Chapter 4 ISLR Classification

Laura Kapitula

11/4/2021, updated

# Classification

## Packages used in this chapter

library(tidyverse)  
library(tidymodels)  
library(discrim)  
library(kknn)  
library(knitr)  
library(kableExtra)  
library(skimr)

Material in these notes are from the book Introduction to Statistical Learning and a Tidyverse take on using the methods given at <https://beaulucas.github.io/tidy_islr/classification.html>. As well as other sources such as your highly opinionated professor’s head.

We will use tidymodels here, [<https://www.tidymodels.org/>]

Linear regression is concerned with predicting a quantitative response variable. What if the response variable is *qualitative*? Eye color is an example of a qualitative variable, which takes discrete value such as blue, brown, green. These are also referred to as *categorical*. We saw in the Oring example how logistic regression can be useful for modeling binomial data.

The approach of predicting qualitative responses is known as *classification*. Often, we predict the probability of the occurences of each category of a qualitative variable, and then make a decision based off of that.

In chapter 4 of ISLR three of the most widely-used classifiers are discussed. We illustrate those methods further here.

* logistic regression
* linear discriminant analysis
* *k*-nearest neighbors

## An Overview of Classification

Classification is a common scenario.

1. Person arrives at ER exhibiting particular symptoms. What illness does he have?
2. Money is wired to an external account at a bank. Is this fraud?
3. Email is sent to your account. Is it legit, or spam?

Similar to regression, we have a set of training observations that we can use to build a classifier or estimate probabilities. We want the classifier to perform well on both training and test observations.

We will use the dataset ISLR::Default. First, let’s convert it to tidy format.

default <- ISLR2::Default %>% as\_tibble()

We are interested in the ability to predict whether an individual will default on their credit card payment, based on their credit card balance and annual income.

If we look at the summary statistics, we see the data is clean, and that very few people default on their balances.

#default %>% skimr::skim()

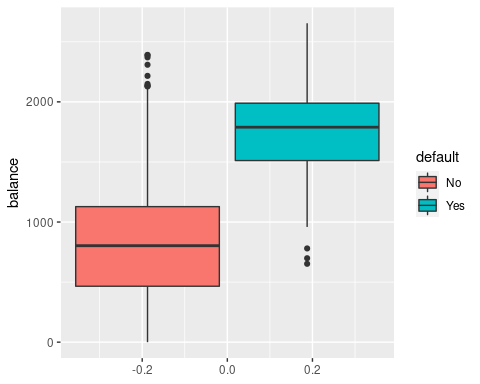
The hex scatterplot signals a strong relationship between balance and default. The hex scatterplot can be a nice way of visualizing relationships in larger data sets.

default %>%  
 ggplot(aes(x = balance, y = income, fill = default)) +   
 geom\_hex(alpha=2/3) #the 2/3 is transparency



The boxplot captures the difference in balance between those who default and those who do not.

default %>%  
 ggplot(aes(y = balance, fill = default)) +  
 geom\_boxplot()



## Why Not Linear Regression?

Imagine we were trying to predict the medical outcome of a patient on the basis of their symptoms. Let’s say there are three possible diagnoses: stroke, overdose, and seizure. We could encode these into a quantitative variable . that takes values from 1 to 3. Using least squares, we could then fit a regression model to predict .

Unfortunately, this coding implies an ordering of the outcomes. It also insists that the difference between levels is quantitative, and equivalent across all sequences of levels.

Thus, changing the order of encodings would change relationship among the conditions, producing fundamentally different linear models.

There could be a case where a response variables took on a natural ordering, such as mild, moderate, severe. We would also need to believe that the gap between each level is equivalent. Unfortunately, there is no natural way to convert a qualitative response variable with more than two levels into a quantitative response that is appropriate for linear regression.

For cases of *binary* qualitative response, we can utilize the indicator(dummy) variable solution seen in Chapter 3. Linear regression will provide a fit model for this binary response scenario. However, it is possible for linear regression to produce estimates outside of the [0, 1] interval, which affects their interpretability as probabilities and we are clearly not fitting a model appropriate to the data.

When the qualitative response has more than two levels, we need to use classification methods that are appropriate.

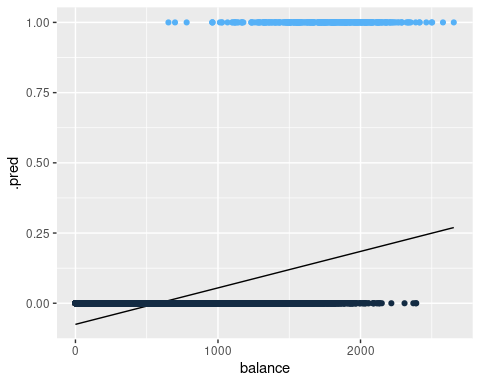
## Logistic Regression

Let’s consider the default dataset. Rather than modeling this response directly, logistic regression models the *probability* that belongs to a particular category. It is always much more useful to model probability than to just make a prediction, as if we know the probability we can make decisions under different conditions using different assumptions and loss functions. As we saw before when we studied probability and Baye’s rule, our conditional probability of an outcome such as disease will depend on the proportions in the population of interest.

If we estimate using linear regression, we see that some estimated probabilities are negative and given probability is never negative this alerts us that our model is not very good. Our model would ideally not give nonsense values.

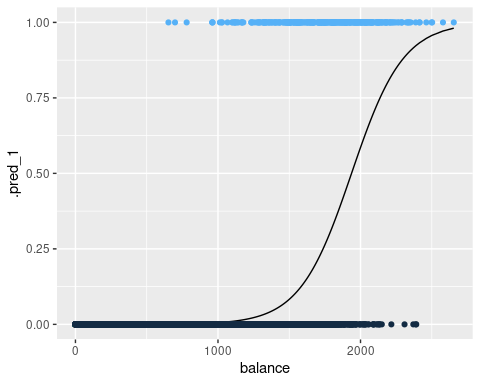
We are using the tidymodels package here. I am learning this method of model fitting along with you but so far it seems rather nice.

default <- default %>%  
 mutate(default\_bool = if\_else(default == "Yes", 1, 0))  
  
lm\_default <- linear\_reg() %>%  
 set\_engine("lm") %>%  
 fit(data = default, default\_bool ~ balance)  
  
default %>%  
 bind\_cols(predict(lm\_default, default)) %>%  
 ggplot(aes(x = balance)) +  
 geom\_line(aes(y = .pred)) +  
 geom\_point(aes(y = default\_bool, colour = default\_bool)) +  
 guides(colour="none")



Below is the classification using logistic regression, where the probabilities fall between 0 and 1.

logi\_default <- logistic\_reg(mode = "classification") %>%  
 set\_engine("glm") %>%  
 fit(data = default, as.factor(default\_bool) ~ balance)   
  
default %>%  
 bind\_cols(predict(logi\_default, default, type = "prob")) %>%  
 ggplot(aes(x = balance)) +  
 geom\_line(aes(y = .pred\_1)) +  
 geom\_point(aes(y = default\_bool, colour = default\_bool)) +  
 guides(colour="none")



Logistic regression in this example is modelling the probability of default, given the value of balance.

Pr(default = Yes|balance)

These values, which we abbreviate as *p*(balance), range between 0 and 1. Logistic regression will always produce an *S-shaped* curve. Regardless of the value of , we will receive a sensible prediction.

From this, we could make a classification prediction for default but the probability itself is much more informative, so I would much rather report the probability than just the prediction to a client. Similar to the reason I did not step functions I do not like just a classification without a probability because just classifying is basically binning my estimated probabilities and how I want to make a decision will vary. This to me is one of the main advantages of modeling instead of just using a heuristic method such as K Nearest Neighbors.

### The Logistic Model

The problem of using a linear regression model is evident in the chart above, where probabilities can fall below 0 or greater than 1.

To avoid this, we must model using a function that gives outputs between 0 and 1 for all values of . In logistic regression, we use the *logistic function*,

*logistic function*

To fit the model, we use a method called *maximum likelihood* if we are using the classical frequentist statistics method or we combine prior information with the likelihood and sample from our posterior using Bayesian methods. The estimates obtained via MLE will typically agree with the Bayesian medians with uninformative or weakly informative priors.

If we manipulate the logistic function, we find that

*odds*

This is called the *odds*, and takes any value from to . This is the same type of odds used in sporting events (“9:1 odds to win this match”, etc). If , then odds are .

If we take the logarithm of the odds, we arrive at

*log-odds  logit*

The left-handed side is called the *log-odds* or *logit*. The logistic regression model has a logit that is linear in .

The contrast to linear regression is that increasing by one-unit changes the log odds by (or the odds by . However, since and relationship is not a straight line (see plot above), does not correspond to the the change in associated with a one-unit increase in . The amount that changes depends on the current value of . See how the slope approaches 0 more and more slowly as balance increases.

Regardless of how much moves, if is positive then increasing will be associated with increasing . The opposite is also true.

### Estimating the Regression Coefficients

The coefficients in the logistic regression equation must be estimated used training data. Linear regression used the least squares approach to estimate the coefficients. Here *maximum likelihood* is typically used.

Maximum likelihood seeks to to find estimates for and such that the predicted probability of default for each individual corresponds as closely as possible to to the individual’s observed default status. We want estimates that produce low probabilities for individuals who did not default, and high probabilities for those who did.

We can examine the coefficients and other information from our logistic regression model.

logi\_default %>% tidy()

## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -10.7 0.361 -29.5 3.62e-191  
## 2 balance 0.00550 0.000220 25.0 1.98e-137

If we look at the terms of our logistic regression, we see that the coefficient for balance is positive. This means that higher balance increases . A one-unit increase in balance will increase the log odds of defaulting by about 0.0055.

The test-statistic also behaves similarly. Coefficients with large statistics indicate evidence against the null hypothesis . For logistic regression, the null hypothesis implies that , which means that the probability of defaulting does not depend on balance.This is of course a nonsense hypothesis, so it is not at all informative that we rejected it.

The intercept () is typically not of interest; it’s main purpose is to adjust the average fitted probabilities to the proportion of ones in the data. Note that in Case-Control studies we can’t properly estimate the but we can estimate the other ’s

### Making Predictions

Once we have the coefficients, we simply compute the probability of default for any given observation.

Let’s take an individual with a balance of $1000. Using our model terms, we can compute the probability. Let’s extract the terms from the model and plug in a balance of $1000.

logi\_coef <- logi\_default %>%  
 tidy() %>%  
 # widen it and clean up names  
 select(term, estimate) %>%  
 pivot\_wider(names\_from = term, values\_from = estimate) %>%  
 janitor::clean\_names()  
logi\_coef %>%  
 mutate(prob\_1000 = exp(intercept + balance \* 1000) /  
 (1 + exp(intercept + balance \* 1000)))

## # A tibble: 1 x 3  
## intercept balance prob\_1000  
## <dbl> <dbl> <dbl>  
## 1 -10.7 0.00550 0.00575

We find the estimated probability to be less than 1%.

We can also incorporate qualitative predictors with the logistic regression model. Here we encode student in to the model.

logi\_default\_student <-   
 logistic\_reg(mode = "classification") %>%  
 set\_engine("glm") %>%  
 fit(data = default, as.factor(default\_bool) ~ student)  
logi\_default\_student %>% tidy()

## # A tibble: 2 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -3.50 0.0707 -49.6 0   
## 2 studentYes 0.405 0.115 3.52 0.000431

This model indicates that students have a higher rate of defaulting compared to non-students. The idea is the same as what we learned with linear regression.

Note that if we take exp(-3.50)=0.030 we get the estimated odds for non-students. If we take exp(-3.50+.40)= 0.045 we get the estimated odds for students. If we exponentiation the term for studentYes, exp(0.40)=1.49 we note that the odds for students is estimated to be 1.49 times the odds for a non-student.

### Multiple Logistic Regression

We now consider the scenario of multiple predictors.

We can rewrite as

/p>

And again use the maximum likelihood method to estimate the coefficients.

Let’s estimate the probability of default using balance, income and student.

multiple\_logi\_default<- logistic\_reg(mode = "classification") %>%  
 set\_engine('glm') %>%  
 fit(data = default, as.factor(default\_bool) ~ balance + student + income)  
multiple\_logi\_default %>% tidy()

## # A tibble: 4 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) -10.9 0.492 -22.1 4.91e-108  
## 2 balance 0.00574 0.000232 24.7 4.22e-135  
## 3 studentYes -0.647 0.236 -2.74 6.19e- 3  
## 4 income 0.00000303 0.00000820 0.370 7.12e- 1

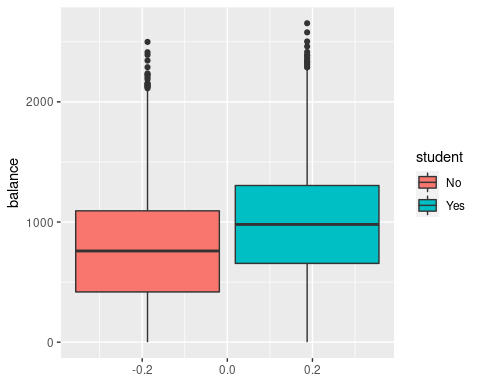
kableExtra::kable(x = broom::tidy(multiple\_logi\_default), format = "pipe")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| term | estimate | std.error | statistic | p.value |
| (Intercept) | -10.8690452 | 0.4922555 | -22.080088 | 0.0000000 |
| balance | 0.0057365 | 0.0002319 | 24.737563 | 0.0000000 |
| studentYes | -0.6467758 | 0.2362525 | -2.737646 | 0.0061881 |
| income | 0.0000030 | 0.0000082 | 0.369815 | 0.7115203 |

Notice that being a student now *decreases* the chances of default, whereas in our previous model (which only contained student as a predictor), it increased the chances.

Why is that? This model is showing that, for a fixed value of income and balance, students actually default less. This is because student and balance are correlated.

default %>%  
 ggplot(aes(y = balance, fill = student)) +  
 geom\_boxplot()



If we plot the distribution of balance across student, we see that students tend to carry larger credit card balances.

This example illustrates the dangers of drawing insights from single predictor regressions when other predictors may be relevant. The results from using one predictor can be substantially different compared to using multiple predictors. This phenomenon is known as *confounding*. This is why in observational studies we can’t really say which variable is most important or how an individual variable impacts the estimated probability of success.

### Logistic Regression for >2 Response Classes

Sometimes we wish to classify a response variable that has more than two classes. This could be the medical example where a patient outcomes falls into stroke, overdose, and seizure. It is possible to extend the two-class logistic regression model into multiple-class, these methods are useful but I will not cover them here.

A method that is popular for multi-class classification is *discriminant analysis*.

## Linear Discriminant Analysis

Logistic regression models the distribution of response given the predictor(s) . In discriminant analysis, we model the distribution of the predictors in each of the response classes, and then use Bayes’ theorem to flip these around into estimates for .

Why do we need this method?

* Well-separated classes produce unstable parameter estimates for logistic regression models
* If is small and distribution of predictors is normal across the classes, the linear discriminant model is more stable than logistic regression

### Using Bayes’ Theorem for Classification

Consider the scenario where we want to classify an observation into one of classes, where .

* Let represent the overall or *prior* probability that a randomly chosen observation comes from the th class
* Let denote the *density function* of for an observation that comes from the th class.

In other words, being large means that there is a high probability that an observation in the th class has .

We can use Bayes’ theorem

And call the left-hand side . We can plug in estimates of and into Bayes’ theorem above to get the probability of a certain class, given an observation.

* Solving for is easy if we have a random sample of s from the population. We simply calculate the fraction of observations that fall into a class.
* Estimating is more challenging unless we assume simple forms for these densities

We refer to as the posterior probability that an observation belongs to the th class. This is the probability that the observation belongs to the th class, *given* the predictor value for that observation.

The Bayes’ classifier classifies an observation to the class for which is largest. If we can find a way to estimate , we can develop a classifier that approximates the Bayes classifier.

### Linear Discriminant Analysis for p = 1

Let’s assume we have one predictor. We need to obtain an estimate for (the density function for given a class ). This will obtain a value for . We will then classify this observation for which is greatest.

To estimate , we need to make some assumptions about its form.

Let’s assume is *normal* or *Gaussian*. The normal density takes the form

Plugging this back in to , we obtain

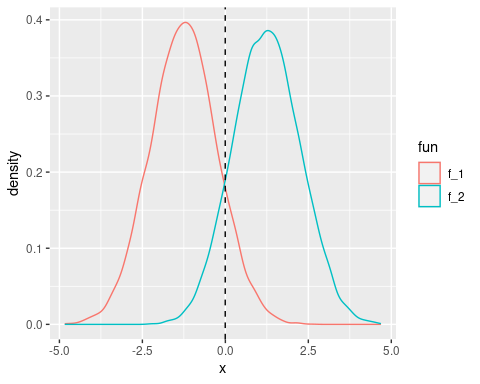
Taking the log and rearranging results in

In this case, the Bayes decision boundary corresponds to

We can simulate some data to show a simple example.

In this data we have two classes:

var\_1 = 1  
var\_2 = var\_1  
n=10000  
f\_1 = tibble(fun = "f\_1", x = rnorm(n = n, mean = -1.25, sd = var\_1))  
f\_2 = tibble(fun = "f\_2", x = rnorm(n = n, mean = 1.25, sd = var\_2))  
f\_x = bind\_rows(f\_1, f\_2)  
# add summary statistics  
f\_x <- f\_x %>%  
 group\_by(fun) %>%  
 mutate(pi = n()/(2\*n),  
 var = var(x),  
 mu = mean(x))   
decision\_boundary <- f\_x %>%  
 group\_by(fun) %>%  
 summarise(mu = mean(x)) %>%  
 summarise(decision\_boundary = sum(mu) / 2) %>%  
 pull() #this makes decision\_boundary into a constant   
f\_x %>%  
 ggplot(aes(x = x, colour = fun)) +  
 geom\_density() +  
 geom\_vline(xintercept = decision\_boundary, linetype = "dashed")



These two densities overlap, and so given , we still have uncertaintly about which class the observation belongs to. If both classes are equally likely for a random observation , then we see the Bayes classifier assigns the observation to class 1 if and class 2 otherwise.

Even if we are sure that is drawn from a Gaussian distribution within each class, we still need to estimate , , and . The *linear discriminant analysis* method approximates these by plugging in estimates as follows

\*

\*

The estimate for is the average of all training observations from the th class. The estimate for is the weighted average of the sample variances for each of the K classes.

To estimate , we simply take the proportion of training observations that belong to the th class

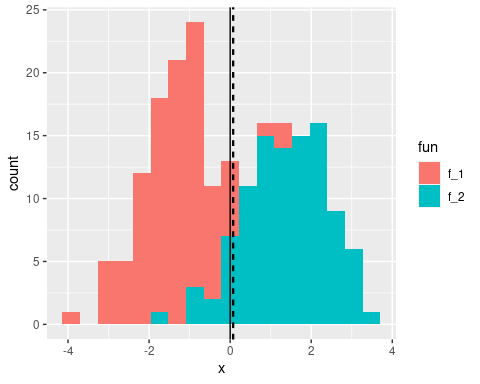
\*

From these estimates, we can achieve a decision boundary

This classifier has *linear* in the name due to the fact that the *discriminant function* above are linear functions of .

Let’s take a sample from our earlier distribution and see how it performs.

library(discrim)  
f\_sample = f\_x %>% sample\_frac(size = 0.01) #sample a proportion of the data  
lda\_f <- discrim::discrim\_linear() %>%  
 set\_engine('MASS') %>%  
 fit(data = f\_sample, as.factor(fun) ~ x)  
preds <- predict(lda\_f, f\_sample, type = "class")  
f\_sample <- f\_sample %>% bind\_cols(preds)  
est\_decision <- f\_sample %>% arrange(x) %>% filter(.pred\_class == 'f\_2') %>%  
 slice(1) %>% pull(x)  
ggplot(f\_sample, aes(x = x, fill = fun)) +  
 geom\_histogram(bins=18) +  
 geom\_vline(xintercept = est\_decision, linetype = "dashed") +  
 geom\_vline(xintercept = 0)



Notice the estimated decision boundary (dashed line) being very close to the Bayes decision boundary.

### Linear Discriminant Analysis for p > 1

We can extend LDA classifier to multiple predictors.

The multivariate Gaussian distribution assumes that each predictor follows a one-dimensional normal distribution, with some correlation between each pair of predictors.

* [Andrew Ng on Multivariate Gaussian Distribution](https://www.youtube.com/watch?v=JjB58InuTqM)

To indicate that a -dimensional random variable has a multi-variate Gaussian distribution, we write $ X N(, )$

* is the mean of (a vector with components)
* is the covariance matrix of .

The multivariate Gaussian density is defined as

In the case of predictors, the LDA classifier assumes that the observations in the th class are drawn from a multivariate Gaussian distribution , where is a class-specific mean vector, and is the covariance matrix that is common to all classes.

Plugging the density function for the th class, , into

and performing some algebra reveals that the Bayes classifier will assign observation by identifying the class for which

is largest.

#### Performing LDA on Default data

If we run an LDA model on our default dataset, predicting the probability of default based off of student and balance, we achieve a respectable 3.0% error rate.

set.seed(1)  
lda\_default <- discrim::discrim\_linear() %>%  
 set\_engine('MASS') %>%  
 fit(data = default, default ~ student + balance)  
preds <- predict(lda\_default, default, type = "class")  
# error rate  
default %>%  
 bind\_cols(preds) %>%  
 metrics(truth = default, estimate = .pred\_class)

## # A tibble: 2 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 accuracy binary 0.972  
## 2 kap binary 0.361

While this may seem impressive, let’s remember that only 3.6% of observations in the dataset end up in default. This means that if we assigned a *null* classifier, which simply predicted every observation to not end in default, our error rate would be 3.6%. This is worse, but not by much, compared to our LDA error rate. The kap (short for kappa, not kapitula) statistic is designed to adjust the accuracy for the proportion we would get correct just by selecting the majority category. Notice it is rather low.

# null error rate  
default %>%  
 group\_by(default) %>%  
 count() %>%  
 ungroup() %>%  
 mutate(prop = n / sum(n))

## # A tibble: 2 x 3  
## default n prop  
## <fct> <int> <dbl>  
## 1 No 9667 0.967   
## 2 Yes 333 0.0333

In our data, we only have 3.3% that are defaults.

Binary decision makers can make two types of errors:

* Incorrectly assigning an individual who defaults to the “no default” category
* Incorrectly assigning an individual who doesn’t default to the “default” category.

We can identify the breakdown by using a *confusion matrix*

cm <- default %>%  
 bind\_cols(preds) %>%  
 conf\_mat(truth = default, estimate = .pred\_class)  
cm

## Truth  
## Prediction No Yes  
## No 9644 252  
## Yes 23 81

We see that our LDA only predicted 104 people to default. Of these, 81 actually defaulted. So, only 23 of out of the 9667 people who did not default were incorrectly labeled.

However, of the 333 people in our test set who defaulted, we only predicted this correctly for 23 of them. That means 24% of individuals who default were incorrectly classified. Having an error rate this high for the problematic class (those who default) is unacceptable.

Class-specific performance is an important concept. *Sensitivity* and *specificity* characterize the performance of a classifier or screening test. In this case, the sensitivity is the percentage of true defaults who are identified (a low ~%). The specificity is the percentage of non-defaulters who are correctly identified (??/?? ~ ?%).

Remember that LDA is trying to approximate the Bayes classifier, which has the lowest *total* error rate out of all classifiers (assuming Gaussian assumption is correct). The classifier will yield the smallest total number of misclassifications, regardless of which class the errors came from. In this credit card scenario, the credit card company might wish to avoid incorrectly misclassifying a user who defaults. In this case, they value sensitivity. For them, the cost of misclassifying a defaulter is higher than the cost of misclassifying a non-defaulter (which they still desire to avoid).

It’s possible to modify LDA for such circumstances. Given the Bayes classifier works by assigning an observation to a class in which the posterior probability is greatest (in the two-class scenario, this decision boundary is at 0.5), we can modify the probability threshold to suit our needs. If we wish to increase our sensitivity, we can lower this threshold.

Imagine we lowered the threshold to 0.2. Sure, we would classify more people as defaulters than before (decreasing our specificity) but we would also catch more defaulters we previously missed (increasing our sensitivity).

preds <- predict(lda\_default, default, type = "prob")  
# error rate  
default %>%  
 bind\_cols(preds) %>%  
 mutate(.pred\_class = as.factor(if\_else(.pred\_Yes > 0.2, "Yes", "No"))) %>%  
 conf\_mat(truth = default, estimate = .pred\_class)

## Truth  
## Prediction No Yes  
## No 9432 138  
## Yes 235 195

Now our sensitivy has increased. Of the ?? people who defaulted, we correctly identified ??, or ~??% of them (up from ~??% previously).

This came at a cost, as our specificity decreased. This time, we predicted ? people to default. Of those, 53 actually defaulted. This means that ? of the ? people who didn’t default were incorrectly labelled. This gives us a specificity of (?/? ~ ?%)

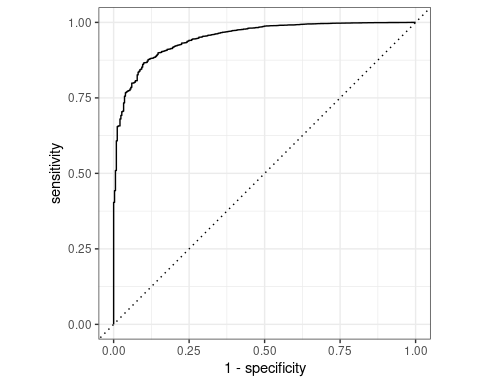
Despite the overall increase in error rate, the lower threshold may be chosen, depending on the context of the problem. To make a decision, an extensive amount of *domain knowledge* is required.

The *ROC curve* is a popular graphic for displaying the two types of errors for all possible thresholds. “ROC” stands for *receiver operating characteristics*.

The overall performance of a classifier, summarized over all possible thresholds, is given by the *area under the (ROC) curve* (AUC). An ideal ROC curve will hug the top left corner. Think of it this way: ideal ROC curves are able to increase sensitivity at a much higher rate than reduction in specificity.

We can use yardstick:: (part of tidymodels::) to plot an ROC curve.

default %>%  
 bind\_cols(preds) %>%  
 roc\_curve(default, .pred\_No) %>%  
 autoplot()



We can think of the *sensitivity* as the true positive, and *1 - specificity* as the false positive.

### Quadratic Discriminant Analysis

LDA assumes that the observations within each class are drawn from a multivariate Gaussian distribution, with a class-specific mean vector and a covariance matrix that is common to all classes. *Quadratic discriminant analysis* (QDA) assumes that class has its own covariance matrix.

It assumes that each observation from the th class has the form , where is a covariance matrix for the th class. Under this assumption, the Bayes classifier assigns an observation to the class for which

is largest. In this case, we plug in estimates for , , and . Notice the quantity appears as a quadratic function, hence the name.

So why would one prefer LDA to QDA, or vice-versa? We again approach the bias-variance trade-off. With predictors, estimating a class-independent covariance matrix requires estimating parameters. For example, a covariance matrix with 4 predictors would require estimating 4(4+1)/2 = 10 parameters. To estimate a covariance matrix for each class, the number of parameters is paramters. With 50 predictors, this becomes some multiple of 1,275, depending on . The assumption of the common covariance matrix in LDA causes the model to become linear in , which means there are linear coefficients to estimate. As a result, LDA is much less flexible clasifier than QDA, and has lower variance.

The consequence of this is that if LDA’s assumption of a common covariance matrix is significantly off, the LDA can suffer from high bias. In general, LDA tends to be a better bet than QDA when there are relatively few training observations and so reduction of variance is crucial. In contrast, with large data sets, QDA can be recommended as the variance of the classifier is not a major concern, or the assumption of a common covariance matrix for the classes is clearly not correct.

Breaking the assumption of a common covariance matrix can “curve” the decision boundary, and so the use of a more flexible model (QDA) could yield better results.

## A Comparison of Classification Methods

Let’s discuss the classification methods we have considered and the scenarios for which one might be superior.

* Logistic regression
* LDA
* QDA
* K-nearest neighbors

There is a connection between LDA and logistic regression, particularyly in the two-class setting with predictor. The difference being that logistic regression estimates coefficients via maximum likelihood, and LDA uses the estimated mean and variance from a normal distribution.

The similarity in fitting procedure means that LDA and logistic regression often give similar results. When the assumption that observations are drawn from a Gaussian distribution with a common covariance matrix in each class are in fact true, the LDA can perform better than logistic regression. If the assumptions are in fact false, logistic regression can outperform LDA.

KNN, on the other hand, is completely non-parametric. KNN looks at observations “closest” to , and assigns it to the class to which the plurality of these observations belong. No assumptions are made about the shape of the decision boundary. We can expect KNN to outperform both LDA and logistic regression when the decision boundary is highly non-linear. A downside of KNN, even when it does outperform, is its lack of interpretability. KNN does not tell us which predictors are important.

QDA serves as a compromise between the non-parametric KNN method and the linear LDA and logistic regression approaches. The assumption of quadratic decision boundary allows it to accurately model a wider range of problems. It’s reduced flexibility compared to KNN allows it to produce a lower variance with a limited number of training observations due to it making some assumptions about the form of the decision boundary.

## Lab: Logistic Regression, LDA, and KNN

### Churn Dataset

We will be using the modeldata::wa\_churn dataset to test our classification techniques.

library(modeldata)  
data(wa\_churn)

These data were downloaded from the IBM Watson site (see below) in September 2018. The data contain a factor for whether a customer churned or not. Alternatively, the tenure column presumably contains information on how long the customer has had an account. A survival analysis can be done on this column using the churn outcome as the censoring information. A data dictionary can be found on the source website.

Our interest for this dataset is predicting whether a customer will churn or not. If a customer is likely to churn, we can offer them an incentive to stay. These incentives cost us money, so we want to minimize the incentives given out to customers that won’t churn. We will identify a balance between *sensitivity* and *specificity* that maximizes the ROI of our incentive.

wa\_churn %>% skimr::skim()

Data summary

Name

Piped data

Number of rows

7043

Number of columns

20

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Column type frequency:

factor

11

numeric

9

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Group variables

None

**Variable type: factor**

skim\_variable

n\_missing

complete\_rate

ordered

n\_unique

top\_counts

churn

0

1

FALSE

2

No: 5174, Yes: 1869

multiple\_lines

0

1

FALSE

3

No: 3390, Yes: 2971, No : 682

internet\_service

0

1

FALSE

3

Fib: 3096, DSL: 2421, No: 1526

online\_security

0

1

FALSE

3

No: 3498, Yes: 2019, No : 1526

online\_backup

0

1

FALSE

3

No: 3088, Yes: 2429, No : 1526

device\_protection

0

1

FALSE

3

No: 3095, Yes: 2422, No : 1526

tech\_support

0

1

FALSE

3

No: 3473, Yes: 2044, No : 1526

streaming\_tv

0

1

FALSE

3

No: 2810, Yes: 2707, No : 1526

streaming\_movies

0

1

FALSE

3

No: 2785, Yes: 2732, No : 1526

contract

0

1

FALSE

3

Mon: 3875, Two: 1695, One: 1473

payment\_method

0

1

FALSE

4

Ele: 2365, Mai: 1612, Ban: 1544, Cre: 1522

**Variable type: numeric**

skim\_variable

n\_missing

complete\_rate

mean

sd

p0

p25

p50

p75

p100

hist

female

0

1

0.50

0.50

0.00

0.00

0.00

1.00

1.00

▇▁▁▁▇

senior\_citizen

0

1

0.16

0.37

0.00

0.00

0.00

0.00

1.00

▇▁▁▁▂

partner

0

1

0.48

0.50

0.00

0.00

0.00

1.00

1.00

▇▁▁▁▇

dependents

0

1

0.30

0.46

0.00

0.00

0.00

1.00

1.00

▇▁▁▁▃

tenure

0

1

32.37

24.56

0.00

9.00

29.00

55.00

72.00

▇▃▃▃▆

phone\_service

0

1

0.90

0.30

0.00

1.00

1.00

1.00

1.00

▁▁▁▁▇

paperless\_billing

0

1

0.59

0.49

0.00

0.00

1.00

1.00

1.00

▆▁▁▁▇

monthly\_charges

0

1

64.76

30.09

18.25

35.50

70.35

89.85

118.75

▇▅▆▇▅

total\_charges

11

1

2283.30

2266.77

18.80

401.45

1397.47

3794.74

8684.80

▇▂▂▂▁

The dataset contains 7043 observations and 20 variables. The dataset consists mostly of factor variables, all with a small amount of unique levels. We are going to try out logistic regression, LDA, and K-nearest neighbors on this dataset. For now, we will only remove the 11 observations which have missing values for the total\_charges column. We will also reorder the churn column levels so that a “success” corresponds with a customer churning.

wa\_churn <- wa\_churn %>% filter(!is.na(total\_charges)) %>%  
mutate(churn = fct\_relevel(churn, "No", "Yes"))

Now let’s take a look at the response variable.

wa\_churn %>% group\_by(churn) %>% tally() %>% mutate(prop = n / sum(n))

## # A tibble: 2 x 3  
## churn n prop  
## <fct> <int> <dbl>  
## 1 No 5163 0.734  
## 2 Yes 1869 0.266

In this dataset, 73.5% of the observations do not churn. There is some skew, but nothing drastic. We will now prepare the data and apply binary classification techniques to see which model performs best.

set.seed(40)  
split\_churn <- initial\_split(wa\_churn, prop = 3/4)  
train\_churn <- training(split\_churn)  
test\_churn <- testing(split\_churn)

### Logistic Regression

Let’s run a logistic regression to predict churn using the available variables.

Unlike ISLR, we will use the parsnip::logistic\_reg function over glm due to its API design and machine learning workflow provided by its parent package, tidymodels. Models in the {parsnip} package also allow for choice of different computational engines. This reduces cognitive overhead by standardizing the high-level arguments for training a model without rembembering the specifications of different engine. In our case, we will be using the glm engine.

logistic\_reg() is a way to generate a specification of a model before fitting and allows the model to be created using different packages in R, Stan, keras, or via Spark.

logi\_churn <- logistic\_reg(mode = "classification") %>%  
 fit(data = train\_churn, churn ~ .)

## Warning: Engine set to `glm`.

broom::tidy(logi\_churn)

## # A tibble: 31 x 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 0.845 0.948 0.891 3.73e- 1  
## 2 female 0.000921 0.0751 0.0123 9.90e- 1  
## 3 senior\_citizen 0.128 0.0987 1.30 1.93e- 1  
## 4 partner 0.00365 0.0907 0.0402 9.68e- 1  
## 5 dependents -0.185 0.104 -1.78 7.55e- 2  
## 6 tenure -0.0639 0.00741 -8.62 6.44e-18  
## 7 phone\_service -0.0198 0.752 -0.0263 9.79e- 1  
## 8 multiple\_linesNo phone service NA NA NA NA   
## 9 multiple\_linesYes 0.378 0.206 1.84 6.58e- 2  
## 10 internet\_serviceFiber optic 1.51 0.926 1.63 1.03e- 1  
## # … with 21 more rows

We can interpret the negative coefficients as reducing the chance a customer churns in the context of the model, and the positive coefficients as increasing the chance a customer churns in the context of the model.

#### Measuring model performance

We can use the predict() function with our {tidymodels} workflow. The type parameter specifies whether we want probabilities or classifications returned. The object returned is a tibble with columns of the predicted probability of the observation being in each class.

logi\_preds <- predict(object = logi\_churn, new\_data = test\_churn,  
 type = "class")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

Note that we get a warning here about rank, this is because we have a lot of categorical variables in our analysis, so our design is not ‘full-rank’ if we created our dummy variables ourselves and always left off one category we would not get this error. Specifying type = "class" will generate classification predictions based off of a 0.5 threshold (this might not be optimal for the problem).

Let’s add them to our test set.

test\_metrics <- test\_churn %>%  
 bind\_cols(logi\_preds)

This adds a column, .pred\_class at each observation.

We can produce a confusion matrix using conf\_mat() function of the {yardstick} package (used for measuring model performance, also part of {tidymodels}).

We tell conf\_mat() that the direction column is our source of truth, and our classifications are contained in the .pred\_class column.

cm\_churn <- test\_metrics %>%  
 conf\_mat(truth = churn, estimate = .pred\_class)  
cm\_churn

## Truth  
## Prediction No Yes  
## No 1142 224  
## Yes 132 260

conf\_mat objects also have a summary() method that computes various classification metrics.

summary(cm\_churn) %>%  
 filter(.metric %in% c('accuracy', 'sens', 'spec'))

## # A tibble: 3 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 accuracy binary 0.797  
## 2 sens binary 0.896  
## 3 spec binary 0.537

Our overall accuracy is around ~80.1%. This may seem high, but if we look at the original dataset, the data is skewed. A naive classifier would produce a 73.5% accuracy (the proportion of yes/no churn values).Still, it suggests that we are better off than randomly guessing.

Can we account for the skew of data when assessing our model? If we utilize the yardstick::metrics() function on our test data fitted with class predictions, we can get a dataframe containing overall model performance.

test\_metrics %>%  
 metrics(truth = churn, estimate = .pred\_class)

## # A tibble: 2 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 accuracy binary 0.797  
## 2 kap binary 0.461

Notice the metric of kap. The Kappa statistic compares an observed accuracy with expected accuracy (random chance). Kappa is a similar measure to accuracy(), but is normalized by the accuracy that would be expected by chance alone and is very useful when one or more classes have large frequency distributions.

We will be using kap to evaluate how our logistic regression fits to other techniques.

### Linear discriminant analysis

set.seed(40)  
lda\_churn <- discrim\_linear() %>%  
 fit(data = train\_churn, churn ~ .)

## Warning: Engine set to `MASS`.

## Warning in lda.default(x, grouping, ...): variables are collinear

Calling the parsnip:: model object gives us information about the model fit. We can see the prior probabilities as well as the *coefficients of linear discriminants*. These provide the linear combination of the variables used to create the decision rule. If the combination is large, the model will predict an increase.

lda\_churn$fit$scaling %>%  
 as\_tibble(rownames = "term") %>%  
 arrange(LD1)

## # A tibble: 30 x 2  
## term LD1  
## <chr> <dbl>  
## 1 contractOne year -0.507  
## 2 contractTwo year -0.274  
## 3 tech\_supportYes -0.273  
## 4 online\_securityYes -0.228  
## 5 online\_backupYes -0.139  
## 6 dependents -0.129  
## 7 tech\_supportNo internet service -0.113  
## 8 internet\_serviceNo -0.113  
## 9 streaming\_tvNo internet service -0.113  
## 10 streaming\_moviesNo internet service -0.113  
## # … with 20 more rows

We see similar important terms as the logistic regression model. Customers on long contracts are less likely to churn.

lda\_preds <- predict(lda\_churn, test\_churn, type = "class")   
test\_churn %>%  
 bind\_cols(lda\_preds) %>%  
 metrics(truth = churn, estimate = .pred\_class)

## # A tibble: 2 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 accuracy binary 0.787  
## 2 kap binary 0.444

Here we achieve a kap of 42.9%.

### K-Nearest Neighbors

Next, we will utilize the parsnip::nearest\_neighbor() function.

set.seed(40)  
knn\_churn <- nearest\_neighbor(mode = "classification") %>%  
 fit(data = train\_churn, churn ~ .)

## Warning: Engine set to `kknn`.

knn\_preds <- predict(object = knn\_churn, new\_data = test\_churn)  
test\_churn %>%  
 bind\_cols(knn\_preds) %>%  
 metrics(truth = churn, estimate = .pred\_class)

## # A tibble: 2 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 accuracy binary 0.743  
## 2 kap binary 0.345

K-nearest neighbors performs much worse, with a kap of 34.1%.

KNN’s poor performance is most likely due to the predictor variables and true decision boundary not being highly non-linear. In this case, KNN’s flexibility doesn’t match the shape of the data.

### Choosing the model

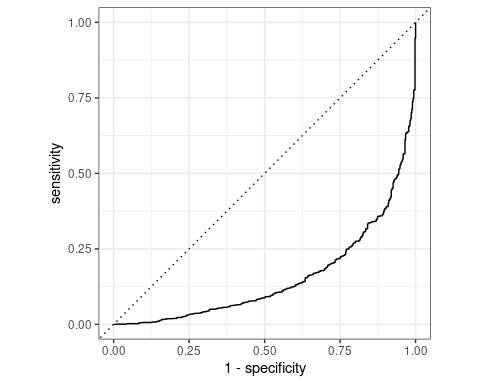
Given logistic regression and LDA exhibiting similar performance metrics, we could use either model. However, logistic regression makes less assumptions about the underlying data.

Let’s take a look at the ROC curve using yardstick::roc\_curve() function.

logi\_preds <- predict(logi\_churn, test\_churn, type = "prob")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

test\_churn %>%  
 bind\_cols(logi\_preds) %>%  
 roc\_curve(churn, .pred\_Yes) %>%  
 autoplot()



Our sensitivity (ability to correctly identify those who will churn) has two visible inflection points where the slope changes, one around ~55% and the other around ~80%. What should we pick as our classifier threshold? This will depend, again, estimate the probability of churn, then decide what to do, that might change from week to week or based on what you are doing to prevent churn.

### Evaluating the threshold

Choosing a classification threshold greatly depends on the problem context.

Imagine we want to hand out an incentive to customers we believe will churn. This incentive costs us money, but has a 100% success rate of retaining a customer (turning them from a Yes to No churn).

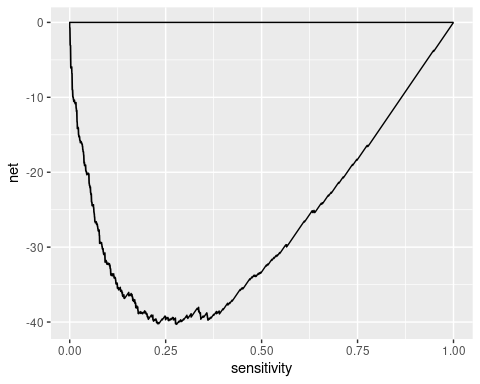
Let’s assume the cost of this incentive for us is $75, but the value of retaining the user is $150. We want to maximize the return we get from this incentive.

In this case, every true positive will net us $75 of profit, and every false positive will cost us $75 (the customer doesn’t churn, so giving the incentive was a waste). We want to find the optimal decision boundary where we maximize our return from this incentive.

ltv = 150  
incentive = 75  
logi\_class <- predict(logi\_churn, test\_churn, type = "class")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from a rank-deficient fit may be misleading

costs <- test\_churn %>%  
 bind\_cols(logi\_preds) %>%  
 bind\_cols(logi\_class) %>%  
 roc\_curve(churn, .pred\_Yes) %>%  
 mutate(profit = sensitivity \* (ltv - incentive),  
 cost = (1 - specificity) \* incentive,  
 net = profit - cost)  
costs %>%  
 ggplot(aes(x = sensitivity, y = net)) +  
 geom\_line() +  
 geom\_line(aes(y = max(net)))



We can grab the optimal sensitivity and find the corresponding decision threshold.

optimal\_sens <- costs %>%  
 filter(net == max(net))  
optimal\_sens

## # A tibble: 3 x 6  
## .threshold specificity sensitivity profit cost net  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 -Inf 0 1 75 75 0  
## 2 0.00108 0 1 75 75 0  
## 3 Inf 1 0 0 0 0

We find that our ideal sensitivity of 76.4% corresponds with a specificity of 75.7%. For whatever reason, roc\_curve() gives the .threshold column the predicted No churn class (despite specifying an ROC curve with .pred\_Yes column). Thus, if we estimate a customer to have a churn percentage greater or equal to 1 - 0.308 = 0.692, we can offer them this incentive.

## Conclusion

We demonstrated how a classification model could be used to solve a real-world problem in which we want to maximize the value of sending incentives to customers that are likely to churn.