Introduction to Machine Learning **Fall Semester, 2013**

Recitation 12: January 5

Lecturer: Mariano Schain *Scribe: ym*

12.1 Project

There will be an optional intermediate stage, where groups that would like can submit their classification on the test data and receive a feedback (the percent of errors) Details in the web site.

12.2 Decision Trees

12.2.1 Terminology and Reminder

Assume a binary classification setting (for every training sample, let f be the binary label). We like to decide in each node on the split, i.e., the predicate h to assign to the node. The local parameters are $q = \Pr[f = 1]$, which is the fraction of 1s in the examples reaching the node, $u = pr[h = 0]$[1] is the fraction of samples for which $h = 0$ out of the samples reaching the node, $p = \Pr[f = 1|h = 0]$ is the fraction of 1s in the samples reaching the node and having $h = 0$, and $r = \Pr[f = 1|h = 1]$ is the fraction of 1s in the samples reaching the node and having $h = 1$. We have that $q = up + (1 - u)r$. (See Figure 12.1.)

Recall the decision-tree algorithm from class: We use a strictly convex node index function $v(\cdot)$[2] that associates a value to a node as a function of the proportion of positively labeled examples in the node (q using our above terminology). Now, by strict convexity of $v(\cdot)$ we have

$$v(q) > u \cdot v(p) + (1 - u)v(r)$$

And at a given node we seek to find a predicate h that splits in a way that mostly reduces the right hand side of the above inequality (the resulting node *potential*).

[1] We use $h = 0$ to indicate that the predicate h is *false* and $h = 1$ for the case h is *true*

[2] An example of a split index is $v(p) = -p \log_2 p - (1-p) \log_2(1-p)$ which is the binary entropy function. (In class we normalized by multiplying by a half, but this will not make a difference.)

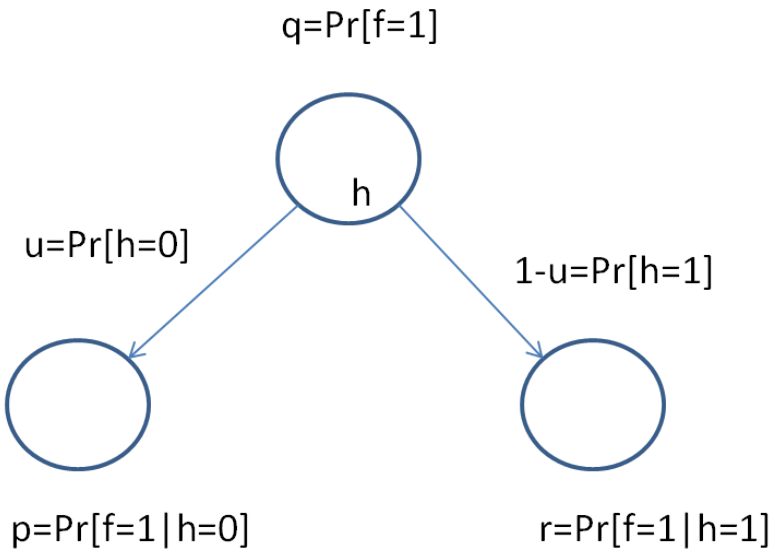


Figure 12.1: The split in a node

12.2.2 Instability Example

We consider the sample 2-feature binary labeled data in figure 12.2a [3]. The root's optimal decision stump $h =$ "$x_1 < 0.6$" reduces the potential [4] from the initial 1 (since the sample contains an equall number of positive and negative samples) to

$$\frac{10}{16}v\left(\frac{7}{10}\right) + \frac{6}{16}v\left(\frac{1}{6}\right) \approx 0.79$$

We continue performing the splits and derive the decision tree of Figure 12.3.

We can now consider what will happen if we slightly modify the location of a single point as follows. (See Figure 12.2b.)

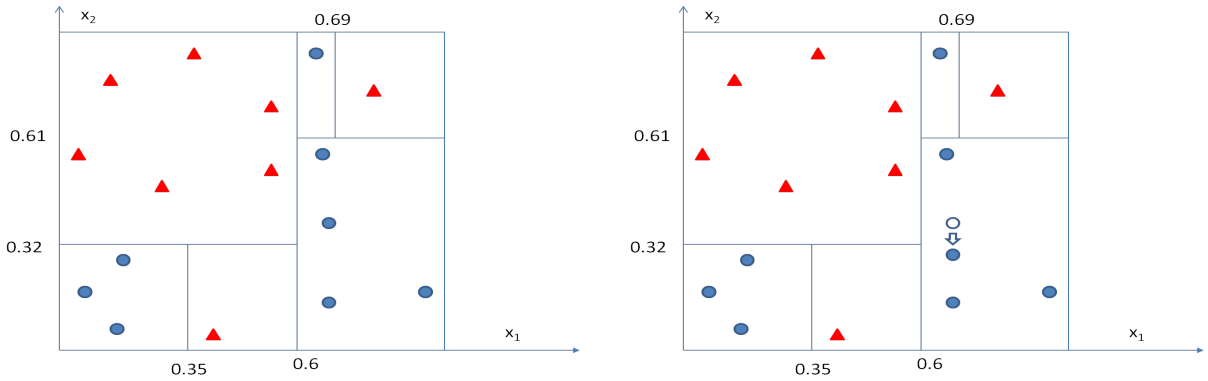
The modified data still has the root split $h =$ "$x_1 < 0.6$" resulting in the same value ≈ 0.79, but for the root split $h =$ "$x_2 < 0.32$" we have

$$\frac{7}{16}v\left(\frac{1}{7}\right) + \frac{9}{16}v\left(\frac{7}{9}\right) \approx 0.68 < 0.79$$

This implies that the minor change will change the optimal predicate at the root and might impact the entire tree.

[3] Example from http://www.lsv.uni-saarland.de/pattern_sr_ws0607/psr_0607_Chap10.pdf, slide 30

[4] We use the entropy function throughout.



(a) Original sample set

(b) Slight change in one sample.

Figure 12.2: Example of data for decision tree instability. Triangles are positively labeled and circles are negatively labeled

12.3 Using decision trees in other algorithms

12.3.1 Boosting Trees

We run AdaBoost as it is, with simple trees (e.g. with a single split) as weak learners. At each AdaBoost iteration, the weight w_i assigned to each training sample (x_i, y_i) is used to compute the probabilities q, u, p, and r of Figure 12.1: Instead of just counting positive and negative labeles, we need to compute frequencies we weigh them according to the weights. For example we replace $\Pr[f = 1]$ by $\sum_{i:f(x_i)=1} w_i/W$, where $W = \sum_{i=1}^{m} w_i$. Combining the decision trees is done using the standard AdaBoost weights α_t.

12.3.2 Random Forest

We first review the Bagging and Stacking patterns:

Bagging

In order to reduce variance, the original sample set S is sub-sampled (with repetitions) to create k new sample sets $S_1, \ldots S_k$ which are fed to the learning algorithm A. The k resulting hypotheses $h_1, \ldots h_k$ form a new (resulting) hypothesis that given x classifies using the majority among $\{h_1(x), \ldots, h_k(x)\}$. See Figure 12.4

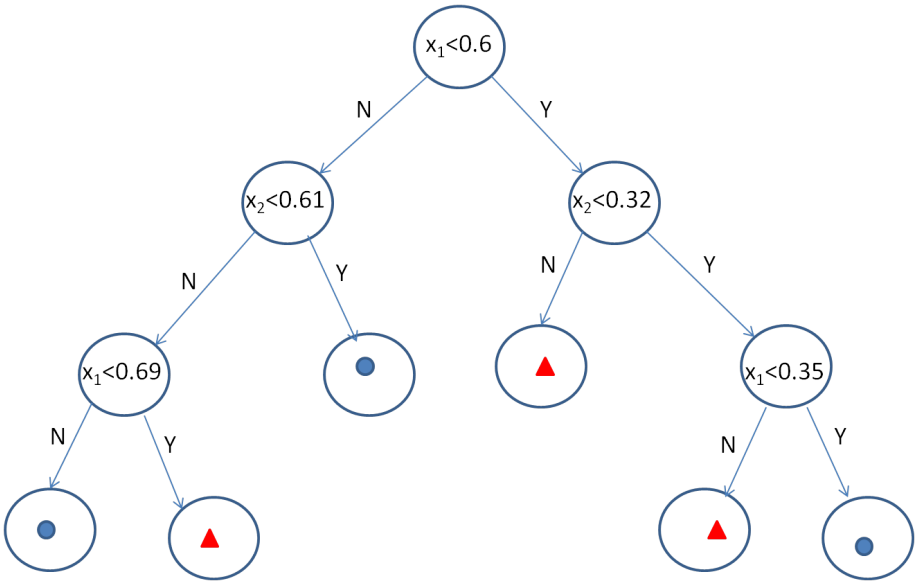


Figure 12.3: The tree that is built

Stacking

Here, in order to find a way to best combine learning algorithms, the original sample S is fed to the k independent learning algorithms $A_1, \dots A_k$, resulting in hypotheses $h_1, \dots, h_k$ (respectively) which are used to create a new sample set

$$S' = \{(h_1(x), \dots, h_k(x)), y) | (x, y) \in S\}$$

S' may now be used by any other learning algorithm to result in a hypothesis h that classifies by first mapping any input x to $(h_1(x), \dots, h_k(x))$.

Scaling Random Forest

The Random Forest flow (see Figure 12.6) has ingredients from both Bagging and Stacking patterns: First, as in Bagging, the original sample set S is sub-sampled to k sample sets $S_1, \dots, S_k$. Subsequently, as in Stacking, each sample set S_i is used to build a random decision tree h_i (random in the the sense that the attributes used for the nodes predicates are randomly restricted as explained in class) and in that sense the algorithm used to build h_i is designated A_i. The resulting decision trees are finally combined to a majority voting h. Note that the k threads of sub-sampling S and using A_i to build h_i are independent and may therefore be easily paralelized! [5]

[5] Challenge: Identify other machine learning patterns and algorithms that are similarly comprised of independent threads and therefore may also be easily parallelised.

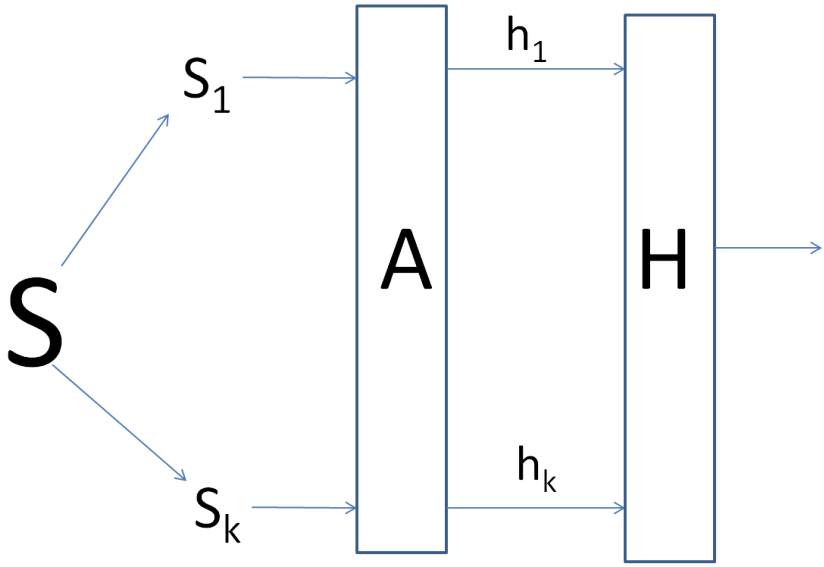


Figure 12.4: Flow of Bagging

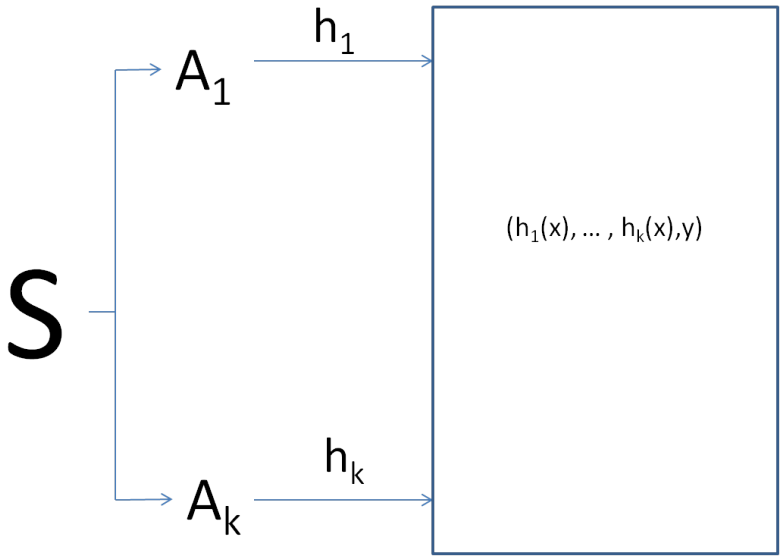


Figure 12.5: Flow of Stacking

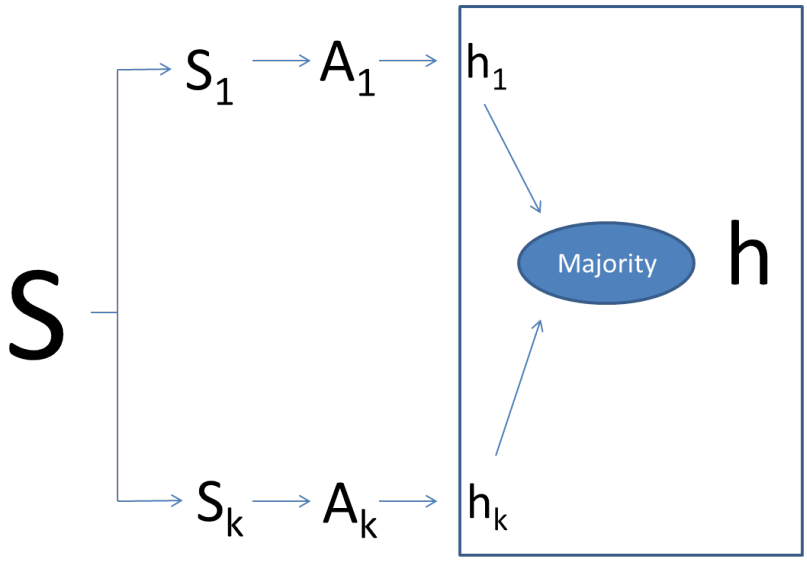


Figure 12.6: Flow of Random Forest