Naive Bayes Classifiers

Shane T. Mueller shanem@mtu.edu

2021-02-28

Naive Bayes Classifiers

A naive bayes classifier uses Bayes rule to combine information about a set of predictors. Although many Bayesian approaches can be quite complex and computationally-intensive, Naive Bayes classifiers are simple such that they can often be implemented without any special library. They are also easy to use even when you have hundreds or thousands of features, such as when trying to classify text (e.g., each word is a feature/predictor), for classification problems like sentiment analysis or spam detection. Let's start with an example application for the iPhone data set:

```
dat <- read.csv("data_study1.csv")
dat$Smartphone <- factor(dat$Smartphone)
library(klaR)
nb <- NaiveBayes(Smartphone ~ ., data = dat)
library(DAAG)
summary(nb)
```

```
          Length Class      Mode
apriori    2     table      numeric
tables    12     -none-     list
levels     2     -none-     character
call       3     -none-     call
x         12     data.frame list
usekernel  1     -none-     logical
varnames  12     -none-     character
```

```
nb$apriori
```

```
grouping
  Android    iPhone
0.4139887 0.5860113
```

```
nb$tables
```

```
$Gender
         var
grouping     female      male
  Android 0.5753425 0.4246575
  iPhone  0.7516129 0.2483871

$Age
            [,1]     [,2]
Android 31.42466 13.52190
iPhone  26.84839 12.19792
```

```
$Honesty.Humility
            [,1]      [,2]
Android 3.598174 0.5971617
iPhone  3.351935 0.6269406

$Emotionality
            [,1]      [,2]
Android 3.231963 0.7187408
iPhone  3.455806 0.6648467

$Extraversion
            [,1]      [,2]
Android 3.196347 0.6842832
iPhone  3.288710 0.6360540

$Agreeableness
            [,1]      [,2]
Android 3.175799 0.5900619
iPhone  3.123226 0.6390040

$Conscientiousness
            [,1]      [,2]
Android 3.544292 0.5910664
iPhone  3.573871 0.6173289

$Openness
            [,1]      [,2]
Android 3.542922 0.6218371
iPhone  3.415806 0.6224821

$Avoidance.Similarity
            [,1]      [,2]
Android 2.545205 0.8413687
iPhone  2.306452 0.8150825

$Phone.as.status.object
            [,1]      [,2]
Android 2.086758 0.5411150
iPhone  2.375484 0.6006299

$Social.Economic.Status
            [,1]     [,2]
Android 5.949772 1.586132
iPhone  6.012903 1.485582

$Time.owned.current.phone
            [,1]      [,2]
Android 12.72146 11.497509
iPhone  11.68065  8.748811
```

To understand how it works, let's look at the output. First, we have the apriori values–this determines the overall base rate of the outcomes. Then, we have the mean and standard deviation of normal distribution describing of each feature for each group. To confirm, look at the values for age:

```
means <- tapply(as.numeric(dat$Age), list(dat$Smartphone), mean)
sds <- tapply(as.numeric(dat$Age), list(dat$Smartphone), sd)
means
```

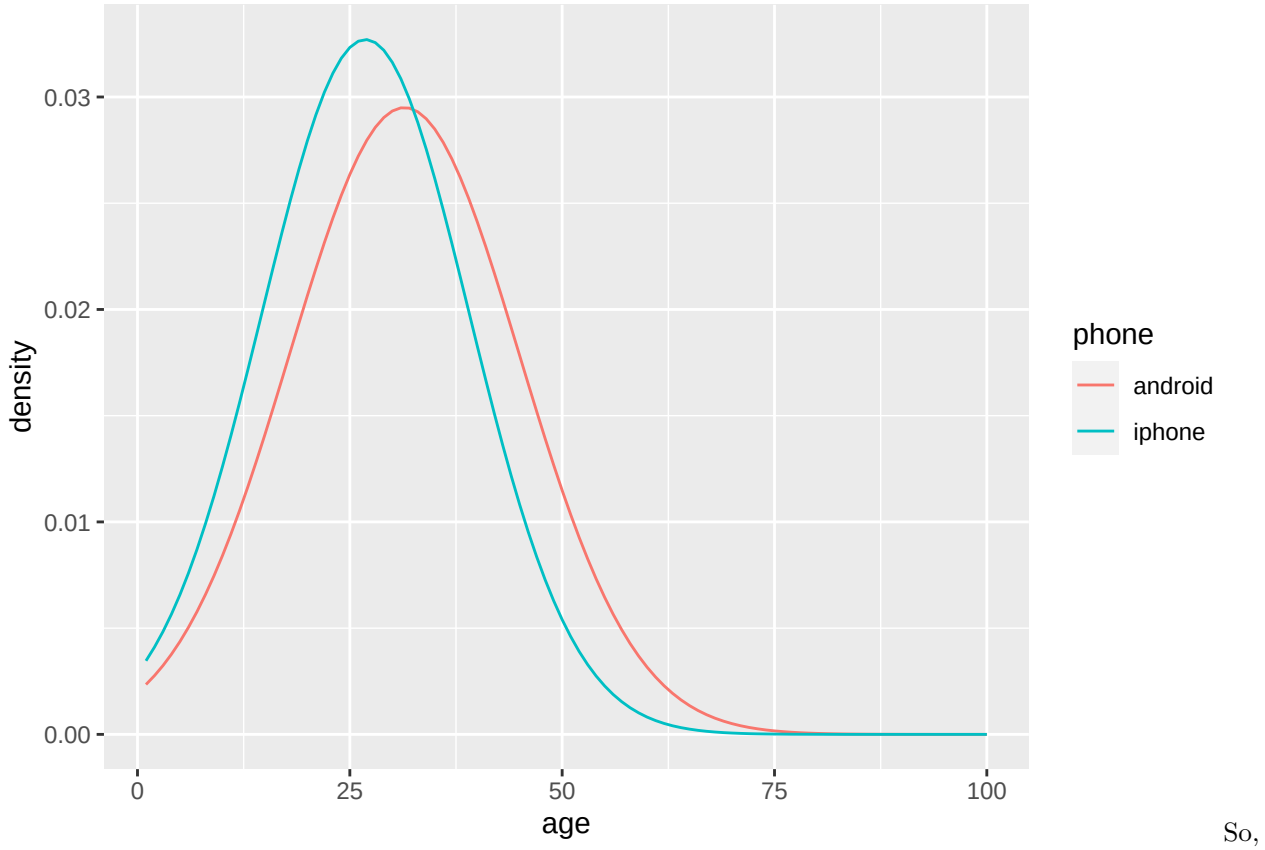
```
 Android   iPhone
31.42466 26.84839
```

```
sds
```

```
 Android   iPhone
13.52190 12.19792
```

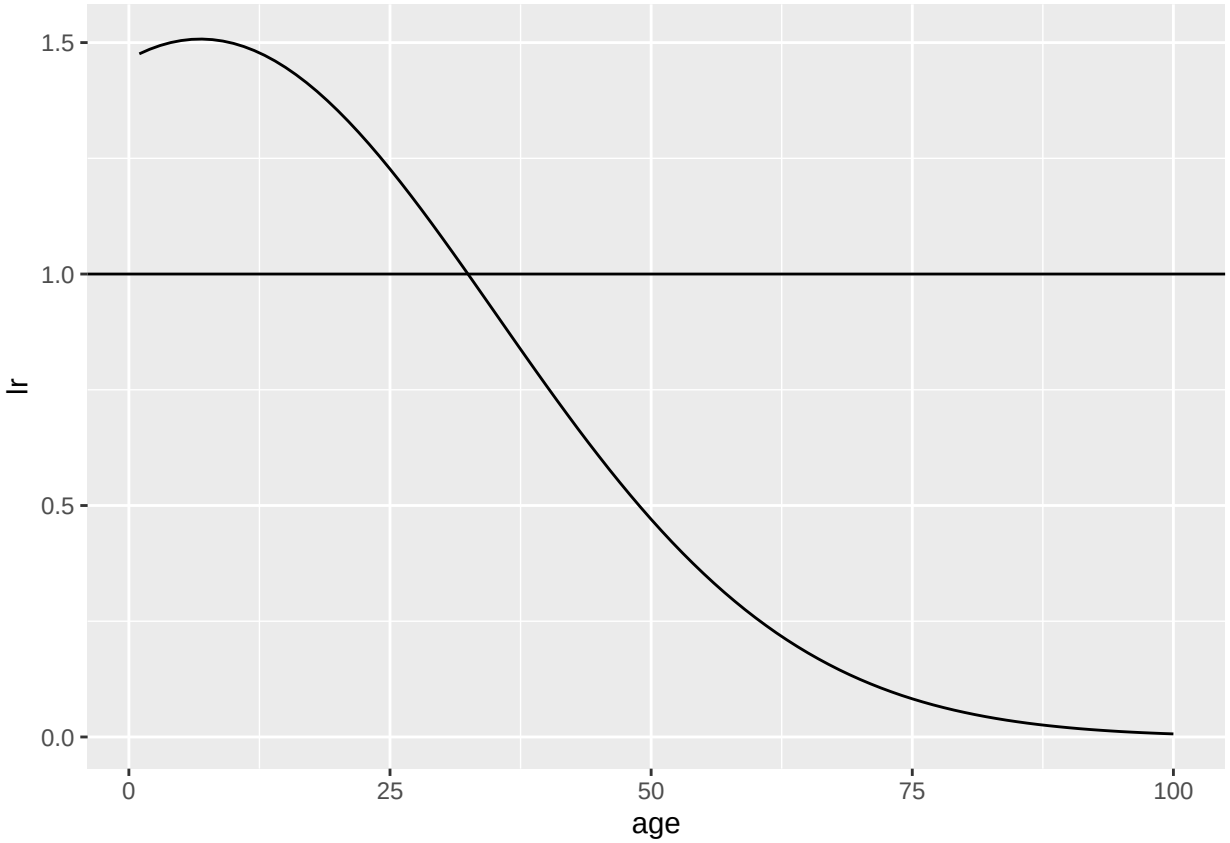
Naive Bayes works very similarly to qda. QDA attempts to find a decision point consistent with the most likely outcome that combines all features optimally. In contrast, NB combines all features equally–for each feature computing the likelihood of each option, and combining those one at a time. For the Age predictor, the two densities look like this:

```
library(ggplot2)
phone <- rep(c("iphone", "android"), each = 100)
density <- c(dnorm(1:100, mean = means[2], sd = sds[2]), dnorm(1:100, mean = means[1],
    sd = sds[1]))
df <- data.frame(age = c(1:100, 1:100), phone, density)

ggplot(df, aes(x = age, y = density, colour = phone)) + geom_line()
```



So, in a way this is very similar to LDA/QDA, because it uses a model-based approach, assuming a normal distribution, and permits different means and standard deviation for each group. Notice how for different ages, there is a higher likelihood for one group or the other. We can compute a ratio easily:

```
df <- data.frame(age = c(1:100), lr = (dnorm(1:100, mean = means[2], sd = sds[2])/dnorm(1:100,
    mean = means[1], sd = sds[1])))


ggplot(df, aes(x = age, y = lr)) + geom_line() + geom_hline(yintercept = 1)
```



This shows the likelihood of ratio in favor of being an iPhone user across age. Notice that this is highly model-based–if your assumed model is wrong, like it if it is not normally-distributed, the ratio can be wrong. This is especially true out in the tails of the distribution, where we probably have the least amount of data, but where the observations can have the greatest impact. If this were a decision tree, we'd use the point the likelihood ratio crosses 1.0 as the place to put the cut-off for that dimension. If we were doing LDA or QDA, we'd combine all axes and find a cutoff along a single vector. NB combines the likelihood ratio curves for each dimension using Bayes Rule, which also permits combining prior probability (base rate). So, like LDA, it combines all the dimensions, but like a decision tree, it computes a fit on dimension.

In contrast to QDA, the weighing of features technically differs in Naive Bayes, but they differ in details a little. One important aspect is that QDA and LDA essentially use a regression equation that incorporates base rate in an intercept value. Here, each features is ignorant of the bias. If you have a missing value, you can ignore the contribution of that feature, as the best guess is 1.0–which has no impact on the classification. For LDA and QDA, it is not as easy to deal with missing values. This is one benefit of Bayes Factor–it is useful for data that are often likely to be missing. This is one of the reasons why it is frequently used for classifying text data, which may have thousands of features (i.e., words) but most of them are missing for any given document.

We can use this for prediction:

```
Overall accuracy = 0.667
```

```
Confusion matrix
         Predicted (cv)
Actual    Android iPhone
  Android   0.543  0.457
  iPhone    0.245  0.755
```

This performs about as well as any of the other models. There is no simple way to do cross-validation with this implementation–you would have to do this by hand if you wanted to implement cross-validation. But also note that if a feature is non-predictive, it will essentially have very little impact on the outcome.

Using alternative kernel

The model assumes a normal/gaussian distribution, but Naive Bayes will permit us to estimate the of each feature empirically, which happens when we specify usekernel=T.

```
nb2 <- NaiveBayes(Smartphone ~ ., usekernel = T, data = dat)
nb2$tables
```

```
$Gender
         var
grouping     female      male
  Android 0.5753425 0.4246575
  iPhone  0.7516129 0.2483871

$Age
$Age$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 4.142

       x                y
 Min.   : 3.575   Min.   :5.350e-06
 1st Qu.:23.537   1st Qu.:2.524e-03
 Median :43.500   Median :8.884e-03
 Mean   :43.500   Mean   :1.251e-02
 3rd Qu.:63.463   3rd Qu.:1.954e-02
 Max.   :83.425   Max.   :4.128e-02

$Age$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 2.292

       x                y
 Min.   : 9.123   Min.   :1.259e-05
 1st Qu.:27.062   1st Qu.:8.960e-04
 Median :45.000   Median :7.547e-03
 Mean   :45.000   Mean   :1.392e-02
 3rd Qu.:62.938   3rd Qu.:1.111e-02
 Max.   :80.877   Max.   :9.380e-02
```

```
$Honesty.Humility
$Honesty.Humility$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.1829

       x               y
 Min.   :1.151   Min.   :0.0001146
 1st Qu.:2.226   1st Qu.:0.0197670
 Median :3.300   Median :0.1441117
 Mean   :3.300   Mean   :0.2324787
 3rd Qu.:4.374   3rd Qu.:0.4797145
 Max.   :5.449   Max.   :0.5965253

$Honesty.Humility$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1791

       x               y
 Min.   :1.163   Min.   :0.0000963
 1st Qu.:2.206   1st Qu.:0.0382611
 Median :3.250   Median :0.1789519
 Mean   :3.250   Mean   :0.2392920
 3rd Qu.:4.294   3rd Qu.:0.4296896
 Max.   :5.337   Max.   :0.6257473


$Emotionality
$Emotionality$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.2202

       x                y
 Min.   :0.4395   Min.   :0.0000934
 1st Qu.:1.7448   1st Qu.:0.0166069
 Median :3.0500   Median :0.1157574
 Mean   :3.0500   Mean   :0.1913477
 3rd Qu.:4.3552   3rd Qu.:0.3759779
 Max.   :5.6605   Max.   :0.5323054

$Emotionality$iPhone

Call:
    density.default(x = xx)
```

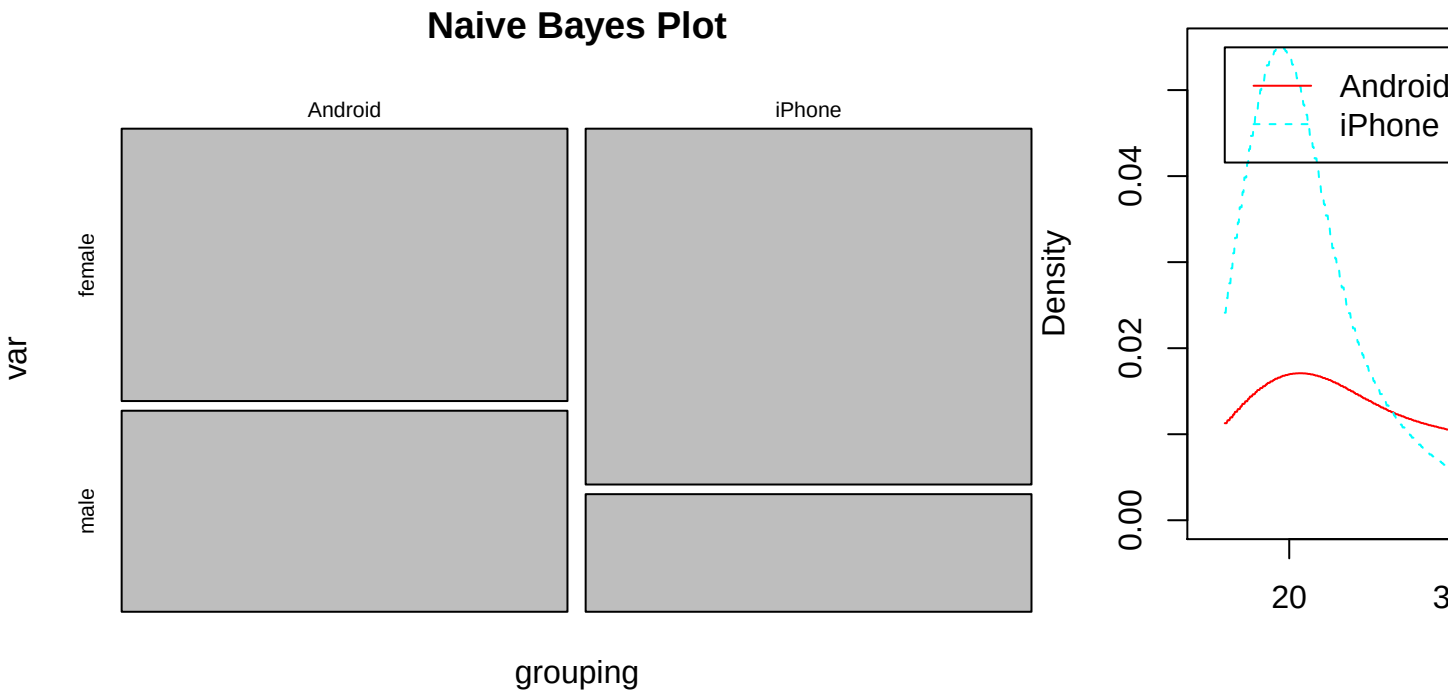
```
Data: xx (310 obs.);    Bandwidth 'bw' = 0.19

       x                y
 Min.   :0.9301   Min.   :0.0000773
 1st Qu.:2.0900   1st Qu.:0.0219588
 Median :3.2500   Median :0.1508402
 Mean   :3.2500   Mean   :0.2153124
 3rd Qu.:4.4100   3rd Qu.:0.3922926
 Max.   :5.5699   Max.   :0.6291699


$Extraversion
$Extraversion$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.2057

       x                y
 Min.   :0.7828   Min.   :0.0001224
 1st Qu.:1.9164   1st Qu.:0.0407787
 Median :3.0500   Median :0.1578239
 Mean   :3.0500   Mean   :0.2203167
 3rd Qu.:4.1836   3rd Qu.:0.3968226
 Max.   :5.3172   Max.   :0.5870843

$Extraversion$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1817

       x               y
 Min.   :1.055   Min.   :0.0000833
 1st Qu.:2.077   1st Qu.:0.0339189
 Median :3.100   Median :0.2100553
 Mean   :3.100   Mean   :0.2442280
 3rd Qu.:4.123   3rd Qu.:0.4375608
 Max.   :5.145   Max.   :0.5799863


$Agreeableness
$Agreeableness$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.1807

       x               y
 Min.   :1.058   Min.   :0.0001347
 1st Qu.:2.104   1st Qu.:0.0270296
```

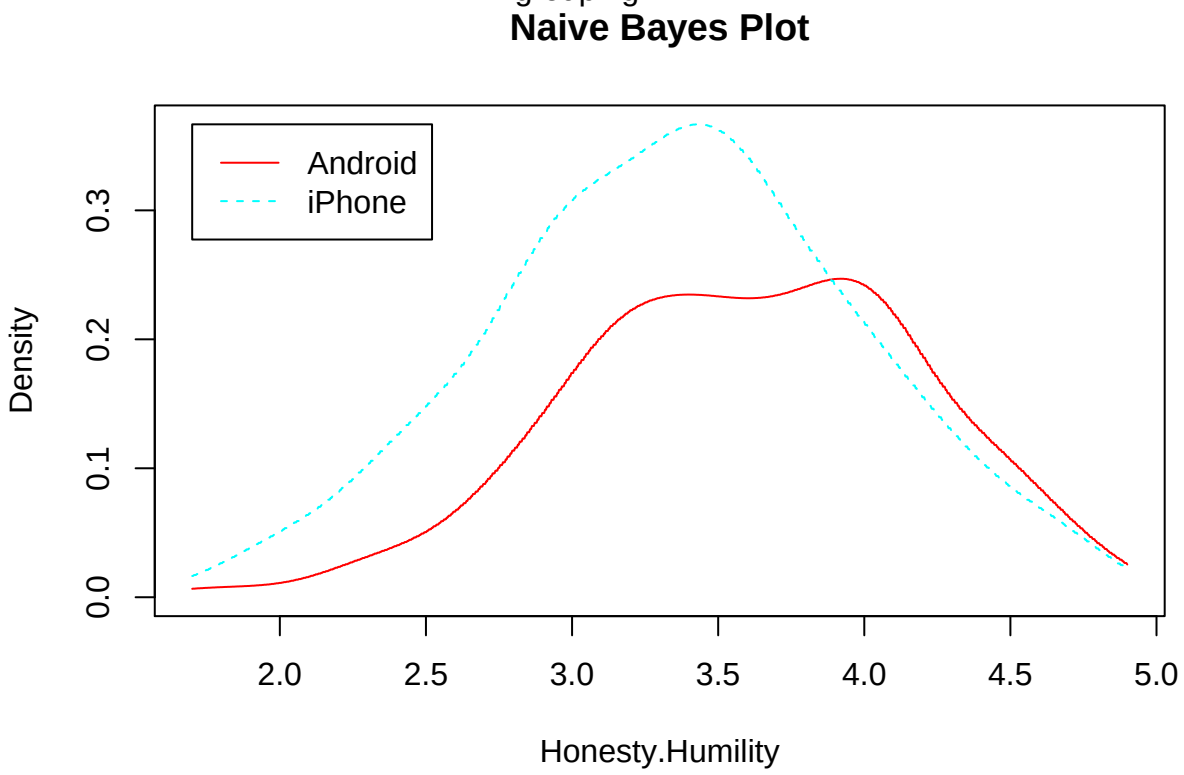
```
 Median :3.150   Median :0.1628894
 Mean   :3.150   Mean   :0.2387443
 3rd Qu.:4.196   3rd Qu.:0.4663836
 Max.   :5.242   Max.   :0.6169762

$Agreeableness$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1826

       x                y
 Min.   :0.8522   Min.   :0.000095
 1st Qu.:1.9261   1st Qu.:0.027074
 Median :3.0000   Median :0.181926
 Mean   :3.0000   Mean   :0.232570
 3rd Qu.:4.0739   3rd Qu.:0.445709
 Max.   :5.1478   Max.   :0.568577


$Conscientiousness
$Conscientiousness$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.181

       x                y
 Min.   :0.9569   Min.   :0.000114
 1st Qu.:2.1034   1st Qu.:0.021708
 Median :3.2500   Median :0.101329
 Mean   :3.2500   Mean   :0.217826
 3rd Qu.:4.3966   3rd Qu.:0.402436
 Max.   :5.5431   Max.   :0.706375

$Conscientiousness$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1706

       x               y
 Min.   :1.188   Min.   :0.0001964
 1st Qu.:2.244   1st Qu.:0.0359617
 Median :3.300   Median :0.1567514
 Mean   :3.300   Mean   :0.2365316
 3rd Qu.:4.356   3rd Qu.:0.4422605
 Max.   :5.412   Max.   :0.6719030


$Openness
```

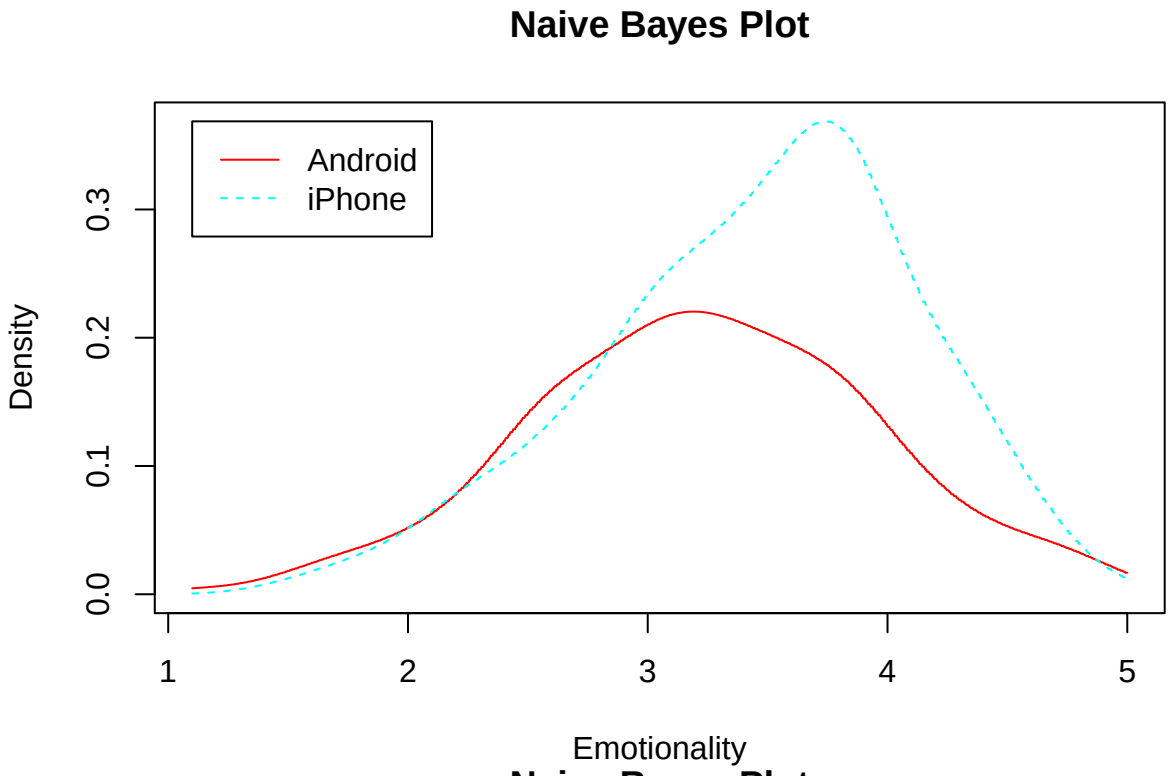
```
$Openness$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.1905

       x               y
 Min.   :1.029   Min.   :0.0001105
 1st Qu.:2.139   1st Qu.:0.0234947
 Median :3.250   Median :0.1359343
 Mean   :3.250   Mean   :0.2248574
 3rd Qu.:4.361   3rd Qu.:0.4378584
 Max.   :5.471   Max.   :0.6162173

$Openness$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1779

       x               y
 Min.   :1.166   Min.   :0.0000961
 1st Qu.:2.233   1st Qu.:0.0252414
 Median :3.300   Median :0.1804943
 Mean   :3.300   Mean   :0.2341143
 3rd Qu.:4.367   3rd Qu.:0.4645641
 Max.   :5.434   Max.   :0.5693733


$Avoidance.Similarity
$Avoidance.Similarity$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.2286

       x                y
 Min.   :0.3142   Min.   :0.00027
 1st Qu.:1.6071   1st Qu.:0.03275
 Median :2.9000   Median :0.15129
 Mean   :2.9000   Mean   :0.19315
 3rd Qu.:4.1929   3rd Qu.:0.37572
 Max.   :5.4858   Max.   :0.46712

$Avoidance.Similarity$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1706
```

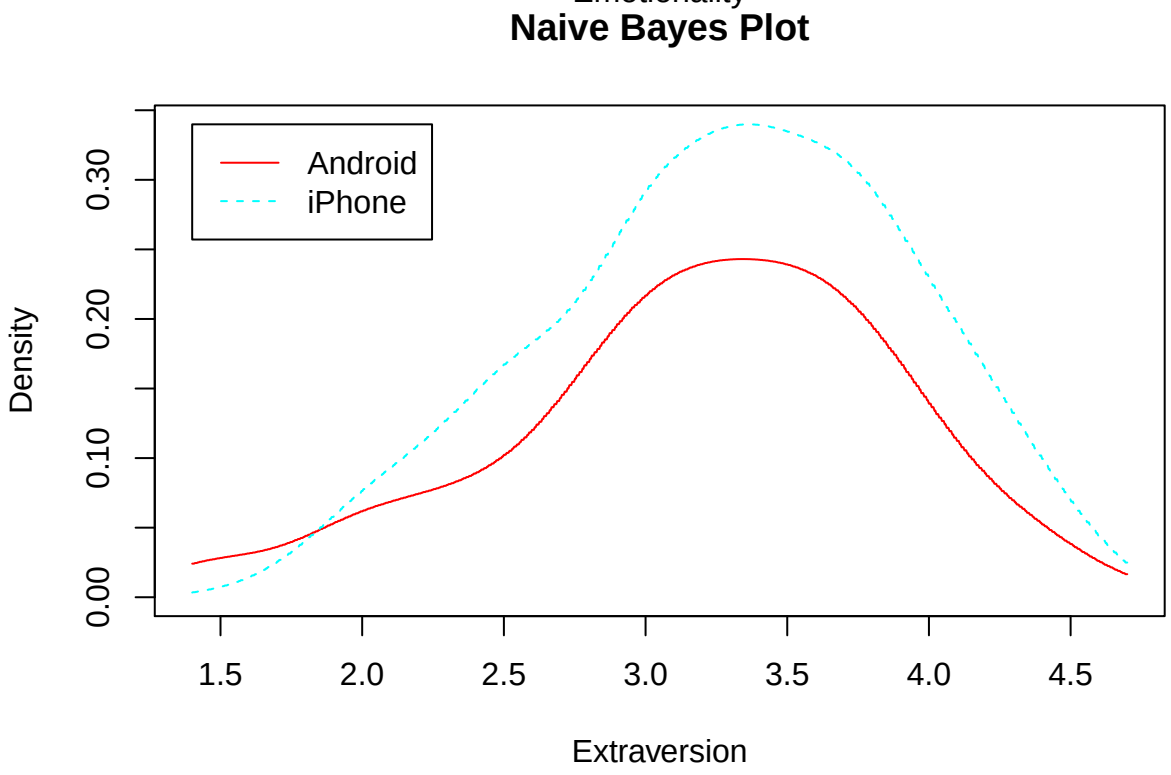
```
       x                y
 Min.   :0.4882   Min.   :0.0004298
 1st Qu.:1.7441   1st Qu.:0.0316214
 Median :3.0000   Median :0.1668941
 Mean   :3.0000   Mean   :0.1988441
 3rd Qu.:4.2559   3rd Qu.:0.2669040
 Max.   :5.5118   Max.   :0.8710047


$Phone.as.status.object
$Phone.as.status.object$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.1657

       x                y
 Min.   :0.5028   Min.   :0.0001238
 1st Qu.:1.4514   1st Qu.:0.0213441
 Median :2.4000   Median :0.1708817
 Mean   :2.4000   Mean   :0.2632744
 3rd Qu.:3.3486   3rd Qu.:0.4800005
 Max.   :4.2972   Max.   :0.6997651

$Phone.as.status.object$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.1706

       x                y
 Min.   :0.4882   Min.   :0.0001104
 1st Qu.:1.4441   1st Qu.:0.0385170
 Median :2.4000   Median :0.2001546
 Mean   :2.4000   Mean   :0.2612763
 3rd Qu.:3.3559   3rd Qu.:0.5033981
 Max.   :4.3118   Max.   :0.6255264


$Social.Economic.Status
$Social.Economic.Status$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 0.4572

       x                 y
 Min.   :-0.3715   Min.   :4.495e-05
 1st Qu.: 2.5642   1st Qu.:7.736e-03
 Median : 5.5000   Median :5.593e-02
 Mean   : 5.5000   Mean   :8.507e-02
```

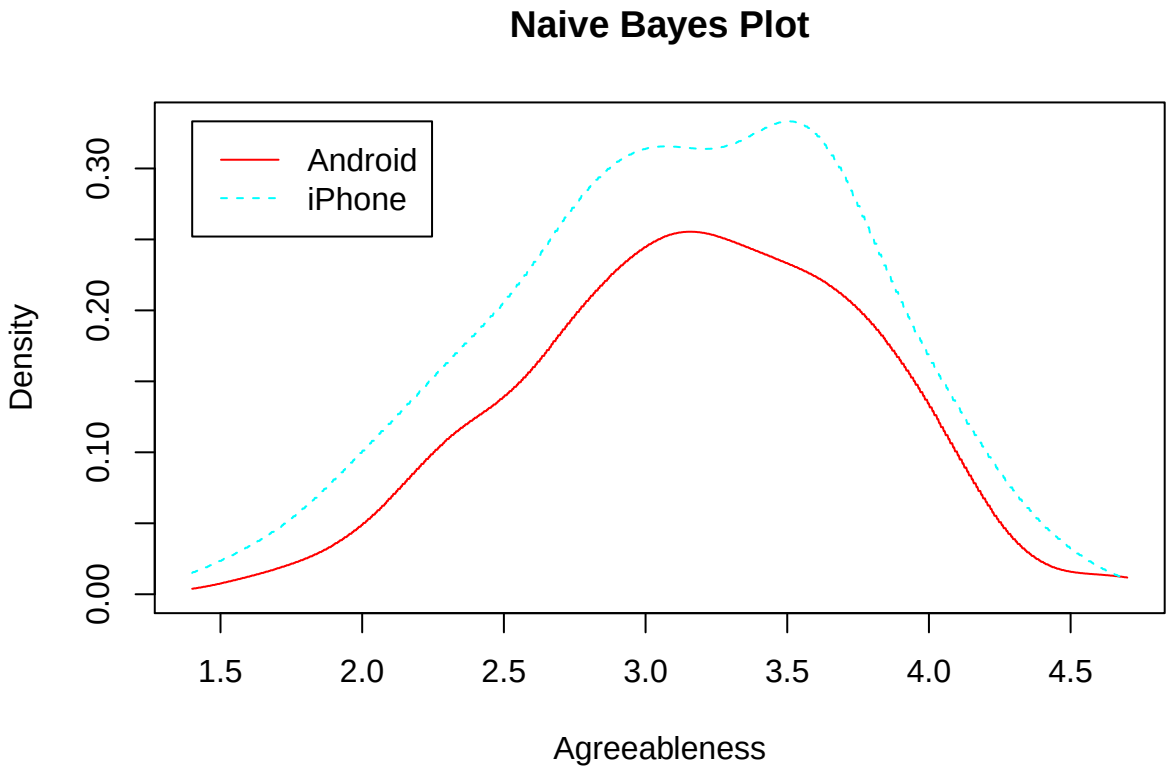
```
 3rd Qu.: 8.4358   3rd Qu.:1.503e-01
 Max.   :11.3715   Max.   :2.785e-01

$Social.Economic.Status$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 0.4245

       x                y
 Min.   :-0.2735   Min.   :6.835e-05
 1st Qu.: 2.6133   1st Qu.:7.145e-03
 Median : 5.5000   Median :3.632e-02
 Mean   : 5.5000   Mean   :8.652e-02
 3rd Qu.: 8.3867   3rd Qu.:1.511e-01
 Max.   :11.2735   Max.   :2.871e-01


$Time.owned.current.phone
$Time.owned.current.phone$Android

Call:
    density.default(x = xx)

Data: xx (219 obs.);    Bandwidth 'bw' = 2.972

       x                y
 Min.   : -8.915   Min.   :0.000e+00
 1st Qu.: 20.293   1st Qu.:8.400e-07
 Median : 49.500   Median :5.839e-04
 Mean   : 49.500   Mean   :8.550e-03
 3rd Qu.: 78.707   3rd Qu.:1.160e-02
 Max.   :107.915   Max.   :4.714e-02

$Time.owned.current.phone$iPhone

Call:
    density.default(x = xx)

Data: xx (310 obs.);    Bandwidth 'bw' = 2.5

       x               y
 Min.   :-7.50   Min.   :5.810e-06
 1st Qu.: 8.25   1st Qu.:9.996e-04
 Median :24.00   Median :5.201e-03
 Mean   :24.00   Mean   :1.586e-02
 3rd Qu.:39.75   3rd Qu.:3.214e-02
 Max.   :55.50   Max.   :5.031e-02
plot(nb2)
```

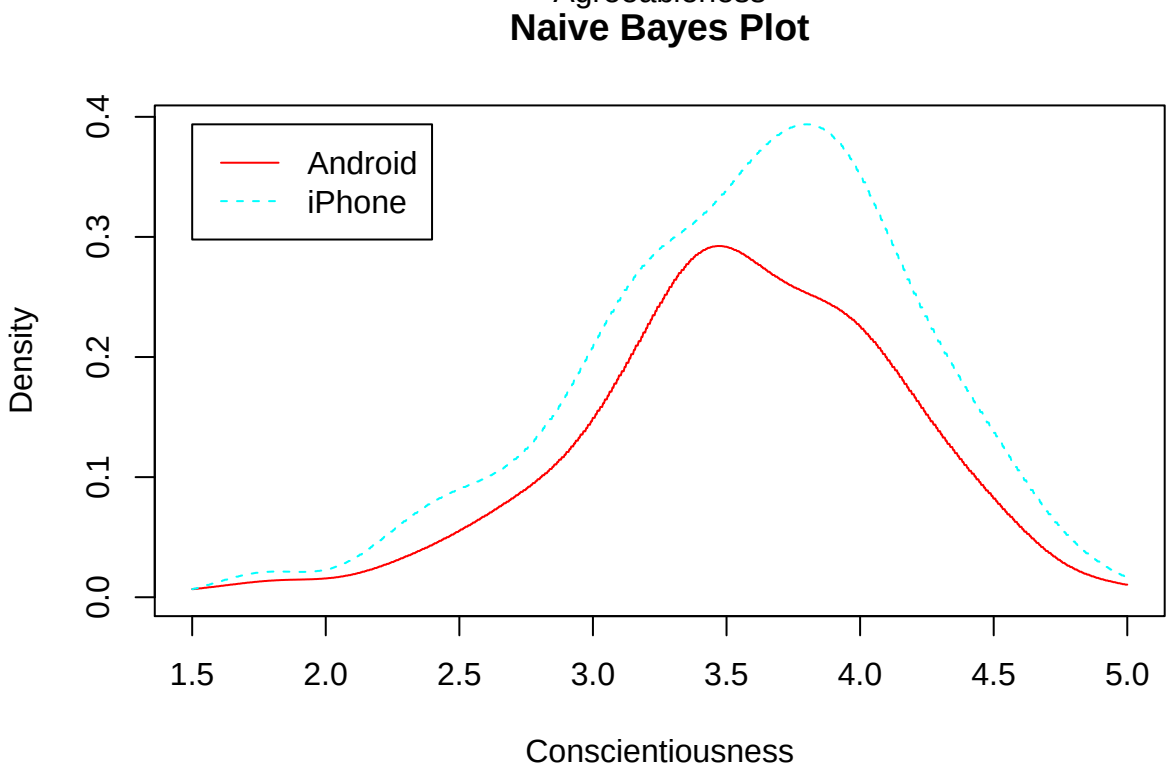

Naive Bayes Plot
Android
iPhone
female
male
var
grouping
Density
Android
iPhone
0.00
0.02
0.04
20

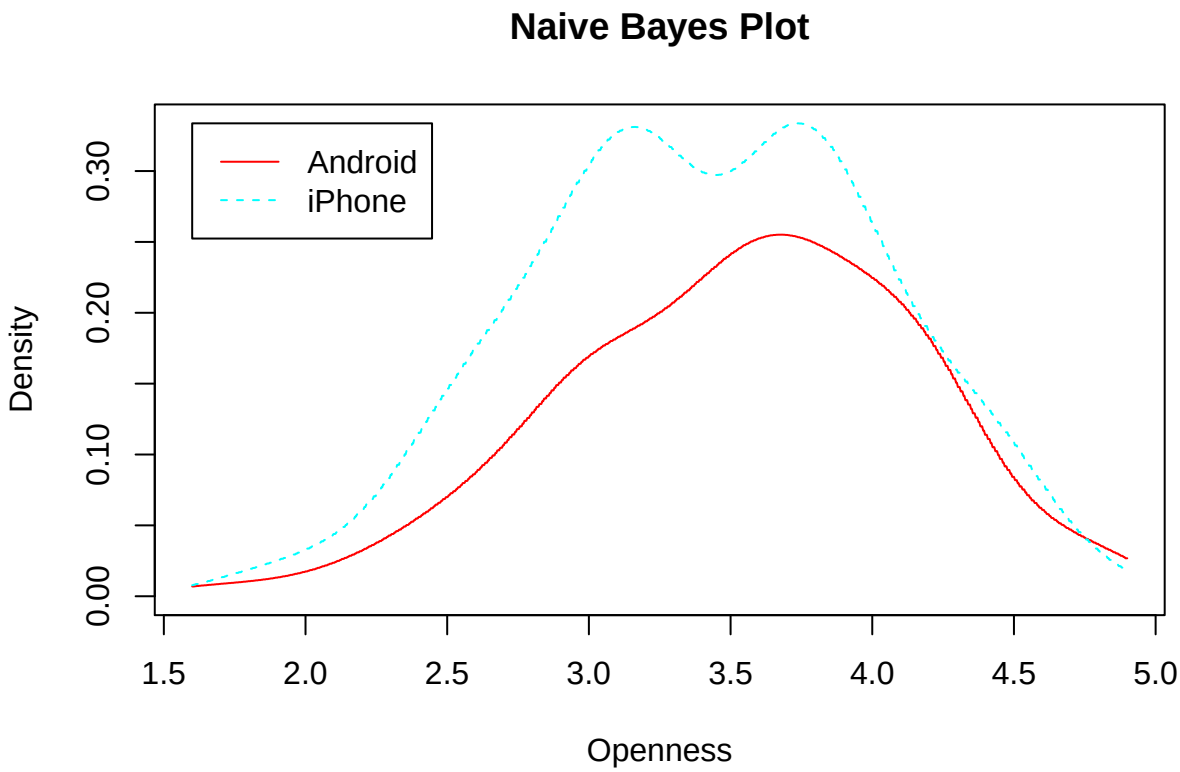

Naive Bayes Plot
Android
iPhone
Density
0.0
0.1
0.2
0.3
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Honesty.Humility

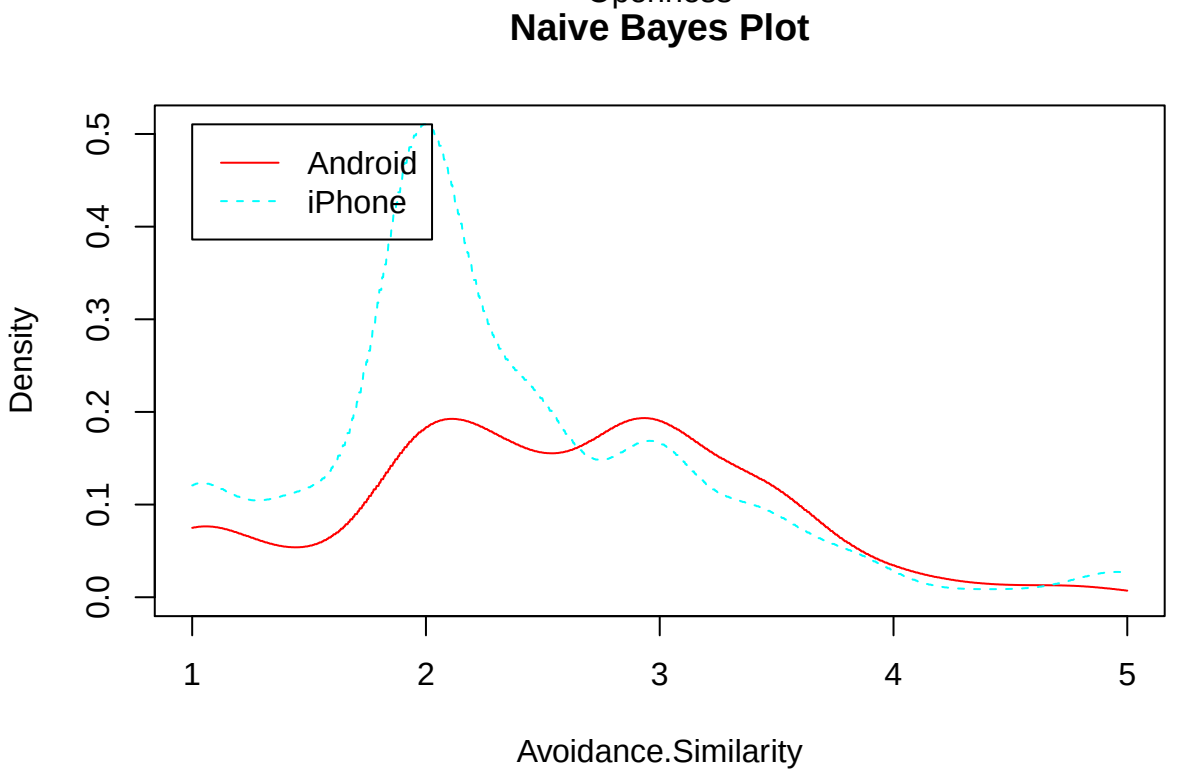

Naive Bayes Plot
Android
iPhone
Density
0.0
0.1
0.2
0.3
1
2
3
4
5
Emotionality

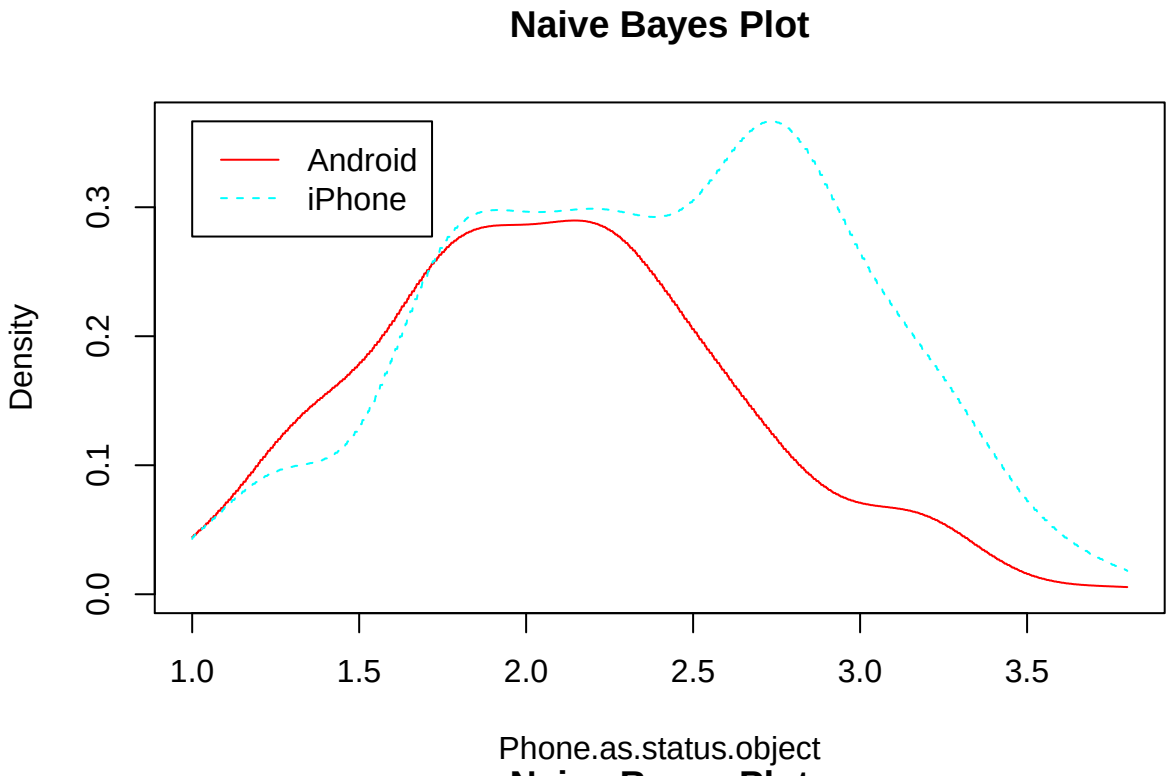

Naive Bayes Plot
Android
iPhone
Density
0.00
0.10
0.20
0.30
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Extraversion

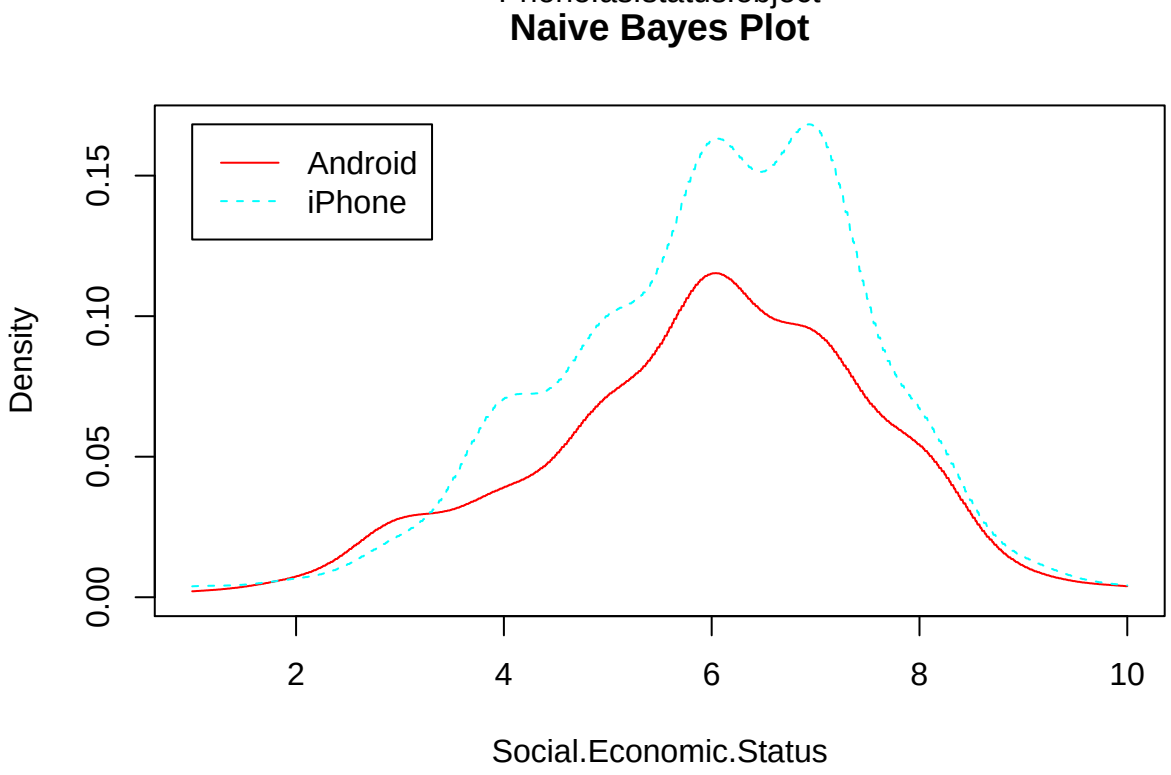

Naive Bayes Plot
Android
iPhone
Density
0.00
0.10
0.20
0.30
1.5
2.0
2.5
3.0
3.5
4.0
4.5
Agreeableness

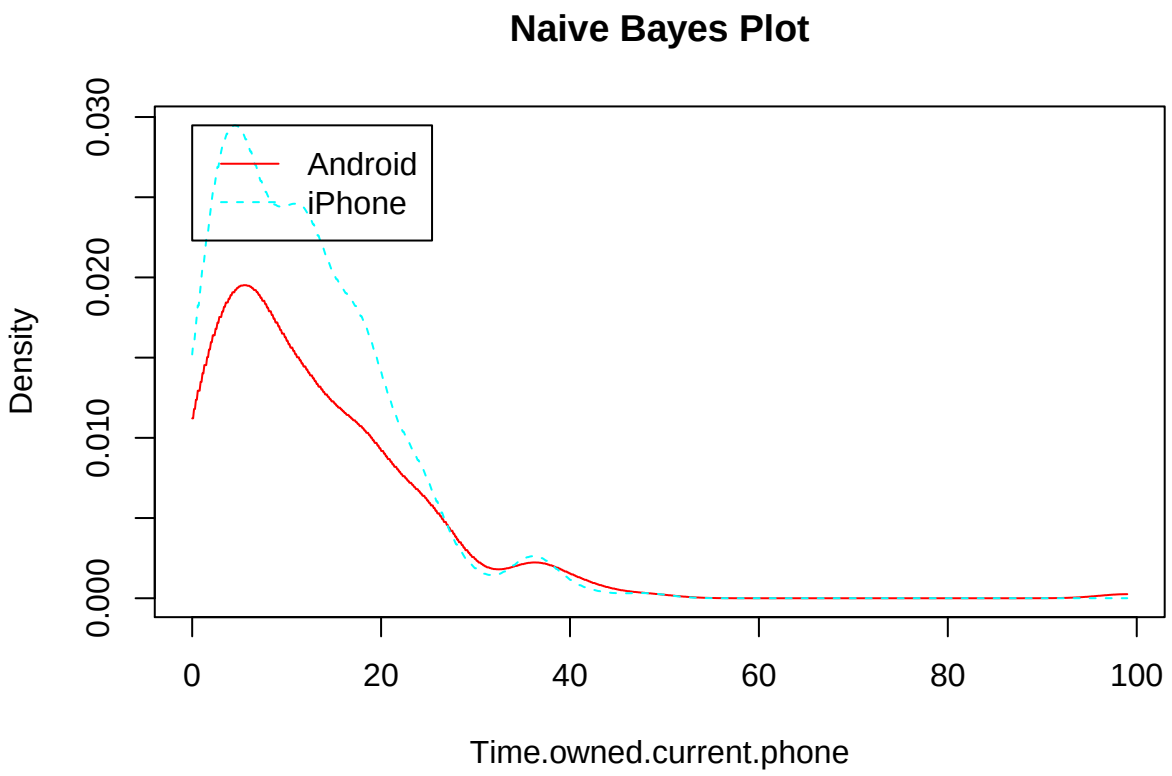

Naive Bayes Plot
Android
iPhone
Density
0.0
0.1
0.2
0.3
0.4
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Conscientiousness


Naive Bayes Plot
Android
iPhone
Density
0.00
0.10
0.20
0.30
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Openness


Naive Bayes Plot
Android
iPhone
Density
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
4
5
Avoidance.Similarity


Naive Bayes Plot
Android
iPhone
Density
0.0
0.1
0.2
0.3
1.0
1.5
2.0
2.5
3.0
3.5
Phone.as.status.object


Naive Bayes Plot
Android
iPhone
Density
0.00
0.05
0.10
0.15
2
4
6
8
10
Social.Economic.Status



Now, instead of just the mean and standard deviation, we have estimated quantiles of the distribution. This might help us when we have skewed distributions, but we need a lot of observations in order to get reliable estimates. 200-300 observations in each group might not be enough to do well. If we look at the predictions:

```
pred2 <- predict(nb2)
confusion(dat$Smartphone, pred2$class)
```

```
Overall accuracy = 0.699

Confusion matrix
         Predicted (cv)
Actual    Android iPhone
  Android   0.557  0.443
  iPhone    0.200  0.800
```

Here, we do a bit better, and in contrast to some of the other classification methods, we have a 55% chance of getting Android users right (the lda model sometimes had less than a 50% chance of getting them correct.)

Again, it would be good to implement a cross-validation here.

Example: Predicting Dengue Fever

The NaiveBayes function is sensitive to missing data and zero-variances. If you have a variable with no variance, any new value will have likelihood of 0, and we have a chance of getting a likelihood ratio that is infinite. Similarly, an NA in the values can cause trouble, dependent on how the model handles it. It might be useful to impute NA data, or add small amounts of noise to the training set to smooth out the values.

```
library(e1071)
data(dengue)
summary(dengue)
```

```
     humid            humid90            temp            temp90
 Min.   : 0.6714   Min.   : 1.066   Min.   :-18.68   Min.   :-10.07
 1st Qu.:10.0088   1st Qu.:10.307   1st Qu.: 11.10   1st Qu.: 12.76
 Median :16.1433   Median :16.870   Median : 20.99   Median : 22.03
 Mean   :16.7013   Mean   :17.244   Mean   : 18.41   Mean   : 19.41
 3rd Qu.:23.6184   3rd Qu.:24.131   3rd Qu.: 25.47   3rd Qu.: 25.98
     h10pix          h10pix90          trees          trees90
 Min.   : 4.317   Min.   : 5.848   Min.   : 0.0   Min.   : 0.00
 1st Qu.:14.584   1st Qu.:14.918   1st Qu.: 1.0   1st Qu.: 6.00
 Median :23.115   Median :24.130   Median :15.0   Median :30.60
 Mean   :21.199   Mean   :21.557   Mean   :22.7   Mean   :35.21
 3rd Qu.:28.509   3rd Qu.:28.627   3rd Qu.:37.0   3rd Qu.:63.62
     NoYes             Xmin              Xmax              Ymin
 Min.   :0.0000   Min.   :-179.50   Min.   :-172.00   Min.   :-54.50
 1st Qu.:0.0000   1st Qu.: -12.00   1st Qu.: -10.00   1st Qu.:  6.00
 Median :0.0000   Median :  16.00   Median :  17.75   Median : 18.00
 Mean   :0.4155   Mean   :  13.31   Mean   :  15.63   Mean   : 19.78
 3rd Qu.:1.0000   3rd Qu.:  42.62   3rd Qu.:  44.50   3rd Qu.: 39.00
      Ymax
 Min.   :-55.50
 1st Qu.:  5.00
 Median : 17.00
 Mean   : 18.16
 3rd Qu.: 37.00
 [ reached getOption("max.print") -- omitted 2 rows ]
```

```
dengue$NoYes <- as.factor(dengue$NoYes)
```

Notice that there areabout a dozen or so values with NA data.

```
# This doesn't work nb.dengue <- NaiveBayes(NoYes~.,data=dengue)

nb.dengue <- NaiveBayes(NoYes ~ ., data = dengue, na.action = "na.omit")  ##this works

# This one works with the klaR NaiveBayes:
nb.dengue2 <- NaiveBayes(NoYes ~ h10pix + Xmin + Ymin, data = dengue)


# this one works--from e1071 package
nb.dengue3 <- naiveBayes(NoYes ~ ., data = dengue)


# when we remove the na, we need to remove it from the ground truth too:
confusion(dengue$NoYes[!is.na(rowMeans(dengue[, -9]))], predict(nb.dengue)$class)
```

```
Overall accuracy = 0.882

Confusion matrix
      Predicted (cv)
Actual     0     1
     0 0.845 0.155
     1 0.066 0.934
```

```
confusion(dengue$NoYes, predict(nb.dengue2)$class)
```

```
Overall accuracy = 0.883
```

```
Confusion matrix
      Predicted (cv)
Actual     0     1
     0 0.843 0.157
     1 0.059 0.941
```

```
confusion(dengue$NoYes, predict(nb.dengue3, newdata = dengue))
```

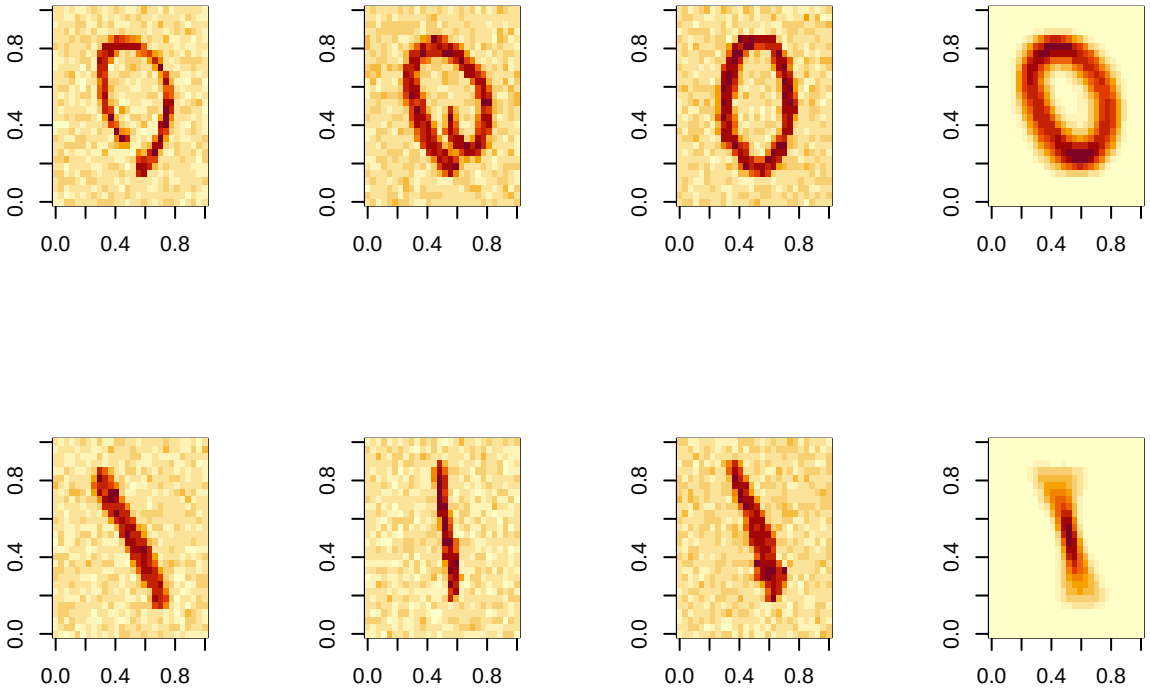
```
Overall accuracy = 0.881

Confusion matrix
      Predicted (cv)
Actual     0     1
     0 0.844 0.156
     1 0.066 0.934
```

Naive Bayes with the mnist (handwriting) set

```
##this code will dowload a create a 500-letter two-class training set from
##mnist via tensorflow
library(tensorflow)
datasets <- tf$contrib$learn$datasets
mnist <- datasets$mnist$read_data_sets("MNIST-data", one_hot = TRUE)

##extract just two labels and sample images

mnist.1 <- mnist$train$labels[,1]
mnist.2 <- mnist$train$labels[,2]

mnist.img1 <- mnist$train$images[mnist.1==1,]
mnist.img2 <- mnist$train$images[mnist.2==1,]


##plot prototypes
par(mfrow=c(1,2))
image((matrix(colMeans(mnist.img1),28,28)))
image((matrix(colMeans(mnist.img2),28,28)))

##these are too many. Sample 250 from each
traintest1 <- sample(1:nrow(mnist.img1),size=500)
traintest2 <- sample(1:nrow(mnist.img2),size=500)

train <- rbind(mnist.img1[traintest1[1:250],],
               mnist.img2[traintest2[1:250],])

test <- rbind(mnist.img1[traintest1[251:500],],
               mnist.img2[traintest2[251:500],])
train<-as.data.frame(train)
test<- as.data.frame(test)
train$labels <- rep(0:1,each=250)



write.csv(train,"trainmnist.csv",row.names=F)
```

```
write.csv(test,"testmnist.csv",row.names=F)
```

```
library(klaR)
library(DAAG)
train <- read.csv("trainmnist.csv")
test <- read.csv("testmnist.csv")

## smooth out the training set a bit by adding some noise, so no pixel has a sd of
## 0.
for (i in 1:ncol(train)) {
    train[, i] <- rnorm(500, as.numeric(train[, i]), 0.1)
}

par(mfrow = c(2, 4))
image(matrix(unlist(train[1, ]), nrow = 28))
image(matrix(unlist(train[2, ]), nrow = 28))
image(matrix(unlist(train[3, ]), nrow = 28))
image(matrix(colMeans(train[1:250, ]), nrow = 28))

image(matrix(unlist(train[251, ]), nrow = 28))
image(matrix(unlist(train[252, ]), nrow = 28))
image(matrix(unlist(train[253, ]), nrow = 28))
image(matrix(colMeans(train[251:500, ]), nrow = 28))
```



Build Naive Bayes model

```
train$labels <- as.factor(rep(0:1, each = 250))
nb3 <- NaiveBayes(labels ~ ., usekernel = T, data = train)
p3a <- predict(nb3)  ##this takes a while
confusion(train$labels, p3a$class)  ##almost perfect
```

```
Overall accuracy = 0.998
```

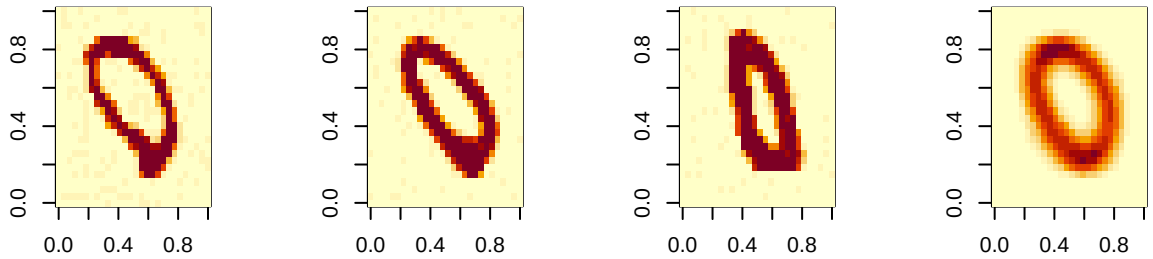
```
Confusion matrix
      Predicted (cv)
Actual     0     1
     0 0.996 0.004
     1 0.000 1.000
```

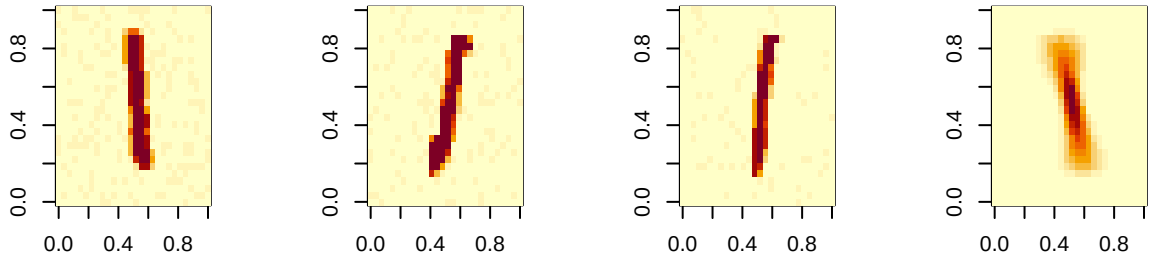
```
p3b <- predict(nb3, test)  ##equally good
confusion(rep(0:1, each = 250), p3b$class)
```

```
Overall accuracy = 0.998

Confusion matrix
      Predicted (cv)
Actual  [,1]  [,2]
  [1,] 1.000 0.000
  [2,] 0.004 0.996
```

What if we add noise to the test?

```
for (i in 1:ncol(test)) {
    test[, i] <- rnorm(500, as.numeric(test[, i]), 0.02)

}

par(mfrow = c(2, 4))
image(matrix(unlist(test[1, ]), nrow = 28))
image(matrix(unlist(test[2, ]), nrow = 28))
image(matrix(unlist(test[3, ]), nrow = 28))
image(matrix(colMeans(test[1:250, ]), nrow = 28))

image(matrix(unlist(test[251, ]), nrow = 28))
image(matrix(unlist(test[252, ]), nrow = 28))
image(matrix(unlist(test[253, ]), nrow = 28))
image(matrix(colMeans(test[251:500, ]), nrow = 28))
```



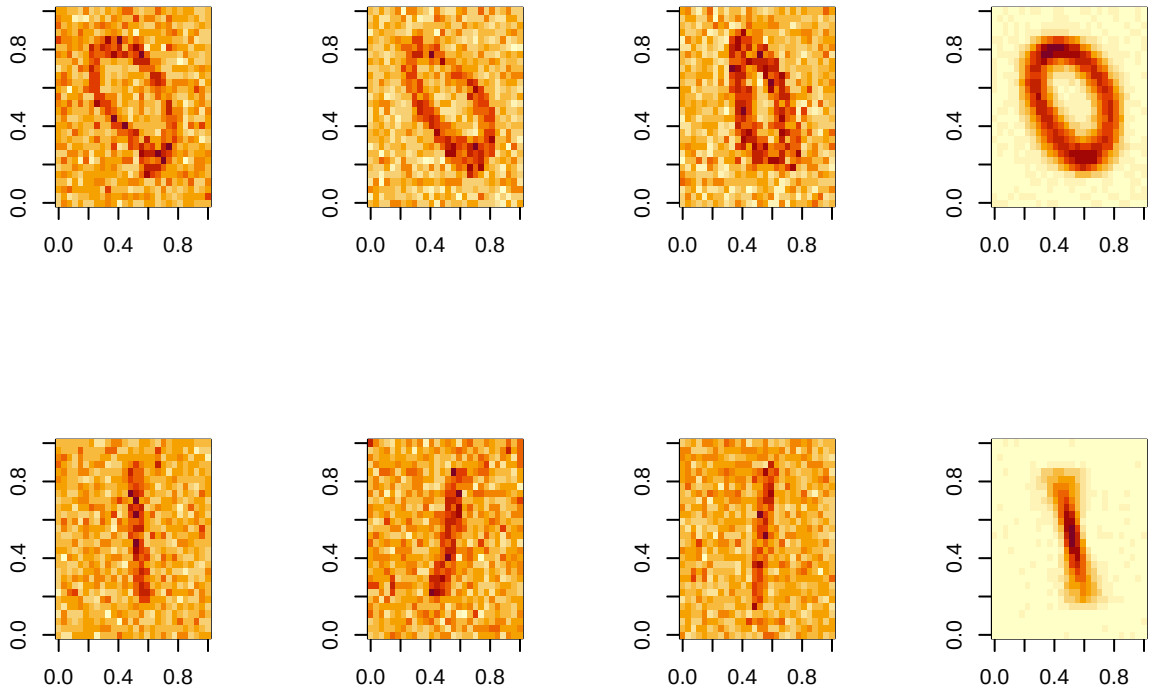


```
p3c <- predict(nb3, test)  ##This one is still pretty good
confusion(rep(0:1, each = 250), p3c$class)
```

```
Overall accuracy = 0.994

Confusion matrix
      Predicted (cv)
Actual  [,1]  [,2]
  [1,] 0.996 0.004
  [2,] 0.008 0.992
```

##add larger amount of noise

```
for (i in 1:ncol(test)) {
    test[, i] <- rnorm(500, as.numeric(test[, i]), 0.35)
}

par(mfrow = c(2, 4))
image(matrix(unlist(test[1, ]), nrow = 28))
image(matrix(unlist(test[2, ]), nrow = 28))
image(matrix(unlist(test[3, ]), nrow = 28))
image(matrix(colMeans(test[1:250, ]), nrow = 28))

image(matrix(unlist(test[251, ]), nrow = 28))
image(matrix(unlist(test[252, ]), nrow = 28))
image(matrix(unlist(test[253, ]), nrow = 28))
image(matrix(colMeans(test[251:500, ]), nrow = 28))
```




```
p3c <- predict(nb3, test)  ##This one is pretty bad.
confusion(rep(0:1, each = 250), p3c$class)
```

```
Overall accuracy = 0.5

Confusion matrix
      Predicted (cv)
Actual [,1] [,2]
  [1,]    1    0
  [2,]    1    0
```