Lecture 8: Linear methods for classification

Alan Lee

Department of Statistics
STATS 784 Lecture 8

August 18, 2017

Outline

Today's agenda

In this lecture we discuss linear methods for classification, (although many of the concepts covered will be applicable to other methods). Some of the ideas should be familiar from other courses, particularly STATS 330/762. Today we will cover

- Classification: general principles;
- Bayes classifier;
- Logistic regression;
- Linear and Quadratic discrimination analysis;
- Cross-validation and bootstrap estimates of classification error;
- K-nearest neighbour classifier;

We will use the credit card default data from *Introduction to Statistical Learning* as a running example.

Classification

In today's lecture we consider the case when our output is a categorical variable, indicating which one of a number of classes the observation belongs to e.g. alive/dead, win/draw/loss, 1-5 in a Likert scale in a questionnaire.

Given we have observed inputs $x_1, \ldots, x_k$, how should we classify an observation? For example, given a person's age, can we predict if they suffer from arthritis?

Bayes classifier

Suppose for each age x, we know the conditional probabilities

$$\text{Prob[has arthritis } | \text{age } = x],$$

$$\text{Prob[does not have arthritis } | \text{age } = x].$$

Then the best classifier (one that minimizes the probability of misclassification) is the one that assigns to a individual aged x the class with the highest conditional probability. (In the case of just two classes, as is the case here, this is the same as classifying an individual as having arthritis if

$$\text{Prob[has arthritis} | \text{age } = x] > 0.5.$$

Estimating conditional probabilities

The problem with this approach is that we don't usually know these conditional probabilities. However, if we could **estimate** them, we could use the Bayes classifier with the estimated probabilities.

We will look at three ways of doing this: using logistic regression, using discriminant analysis, and K-nearest neighbours.

Logistic regression

We first discuss the case where there are just two categories, which we label 0 and 1. We denote the output as Y, so either $Y = 0$ or $Y = 1$, and the inputs as x. In logistic regression, we model the probabilities as

$$\text{Prob}[Y = 1|x] = \frac{\exp\{b_0 + b_1x_1 + \cdots b_kx_k\}}{1 + \exp\{b_0 + b_1x_1 + \cdots b_1x_k\}},$$

$$\text{Prob}[Y = 0|x] = \frac{1}{1 + \exp\{b_0 + b_1x_1 + \cdots b_1x_1\}},$$

where we have to choose the values of b_0, b_1, b_k, based on a training set.

Odds & log-odds form

Odds form:

$$\frac{\text{Prob}}{1 - \text{Prob}} = \exp(b_0 + b_1 x_1 + \cdots + b_k x_k)$$

Log-odds form:

$$\log \frac{\text{Prob}}{1 - \text{Prob}} = b_0 + b_1 x_1 + \cdots + b_k x_k.$$

Choosing the b's

The b's are chosen by the method of maximum likelihood - this is a statistical technique based on assumptions on how the data was generated. See STATS 330 for more detail. In R, this is done using the function `glm`. See the example below.

Prediction error

In classification problems, the prediction error is measured by the probability of misclassification. This can be estimated from the training set by the proportion of misclassified observations, or from the test set in the same way. We illustrate with an example, from *An Introduction to Statistical Learning*. In the package `ISLR` there is a data set `Defaults` containing data on 10,000 monthly credit card bills. The variables are

income	: annual income of card holder;
balance	: the monthly balance;
student	:student status Yes=student, No=Not a student;
default	: defaulted on payment? Yes/No

Note: The variable student is categorical, so the effect of being a student is to increase the predicted log-odds by the amount of the student coefficient.

Calculating the b's

```
> library(ISLR)
> data(Default)
> default.logistic = glm(default~ student+balance+income,
+      data=Default, family=binomial)
> coefficients(default.logistic)
  (Intercept)    studentYes       balance        income
-1.086905e+01 -6.467758e-01  5.736505e-03  3.033450e-06
```

Note: The effect of being a student is to decrease the predicted log-odds by 6.467758e-01.

Predicting

To make the predictions, we use the generic `predict` function to calculate the estimated conditional probabilities, and then make a table of predictions versus actuals. We predict a default if the estimated probability is greater than 0.5:

```
> probs = predict(default.logistic, type="response")
> my.table = table(Default$default, probs>0.5)
> my.table
       FALSE TRUE
  No    9627   40
  Yes    228  105
```

The estimated error is the proportion of individuals on the off-diagonals:

```
> 1-sum(diag(my.table))/sum(my.table)
[1] 0.0268
```

Error rates: general loss functions

As in regression, we can get estimates of the out-of-sample error rate using either a test set, cross-validation or the bootstrap. In both regression and classification, the error when using a prediction rule $f(x)$ is measured by the aveage value of the loss function

$$\frac{1}{M}\sum_{i=1}^{M} L(y_i, f(x_i))$$

where $L(y, f(x))$ is the error when predicting a case whose true response is y with the prediction $f(x)$. For least squares, the loss function is $(y - f(x))^2$. For a binary response, with a predictor $f(x)$ taking values 0 and 1, it is

$$L(y, f(x)) = \begin{cases} 0, & y = f(x) \\ 1, & y \neq f(x) \end{cases}$$

Sensitivity and specificity

The predictor does well when predicting the non-defaults, but poorly when predicting the defaults. We can distinguish between these:

- Sensitivity: probability of predicting a 1 when the case is truly a 1: the "true positive rate"
- Specificity: probability of predicting a 0 when the case is truly a 0: the "true negative rate" (1-specificity is called the "false positive rate")
- Ideally, want both to be close to 1
- Thus, the sensitivity is the probability of correctly identifying a default - i.e. $105/(228+105) = 0.315$, the specificity is $9627/(9627+40)=0.996$.

Sensitivity and specificity

We can trade off the specificity against the sensitivity by moving away from the Bayes rule. For example, if we decide to classify a customer as a default if the probability of default is more than 0.1 (instead of 0.5) we would get

```
> my.table = table(Default$default, probs>0.1)
my.table
      FALSE TRUE
  No   9107  560
  Yes    85  248
> 9107/(9107+560)
[1] 0.942071
> 248/(85+248)
[1] 0.7447447
```

so the sensitivity has gone down a bit (from 0.996 to 0.942) but the specificity has improved from 0.315 to 0.745. The overall prediction error has increased (from 0.0268 to 0.0645).

Predicting rare events

Defaults are quite rare: only about 3% of the data set is a default. In data sets like this, very few, if any of the estimated probailities of success (e.g. default) will be more than 0.50. In this case almost all cases will be predicted to be non-defaults. This is not very helpful.

Sometimes we are interested in the cases that have the highest probability of default, even if this probability is less than 0.5. We might set the threshold lower, say 0.2, and flag customers that have a probability of more than 0,2 of defaulting. Thus, we are more interested in predicting probabilities rather than outcomes. We may be interested in a customer with default probability 0.2, even though by the Bayes rule, we would predict that he or she won't default.

Example: cold calling

Suppose an insurance company is peddling insurance on caravans. One option is to cold-call previous customers, but this will result in very few successful sales. A better option is to estimate the probability of a successful sale, given the information in the company's database. The calls can then be concentrated on the customers with the highest probability of actually purchasing insurance.

This example was the basis of the 2000 COIL Challenge (a data mining competition): see liacs.leidenuniv.nl/∼puttenpwhvander/library/cc2000/

Bootstrapping and cross-validation

This works exactly as before, except that instead of averaging the squared errors, we average the quantity

$$L(y, \hat{y}) = \begin{cases} 1, & y \neq \hat{y}, \\ 0, & y = \hat{y}, \end{cases} .$$

where y is the actual category and $\hat{y}$ the predicted one. (For numerical outputs we use $L(y, \hat{y}) = (y - \hat{y})^2$)

Example: the default data

```
> library(R330)
> cross.val(default.logistic,nfold=10)
Mean Specificity =  0.9958682
Mean Sensitivity =  0.3174139
Mean Correctly classified =  0.973215
> 1-0.973215
[1] 0.026785

err.boot(default.logistic,B=100)
$err
[1] 0.0268

$Err
[1] 0.026654
```

Example: More than two output categories

Suppose that there are J output categories, $C_1, C_2, \ldots, C_J$. We estimate the conditional probabilities by

$$\text{Prob}[Y \text{ in } C_j|x] = \frac{\exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}{1 + \sum_{j=1}^{J-1} \exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}$$

for $j = 1, 2, \ldots, (J-1)$ and

$$\text{Prob}[Y \text{ in } C_J|x] = \frac{1}{1 + \sum_{j=1}^{J-1} \exp\{b_{j0} + b_{j1}x_1 + \cdots b_{jk}x_k\}}$$

The coefficients b_{jk} are found using the `nnet` function - see later.

Using Bayes Theorem

Logistic regression works by modeling the conditional probability of being in a category, given the features x, i.e. we model

$$\text{Prob}[Y \text{ in } C_j|x].$$

An alternative is to turn this around, and instead model the conditional density $f(x|C_j)$ of x, given the class C_j. Then we can recover $\text{Prob}[Y \text{ in } C_j|x]$ using Bayes Theorem:

$$\text{Prob}[Y \text{ in } C_j|x] = \frac{f(x|C_j)\pi_j}{\sum_{j=1}^{J} f(x|C_j)\pi_j}.$$

Here the π_j are the prior probabilities that a case will be in class C_j. As before, we assign a case with features x to the class with the highest conditional probability. As the denominators are the same, this amounts to assigning to the class with the highest value of $f(x|C_j)\pi_j$.

Conditional densities

How should we model the conditional densities? There are several possibilities (we will assume for simplicity that all the features are numeric)

1. Use a non-parametric density estimate, estimated from the training data.
2. Use a multivariate normal distribution.
3. Use a mixture of multivariate normals (see later in the Unsupervised learning lectures)

Multivariate normal

An equivalent to finding the maximum $f(x|C_j)\pi_j$ is to find the maximum $\log(f(x|C_j)\pi_j)$. In the case of the multivariate normal this is the maximum value of

$$-\frac{1}{2}\log\det(\Sigma_j) - \frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j) + \log\pi_j$$

Here μ_j is the mean vector and Σ_j the covariance matrix of the jth conditional distribution. These are usually unknown, so to get a practical rule we replace these with estimates calculated from the training set.

In the case of two classes, the rule divides the feature space into two regions separated by a quadratic boundary, so this technique is called Quadratic Discriminant Analysis.

Linear discriminant analysis

If we assume that the covariance matrices Σ_j are all equal (to Σ say) the rule is to assign the case to the class having the largest value of

$$x^T \Sigma^{-1} \mu_j - \frac{1}{2} \mu_j^T \Sigma^{-1} \mu_j + \log \pi_j$$

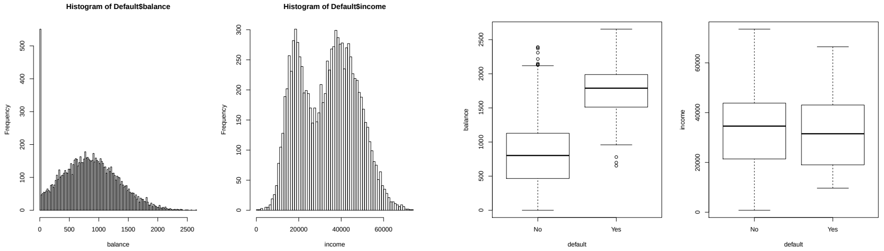
which results in linear discriminant functions. For two classes, the boundary is a linear boundary. The common covariance is estimated by a weighted average of the separate sample covariance matrices:

$$\hat{\Sigma} = \frac{(n_1 - 1)\hat{\Sigma}_1 + \cdots + (n_J - 1)\hat{\Sigma}_J}{n_1 + \cdots n_J - J}$$

(the within-groups covariance matrix). Here $\hat{\Sigma}_j$ and n_j are the sample covariance matrices and the sample size for the jth category in the training data. This is Linear Discriminant Analysis.

Example

We will us the default data, ignoring the student feature.



Seems like some discrimination should be possible.

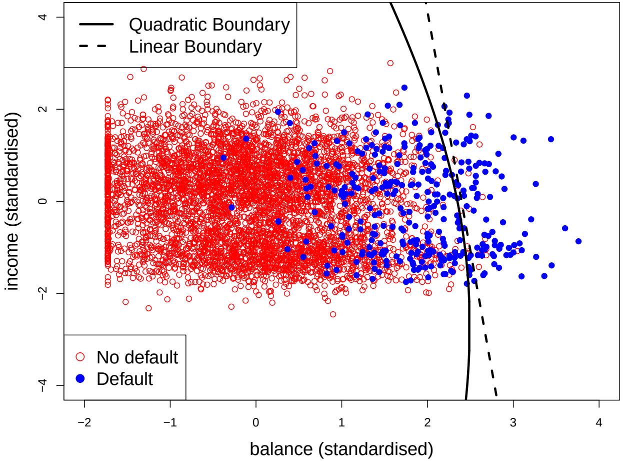
Example: QDA

```
> fit.qda = qda(default~balance + income, data = scaledDefault)
> predClasses = predict(fit.qda)$class
> head(predClasses)
[1] No No No No No No
Levels: No Yes
> predProbs = predict(fit.qda)$posterior
> head(predProbs)
         No          Yes
1 0.9993344 6.655567e-04
2 0.9993649 6.350755e-04
3 0.9929750 7.024999e-03
4 0.9999138 8.617529e-05
5 0.9991055 8.945327e-04
6 0.9986286 1.371356e-03
> qdaTable = table(predClasses, scaledDefault$default)
> 1 - sum(diag(qdaTable))/sum(qdaTable)
```

Example: LDA

```
> fit.lda = lda(default~balance + income, data = scaledDefault)
> predClasses = predict(fit.lda)$class
> head(predClasses)
[1] No No No No No No
Levels: No Yes
> predProbs = predict(fit.lda)$posterior
> head(predProbs)
         No         Yes
1 0.9967714 0.003228590
2 0.9971267 0.002873331
3 0.9870147 0.012985269
4 0.9988978 0.001102160
5 0.9961908 0.003809204
6 0.9957063 0.004293704
> qdaTable = table(predClasses, scaledDefault$default)
> 1 - sum(diag(qdaTable))/sum(qdaTable)
[1] 0.0276
```

Example: Boundaries



K-nearest neighbours

Here we estimate

$$\text{Prob}[Y \text{ in } C_j | x]$$

by the proportion of the K data points closest to x that are in category C_j.

To define "closest" we need a definition of distance - in the case of numeric variables we use Euclidean distance, in the case of categorical inputs we can use dummy variables - see STATS 330 for details.

We should standardise the variables to prevent one variable dominating the distance.

Example

```
# knn
library(class)
# scale the inputs, turn student into numeric
defaultScaled  = data.frame(student=as.numeric(Default[,2]),
   as.data.frame(scale(Default[,3:4])))
predictions = knn(defaultScaled, defaultScaled,
   Default$default, k=5)
knn.table = table(Default$default,predictions)
> knn.table
     predictions
        No  Yes
  No  9614   53
  Yes  199  134
> 1-sum(diag(knn.table))/sum(knn.table)
[1] 0.0252
```