Lecture 7: Classifiers I

Intro to Data Science for Public Policy, Spring 2016

by Jeff Chen & Dan Hammer, Georgetown University McCourt School of Public Policy

Contents

Introduction

According to the American Community Survey (ACS), an annual survey of approximately 3.5% of the US population as conducted by the US Census Bureau, over 16.5% of residents of the U.S. state of Georgia were without healthcare coverage in 2015. To some degree, this is [un]surprising. While the statistic is informative, it is not actionable as more specifics are required. How can a policy or program close the gap?

In order to close the gap, the data needs to enable the prediction and classification of a population into two classes: covered and not covered This binary problem or membership problem is known as a classification problem. By correctly classifying a population as covered and not covered, decision makers and outreach staff can mobilize targeted outreach. From a data science perspective, the real objective is to be able to identify and replicate re-occurring patterns in the training data, then generalize the insights onto a sample or population that is not contained in the sample. In other words, the production of actionable data insights means that the results of data analysis and modeling is worth more than a sound bite, but can be applied and deployed directly to solve a problem.

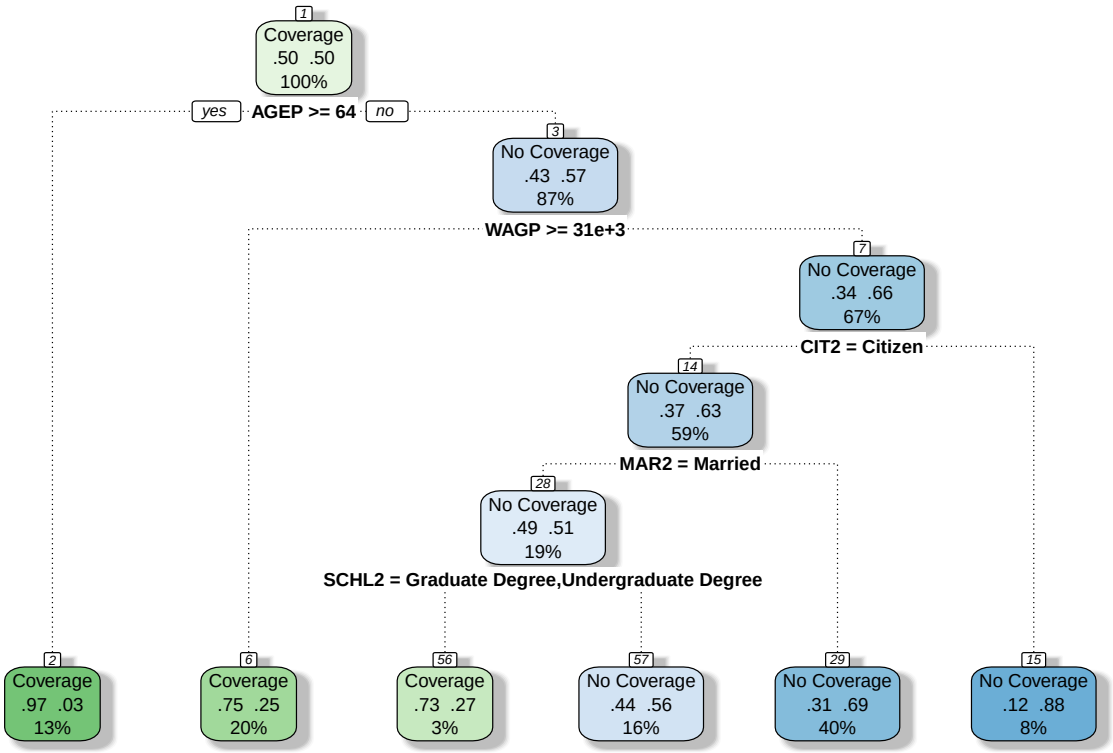
In most policy environments, a data analyst will typically manually select population characteristics to use in cross tabulations to find statistical patterns; however, this tradtional approach can suffer from human bias that may yield misleading results. Some features may be more important than others, and humans usually do not systematically check all features. For example, the table below compares healthcare coverage and citizenship. Each of the cells are quite interpretable: 49.4% of non-citizens are without coverage, but that sub-population is only 3.5% of the population.

	Coverage	Without coverage	% Without coverage
Citizen	6,387,690	1,051,815	14.1%
Non-citizen	276,192	282,769	49.4%
All	6,663,882	1,334,584	16.6%
% Non-citizen	4.1%	21.1%	3.5%

A cross-tabulation does not, however, provide sufficient predictive power. Expanding the table to include more features such as age, gender, wages, etc. may not improve inference either as this will lead to "analysis paralysis".

Supervised learning offers a number of options to more efficiently identify patterns surface important variables, and predict membership. Given the label $Y(Coverage)$, we will use supervised learning techniques to determine how to split the survey sample into smaller cells using the following features: Sex, Age, Education, Marital Status, Race, Citizenship).

Below is a visualization of the results, which are interpretable as well as defined by discrete cells of Georgians who have and do not have healthcare coverage. For example, non-citizens under the age of 64 without a college education are 90% likely to not have coverage, and citizens between 16 and 64 who are not married have a 71% chance of not having health coverage.



The above diagram was produced using a decision tree algorithm, which is one of many forms of supervised learning known as *classifiers* or *classification algorithms*. Classifiers take on many forms. Some use recursive partitioning to break a population into many, more homogeneous subpopulations. Others estimate a series of equations to fit a line or plane between two or more classes. Others will average the results of an ensemble of models to predict membership. Each class of model is defined with mathematical scenarios in mind. This chapter is organized into three sections. Section 1 describes common considerations in classification models. Section 2 provides an overview of a number of classifiers, including kNN, logistic regression, decision trees, as well as random forests, support vector machines, and ensemble methods. Section 3 describes the types of applications of these methods.

Section 1 - An Overview

What goes into a classifier?

Classifiers are a type of supervised learning problem that handles target features that contain discrete labels, otherwise known as classes. Using the example above, having and not having health insurance would be two classes. Being part of Generations X, Y, and Z would be three classes. Classification algorithms accept a discrete target feature and input features to produce a probability that a given record belongs to target class given the input features.

The output probability is the key to evaluating the accuracy of a model. Unlike regression, classifiers rely on entirely different measures of accuracy given the nature of the labeled data. All measures, however, rely on metrics that can be derived from a confusion matrix, or a 2×2 table where the rows typically represent actual classes and columns represent predicted classes.

	Predicted (+)	Predicted (-)
Actual (+)	True Positive (TP)	False Negative (FN)
Actual (-)	False Positive (FP)	True Negative (TN)

Each of the cells contains the building blocks of accuracy measures:

- The True Positive (TP) is the count of all cases where the actual positive ($Y = 1$) case is accurately predicted.
- The True Negative (TN) is the count of all cases where the actual positive ($Y = 0$) case is accurately predicted.
- The False Positive (FP) is count of all cases where the actual label was $Y = 0$, but the model classified a record as $\hat{Y} = 1$. This is also known as Type I error.
- The False Negative (FN) is count of all cases where the actual label was $Y = 1$, but the model classified a record as $\hat{Y} = 0$. This is also known as Type II error.

Accuracy. Overall accuracy is measured as the sum of the main diagonal divided by the population (below).

$$TPR = \frac{TP + TN}{TP + FN + FP + TN}$$

True Negative Rate. By combining TN and FP, we can calculate the True Negative Rate (TPR), which is proportion of $Y = 0$ cases that are accurately predicted. TNR is also referred to as the "specificity".

$$TNR = \frac{TN}{TN + FP} = \frac{TN}{Actual(-)}$$

True Positive Rate. By combining TP and FN, we can calculate the True Positive Rate (TPR), which is proportion of $Y = 1$ cases that are accurately predicted. TPR is also referred to as the "sensitivity" or "recall".

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{Actual(+)}$$

Positive Predicted Value. By combining TP and FP, we can calculate the Positive Predicted Value (PPV), which is proportion of predicted $Y = 1$ cases that actually are of tht class. PPV is also referred to as "precision".

$$PPV = \frac{TP}{TP + FP}$$

What does accuracy look like? To illustrate this, the next series of tables provides simulated results of a classifer. Let's assume that a health insurance classifier was tested on a sample of $n = 100$ with actual labels perfectly split between $Y = 1$ and $Y = 0$. A perfect performing model would resemble the following table, where TP = 50 and FP = 50. With perfect predictions with $Accuracy = \frac{50+50}{100} = 100$, the $TPR = \frac{50}{50+0} = 100$ and $PPV = \frac{50}{50+0} = 100$ indicate that model is perfectly balanced and precise.

	Predicted (+)	Predicted (-)
Actual (+)	50	0
Actual (-)	0	50

A model with little discriminant power or ability to distinguish between classes would look like the following.

While the $TPR = \frac{35}{35+5} = 87.5$ is high, overall $Accuracy = \frac{35+0}{100} = 45$, which is largely driven by low precision $PPV = \frac{35}{35+60} = 36.8$.

	Predicted (+)	Predicted (-)
Actual (+)	35	5
Actual (-)	60	0

While these calculations are simple and understandable, determining the predicted label is not as simple. In a simple case, given an outcome $Y = 1$, a voting rule would classify a probability of greater than 50% as $Y = 1$. However, it is fairly common that a trained classifier with strong performance may never produce a probability of more than 50%. In order to generalize accuracy, we can rely on one or a combination of the following measures.

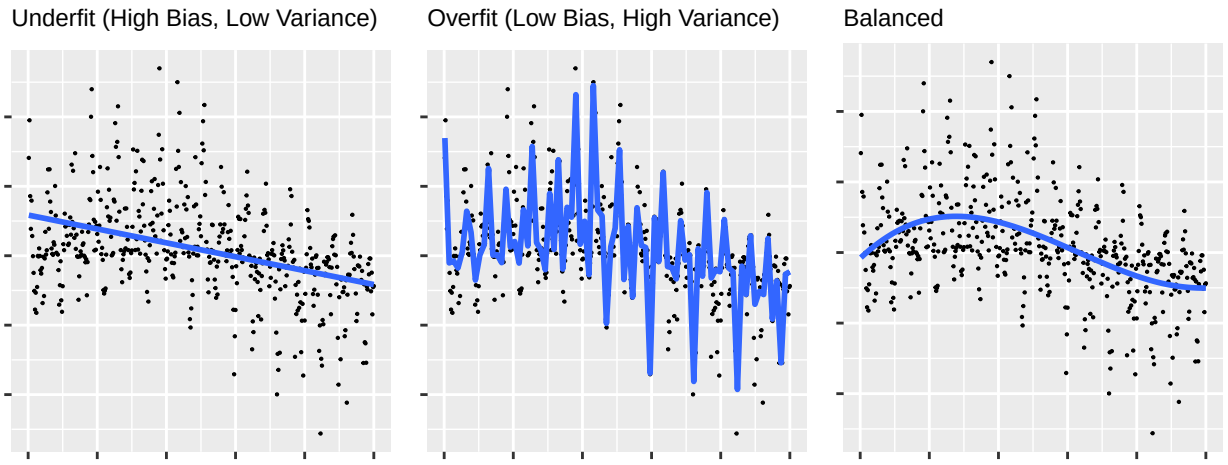
Measure	Description	Interpretation
Receiving Operating Characteristic (ROC) Curve	ROC curves plotpairs of TPRs and FPRs that correspond to varied discriminant thresholds between 0 and 1. By systematically testing thresholds. For example, TPRs and FPRs are calculated and plotted given probability thresholds $p = 0.2$, $p = 0.5$, and $p = 0.8$.	Once plotting the curve with TPR as Y and FPR as X, the area under the curve (AUC) represents robustness of the model, ranging from 0.5 (model is as good as a coin toss) to 1.0 (perfectly robust model). The AUC statistic is sometimes referred to as the "concordance".
F_1 Score	The score is formulated as $F_1 = 2 \times \frac{precision \times recall}{precision + recall} = 2 \times \frac{PPV \times TPR}{PPV + TPR}$ where precision or PPV $= \frac{TP}{TP+FP}$ and recall or TPR $= \frac{TP}{TP+FN}$	The measure is bound between 0 and 1, where 1 is the top score indicating a better model.

Bias & variance

Humans tend towards simple and seemingly elegant explanations of phenomena that allow for comfortably generalizing insights across populations. The simplicity is favored when generalizing insights as it is more accessible to more people. Generalizations sometimes do not, however, accurately describe a phenomenon. Thus, as humans seek to improve theories and representations of phenomena, explanations become increasingly complex to account for progressively more intricate patterns based on observation. As patterns become more complex, humans will tend to account for edge cases, but these patterns may simply be noise. This balance between generalizability and accuracy is the basis of the *Bias-Variance Tradeoff*.

To illustrate this tradeoff, the three graphs below exemplify the two extremes and the balanced case.

- **Underfitting**. Starting from the left, an overly simplistic explanation might boil down a relationship too much to the point that the model is "underfitted" – that the true shape of the relationship is not captured and the model is not responsible when generalized to new data. The variance of predictions in the training set is likely to be high and the same tends to be true for the test set. Models that have high bias (overly simplfified) tend to tell a digestable story, but have little predictive value.
- **Overfitting**. The middle graph shows a low bias model, which is often a far more complex model with more features and polynomials to improve the model fit. At first glance, captures much of the variability in the point-spread. While low bias models tend to have low bias in the training step, their predictions in out-of-sample testing tend to have high varaince as the model was calibrated to noise and is thus extra sensitive to noise.
- **Goldilocks**. The last graph shows a good balance between the two extremes: Just enough complexity to capture the curvature, but not so much to mold to every crevice of the data.



As we move through this chapter, the bias-variance tradeoff is a critical consideration. It informs how models are formulated and deployed.

Section 2 - Methods

In the context of healthcare coverage, we will review an application of KNNs, then explore two types of tree-based learning, then wrap up with logistic regression. We will need to import pre-cleaned healthcare coverage data from the American Community Survey (ACS), which is available from US Census Bureau website.

```
health <- read.csv("data/lecture7.csv")
str(health)
```

```
## 'data.frame':    22354 obs. of  7 variables:
##  $ coverage: Factor w/ 2 levels "Coverage","No Coverage": 2 2 2 2 2 2 2 2 2 2 ...
##  $ age     : int  33 22 28 25 42 51 36 26 55 23 ...
##  $ wage    : int  0 0 0 0 50000 0 0 0 37000 9400 ...
##  $ cit     : Factor w/ 2 levels "Citizen","Non-citizen": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mar     : Factor w/ 5 levels "Divorced","Married",..: 2 2 3 2 2 3 3 3 1 3 ...
##  $ educ    : Factor w/ 4 levels "Graduate Degree",..: 2 2 2 3 2 2 2 3 3 2 ...
##  $ race    : Factor w/ 8 levels "Amer. Ind.","Asian",..: 8 8 8 8 8 3 8 3 8 3 ...
```

KNNs in applications

Review of KNNs

Review

Recall that in the case of KNNs, all variables should be in the same scale such that each input feature has equal weight. A review of the data indicates that the health data is not in the appropriate form to be used.

Data preparation: Mixed variable formats

The continuous variable can be discretized by first binning records at equal intervals. For simplicity, we'll bin the age and wage varaibles in the following manner:

- `age`: 10 year intervals.
- `wage`: $20,000 intervals, topcoded at $200,000.

Upon binning, each variable needs to be set as a factor.

```
#Age
  age <- round(health$age / 10) * 10
  age <- factor(age)

#Wage
  wage <- round(health$wage / 20000) * 20000
  wage[wage > 200000] <- 200000
  wage <- factor(wage)
```

For all discrete features including the newly added `age` and `wage` variables, we can convert them into dummy matrices (e.g. all except one level in a discrete feature is converted into a binary variable). The former can be easily achieved by using the `model.matrix()` method, which returns a binary matrix for all levels:

```
  model.matrix(~health$variable - 1)
```

As is proper in preparation of dummy variables, if there are k levels in a given discrete variable, we should only keep $k - 1$ dummy variables For example, citizenship is a two level variable, thus we only need to keep one of two dummies. It's common to leave out the level with the most records, but any consistently appearing level will do.

```
#Keep only column 2
  cit <- as.data.frame(model.matrix(~ health$cit - 1)[, 2])

#Keep columns 1 to 4, leave out column 5
  mar <- as.data.frame(model.matrix(~ health$mar - 1)[,1:4])

#Keep columns 1 to 4, leave out column 5
  educ <- as.data.frame(model.matrix(~ health$mar - 1)[,1:4])

#Keep columns 1 to 7, leave out column 8
  race <- as.data.frame(model.matrix(~ health$race - 1)[,1:7])

#Keep columns 2 to 11, leave out column 1
  wage <- as.data.frame(model.matrix(~ wage - 1)[,2:11])

#Keep columns 2 to 8, leave out column 1
  age <- as.data.frame(model.matrix(~ age - 1)[,2:8])
```

Now the data can be combined. Notice that the new dataset `knn.data` has 34 features. Compared with the `health` dataset's 7 features. Note that perform these transformations are necessary given mixed variable types; however, a datasets containing continuous variables only does not require any manipulation other than scaling.

```
#Combine all the newly transformed data
  knn.data <- as.data.frame(cbind(coverage = as.character(health$coverage),
                                  wage, age, cit, educ, mar, race))

#Dimensions
  dim(health)
```

```
## [1] 22354     7
```

```
  dim(knn.data)
```

```
## [1] 22354    34
```

Sample partition

As is proper, the next step is to partition the data. For simplicity, we'll create a vector that will split the data into two halves, denoting the training set as `TRUE` and the test set as `FALSE`. We then split the data into four objects:

- Two objects contain the input features for each train and test sets.
- Two objects contain the labels for each train and test sets.

Splitting into four objects is common practice. `ytrain` and `xtrain` are used to train the model. Then, `xtest` is used to produce predictions. Then, `ytest` is used to calculate accuracy of the predictions.

```
#Train-Test
  rand <- runif(nrow(knn.data))
  rand <- rand > 0.5

#Create x-matrix. Use "-1" in the column argument to keep everything except column 1
  xtrain <- knn.data[rand == T, -1]
  xtest <- knn.data[rand == F, -1]

#Create y-matrix
  ytrain <- knn.data[rand == T, 1]
  ytest <- knn.data[rand == F, 1]
```

Model

Now the data is ready, the process is fairly simple. The supervised learning library "class" needs to be called to be able to take advantage of the `knn()` method.

```
#Call "class" library
  library(class)
```

The `knn()` syntax is simple, only requiring a few parameters:

- training features: `xtrain`
- test features: `xtest`
- training labels: `cl`
- number of neighbors: `k`
- return probability?: `prob`

In this trial run, we set $k = 10$, meaning the 10 nearest neighbors in euclidean distance are used to predict a given observation's value. The method returns an object containing the predicted labels of the test set. If `prob = TRUE`, the object also contains the probabilities, which can be accessed by using the `attr()` method.

```
#Run model
  pred <- knn(xtrain, xtest, cl = ytrain, k = 10, prob = TRUE)

#Check model object
  str(pred)
```

```
##  Factor w/ 2 levels "Coverage","No Coverage": 2 2 2 1 2 2 2 2 2 1 ...
##  - attr(*, "prob")= num [1:11309] 0.698 0.689 0.851 0.718 0.806 ...
```

```
#Extract probabilities
  test.prob <- attr(pred, "prob")
```

Using the extracted probabilities, we now can calculate the accuracy using the True Positive Rate (TPR) using a probability cutoff of 0.5. Typically, one would expect a 2×2 matrix given a binary label; However, our confusion matrix is a 1×2 matrix with a TPR of 49.9%, indicating that the model does not perform well

at the $p = 0.5$ cutoff does not yield promising results. The selected cutoff is fairly arbitrary and one cutoff does not necessarily provide an indication of the model's robustness [or lack thereof].

```
#TPR
  pred.class <- test.prob
  pred.class[test.prob >= 0.5] <- "Coverage"
  pred.class[test.prob < 0.5] <- "No Coverage"

#Confusion matrix
  table(ytest, pred.class)
```

```
##              pred.class
## ytest         Coverage
##   Coverage        5669
##   No Coverage     5640
```

To more robustly test the model, we will rely on the Receiving-Operating Characteristic (ROC) Curve and the Area Under the Curve (AUC). The ROC calculates the TPR and FPR at many thresholds, that produces a curve that indicates the general robustness of a model. The AUC is literally the area under that curve, which is a measure between 0.5 and 1 where the former indicates no predictive power and 1.0 indicates a perfect model.

Upon installing and loading the `ROCR` library, the library requires a minimum of two methods to be run: `prediction()` and `performance()`. The `prediction()` method takes two arguments: (1) the predicted probabilities of the test set, (2) the labels of the test set. This method calculates TPR, FPR and other accuracy measures at many different cutoffs from Infinity to 1.0 – at discrete intervals, of course – and outputs a prediction object (`pred.obj` in example below) containing the accuracies.

```
#Call the "ROCR" library"
  library(ROCR)

#Calculate accuracies given different cutoffs.
  pred.obj <- prediction(test.prob, ytest)
```

The `performance()` method is used to extract accuracy statistics from `pred.obj`. The AUC, for example, can be extracted using the command below and the AUC statistic is captured under `pred.obj@y.values`. Note that the AUC is less than 0.5, indicating that this is an egregiously poor performing model.

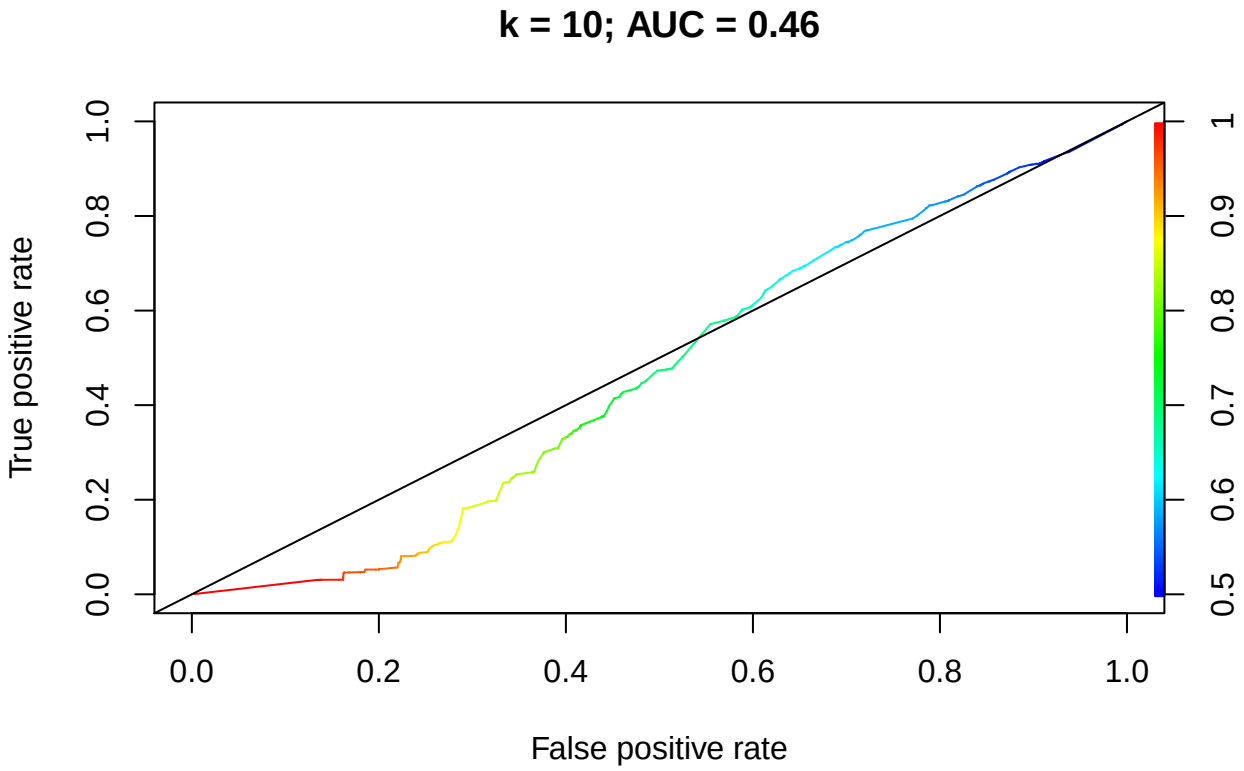
```
#Calculate AUC
  auc.knn <- performance(pred.obj, "auc")
  str(auc.knn)
```

```
## Formal class 'performance' [package "ROCR"] with 6 slots
##   ..@ x.name      : chr "None"
##   ..@ y.name      : chr "Area under the ROC curve"
##   ..@ alpha.name  : chr "none"
##   ..@ x.values    : list()
##   ..@ y.values    :List of 1
##   .. ..$ : num 0.464
##   ..@ alpha.values: list()
```

Nonetheless, it is worth seeing the shape of the curve. To plot the ROC, we need to extract the FPR and TPR accuracies from `pred.obj`. The resulting object can be plotted. Typically, a remotely well-performing model will curve above the diagonal line with an $AUC > 0.5$. Closer the line is to reaching TPR = 1 for all values of FPR, the more robust the model. While we can optimize for different values of k, KNNs are not well-suited for this task.

```r
#Calculate ROC
  pred.knn <- performance(pred.obj, "tpr", "fpr")

#Plot
  plot(pred.knn, colorize = TRUE,
       main = paste0("k = 10; AUC = ",round(unlist(auc.knn@y.values), 2)))
  abline(a = 0, b = 1)
```



Decision Trees

In everyday policy setting and operations, decision trees are a common tool used for communicating complex processes, whether for how an actor moves through intricate and convoluted bureaucracy or how a sub-population can be described based on a set of criteria. While the garden variety decision tree can be laid out qualitatively, supervised learning allows decision trees to be create in an empirical fashion that not only have the power to aesthetically communicate patterns, but also predict how a non-linear system behaves.

As was demonstrated at the beginning of this chapter, decision trees use a form of recursive partitioning to learn patterns, doing so using central concepts of *information theory*. There are a number key concepts that guide the use of the decision tree algorithm, including (1) nodes and splits, (2) termination criteria, and (3) normalized information gain.

(1) Nodes + Splits

Recalling the healthcare insurance decision tree, the tree can be characterized by circles and lines.

(2) Termination Criteria

Recalling the healthcare insurance decision tree, the tree can be characterized by circles and lines.

(3) Normalized Gain

To understand information gain means to understand the concept of *entropy*, which is a measure of purity or certainty of information.

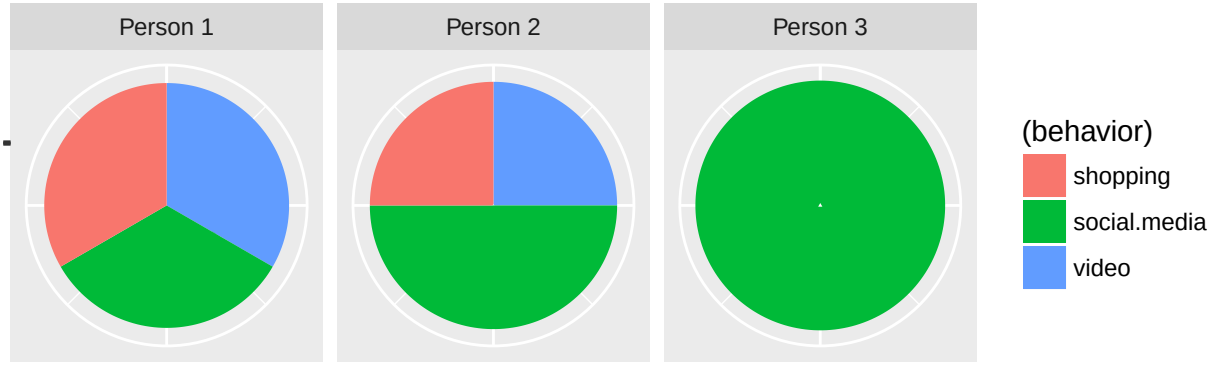
Given discrete classes or "states", entropy is defined as:

$$\text{Entropy} = \sum -p_i log_2(p_i)$$

where i is an index of states, p is the proportion of observations that are in state i, and $log_2(p_i)$ is the Base 2 logarithm of the proportion for state i. For each record in a data set, entropy can be calculated as a measure of homogeneity and consistency. For example, the data below describes the behavior of three hypothetical people.

Person	% Social Media	% Shopping	% Video Watching
1	33.33%	33.33%	33.33%
2	50%	25%	25%
3	100%	0%	0%

Each person's online behavior can be represented using pie charts. Person 3 spend all his/her online time on social media, whereas Person 1 shows the most heterogeneous behavior, splitting time evenly across all types of activity. Using entropy, we can easily quantify whether someone splits their time evenly, spends it all on one type of activity, and all combinations in between.



[An entropy equation can be]]

```
entropy <- function(x, start, end){
    prop <- x[,start:end]/rowSums(x[,start:end])
    logprop <- log(prop)
    return(rowSums(prop*logprop, na.rm = T))
}
```

that helps to identify when an empirical variable contains information. Whether its failure analysis of engineering mechanisms or developing customer profiles of program participation, decision trees can help characterize intricate, non-linear patterns in data.

The point at which a feature is split is known as a decision node, the trunk of the tree from which all branches spring is known as the root node, and the termini of the tree with the most homogeneous sub-samples are known as leafs.

The Gist. The structure of a decision tree can be likened to branches of a tree: moving from the base of the tree upwards, the tree trunk splits into two or more large branches, which then in turn split into even smaller branches, eventually reaching even small twigs with leaves. Given a labeled set of data that contains input features, the branches of a decision tree is grown by subsetting a population into smaller, more homogeneous

units. In other words, moving from the root of the tree to the terminating branches, each subsequent set of branches should contain records that are more similar, more homogeneous or purer.

How is a decision tree empirically grown? For this, we can rely on pseudo-code to lay out the process for the C4.5 algorithm, which is perhaps the most commonly implemented decision tree algorithm described in Quinlan (1993):

```
C4.5 (Sample = S, Target = Y, Input Features = X)
  Screen records for cases that meet termination criteria.
      If each base case that is met, partition sample to isolate homogeneous cases.
  For each input feature X, calculate the normalized information gain IG from splitting X into two subs
  Partition S into two partitions using feature X that corresponds to the highest IG.
  Repeat process for each sub-partition until termination criteria is met.
```

Issues. While decision trees are can be interpreta and discrete method for classification [and regression], a fully grown tree may suffer from overfitting. As a tree is grown, nodes become smaller and smaller, eventually causing nodes to overstate their accuracy while surfacing irregular patterns. [bias-variance]

Decision Trees in Practice

```
#Train-Test
  rand <- runif(nrow(health))
  rand <- rand > 0.5

#Create x-matrix. Use "-1" in the column argument to keep everything except column 1
  train <- health[rand == T, ]
  test <- health[rand == F, ]
```

```
library(rpart)
fit <- rpart(coverage ~ age + wage + cit + mar + educ + race, method = "class", data = train)

#Accuracy
  tree.pred <- predict(fit, test, type='prob')
  pred.obj <- prediction(tree.pred[,2], test[,1])
  pred.rpart <- performance(pred.obj, "tpr", "fpr")
  auc.rpart <- performance(pred.obj, "auc")

#Plot AUC
  plot(pred.knn, colorize=T, lwd=2, main = paste0("AUC = ",round(unlist(auc.rpart@y.values),2)))
  abline(a = 0, b = 1)
```

Random Forests

- statistical assumptions and mechanics, risks/strengths, implementation, non-technical explanation
- Gini Decrease or Mean Gini Decrease "Every time a split of a node is made on variable m the gini impurity criterion for the two descendent nodes is less than the parent node. Adding up the gini decreases for each individual variable over all trees in the forest gives a fast variable importance that is often very consistent with the permutation importance measure."

```
  randomForest(<formula>, <data>)
```

Using the

```
  library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:gridExtra':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
#Run Random Forest
  fit <- randomForest(coverage ~ age + wage + cit + mar + educ + race, method = "class", data = train)

#Check output
  summary(fit)
```

```
##                 Length Class  Mode
## call               4  -none- call
## type               1  -none- character
## predicted      11031  factor numeric
## err.rate        1500  -none- numeric
## confusion          6  -none- numeric
## votes          22062  matrix numeric
## oob.times      11031  -none- numeric
## classes            2  -none- character
## importance         6  -none- numeric
## importanceSD       0  -none- NULL
## localImportance    0  -none- NULL
## proximity          0  -none- NULL
## ntree              1  -none- numeric
## mtry               1  -none- numeric
## forest            14  -none- list
## y              11031  factor numeric
## test               0  -none- NULL
## inbag              0  -none- NULL
## terms              3  terms  call
```

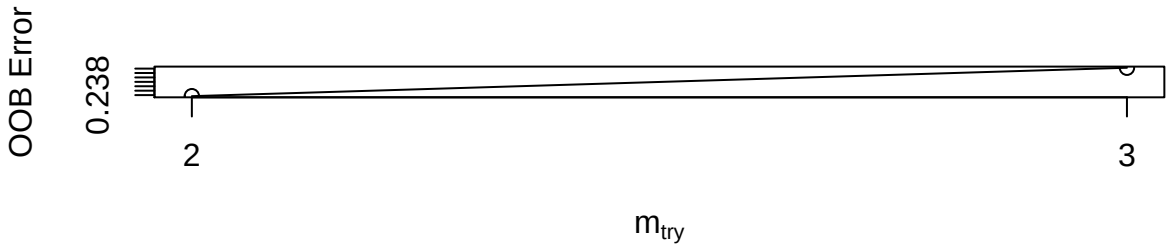
```
  print(importance(fit, type = 2))
```

```
##      MeanDecreaseGini
## age         1101.3834
## wage         709.0918
## cit          192.4973
## mar          362.0141
## educ         353.4087
## race         167.3363
```

```
#Search for most optimal number of input features
  fit.tune <- tuneRF(train[,-1], train$coverage, ntreeTry = 300, stepFactor = 1.5,
                improve = 0.01, trace = TRUE, plot = TRUE)
```

```
## mtry = 2  OOB error = 23.75%
## Searching left ...
## Searching right ...
## mtry = 3     OOB error = 25.04%
```

```
## -0.05419847 0.01
```



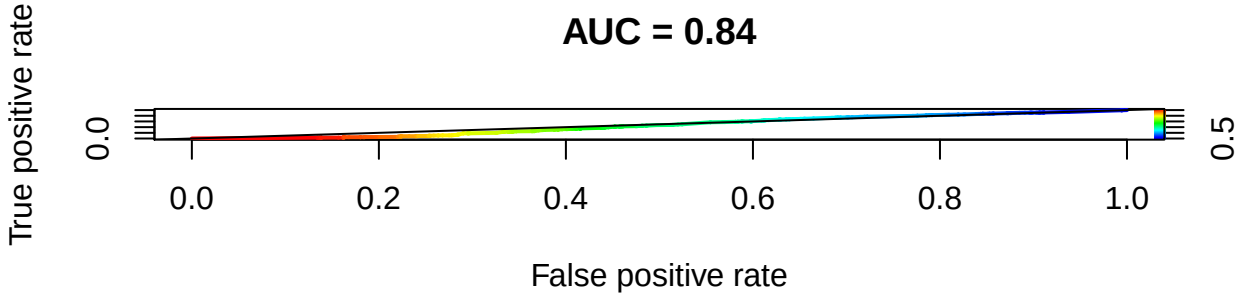

```
  tune.param <- fit.tune[fit.tune[, 2] == min(fit.tune[, 2]), 1]

#Re-run with tuned parameters
  set.seed(123)
  fit <-randomForest(coverage ~ ., data = train, mtry = tune.param,
                     importance = TRUE, ntree = 200)
  print(fit)
```

```
##
## Call:
##  randomForest(formula = coverage ~ ., data = train, mtry = tune.param,      importance = TRUE, ntree
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 23.86%
## Confusion matrix:
##             Coverage No Coverage class.error
## Coverage        4058        1515   0.2718464
## No Coverage     1117        4341   0.2046537
```

```
#Accuracy
  tree.pred <- predict(fit, test, type='prob')
  pred.obj <- prediction(tree.pred[,2], test[,1])
  pred.rf <- performance(pred.obj, "tpr", "fpr")
  auc.rf <- performance(pred.obj, "auc")

#Plot AUC
  plot(pred.knn, colorize=T, lwd=2, main = paste0("AUC = ",round(unlist(auc.rf@y.values),2)))
  abline(a = 0, b = 1)
```



Logistic Regression

- statistical assumptions and mechanics, risks/strengths, implementation, non-technical explanation

```r
fit2 <- glm(coverage ~ AGEP+ CIT2 + MAR2 + SCHL2 +RAC1P + WAGP,
            data=df_sub, family = "binomial")
```