Inducing knowledge from examples

Dunja Mladenić
J. Stefan Institute
Jamova 39, 61000 Ljubljana, Slovenia

Abstract

The paper presents way of explaining knowledge induction process to help students to understand it, and experiment by means of computer program. The knowledge is induced from examples of some situation with a sort of generalization that use entropy measure to guide the generalization process. Set of examples can be changed to enable analysis of influence of new examples to the induced knowledge. Simple addition of a new example that describe the same situation, possibly leads to modification of the induced knowledge. The process is illustrated with the simple algorithm implemented in LOGO. Furthermore, the algorithm is explained in details to enable simple modification and use with some other input examples. The induction algorithm implemented in LOGO is one of the well known machine learning algorithms. It has been developed in a few ways to some poverful algorithms that have been successfully applied on a lot of real-world problems, to induce knowledge about some real situation. One of widely known real-world application is in medical diagnosis where machine learning system is used to predict a patient diagnosis from the symptoms.

Key words: teaching your computer, knowledge induction, machine learning

1 Introduction

Knowledge induction is the ability that we oftenly need in our everyday life. For example, observing the current weather situation, we can try to predict the weather for the next day. With days, or better with years of observation our predictions are becoming more and more correct. This process can be viewed as induction of knowledge contained in the weather examples. Each example consists of some values describing the weather situation at the appropriate day and is connected to prediction of the next-day weather. Such induction process can be used as the basis for programming the computer to induce knowledge. The idea of knowledge induction from examples is already known in the *machine learning*, area of the computer science field called *artificial intelligence*.

Section 2 briefly describes the idea of learning from examples. Section 3 introduces the machine learning algorithm used in LOGO program for knowledge induction. The LOGO program is illustrated and discussed in Section 4. Section 5 gives a brief discussion of the presented work. The part of LOGO program that is not in Section 4 is given in Appendix.

2 Problem description

When *learning from examples* set of examples containing knowledge about the target situation is used as program input. Each example can be viewed as a n-tuple of *attributes* where each attribute describes one property of the concrete situation (e.g. weather situation at the appropriate day). To each example, a value called *class* is assigned (e.g. prediction of the next-day weather). The problem is to induce knowledge describing value of the class by means of the attribute values.

To illustrate the way of knowledge induction with computer, one of the basic machine learning algorithms ID3 [3] has been implemented in LOGO. The algorithm belongs to the family of machine learning algorithms called Top Down Induction of Decision Trees. The main characteristic of these algorithms is that knowledge induction is performed using top-down approach and is represented by means of decision tree. *Decision tree* consists of nodes containing tests of attribute values and leaves containing the predicted class values. Once generated, the tree can be used to classify new (unseen) examples, i.e. to decided which class value a new example has. This is done by proceeding the new example from the root of the tree to the appropriate leaf. Each step consists of testing the current node attribute value example has, and proceeding to the subtree according to the test result. The class value assigned to the final node-leaf is the class predicted to the proceeded example.

2.1 An example problem domain

example	class	attributes			
number	suitable	outlook	temperature	humidity	windy
1	no	sunny	hot	high	no
2	no	sunny	hot	high	yes
3	yes	overcast	hot	high	no
4	yes	rain	mild	high	no
5	yes	rain	cool	normal	no
6	no	rain	cool	normal	yes
7	yes	overcast	cool	normal	yes
8	no	sunny	mild	high	no
9	yes	sunny	cool	normal	no
10	yes	rain	mild	normal	no
11	yes	sunny	mild	normal	yes
12	yes	overcast	mild	high	yes
13	yes	overcast	hot	normal	no
14	no	rain	mild	high	yes

Table 1: Examples for the 'Saturday morning' domain.

To describe learning from examples as performed by ID3 algorithm a simple 'Saturday morning' [3] domain is used. In this machine learning domain, 14 training examples are given (see Table 1). Each example is labelled with its class: suitable (no, yes), saying if a morning is suitable for some unspecified activity, and 4 attributes: outlook (sunny, overcast, rain), temperature (hot, mild, cool), humidity (high, normal) and windy (no, yes), describing a Saturday morning.

The set of examples given in Table 1 contains the information about the 'Saturday morning'. Some examples describe mornings that are suitable for some unspecified activity (class = `yes`) and the other describe unsuitable mornings (class = `no`). Looking at the first example in Table 1, (`no, sunny, hot, high, no`), one can see that morning is not suitable if `outlook` is `sunny`, `temperature` is `hot`, `humidity` is `high` and `windy` is `no`.

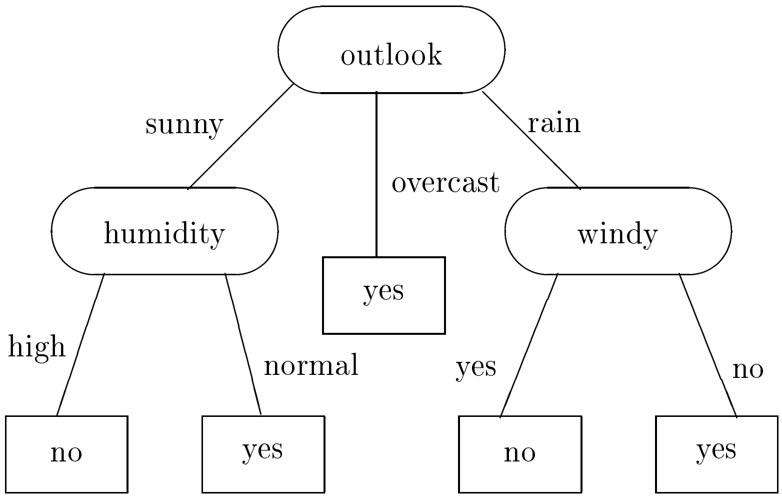

Figure 1: Desicion tree for 'Saturday morning' domain.

To induce some knowledge about the suitability (class) from the given set of examples, one will search some regularities (correlations) between the class and the attribute values. Focusing the attribute `outlook` in Table 1, it is easy to see that always when `outlook` is `overcast` the class is `yes`.[1] Described machine learning algorithm is performing its induction in the similar way. Decision tree induced by the ID3 algorithm from the whole set of examples is given in Figure 1 where ovals labelled with the attribute name denote nodes that test attribute value, and rectangles labelled with the class denote leaves predicting the class.

The tree can be interpreted (starting with the most left leaf) in the following way:

if outlook = sunny and humidity = high **then** suitable = no
if outlook = sunny and humidity = normal **then** suitable = yes
if outlook = overcast **then** suitable = yes
if outlook = rain and windy = yes **then** suitable = no
if outlook = rainy and windy = no **then** suitable = yes

A new example about the same situation (same class and attributes) can be classified using the above tree. Suppose that today's weather can be described by means of attributes as `outlook` is `rain`, `temperature` is `hot`, `humidity` is `normal` and `windy` is `no`. We want to know if the day is suitable (if class is `yes`). The class is predicted using the above tree so

[1]Notice that the middle leaf of the tree in Figure 1 is saying exactly the same as we have discovered just looking at the table.

that example is proceeded from the root of the tree to appropriate leaf. `Outlook` is `rain` and `windy` is `no` leads to the most right leaf labelled with class `yes`. In this situation it is obvious that `temperature` and`humidity` are not important.

3 Algorithm used for knowledge induction

Machine learning algorithm ID3 [3] induces knowledge from examples and presents it in the form of decision tree. Decision tree consists of nodes, each containing test of an attribute value, and leaves labelled with the class value. Illustrated on the tree from Figure 1, the root of the tree tests value of the `outlook` attribute and has three subtrees, one for each value of the attribute. To each node/leaf a set of learning examples is assigned, so that all examples satisfy all the tests (of attribute values) that appear on the path from the root to the current node. Illustrated on the same tree, to the root the whole learning example set is assigned, since there is no test before the root. To the left branch (node with attribute `humidity`) only examples with the `outlook = sunny` are assigned, to the most left leaf (label with class `no`) only examples with the `outlook = sunny` and `humidity = high` are assigned. The union of example sets assigned to all leaves should be the whole learning example set.
The main ID3 algorithm is as follows:

> $ExSet$:= set of examples;
> **if** all examples in $ExSet$ have the same class C **then**
>   Make a leaf labelled with the class C
> **else**
>   Find the most informative attribute to be the root of the current decision tree;
>   According to the value of the chosen attribute split $ExSet$ into subsets;
>   Repeat the whole process for each subset; { generate subtrees}
> **endif**
> **output** decision tree;

The main ID3 algorithm constructs decision tree from examples starting with the whole set of examples (root), splitting it into subsets according to the value of chosen attribute and preforming the same procedure to each subset. When some subset consists only of examples of the same class, the process is stopped by construction of the leaf. The most informative attribute is find according to the informativity calculated for each attribute using the following formula:

$$Informativity(Attribute) = Entropy(ExSet) - Entropy(Attribute)$$

$$Entropy(ExSet) = \sum_i p_i \log(p_i); Entropy(Attribute) = \sum_v p_v * Entropy(ExSet_v)$$

where p_i is the probability of i-th class in the set $ExSet$, p_v is the probability of the v-th value of attribute and $ExSet_v$ is the example set assigned to v-th branch.

Illustrated on the examples give in Table 1, the entropy of the whole set is calculated from the set of 14 examples, 5 with class `no` and 9 with class `yes`. $Entropy(ExSet) =$

$-\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$ bits. Calculating the informativity for the `outlook` attribute that has three values (`sunny, overcast, rain`) 5 of the 14 examples have the first value, 2 of them from class `yes` and 3 from class `no`. The probability for the first branch is $p_1 = \frac{5}{14}$ and its entropy is $Entropy(ExSet_1) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$ bits. Entropy of the rest two branches is calculated in the same way. The entropy and informativity of attribute `outlook` are then calculated:

$$Entropy(outlook) = \frac{5}{14} * 0.971 + \frac{4}{14} * 0 + \frac{5}{14} * 0.971 = 0.694$$

$$Informativity(Attribute) = 0.940 - Entropy(outlook) = 0.246$$

4 Description of the Logo implementation

Program is written in LogoS ([1], [2]), but can be easily adapted for any other Logo interpreter. It implements described knowledge induction algorithm and consist of three main parts; the first reading domain (description of attributes and classes, and examples), the second generating decision tree and the third writing decision tree to screen. The main procedure `To ID3` illustrates knowledge induction process starting with tree generation from a single example and adding examples one by one. The last generated tree (from the 14 examples) is the same as given in Figure 1.

```
CmLn "ID3 End
To ID3
  TxScr
  Local "TrainExL
  Make "TrainExL []
  RdDomain ;read domain into Class, AttrL and ExL
  Repeat Length :ExL {
    Make "TrainExL AddLast ( Item GetIter :ExL ) :TrainExL
    Make "Tree ( GenTree :Class :AttrL :TrainExL ) ;generate decision tree
    Clr
    print "' ' type GetIter
    type "'. example added :' print ( Item GetIter :ExL ) print "' '
    WrTree :Tree 1 ;write decision tree
    Wait
  }
End
```

4.1 Read of domain description and examples

The `RdDomain` procedure sets values of three variables. `Class` is list of class values (`[ no yes ]`) and must contain all possible values of the class. `AttrL` is list of attributes, each attribute is represented with list of two elements (`[ outlook [ sunny overcast rain ]`) the first is attribute name and the second is list of attribute values. `ExL` is list of learning examples where each example is represented with list of values (`[ no sunny hot high no ]`), where the first (class) value is followed by the attribute values given in the same order as the attributes in the domain description (`AttrL`). Example set thus looks very similar to the examples given in Table 1. Experiment with some other domain is simply performed by changing the contents of these three variables in procedures `GetClass`, `GetAttrL` and `GetExL`.

```
To RdDomain
  Make "Class GetClass
  Make "AttrL GetAttrL
  Make "ExL GetExL
End

To GetClass
  Output [ false true ]
End

To GetAttrL
  Output [
    [ outlook [ sunny overcast rain ]]
    [ temperature [ hot mild cool ]]
    [ humidity [ high normal ]]
    [ windy [ false true ]]
  ]
End
```

```
To GetExL
  Output [
    [ false sunny hot high false ]
    [ false sunny hot high true ]
    [ true overcast hot high false ]
    [ true rain mild high false ]
    [ true rain cool normal false ]
    [ false rain cool normal true ]
    [ true overcast cool normal true ]
    [ false sunny mild high false ]
    [ true sunny cool normal false ]
    [ true rain mild normal false ]
    [ true sunny mild normal true ]
    [ true overcast mild high true ]
    [ false rain mild high true ]
    [ true overcast hot normal false ]
  ]
End
```

4.2 Write decision tree

Decision tree is written simply to text screen by the procedure `WrTree` that has two parameters, `Tree` and `Level`. The first is generated decision tree and the second is current level in the tree. `Level` enables indentation, by means of typing ' ', so that all nodes from the same level of the tree (same path length from the tree root) are left justified. For example, the tree shown in Figure 1 is written as follows :

```
attr = outlook
(sunny)
    attr = humidity
    (high)
      class = no
    (normal)
      class = yes
(overcast)
    class = yes
(rain)
    attr = windy
    (yes)
      class = no
    (no)
      class = yes
```

What follows is the procedure that writes decision tree in the described way.

```
To WrTree :Tree :Level
  ( Local "Attr "AVal "SubTrees )
  If (( Item 1 :Tree ) = "Null ) {
    Repeat :Level{ type "' ' }
    print "Null
  }{
  If (( Item 1 :Tree ) = "Leaf ) {
    Repeat :Level{ type "' ' }
    type "'class ' type "'= ' print ( Item 2 :Tree )
  }{ ; ( Item 1 :Tree ) = "Branch
    Repeat :Level{ type "' ' }
    type "'attr ' type "'= ' Make "Attr ( Item  2 :Tree )
    print ( Item 1 :Attr ) ;attribute name
    Make "AVal ( Item 2 :Attr ) ;attribute values
    Make "SubTrees ( Item 3 :Tree ) ;subtrees
    Repeat Length :SubTrees {
      Repeat :Level { type "' ' }
      type "'( ' type ( Item GetIter :AVal ) print "' )'
      WrTree ( Item GetIter :SubTrees ) ( :Level + 1 )
    }
  }}
End
```

4.3 Decision tree generation

Decision tree is generated with the procedure GenTree that implements the main ID3 algorithm described in Section 3. It has class, attributes and examples as parameters. The first line in the below code is needed because the procedures GenTree and GenSubTrees are calling each other.

```
ForwTo GenTree, 3 End

To GenSubTrees :Class :AttrL :SplitedExL
  Local "TreeL
  Make "TreeL [ ]
  Repeat Length :SplitedExL {
    Make "Tree GenTree :Class :AttrL ( Item GetIter :SplitedExL )
    Make "TreeL AddLast :Tree :TreeL
  }
  Output :TreeL
End

To GenTree :Class :AttrL :ExL
  ( Local "Best "BestAttr "BestSplitedExL )
  If EmptyP :ExL {
    Output List "Null
  }{
  If SameClass :ExL {
    Output ( List "Leaf ( Item 1 ( Item 1 :ExL )))
```

```
  }{
    Make "Best FindBestAttr :Class :AttrL :ExL
    Make "BestAttr Item 1 :Best
    Make "BestSplitedExL Item 2 :Best
    Output ( List "Branch :BestAttr
     ( GenSubTrees :Class :AttrL :BestSplitedExL ))
  }}
End

To SameClass :ExL; check if all examples are of the same class
  Repeat Length :ExL{
    If ( Item 1 ( Item GetIter :ExL )) <> Item 1 ( Item 1 :ExL ) {
      Output "false
    }{}
  }
  Output "true
End
```

Procedure `GenTree` consists of three parts, two simple stopping criterion and the third containing recursive call. In the third part the most informative attribute is found and example set is splited according to it values (`FindBestAttr`). Procedure `GenSubTrees` is called with the domain information (class, attributes) and the list of splited example set. It just calls `GenTree` again for each element of the list (each example set) and generates list of subtrees `TreeL`.

`FindBestAttr` is procedure that implements informativity/entropy formula described in section 3 and needs procedures given in Appendix.

5 Discussion

Simple knowledge induction algorithm described in the paper, can be used as illustration of the way of learning from examples. It is also implemented in Logo and can be typed in computer to enable performing experiments. For the purpose of experiments and analyses, input data can be modified to describe some new situation. The idea is also for the students to get some opinions about way computer can perform learning, to try to be a teacher to computer.

References

[1] Batagelj, V., Grobelnik, M. and Zupanič, D. (1991) The design of LogoS, *Third Europen Logo Conference Proceedings*, pp. 287-296, Parma, Italy.

[2] Grobelnik, M. and Zupanič, D. (1991) Inside LogoS implementation, *Third Europen Logo Conference Proceedings*, pp. 297-311, Parma, Italy.

[3] Quinlan, J.R. (1986) Induction of decision trees. *Machine Learning 1*, pp. 81-106. Kluwer Academic Publishers.

Appendix

```
To SplitExL :Attr :AttrL :ExL; split set of ex. according to given attr.
  ( Local "SplitedExL  "Ex )
  ( Local "AValExL "AVal "AValL "AttrNum )
  Make "AttrNum NMember :Attr :AttrL
  Make "AValL Item 2 :Attr
  Make "SplitedExL []
  Repeat Length :AValL {
    Make "SplitedExL AddLast [] :SplitedExL
  }
  Repeat Length :ExL {
    Make "Ex ( Item GetIter :ExL )
    Make "AValNum NMember ( Item ( :AttrNum + 1 ) :Ex ) :AValL
    Make "AValExL AddLast :Ex ( Item :AValNum :SplitedExL )
    Make "SplitedExL Replace :AValNum :AValExL :SplitedExL
  }
  Output :SplitedExL
End

To CalcClassProbL :Class :ExL :Exs; calculate probability for each class
  ( Local "ProbL "Ex "ClassNum )
  Make "ProbL []
  Repeat Length :Class {
    Make "ProbL AddLast 0 :ProbL
  }
  Repeat Length :ExL {
    Make "Ex Item GetIter :ExL
    Make "ClassNum NMember ( Item 1 :Ex ) :Class
    Make "ProbL Replace :ClassNum ( Sum ( Item :ClassNum :ProbL ) 1 ) :ProbL
  }
  Repeat Length :ProbL {
    Make "ProbL Replace GetIter ( Div ( Item GetIter :ProbL ) :Exs ) :ProbL
  }
  Output :ProbL
End

To CalcEntropy :Class :ExL :Exs; calculate entropy for given ex. set
  ( Local "ClassProbL "ClassProb "Log2Prob "Entropy )
  Make "Entropy 0
  Make "ClassProbL CalcClassProbL :Class :ExL :Exs
  Repeat Length :Class {
    Make "ClassProb ( Item GetIter :ClassProbL )
    If :ClassProb > 0 {
      Make "Log2Prob ( Div Log :ClassProb Log 2 )
    }{
      Make "Log2Prob 0
    }
    Make "Entropy ( Diff :Entropy ( Prod :ClassProb :Log2Prob ))
```

```
  }
  Output( :Entropy )
End

To AttrEntropy :Class :AttrL :SplitedExL :Exs; calculate attr. entropy
  ( Local "ExL "AEntropy )
  Make "AEntropy 0
  Repeat Length :SplitedExL {
    Make "ExL ( Item GetIter :SplitedExL )
    Make "AEntropy ( Sum :AEntropy ( Prod ( Div Length :ExL :Exs )
     CalcEntropy :Class :ExL :Exs ))
  }
  Output :AEntropy
End

To FindBestAttr :Class :AttrL :ExL; find best attr. and split ex. set
  ( Local "BestAttr "BestSplitedExL "BestAttrInf )
  ( Local "Attr "SplitedExL "AttrInf )
  ( Local "Exs "Entropy )
  Make "Exs Length :ExL
  Make "Entropy CalcEntropy :Class :ExL :Exs
  Make "BestAttr 1
  Make "BestAttrInf 0
  Repeat Length :AttrL {
    Make "Attr Item GetIter :AttrL
    Make "SplitedExL SplitExL :Attr :AttrL :ExL
    Make "AttrInf ( Diff :Entropy ( AttrEntropy :Class :AttrL :SplitedExL :Exs))
    If :AttrInf > :BestAttrInf {
      Make "BestAttr :Attr
      Make "BestSplitedExL :SplitedExL
      Make "BestAttrInf :AttrInf
    }{}
  }
  Output ( List :BestAttr :BestSplitedExL )
End
```