D

Data Augmentation

▸ Data Enrichment

Data Cleaning

▸ Data Cleansing

Data Cleansing

Synonyms

Data cleaning; Data reconciliation; Data scrubbing

Data cleansing is the process of detecting and correcting (or removing) corrupt or inaccurate records from data.

Cross-References

▸ Data Preparation

Data Enrichment

Synonyms

Data augmentation; Data integration

Data enrichment is the process of adding to an existing data collection. This commonly involves sourcing of additional information about the data points on which data are already held.

Cross-References

▸ Data Preparation

Data Integration

▸ Data Enrichment

Data Linkage

▸ Record Linkage

Data Matching

▸ Record Linkage

Data mining on Text

▸ Text Mining

C. Sammut, G.I. Webb (eds.), *Encyclopedia of Machine Learning and Data Mining*,
DOI 10.1007/978-1-4899-7687-1

Data Preparation

Zahraa S. Abdallah[1], Lan Du[1], and Geoffrey I. Webb[2]
[1]Faculty of Information Technology, Monash University, Clayton, Melbourne, VIC, Australia
[2]Faculty of Information Technology, Monash University, Victoria, Australia

Abstract

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.

Data preparation is typically an iterative process of manipulating raw data, which is often unstructured and messy, into a more structured and useful form that is ready for further analysis. The whole preparation process consists of a series of major activities (or tasks) including data profiling, cleansing, integration, and transformation.

Synonyms

Data preprocessing; Data wrangling

Motivation and Background

Data are collected for many purposes, not necessarily with machine learning or data mining 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 and is often the most time-consuming part of any nontrivial machine learning or data mining project.

In most cases, the preparation process consists of dozens of transformations and needs to be repeated several times. Despite advances in technologies for working with data, each of those transformations may involve much-handcrafted work and can consume a significant amount of time and effort. Thus, working with huge and diverse data remains a challenge. It is often agreed that data wrangling/preparation is the most tedious and time-consuming aspect of data analysis. It has become a big bottleneck or "iceberg" for performing advanced data analysis, particularly on big data. A recent article in the New York Times For Big-Data Scientists reported that the whole process of data wrangling could account up to 80 % of the time in the analysis cycle. In other words, there is only a small fraction of time for data analysts and scientists to do analysis work. According to the data science report Data science report, published by Crown in 2015, messy and disorganized data are the number one obstacle holding data scientists back. The same study reports that 70 % of a data scientist's time is spent in cleaning data.

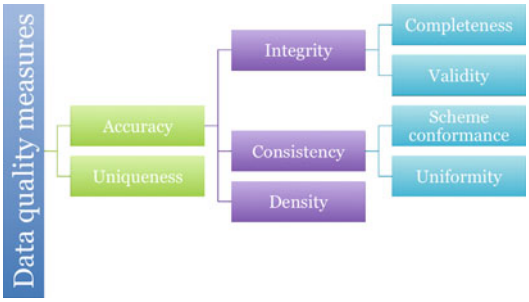
Processes and Techniques

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.

Data Profiling: 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. It is also important to assess whether there are sufficient data to realistically obtain the desired machine learning outcomes.

Data quality should also be investigated, as data sets are often 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



Data Preparation, Fig. 1 Data quality measures (Adapted from Müller and Freytag 2005)

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.

Assuring and improving data quality are two of the primary reasons for data preprocessing. There are common criteria to measure and evaluate the quality of data, which can be categorized into two main elements, accuracy and uniqueness (Müller and Freytag 2005), as explained in Fig. 1.

Accuracy is described as an aggregated value over the quality criteria: integrity, consistency, and density. Intuitively this describes the extent to which the data are an exact, uniform, and complete representation of the *mini-world*: the aspects of the world that the data describe. We describe each accuracy criterion as follows:

- **Integrity:** An integral data collection contains representations of all the entities in the mini-world and only of those. Integrity requires both completeness and validity.
  - **Completeness:** Complete data give a comprehensive representation of the mini-world and contain no missing values. We achieve completeness within data cleansing by correcting anomalies and not just deleting them. It is also possible that additional data are generated, representing existing entities that are currently unrepresented in the data. A problem with assessing completeness is that you do not know what you do not know. As a result, there are no known gold standard data, which can be used as a reference to measure completeness.
  - **Validity:** Data are valid when there are no constraints violated. There are numerous mechanisms to increase validity including mandatory fields, enforcing unique values, and data schema/structure.
- **Consistency:** This quality concerns syntactic anomalies as well as contradictions. The main challenge concerning data consistency is choosing which data source you trust for reliable agreement among data across different sources.
  - **Schema conformance:** This is especially true for the relational database systems where the adherence of domain formats relies on the user.
  - **Uniformity:** This is directly related to irregularities.
- **Density:** This criterion concerns the quotient of missing values in the data. There still can be nonexistent values or properties that have to be represented by null values having the exact meaning of not being known.

The above three criteria of integrity, consistency, and density collectively represent the accuracy measure.

The other major quality measure that is also crucial to measure data quality is uniqueness. Uniqueness is satisfied when the data do not contain any duplicates.

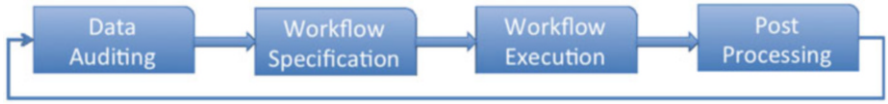
Timeliness is another criterion that also has been considered for data quality. This criterion refers to the currency of the data that keeps it up to date.

More information about data quality can be found in Dasu and Johnson (2003) and Müller and Freytag (2005).

Data Cleansing

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

Data cleansing is defined as the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database. Data cleansing can also be referred to as data cleaning, data scrubbing, or data reconcilia-



Data Preparation, Fig. 2 Data cleansing process (Adapted from Müller and Freytag 2005)

tion. More precisely, the process of data cleansing could be explained as a four-stage process:

1. Define and identify errors in data such as incompleteness, incorrectness, inaccuracy, or irrelevancy.
2. Clean and rectify these errors by replacing, modifying, or deleting them.
3. Document error instances and error types.
4. Measure and verify to see whether the cleansing meets the user's specified tolerance limits in terms of cleanliness.

Data Anomalies

Data are symbolic representations of information, i.e., facts or entities from parts of the world, called a mini-world, depicted by symbolic values. Imperfections in the data set correspond to differences between an ideal (i.e., error-free) data set (DI) and the real data (DR). In this context, anomalousness is a property of data that renders an erroneous representation of the mini-world.

The term *data anomaly* describes any distortion of data resulting from the data collection process. From this perspective, anomalies include duplication, inconsistency, missing values, outliers, noisy data, or any kind of distortion that can cause data imperfections.

Anomalies can be classified at a high level into three categories:

- **Syntactic anomalies:** describe characteristics concerning the format and values used for the representation of the entities. Syntactic anomalies include lexical errors, domain format errors, syntactical errors, and irregularities.
- **Semantic anomalies:** hinder the data collection from being a comprehensive and nonredundant representation of the mini-world. These types of anomalies include integrity constraint violations, contradictions, duplicates, and invalid tuples.
- **Coverage anomalies:** decrease the number of entities and entity properties from the mini-world that is represented in the data collection. Coverage anomalies are categorized as missing values and missing tuples.

Therefore, it is clear that data anomalies can take a number of different forms, each with a different range of analytical consequences.

Data Cleansing Process

Data cleansing is an iterative process that consists of the four consecutive steps (Müller and Freytag 2005), as depicted in Fig. 2:

1. **Data auditing:** This first step mainly identifies the types of anomalies that reduce data quality. Data auditing checks the data using validation rules that are prespecified and then creates a report of the quality of the data and its problems. We often apply some statistical tests in this step for examining the data.
2. **Workflow specification:** The next step is to detect and eliminate anomalies by a sequence of operations on the data. The information collected from data auditing is then used to create a data-cleaning plan. It identifies the causes of the dirty data and plans steps to resolve them.
3. **Workflow execution:** The data-cleaning plan is executed, applying a variety of methods on the data set.
4. **Post-processing and controlling:** The post-processing or control step involves examination of the workflow results and performs exception handling for the data mishandled by the workflow.

Dealing with Missing Values

One major task in data cleansing is dealing with missing values. 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 See ▸ missing attribute values.

Handling data that contain missing values is crucial for the data cleansing process and data wrangling in general. In real-life data, most of existing data sets contain missing values that were not introduced or were lost in the recording process for many reasons.

Handling Outliers

An outlier is another type of data anomaly that requires attention in the cleansing process. Outliers are data that do not conform to the overall data distribution.

Outliers can be seen from two different perspectives; first, they might be seen as glitches in the data. Alternatively, they might be also seen as interesting elements that could potentially represent significant elements in the data. For example, outliers in sales records for a store might reflect a successful marketing campaign. Therefore, to classify data as outliers, we must define what the normal behavior of the data is and therefore how different or significant the outlier is relative to normal behavior. There might be different normal behaviors for data and thus different classes of outliers. From the above definition, we can see that as the normality in data differs, various classes of outliers can be detected. To be able to do that, we need to formalize both the normality in the data and inconsistency of the outliers. Read more about handling outliers for data preprocessing in Han et al. (2011).

Data Enrichment/Integration

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 obtaining socioeconomic data about individual customers. The imported data must be integrated with the other data for a unified view of all data sources.

Data integration is a crucial task in data preparation. Combining data from different sources is not trivial especially when dealing with large amounts of data and heterogeneous sources. Data are typically presented in different forms (structured, semi-structured, or unstructured) as well as from different sources (web, database) that could be stored locally or distributed. Moreover, structured data coming from a single source might have different schemas. The combination of these variations is not an easy task.

Integration of data brings many opportunities, yet it also comes with various challenges. We highlight the most relevant challenges below:

1. **Data are heterogeneous:** Data integration involves a combination of data coming from different sources that have been developed independently of each other and thus vary in data format. Each source will have its own schemas, definition of objects, and structure of data (tables, XML, unstructured text, etc.).
2. **The number of sources:** Data integration is already a challenge for a small number of sources, but the challenges are exacerbated when the number of sources grows (such as Web-scale data integration).
3. **Object identity and separate schemas:** Differences exist both on the level of individual objects and the schema level. Every source classifies their data according to taxonomies pertinent to a certain domain.
4. **Time synchronization:** Each source might have a different time window over which data have been captured, different granularities at which events are modeled (daily, weekly, annually), and frequency at which they are updated. Synchronization of these differences and making time-sensitive data compatible are another challenge.
5. **Dealing with legacy data:** There are still important data stored in a legacy form such as IMS, spreadsheets, and ad hoc structures. Combining legacy data with other modern data structures such as XML is a challenging task.
6. **Abstraction levels:** Different data sources might provide data at incompatible levels of abstraction. When combining data, differences in levels of specificity must be resolved.
7. **Data quality:** Data are often erroneous, and combining data often aggravates the problem. Erroneous data has a potentially devastating

impact on the overall quality of the integration process.

The integration process can be divided into two main subtasks, schema integration and data integration, where each has its own techniques and challenges. Schema integration concerns a holistic view across data sources. It focuses on formats, structures, and identification of objects and their level of abstraction. This includes semantic mapping, matching, resolving naming conflicts, and entity resolution. The contents of data add another clue to the integration process.

Even with data from different sources that have identical schemas, integration on the data level is still essential. Data integration deals with different types of problems that concern the data itself rather than the overall structure as in schema integration. Common data integration problems are duplication in data and inconsistency. Correlated or duplicated values/attributes may increase both size and complexity of the data. Resolving conflicts at the data level enhances the overall performance of the integration process.

Data Transformation

It is frequently necessary to transform data from one representation to another. There are many reasons for changing representations:

- **To generate symmetric distributions instead of the original skewed distributions**.
- **Transformation improves visualization** of data that might be tightly clustered relative to a few outliers.
- Data are transformed to achieve **better interpretability.**
- Transformations are often used to **improve the compatibility of the data with assumptions underlying a modeling process**, for example, to linearize (straighten) the relation between two variables whose relationship is nonlinear. Some of the data mining algorithms require the relationship between data to be linear.

In the following, we will discuss different types of transformation whereby each data point $\mathbf{x_i}$ is replaced with a **transformed value** $\mathbf{y_i = f(x_i)}$, where $\mathbf{f}$ is the transformation function. Many techniques are applied for data transformation. Each technique has its own purpose and dependency on the nature of data. Some of the major transformations are discussed below.

Numeric to Numeric Transformation

Normalization and Rescaling

It is usually the case that raw data are not in a suitable form to be processed by machine learning and data mining techniques. Data normalization is the process of transforming raw data values to another form with properties that are more suitable for modeling and analysis. The normalization process focuses on scaling data in terms of range and distribution. Therefore, it consists of two main processes:

- *Min-max normalization* projects the original range of data onto a new range. Very common normalization intervals are [0,1] and [−1,1]. This normalization method is very useful when we apply a machine learning or data mining approach that utilizes distance. For example, in k-nearest neighbor methods, using un-normalized values might cause attributes whose values have greater magnitudes to dominate over other attributes. Therefore, normalization aims to standardize magnitudes across variables. A useful application for min-max scaling is image processing where pixel intensities have to be normalized to fit within a certain range (i.e., 0–255 for the RGB color range). Also, typical neural network algorithms (ANN) require data that is on a 0–1 scale. Normalization provides the same range of values for each of the inputs to the model.
- **Z-score normalization** (also referred to as standardization) is a normalization method that transforms not only the data magnitude but also the dispersion. Some data mining methods are based on the assumption that data follow a certain distribution. For example, methods such as logistic regression, SVM, and neural network when

using gradient descent/ascent optimization methods assume data follow a Gaussian distribution. Otherwise, the approaches will be ill conditioned and might not guarantee a stable convergence of weight and biases. Other approaches such as linear discriminant analysis (LDA), principal component analysis (PCA), and kernel principal component analysis require features to be on the same scale to find directions that maximize the variance (under the constraints that those directions/eigenvectors/principal components are orthogonal). Z-score normalization overcomes the problem of variables with different units as it transforms variables so that they are centered on 0 with a standard deviation of 1.

- **Decimal scaling** is another type of scaling transformation where the decimal place of a numeric value is shifted so the maximum absolute value will be always less than 1.

Linear Transformation

Linear transformations preserve linear relationships within data. A function f(.) results in a linear transformation if and only if for all values x and y in the original representation, f(x)+f(y) = f(x+y) and f(x)−f(y) = f(x−y). Examples of a linear transformation are transforming Celsius to Fahrenheit, miles to kilometers, and inches to centimeters. All linear transformations follow the standard linear regression formula to convert variables linearly.

Many other transformations are not linear. A nonlinear transformation changes (increases or decreases) linear relationships between variables and, thus, changes the correlation between variables. Examples of nonlinear transformations are square root, raising to a power, logarithm, and any of the trigonometric functions. In the following, we discuss some nonlinear transformation methods.

Power Transformation (Tukey's Ladder of Powers)

Tukey describes a way of re-expressing variables using a power transformation (Tukey 1977). The aim of this transformation is to improve the linearity between variables. When we consider two variables (x and y), transformation can be applied to one variable or both of them depending on the relationship between the two variables. This kind of transformation fits when the relationship between the two variables is monotonic and has a single bend. When the data are represented as pairs of (x,y), Tukey has expressed data transformation as

$$y^{a} = \beta_0 + \beta_1 + x^{b}.$$

The choice of a and b decides on the transformation type in the relationship between x and y. Figure 3 shows a visual rule of thumb that has been proposed by John Tukey. The following diagram gives us an insight to understand which transformations are likely to work with different types of data.

We explain Tukey's ladder rule as follows: Suppose the data patterns follow a similar curve as the blue line in Q1; thus the data could be transformed by going up the ladder for x, y, or both. If the data pattern is shaped similar to that shown in Q2, then we should try to transform the data by going the down-ladder for x and/or up-ladder for y. Similar procedures can be applied for the other two quarters. Figure 4 explains the ladder of power for variable y. The transformation is stronger when the power value is away from 1 (the original data) in both directions (up and down).

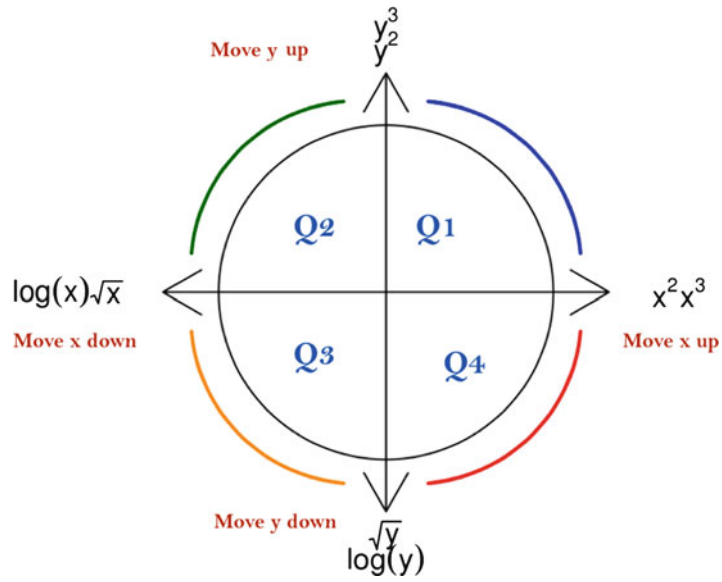
Choosing the Right Numeric Transformation

There is no definite answer to what is the best transformation method to use for a particular data set. The choice is very data dependent and requires an understanding of the domain as well as the data distribution. Trial and error for the common transformation methods may also be required. Table 1 summarizes the main transformation methods.

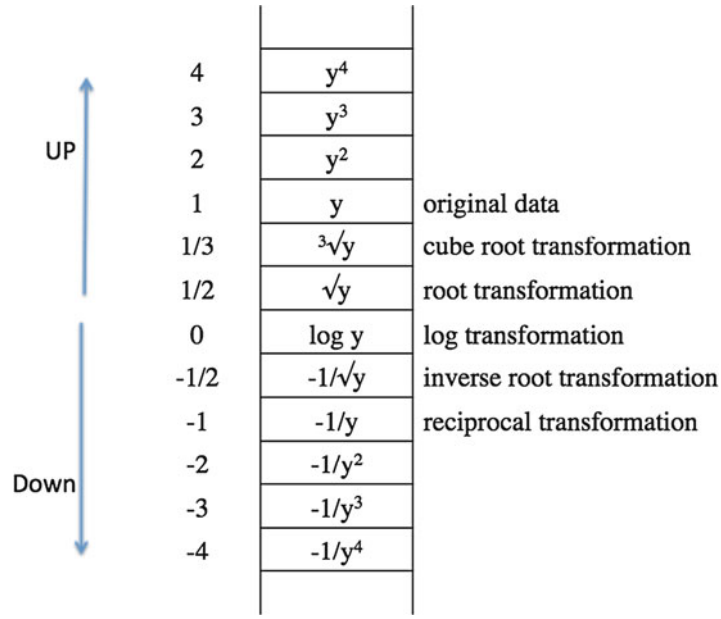
Nominal to Numeric Transformation

All the aforementioned methods transform and re-express numerical variables. However, the transformation of nominal variables is equally

Data Preparation, Fig. 3
Tukey's ladder rule



Data Preparation, Fig. 4
The ladder of power



important, especially for machine learning and data mining methods that only accept numerical values such as SVM and ANN. Assume that we have a nominal variable x with N different nominal values. There are two approaches to transform x into a numeric variable:

1. The first and simplest approach is to map nominal values to the integers 1 to N. Although simple, this method has two major drawbacks:
   - Integer substitution may impose an ordering that does not actually exist in the original data.
   - The integer value might be used as part of the calculation in the mining algorithm giving an incorrect meaning of weights based on the assigned values.

Data Preparation, Table 1 Summary of key transformation methods

Method	Pros	Cons
Standard linear regression	–Preserves the relationship between variables	–No actual transformation occurred
Reciprocal transformation	–Making small values bigger and big values smaller –Reducing the effect of outliers	–Not applicable for zero
Log transformation	–Good for right skewed data –$\log_{10}(x)$ is especially good at handling higher-order powers of 10 (e.g., 1000,100,000)	–Not applicable for zero and negative values (constant can be added to overcome this)
Root transformation	–Simple counts –Good for right skewed data	–Not applicable for negative values (constant can be added to overcome this)
Logit transformation	–Works with proportions and percents	–Not applicable for 0 and 1 values
Cube root transformation	–Can be applied on negative and 0 values	–Not effective in transformation as long model

2. The other main approach is to first binarize the variable (see 3.7 Binarization) and then map each of the N new binary attributes to the integer values 0 and 1. This approach is generally viewed as safer than the first and hence is more widely used.

Propositionalization

Some data sets contain information expressed in a relational format, 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 re-expressed 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.

Discretization

Discretization transforms continuous data into a discrete form. This is useful in many cases for better data representation, data volume reduction, better data visualization, and representing data at various levels of granularity for data analysis. Data discretization approaches are categorized as supervised, unsupervised, bottom-up, or top-down. Approaches for data discretization include binning, entropy based, nominal to numeric, 3-4-5 rule, and concept hierarchy. See ▶ Discretization.

Binarization

Some systems cannot process multivalued categorical variables. This limitation can be circumvented by binarization, a process that converts a multivalued 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 multivalued categorical variable.

Granularity

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 dairy 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 be 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.

Dimensionality Reduction

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 component analysis and kernel methods.

Feature Engineering

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.

Feature engineering can be considered as means for dimensionality reduction also, by replacing the original features by a smaller number of derived features.

See ▸ Feature Selection and ▸ Feature Construction in Text Mining.

Sampling

Much of the theory on which learning systems are based assumes that the training data are randomly sampled from the population about which the user wishes to learn a model. However, much historical data contains sampling biases, for example, data that were easy to collect or were considered interesting for some other purpose. It is important to consider whether the available data are sufficiently representative of the future data to which a learned model is to be applied.

In all sampling methods, the aim is to select a sample S containing N instances from the entire data D. Each method models the relationship between a population and a sample with an underlying mathematical process. We discuss in the following some of these methods:

Random Sampling

In this method, S is selected randomly from D with a probability of $1/N$ for any instance in D to be selected. *Simple random sampling* may have very poor performance in the presence of skew in data. There are two main variants of simple random sampling, *with replacement* (SRSWR) and *without replacement* (SRSWOR). For sampling with replacement, the instance that is drawn from the population is replaced, and therefore it might be chosen again. For sampling without replacement, each instance that is drawn from D is removed, and hence S must contain N distinct instances.

Simple random sampling is usually easy to implement and to understand. However, it might cause loss of accuracy if applied to skewed data by failing to include sufficient data to accurately represent the tail of the distribution. The simple random sample might also result in substantial variance across samples.

Cluster Sampling

This method approximates the percentage of each class (or subpopulation of interest) in the overall data set; then it draws a simple random sample from each cluster. In this method, we might not have a complete list of population members (i.e., not all data available). However, a list of groups or "clusters" of this population is available and complete. That means the clusters could be incomplete, but a list of them is complete. Therefore, cluster sampling is a cost-efficient sampling method, as it does not require data to be complete. A drawback for cluster sampling is the possible poor representation of the diversity in clusters.

Stratified Sampling

If D is divided into mutually disjoint parts called strata, obtaining a simple random sample from each stratum generates a stratified sample.

Stratified sampling has a number of advantages. First, inferences can be made about specific subgroups for more efficient statistical estimates. Since each stratum is treated as an independent population, different sampling approaches can be applied to different strata. Second, this method will never result in lower efficiency than the simple random sample, provided that each stratum is proportional to the group's size in the population. Finally, it increases data readability as it represents individual preexisting strata within a population rather than the overall population.

Stratified sampling is complex to implement and estimate. It also can be sensitive to parameters such as selection criteria and minimum group

size. Finally, stratified sampling techniques are generally used when the population is heterogeneous, or dissimilar, where certain homogeneous, or similar, subpopulations can be isolated (strata). Thus, this method will not be useful when there are no homogeneous subgroups. Read more about sampling techniques in García et al. (2015).

Balanced sampling is a special case of stratified sampling where the strata correspond to the classes and the sample drawn from each strata is proportional to the class's size in the population.

Cross-References

- ▸ Anomaly Detection
- ▸ Binning
- ▸ Data Set
- ▸ Dimensionality Reduction
- ▸ Discretization
- ▸ 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
- ▸ Record Linkage

Recommended Reading

Barnett V, Lewis T (1994) Outliers in statistical data, 3rd edn. Wiley series in probability and mathematical statistics. Applied probability and statistics. Wiley, Chichester/New York

Crown (2015) Data science report. http://visit.crowdflower.com/2015-data-scientist-report.html

Dasu T, Johnson T (2003) Exploratory data mining and data cleaning, vol 479. Wiley, New York

Data science report (2014) http://visit.crowdflower.com/2015-data-scientist-report.html

Doan A, Halevy A, Ives Z (2012) Principles of data integration. Morgan Kaufmann, Waltham

For Big-Data Scientists (2014) 'Janitor Work' Is Key Hurdle to Insights. http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?_r=0&module=ArrowsNav&contentCollection=Technology&action=keypress&region=FixedLeft&pgtype=article (The NYT article by Steve Lohr)

García S, Luengo J, Herrera F (2015) Data preprocessing in data mining. Springer, Cham

Han J, Pei J, Kamber M (2011) Data mining: concepts and techniques. Elsevier, Burlington

Müller H, Freytag J-C (2005) Problems, methods, and challenges in comprehensive data cleansing. Professoren des Inst. Für Informatik, Berlin

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

Tukey JW (1977) Exploratory data analysis, pp 2–3

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, Amsterdam/Boston

Data Preprocessing

- ▸ Data Preparation

Data Scrubbing

- ▸ Data Cleansing

Data Reconciliation

- ▸ Data Cleansing
- ▸ Record Linkage

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.

Data Wrangling

- ▸ Data Preparation

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
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

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
▸ Decision Lists and Decision Trees
▸ Rule Learning
▸ Rule Set

Recommended Reading

Rivest RL (1987) Learning decision lists. Mach Learn 2:229–246

Decision Lists and Decision Trees

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Definition

▸ Decision trees and ▸ decision lists are two popular ▸ hypothesis languages, which share quite a few similarities, but also have important differences with respect to expressivity and learnability.

Discussion

The key difference between decision trees and decision lists is that the former 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 non-overlapping, i.e., each example is covered by exactly one 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).

This is also reflected in the learning strategies that are typically used for learning these concept classes. Decision trees are traditionally learned with a ▸ divide-and-conquer strategy, which successively divides the example space into non-overlapping regions, whereas the ▸ covering algorithm that is typically used for learning rule sets is also known as ▸ separate-and-conquer Fürnkranz (1990) because it successively removes (separates) examples covered by previously learned rule. For a comparison between the two strategies we refer to Boström (1995).

Moreover, the restriction of decision tree learning algorithms to non-overlapping 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 non-overlapping 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 and Witten (1998), which learns a decision list that consists of a 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 non-overlapping rules Quinlan (1987a).

See Also

- ▸ Covering Algorithm
- ▸ Decision Tree
- ▸ Divide-and-Conquer Learning
- ▸ Rule Learning

Recommended Reading

Henrik Boström. 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)*, pages 1194–1200, 1995.

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

Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 144–151, Madison, Wisconsin, 1998. Morgan Kaufmann.

Johannes Fürnkranz Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

John Ross Quinlan. Generating production rules from decision trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 304–307. Morgan Kaufmann, 1987a.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

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

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

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.

Discussion

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 and Schapire (1996).

Cross-References

▸ Bias Variance Decomposition
▸ Decision Tree
▸ Overfitting

Recommended Reading

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufmann, pp 148–156

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learn 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
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

Abstract

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 et al. 1984; Kass 1980) and machine learning (Hunt et al. 1966; Quinlan 1983, 1986) communities. A *decision tree* is a tree-structured classification model, which is easy to understand, even by non-expert users, and can be efficiently induced from data. An extensive survey of decision-tree learning can be found in Murthy (1998).

Synonyms

Classification tree

Representation

Figure 1 shows a well-known 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). On the right, a simple decision tree that can be induced from such data is shown. Classification of a new example starts at the top node – the *root* – and the value of the attribute that corresponds to this tree is considered (*outlook* in the example). Classification then proceeds by moving 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 we arrive at a terminal node – a so-called *leaf* – which is not labeled with an attribute but with a value of the target attribute (*play golf?*). For all examples that arrive at the same leaf, the same target value will be predicted. Figure 1 shows leaves as rectangular boxes.

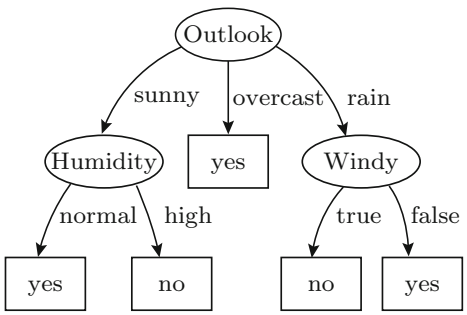
Note that some of the attributes may not occur at all in the tree. For example, the tree in Fig. 1 does not contain a test on *temperature* because the training data can be classified without making a reference to this variable. More generally, one can say that the attributes in the upper parts of the 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 t. 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 beforehand using a ▸ discretization algorithm (Fig. 2).

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

Outlook	*Temp*	*Humidity*	*Windy*	*Golf?*
rainy	hot	high	false	no
rainy	hot	high	true	no
overcast	hot	high	false	yes
sunny	mild	high	false	yes
sunny	cool	normal	false	yes
sunny	cool	normal	true	no
overcast	cool	normal	true	yes
rainy	mild	high	false	no
rainy	cool	normal	false	yes
sunny	mild	normal	false	yes
rainy	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
sunny	mild	high	true	no



Decision Tree, Fig. 1 A data set describing weather conditions and a target variable (*Play Golf?*) and a decision tree learned for this dataset (Quinlan 1986)

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 non-overlapping, 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, i.e., examples with the same feature values but different class values.

function TDIDT(S)

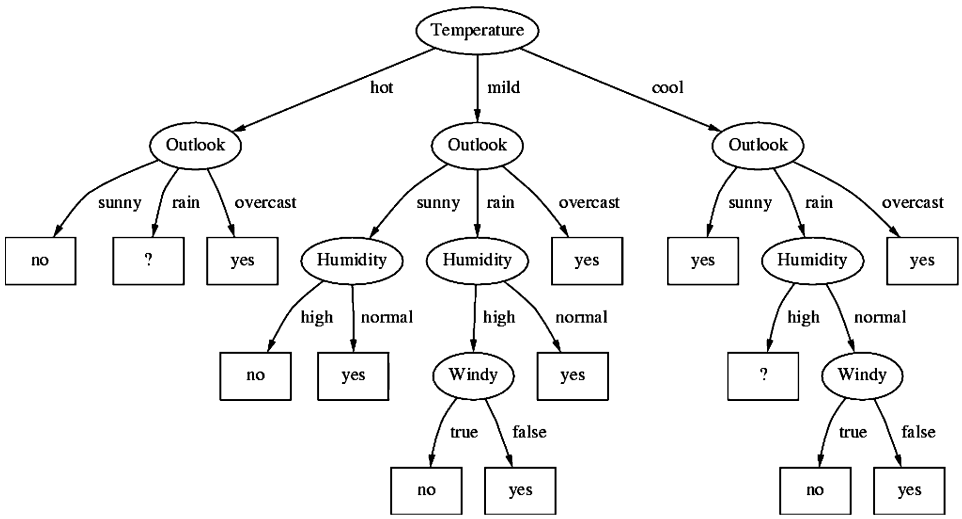
Input: S, a set of labeled examples.

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*

Decision Tree, Fig. 2 Top-down induction of decision trees

Attribute Selection

The crucial step in decision-tree induction is the choice of an adequate attribute. In the sample tree of Fig. 3, which has been generated from the same 14 training examples as the tree of Fig 1, most leaves contain only single training example, i.e., 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.



Decision Tree, Fig. 3 A needlessly complex decision tree describing the same dataset

Typical attribute selection criteria use a function that measures the *impurity* of a node, i.e., 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 Gini index (Breiman et al. 1984), which are defined as

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

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

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, we want 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 S_t are non-overlapping disjoint subsets $S_t \in S$ that are induced by splitting the attribute A, 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 fits 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 Mingers (1989a) and Buntine and Niblett (1992).

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, i.e., 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 (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 and 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 Trees
- ▸ Pruning
- ▸ Regression Trees
- ▸ Rule Learning

Recommended Reading

Breiman L (2001) Random forests. Mach Learn 45(1): 5–32
Breiman L, Friedman JH, Olshen R, Stone C (1984) Classification and regression trees. Wadsworth & Brooks, Pacific Grove
Buntine W, Niblett T (1992) A further comparison of splitting rules for decision-tree induction. Mach Learn 8:75–85
Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Proceedings of the 13th international conference on machine learning, Bari. Morgan Kaufmann, pp 148–156
Hunt EB, Marin J, Stone PJ (1966) Experiments in induction. Academic, New York
Kass GV (1980) An exploratory technique for investigating large quantities of categorical data. Appl Stat 29:119–127
Mingers J (1989a) An empirical comparison of selection measures for decision-tree induction. Mach Learn 3:319–342
Mingers J (1989b) An empirical comparison of pruning methods for decision tree induction. Mach Learn 4:227–243
Murthy SK (1998) Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min Knowl Discov 2(4):345–389
Quinlan JR (1983) Learning efficient classification procedures and their application to chess end games. In: Michalski RS, Carbonell JG, Mitchell TM (eds) Machine learning. An artificial intelligence approach, Tioga, Palo Alto, pp 463–482
Quinlan JR (1986) Induction of decision trees. Mach Learn 1:81–106
Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo

Quinlan JR (1996) Improved use of continuous attributes in C4.5. J Artif Intell Res 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

Deduplication or Duplicate Detection (When Applied to One Database Only)

▸ Record Linkage

Deep Belief Nets

Geoffrey Hinton
University of Toronto, Toronto, ON, 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 (Werbos 1974; Rumelhart et al. 1986) 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 and 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

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

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 et al. (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 (Welling et al. 2005; Movellan and Marks 2001) 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 (Hinton 1989; Bengio et al. 2007; LeCun and 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 et al. 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 et al. 2007), video sequences (Sutskever and Hinton 2007), and motion-capture data (Taylor et al. 2007). If the number of units in the highest layer is small, deep belief nets perform nonlinear dimensionality reduction (Hinton and 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 and 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 component analysis, and if the highest level code is forced to be binary, they allow extremely fast retrieval of documents or images (Salakhutdinov and Hinton 2007; Torralba et al. 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, Vancouver, vol 19. MIT, Cambridge

Hinton GE (1989) Connectionist learning procedures. Artif Intell 40(1–3):185–234

Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18:1527–1554

Hinton GE, Salakhutdinov RR (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, Corvalis. ACM, New York

LeCun Y, Bengio Y (2007) Scaling learning algorithms towards AI. In: Bottou L et al (eds) Large-scale kernel machines. MIT, Cambridge

Movellan JR, Marks TK (2001) Diffusion networks, product of experts, and factor analysis

Ranzato M, Huang FJ, 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

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

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536

Salakhutdinov RR, Hinton GE (2007) Semantic hashing. In: Proceedings of the SIGIR workshop on information retrieval and applications of graphical models, Amsterdam

Selfridge OG (1958) Pandemonium: a paradigm for learning. In: Proceedings of a symposium on mechanisation of though processes, National Physical Laboratory. HMSO, London

Sutskever I, Hinton GE (2007) Learning multilevel distributed representations for high-dimensional sequences. In: Proceedings of the eleventh international conference on artificial intelligence and statistics, San Juan

Taylor GW, Hinton GE, Roweis S (2007) Modeling human motion using binary latent variables. In: Advances in neural information processing systems, Vancouver, vol 19. MIT, Cambridge

Torralba A, Fergus R, Weiss Y (2008) Small codes and large image databases for recognition. In: IEEE conference on computer vision and pattern recognition, Anchorage, pp 1–8
Welling M, Rosen-Zvi M, Hinton GE (2005) Exponential family harmoniums with an application to information retrieval. In: Advances in neural information processing systems, Vancouver, vol 17. MIT, Cambridge, pp 1481–1488
Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University, Cambridge

Deep Belief Networks

▸ Deep Belief Nets

Deep Learning

Jürgen Schmidhuber
The Swiss AI Lab, IDSIA, USI & SUPSI,
Manno & Lugano, Switzerland

Abstract

Deep learning artificial neural networks have won numerous contests in pattern recognition and machine learning. They are now widely used by the worlds most valuable public companies. I review the most popular algorithms for feedforward and recurrent networks and their history.

Introduction

Deep learning has revolutionized Pattern Recognition and Machine Learning. It is about credit assignment in adaptive systems with long chains of potentially causal links between actions and consequences.

The ancient term "deep learning" was first introduced to Machine Learning by Dechter (1986) and to artificial neural networks (NNs) by Aizenberg et al. (2000). Subsequently it became especially popular in the context of deep NNs, the most successful deep learners, which are much older though, dating back half a century. This article will focus on essential developments since the 1960s, addressing supervised, unsupervised, and (briefly) reinforcement learning. There is a recent, more detailed survey with 888 references (Schmidhuber 2015). LeCun et al. (2015) provide a more limited view of more recent deep learning history. The present condensed survey is based on the Scholarpedia article (Schmidhuber 2015b).

A standard NN consists of many simple, connected processors called units, each producing a sequence of real-valued activations. Input units get activated through sensors perceiving the environment, other units through connections with real-valued weights from previously active units. Some units may influence the environment by triggering actions. Learning or credit assignment is about finding weights that make the NN exhibit desired behavior, such as controlling a robot. Depending on the problem and how the units are connected, such behavior may require long causal chains of computational stages, where each stage transforms (often in a nonlinear way) the aggregate activation of the network. Deep learning in NNs is about accurately assigning credit across many such stages.

In a sense, sequence-processing recurrent NNs (RNNs) are the ultimate NNs, because they are general computers (an RNN can emulate the circuits of a microchip). In fully connected RNNs, all units have connections to all non-input units. Unlike feedforward NNs, RNNs can implement while loops, recursion, etc. The program of an RNN is its weight matrix. RNNs can learn programs that mix sequential and parallel information processing in a natural and efficient way.

To measure whether credit assignment in a given NN application is of the deep or shallow type, we consider the length of the corresponding credit assignment paths, which are chains of possibly causal connections between subsequent unit activations, e.g., from input units through hidden units to output units in feedforward NNs (FNNs) without feedback connections or through transformations over time in RNNs. FNNs with fixed topology have a problem-independent maximal problem depth bounded by the number of layers

of units. RNNs, the deepest of all NNs, may learn to solve problems of potentially unlimited depth, for example, by learning to store in their activation-based "short-term memory" representations of certain important previous observations for arbitrary time intervals.

The difficulty of a problem may have little to do with its depth. Some NNs can quickly learn to solve certain deep but simple problems through random weight guessing (e.g., Hochreiter and Schmidhuber 1997b). In general, however, finding an NN that precisely models a given training set (of input patterns and corresponding labels) is an NP-complete problem and also in the case of deep NNs (e.g., Sima 1994).

First Deep Learners

Certain early NNs (McCulloch and Pitts 1943) did not learn at all. Hebb (1949) published ideas about unsupervised learning. The following decades brought shallow unsupervised NNs and supervised NNs (e.g., Rosenblatt 1958). Early supervised NNs were essentially variants of linear regressors dating back two centuries (Gauss, Legendre).

Deep learning networks originated in the 1960s when Ivakhnenko and Lapa (1965) published the first general, working learning algorithm for supervised deep feedforward multilayer perceptrons. Their units had polynomial activation functions combining additions and multiplications in Kolmogorov-Gabor polynomials. Ivakhnenko (1971) already described a deep network with eight layers trained by the "group method of data handling," still popular in the new millennium. Given a training set of input vectors with corresponding target output vectors, layers are incrementally grown and trained by regression analysis and then pruned with the help of a separate validation set, where regularization is used to weed out superfluous units. The numbers of layers and units per layer can be learned in problem-dependent fashion.

Like later deep NNs, Ivakhnenko's nets learned to create hierarchical, distributed, internal representations of incoming data. Many later nonneural methods of Artificial Intelligence and Machine Learning also learn more and more abstract, hierarchical data representations. For example, syntactic pattern recognition methods (Fu 1977) such as grammar induction discover hierarchies of formal rules to model observations.

Architectures of Convolutional NNs (CNNs)

The 1970s also saw the birth of the convolutional NN (CNN) architecture (Fukushima's Neocognitron, 1979) inspired by neurophysiological insights. Today such architectures are widely used for computer vision. Here the (typically rectangular) receptive field of a unit with given weight vector (a filter) is shifted step by step across a two-dimensional array of input values, such as the pixels of an image (usually there are several such filters). The resulting array of subsequent activation events of this unit can then provide inputs to higher-level units and so on. Due to massive weight replication, relatively few parameters may be necessary to describe the behavior of such convolutional layers, which typically feed downsampling layers consisting of units whose fixed-weight connections originate from physical neighbors in the convolutional layers below. Downsampling units use "spatial averaging" to become active if at least one of their inputs is active; their responses are insensitive to certain small image shifts. Weng (1993) later replaced spatial averaging by "max-pooling" (MP), which is widely used today. Here a two-dimensional layer or array of unit activations is partitioned into smaller rectangular arrays. Each is replaced in a downsampling layer by the activation of its maximally active unit.

Backpropagation

Ivakhnenko and Fukushima did not yet use supervised backpropagation (BP) to train the weights of their nets by gradient descent in an objective function, such as the total classification error on a given training set of input patterns and corresponding labels, although BP was also developed back then.

BP's continuous form was derived in the early 1960s (Kelley 1960; Bryson 1961; Bryson and Ho 1969). Dreyfus (1962) published the elegant derivation of BP based on the chain rule only. BP's modern efficient version for discrete sparse networks (including FORTRAN code) was published by Linnainmaa (1970). Here the complexity of computing the derivatives of the output error with respect to each weight is proportional to the number of weights. That's the method still used today. Dreyfus (1973) used BP to change weights of controllers in proportion to such gradients. By 1980, automatic differentiation could derive BP for any differentiable graph (Speelpenning 1980). Werbos (1982) published the first application of BP to NNs, extending thoughts in his 1974 thesis, which did not yet have Linnainmaa's modern, efficient form of BP. In 1980–1990, computers became 10,000 times faster than those of 1960–1970 and widely accessible in academic labs. Computational experiments then demonstrated that BP in NNs can indeed yield useful internal representations in hidden layers of NNs (Rumelhart et al. 1986). Wan (1994) produced the first BP-trained NN to win a controlled pattern recognition contest with secret test set. Amari (1998) described BP for natural gradient-based NNs. By 2003, deep BP-based standard FNNs with up to seven layers were used to successfully classify high-dimensional data (e.g., Vieira and Barradas 2003).

In the 2000s, computing hardware had again become 10,000 times faster than in the 1980s. Cheap massively parallel graphics processing units (GPUs, originally developed for video games) started to revolutionize NN research. Standard FNNs implemented on GPU were 20 times faster than on CPU (Oh and Jung 2004). A plain GPU-based FNN trained by BP with pattern distortions (Baird 1990) set a new record of 0.35 % error rate (Ciresan et al. 2010) on the MNIST handwritten digit dataset, which by then had been the perhaps most famous machine learning benchmark for decades. This seemed to suggest that advances in exploiting modern computing hardware were more important than advances in algorithms.

Backpropagation for CNNs

LeCun et al. (1989) first applied BP to Neocognitron-like CNNs, achieving good performance on MNIST. Similar CNNs were used commercially in the 1990s. Ranzato et al. (2007) first applied BP to max-pooling CNNs (MPCNNs); advantages of doing this were pointed out subsequently (Scherer et al. 2010).

Efficient parallelized GPU-based MPCNNs (Ciresan et al. 2011) further improved the MNIST record dramatically, achieving human performance (around 0.2 %) for the first time (Ciresan et al. 2012c). To detect human actions in surveillance videos, a three-dimensional CNN, combined with support vector machines, was part of a larger system using a bag of features approach to extract regions of interest. The system won three 2009 TRECVID competitions. These were possibly the first official international contests won with the help of (MP)CNNs; compare (Ji et al. 2013).

In 2011, an ensemble (Breiman 1996; Schapire 1990) of GPU-based MPCNNs also was the first system to achieve superhuman visual pattern recognition in a controlled competition, namely, the IJCNN 2011 traffic sign recognition contest in Silicon Valley (Ciresan et al. 2012c). The system was twice better than humans and three times better than the nearest nonhuman competitor. Subsequently, similar committees of GPU-MPCNNs became widely used and also won the 2012 ImageNet classification contest (Krizhevsky et al. 2012), which is popular in the computer vision community. Further progress on ImageNet was achieved through variants of such systems (e.g., Zeiler and Fergus 2013; Szegedy et al. 2014; Simonyan and Zisserman 2015).

In 2012, a GPU-MPCNN committee also was the first deep learning NN to win a contest on visual object discovery in large images (Ciresan et al. 2013), namely, the ICPR 2012 Contest on Mitosis Detection in Breast Cancer Histological Images. Here deep MPCNNs are trained on labelled patches of big images and then used as feature detectors to be shifted across unknown visual scenes, using various rotations and zoom

factors. Image parts that yield highly active output units are likely to contain objects similar to those the NN was trained on. A similar GPU-MPCNN committee was the first deep learner to win a pure image segmentation contest (Ciresan et al. 2012a), namely, the ISBI 2012 segmentation of neuronal structures in EM stacks challenge. The MPCNN learned to predict for each pixel whether it belongs to the background. Fast MPCNN image scanners avoid redundant computations and speed up naive implementations by up to three orders of magnitude (Masci et al. 2013), extending earlier efficient methods for CNNs without MP (Vaillant et al. 1994).

It is fair to say that deep GPU-CNNs have revolutionized computer vision. For example, GPU-MPCNNs helped to recognize multi-digit numbers in Google Street View images (Goodfellow et al. 2014b), where part of the NN was trained to count visible digits. Other successful recent CNN applications include scene parsing (Farabet et al. 2013), shadow detection (Khan et al. 2014), and video classification (Karpathy et al. 2014), to name a few.

Fundamental Deep Learning Problem and Unsupervised Pre-training of RNNs and FNNs

There are extensions of backpropagation (BP) for supervised RNNs (e.g., Williams 1989; Robinson and Fallside 1987; Werbos 1988). During training by "BP through time" (BPTT), the RNN is "unfolded" into an FNN that has essentially as many layers as there are time steps in the observed sequence of input vectors.

The drawbacks of BP and BPTT became obvious in 1991, when the vanishing/exploding gradient problem or "Fundamental Deep Learning Problem" was identified and analyzed (Hochreiter 1991): With standard activation functions, cumulative backpropagated error signals either shrink exponentially in the number of layers (or time steps) or grow out of bounds. The problem is most apparent in RNNs, the deepest of all NNs.

To some extent, Hessian-free optimization can alleviate the problem for FNNs (Moller 1993; Pearlmutter 1994) and RNNs (Martens and Sutskever 2011).

To overcome the vanishing gradient problem, an early generative model was proposed, namely, an unsupervised stack of RNNs called the neural history compressor (Schmidhuber 1992b). A first RNN uses unsupervised learning to predict its next input. Each higher level RNN tries to learn a compressed representation of the info in the RNN below, trying to minimize the description length (or negative log probability) of the data. The top RNN may then find it easy to classify the data by supervised learning. One can also "distill" the knowledge of a higher RNN (the teacher) into a lower RNN (the student) by forcing the lower RNN to predict the hidden units of the higher one. In the early 1990s, such systems could solve previously unsolvable "very deep learning" tasks involving hundreds of subsequent computational stages.

A conceptually very similar but FNN-based system was the deep belief network (DBN, Hinton and Salakhutdinov 2006), a stack of restricted Boltzmann machines (RBMs, Smolensky 1986) with a single layer of feature-detecting units. They can be trained by the contrastive divergence algorithm (Hinton 2002). At least in theory under certain assumptions, adding more layers improves a bound on the data's negative log probability (Hinton et al. 2006), equivalent to the data's description length – just like with the RNN history compressor above. A GPU-DBN implementation (Raina et al. 2009) was orders of magnitudes faster than previous CPU-DBNs; see also Coates et al. (2013). DBNs achieved good results on phoneme recognition (Mohamed and Hinton 2010). Autoencoder stacks (Ballard 1987) became a popular alternative way of pre-training deep FNNs in unsupervised fashion, before fine-tuning them through BP (e.g., Bengio et al. 2007).

Generally speaking, unsupervised learning (UL) can help to encode input data in a form advantageous for further processing. For example, FNNs may profit from pre-training by competitive UL prior to BP-based fine-tuning

(Maclin and Shavlik 1995). Many UL methods generate distributed, sparse representations of input patterns. Ideally, given an ensemble of input patterns, redundancy reduction through a deep NN will create a factorial code (a code with statistically independent components) of the ensemble (Barlow et al. 1989). Such codes may be sparse and can be advantageous for (1) data compression, (2) speeding up subsequent BP, and (3) trivializing the task of subsequent naive yet optimal Bayes classifiers. Methods for deep UL FNNs include hierarchical self-organizing Kohonen maps (e.g., Koikkalainen and Oja 1990), hierarchical Gaussian potential function networks (Lee and Kil 1991), layer-wise UL of feature hierarchies fed into SL classifiers (Behnke 1999), the self-organizing tree algorithm (Herrero et al. 2001), and nonlinear autoencoders (AEs) with five or more layers (e.g., Kramer 1991). Predictability minimization (Schmidhuber 1992c) searches for factorial codes through nonlinear feature detectors that fight nonlinear predictors, trying to become both as informative and as unpredictable as possible. Hierarchical CNNs in a Neural Abstraction Pyramid (e.g., Behnke 2003b) can be trained to reconstruct images corrupted by structured noise, thus enforcing increasingly abstract image representations in deeper and deeper layers.

In many applications of the 2000s, however, DBNs and other unsupervised methods were largely replaced by purely supervised FNNs, especially MPCNNs (see above). Here history repeated itself, because already in the 1990s, unsupervised RNN-based history compressors (see above) were largely replaced by purely supervised LSTM RNNs (see below).

Very Deep Learning in Supervised Sequence-Processing RNNs

Supervised long short-term memory (LSTM) RNNs have been developed since the 1990s (e.g., Hochreiter and Schmidhuber 1997b; Gers and Schmidhuber 2001; Graves et al. 2009). Parts of LSTM RNNs are designed such that backpropagated errors can neither vanish nor explode but flow backward in "civilized" fashion for thousands or even more steps. Thus, LSTM variants could learn previously unlearnable very deep learning tasks (including some unlearnable by the 1992 history compressor above) that require to discover the importance of (and memorize) events that happened thousands of discrete time steps ago, while previous standard RNNs already failed in case of minimal time lags of ten steps. It is possible to evolve good problem-specific LSTM-like topologies (Bayer et al. 2009).

Recursive NNs (Goller and Küchler 1996) generalize RNNs, by operating on hierarchical structures, recursively combining child representations into parent representations. Bidirectional RNNs (BRNNs) (Schuster and Paliwal 1997) are designed for input sequences whose starts and ends are known in advance, such as spoken sentences to be labeled by their phonemes. DAG-RNNs (Baldi and Pollastri 2003) generalize BRNNs to multiple dimensions. Recursive NNs, BRNNs, and DAG-RNNs unfold their full potential when combined with LSTM (Graves et al. 2009).

Particularly successful in competitions were stacks of LSTM RNNs (Fernandez et al. 2007b) trained by connectionist temporal classification (CTC, Graves et al. 2006), a gradient-based method for finding RNN weights that maximize the probability of teacher-given label sequences, given (typically much longer and more high-dimensional) streams of real-valued input vectors. CTC performs simultaneous segmentation (alignment) and recognition. In 2009, CTC-trained LSTM became the first RNN to win controlled international contests, namely, three competitions in connected handwriting recognition. Hannun et al. (2014) used CTC-trained RNNs to break a famous speech recognition benchmark record, without using any traditional speech processing methods such as hidden Markov models (HMMs) or Gaussian mixture models.

Unlike HMMs and previous RNNs, LSTM can learn to recognize context-sensitive languages. By 2007, LSTM had started to revolutionize speech recognition, outperforming traditional

HMMs in keyword spotting tasks (Fernandez et al. 2007b). By 2013, LSTM achieved best known results on the famous TIMIT phoneme recognition benchmark (Graves et al. 2013). Hybrids of traditional methods and LSTM RNNs obtained best known performance on large-vocabulary speech recognition (Sak et al.; Google 2014a; Li and Wu 2015). LSTM also helped to improve the state of the art in numerous other fields, including image caption generation (in conjunction with CNNs) (Vinyals et al.; Google 2014a), machine translation (Sutskever et al.; Google 2014), text-to-speech synthesis (Fan et al. 2015; Zen and Sak 2015, now available for Google Android), photo-real talking heads (Fan et al.; Microsoft 2015), syntactic parsing for natural language processing (Vinyals et al.; Google, 2014b), and many other applications. In 2015, CTC-trained LSTM dramatically improved Google Voice (by 49 %) and is now available to a billion smartphone users (Sak et al. 2015).

Gradient-based LSTM is no panacea though. Other methods sometimes outperformed LSTM at least on certain tasks (e.g., Jaeger 2004; Schmidhuber et al. 2007; Martens and Sutskever 2011; Zimmermann et al. 2012; Pascanu et al. 2013b; Koutnik et al. 2014). Several alternative RNN-related methods with fast memory control have been proposed over the decades (e.g., AMAmemory 2015).

Some Tricks to Improve NNs

BP-like methods can be used to search for "simple," low-complexity NNs with high generalization capability. For example, weight decay (e.g., Hanson and Pratt 1989) encourages near-zero weights, by penalizing large weights. Related weight priors are implicit in additional penalty terms (MacKay 1992) or in methods based on validation sets (e.g., Hastie and Tibshirani 1990). Similar priors (or biases towards simplicity) are implicit in constructive and pruning algorithms, e.g., layer-by-layer sequential network construction (e.g., Ivakhnenko 1971), input pruning (Moody 1992), unit pruning (e.g., Ivakhnenko 1971; Mozer and Smolensky 1989), weight pruning (e.g., LeCun et al. 1990b), fast and short weight matrix-computing programs (Schmidhuber 1997), and flat minimum search (FMS, Hochreiter and Schmidhuber 1999). DBN training can be improved (Cho et al. 2012) through Tikhonov-type regularization (Tikhonov et al. 1977). See also sparsity-enforcing methods mentioned earlier.

Dropout (Hinton et al. 2012b) removes units from NNs during training to improve generalization. It is closely related to older, biologically plausible techniques for adding noise to neurons or synapses during training (e.g., Hanson 1990). NNs with competing units (e.g., Schmidhuber 1989b; Maass 2000; Goodfellow et al. 2013) tend to outperform those with noncompeting units and avoid catastrophic forgetting through BP when training sets change over time (Srivastava et al. 2013).

The popular activation function f of rectified linear units (ReLUs) is $f(x) = x$ for $x > 0$; $f(x) = 0$ otherwise. ReLU NNs are useful for RBMs (Nair and Hinton 2010; Maas et al. 2013), outperformed sigmoidal activation functions in deep NNs (Glorot et al. 2011), and helped to obtain best results on several benchmark problems across multiple domains (e.g., Krizhevsky et al. 2012).

Many additional tricks for improving NNs have been described (e.g., Montavon et al. 2012; Schmidhuber 2015).

Consequences for Neuroscience

Artificial NNs (ANNs) can help to better understand biological NNs (BNNs). The feature detectors learned by single-layer visual ANNs are similar to those found in early visual processing stages of BNNs. Likewise, the feature detectors learned in deep layers of visual ANNs should be highly predictive of what neuroscientists will find in deep layers of BNNs. While the visual cortex of BNNs may use quite different learning algorithms, its objective function to be minimized may be rather similar to the one of visual ANNs. In fact, results obtained with relatively deep artificial NNs (e.g., Yamins et al. 2013)

seem compatible with insights about the visual pathway in the primate cerebral cortex, which has been studied for many decades.

Deep Learning with Spiking Neurons?

Current deep NNs greatly profit from GPUs, which are little ovens, much hungrier for energy than biological brains, whose neurons efficiently communicate by brief spikes (e.g., Hodgkin and Huxley 1952) and often remain quiet. Many computational models of such spiking neurons have been proposed and analyzed (e.g., Gerstner and Kistler 2002). Future energy-efficient hardware for DL in NNs may implement aspects of such models – see numerous references in the survey (Schmidhuber 2015, Sect. 5.26). In practical applications, however, current artificial networks of spiking neurons cannot yet compete with the best traditional deep NNs.

Deep Reinforcement Learning (RL)

Reinforcement learning (RL) is the most general type of learning. General RL agents must discover, without the aid of a teacher, how to interact with a dynamic, initially unknown, partially observable environment in order to maximize their expected cumulative reward signals (e.g., Kaelbling et al. 1996; Sutton and Barto 1998; Wiering and van Otterlo 2012). There may be arbitrary, a priori unknown delays between actions and perceivable consequences. The RL problem is as hard as any problem of computer science, since any task with a computable description can be formulated in the general RL framework (e.g., Hutter 2005). Deep FNNs and RNNs are useful tools for various types of RL. Many references on this since the 1980s can be found in the recent survey (Schmidhuber 2015, Sect. 6).

Outlook

Deep learning in NNs is more than a temporary fad. Physics seems to dictate that any future efficient computational hardware will have to be brain-like, with many compactly placed processors in three-dimensional space, sparsely connected by many short and few long wires, to minimize total connection cost (even if the "wires" are actually light beams). The basic architecture is essentially the one of a deep, sparsely connected, three-dimensional RNN, and deep learning methods for such RNNs are expected to become even much more important than they are today.

The contents of this article may be used for educational and noncommercial purposes, including articles for Wikipedia and similar sites.

Recommended Reading

Aizenberg I, Aizenberg NN, Vandewalle JPL (2000) Multi-valued and universal binary neurons: theory, learning and applications. Springer, Boston. First work to introduce the term "Deep Learning" to Neural Networks

AMAmemory (2015) Answer at reddit AMA (Ask Me Anything) on "memory networks" etc (with references) http://www.reddit.com/r/MachineLearning/comments/2xcyrl/i_am_j%C3%BCrgen_schmidhuber_ama/cp0q12t

Amari S-I (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276

Baird H (1990) Document image defect models. In: Proceedings of IAPR workshop on syntactic and structural pattern recognition, Murray Hill

Baldi P, Pollastri G (2003) The principled design of large-scale recursive neural network architectures – DAG-RNNs and the protein structure prediction problem. J Mach Learn Res 4:575–602

Ballard DH (1987) Modular learning in neural networks. In: Proceedings of AAAI, Seattle, pp 279–284

Barlow HB, Kaushal TP, Mitchison GJ (1989) Finding minimum entropy codes. Neural Comput 1(3):412–423

Bayer J, Wierstra D, Togelius J, Schmidhuber J (2009) Evolving memory cell structures for sequence learning. In: Proceedings of ICANN, vol 2. Springer, Berlin/New York, pp 755–764

Behnke S (1999) Hebbian learning and competition in the neural abstraction pyramid. In: Proceedings of IJCNN, vol 2. Washington, pp 1356–1361

Behnke S (2003) Hierarchical neural networks for image interpretation. Lecture notes in computer science, vol LNCS 2766. Springer, Berlin/New York

Bengio Y, Lamblin P, Popovici D, Larochelle H (2007) Greedy layer-wise training of deep networks. In:

Cowan JD, Tesauro G, Alspector J (eds) Proceedings of NIPS 19, MIT Press, Cambridge, pp 153–160

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Bryson AE (1961) A gradient method for optimizing multi-stage allocation processes. In: Proceedings of Harvard university symposium on digital computers and their applications, Harvard University Press, Cambridge

Bryson A, Ho Y (1969) Applied optimal control: optimization, estimation, and control. Blaisdell Publishing Company, Washington

Cho K, Ilin A, Raiko T (2012) Tikhonov-type regularization for restricted Boltzmann machines. In: Proceedings of ICANN 2012, Springer, Berlin/New York, pp 81–88

Ciresan DC, Meier U, Gambardella LM, Schmidhuber J (2010) Deep big simple neural nets for handwritten digit recogntion. Neural Comput 22(12):3207–3220

Ciresan DC, Meier U, Masci J, Gambardella LM, Schmidhuber J (2011) Flexible, high performance convolutional neural networks for image classification. In: Proceedings of IJCAI, pp 1237–1242

Ciresan DC, Giusti A, Gambardella LM, Schmidhuber J (2012a) Deep neural networks segment neuronal membranes in electron microscopy images. In: Proceedings of NIPS, Quebec City, pp 2852–2860

Ciresan DC, Meier U, Masci J, Schmidhuber J (2012b) Multi-column deep neural network for traffic sign classification. Neural Netw 32:333–338

Ciresan DC, Meier U, Schmidhuber J (2012c) Multi-column deep neural networks for image classification. In: Proceedings of CVPR 2012, Long preprint. arXiv:1202.2745v1 [cs.CV]

Ciresan DC, Giusti A, Gambardella LM, Schmidhuber J (2013) Mitosis detection in breast cancer histology images with deep neural networks. In: Proceedings of MICCAI, vol 2. Nagoya, pp 411–418

Coates A, Huval B, Wang T, Wu DJ, Ng AY, Catanzaro, B (2013) Deep learning with COTS HPC systems. In: Proceedings of ICML'13

Dechter R (1986) Learning while searching in constraint-satisfaction problems. University of California, Computer Science Department, Cognitive Systems Laboratory. First paper to introduce the term "Deep Learning" to Machine Learning; compare a popular G+ post on this. https://plus.google.com/100849856540000067209/posts/7N6z251w2Wd?pid=6127540521703625346&oid=100849856540000067209

Dreyfus SE (1962) The numerical solution of variational problems. J Math Anal Appl 5(1):30–45

Dreyfus SE (1973) The computational solution of optimal control problems with time lag. IEEE Trans Autom Control 18(4):383–385

Fan B, Wang L, Soong FK, Xie L (2015) Photo-real talking head with deep bidirectional LSTM. In: Proceedings of ICASSP 2015, Brisbane

Farabet C, Couprie C, Najman L, LeCun Y (2013) Learning hierarchical features for scene labeling. IEEE Trans Pattern Anal Mach Intell 35(8):1915–1929

Fernandez S, Graves A, Schmidhuber J (2007a) An application of recurrent neural networks to discriminative keyword spotting. In: Proceedings of ICANN, vol 2. pp 220–229

Fernandez S, Graves A, Schmidhuber J (2007b) Sequence labelling in structured domains with hierarchical recurrent neural networks. In: Proceedings of IJCAI

Fu KS (1977) Syntactic pattern recognition and applications. Springer, Berlin

Fukushima K (1979) Neural network model for a mechanism of pattern recognition unaffected by shift in position – neocognitron. Trans. IECE J62-A(10):658–665

Gers FA, Schmidhuber J (2001) LSTM recurrent networks learn simple context free and context sensitive languages. IEEE Trans Neural Netw 12(6):1333–1340

Gerstner W, Kistler WK (2002) Spiking neuron models. Cambridge University Press, Cambridge

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier networks. In: Proceedings of AISTATS, vol 15. Fort Lauderdale, pp 315–323

Goodfellow IJ, Warde-Farley D, Mirza M, Courville A, Bengio Y (2013) Maxout networks. In: Proceedings of ICML, Atlanta

Goodfellow IJ, Bulatov Y, Ibarz J, Arnoud S, Shet V (2014b) Multi-digit number recognition from street view imagery using deep convolutional neural networks. arXiv preprint arXiv:1312.6082 v4

Goller C, Küchler A (1996) Learning task-dependent distributed representations by backpropagation through structure. In: IEEE international conference on neural networks 1996, vol 1, pp 347–352

Graves A, Fernandez S, Gomez FJ, Schmidhuber J(2006) Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural nets. In: Proceedings of ICML'06, Pittsburgh, pp 369–376

Graves A, Liwicki M, Fernandez S, Bertolami R, Bunke H, Schmidhuber J (2009) A novel connectionist system for improved unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 31(5):855–868

Graves A, Mohamed A-R, Hinton GE (2013) Speech recognition with deep recurrent neural networks. In: Proceedings of ICASSP, Vancouver, pp 6645–6649

Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY (2014) Deep speech: scaling up end-to-end speech recognition. arXiv preprint http://arxiv.org/abs/1412.5567

Hanson SJ, Pratt LY (1989) Comparing biases for minimal network construction with back-propagation. In: Touretzky DS (ed) Proceedings of NIPS, vol 1. Morgan Kaufmann, San Mateo, pp 177–185

Hanson SJ (1990) A stochastic version of the delta rule. Phys D: Nonlinear Phenom 42(1):265–272
Hastie TJ, Tibshirani RJ (1990) Generalized additive models, vol 43. CRC Press
Hebb DO (1949) The organization of behavior. Wiley, New York
Herrero J, Valencia A, Dopazo J (2001) A hierarchical unsupervised growing neural network for clustering gene expression patterns. Bioinformatics 17(2):126–136
Hinton G, Salakhutdinov R (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14(8):1771–1800
Hinton GE, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012b) Improving neural networks by preventing co-adaptation of feature detectors. Technical report. arXiv:1207.0580
Hochreiter S (1991) Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut fuer Informatik, Lehrstuhl Prof. Brauer, Tech. Univ. Munich. Advisor: J. Schmidhuber
Hochreiter S, Schmidhuber J (1997a) Flat minima. Neural Comput 9(1):1–42
Hochreiter S, Schmidhuber J (1997b) Long short-term memory. Neural Comput 9(8):1735–1780. Based on TR FKI-207-95, TUM (1995)
Hochreiter S, Schmidhuber J (1999) Feature extraction through LOCOCODE. Neural Comput 11(3):679–714
Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500
Hutter M (2005) Universal artificial intelligence: sequential decisions based on algorithmic probability. Springer, Berlin
Ivakhnenko AG, Lapa VG (1965) Cybernetic Predicting Devices. CCM Information Corporation, New York
Ivakhnenko AG (1971) Polynomial theory of complex systems. IEEE Trans Syst Man Cybern (4):364–378
Jaeger H (2004) Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304:78–80
Ji S, Xu W, Yang M, Yu K (2013) 3D convolutional neural networks for human action recognition. IEEE Trans Pattern Anal Mach Intell 35(1):221–231
Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J AI Res 4:237–285
Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L (2014) Large-scale video classification with convolutional neural networks. In: Proceedings of CVPR, Columbus
Kelley HJ (1960) Gradient theory of optimal flight paths. ARS J 30(10):947–954
Khan SH, Bennamoun M, Sohel F, Togneri R (2014) Automatic feature learning for robust shadow detection. In: Proceedings of CVPR, Columbus
Koikkalainen P and Oja E (1990) Self-organizing hierarchical feature maps. In: Proceedings of IJCNN, pp 279–284
Koutnik J, Greff K, Gomez F, Schmidhuber J (2014) A Clockwork RNN. In: Proceedings of ICML, vol 32. pp 1845–1853. arXiv:1402.3511 [cs.NE]
Kramer M (1991) Nonlinear principal component analysis using autoassociative neural networks. AIChE J 37:233–243
Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of NIPS, Nevada, p 4
LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
LeCun Y, Denker JS, Solla SA (1990b) Optimal brain damage. In: Touretzky DS (ed) Proceedings of NIPS 2, Morgan Kaufmann, San Mateo, pp 598–605
LeCun Y, Bengio Y, Hinton G (2015) Deep Learning. Nature 521:436–444. Link. See critique by J. Schmidhuber (2015) http://people.idsia.ch/~juergen/deep-learning-conspiracy.html
Lee S, Kil RM (1991) A Gaussian potential function network with hierarchically selforganizing learning. Neural Netw 4(2):207–224
Li X, Wu X (2015) Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In: Proceedings of ICASSP 2015. http://arxiv.org/abs/1410.4281
Linnainmaa S (1970) The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, University of Helsinki
Linnainmaa S (1976) Taylor expansion of the accumulated rounding error. BIT Numer Math 16(2):146–160
Maas AL, Hannun AY, Ng AY (2013) Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of ICML, Atlanta
Maass W (2000) On the computational power of winner-take-all. Neural Comput 12:2519–2535
MacKay, DJC (1992) A practical Bayesian framework for backprop networks. Neural Comput 4:448–472
Maclin R, Shavlik JW (1995) Combining the predictions of multiple classifiers: using competitive learning to initialize neural networks. In: Proceedings of IJCAI, pp 524–531
Martens J, Sutskever I (2011) Learning recurrent neural networks with Hessian-free optimization. In: Proceedings of ICML, pp 1033–1040
Masci J, Giusti A, Ciresan DC, Fricout G, Schmidhuber J (2013) A fast learning algorithm for image segmentation with max-pooling convolutional networks. In: Proceedings of ICIP13, pp 2713–2717

McCulloch W, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 7:115–133
Mohamed A, Hinton GE (2010) Phone recognition using restricted Boltzmann machines. In: Proceedings of ICASSP, Dallas, pp 4354–4357
Moller MF (1993) Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in O(N) time. Technical report PB-432, Computer Science Department, Aarhus University
Montavon G, Orr G, Mueller K (2012) Neural networks: tricks of the trade. Lecture notes in computer science, vol LNCS 7700. Springer, Berlin/Heidelberg
Moody JE (1992) The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In: Proceedings of NIPS'4, Morgan Kaufmann, San Mateo, pp 847–854
Mozer MC, Smolensky P (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. In: Proceedings of NIPS 1, Morgan Kaufmann, San Mateo, pp 107–115
Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of ICML, Dallas
Oh K-S, Jung K (2004) GPU implementation of neural networks. Pattern Recognit 37(6):1311–1314
Pascanu R, Mikolov T, Bengio Y (2013b) On the difficulty of training recurrent neural networks. In: ICML'13: JMLR: W&CP, vol 28
Pearlmutter BA (1994) Fast exact multiplication by the Hessian. Neural Comput 6(1):147–160
Raina R, Madhavan A, Ng A (2009) Large-scale deep unsupervised learning using graphics processors. In: Proceedings of ICML, Montreal, pp 873–880
Ranzato MA, Huang F, Boureau Y, LeCun Y (2007) Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: Proceedings of CVPR, Minneapolis, pp 1–8
Robinson AJ, Fallside F (1987) The utility driven dynamic error propagation network. Technical report CUED/F-INFENG/TR.1, Cambridge University Engineering Department
Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386
Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Rumelhart DE, McClelland JL (eds) Parallel distributed processing, vol 1, MIT Press, Cambridge, pp 318–362
Sak H, Senior AW, Beaufays F (2014) Long short-term memory recurrent neural network architectures for large scale acoustic modeling. INTERSPEECH
Sak H, Senior A, Rao K, Beaufays F, Schalkwyk J (2015) Google research blog. http://googleresearch.blogspot.ch/2015/09/google-voice-search-faster-and-more.html
Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
Scherer D, Mueller A, Behnke S (2010) Evaluation of pooling operations in convolutional architectures for object recognition. In: Proceedings of ICANN, Thessaloniki, pp 92–101
Schmidhuber J (1989b) A local learning algorithm for dynamic feedforward and recurrent networks. Connect Sci 1(4):403–412
Schmidhuber J (1992b) Learning complex, extended sequences using the principle of history compression. Neural Comput 4(2):234–242. Based on TR FKI-148-91, TUM, 1991
Schmidhuber J (1992c) Learning factorial codes by predictability minimization. Neural Comput 4(6):863–879
Schmidhuber J (1997) Discovering neural nets with low Kolmogorov complexity and high generalization capability. Neural Netw 10(5):857–873
Schmidhuber J, Wierstra D, Gagliolo M, Gomez FJ (2007) Training recurrent networks by Evolino. Neural Comput 19(3):757–779
Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117. arXiv preprint 1404.7828
Schmidhuber J (2015) Deep learning. Scholarpedia 10(11):32832
Schuster M, Paliwal KK (1997) Bidirectional recurrent neural networks. IEEE Trans Signal Process 45:2673–2681
Sima J (1994) Loading deep networks is hard. Neural Comput 6(5):842–850
Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. arXiv preprint http://arxiv.org/abs/1409.1556
Smolensky P (1986) Parallel distributed processing: explorations in the microstructure of cognition, chapter information processing in dynamical systems: foundations of Harmony theory, vol 1. MIT Press, Cambridge, pp 194–281
Speelpenning B (1980) Compiling fast partial derivatives of functions given by algorithms. Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana-Champaign
Srivastava RK, Masci J, Kazerounian S, Gomez F, Schmidhuber J (2013) Compete to compute. In: Proceedings of NIPS, Nevada, pp 2310–2318
Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Proceedings of NIPS'2014. arXiv preprint arXiv:1409.3215 [cs.CL]
Sutton R, Barto A (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014) Going deeper with convolutions. arXiv preprint arXiv:1409.4842 [cs.CV]
Tikhonov AN, Arsenin VI, John F (1977) Solutions of ill-posed problems. Winston, New York

Vaillant R, Monrocq C, LeCun Y (1994) Original approach for the localisation of objects in images. IEE Proc Vision Image Signal Process 141(4):245–250
Vieira A, Barradas N (2003) A training algorithm for classification of high-dimensional data. Neurocomputing 50:461–472
Vinyals O, Toshev A, Bengio S, Erhan D (2014a) Show and tell: a neural image caption generator. arXiv Preprint http://arxiv.org/pdf/1411.4555v1.pdf
Vinyals O, Kaiser L, Koo T, Petrov S, Sutskever I, Hinton G (2014b) Grammar as a foreign language. Preprint http://arxiv.org/abs/1412.7449
Wan EA (1994) Time series prediction by using a connectionist network with internal delay lines. In: Weigend AS, Gershenfeld NA (eds) Time series prediction: forecasting the future and understanding the past. Addison-Wesley, Reading, pp 265–295
Weng JJ, Ahuja N, Huang TS (1993) Learning recognition and segmentation of 3-d objects from 2-d images. Proceedings of the fourth international conference on computer vision. IEEE
Williams RJ (1989) Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report NU-CCS-89-27, Northeastern University, College of Computer Science, Boston
Wiering M, van Otterlo M (2012) Reinforcement learning. Springer, Berlin/Heidelberg
Werbos PJ (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University
Werbos PJ (1982) Applications of advances in nonlinear sensitivity analysis. In: Proceedings of the 10th IFIP conference, 31.8–4.9, NYC, pp 762–770
Werbos PJ (1988) Generalization of backpropagation with application to a recurrent gas market model. Neural Netw 1(4):339–356
Yamins D, Hong H, Cadieu C, DiCarlo JJ (2013) Hierarchical modular optimization of convolutional networks achieves representations similar to macaque IT and human ventral stream. In: Proceedings of NIPS, Nevada, pp 1–9
Zeiler MD, Fergus R (2013) Visualizing and understanding convolutional networks. Technical report arXiv:1311.2901 [cs.CV], NYU
Zen H, Sak H (2015) Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. In: Proceedings of ICASSP, Brisbane, pp 4470–4474
Zimmermann H-G, Tietz C, Grothmann R (2012) Forecasting with recurrent neural networks: 12 tricks. In: Montavon G, Orr GB, Mueller K-R (eds) Neural networks: tricks of the trade, 2nd edn. Lecture Notes in Computer Science, vol 7700. Springer, Berlin/New York, pp 687–707

Density Estimation

Claude Sammut
The University of New South Wales, Sydney, NSW, 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 et al. (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:

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

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
- ▸ Mean Shift
- ▸ Mixture Model
- ▸ Nearest Neighbor
- ▸ Support Vector Machines

Recommended Reading

Kernel Density estimation is well covered in texts including Hastie et al. (2009), Duda et al. (2001) and Ripley (1996)

Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd edn. Wiley, New York

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

Parzen E (1962) On the estimation of a probability density function and the mode. Ann Math Stat 33:1065–1076

Ripley BD (1996) Pattern recognition and neural networks. Cambridge University Press, Cambridge

Density-Based Clustering

Joerg Sander
University of Alberta, Edmonton, AB, Canada
Statistical Machine Learning Group, NICTA, Canberra, ACT, Australia

Abstract

The chapter gives a concise explanation of the basic principles of density-based clustering and points out important ”milestone papers” in this area.

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

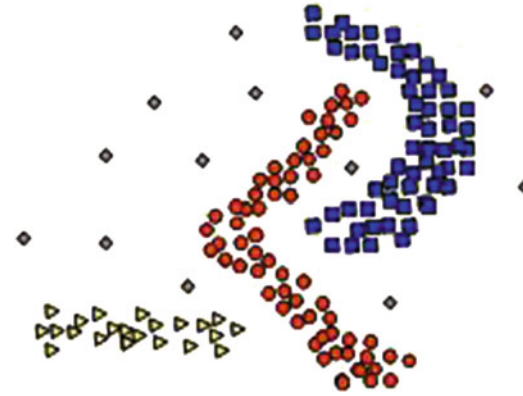
Clustering in general is an unsupervised learning task that aims at finding distinct groups in data, called “clusters.” The minimum requirements for this task are that the data is given as some set of objects O for which a dissimilarity-distance

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 characterizing what establishes 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 the within-cluster dissimilarities are minimized, while the between-cluster dissimilarities are maximized. The notions of within-cluster and between-cluster dissimilarity are defined using the given distance function d. Such methods correspond, from a statistical point of view, 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 "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 a rivers, roads, seismic faults, etc. Figure 1 illustrates density-based clusters using 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 have clearly a higher point density than the region between them, and they can easily be separated.



Density-Based Clustering, Fig. 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 which are directly or transitively connected to objects whose density

exceeds some threshold λ. The set $\{x \mid p(x) > \lambda\}$ 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 exceed the threshold λ; 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 the first time by Wishart (1969). His algorithm for *one level mode analysis* consists of six steps: "(a) Select a distance threshold r, and a frequency (or density) threshold k. (b) Compute the triangular similarity matrix of all inter-point distances. (c) Evaluate the frequency k_i of each data point, defined as the number of points which lie within a distance r of point i (. . .). (d) Remove the 'noise' or non-dense points, those for which $k_i < k$. (e) Cluster the remaining dense points ($k_i > k$) by single linkage, forming the mode nuclei. (f) 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 Computing 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 as *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 algorithm that performs well in practical situations when spatial index structures are effective (typically for low- to medium dimensional data), and has quadratic worst case runtime when index structures are not effective (e.g., for high-dimensional data).

DENCLUE (Hinneburg and Keim 1998) proposed a notion of density-based clusters using 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 truncated 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; nonempty 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) report 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 computing 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, that 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, 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 when 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), Stuetzle (2003), and Campello et al. (2013). These hierarchical methods are closely related and are essentially processing and rendering a minimum spanning tree of the data –with edge weights defined in different ways– 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. Recent work (Campello et al. 2013) provides an efficient hierarchical version DBSCAN, called HDBSCAN, which includes a method for automatically extracting a flat partitioning from possibly different levels of a density-based clustering hierarchy,

containing only significant clusters according to a cluster stability measure.

Cross-References

▸ Clustering
▸ Density Estimation

Recommended Reading

Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) OPTICS: ordering points to identify the clustering structure. In: Delis A, Faloutsos C, Ghandeharizadeh S (eds) Proceedings of the 1999 ACM SIGMOD international conference on management of data, Philadelphia

Campello RJGB, Moulavi D, Sander J (2013) Density-based clustering based on hierarchical density estimates. In: Proceedings of the 17th Pacific-Asia conference on knowledge discovery in databases, PAKDD 2013. Lecture notes in computer science, vol 7819, p 160

Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis E, Han J, Fayyad UM (eds) Proceedings of the 2nd international conference on knowledge discovery and data mining, Portland

Hartigan JA (1975) Clustering algorithms. Wiley, New York

Hinneburg A, Keim DA (1998) An efficient approach to clustering in large multimedia databases with noise. In: Agrawal R, Stolorz P (eds) Proceedings of the 4th international conference on knowledge discovery and data mining, New York City

Sander J, Ester M, Kriegel H-P, Xu X (1998) Density-Based clustering in spatial databases: the algorithm GDBSCAN and its applications. Data Min Knowl Discov 2(2):169–194

Stuetzle W (2003) Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. J Classif 20(1):025–047

Wishart D (1969) Mode analysis: a generalization of nearest neighbor which reduces chaining effects. In: Numerical Taxonomy, ed. A. J. Cole, London: Academic Press, 282–311

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.

Diagonal Matrix

▸ K-Way Spectral Clustering

Differential Prediction

▸ Uplift Modeling

Digraphs

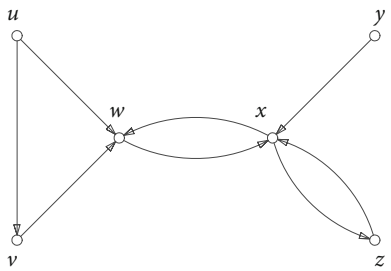
Synonyms

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 $A(D) = \{(u, v), (u, w), (v, w), (w, x), (x, w), (x, z), (y, x),$



Digraphs, Fig. 1 A digraph

$(z, x)\}$. Digraphs can be viewed as generalizations of ▸ graphs.

Dimensionality Reduction

Michail Vlachos
IBM Research, Zurich, Switzerland

Abstract

Dimensionality reduction in an important data pre-processing when dealing with Big Data. We explain how it can be used for speeding up search operation and show applications for time-series datasets.

Synonyms

Feature selection; Feature projection; lossy compression

Introduction

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 interchangeably use 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 (dR) 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 in general; hence, dimensionality reduction can be seen as a method for data compression. In addition, 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 (a) making tractable classification schemes that are superlinear with respect to dimensionality tractable, (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 these techniques is to appropriately select the k dimensions (and also the number k) so that the important characteristics of the original object are retained. For example, when performing dimensionality reduction on an image, e.g., using a wavelet-based technique, the desirable outcome is that the difference between the original and the final images is almost imperceptible.

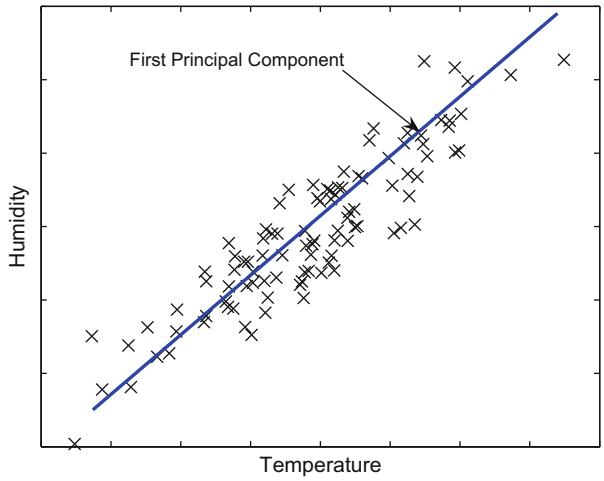
When performing dimensionality reduction not on a single object, but on a dataset, an additional requirement is that the relationship between the objects in the original space be preserved. This is particularly important for reasons of classification and visualization in the new space.

Two important categories of dimensionality reduction techniques exist:

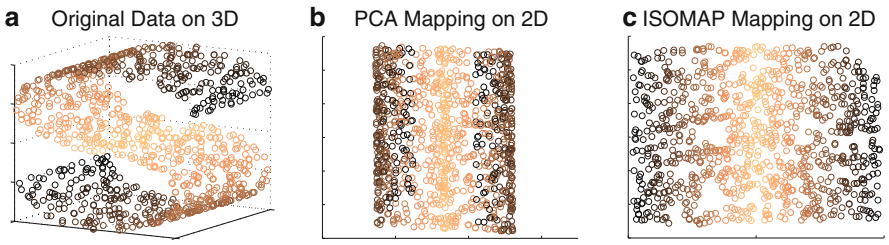
- **Feature selection** techniques, in which only the most important or descriptive features/dimensions are retained, and the rest are discarded. More details on such techniques can be found under the entry ▸ Feature Selection
- **Feature projection** methodologies, which project the existing features onto different dimensions or axes. The aim here is, again, to find those new data axes that retain the dataset structure and preserve 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 component analysis* or PCA. It attempts to discover those axes (or



Dimensionality Reduction, Fig. 1 Principal component analysis



Dimensionality Reduction, Fig. 2 Nonlinear dimensionality reduction techniques produce a better low-dimensional data mapping than PCA if the original data lie on a high-dimensional manifold

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 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 onto 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 objects. The first principal component (or axis) indicates the direction of maximum variance in the original dimensions. The second component shows the direction of the next highest variance (and is uncorrelated to the first component), etc.

Other dimensionality reduction techniques optimize or preserve other criteria than PCA does. Manifold-inspired methods such as 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 illustrated in Fig. 2, where it is also compared with the respective PCA mapping.

Other recent dimensionality reduction techniques include locally linear embedding (LLE) (Roweis and Saul 2000) and Laplacian eigenmaps (Belkin and Niyogi 2002). We also refer the interested practitioner to van der Maaten et al. (2009), for a detailed comparison of various techniques and also for Matlab implemen-

tations 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 offers the following desirable properties:

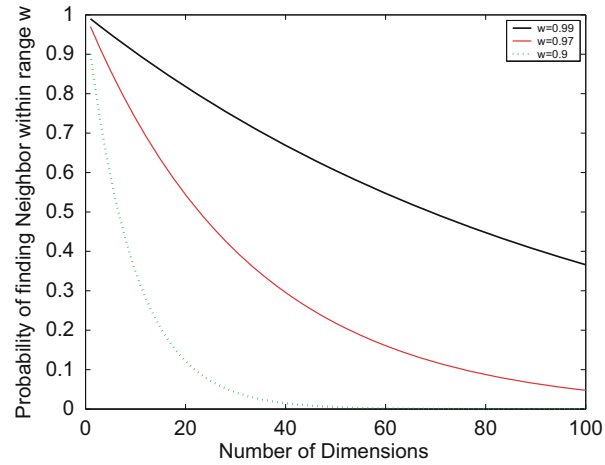
- Data compression: the dataset objects are represented in fewer dimensions, hence saving important disk storage space and offering faster loading of the compressed data from the disk.
- Better data visualization: the relationships between the original high-dimensional objects can be visualized in two- or three-dimensional projections.
- Improved classification accuracy: this can be attributed to both variance reduction and noise removal from the original high-dimensional dataset.
- More efficient data retrieval: dimensionality reduction techniques can also assist in making the retrieval of the original uncompressed data faster and more efficient, by offering very fast prefiltering with the help of the compressed data representation.
- High index performance: more effective use of indexing structures can be achieved by using the compressed data, because 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 the ▸ 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 $P_w(d) = w^d$.

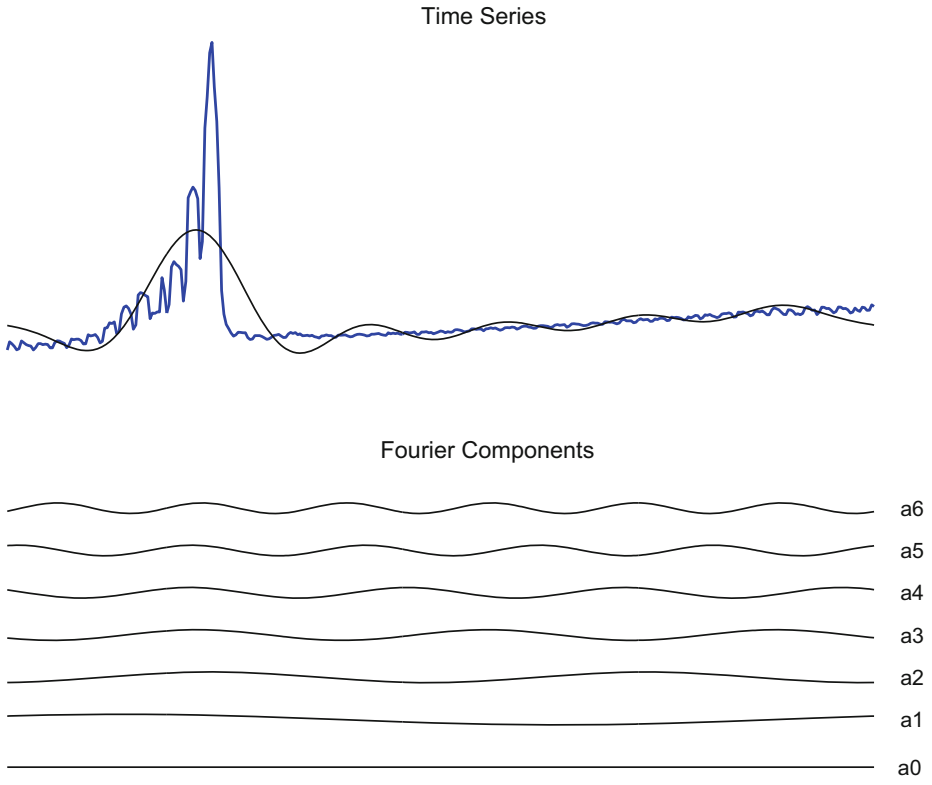
Figure 3 illustrates this probability for various values of w. Evidently, at higher dimensionalities the data becomes very sparse, and even at large radii, only a small portion of the entire space is covered. In simple terms the "curse of dimensionality" translates into the following fact: for large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

Applications: Dimensionality Reduction for Time-Series Data

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



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



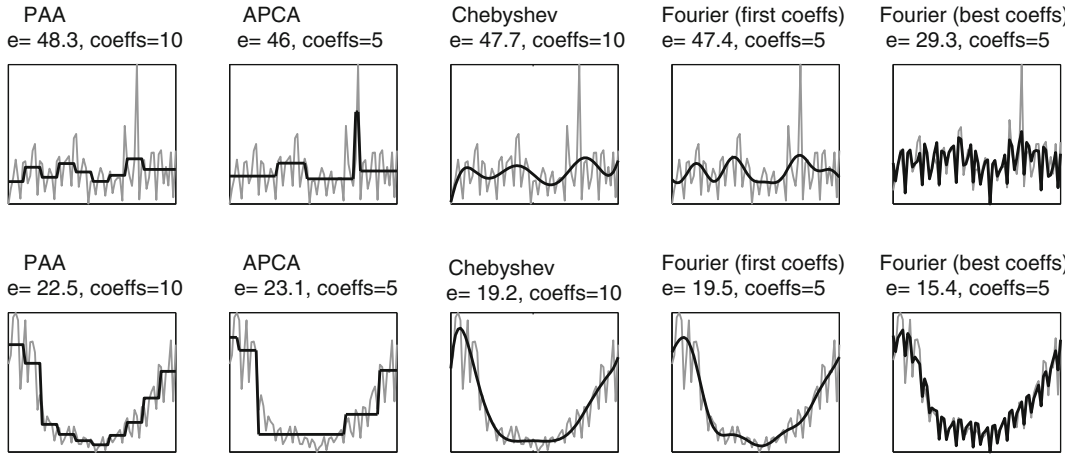
Dimensionality Reduction, Fig. 4 Decomposition of a signal into the first 7 Fourier coefficients. We can see that by using even only few of the Fourier coefficients we can achieve a good reconstruction of the original signal

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

Dimensionality reduction for one- and two-dimensional signals is commonly accomplished using Fourier decomposition. This method for data representation 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 came to the conclusion that every function could be expressed as a sum of trigonometrical series (i.e., sines and cosines). This original work was initially met with doubt (even by famous mathematicians such as Lagrange and Laplace), because of its unexpected result, and moreover, the solution was considered impractical because 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 was lifted. What the Fourier transform attempts to achieve is to represent the original signal as a linear combination of sinusoids. Therefore, each Fourier coefficient is a complex number that essentially encodes the amplitude and the phase of each of these sinusoids, after the original signal has been projected on them.

For most signals, the original sequence can be reconstructed with high accuracy using just few of the coefficients. 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



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

time series that follow the pattern of a random walk, the first few coefficients, which capture the low frequencies of the signal, are sufficient to describe the signal accurately (or, equivalently, to capture most of its energy). Figure 4 depicts a signal of 1024 points and its reconstruction using 7 Fourier coefficients (i.e., using $7 \times 2 = 14$ numbers).

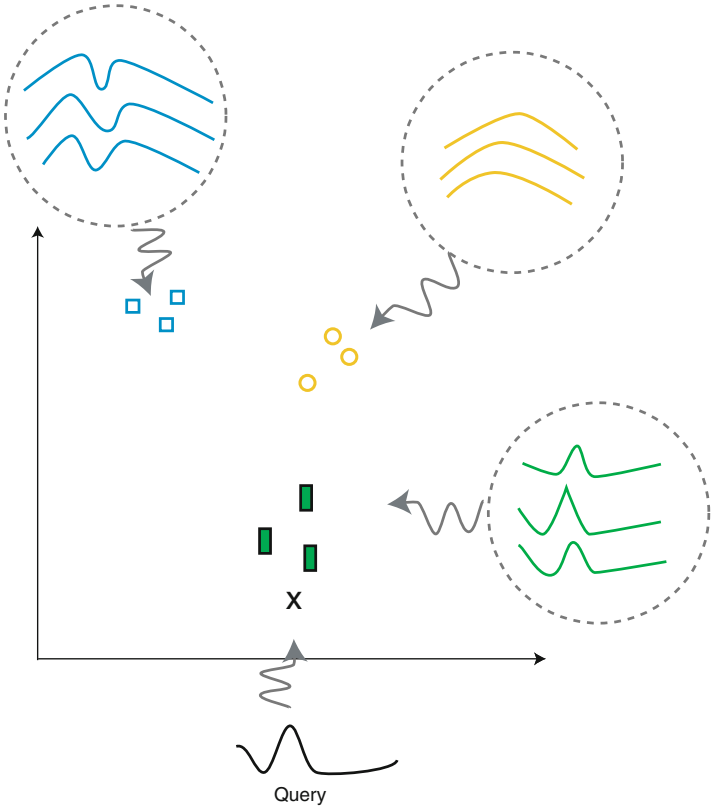
Other popular dimensionality reduction techniques for time-series data are the various wavelet transforms; piecewise linear approximations; piecewise aggregate approximation (PAA), which can be regarded as a projection in time of the wavelet coefficients adaptive piecewise constant approximation (APCA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

No dimensionality reduction technique is universally better than all the others. Depending on the dataset characteristics, one method may provide a better approximation of a dataset than the other techniques. Therefore, the key is to carefully pick the representation that best 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 methods, the same storage space is allocated for the compressed sequences. The time-series reconstruction is shown in a darker color, and the approximation error to the original sequence is also reported. In general, we notice that dimensionality reduction techniques based on selection of the highest energy coefficients 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 illustrates 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. Then, suppose that a user poses another high-dimensional object as query and wishes to find all the objects closest to this query.

To avoid the search on the original high-dimensional space, the query is also transformed into a point in the lower-dimensional space, and its closest matches can be discovered in the vicinity of the projected query point. However,



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

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 the following.

Suppose that we are seeking the 1-Nearest-Neighbor (1-NN) a query Q in a database $\mathcal{D}$. By examining all 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:

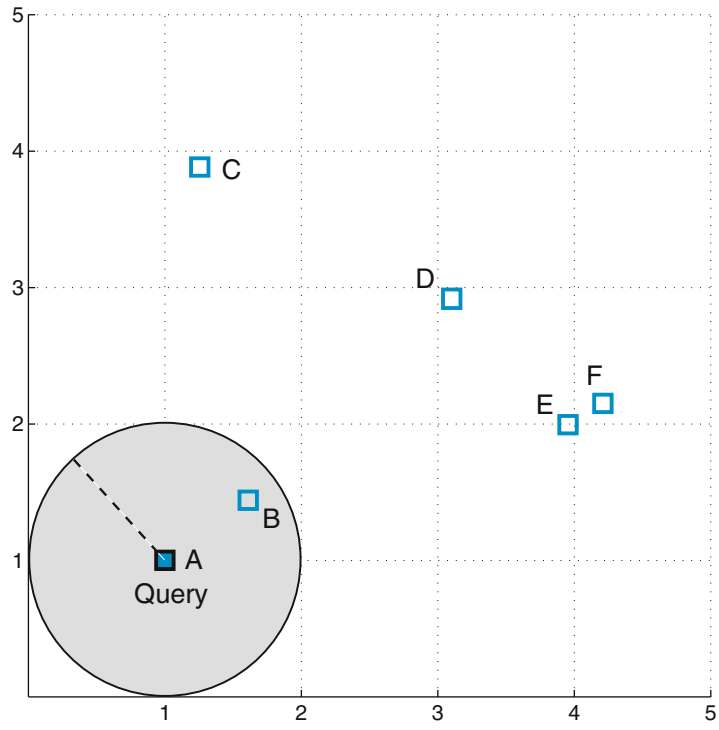
$$\text{Having} \quad A \subset \mathcal{D} \xrightarrow{dR} a \quad \text{and} \quad Q \xrightarrow{dR} q$$

$$\text{then}$$

$$\Delta(q, a) \leq \Delta(Q, A)$$

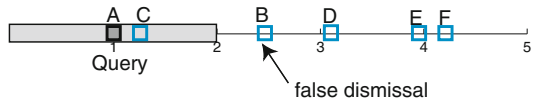
As the computed distance Δ between any two compressed objects is underestimated, *false alarms* may arise. Suppose, for example, that our database consists of 6 two-dimensional points (Fig. 7). If the user query is: "Find everything that lies within a radius of 1 around A," then B is the only result.

Let us assume for a minute that the dimensionality reduction 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*. However, this does not constitute a problem; in a post-processing, in a post-processing phase, the calculation of the exact



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

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



Dimensionality Reduction, Fig. 9 False dismissals may happen when the lower-bounding lemma is not obeyed

distance will remove 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*, because object B lies outside the range of search.

This generic framework for similarity search using dimensionality reduction and lower-bounding distance 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, by using dimensionality reduction for search operations, one can first examine the compressed objects and eliminate many of the uncompressed objects from examination by 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

Recommended Reading

Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: Proceedings of the foundations of data organization and algorithms, Chicago, pp 69–84

Belkin M, Niyogi P (2002) Laplacian eigenmaps and spectral techniques for embedding and clustering. Adv Neural Inf Process Syst 1:585–591

Jolliffee IT (2002) Principal component analysis. 2nd edn. Springer, New York

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

Lin J, Keogh E, Lonardi S, Chiu B (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, San Diego

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

van der Maaten LJP, Postma EO, van den Herik HJ (2009) Dimensionality reduction: a comparative review. Technical report, Tilburg University, TiCC-TR 2009-005

Tenenbaum JB, de Silva V, Langford JC (2000) A global geometric framework for nonlinear dimensionality reduction. Science 290(5500): 2319–2323

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 under-fitting 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 Nonparametric Models.

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 and 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:

$$\begin{array}{ll} \pi|\alpha \sim Dir\left(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}\right) & \theta_k^*|H \sim H \\ z_i|\pi \sim Mult(\pi) & x_i|z_i, \{\theta_k^*\} \sim F\left(\theta_{z_i}^*\right) \end{array} \tag{1}$$

where π is the mixing proportion, α is the pseudocount 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 $A_1, \ldots, A_r$ of Θ the vector $(G(A_1), \ldots, G(A_r))$ 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

$$(G(A_1), \ldots, G(A_r)) \sim Dir(\alpha H(A_1), \ldots, \alpha H(A_r)) \quad (2)$$

for every finite measurable partition $A_1, \ldots, A_r$ 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

$$(G(A_1), \ldots, G(A_r))|\theta_1, \ldots, \theta_n \sim \text{Dir}(\alpha H(A_1) + n_1, \ldots, \alpha H(A_r) + n_r) \quad (3)$$

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

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

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

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

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 and 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 and 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

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

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

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

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:

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

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

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

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 $\theta_1,\ldots,\theta_n$ induce a partitioning of the set $[n] = \{1,\ldots,n\}$ into clusters such that within each cluster, say cluster k, the θ_i's take on the same value θ_k^*. Given that $\theta_1,\ldots,\theta_n$ 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 $\theta_1,\ldots,\theta_n$, 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 $1,2,\ldots$ 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:

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

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

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:

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

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 and 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 and 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 and 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

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

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 and James 2001; Pitman and Yor 1997), which has recently seen successful applications modeling data exhibiting power-law properties (Goldwater 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 et al. 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 Nonparametric Models
- Clustering
- Density Estimation
- Gaussian Process
- Prior Probability

Further Reading

In addition to the references embedded in the text above, we recommend the book (Hjort et al. 2010) on Bayesian nonparametrics.

Recommended Reading

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

Antoniak CE (1974) Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. Ann Stat 2(6):1152–1174

Blackwell D, MacQueen JB (1973) Ferguson distributions via Pólya urn schemes. Ann Stat 1:353–355

Escobar MD, West M (1995) Bayesian density estimation and inference using mixtures. J Am Stat Assoc 90:577–588

Ewens WJ (1972) The sampling theory of selectively neutral alleles. Theor Popul Biol 3:87–112

Ferguson TS (1973) A Bayesian analysis of some nonparametric problems. Ann Stat 1(2):209–230

Goldwater S, Griffiths TL, Johnson M (2006) Interpolating between types and tokens by estimating power-law generators. Adv Neural Inf Process Syst 18:459–466

Hjort N, Holmes C, Müller P, Walker S (eds) (2010) Bayesian nonparametrics. Cambridge series in statistical and probabilistic mathematics, vol 28. Cambridge University Press, Cambridge/New York

Ishwaran H, James LF (2001) Gibbs sampling methods for stick-breaking priors. J Am Stat Assoc 96(453): 161–173

Lo AY (1984) On a class of Bayesian nonparametric estimates: I. Density estimates. Ann Stat 12(1): 351–357

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

Neal RM (1992) Bayesian mixture modeling. In: Proceedings of the workshop on maximum entropy and Bayesian methods of statistical analysis, vol 11. Kluwer Academic Publishers, Dordrecht/Boston, pp 197–211

Neal RM (2000) Markov chain sampling methods for Dirichlet process mixture models. J Comput Graph Stat 9:249–265

Pitman J (2002) Combinatorial stochastic processes (Technical Report 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. Ann Probab 25:855–900

Rasmussen CE (2000) The infinite Gaussian mixture model. Adv Neural Inf Process Syst 12:554–560

Sethuraman J (1994) A constructive definition of Dirichlet priors. Stat Sin 4:639–650

Teh YW (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, Sydney, pp 985–992
Teh YW, Jordan MI, Beal MJ, Blei DM (2006) Hierarchical Dirichlet processes. J Am Stat Assoc 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, Box Hill, VIC, 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 et al. 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 et al. 2000). Eager methods perform discretization prior to classification time. Lazy methods perform discretization during the process of classification.
Disjoint vs. **Nondisjoint** (Yang and 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 and 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 and 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 SD (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 UM, Irani KB (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 CN, Huang HJ, Wong TT (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 JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers, San Francisco

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 $P(y|\mathbf{x})$ without reference to an explicit estimate of any of $P(\mathbf{x})$, $P(y, \mathbf{x})$, or $P(\mathbf{x}|y)$, 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 $P(\mathbf{x}|y)$.

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 $P(\mathbf{x}|y)$.

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.

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

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 edn. Chapman & Hall, Princeton, p 30

Distance

▸ Similarity Measures

Distance Functions

▸ Similarity Measures

Distance Measures

▸ Similarity Measures

Distance Metrics

▸ Similarity Measures

Distribution-Free Learning

▸ PAC Learning

Divide-and-Conquer Learning

Johannes Fürnkranz
Knowledge Engineering Group, TU Darmstadt, Darmstadt, Deutschland
Department of Information Technology, University of Leoben, Leoben, Austria

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 Categorization

▸ Document Classification

Document Classification

Dunja Mladenić[1], Janez Brank[2], and Marko Grobelnik[2]
[1]Artificial Intelligence Laboratory, Jožef Stefan Insitute, Ljubljana, Slovenia
[2]Jožef Stefan Institute, Ljubljana, Slovenia

Abstract
Document Classification analogous to general classification of instances, deals with

assigning labels to documents. The documents can be written in different natural languages, can be of different length and structure, can be written by different authors using variety of writing styles. Moreover, the documents to classify can be obtained from different sources including official news, internal company documentation, as well as public Web pages and texts from social media. In document classification, in addition to the algorithm we are using for constructing a classifier, data representation is crucial. Commonly used is word vector representation, where either raw data in the form of words and phrases is used or a more abstract form is constructed. Deep learning has been shown very effective for learning text representation using various deep learning architectures.

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 (also referred to class values). The main issues in document classification are connected to classification of free text giving document content, for instance, classifying Web documents on the content topic as being about arts, education, science, etc., or on the page type (personal homepage, company page, etc.), classifying news articles by their topic (politics, technology, science, health, etc.), and classifying movie reviews by their opinion (positive review, negative review). 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 vectors of feature; usually the features are words so each document is a word vector and the representation is referred to as the "bag-of-words" or "vector space model" representation. Classifier is then built using a set of documents that have been manually classified (Cohen and Singer 1996; Mladenić and 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 regard to the whole corpus (▸ (IDF) inverse document frequency). The IDF weight for the ith 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 ith word (the number of documents from the whole corpus in which the ith 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 $P(c|d) \propto P(c)\Pi_t P(t|c)$ TF(t, d), where $P(c)$ is the ▸ prior probability that a document belongs to class c, $P(t|c)$ 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 $P(c|d)$ is equivalent to taking the sign of ln $P(c_+|d)/P(c_-|d)$, 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 $P(t|c)$ 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 sgn$(\mathbf{w}^T\ \mathbf{x})$, where $\mathbf{w}$ is a vector of weights obtained during training. Computation starts with $\mathbf{w} = 0$ and 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 and $y_i = 1$ 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 programming 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 et al. 1996). Moreover, experimental evaluation on some standard Reuters news datasets shows that SVM tends to outperform other classifiers including naive Bayes and perceptron (Mladenic et al. 2004).

In many applications, a document may belong to multiple classes, e.g., because it includes discussion relevant to several topics. The classifiers described above can naturally provide multi-label predictions in a multi-class learning problem simply by treating each class as a two-class problem separately from the other classes. However, there are also methods that try to model the multi-class nature of individual documents directly. For example, recently there has been an increase of interest in using artificial neural networks for text classification, with a multiunit output layer that can generate predictions for all classes at once (Zhang and Zhou 2006; Nam et al. 2014).

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:

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

- Precision $=$ TP/(TP + FP)
- Recall $=$ TP_{rate} $=$ TP/(TP + FN)
- FP_{rate} $=$ FP/(TN + FP)
- $F_1 = 2 \bullet precision \bullet recall/(precision + recall)$

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

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 $\langle TP_{rate}, FP_{rate} \rangle$ pairs, resulting in an ▸ 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
- ▸ Feature Selection
- ▸ Precision
- ▸ Semi-supervised Text Processing
- ▸ Support Vector Machines
- ▸ Text Visualization

Recommended Reading

Cohen WW, 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. ACM, Zurich, pp 307–315

Lewis DD (1991) Representation and learning in information retrieval. PhD thesis, Department of computer science, University of Massachusetts, Amherst

Lewis DD, Schapire RE, Callan JP, 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. ACM, New York, pp 298–306

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. ACM, New York, pp 234–241
Mladenić D, Grobelnik M (2003) Feature selection on hierarchy of Web documents. J Decis Support Syst 35:45–87
Nam J, Kim J, Mencia EL, Gurevych I, Fürnkranz J (2014) Large-scale multi-label text classification – revisiting neural networks. In: Proceedings of ECML/PKDD, Nancy, pp 437–452
Sebastiani F (2002) Machine learning for automated text categorization. ACM Comput Surv 34(1):1–47
Yang Y (1997) An evaluation of statistical approaches to text categorization. J Info Retr 1:67–88
Zhang ML, Zhou ZH (2006) Multilabel neural networks with applications to functional genomics and text categorization. IEEE Trans Knowl Data Eng 18:1338–1351

Domain Adaptation

▸ Inductive Transfer

Dual Control

▸ Bayesian Reinforcement Learning
▸ Partially Observable Markov Decision Processes

Duplicate Detection

▸ Entity Resolution

Dynamic Bayesian Network

▸ Learning Graphical Models

Dynamic Decision Networks

▸ Partially Observable Markov Decision Processes

Dynamic Programming

Martin L. Puterman[1] and Jonathan Patrick[2]
[1]University of British Columbia, Vancouver, BC, Canada
[2]University of Ottawa, Ottawa, ON, 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 entry 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 variation problems in the seventeenth century. However, the modern investigation of stochastic sequential decision problems arguably dates back to the 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 Processes). 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 $n = 1, \ldots, N$ (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 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.

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 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 runoff

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 entry, 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,

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

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 leftover 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 underlie 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 setup 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:

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

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 Discounted Infinite-Horizon 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

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

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:

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

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

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

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

$$
\begin{aligned}
&v^{n+1}(s) \\
&\quad = \max_{a\in A(s)}\left\{r(s,a) + \sum_{j\in S}\lambda p(j|s,a)v^n(j)\right\}.
\end{aligned}
$$

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

$$d_{n+1} \in \text{argmax}_{d \in D}\{r_d + \lambda P_d v^n\}$$

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

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

(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

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

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

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^∞, $X(s,a)$ 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

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

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

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

   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:

D

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

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

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

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

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 (e.g., 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 predefined 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. Athena Scientific, Belmont

Bertsekas D, Tsitsiklis J (1996) Neuro-dynamic programming. Athena Scientific, Belmont

Feinberg E, Shwartz A (2002) Handbook of Markov decision processes. Kluwer Academic, Boston

Puterman M (1994) Markov decision processes. Wiley, New York

Sutton R, Barto A (1998) Reinforcement learning. MIT, Cambridge

Dynamic Programming for Relational Domains

▸ Symbolic Dynamic Programming

Dynamic Selection of Bias

▸ Metalearning

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.