Naive Bayes model

Brooke Anderson

January 20, 2016

Load required libraries.

```
library(dplyr) ## Data wrangling
library(e1071) ## Includes `naiveBayes` function
library(ggplot2)
library(stringr) ## Helps with regular expression
```

Read in the data. (I have it in the subdirectory `data` of the parent directory- `..`– of my current working directory.) Sometimes, some of the machine learning functions can be particularly fussy about class type for variables, so I used `mutate` to specify those for any problem ones. It doesn't hurt to make sure the all the factor variables end up with the same levels and saved order of levels after reading in the data.

```
train <- read.csv("../data/train.csv") %>%
  mutate(Survived = factor(Survived),
         Pclass = factor(Pclass),
         Name = as.character(Name),
         Sex = factor(Sex))
test <- read.csv("../data/test.csv") %>%
  mutate(Pclass = factor(Pclass),
         Name = as.character(Name),
         Sex = factor(Sex))

# Look through columns besides code about survival.
for(column in c("Pclass", "Sex",
               "Ticket", "Cabin", "Embarked")){
  cat(column, " train levels: ", head(levels(train[ , column]), 5), "\n")
  cat(column, " test levels: ", head(levels(test[ , column]), 5), "\n")
               }
```

```
## Pclass  train levels:  1 2 3
## Pclass  test levels:  1 2 3
## Sex  train levels:  female male
## Sex  test levels:  female male
## Ticket  train levels:  110152 110413 110465 110564 110813
## Ticket  test levels:  110469 110489 110813 111163 112051
## Cabin  train levels:   A10 A14 A16 A19
## Cabin  test levels:   A11 A18 A21 A29
## Embarked  train levels:   C Q S
## Embarked  test levels:  C Q S
```

Everything looks fine for `Pclass`, `Sex`, and `Embarked`, but it looks like there will not always be values of all levels of `Ticket` and `Cabin` in both datasets, so we may need to think carefully about thow we use these variables.

Null model

As a baseline, you could fit a null model that just takes the most common value of `Survival` and predicts that everyone will have that. Since more people died than survived in the training set– $P(Y) = 0.38$ in the training data, where Y is a 0 / 1 indicator of survival–here you would predict that everyone died.

This model will predict that `Survived` is always 0:

```
pred_train <- rep(0, length = nrow(train))
pred_test <- rep(0, length = nrow(test))
```

The accuracy of the `train` dataset is the percent of times, in this case, that `Survived` actually was 0:

```
mean(train$Survived == "0")
```

```
## [1] 0.6161616
```

To figure out the accuracy of the predictions from the null model for the `test` data, you need to save a csv with the predictions and submit to Kaggle:

```
out <- cbind(test$PassengerId, 0)
colnames(out) <- c("PassengerId", "Survived")
write.csv(out, file = "../predictions/null_model.csv", row.names = FALSE)
```

Now there is a `null_model.csv` file in my `predictions` directory. After submitting to Kaggle, I found the accuracy of this model based on the Public Leaderboard observations was 0.62679.

Single categorical predictor

First, try with a single, categorical predictor, `Pclass`.

```
# Fit the model-- note same conventions as `glm` formula call
nb_mod <- naiveBayes(Survived ~ Pclass, data = train)
```

Check out the model. It gives you the distribution of the outcome variable (`apriori`; $P(Y)$) and also the probability of each value of `Pclass` conditional on the level of `Survival`: $Pr(X_1|Y)$ where X_1 is the passenger's ticket class and Y is the passenger's survival status.

```
nb_mod$apriori ## Class distribution for `Survived`
```

```
## Y
##   0   1
## 549 342
```

```
nb_mod$tables ## Conditional probabilities given of `Pclass` given `Survived`
```

```
## $Pclass
##    Pclass
## Y           1         2         3
##   0 0.1457195 0.1766849 0.6775956
##   1 0.3976608 0.2543860 0.3479532
```

Note that each row of the `tables` element sums to 1:

```
apply(nb_mod$tables$Pclass, 1, sum)
```

```
## 0 1
## 1 1
```

Predict and assess accuracy within training class:

```
pred_train <- predict(nb_mod, train)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
mean(pred_train == "1") ## For a T / F, gives the proportion T's
```

```
## [1] 0.2424242
```

(If you're having some problems getting `predict` to work with `naiveBayes`, see here.)

This model predicts that everyone in First class survives and no one in Second or Third:

```
table(pred_train, train$Survived, train$Pclass)
```

```
## , ,  = 1
##
##
## pred_train   0   1
##          0   0   0
##          1  80 136
##
## , ,  = 2
##
##
## pred_train   0   1
##          0  97  87
##          1   0   0
##
## , ,  = 3
##
##
## pred_train   0   1
##          0 372 119
##          1   0   0
```

To determine the accuracy, calculate the percent of time that the predicted value equals the true value for `Survived`:

```
sum(pred_train == train$Survived) / length(pred_train)
```

```
## [1] 0.6790123
```

You can also predict for the `test` data to generate a prediction to submit to Kaggle.

```
pred_test <- predict(nb_mod, test)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
head(pred_test)
```

```
## [1] 0 0 0 0 0 0
## Levels: 0 1
```

```
table(pred_test)
```

```
## pred_test
##   0   1
## 311 107
```

To submit, you need to join with the passenger IDs from `test` and write to a csv. I'm saving in a subdirectory of my parent directory called `predictions`. Also, need to convert the factor of `pred_test` results to character then numeric to have it saved properly as 0 / 1.

```
out <- cbind(test$PassengerId, as.numeric(as.character(pred_test)))
colnames(out) <- c("PassengerId", "Survived")
head(out, 3)
```

```
##      PassengerId Survived
## [1,]         892        0
## [2,]         893        0
## [3,]         894        0
```

```
write.csv(out, file = "../predictions/nb_pclass.csv", row.names = FALSE)
```

This file is now ready to upload to Kaggle. I did, and the accuracy was 0.65550, not too much lower than the testing set accuracy of 0.6790123 for this model.

Note: to find the help file for predicting with `naiveBayes`, use:

```
?predict.naiveBayes
```

Single continuous predictor

You can also try Naive Bayes with a continuous varible, like `Age`.

```
nb_mod <- naiveBayes(Survived ~ Age, data = train)
```

Now, the "Conditional probabilities" part of the model output gives, for each class of `Survived`, the mean (first column) and standard deviations (second column) of the independent variable put into the model (`Age`).

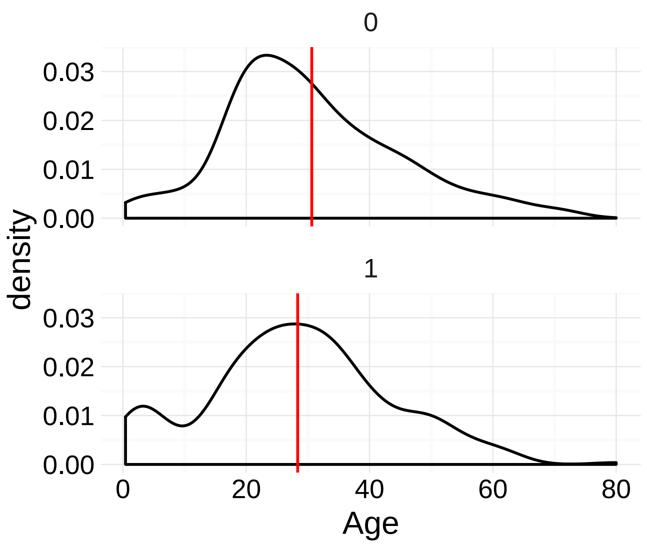
```
nb_mod
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.6161616 0.3838384
##
## Conditional probabilities:
##    Age
## Y       [,1]     [,2]
##   0 30.62618 14.17211
##   1 28.34369 14.95095
```

To visualize, here are density plots for `Age` separated by `Survived`, with red lines showing the mean values given by `nb_mod`:

```
vlines <- data.frame(Age = nb_mod$table$Age[ , 1],
                     Survived = factor(rownames(nb_mod$table$Age)))

ggplot(train, aes(Age)) +
  geom_density() +
  facet_wrap(~ Survived, ncol = 1) +
  geom_vline(data = vlines, aes(xintercept = Age), color = "red") +
  theme_minimal()
```



Here's a comparison of the model output with the means of age calculated by survival, and you can see they're identical.

```
nb_mod$table$Age
```

```
##    Age
## Y       [,1]     [,2]
##   0 30.62618 14.17211
##   1 28.34369 14.95095
```

```
by(train$Age, train$Survived, mean, na.rm = TRUE)
```

```
## train$Survived: 0
## [1] 30.62618
## ---------------------------------------------------------
## train$Survived: 1
## [1] 28.34369
```

Again, you can predict with this model. However, now you have the problem that `Age` is missing for some of your observations. We can talk about strategies for dealing with that– I'm going to use a very simple approach and replace any of those with the most common value of `Survival` in the dataset, "0".

```
pred_train <- predict(nb_mod, train)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
pred_train[is.na(pred_train)] <- factor(0)
table(pred_train)
```

```
## pred_train
##   0   1
## 890   1
```

The accuracy in the training data is:

```
mean(train$Survived == pred_train)
```

```
## [1] 0.617284
```

Only one person is predicted to survive under this model. This happens to be the youngest person on board the ship among the training dataset.

```
train[pred_train == "1", ]
```

```
##     PassengerId Survived Pclass                            Name  Sex  Age
## 804         804        1      3 Thomas, Master. Assad Alexander male 0.42
##     SibSp Parch Ticket   Fare Cabin Embarked
## 804     0     1   2625 8.5167              C
```

```
min(train$Age, na.rm = TRUE)
```

```
## [1] 0.42
```

Fit the predictive model to the testing dataset and then try it on Kaggle:

```
pred_test <- predict(nb_mod, test)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
pred_test[is.na(pred_test)] <- factor(0)
table(pred_test)
```

```
## pred_test
##   0   1
## 416   2
```

```
out <- cbind(test$PassengerId, as.numeric(as.character(pred_test)))
colnames(out) <- c("PassengerId", "Survived")
head(out, 3)
```

```
##      PassengerId Survived
## [1,]         892        0
## [2,]         893        0
## [3,]         894        0
```

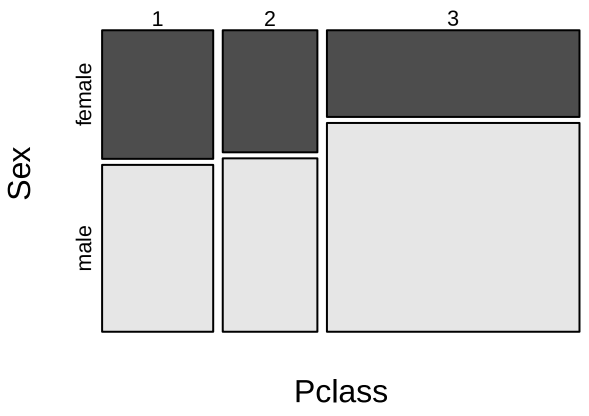
```
write.csv(out, file = "../predictions/nb_age.csv", row.names = FALSE)
```

The accuracy of this model on the Leaderboard was 0.617284, which is worse than that of the null model.

Multiple predictors

You can also fit this with multiple predictors. For example, maybe fit a model with `Pclass`, `Sex`, and `Embarked`. Just to keep in mind, although the Naive Bayes model assumes they're all independent of each other, here that's not the case. For example, a much higher percentage of 1st and 2nd class were female then 3rd class:

```
mosaicplot(~ Pclass + Sex, data = train, color = TRUE,
           main = "")
```



Something to think about: what are the implications of using Naive Bayes when you violate these assumptions of independence between predictors?

Next, fit the model:

```r
(nb_mod <- naiveBayes(Survived ~ Pclass + Sex + Embarked, data = train))
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.6161616 0.3838384
##
## Conditional probabilities:
##    Pclass
## Y           1         2         3
##   0 0.1457195 0.1766849 0.6775956
##   1 0.3976608 0.2543860 0.3479532
##
##    Sex
## Y      female      male
##   0 0.1475410 0.8524590
##   1 0.6812865 0.3187135
##
##    Embarked
## Y                        C           Q           S
##   0 0.000000000 0.136612022 0.085610200 0.777777778
##   1 0.005847953 0.271929825 0.087719298 0.634502924
```

Now you get separate conditional probabilities for each predictor. Evidently, some values of `Embarked` aren't listed as `NA`s but rather as `""`. They get their own probabilities in `nb_mod`.

```r
sum(is.na(train$Embarked))
```

```
## [1] 0
```

```r
sum(train$Embarked == "")
```

```
## [1] 2
```

```r
train[train$Embarked == "", ]
```

```
##     PassengerId Survived Pclass                                      Name
## 62           62        1      1                       Icard, Miss. Amelie
## 830         830        1      1 Stone, Mrs. George Nelson (Martha Evelyn)
##        Sex Age SibSp Parch Ticket Fare Cabin Embarked
## 62  female  38     0     0 113572   80   B28
## 830 female  62     0     0 113572   80   B28
```

I think that how we treat these shouldn't affect the Kaggle score, because none of the test observations have `Embarked` equal to `""`.

Predicting this to the training data. Again, if something is missing, I'll replace with "0" (better solutions?):

```
pred_train <- predict(nb_mod, train)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
pred_train[is.na(pred_train)] <- factor(0)
table(pred_train)
```

```
## pred_train
##   0   1
## 535 356
```

```
mean(train$Survived == pred_train)
```

```
## [1] 0.7777778
```

On the training data, this model has an accuracy of 0.7777778, the highest so far by a bit.

To see how it does on the testing data:

```
pred_test <- predict(nb_mod, test)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```
pred_test[is.na(pred_test)] <- factor(0)
table(pred_test)
```

```
## pred_test
##   0   1
## 231 187
```

```
out <- cbind(test$PassengerId, as.numeric(as.character(pred_test)))
colnames(out) <- c("PassengerId", "Survived")
head(out, 3)
```

```
##      PassengerId Survived
## [1,]         892        0
## [2,]         893        1
## [3,]         894        0
```

```
write.csv(out, file = "../predictions/nb_class_sex_embark.csv",
          row.names = FALSE)
```

This model had an accuracy of 0.73684 on the Kaggle Leaderboard data, the best so far. Also, this model had the biggest reduction in accuracy going from the training to the testing data.

Kitchen sink model

Last, I tried chucking in everything I could think of, incuding some "engineered" features. First, some code to add some features:

```r
# Add honorific (Mr., Mrs., Dr., etc.)
honorific <- str_extract(train$Name, ",\\ .+?\\.") # Uses `stringr` package
honorific <- gsub("[\\,\\.\\ ]", "", honorific)
head(honorific, 3)
```

```
## [1] "Mr"   "Mrs"  "Miss"
```

```r
train <- cbind(train, honorific)

# Add if age is missing. Note-- this might cover some data leakage--
# possible that it was easier to find out ages of survivors than victims
train <- mutate(train,
                missing = factor(is.na(Age), levels = c(TRUE, FALSE),
                                 labels = c("Age missing",
                                            "Age available")))

# Mark if they were using a ticket that covered more than 5 people
common_tickets <- names(table(train$Ticket)[table(train$Ticket) > 5])
head(common_tickets)
```

```
## [1] "1601"     "3101295"  "347082"   "347088"   "CA 2144"  "CA. 2343"
```

```r
train$common_ticket <- factor("0", levels = c("0", "1"))
train$common_ticket[train$Ticket %in% common_tickets] <- "1"
table(train$common_ticket)
```

```
##
##   0   1
## 852  39
```

Here are all the variables currently in `train`:

```r
str(train)
```

```
## 'data.frame':    891 obs. of  15 variables:
##  $ PassengerId  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Survived     : Factor w/ 2 levels "0","1": 1 2 2 2 1 1 1 1 2 2 ...
##  $ Pclass       : Factor w/ 3 levels "1","2","3": 3 1 3 1 3 3 1 3 3 2 ...
##  $ Name         : chr  "Braund, Mr. Owen Harris" "Cumings, Mrs. John Bradley (Florence Briggs Thayer)
##  $ Sex          : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
##  $ Age          : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ SibSp        : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ Parch        : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ Ticket       : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 .
##  $ Fare         : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ Cabin        : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
##  $ Embarked     : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
##  $ honorific    : Factor w/ 17 levels "Capt","Col","Don",..: 12 13 9 13 12 12 12 8 13 13 ...
##  $ missing      : Factor w/ 2 levels "Age missing",..: 2 2 2 2 2 1 2 2 2 2 ...
##  $ common_ticket: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

So I'll fit the model:

```r
kitchen_sink <- naiveBayes(Survived ~ Pclass + Sex + SibSp +
                             Parch + Fare + Embarked + honorific +
                             missing , data = train)
kitchen_sink
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.6161616 0.3838384
##
## Conditional probabilities:
##    Pclass
## Y           1         2         3
##   0 0.1457195 0.1766849 0.6775956
##   1 0.3976608 0.2543860 0.3479532
##
##    Sex
## Y      female      male
##   0 0.1475410 0.8524590
##   1 0.6812865 0.3187135
##
##    SibSp
## Y        [,1]      [,2]
##   0 0.5537341 1.2883991
##   1 0.4736842 0.7086875
##
##    Parch
## Y        [,1]     [,2]
##   0 0.3296903 0.823166
##   1 0.4649123 0.771712
##
##    Fare
## Y       [,1]     [,2]
##   0 22.11789 31.38821
##   1 48.39541 66.59700
##
##    Embarked
## Y                       C           Q           S
##   0 0.000000000 0.136612022 0.085610200 0.777777778
##   1 0.005847953 0.271929825 0.087719298 0.634502924
##
##    honorific
## Y          Capt         Col         Don          Dr    Jonkheer
##   0 0.001821494 0.001821494 0.001821494 0.007285974 0.001821494
##   1 0.000000000 0.002923977 0.000000000 0.008771930 0.000000000
##    honorific
## Y           Lady       Major      Master        Miss        Mlle
##   0 0.000000000 0.001821494 0.030965392 0.100182149 0.000000000
##   1 0.002923977 0.002923977 0.067251462 0.371345029 0.005847953
##    honorific
## Y            Mme          Mr         Mrs          Ms         Rev
##   0 0.000000000 0.794171220 0.047358834 0.000000000 0.010928962
##   1 0.002923977 0.236842105 0.289473684 0.002923977 0.000000000
##    honorific
## Y            Sir theCountess
##   0 0.000000000 0.000000000
##   1 0.002923977 0.002923977
##
##    missing
## Y   Age missing Age available
##   0   0.2276867     0.7723133
##   1   0.1520468     0.8479532
```

Predict on training data:

```r
pred_train <- predict(kitchen_sink, train)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```r
pred_train[is.na(pred_train)] <- factor(0)
table(pred_train)
```

```
## pred_train
##   0   1
## 567 324
```

```r
mean(train$Survived == pred_train)
```

```
## [1] 0.8024691
```

This accuracy is even better than for the last model, at least for the training dataset.

To try on the `test` data, I need to add the same new features to that, as well (I'll now use `test` and `train` to figure out the common tickets, though):

```r
# Add honorific (Mr., Mrs., Dr., etc.)
honorific <- str_extract(test$Name, ",\\ .+?\\.") # Uses `stringr` package
honorific <- gsub("[\\,\\.\\ ]", "", honorific)
head(honorific, 3)
```

```
## [1] "Mr"  "Mrs" "Mr"
```

```r
test <- cbind(test, honorific)

# Add if age is missing. Note-- this might cover some data leakage--
# possible that it was easier to find out ages of survivors than victims
test <- mutate(test,
                  missing = factor(is.na(Age), levels = c(TRUE, FALSE),
                                   labels = c("Age missing",
                                              "Age available")))

# Mark if they were using a ticket that covered more than 5 people
all_tickets <- c(as.character(train$Ticket), as.character(test$Ticket))
common_tickets <- names(table(all_tickets)[table(all_tickets) > 5])
head(common_tickets)
```

```
## [1] "113781"  "1601"    "19950"   "3101295" "347077"  "347082"
```

```r
test$common_ticket <- factor("0", levels = c("0", "1"))
test$common_ticket[test$Ticket %in% common_tickets] <- "1"
table(test$common_ticket)
```

```
##
##   0   1
## 395  23
```

```r
pred_test <- predict(kitchen_sink, test)
```

```
## Warning in data.matrix(newdata): NAs introduced by coercion
```

```r
pred_test[is.na(pred_test)] <- factor(0)
table(pred_test)
```

```
## pred_test
##   0   1
## 227 191
```

```r
out <- cbind(test$PassengerId, as.numeric(as.character(pred_test)))
colnames(out) <- c("PassengerId", "Survived")
head(out, 3)
```

```
##      PassengerId Survived
## [1,]         892        0
## [2,]         893        1
## [3,]         894        1
```

```r
write.csv(out, file = "../predictions/nb_kitchen_sink.csv",
          row.names = FALSE)
```

This model had an accuracy of 0.67464 on the Leaderboard test data, so it was not an improvement over the Naive Bayes with just three predictors.

Things to think more about

- Laplace smoothing
- Better to pool predictors with lots of categories into just a few categories?

- Predicting observations with 1+ feature values missing
- Way to automate finding the best predictors to include? Overfitting repercussions of trying to automate that?
- Can violate the assumption of independence between predictors but still be a good model?
- How does Naive Bayes work if you have a continuous predictor that is not normally distributed? Something really skewed like `Fare` here. Can you set up a Naive Bayes model to use a different distribution for continuous predictors?