Logistic Regression, Linear and Quadratic Discriminant Analysis, and Generalized Additive Models and K-Cross Fold validation

Chris Schmidt

2019

Classification Methods Exploration

When we have a qualitative response variable we are more likely to build an effective model using a classifier than using a regression with different levels or dummy variables.

Perform Logistic Regression, Linear Discriminate Analysis, Quadratic Discriminant Analysis, KNN and GAM on the S&P 500 data set, Smarket, found in the ISLR package.

Use 10-fold cross validation to decide which model performs the best for the Smarket data.

The gam() model *b3* performs the best as can be shown in the output below.

Build and run lda and qda models and logistic regression, using knn with k=3, knn k=4, and b, b2 and b3.

Note to reader: The output of MAE values is cumulative in the document. The rightmost value in the output for each model is the output for that particular model. Use the set.seed(1) function for reproducible results and split the data into test and train set.

```
set.seed(1)
train = sample (1250, 1250)
```

Examine the data set SMarket

The *Smarket* data set is a collection of daily percentage returns from 2001 to 2005 with columns representing lagged percentage returns for previous days 1 to 5 days from the current day along with current day percentage return, volume traded for the day and the direction that the market finished, up or down from the previous day.

An examination of the data set using the *head()* and *tail()* function and the *str()* function gives a flavor of the data.

```
head(Smarket)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

```
tail(Smarket)
```

```
##      Year   Lag1   Lag2   Lag3   Lag4   Lag5  Volume  Today Direction
## 1245 2005  0.252 -0.024 -0.584 -0.285 -0.141 2.06517  0.422        Up
## 1246 2005  0.422  0.252 -0.024 -0.584 -0.285 1.88850  0.043        Up
## 1247 2005  0.043  0.422  0.252 -0.024 -0.584 1.28581 -0.955      Down
## 1248 2005 -0.955  0.043  0.422  0.252 -0.024 1.54047  0.130        Up
## 1249 2005  0.130 -0.955  0.043  0.422  0.252 1.42236 -0.298      Down
## 1250 2005 -0.298  0.130 -0.955  0.043  0.422 1.38254 -0.489      Down
```

```
str(Smarket)
```

```
## 'data.frame':    1250 obs. of  9 variables:
##  $ Year     : num  2001 2001 2001 2001 2001 ...
##  $ Lag1     : num  0.381 0.959 1.032 -0.623 0.614 ...
##  $ Lag2     : num  -0.192 0.381 0.959 1.032 -0.623 ...
##  $ Lag3     : num  -2.624 -0.192 0.381 0.959 1.032 ...
##  $ Lag4     : num  -1.055 -2.624 -0.192 0.381 0.959 ...
##  $ Lag5     : num  5.01 -1.055 -2.624 -0.192 0.381 ...
##  $ Volume   : num  1.19 1.3 1.41 1.28 1.21 ...
##  $ Today    : num  0.959 1.032 -0.623 0.614 0.213 ...
##  $ Direction: Factor w/ 2 levels "Down","Up": 2 2 1 2 2 2 1 2 2 2 ...
```

Logistic Regression with 10-Fold Cross-Validation

Logistic regression models the probability that the response variable belongs to a specific category where $Y = Pr(X)$ where $Pr(X) = Pr(Y = k|X = x)$ is the conditional probability of the response Y given the predictor(s) X. Because this is a probability we need outputs bounded from 0 to 1 for all values of X. In simple logistic regression we use the logistic function for this purpose where

$$Pr(X) = \frac{e^{\beta_0+\beta_1 X_1}}{1 + e^{\beta_0+\beta_1 X_1}}$$

To fit this model we use the maximum likelihood function where we seek to maximize the likelihood function to find the estimates of our predictor coefficients

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y'_i} (1 - p(x'_i))$$

where the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize this likelihood.

We find the *odds* by manipulating $Pr(X)$ to get

$$log\Big(\frac{p(X)}{1 - p(X)}\Big) = \beta_0 + \beta_1 X_1$$

where the left hand side is the logit function or log-odds function.

Multiple logistic regression simply expands upon this model as in multiple linear regression.

$$Pr(X) = \frac{e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}}$$

and the logit function can be generalized as

$$log\Big(\frac{p(X)}{1 - p(X)}\Big) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$$

And, as with the simple logistic regression, we use the maximum likelihood function to estimate $\beta_0, \beta_1, ..., \beta_p$

Logistic Regression with k-Fold Cross Validation

k-fold cross validation is a resampling technique that randomly divides the data set into k groups of observations or roughly equal size. The first fold is the validation set and the model method is run on the remaining k-1 folds of data. The MSE_1 (mean squared error) is computed on the observations on the held-out fold and the procedure is repeated k-times with a different held-out fold as the validation set. This generates k estimates of the test error, $MSE_1, MSE_2, ..., MSE_k$. The k-fold CV (cross validation) estimate is then generated by averaging these values.

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

There is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically a k-value of 5 or 10 is used as these have been shown empirically to generate test-error rate estimates that do not have either high bias or high variance.

The **cv.glm()** function can be used to implement a k-fold Cross Validation. Below, we use k = 10, a common choice for k, on the **Smarket** data set with the S&P 500 stock market index data. We set a random seed for reproducibility above and initialize a vector in which we will store the CV errors corresponding to the polynomial fits of orders one to ten.

Using the **glm()** function and the *family = binomial* argument for the logit function we create a logistic regression model where the response is the direction and the predictors are Lag1 and Lag2. We create a call to generate the mean absolute error (MAE)

```
MAE=c()
mae=c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  m_glm <- glm(Direction ~ Lag1 + Lag2, data = train_data, family = binomial)
  m_pred <- predict(m_glm, test_data, type = "response")
  glm.pred <- rep("Down", 1125)
  glm.pred[m_pred > 0.5] = "Up"

  mae <- c(mae, mean(abs(as.integer(glm.pred=="Up")-as.integer(test_data$Direction == "Up"))))
}
MAE=c(MAE, mean(mae))
MAE
```

```
## [1] 0.4872
```

3. Linear Discriminant Analysis (LDA) with 10-Fold Cross-Validation

There are several reasons why LDA is a better option than logistic regression:

- If there are well-separated classes the parameter estimates for logistic regression can be unstable,
- If we have a small sample size and an approximately normal distribution of X in each class, the LDA is more stable than logistic regression,
- When there are more than two response classes, LDA is a more popular method.

LDA uses Bayes Theorem for classification which we can explain by noting that if we have K classes and we want to classify the qualitative response variable Y where there are K possible distinct and unordered values we use the following method.

Let π_k be the *prior* probability that a given randomly chosen observation comes from the kth class. Let $f_k(x) \equiv Pr(X = x|Y = k)$ be the density function of X for an observation from the kth class. $f_k(x)$ is relatively large if there is a high probability that an observation in the kth class has $X \approx x$ and $f_k(x)$ is relatively small if it is very unlikely that an observation in the kth class has $X \approx x$.

Bayes Theorem states that

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Letting $p_k(x) = Pr(Y = k|X)$ we see that we can simply plug in estimates of π_k and $f_k(X)$ into the formula which can be generated with the software that then takes care of the rest. We refer to $p_k(x)$ as the *posterior* probability that an observation $X = x$ belongs to the kth class given the predictor value for that observation.

Estimating π_k is easy if we have a random sample of Y's from the population but estimating $f_k(X)$ is more difficult. However, if we have an estimate for $f_k(x)$ then we can build a classifier that approximates the Bayes classifier.

By assuming that $X = (X_1, X_2, ..., X_p)$ is drawn from a multivariate Gaussian distribution, with a class specific mean vector and a common covariance matrix which we can write as $X \sim N(\mu, \Sigma)$ to indicate that p has a multivariate Gaussian distribution. $E(X) = \mu$ is the mean of the X vector with p components and $Cov(X) = \Sigma$ is the $p \times p$ covariance matrix of X.

Formally, the multivariate Gaussian density is

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} exp\Big(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\Big)$$

plugging the density function for the kth class, $f_k(X = x)$ into

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

and applying some algebra we see that the Bayes classifier assigns $X = x$ to the class for which

$$\delta_k(x) = x^T \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k + \log \pi_k$$

is the largest. The Bayes decision boundaries represent the set of values x for which $\delta_k(x) = \delta_l(x)$. In other words for which

$$x^T \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1}\mu_k = x^T \Sigma^{-1}\mu_l - \frac{1}{2}\mu_l^T \Sigma^{-1}\mu_l, \text{ for } k \neq l$$

The log π_k term has disappeared because each of the three classes has the same number of training observations, thus π_k is the same for each class. To estimate $\mu_1, ..., \mu_k, \pi_1, ..., \pi_k$ and Σ we use similar conventions for the case where $p = 1$

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

$$\hat{\pi}_k = \frac{n_k}{n}$$

The estimates are plugged into

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k$$

in order to assign a new observation $X = x$ to the class for which $\hat{\delta}_k(x)$ is the largest. This is a linear function of x so the LDA decision rule depends on x only through a linear combination of the elements.

The output for LDA often uses a *confusion matrix* to display the True status versus the predicted status for the qualitative response variable. Elements on the diagonal represent correct predictions and off-diagonal represent misclassifications.

This introduces the terms *sensitivity* and *specificity* to characterize the performance of a classifier. We call the true positive rate the *sensitivity* and the false positive rate, 1 - *specificity*

The Bayes classifier works by assigning an observation to the class for which the posterior probability $p_k(X)$ is the largest. If we have two classes, say "*wrong*" and "*right*" we assign the observation to the "*wrong*" class if

$$Pr(\text{ wrong} = \text{Yes } |X = x) > 0.5$$

This creates a threshold of 50% for the *posterior* probability of default in order to assign an observation to the "*wrong*" class. If we have concerns about mislabeling the prediction for the "*wrong*" class we can lower this threshold. We could, for example, label an observation with a posterior probability of being in the "*wrong*" class about 20% to the "*wrong*" class

$$Pr(\text{ wrong} = \text{Yes } |X = x) > 0.2$$

We use the *receiver operating characteristics* curve, ROC curve, to simultaneously display the two types of errors for all possible thresholds where the overall performance of the classifier is given by the area under the ROC curve (the AUC) where the larger the percentage, the better the classifier.

Run the k-fold cross validation with k = 10.

```
#MAE=c()
mae=c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  m_lda <- lda(Direction ~ Lag1 + Lag2, data = train_data)
  m_pred <- predict(m_lda, test_data)
  mae <- c(mae, mean(abs( as.integer(m_pred$class == 'Up') - as.integer(train_data$Direction == 'Up'))))
}

MAE=c(MAE, mean(mae))
MAE
```

```
## [1] 0.4872000 0.4826667
```

4. Quadratic Discriminant Analysis with 10-fold Cross-Validation

QDA assumes the observations come from a Gaussian distribution like LDA but QDA assumes each class has its own covariance matrix. QDA assumes that an observation from the kth class is of the form $X \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the kth class.

In this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\Sigma_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \log|\Sigma_k| + \log\pi_k\end{aligned}$$

is the largest. The QDA classifier plugs estimates for Σ_k, μ_k, and π_k into the equation above and then assigning $X = x$ to the class for which the quantity is largest. Since x appears as a quadratic equation, we call this classifier QDA.

The reasons for choosing LDA over QDA or vice versa have to do with the bias-variance tradeoff. When there are p predictors, estimating the covariance matrix requires estimating $p(p+1)/2$ parameters. QDA estimates a separate covariance matrix for each class for a total of $Kp(p+1)/2$ parameters. If we have 50 predictors this is some multiple of $(50 * 51)/2 = 1275$ for a significant jump in predictors. The LDA model assumes the K classes share a common covariance matrix so that the LDA model becomes linear in x so that there are kp linear coefficients to estimate. Thus LDA is a less flexible classifer than QDA and has a significantly lower variance. The tradeoff comes from noting that if the LDA assumption of a common covariance matrix is incorrect then high bias can be an issue.

Run the k-fold cross validation with k = 10.

```
#MAE <- c()
mae <- c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  m_qda <- qda(Direction ~ Lag1 + Lag2, data = train_data)
  m_pred <- predict(m_qda, test_data)
  mae <- c(mae, mean(abs( as.integer(m_pred$class == 'Up') - as.integer(train_data$Direction == 'Up'))))
}

MAE=c(MAE, mean(mae))
MAE
```

```
## [1] 0.4872000 0.4826667 0.4868444
```

5. 10-Fold Cross-Validation for KNN

```
#MAE <- c()
mae <- c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  train.X = cbind(train_data$Lag1, train_data$Lag2)
  test.X = cbind(test_data$Lag1, test_data$Lag2)
  train.Direction = train_data$Direction
  test.Direction = test_data$Direction
  knn.pred = knn(train.X, test.X, train.Direction, k = 3)
  mae = c(mae,mean(knn.pred != test.Direction))
}

MAE=c(MAE, mean(mae))
MAE
```

K = 3

```
## [1] 0.4872000 0.4826667 0.4868444 0.5152000
```

K = 4

```
#MAE <- c()
mae <- c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  train.X = cbind(train_data$Lag1, train_data$Lag2)
  test.X = cbind(test_data$Lag1, test_data$Lag2)
  train.Direction = train_data$Direction
  test.Direction = test_data$Direction
  knn.pred = knn(train.X, test.X, train.Direction, k = 4)
  mae = c(mae,mean(knn.pred != test_data$Direction))
}

MAE = c(MAE, mean(mae))
MAE_knn4 = mean(mae)
MAE_knn4
```

```
## [1] 0.504
```

```
#MAE
```

6. 10-Fold Cross-Validation with GAM.

```
#MAE=c()
mae=c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  #train <- (train_data$Direction == "Up")
  b <- gam(Direction ~ s(Lag1) + s(Lag2),
           data = Smarket, family = binomial)
  pred_b = predict(b, newdata = test_data, type = "response")
  pred_b_ud = rep("Down", 125)
  pred_b_ud[pred_b > 0.5] = "Up"

  mae <- c(mae, mean(abs(as.integer(pred_b_ud=="Up")
                         -as.integer(test_data$Direction == "Up"))))
}

MAE=c(MAE, mean(mae))
MAE
```

Model B

```
## [1] 0.4872000 0.4826667 0.4868444 0.5152000 0.5040000 0.4600000
```

```
#MAE=c()
mae=c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  #train <- (train_data$Direction == "Up")
  b2 <- gam(Direction ~ ti(Lag1) + ti(Lag2) + ti(Lag1, Lag2),
            data = Smarket, family = binomial)
  pred_b = predict(b2, newdata = test_data, type = "response")
  pred_b_ud = rep("Down", 125)
  pred_b_ud[pred_b > 0.5] = "Up"

  mae <- c(mae, mean(abs(as.integer(pred_b_ud=="Up")
                         -as.integer(test_data$Direction == "Up"))))
}

MAE = c(MAE, mean(mae))
MAE
```

Model B2

```
## [1] 0.4872000 0.4826667 0.4868444 0.5152000 0.5040000 0.4600000 0.4616000
```

```
#MAE=c()
mae=c()

for (i in 1:10){
  train_data <- Smarket[-((i*125-124):(i*125)), ]
  test_data <- Smarket[((i*125-124):(i*125)), ]
  #train <- (train_data$Direction == "Up")
  b3 <- gam(Direction ~ te(Lag1, Lag2),
            data = Smarket, family = binomial)
  pred_b = predict(b3, newdata = test_data,
                   type = "response")
  pred_b_ud = rep("Down", 125)
  pred_b_ud[pred_b > 0.5] = "Up"

  mae <- c(mae, mean(abs(as.integer(pred_b_ud=="Up")
                         -as.integer(test_data$Direction == "Up"))))
}

MAE=c(MAE, mean(mae))
MAE
```

Model B3

```
## [1] 0.4872000 0.4826667 0.4868444 0.5152000 0.5040000 0.4600000 0.4616000
## [8] 0.4528000
```

Visualization of Best Model as Identified using 10-Fold Cross-Validation

```
plot(MAE, ylim=c(.45,.52) , col='green', type='p',
     pch=19, main='cross validation for the Smarket data',
     ylab='mean absolute error')

text(MAE, c('lda', 'qda', 'glm', 'knn3', 'knn4',
            'b', 'b2', 'b3'), pos=c(3,3), cex=1.25, col="purple")
```