CSE 5523: Lecture Notes 7
Decision trees

Contents

7.1 Decision trees

Decision trees are simple machine learning algorithms with information-theoretic loss functions.

Specifically, they use conditional entropy as a loss function for predictor values as estimators (assuming a probability space $\langle Y \times X_1 \times \ldots \times X_V, 2^{Y \times X_1 \times \ldots \times X_V}, \mathsf{P} \rangle$):

$$v^*_{\mathcal{D}}, j^*_{\mathcal{D}} \stackrel{\text{def}}{=} \operatorname*{argmin}_{v \in \{1,\ldots,V\}, j \in X_v} \mathsf{H}(Y \,|\, X_v{=}j) \;+\; \mathsf{H}(Y \,|\, X_v{\neq}j)$$

Winning predictors $X_{v^*_{\mathcal{D}}}$ and values $j^*_{\mathcal{D}}$ are then used to define recursive splits of data $\mathcal{D}$ (until no non-trivial splits are possible):

$$\tau_{\mathcal{D}} = \begin{cases} \langle v_{\tau_{\mathcal{D}}}, j_{\tau_{\mathcal{D}}}, t_{\tau_{\mathcal{D}}}, u_{\tau_{\mathcal{D}}} \rangle = \langle v^*_{\mathcal{D}}, j^*_{\mathcal{D}}, \overbrace{\tau_{\{y,x_1,\ldots,x_V \in \mathcal{D} \,|\, x_{v^*_{\mathcal{D}}} = j^*_{\mathcal{D}}\}}}^{\text{subtree for positive subset}}, \overbrace{\tau_{\{y,x_1,\ldots,x_V \in \mathcal{D} \,|\, x_{v^*_{\mathcal{D}}} \neq j^*_{\mathcal{D}}\}}}^{\text{subtree for negative subset}} \rangle & \text{if } 0 < \tilde{\mathsf{P}}_{\mathcal{D}}(x_{v^*_{\mathcal{D}}} = j^*_{\mathcal{D}}) < 1 \\ \operatorname{argmax}_y \; \tilde{\mathsf{P}}_{\mathcal{D}}(y) & \text{otherwise} \end{cases}$$

The result is a tree $\tau_{\mathcal{D}}$ with tuples of subtrees $t_{\tau_{\mathcal{D}}}, u_{\tau_{\mathcal{D}}}$ as non-terminals and y values as terminals. (Here, $\tilde{\mathsf{P}}_{\mathcal{D}}(\ldots)$ is the empirical probability of '$\ldots$' in data $\mathcal{D}$.)

Once trained (i.e. at test time), this tree $\tau_{\mathcal{D}}$ is queried recursively for each test data item $x_1, \ldots, x_V$:

$$\delta_{\tau_{\mathcal{D}}}(x_1, \ldots, x_V) = \begin{cases} \tau_{\mathcal{D}} & \text{if } \tau_{\mathcal{D}} \in Y \\ \delta_{t_{\tau_{\mathcal{D}}}}(x_1, \ldots, x_V) & \text{if } \tau_{\mathcal{D}} \notin Y \wedge x_{v_{\tau_{\mathcal{D}}}} = j_{\tau_{\mathcal{D}}} \\ \delta_{u_{\tau_{\mathcal{D}}}}(x_1, \ldots, x_V) & \text{if } \tau_{\mathcal{D}} \notin Y \wedge x_{v_{\tau_{\mathcal{D}}}} \neq j_{\tau_{\mathcal{D}}} \end{cases}$$

(where $\delta_{\ldots}$ is a decision procedure parametrized by '$\ldots$', as defined in the earlier notes).

7.2 Sample code

Here is a sample decision tree learner (which halts at depth 3 to save time):

```
import sys
import numpy
import pandas
```

```
def I(p):                                   ## calculate information (I) of distrib p
  return -p*numpy.log2(p) if p>0.0 else 0.0

def H( Y ):                                 ## calculate entropy (H)
  return sum([ I( len(Y[Y==y])/len(Y) )  for y in Y.unique() ])

def split( D ):                             ## calc argmin_v,j of h using min of <h,v,j>
  YandXvars = D.columns.values.tolist()
  return min([ (   len(D[D[v]==j])/len(D) * H( D[ D[v]==j ][ YandXvars[0] ] )
                 + len(D[D[v]!=j])/len(D) * H( D[ D[v]!=j ][ YandXvars[0] ] ), v, j)
               for v in YandXvars[1:]  for j in D[v].unique() ])

def train( D, d=0 ):
  if len(D)>0 and d<3:                      ## form tree if below arbitrary depth cutoff
    h,v,j = split(D)                        ## call our split function to get best split
    if len(D[D[v]==j]) > 0 and len(D[D[v]!=j]) > 0:              ## recurse, divide data
      return( v, j, train(D[D[v]==j],d+1), train(D[D[v]!=j],d+1) )
  y = D.columns.values.tolist()[0]          ## y is the first column of the data frame
  return D[y].value_counts().idxmax()       ## if too deep, return the most common y value

D = pandas.read_csv( sys.argv[1] )          ## read in data

t = train(D)                                ## call our train function

print( t )
```

When run on 'investor-train.csv':

```
investor,birthyr,employ,hadjob,industryclass,ownhome,votereg,phone,internethome,internetwork,union
1.0,1951,5.0,2.0,18.0,1.0,1,1.0,1.0,1.0,1
1.0,1968,7.0,2.0,,1.0,1,2.0,1.0,3.0,2
1.0,1945,2.0,,20.0,1.0,1,2.0,1.0,3.0,3
1.0,1973,4.0,1.0,20.0,1.0,1,2.0,1.0,1.0,3
2.0,1956,5.0,2.0,21.0,1.0,1,2.0,1.0,3.0,3
1.0,1960,1.0,,18.0,1.0,1,1.0,1.0,1.0,3
1.0,1987,2.0,,19.0,2.0,1,2.0,1.0,1.0,3
2.0,1963,1.0,,23.0,1.0,1,1.0,1.0,1.0,2
1.0,1984,1.0,,22.0,1.0,1,2.0,1.0,1.0,3

⋮
```

It outputs this tree (my indentation):

```
('ownhome', 1.0, ('votereg', 1, ('internetwork', 3.0, 2.0, 1.0),
                                 ('union', 3, 2.0, 2.0)),
                 ('internetwork', 3.0, ('employ', 6.0, 2.0, 2.0),
                                       ('employ', 1.0, 2.0, 2.0)))
```