I think that most of these algorithms trade off the SIZE of the classifier (a measure of likely generalization ability) versus the ERROR WORSENING of the classifier. So you could do this:

1. Compute the score f_n for each node n which roots a subtree
2. Let f_n^* be the highest such score. If $f_n^* \geq a$ for some threshold $a > 1$, then replace n with a leaf.
3. Go to 1

So the question is what constitutes f_n. Here are some variables of interest: s_{old} is the old size of the rooted subtree. s_{new} is the new size of the rooted subtree (normally $= 1$). S_{old} is the old size of the entire tree. S_{new} is the new size of the entire tree. Likewise e_{old} is the error rate in the old subtree. e_{new} is the error rate in the new subtree (that is, the leaf). And E_{old} and E_{new} are the error rates of the old and new entire trees.

You want the tree to get as SMALL as possible while its error staying as SMALL as possible. Let's consider just looking at local improvements. Then you have

$$\gamma(s_{\text{old}} - s_{\text{snew}}) + \beta(e_{\text{old}} - e_{\text{new}}) \stackrel{?}{\geq} a$$

I don't like the linear nature: this suggests that going from 10% error to 15% error is better than going from 55% error to 65% error, which doesn't sound right to me. How about:

$$\gamma \frac{s_{\text{old}}}{s_{\text{new}}} + \beta \frac{e_{\text{old}}}{e_{\text{new}}} \stackrel{?}{\geq} a$$

This is too local too. Maybe it's better to consider the impact on the tree as a whole. So you might have:

$$\gamma \frac{S_{\text{old}}}{S_{\text{new}}} + \beta \frac{E_{\text{old}}}{E_{\text{new}}} \stackrel{?}{\geq} a$$

Perhaps it might make sense to go "fully nonlinear", I dunno:

$$\frac{S_{\text{old}}\ E_{\text{old}}}{S_{\text{new}}\ E_{\text{new}}} \stackrel{?}{\geq} a$$

Finally, we might want to change S to $\log_n S$, so that big trees are less likely to get pruned.

$$\gamma \frac{\log_n S_{\text{old}}}{\log_n S_{\text{new}}} + \beta \frac{E_{\text{old}}}{E_{\text{new}}} \stackrel{?}{\geq} a \qquad \text{or} \qquad \frac{\log_n S_{\text{old}}\ E_{\text{old}}}{\log_n S_{\text{new}}\ E_{\text{new}}} \stackrel{?}{\geq} a$$

0.1 Revised Idea - I

First I thought that the PEP also makes pruning decision based on the global error (total misclassification on the whole original examples), however after discussing with sean I realized that the PEP calculates the error for a node based on the local examples (number of examples reaching to that node). But I was a bit uncomfortable about this idea, what I would do is to make the decision on the misclassification on the total number of examples (i.e. E_*, according to sean's notation). Here we have two goals –

1. A node at a deeper position in the tree is more capable of capturing noise in the data, so we are eager to prune it.
2. A node which makes less misclassification error on the whole examples (if it were replaces by a majority class label), we are eager to prune it.

But what if node n_i and node n_j having depth of $depth(n_i)$ and $depth(n_j)$, respectively. Where

$$depth(n_j) > depth(n_i) \quad \text{and}$$

$$Error_{total}(n_j) < Error_{total}(n_j) \quad ?$$

Here $Error_{total}(x)$ means the misclassification error made by node x on the whole example set and the above condition may happen when one class of data is surrounded by another class of data (i.e. multiple co-centric clusters) – although in such cases, SVM is a better solution. Anyway, eventually we are having a multiobjective problem here – because if we consider the criteria 1 above, we need to prune n_j, but if we consider the criteria 2, n_i is going to be pruned.

So, my idea is to do a non-dominated sort on all the nodes and create a Pareto-front. i.e. We need to consider like –

$$f_1(n_i) = \max_i [depth(n_i)] \quad \text{and}$$

$$f_2(n_i) = \min_i [Error_{total}(n_i)]$$

and then start pruning from the non-dominated nodes. Morever setting a good threshold a is not easy, this method I think is totally parameter free.

0.2 Revised Idea - II

As the PEP makes its decision solely depending on the misclassification error made in the local node (number of examples reaching to that node), why don't we pour some global information to it? It can be done like this –

We have to calculate two types of error here, namely *local error* and *global error*. For a particular node n_i, lets define its global and local error as follows – the total number of examples reaching n_i is N_i and the total number of misclassified examples is e_i at n_i, i n_i were replaced by a majority class label, then –

$$Error_{local}(n_i) = \frac{e_i}{N_i}$$

and if we are learning the tree with total N number of examples, then –

$$Error_{global}(n_i) = \frac{e_i}{N}$$

and the total error at node n_i is –

$$Error_{total}(n_i) = Error_{local}(n_i) + \gamma Error_{global}(n_i)$$

where $\gamma = \frac{depth(n_i)}{depth_{total}}$ which means that a node on the deeper level, will get more share on $Error_{global}$. Or we can also make a nonlinear relation like –

$$Error_{total}(n_i) = (Error_{local}(n_i))^{\gamma Error_{global}(n_i)}$$

But how do I decide the threshold a here ??