797ML Handbook

Steve Linberg

2022-04-17

Contents

Chapter 1

About

This book is being written as part of a final project for 797ML at UMass Amherst, spring 2022. It contains a simple reference and breakdown for a couple of dozen core methods used in machine learning.

The intent is twofold:

1. Serve as a reference for the basics of the material covered in the class, using language and examples that are as simple as possible to explain the core concepts and how to do them;
2. Force myself to learn these techniques better by carrying out the above.

The main purpose of this work is to be *simple*, not to be *comprehensive*. We won't cover every facet of every technique, or every possible permutations of outcomes. The goal is to simply express the broad strokes and core concepts in a way that can be easily remembered, and to serve as a jumping-off point when more information is needed.

1.1 Authoring guidelines

The goal is to have no more than a few short paragraphs for each section, and to keep each explanation of the meanings of outcome variables to one sentence each.

Each chapter will have a TL;DR section at the top with *one-sentence answers* to following questions:

- What it does (answer the question starting with "It…")
- When to do it
- How to do it
- How to assess it

and it will be possible to skim through the book, just surfing the TL;DR at the top of each chapter, and get a reasonable overview without reading the rest. The rest of each chapter will deliver more fleshed-out, but *short*, answers to the same questions, covering the basics and the most general concepts.

1.2 Contact

Steve Linberg: steve@slinberg.net || https://slinberg.net

Project home: https://stevelinberg.github.io/797MS-handbook/

Github repo: https://github.com/stevelinberg/797MS-handbook

1.3 Resources

A lot of the material from this work is from the class textbook, James et al. [2021]. I also find UNC geneticist Josh Starmer's StatQuest video series on YouTube to be immensely helpful for simple explanations of statistics and machine learning concepts. His website statquest.org has many additional useful resources.

Chapter 2

Simple Linear Regression

2.1 TL;DR

What it does Looks to see how well a single predictor variable predicts an outcome, like *how well do years of education predict salary?*

When to do it When you want to see if pretty much the simplest possible model provides enough of an explanation of variance for your purposes

How to do it With the `lm()` function, among other ways

How to assess it Look for a significant p-value for the predictor, and a reasonable R^2

Note "Linear" does not necessarily mean "straight" in this context; a linear regression line can curve.

2.2 What it does

Simple linear regression is where it all begins; among the simplest of all of the regression techniques in analysis, which attempts to estimate a slope and an intercept line for a set of observations using a single predictor variable X and an output variable Y. It uses ordinary least squares (OLS) to build its model, looking for the line through the mean of X and Y that has the smallest sum of squares between the predicted and observed values.

The figure above shows a plot of a simple linear regression, attempting to use the variable `TV` to predict `Sales`. The blue line is the line defined by the regression with its Y intercept and slope; the red dots are the actual observations of `Sales` for each measure of the predictor `TV` on the X axis. The thin blue lines are the error in the prediction.

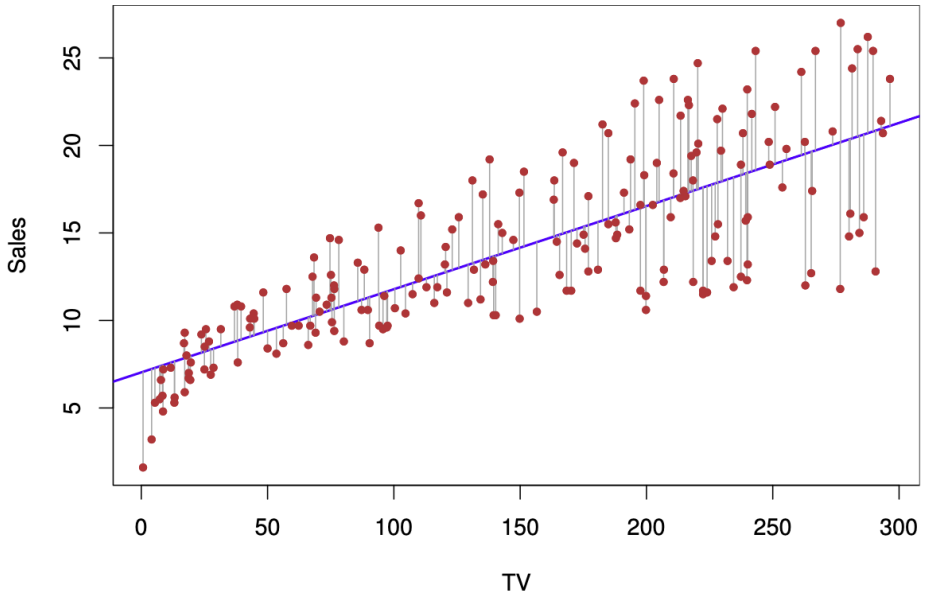


Figure 2.1: Simple linear regression plot (source: James et al. [2021], p. 62)

2.3 When to do it

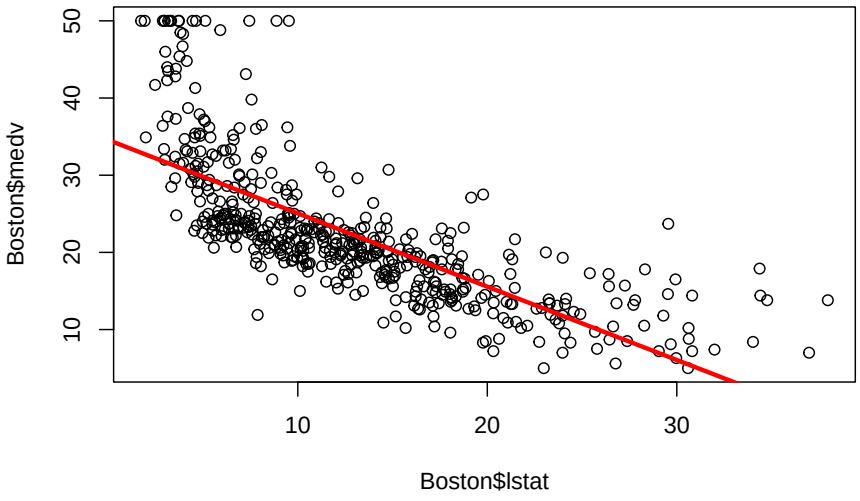
It is a simple first step for looking at data to see if there is an easy single-variable model that does a reasonable job predicting outcomes using one predictor variable. Sometimes, it can be good enough! It has the advantage of being easy to execute, to understand and to communicate, and the value of these factors should not be underestimated. Communicating with non-specialists is an important aspect of a data scientist's job.

Linear regression requires a dataset with a continuous outcome variable; it is easiest and most effective if the predictor variable is also numeric, whether continuous or discrete. It is possible to do linear regression with non-numeric predictors, such as true/false or ordered responses, by converting the predictors to a numeric scale.

2.4 How to do it

This example from James et al. [2021] shows a simple linear regression of **medv** onto **lstat**, attempting to predict median housing prices from percentage of "lower-status" population in the Boston data set from the late 1970s.

```
lm.fit <- lm(medv ~ lstat, data = Boston)
plot(Boston$lstat, Boston$medv)
abline(lm.fit, lwd = 3, col = "red")
```



The results of the regression are stored in the output of `lm()`, and may be viewed with `summary()`:

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

2.5 How to assess it

The output contains a lot of information, but the key points are:

- `Adjusted R-squared`: 0.5432 is the percentage of the variation in the model that is explained by the fit's prediction, compared to just looking at how much the observations vary from their own average with no predictor variable.
- `Residual standard error`: 6.216 is "the average amount that the response will deviate from the true regression line", or the standard deviation of the error. In this case, since the unit of `medv` is in thousands of dollars, it means the average prediction will be off by $6,216.
- `p-value`: $<$ 2.2e-16 (basically zero) means that the model is (very) statistically significant, with a near-zero percent chance that the data in this set could have resulted from a random draw from the (unknown) population if there were no relationship between `medv` and `lstat`.
- `Estimated intercept` of 34.55 is the Y intercept on the graph, or the estimated value of `medv` when `lstat` is zero.
- `Estimated (lstat)`: -0.95 is the estimated coefficient of `lstat`, or the amount that `medv` changes by for each unit change in `lstat`.

The intercept and coefficient are β_0 and β_1, respectively, of the linear regression formula

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Substituting the coefficients and variables, we transform this to:

$$\widehat{\text{medv}} = 34.55 - 0.95 \times \text{lstat}$$

The low p-value shows that there is (almost) definitely a relationship between `medv` and `lstat`, but the R^2 of 0.54 shows that the model only explains slightly more than half of the variation in the data. It's not a bad start, but we would probably want to find a model that explains more of it.

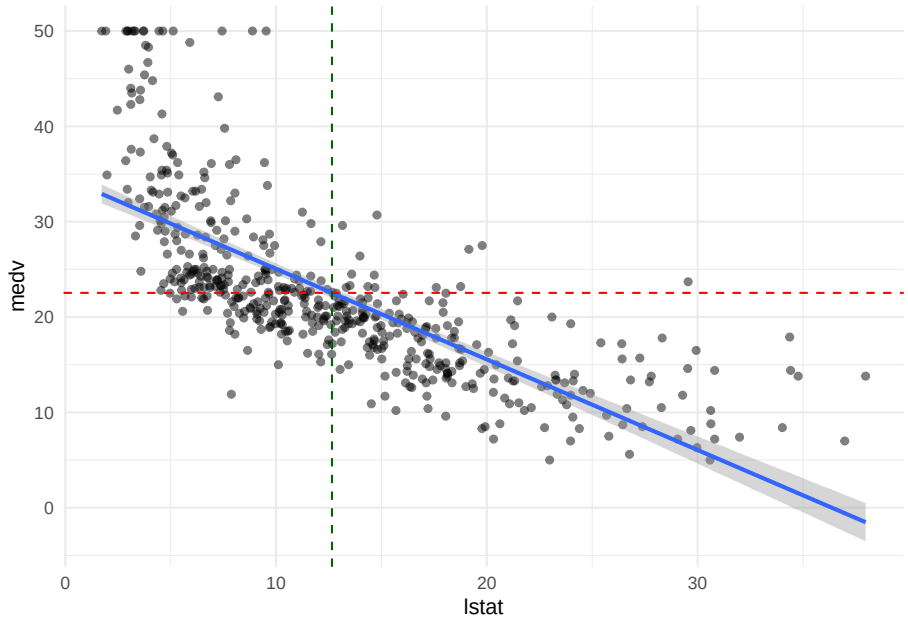
2.6 Where to learn more

- Chapter 3 - 3.1 in James et al. [2021]
- StatQuest: Linear Regression - this goes into good depth on the meaning of the F statistic as well, and how it used to calculate the p value.

2.7 Notes

You can also do a scatterplot in ggplot2 and add a regression line with `geom_smooth()`; it doesn't show the coefficients or other output information, but it gives a quick visual that's a little prettier than the base R plot:

```
ggplot(data = Boston, aes(x = lstat, y = medv)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", formula = y ~ x) +
  geom_hline(yintercept = mean(Boston$medv), linetype = "dashed", color = "red") +
  geom_vline(xintercept = mean(Boston$lstat), linetype = "dashed", color = "darkgreen") +
  theme_minimal()
```



The means of `lstat` and `medv` are shown with dashed red and green lines, showing that the regression line goes through the center of the data.

Chapter 3

Multiple Linear Regression

3.1 TL;DR

What it does Looks to see how well multiple predictor variables predict an outcome, like *how well do years of education and age predict salary?*

When to do it When a simple linear regression doesn't provide a good enough explanation of variance, and you want to see if adding additional variables provides a better one

How to do it With the `lm()` function, utilizing more than one predictor

How to assess it Look for significant p-values for the predictors, and a reasonable adjusted-R^2

3.2 What it does

Multiple linear regression is the first natural extension of simple linear regression. It allows for more than one predictor variable to be specified. It is also possible to combine predictors in interactions, to find out if combinations of predictors have different effects than simply adding them to the model. XXX explain/demo

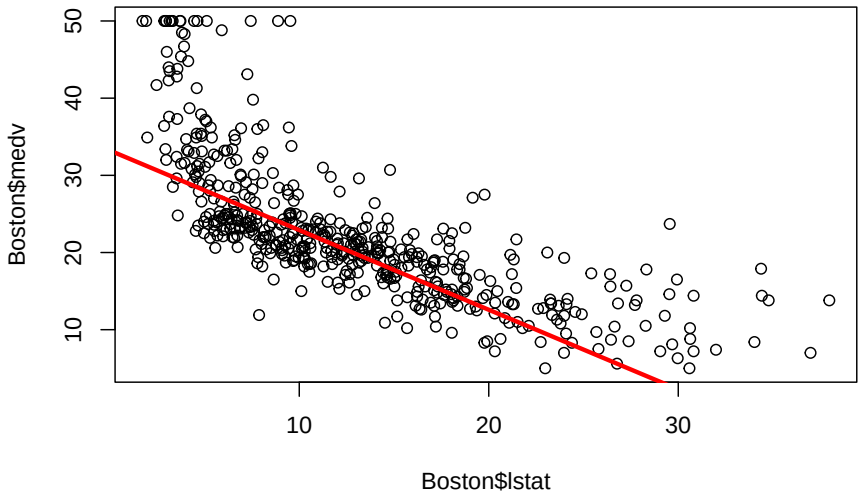
3.3 When to do it

Use multiple linear regression when a simple linear regression doesn't provide a good enough explanation of the variance you're observing, and you want to see if adding more predictors provides a better fit. Typically, this would be in response to either a low R^2 that leaves a lot of unexplained variance, or even just a visual conclusion drawn from seeing a plot of a linear model with an unsatisfactory regression line.

3.4 How to do it

Use the same `lm()` function, with more than one predictor in the formula.

```
lm.fit <- lm(medv ~ lstat + age, data = Boston)
plot(Boston$lstat, Boston$medv)
abline(lm.fit, lwd = 3, col = "red")
```



As with simplr linear regression, the results of the regression are stored in the output of `lm()`, and may be viewed with `summary()`:

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat + age, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.981  -3.978  -1.283   1.968  23.158
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.22276    0.73085  45.458  < 2e-16 ***
## lstat       -1.03207    0.04819 -21.416  < 2e-16 ***
## age          0.03454    0.01223   2.826  0.00491 **
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.173 on 503 degrees of freedom
## Multiple R-squared:  0.5513, Adjusted R-squared:  0.5495
## F-statistic:   309 on 2 and 503 DF,  p-value: < 2.2e-16
```

All of the same parameters are there, with the addition of one coefficient line for each parameter specified in the model. Each coefficient has its own estimate, standard error, t value and p value, and should be evaluated independently.

The main task is to get the ideal subset of parameters that provide the best fit, which can involve a lot of trial and error. *Forward selection* involves starting with a null model (no parameters) and trying all available parameters to see which has the lowest p-value, and adding that and continuing until the parameters are no longer significant or don't reduce the adjusted R^2; *Backward selection* starts with all parameters and removes the least significant one at a time until all remaining parameters are significant; *Mixed selection* starts with forward selection but also removes parameters that lose their significance along the way.

Other tidbits:

- *interaction terms* can be specified with syntax like `lstat * age`, which adds `lstat`, `age` and `lstat * age` as predictors; if an interaction is significant but an individual predictor from the interaction isn't, it should still be left in the model
- *nonlinear transformations* like age^2 can be added by escaping them inside the `I()` function, like `I(age^2)`, to escape the `^` character inside the formula

3.5 How to assess it

With the exception of the additional information for each parameter, the assessment is the same as for simple linear regression.

In the example above, we see that the addition of `age` only added a tiny bit of improvement to the model from the simple linear regression on just `lstat`. Other variables, or combinations of variables, might do a better job.

3.6 Where to learn more

- Chapter 3.2 in James et al. [2021]

Chapter 4

Logistic Regression

4.1 TL;DR

What it does Models the *probability* that an observation falls into one of two categories

When to do it When you are trying to predict a categorical variable with two possible outcomes, like true/false, instead of something continuous like in linear regression

How to do it With the `glm()` function, specifying `family = "binomial"`

How to assess it Look for a significant p-value for the predictor, and assess its accuracy against training data

4.2 What it does

Logistic regression is used to make a prediction about whether an observation falls into one of two categories. This is an alternative to linear regression, which predicts a continuous outcome. A logistic regression results in a model that gives the probability of Y given X.

4.3 When to do it

As the TL;DR says: When you are trying to predict a categorical variable with two possible outcomes, like true/false, instead of something continuous.

4.4 How to do it

First, it's important to note that logistic regression will give better results when the model is fit using *held out* or *training* data, and then tested against other

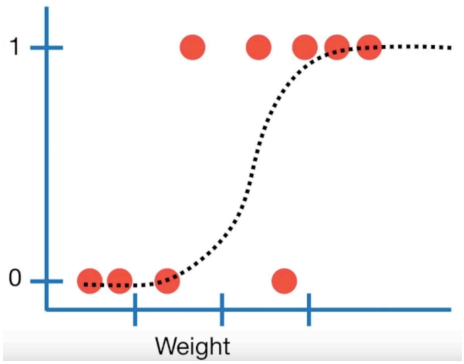


Figure 4.1: Linear regression plot (source: StatQuest)

data, to help avoid overfitting. It is not required, but it's recommended where possible.

First, before that, fit a model using `glm()`, making sure in this case that we have a categorical value to test for:

```
data(Boston)
boston <- Boston %>%
  mutate(
    # Create the categorical crim_above_med response variable
    crim_above_med = as.factor(ifelse(crim > median(crim), "Yes", "No")),
    # Also make a numeric version of crim_above_med for the plot
    crim_above_med_num = ifelse(crim > median(crim), 1, 0)
  )
boston.fits <- glm(crim_above_med ~ nox, data = boston, family = binomial)
summary(boston.fits)
```

```
##
## Call:
## glm(formula = crim_above_med ~ nox, family = binomial, data = boston)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.27324  -0.37245  -0.06847   0.39620   2.53124
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -15.818      1.386  -11.41   <2e-16 ***
## nox           29.365      2.599   11.30   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 701.46  on 505  degrees of freedom
## Residual deviance: 320.39  on 504  degrees of freedom
## AIC: 324.39
##
## Number of Fisher Scoring iterations: 6
```

We can see the that p value for the `nox` variable is nearly 0, suggesting a strong association between `nox` and `crim_above_med`.

Now split the data into training and test sets, using, say, 80% of the data for training:

```
set.seed(1235)
boston.training <- rep(FALSE, nrow(boston))
boston.training[sample(nrow(boston), nrow(boston) * 0.8)] <- TRUE
boston.test <- !boston.training
```

Fit the above model to the training data:

```
boston.fits <-
  glm(
    crim_above_med ~ nox,
    data = boston,
    subset = boston.training,
    family = binomial
  )
```

And use the fit to predict the test data with `predict()`:

```
boston.probs <- predict(boston.fits, boston[boston.test,], type = "response")
```

4.5 How to assess it

Assess the accuracy of the predictions by building a confusion matrix, using the same values ("No" and "Yes" in this case) as contained in the outcome variable we want to predict:

```
boston.pred <- rep("No", sum(boston.test))
boston.pred[boston.probs > 0.5] <- "Yes"
table(boston.pred, boston[boston.test,]$crim_above_med)
```

```
##
## boston.pred No Yes
##         No  36  12
##         Yes  5  49
```

The above table shows that the model correctly predicted `crim_above_med` from

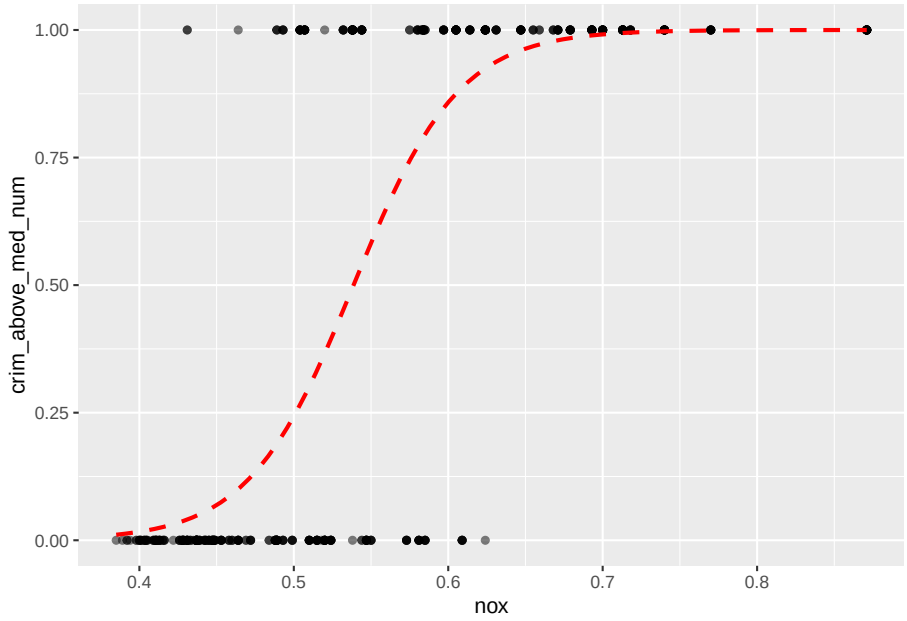
`nox` 85 times out of 102, with 5 false positives and 12 false negatives. The accuracy of the model can be computed via the `mean()` of how often the predictions match the test data, a neat trick:

```
mean(boston.pred == boston[boston.test,]$crim_above_med)
```

```
## [1] 0.8333333
```

The regression can be plotted with `ggplot2` and the `stat_smooth()` function, utilizing the `glm` method, like so:

```
ggplot(boston, aes(x = nox, y = crim_above_med_num)) +
  geom_point(alpha = .5) +
  stat_smooth(
    method = "glm",
    formula = y ~ x,
    se = FALSE,
    method.args = list(family = binomial),
    col = "red",
    lty = 2
  )
```



This shows that an observation with a `nox` value of 0.6 would have around a 90% probability of `crim_above_med` being true, or 1, while an observation with a `nox` value of 0.5 would only have around a 25% probability of `crim_above_med` being true.

4.6 Where to learn more

- Chapter 4 - 4.4 in James et al. [2021]
- StatQuest: Logistic Regression

4.7 Notes

Like with linear regression, the "multiple" flavor of logistic regression works much the same way, with the specification of additional predictors in the formula, and so doesn't really warrant a whole chapter of its own. The selection of variables requires some effort, and is addressed under best subser selection.

Chapter 5

Linear Discriminant Analysis (LDA)

5.1 TL;DR

What it does Separates observations into categories, similar logistic regression, but creating actual groups (separated by a line) rather than per-observation probabilities

When to do it When exploring various classifiers to see which works best for a given data set

How to do it With the `lda()` function from the MASS library, using training and testing sets

How to assess it Assess the accuracy of the predictions on the test set after training

5.2 What it does

Similar to principal component analysys (PCA), LDA reduces dimensionality and finds the best single axis to separate two (or more) groups of observations into categories using a least-squares method of distance from a mean. Like PCA, it returns a set of new axes for the data, organized by importance, so that the first axis is the one that explains the largest amount of variance, and so on, down to $p - 1$ axes where p is the number of categories/dimensions.

So, LDA will return 3 axes for a set of observations with 4 categories, or 1 for a set of observations with 2 categories. Each axis will have a loading score that indicates which variables had the biggest impacts on it.

5.3 When to do it

When you want to see if it will work better than other classification methods! It should always be tried along with other classifiers like logistic regression, quadratic discriminant analysis, and Naive Bayes.

5.4 How to do it

Again, using the Boston data from the logistic regression chapter:

```
data(Boston)
boston <- Boston %>%
  mutate(
    # Create the categorical crim_above_med response variable
    crim_above_med = as.factor(ifelse(crim > median(crim), "Yes", "No")),
  )
```

We again split the data into training and test sets:

```
set.seed(1235)
boston.training <- rep(FALSE, nrow(boston))
boston.training[sample(nrow(boston), nrow(boston) * 0.8)] <- TRUE
boston.test <- !boston.training
```

And fit the above model to the training data using the `lda()` function from the MASS library[1] (note: there is no `family` argument as with `glm()` in logistic regression, but the calls are otherwise identical):

```
boston_lda.fits <-
  lda(
    crim_above_med ~ nox,
    data = boston,
    subset = boston.training,
  )
```

5.5 How to assess it

The fit can be directly examined:

```
boston_lda.fits
```

```
## Call:
## lda(crim_above_med ~ nox, data = boston, , subset = boston.training)
##
```

[1] Note to my fellow New Englanders: the MASS library is named for a book titled "Modern Applied Statistics with S", and although it does contain an earlier version of the Boston dataset (among many others), it has nothing to do with Massachusetts, and our Boston dataset comes from the ISLR2 library.

```
## Prior probabilities of groups:
##        No       Yes
## 0.5247525 0.4752475
##
## Group means:
##           nox
## No  0.4710519
## Yes 0.6357865
##
## Coefficients of linear discriminants:
##          LD1
## nox 12.29052
```

In the above output, the `Prior probabilities of groups` information is just the percentage of observations in each of the outcome categories:

```
> mean(boston[boston.training,]$crim_above_med == "Yes")
[1] 0.4752475
> mean(boston[boston.training,]$crim_above_med == "No")
[1] 0.5247525
```

The `Group means` section shows the means of the predictor variable (`nox` in this case) for each of the categories:

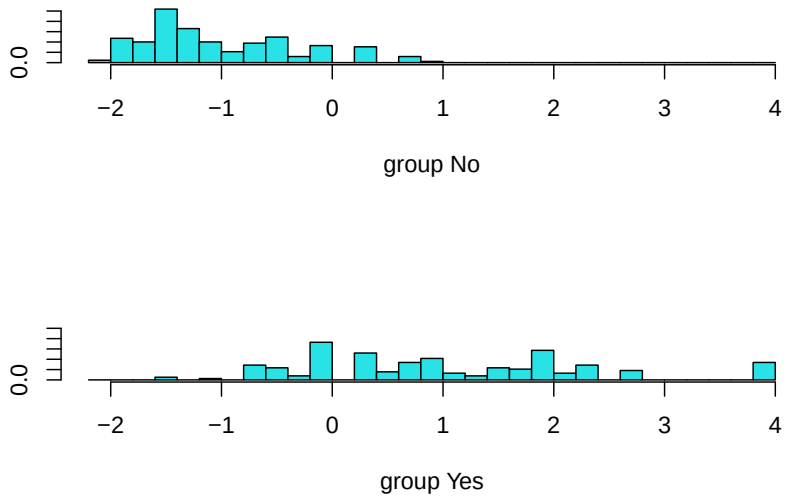
```
boston[boston.training, ] %>%
  group_by(crim_above_med) %>%
  summarize("Group means" = mean(nox))
```

```
## # A tibble: 2 x 2
##   crim_above_med `Group means`
##   <fct>                  <dbl>
## 1 No                     0.471
## 2 Yes                    0.636
```

And the "Coefficient" is the linear coefficient for the predictor variable `nox`, which determines the slope of the line that separates the categories.

The fit can be plotted:

```
plot(boston_lda.fits)
```



The fit should be used to predict the test data, and the model can be assessed on its results:

```
boston_lda.pred <- predict(boston_lda.fits, boston[boston.test, ])
table(boston_lda.pred$class, boston[boston.test, ]$crim_above_med)
```

```
##
##       No Yes
##   No  38  12
##   Yes  3  49
```

In this example, LDA made the correct categorization 87 times out of 102, with 3 false positives and 12 false negatives. Again, we can compute the prediction accuracy by the mean of the correct-to-wrong guesses:

```
mean(boston_lda.pred$class == boston[boston.test, ]$crim_above_med)
```

```
## [1] 0.8529412
```

If the accuracy rate is good enough for your purposes, you can use the fit in the same manner to make predictions for new data.

5.6 Where to learn more

- Chapter 4.4.1 - 4.4.2 in James et al. [2021]
- StatQuest: Linear Discriminant Analysis

Chapter 6

Quadratic Discriminant Analysis

6.1 TL;DR

What it does Assigns observations to categories like linear discriminant analysis, but with a more flexible / curved boundary rather than a straight line

When to do it When linear discriminant analysis isn't giving good enough results (esp. if the dataset is very large)

How to do it With the `qda()` function from the MASS library, using training and testing sets

How to assess it Assess the accuracy of the predictions on the test set after training

6.2 What it does

Just like linear discriminant analysis, quadratic discriminant analysis attempts to separate observations into two or more classes or categories, but it allows for a curved boundary between the classes. Which approach gives better results depends on the shape of the Bayes decision boundary for any particular dataset. The following figure from James et al. [2021] shows the two approaches:

The purple dashed line in both diagrams shows the Bayes decision boundary. The LDA line in the left plot shows this class separation line as straight; the QDA line on the right is curved.

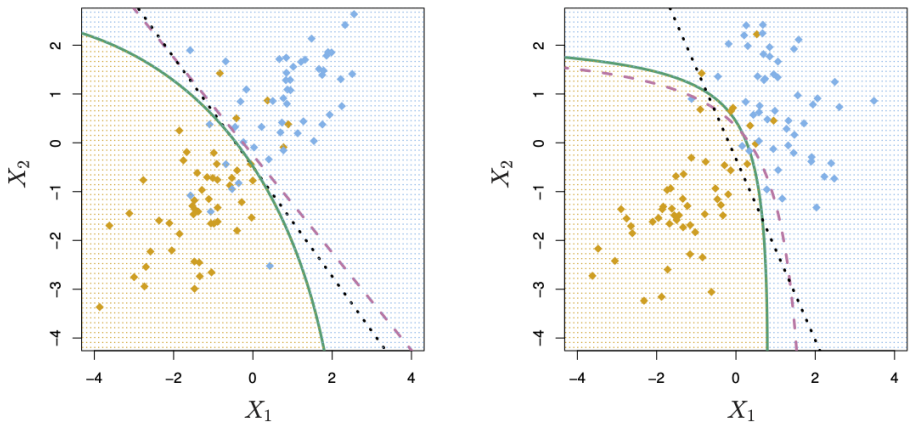


Figure 6.1: LDA and QDA plots (source: James et al. [2021], p. 154)

6.3 When to do it

Use QDA when the linear separation of linear discriminant analysis isn't giving good enough results, and you think a curved (or quadratic) line might fit better.

6.4 How to do it

Exactly like linear discriminant analysis, except for the `qda()` call instead of `lda()`. Train the model on training data, assess it against test data.

```
data(Boston)
boston <- Boston %>%
  mutate(
    # Create the categorical crim_above_med response variable
    crim_above_med = as.factor(ifelse(crim > median(crim), "Yes", "No")),
  )
```

We again split the data into training and test sets:

```
set.seed(1235)
boston.training <- rep(FALSE, nrow(boston))
boston.training[sample(nrow(boston), nrow(boston) * 0.8)] <- TRUE
boston.test <- !boston.training
```

and call `qda()` on the training data:

```
boston_qda.fits <-
  qda(
    crim_above_med ~ nox,
    data = boston,
```

```
    subset = boston.training,
  )
```

6.5 How to assess it

As with `lda()`, the resulting fit can be directly examined:

```
boston_qda.fits
```

```
## Call:
## qda(crim_above_med ~ nox, data = boston, , subset = boston.training)
##
## Prior probabilities of groups:
##        No       Yes
## 0.5247525 0.4752475
##
## Group means:
##           nox
## No  0.4710519
## Yes 0.6357865
```

Note that in this example, the `Prior probabilities of groups` and `Group means` assessments are identical to those from `lda()`. There is no linear discriminant coefficient, because this is not a linear discriminant! The rest of the output is identical.

We then call `predict` on the QDA fit, to predict the `crim_above_med` variable from `nox`:

```
boston_qda.pred <- predict(boston_qda.fits, boston[boston.test, ])
table(boston_qda.pred$class, boston[boston.test, ]$crim_above_med)
```

```
##
##       No Yes
##   No  38  12
##   Yes  3  49
```

In this example, just like LDA, QDA made the correct categorization 87 times out of 102, with 3 false positives and 12 false negatives. And again, we can compute the prediction accuracy by the mean of the correct-to-wrong guesses:

```
mean(boston_qda.pred$class == boston[boston.test, ]$crim_above_med)
```

```
## [1] 0.8529412
```

In this case, QDA offered no increase in prediction accuracy over LDA, so there is no reason to prefer it (although it also performed no poorer).

6.6 Where to learn more

- Chapter 4.4.3 in James et al. [2021]

Chapter 7

Naive Bayes

7.1 TL;DR

What it does Provides yet another way to try to classify observations into one or more categories

When to do it When you're rounding out your exhaustive try-them-all process to try to determine the best classifier for your data after linear discriminant analysis and quadratic discriminant analysis

How to do it With the `naiveBayes()` function from the e1071 library

How to assess it As with the other methods, assess it on test data after training and see if its accuracy is better than other methods

7.2 What it does

Naive Bayes makes the assumption that there is no association between the predictors. According to James et al. [2021], this still yields "decent results" even though that's generally not true, especially in datasets where the number of observations n is smallish relative to the number of variables p.

7.3 When to do it

When you're rounding our your assessment of the accuracy of the various classification methods looking for the best one.

TODO: there must be a better explanation for this.

7.4 How to do it

Exactly like linear discriminant analysis and quadratic discriminant analysis, except for the `naiveBayes()` call instead of `qda()` or `lda()`. Train the model on training data, assess it against test data.

```
data(Boston)
boston <- Boston %>%
  mutate(
    # Create the categorical crim_above_med response variable
    crim_above_med = as.factor(ifelse(crim > median(crim), "Yes", "No")),
  )
```

We again split the data into training and test sets:

```
set.seed(1235)
boston.training <- rep(FALSE, nrow(boston))
boston.training[sample(nrow(boston), nrow(boston) * 0.8)] <- TRUE
boston.test <- !boston.training
```

and call `naiveBayes()` on the training data:

```
boston_nb.fit <-
  naiveBayes(
    crim_above_med ~ nox,
    data = boston,
    subset = boston.training,
  )
```

7.5 How to assess it

```
boston_nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        No       Yes
## 0.5247525 0.4752475
##
## Conditional probabilities:
##      nox
## Y          [,1]       [,2]
```

```
##   No  0.4710519 0.05635737
##   Yes 0.6357865 0.10210051
```

In the output above, we see our `A-priori probabilities`, which like the `Prior probabilities` of `lda()` and `qda()` show the percentage of observations in each of the outcome categories:

```
> mean(boston[boston.training,]$crim_above_med == "Yes")
[1] 0.4752475
> mean(boston[boston.training,]$crim_above_med == "No")
[1] 0.5247525
```

The `Conditional probabilities` are identical to the `Group means` from `lda()` and `qda()`, showing the means of the predictor variable (`nox` in this case) for each of the categories:

```
boston[boston.training, ] %>%
  group_by(crim_above_med) %>%
  summarize("Group means" = mean(nox))
```

```
## # A tibble: 2 x 2
##   crim_above_med `Group means`
##   <fct>                  <dbl>
## 1 No                     0.471
## 2 Yes                    0.636
```

All of these results are the same as `lda()` and `qda()`. The Naive Bayes output also includes a second column in the `Conditional probabilities` section, which is the standard deviation for the (in this case `nox`) variable in each category.

After the fit is built, we can run prediction as with the other methods, to predict the `crim_above_med` variable from `nox` (note that we just make a table of the prediction output itself, without the `$class` attribute):

```
boston_nb.pred <- predict(boston_nb.fit, boston[boston.test, ])
table(boston_nb.pred, boston[boston.test, ]$crim_above_med)
```

```
##
## boston_nb.pred No Yes
##            No  38  12
##            Yes  3  49
```

In this example, just like LDA and QDA, Naive Bayes made the correct categorization 87 times out of 102, with 3 false positives and 12 false negatives. And again, we can compute the prediction accuracy by the mean of the correct-to-wrong guesses:

```
mean(boston_nb.pred == boston[boston.test, ]$crim_above_med)
```

```
## [1] 0.8529412
```

We see again here that Naive Bayes performed identically to `lda()` and `qda()`. This means we probably need a better example for this document.

7.6 Where to learn more

- Chapter 4.4.4 in James et al. [2021]

Chapter 8

K-Nearest Neighbors

8.1 TL;DR

What it does K-Nearest Neighbors classifies observations by judging their proximity to (k) other nearby classified neighbors.

When to do it When you're working through all of the various classification methods to see which one works the best for your data

How to do it With the `knn()` function from the class library

How to assess it Check the prediction accuracy, as with the other classification methods

8.2 What it does

K-Nearest Neighbors starts with known categories of clustered data, and assigns new data to one of these categories by grouping it with its (k) nearest neighbor(s). This is judged in whatever dimensional space is appropriate to the number of predictor variables; if only using one predictor, as in the example here, then the points are simply arranged on a line; 2 predictors puts them on a plane, and so forth.

Once the new observation is located against the known categories, a sort of "lasso" extends out from the point in all available directions until it contains k other neighbors. Whichever category has the most points is assigned to the new observation.

8.3 When to do it

When you're still looking for the best classification method for your data. Maybe this will be it this time!

8.4 How to do it

We will continue with the Boston data set and the same training split.

```
data(Boston)
boston <- Boston %>%
  mutate(
    # Create the categorical crim_above_med response variable
    crim_above_med = as.factor(ifelse(crim > median(crim), "Yes", "No")),
  )
```

We again split the data into training and test sets:

```
set.seed(1235)
boston.training <- rep(FALSE, nrow(boston))
boston.training[sample(nrow(boston), nrow(boston) * 0.8)] <- TRUE
boston.test <- !boston.training
```

Unlike the other classification methods used so far, the `knn()` function needs the data columns in a simple matrix. Also, we supply the training and test data together in a single call to `knn()`.

So first, create the matrices from the data columns. We only have one column in our model here, so we need to convert it to a `data.frame` because `knn()` won't accept a plain vector:

```
train.X <- as.data.frame(boston[boston.training, ]$nox)
test.X <- as.data.frame(boston[boston.test, ]$nox)
```

If we had more than one column, we could instead bind them with `cbind()`:

```
train.X <- cbind(boston$nox, boston$something_else)[boston.training, ]
test.X <- cbind(boston$nox, boston$something_else)[boston.test, ]
```

Note that it is highly recommended to standardize all columns to the same scale when using more than one, unless a weighted scale is desired, and even then it would likely be better to standardize first and then scale from there.

```
train.X <- cbind(scale(boston$nox), scale(boston$something_else))[boston.training, ]
test.X <- cbind(scale(boston$nox), scale(boston$something_else))[boston.test, ]
```

We also extract the outcome variable into a vector.

```
# Also extract the outcome variable.
train.crim_above_med <- boston$crim_above_med[boston.training]
```

Now set a seed and run the `knn()` function. Select a value for k, the number of nearest neighbors to use in the classifier; here we will start with 1, which means each new observation will be classified according to its single closest neighbor. (Higher values will decide based on the majority of the k nearest

neighbors; keeping k prime, if more than 1, will avoid ties; alternately, ties can be randomly assigned.)

```
set.seed(1235)
knn.pred <- knn(train.X, test.X, train.crim_above_med, k = 1)
```

8.5 How to assess it

Check the table of predictions against the test data and see how it did.

```
table(knn.pred, boston[boston.test,]$crim_above_med)
```

```
##
## knn.pred No Yes
##      No  40   2
##      Yes  1  59
```

```
mean(knn.pred == boston[boston.test,]$crim_above_med)
```

```
## [1] 0.9705882
```

In this case, we achieved 97% accuracy with `knn()` and $k = 1$. If we didn't get a good enough result, we could run `knn()` again with different values for k to see if we could improve the results. The best result here can then be compared against the results with other classification methods.

(It's unusual for $k = 1$ to be the best number, as it will tend to overfit; some exploration will be needed to find the best results, and it's simple enough to try a range in a loop.)

8.6 Where to learn more

- Chapter 4.7.6 (lab) in James et al. [2021]
- StatQuest: K nearest neighbors

Chapter 9

TODO: Poisson Regression

9.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

9.2 What it does

TODO

9.3 When to do it

TODO

9.4 How to do it

TODO

9.5 How to assess it

TODO

9.6 Where to learn more

TODO

Chapter 10

TODO: Cross-Validation

10.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

10.2 What it does

TODO

10.3 When to do it

TODO

10.4 How to do it

TODO

10.5 How to assess it

TODO

10.6 Where to learn more

TODO

Chapter 11

TODO: Bootstrap

11.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

11.2 What it does

TODO

11.3 When to do it

TODO

11.4 How to do it

TODO

11.5 How to assess it

TODO

11.6 Where to learn more

TODO

Chapter 12

TODO: Best Subset Selection

12.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

12.2 What it does

TODO

12.3 When to do it

TODO

12.4 How to do it

TODO

12.5 How to assess it

TODO

12.6 Where to learn more

TODO

Chapter 13

TODO: Stepwise Selection

13.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

13.2 What it does

TODO

13.3 When to do it

TODO

13.4 How to do it

TODO

13.5 How to assess it

TODO

13.6 Where to learn more

TODO

Chapter 14

TODO: Ridge Regression

14.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

14.2 What it does

TODO

14.3 When to do it

TODO

14.4 How to do it

TODO

14.5 How to assess it

TODO

14.6 Where to learn more

TODO

Chapter 15

TODO: Lasso

15.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

15.2 What it does

TODO

15.3 When to do it

TODO

15.4 How to do it

TODO

15.5 How to assess it

TODO

15.6 Where to learn more

TODO

Chapter 16

TODO: Principal Component Regression

16.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

16.2 What it does

TODO

16.3 When to do it

TODO

16.4 How to do it

TODO

16.5 How to assess it

TODO

16.6 Where to learn more

TODO

Chapter 17

TODO: Bagging

17.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

17.2 What it does

TODO

17.3 When to do it

TODO

17.4 How to do it

TODO

17.5 How to assess it

TODO

17.6 Where to learn more

TODO

Chapter 18

TODO: Random Forests

18.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

18.2 What it does

TODO

18.3 When to do it

TODO

18.4 How to do it

TODO

18.5 How to assess it

TODO

18.6 Where to learn more

TODO

Chapter 19

TODO: Boosting

19.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

19.2 What it does

TODO

19.3 When to do it

TODO

19.4 How to do it

TODO

19.5 How to assess it

TODO

19.6 Where to learn more

TODO

Chapter 20

TODO: Bayesian Additive Regression Trees

20.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

20.2 What it does

TODO

20.3 When to do it

TODO

20.4 How to do it

TODO

20.5 How to assess it

TODO

20.6 Where to learn more

TODO

Chapter 21

TODO: Support Vector Machines

21.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

21.2 What it does

TODO

21.3 When to do it

TODO

21.4 How to do it

TODO

21.5 How to assess it

TODO

21.6 Where to learn more

TODO

Chapter 22

TODO: Principal Component Analysis

22.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

22.2 What it does

TODO

22.3 When to do it

TODO

22.4 How to do it

TODO

22.5 How to assess it

TODO

22.6 Where to learn more

TODO

Chapter 23

TODO: K-Means Clustering

23.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

23.2 What it does

TODO

23.3 When to do it

TODO

23.4 How to do it

TODO

23.5 How to assess it

TODO

23.6 Where to learn more

TODO

Chapter 24

TODO: Hierarchical Clustering

24.1 TL;DR

What it does TODO
When to do it TODO
How to do it TODO
How to assess it TODO

24.2 What it does

TODO

24.3 When to do it

TODO

24.4 How to do it

TODO

24.5 How to assess it

TODO

24.6 Where to learn more

TODO

Bibliography

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2nd edition, 2021. URL https://link.springer.com/book/10.1007/978-1-0716-1418-1. ISBN 978-1-0716-1417-4.