Lab 03 - Classification

Ken Ye

9/13/2023

1. Stock Market Data

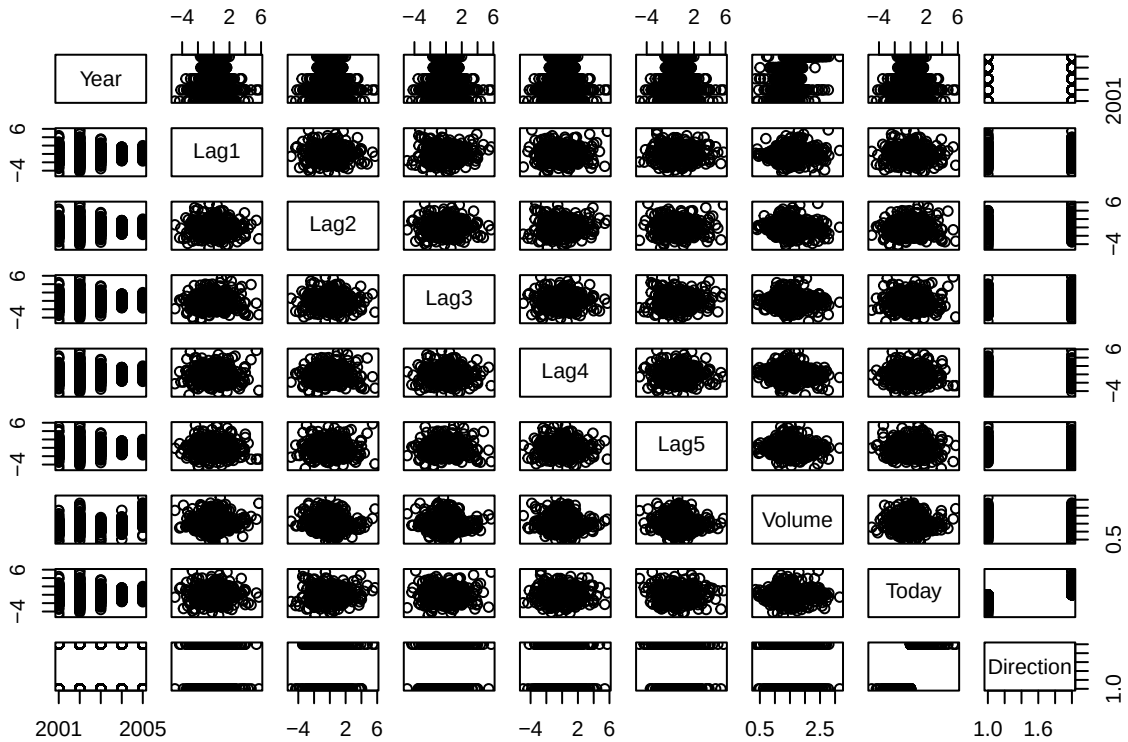
This lab will work with the `Smarket` stock market data from the `ISLR` library. The data consists of the percentage returns for the S&P 500 stock index for each day between the beginning of 2001 to the end of 2005. Each date has information about the percentage returns for the 5 previous trading days (`Lag1` through `Lag5`), the `Volume` (number of shares traded on the previous day), `Today` (percentage return on given date) and `Direction` (if the market was up or down on this date.)

We'll start by loading and examining the data.

```
library(ISLR)
attach(Smarket)
names(Smarket)
```

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

```
pairs(Smarket)
```



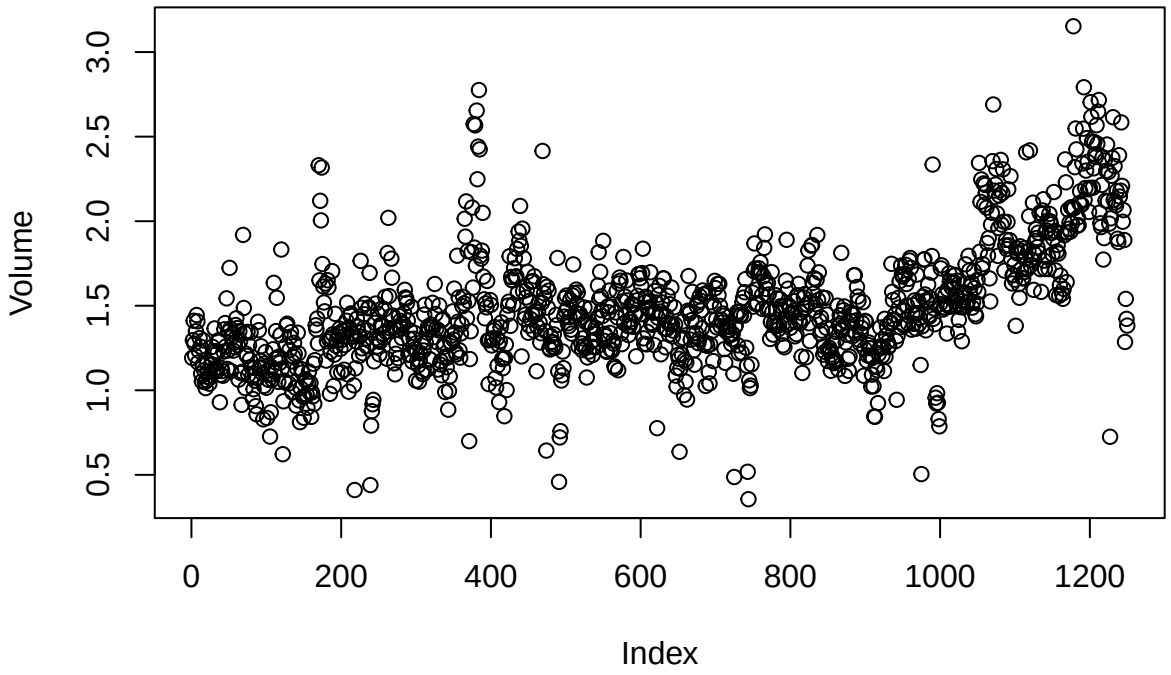
The `cor()` function can be used to calculate the matrix of all pairwise correlations between predictors.

```
cor(Smarket[, -9])
```

```
##              Year         Lag1         Lag2         Lag3         Lag4
## Year   1.00000000  0.029699649  0.030596422  0.033194581  0.035688718
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402 -0.002985911
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670 -0.010853533
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000 -0.024051036
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036  1.000000000
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686 -0.048414246
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527
##                Lag5      Volume        Today
## Year    0.029787995  0.53900647  0.030095229
## Lag1   -0.005674606  0.04090991 -0.026155045
## Lag2   -0.003557949 -0.04338321 -0.010250033
## Lag3   -0.018808338 -0.04182369 -0.002447647
## Lag4   -0.027083641 -0.04841425 -0.006899527
## Lag5    1.000000000 -0.02200231 -0.034860083
## Volume -0.022002315  1.00000000  0.014591823
## Today  -0.034860083  0.01459182  1.000000000
```

We can also look at the `Volume` variable over the time period considered.

```
plot(Volume)
```



2. Logistic Regression

The variable of interest is predicting the `Direction`, whether a stock moved `Up` or `Down`. We will fit a logistic regression model to predict `Direction` using the other variables (except `Today`).

The `glm()` function can be used to fit generalized linear models, of which logistic regression is an example. The functionality of `glm()` is similar to that of the `lm()` function from last week. We will need to pass the argument `family = binomial` to the `glm()` function to specify logistic regression.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Smarket, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## 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 `summary()` and `coef()` functions can also be used with `glm()` model fits.

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

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

```
summary(glm.fit)$coef
```

```
##                 Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.126000257 0.24073574 -0.5233966 0.6006983
## Lag1        -0.073073746 0.05016739 -1.4565986 0.1452272
## Lag2        -0.042301344 0.05008605 -0.8445733 0.3983491
## Lag3         0.011085108 0.04993854  0.2219750 0.8243333
## Lag4         0.009358938 0.04997413  0.1872757 0.8514445
## Lag5         0.010313068 0.04951146  0.2082966 0.8349974
## Volume       0.135440659 0.15835970  0.8552723 0.3924004
```

```
summary(glm.fit)$coef[, 4]
```

```
## (Intercept)        Lag1        Lag2        Lag3        Lag4        Lag5
##   0.6006983   0.1452272   0.3983491   0.8243333   0.8514445   0.8349974
##      Volume
##   0.3924004
```

The `predict()` function can also be used, similar to its use with the `lm()` function.

```
glm.probs <- predict(glm.fit, type = "response")
glm.probs[1:10]
```

```
##         1         2         3         4         5         6         7         8
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509 0.5092292
##         9        10
## 0.5176135 0.4888378
```

The `contrasts()` function can be used to see the dummy variable encoding for the `Direction` variable.

```
contrasts(Direction)
```

```
##      Up
## Down  0
## Up    1
```

This means that when `Up = 0`, `Direction = Down`, while when `Up=1`, we have that `Direction = Up`.

Next, we will want to convert the predicted probabilities to either `Up` or `Down` based on the value of the probability. Probabilities greater than 0.5 will be classified as `Up`.

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

We can create a confusion matrix using the `table()` function. The confusion matrix gives us the number of True Positives, True Negatives, False Positives and False Negatives.

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

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

We can calculate the accuracy of the classifier by finding the percentage of points that are correctly classified.

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

```
## [1] 0.5216
```

Now, we will divide our dataset into training and validation sets. Observations from 2001-2004 will be in the training set and observations from 2005 will be in the validation set.

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

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

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

We can again fit the logistic regression model using `glm()`, this time restricted to just our training set. We can then predict the `Direction` on the validation set using the `predict()` function.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Smarket, family = binomial, subset = train)
glm.probs <- predict(glm.fit, Smarket.2005, type = "response")
```

Then, we can compare the predictions on the validation set to the true labels and calculate the accuracy of our logistic regression classifier.

```
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) ## Accuracy
```

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

```
mean(glm.pred != Direction.2005) ## Misclassification rate
```

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

We can improve the predictive performance of our classifier by considering a subset of the predictor variables; only those that have the strongest relationship to the response. From the correlation matrix calculated above, we can see that these variables are `Lag1` and `Lag2`.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Smarket, family = binomial, subset = train)
glm.probs <- predict(glm.fit, 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
```

The predictive performance has slightly improved by limiting the predictor variables to a subset.

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

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

We can also predict the `Direction` for two new days with the lags given below using the `predict()` function.

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

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

3. K-Nearest Neighbors

To fit a K-nearest neighbors model, we need to use the `class()` package. First, however, we need to split our data into training and validation sets. We can use `cbind()` (column bind) to bind `Lag1` and `Lag2` into a matrix for each subset.

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

We need to use `set.seed()` to initialize the random number generator for consistent results and then can call the `knn()` function to fit the classifier and make predictions about `Direction`.

```
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Direction, k = 3)
table(knn.pred, Direction.2005)
```

```
##         Direction.2005
## knn.pred Down Up
##     Down   48 55
##     Up     63 86
```

The accuracy of the classifier on the test set can be found using the diagonal of the confusion matrix.

```
accuracy <- sum(diag(table(knn.pred, Direction.2005)))/nrow(Smarket)
```

We can also repeat the fit with $K = 3$ and compute the confusion matrix and accuracy.

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

```
##         Direction.2005
## knn.pred Down Up
##     Down   48 55
##     Up     63 86
```

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

```
## [1] 0.531746
```

4. Application

We can also apply the KNN classifier to the `Caravan` dataset. The response variable is `Purchase`, which indicates whether or not an individual purchases a caravan insurance policy. This dataset is an imbalanced classification problem, as only 6% of people in the dataset purchase the caravan insurance.

```r
attach(Caravan)
dim(Caravan)
```

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

```r
summary(Purchase)
```

```
##   No  Yes
## 5474  348
```

```r
348/5822
```

```
## [1] 0.05977327
```

We can use the `scale()` function to scale the data to have a mean of 0 and a standard deviation of 1.

```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
```

Again, we can split the data into a training and test set and make predictions about the response variable `Purchase` using a KNN model.

```
test <- 1:1000
train.X <- standardized.X[-test, ]
test.X <- standardized.X[test, ]
train.Y <- Purchase[-test]
test.Y <- Purchase[test]
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Y, k = 1)
mean(test.Y != knn.pred)
```

```
## [1] 0.118
```

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

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

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

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

```
9/(68 + 9)
```

```
## [1] 0.1168831
```

We can also fit the model with other values of K, for example $K = 3$ and $K = 5$.

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

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

```
5/26
```

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

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

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

```
4/15
```

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

Finally, we can compare the KNN model with a logistic regression model fit using `glm()`.

```
glm.fit <- glm(Purchase ~ ., data = Caravan, family = binomial, subset = -test)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.probs <- predict(glm.fit, 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
```

```
glm.pred <- rep("No", 1000)
glm.pred[glm.probs > 0.25] <- " Yes"
table(glm.pred, test.Y)
```

```
##         test.Y
## glm.pred  No Yes
##      Yes  22  11
##     No   919  48
```

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

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

5. Exercises

1. Logistic Regression

We will again use the `iris` data for classification.

```
data("iris")
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
attach(iris)
```

(a) Find the number of unique `Species` for the `iris` data.

There are 3 unique species.

```r
unique(Species)
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

(b) Since there are more than 2 classes for the `iris` data, we will consider a 1-vs-rest approach to classification. Add a new column to the `iris` dataset that has the labels that we wil use for classificaition. Call this column `Species2`. Encode this column such that the `setosa` is the positive class (`Speicies2 = 1`), while the other Species correspond to the negative class (the outcome variable is 0 for these other classes).

```r
iris$Species2 <- ifelse(Species == "setosa", 1, 0)
```

(c) Next, we need to divide our data into a training and validation set. Examine the `iris` data and propose a method for this division. What do we need to be careful about based on the original `iris` data?

We need to make sure the training and the testing set has similar distribution of the species.

```r
## Need to shuffle data, all classes in order right now
set.seed(1)
iris <- iris[sample(nrow(iris)), ]
```

(d) What are some cases where we need to be careful how we split our data into training and validation sets? What is the usual assumption when we split the data randomly?

Imbalanced Classes: If the dataset has imbalanced classes, a random split might lead to a training set with insufficient examples of the minority class.

Small Datasets: For small datasets, random splitting can result in training sets that are too small to build a robust model.

Independent and Identically Distributed (IID): The usual assumption when splitting data randomly is that the observations are independent and identically distributed. This means that each observation is generated from the same distribution and is independent of all other observations. Under this assumption, random splitting is appropriate because it ensures that both training and validation sets are representative samples of the overall distribution of the data.

(e) Now, split the `iris` data into a training and validation set. Use 30% of the data for the validation set. Make sure to use `set.seed()` (see Lab 01 for a reminder about this function). Print the dimensions of you training and validation sets.

```r
set.seed(1)
num_train <- floor(0.7*nrow(iris))
train <- iris[1:num_train,]
val <- iris[(num_train+1):nrow(iris),]
print(dim(train))
```

```
## [1] 105   6
```

```r
print(dim(val))
```

```
## [1] 45  6
```

(f) Fit a logistic regression classifier to your `iris` training data. Display a summary of the model fit. Use `Sepal.Length`, `Sepal.Width`, `Petal.Length` and `Petal.Width` as predictors.

```
glm.fit <- glm(Species2 ~ Sepal.Length + Sepal.Width +
                 Petal.Length + Petal.Width,
               data = train, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Species2 ~ Sepal.Length + Sepal.Width + Petal.Length +
##     Petal.Width, family = binomial, data = train)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -24.533 238278.898       0        1
## Sepal.Length     14.585  76221.596       0        1
## Sepal.Width       5.409  63229.673       0        1
## Petal.Length    -19.480  68101.956       0        1
## Petal.Width     -19.322 145833.119       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.3367e+02  on 104  degrees of freedom
## Residual deviance: 1.7013e-09  on 100  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

(g) Explore adding some interaction terms or dropping terms from the model that do not appear significant. You do not need to perform forward/backward selection, just explore the data and model fits. Interpret the parameters in your final model.

```
glm.fit2 <- glm(Species2 ~ Sepal.Length * Sepal.Width +
                  Petal.Length * Petal.Width,
                data = train, family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm.fit2)
```

```
##
## Call:
```

```
## glm(formula = Species2 ~ Sepal.Length * Sepal.Width + Petal.Length *
##     Petal.Width, family = binomial, data = train)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -4.616e+01  1.283e+06       0        1
## Sepal.Length                1.914e+01  2.685e+05       0        1
## Sepal.Width                 3.248e+01  5.195e+05       0        1
## Petal.Length               -2.507e+01  1.366e+05       0        1
## Petal.Width                -4.545e+01  3.089e+05       0        1
## Sepal.Length:Sepal.Width -4.686e+00  9.783e+04       0        1
## Petal.Length:Petal.Width  1.016e+01  3.679e+04       0        1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.3367e+02  on 104  degrees of freedom
## Residual deviance: 1.0070e-09  on  98  degrees of freedom
## AIC: 14
##
## Number of Fisher Scoring iterations: 25
```

(h) Use the `predict()` function to predict the species for your validation data. Use `predict()` for both models, from part (f) and part (g).

```
glm.probs <- predict(glm.fit, val, type = "response")
glm.probs2 <- predict(glm.fit2, val, type = "response")
```

(i) Calculate the confusion matrix and accuracy for your two models, from parts (f) and (g). Make sure to first convert your predicted probabilities from part (h) to `Species2`. Assume that a probability of greater than 0.5 corresponds to the positive class species, i.e. `Species2=1` and `Species = setosa`. Which model would you select and why?

Both model have a 100% accuracy, possibly due to perfect or near-perfect separation in the data, where the predictors can perfectly (or almost perfectly) predict the outcome variable. In this case, Species2 can be perfectly predicted.

```
# model 1
glm.pred <- rep(0, 45)
glm.pred[glm.probs > 0.5] <- 1
table(glm.pred, val$Species2)
```

```
##
## glm.pred  0  1
##        0 30  0
##        1  0 15
```

```
mean(glm.pred == val$Species2) # accuracy
```

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

```r
# model 2
glm.pred2 <- rep(0, 45)
glm.pred2[glm.probs2 > 0.5] <- 1
table(glm.pred2, val$Species2)
```

```
##
## glm.pred2  0  1
##         0 30  0
##         1  0 15
```

```r
mean(glm.pred2 == val$Species2) # accuracy
```

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

2. Conceptual

Think about the `Caravan` example above. Is accuracy a good measure to evaluate potential classifiers on this data set? Why or why not? What might be some better metrics to use to evaluate classifiers for imbalanced data sets?

No, accuracy may not be a good measure because a model that always predicts the majority class (no purchase) will achieve an accuracy of 94%, which might give a false sense of a well-performing mode.

A metric we could use is precision, which is calculated by TP/(TP + FP). It measures the proportion of true positive predictions among all instances that are predicted as positive.

Another one, which is similar, is sensitivity, which is calculated by TP/(TP + FN). It measures the proportion of true positive predictions among all actual positive instances.