Lab 4: Classification

Phuong Dong Le

5/12/2020

```r
library(ggplot2)
library(ggthemes)
library(ISLR)
```

4.6.1: The stock market data:

- We begin with examine the stock market data. This data consists of the percentage returns for the S&P 500 stock index over 1250 days from the beginning of 2001 until the end of 2005. For each date, we record the percentage returns for each of the five previous trading days, `Lag1` through `Lag5`. We also record the `Volume` (the number of shares traded on the previous day in billions), `Today` the percentage return on the date in question and `Direction` whether the market was `Up` or `Down` on this date.

```r
names(Smarket)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```r
dim(Smarket)
```

```
## [1] 1250    9
```

```r
summary(Smarket)
```

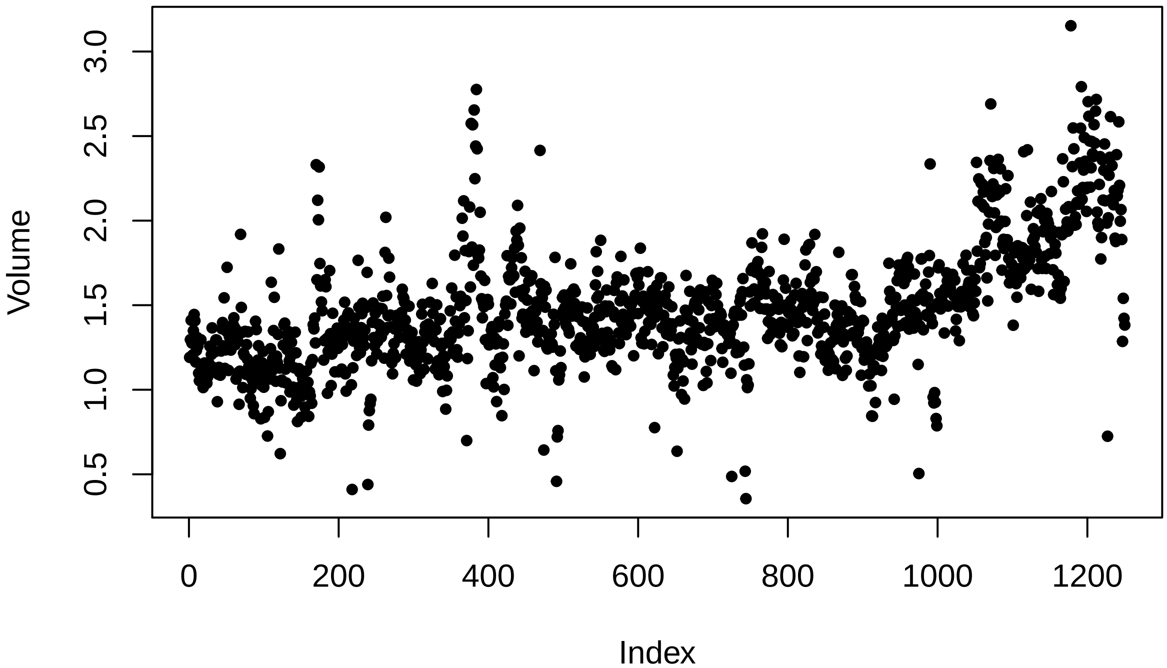
```
##       Year           Lag1                Lag2
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500
##  Median :2003   Median : 0.039000   Median : 0.039000
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000
##       Lag3                Lag4                Lag5
##  Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92200
##  1st Qu.:-0.640000   1st Qu.:-0.640000   1st Qu.:-0.64000
##  Median : 0.038500   Median : 0.038500   Median : 0.03850
##  Mean   : 0.001716   Mean   : 0.001636   Mean   : 0.00561
##  3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59700
##  Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73300
##      Volume           Today           Direction
##  Min.   :0.3561   Min.   :-4.922000   Down:602
##  1st Qu.:1.2574   1st Qu.:-0.639500   Up  :648
##  Median :1.4229   Median : 0.038500
##  Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   :3.1525   Max.   : 5.733000
```

```r
round(cor(Smarket[,-9]), digits = 3)
```

```
##         Year  Lag1  Lag2  Lag3  Lag4  Lag5 Volume Today
## Year   1.000 0.030 0.031 0.033 0.036 0.030  0.539 0.030
```

```
## Lag1   0.030  1.000 -0.026 -0.011 -0.003 -0.006  0.041 -0.026
## Lag2   0.031 -0.026  1.000 -0.026 -0.011 -0.004 -0.043 -0.010
## Lag3   0.033 -0.011 -0.026  1.000 -0.024 -0.019 -0.042 -0.002
## Lag4   0.036 -0.003 -0.011 -0.024  1.000 -0.027 -0.048 -0.007
## Lag5   0.030 -0.006 -0.004 -0.019 -0.027  1.000 -0.022 -0.035
## Volume 0.539  0.041 -0.043 -0.042 -0.048 -0.022  1.000  0.015
## Today  0.030 -0.026 -0.010 -0.002 -0.007 -0.035  0.015  1.000
```

```r
attach(Smarket)
plot(Volume, pch = 20)
```



- We see that `Volume` is increasing over time. The average number of shares traded daily increased from 2001 to 2005.

4.6.2: Logistic Regression

- We fit the logistic regression model to predict Direction using Lag1 through Lag5 and Volume.

```r
glm.fit = glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
              family = "binomial",
              data = Smarket)

summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = "binomial", data = Smarket)
##
## Deviance Residuals:
##    Min       1Q   Median       3Q      Max
## -1.446   -1.203    1.065    1.145    1.326
##
```

```
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

- The smallest p value here is associated with Lag1. Negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is likely to go up today.
- We access the coefficients in the model:

```
coef(glm.fit)
```

```
##  (Intercept)         Lag1         Lag2         Lag3         Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##         Lag5       Volume
##  0.010313068  0.135440659
```

- The `predict()` function is to predict the probabilty that the market will go up for a given values of predictors. The `type = response` tells `R` to output the probabilities of the form $P(Y = j|X)$ as opposed to other information such as the logit. Here we have printed the first ten probabilities. These values correspond to the probability that the market is going up rather than going down because the `contrasts()` function indicates that a dummy variable with 1 for `Up`.

```
glm.probs = predict(glm.fit, type = "response")
print(
  round(glm.probs[1:10], digits = 2)
)
```

```
##    1    2    3    4    5    6    7    8    9   10
## 0.51 0.48 0.48 0.52 0.51 0.51 0.49 0.51 0.52 0.49
```

- We create a vector of class predictions based on whether the pred prob. of a market increase is greater than or less than 0.5

```
glm.pred = rep("Down", 1250)
glm.pred[glm.probs > 0.5]  = "Up"
```

- The `table()` function can be used to produce a confusion matrix to determine how many observations were correctly or incorrectly classfied.

```
table(glm.pred, Direction)
```

```
##         Direction
## glm.pred Down  Up
##     Down  145 141
##     Up    457 507
```

- Here, our model correctly predicts that the market goes up on 507 days and that it would go down on 145 days for a total of 652 correct predictions. The `mean()` function can be used to compute the fraction of days for which the prediction was correct. In this case, logistic regression correctly predicts the movement of the market 52.2 % of the time.
- With training data: we first create a vector corresponding to the observations from 2001 to 2004. We then use this vector to create a held out dataset of observations from 2005.

```
train = (Year < 2005)
Smarket.2005 = Smarket[!train,]
dim(Smarket.2005)
```

```
## [1] 252   9
```

```
Direction.2005 = Direction[!train]
```

- The object `train` is a vector of 1250 elements corresponding to the observations in the data set. The elements of the vector correspond to observations that occured before 2005 are set to TRUE while those that correspond to observations in 2005 are set to FALSE. The command `Smarket[train, ]` picks out the submatrix of the stock market data corresponding to the dates before 2005. So `Smarket[!train,]` indicates a submatrix of the stock market data containing only the observations for which `train` is FALSE - the observations with dates in 2005.
- We now fit a logistic regression model using the subset of the observations that correspond to dates before 2005 using the `subset` argument.

```
glm.fits = glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               family = "binomial",
               data = Smarket,
               subset = train)

glm.probs = predict(glm.fits, Smarket.2005, type = "response")
```

- Notice, we have trained and tested our model on two completely seperate data sets: training - perfomed using only the dates before 2005, and testing - performed using only the dates in 2005. Finally, we compute the predictions for 2005 and compare them to the actual movements of the market over that time period.

```
glm.pred = rep("Down", 252)
glm.pred[glm.probs > 0.5] = "Up"
table(glm.pred, Direction.2005)
```

```
##         Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44
```

```
mean(glm.pred == Direction.2005)
```

```
## [1] 0.4801587
```

```
mean(glm.pred != Direction.2005)
```

```
## [1] 0.5198413
```

- The test error rate is 52% which is worse than random guessing!. We can remove the variables that appear not to be helpful in predicting Direction we can obtain a more effective model. We refit using Lag1 and Lag2 - seemed to have the highest predictive power in the ordinal logistic regression model.

```r
glm.fits = glm(Direction ~ Lag1 + Lag2, data = Smarket, subset = train,
               family= "binomial")

glm.probs = predict(object = glm.fits, Smarket.2005 ,type = "response")

glm.pred = rep("Down", 252)
glm.pred[glm.probs > 0.5] = "Up"

table(glm.pred, Direction.2005)
```

```
##         Direction.2005
## glm.pred Down  Up
##     Down   35  35
##     Up     76 106
```

```r
mean(glm.pred == Direction.2005)
```

```
## [1] 0.5595238
```

- 56% of daily movements have been correctly predicted. It is worth noting that in this case, a much simpler strategy of predicting that the market will increase every day will also be correct 56 % of the time.
- Suppose that we want to predict the returns associated with particular values of Lag1 and Lag2. We can predict Direction on a day when Lag1 and Lag2 equal to 1.2 and 1.1 respectively, and on a day when they equal to 1.5 and -0.8 respectively. We do this using `predict()` function:

```r
Direction.Predict = predict(glm.fits, newdata = data.frame(Lag1 = c(1.2,1.5),
                                                            Lag2 = c(1.1,-0.8)), type = "response")

print(Direction.Predict)
```

```
##         1         2
## 0.4791462 0.4960939
```

4.6.3: Linear Discriminant Analysis

- We perform the LDA on the Smarket data. we fit an LDA model with `lda()` function part of the `MASS` library.

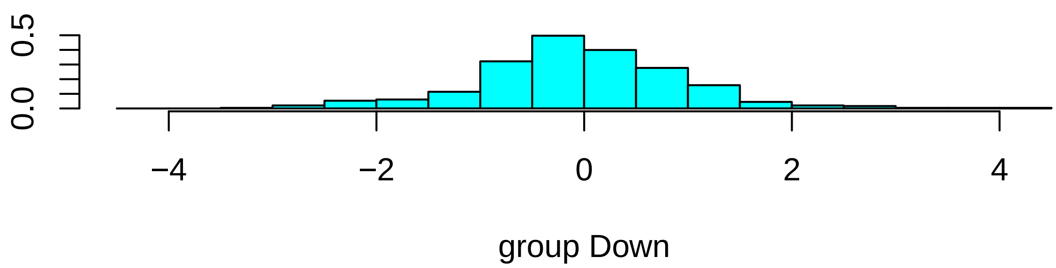
```r
library(MASS)
lda.fit = lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
summary(lda.fit)
```

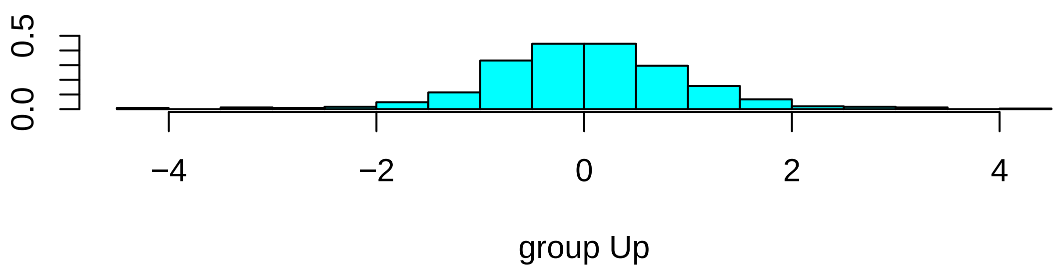
```
##         Length Class  Mode
## prior   2      -none- numeric
## counts  2      -none- numeric
## means   4      -none- numeric
## scaling 2      -none- numeric
## lev     2      -none- character
## svd     1      -none- numeric
## N       1      -none- numeric
## call    4      -none- call
## terms   3      terms  call
## xlevels 0      -none- list
```

```r
lda.fit
```

```
## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
```

```r
par(mfrow = c(2,2))
plot(lda.fit)
```





- LDA output indicates that $\hat{\pi}_1 = 0.492$ and $\hat{\pi}_2 = 0.508$. In other words, 49.2% of the training observations correspond to the days during which the market went down. The coefficients of linear discriminants ouput provides the linear combination of Lag1 and Lag2 that are used to form the LDA decision rule. If $-0.642 \times Lag1 - 0.514 \times Lag2$ is large, then the LDA classifier will predict a market increase, and if it is small then the LDA classifier will predict the market decline.
- The `predict()` functions return a list with 3 elements. The first element, `class`, contains LDA predictions about the movement of the market. The second element, `posterior` is a matrix whose kth column contains the posterior probability that the corresponding observation belongs to hte kth class.

```r
lda.pred = predict(object = lda.fit, Smarket.2005)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```r
lda.class = lda.pred$class
table(lda.class, Direction.2005)
```

```
##          Direction.2005
## lda.class Down  Up
##      Down   35  35
##      Up     76 106
```

```r
mean(lda.class == Direction.2005)
```

```
## [1] 0.5595238
```

- Applying a 50% threshold to the posterior probabilities allows us to recreate the predictions contained in `lda.pred$class`:

```r
sum(lda.pred$posterior[,1]>=0.5)
```

```
## [1] 70
```

```r
sum(lda.pred$posterior[,1]<0.5)
```

```
## [1] 182
```

- Notice the posterior probability output by the model corresponds to the probability that the market will decrease.

```r
lda.pred$posterior[1:20,1]
```

```
##       999      1000      1001      1002      1003      1004      1005
## 0.4901792 0.4792185 0.4668185 0.4740011 0.4927877 0.4938562 0.4951016
##      1006      1007      1008      1009      1010      1011      1012
## 0.4872861 0.4907013 0.4844026 0.4906963 0.5119988 0.4895152 0.4706761
##      1013      1014      1015      1016      1017      1018
## 0.4744593 0.4799583 0.4935775 0.5030894 0.4978806 0.4886331
```

```r
lda.class[1:20]
```

```
##  [1] Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Up   Down Up   Up
## [15] Up   Up   Up   Down Up   Up
## Levels: Down Up
```

- If we want to use a posterior probabilitiy threshold other than 50% to make the predictions, we could easily do so. If the posterior probability is at least 90%.

```r
sum(lda.pred$posterior[,1]>0.9)
```

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

- No days in 2005 meet that threshold. In fact, the greatest posterior probability of decrease in all of 2005 was 52.02%.

4.6.4: Quadratic Discriminant Analysis.

- We fit a QDA model to the Smarket data. We use `qda()` function which is part of `MASS` library.

```r
qda.fit = qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

- The output contains the group means. But it does not contain the coefficients of the linear discriminants because the QDA classifier involves a quadratic rather than a linear function of the predictors. The `predict()` function works in exactly the same fashion as for LDA.

```r
qda.pred = predict(object = qda.fit, Smarket.2005)
names(qda.pred)
```

```
## [1] "class"     "posterior"
```

```r
## Classifiers:

qda.class = qda.pred$class
table(qda.class, Direction.2005)
```

```
##          Direction.2005
## qda.class Down  Up
##      Down   30  20
##      Up     81 121
```

```r
## Accuracy of the model:
round(mean(qda.class == Direction.2005), digits = 2)
```

```
## [1] 0.6
```

- The QDA predictions are acurate almost 60% of the time even though the 2005 data was not used to fit the model. This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear forms assumed by LDA and logistic regression.

4.6.5: K-Nearest Neighbors:

```r
train = (Year < 2005)
Smarket.2005 = Smarket[!train,]
Direction.2005 = Direction[!train]
```

- We perform KNN using the `knn()` function which is part of the `class` library. The function requires 4 inputs:

1. A matrix containing the predictors associated with the training data labeled train.X
2. A matrix containing the predictors associated with the data we want to make predictions test.X
3. A vector containing the class labels for the training observations labeled train.Direction.

4. A value for K integer, the number of nearest neighbors to be used by the classifier.

```r
library(class)
train.X = cbind(Lag1, Lag2)[train,]
test.X = cbind(Lag1, Lag2)[!train,]
train.Direction = Direction[train]



### Now we use the knn function:

set.seed(1)
knn.pred = knn(train = train.X, test = test.X, train.Direction, k = 1)

table(knn.pred, Direction.2005)
```

```
##          Direction.2005
## knn.pred Down Up
##     Down   43 58
##     Up     68 83
```

```r
(83+43)/252
```

```
## [1] 0.5
```

- The result using $K = 1$ are not very good since only 50% of the observations are correctly predicted. Of course, it may be that $K = 1$ results in an overly flexible fit to the data. We repeat the analysis using $K = 3$.

```r
knn.pred = knn(train = train.X, test = test.X, train.Direction, k = 3)
table(knn.pred, Direction.2005)
```

```
##          Direction.2005
## knn.pred Down Up
##     Down   48 54
##     Up     63 87
```

```r
## Accuracy:
mean(knn.pred == Direction.2005)
```

```
## [1] 0.5357143
```

- The results have improved slightly. It appears that for this data, QDA provides the best results of the methods that we have examined so far.

4.6.6: Caravan Insurance Data:

- The data set contains 85 predictors that measure demographic characteristics of 5822 individuals. The response variable is Purchase indicating whether or not a given individual purchases a caravan insurance policy. In this data set, only 6% of people purchased caravan insurance.

```r
dim(Caravan)
```

```
## [1] 5822   86
```

```r
attach(Caravan)
summary(Purchase)
```

```
##   No  Yes
```

```
## 5474  348
```

- A good way to handle this problem is to standardize the data so that all variables are given a mean of zero and a standard deviation of one. Then all variables will be on a comparable scale. We exclude column 86 because that is the qualitative Purchase variable.

```r
standardized.X = scale(Caravan[,-86])

var(Caravan[,1])
```

```
## [1] 165.0378
```

```r
var(Caravan[,2])
```

```
## [1] 0.1647078
```

```r
var(standardized.X[,1])
```

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

```r
var(standardized.X[,2])
```

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

- We split the observations into a test set containing the first 1000 observations, and a training data set containing the remaining observations:

```r
test = 1:1000

train.X = standardized.X[-test,]
test.X = standardized.X[test, ]

train.Y = Purchase[-test]
test.Y = Purchase[test]


### Perform KNN method with K  = 1:

set.seed(1)

knn.pred = knn(train = train.X, test = test.X, train.Y, k = 1)
table(knn.pred, test.Y)
```

```
##          test.Y
## knn.pred   No Yes
##       No  873  50
##       Yes  68   9
```

```r
### Accuracy:
round(mean(knn.pred == test.Y), digits = 2)
```

```
## [1] 0.88
```

```r
round(mean(knn.pred != test.Y), digits = 2)
```

```
## [1] 0.12
```

```r
mean(test.Y != "No")
```

```
## [1] 0.059
```

- It turns out that KNN with K = 1 does far better than random guessing among the customers that are predicted to buy insurance. Among 77 customers, only 11.7% actually do purchase the insurance. This is double the rate that one would obtain from random guessing. Using K = 3, the success rate increases to 19% and with K = 5, the rate is 5%. This is four times the rate that results from random guessing.

```r
knn.pred = knn(train = train.X, test = test.X, train.Y, k = 3)
table(knn.pred, test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  920  54
##      Yes  21   5
```

```r
5/26
```

```
## [1] 0.1923077
```

```r
knn.pred = knn(train = train.X, test = test.X, train.Y, k = 5)
table(knn.pred, test.Y)
```

```
##         test.Y
## knn.pred  No Yes
##      No  930  55
##      Yes  11   4
```

```r
4/15
```

```
## [1] 0.2666667
```

- We can also fit a logistic regression model to the data. If we use 0.5 as the predicted probability cut off for the classifer, then we have a problem. If we instead choose a threshold of 0.25, we get much better result: we predict that 33 people will purchase the insurance and we are correct for about 33% of theese people. This is over 5 times better than random guessing.

```r
glm.fits = glm(Purchase~., data = Caravan, family = "binomial", subset = -test)

glm.probs = predict(glm.fits, Caravan[test,], type = "response")
glm.pred = rep("No", 1000)
glm.pred[glm.probs>0.5] = "Yes"

table(glm.pred, test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      No  934  59
##      Yes   7   0
```

```r
### with different threshold 0.25:

glm.fits = glm(Purchase~., data = Caravan, family = "binomial", subset = -test)

glm.probs = predict(glm.fits, Caravan[test,], type = "response")
glm.pred = rep("No", 1000)
glm.pred[glm.probs>0.25] = "Yes"

table(glm.pred, test.Y)
```

```
##         test.Y
## glm.pred  No Yes
```

```
##      No  919  48
##      Yes  22  11
```

```r
11/(22+11)
```

```
## [1] 0.3333333
```