Applied Statistical Genetics in R

for population-based association studies

PART IV: High-dimensional data methods

Outline

Classification and regression trees (CART)

- Classification and regression trees (CARTs) are an approach to discovering relationships among a large number of independent (predictor) variables and a categorical or continuous trait.
- Classification trees are applied to categorical outcomes while regression trees apply to continuous traits.
- Both involve the application of a recursive algorithm that aims to partition individuals into groups in a way that minimizes the within group heterogeneity.

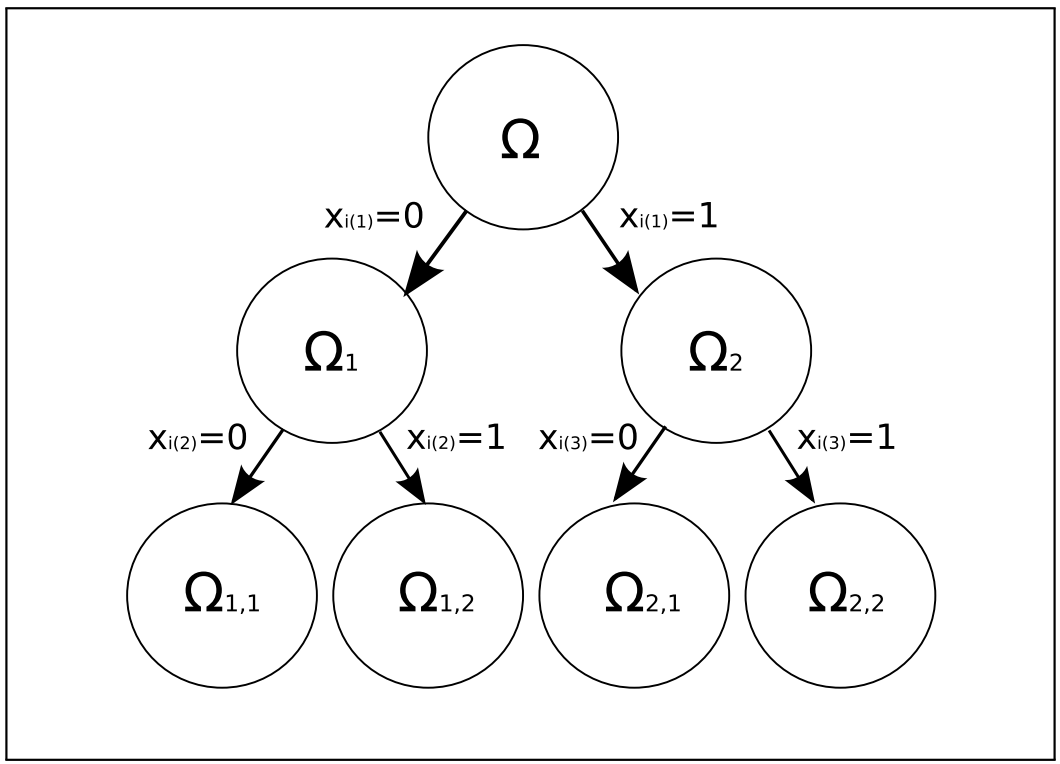
Building a tree

- A tree is constructed by first determining the variable x_{ik}, among the set of potential predictors $\mathbf{x}_i$, that is *most predictive* of the trait y_i.
- The phrase "most predictive" is used loosely here and is defined explicitly below.
- Individuals in our sample are then divided into two groups based on the value of x_{ik}.
- Suppose, for example, that x_{ik} is a binary indicator for the presence of the variant allele at the kth SNP under investigation. If x_{ik} is identified as the most predictive variable, then individuals with at least one variant allele at this SNP are assigned to one group while individuals that are homozygous wildtype are assigned to a second group.

Building a tree

- Process is repeated recursively to generate a tree:

Figure: Tree structure



Building a tree

- Each circle is called a *node*.
- The first node is termed the *root node* or *parent node*.
- Subsequent splits result in the formation of left and right *daughter nodes*.
- A node is simply a representation of a group of individuals, which we denote with Ω.

Building a tree

- Importantly, at each stage of the splitting algorithm, we are identifying the variable that is *most* predictive of the trait of interest among all variables under consideration.
- In general, this is not based on a formal statistical test. That is, these variables may not be significantly associated with the trait, even though they are more predictive than any other variable.
- Therefore, after building a tree in this manner, it is important to "prune" it back in order to avoid over-fitting.

Splitting rules

- Generating a tree typically begins by defining a measure of *node impurity*, or heterogeneity,
- The splitting variable is the variable the that maximizes the reduction in this measure.
- Let the impurity for node Ω be given by $I(\Omega)$. Formally, the goal is to choose the split that maximizes:

$$\phi = I(\Omega) - I(\Omega_L) - I(\Omega_R)$$

where Ω_L and Ω_R are the left and right daughter nodes of Ω, respectively.

Splitting rules: Binary Trait

- Consider a binary trait, Y that takes on the values 0 and 1, such as an indicator for the presence of disease. In this case we replace $I(\Omega)$ with $\pi i(\Omega)$ where π is the proportion of cases in Ω, giving us:

$$\phi = i(\Omega) - \pi_L i(\Omega_L) - \pi_R i(\Omega_R)$$

- One of the simplest measure of node impurity that has been proposed for this binary trait setting is given by:

$$i(\Omega) = \min(p_\Omega, 1 - p_\Omega)$$

where $p_\Omega = Pr(Y = 1|\Omega)$ is the probability of being a case given membership to node Ω.

- This impurity measure is commonly referred to as the *Bayes error*, *minimum error* or *misclassification cost*.

Splitting rules: Binary Trait

- Another commonly used measure of impurity is the *Gini index*, also called the *nearest neighbor error*, which is defined as:

$$i(\Omega) = 2p_\Omega(1 - p_\Omega)$$

  where again p_Ω is the conditional probability of being a case.
- Represents the sum of variances of bernoulli random variables.

Splitting rules: Binary Trait

- **Example 1** (Creating a classification tree):
  - Suppose we are interested in identifying polymorphisms within the protease region of the viral genome that are associated with greater *in vitro* sensitivity to nelfinavir (NFV) than indinavir (IDV) based on the Virco data. That is, we aim to characterize the relationship between mutations in the variables labelled `P1`, ..., `P99` and a binary indicator for whether `IDV.Fold` is greater than `NFV.fold`.
  - We begin by defining a data frame, entitled `VircoGeno`, that contains binary indicators for the presence of a mutation at each of the 99 sites within the protease region:

```
> attach(virco)
> VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")
```

Splitting rules: Binary Trait

- (example continued)
  - We then define our trait and construct a classification tree as follows. Note that we define our trait as a `factor` variable since we interested in creating a classification tree. We additionally specify `method="class"`, though this is the default setting when the trait is a factor:

```
> Trait <- as.factor(IDV.Fold > NFV.Fold)
> library(rpart)
> ClassTree <- rpart(Trait~., method="class", data=VircoGeno)
> ClassTree
```

Splitting rules: Binary Trait

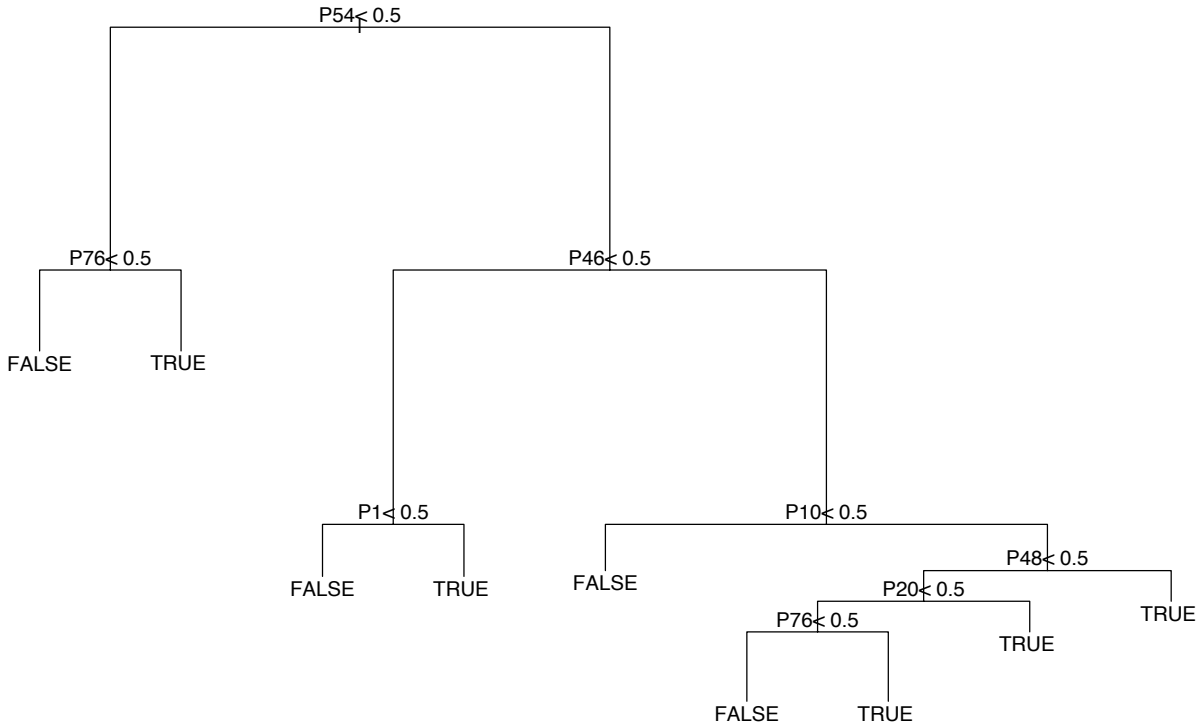
- (example continued)
  - This yields the following output:

```
n=976 (90 observations deleted due to missingness)
node), split, n, loss, yval, (yprob)
      * denotes terminal node
  1) root 976 399 FALSE (0.5911885 0.4088115)
    2) P54< 0.5 480 130 FALSE (0.7291667 0.2708333)
      4) P76< 0.5 466 116 FALSE (0.7510730 0.2489270) *
      5) P76>=0.5 14   0 TRUE (0.0000000 1.0000000) *
    3) P54>=0.5 496 227 TRUE (0.4576613 0.5423387)
      6) P46< 0.5 158  57 FALSE (0.6392405 0.3607595)
       12) P1< 0.5 115  31 FALSE (0.7304348 0.2695652) *
       13) P1>=0.5 43  17 TRUE (0.3953488 0.6046512) *
      7) P46>=0.5 338 126 TRUE (0.3727811 0.6272189)
       14) P10< 0.5 22   7 FALSE (0.6818182 0.3181818) *
       15) P10>=0.5 316 111 TRUE (0.3512658 0.6487342)
         30) P48< 0.5 278 106 TRUE (0.3812950 0.6187050)
           60) P20< 0.5 113  55 TRUE (0.4867257 0.5132743)
            120) P76< 0.5 92  40 FALSE (0.5652174 0.4347826) *
            121) P76>=0.5 21   3 TRUE (0.1428571 0.8571429) *
           61) P20>=0.5 165  51 TRUE (0.3090909 0.6909091) *
         31) P48>=0.5 38   5 TRUE (0.1315789 0.8684211) *
```

Splitting rules: Binary Trait

- (example continued)
  - We can also plot the results, as follows:

```
> plot(ClassTree)
> text(ClassTree)
```



Splitting rules: Binary Trait

- (example continued)
  - The most predictive variable of greater NFV sensitivity is `P54`.
  - Among viruses with a mutation at this site, indicated by `P54 > 0.5`, the estimated probability that `IDV.Fold` is greater than `NFV.Fold` is 0.54. On the other hand, this probability is 0.27 for viruses that are wildtype at this site.
  - The next split in the tree is described in rows `4)` and `5)` of the output. Here we see that among viruses that are wildtype at site `P54`, the most predictive variable is `P76`. In this case, the estimated probability of greater IDV than NFV fold resistance is 0.25 among viruses that are additionally wildtype at `P76`. On the other hand, this probability estimate is 1 for viruses that are mutant at `P76`.
  - The final nodes, also referred to as *terminal* nodes, are indicated with an asterisk in the output.
  - Warning: These are the most predictive variables, but we have not yet assessed whether they are statistically significant or simply chance findings.

Splitting rules: Binary Trait

- Several additional parameters can be specified within the `rpart()` function.
  - For example, we can specify the split criterion to be used. The default criterion is the gini index and the information criterion is given as an alternative option. To specify this, we use the following code:

```
> rpart(Trait~., method="class", parms=list(split='information'),
+ data=VircoGeno)
```

Splitting rules: Binary Trait

- (example continued)
  - n=976 (90 observations deleted due to missingness)
    node), split, n, loss, yval, (yprob)
    * denotes terminal node

```
 1) root 976 399 FALSE (0.5911885 0.4088115)
   2) P54< 0.5 480 130 FALSE (0.7291667 0.2708333)
     4) P76< 0.5 466 116 FALSE (0.7510730 0.2489270) *
     5) P76>=0.5 14   0 TRUE (0.0000000 1.0000000) *
   3) P54>=0.5 496 227 TRUE (0.4576613 0.5423387)
     6) P46< 0.5 158  57 FALSE (0.6392405 0.3607595)
      12) P1< 0.5 115  31 FALSE (0.7304348 0.2695652) *
      13) P1>=0.5 43  17 TRUE (0.3953488 0.6046512) *
     7) P46>=0.5 338 126 TRUE (0.3727811 0.6272189)
      14) P48< 0.5 299 120 TRUE (0.4013378 0.5986622)
        28) P20< 0.5 125  62 FALSE (0.5040000 0.4960000)
          56) P76< 0.5 104  44 FALSE (0.5769231 0.4230769) *
          57) P76>=0.5 21   3 TRUE (0.1428571 0.8571429) *
        29) P20>=0.5 174  57 TRUE (0.3275862 0.6724138) *
      15) P48>=0.5 39   6 TRUE (0.1538462 0.8461538) *
```

  - Notably, this results in a different tree than we saw above, though the first several splits are identical. Since splits that are lower down in the tree tend to be pruned back, these difference may not be relevant.

Splitting rules: Binary Trait

- (example continued)
  - We are also able to specify relevant criteria regarding the numbers of observations within nodes.
  - The control parameters `minsplit` and `minbucket` allows us to specify the number of observations that are required in a node in order to consider additional splits and the minimum number of observations required in a terminal node:

```
> rpart(Trait~., method="class", parms=list(split='gini'),
+   control=rpart.control(minsplit=150, minbucket=50),
+   data=VircoGeno)

n=976 (90 observations deleted due to missingness)
node), split, n, loss, yval, (yprob)
      * denotes terminal node
1) root 976 399 FALSE (0.5911885 0.4088115)
  2) P54< 0.5 480 130 FALSE (0.7291667 0.2708333) *
  3) P54>=0.5 496 227 TRUE (0.4576613 0.5423387)
    6) P46< 0.5 158  57 FALSE (0.6392405 0.3607595) *
    7) P46>=0.5 338 126 TRUE (0.3727811 0.6272189) *
```

  - The tree is smaller since splits at some nodes resulted in daughter nodes that contain less than the required number of observations, given by `minbucket=50`.

Splitting rules: Quantitative Trait

- The most commonly used measure of node impurity for a quantitative trait is the average sum of squared deviations from the mean, also referred to as the mean squared error (MSE), given by:

$$I(\Omega) = \frac{1}{n}\sum_i (Y_i - \bar{Y})^2$$

  Recall, this is the same as the least squares criterion used to find coefficient estimates in a linear regression

- **Example 2** (Generating a regression tree):
  - Suppose again that we are interested in identifying mutations in the protease region that are associated with a difference in IDV and NFV fold resistance based on the Virco data.

Splitting rules: Quantitative Trait

- (example continued)
  - In this case, we define a quantitative as the difference in these two values:

```
> Trait <- NFV.Fold - IDV.Fold
```

  - Using the `VircoGeno` matrix created in the Example above, we fit a regression tree as follows:

```
> RegTree <- rpart(Trait~., method="anova", data=VircoGeno)
> RegTree
```

  - Notably here we specify `method="anova"` although this is the default for a numeric trait.

Splitting rules: Quantitative Trait

- (example continued)
  - This gives the following output:

```
n=976 (90 observations deleted due to missingness)

node), split, n, deviance, yval
      * denotes terminal node

 1) root 976 6437933.00   4.288320
   2) P54>=0.5 496 1247111.00  -3.916935
     4) P46>=0.5 338  343395.20 -10.567160 *
     5) P46< 0.5 158  856789.90  10.309490
      10) P58< 0.5 144  110944.10   2.570139 *
      11) P58>=0.5 14  648503.60  89.914290 *
   3) P54< 0.5 480 5122921.00  12.767080
     6) P73< 0.5 422  145579.90   5.706635 *
     7) P73>=0.5 58 4803244.00  64.137930
      14) P35< 0.5 45   26136.17   8.171111 *
      15) P35>=0.5 13 4148242.00 257.869200 *
```

  - Based on this output we again see that n=976 observations contributed to this analysis.

Splitting rules: Quantitative Trait

- (example continued)
  - In this setting, `yval` corresponds to the mean trait or predicted value for observations within the corresponding node.
  - For example, at the `root` node, the mean difference in NFV and IDV fold resistance is 4.29. Among sequences that are mutant at `P54`, this mean difference is -3.92 while sequences that are wildtype at `P54` are predicted to have a difference of 12.77.
  - The `deviance` is defined as the sum of the squared differences between the observed trait and the predictive value over all observations within the corresponding node.
  - Dividing this quantity by the within node value of `n` yields $I(\Omega)$

Honest estimates

- Any tree we construct has an associated error that provides a measure of how well the tree predicts the observed data. Formally, the overall error associated with a tree $\mathcal{T}$, also referred to as *tree impurity*, is given by:

$$R(\mathcal{T}) = \sum_{\tau \in \widetilde{\mathcal{T}}} Pr(\tau) r(\tau)$$

  where $\widetilde{\mathcal{T}}$ is the set of terminal nodes in $\mathcal{T}$ and $r(\tau)$ is a measure of error for node τ.
- One estimate of this error is based on re-substituting the original data into the tree, called the *resubstitution estimate*. Inclusion of additional splits in a tree will always improve this resubstitution estimate of the error.
- An *honest estimate* of the error applies to any sample taken from the population of interest.

Honest estimates

- 10-fold Cross-Validation (CV) gives an honest estimate of the error:

$$R^{CV}(\mathcal{T}) = \frac{1}{10}\sum_i R^{ts}(\mathcal{T}_{-i})$$

  where $R^{ts}(\mathcal{T}_{-i})$ is the error running the *test sample* data, $\mathcal{L}_i$, through a tree generated using the *learning sample* $\mathcal{L}_{-i}$.
- Note that this is an estimate of the error associated with the tree $\mathcal{T}$ constructed based on the entire sample, $\mathcal{L}_1 \cup \mathcal{L}_2 \cup \ldots \cup \mathcal{L}_{10}$.

Cost-complexity pruning

Random Forests (RF)

1. Randomly sample with replacement n_1 (approximately equal to $2n/3$) individuals and call this the learning sample (LS). Let the remaining $n_2 = n - n_1$ individuals represent the out-of-bag (OOB) data.
2. Using the LS data only, generate an unpruned tree by randomly sampling a subset of the p predictors at each node to be considered as potential splitting variables.
3. Based on the OOB data only:
   - Record the overall tree impurity and let this be denoted π_b.
   - Permute $\mathbf{X}_j$ and record overall tree impurity using the permuted data, for each $j = 1, \ldots, p$. Denote this π_{bj} and define variable importance for the jth predictor as $\delta_{bj} = \pi_{bj} - \pi_b$.

Random Forests (RF)

4. Repeat steps (1) – (3) for $b = 2, \ldots, B$ to obtain $\delta_{1j}, \ldots, \delta_{Bj}$ for each j.
5. Record the overall variable importance score for $\mathbf{X}_1, \ldots, \mathbf{X}_p$, defined for the jth predictor as:

$$\widehat{\theta}_j = \frac{1}{B} \sum_{b=1}^{B} \delta_{bj}$$

Random Forests (RF)

- **Example 3** (An application of random forests):
  - We begin by uploading the randomForest package:

```
> install.packages("randomForest")
> library(randomForest)
```

  - We then define the trait and a matrix of potential predictor variables, given by `Trait` and `VircoGeno`, respectively, using the same code as in the Example above. The `randomForest()` function does not permit missing data in the response variable, and so we next subset our data to include only those individuals with complete information on the trait:

```
> attach(virco)
> Trait <- NFV.Fold - IDV.Fold
> VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")
> Trait.c <- Trait[!is.na(Trait)]
> VircoGeno.c <- VircoGeno[!is.na(Trait),]
```

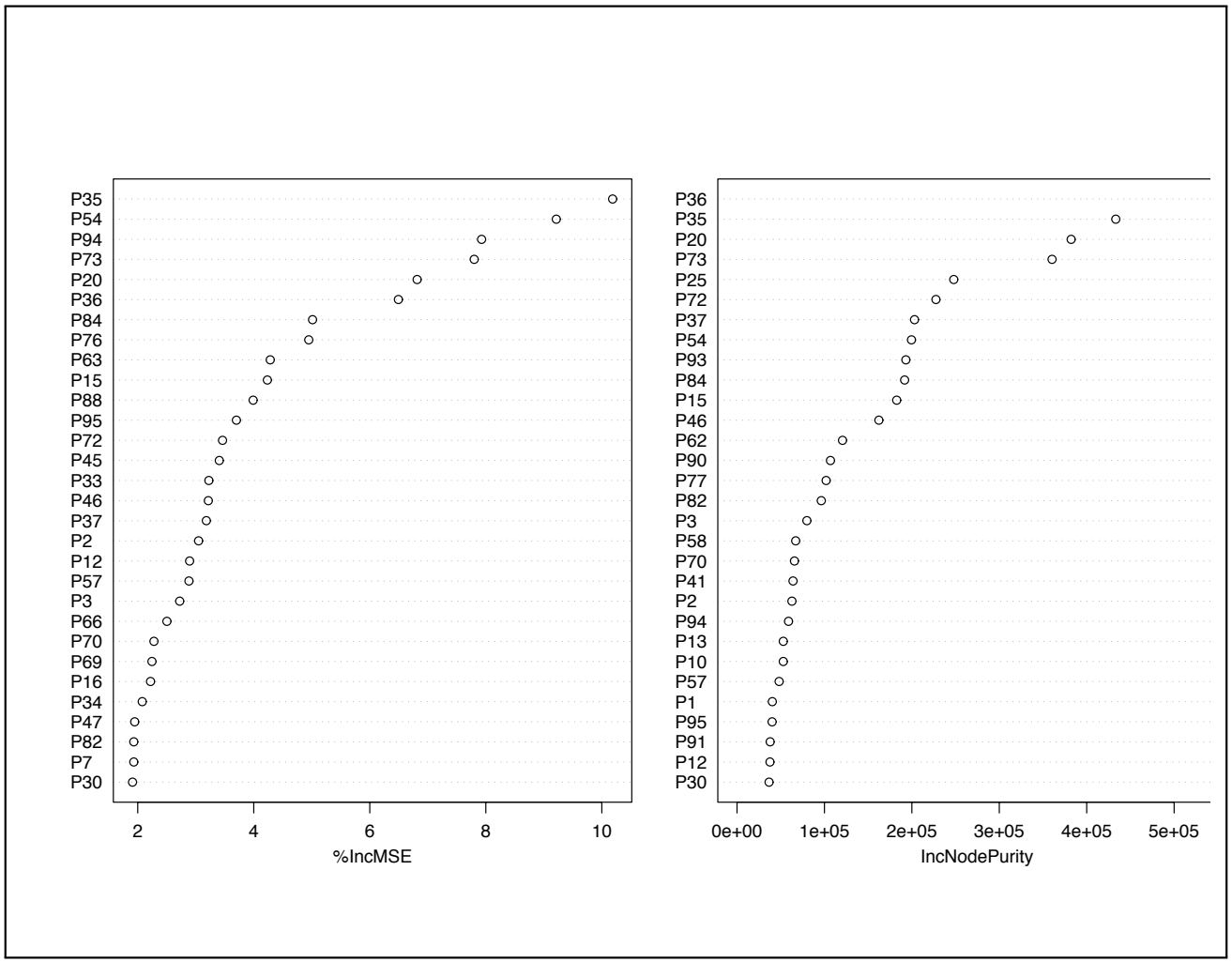
Random Forests (RF)

- (example continued)
  - We fit a random forest and plot the ordered variable importance measures, given by mean decrease in accuracy (%IncMSE) and mean decrease in node impurity (%IncMSE):

```
> RegRF <- randomForest(VircoGeno.c, Trait.c, importance=TRUE)
> RegRF

Call:
 randomForest(x = VircoGeno.c, y = Trait.c, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 33

          Mean of squared residuals: 5745.777
                    % Var explained: 12.89

> varImpPlot(RegRF)
```

Random Forests (RF)

Figure: Ordered variable importance scores from random forest



Additional Methods

- Multivariable Adaptive Regression Splines (MARS)
  - earth: earth()
- Logic Regression
  - LogicReg: logreg()

Summary

- Topics covered:
  - Classification trees – Bayes error and Gini index
  - Regression trees – least squares criterion
  - Honest estimates of error
  - Random forests – importance scores
- Useful R packages and functions:
  - rpart: rpart(); printcp(); plotcp(); prune()
  - tree: tree(), cv.tree(), prune.tree()
  - randomForest: randomForest()