# Applied Machine Learning - Classification

Max Kuhn (RStudio)

# Outline

- Performance Measures
- OkC Data
- Classification Trees
- More Bagging
- Naive Bayes Models

# Measuring Performance in Classification

# Load Packages

```
library(tidymodels)
```

```
## ── Attaching packages ──────────────────────────────────────── tidymodels 0.0.1 ──
```

```
## ✔ tibble    1.4.2     ✔ recipes   0.1.3
## ✔ purrr     0.2.5     ✔ yardstick 0.0.1
## ✔ dplyr     0.7.6     ✔ infer     0.3.1
## ✔ rsample   0.0.2
```

```
## ── Conflicts ─────────────────────────────────────────── tidymodels_conflicts() ──
## ✖ dplyr::combine()         masks Biobase::combine(), BiocGenerics::combine()
## ✖ rsample::fill()          masks tidyr::fill()
## ✖ dplyr::filter()          masks stats::filter()
## ✖ dplyr::lag()             masks stats::lag()
## ✖ purrr::lift()            masks caret::lift()
## ✖ yardstick::mnLogLoss()   masks caret::mnLogLoss()
## ✖ BiocGenerics::Position() masks ggplot2::Position(), base::Position()
## ✖ dplyr::select()          masks MASS::select()
## ✖ recipes::step()          masks stats::step()
```

# Illustrative Example

`yardstick` contains another test set example in a data frame called `two_class_example`:

```
two_class_example %>% head(4)
```

```
##    truth  Class1 Class2 predicted
## 1 Class2 0.00359  0.996    Class2
## 2 Class1 0.67862  0.321    Class1
## 3 Class2 0.11089  0.889    Class2
## 4 Class1 0.73516  0.265    Class1
```

Both `truth` and `predicted` are factors with the same levels. The other two columns represent *class probabilities*.

This reflects that most classification models can generate "hard" and "soft" predictions for models.

The class predictions are usually created by thresholding some numeric output of the model (e.g. a class probability) or by choosing the largest value.

# Class Prediction Metrics

With class predictions, a common summary method is to produce a *confusion matrix* which is a simple cross-tabulation between the observed and predicted classes:

```
two_class_example %>%
    conf_mat(truth = truth, estimate = predicted)
```

```
##           Truth
## Prediction Class1 Class2
##     Class1    227     50
##     Class2     31    192
```

These can be visualized using mosaic plots.

Accuracy is the most obvious metric for characterizing the performance of models.

```
two_class_example %>%
    accuracy(truth = truth, estimate = predicted)
```

```
## [1] 0.838
```

However, it suffers when there is a *class imbalance*; suppose 95% of the data have a specific class. 95% accuracy can be achieved by predicting samples to be the majority class.

There are measures that correct for the natural event rate, such as Cohen's Kappa.

# Two Classes

There are a number of specialized metrics that can be used when there are two classes. Usually, one of these classes can be considered the *event of interest* or the *positive class*.

One common way to think about performance is to consider false negatives and false positives.

- the sensitivity is the *true positive rate*
- the specificity is the rate of correctly predicted negatives, or 1 - *false positive rate*.

From this, assuming that `Class1` is the event of interest:

```
##            Truth
## Prediction Class1 Class2
##     Class1    227     50
##     Class2     31    192
```

sensitivity = 227/(227 + 31) = 0.88

specificity = 192/(192 + 50) = 0.79

# Conditional and Unconditional Measures

Sensitivity and specificity can be computed from the `sens` and `spec` functions, respectively.

It should be noted that these are *conditional measures* since we need to know the true outcome.

The event rate is the *prevalence* (or the Bayesian *prior*). Sensitivity and specificity analogous to the *likelihood values*.

There are *unconditional* analogs to the *posterior values* called the positive predictive values and the negative predicted values.

A variety of other measures are available for two class systems, especially for *information retrieval*.

One thing to consider: what happens if our **threshold to call a sample an event is not optimal**?

# Changing the Probability Threshold

For two classes, the 50% cutoff is customary; if the probability of class #1 is >= 50%, they would be labelled as `Class1`.

What happens when you change the cutoff?

- Increasing it makes it harder to be called `Class1` $\Rightarrow$ fewer predicted events, specificity $\uparrow$, sensitivity $\downarrow$
- Decreasing the cutoff makes it easier to be called `Class1` $\Rightarrow$ more predicted events, specificity $\downarrow$, sensitivity $\uparrow$

With two classes, the **Receiver Operating Characteristic (ROC) curve** can be used to estimate performance using a combination of sensitivity and specificity.

To create the curve, many alternative cutoffs are evaluated.

For each cutoff, we calculate the sensitivity and specificity.

The ROC curve plots the sensitivity (eg. true positive rate) versus 1 - specificity (eg. the false positive rate).

The area under the ROC curve is a common metric of performance.

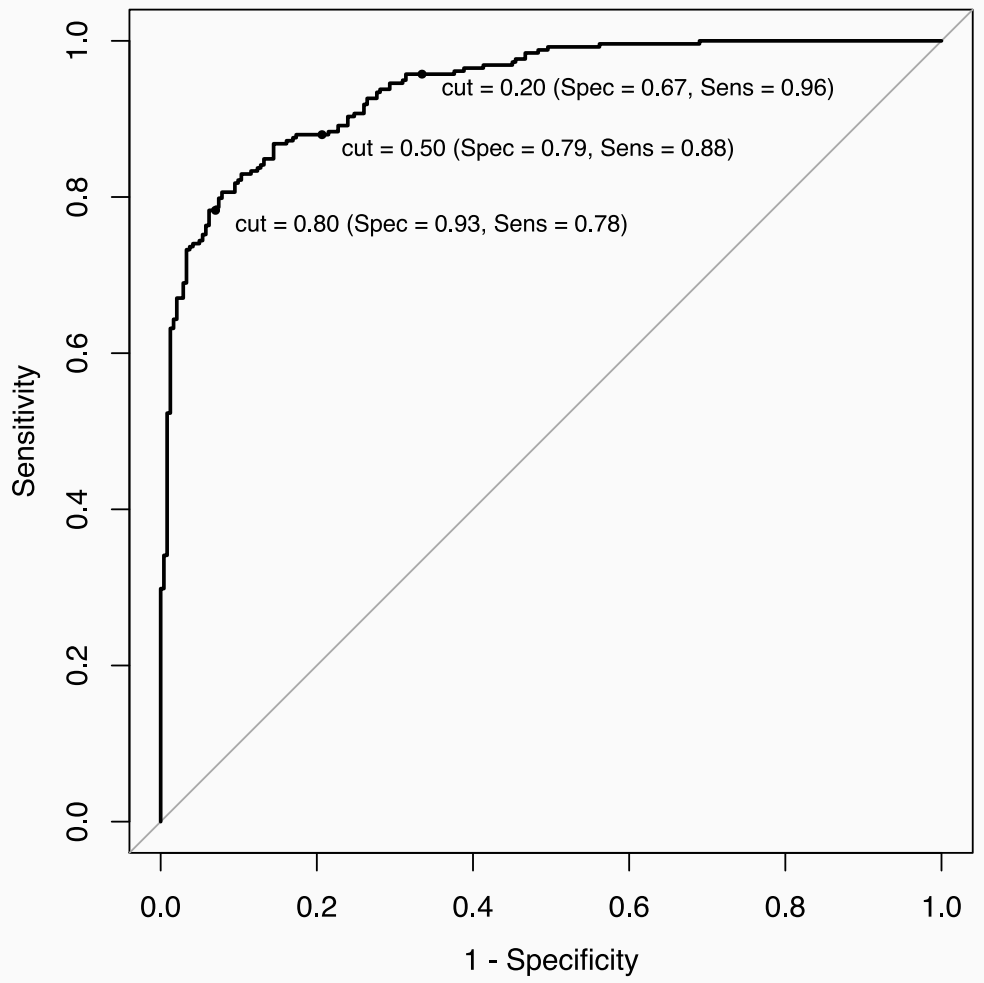
# The Receiver Operating Characteristic (ROC) Curve

```
library(pROC)
roc_obj <- roc(
    response = two_class_example$truth,
    predictor = two_class_example$Class1,
    # If the first level is the event of interest:
    levels = rev(levels(two_class_example$truth))
    )
```

```
auc(roc_obj)
```

```
## Area under the curve: 0.939
```

```
plot(
    roc_obj,
    legacy.axes = TRUE,
    print.thres = c(.2, .5, .8),
    print.thres.pattern = "cut = %.2f (Sp = %.3f, Sn =
    print.thres.cex = .8
)
```



# The Receiver Operating Characteristic (ROC) Curve

The ROC curve has some major advantages:

- It can allow models to be optimized for performance before a definitive cutoff is determined.
- It is *robust* to class imbalances; no matter the event rate, it does a good job at characterizing model performance.
- The ROC curve can be used to pick an optimal cutoff based on the trade-offs between the types of errors that can occur.

When there are two classes, it is advisable to focus on the area under the ROC curve instead of sensitivity and specificity.

Once an acceptable model is determined, a proper cutoff can be determined.

# OkC Data

These data contains several types of fields:

- a number of open text essays related to interests and personal descriptions
- single choice type fields, such as profession, diet, gender, body type, etc.
- multiple choice data, including languages spoken, etc.

We will try to predict whether someone has a profession in the STEM fields (science, technology, engineering, and math) using a random sample of the overall dataset.

The data are included in the workshop's GitHub repo:

```
load("Data/okc.RData")
okc_train %>% dim()
```

```
## [1] 4000   91
```

```
okc_test %>% nrow()
```

```
## [1] 1000
```

```
table(okc_train$Class)
```

```
##
##  stem other
##   729  3271
```

# Dealing with Class Imbalances

In our data, only 18.2% of the profiles are in STEM professions. This complicates the analysis since many models will overfit to the majority class.

There are two main strategies to deal with this:

- *Cost-sensitive learning* where a higher cost is attached to the minority classes. In this way, the fitting process puts more emphasis on those samples.
- *Sampling procedures* that modify the rows of the data to re-balance the training set.

Cost-sensitive models tend to only produce hard classifications so we will focus on the latter.

# Class Imbalance Sampling

There are a variety of methods for subsampling the data. Some exclude or replicate rows in the training set while others try to *synthesize* new data points to balance the classes.

The simplest method for dealing with the problem is to *down-sample* the data to make the number of STEM and non-STEM profiles the same.

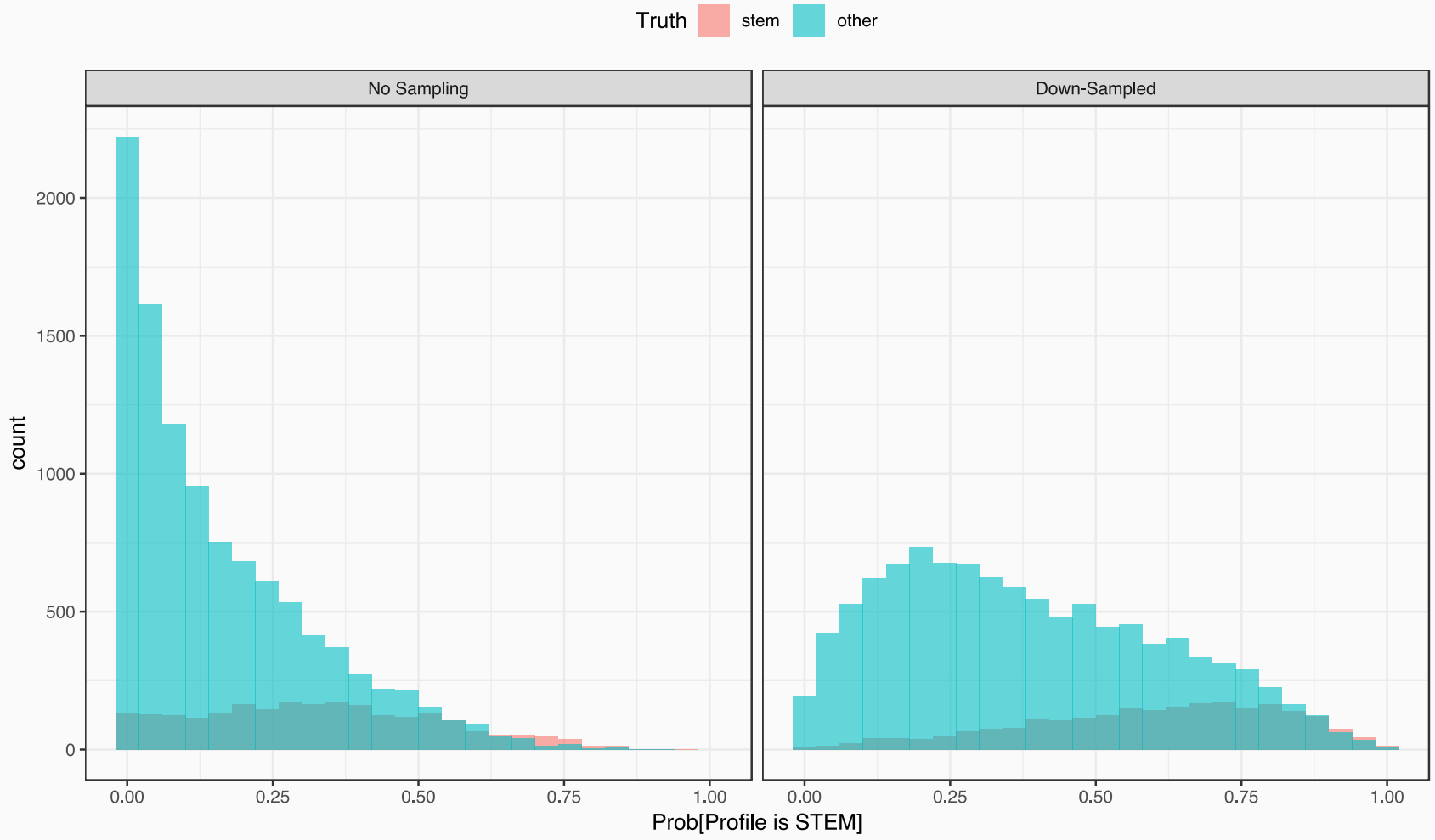
While it seems like throwing away most of the data is a bad idea, it tends top produce less pathological distributions of the class probabilities and *might* improve the ROC curve.

It is critical that:

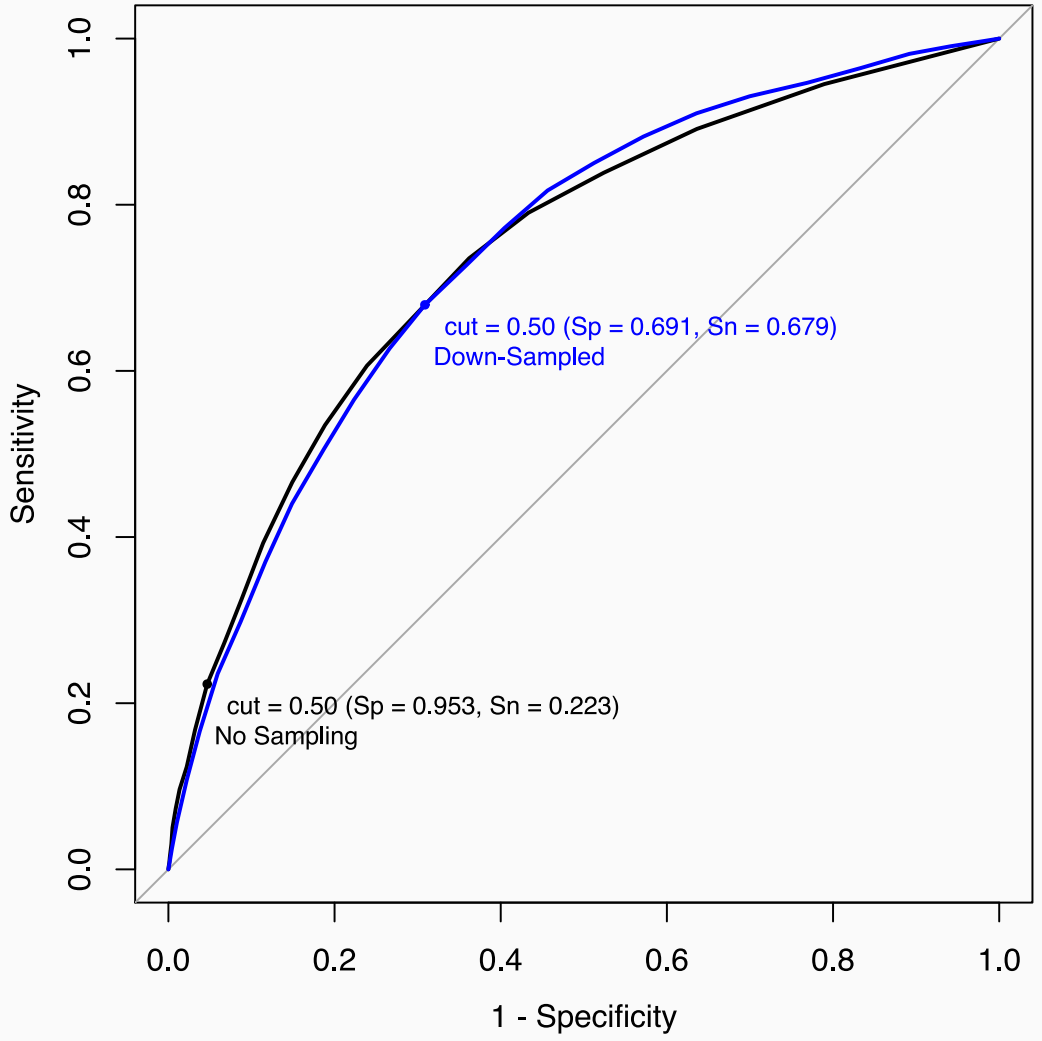
- the sampling should be done *inside of resampling*. Otherwise, the performance estimates can be optimistic.
- these sampling methods take place on the analysis set and not the assessment set

Note that for a simple logistic regression model, this mainly has the effect of changing the intercept.

# Calibration Effect (Test Set Example)



# Calibration Effect (Test Set Example)



# Resampling and Analysis Strategy

If we down-sample the data, the analysis set will consist of 1458 profiles (equally balanced). We'll again use 10-fold CV to resample the data.

Within each resample, the analysis data are down-sampled and the assessment sets are left alone.

The number of STEM profiles held-out would be about 72 and this should be sufficient to compute sensitivity.

The models will be optimized on the area under the ROC curve.

```
library(caret)
ctrl <- trainControl(
    method = "cv",
    # Also predict the probabilities
    classProbs = TRUE,
    # Compute the ROC AUC as well as the sens and
    # spec from the default 50% cutoff. The
    # function `twoClassSummary` will produce those.
    summaryFunction = twoClassSummary,
    savePredictions = "final",
    sampling = "down"
)
```

# Classification Trees

# Classification Trees

Tree-based classifiers conduct searches of the predictors to find the best split of the data to create two subsets.

"Best", in most cases, means that the class distribution is as *pure* as possible in the subsets (i.e. is mostly one class).

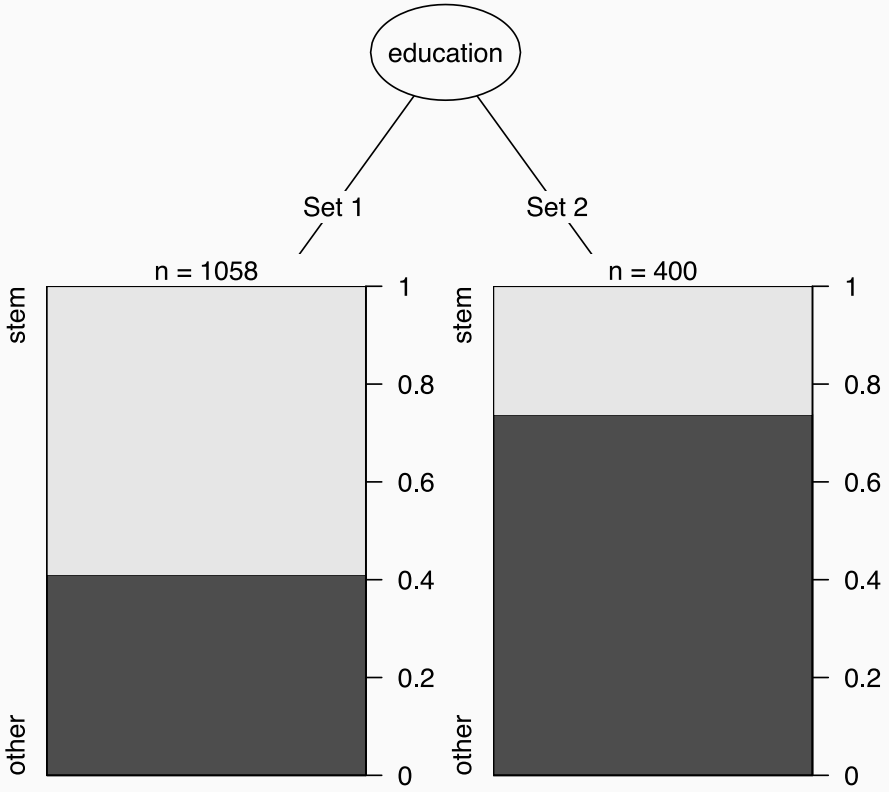
There are many different types of classification trees, including conditional inference trees, C5.0, Bayesian additive regression trees, globally optimal trees, CART, and others.

We'll focus on CART via the `rpart` package for these notes.

# Two Possible Splits - Which One is Better?

The data have been down-sampled so that classes have equal frequencies.

# Best First Split




The two sets were derived by the model and are not listed here due to their sizes.

"Set 1" includes

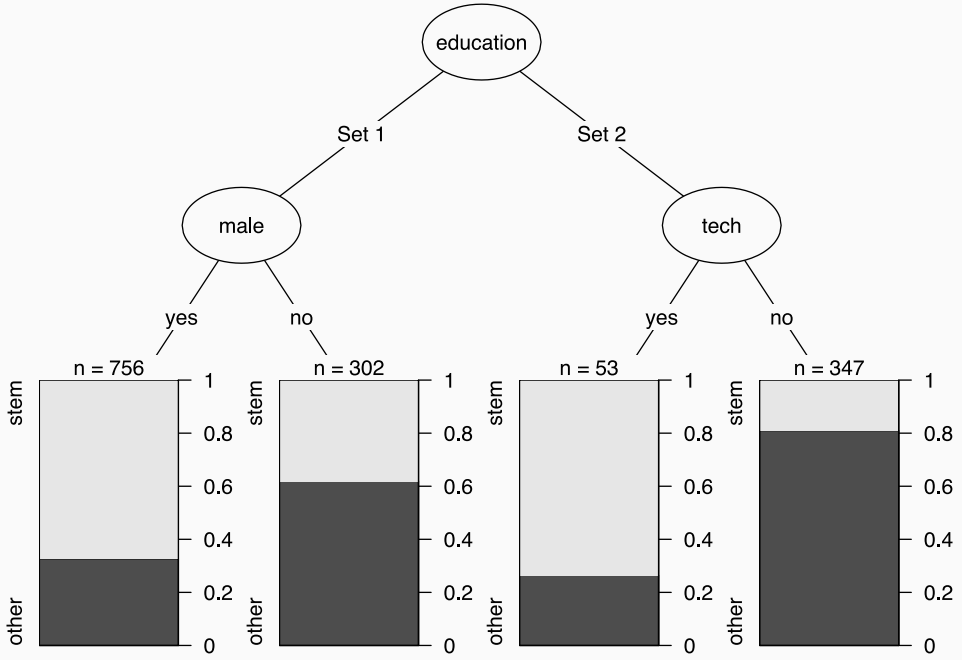
```
##  [1] "college_university"
##  [2] "dropped_out_of_college_university"
##  [3] "dropped_out_of_high_school"
##  [4] "dropped_out_of_ph_d_program"
##  [5] "graduated_from_college_university"
##  [6] "graduated_from_masters_program"
##  [7] "graduated_from_ph_d_program"
##  [8] "graduated_from_space_camp"
##  [9] "masters_program"
## [10] "working_on_space_camp"
```

# The Next Two splits

Once the initial split is made, the model will then split the resulting two data sets using new searches in those *leaves*.

The process continues until there are not enough data points left to accurately split or a pre-defined split limit has been reached.

This is the *tree growing* process and, once complete, most tree-based models begin to *prune* the trees using some method that balances complexity with performance.



# Classification Trees

`caret` contains multiple method for training CART models.

We'll go with `method = "rpart2"` which uses `maxdepth` as the tuning parameter.

Also, we'll use the non-formula interface so that the factor predictors are not converted to indicator variables.

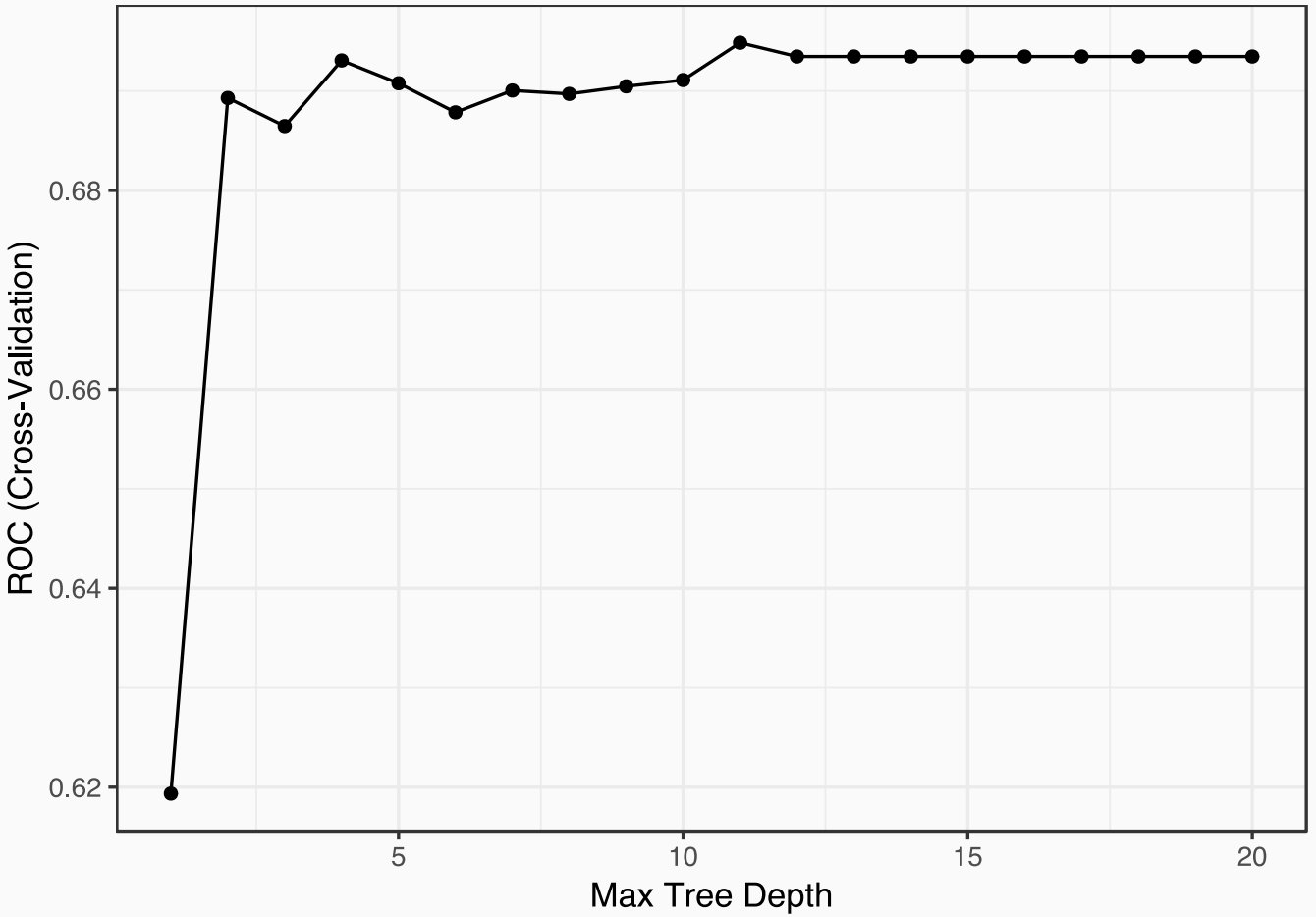
```
set.seed(5515)
cart_mod <- train(
    x = okc_train[, names(okc_train) != "Class"],
    y = okc_train$Class,
    method = "rpart2",
    metric = "ROC",
    tuneGrid = data.frame(maxdepth = 1:20),
    trControl = ctrl
)
```

# CART Model Results

```
cart_mod
```

```
## CART
##
## 4000 samples
##   90 predictor
##    2 classes: 'stem', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3600, 3601, 3599, 3600, 3600, 3600, ...
## Addtional sampling using down-sampling
##
## Resampling results across tuning parameters:
##
##   maxdepth  ROC    Sens   Spec
##    1        0.619  0.779  0.459
##    2        0.689  0.716  0.647
##    3        0.686  0.735  0.629
##    4        0.693  0.732  0.631
##    5        0.691  0.731  0.633
##    6        0.688  0.715  0.640
##    7        0.690  0.726  0.633
##    8        0.690  0.727  0.629
##    9        0.690  0.732  0.629
##   10        0.691  0.737  0.628
##   11        0.695  0.727  0.631
##   12        0.693  0.730  0.625
##   13        0.693  0.730  0.625
##   14        0.693  0.730  0.625
```

# CART Model Results



# CART Model

```
cart_mod$finalModel
```

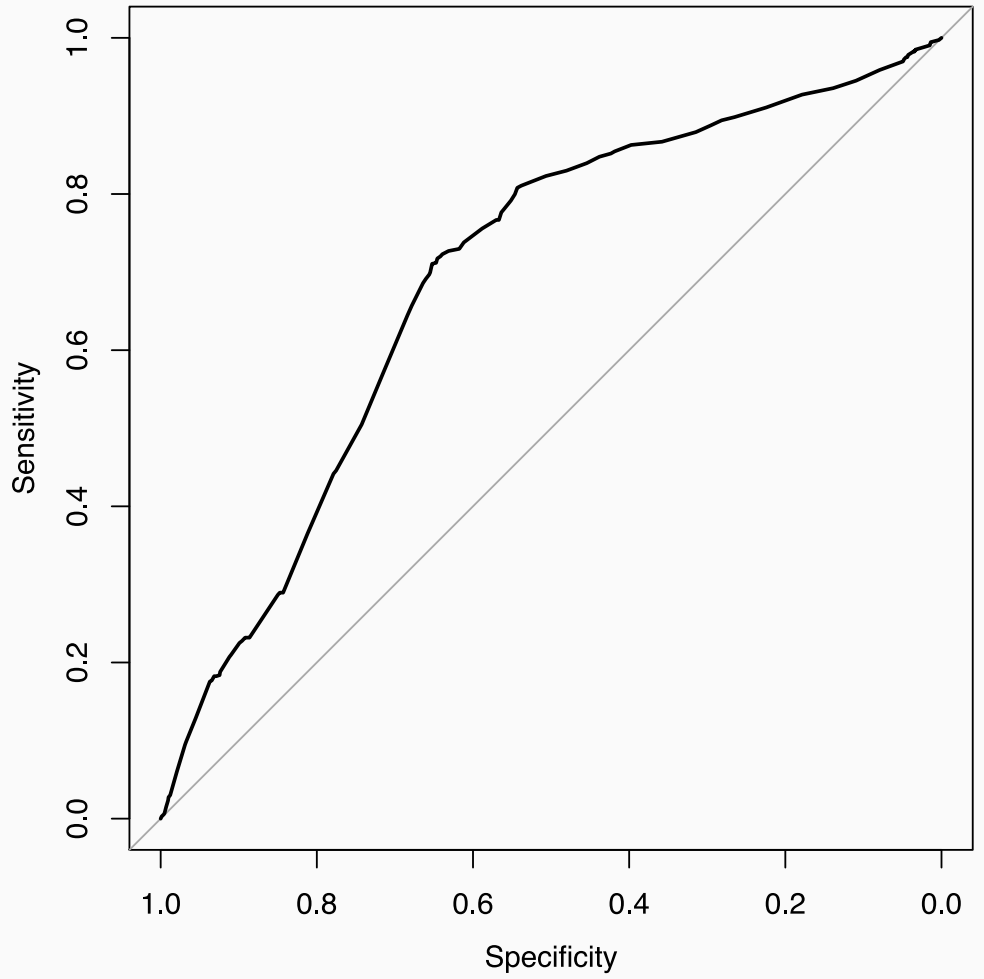
```
## n= 1458
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1458 729 stem (0.5000 0.5000)
##    2) male>=0.5 1018 414 stem (0.5933 0.4067)
##      4) education=college_university,dropped_out_of_college_university,dropped_out_of_high_school,dropped_out_of_masters_program,dropped_out_of_ph_d_program,graduated_from_college_unive
##        8) tech>=0.5 190  29 stem (0.8474 0.1526) *
##        9) tech< 0.5 616 243 stem (0.6055 0.3945)
##         18) income=missing,inc50000,inc60000,inc70000,inc80000,inc100000,inc150000 580 215 stem (0.6293 0.3707)
##           36) where_town=alameda,belmont,benicia,berkeley,castro_valley,daly_city,emeryville,fairfax,hayward,hercules,larkspur,martinez,menlo_park,millbrae,mountain_view,novato,oakland,
##             72) diet=diet_missing,anything,mostly_anything,strictly_anything,strictly_other,strictly_vegetarian 508 163 stem (0.6791 0.3209) *
##             73) diet=mostly_halal,mostly_other,mostly_vegan,mostly_vegetarian,other,vegan,vegetarian 38  12 other (0.3158 0.6842) *
##           37) where_town=albany,burlingame,el_cerrito,el_sobrante,pleasant_hill,richmond,san_anselmo,san_carlos,san_leandro,walnut_creek 34   8 other (0.2353 0.7647) *
##         19) income=inc20000,inc30000,inc40000,inc250000,inc1000000 36   8 other (0.2222 0.7778) *
##      5) education=ed_missing,dropped_out_of_space_camp,dropped_out_of_two_year_college,graduated_from_high_school,graduated_from_law_school,graduated_from_med_school,high_school,two_yea
##       10) tech>=0.5 43  16 stem (0.6279 0.3721) *
##       11) tech< 0.5 169  43 other (0.2544 0.7456) *
##    3) male< 0.5 440 125 other (0.2841 0.7159)
##      6) tech>=0.5 66  25 stem (0.6212 0.3788)
##       12) where_town=alameda,corte_madera,el_cerrito,mill_valley,mountain_view,oakland,other,pacifica,palo_alto,san_francisco,san_lorenzo,san_mateo 54  14 stem (0.7407 0.2593) *
##       13) where_town=albany,daly_city,emeryville,fremont,hayward,menlo_park,richmond,san_anselmo,san_leandro,vallejo 12   1 other (0.0833 0.9167) *
##      7) tech< 0.5 374  84 other (0.2246 0.7754) *
```

Note that there are 10 terminal nodes and thus 10 possible probability values.

# Classification Tree Average ROC Curve

ROC curves will be created for each model so a convenience function will be used repeatedly to create an *approximate* curve:

```
plot_roc <- function(x, ...) {
  roc_obj <- roc(
    response = x[["obs"]],
    predictor = x[["stem"]],
    levels = rev(levels(x$obs))
  )
  plot(roc_obj, ...)
}
plot_roc(cart_mod$pred)
```



# Resampled Confusion Matrix

Since there are 10 different assessment sets, separate confusion matrices can be constructed for each.

`train` has its own `confusionMatrix` method that can show aggregations of these matrices.

By default, the overall rates are shown in each cell.

```
confusionMatrix(cart_mod)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across r
##
##            Reference
## Prediction stem other
##      stem  13.2  30.1
##      other  5.0  51.6
##
##  Accuracy (average) : 0.6487
```

# Variable Importance Scores

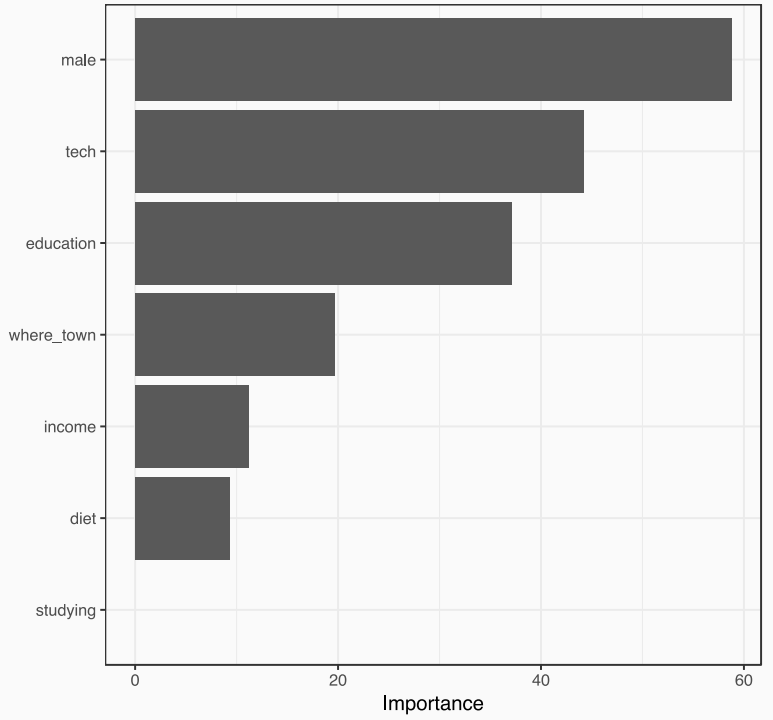
CART tracks the decrease in impurity in the nodes when splits are made.

These can be aggregated over the tree to produce another general importance score.

`rpart` can also keep track of splits not used in the model (e.g. surrogate, completing) that can be used when there are missing predictor values. The `varImp` method has options to turn these on/off.

Turning these off gives only the splits uses in the official tree.

```
cart_imp <- varImp(cart_mod, scale = FALSE,
                   surrogates = FALSE,
                   competes = FALSE)
ggplot(cart_imp, top = 7) + xlab("")
```



# Hands-On: I Don't Want to be a Dummy!

Take the previous code and *use the formula method* to create the model. For `train`, the formula method *always* creates dummy variables for predictors that are factors.

Is there any difference in performance?

Is the final model affected? How?

Take 15 min to answer these questions.

# Bagging (Again)

# Instability of Trees

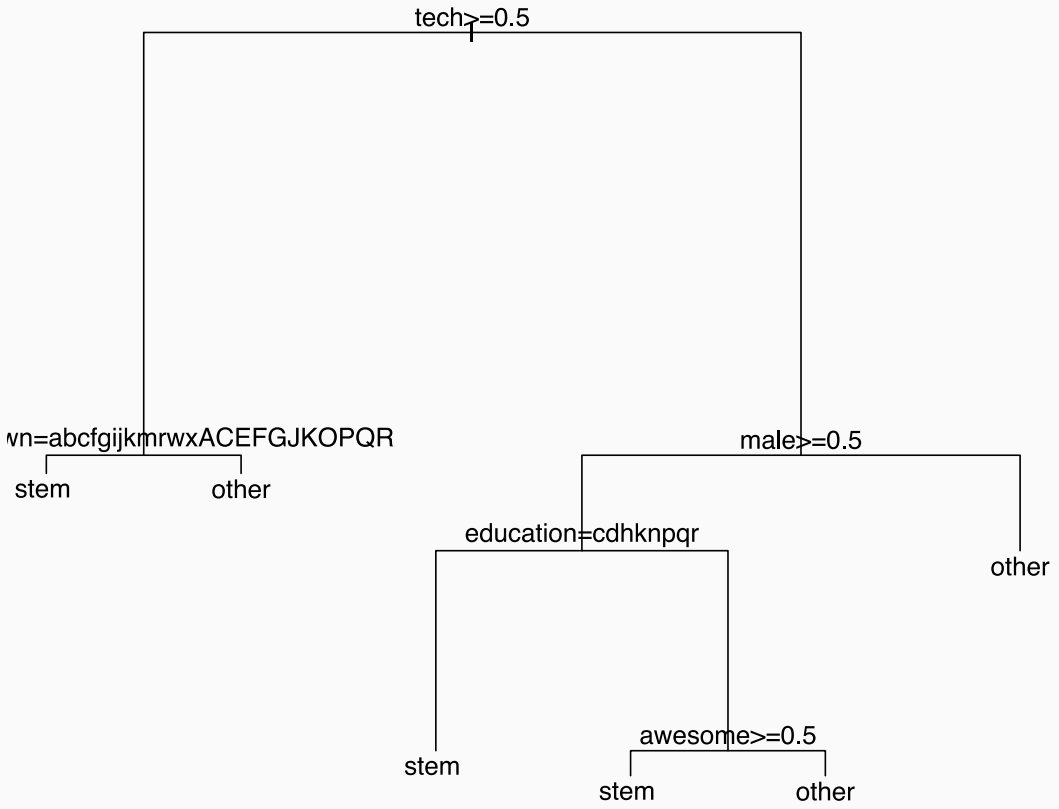
Many types of trees are *unstable* in that they have high variance; if the data are slightly changed, a large impact can be seen on the structure of the model.

This is generally bad and might make you question the interpretability of trees.

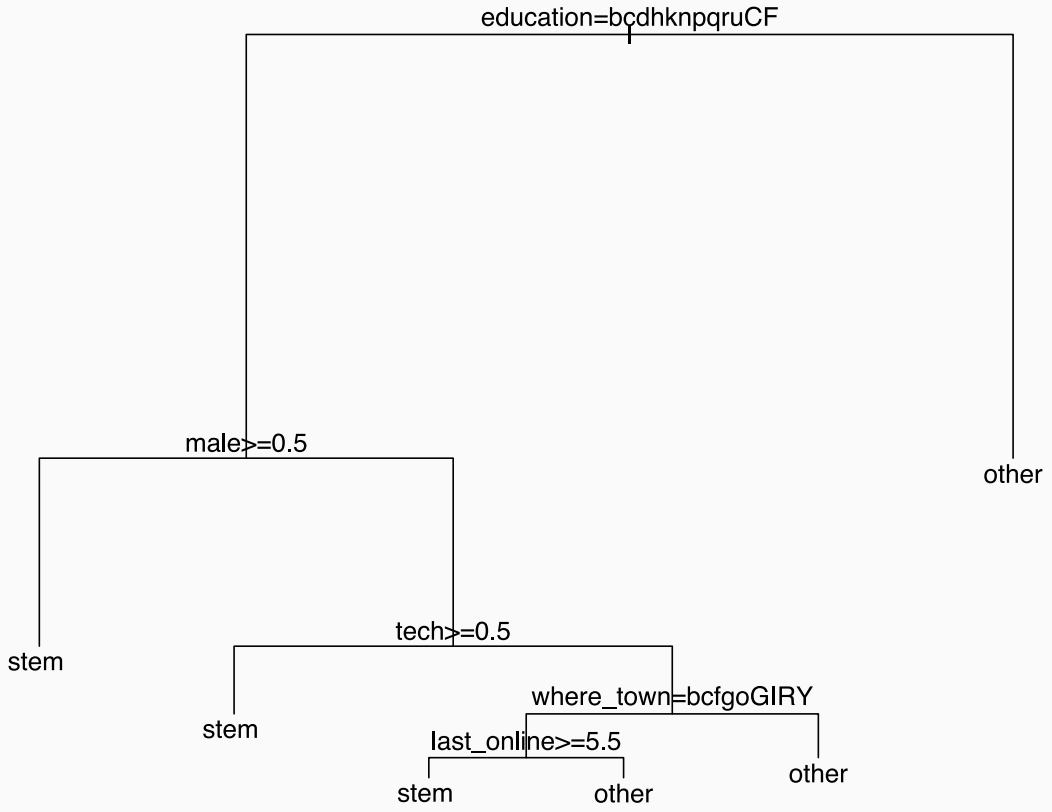
For example, what happens if we were to build our model on bootstrap samples of the training set?

In the three following plots, the maximum tree depth was capped at 5 to make the trees easier to visualize.

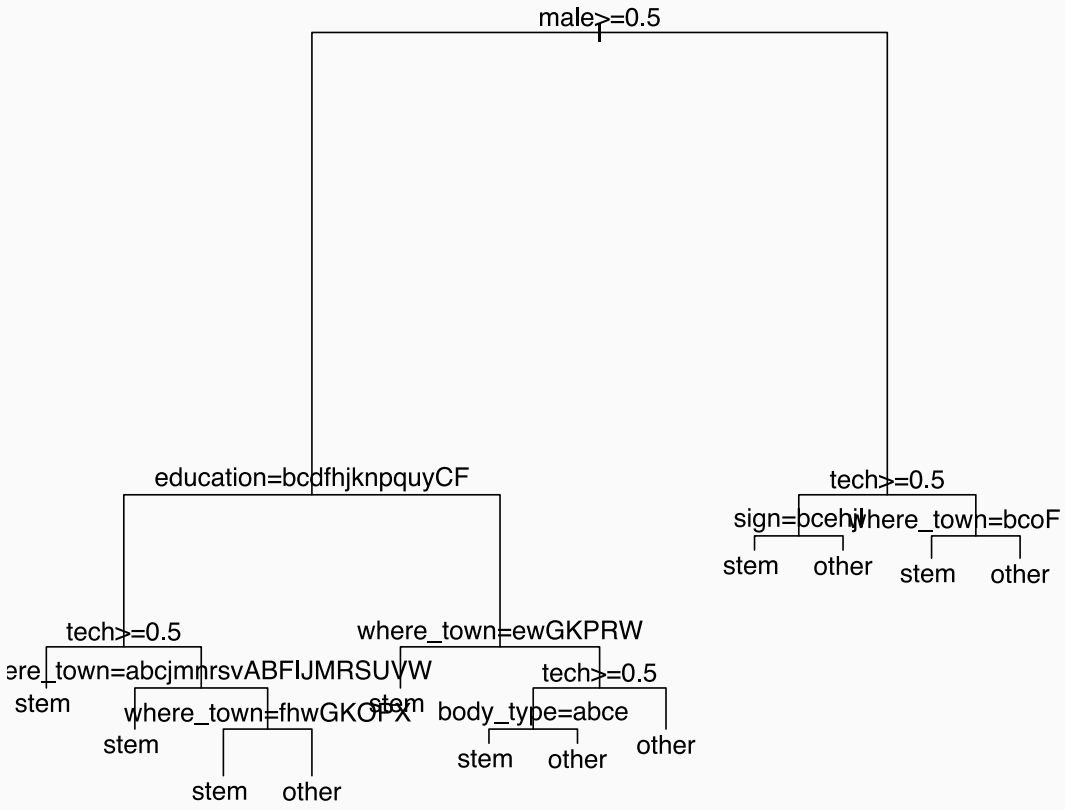
# Bootstrap Sample #1



# Bootstrap Sample #2



# Bootstrap Sample #3



# Lemonade from Lemons

The *nice* thing about this instability is that it makes trees great candidates for ensemble methods.

Since ensembles use multiple models, they are only effective when the constituent models are *diverse*; otherwise the same predictions are averaged.

Let's bag the CART trees by using bootstrap samples of the training set and growing the largest possible tree.

`caret` wraps `ipred::bagging` using `method = "treebag"`.

# Bagging CART Trees

```
set.seed(5515)
cart_bag <- train(
    x = okc_train[, names(okc_train) != "Class"],
    y = okc_train$Class,
    method = "treebag",
    metric = "ROC",
    trControl = ctrl
)
```

# Bagged CART Results

```
cart_bag
```

```
## Bagged CART
##
## 4000 samples
##   90 predictor
##    2 classes: 'stem', 'other'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3600, 3601, 3599, 3600, 3600, 3600, ...
## Addtional sampling using down-sampling
##
## Resampling results:
##
##   ROC    Sens   Spec
##   0.746  0.708  0.67
```

# Resampled Confusion Matrix

As measured by the default cutoffs, there is some increase in accuracy that is achieved by improving the specificity (27% versus CART's 30.1%).

```
confusionMatrix(cart_bag)
```

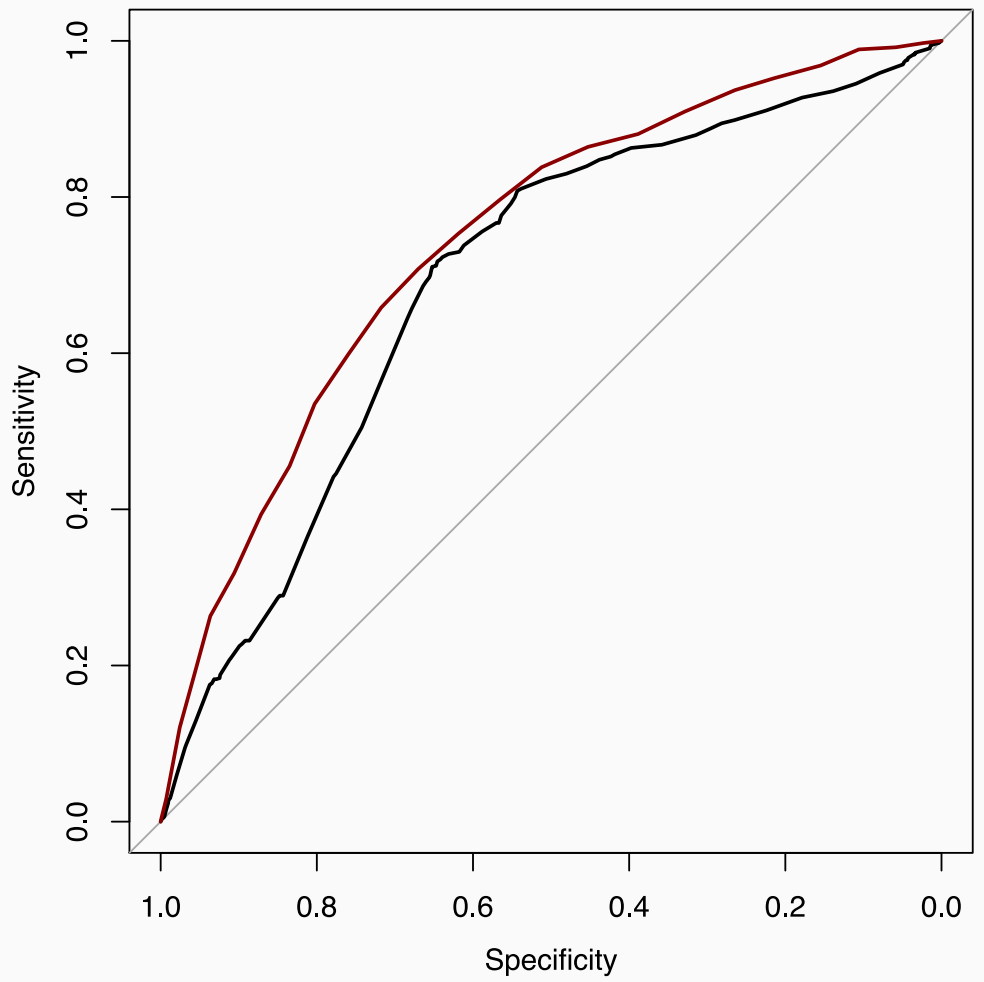
```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction stem other
##      stem  12.9  27.0
##      other  5.3  54.8
##
##  Accuracy (average) : 0.677
```

]

# Bagged Classification Tree Average ROC Curve

```
plot_roc(cart_mod$pred)
plot_roc(cart_bag$pred,
         col = "darkred",
         add = TRUE)
```

Although the bagged model's curve is uniformly better than the single tree, their performance is very similar for the cutoffs closest to upper left-hand corner.

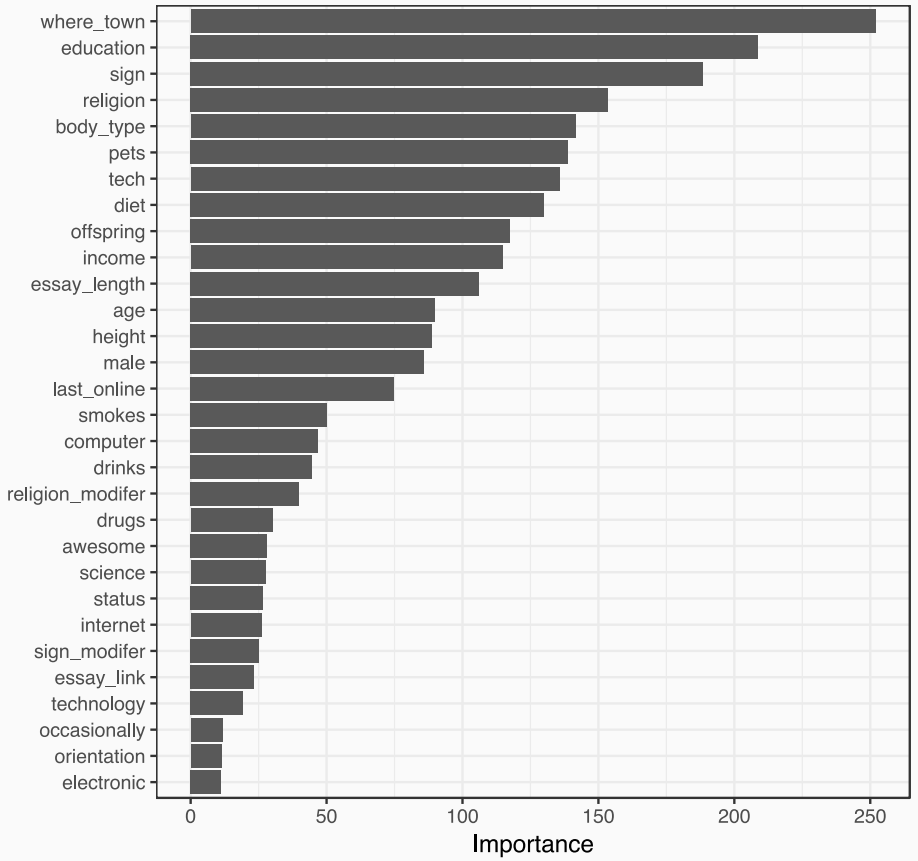


# Variable Importance Scores

When bagging, the CART importances scores for each tree are aggregated across trees.

Many more predictors are used in this model.

```
bag_imp <- varImp(cart_bag, scale = FALSE)
ggplot(bag_imp, top = 30) + xlab("")
```



# Hands-On: How Many Trees?

As previously mentioned, `caret` uses `ipred::bagging` to create the model.

Look at the help function to determine which `bagging` argument controls the number of bootstraps.

How can we make `train` use a different value? (hint: `?train`)

Does changing this affect the area under the ROC curve?

Take another 10 mins.

# Other Ensemble Methods

There are a variety of other methods for creating ensembles

- **Random forests** are just like bagging but the trees are made more diverse by randomly sampling a subset of predictors to be used in each split. (`ranger`)
- **Boosting** fits a *sequence* of trees and modifies the case-weights of each data point to increase diversity. (`xgboost`, `C50`)
- **Rotation forests** is PCA signal extraction on random subset of the data prior to creating the trees. (`rotationForest`)
- Regression **committees** adjust the outcome data over a sequence of models. (`Cubist`)
- **Stacking** is an ensemble method where different types of models can be blended together through averaging. (`caretEnsemble`)

R has multiple implmentations of these methods.

# Bayes' Rule

# Naive Bayes Models

This classification model is motivated directly from statistical theory based on Bayes' Rule:

$$Pr[Class|Predictors] = \frac{Pr[Class] \times Pr[Predictors|Class]}{Pr[Predictors]} = \frac{Prior \times Likelihood}{Evidence}$$

In English:

> Given our predictor data, what is the probability of each class?

The *prior* is the prevalence that was mentioned earlier (e.g. the rate of STEM profiles). This can be estimated or set.

Most of the action is in `Pr[Predictors|Class]`, which is based on the observed training set.

# So Why is it Naive?

Determining $Pr[Predictors|Class]$ can be very difficult without strong assumptions because it measures the *joint probability* of all of the predictors.

- For example, what is the correlation between a person's essay length and their religion?

To resolve this, **naive** Bayes assumes that all of the predictors are *independent* and that their probabilities can be estimated separately.

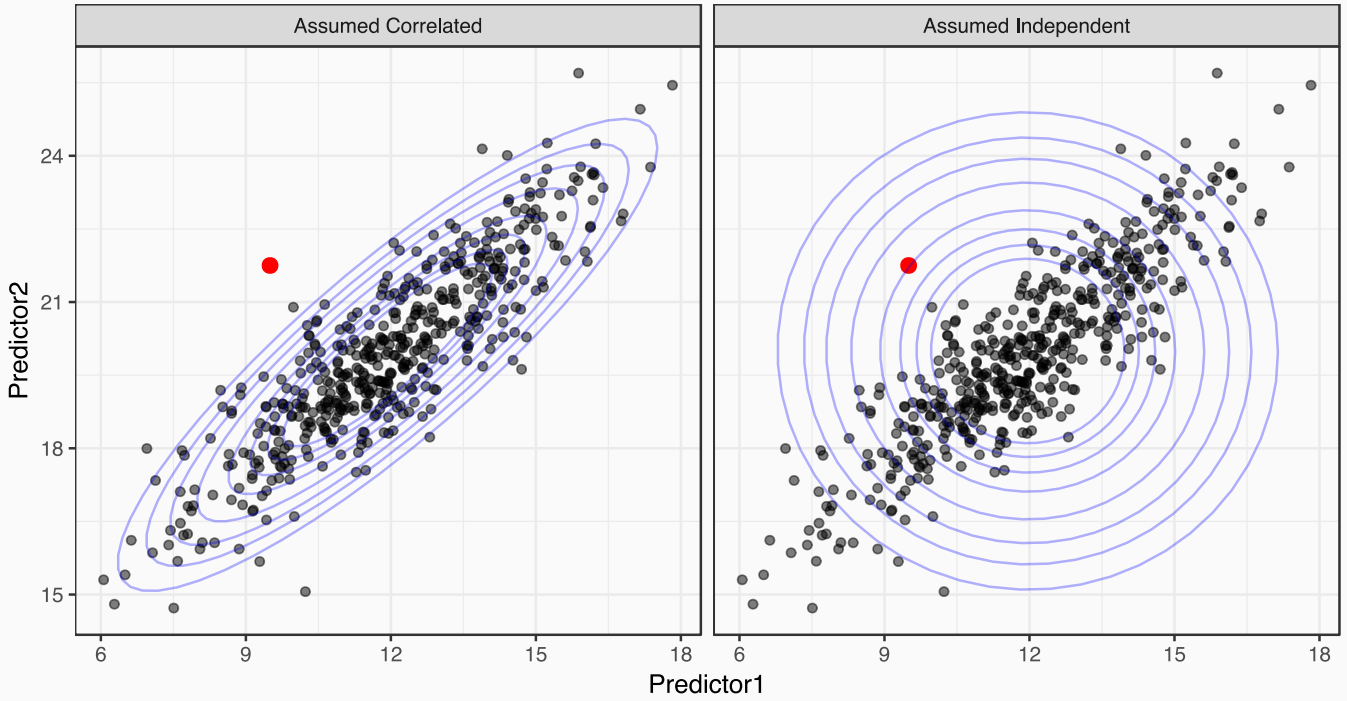
The joint probability is then the product of all of the individual probabilities (an example follows soon).

This assumption is almost certainly bogus but the model tends to do well despite this.

# The Effect of Independence

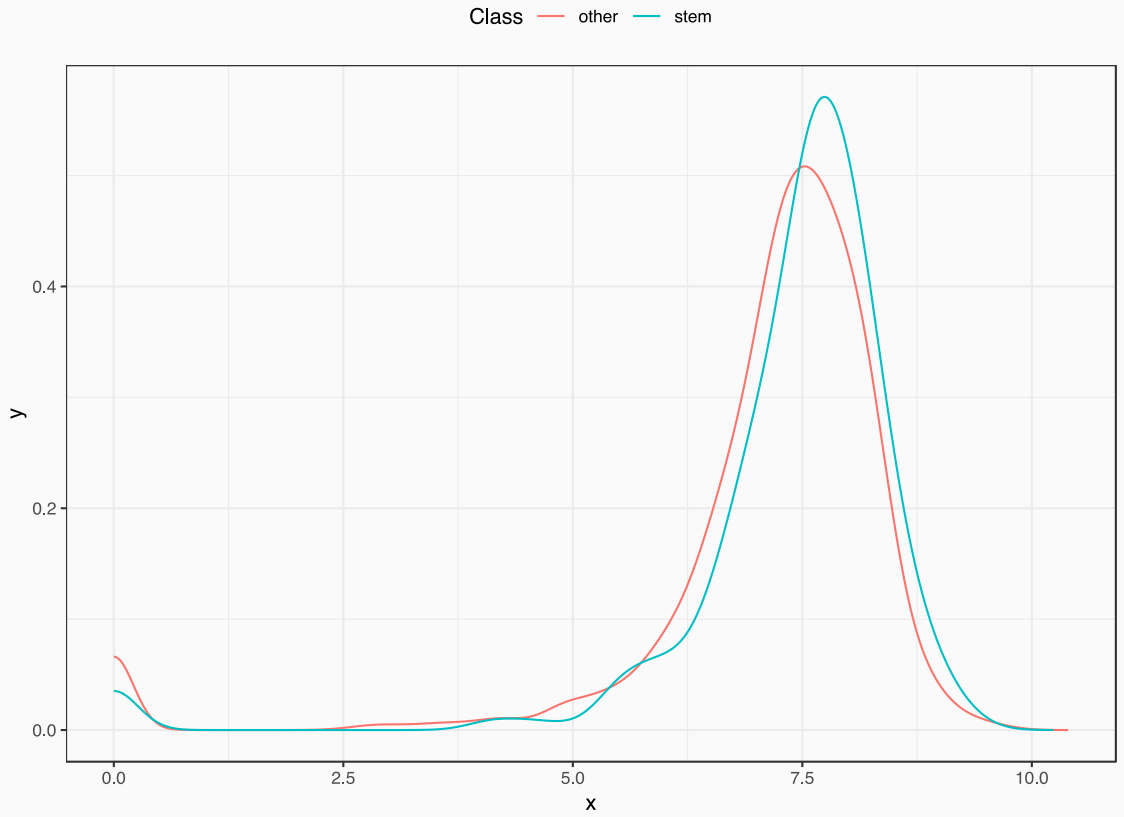
The probability contours assume multivariate normality with different assumptions.

Suppose the red dot is a new sample.



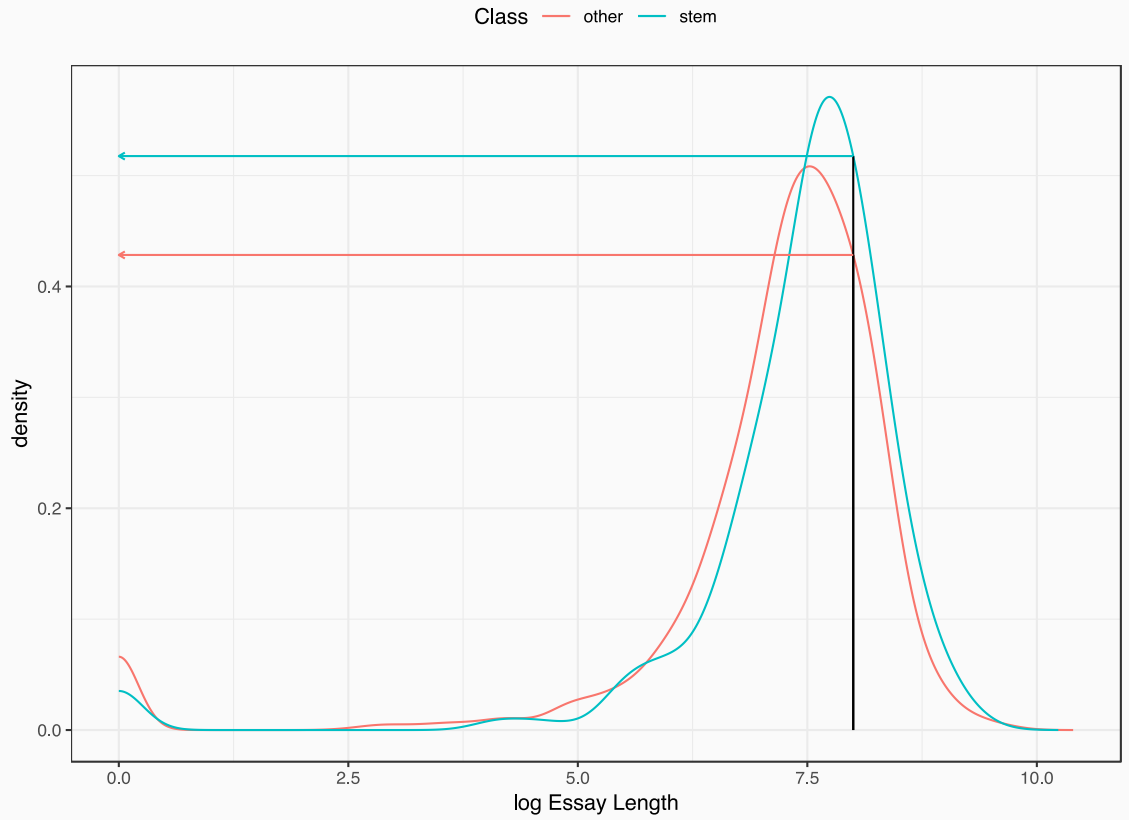
# Conditional Densities for Each Class

$Pr[Essay\ Length|Class]$



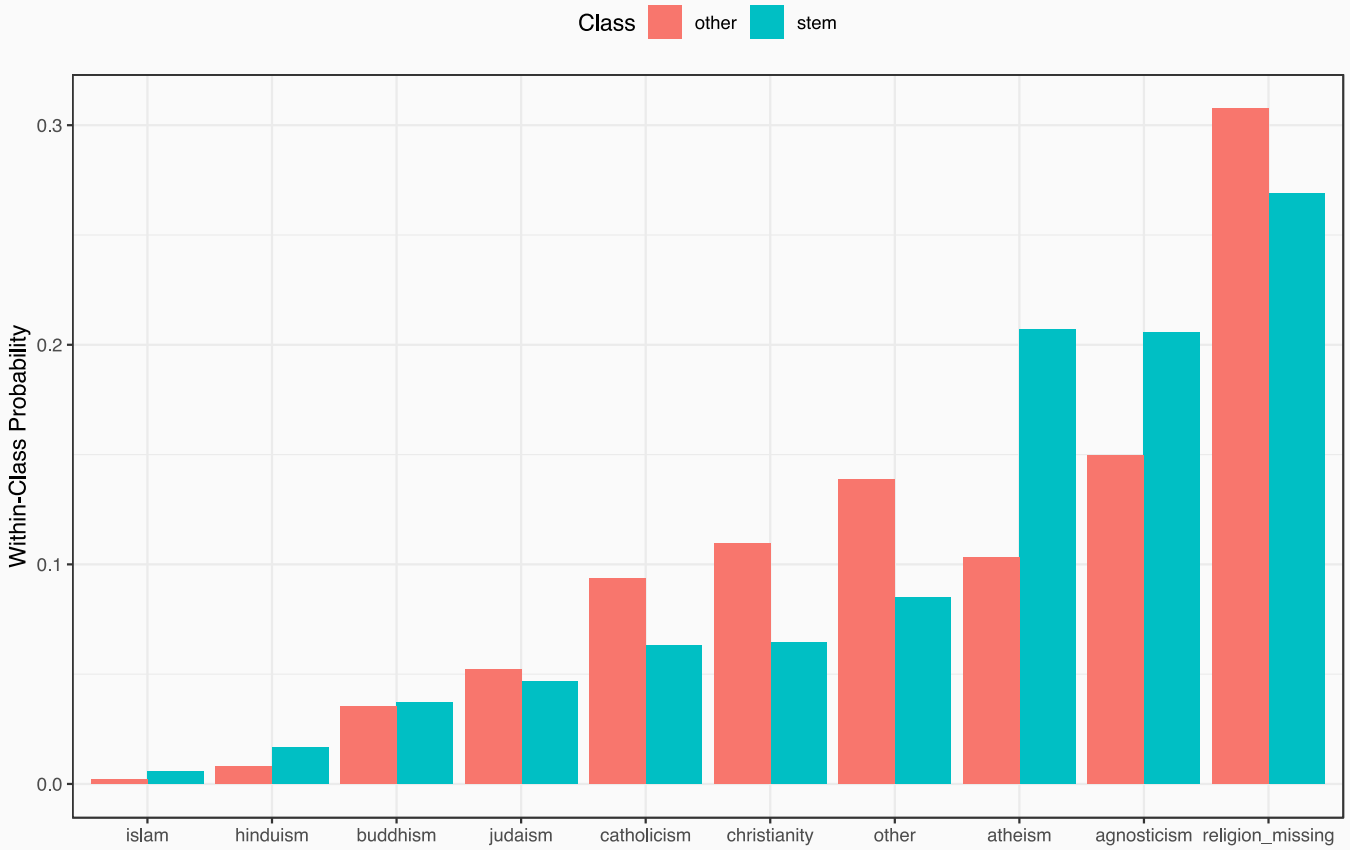
# Conditional Values for Numeric Predictors

$$Pr[Essay\ Length = 8|Class]$$



# Conditional Probabilities for Categorical Predictors

$Pr[Religion|Class]$



# Combining Predictor Scores with the Prior

For an Atheist who wrote $e^8$ words, their likelihood values were:

- $Pr[Essay\ Length = 8|STEM] \times Pr[Religion = Atheism|STEM]$ = 0.518 x 0.207 = 0.107
- $Pr[Essay\ Length = 8|Other] \times Pr[Religion = Atheism|Other]$ =0.428 x 0.103 = 0.044

However, when these are combined with the *prior probability* for each class, the *relative probabilities* show:

- $Pr[Predictors|STEM] \times Pr[STEM]$ = 0.107 x 0.182 = 0.02
- $Pr[Predictors|Other] \times Pr[Other]$ = 0.044 x 0.818 = 0.036

We don't need to compute the evidence; we can just normalize these values to add up to 1.

The results is that the *posterior probability* that this person is in a STEM field is 35.1%.

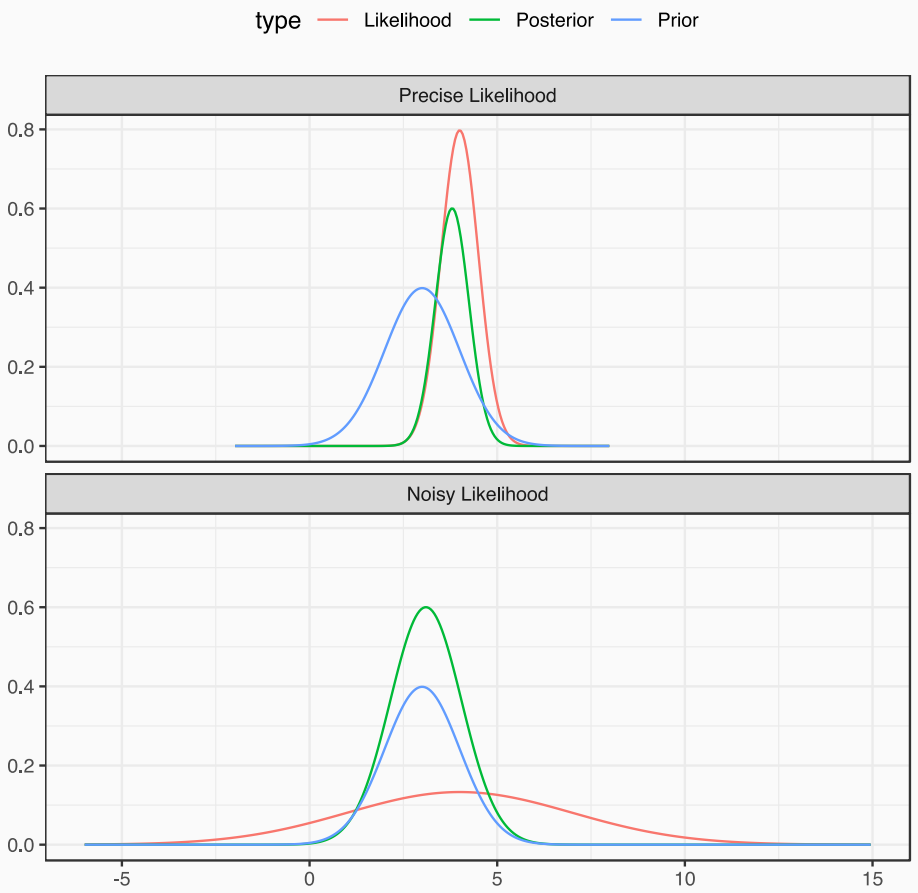
# "Shrinkage" Properties of Bayesian Models

When our observed data is of high quality (top panel), the likelihood is narrow and dominates the equation.

However, when our observed information is poor, the likelihood can be very wide and diffuse.

When this occurs, Bayes' rule relies more on our prior belief than on the data in-hand.

This is generally called "shrinkage".



# Pros and Cons

Good:

- This model can be very quickly trained (and theoretically in parallel).
- Once trained, the prediction is basically a look-up table (i.e. fast).
- Nonlinear class boundaries can be generated.

Bad:

- Linearly diagonal boundaries can be difficult.
- With many predictors, the class probabilities become poorly calibrated and U shaped with most values near zero or one. For example:

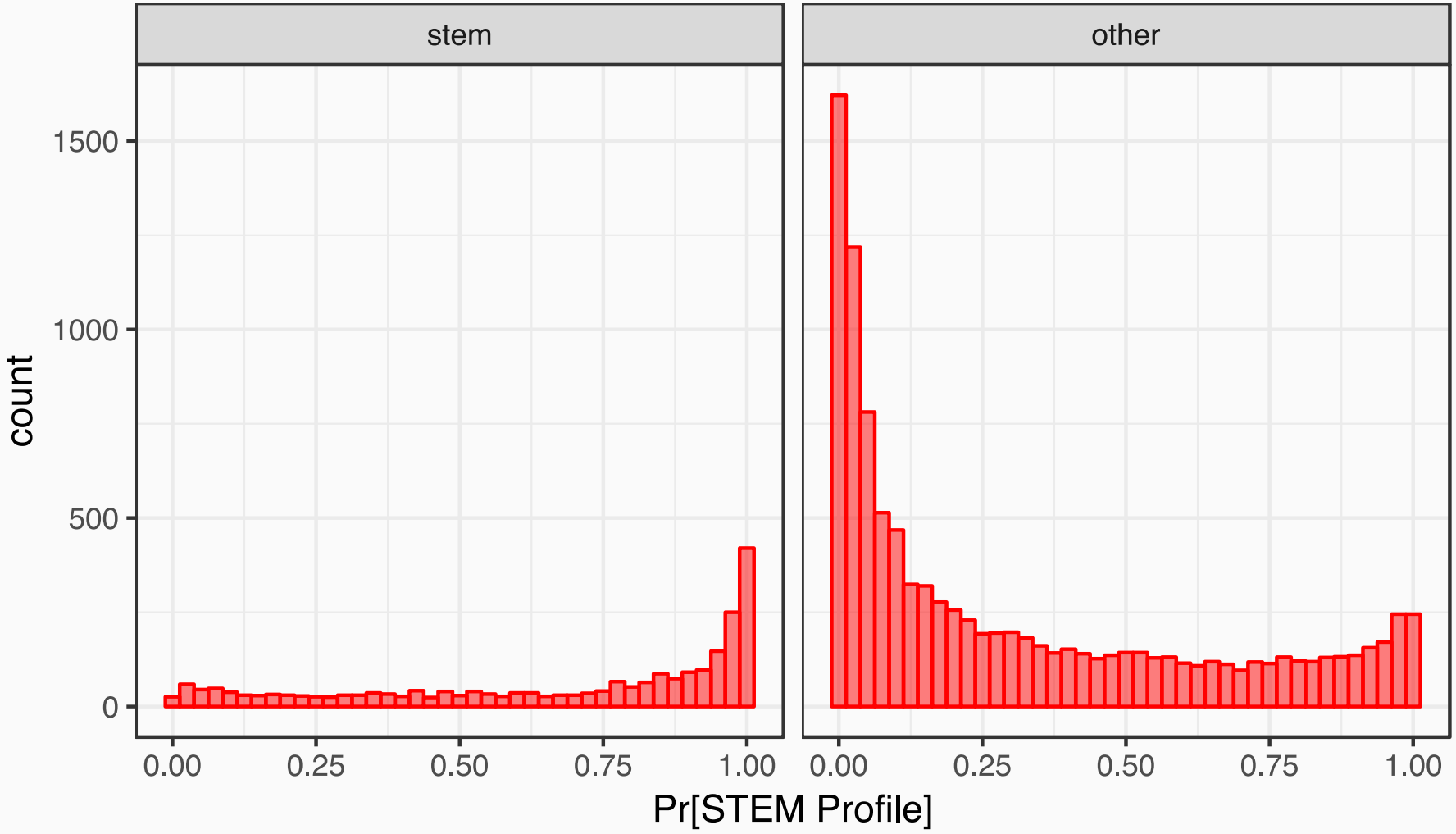
```
prod(rep(.1, 100))
```

```
## [1] 1e-100
```

```
prod(rep(.5, 100))
```

```
## [1] 7.89e-31
```

# U-Shaped Class Probability Distributions



# Naive Bayes Data Preparations

There is a step specifically designed for converting binary dummy variables into factors.

First, we identify them, then use quasiquotation (via `!!!`) to pass them to the step function.

```
library(recipes)
is_dummy <- vapply(okc_train, function(x) length(unique(x)) == 2 & is.numeric(x), logical(1))
dummies <- names(is_dummy)[is_dummy]
no_dummies <- recipe(Class ~ ., data = okc_train) %>%
    step_bin2factor(!!! dummies) %>%
    step_zv(all_predictors())
```

We can also modulate the smoothness of the density estimation for the continuous predictors using the `adjust` option to the `density` function.

```
smoothing_grid <- expand.grid(usekernel = TRUE, fL = 0, adjust = seq(0.5, 3.5, by = 0.5))
```

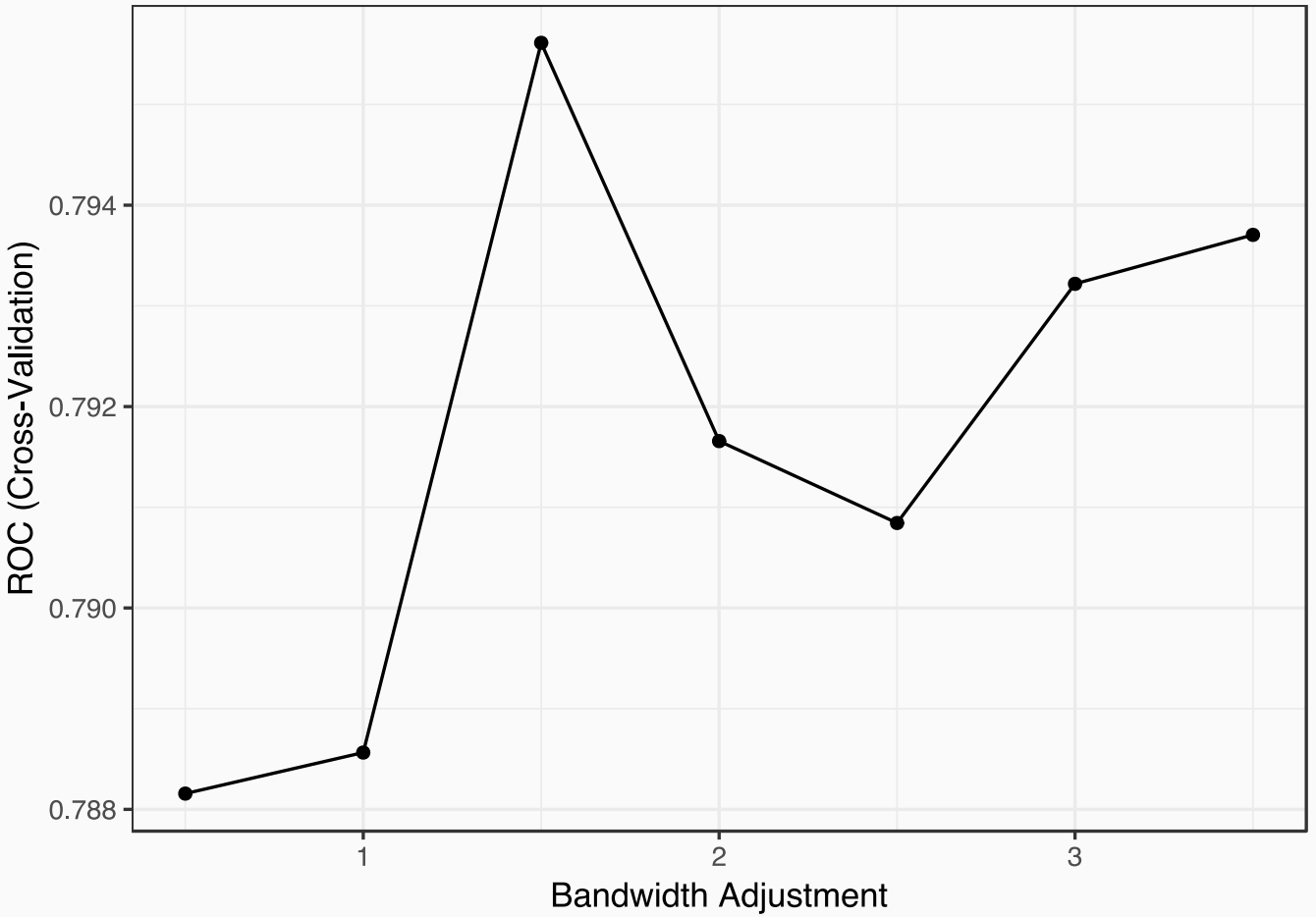
# Naive Bayes Training

```
set.seed(5515)
nb_mod <- train(
    no_dummies,
    data = okc_train,
    method = "nb",
    metric = "ROC",
    tuneGrid = smoothing_grid,
    trControl = ctrl
)
```

Some warnings: `## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with observation 1`

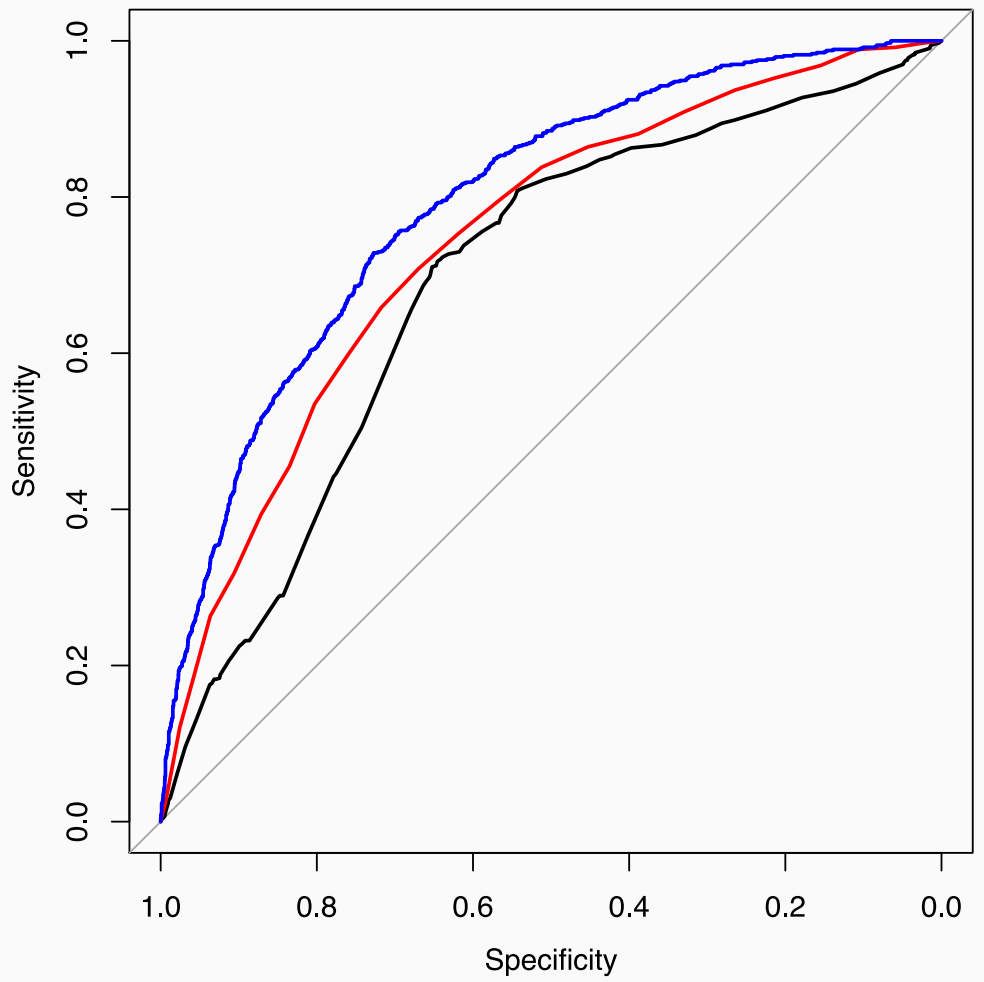
This is due to the poorly calibrated probabilities although the warning is a bit misleading. This issue does not generally affect performance and can be ignored.

# Naive Bayes Resampling Profile



# Three ROC Curves

```
plot_roc(cart_mod$pred)
plot_roc(cart_bag$pred, col = "red", add = TRUE)
plot_roc(nb_mod$pred, col = "blue", add = TRUE)
```



# Finalizing the Model and Predicting the Test Set

# Formally Comparing the Models


As before:

```
rs <- resamples(
  list(CART = cart_mod, Bagged = cart_bag, Bayes = nb_mod)
)
library(tidyposterior)
roc_mod <- perf_mod(rs, seed = 2560, iter = 5000)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
##
## Gradient evaluation took 0.000128 seconds
## 1000 transitions using 10 leapfrog steps per transition would take 1.28 seconds.
## Adjust your expectations accordingly!
##
##
## Iteration:    1 / 5000 [  0%]  (Warmup)
## Iteration:  500 / 5000 [ 10%]  (Warmup)
## Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Iteration: 2500 / 5000 [ 50%]  (Warmup)
```

# Formally Comparing the Models

```
roc_dist <- tidy(roc_mod)
summary(roc_dist)
```

```
## # A tibble: 3 x 4
##   model   mean lower upper
##   <chr>  <dbl> <dbl> <dbl>
## 1 Bagged 0.745 0.725 0.765
## 2 Bayes  0.795 0.775 0.815
## 3 CART   0.695 0.675 0.715
```

```
differences <-
    contrast_models(
        roc_mod,
        list_1 = c("Bagged", "Bayes"),
        list_2 = c("CART", "Bagged"),
        seed = 650
    )
```

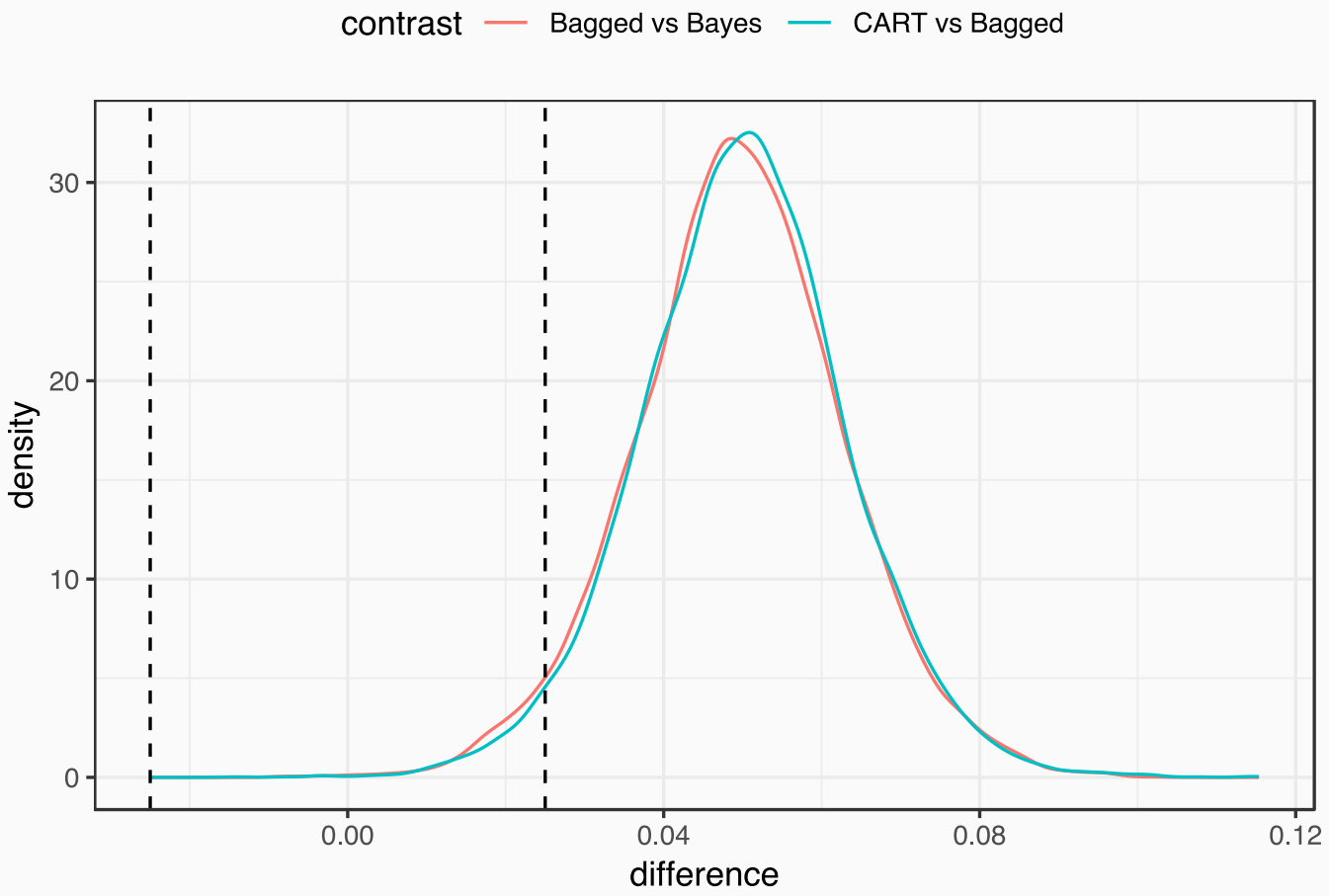
# Summarizing the Results

```
summary(differences, size = 0.025)
```

```
## # A tibble: 2 x 9
##   contrast        probability   mean  lower  upper  size pract_neg pract_equiv pract_pos
##   <chr>                 <dbl>  <dbl>  <dbl>  <dbl> <dbl>     <dbl>       <dbl>     <dbl>
## 1 Bagged vs CART        0.999 0.0502 0.0287 0.0717 0.025         0      0.0298     0.970
## 2 Bayes vs Bagged       0.999 0.0496 0.0275 0.0714 0.025         0      0.0361     0.964
```

```
differences %>%
    mutate(contrast = paste(model_2, "vs", model_1)) %>%
    ggplot(aes(x = difference, col = contrast)) +
    geom_line(stat = "density") +
    geom_vline(xintercept = c(-0.025, 0.025), lty = 2)
```

# The Posterior Distributions



# Test Set Results

```
test_res <- okc_test %>%
    dplyr::select(Class) %>%
    mutate(
        prob = predict(nb_mod, okc_test, type = "prob")[, "stem"],
        pred = predict(nb_mod, okc_test)
    )
roc_curve <- roc(test_res$Class, test_res$prob, levels = c("other", "stem"))
```

# Test Set ROC Curve

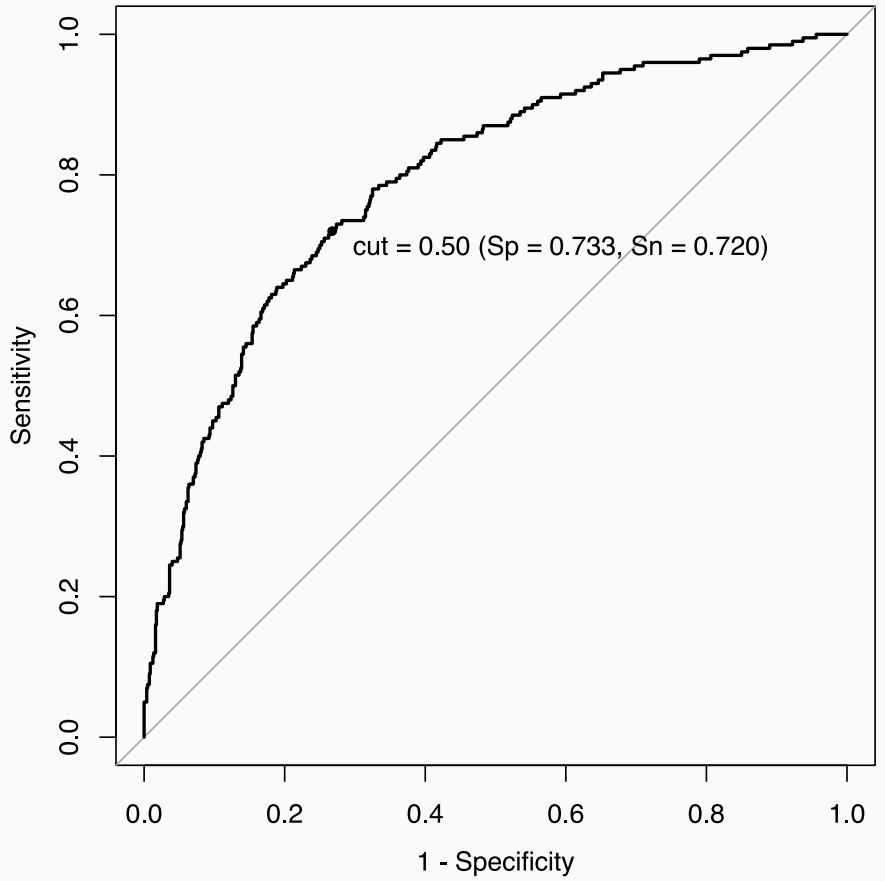
```
plot(
    roc_curve,
    print.thres = .5,
    print.thres.pattern = "cut = %.2f (Sp = %.3f, Sn =
    legacy.axes = TRUE
)
```

```
roc_curve
```

```
##
## Call:
## roc.default(response = test_res$Class, predictor = t
##
## Data: test_res$prob in 800 controls (test_res$Class
## Area under the curve: 0.792
```

```
getTrainPerf(nb_mod)
```

```
##   TrainROC TrainSens TrainSpec method
## 1    0.796     0.735      0.71     nb
```



# Test Set Class Probabilities

```
ggplot(test_res, aes(x = prob)) + geom_histogram(binwidth = .04) + facet_wrap(~Class)
```