Pruning a tree and assessing its quality

When constructing a tree, it is important to know when to stop. A tree which is too big will fit the training data very well, but predict future data badly. On the other hand, a tree which is too small will perform consistently badly. To make the decision, you need a way of estimating how well a tree will perform on future data (not just the training data).

Computational learning theory—Construct a sampling distribution for the training error rate given the population error rate and type of model. From the sampling distribution, derive a confidence interval on the population error rate. AIC (handout 17) is an instance of this method.

Holdout method and cross-validation (handout 19)—Train on part of the data, test on the rest, to estimate the performance of a given model type. For a given tree size, these methods will estimate how well it will perform on future data. Do this for several tree sizes, pick the best, and then fit a tree of that size to all the data.

Suppose you have a tree of size 5. How do you get a tree of size 4? Starting from scratch is wasteful. Instead you can **prune** away the least informative split. Even fancier is to use cross-validation to decide which split to prune.

Scoring a tree on a test set:

Misclassification rate does not account for costs.

$$Misclass = \frac{1}{N}\sum_i I(\text{predicted class of point } i \neq \text{actual class of point } i)$$

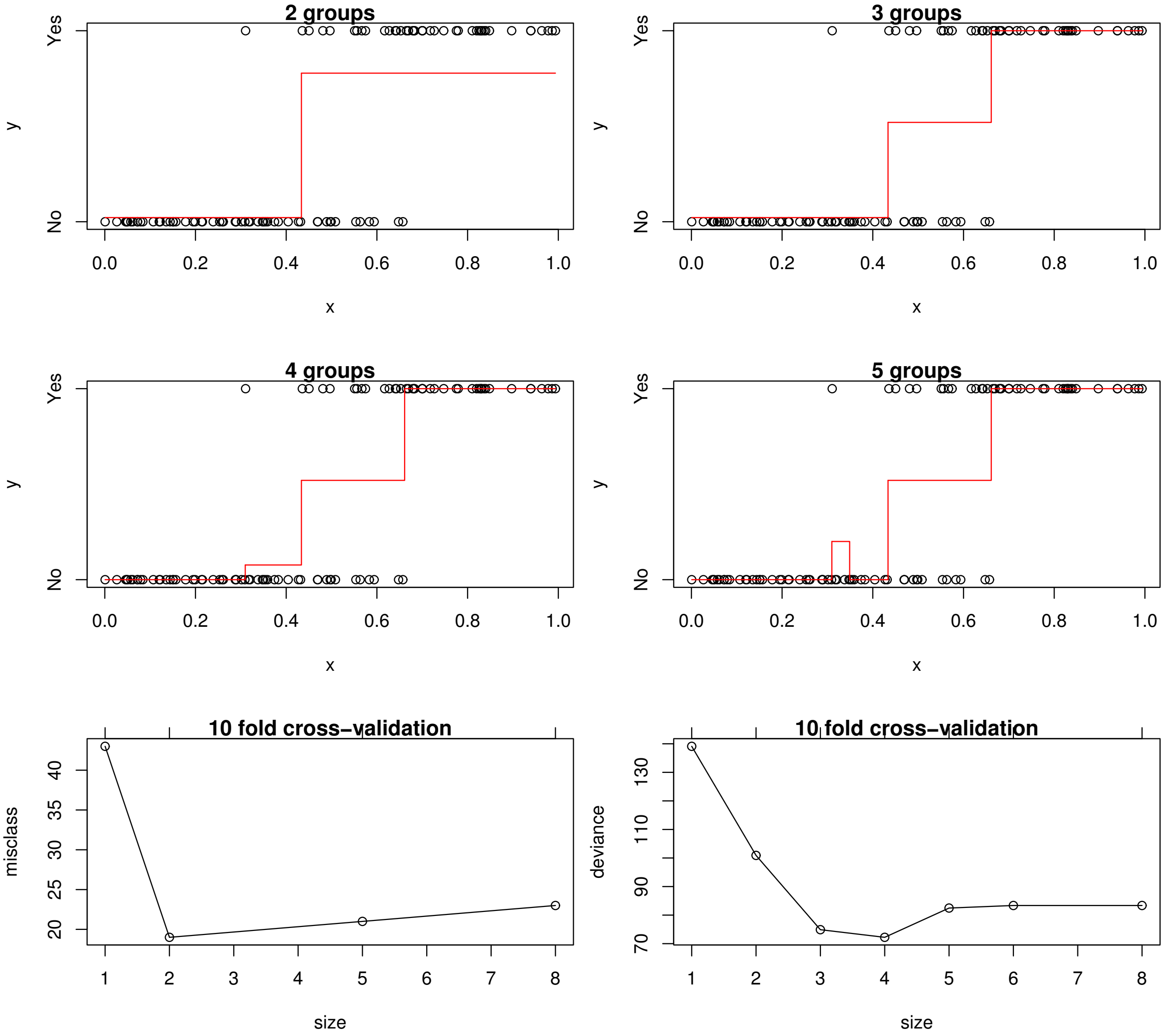
Misclassifiation cost requires you to know the costs in advance.

$$Cost = \frac{1}{N}\sum_i Cost(\text{predicted class of point } i|\text{actual class of point } i)$$

Deviance evaluates the probabilities themselves, making the tree suitable for any cost matrix. It also provides a more precise score than the other two.

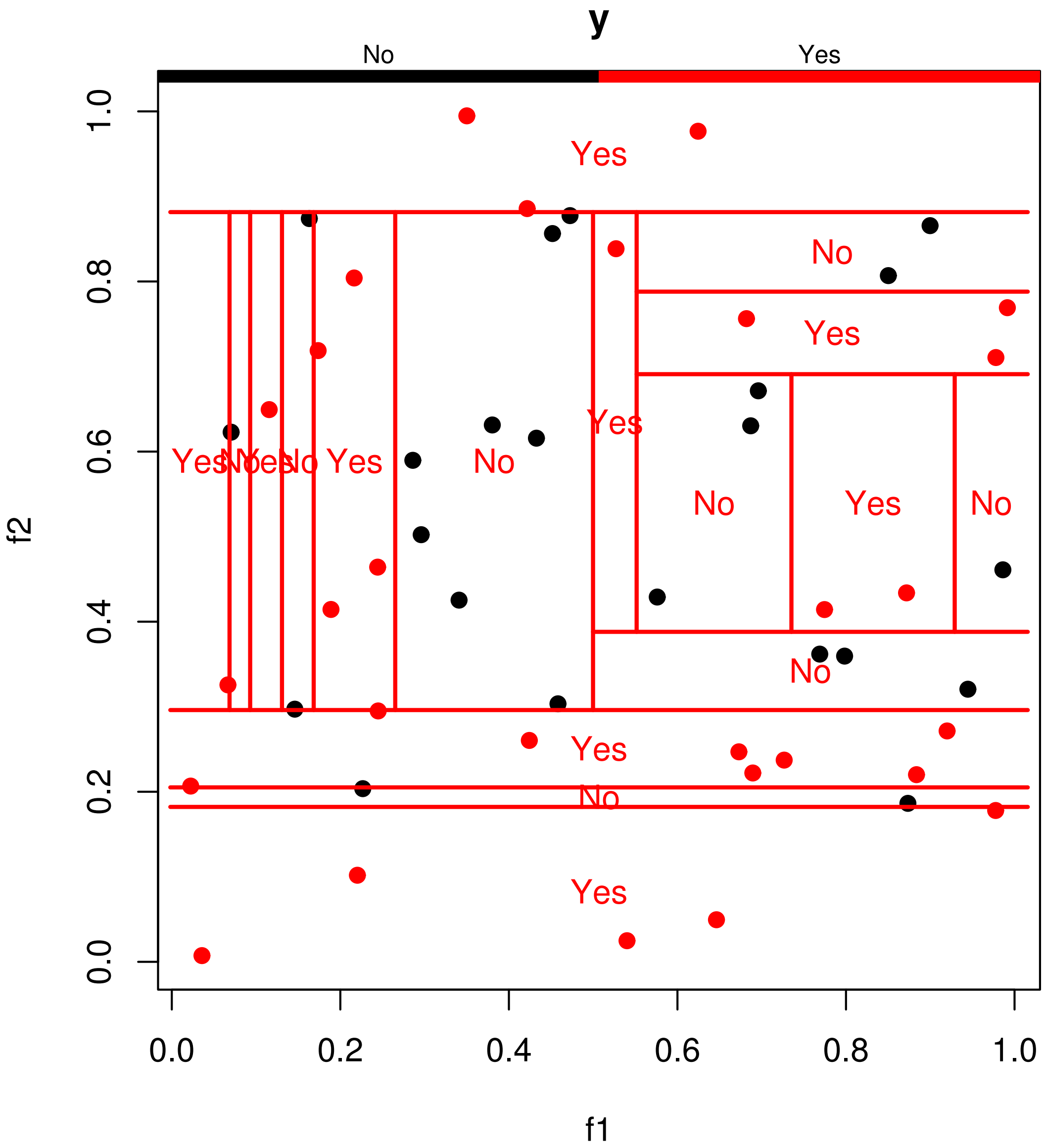
$$Deviance = \frac{-2}{N}\sum_i \log p(\text{actual class of point } i|\text{point } i, \text{model})$$

For example, if a future customer did churn, and the tree gave a churn probability of 0.4 for that customer, then the deviance is $-2\log(0.4)$. If the tree gives probability 1 to the correct answer, it has deviance zero. If the tree gives probability 0 to the correct answer, it has deviance ∞. Thus it is good to be confident if you are right and bad to be confident if you are wrong.

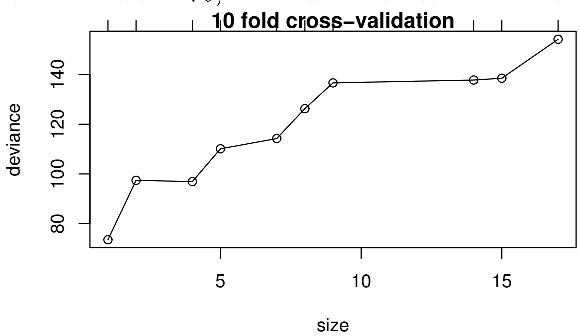


To minimize misclassification rate, you only need to know if $p(Yes) > 0.5$ (the **decision boundary**). This can be done with a small tree. To minimize deviance, you need accurate probabilities, which requires a bigger tree. In this case, cross-validation suggests 4 groups as providing the best fit without over-fitting.

An extreme example of overfitting:



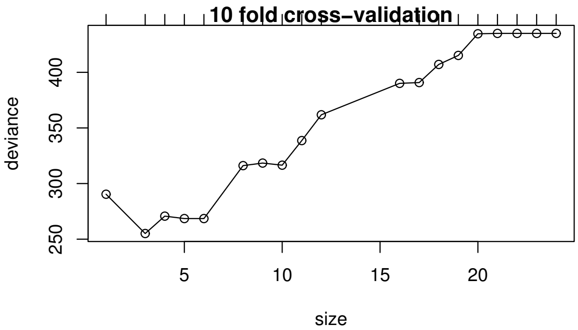
This tree makes no errors on the training set. But in fact the y values were generated randomly, so on test data the error rate will be 50%, no matter what the tree is.

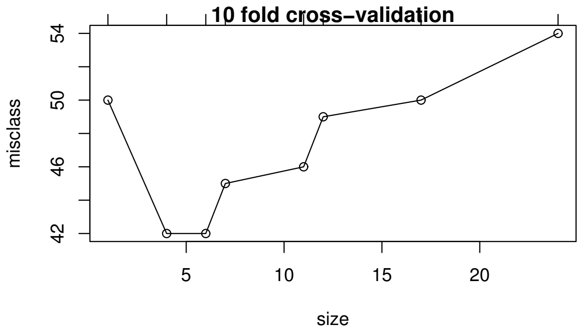


Churn dataset, holdout method, train on 10% of data:

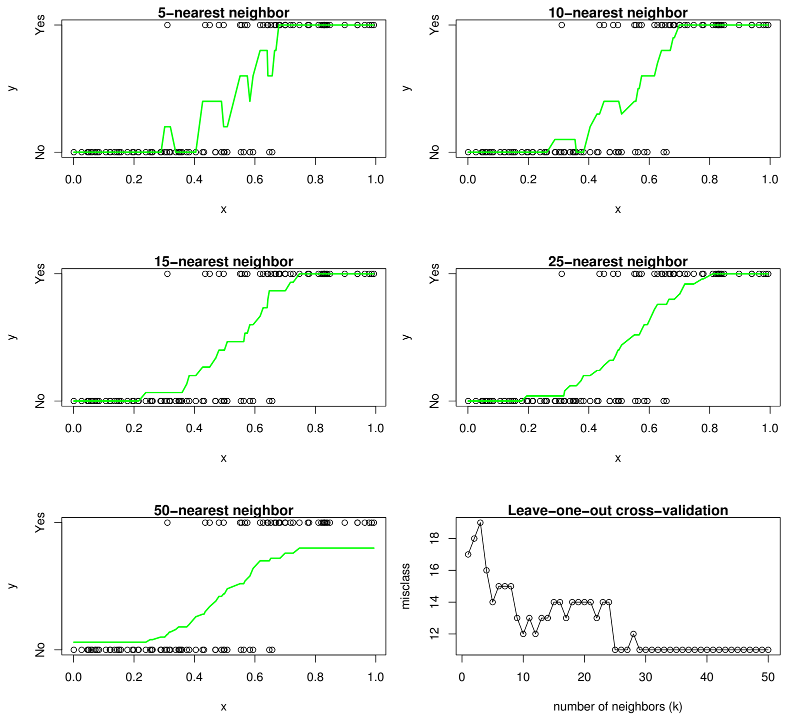
	training set	testing set
21 leaves, misclass	6%	13%
21 leaves, deviance	0.23	1.15
4 leaves, misclass	10%	12.5%
4 leaves, deviance	0.61	0.72
3 leaves, misclass	13%	13.5%
3 leaves, deviance	0.66	0.72

The default tree has 21 leaves, which is apparently too much. Cross-validation on misclassification rate (based on the training set alone) picks 4 leaves. Cross-validation on deviance picks 3 leaves.



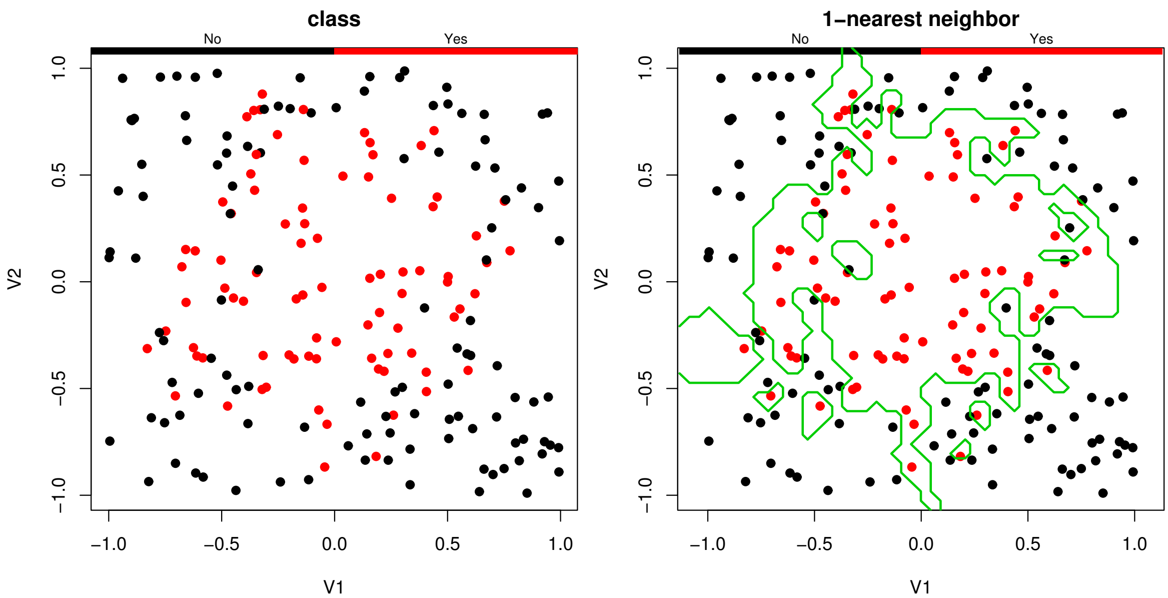


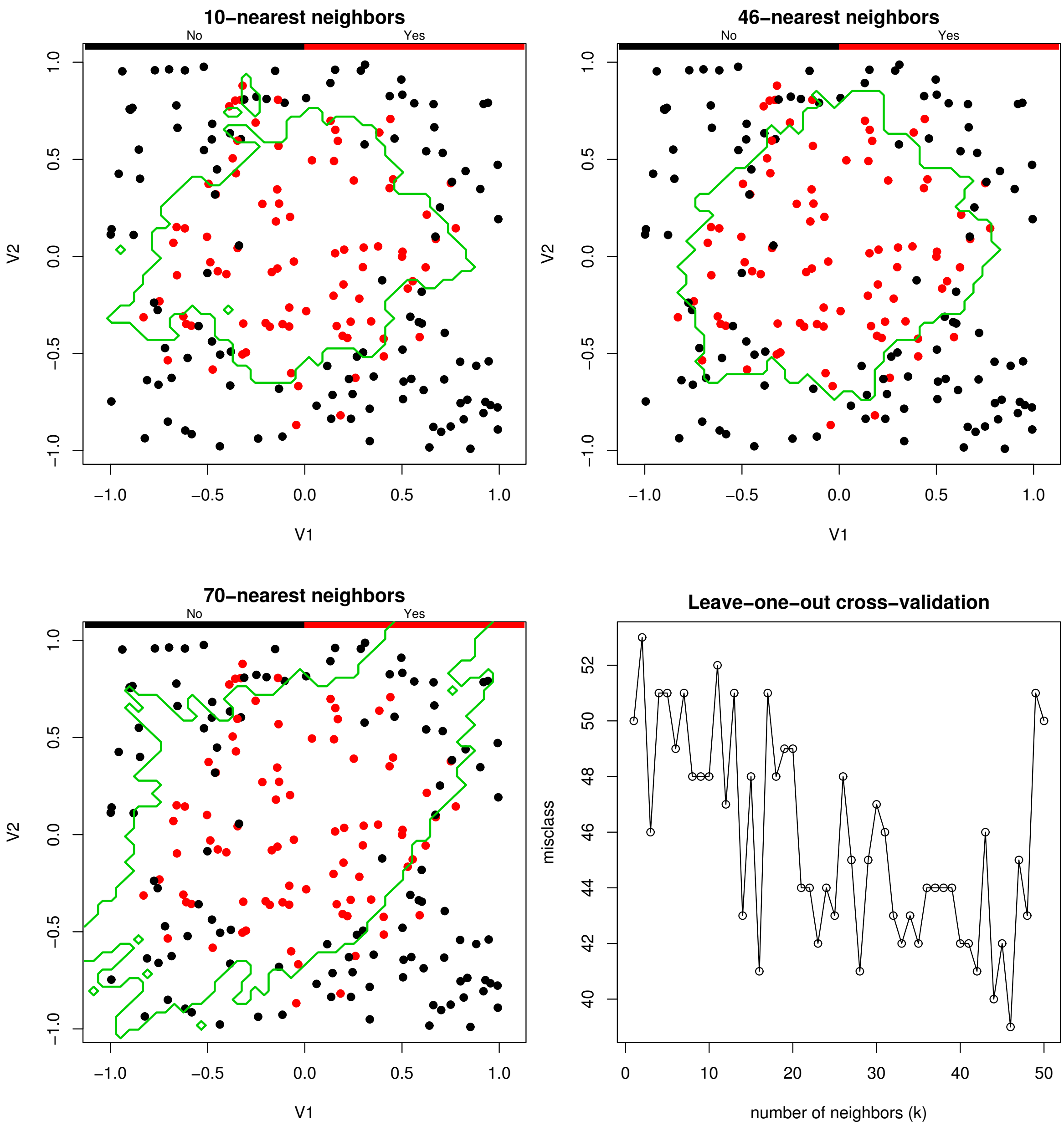
The basic nearest-neighbor classifier doesn't provide probabilities. But suppose we take k nearest neighbors, instead of 1, to estimate the local class probabilities.



Cross-validation picks $k = 25$. Leave-one-out is normally used with k-NN because it is especially simple.

Using $k > 1$ tends to give lower misclassification rate as well, due to noise averaging. An example in two dimensions:





The decision boundary gets smoother with larger k. On a test set, $k = 1$ has 28% misclassification error, while $k = 46$ has 22%. k-NN is good at rough and curvy decision boundaries. What would the decision boundary look like for a tree?