Lecture 20: Decision Trees and Random Forests
STAT 632, Spring 2020

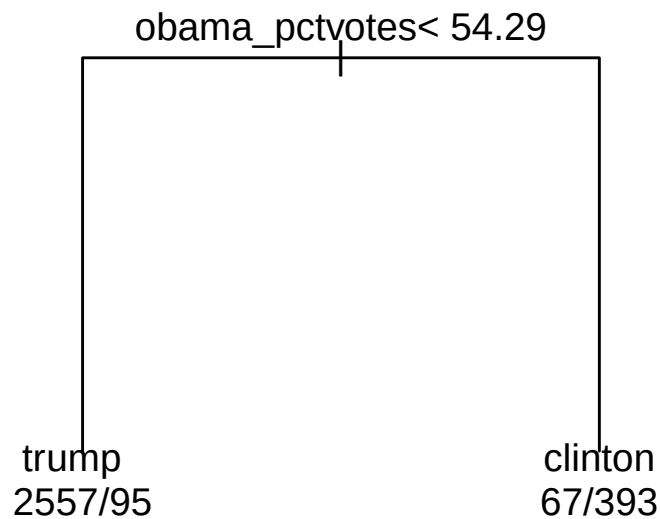
Decision Trees

- A **decision tree** is a flowchart-like structure that can be used for both regression and classification problems.
- Decision trees involve a finding a set of splitting rules for the predictor variables that stratify the data into smaller subsets. The set of splitting rules can be visualized as a tree.
- For classification tasks, predictions are made as the most commonly occurring class in each terminal node of the tree. Whereas, for regression tasks, predictions are made as the mean of the response data in each terminal node of the tree.
- We will focus on classification trees here. As an illustration, we will use the 2012/16 presidential election data sets for US counties from HW 7.

Example: Election Data

```
> library(rpart)

> county_votes16 <- readRDS(url("https://ericwfox.github.io/data/county_votes16.rds"))

# reformat response variable to a factor type
# with 1/0 coded as trump/clinton
> county_votes16$trump_win2 <- factor(county_votes16$trump_win,
    levels = c(1,0), labels=c("trump", "clinton"))

> table(county_votes16$trump_win2)
  trump clinton
   2624     488
```

```
# fit and plot tree model
> t1 <- rpart(trump_win2 ~ obama_pctvotes, data=county_votes16)
> par(cex=0.8, xpd=NA)
> plot(t1)
> text(t1, use.n=T)
```



- Here we fit a classification tree to predict whether Trump or Clinton wins a county using `obama_pctvotes`, the percent of votes cast for Obama in 2012, as a single predictor variable.

- The decision tree partitions the data into 2 sets. Counties with `obama_pctvotes` < 54.29 are assigned to the left-branch, which predicts that Trump wins. Otherwise, the tree predicts Clinton wins.

- This kind of tree-model is called a **stump** since there is just one splitting rule.

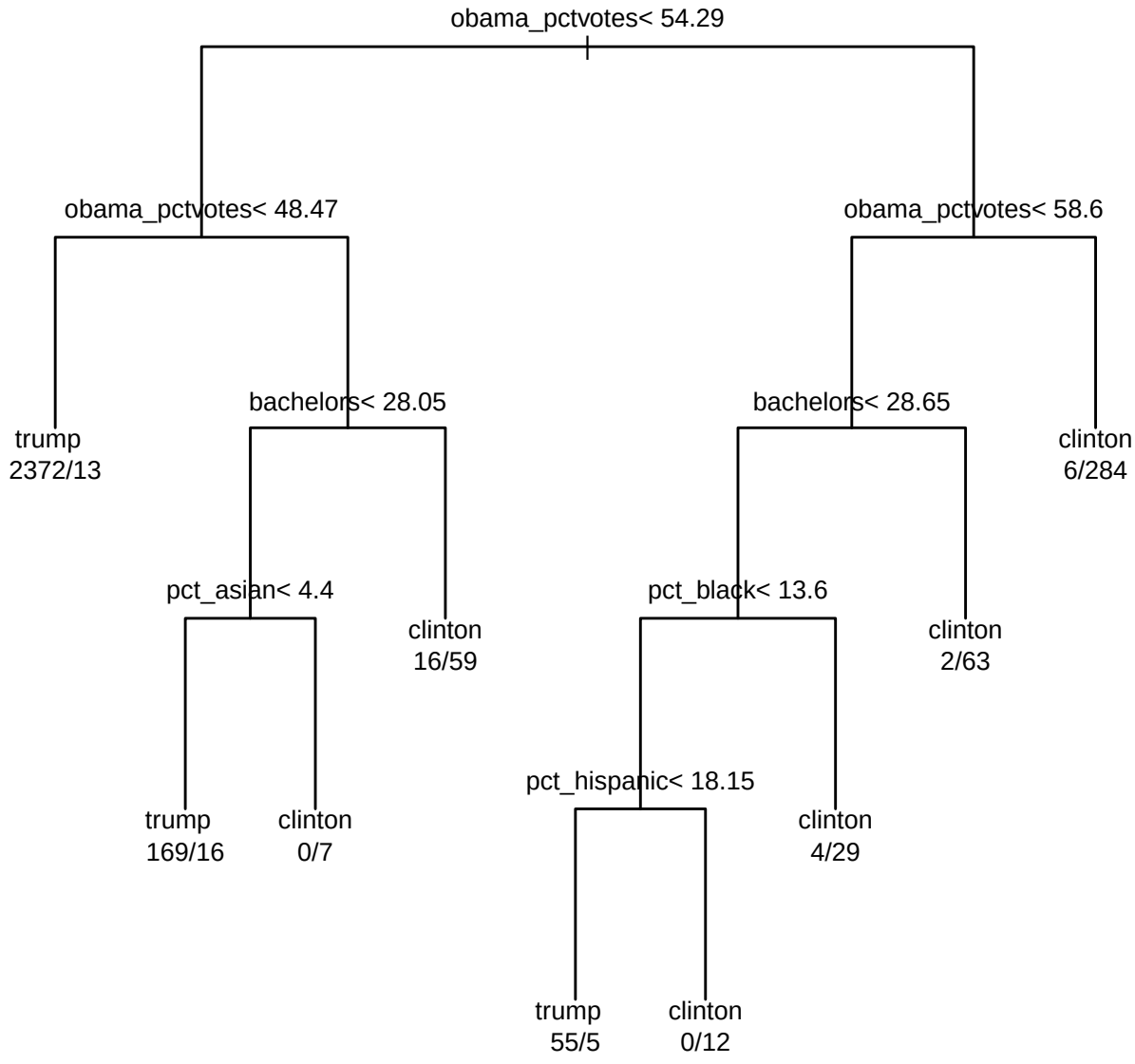
How did the algorithm select `obama_pctvotes` < 54.29 as the splitting rule?

- One approach is to loop through all sensible splits of `obama_pctvotes`, and choose the one that minimizes the **misclassification error rate**.
- Another common approach is to minimize the **Gini index**, $G_i = \sum_{k=1}^{K} \hat{p}_{ik}(1 - \hat{p}_{ik})$, where $\hat{p}_{ik}$ represents the proportion of observations of class k that are in the i^{th} terminal node. It is minimized when all the $\hat{p}_{ik}$'s are close to 0 and 1, and so the data in a terminal node are predominantly of the same class. By default `rpart()` uses the Gini index.[1]

[1] For more details see pp. 311–313 from *An Introduction to Statistical Learning*

We can also use `rpart()` fit a decision tree using more than one variable.

```
# fit tree using all 9 predictors
> t2 <- rpart(trump_win2 ~ obama_pctvotes + pct_pop65 +
                pct_black + pct_white + pct_hispanic + pct_asian +
                highschool + bachelors + income, data=county_votes16)

# plot tree
> par(cex=0.6, xpd=NA)
> plot(t2, uniform=T)
> text(t2, use.n=T)
```


obama_pctvotes< 54.29
obama_pctvotes< 48.47
obama_pctvotes< 58.6
trump
2372/13
bachelors< 28.05
bachelors< 28.65
clinton
6/284
pct_asian< 4.4
clinton
16/59
pct_black< 13.6
clinton
2/63
trump
169/16
clinton
0/7
pct_hispanic< 18.15
clinton
4/29
trump
55/5
clinton
0/12

- The `rpart()` function uses a method called **recursive binary splitting** to fit the decision tree.

- Trees that have too many nodes can overfit the data. One strategy is to grow a very large tree, and stop splitting when each terminal node contains no more than 5 observations. Then a **pruning** algorithm is used to find a subtree that is less complex.

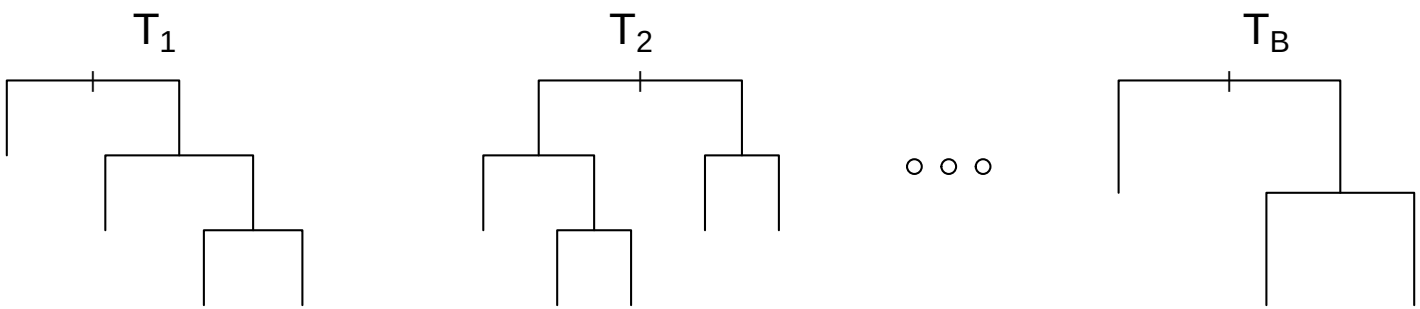
- The details of this algorithmic approach to tree fitting are beyond the scope of what we can cover today. Chapter 8 of *An Introduction to Statistical Learning* provides an excellent explanation.

Random Forests

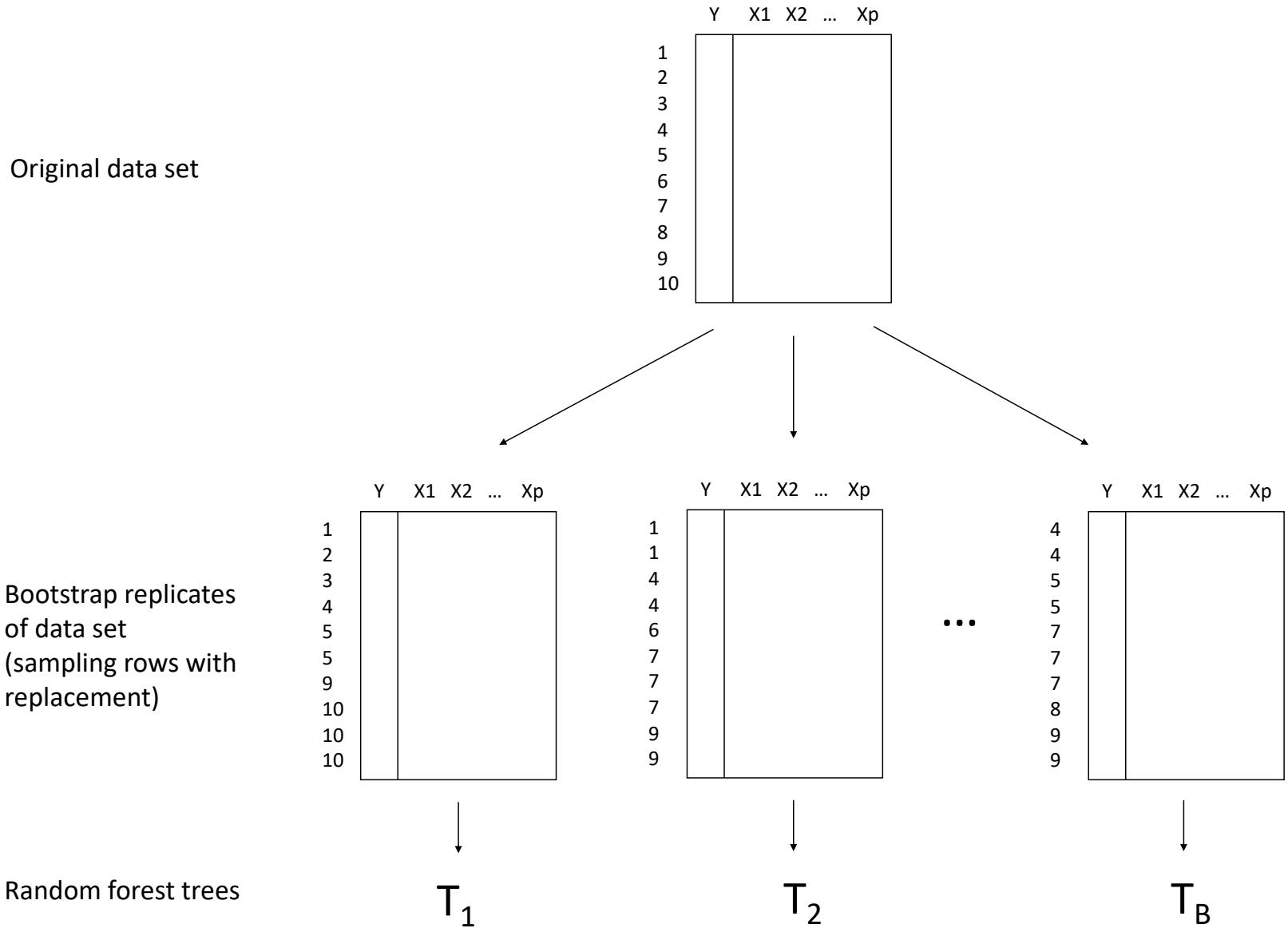
- Decision trees are simple and useful for interpretation.
- However they typically are not competitive with the best statistical learning methods in terms of prediction accuracy.
- Random forests is a popular method that involves fitting a large number of decision trees which are then combined to yield a single consensus prediction.
- The random forests approach to combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation.

Random Forests

A random forest (RF) model is a collection of decision trees $\{T_b : b = 1, \cdots, B\}$ built from bootstrap samples of the data set.



Additionally, at each internal node $m \leq p$ predictors are randomly sampled as candidates for splitting.


Original data set
Y X1 X2 ... Xp
1
2
3
4
5
6
7
8
9
10
Bootstrap replicates of data set (sampling rows with replacement)
Y X1 X2 ... Xp
1
2
3
4
5
5
9
10
10
10
Y X1 X2 ... Xp
1
1
4
4
6
7
7
7
9
9
...
Y X1 X2 ... Xp
4
4
5
5
7
7
7
8
9
9
Random forest trees
T_1
T_2
T_B

Random Forests: Prediction

Let $\boldsymbol{x} = \begin{pmatrix} x_1 & x_2 \cdots x_p \end{pmatrix}$ be a vector of new values for the predictors, and $T_b(\boldsymbol{x})$ the response prediction of the b^{th} tree.

The random forest prediction for a quantitative response is found by averaging over the predictions made by each tree in the ensemble:

$$\hat{f}_{RF}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} T_b(\boldsymbol{x})$$

Random Forests: Prediction

How can random forests be used for binary classification?

Let $T_b(\boldsymbol{x}) \in \{0, 1\}$ be the class prediction of the b^{th} tree.

Then the random forest prediction is given by the *majority vote* of $\{T_b(\boldsymbol{x}) : b = 1, \cdots, B\}$. That is, the most commonly occurring class among the B predictions.

This concept also generalizes to problems with more than 2 response classes.

Random Forests: Tuning

RF has two main tuning parameters:

`mtry`: Number of predictors randomly sampled as candidates at each split.

- Has the effect of decorrelating, or diverisfying, the trees in the ensemble.
- Some special cases:
  - `mtry` $= p$ is called *bagging* (all the predictors are considered at each split)
  - `mtry` $= p/3$ is the default for regression
  - `mtry` $= \sqrt{p}$ is the default for classification

`ntree`: Number of trees; the default is 500.

RF is generally insensitive to the choice of these tuning parameters. The defaults work adequately well for most data sets.

Random Forests: Out of Bag Data

- The trees in a random forest model are fit to bootstrap samples of the data set.
- One can show that each tree in a random forest model makes use of about two-thirds of the observations.
- The remaining one-third of the observations not used to fit a random forest tree are referred to as the **out-of-bag (OOB)** data.
- Since the OOB data was not used to fit the model it can be used to compute essentially cross-validated performance measures (e.g., classification accuracy).

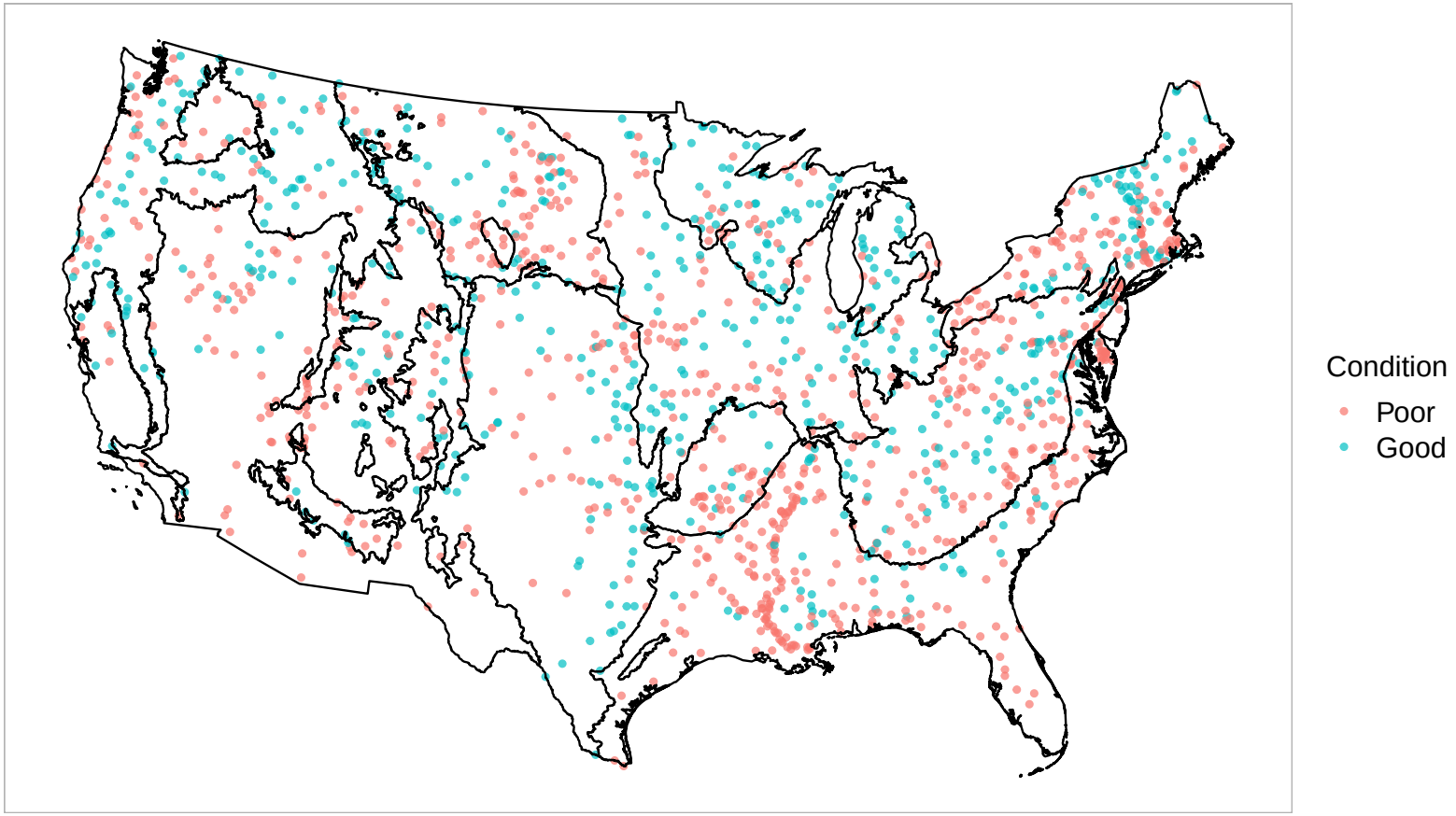
Example: EPA Stream Condition Data

- The Environmental Protection Agency (EPA) randomly sampled stream sites across the conterminous US during the summer months of 2008/09. The effort was part of a larger environmental monitoring program called the National Rivers and Stream Assessment (NRSA).[2]

- Streams were classified as being in `Good` or `Poor` condition according to an aquatic health index. Sampled macroinvertebrates were used as the primary indicator of aquatic health.

[2] `https://www.epa.gov/national-aquatic-resource-surveys/nrsa`

Map of sampled stream sites from 2008/09 National Rivers and Stream Assessment.



Random Forest Model

Response: Binary `Good`/`Poor` condition of $n = 1433$ sampled stream sites.

Predictors: $p = 210$ landscape features for each stream's catchment (i.e., local drainage area around each stream segment). The predictors are from the StreamCat data set.[3]

- Examples: % urbanization, % agriculture, road densities, dams, temperature, precipitation, forest change, etc.

[3] `https://www.epa.gov/national-aquatic-resource-surveys/streamcat`

```
# load EPA stream data set
> streams <- readRDS(url("https://ericwfox.github.io/data/streams.rds"))

> dim(streams)
[1] 1433  211

> table(streams$Condition)
Poor Good
 862  571

# fit random forest model using all predictors
> library(randomForest)
> set.seed(999) # set seed for reproducibility
> rf1 <- randomForest(Condition ~ ., data=streams)

# make predictions for stream condition (Good/Poor)
# predictions are made using the OOB data
rf_preds <- predict(rf1, type = "response")
```

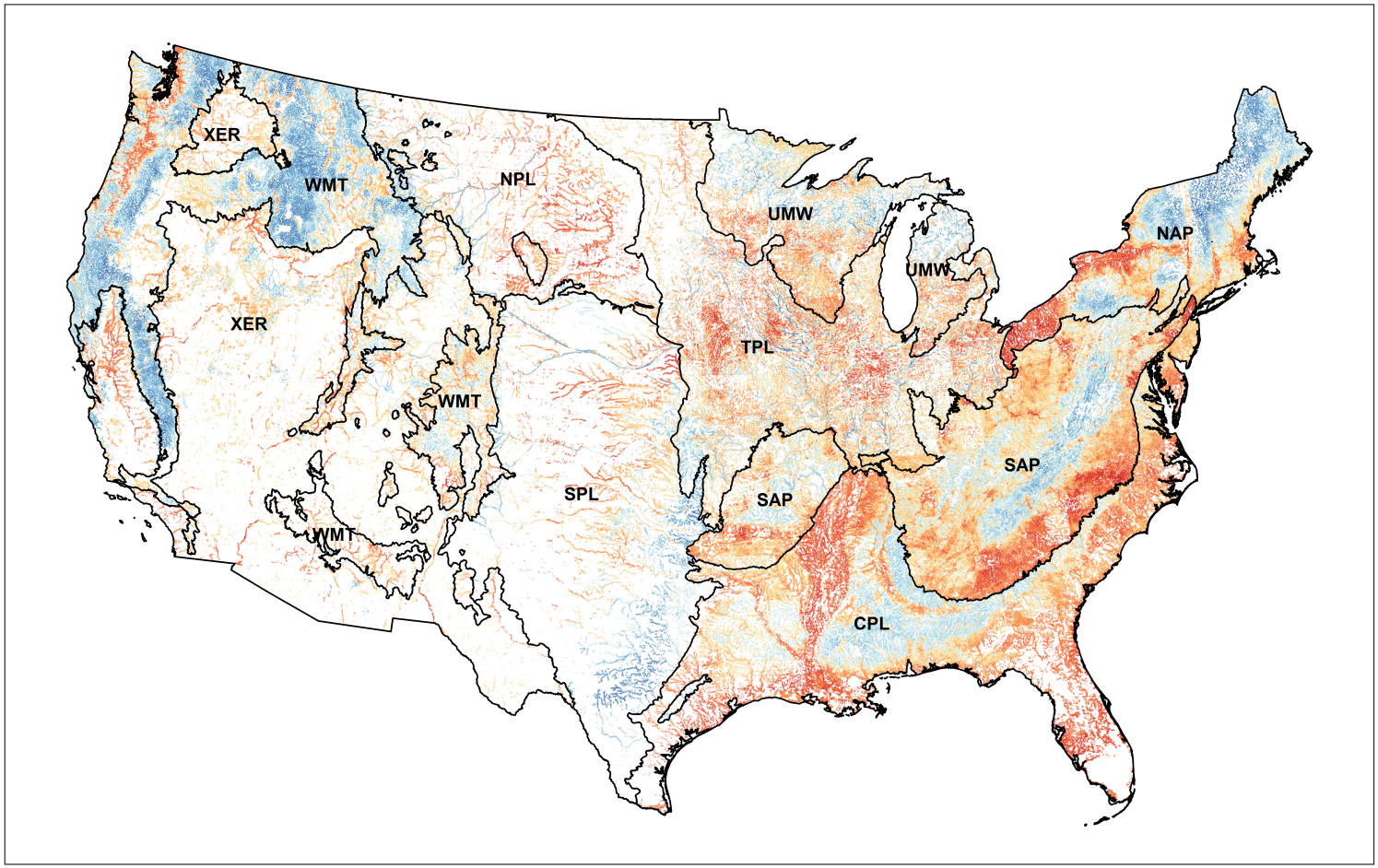
```
# make confusion matrix
> tb <- table(actual = streams$Condition, predicted = rf_preds)
> addmargins(tb)
      predicted
actual Poor Good  Sum
  Poor  752  110  862
  Good  211  360  571
  Sum   963  470 1433

# Accuracy (percent correctly classified)
> (752 + 360) / 1433
[1] 0.7759944

# Sensitivity (percent of good streams correctly classified)
> 360 / 571
[1] 0.6304729

# Specificity (percent of poor streams correctly classified)
> 752 / 862
[1] 0.8723898
```

Mapping Predictions

- One application of the random forest model is to map the predicted probability of good stream condition for all 1.1 million perennial stream reaches in the US.
- The $p = 210$ predictors (i.e., landscape features) from the StreamCat data set are available for all 1.1 million perennial stream reaches across the US, and thus make mapping of the predictions possible.
- Let $\boldsymbol{x}$ be the predictor values at a new, unsampled location and $T_b(\boldsymbol{x}) \in \{0, 1\}$ the predicted `Poor`/`Good` condition of the b^{th} tree in the random forest model. Then we define the predicted probability of good stream condition as

$$\hat{p}(\boldsymbol{x}) = \frac{1}{B} \sum_{i=1}^{B} T_b(\boldsymbol{x})$$

That is, the predicted probability is the proportion of the random forest trees that voted `Good`.

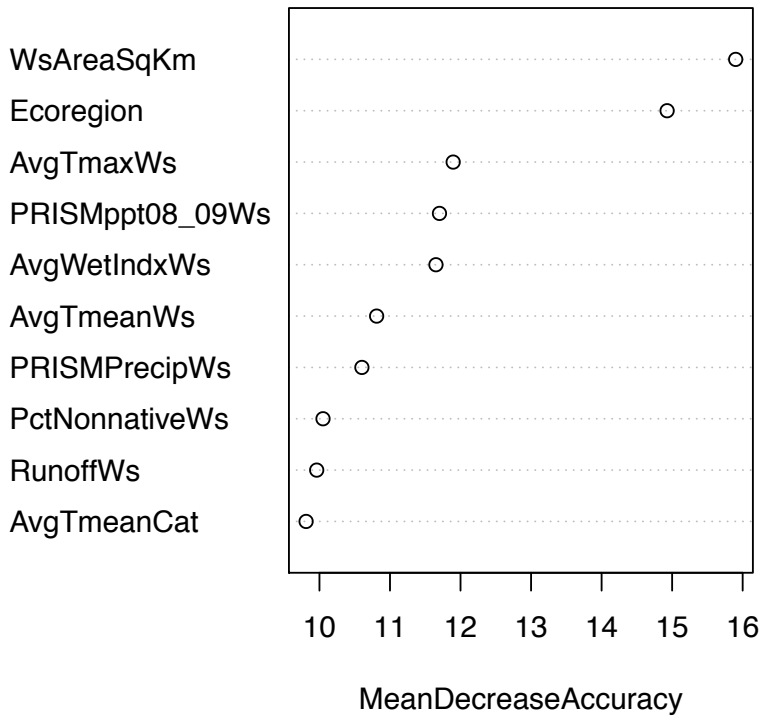


Ecoregions: Coastal Plains (CPL), Northern Appalachians (NAP), Northern Plains (NPL), Southern Appalachians (SAP), Southern Plains (SPL), Temperate Plains (TPL), Upper Midwest (UMW), Western Mountains (WMT), and Xeric (XER)

Variable Importance

- Random forests also provides measures of variable importance. Here we use a measure based on the decrease in prediction accuracy when permuting each predictor in the OOB data.
- For modeling stream condition, the two most important predictors are the watershed area and ecoregion. Essentially, this means that the size and location of the stream are important.

```
> set.seed(999) # set seed for reproducibility
> rf1 <- randomForest(Condition ~ ., data=streams, importance = TRUE)
> varImpPlot(rf1, type=1, n.var=10, main="")
```



Concluding Remarks

- Random forest models often performs well since they can handle nonlinearities and high-order interactions between the predictors and response variable.
- Random forest models are also robust to the inclusion of a large number of predictor variables. For instance, the data set used for modeling stream condition contained $p = 210$ predictors.
- One shortcoming of random forests is that it is a so-called "black-box" method. In contrast to linear regression, or individual decision trees, we do not know what the relationships are between the response and predictor variables. However, variable importance plots allow for some interpretation with random forests.

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. *An Introduction to Statistical Learning with Applications in R*. Springer, 2013.

Hill, R.A., Fox, E.W., Leibowitz, S.G., Olsen, A.R., Thornbrugh, D.J., and Weber, M.H. (2017). Predictive mapping of the biotic condition of conterminous-USA rivers and streams. *Ecological Applications*, 27(8), 2397-2415.

Hill, R.A., Weber, M.H., Leibowitz, S.G., Olsen, A.R., and Thornbrugh, D.J. (2016). The Stream-Catchment (StreamCat) Dataset: A database of watershed metrics for the conterminous Unites States. *Journal of the American Water Resources Association*, 52(1):120–218.