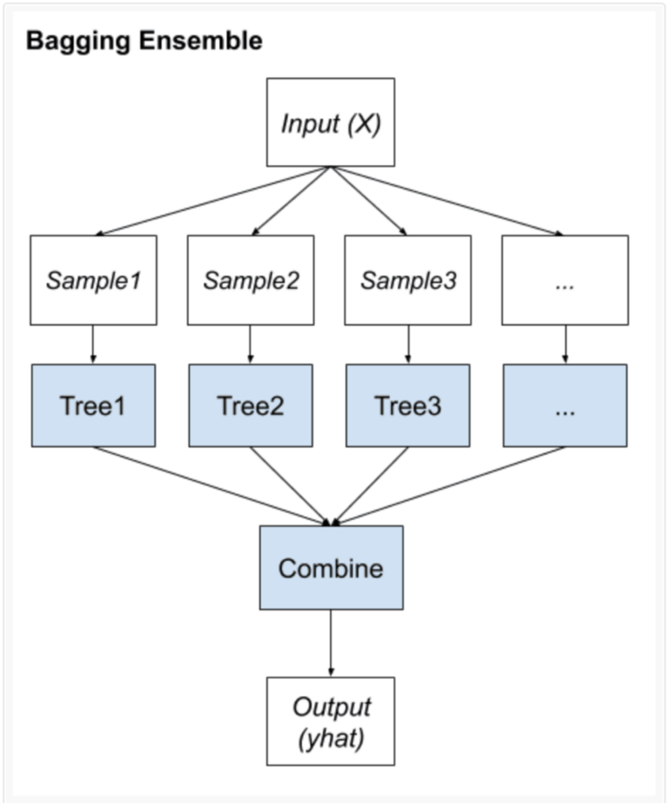
STAT 380 – Random Forests (Lecture 17)

Definition: Ensemble Learning is an approach to machine learning that involves combining predictions from multiple models, often called "weak learners", in order to achieve more accurate predictions. Three classes of ensemble learning are bagging, stacking, and boosting.

IDEA: In this lecture, we will explore the concept of a random forest, which is a variation of bagging decision trees.

Definition: Bagging is short for Bootstrap Aggregating. Bagging decision trees involves fitting many decision trees, each on a bootstrap sample of the dataset. The predictions from each tree are then combined by averaging (or majority voting). The following image, taken from https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/, depicts the idea of bagging.



Bagging Ensemble

RECALL: A bootstrap sample is a sample of size n, selected with replacement, from a dataset consisting of n observations. Since the bootstrap sample is selecting based on sampling with replacement, some observations (rows) are selected multiple times while other rows are not selected at all.

Definition: (.632 Rule). The average number of unique observations in each bootstrap sample is about $0.632n$ where n is the number of observations in the dataset. The value comes from considering the chances of NOT being selected as any of the n draws with replacement from the n rows of the sample. As the sample size, n, grows without bound, the probability of not being selected is given by:

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n$$

It can be shown that this limit is equal to $e^{-1} \approx 0.368$. Thus, the probability of being selected is approximately 1 – 0.368 = 0.632.

NOTE: There is a very straightforward way to estimate the test error of a random forest, or more generally a bagged, model, WITHOUT the need to perform cross-validation or the validation set (training/testing split) approach.

IDEA: The key to bagging (and random forest) is that trees are repeatedly fit to bootstrapped subsets of the observations. Using the .632 Rule, approximately 0.368 of the observations are held out when constructing each tree. We refer to those observations that are not used in the construction of a given tree as the out-of-bag (OOB) observations.

NOTE: We can predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This will yield approximately number_of_trees/3 predictions for the i^{th} observation. To obtain a single prediction for the i^{th} observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the i^{th} observation.

NOTE: An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.

CAUTION: Even though random forests include a built-in method for cross validation, we must still perform a training/testing split (or k-fold CV) if we want to compare the random forest performance to other methods (such as multiple linear regression, kNN, logistic regression, etc.) using a consistent test set.

Random Forest

NOTE: Random forest is an example of ensemble learning. In ensemble learning, the idea is to build a large number of weak learners and combine their predictions. Random forest achieves this through a variation of bootstrap aggregating (bagging). The random forest algorithm is as follows:

```
for (i in 1:number_of_trees){

        1. Generate a bootstrap sample of the data
        2. Grow a regression/classification tree using the bootstrap sample
             • Each time we consider making the next split in the tree, we only consider a random
               subset of the input variables – done to decorrelate the trees
        3. Record the predictions for the OOB data
}

Combine the predictions from the trees
```

EXAMPLE 1: Our goal is to use random forest to predict whether a person will survive based on the other variables in the dataset. The data are found in L12_titanic3.csv. The dataset contains:

Variable Name	Variable Meaning	Notes
Survived	Whether the passenger survived	Values include 'Yes' and 'No'
PClass	Purchased ticket class	1 = First class 2 = Second class 3 = Third class
Sex	Passenger's sex	Values include 'male' and 'female'
Age	Passenger's age	
Siblings	Number of siblings or spouses aboard	
Parch	Number of parents or children aboard	
Fare	Passenger's fare	

a. Perform an 80/20 training/testing split based on a seed of 123. NOTE: Although random forest has a built-in method for assessing performance on heldout data, we still must do a split if we want to compare the performance to other methods (such as knn, multiple linear regression, etc.)

```
set.seed(123)
train_ind <- sample(1:nrow(Titanic), floor(0.8 * nrow(Titanic)))
set.seed(NULL)

Train <- Titanic[train_ind, ]
Test <- Titanic[-train_ind, ]
```

b. Using the randomForest function from the randomForest library, build a random forest for predicting whether a person survives based on the other variables in the dataset. The forest should consist of 500 trees. Each time a split is considered, use 3 variables. Explain the error message:

```
rf <- randomForest(Survived ~ ., data = Train, ntree = 500, mtry = 3)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in mean.default(y): argument is not numeric or logical: returning NA
```

```
## Error in y - ymean: non-numeric argument to binary operator
```

b. (More Space)

c. Fix the code to eliminate the error message and build the model.

d. What are the default options for ntree and mtry? You will find the answers in the documentation.

e. Find the probabilities of survival for the first 6 passengers in the testing set. Are you getting the same values?

```
pred_prob <- predict(rf, newdata = Test, type = "prob")
head(pred_prob)
```

```
##        No   Yes
## 1  1.000 0.000
## 3  0.598 0.402
## 7  0.732 0.268
## 12 0.302 0.698
## 15 0.222 0.778
## 18 1.000 0.000
```

f. Using a seed of 123 for constructing the forest, report the confusion matrix for the testing data and the overall accuracy of the method.

```r
#Build model using a seed
set.seed(123)
rf <- randomForest(as.factor(Survived) ~ ., data = Train,
                    ntree = 500, mtry = 3)
set.seed(NULL)

#Obtain predicted probabilities
pred_prob <- predict(rf, newdata = Test, type = "prob")
head(pred_prob)
```

```
##       No   Yes
## 1  0.996 0.004
## 3  0.612 0.388
## 7  0.770 0.230
## 12 0.314 0.686
## 15 0.222 0.778
## 18 0.998 0.002
```

```r
#To create the confusion matrix, we need to determine which class (No o
r Yes) is more likely. Although we could define a threshold and compare
 probabilities as we have done in the past, things are more challenging
 if y has more than 2 classes. So, let R pick
pred_surv <- predict(rf, newdata = Test, type = "response")

#Create confusion matrix
table(pred_surv, Test$Survived)
```

```
## 
## pred_surv No Yes
##       No  92  23
##       Yes 14  49
```

```r
#Calculate accuracy
mean(pred_surv == Test$Survived)
```

```
## [1] 0.7921348
```

NOTE: Do ***not*** use rf$confusion. This confusion matrix is based on OOB data, not Test.

Variable Importance

RECALL: One of the advantages of decision trees is the attractive and easily interpreted diagram that results.

ISSUE: When we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure. Thus, bagging improves prediction accuracy at the expense of interpretability. (Random forest is a special case of bagging in which we also add random selection of inputs for considering tree splits.)

IDEA: We can obtain an overall summary of the importance of each predictor using:

- Residual Sum of Squares, RSS, for bagging regression trees
- the Gini index for bagging classification trees

RECALL: Definition: Let $\hat{p}_{mk}$ represent the proportion of training observations in the mth region/node that are from the kth class. The Gini Index is defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}\,(1 - \hat{p}_{mk})$$

which is a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $\hat{p}_{mk}'s$ are close to zero or one. For this reason, the Gini index is referred to as a measure of node impurity—a small value indicates that a node contains predominantly observations from a single class.

EXAMPLE 2: Titanic data continued.

a. Build a classification random forest as in Example 1f. Generate the variable importance values by including the importance = TRUE option in the randomForest function.

```
set.seed(123)
rf <- randomForest(as.factor(Survived) ~ ., data = Train,
                   ntree = 500, mtry = 3, importance = TRUE)
set.seed(NULL)
```

b. There are several ways to access the variable importance values.

```r
rf$importance
```

```
##                   No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## Pclass   0.036931920 0.12814243          0.071609042         33.57867
## Sex      0.105872033 0.20031397          0.141680920         83.24320
## Age      0.044598457 0.04764105          0.045674947         69.11592
## Siblings 0.020312316 0.00994687          0.016384008         16.63690
## Parch    0.007242857 0.01149273          0.008886679         10.20615
## Fare     0.043739018 0.10153287          0.065615610         80.53885
```

```r
importance(rf)
```

```
##                 No       Yes MeanDecreaseAccuracy MeanDecreaseGini
## Pclass   18.655060 34.928829             41.30628         33.57867
## Sex      67.497153 87.624109             98.40891         83.24320
## Age      24.157107 23.343567             36.43599         69.11592
## Siblings 18.166660  8.214273             20.93938         16.63690
## Parch     7.184154  9.819590             11.69148         10.20615
## Fare     17.888774 29.376226             37.10056         80.53885
```

```r
importance(rf, scale = FALSE)
```

```
##                   No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## Pclass   0.036931920 0.12814243          0.071609042         33.57867
## Sex      0.105872033 0.20031397          0.141680920         83.24320
## Age      0.044598457 0.04764105          0.045674947         69.11592
## Siblings 0.020312316 0.00994687          0.016384008         16.63690
## Parch    0.007242857 0.01149273          0.008886679         10.20615
## Fare     0.043739018 0.10153287          0.065615610         80.53885
```

NOTE: Regarding the variable importance metrics, larger values suggest greater importance.

From: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

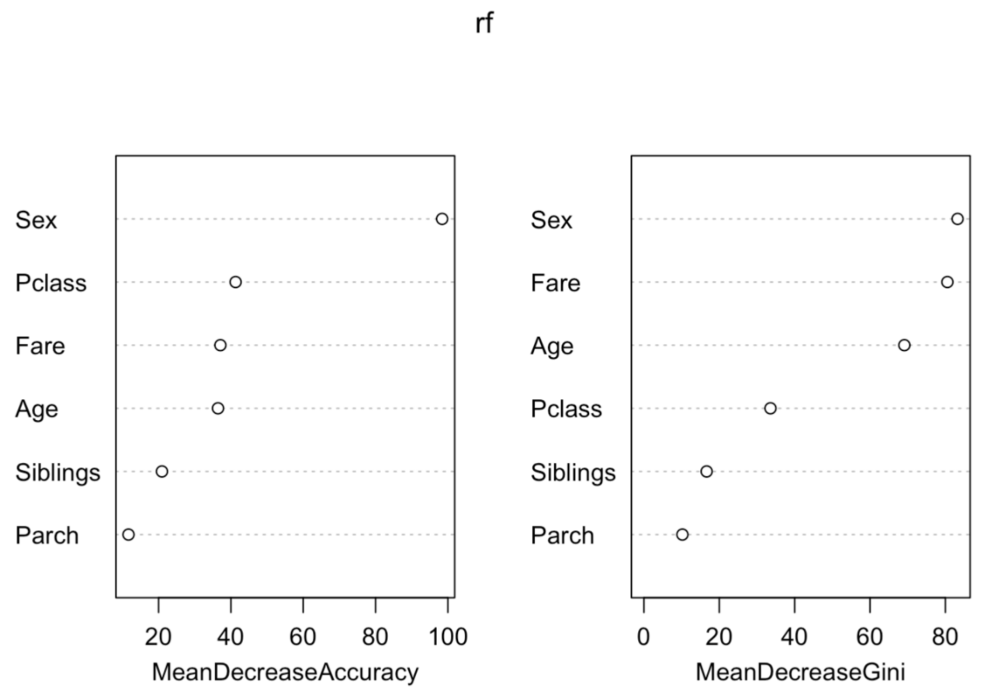
Details

Here are the definitions of the variable importance measures. The first measure is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences. If the standard deviation of the differences is equal to 0 for a variable, the division is not done (but the average is almost always equal to 0 in that case).

The second measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares.

c. Plot the variable importance values using the varImpPlot function from the randomForest package.

```
varImpPlot(rf, n.var = 6)
```



d. Which 3 variables are the most important?

NOTE: When running random forest, you would have to choose the values for ntree and mtry. This is often done by performing a grid search across a number of combinations. The optimal combination is chosen based on a specific metric (such as smallest RMSE/highest accuracy on test set (or OOB) data).