Volume : 2 | Issue : 8 | August 2013 • ISSN No 2277 - 8179

Research Paper

The Computational Efficiency of the Distribution-Based Algotithm

Engineering

KEYWORDS : Classification, decision tree model, data uncertainty, probability distribution.

S. GOVINDA RAO	Department of C. Sc. , Avanthi's St.Theressa, Institute of Engg.&Tech., GARIVIDI, A.P., India.
CHNNA BABU GALINKI	Assistant Professor, Department of C. Sc. Avanthi's St.Theressa, Institute of Engg.&Tech., GARIVIDI, A.P., India.
HEMALATA KONDALA	Assistant Professor, Department of C. Sc. Avanthi's St.Theressa, Institute of Engg.&Tech., GARIVIDI, A.P., India.

ABSTRACT

Classification is a classical problem in machine learning and data mining. One of the most popular classification models is the decision tree model. Many algorithms have been devised for decision tree construction. In traditional decision-tree classification, a feature (an attribute) of a tuple is either categorical or numerical. For the latter, a precise and definite point value is usually assumed. In many applications, however, data uncertainty is common. The value of a feature/attribute is thus best captured not by a single point value, but by a range of values giving rise to a probability distribution. A simple way to handle data uncertainty is to abstract probability distributions by summary statistics such as means and variances. This approach is known as Averaging. Another approach is to consider the complete information carried by the probability distributions to build a decision tree. This approach is known as Distribution-based.

INTRODUCTION

Classification is a classical problem in machine learning and data mining. Given a set of training data tuples, each having a class label and being represented by a feature vector, the task is to algorithmically build a model that predicts the class label of an unseen test tuple based on the tuple's feature vector. One of the most popular classification models is the decision tree model. Decision trees are popular because they are practical and easy to understand. Rules can also be extracted from decision trees easily. Many algorithms have been devised for decision tree construction. These algorithms are widely adopted and used in a wide range of applications such as image recognition, medical diagnosis, and credit rating of loan applicants, scientific tests, fraud detection, and target marketing.

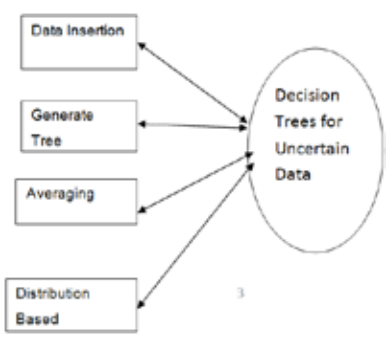
In traditional decision-tree classification, a feature (an attribute) of a tuple is either categorical or numerical. For the latter, a precise and definite point value is usually assumed. In many applications, however, data uncertainty is common. The value of a feature/attribute is thus best captured not by a single point value, but by a range of values giving rise to a probability distribution. A simple way to handle data uncertainty is to abstract probability distributions by summary statistics such as means and variances. We call this approach averaging. Another approach is to consider the complete information carried by the probability distributions to build a decision tree. We call this approach Distribution-based.

EXISTING SYSTEM:

In traditional decision-tree classification, a feature (an attribute) of a tuple is either categorical or numerical. For the latter, a precise and definite point value is usually assumed. In many applications, however, data uncertainty is common. The value of a feature/attribute is thus best captured not by a single point value, but by a range of values giving rise to a probability distribution. Although the previous techniques can improve the efficiency of means, they do not consider the spatial relationship among cluster representatives, nor make use of the proximity between groups of uncertain objects to perform pruning in batch. A simple way to handle data uncertainty is to abstract probability distributions by summary statistics such as means and variances. We call this approach averaging. Another approach is to consider the complete information carried by the probability distributions to build a decision tree. We call this approach Distribution-based.

PROPOSED SYSTEM:

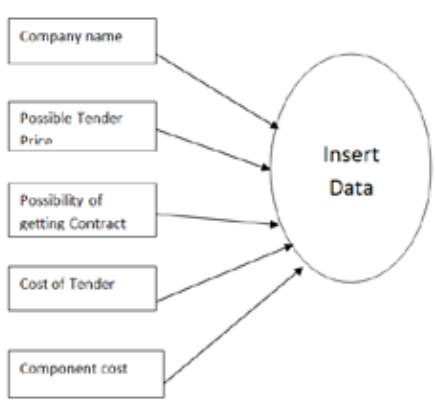
We study the problem of constructing decision tree classifiers on data with uncertain numerical attributes. Our goals are (1) to devise an algorithm for building decision trees from uncertain data using the Distribution-based approach; (2) to investigate whether the Distribution-based approach could lead to a higher classification accuracy compared with the Averaging approach; and (3) to establish a theoretical foundation on which pruning techniques are derived that can significantly improve the computational efficiency of the Distribution-based algorithms.



Modules:

Data Insertion

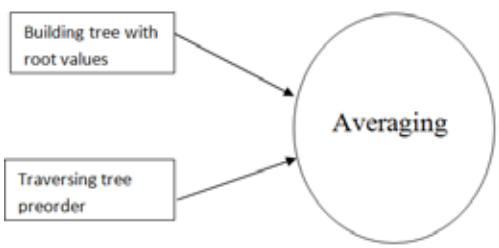
In many applications, however, data uncertainty is common. The value of a feature/attribute is thus best captured not by a single point value, but by a range of values giving rise to a probability distribution. With uncertainty, the value of a data item is often represented not by one single value, but by multiple values forming a probability distribution. This uncertain data is inserted by user.



Volume : 2 | Issue : 8 | August 2013 • ISSN No 2277 - 8179

Generate Tree
Building a decision tree on tuples with numerical, point valued data is computationally demanding. A numerical attribute usually has a possibly infinite domain of real or integral numbers, inducing a large search space for the best "split point". Given a set of n training tuples with a numerical attribute, there are as many as n-1 binary split points or ways to partition the set of tuples into two non-empty groups. Finding the best split point is thus computationally expensive. To improve efficiency, many techniques have been proposed to reduce the number of candidate split points.

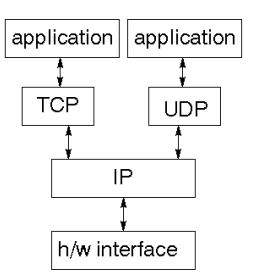
Averaging
A simple way to handle data uncertainty is to abstract probability distributions by summary statistics such as means and variances. We call this approach Averaging. A straight-forward way to deal with the uncertain information is to replace each pdf with its expected value, thus effectively converting the data tuples to point-valued tuples. This reduces the problem back to that for point-valued data. AVG is a greedy algorithm that builds a tree top-down. When processing a node, we examine a set of tuples S. The algorithm starts with the root node and with S being the set of all training tuples. At each node n, we first check if all the tuples in S have the same class label.



Distribution Based
An approach is to consider the complete information carried by the probability distributions to build a decision tree. We call this approach Distribution-based. Our goals are,

(1) To devise an algorithm for building decision trees from uncertain data using the Distribution-based approach;
(2) To investigate whether the Distribution-based approach could lead to a higher classification accuracy compared with the Averaging approach;
(3) To establish a theoretical foundation on which pruning techniques are derived that can significantly improve the computational efficiency of the Distribution-based algorithms.

Networking
TCP/IP stack
The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

IP datagram's

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

UDP
UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

TCP
TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

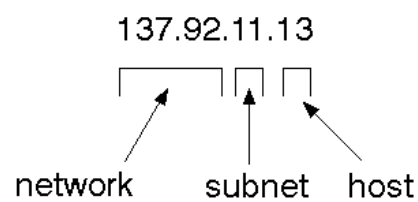
Internet addresses
In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address**.**

Network address
Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

Subnet address
Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

Host address
8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

Total address



The 32 bit address is usually written as 4 integers separated by dots.

Port addresses
A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

Sockets
A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h>

#include <sys/socket.h>

int socket(int family, int type, int protocol);
```

Here "family" will be AF_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

IMPLEMENTATION
Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investiga-

tion of the existing system and it's constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

Implementation is the process of converting a new system design into operation. It is the phase that focuses on user training, site preparation and file conversion for installing a candidate system. The important factor that should be considered here is that the conversion should not disrupt the functioning of the organization.

CONCLUSION

In this article, decision-tree classification model is extended to accommodate data tuples having numerical attributes with uncertainty described by arbitrary pdf's. The classical decision tree building algorithms are modified to build decision trees for classifying uncertain data. It is found empirically that when suitable pdf's are used, exploiting data uncertainty leads to decision trees with remarkably higher accuracies. Therefore it is proved that data be collected and stored with the pdf information intact. Performance is an issue, though, because of the increased amount of information to be processed, as well as the more complicated entropy computations involved.

REFERENCE

1. Accoria. Rock web server and load balancer. http://www.accoria.com. | 2. Amazon Web Services. Amazon Web Services (AWS). http://aws.amazon.com. | 3. V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. IEEE Internet Computing, 3(3):28{39, 1999. | 4. L. Cherkasova. FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service. IEEE Symposium on Computers and Communications, 0:8, 2000. | 5. F5 Networks. F5 Networks. http://www.f5.com. | 6. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol { http/1.1. In IETF RFC 2616, 1999. | 7. Google Inc. Google App Engine. http://code.google.com/appengine/. | 8. HaProxy. HaProxy load balancer. http://haproxy.1wt.eu/. | 9. G. Hunt, E. Nahum, and J. Tracey. Enabling content-based load distribution for scalable services. Technical report, 1997. | 10. E. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In Proc. First International Conference on the World Wide Web, Apr. 1994.