
▶Decision Lists and Decision Trees
▶Decision Stump
▶Divide-and-Conquer Learning
▶Model Tree
▶Pruning
▶Regression Tree
▶Rule Learning

Recommended Reading

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Pacific Grove: Wadsworth & Brooks.

Buntine, W., & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning, 8*, 75–85.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed), *Proceedings of the 13th international conference on machine learning, Bari, Italy* (pp. 148–156). San Francisco: Morgan Kaufmann.

Hunt, E. B., Marin, J., & Stone, P. J. (1966). *Experiments in induction*. New York: Academic.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics, 29*, 119–127.

Mingers, J. (1989a). An empirical comparison of selection measures for decision-tree induction. *Machine Learning, 3*, 319–342.

Mingers, J. (1989b). An empirical comparison of pruning methods for decision tree induction. *Machine Learning, 4*, 227–243.

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery, 2*(4), 345–389.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning. An artificial intelligence approach* (pp. 463–482). Palo Alto: Tioga.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research, 4*, 77–90.

Decision Trees For Regression

►Regression Trees

Deductive Learning

Synonyms

Analytical learning; Explanation-based learning

Definition

Deductive learning is a subclass of machine learning that studies algorithms for learning provably correct knowledge. Typically such methods are used to speedup problem solvers by adding knowledge to them that is deductively entailed by existing knowledge, but that may result in faster solutions.

Deduplication

►Entity Resolution

Deep Belief Nets

Geoffrey Hinton
University of Toronto, Toronto, Canada

Synonyms

Deep belief networks

Definition

Deep belief nets are probabilistic generative models that are composed of multiple layers of stochastic latent variables (also called "feature detectors" or "hidden units"). The top two layers have undirected, symmetric connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. Deep belief nets have two important computational properties. First, there is an efficient procedure for learning the top-down, generative weights that specify how the variables in one layer determine the probabilities of variables in the layer below. This procedure learns one layer of latent variables at a time. Second, after learning multiple layers, the values of the latent variables in every layer can be inferred by a single, bottom-up pass that starts with an observed data vector in the bottom layer and uses the generative weights in the reverse direction.

Motivation and Background

The perceptual systems of humans and other animals show that high-quality pattern recognition can be achieved by using multiple layers of adaptive nonlinear features, and researchers have been trying to understand how this type of perceptual system could be learned, since the 1950s (Selfridge, 1958). Perceptrons (Rosenblatt, 1962) were an early attempt to learn a biologically inspired perceptual system, but they did not have an efficient learning procedure for multiple layers of features. Backpropagation (Rumelhart, Hinton, & Williams, 1986; Werbos, 1974) is a supervised learning procedure that became popular in the 1980s because it provided a fairly efficient way of learning multiple layers of nonlinear features by propagating derivatives of the error in the output backward through the multilayer network. Unfortunately, backpropagation has difficulty optimizing the weights in deep networks that contain many layers of hidden units and it requires labeled training data, which is often expensive to obtain. Deep belief nets overcome the limitations of backpropagation by using *unsupervised* learning to create layers of feature detectors that model the statistical structure of the input data without using any information about the required output. High-level feature detectors that capture complicated higher-order statistical structure in the input data can then be used to predict the labels.

Structure of the Learning System

Deep belief nets are learned one layer at a time by treating the values of the latent variables in one layer, when they are being inferred from data, as the data for training the next layer. This efficient, greedy learning can be followed by, or combined with, other learning procedures that fine-tune all of the weights to improve the generative or discriminative performance of the whole network. Discriminative fine-tuning can be performed by adding a final layer of variables that represent the desired outputs and backpropagating error derivatives. When networks with many hidden layers are applied in domains that contain highly structured input vectors, backpropagation learning works much better if the feature detectors in the hidden layers are initialized by learning a deep belief net that models the structure in the input data (Hinton & Salakhutdinov, 2006). Matlab code for learning and fine-tuning deep belief nets can be found at http://cs.toronto.edu/~hinton.

Composing Simple Learning Modules

Early deep belief networks could be viewed as a composition of simple learning modules, each of which is a "restricted Boltzmann machine." Restricted Boltzmann machines contain a layer of "visible units" that represent the data and a layer of "hidden units" that learn to represent features that capture higher-order correlations in the data. The two layers are connected by a matrix of symmetrically weighted connections, W, and there are no connections within a layer. Given a vector of activities $\mathbf{v}$ for the visible units, the hidden units are all conditionally independent so it is easy to sample a vector, $\mathbf{h}$, from the posterior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{v}, \mathbf{W})$. It is also easy to sample from $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$. By starting with an observed data vector on the visible units and alternating several times between sampling from $p(\mathbf{h}|\mathbf{v}, \mathbf{W})$ and $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$, it is easy to get a learning signal which is simply the difference between the pairwise correlations of the visible and hidden units at the beginning and end of the sampling (see Chapter Boltzmann Machines for details).

The Theoretical Justification of the Learning Procedure

The key idea behind deep belief nets is that the weights, $\mathbf{W}$, learned by a restricted Boltzmann machine define both $p(\mathbf{v}|\mathbf{h}, \mathbf{W})$ and the prior distribution over hidden vectors, $p(\mathbf{h}|\mathbf{W})$, so the probability of generating a visible vector, $\mathbf{v}$, can be written as:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{W})p(\mathbf{v}|\mathbf{h}, \mathbf{W}) \tag{1}$$

After learning $\mathbf{W}$, we keep $p(\mathbf{v}|\mathbf{h},\mathbf{W})$ but we replace $p(\mathbf{h}|\mathbf{W})$ by a better model of the *aggregated* posterior distribution over hidden vectors – i.e., the nonfactorial distribution produced by averaging the factorial posterior distributions produced by the individual data vectors. The better model is learned by treating the hidden activity vectors produced from the training data as the training data for the next learning module. Hinton, Osindero, and Teh (2006) show that this replacement improves a variational lower bound on the probability of the training data under the composite model.

Deep Belief Nets with Other Types of Variable

Deep belief nets typically use the logistic function $y = 1/(1 + \exp(-x))$ of the weighted input, x, received from above or below to determine the probability that a binary latent variable has a value of 1 during top-down generation or bottom-up inference. Other types of variable within the exponential family, such as Gaussian, Poisson, or multinomial can also be used (Movellan & Marks, 2001; Welling, Rosen-Zvi, & Hinton, 2005) and the variational bound still applies. However, networks with multiple layers of Gaussian or Poisson units are difficult to train and can become unstable. To avoid these problems, the function $\log(1 + \exp(x))$ can be used as a smooth approximation to a rectified linear unit. Units of this type often learn features that are easier to interpret than those learned by logistic units. $\log(1+\exp(x))$

is not in the exponential family, but it can be approximated very accurately as a sum of a set of logistic units that all share the same weight vector and adaptive bias term, but differ by having offsets to the shared bias of $-0.5, -1.5, -2.5, \ldots$.

Using Autoencoders as the Learning Module

A closely related approach that is also called a "deep belief net" uses the same type of greedy, layer-by-layer learning with a different kind of learning module – an "autoencoder" that simply tries to reproduce each data vector from the feature activations that it causes (Bengio, Lamblin, Popovici, & Larochelle, 2007; Hinton, 1989; LeCun & Bengio, 2007). However, the variational bound no longer applies, and an autoencoder module is less good at ignoring random noise in its training data (Larochelle, Erhan, Courville, Bergstra, & Bengio, 2007).

Applications of Deep Belief Nets

Deep belief nets have been used for generating and recognizing images (Bengio, et al., 2007; Hinton et al., 2006; Ranzato, Huang, Boureau, & LeCun, 2007), video sequences (Sutskever & Hinton, 2007), and motion-capture data (Taylor, Hinton, & Roweis, 2007). If the number of units in the highest layer is small, deep belief nets perform nonlinear dimensionality reduction (Hinton & Salakhutdinov, 2006), and by pretraining each layer separately it is possible to learn very deep autoencoders that can then be fine-tuned with backpropagation (Hinton & Salakhutdinov, 2006). Such networks cannot be learned in reasonable time using backpropagation alone. Deep autoencoders learn compact representations of their input vectors that are much better than those found by linear methods such as Principal Components Analysis, and if the highest level code is forced to be binary, they allow extremely fast retrieval of documents or images (Salakhutdinov & Hinton, 2007; Torralba, Fergus, & Weiss, 2008).

Recommended Reading

Bengio, Y., Lamblin, P., Popovici, P., & Larochelle, H. (2007). Greedy layer-wise training of deep networks, In *Advances in neural information processing systems* (Vol. 19). Cambridge, MA: MIT Press.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence, 40*(1–3), 185–234.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*, 1527–1554.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*, 504–507.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on machine learning*. New York: ACM.

LeCun, Y., & Bengio, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou et al. (Eds.), *Large-scale kernel machines*. MA: MIT Press.

Movellan, J. R., & Marks, T. K. (2001). Diffusion networks, product of experts, and factor analysis.

Ranzato, M., Huang, F. J., Boureau, Y., & LeCun, Y. (2007) Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of computer vision and pattern recognition conference (CVPR 2007)*. Minneapolis, MN.

Rosenblatt, F. (1962). *Principles of neurodynamics*. Washington, DC: Spartan Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533-536.

Salakhutdinov, R. R., & Hinton, G. E. (2007). Semantic hashing. In *Proceedings of the SIGIR workshop on information retrieval and applications of graphical models*. Amsterdam, the Netherlands.

Selfridge, O. G. (1958) Pandemonium: A paradigm for learning. In *Mechanisation of though processes: Proceedings of a symposium held at the National Physical Laboratory*. London: HMSO.

Sutskever, I., & Hinton, G. E. (2007). Learning multilevel distributed representations for high-dimensional sequences. In *Proceedings of the eleventh international conference on artificial intelligence and statistics, San Juan,* Puerto Rico.

Taylor, G. W., Hinton, G. E., & Roweis, S. (2007). Modeling human motion using binary latent variables. In *Advances in neural information processing systems* (Vol. 19). Cambridge, MA: MIT Press.

Torralba, A., Fergus, R., & Weiss, Y. (2008). Small codes and large image databases for recognition. In *IEEE conference on computer vision and pattern recognition* (pp. 1–8). Anchorage, AK.

Welling, M., Rosen-Zvi, M., & Hinton, G. E. (2005). Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems* (Vol. 17, pp. 1481–1488). Cambridge, MA: MIT Press.

Werbos, P. (1974). *Beyond Regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA.

Deep Belief Networks

►Deep Belief Nets

Density Estimation

Claude Sammut
University of New South Wales, Sydney, Australia

Synonyms

Kernel density estimation

Definition

Given a set of observations, $x_1, \ldots, x_N$, which is a random sample from a probability density function $f_X(x)$, density estimation attempts to approximate $f_X(x)$ by $\widehat{f}_X(x_0)$.

A simple way of estimating a probability density function is to plot a histogram from a random sample drawn from the population. Usually, the range of data values is subdivided into equally sized intervals or *bins*. How well the histogram estimates the function depends on the bin width and the placement of the boundaries of the bins. The latter can be somewhat improved by modifying the histogram so that fixed boundaries are not used for the estimate. That is, the estimate of the probability density function at a point uses that point as the centre of a neighborhood. Following Hastie, Tibshirani and Friedman (2009), the estimate can be expressed as:

$$\widehat{f}_X(x_0) = \frac{\# x_i \in N(x_0)}{N\lambda} \tag{1}$$

where $x_1, \ldots, x_N$ is a random sample drawn from a probability density function $f_X(x)$ and $\widehat{f}_X(x_0)$ is the estimate of f_X at point x_0. $N(x_0)$ is a neighborhood of width λ, around x_0. That is, the estimate is the normalized count of the number of values that fall within the neighborhood of x_0.

The estimate above is still *bumpy*, like the histogram. A smoother approximation can be obtained by using a *kernel function*. Each x_i in the sample is associated with a kernel function, usually Gaussian. The count in formula (1) above is replaced by the sum of the kernel function applied to the points in the neighborhood of x_0:

$$\widehat{f}_X(x_0) = \frac{1}{N\lambda} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \tag{2}$$

where K is the kernel function associated with sample x_i near x_0. This is called the *Parzen* estimate (Parzen, 1962). The *bandwidth*, λ, affects the roughness or smoothness of the kernel histogram. The kernel density estimate is said to be under-smoothed if the bandwidth is too small. The estimate is over-smoothed if the bandwidth is too large.

Density estimation is most often used in association with memory-based classification methods, which can be thought of as weighted ►nearest neighbor classifiers.

►Mixture models and ►Locally weighted regression are forms of kernel density estimation.

Cross References

►Kernel Methods
►Locally Weighted Regression for Control
►Mixture Models
►Nearest Neighbor
►Support Vector Machine

Recommended Reading

Kernel Density estimation is well covered in texts including Hastie, Tibshirani and Friedman (2009), Duda, Hart and Stork (2001) and Ripley (Ripley, 1996).

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference and perception* (2nd ed.). New York: Springer.

Parzen, E. (1962). On the estimation of a probability density function and the mode. *Annals of Mathematics and Statistics*, *33*, 1065–1076.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge: Cambridge University Press.

Density-Based Clustering

Joerg Sander
University of Alberta
Edmonton, AB, Canada

Synonyms

Estimation of density level sets; Mode analysis; Nonparametric cluster analysis

Definition

Density-Based Clustering refers to ►unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in a data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density. The data points in the separating regions of low point density are typically considered noise/outliers.

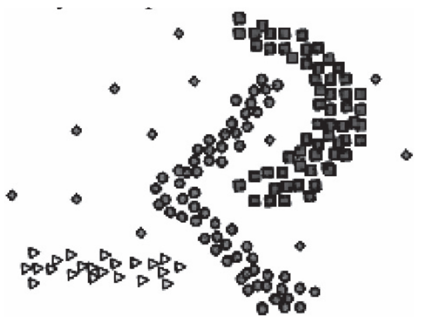
Motivation and Background

Clustering in general is an unsupervised learning task that aims at finding distinct groups in data, called clusters. The minimum requirements for this task are that the data is given as some set of objects O for which a dissimilarity-distance function $d : O \times O \rightarrow R^+$ is given. Often, O is a set of d-dimensional real valued points, $O \subset R^d$, which can be viewed as a sample from some unknown probability density $p(x)$, with d as the Euclidean or some other form of distance.

There are different approaches to classifying what characterizes distinct groups in the data.

From a procedural point of view, many clustering methods try to find a partition of the data into k groups, so that within-cluster dissimilarities are minimized while the between-cluster dissimilarities are maximized. The notions of within-cluster dissimilarity and between-cluster dissimilarity are defined using the given distance function d. From a statistical point of view, such methods correspond to a parametric approach, where the unknown density $p(x)$ of the data is assumed to be a mixture of k densities $p_i(x)$, each corresponding to one of the k groups in the data; the $p_i(x)$ are assumed to come from some parametric family (e.g., Gaussian distributions) with unknown parameters, which are then estimated from the data.

In contrast, *density-based* clustering is a nonparametric approach, where the groups in the data are considered to be the high density areas of the density $p(x)$. Density-based clustering methods do not require the number of clusters as input parameters, nor do they make assumptions about the underlying density $p(x)$ or the variance within the groups that may exist in the data. Consequently, density-based clusters are not necessarily groups of points with high within-cluster similarity as measured by the distance function d, but can have an "arbitrary shape" in the feature space; they are sometimes also referred to as "natural clusters." This property makes density-based clustering particularly suitable for applications where clusters cannot be well described as distinct groups of low within-cluster dissimilarity, as, for instance, in spatial data, where clusters of points in the space may form along natural structures such as rivers, roads, and seismic faults. Figure 1 illustrates density-based clusters using a two-dimensional example, where the assumed dissimilarity function between the points is the Euclidean distance: there are three clusters indicated by triangles, points, and rectangles, as well as some noise points, indicated by diamond shapes. Note that the distance between some points within the clusters is much larger than the distance between some points from different clusters, yet the regions containing the clusters clearly have a higher point density than the region between them, and they can easily be separated.


Density-Based Clustering. Figure 1. Illustration of a density-based clustering, showing three distinguishable groups

Density-based clustering is one of the prominent paradigms for clustering large data sets in the data mining community. It has been extensively studied and successfully used in many applications.

Structure of Learning System

Assuming that the data set $O \subset R^d$ is a sample from some unknown probability density $p(x)$, there are different ways of determining high density areas of the density $p(x)$. Commonly, the notion of a high density area is (implicitly or explicitly) based on a local density estimate at each point (typically, some kernel or nearest neighbor density estimate), and a notion of connection between objects (typically, points are connected if they are within a certain distance ε from each other); clusters are essentially constructed as maximal sets of objects that are directly or transitively connected to objects whose density exceeds some threshold λ. The set $\{x | p(x) > \lambda\}$ of all high density objects is called the *density level set* of p at λ. Objects that are not part of such clusters are called *noise* or *outliers*.

Different proposed density-based methods distinguish themselves mainly by how the density $p(x)$ is estimated, how the notion of connectivity is defined, and how the algorithm for finding connected components of the induced graph is implemented and supported by suitable data structures to achieve scalability for large data sets. Some methods include in a cluster only objects whose density exceeds the threshold λ, while others also include objects with lower density if they are connected to an object with density above the threshold λ.

Density-based clustering was probably introduced for the first time by Wishart (1969). His algorithm for *one level mode analysis* consists of six steps: "(1) Select a distance threshold r, and a frequency (or density) threshold k, (2) Compute the triangular similarity matrix of all inter-point distances, (3) Evaluate the frequency k_i of each data point, defined as the number of points which lie within a distance r of point i (...), (4) Remove the "noise" or non-dense points, those for which $k_i < k$, (5) Cluster the remaining dense points ($k_i > k$) by single linkage, forming the mode nuclei, (6) Reallocate each non-dense point to a suitable cluster according to some criterion (...) (Wishart, 1969).

Hartigan (1975) suggested a more general definition of a density-based cluster, a *density contour cluster* at level λ, as a maximally connected set of points x for which $p(x) > \lambda$, given a density $p(x)$ at each point x, a density threshold λ, and links specified for some pairs of objects. For instance, given a particular distance function, points can be defined as linked if the distance between them is no greater than some threshold r, or, if only direct links are available, one can define a "distance" for pairs of objects x and y in the following way:

$$d(x,y) = \begin{cases} -\min[p(x), p(y)], & x \text{ and } y \text{ are linked,} \\ 0 & \text{otherwise.} \end{cases}$$

To compute the density-contour clusters, Hartigan, like Wishart, suggest a version of single linkage clustering, which will construct the maximal connected sets of objects of density greater than the given threshold λ.

The DBSCAN algorithm (Ester et al., 1996) introduced density-based clustering independently to the Computer Science Community, also proposing the use of spatial index structures to achieve a scalable clustering algorithm. Assuming a distance threshold r and a density threshold k, DBSCAN, like Wishart's method, estimates the density for each point x_i as the number k_i of points that lie inside a radius r around x. *Core points* are defined as data points for which $k_i > k$. Points are considered directly connected if the distance between them is no greater than r. Density-based clusters are defined as maximally connected components of the set of points that lie within distance r from some core object (i.e., a cluster may contain points x_i with $k_i < k$, called *border objects*, if they are within distance r of a core object of that cluster). Objects not part of a cluster are considered *noise*. The algorithm DBSCAN constructs clusters iteratively, starting a new cluster C with a non-assigned core object x, and assigning all points to C that are directly or transitively connected to x. To determine directly and transitively connected points for a given point, a spatial index structure is used to perform range queries with radius r for each object that is newly added to a current cluster, resulting in an efficient runtime complexity for moderately dimensional data of $O(N \log N)$, where N is the total number of points in the data set, and a worst case runtime of $O(N^2)$, e.g., for high-dimensional data when the performance of spatial index structures deteriorates.

DENCLUE (Hinneburg and Keim, 1998) proposed a notion of density-based clusters using a kernel density estimation. Each data point x is associated with ("attracted by") a local maximum ("density attractor") of the overall density function that lies in the direction of maximum increase in density from x. Density-based clusters are defined as connected components of density attractors with their associated points, whose density estimate is above a given threshold λ. In this formulation, DBSCAN and Wishart's method can be seen as special cases of DENCLUE, using a uniform spherical kernel and, for Wishart's method, not including attracted points whose density is below λ. DENCLUE essentially uses a Gaussian kernel for the implementation, which is based on a clever data structure to speed up local density estimation. The data space is partitioned into d-dimensional cells. Non-empty cells are mapped to one-dimensional keys, which are stored, together with some sufficient statistics about the cell (number of points, pointers to points, and linear sum of the points belonging to the cell), in a search tree for efficient retrieval of neighboring cells, and local density estimation (Hinneburg and Keim (1998) reports that in an experimental comparison on 11-dimensional data sets of different sizes, DENCLUE runs up to 45 times faster than DBSCAN).

A large number of related methods and extensions have been proposed, particularly in computer science and application-oriented domains, some motivated by algorithmic considerations that could improve efficiency of the computation of density-based clusters, others motivated by special applications, proposing essentially density based clustering algorithms using specific density measures and notions of connectivity. An algorithmic framework, called GDBSCAN, which generalizes the topological properties of density-based clusters, can be found in Sander et al. (1998). GDBSCAN generalizes the notion of a density-based clustering to that of a *density-connected decomposition*, assuming only a reflexive and symmetric *neighborhood* relation for pairs of objects (direct links between some objects), and an arbitrary predicate, called "*MinWeight*," that evaluates to *true* for some neighborhood sets of objects and *false* on others, so that a core object can be defined as an object whose neighborhood satisfies the *MinWeight* predicate. Then, a density-connected decomposition consists of the maximally connected components of the set of objects that are in the neighborhood of some core object, and they can be computed with the same algorithmic scheme as density-based clusters by DBSCAN.

One of the principal problems of finding the density-based clusters of a density level set for a single level λ is how to determine a suitable level λ. The result of a density-based clustering method depends critically on the choice of λ, which may be difficult to determine even in situations when a meaningful level exists, depending on how well the clusters are separated in the given sample. In other situations, it may not even be possible to characterize the cluster structure appropriately using a single density threshold, when modes exist in different regions of the data space that have very different local densities, or clusters are nested within clusters. The problem of selecting suitable density threshold parameters has been already observed by Wishart (1969) who also proposed a hierarchical algorithm to represent the clusters at different density levels. Hartigan (1975) also observed that density-based clusters at different density levels have a hierarchical structure, a *density contour tree*, based on the fact that two clusters (i.e., connected components) of different density levels are either disjoint, or the cluster of higher density is completely contained in the cluster of lower density. Recent proposals for hierarchical clustering methods based on a density estimate and a notion of linkage are, e.g., Ankerst et al. (1999) and Stuetzle (2003). These hierarchical methods are closely related, and are essentially processing and rendering a Minimum Spanning Tree of the data (with pairwise distances or reachability distances as defined in Stuetzle (2003) as edge weights), and are thus also closely related to single linkage clustering. Hierarchical methods do not, in a strict sense, compute a partition of the data, but compute a representation of the overall hierarchical density structure of the data from which particular density-based clusters at different density levels or a global density threshold (a "cut level") could be determined.

Cross References

►Clustering
►Density Estimation

Recommended Reading

Ankerst, M., Breunig, M. M., Kriegel, H.-P., Sander, J. (1999). OPTICS: Ordering Points to Identify the Clustering Structure. In A. Delis, C. Faloutsos, S. Ghandeharizadeh (Eds.), *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. Philadelphia: ACM.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In E. Simoudis, J. Han, & U.M. Fayyad (Eds.), *Proceedings of the second International Conference on Knowledge Discovery and Data Mining*. Portland: AAAI Press.

Hartigan, J. A. (1975). *Clustering Algorithms*. New York: Wiley.

Hinneburg, A., Keim, D. A. (1998). En Efficient Approach to Clustering in Large Multimedia Databases with Noise. In R. Agrawal, & P. Stolorz (Eds.), *Proceedings of the fourth International Conference on Knowledge Discovery and Data Mining*. New York City: AAAI Press.

Sander, J., Ester, M., Kriegel, H.-P., Xu, X. (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery, 2*(2), 169–194.

Stuetzle, W. (2003). Estimating the Cluster Tree of a Density by Analyzing the Minimal Spanning Tree of a Sample. *Journal of Classification, 20*(1), 025–047.

Wishart, D. (1969). Mode analysis: A generalization of nearest neighbor which reduces chaining effects. In A. J. Cole (Ed.), *Proceedings of the Colloquium in Numerical Taxonomy*. Scotland: St. Andrews.

Dependency Directed Backtracking

►Intelligent Backtracking

Detail

In ►Minimum Message Length, *detail* is the code or language shared between sender and receiver that is used to describe the data conditional on the asserted model.

Deterministic Decision Rule

►Decision Rule

Digraphs

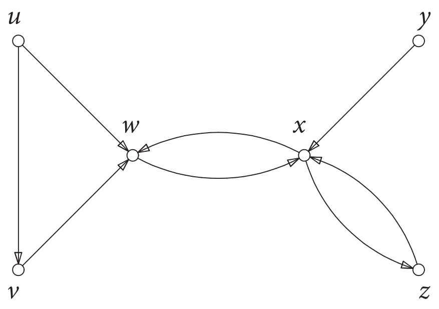
Synonyms

Directed graphs

Definition

A *digraph* D consists of a (finite) set of *vertices* $V(D)$ and a set $A(D)$ of ordered pairs, called *arcs*, of distinct vertices. An arc (u, v) has *tail* u and *head* v, and it is said to leave u and enter v.

Figure 1 shows a digraph D with vertex set $V(D) = \{u, v, w, x, y, z\}$ and arc set $A(D) = \{(u, v), (u, w), (v, w), (w, x), (x, w), (x, z), (y, x), (z, x)\}$. Digraphs can be viewed as generalizations of ►graphs.



Digraphs. Figure 1. A digraph

Dimensionality Reduction

Michail Vlachos
IBM Zürich Research Laboratory
Rüschlikon
Switzerland

Synonyms

Feature extraction

Definition

Every data object in a computer is represented and stored as a set of features, for example, color, price, dimensions, and so on. Instead of the term *features* one can use interchangeably the term *dimensions*, because an object with n features can also be represented as a multidimensional point in an n-dimensional space. Therefore, dimensionality reduction refers to the process of mapping an n-dimensional point, into a lower k-dimensional space. This operation reduces the size for representing and storing an object or a dataset generally; hence, dimensionality reduction can be seen as a method for data compression. Additionally, this process promotes data visualization, particularly when objects are mapped onto two or three dimensions. Finally, in the context of classification, dimensionality reduction can be a useful tool for the following: (a) making tractable classification schemes that are super-linear with respect to dimensionality, (b) reducing the variance of classifiers that are plagued by large variance in higher dimensionalities, and (c) removing the noise that may be present, thus boosting classification accuracy.

Motivation and Background

There are many techniques for dimensionality reduction. The objective of dimensionality reduction techniques is to appropriately select the k dimensions (and also the number k) that would retain the important characteristics of the original object. For example, when performing dimensionality reduction on an image, using a wavelet technique, then the desirable outcome is for the difference between the original and final images to be almost imperceptible.

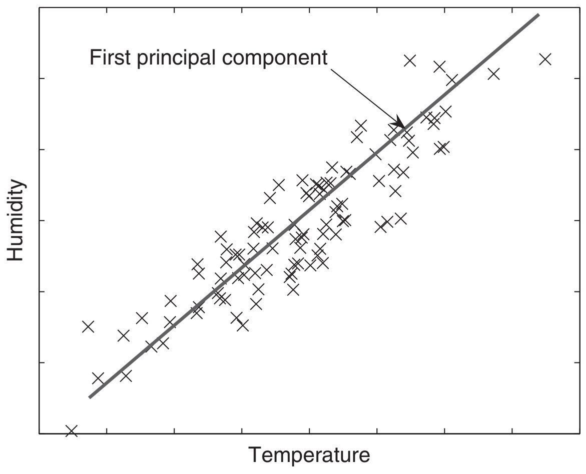
When performing dimensionality reduction not on a single object, but on a dataset, an additional requirement is for the method to preserve the relationship between the objects in the original space. This is particularly important for reasons of classification and visualization in the new space.

There exist two important categories of dimensionality reduction techniques:

- *Feature selection* techniques, where only the most important or descriptive features/dimensions are retained and the remaining are discarded. More details on such techniques can be found under the entry ►Feature Selection.
- *Feature projection* methodologies, which project the existing features onto different dimensions or axes. The aim here is again, to find these new data axes that retain the dataset structure and its variance as closely as possible.

Feature projection techniques typically exploit the correlations between the various data dimensions, with the goal of creating dimensions/axes that are uncorrelated and sufficiently describe the data.

One of the most popular dimensionality reduction techniques is *Principal Components Analysis* or PCA. It attempts to discover those axes (or components) onto which the data can be projected, while maintaining the original correlation between the dimensions. Consider, for example, a dataset that contains records of environmental measurements over a period of time, such as humidity and temperature. The two attributes can



Dimensionality Reduction. Figure 1. Principal components analysis (PCA)

be highly correlated, as shown in Fig. 1. By deploying PCA this trend will be discovered and the original two-dimensional points can be reduced to one-dimensional, by projecting the original points on the first *principal component*. In that way the derived dataset can be stored in less space.

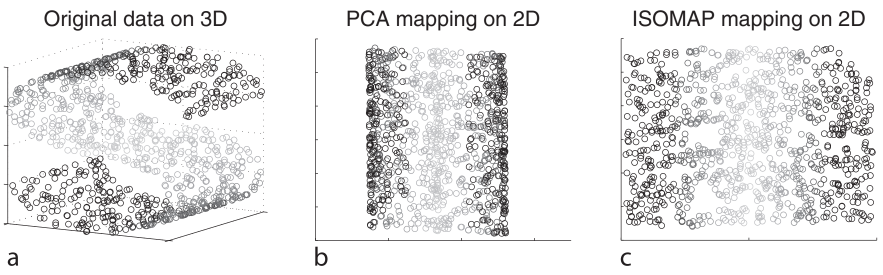
PCA uses the Euclidean distance as the measure of dissimilarity among the objects. The first principal component (or axis) indicates the direction of maximum variance in the original dimensions. The second component shows the direction of next highest variance (and is uncorrelated to the first component), etc.

Other dimensionality reduction techniques optimize or preserve different criteria than PCA. Manifold inspired methods like ISOMAP (Tenenbaum et al., 2000) preserve the geodesic distances between objects. The notion here is to approximate the distance between objects "through" the remaining ones. The result of such dimensionality reduction techniques, is that when the data lie on a manifold, the projected dimensions effectively 'unfold' the underlying high-dimensional manifold. An example of this mapping is portrayed in Fig. 2, where it is also compared with the respective PCA mapping.

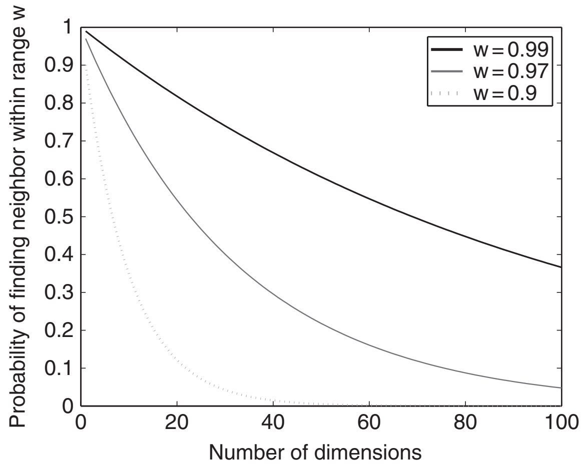
Other recent dimensionality reduction techniques include locally linear embeddings (LLE) (Roweis and Saul, 2000) and Laplacian Eigenmaps (Belkin and Niyogi, 2002). We also refer the interested practitioners to (van der Maaten et al., 2009), for a detailed comparison of various techniques and also for Matlab implementations on a variety of dimensionality reduction algorithms.

In general, dimensionality reduction is a commonly practiced and useful operation in database and machine learning systems because it generally offers the following desirable properties:

- Data compression: the dataset objects are represented in fewer dimensions, hence saving important disk storage space and offering faster loading of the compressed data from the disk.
- Better data visualization: the relationships between the original high-dimensional objects can be visualized in two- or three-dimensional projections.
- Improved classification accuracy: this can be attributed to both variance reduction and noise removal from the original high-dimensional dataset.



Dimensionality Reduction. Figure 2. Nonlinear dimensionality reduction techniques produce a better low-dimensional data mapping, when the original data lie on a high-dimensional manifold



Dimensionality Reduction. Figure 3. Probability $P_w(d)$ against dimensionality d. The data becomes sparse in higher dimensions

- More efficient data retrieval: dimensionality reduction techniques can also assist in making faster and more efficient the retrieval of the original uncompressed data, by offering very fast prefiltering with the help of the compressed data representation.
- Boosting index performance: more effective use of indexing structures can be achieved by utilizing the compressed data, since indexing techniques only work efficiently with lower-dimensional data (e.g., from 1 to 30 dimensions, depending on the type of the index).

The fact that indexing structures do not perform efficiently for higher-dimensional data is also known as ►"curse of dimensionality." Suppose that we are interested in performing search operations on a set of high-dimensional data. For simplicity let us assume that the data lie in a unit hypercube $C = [0,1]^d$, where d is the data dimensionality. Given a query point, the probability P_w that a match (neighbor) exists within radius w in the data space of dimensionality d is given by $P_w(d) = w^d$.

Figure 3 illustrates this probability for various values of w. Evidently, at higher dimensionalities the data becomes very sparse and even at large radii, only a small portion of the entire space is covered. This fact is coined under the term 'curse of dimensionality,' which in simple terms translates into the following fact: for large dimensionalities existing indexing structures outperform sequential scan only when the dataset size (number of objects) grows exponentially with respect to dimensionality.

Dimensionality Reduction for Time-Series Data

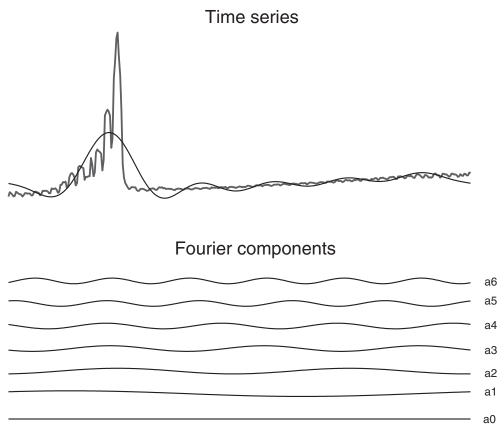
In this section we provide more detailed examples on dimensionality reduction techniques for ►time-series data. We chose time-series in order to convey more visually the effect of dimensionality reduction particularly for high-dimensional data such as time-series.

Later, we also show how dimensionality reduction on large datasets can help speed up the search operations over the original uncompressed data.

Dimensionality reduction for one- and two-dimensional signals is commonly accomplished using the Fourier decomposition. Fourier decomposition was first presented in the beginning of the nineteenth century by Jean Baptiste Fourier (1768–1830), in his seminal work *On the Propagation of Heat in Solid Bodies*. Fourier reached the conclusion that every function could be expressed as a sum of trigonometrical series (i.e., sines and cosines). This original work was initially faced with doubt (even by famous mathematicians such as Lagrange and Laplace), because of its unexpected result and because the solution was considered impractical due of the complex integration functions.

However, in the twentieth century no one can deny the importance of Fourier's findings. With the introduction of fast ways to compute the Fourier decomposition in the 1960s (Fast Fourier Transform or FFT), the barrier of the high computational complexity has been lifted. What the Fourier transform attempts to achieve is, represent the original signal as a linear combination of sinusoids. Therefore, each Fourier coefficient is a complex number that essentially encodes the amplitude and phase of each of these sinusoids, after the original signal is projected on them.

For most signals, utilizing just few of the coefficients we can reconstruct with high accuracy the original sequence. This is where the great power of the Fourier transformation lies; by neglecting the majority of the coefficients, we can essentially compress the signal or describe it with fewer numbers. For stock market data or other time-series that follow the pattern of a random walk, the first few coefficients, which capture the low frequencies of the signal, are adequate to describe accurately the signal (or capture most of its energy). Figure 4 depicts a signal of 1,024 points and its reconstruction using seven Fourier coefficients (i.e., using $7 \times 2 = 14$ numbers).



Dimensionality Reduction. Figure 4. Decomposition of a signal into the first seven Fourier coefficients. We can see that using only of few of the Fourier coefficients we can achieve a good reconstruction of the original signal

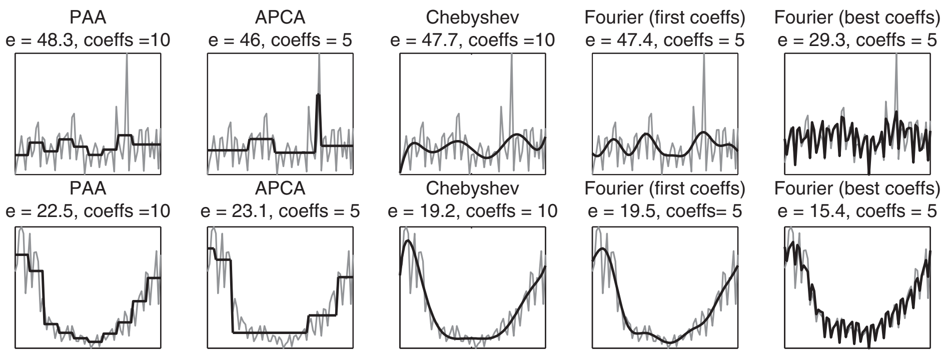
Other popular dimensionality reduction techniques for time-series data are the various wavelet transforms, piecewise linear approximations, piecewise aggregate approximation (PAA), which can be regarded as a projection in time of the wavelet coefficients, adaptive piecewise constant approximation (APCA (Keogh et al., 2001)), that utilizes the highest energy wavelet coefficients, Chebyshev Polynomial Approximation and Symbolic Approximation of time-series (such as the SAX representation (Lin et al., 2003)).

No dimensionality reduction technique is universally better than all the rest. According to the dataset characteristics, one method may provide better approximation of a dataset compared to other techniques. Therefore, the key is to carefully pick the representation that better suits the specific application or the task at hand. In Fig. 5 we demonstrate various dimensionality reduction techniques and the quality of the time-series approximation. For all of the methods, the same storage space is allocated for the compressed sequences. The time-series reconstruction is shown in darker color, and the approximation error to the original sequence is also reported. In general, we can notice that dimensionality reduction techniques based on the selection of the highest energy coefficients can consistently provide a high quality sequence approximation.

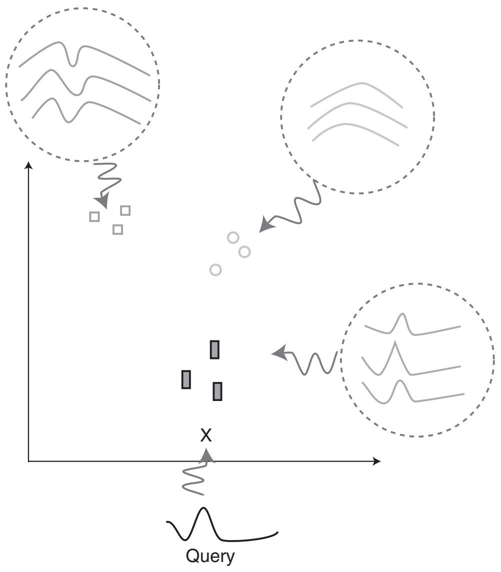
Dimensionality Reduction and Lower-Bounding

Dimensionality reduction can be a useful tool for speeding up search operations. Figure 6 elucidates dimensionality reduction for high-dimensional time-series data. After dimensionality reduction, each object is represented using fewer dimensions (attributes), so it is represented in a lower-dimensional space. Suppose that a user poses another high-dimensional object as a query and wishes to find all the objects closest to this query.

In order to avoid the search on the original high-dimensional space, the query is also transformed into a point in the low-dimensional space and its closest matches can be discovered in the vicinity of the projected query point. However, when searching using the compressed objects, one needs to provide an estimate of the distance between the original objects. Typically, it is preferable that the distance in the new space underestimates (or lower bounds) the distance in the original high-dimensional space. The reason for this is explained as follows.



Dimensionality Reduction. Figure 5. Comparison of various dimensionality reduction techniques for time-series data. The *darker series* indicates the approximation using the indicated number of coefficients. Each figure also reports the error *e* introduced by the dimensionality reduction technique. Lower errors indicate better low-dimensional approximation of the original object



Dimensionality Reduction. Figure 6. Search and dimensionality reduction. Every object (time-series in this case) is tranformed into a lower-dimensional point. User queries are also projected into the new space. Similarity search consists in finding the closest points to the query projection

Suppose that we are seeking for the 1-NN (►Nearest-Neighbor) of a query Q in a database $\mathcal{D}$. By examining all the objects (linear scan) one can guarantee that the best match will be found. Can one provide the same guarantee (i.e., that the same best match will be returned) when examining the compressed objects (after dimensionality reduction)?

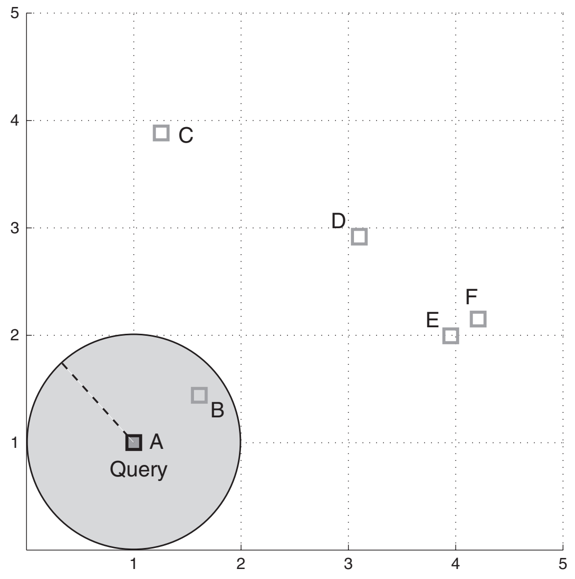
The answer is positive, as long as the distance on the compressed data *underestimates* or *lower bounds* the distance on the raw data. In other words, the dimensionality reduction (dR) that is performed on the raw data must have the following property:

$$\text{Having} \quad A \subset \mathcal{D} \xrightarrow{dR} a \quad \text{and} \quad Q \xrightarrow{dR} q$$
$$\text{then}$$
$$\Delta(q, a) \leq \Delta(Q, A)$$

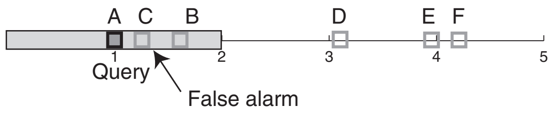
Since the computed distance Δ between any two compressed objects is underestimated, *false alarms* may arise. Suppose, for example, that our database consists of six two-dimensional point (Fig. 7). If the user query is "Find everything that lies within a radius of 1 around A," then B is the only result.

Let us assume for a minute that the dimensionality reduction that is performed on the data is simply a projection on the x-axis (Fig. 8). In this new space, seeking for points within a range of 1 from A, would also retrieve point C, which is called a *false alarm*. This does not constitute a problem, because in a post-processing phase, the calculation of the exact distance will eliminate any false alarms. Suppose now, that another dimensionality reduction results in the projection of Fig. 9. Here, we have a case of a *false dismissal*, since object B lies outside the range of search.

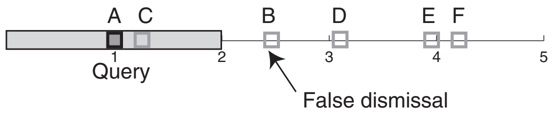
This generic framework for similarity search using dimensionality reduction and lower-bounding distance



Dimensionality Reduction. Figure 7. **Range search in the original space, returns only object *B***



Dimensionality Reduction. Figure 8. **Because of the dimensionality reduction, false alarms may arise**



Dimensionality Reduction. Figure 9. **False dismissals may happen when the lower bounding lemma is not obeyed**

functions was proposed in (Agrawal et al., 1993) and is called GEMINI (**GE**neric **M**ultimedia **IN**dex**I**ng). One can show that orthonormal dimensionality reduction techniques (PCA, Fourier, Wavelets) satisfy the lower bounding lemma when the distance used is the Euclidean distance.

In conclusion, for search operations, by using dimensionality reduction one can examine first the compressed objects and eliminate many of the uncompressed objects from examination using a lower-bounding approximation of the distance function. This initial search will return a superset of the correct answers (no false dismissals). False alarms can be filtered out by computing the original distance between the remaining uncompressed objects and the query. Therefore, a significant speedup is achieved by examining only a small subset of the original raw data.

Cross References

►Curse of Dimensionality
►Feature Selection

Recommended Reading

Agrawal, R., Faloutsos, C., & Swami, A. (1993). Efficient similarity search in sequence databases. *Proceedings of Foundations of Data Organization and Algorithms*, Chicago, Illinois. (pp. 69–84).

Belkin, M., & Niyogi, P. (2002). Laplacian Eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*. (Vol. 14, pp. 585–591). Canada: Vancouver.

Jolliffee, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer.

Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. *Proceedings of ACM SIGMOD*, (pp. 151–162). Santa Barbara, CA.

Lin, J., Keogh, E., Lonardi, S., & Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. San Diego, CA.

Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*(5500), 2319–2323.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science, 290*(5500), 2323–2326.

van der Maaten, L. J. P., Postma, E. O., and van den Herik, H. J. (2009). *Dimensionality reduction: a comparative review* (TiCC-TR 2009-005), Tilburg University Technical Report.

Dimensionality Reduction on Text via Feature Selection

►Feature Selection in Text Mining

Directed Graphs

►Digraphs

Dirichlet Process

Yee Whye Teh
University College London,
London, UK

Definition

The Dirichlet process (DP) is a stochastic process used in ►Bayesian nonparametric models of data, particularly in Dirichlet process mixture models (also known as infinite mixture models). It is a distribution over distributions, that is, each draw from a Dirichlet process is itself a distribution. It is called a Dirichlet process because it has Dirichlet distributed finite dimensional marginal distributions, just as the ►Gaussian process, another popular stochastic process used for Bayesian nonparametric regression, has Gaussian distributed finite dimensional marginal distributions. Distributions drawn from a Dirichlet process are discrete, but cannot be described using a finite number of parameters, thus the classification as a nonparametric model.

Motivation and Background

Probabilistic models are used throughout machine learning to model distributions over observed data. Traditional parametric models using a fixed and finite number of parameters can suffer from over- or underfitting of data when there is a misfit between the complexity of the model (often expressed in terms of the number of parameters) and the amount of data available. As a result, model selection, or the choice of a model with the right complexity, is often an important issue in parametric modeling. Unfortunately, model selection is an operation that is fraught with difficulties, whether we use ►cross validation or marginal probabilities as the basis for selection. The Bayesian nonparametric approach is an alternative to parametric modeling and selection. By using a model with an unbounded complexity, underfitting is mitigated, while the Bayesian approach of computing or approximating the full posterior over parameters mitigates overfitting. For a general overview of Bayesian nonparametrics, see ►Bayesian Nonparametrics.

Nonparametric models are also motivated philosophically by Bayesian modeling. Typically we assume that we have an underlying and unknown distribution which we wish to infer given some observed data. Say we observe $x_1, \ldots, x_n$, with $x_i \sim F$ independent and identical draws from the unknown distribution F. A Bayesian would approach this problem by placing a prior over F then computing the posterior over F given data. Traditionally, this prior over distributions is given by a parametric family. But constraining distributions to lie within parametric families limits the scope and type of inferences that can be made. The nonparametric approach instead uses a prior over distributions with wide support, typically the support being the space of all distributions. Given such a large space over which we make our inferences, it is important that posterior computations are tractable.

The Dirichlet process is currently one of the most popular Bayesian nonparametric models. It was first formalized in Ferguson (1973) for general Bayesian statistical modeling, as a prior over distributions with wide support yet tractable posteriors. (Note however that related models in population genetics date back to Ewens (1972)). Unfortunately the Dirichlet process is limited by the fact that draws from it are discrete distributions, and generalizations to more general priors did not have tractable posterior inference until the development of MCMC (►Markov chain Monte Carlo) techniques (Escobar & West, 1995; Neal, 2000). Since then there has been significant developments in terms of inference algorithms, extensions, theory and applications. In the machine learning, community work on Dirichlet processes date back to Neal (1992) and Rasmussen (2000).

Theory

The Dirichlet process (DP) is a stochastic process whose sample paths are probability measures with probability one. Stochastic processes are distributions over function spaces, with sample paths being random functions drawn from the distribution. In the case of the DP, it is a distribution over probability measures, which are functions with certain special properties, which allow them to be interpreted as distributions over some probability space Θ. Thus draws from a DP can be interpreted as random distributions. For a distribution over probability measures to be a DP, its marginal distributions have to take on a specific form which we shall give below. We assume that the user is familiar with a modicum of measure theory and Dirichlet distributions.

Before we proceed to the formal definition, we will first give an intuitive explanation of the DP as an infinite dimensional generalization of Dirichlet distributions. Consider a Bayesian mixture model consisting of K components:

$$\begin{aligned} \pi|\alpha &\sim Dir\left(\tfrac{\alpha}{K},\dots,\tfrac{\alpha}{K}\right) & \theta_k^*|H &\sim H \\ z_i|\pi &\sim Mult\left(\pi\right) & x_i|z_i,\{\theta_k^*\} &\sim F\left(\theta_{z_i}^*\right) \end{aligned} \quad (1)$$

where π is the mixing proportion, α is the pseudo-count hyperparameter of the Dirichlet prior, H is the prior distribution over component parameters θ_k^*, and $F(\theta)$ is the component distribution parametrized by θ. It can be shown that for large K, because of the particular way we parametrized the Dirichlet prior over π, the number of components typically used to model n data items becomes independent of K and is approximately $O(\alpha \log n)$. This implies that the mixture model stays well defined as $K \to \infty$, leading to what is known as an infinite mixture model (Neal, 1992; Rasmussen, 2000). This model was first proposed as a way to sidestep the difficult problem of determining the number of components in a mixture, and as a nonparametric alternative to finite mixtures whose size can grow naturally with the number of data items. The more modern definition of this model uses a DP and with the resulting model called a DP mixture model. The DP itself appears as the $K \to \infty$ limit of the random discrete probability measure $\sum_{k=1}^{K} \pi_k \delta_{\theta_k^*}$, where δ_θ is a point mass centered at θ. We will return to the DP mixture toward the end of this entry.

Dirichlet Process

For a random distribution G to be distributed according to a DP, its marginal distributions have to be Dirichlet distributed (Ferguson, 1973). Specifically, let H be a distribution over Θ and α be a positive real number. Then for any finite measurable partition $A_1,\dots,A_r$ of Θ the vector $(G(A_1),\dots,G(A_r))$ is random since G is random. We say G is Dirichlet process distributed with base distribution H and concentration parameter α, written $G \sim DP(\alpha, H)$, if

$$(G(A_1),\dots,G(A_r)) \sim Dir\left(\alpha H(A_1),\dots,\alpha H(A_r)\right) \quad (2)$$

for every finite measurable partition $A_1,\dots,A_r$ of Θ.

The parameters H and α play intuitive roles in the definition of the DP. The base distribution is basically the mean of the DP: for any measurable set $A \subset \Theta$, we have $E[G(A)] = H(A)$. On the other hand, the concentration parameter can be understood as an inverse variance: $V[G(A)] = H(A)(1-H(A))/(\alpha+1)$. The larger α is, the smaller the variance, and the DP will concentrate more of its mass around the mean. The concentration parameter is also called the strength parameter, referring to the strength of the prior when using the DP as a nonparametric prior over distributions in a Bayesian nonparametric model, and the mass parameter, as this prior strength can be measured in units of sample size (or mass) of observations. Also, notice that α and H only appear as their product in the definition (3) of the DP. Some authors thus treat $\widetilde{H} = \alpha H$, as the single (positive measure) parameter of the DP, writing $DP(\widetilde{H})$ instead of $DP(\alpha, H)$. This parametrization can be notationally convenient, but loses the distinct roles α and H play in describing the DP.

Since α describes the concentration of mass around the mean of the DP, as $\alpha \to \infty$, we will have $G(A) \to H(A)$ for any measurable A, that is $G \to H$ weakly or pointwise. However this not equivalent to saying that $G \to H$. As we shall see later, draws from a DP will be discrete distributions with probability one, even if H is smooth. Thus G and H need not even be absolutely continuous with respect to each other. This has not stopped some authors from using the DP as a nonparametric relaxation of a parametric model given by H. However, if smoothness is a concern, it is possible to extend the DP by convolving G with kernels so that the resulting random distribution has a density.

A related issue to the above is the coverage of the DP within the class of all distributions over Θ. We already noted that samples from the DP are discrete, thus the set of distributions with positive probability under the DP is small. However it turns out that this set is also large in a different sense: if the topological support of H (the smallest closed set S in Θ with $H(S) = 1$) is all of Θ, then any distribution over Θ can be approximated arbitrarily accurately in the weak or pointwise sense by a sequence of draws from $DP(\alpha, H)$. This property has consequence in the consistency of DPs discussed later.

For all but the simplest probability spaces, the number of measurable partitions in the definition (3) of the DP can be uncountably large. The natural question to ask here is whether objects satisfying such a large number of conditions as (3) can exist. There are a number of approaches to establish existence. Ferguson (1973) noted that the conditions (3) are consistent with each other, and made use of Kolmogorov's consistency theorem to show that a distribution over functions from the measurable subsets of Θ to $[0,1]$ exists satisfying (3) for all finite measurable partitions of Θ. However it turns out that this construction does not necessarily guarantee a distribution over probability measures. Ferguson (1973) also provided a construction of the DP by normalizing a gamma process. In a later section we will see that the predictive distributions of the DP are related to the Blackwell–MacQueen urn scheme. Blackwell and MacQueen (1973) made use of this, along with de Finetti's theorem on exchangeable sequences, to prove existence of the DP. All the above methods made use of powerful and general mathematical machinery to establish existence, and often require regularity assumptions on H and Θ to apply these machinery. In a later section, we describe a stick-breaking construction of the DP due to Sethuraman (1994), which is a direct and elegant construction of the DP, which need not impose such regularity assumptions.

Posterior Distribution

Let $G \sim DP(\alpha, H)$. Since G is a (random) distribution, we can in turn draw samples from G itself. Let $\theta_1, \ldots, \theta_n$ be a sequence of independent draws from G. Note that the θ_i's take values in Θ since G is a distribution over Θ. We are interested in the posterior distribution of G given observed values of $\theta_1, \ldots, \theta_n$. Let $A_1, \ldots, A_r$ be a finite measurable partition of Θ, and let $n_k = \#\{i : \theta_i \in A_k\}$ be the number of observed values in A_k. By (3) and the conjugacy between the Dirichlet and the multinomial distributions, we have

$$\begin{aligned}&(G(A_1), \ldots, G(A_r))|\theta_1, \ldots, \theta_n \\ &\sim \mathrm{Dir}(\alpha H(A_1) + n_1, \ldots, \alpha H(A_r) + n_r)\end{aligned} \tag{3}$$

Since the above is true for all finite measurable partitions, the posterior distribution over G must be a DP as well. A little algebra shows that the posterior DP has updated concentration parameter $\alpha + n$ and base distribution $\frac{\alpha H + \sum_{i=1}^n \delta_{\theta_i}}{\alpha + n}$, where δ_i is a point mass located at θ_i and $n_k = \sum_{i=1}^n \delta_i(A_k)$. In other words, the DP provides a conjugate family of priors over distributions that is closed under posterior updates given observations. Rewriting the posterior DP, we have

$$G|\theta_1, \ldots, \theta_n \sim DP\left(\alpha + n, \frac{\alpha}{\alpha+n} H + \frac{n}{\alpha+n} \frac{\sum_{i=1}^n \delta_{\theta_i}}{n}\right) \tag{4}$$

Notice that the posterior base distribution is a weighted average between the prior base distribution H and the empirical distribution $\frac{\sum_{i=1}^n \delta_{\theta_i}}{n}$. The weight associated with the prior base distribution is proportional to α, while the empirical distribution has weight proportional to the number of observations n. Thus we can interpret α as the strength or mass associated with the prior. In the next section we will see that the posterior base distribution is also the predictive distribution of θ_{n+1} given $\theta_1, \ldots, \theta_n$. Taking $\alpha \to 0$, the prior becomes non-informative in the sense that the predictive distribution is just given by the empirical distribution. On the other hand, as the amount of observations grows large, $n \gg \alpha$, the posterior is simply dominated by the empirical distribution, which is in turn a close approximation of the true underlying distribution. This gives a consistency property of the DP: the posterior DP approaches the true underlying distribution.

Predictive Distribution and the Blackwell–MacQueen Urn Scheme

Consider again drawing $G \sim DP(\alpha, H)$, and drawing an i.i.d. (independently and identically distributed) sequence $\theta_1, \theta_2, \ldots \sim G$. Consider the predictive distribution for θ_{n+1}, conditioned on $\theta_1, \ldots, \theta_n$ and with G marginalized out. Since $\theta_{n+1}|G, \theta_1, \ldots, \theta_n \sim G$, for a measurable $A \subset \Theta$, we have

$$\begin{aligned}P(\theta_{n+1} \in A|\theta_1, \ldots, \theta_n) &= E[G(A)|\theta_1, \ldots, \theta_n] \\ &= \frac{1}{\alpha + n}\left(\alpha H(A) + \sum_{i=1}^n \delta_{\theta_i}(A)\right)\end{aligned} \tag{5}$$

where the last step follows from the posterior base distribution of G given the first n observations. Thus with G marginalized out:

$$\theta_{n+1}|\theta_1, \ldots, \theta_n \sim \frac{1}{\alpha + n}\left(\alpha H + \sum_{i=1}^n \delta_{\theta_i}\right) \tag{6}$$

Therefore the posterior base distribution given $\theta_1, \ldots, \theta_n$ is also the predictive distribution of θ_{n+1}.

The sequence of predictive distributions (6) for $\theta_1, \theta_2, \ldots$ is called the Blackwell–MacQueen urn scheme (Blackwell & MacQueen, 1973). The name stems from a metaphor useful in interpreting (6). Specifically, each value in Θ is a unique color, and draws $\theta \sim G$ are balls with the drawn value being the color of the ball. In addition we have an urn containing previously seen balls. In the beginning there are no balls in the urn, and we pick a color drawn from H, that is, draw $\theta_1 \sim H$, paint a ball with that color, and drop it into the urn. In subsequent steps, say the $n+1$st, we will either, with probability $\frac{\alpha}{\alpha+n}$, pick a new color (draw $\theta_{n+1} \sim H$), paint a ball with that color and drop the ball into the urn, or, with probability $\frac{n}{\alpha+n}$, reach into the urn to pick a random ball out (draw θ_{n+1} from the empirical distribution), paint a new ball with the same color, and drop both balls back into the urn.

The Blackwell–MacQueen urn scheme has been used to show the existence of the DP (Blackwell & MacQueen, 1973). Starting from (6), which are perfectly well defined conditional distributions regardless of the question of the existence of DPs, we can construct a distribution over sequences $\theta_1, \theta_2, \ldots$ by iteratively drawing each θ_i given $\theta_1, \ldots, \theta_{i-1}$. For $n \geq 1$ let

$$P(\theta_1, \ldots, \theta_n) = \prod_{i=1}^{n} P(\theta_i | \theta_1, \ldots, \theta_{i-1}) \qquad (7)$$

be the joint distribution over the first n observations, where the conditional distributions are given by (6). It is straightforward to verify that this random sequence is infinitely exchangeable. That is, for every n, the probability of generating $\theta_1, \ldots, \theta_n$ using (6), in that order, is equal to the probability of drawing them in any alternative order. More precisely, given any permutation σ on $1, \ldots, n$, we have

$$P(\theta_1, \ldots, \theta_n) = P(\theta_{\sigma(1)}, \ldots, \theta_{\sigma(n)}) \qquad (8)$$

Now de Finetti's theorem states that for any infinitely exchangeable sequence $\theta_1, \theta_2, \ldots$ there is a random distribution G such that the sequence is composed of i.i.d. draws from it:

$$P(\theta_1, \ldots, \theta_n) = \int \prod_{i=1}^{n} G(\theta_i) \, dP(G) \qquad (9)$$

In our setting, the prior over the random distribution $P(G)$ is precisely the Dirichlet process $DP(\alpha, H)$, thus establishing existence.

A salient property of the predictive distribution (6) is that it has point masses located at the previous draws $\theta_1, \ldots, \theta_n$. A first observation is that with positive probability draws from G will take on the same value, regardless of smoothness of H. This implies that the distribution G itself has point masses. A further observation is that for a long enough sequence of draws from G, the value of any draw will be repeated by another draw, implying that G is composed only of a weighted sum of point masses, that is, it is a discrete distribution. We will see two sections below that this is indeed the case, and give a simple construction for G called the stick-breaking construction. Before that, we shall investigate the clustering property of the DP.

Clustering, Partitions, and the Chinese Restaurant Process

In addition to the discreteness property of draws from a DP, (6) also implies a ▶clustering property. The discreteness and clustering properties of the DP play crucial roles in the use of DPs for clustering via DP mixture models, described in the application section. For now we assume that H is smooth, so that all repeated values are due to the discreteness property of the DP and not due to H itself. (Similar conclusions can be drawn when H has atoms, there is just more bookkeeping.) Since the values of draws are repeated, let $\theta_1^*, \ldots, \theta_m^*$ be the unique values among $\theta_1, \ldots, \theta_n$, and n_k be the number of repeats of θ_k^*. The predictive distribution can be equivalently written as

$$\theta_{n+1} \Big| \theta_1, \ldots, \theta_n \sim \frac{1}{\alpha+n} \left(\alpha H + \sum_{k=1}^{m} n_k \delta_{\theta_k^*} \right) \qquad (10)$$

Notice that value θ_k^* will be repeated by θ_{n+1} with probability proportional to n_k, the number of times it has already been observed. The larger n_k is, the higher the probability that it will grow. This is a rich-gets-richer phenomenon, where large clusters (a set of θ_i's with identical values θ_k^* being considered a cluster) grow larger faster.

We can delve further into the clustering property of the DP by looking at partitions induced by the clustering. The unique values of $\theta_1, \dots, \theta_n$ induce a partitioning of the set $[n] = \{1, \dots, n\}$ into clusters such that within each cluster, say cluster k, the θ_i's take on the same value θ_k^*. Given that $\theta_1, \dots, \theta_n$ are random, this induces a random partition of $[n]$. This random partition in fact encapsulates all the properties of the DP, and is a very well-studied mathematical object in its own right, predating even the DP itself (Aldous, 1985; Ewens, 1972; Pitman, 2002). To see how it encapsulates the DP, we simply invert the generative process. Starting from the distribution over random partitions, we can reconstruct the joint distribution (7) over $\theta_1, \dots, \theta_n$, by first drawing a random partition on $[n]$, then for each cluster k in the partition draw a $\theta_k^* \sim H$, and finally assign $\theta_i = \theta_k^*$ for each i in cluster k. From the joint distribution (7) we can obtain the DP by appealing to de Finetti's theorem.

The distribution over partitions is called the Chinese restaurant process (CRP) due to a different metaphor. (The name was coined by Lester Dubins and Jim Pitman in the early 1980s (Aldous, 1985)) In this metaphor we have a Chinese restaurant with an infinite number of tables, each of which can seat an infinite number of customers. The first customer enters the restaurant and sits at the first table. The second customer enters and decides either to sit with the first customer, or by herself at a new table. In general, the $n+1$st customer either joins an already occupied table k with probability proportional to the number n_k of customers already sitting there, or sits at a new table with probability proportional to α. Identifying customers with integers $1, 2, \dots$ and tables as clusters, after n customers have sat down the tables define a partition of $[n]$ with the distribution over partitions being the same as the one above. The fact that most Chinese restaurants have round tables is an important aspect of the CRP. This is because it does not just define a distribution over partitions of $[n]$, it also defines a distribution over permutations of $[n]$, with each table corresponding to a cycle of the permutation. We do not need to explore this aspect further and refer the interested reader to Aldous (1985) and Pitman (2002).

This distribution over partitions first appeared in population genetics, where it was found to be a robust distribution over alleles (clusters) among gametes (observations) under simplifying assumptions on the population, and is known under the name of Ewens sampling formula (Ewens, 1972). Before moving on we shall consider just one illuminating aspect, specifically the distribution of the number of clusters among n observations. Notice that for $i \geq 1$, the observation θ_i takes on a new value (thus incrementing m by one) with probability $\frac{\alpha}{\alpha+i-1}$ independently of the number of clusters among previous θ's. Thus the number of cluster m has mean and variance:

$$\begin{aligned} E[m|n] &= \sum_{i=1}^{n} \frac{\alpha}{\alpha+i-1} = \alpha(\psi(\alpha+n) - \psi(\alpha)) \\ &\simeq \alpha \log\left(1 + \frac{n}{\alpha}\right) \qquad \text{for } N, \alpha \gg 0, \qquad (11) \\ V[m|n] &= \alpha(\psi(\alpha+n) - \psi(\alpha)) \\ &\quad + \alpha^2(\psi'(\alpha+n) - \psi'(\alpha)) \\ &\simeq \alpha \log\left(1 + \frac{n}{\alpha}\right) \qquad \text{for } n > \alpha \gg 0, \qquad (12) \end{aligned}$$

where $\psi(\cdot)$ is the digamma function. Note that the number of clusters grows only logarithmically in the number of observations. This slow growth of the number of clusters makes sense because of the rich-gets-richer phenomenon: we expect there to be large clusters thus the number of clusters m has to be smaller than the number of observations n. Notice that α controls the number of clusters in a direct manner, with larger α implying a larger number of clusters a priori. This intuition will help in the application of DPs to mixture models.

Stick-Breaking Construction

We have already intuited that draws from a DP are composed of a weighted sum of point masses. Sethuraman (1994) made this precise by providing a constructive definition of the DP as such, called the stick-breaking construction. This construction is also significantly more straightforward and general than previous proofs of the existence of DPs. It is simply given as follows:

$$\begin{aligned} &\beta_k \sim Beta(1, \alpha) && \theta_k^* \sim H \\ &\pi_k = \beta_k \prod_{l=1}^{k-1}(1 - \beta_k) && G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*} \qquad (13) \end{aligned}$$

Then $G \sim DP(\alpha, H)$. The construction of π can be understood metaphorically as follows. Starting with a stick of length 1, we break it at β_1, assigning π_1 to be the length of stick we just broke off. Now recursively break the other portion to obtain π_2, π_3, and so forth. The stick-breaking distribution over π is sometimes written $\pi \sim GEM(\alpha)$, where the letters stand for Griffiths, Engen, and McCloskey (Pitman, 2002). Because of its simplicity, the stick-breaking construction has lead to a variety of extensions as well as novel inference techniques for the Dirichlet process (Ishwaran & James, 2001).

Applications

Because of its simplicity, DPs are used across a wide variety of applications of Bayesian analysis in both statistics and machine learning. The simplest and most prevalent applications include Bayesian model validation, density estimation, and clustering via mixture models. We shall briefly describe the first two classes before detailing DP mixture models.

How does one validate that a model gives a good fit to some observed data? The Bayesian approach would usually involve computing the marginal probability of the observed data under the model, and comparing this marginal probability to that for other models. If the marginal probability of the model of interest is highest we may conclude that we have a good fit. The choice of models to compare against is an issue in this approach, since it is desirable to compare against as large a class of models as possible. The Bayesian nonparametric approach gives an answer to this question: use the space of all possible distributions as our comparison class, with a prior over distributions. The DP is a popular choice for this prior, due to its simplicity, wide coverage of the class of all distributions, and recent advances in computationally efficient inference in DP models. The approach is usually to use the given parametric model as the base distribution of the DP, with the DP serving as a nonparametric relaxation around this parametric model. If the parametric model performs as well or better than the DP relaxed model, we have convincing evidence of the validity of the model.

Another application of DPs is in ▶density estimation (Escobar & West, 1995; Lo, 1984; Neal, 1992; Rasmussen, 2000). Here we are interested in modeling the density from which a given set of observations is drawn. To avoid limiting ourselves to any parametric class, we may again use a nonparametric prior over all densities. Here again DPs are a popular. However note that distributions drawn from a DP are discrete, thus do not have densities. The solution is to smooth out draws from the DP with a kernel. Let $G \sim DP(\alpha, H)$ and let $f(x|\theta)$ be a family of densities (kernels) indexed by θ. We use the following as our nonparametric density of x:

$$p(x) = \int f(x|\theta)G(\theta)\,d\theta \tag{14}$$

Similarly, smoothing out DPs in this way is also useful in the nonparametric relaxation setting above. As we see below, this way of smoothing out DPs is equivalent to DP mixture models, if the data distributions $F(\theta)$ below are smooth with densities given by $f(x|\theta)$.

Dirichlet Process Mixture Models

The most common application of the Dirichlet process is in clustering data using mixture models (Escobar & West, 1995; Lo, 1984; Neal, 1992; Rasmussen, 2000). Here the nonparametric nature of the Dirichlet process translates to mixture models with a countably infinite number of components. We model a set of observations $\{x_1, \ldots, x_n\}$ using a set of latent parameters $\{\theta_1, \ldots, \theta_n\}$. Each θ_i is drawn independently and identically from G, while each x_i has distribution $F(\theta_i)$ parametrized by θ_i:

$$\begin{aligned} x_i|\theta_i &\sim F(\theta_i) \\ \theta_i|G &\sim G \\ G|\alpha, H &\sim DP(\alpha, H) \end{aligned} \tag{15}$$

Because G is discrete, multiple θ_i's can take on the same value simultaneously, and the above model can be seen as a mixture model, where x_i's with the same value of θ_i belong to the same cluster. The mixture perspective can be made more in agreement with the usual representation of mixture models using the stick-breaking construction (13). Let z_i be a cluster assignment variable, which takes on value k with probability π_k. Then (15) can be equivalently expressed as

$$\begin{aligned} \pi|\alpha &\sim GEM(\alpha) & \theta_k^*|H &\sim H \\ z_i|\pi &\sim Mult(\pi) & x_i|z_i, \{\theta_k^*\} &\sim F\left(\theta_{z_i}^*\right) \end{aligned} \tag{16}$$

with $G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}$ and $\theta_i = \theta_{z_i}^*$. In mixture modeling terminology, π is the mixing proportion, θ_k^* are the cluster parameters, $F(\theta_k^*)$ is the distribution over data in cluster k, and H the prior over cluster parameters.

The DP mixture model is an *infinite* mixture model – a mixture model with a countably infinite number of clusters. However, because the π_k's decrease exponentially quickly, only a small number of clusters will be used to model the data a priori (in fact, as we saw previously, the expected number of components used a priori is logarithmic in the number of observations). This is different than a finite mixture model, which uses a fixed number of clusters to model the data. In the DP mixture model, the actual number of clusters used to model data is not fixed, and can be automatically inferred from data using the usual Bayesian posterior inference framework (see Neal (2000) for a survey of MCMC inference procedures for DP mixture models). The equivalent operation for finite mixture models would be model averaging or model selection for the appropriate number of components, an approach that is fraught with difficulties. Thus infinite mixture models as exemplified by DP mixture models provide a compelling alternative to the traditional finite mixture model paradigm.

Generalizations and Extensions

The DP is the canonical distribution over probability measures and a wide range of generalizations have been proposed in the literature. First and foremost is the *Pitman–Yor process* (Ishwaran & James, 2001; Pitman & Yor, 1997), which has recently seen successful applications modeling data exhibiting power-law properties (Goldwater, Griffiths, & Johnson, 2006; Teh, 2006). The Pitman–Yor process includes a third parameter $d \in [0,1)$, with $d = 0$ reducing to the DP. The various representations of the DP, including the Chinese restaurant process and the stick-breaking construction, have analogues for the Pitman–Yor process. Other generalizations of the DP are obtained by generalizing one of its representations. These include Pólya trees, normalized random measure, Poisson–Kingman models, species sampling models and stick-breaking priors.

The DP has also been used in more complex models involving more than one random probability measure. For example, in nonparametric regression we might have one probability measure for each value of a covariate, and in multitask settings each task might be associated with a probability measure with dependence across tasks implemented using a hierarchical Bayesian model. In the first situation, the class of models is typically called dependent Dirichlet processes (MacEachern, 1999), while in the second the appropriate model is a hierarchical Dirichlet process (Teh, Jordan, Beal, & Blei, 2006).

Future Directions

The Dirichlet process, and Bayesian nonparametrics in general, is an active area of research within both machine learning and statistics. Current research trends span a number of directions. Firstly, there is the issue of efficient inference in DP models. Reference Neal (2000) is an excellent survey of the state-of-the-art in 2000, with all algorithms based on Gibbs sampling or small-step Metropolis–Hastings MCMC sampling. Since then there has been much work, including split-and-merge and large-step auxiliary variable MCMC sampling, sequential Monte Carlo, expectation propagation, and variational methods. Secondly, there has been interest in extending the DP, both in terms of new random distributions, as well as novel classes of nonparametric objects inspired by the DP. Thirdly, theoretical issues of convergence and consistency are being explored to provide frequentist guarantees for Bayesian nonparametric models. Finally, there are applications of such models, to clustering, transfer learning, relational learning, models of cognition, sequence learning, and regression and classification among others. We believe DPs and Bayesian nonparametrics will prove to be rich and fertile grounds for research for years to come.

Cross References

- ►Bayesian Methods
- ►Bayesian Nonparametrics
- ►Clustering
- ►Density Estimation
- ►Gaussian Process
- ►Prior Probabilities

Further Reading

In addition to the references embedded in the text above, we recommend the book (Hjort, Holmes, Müller, & Walker, 2010) on Bayesian nonparametrics.

Recommended Reading

Aldous, D. (1985). Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII-1983* (pp. 1–198). Berlin: Springer.

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics, 2*(6), 1152–1174.

Blackwell, D., & MacQueen, J. B. (1973). Ferguson distributions via Pólya urn schemes. *Annals of Statistics, 1*, 353–355.

Escobar, M. D., & West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association, 90*, 577–588.

Ewens, W. J. (1972). The sampling theory of selectively neutral alleles. *Theoretical Population Biology, 3*, 87–112.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *Annals of Statistics, 1*(2), 209–230.

Goldwater, S., Griffiths, T. L., & Johnson, M. (2006). Interpolating between types and tokens by estimating power-law generators. In *Advances in neural information processing systems* (Vol. 18).

Hjort, N., Holmes, C., Müller, P., & Walker, S. (Eds.). (2010). *Bayesian nonparametrics. Cambridge series in statistical and probabilistic mathematics* (Vol. 28). Cambridge University Press.

Ishwaran, H., & James, L. F. (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association, 96*(453), 161–173.

Lo, A. Y. (1984). On a class of Bayesian nonparametric estimates: I. Density estimates. *Annals of Statistics, 12*(1), 351–357.

MacEachern, S. (1999). Dependent nonparametric processes. In *Proceedings of the section on Bayesian statistical science*. American Statistical Association. Alexandria, VA, USA.

Neal, R. M. (1992). Bayesian mixture modeling. In *Proceedings of the workshop on maximum entropy and Bayesian methods of statistical analysis* (Vol. 11, pp. 197–211). Neal (1992): Kluwer Academic Publishers, The Netherlands.

Neal, R. M. (2000). Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics, 9*, 249–265.

Pitman, J. (2002). *Combinatorial stochastic processes* (Tech. Rep. 621). Department of Statistics, University of California at Berkeley. Lecture notes for St. Flour Summer School.

Pitman, J., & Yor, M. (1997). The two-parameter Poisson–Dirichlet distribution derived from a stable subordinator. *Annals of Probability, 25*, 855–900.

Rasmussen, C. E. (2000). The infinite Gaussian mixture model. In *Advances in neural information processing systems* (Vol. 12).

Sethuraman, J. (1994). A constructive definition of Dirichlet priors. *Statistica Sinica, 4*, 639–650.

Teh, Y. W. (2006). A hierarchical Bayesian language model based on Pitman–Yor processes. In *Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the association for computational linguistics* (pp. 985–992).

Teh, Y. W., Jordan, M. I., Beal, M. J., & Blei, D. M. (2006). Hierarchical Dirichlet processes. *Journal of the American Statistical Association, 101*(476), 1566–1581.

Discrete Attribute

A **discrete attribute** assumes values that can be counted. The attribute cannot assume all values on the number line within its value range. See ►Attribute and ►Measurement Scales.

Discretization

Ying Yang
Australian Taxation Office, Australia

Synonyms

Binning

Definition

Discretization is a process that transforms a ►numeric attribute into a ►categorical attribute. Under discretization, a new categorical attribute X' is formed from and replaces an existing numeric attribute X. Each value x' of X' corresponds to an interval $(a,b]$ of X. Any original numeric value x of X that belongs to $(a,b]$ is replaced by x'. The boundary values of formed intervals are often called "cut points."

Motivation and Background

Many learning systems require categorical data, while many data are numeric. Discretization allows numeric data to be transformed into categorical form suited to processing by such systems. Further, in some cases effective discretization can improve either computational or prediction performance relative to learning from the original numeric data.

Taxonomy

The following taxonomy identifies many key dimensions along which alternative discretization techniques can be distinguished.

►Supervised vs. **►Unsupervised** (Dougherty, Kohavi, & Sahami, 1995). Supervised methods use the class information of the training instances to select discretization cut points. Methods that do not use the class information are unsupervised.

Global vs. **Local** (Dougherty et al., 1995). Global methods discretize with respect to the whole training data space. They perform discretization only once, using a single set of intervals throughout a single classification task. Local methods allow different sets of intervals to be formed for a single attribute, each set being applied in a different classification context. For example, different discretizations of a single attribute might be applied at different nodes of a decision tree (Quinlan, 1993).

Eager vs. **Lazy** (Hsu, Huang, & Wong, 2000). Eager methods perform discretization prior to classification time. Lazy methods perform discretization during the process of classification.

Disjoint vs. **Nondisjoint** (Yang & Webb, 2002). Disjoint methods discretize the value range of a numeric attribute into disjoint intervals. No intervals overlap. Nondisjoint methods discretize the value range into intervals that can overlap.

Parameterized vs. **Unparameterized**. Parameterized discretization requires input from the user, such as the maximum number of discretized intervals. Unparameterized discretization uses information only from data and does not need input from the user, for instance, the entropy minimization discretization (Fayyad & Irani, 1993).

Univariate vs. **Multivariate** (Bay, 2000). Methods that discretize each attribute in isolation are univariate. Methods that take into consideration relationships among attributes during discretization are multivariate.

Split vs. **Merge** (Kerber, 1992) vs. **Single-scan** (Yang & Webb, 2001). Split discretization initially has the whole value range as an interval and then continues splitting it into subintervals until some threshold is met. Merge discretization initially puts each value into an interval and then continues merging adjacent intervals until some threshold is met. Single-scan discretization uses neither split nor merge process. Instead, it scans the ordered values only once, sequentially forming the intervals.

Recommended Reading

Bay, S. D. (2000). Multivariate discretization of continuous variables for set mining. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 315–319).

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proceedings of the twelfth international conference on machine learning* (pp. 194–202).

Fayyad, U. M. & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the thirteenth international joint conference on artificial intelligence* (pp. 1022–1027).

Hsu, C. N., Huang, H. J., & Wong, T. T. (2000). Why discretization works for naïve Bayesian classifiers. In *Proceedings of the seventeenth international conference on machine learning* (pp. 309–406).

Kerber, R. (1992). ChiMerge: Discretization for numeric attributes. In *AAAI national conference on artificial intelligence* (pp. 123–128).

Kononenko, I. (1992). Naive Bayesian classifier and continuous Attributes. *Informatica*, *16*(1), 1–8.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco: Morgan Kaufmann Publishers.

Yang, Y., & Webb, G. (2001). Proportional k-interval discretization for naive-Bayes classifiers. In *Proceedings of the twelfth european conference on machine learning* (pp. 564–575).

Yang, Y. & Webb, G. (2002). Non-disjoint discretization for naive-Bayes classifiers. In *Proceedings of the nineteenth international conference on machine learning* (pp. 666–673).

Discriminative Learning

Definition

Discriminative learning refers to any ►classification learning process that classifies by using a ►model or estimate of the probability $P(y \mid \mathbf{x})$ without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x} \mid y)$, where y is a class and $\mathbf{x}$ is a description of an object to be classified. Discriminative learning contrasts to ►generative learning which classifies by using an estimate of the joint probability $P(y, \mathbf{x})$ or of the prior probability P(y) and the conditional probability $P(\mathbf{x} \mid y)$.

It is also common to categorize as discriminative any approaches that are directly based on a decision risk function (such as ►Support Vector Machines, ►Artificial Neural Networks, and ►Decision Trees), where the decision risk is minimized without estimation of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x} \mid y)$.

Cross References

►Generative and Discriminative Learning

Disjunctive Normal Form

Bernhard Pfahringer
University of Waikato, Hamilton, New Zealand

Disjunctive normal form is an important normal form for propositional logic. A logic formula is in disjunctive normal form if it is a single disjunction of conjunctions of (possibly negated) literals. No more nesting and no other negations are allowed. Examples are:

$$a$$
$$\neg b$$
$$a \vee b$$
$$(a \wedge \neg b) \vee (c \wedge d)$$
$$\neg a \vee (b \wedge \neg c \wedge d) \vee (a \wedge \neg d)$$

Any arbitrary formula in propositional logic can be transformed into disjunctive normal form by application of the laws of distribution, De Morgan's laws, and by removing double negations. It is important to note that this process can lead to exponentially larger formulas which implies that the process in the worst case runs in exponential time. An example for this behavior is the following formula given in ►conjunctive normal form (CNF), which is linear in the number of propositional variables in this form. When transformed into disjunctive normal form (DNF), its size is exponentially larger.

$$\text{CNF: } (a_0 \vee a_1) \wedge (a_2 \vee a_3) \wedge \cdots \wedge (a_{2n} \vee a_{2n+1})$$

$$\text{DNF: } (a_0 \wedge a_2 \wedge \cdots \wedge a_{2n}) \vee (a_1 \wedge a_2 \wedge \cdots \wedge a_{2n}) \vee \cdots \vee (a_1 \wedge a_3 \wedge \cdots \wedge a_{2n+1})$$

Recommended Reading

Mendelson, E. (1997). *Introduction to mathematical logic* (4th ed.) (p.30). Chapma & Hall.

Distance

►Similarity Measures

Distance Functions

►Similarity Measures

Distance Measures

►Similarity Measures

Distance Metrics

►Similarity Measures

Distribution-Free Learning

►PAC Learning

Divide-and-Conquer Learning

Synonyms

Recursive partitioning; TDIDT strategy

Definition

The *divide-and-conquer* strategy is a learning algorithm for inducing ►Decision Trees. Its name reflects its key idea, which is to successively partition the dataset into smaller sets (the *divide* part), and recursively call itself on each subset (the *conquer* part). It should not be confused with the *separate-and-conquer* strategy which is used in the ►Covering Algorithm for rule learning.

Cross References

►Covering Algorithm
►Decision Tree

Document Classification

Dunja Mladeni, Janez Brank, Marko Grobelnik
Jožef Stefan Institute, Ljubljana, Slovenia

Synonyms

Document categorization; Supervised learning on text data

Definition

Document classification refers to a process of assigning one or more ►labels for a document from a predefined set of labels. The main issues in document classification are connected to classification of free text giving document content. For instance, classifying Web documents as being about arts, education, science, etc. or classifying news articles by their topic. In general, one can consider different properties of a document in document classification and combine them, such as document type, authors, links to other documents, content, etc. Machine learning methods applied to document classification are based on general classification methods adjusted to handle some specifics of text data.

Motivation and Background

Documents and text data provide for valuable sources of information and their growing availability in electronic form naturally led to application of different analytic methods. One of the common ways is to take a whole vocabulary of the natural language in which the text is written as a feature set, resulting in several tens of thousands of features. In a simple setting, each feature gives a count of the word occurrences in a document. In this way, text of a document is represented as a vector of numbers. The representation of a particular document contains many zeros, as most of the words from the vocabulary do not occur in a particular document. In addition to the already mentioned two common specifics of text data, having a large number of features and a sparse data representation, it was observed that frequency of words in text generally follows Zipf's law – a small subset of words occur very frequently in texts while a large number of words occur only rarely. Document classification takes these and some other data specifics into account when developing the appropriate classification methods.

Structure of Learning System

Document classification is usually performed by representing documents as word-vectors, usually referred to as the "bag-of-words" or "vector space model" representation, and using documents that have been manually classified to generate a model for document classification (Cohen & Singer, 1996, Mladenić & Grobelnik, 2003; Sebastiani, 2002; Yang, 1997).

Data Representation

In the word-vector representation of a document, a vector of word weights is formed taking all the words occurring in all the documents. Most researchers have used single words when representing text, but there is also research that proposes using additional information to improve classification results. For instance, the feature set might be extended with various multi-word features, e.g., *n*-grams (sequences of *n* adjacent words), loose phrases (*n*-grams in which word order is ignored), or phrases based on grammatical analysis (noun phrases, verb phrases, etc.). Information external to the documents might also be used if it is available; for example, when dealing with Web pages, their graph organization can be a source of additional features (e.g., features corresponding to the adjacency matrix; features based on graph vertex statistics such as degree or PageRank; or features taken from the documents that are adjacent to the current document in the Web graph).

The commonly used approach to weighting words is based on TF–IDF weights where the number of occurrences of the word in the document, referred to as term frequency (TF), is multiplied by the importance of the word with regards to the whole corpus (IDF – inverse document frequency). The IDF weight for the *i*th word is defined as $\mathrm{IDF}_i = \log(N/\mathrm{DF}_i)$, where N is total number of documents and DF_i is the document frequency of the *i*th word (the number of documents from the whole corpus in which the *i*th word appears). The IDF weight decreases the influence of common words (which are not as likely to be useful for discriminating between classes of documents) and favors the less common words. However, the least frequently occurring words are often deleted from the documents as a preprocessing step, based on the notion that if a word that does not occur often enough in the training set cannot be useful for learning and generalization, and would effectively be perceived as noise by the learning algorithm. A stopword list is also often used to delete some of the most common and low-content words (such as "the," "of," "in," etc.) during preprocessing. For many purposes, the vectors used to represent documents should be normalized to unit length so that the vector reflects the contents and themes of the document but not its length (which is typically not relevant for the purposes of document categorization).

Even in a corpus of just a few thousand documents, this approach to document representation can easily lead to a feature space of thousands, possibly tens of thousands, of features. Therefore, feature selection is sometimes used to reduce the feature set before training. Such questions as whether feature selection is needed and/or beneficial, and which feature selection method should be used, depend considerably on the learning algorithm used; the number of features to be retained depends both on the learning algorithm and on the feature selection method used. For example, naive Bayes tends to benefit, indeed require, heavy feature selection while support vector machines (SVMs) tend to benefit little or nothing from it. Similarly, odds ratio tends to value (some) rare features highly and therefore requires a lot of features to be kept, while information gain tends to score some of the more frequent features highly and thus often works better if a smaller number of features is kept (see also ▶Feature Selection in Text Mining).

Due to the large number of features in the original data representation, some of the more computationally expensive feature selection methods from traditional machine learning cannot be used with textual data. Typically, simple feature scoring measures, such as information gain, odds ratio, and chi-squared are used to rank the features and the features whose score falls below a certain threshold are discarded. A better, but computationally more expensive feature scoring method is to train a linear classifier on the full feature set first (e.g., using linear SVM, see below) and rank the features by the absolute value of their weights in the resulting linear model (see also ▶Feature Construction in Text Mining).

Classification

Different classification algorithms have been adjusted and applied on text data. A few more popular are described here.

▶Naive Bayes based on the multinomial model, where the predicted class for document d is the one that maximizes the posterior probability $P(c \mid d) \propto P(c)\Pi_t P(t \mid c)$ TF(t,d), where $P(c)$ is the prior probability that a document belongs to class c, $P(t \mid c)$ is the probability that a word chosen randomly in a document from class c equals t, and TF(t, d) is the "term frequency," or the number of occurrences of word t in a document d. Where there are only two classes, say c_+ and c_-, maximizing $P(c \mid d)$ is equivalent to taking the sign of ln $P(c_+ \mid d)/P(c_t \mid d)$, which is a linear combination of TF(w, d). Thus, the naive Bayes classifier can be seen as a linear classifier as well. The training consists simply of estimating the probabilities $P(t \mid c)$ and $P(c)$ from the training documents.

▶Perceptron trains a linear classifier in an incremental way as a neural unit using an additive update rule. The prediction for a document represented by the vector $\mathbf{x}$ is $\text{sgn}(\mathbf{w}^T\mathbf{x})$, where $\mathbf{w}$ is a vector of weights obtained during training. Computation starts with $\mathbf{w} = 0$, then considers each training example $\mathbf{x}_i$ in turn. If the present $\mathbf{w}$ classifies document $\mathbf{x}_i$ correctly it is left unchanged, otherwise it is updated according to the additive rule: $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$, where y_i is the correct class label of the document $\mathbf{x}_i$, namely $y_i = +1$ for a positive document, $y_i = 1$ for a negative one.

▶*SVM* trains a linear classifier of the form sgn $(\mathbf{w}^T\mathbf{x} + b)$. Learning is posed as an optimization problem with the goal of maximizing the *margin*, i.e., the distance between the separating hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ and the nearest training vectors. An extension of this formulation, known as the *soft margin*, also allows for a wider margin at the cost of misclassifying some of the training examples. The dual form of this optimization task is a quadratic programing problem and can be solved numerically.

Results of numerous experiments reported in research papers suggest that among the classification algorithms that have been adjusted to text data SVM, Naive Bayes and k-Nearest Neighbor are among the best performing (Lewis, Schapire, Callan, & Ron Papka, 1996). Moreover, experimental evaluation on some standard Reuters news datasets shows that SVM tends to outperform other classifiers including Naive Bayes and Perceptron (Mladenic, Brank, Grobelnik, & Milic-Frayling, 2004).

Evaluation Measures

A characteristic property of machine learning problems arising in document classification is a very unbalanced class distribution. In a typical dataset there may be tens (or sometimes hundreds or thousands) of categories, most of which are very small. When we train a binary (two-class) classification model for a particular category, documents belonging to that category are treated as the positive class while all other documents are treated as the negative class. Thus, the negative class is typically vastly larger as the positive one. These circumstances are not well suited to some traditional machine learning evaluation measures, such as ►accuracy (if almost all documents are negative, then a useless classifier that always predicts the negative class will have very high accuracy). Instead, evaluation measures from information retrieval are more commonly used, such as ►precision, ►recall, the F_1-measure, the breakeven point (BEP), and the area under the receiver operating characteristic (ROC) curve (see also ►ROC Analysis).

The evaluation of a binary classifier for a given category c on a given test set can be conveniently summarized in a contingency table. We can divide documents into four groups depending on whether they belong to c and whether our classifier predicted them as positive (i.e., supposedly belonging to c) or not:

	Belongs to c	Not in c
Predicted negative	TP (true positives)	FP (false positives)
Predicted negative	FN (false negatives)	TN (true negatives)

Given the number of documents in each of the four groups (TP, FP, TN, and FN), we can compute various evaluation measures as follows:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP})$$
$$\text{Recall} = \text{TP}_{\text{rate}} = \text{TP}/(\text{TP} + \text{FN})$$
$$\text{FP}_{\text{rate}} = \text{FP}/(\text{TN} + \text{FP})$$
$$F_1 = 2 \bullet \mathit{precision} \bullet \mathit{recall}/(\mathit{precision} + \mathit{recall})$$

Thus, precision is the proportion of documents predicted positive that are really positive, while recall is the proportion of positive documents that have been correctly predicted as positive. The F_1 is the harmonic mean of precision and recall; thus, it lies between precision and recall, but is closer to the lower of these two values. This means that a classifier with high F_1 has both good precision and good recall. In practice, there is usually a tradeoff between precision and recall; by making the classifier more liberal (i.e., more likely to predict positive), we can increase recall at the expense of precision, while by making it more conservative (less likely to predict positive) we can usually increase precision at the expense of recall. Often the classification model involves a threshold which can be varied at will to obtain various $\langle$*precision*, *recall*$\rangle$ pairs. These can be plotted on a chart, resulting in the *precision–recall curve*. As we decrease the threshold (thus making the classifier more liberal), precision decreases and recall increases until at some point precision and recall are equal; this value is known as the (*precision–recall*) *BEP* (Lewis, 1991). Instead of $\langle$*precision*, *recall*$\rangle$ pairs, one can measure $\langle \text{TP}_{\text{rate}}, \text{FP}_{\text{rate}} \rangle$ pairs, resulting in a *ROC curve* (see ROC analysis). The *area under the ROC curve* is another valuable measure of the classifier quality.

Document classification problems are typically multi-class, multi-label problems, which are treated by regarding each category as a separate two-class classification problem. After training a two-class classifier for each category and evaluating it, the question arises how to combine these evaluation measures into an overall evaluation measure. One way is *macroaveraging*, which means that the values of precision, recall, F_1, or whatever other measure we are interested in are simply averaged over all the categories. Since small categories tend to be much more numerous than large ones, macroaveraging tends to emphasize the performance of our learning algorithm on small categories. An alternative approach is *microaveraging*, in which the contingency tables for individual two-class classifiers are summed up and measures such as precision, recall, and F_1 computed from the resulting aggregated table. This approach emphasizes the performance of our learning algorithm on larger categories.

Cross References

►Classification
►Document Clustering
►Feature Selection
►Perceptron
►Semi-Supervised Text Processing
►Support Vector Machine
►Text Visualization

Recommended Reading

Cohen, W. W., & Singer, Y. (1996). Context sensitive learning methods for text categorization. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 307–315). Zurich: ACM.

Lewis, D. D. (1991). *Representation and learning in information retrieval*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA.

Lewis, D. D., Schapire, R. E., Callan, J. P., & Ron Papka, R. (1996) Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on research and development in information retrieval SIGIR-1996* (pp. 298–306). New York: ACM.
Mladenic, D., Brank, J., Grobelnik, M., & Milic-Frayling, N. (2004). Feature selection using linear classifier weights: Interaction with classification models. In *Proceedings of the twenty-seventh annual international ACM SIGIR conference on research and development in information retrieval SIGIR-2004* (pp. 234–241). New York: ACM.
Mladenić, D., & Grobelnik, M. (2003). Feature selection on hierarchy of web documents. *Journal of Decision Support Systems, 35*, 45–87.
Sebastiani, F. (2002). Machine learning for automated text categorization. *ACM Computing Surveys, 34*(1), 1–47.
Yang, Y. (1997). An evaluation of statistical approaches to text categorization. *Journal of Information Retrieval, 1*, 67–88.

Document Clustering

Ying Zhao[1], George Karypis[2]
[1]Tsinghua University, Beijing, China
[2]University of Minnesota, Minneapolis, USA

Synonyms

High-dimensional clustering; Text clustering; Unsupervised learning on document datasets

Definition

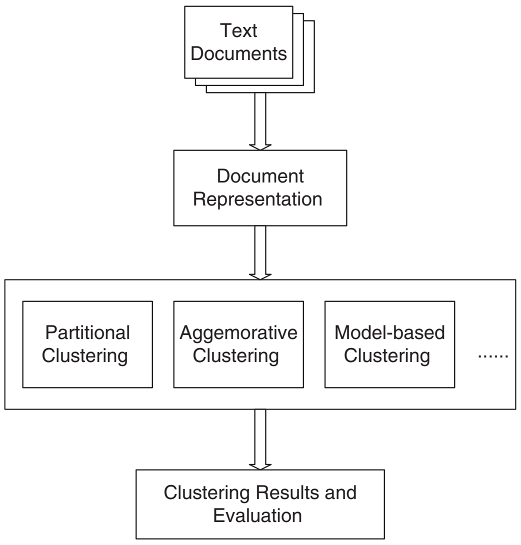
At a high-level, the problem of document clustering is defined as follows. Given a set S of n documents, we would like to partition them into a predetermined number of k subsets $S_1, S_2, \ldots, S_k$, such that the documents assigned to each subset are more similar to each other than the documents assigned to different subsets. Document clustering is an essential part of text mining and has many applications in information retrieval and knowledge management. Document clustering faces two big challenges: the dimensionality of the feature space tends to be high (i.e., a document collection often consists of thousands or tens of thousands unique words) and the size of a document collection tends to be large.

Motivation and Background

►Clustering is an essential component of data mining and a fundamental means of knowledge discovery in data exploration. Fast and high-quality document clustering algorithms play an important role in providing intuitive navigation and browsing mechanisms as well as in facilitating knowledge management. In recent years, we have witnessed a tremendous growth in the volume of text documents available on the Internet, digital libraries, news sources, and company-wide intranets. This has led to an increased interest in developing methods that can help users effectively navigate, summarize, and organize this information with the ultimate goal of helping them find what they are looking for. Fast and high-quality document clustering algorithms play an important role toward this goal as they have been shown to provide both an intuitive navigation/browsing mechanism by organizing large amounts of information into a small number of meaningful clusters as well as to greatly improve the retrieval performance either via cluster-driven dimensionality reduction, term-weighting, or query expansion.

Structure of Learning System

Figure 1 shows the three procedures of transferring a document collection to clustering results that are valuable to users. Original documents are often plain text files, html files, xml files, or a mixture of them. However, most clustering algorithms cannot operate



Document Clustering. Figure 1. Structure of document clustering learning system

on such textual files directly. Hence, *document representation* is needed to prepare original documents into the data model on which clustering algorithms can operate. The actual clustering process can choose clustering algorithms of various kinds: partitional clustering, agglomerative clustering, model-based clustering, etc., depending on the characteristics of the dataset and requirements of the application. We focus on two kinds of clustering algorithms that have been widely used in document clustering: partitional clustering and agglomerative clustering. Finally, clustering results need to be presented with proper quality evaluation to users.

Structure of Document Clustering

In the section, we describe document representation, partitional document clustering, agglomerative document clustering, and clustering evaluation in details.

Document Representation

We introduce here the most widely used document model for clustering and information retrieval: *term frequency-inverse document frequency* (tf-idf) vector-space model (Salton, 1989). In this model, each document d is considered to be a vector in the term-space and is represented by the vector

$$d_{\mathrm{tfidf}} = (tf_1 \log(n/\mathrm{df}_1), tf_2 \log(n/\mathrm{df}_2), \ldots, \\ \times\, tf_m \log(n/\mathrm{df}_m)),$$

where tf_i is the frequency of the ith term (i.e., term frequency), n is the total number of documents, and df_i is the number of documents that contain the ith term (i.e., document frequency). To account for documents of different lengths, the length of each document vector is normalized so that it is of unit length.

Similarity Measures

We need to define similarity between two documents under tf-idf model, which is essential to a clustering algorithm. Two prominent ways have been proposed to compute the similarity between two documents d_i and d_j. The first method is based on the commonly-used (Salton, 1989) cosine function

$$\cos(d_i, d_j) = d_i^t d_j / (\|d_i\| \, \|d_j\|),$$

and since the document vectors are of unit length, it simplifies to $d_i^t d_j$. The second method computes the similarity between the documents using the Euclidean distance $\mathrm{dis}(d_i, d_j) = \|d_i - d_j\|$. Note that besides the fact that one measures similarity and the other measures distance, these measures are quite similar to each other because the document vectors are of unit length.

Partitional Document Clustering

Partitional algorithms, such as K-means (MacQueen, 1967), K-medoids (Jain & Dubes, 1988), probabilistic (Dempster, Laird, & Rubin, 1977), graph-partitioning-based (Zahn, 1971), or spectral-based (Boley, 1998), find the clusters by partitioning the entire dataset into either a predetermined or an automatically derived number of clusters. A key characteristic of many partitional clustering algorithms is that they use a global criterion function whose optimization drives the entire clustering process. For some of these algorithms the criterion function is implicit (e.g., PDDP, Boley, 1998), whereas for other algorithms (e.g., K-means, MacQueen, 1967) the criterion function is explicit and can be easily stated. This latter class of algorithms can be thought of as consisting of two key components. First is the criterion function that the clustering solution optimizes, and second is the actual algorithm that achieves this optimization.

Criterion Function Criterion functions used in the partitional clustering reflect the underlying definition of the "goodness" of clusters. The partitional clustering can be considered as an optimization procedure that tries to create high quality clusters according to a particular criterion function. Many criterion functions have been proposed and analyzed (Duda, Hart, & Stork, 2001; Jain & Dubes, 1988; Zhao & Karypis, 2004). We list in Table 1 a total of seven different clustering criterion functions. These functions optimize various aspects of intra-cluster similarity, inter-cluster dissimilarity, and their combinations, and represent some of the most widely used criterion functions for document clustering. These criterion functions utilize different views of the underlying collection, by modeling either the objects as vectors in a high-dimensional space, or the collection as a graph.

The $\mathcal{I}_1$ criterion function (1) maximizes the sum of the average pairwise similarities (as measured by the cosine function) between the documents assigned to each cluster weighted according to the size of each

Document Clustering. Table 1 The mathematical definition of various clustering criterion functions

Criterion function	Optimization function
$\mathcal{I}_1$	maximize $\sum_{i=1}^{k} \frac{1}{n_i}\left(\sum_{v,u\in S_i} \text{sim}(v,u)\right)$ (1)
$\mathcal{I}_2$	maximize $\sum_{i=1}^{k} \sqrt{\sum_{v,u\in S_i} \text{sim}(v,u)}$ (2)
$\mathcal{E}_1$	minimize $\sum_{i=1}^{k} n_i \frac{\sum_{v\in S_i,u\in S} \text{sim}(v,u)}{\sqrt{\sum_{v,u\in S_i} \text{sim}(v,u)}}$ (3)
$\mathcal{G}_1$	minimize $\sum_{i=1}^{k} \frac{\sum_{v\in S_i,u\in S} \text{sim}(v,u)}{\sum_{v,u\in S_i} \text{sim}(v,u)}$ (4)
$\mathcal{G}_2$	minimize $\sum_{r=1}^{k} \frac{\text{cut}(V_r, V-V_r)}{W(V_r)}$ (5)
$\mathcal{H}_1$	maximize $\frac{\mathcal{I}_1}{\mathcal{E}_1}$ (6)
$\mathcal{H}_2$	maximize $\frac{\mathcal{I}_2}{\mathcal{E}_1}$ (7)

The notation in these equations are as follows: k is the total number of clusters, S is the total objects to be clustered, S_i is the set of objects assigned to the ith cluster, n_i is the number of objects in the ith cluster, v and u represent two objects, and sim(v,u) is the similarity between two objects

cluster. The $\mathcal{I}_2$ criterion function (2) is used by the popular vector-space variant of the K-means algorithm (Cutting, Pedersen, Karger, & Tukey, 1992). In this algorithm each cluster is represented by its centroid vector and the goal is to find the solution that maximizes the similarity between each document and the centroid of the cluster that is assigned to. Comparing $\mathcal{I}_1$ and $\mathcal{I}_2$, we see that the essential difference between them is that $\mathcal{I}_2$ scales the within-cluster similarity by the $\|D_r\|$ term as opposed to the n_r term used by $\mathcal{I}_1$. $\|D_r\|$ is the square-root of the pairwise similarity between all the document in S_r and will tend to emphasize clusters whose documents have smaller pairwise similarities compared to clusters with higher pairwise similarities.

The $\mathcal{E}_1$ criterion function (3) computes the clustering by finding a solution that separates the documents of each cluster from the entire collection. Specifically, it tries to minimize the cosine between the centroid vector of each cluster and the centroid vector of the entire collection. The contribution of each cluster is weighted proportionally to its size so that larger clusters will be weighted higher in the overall clustering solution. $\mathcal{E}_1$ was motivated by multiple discriminant analysis and is similar to minimizing the trace of the between-cluster scatter matrix (Duda et al., 2001).

The $\mathcal{H}_1$ and $\mathcal{H}_2$ criterion functions (6) and (7) are obtained by combining criterion $\mathcal{I}_1$ with $\mathcal{E}_1$, and $\mathcal{I}_2$ with $\mathcal{E}_1$, respectively. Since $\mathcal{E}_1$ is minimized, both $\mathcal{H}_1$ and $\mathcal{H}_2$ need to be maximized as they are inversely related to $\mathcal{E}_1$.

The criterion functions that we described so far view each document as a multidimensional vector. An alternate way of modeling the relations between documents is to use graphs. Two types of graphs are commonly used in the context of clustering. The first corresponds to the document-to-document similarity graph G_s and the second to the document-to-term bipartite graph G_b (Dhillon, 2001; Zha, He, Ding, Simon, & Gu, 2001). G_s is obtained by treating the pairwise similarity matrix of the dataset as the adjacency matrix of G_s, whereas G_b is obtained by viewing the documents and the terms as the two sets of vertices (V_d and V_t) of a bipartite graph. In this bipartite graph, if the ith document contains the jth term, then there is an edge connecting the corresponding ith vertex of V_d to the jth vertex of V_t. The weights of these edges are set using the *tf-idf* model.

Viewing the documents in this fashion, edge-cut-based criterion functions can be used to cluster document datasets. $\mathcal{G}_1$ and $\mathcal{G}_2$ ((4) and (5)) are two such criterion functions that are defined on the similarity and bipartite graphs, respectively. The $\mathcal{G}_1$ function (Ding, He, Zha, Gu, & Simon, 2001) views the clustering process as that of partitioning the documents into groups that minimize the edge-cut of each partition. However, because this edge-cut-based criterion function may have trivial solutions the edge-cut of each cluster is scaled by the sum of the cluster's internal edges (Ding et al., 2001). Note that, cut$(S_r, S-S_r)$ in (4) is the edge-cut between the vertices in S_r and the rest of the vertices $S-S_r$ and can be re-written as $D_r^t(D-D_r)$ because the similarity between documents is measured using the cosine function. The $\mathcal{G}_2$ criterion function (Dhillon, 2001; Zha et al., 2001) views the clustering problem as a simultaneous partitioning of the documents and the terms so that it minimizes the normalized edge-cut of the partitioning. Note that, V_r is the set of vertices assigned to the rth cluster and $W(V_r)$ is the sum of the weights of the adjacency lists of the vertices assigned to the rth cluster.

Optimization Method There are many techniques that can be used to optimize the criterion functions described above. They include relatively simple greedy schemes, iterative schemes with varying degree of hill-climbing capabilities, and powerful but computationally expensive spectral-based optimizers (Boley, 1998; Dhillon, 2001; Fisher, 1996; MacQueen, 1967; Zha et al., 2001). We introduce here a simple yet very powerful greedy strategy that has been shown to produce comparable results to those produced by more sophisticated optimization algorithms. In this greedy straggly, a k-way clustering of a set of documents can be computed either directly or via a sequence of repeated bisections. A direct k-way clustering is computed as follows. Initially, a set of k objects is selected from the datasets to act as the *seeds* of the k clusters. Then, for each object, its similarity to these k seeds is computed, and it is assigned to the cluster corresponding to its most similar seed. This forms the initial k-way clustering. This clustering is then repeatedly refined so that it optimizes a desired clustering criterion function. A k-way partitioning via repeated bisections is obtained by recursively applying the above algorithm to compute 2-way clustering (i.e., bisections). Initially, the objects are partitioned into two clusters, then one of these clusters is selected and is further bisected, and so on. This process continues $k - 1$ times, leading to k clusters. Each of these bisections is performed so that the resulting two-way clustering solution optimizes a particular criterion function.

Agglomerative Document Clustering

Hierarchical agglomerative algorithms find the clusters by initially assigning each object to its own cluster and then repeatedly merging pairs of clusters until a certain stopping criterion is met. Consider an n-object dataset and the clustering solution that has been computed after performing l merging steps. This solution will contain exactly $n - l$ clusters as each merging step reduces the number of clusters by one. Now, given this $(n - l)$-way clustering solution, the pair of clusters that is selected to be merged next is the one that leads to an $(n - l - 1)$-way solution that optimizes a particular criterion function. That is, each one of the $(n - l) \times (n - l - 1)/2$ pairs of possible merges is evaluated, and the one that leads to a clustering solution that has the maximum (or minimum) value of the particular criterion function is selected. Thus, the criterion function is *locally* optimized within each particular stage of agglomerative algorithms. Depending on the desired solution, this process continues until either there are only k clusters left or when the entire agglomerative tree has been obtained.

The three basic criteria to determine which pair of clusters to be merged next are single-link (Sneath & Sokal, 1973), complete-link (King, 1967), and group average (i.e., unweighed pair group method with arithmetic mean, UPGMA) (Jain & Dubes, 1988). The single-link criterion function measures the similarity of two clusters by the maximum similarity between any pair of objects from each cluster, whereas the complete-link uses the minimum similarity. In general, both the single- and the complete-link approaches do not work very well because they either base their decisions to a limited amount of information (single-link), or assume that all the objects in the cluster are very similar to each other (complete-link). On the other hand, the group average approach measures the similarity of two clusters by the average of the pairwise similarity of the objects from each cluster and does not suffer from the problems arising with single- and complete-link.

Evaluation of Document Clustering

Clustering results are hard to be evaluated, especially for high dimensional data and without a priori knowledge of the objects' distribution, which is quite common in practical cases. However, assessing the quality of the resulting clusters is as important as generating the clusters. Given the same dataset, different clustering algorithms with various parameters or initial conditions will give very different clusters. It is essential to know whether the resulting clusters are valid and how to compare the quality of the clustering results, so that the right clustering algorithm can be chosen and the best clustering results can be used for further analysis.

In general, there are two types of metrics for assessing clustering results: metrics that only utilize the information provided to the clustering algorithms (i.e., *internal metrics*) and metrics that utilize a priori knowledge of the classification information of the dataset (i.e., *external metrics*).

The basic idea behind internal quality measures is rooted from the definition of clusters. A meaningful clustering solution should group objects into various clusters, so that the objects within each cluster are more similar to each other than the objects from different clusters. Therefore, most of the internal quality measures evaluate the clustering solution by looking at how similar the objects are within each cluster and how well the objects of different clusters are separated. In particular, the *internal similarity* measure, *ISim*, is defined as the average similarity between the objects of each cluster, and the *external similarity* measure, *ESim*, is defined as the average similarity of the objects of each cluster and the rest of the objects in the data set. The ratio between the internal and external similarity measure is also a good indicator of the quality of the resultant clusters. The higher the ratio values, the better the clustering solution is. One of the limitations of the internal quality measures is that they often use the same information both in discovering and in evaluating the clusters.

The approaches based on external quality measures require a priori knowledge of the natural clusters that exist in the dataset, and validate a clustering result by measuring the agreement between the discovered clusters and the known information. For instance, when clustering document datasets, the known categorization of the documents can be treated as the natural clusters, and the resulting clustering solution will be considered correct if it leads to clusters that preserve this categorization. A key aspect of the external quality measures is that they utilize information other than that used by the clustering algorithms. The *entropy* measure is one such metric that looks how the various classes of documents are distributed within each cluster.

Given a particular cluster, S_r, of size n_r, the entropy of this cluster is defined to be

$$E(S_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\frac{n_r^i}{n_r}\log\frac{n_r^i}{n_r}, \qquad (8)$$

where q is the number of classes in the data set, and n_r^i is the number of documents of the ith class that were assigned to the rth cluster. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropies weighted according to the cluster size. That is,

$$\text{Entropy} = \sum_{r=1}^{k}\frac{n_r}{n}E(S_r). \qquad (9)$$

A perfect clustering solution will be the one that leads to clusters that contain documents from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

Programs and Data

An illustrative example of a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters is CLUTO(Karypis, 2002). CLUTO has implementations of the various clustering algorithms and evaluation metrics described in previous sections. It was designed by the University of Minnesota's data mining's group and is available at www.cs.umn.edu/~karypis/cluto. CLUTO has been developed as a general purpose clustering toolkit. CLUTO's distribution consists of both stand-alone programs (vcluster and scluster) for clustering and analyzing these clusters, as well as a library through which an application program can access directly the various clustering and analysis algorithms implemented in CLUTO. Utility tools for preprocessing documents into vector matrices and some sample document datasets are also available at www.cs.umn.edu/~karypis/cluto.

Cross References

►Clustering
►Information Retrieval
►Text Mining
►Unsupervised Learning

Recommended Reading

Boley, D. (1998). Principal direction divisive partitioning. *Data Mining and Knowledge Discovery, 2*(4), 325–344.

Cutting, D. R., Pedersen, J. O., Karger, D. R., & Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. In *Proceedings of the ACM SIGIR* (pp. 318–329). Copenhagen, Denmark.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, 39*(1), 1–38.

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge discovery and data mining* (pp. 269–274). San Francisco: Morgan Kaufmann.

Ding, C., He, X., Zha, H., Gu, M., & Simon, H. (2001). *Spectral min-max cut for graph partitioning and data clustering*. Technical report TR-2001-XX, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. New York: Wiley.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research, 4*, 147–180.
Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Englewood Cliffs, NJ: Prentice-Hall.
Karypis, G. (2002). CLUTO*: A clustering toolkit*. Technical report 02-017, Department of Computer Science, University of Minnesota. Available at http://www.cs.umn.edu/~cluto.
King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association, 69*, 86–101.
MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th symposium on mathematical statistics and probability* (pp. 281–297). Berkeley, CA: University of California Press.
Salton, G. (1989). *Automatic text processing: The transformation, analysis, and retrieval of information by computer*. Reading, MA: Addison-Wesley.
Sneath, P. H., & Sokal, R. R. (1973). *Numerical taxonomy*. San Francisco: Freeman.
Zahn, K. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers, (C-20)*, 68–86.
Zha H., He X., Ding C., Simon H., and Gu M. Bipartite graph partitioning and data clustering. In *Proceedings of the International Conference on Information and Knowledge Management*, 2001.
Zhao, Y., & Karypis, G. (2004). Criterion functions for document clustering: Experiments and analysis. *Machine Learning, 55*, 311–331.

Dual Control

►Bayesian Reinforcement Learning
►Partially Observable Markov Decision Process

Duplicate Detection

►Entity Resolution

Dynamic Bayesian Network

►Learning Graphical Models

Dynamic Decision Networks

►Partially Observable Markov Decision Processes

Dynamic Memory Model

SUSAN CRAW
The Robert Gordon University, Scotland, UK

Synonyms

Dynamic memory model; Memory organization packets

Definition

Schank's dynamic memory model (Schank, 1982) was designed to capture knowledge of specific experiences. Schank's memory organization packets (MOPs) and Kolodner's E-MOPs (episodic MOPS) (Kolodner, 1983) provide templates about typical scenes. For a restaurant scene these might identify "being seated," "ordering," and "paying."

Cross References

►Case-Based Reasoning

Recommended Reading

Kolodner, J. (1983). Reconstructive memory. A computer model. *Cognitive Science, 7*(4), 281–328.
Schank, R. S. (1982). *Dynamic Memory : A theory of reminding and learning in computers and people*. New York: Cambridge University Press.

Dynamic Programming

MARTIN L. PUTERMAN[1], JONATHAN PATRICK[2]
[1]University of British Columbia, Vancouver, Canada
[2]University of Ottawa, Ottawa, Canada

Definition

Dynamic programming is a method for modeling a sequential decision process in which past decisions impact future possibilities. Decisions can be made at fixed discrete time intervals or at random time intervals triggered by some change in the system. The decision process can last for a finite period of time or run indefinitely – depending on the application. Each time a decision needs to be made, the decision-maker (referred to as "he" in this chapter with no sexist connotation intended) views the current ►state of the system and chooses from a known set of possible ►actions. As a result of the state of the system and the action chosen, the decision-maker receives a ►reward (or pays a ►cost) and the system evolves to a new state based on known probabilities. The challenge faced by the decision-maker is to choose a sequence of actions that will lead to the greatest reward over the length of the decision-making horizon. To do this, he needs to consider not only the current reward (or cost) for taking a given action but the impact such an action might have on future rewards. A ►policy is a complete sequence of decisions that dictates what action to take in any given state and at any given time. Dynamic programming finds the optimal policy by developing mathematical recursions that decompose the multi-decision problem into a series of single-decision problems that are analytically or computationally more tractable.

Background and Motivation

The earliest concepts that later developed into dynamic programming can be traced back to the calculus of variations problems in the seventeenth century. However, the modern investigation of stochastic sequential decision problems arguably dates back to work by Wald in 1947 on sequential statistical analysis. At much the same time, Pierre Masse was analyzing similar problems applied to water resource management in France. However, the major name associated with dynamic programming is that of Richard Bellman who established the optimality equations that form the basis of dynamic programming.

It is not hard to demonstrate the potential scope of dynamic programming. Table 1 gives a sense of the breadth of application as well as highlighting the stochastic nature of most instances.

Structure of the Learning System

A dynamic program is a general representation of a sequential decision problem under uncertainty about the future and is one of the main methods for solving Markov Decision Problems (see ►Markov Decision Process). Like a decision tree, it models a process where the decision we make "today" impacts where we end up tomorrow and therefore what decisions are available to us tomorrow. It has distinct advantages over a decision tree in that:

- It is a more compact representation of a decision process
- It enables efficient calculation
- It allows exploration of the structural properties of optimal decisions
- It can analyze and solve problems with infinite or indefinite time horizons

The Finite Horizon Setting

A finite horizon MDP, is a decision process with a known end date. Thus, the decision-maker is faced with the task of making a finite sequence of decisions at fixed intervals. The MDP model is based on five elements:

►Decision epochs: Sequences of decision times $n = 1, \dots, N$ (in the infinite horizon we set $N = \infty$). In a discrete time MDP, these decision times happen at regular, fixed intervals while in a continuous time model they occur at random times triggered by a change in the system. The time between decision epochs is called a period.

►State space: States represent the possible system configurations facing the decision-maker at each decision epoch. They contain all information available to the decision-maker at each decision epoch. The state space, S, is the set of all such states (often assumed to be finite). In choosing the state space, it is important to include all the information that may be relevant in determining a decision and that may change from decision epoch to decision epoch.

►Actions: Actions are the available choices for the decision-maker at any given decision epoch, in any given state. $A(s)$ is the set of all actions available in state s (usually assumed to be finite for all s). No action is taken in the final decision epoch N.

►Transition probabilities: The probability of being in state s' at time $t+1$, given you take action a from state s at time t, is written as $p_t(s'|s,a)$. It clearly makes sense to allow the transition probabilities to be conditional upon the current state and the action taken.

►Rewards/costs: In most MDP applications, the decision-maker receives a reward each period. This reward can depend on the current state, the action taken, and the next state and is denoted by $r_t(s,a,s')$. Since a decision must be made before knowing the next state, s', the

Dynamic Programming. Table 1 Dynamic programming applications

Application	System state	Actions	Rewards	Stochastic aspect
Capacity	Size of plant	Maintain or add capacity	Costs of expansion and production at current capacity	Demand for a product
Cash mgt	Cash available	Borrow or invest	Transaction costs and less interest	External demand for cash
Catalog mailing	Customer purchase record	Type of catalog to send, if any	Purchases in current period less mailing costs	Customer purchase amount
Clinical trials	Number of successes with each treatment	Stop or continue the trial	Costs of treatment and incorrect decisions	Response of a subject to treatment
Economic growth	State of the economy	Investment or consumption	Utility of consumption	Effect of investment
Fisheries mgt	Fish stock in each age class	Number of fish to to harvest	Value of the catch	Population size
Forest mgt	Size and condition of stand	Harvesting and reforestation activities	Revenues and less harvesting costs	Stand growth and price fluctuation
Gambling	Current wealth	Stop or continue playing	Cost of playing	Outcome of the game
Inventory control	Stock on hand	Order additional stock	Revenue per item sold and less ordering, holding, and penalty costs	Demand for items
Project selection	Status of each project	Project to invest in at present	Return from investing in project	Change in project status
Queueing control	Number in the queue	Accept/reject new customers or control service rate	Revenue from serving customers and less delay costs	Interarrival times and service times
Reliability	Age or status of equipment	Inspect and repair or replace if necessary	Inspection, repair, and failure costs	Failure and deterioration
Reservations	Number of confirmed reservations	Accept, wait-list, or reject new reservation	Profit from satisfied reservations and less overbooking penalties	Number of arrivals and the demand for reservations
Scheduling	Activities completed	Next activity to schedule	Cost of activity	Length of time to complete activity
Selling an asset	Current offer	Accept or reject the offer	The offer is less than the cost of holding the asset for one period	Size of the offer
Water resource management	Level of water in each reservoir	Quantity of water to release	Value of power generated	Rainfall and run-off

MDP formulation deals with the expected reward:

$$r_t(s,a) = \sum_{s' \in S} r_t(s,a,s')p_t(s'|s,a).$$

We also define the terminal rewards as $r_N(s)$ for being in state s at the final decision epoch. These are independent of the action since no action is taken at that point.

The objective in the finite horizon model is to maximize total expected reward:

$$\max\left\{E\left[\sum_{t=1}^{N} r_t(s_t,a_t,s_{t+1}) + r_N(s_N)|s_1 = s\right]\right\}. \quad (1)$$

At any given time t, the decision-maker has observed the history up to time t, represented by $h_t = (s_1,a_1,s_2,a_2,\ldots,a_{t-1},s_t)$, and needs to choose a_t in such a way as to maximize (1). A ▶decision rule, d_t, determines what action to take, based on the history to date at a given decision epoch and for any possible state. It is ▶deterministic if it selects a single member of $A(s)$ with probability 1 for each $s \in S$ and for a given h_t, and it is ▶randomized (▶randomized decision rule) if it selects a member of $A(s)$ at random with probability $q_{d_t(h_t)}(a)$. It is Markovian (▶Markovian decision rule) if it depends on h_t only through s_t. That is, $d_t(h_t) = d_t(s_t)$.

A ▶policy, $\pi = (d_1,\ldots,d_{N-1})$, denotes a complete sequence of decision rules over the whole horizon. It can be viewed as a "contingency plan" that determines the action for each possible state at each decision epoch. One of the major results in MDP theory is that, under reasonable conditions, it is possible to prove that there exists a Markovian, deterministic policy that attains the maximum total expected reward. Thus, for the purposes of this chapter we will concentrate on this subset of all policies.

If we define, $v_t(s)$ as the expected total reward from time t to the end of the planning horizon, given that at time t the system occupies state s, then a recursion formula can be built that represents v_t in terms of v_{t+1}. Specifically,

$$v_t(s) = \max_{a \in A(s)}\left\{r_t(s,a) + \sum_{s' \in S} p(s'|s,a)v_{t+1}(s')\right\} \quad (2)$$

This is often referred to as the ▶Bellman equation, named after Richard Bellman who was responsible for the seminal work in this area. It breaks the total reward at time t, into the immediate reward $r_t(s,a)$ and the total future expected reward, $\sum_{s' \in S} p(s'|s,a)v_{t+1}(s')$. Define $A^*_{s,t}$ as the set of actions that attain the maximum in (2) for a given state s and decision epoch t. Then the finite horizon discrete time MDP can be solved through the following backward induction algorithm.

Backward Induction Algorithm

- Set $t = N$ and $v_t(s) = r_N(s) \quad \forall s \in S$ (since there is no decision at epoch N and no future epochs, it follows that the optimal reward-to-go function is just the terminal reward).
- Let $t = t - 1$ and compute for each $s \in S_t$

$$v_t(s) = \max_{a \in A(s)}\left\{r_t(s,a) + \sum_{s' \in S} p(s'|s,a)v_{t+1}(s')\right\}.$$

- For each $s \in S_t$, compute $A^*_{s,t}$ by solving

$$\text{argmax}_{a \in A(s)}\left\{r_t(s,a) + \sum_{s' \in S} p(s'|s,a)v_{t+1}(s')\right\}.$$

- If $t = 1$ then stop else return to step 2.

The function $v_1(s)$ is the maximum expected reward over the entire planning horizon given the system starts in state s. The optimal policy is constructed by choosing a member of $A^*_{s,t}$ for each $s \in S$ and $t \in \{1,\ldots,N\}$. In essence, the algorithm solves a complex N-period decision problem by solving N simple 1-period decision problems.

Example – inventory control: Periodically (daily, weekly, or monthly), an inventory manager must determine how much of a product to stock in order to satisfy random external demand for the product. If too little is in stock, potential sales are lost. Conversely, if too much is on hand, a cost for carrying inventory is incurred. The objective is to choose an ordering rule that maximizes expected total profit (sales minus holding and ordering costs) over the planning horizon. To formulate an MDP model of this system requires precise assumptions such as:

- The decision regarding the quantity to order is made at the beginning of each period and delivery occurs instantaneously.

- Demand for the product arrives throughout the period, but all orders are filled on the last day of the period.
- If demand exceeds the stock on hand, potential sales are lost.
- The revenues, costs and demand distribution are the same each period.
- The product can only be sold in whole units.
- The warehouse has a capacity for M units.

(These assumptions are not strictly necessary but removing them leads to a different formulation.) Decisions epochs correspond to the start of a period. The state, $s_t \in \{0,\ldots,M\}$, represents the inventory on hand at the start of period t and the action, $a_t \in \{0,1,2,\ldots,M-s\}$, is the number of units to order that period; the action 0 corresponds to not placing an order. Let D_t represent the random demand throughout period t and assume that the distribution of demand is given by $p_t(d) = P(D_t = d), d = 0,1,2,\ldots$. The cost of ordering u units is $O(u) = K + c(u)$ (a fixed cost plus variable cost) and the cost of storing u units is $h(u)$, where $c(u)$ and $h(u)$ are increasing functions in u. We will assume that left-over inventory at the end of the planning horizon has value $g(u)$ and that the sale of u units yields a revenue of $f(u)$. Thus, if there are u units on hand at decision epoch t, the expected revenue is

$$F_t(u) = \sum_{j=0}^{u-1} f(j)p_t(j) + f(u)P(D_t \geq u).$$

The expected reward is therefore

$$r_t(s,a) = F(s+a) - O(a) - h(s+a)$$

and the terminal rewards are $r_N(s,a) = g(s)$. Finally, the transition probabilities depend on whether or not there is enough stock on hand, $s+a$, to meet the demand for that month, D_t. Specifically,

$$p_t(j|s,a) = \begin{cases} 0 & \text{if } j > s+a, \\ p_t(j) & \text{if } j = s+a-D_t, s+a \leq M, \\ & \quad s+a > D_t, \\ \sum_{d=s+a}^{\infty} p_t(d) & \text{if } j = 0, s+a \leq M, s+a \leq D_t. \end{cases}$$

Solving the finite horizon version of this problem through backward induction reveals a simple form to the optimal policy referred to as an (s,S) policy. Specifically, if at time t, the inventory is below some number s^t then it is optimal to order a quantity that raises the inventory level to S^t. It has been shown that a structured policy of this type is optimal for several variants of the inventory management problem with a fixed ordering cost. Many variants of this problem have been studied; these models underly the field of supply chain management.

The Infinite Horizon Setting

In the infinite (or indefinite) horizon setting, the backward induction algorithm described above no longer suffices as there are no terminal rewards with which to begin the process.

In most finite horizon problems, the optimal policy begins to look the same at each decision epoch as the horizon is pushed further and further into the future. For instance, in the inventory example above, $s^t = s^{t+1}$ and $S^t = S^{t+1}$ if t is sufficiently removed from the end of the horizon. The form of the optimal policy only changes as the end of the time horizon approaches. Thus, if there is no fixed time horizon, we should expect the optimal policy to be *stationary* in most cases. We call a policy *stationary* if the same decision rule is applied at each decision epoch (i.e., $d_t = d \; \forall \; t$). One necessary assumption for this to be true is that the rewards and transition probabilities are independent of time (i.e., $r_t(s,a) = r(s,a)$ and $p_t(s'|s,a) = p(s'|s,a) \; \forall \; s,' s \in S$ and $a \in A(s)$). For the infinite horizon MDP, the theory again proves that under mild assumptions there exists an optimal policy that is *stationary, deterministic*, and *Markovian*. This fact greatly simplifies the process of finding the optimal policy as we can concentrate on a small subset of all potential policies.

The set up for the infinite horizon MDP is entirely analogous to the finite horizon setting with the same ►decision epochs, ►states, ►actions, ►rewards, and ►transition probabilities (with the last two assumed to be independent of time).

The most obvious objective is to extend the finite horizon objective to infinity and seek to find the policy, π, that maximizes the total expected reward:

$$v^\pi(s) = \lim_{N\to\infty} \left\{ E_s^\pi \left[\sum_{t=1}^{N} r(s_t, a_t) \right] \right\}. \qquad (3)$$

This, however, is problematic since

1. The sum may be infinite for some or all policies
2. The sum may not even exist, or
3. Even if the sum exists, there may be no maximizing policy

In the first case, just because all (or a subset of all) policies lead to infinite reward in the long run does not mean that they are all equally beneficial. For instance, one may give a reward of \$100 each epoch and the other \$1 per epoch. Alternatively, one may give large rewards earlier on while another gives large rewards only much later. Generally speaking, the first is more appealing but the above objective function will not differentiate between them. Secondly, the limit may not exist if, for instance, the reward each decision epoch oscillates between 1 and −1. Thirdly, there may be no maximizing policy simply because there is an infinite number of policies and thus there may be an infinite sequence of policies that converges to a maximum limit but never reaches it. Thus, instead we look to maximize either the *total expected discounted reward* or the *expected long run average reward* depending on the application.

Let $\lambda \in (0,1)$ be a discount factor. Assuming the rewards are bounded (i.e., there exists an M such that $|r(s,a)| < M \quad \forall (s,a) \in S \times A(s)$), the *total expected discounted reward* for a given policy π is defined as

$$v_\lambda^\pi(s) = \lim_{N\to\infty} E_s^\pi \left\{ \sum_{t=1}^{N} \lambda^{t-1} r(s_t, d_t(s_t)) \right\}$$
$$= E_s^\pi \left\{ \sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right\}.$$

Since, $\lambda < 1$ and the rewards are bounded, this limit always exists. The second objective is the *expected average reward* which, for a given policy π, is defined as

$$g^\pi(s) = \lim_{N\to\infty} \frac{1}{N} E_s^\pi \left\{ \sum_{t=1}^{N} r(s_t, d_t(s_t)) \right\}.$$

Once again, we are dealing with a limit that may or may not exist. As we will see later, whether the above limit exists depends on the structure of the Markov chain induced by the policy.

Let us, at this point, formalize what we mean by an optimal policy. Clearly, that will depend on which objective function we choose to use. We say that

- π^* is *total reward optimal* if $v^{\pi^*}(s) \geq v^\pi(s)\ \forall s \in S$ and $\forall \pi$.
- π^* is *discount optimal* if $v_\lambda^{\pi^*}(s) \geq v_\lambda^\pi(s)\ \forall s \in S$ and $\forall \pi$.
- π^* is *average optimal* if $g^{\pi^*}(s) \geq g^\pi(s)\ \forall s \in S$ and $\forall \pi$.

For simplicity, we introduce matrix and vector notation. Let $r_d(s) = r(s, d(s))$ and $p_d(j|s) = p(j|s, d(s))$. Thus r_d is the vector of rewards for each state under decision rule d, and P_d is the transition matrix of states under decision rule d. We will now take a more in-depth look at the infinite horizon model with the total expected discounted reward as the optimality criterion.

Solving the Infinite Horizon Discounted MDP

Given a Markovian, deterministic policy $\pi = (d_1, d_2, d_3, \ldots)$ and defining $\pi_k = (d_k, d_{k+1}, \ldots)$ we can compute

$$v_\lambda^\pi(s) = E_s^{\pi_1}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right]$$
$$= E_s^{\pi_1}\left[r(s, d_1(s)) + \lambda \sum_{t=2}^{\infty} \lambda^{t-2} r(s_t, d_t(s_t)) \right]$$
$$= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) E_j^{\pi_2}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(s_t, d_t(s_t)) \right]$$
$$= r(s, d_1(s)) + \lambda \sum_{j\in S} p_{d_1}(j|s) v_\lambda^{\pi_2}(j).$$

In matrix notation,

$$v_\lambda^{\pi_1} = r_{d_1} + \lambda P_{d_1} v_\lambda^{\pi_2}.$$

If we follow our supposition that we need to only consider *stationary* policies (so that the same decision rule is applied to every decision epoch), $\pi = d^\infty = (d, d, \ldots)$, then this results in

$$v_\lambda^{d^\infty} = r_d + \lambda P_d v_\lambda^{d^\infty}.$$

This implies that the value function generated by a stationary policy satisfies the equation:

$$v = r_d + \lambda P_d v$$
$$\Rightarrow v = (I - \lambda P_d)^{-1} r_d.$$

The inverse above always exists since P_d is a probability matrix (so that its spectral radius is less than or equal to 1) and $\lambda \in (0,1)$. Moving to the maximization problem of finding the optimal policy, we get the recursion formula

$$v(s) = \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{j \in S} p(s|s,a)v(j) \right\}. \qquad (4)$$

Note that the right hand side can be viewed as a function of a vector v (given r, p, λ). We define a vector-valued function

$$Lv = \max_{d \in D^{MD}} \left\{ r_d + \lambda P_d v \right\},$$

where D^{MD} is the set of all Markovian, ▶deterministic decision rules. There are three methods for solving the above optimization problem in order to determine the optimal policy. The first method, called *value iteration*, creates a sequence of approximations to the value function that eventually converges to the value function associated with the optimal policy.

Value Iteration

1. Start with an arbitrary $|S|$-vector v^0. Let $n = 0$ and choose $\epsilon > 0$ to be small.
2. For every $s \in S$, compute $v^{n+1}(s)$ as
$$v^{n+1}(s) = \max_{a \in A(s)} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^n(j) \right\}.$$
3. If $\max_{s \in S} |v^{n+1}(s) - v^n(s)| \geq \epsilon(1-\lambda)/2\lambda$ let $n \to n+1$ and return to step 2.
4. For each $s \in S$, choose
$$d_\epsilon(s) \in \operatorname{argmax}_{a \in A(s)} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v^{n+1}(j) \right\}.$$

It has been shown that value iteration identifies a policy with expected total discounted reward within ϵ of optimality in a finite number of iterations. Many variants of value iteration are available such as using different stopping criteria to accelerate convergence or combining value iteration with the policy iteration algorithm described below.

A second algorithm, called *policy iteration*, iterates through a sequence of policies eventually converging to the optimal policy.

Policy Iteration

1. Set $d_0 \in D$ to be an arbitrary policy. Let $n = 0$.
2. (Policy evaluation) Obtain v^n by solving
$$v^n = (I - \lambda P_{d_n})^{-1} r_{d_n}.$$
3. (Policy improvement) Choose d_{n+1} to satisfy
$$d_{n+1} \in \operatorname{argmax}_{d \in D} \{ r_d + \lambda P_d v^n \}$$
componentwise. If d_n is in this set, then choose $d_{n+1} = d_n$.
4. If $d_{n+1} = d_n$, set $d^* = d_n$ and stop. Otherwise, let $n \to n+1$ and return to (2).

Note that value iteration and policy iteration have different conceptual underpinnings. Value iteration seeks a *fixed point* of the operator L using successive approximations while policy iteration can be viewed as using Newton's Method to solve $Lv - v = 0$.

Finally, a third method for solving the discounted infinite horizon MDP takes advantage of the fact that, because L is monotone, if $Lv \leq v$ then $L^2 v \leq Lv$ and more generally, $L^k v \leq v$. Thus, induction implies that the value function of the optimal policy, v_λ^* is less than or equal to v for any v, where $Lv \leq v$. We define the set $U := \{v \in V | Lv \leq v\}$. Then, not only is v_λ^* in the set U, it is also the smallest element of U. Therefore, we can solve for v_λ^* by solving the following linear program:

$$\min_v \sum_{s \in S} \alpha(s) v(s)$$

subject to

$$v(s) \geq r(s,a) + \lambda \sum_{j \in S} p(j|s,a) v(j) \quad \forall s \in S, \quad a \in A_s.$$

(Note that the above set of constraints is equivalent to $Lv \leq v$.) We call this the primal LP. The coefficients $\alpha(s)$ are arbitrarily chosen. The surprising fact is that the solution to the above LP will be v_λ^* for any strictly positive α.

We can construct the dual to the above primal to get

$$\max_X \sum_{s \in S} \sum_{a \in A_s} r(s,a) X(s,a)$$

subject to

$$\sum_{a\in A_j} X(j,a) - \sum_{s\in S}\sum_{a\in A_s} \lambda p(j|s,a)X(s,a) = \alpha(j) \quad \forall j \in S$$

$$X(s,a) \geq 0 \quad \forall s \in S, \quad a \in A_s.$$

Let $(X(s,a) : s \in S, a \in A_s)$ be a feasible solution for the dual (i.e., satisfies the constraints but not necessarily optimal). Every such feasible solution corresponds to a randomized Markov policy d^∞ and vice versa. Furthermore, for a given feasible solution, X, and the corresponding policy $d^\infty, X(s,a)$ represents the expected total number of times you will be in state s and take action a following policy d^∞ before stopping in the indefinite horizon problem. Thus, the objective in the dual can be interpreted as the total expected reward over the length of the indefinite horizon. The strong law of duality states that at the optimal solution the objective functions in the primal and dual will be equal. But we already know that at the optimal, the primal objective will correspond to a weighted sum of $v^*_\lambda(s), s \in S$, which is the total expected discounted reward over the infinite (or indefinite) horizon given you start in state s. Thus our interpretations for the primal and dual variables coincide.

Solving the Infinite Horizon Average Reward MDP

Recall that in the average reward model, the objective is to find the policy that has the maximum average reward, often called the *gain*. The gain of a policy can be written as

$$g^\pi(s) = \lim_{n\to\infty} \frac{1}{N} v^\pi_{N+1} = \lim_{n\to\infty} \frac{1}{N} \sum_{n=1}^{N} [P^{n-1}_\pi r_{d_s}](s). \quad (5)$$

As mentioned earlier, the major drawback is that for a given policy π, the gain may not even exist. An important result, however, states that if we confine ourselves to stationary policies, we can in fact be assured that the gain is well defined. Our ability to solve a given infinite horizon average reward problem depends on the form of the Markov chains induced by the deterministic, stationary policies available in the problem. Thus, we divide the set of average reward MDPs according to the structure of the underlying Markov chains. We say that an MDP is

- *Unichain* if the transition matrix corresponding to *every* deterministic stationary policy is unichain, that is, it consists of a single recurrent class plus a possibly empty set of transient states, or
- *Multichain* if the transition matrix corresponding to *at least one* stationary policy contains two or more closed irreducible recurrent classes

If an MDP is unichain, then the gain for any given stationary, deterministic policy can be defined by a single number (independent of starting state). This makes intuitive sense since if we assume that it is possible to visit every state from every other one (possibly minus some set of transient states that may be visited initially but will eventually be abandoned) then it would seem reasonable to assume that over the infinite horizon the initial starting state would not impact the average reward. However, if the initial state impacts what set of states can be visited in the future (i.e., the MDP is multichain) then clearly it is likely that the expected average reward will be dependent on the initial state.

If the average reward MDP is unichain then the *gain* can be uniquely determined by solving

$$v(s) = \max_{a\in A(s)} \left\{ r(s,a) - g + \sum_{s'\in S} p(s'|s,a)v(s') \right\}. \quad (6)$$

Notice that the above equation has $|S|+1$ unknowns but only $|S|$ equations. Thus, v is not uniquely determined. To specify v uniquely, it is sufficient to set $v(s') = 0$ for some $s' \in S$. If this is done, then $v(s)$ is called the relative value function and $v(j) - v(k)$ is the difference in expected total reward obtained in using an optimal policy and starting in state j as opposed to state k. It is also often represented by the letter h and called the *bias*.

As in the discounted infinite horizon MDP, there are three potential methods for solving the average reward case. We present only policy iteration here and refer the reader to the recommended readings for value iteration and linear programming.

Policy Iteration

1. Set $n = 0$, and choose an arbitrary decision d_n.
2. (Policy evaluation) Solve for g_n, v_n:

$$0 = r_{d_n} - ge + (P_{d_n} - I)v.$$

3. Choose d_{n+1} to satisfy

$$d_{n+1} \in \operatorname{argmax}_{d \in D}\{r_d + P_d v_n\}.$$

Setting $d_{n+1} = d_n$ if possible.
4. If $d_{n+1} = d_n$, stop, set $d^* = d_n$. Else, increment n by 1 and return to step 2.

As mentioned earlier, the equation in step 2 fails to provide a unique v_n since we have $|S| + 1$ unknowns and only $|S|$ equations. We therefore need an additional equation. Any one of the following three will suffice:

1. Set $v_n(s_0) = 0$ for some fixed $s_0 \in S$.
2. Choose v_n to satisfy $P^*_{d_n} v_n = 0$.
3. Choose v_n to satisfy $-v_n + (P_d - I)w = 0$ for some $w \in V$.

Continuous Time Models

So far, we have assumed that decision epochs occur at regular intervals but clearly in many applications this is not the case. Consider, for instance, a queueing control model where the service rate can be adjusted in response to the size of the queue. It is reasonable to assume, however, that changing the service rate is only possible following the completion of a service. Thus, if the service time is random then the decision epochs will occur at random time intervals. We will therefore turn our attention now to systems in which the state changes and decision epochs occur at random times. At the most general level, decisions can be made at any point in time but we will focus on the subset of models for which decision epochs only occur at state transitions. It turns out that this is usually sufficient as the added benefit of being able to change decisions apart from state changes does not generally improve performance. Thus, the models we study generalize the discrete time MDP models by:

1. Allowing, or requiring, the decision-maker to choose actions whenever the system changes state
2. Modeling the evolution of the system in continuous time, and
3. Allowing the time spent in a particular state to follow an arbitrary probability distribution

Semi-Markov decision processes (SMDP) are continuous time models where decisions are made at some but not necessarily all state transitions. The most common subset of these, called exponential SMDPs, are SMDPs where the intertransition times are exponentially distributed.

We distinguish between two processes:

1. The natural process that monitors the state of the system as if it were observed continually through time and
2. The embedded Markov chain that monitors the evolution of the system at the decision epochs only

For instance, in a queueing control model one may decide only to change the rate of service every time there is an arrival. Then the embedded Markov chain would only keep track of the system at each arrival while the natural process would keep track of all state changes – including both arrivals and departures.

While the actions are generally only going to depend on the state of the system at each decision epoch, it is possible that the rewards/costs to the system may depend on the natural process. Certainly, in the queueing control model the cost to the system would go down as soon as a departure occurs. In discrete models it was sufficient to let the reward depend on the current state s and the current action a and possibly the next state s'. However, in an SMDP, the natural process may change between now and the next decision epoch and moreover, the time the process stays in a given state is no longer fixed. Thus we need to consider two types of rewards/costs. First, a lump sum reward, $k(s, a)$, for taking action a when in state s. Second, a reward *rate*, $c(j, s, a)$, paid out for each time unit that the natural process spends in state j until the next decision epoch when the state at the last decision epoch was s and the action taken was a. Note that if we insist that every state transition triggers a decision epoch, we can reduce this to $c(s, a)$ since the system remains in s until the next decision epoch.

Before we can state our objective we need to determine what we mean by discounting. Again, because we are dealing with continuous time so that decision epochs are not evenly spaced, it is not sufficient to have a fixed discount factor λ. Instead, we will discount future rewards at rate $e^{-\alpha t}$, for some $\alpha > 0$. If we let $\lambda = e^{-\alpha}$

(the discount rate for one time unit) then $\alpha = 0.11$ corresponds to $\lambda = 0.9$. Thus an α around 0.1 is commonly used.

We can now state our objective. We look to find a policy that maximizes the total expected discounted reward over the infinite horizon. There is an average reward model for continuous time models as well but we will not discuss that here. Given a policy π we can write its total expected discounted reward as:

$$v_\alpha^\pi(s) = E_s^\pi\left[\sum_{n=0}^{\infty} e^{-\alpha\sigma_n}\left(K(X_n, Y_n) + \int_{\sigma_n}^{\sigma_{n+1}} e^{-\alpha(t-\sigma_n)} c(W_t, X_n, Y_n)\,dt\right)\right], \quad (7)$$

where X_n and Y_n are the random variables that represent the state and action at time n respectively, W_t is the random variable that represents the state of the natural process at time t, and σ_n is the random time of the nth decision epoch. Again, if we assume that each state transition triggers a decision epoch, $X_n = W_t$ for all $t \in [\sigma_n, \sigma_{n+1})$. We seek to find a policy π such that

$$v_\alpha^\pi(s) = v_\alpha^*(s) = \max_{\pi \in \Pi^{HR}} v_\alpha^\pi(s) \quad (8)$$

for all $s \in S$. Perhaps surprisingly, (7) can be reduced to one that has the same form as in the discrete time case for any SMDP. As a consequence, all the theory and the algorithms that worked in the discrete version can be transferred to the continuous model! Again, we refer the reader to the recommended readings for the details.

Extensions

►Partially Observed MDPs

In some instances, the state of the system may not be directly observable but instead, the decision-maker receives a signal from the system that provides information about the state. For example, in medical decision making, the health care provider will not know the patient's true health status but will have on hand some diagnostic information that may be related to the patient's true health. These problems are modeled from a Bayesian perspective. The decision-maker uses the signal to update his estimate of the probability distribution of the system state. He then bases his decision on this probability distribution. The computational methods for solving partially observed MDPs are significantly more complex than in the fully observable case and only small problems have been solved numerically.

Parameter-Adaptive Dynamic Programming

Often the transition probabilities in an MDP are derived from a system model, which is determined by a few parameters. Examples include demand distributions in inventory control and arrival and/or service distributions in queueing systems. In these cases the forms of the distributions are known (for example, Poisson for demand models and exponential for arrival or service models) but their parameter values are not. Herein, the decision-maker seeks a policy that combines *learning* with *control*. A Bayesian approach is used. The parameter is related to the system state through a likelihood function and after observing the system state, the probability distribution on the parameter is updated. This updated probability distribution provides the basis for choosing a policy.

Approximate Dynamic Programming

Arguably the greatest challenge to implementing MDP theory in practice is "the curse of dimensionality." As the complexity of a problem grows, the amount of information that needs to be stored in the state space quickly reaches a point where the MDP is no longer computationally tractable. There now exist several methods for dealing with this problem, all of which are grouped under the title of approximate dynamic programming or neuro-dynamic programming. These potential methods begin by restricting the value function to a certain class of functions and then seeking to find the optimal value function within this class. A typical approximation scheme is based on the linear architecture:

$$v^*(s) \approx \tilde{v}(s, r) = \sum_{i=1}^{k} r_i \phi_i(s),$$

where $\phi_i(s), i = 1, \ldots, k$ are pre-defined basis functions that attempt to characterize the state space and r is a set of weights applied to the basis functions. This reduces the problem from one with $|S|$-dimensions to one with $|k|$-dimensions. The questions are (1) how do you determine what class of functions (determined by ϕ) to choose and (2) how to find the best approximate value function within the chosen class (i.e., the best values for r)? The first question is still very much wide open.

Answers to the second question fall into two main camps. On the one hand, there are a number of methods that seek to iteratively improve the approximation through the simulation of sample paths of the decision process. The second method uses linear programming but restricts the value function to the approximate form. This reduces the number of variables in the primal to a reasonable number (equal to the number of basis functions chosen). One can then determine the optimal set of weights, r, through column generation. One of the major challenges facing approximate dynamic programming is that it is difficult to determine how close the approximate value function is to its true value. In other words, how much more reward might have been accumulated had the original MDP been solved directly? Though there are some attempts in the literature to answer this question, it remains a significant challenge.

Cross References

►Markov Decision Processes
►Partially Observable Markov Decision Processes

Recommended Reading

Bertsekas, D. (2000). *Dynamic programming and optimal control*. Belmont: Athena Scientific.
Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.
Feinberg, E., & Shwartz, A. (2002). *Handbook of Markov decision processes*. Boston, MA: Kluwer Academic Publishers.
Puterman, M. (1994). *Markov decision processes*. New York: Wiley.
Sutton, R., & Barto, A. (1998). *Reinforcement learning*. Cambridge, MA: MIT Press.

Dynamic Programming For Relational Domains

►Symbolic Dynamic Programming

Dynamic Systems

The dynamic systems approach emphasizes the human, and animal, interaction with the environment. Interactions are described by partial differential equations. Attractors and limit cycles represent stable states which may be analogous to attribute-values.