Machine Learning: Decision Trees & the ID3 Algorithm

CSE 415: Introduction to Artificial Intelligence
University of Washington
Winter, 2018

Outline

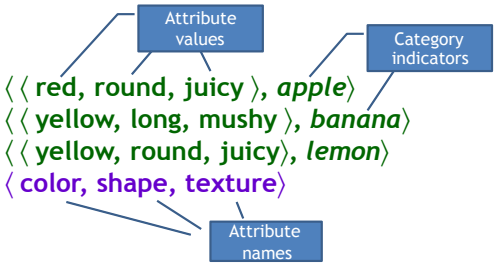
- Motivation
- Training Examples
- General Algorithm
- Maximum Entropy Attribute Selection
- Generalization vs Overfitting

Motivation

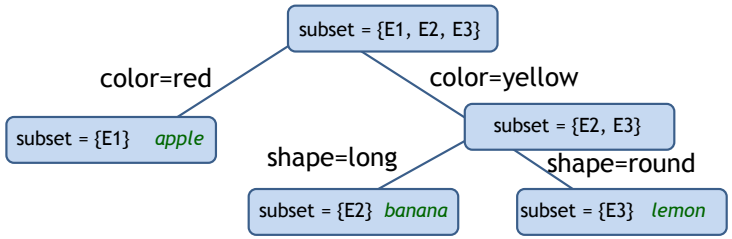
•Perceptron learning requires the data to be linearly separable.
•Decision tree learning will not have this limitation.
•Decision trees provide a more transparent classification method than neural nets do.
•Attributes of input examples are often heterogenous; decision-tree learning can take that into account.

Training Examples

A training example consists of a vector of attribute values with a category indicator.



Decision Tree



E1: ⟨ ⟨ red, round, juicy ⟩, *apple*⟩
E2: ⟨ ⟨ yellow, long, mushy ⟩, *banana*⟩
E3: ⟨ ⟨ yellow, round, juicy⟩, *lemon*⟩
⟨ color, shape, texture⟩

General Tree-Building Algorithm

```
Create root R
S_R={all training examples}
LEAF(R)=False
OPEN = [R]
While OPEN not empty:
    N = OPEN.dequeue()
    if examples in S_N all have the same category K:
        LEAF(N)=true
        category(N) = K
    else:
        A = chooseBestAttribute(S_N)
        VALS = {v | A has value v in an example in S_N}
        for v in VALS:
            C_v = new child of N
            S_Cv = {e in S_N | e has value v for attribute A}
            OPEN.enqueue(C_v)
```


General Tree-Building Algorithm

```
Create root R
SR={all training examples}
LEAF(R)=False
OPEN = [R]
While OPEN not empty:
    N = OPEN.dequeue()
    if examples in SN all have the same category K:
        LEAF(N)=true
        category(N) = K
    else:
        A = chooseBestAttribute(SN)
        VALS = {v | A has value v in an example in SN}
        for v in VALS:
            Cv = new child of N
            SCv = {e in SN | e has value v for attribute A}
            OPEN.enqueue(Cv)
```


What is the Best Attribute?

The best attribute is usually the one that spreads the elements of the set most evenly among the resulting children.

The worst kind of attribute fails to distinguish among examples at all:

E1: ⟨ ⟨ red, round, juicy ⟩, *apple*⟩
E2: ⟨ ⟨ yellow, long, mushy ⟩, *banana*⟩
E3: ⟨ ⟨ yellow, round, juicy⟩, *lemon*⟩

Let S_N = { E1, E3 }.
Bad attributes: shape, texture (because E1 and E3 cannot be distinguished at all using them)
Good attribute: color. (The apple and lemon are distinguished by color.)

How good is an attribute A at distinguishing elements of a set S?
One answer: compute its "entropy".

$$\text{Entropy}(A) = \Sigma -P(v_i) \log P(v_i)$$

Maximize Entropy

$$\text{Entropy}(Q_A) = \sum_{v \text{ in } VALS_A} -P(v) \log_2 P(v)$$

Here Q_A is the population of examples projected onto attribute A alone.
E.g. [red, yellow, yellow]
P(v) is the probability of v within the set.

We usually take P(v) to be the relative frequency with which A has value v in the set.

The entropy associated with attribute A is a measure of the expected information gain (in bits) from asking "What is the value of A?" for a randomly chosen member of the subset of training examples associated with the current node.

Python Code for Entropy

```
import math                    # for the log function

def entropy(lst):
    n = len(lst)+0.0           # denominator for probabilities
    myhash = {}
    for elt in lst:            # Use a hash to obtain occurrence counts
        try:
            val = myhash[elt]
            val += 1
            myhash[elt]=val
        except:
            myhash[elt]=1
    sum = 0.0                  # prepare to accumulate the entropy
    for item in myhash.keys():
        p = myhash[item]/n     # probability of this item within the population
        logp = math.log(p,2)   # log, base 2, of the probability
        sum -= p * logp        # accumulation by subtraction of a negative number
    return sum


print(entropy([9,2,1,1,1,9])) # sample call with a "bag" represented as a list. 1.459
```


Entropy Example

$$\text{Entropy}(Q_A) = \sum_{v \text{ in } VALS_A} -P(v) \log_2 P(v)$$

Example set of vectors with two attributes:
{ (A=1, B=1), (A=1, B=2), (A=2, B=1), (A=1, B=5)}
Project on each attribute to get a multiset ("bag" or "population").
Q_A = [1, 1, 2, 1]; Q_B = [1, 2, 1, 5];

For clarity of presentation, let's also project to sets, and show the probabilities.
$VALS_A$ = {1, 2} P(1)=0.75; P(2)=0.25. Entropy(Q_A)=0.811

$VALS_B$ = {1, 2, 5} P(1)=0.5; P(2)=0.25; P(5)=0.25
Entropy(Q_B)=1.5

B is a better attribute than A.

Limitations of ID3

The ID3 algorithm is a *greedy* algorithm. There is no lookahead to determine if the new subsets can be efficiently dichotomized further.

The policy of maximizing entropy at each node is a *heuristic* - a policy that is usually better than no policy. It tends to produce shallower trees than no policy would.

However, it does *not guarantee* finding an optimum tree overall.

There is no bias towards asking the same questions in the same order for different input examples. Thus the question sequences associated with different paths tend to be different. (However, since the root is the starting point for all input examples, the first question will always be the same.)

ID3 is prone to *overfitting*, particularly when the training set contains "noise" examples that break normal patterns of generalization.

Overfitting: adapted to handle the training examples correctly, but without sufficient generalization to handle new examples properly.

Subsequent Progress

The C4.5 algorithm (also by Ross Quinlan*) improved on ID3 in several ways:

•Allowing continuous attribute values by incorporating thresholding.
•Allowing training examples to have missing attribute values.
•Allowing different costs for each attribute.
•Tree simplification at the end of the algorithm.

*Ross Quinlan was the first person to get a Ph.D. in AI at U.W., according to his advisor, Earl Hunt.
Some textbooks (e.g., Tan, Steinback and Kumar) call the basic version of ID3 "Hunt's Algorithm".
P.S. UW Prof. Earl B. Hunt passed away April 12, 2016.

Random Forests

The classification accuracy and ability to generalize are typically improved by creating many different decision trees, using random subsets of the training data. The results are combined, using bagging (bootstrap aggregation).

Bagging: creating many classifiers from the same training data by generating random subsets (with replacement) and training each classifier with a random subset. Classification by the ensemble is based on the mode of the individual classifier outputs, or if the classifiers output continuous values, by the mean.