Data Mining Lectures - Decision trees

Piotr Wąsiewicz

Institute of Computer Science

pwasiewi@elka.pw.edu.pl

25 marca 2019

Decision trees

- A hierarchical structure representing dataset/domain partitioning nodes: splits based on attribute-value conditions and leaves: class labels or probability distributions.
- Prediction by descending the tree at each node dispatching along a branch at each leaf a class label or probability determined.

Splits for discrete attributes

- Value-based: split outcomes correspond to single attribute values.
- Equality based: split outcomes correspond to binary equality test results.
- Partition-based: split outcomes correspond to attribute value subsets.
- Membership-based: split outcomes correspond to binary membership test results.

Splits for numeric attributes

- Inequality based: split outcomes correspond to binary inequality test results.
- Interval-based: split outcomes correspond to attribute value interval.

Decision tree growing

1. create the root node and mark it as open;
2. assign all training instances from T to the root node;
3. while there are open nodes:
   A. select an open node n;
   B. calculate class distribution $T(d|n)$ based on T[n];
   C. assign class label $argmax[d]T(d|n)$ to n;
   D. if stop criteria are satisfied for n mark n as a closed leaf; else
      1. select a split t for n;
      2. for each outcome r of split t:
      A. create a descendant node n[r] corresponding to r and mark it as open;
      B. assign all instances from T[n,t=r] to n[r];
      C. mark n as a closed node;

Stop criteria

- Uniform class: all training instances in the node are of the same class.
- No instances left: the set of training instances assigned to the node is empty.
- No splits left: there is no split that can be applied to further partition the current subset of training instances.
- Can be relaxed:
  1. most instances of the same class (low class impurity),
  2. less than a specified minimum number of instances,
  3. the best available split is not sufficiently good.

Split selection

- Strict stop criteria guarantee training set error minimization.
- Split selection responsible for overfitting avoidance.
- Ockham's razor: among trees with the same training set error prefer smaller ones and it can be achieved by minimizing class impurity e.g. its entropy.

Pruning and probability classification

- In pruning the complexity parameter (cp) controls the tradeoff between error and size
- Class probability distribution at leaves enables probabilistic prediction
- Can be used to minimize misclassification costs; instead of predicting the most probable class predict the class with the minimum expected cost,
- Can be used to adjust the operating point for binary classification e.g. obtaining the ROC curve

Growing the exemplary decision tree

An exemplary test dataset T

a	b	c	class (C)	nodemap (S)
a_1	b_1	c_1	1	**1**
a_2	b_1	c_1	1	**1**
a_1	b_2	c_1	0	**1**
a_2	b_2	c_2	1	**1**
a_2	b_2	c_2	1	**1**

- The dataset T has three input attributes a, b, c and target class labels C and a nodemap S - current tree terminal node indices which are attached the given instances to. At the beginning of creating (growing) the tree we have only one node with a number 1 and it is a root and also a temporary leaf, so the nodemap has only one node.

Growing the exemplary decision tree

The impurity measure of splits

- In order to select a split condition first the impurity measure for all atrribute value splits e.g. $a(x) = a_1$ are calculated based on entropy e.g. for $a(x) = a_1$:

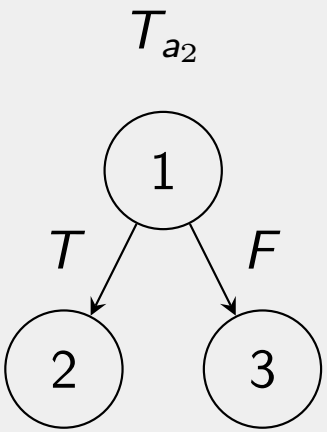
$$E_{a,a_1}(T) = -\frac{|T^0_{a,a_1}|}{|T_{a,a_1}|}\log_2(\frac{|T^0_{a,a_1}|}{|T_{a,a_1}|}) - \frac{|T^1_{a,a_1}|}{|T_{a,a_1}|}\log_2(\frac{|T^1_{a,a_1}|}{|T_{a,a_1}|}) =$$
$$-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$$

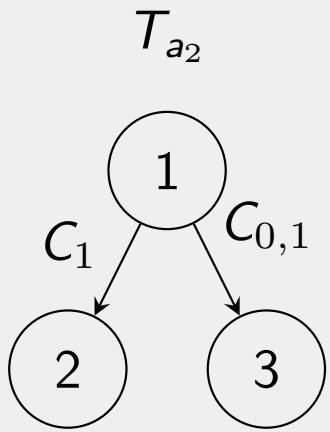
a	b	c	class (C)	nodemap (S)
$\boldsymbol{a}_1$	b_1	c_1	$\mathbf{1}$	1
a_2	b_1	c_1	1	1
$\boldsymbol{a}_1$	b_2	c_1	$\mathbf{0}$	1
a_2	b_2	c_2	1	1
a_2	b_2	c_2	1	1

Growing the exemplary decision tree

Creating new nodes based on the selected split - disjoint instances subsets

- The split with the smallest entropy equal to 0 or as close as possible to 0 is selected. If several entropies are equal to 0, the split e.g. with the largest number of instances is chosen: $a(x) = a_2$ has three instances with the same class 1 this means $C_{a(x)=a_2} = 1$ or $C(a_2)_1 = True$ and stop criteria for the same target class values are applied successfully.
- $E_{a,a_2}(T) = -\frac{|T^0_{a,a_2}|}{|T_{a,a_2}|}\log_2(\frac{|T^0_{a,a_2}|}{|T_{a,a_2}|}) - \frac{|T^1_{a,a_2}|}{|T_{a,a_2}|}\log_2(\frac{|T^1_{a,a_2}|}{|T_{a,a_2}|}) = -\frac{0}{3}\log_2(\frac{0}{3}) - \frac{3}{3}\log_2(\frac{3}{3}) = 0$





Growing the exemplary decision tree

Creating a leaf after the stop criterion is satisfied

- Based on the chosen split condition $a(x) = a_2$ the new nodemap is generated. The instances with the following value attributes $a(x) = a_2$ belong to the branch of the node 2 and as it was earlier mentioned the node 2 becomes the tree leaf (a terminal node with one class value) and the rest of instances belongs to the node 3. Thus, it is not necessary that all target classes in the branch subsets should be equal only to one class, sometimes the majority of the instances with the same class above the given threshold is the admisible criterion for creating leaves.

a	b	c	class (C)	nodemap (S)
a_1	b_1	c_1	1	3
a_2	b_1	c_1	**1**	**2**
a_1	b_2	c_1	0	3
a_2	b_2	c_2	**1**	**2**
a_2	b_2	c_2	**1**	**2**

Growing the exemplary decision tree

The next layer of child binary tree nodes: $2n$ and $2n+1$

- In the next steps for every next node instance subset the split with the smallest entropy equal to 0 or as close as possible to 0 is also chosen.
- But for the node 2 instances denoted in the nodemap the stop criterion is met and it can become a leaf which accurately classifies the attached test dataset instances and hopefully will achieve good accuracy for new similar instances chosen by the condition $a(x) = a_2$.
- For the node 3 instance subset the attribute b values are appriopriate: for each target class it has different values with entropy equal to 0. It creates two binary child nodes numbered $2n = 6$ and $2n + 1 = 7$ (only for binary trees).

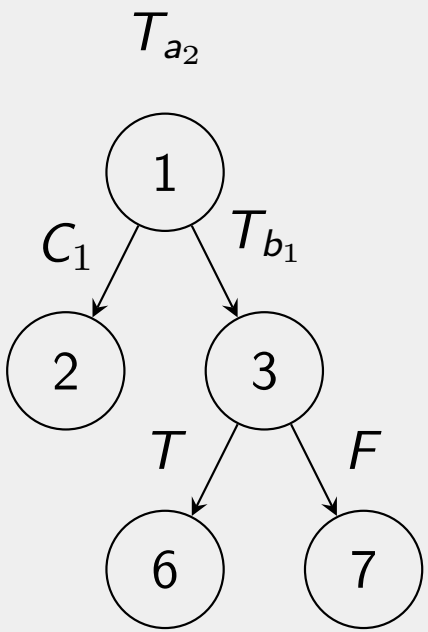
$E_{b,b_1}(T) = -\frac{|T^0_{b,b_1}|}{|T_{b,b_1}|}\log_2(\frac{|T^0_{b,b_1}|}{|T_{b,b_1}|}) - \frac{|T^1_{b,b_1}|}{|T_{b,b_1}|}\log_2(\frac{|T^1_{b,b_1}|}{|T_{b,b_1}|}) = -\frac{0}{1}\log_2(\frac{0}{1}) - \frac{1}{1}\log_2(\frac{1}{1}) = 0$

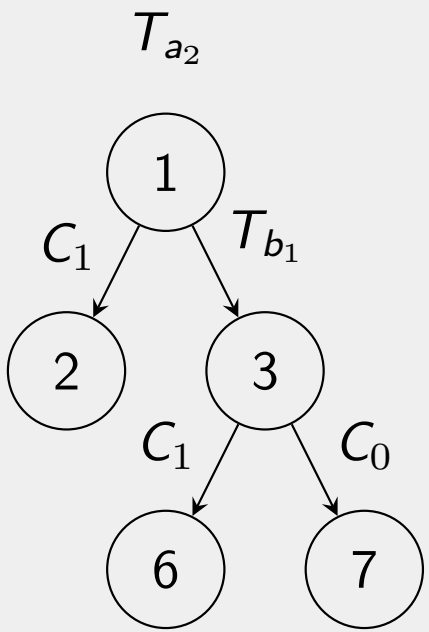
$E_{b,b_2}(T) = 0$

$E_{c,c_1}(T) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$

Growing the exemplary decision tree

Visualization of the final classifier generation





Confusion Matrix

Positives and Negatives

```
rci <- runif(nrow(census))
ci.train <- census[rci>=0.33,]
ci.val <- census[rci<0.33,]
ci.tree.d <- rpart(income ., ci.train)
ci.tree.d.pred <- predict(ci.tree.d, ci.val, type="c")
ci.tree.d.cm <- confmat(ci.tree.d.pred, ci.val$income)
```

	true 0	true 1
pred 0	True Negative TN	False Negative (II error) FN
pred 1	False Positive (I error) FP	True Positive TP

FN is much worse than FP e.g. not knowing about cancer is much worse than to be informed that cancer is detected and it is not really present

Confusion Matrix

Performance measures

- $\frac{FP+FN}{TP+TN+FP+FN}$ misclassification error
- $\frac{TP+TN}{TP+TN+FP+FN}$ accuracy
- $\frac{TP}{TP+FN}$ true positive rate recall, sensitivity - the ratio of correctly classified as positive to all positive
- $\frac{FP}{TN+FP}$ false positive rate - the ratio of incorrectly classified as positive to all negative
- $\frac{TP}{TP+FP}$ precision - the ratio of correctly classified as positive to all instances classified as positive
- $\frac{TN}{TN+FP}$ specificity 1 - fpr - the ratio of correctly classified as negatives to all negative instances
- Complementary pairs for detecting e.g. trivial classifiers which predicts only one class:
  - tpr - fpr
  - precision - recall
  - sensitivity - specificity

Confusion Matrix

Harmonic mean as a compromise between maximizing precision and recall

- F-measure $= \frac{1}{\frac{\frac{1}{precision}+\frac{1}{recall}}{2}}$
- Code in R: function(cm) 1/mean(c(1/precision(cm), 1/recall(cm)))

Handling more than two classes

- $1 - vs - 1$ one class is positive and another one is negative
- $1 - vs - rest$ one class is positive the remaining classes are negative: confmat01 - function in R

  cm.multi<-confmat01(ci.tree.d.pred.multi, ci.val$multiclass)

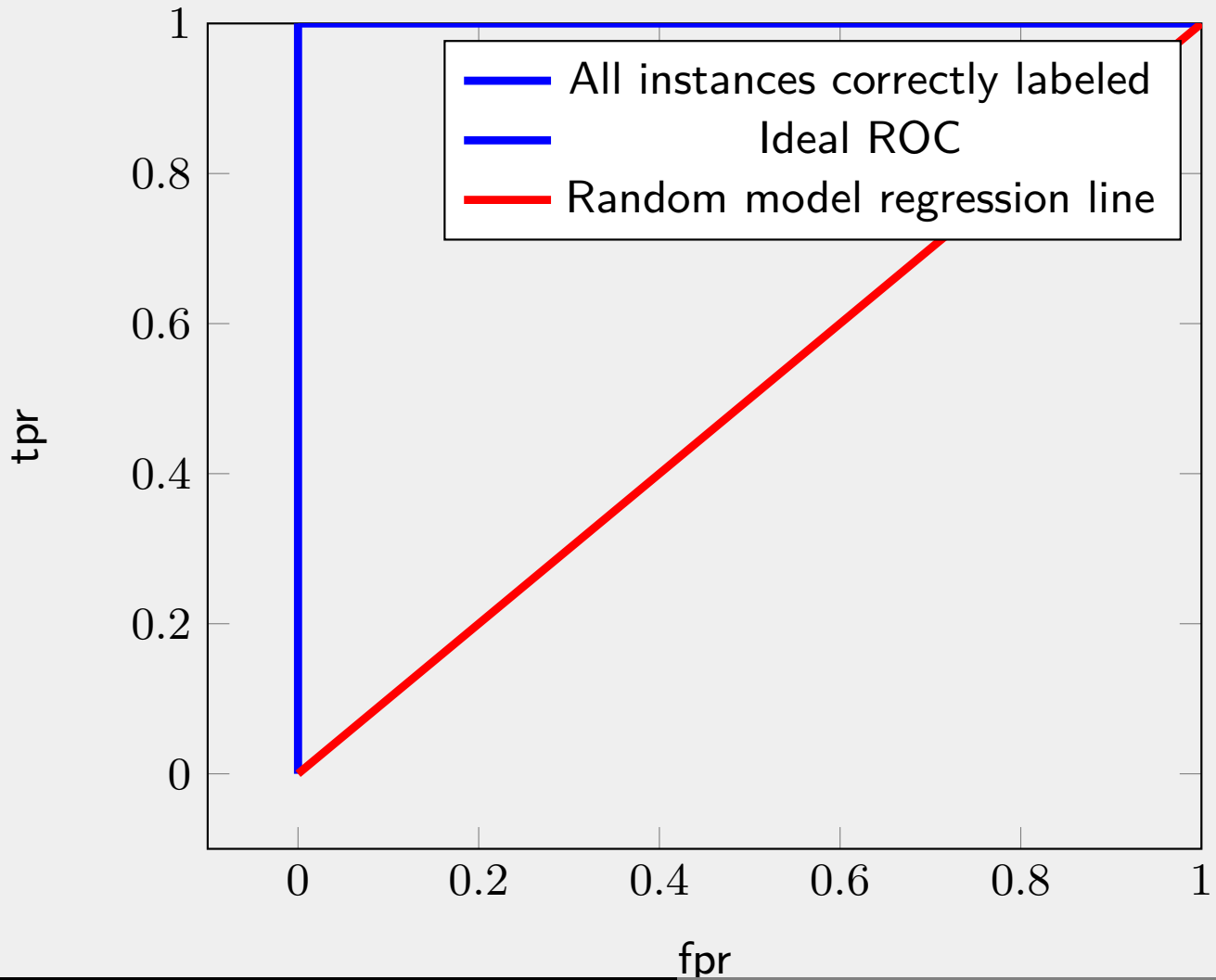
  rowMeans(sapply(cm.multi,function(cm) c(tpr=tpr(cm), fpr=fpr(cm), fm=f.measure(cm))))

Weighted confusion matrix

- For instances with weights: wconfmat
- tpr and fpr are the same for weighted and unweighted matrices if the same weights are assigned to one class instances.

Receiver operating characteristics (ROC)

- In order to obtain ROC characteristics it is a need for the classifier label output with probabilities of positive class 1, if you have a label domain with two values: 0 and 1.



ROC features

- All operating points should be above the red line: tpr should be higher than fpr or in the ideal case the only operating point should be (0,1) - the blue line.
- (1,0) - the worst operating point, (1,1) - the classifier always predicts class 1, (0,0) - the classifier always predicts class 0
- The output probabilities for the positive class are sorted to make a ROC curve.
- For each value of probability treated as a threshold for being the positive class (above the threshold) or the negative class (below the threshold) starting from the highest probability the all TP and FP are counted resulting in tpr and fpr rates, which give operating points and finally after joining them with lines - the ROC curve.
- The default optimal operating point is for the threshold equal to 0.5, but it is often not the best point.
- Moving along the ROC curve shifting the operating point is possible by changing the mentioned threshold.

ROC curve

Real classes	1	1	0	0	1	0	0	1	0	0
Predicted classes	1	1	1	1	1	0	0	0	0	0
Probs of positives	0.9	0.9	0.6	0.6	0.6	0.3	0.3	0.1	0.1	0.1

- In order to draw the ROC curve you forget about predicted classes and assume that probabilities are only for positive classes. Probabilities should be sorted in descending order and assigned to them classes from training set will determine its true positive or false positive nature.
- At the beginning of counting it should be assumed that all negative instances are true negatives TN and all positive instances are false negatives FN, true positives TP and false positives FP are equal to 0.
- For the same probability level:
  - for each such within that level probability with the assigned positive training class the true positive number TP is incremented and the false negative number FN is decremented.
  - for each such within that level probability with the assigned negative training class the false positive number FP is incremented and the true negative number TN is decremented.
  - At the end the true positive rate $tpr = \frac{TP}{TP+FN}$ and the false positive rate $fpr = \frac{FP}{TN+FP}$ should be computed and will become the new ROC point parameters.
- Thus, for each probability level the new ROC point will be obtained and all points joint by lines together will form the ROC curve to measure our classifier efficiency.