4.6 Lab:Logistic Regression, LDA, QDA, and KNN

Zongyi Liu

2023-05-20

4.6 Logistic Regression, LDA, QDA, and KNN

4.6.1 The Stock Market Data

```
library(ISLR)
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
```

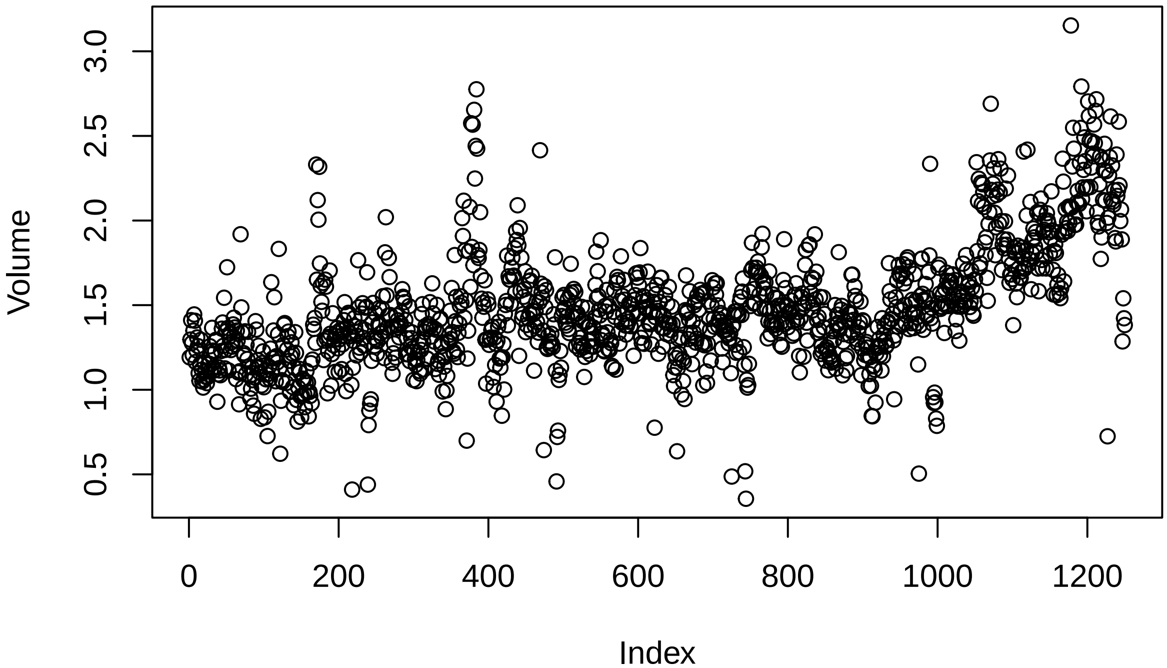
`cor()` can be used to produce the summary of all of the pairwise correlations among predictors in the data set

```
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 plot the `Volume` data to see that it is increasing over year

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



4.6.2 Logistic Regression

In this case, we will use the `glm()` function to do the task. glm stands for *generalized linear model*.

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

```
##
## 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
```

Then we can select its coefficients with `coef()` function or `$coef`

```
coef(glomd)
```

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

```
summary(glomd)$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
```

The `predict()` function can be used to predict the probability that the market will go up.

```
glomd_probability = predict(glomd, type = "response")
glomd_probability[1:20]
```

```
##         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        11        12        13        14        15        16
## 0.5176135 0.4888378 0.4965211 0.5197834 0.5183031 0.4963852 0.4864892 0.5153660
##        17        18        19        20
## 0.5053976 0.5319322 0.5167163 0.4983272
```

```
contrasts(Direction) # A dummy variable that goes up
```

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

To predict whether the market will go up or down

```
glmod_pred = rep("Down", 1250)
glmod_pred[glomd_probability >.5]="Up"
table(glmod_pred,Direction)
```

```
##           Direction
## glmod_pred Down  Up
##       Down  145 141
##       Up    457 507
```

```
mean(glmod_pred==Direction)
```

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

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.

From the results, we can see that the market would go up on 507 days and down for 145 days, adding up to totally 652 days correctly predicted. With the `mean()` function, we can see that the correctly prediction rate is 52.2%.

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

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

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

Here, the `train` is a vector of 1,250 elements, corresponding to the observations in the data set; it is also a Boolean value, for which occurred before 2005 are set to be `TRUE`, but those values occurred after 2005 are set to be `FALSE`.

Then we can fit out a logistic regression model using only the subset of the observations that correspond to dates before 2005.

The `subset` argument can help to obtain predicted probabilities of the stock market going up for each days in the test set.

```
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")
```

Here we trained and tested the model on two completely separated data sets. 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
```

And get the mean of prediction correct rate:

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

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

4.6.3 Linear Discriminant Analysis

Here we perform LDA on the `Smarket` data.

```r
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
## 0.491984 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
```

The LDA output indicates that $\hat{\pi}_1 = 0.492$ and $\hat{\pi}_2 = 0.508$, meaning that 49.2% of the training observations correspond to days during which the market went down and 50.8% is for up.

The coefficients of linear discriminants output 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 a market decline. Then we can use the `plot()` function to draw the graph.

The `predict()` function returns a list with three elements, `class`, `posterior`, and `x`.

```r
lda.pred=predict(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
```

For the mean of the value

```
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`.

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

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

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

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

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

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

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

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

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

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
```

4.6.4 Quadratic Discriminant Analysis

We can 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()`.

```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 coef- ficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors.

```r
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
```

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

```
## [1] 0.5992063
```

QDA predictions are accurate almost 60% of the time, even though the 2005 data was not used to fit the model. This level of accu- racy 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

In this part we will use the `knn()` function to perform the K-Nearest Neighbors method.

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.

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 to bind the `Lag1` and `Lag2` variables together into two matrices, one for the training set and the other for the test set.

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

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
```

The results using K = 1 are not very good, since only 50 % of the observations are correctly predicted. Thus we can do it 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] 0.5357143
```

The results have been improved slightly.

4.6.6 An Application to Caravan Insurance Data

Finally, we will apply the KNN approach to the `Caravan` data set, which is part of the `ISLR` library. This data set includes 85 predictors that measure demographic characteristics for 5,822 individuals.

```
dim(Caravan)
```

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

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

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

```
348/5822
```

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

We can 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.

```
standardized.X=scale(Caravan [,-86])
var( Caravan [ ,1])
```

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

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

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

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

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

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

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

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
```

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.

```
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 then try $K = 3$ and 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
```

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. This is over five times better than random guessing.

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

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

```r
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
```

```r
glm.pred=rep("No",1000)
glm.pred[glm.probs >.25]=" Yes"
table(glm.pred,test.Y)
```

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

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

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