Lab: Classification Methods

The Stock Market Data

We will begin by examining some numerical and graphical summaries of the `Smarket` data, which is part of the `ISLR2` library. This data set consists of percentage returns for the S&P 500 stock index over $1,250$ days, from the beginning of 2001 until the end of 2005. For each date, we have recorded the percentage returns for each of the five previous trading days, `Lag1` through `Lag5`. We have also recorded `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). Our goal is to predict `Direction` (a qualitative response) using the other features.

```
> library(ISLR2)
> names(Smarket)
[1] "Year"       "Lag1"       "Lag2"       "Lag3"       "Lag4"
[6] "Lag5"       "Volume"     "Today"      "Direction"
> dim(Smarket)
[1] 1250    9
> summary(Smarket)
      Year            Lag1                Lag2
 Min.   :2001   Min.   :-4.92200   Min.   :-4.92200
 1st Qu.:2002   1st Qu.:-0.63950   1st Qu.:-0.63950
 Median :2003   Median : 0.03900   Median : 0.03900
 Mean   :2003   Mean   : 0.00383   Mean   : 0.00392
 3rd Qu.:2004   3rd Qu.: 0.59675   3rd Qu.: 0.59675
 Max.   :2005   Max.   : 5.73300   Max.   : 5.73300
      Lag3                Lag4                Lag5
 Min.   :-4.92200   Min.   :-4.92200   Min.   :-4.92200
 1st Qu.:-0.64000   1st Qu.:-0.64000   1st Qu.:-0.64000
 Median : 0.03850   Median : 0.03850   Median : 0.03850
 Mean   : 0.00172   Mean   : 0.00164   Mean   : 0.00561
 3rd Qu.: 0.59675   3rd Qu.: 0.59675   3rd Qu.: 0.59700
 Max.   : 5.73300   Max.   : 5.73300   Max.   : 5.73300
     Volume           Today          Direction
 Min.   :0.356   Min.   :-4.92200   Down:602
 1st Qu.:1.257   1st Qu.:-0.63950   Up  :648
 Median :1.423   Median : 0.03850
 Mean   :1.478   Mean   : 0.00314
 3rd Qu.:1.642   3rd Qu.: 0.59675
 Max.   :3.152   Max.   : 5.73300
> pairs(Smarket)
```

The `cor()` function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. The first command below gives an error message because the `Direction` variable is qualitative.

```
> cor(Smarket)
Error in cor(Smarket) : 'x' must be numeric
> cor(Smarket[, -9])
```

```
          Year      Lag1      Lag2      Lag3      Lag4      Lag5
Year    1.0000   0.02970   0.03060   0.03319   0.03569   0.02979
Lag1    0.0297   1.00000  -0.02629  -0.01080  -0.00299  -0.00567
Lag2    0.0306  -0.02629   1.00000  -0.02590  -0.01085  -0.00356
Lag3    0.0332  -0.01080  -0.02590   1.00000  -0.02405  -0.01881
Lag4    0.0357  -0.00299  -0.01085  -0.02405   1.00000  -0.02708
Lag5    0.0298  -0.00567  -0.00356  -0.01881  -0.02708   1.00000
Volume  0.5390   0.04091  -0.04338  -0.04182  -0.04841  -0.02200
Today   0.0301  -0.02616  -0.01025  -0.00245  -0.00690  -0.03486
         Volume     Today
Year     0.5390   0.03010
Lag1     0.0409  -0.02616
Lag2    -0.0434  -0.01025
Lag3    -0.0418  -0.00245
Lag4    -0.0484  -0.00690
Lag5    -0.0220  -0.03486
Volume   1.0000   0.01459
Today    0.0146   1.00000
```

As one would expect, the correlations between the lag variables and today's returns are close to zero. In other words, there appears to be little correlation between today's returns and previous days' returns. The only substantial correlation is between `Year` and `Volume`. By plotting the data, which is ordered chronologically, we see that `Volume` is increasing over time. In other words, the average number of shares traded daily increased from 2001 to 2005.

```
> attach(Smarket)
> plot(Volume)
```

4.7.2 Logistic Regression

Next, we will fit a logistic regression model in order to predict `Direction` using `Lag1` through `Lag5` and `Volume`. The `glm()` function can be used to fit many types of generalized linear models, including logistic regression. The syntax of the `glm()` function is similar to that of `lm()`, except that we must pass in the argument `family = binomial` in order to tell `R` to run a logistic regression rather than some other type of generalized linear model.

`glm()`
generalized linear model

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

Call:
glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5
    + Volume, family = binomial, data = Smarket)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
```

```
 -1.45   -1.20    1.07    1.15    1.33

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12600    0.24074   -0.52     0.60
Lag1        -0.07307    0.05017   -1.46     0.15
Lag2        -0.04230    0.05009   -0.84     0.40
Lag3         0.01109    0.04994    0.22     0.82
Lag4         0.00936    0.04997    0.19     0.85
Lag5         0.01031    0.04951    0.21     0.83
Volume       0.13544    0.15836    0.86     0.39

(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: 1742

Number of Fisher Scoring iterations: 3
```

The smallest p-value here is associated with `Lag1`. The negative coefficient for this predictor suggests that if the market had a positive return yesterday, then it is less likely to go up today. However, at a value of 0.15, the p-value is still relatively large, and so there is no clear evidence of a real association between `Lag1` and `Direction`.

We use the `coef()` function in order to access just the coefficients for this fitted model. We can also use the `summary()` function to access particular aspects of the fitted model, such as the p-values for the coefficients.

```
> coef(glm.fits)
(Intercept)         Lag1         Lag2         Lag3         Lag4
   -0.12600     -0.07307     -0.04230      0.01109      0.00936
       Lag5       Volume
    0.01031      0.13544
> summary(glm.fits)$coef
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.12600     0.2407  -0.523    0.601
Lag1        -0.07307     0.0502  -1.457    0.145
Lag2        -0.04230     0.0501  -0.845    0.398
Lag3         0.01109     0.0499   0.222    0.824
Lag4         0.00936     0.0500   0.187    0.851
Lag5         0.01031     0.0495   0.208    0.835
Volume       0.13544     0.1584   0.855    0.392
> summary(glm.fits)$coef[, 4]
(Intercept)         Lag1         Lag2         Lag3         Lag4
      0.601        0.145        0.398        0.824        0.851
       Lag5       Volume
      0.835        0.392
```

The `predict()` function can be used to predict the probability that the market will go up, given values of the predictors. The `type = "response"` option tells `R` to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the logit. If no data set is supplied to the

predict() function, then the probabilities are computed for the training data that was used to fit the logistic regression model. Here we have printed only the first ten probabilities. We know that these values correspond to the probability of the market going up, rather than down, because the contrasts() function indicates that R has created a dummy variable with a 1 for Up.

```
> glm.probs <- predict(glm.fits, type = "response")
> glm.probs[1:10]
    1     2     3     4     5     6     7     8     9    10
0.507 0.481 0.481 0.515 0.511 0.507 0.493 0.509 0.518 0.489
> contrasts(Direction)
     Up
Down  0
Up    1
```

In order to make a prediction as to whether the market will go up or down on a particular day, we must convert these predicted probabilities into class labels, Up or Down. The following two commands create a vector of class predictions based on whether the predicted probability of a market increase is greater than or less than 0.5.

```
> glm.pred <- rep("Down", 1250)
> glm.pred[glm.probs > .5] = "Up"
```

The first command creates a vector of 1,250 Down elements. The second line transforms to Up all of the elements for which the predicted probability of a market increase exceeds 0.5. Given these predictions, the table() function can be used to produce a confusion matrix in order to determine how many observations were correctly or incorrectly classified.

table()

```
> table(glm.pred, Direction)
        Direction
glm.pred Down  Up
    Down  145 141
    Up    457 507
> (507 + 145) / 1250
[1] 0.5216
> mean(glm.pred == Direction)
[1] 0.5216
```

The diagonal elements of the confusion matrix indicate correct predictions, while the off-diagonals represent incorrect predictions. Hence our model correctly predicted that the market would go up on 507 days and that it would go down on 145 days, for a total of $507 + 145 = 652$ 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 predicted the movement of the market 52.2 % of the time.

At first glance, it appears that the logistic regression model is working a little better than random guessing. However, this result is misleading because we trained and tested the model on the same set of $1,250$ observations. In other words, $100\% - 52.2\% = 47.8\%$, is the *training* error

rate. As we have seen previously, the training error rate is often overly optimistic—it tends to underestimate the test error rate. In order to better assess the accuracy of the logistic regression model in this setting, we can fit the model using part of the data, and then examine how well it predicts the *held out* data. This will yield a more realistic error rate, in the sense that in practice we will be interested in our model's performance not on the data that we used to fit the model, but rather on days in the future for which the market's movements are unknown.

To implement this strategy, we will first create a vector corresponding to the observations from 2001 through 2004. We will then use this vector to create a held out data set 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 1,250 elements, corresponding to the observations in our data set. The elements of the vector that correspond to observations that occurred before 2005 are set to `TRUE`, whereas those that correspond to observations in 2005 are set to `FALSE`. The object `train` is a *Boolean* vector, since its elements are `TRUE` and `FALSE`. Boolean vectors can be used to obtain a subset of the rows or columns of a matrix. For instance, the command `Smarket[train, ]` would pick out a submatrix of the stock market data set, corresponding only to the dates before 2005, since those are the ones for which the elements of `train` are `TRUE`. The `!` symbol can be used to reverse all of the elements of a Boolean vector. That is, `!train` is a vector similar to `train`, except that the elements that are `TRUE` in `train` get swapped to `FALSE` in `!train`, and the elements that are `FALSE` in `train` get swapped to `TRUE` in `!train`. Therefore, `Smarket[!train, ]` yields a submatrix of the stock market data containing only the observations for which `train` is `FALSE`—that is, the observations with dates in 2005. The output above indicates that there are 252 such observations.

boolean

We now fit a logistic regression model using only the subset of the observations that correspond to dates before 2005, using the `subset` argument. We then obtain predicted probabilities of the stock market going up for each of the days in our test set—that is, for the days in 2005.

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

Notice that we have trained and tested our model on two completely separate data sets: training was performed using only the dates before 2005,

and testing was 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 > .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] 0.48
> mean(glm.pred != Direction.2005)
[1] 0.52
```

The `!=` notation means *not equal to*, and so the last command computes the test set error rate. The results are rather disappointing: the test error rate is 52 %, which is worse than random guessing! Of course this result is not all that surprising, given that one would not generally expect to be able to use previous days' returns to predict future market performance. (After all, if it were possible to do so, then the authors of this book would be out striking it rich rather than writing a statistics textbook.)

We recall that the logistic regression model had very underwhelming p-values associated with all of the predictors, and that the smallest p-value, though not very small, corresponded to `Lag1`. Perhaps by removing the variables that appear not to be helpful in predicting `Direction`, we can obtain a more effective model. After all, using predictors that have no relationship with the response tends to cause a deterioration in the test error rate (since such predictors cause an increase in variance without a corresponding decrease in bias), and so removing such predictors may in turn yield an improvement. Below we have refit the logistic regression using just `Lag1` and `Lag2`, which seemed to have the highest predictive power in the original logistic regression model.

```
> glm.fits <- glm(Direction ~ Lag1 + Lag2, data = Smarket,
    family = binomial, subset = train)
> glm.probs <- predict(glm.fits, Smarket.2005,
    type = "response")
> glm.pred <- rep("Down", 252)
> glm.pred[glm.probs > .5] <- "Up"
> table(glm.pred, Direction.2005)
        Direction.2005
glm.pred Down  Up
    Down   35  35
    Up     76 106
> mean(glm.pred == Direction.2005)
[1] 0.56
> 106 / (106 + 76)
[1] 0.582
```

Now the results appear to be a little better: 56% of the 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! Hence, in terms of overall error rate, the logistic regression method is no better than the naive approach. However, the confusion matrix shows that on days when logistic regression predicts an increase in the market, it has a 58% accuracy rate. This suggests a possible trading strategy of buying on days when the model predicts an increasing market, and avoiding trades on days when a decrease is predicted. Of course one would need to investigate more carefully whether this small improvement was real or just due to random chance.

Suppose that we want to predict the returns associated with particular values of `Lag1` and `Lag2`. In particular, we want to predict `Direction` on a day when `Lag1` and `Lag2` equal 1.2 and 1.1, respectively, and on a day when they equal 1.5 and −0.8. We do this using the `predict()` function.

```
> predict(glm.fits,
    newdata =
      data.frame(Lag1 = c(1.2, 1.5),  Lag2 = c(1.1, -0.8)),
    type = "response"
  )
       1         2
  0.4791    0.4961
```

Linear Discriminant Analysis

Now we will perform LDA on the `Smarket` data. In `R`, we fit an LDA model using the `lda()` function, which is part of the `MASS` library. Notice that the syntax for the `lda()` function is identical to that of `lm()`, and to that of `glm()` except for the absence of the `family` option. We fit the model using only the observations before 2005.

`lda()`

```
> library(MASS)
> lda.fit <- lda(Direction ~ Lag1 + Lag2, data = Smarket,
    subset = train)
> lda.fit
Call:
lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)

Prior probabilities of groups:
 Down    Up
0.492 0.508

Group means:
        Lag1    Lag2
Down  0.0428  0.0339
Up   -0.0395 -0.0313
```

```
Coefficients of linear discriminants:
        LD1
Lag1 -0.642
Lag2 -0.514
> plot(lda.fit)
```

The 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 days during which the market went down. It also provides the group means; these are the average of each predictor within each class, and are used by LDA as estimates of μ_k. These suggest that there is a tendency for the previous 2 days' returns to be negative on days when the market increases, and a tendency for the previous days' returns to be positive on days when the market declines. The *coefficients of linear discriminants* output provides the linear combination of `Lag1` and `Lag2` that are used to form the LDA decision rule. In other words, these are the multipliers of the elements of $X = x$ in (4.24). 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 a market decline.

The `plot()` function produces plots of the *linear discriminants*, obtained by computing $-0.642 \times$ `Lag1` $- 0.514 \times$ `Lag2` for each of the training observations. The `Up` and `Down` observations are displayed separately.

The `predict()` function returns a list with three elements. The first element, `class`, contains LDA's 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 the kth class, computed from (4.15). Finally, `x` contains the linear discriminants, described earlier.

```
> lda.pred <- predict(lda.fit, Smarket.2005)
> names(lda.pred)
[1] "class"     "posterior" "x"
```

As we observed in Section 4.5, the LDA and logistic regression predictions are almost identical.

```
> lda.class <- lda.pred$class
> table(lda.class, Direction.2005)

         Direction.2005
lda.pred Down  Up
    Down   35  35
    Up     76 106
> mean(lda.class == Direction.2005)
[1] 0.56
```

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

```
> sum(lda.pred$posterior[, 1] >= .5)
[1] 70
```

```
> sum(lda.pred$posterior[, 1] < .5)
[1] 182
```

Notice that the posterior probability output by the model corresponds to the probability that the market will *decrease*:

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

If we wanted to use a posterior probability threshold other than 50 % in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a market decrease only if we are very certain that the market will indeed decrease on that day—say, if the posterior probability is at least 90 %.

```
> sum(lda.pred$posterior[, 1] > .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 %.

Quadratic Discriminant Analysis

We will now fit a QDA model to the `Smarket` data. QDA is implemented in `R` using the `qda()` function, which is also part of the `MASS` library. The syntax is identical to that of `lda()`.

`qda()`

```
> 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
0.492 0.508

Group means:
        Lag1    Lag2
Down  0.0428  0.0339
Up   -0.0395 -0.0313
```

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.

```
> qda.class <- predict(qda.fit, Smarket.2005)$class
> table(qda.class, Direction.2005)
        Direction.2005
qda.class Down  Up
    Down   30  20
```

```
    Up      81 121
> mean(qda.class == Direction.2005)
[1] 0.599
```

Interestingly, the QDA predictions are accurate 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. However, we recommend evaluating this method's performance on a larger test set before betting that this approach will consistently beat the market!

Naive Bayes

Next, we fit a naive Bayes model to the `Smarket` data. Naive Bayes is implemented in `R` using the `naiveBayes()` function, which is part of the `e1071` library. The syntax is identical to that of `lda()` and `qda()`. By default, this implementation of the naive Bayes classifier models each quantitative feature using a Gaussian distribution. However, a kernel density method can also be used to estimate the distributions.

naiveBayes()

```
> library(e1071)
> nb.fit <- naiveBayes(Direction ~ Lag1 + Lag2, data = Smarket,
    subset = train)
> nb.fit
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
 Down    Up
0.492 0.508

Conditional probabilities:
      Lag1
Y          [,1] [,2]
  Down  0.0428 1.23
  Up   -0.0395 1.23
      Lag2
Y          [,1] [,2]
  Down  0.0339 1.24
  Up   -0.0313 1.22
```

The output contains the estimated mean and standard deviation for each variable in each class. For example, the mean for `Lag1` is 0.0428 for `Direction=Down`, and the standard deviation is 1.23. We can easily verify this:

```
> mean(Lag1[train][Direction[train] == "Down"])
[1] 0.0428
> sd(Lag1[train][Direction[train] == "Down"])
[1] 1.23
```

The `predict()` function is straightforward.

```
> nb.class <- predict(nb.fit, Smarket.2005)
> table(nb.class, Direction.2005)
        Direction.2005
nb.class Down  Up
    Down   28  20
    Up     83 121
> mean(nb.class == Direction.2005)
[1] 0.591
```

Naive Bayes performs very well on this data, with accurate predictions over 59% of the time. This is slightly worse than QDA, but much better than LDA.

The `predict()` function can also generate estimates of the probability that each observation belongs to a particular class.

```
> nb.preds <- predict(nb.fit, Smarket.2005, type = "raw")
> nb.preds[1:5, ]
       Down    Up
 [1,] 0.487 0.513
 [2,] 0.476 0.524
 [3,] 0.465 0.535
 [4,] 0.475 0.525
 [5,] 0.490 0.510
```

K-Nearest Neighbors

We will now perform KNN using the `knn()` function, which is part of the `class` library. This function works rather differently from the other model-fitting functions that we have encountered thus far. Rather than a two-step approach in which we first fit the model and then we use the model to make predictions, `knn()` forms predictions using a single command. The function requires four inputs.

`knn()`

1. A matrix containing the predictors associated with the training data, labeled `train.X` below.
2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled `test.X` below.
3. A vector containing the class labels for the training observations, labeled `train.Direction` below.
4. A value for K, the number of nearest neighbors to be used by the classifier.

We use the `cbind()` function, short for *column bind*, to bind the `Lag1` and `Lag2` variables together into two matrices, one for the training set and the other for the test set.

`cbind()`

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

Now the `knn()` function can be used to predict the market's movement for the dates in 2005. We set a random seed before we apply `knn()` because if several observations are tied as nearest neighbors, then `R` will randomly break the tie. Therefore, a seed must be set in order to ensure reproducibility of results.

```
> set.seed(1)
> knn.pred <- knn(train.X, test.X, train.Direction, k = 1)
> table(knn.pred, Direction.2005)
        Direction.2005
knn.pred Down Up
    Down   43 58
    Up     68 83
> (83 + 43) / 252
[1] 0.5
```

The results 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. Below, we repeat the analysis using $K = 3$.

```
> knn.pred <- knn(train.X, test.X, train.Direction, k = 3)
> table(knn.pred, Direction.2005)
        Direction.2005
knn.pred Down Up
    Down   48 54
    Up     63 87
> mean(knn.pred == Direction.2005)
[1] 0.536
```

The results have improved slightly. But increasing K further turns out to provide no further improvements. It appears that for this data, QDA provides the best results of the methods that we have examined so far.

KNN does not perform well on the `Smarket` data but it does often provide impressive results. As an example we will apply the KNN approach to the `Caravan` data set, which is part of the `ISLR2` library. This data set includes 85 predictors that measure demographic characteristics for 5,822 individuals. The response variable is `Purchase`, which indicates whether or not a given individual purchases a caravan insurance policy. In this data set, only 6 % of people purchased caravan insurance.

```
> dim(Caravan)
[1] 5822   86
```

```
> attach(Caravan)
> summary(Purchase)
  No  Yes
5474  348
> 348 / 5822
[1] 0.0598
```

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Variables that are on a large scale will have a much larger effect on the *distance* between the observations, and hence on the KNN classifier, than variables that are on a small scale. For instance, imagine a data set that contains two variables, `salary` and `age` (measured in dollars and years, respectively). As far as KNN is concerned, a difference of \$1,000 in salary is enormous compared to a difference of 50 years in age. Consequently, `salary` will drive the KNN classification results, and `age` will have almost no effect. This is contrary to our intuition that a salary difference of \$1,000 is quite small compared to an age difference of 50 years. Furthermore, the importance of scale to the KNN classifier leads to another issue: if we measured `salary` in Japanese yen, or if we measured `age` in minutes, then we'd get quite different classification results from what we get if these two variables are measured in dollars and years.

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. The `scale()` function does just this. In standardizing the data, we exclude column 86, because that is the qualitative `Purchase` variable.

standardize

`scale()`

```
> standardized.X <- scale(Caravan[, -86])
> var(Caravan[, 1])
[1] 165
> var(Caravan[, 2])
[1] 0.165
> var(standardized.X[, 1])
[1] 1
> var(standardized.X[, 2])
[1] 1
```

Now every column of `standardized.X` has a standard deviation of one and a mean of zero.

We now split the observations into a test set, containing the first 1,000 observations, and a training set, containing the remaining observations. We fit a KNN model on the training data using $K = 1$, and evaluate its performance on the test data.

```
> 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
```

The vector `test` is numeric, with values from 1 through 1,000. Typing `standardized.X[test, ]` yields the submatrix of the data containing the observations whose indices range from 1 to 1,000, whereas typing `standardized.X[-test, ]` yields the submatrix containing the observations whose indices do *not* range from 1 to 1,000. The KNN error rate on the 1,000 test observations is just under 12 %. At first glance, this may appear to be fairly good. However, since only 6 % of customers purchased insurance, we could get the error rate down to 6 % by always predicting `No` regardless of the values of the predictors!

Suppose that there is some non-trivial cost to trying to sell insurance to a given individual. For instance, perhaps a salesperson must visit each potential customer. If the company tries to sell insurance to a random selection of customers, then the success rate will be only 6 %, which may be far too low given the costs involved. Instead, the company would like to try to sell insurance only to customers who are likely to buy it. So the overall error rate is not of interest. Instead, the fraction of individuals that are correctly predicted to buy insurance is of interest.

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 such customers, 9, or 11.7 %, actually do purchase insurance. This is double the rate that one would obtain from random guessing.

```
> table(knn.pred, test.Y)
        test.Y
knn.pred  No Yes
     No  873  50
     Yes  68   9
> 9 / (68 + 9)
[1] 0.117
```

Using $K = 3$, the success rate increases to 19 %, and with $K = 5$ the rate is 26.7 %. This is over four times the rate that results from random guessing. It appears that KNN is finding some real patterns in a difficult data set!

```
> 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] 0.192
> 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] 0.267
```

However, while this strategy is cost-effective, it is worth noting that only 15 customers are predicted to purchase insurance using KNN with $K = 5$. In practice, the insurance company may wish to expend resources on convincing more than just 15 potential customers to buy insurance.

As a comparison, we can also fit a logistic regression model to the data. If we use 0.5 as the predicted probability cut-off for the classifier, then we have a problem: only seven of the test observations are predicted to purchase insurance. Even worse, we are wrong about all of these! However, we are not required to use a cut-off of 0.5. If we instead predict a purchase any time the predicted probability of purchase exceeds 0.25, we get much better results: we predict that 33 people will purchase insurance, and we are correct for about 33 % of these people. This is over five times better than random guessing!

```
> glm.fits <- glm(Purchase ~ ., data = Caravan,
    family = binomial, subset = -test)
Warning message:
glm.fits: fitted probabilities numerically 0 or 1 occurred
> glm.probs <- predict(glm.fits, Caravan[test, ],
    type = "response")
> glm.pred <- rep("No", 1000)
> glm.pred[glm.probs > .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 > .25] <- "Yes"
> table(glm.pred, test.Y)
        test.Y
glm.pred  No Yes
     No  919  48
     Yes  22  11
> 11 / (22 + 11)
[1] 0.333
```

4.7.7 Poisson Regression

Finally, we fit a Poisson regression model to the `Bikeshare` data set, which measures the number of bike rentals (`bikers`) per hour in Washington, DC. The data can be found in the `ISLR2` library.

```
> attach(Bikeshare)
> dim(Bikeshare)
[1] 8645   15
> names(Bikeshare)
 [1] "season"     "mnth"       "day"        "hr"
 [5] "holiday"    "weekday"    "workingday" "weathersit"
 [9] "temp"       "atemp"      "hum"        "windspeed"
[13] "casual"     "registered" "bikers"
```

We begin by fitting a least squares linear regression model to the data.

```
> mod.lm <- lm(
    bikers ~ mnth + hr + workingday + temp + weathersit,
    data = Bikeshare
  )
> summary(mod.lm)
Call:
lm(formula = bikers ~ mnth + hr + workingday + temp +
  weathersit, data = Bikeshare)

Residuals:
    Min      1Q  Median      3Q     Max
-299.00  -45.70   -6.23   41.08  425.29

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -68.632      5.307 -12.932  < 2e-16 ***
mnthFeb        6.845      4.287   1.597 0.110398
mnthMarch     16.551      4.301   3.848 0.000120 ***
mnthApril     41.425      4.972   8.331  < 2e-16 ***
mnthMay       72.557      5.641  12.862  < 2e-16 ***
```

Due to space constraints, we truncate the output of `summary(mod.lm)`. In `mod.lm`, the first level of `hr` (0) and `mnth` (Jan) are treated as the baseline values, and so no coefficient estimates are provided for them: implicitly, their coefficient estimates are zero, and all other levels are measured relative to these baselines. For example, the Feb coefficient of 6.845 signifies that, holding all other variables constant, there are on average about 7 more riders in February than in January. Similarly there are about 16.5 more riders in March than in January.

The results seen in Section 4.6.1 used a slightly different coding of the variables `hr` and `mnth`, as follows:

```
> contrasts(Bikeshare$hr) = contr.sum(24)
> contrasts(Bikeshare$mnth) = contr.sum(12)
> mod.lm2 <- lm(
    bikers ~ mnth + hr + workingday + temp + weathersit,
    data = Bikeshare
  )
> summary(mod.lm2)
Call:
lm(formula = bikers ~ mnth + hr + workingday + temp +
  weathersit, data = Bikeshare)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-299.00  -45.70   -6.23   41.08  425.29

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)             73.597      5.132  14.340  < 2e-16 ***
mnth1                  -46.087      4.086 -11.281  < 2e-16 ***
mnth2                  -39.242      3.539 -11.088  < 2e-16 ***
mnth3                  -29.536      3.155  -9.361  < 2e-16 ***
mnth4                   -4.662      2.741  -1.701  0.08895 .
```

What is the difference between the two codings? In `mod.lm2`, a coefficient estimate is reported for all but the last level of `hr` and `mnth`. Importantly, in `mod.lm2`, the coefficient estimate for the last level of `mnth` is not zero: instead, it equals the *negative of the sum of the coefficient estimates for all of the other levels*. Similarly, in `mod.lm2`, the coefficient estimate for the last level of `hr` is the negative of the sum of the coefficient estimates for all of the other levels. This means that the coefficients of `hr` and `mnth` in `mod.lm2` will always sum to zero, and can be interpreted as the difference from the mean level. For example, the coefficient for January of -46.087 indicates that, holding all other variables constant, there are typically 46 fewer riders in January relative to the yearly average.

It is important to realize that the choice of coding really does not matter, provided that we interpret the model output correctly in light of the coding used. For example, we see that the predictions from the linear model are the same regardless of coding:

```
> sum((predict(mod.lm) - predict(mod.lm2))^2)
[1] 1.426e-18
```

The sum of squared differences is zero. We can also see this using the `all.equal()` function:

`all.equal()`

```
> all.equal(predict(mod.lm), predict(mod.lm2))
```

To reproduce the left-hand side of Figure 4.13, we must first obtain the coefficient estimates associated with `mnth`. The coefficients for January through November can be obtained directly from the `mod.lm2` object. The coefficient for December must be explicitly computed as the negative sum of all the other months.

```
> coef.months <- c(coef(mod.lm2)[2:12],
    -sum(coef(mod.lm2)[2:12]))
```

To make the plot, we manually label the x-axis with the names of the months.

```
> plot(coef.months, xlab = "Month", ylab = "Coefficient",
    xaxt = "n", col = "blue", pch = 19, type = "o")
```

```
> axis(side = 1, at = 1:12, labels = c("J", "F", "M", "A",
    "M", "J", "J", "A", "S", "O", "N", "D"))
```

Reproducing the right-hand side of Figure 4.13 follows a similar process.

```
> coef.hours <- c(coef(mod.lm2)[13:35],
    -sum(coef(mod.lm2)[13:35]))
> plot(coef.hours, xlab = "Hour", ylab = "Coefficient",
    col = "blue", pch = 19, type = "o")
```

Now, we consider instead fitting a Poisson regression model to the `Bikeshare` data. Very little changes, except that we now use the function `glm()` with the argument `family = poisson` to specify that we wish to fit a Poisson regression model:

```
> mod.pois <- glm(
    bikers ~ mnth + hr + workingday + temp + weathersit,
    data = Bikeshare, family = poisson
  )
> summary(mod.pois)
Call:
glm(formula = bikers ~ mnth + hr + workingday + temp +
  weathersit, family = poisson, data = Bikeshare)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-20.7574   -3.3441   -0.6549    2.6999   21.9628

Coefficients:
             Estimate Std. Error  z value Pr(>|z|)
(Intercept)  4.118245   0.006021  683.964  < 2e-16 ***
mnth1       -0.670170   0.005907 -113.445  < 2e-16 ***
mnth2       -0.444124   0.004860  -91.379  < 2e-16 ***
mnth3       -0.293733   0.004144  -70.886  < 2e-16 ***
mnth4        0.021523   0.003125    6.888 5.66e-12 ***
```

We can plot the coefficients associated with `mnth` and `hr`, in order to reproduce Figure 4.15:

```
> coef.mnth <- c(coef(mod.pois)[2:12],
    -sum(coef(mod.pois)[2:12]))
> plot(coef.mnth, xlab = "Month", ylab = "Coefficient",
    xaxt = "n", col = "blue", pch = 19, type = "o")
> axis(side = 1, at = 1:12, labels = c("J", "F", "M", "A", "M",
    "J", "J", "A", "S", "O", "N", "D"))
> coef.hours <- c(coef(mod.pois)[13:35],
    -sum(coef(mod.pois)[13:35]))
> plot(coef.hours, xlab = "Hour", ylab = "Coefficient",
    col = "blue", pch = 19, type = "o")
```

We can once again use the `predict()` function to obtain the fitted values (predictions) from this Poisson regression model. However, we must use the argument `type = "response"` to specify that we want `R` to output $\exp(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p)$ rather than $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \ldots + \hat{\beta}_p X_p$, which it will output by default.

```
> plot(predict(mod.lm2), predict(mod.pois, type = "response"))
> abline(0, 1, col = 2, lwd = 3)
```

The predictions from the Poisson regression model are correlated with those from the linear model; however, the former are non-negative. As a result the Poisson regression predictions tend to be larger than those from the linear model for either very low or very high levels of ridership.

In this section, we used the `glm()` function with the argument `family = poisson` in order to perform Poisson regression. Earlier in this lab we used the `glm()` function with `family = binomial` to perform logistic regression. Other choices for the `family` argument can be used to fit other types of GLMs. For instance, `family = Gamma` fits a gamma regression model.