Business 4720

Regression and Classification Models

Joerg Evermann

Learning Goals

After reading this chapter, you should be able to:

- Explain the importance of visually assessing data before predictive modelling.
- Build a linear regression model, including interaction effects and categorical predictors, and be able to assess its quality using cross-validation methods.
- Explain the goals and the differences between ridge regression, LASSO and Elasticnet
- Build regression models using different penalized linear regressions and evaluate their quality using cross-validation methods.
- Explain logistic regression and the purpose of the link function, including the concepts of log-odds or logits.
- Build a classification model using logistic regression and evaluate its quality using common metrics such as accuracy, and the ROC and AUC.
- Build a classification model using the KNN method and evaluate its quality using common metrics such as accuracy, and the ROC and AUC.

Sources and Further Reading

The material in this chapter is based on the following sources. They are freely available. Consult them for additional information.

Gareth James, Daniel Witten, Trevor Hastie and Robert Tibshirani: *An Introduction to Statistical Learning with Applications in R*. 2nd edition, corrected printing, June 2023. (ISLR2)
`https://www.statlearning.com`
Chapters 2, 3, 4, 5

The book by James et al. provides an easy introduction to machine learning at the introductory undergraduate level. It focuses on applications, not mathematics, and contains many exercises using R. Concepts are well explained and illustrated. There is a similar book available by the same authors with applications in Python. This book is a more accessible of the following book.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman: *The Elements of Statistical Learning*. 2nd edition, 12th corrected printing, 2017. (ESL)
`https://hastie.su.domains/ElemStatLearn/`
Chapters 2, 3, 4, 7

The book by Hastie et al. still sets the standard for statistical learning. It is widely used and cited. It's treatment is more technical than the previous book and there are no exercises in R or Python. However, it covers the concepts in more depth (and a few more formulas). However, it is still very accessible even to an undergraduate audience.

> Kevin P. Murphy: *Probabilistic Machine Learning – An Introduction*. MIT Press 2022.
> https://probml.github.io/pml-book/book1.html
> Chapters 4, 6, 9, 10, 11

Murphy's book is available under a creative-commons license. It is a somewhat more technical treatment of the material, but with many illustrations and examples. It is quite comprehensive in its coverage and targeted at the advanced undergraduate or graduate student.

1 Introduction

This chapter is an introduction to supervised machine learning using R and includes both regression and classification problems. The chapter introduces linear regression and penalized regression models (ridge regression and LASSO). For classification, it introduces the methods of logistic regression and k-Nearest-Neighbours, which was already briefly discussed in the previous chapter. This chapter builds on the introductory material to supervised learning in the previous chapter.

2 Linear Regression

Linear regression fits a simple statistical model to a set of input and output variables. The true model is assumed to take the form:

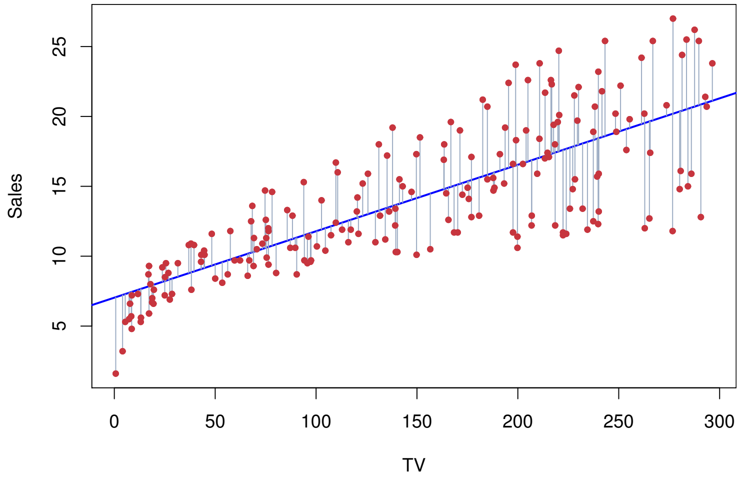
$$f(X) = Y = \beta_0 + \beta_1 X + \epsilon$$

while the fitted, approximate model is:

$$\hat{f}(X) = \hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

The values of $\hat{f}(X) = \hat{Y}$ are called the *fitted values* in statistics or the *predicted values* in machine learning. The difference in terminology reflects the fact that traditional statistics looks back at the training data for the output values determined by the model after it has been fit to the training data, whereas machine learning emphasizes prediction of output values for new inputs.

This equation shows the form of the assumed functional relationship between X and Y is linear in the parameters β. In other words, there are no polynomials or other



Source: ISLR2 Figure 3.1

Figure 1: A linear regression model

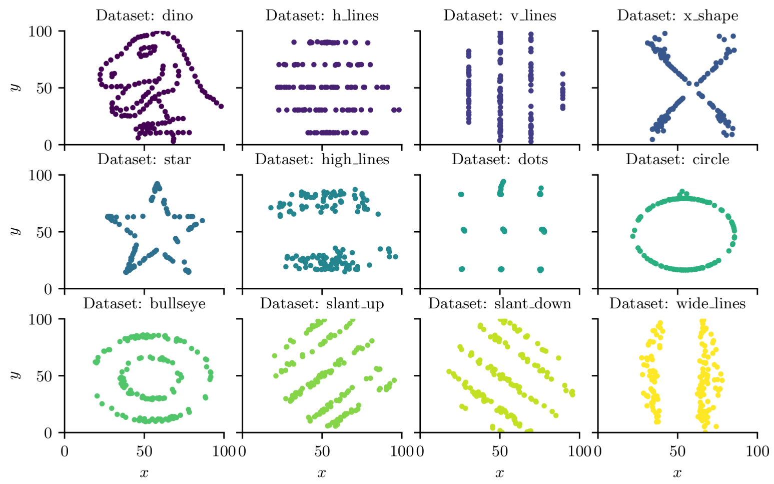
transformations of β, such as β_1^2 or $\log \beta_1$. It is the linearity of the parameters that makes a regression linear, *not* the linearity of the predictors. For example, adding a polynomial term $\beta_2 X^2$ to the right-hand side of this regression equation would still make this a linear regression.

Figure 1 shows the fitted regression line expressed by the intercept ($\hat{\beta}_0$) and slope ($\hat{\beta}_1$) parameters. The distance between this line and the data points (coloured red) represents the error term ϵ in the regression equation. Importantly, the error is the vertical differences in the Y direction between fitted regression line and the data point, *not* the shortest distance between the regression line and the data point.

The first task in performing a regression analysis is to identify the form of the regression model or regression equation. While Figure 1 fitted a model that is linear in X, it is clear that this model is not a good fit for small X values, as the corresponding observations are all below the regression line.

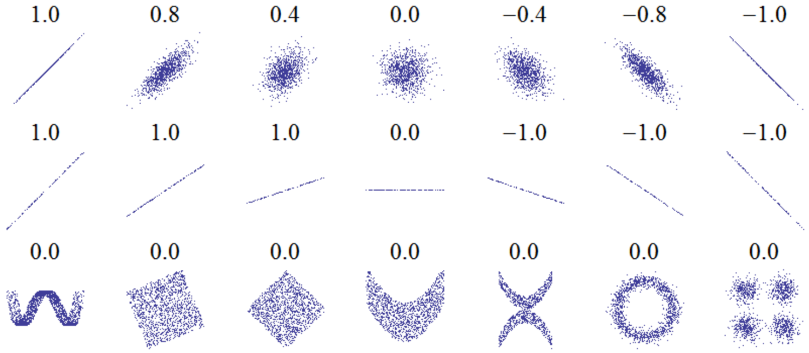
To identify an appropriate functional form, it is insufficient to simply examine summary statistics like correlations or covariances between variables. Figure 2 shows twelve data sets with the same correlation between the two variables. It is clear that none of them can be fitted to a simple moodel that is linear in X. While some, like the circle or the bullseye data set, can be appropriately transformed and fitted to linear regression models that involve polynomials or other functions of X, it is not clear what functional form the dinosaur head might take.

It is also insufficient to fit a model that is linear in X and use the fit as indication for the appropriateness of the model. Consider the data sets shown in Figure 3. The correlations between the two variables are shown above each data set. Visual inspection is sufficient to show that data sets with the same correlation do not necessarily have the



Source: Murphy Figure 2.6

Figure 2: The "Datasaurus Dozen" – All datasets have the same correlation between the two variables



Source: Murphy Figure 3.1

Figure 3: Datasets with the same correlation (as indicated above each dataset) between two variables do not need to have the same regression slope

same regression slope, or should even be fitted to the same linear model.

The objective for estimating the parameters β_0 and β_1 from the data set is to minimize the residual or error, that is, the difference between f and $\hat{f}$. So that positive and negative differences do not compensate for each other, the difference is either squared or the absolute value is taken. The former is called *mean squared error* (MSE) while the latter is called *mean absolute error* (MAE). In statistics, a closely related concept is the *residual sum of squares* (RSS). The MSE is the RSS divided by the number of observations n, and minimizing one also minimizes the other. While the discussion in the previous chapter indicated that the MAE is more robust in that outliers, i.e. observations with large errors, have less influence on the estimated values of the parameters, it is customary to fit linear regression models using the RSS or the MSE.

$$
\begin{aligned}
RSS &= \sum_i \left(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i\right)^2 \\
MSE &= \frac{1}{n} RSS
\end{aligned}
$$

The linear regression model is simple enough that an optimal solution can be derived analytically. The optimal least-squares estimates are:

$$
\begin{aligned}
\hat{\beta}_1 &= \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \\
\hat{\beta}_0 &= \bar{y} - \hat{\beta}_1 \bar{x}
\end{aligned}
$$

where $\bar{x}$ and $\bar{y}$ are the sample means.

An important statistic in linear regression analysis is the proportion of explained variance, designated as R^2. It is defined as:

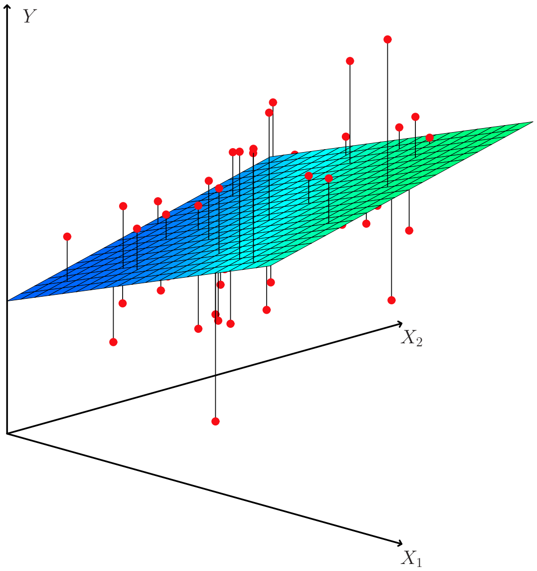
$$
R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}
$$

where $TSS = \sum_i (y_i - \bar{y})^2$ is the total sum of squares. An R^2 value close to 1 indicates that the fitted regression model explains a large proportion of the variability in the data set, that is, it explains the data well. In contrast, a value close to 0 indicates that the fitted regression model explains very little of the observed variability, it does not explain the observed data well. Another important interpretation of the R^2 is as the correlation between the true Y and the fitted or predicted values $\hat{Y}$.

Adding additional predictors into the linear regression model is straightforward, as is the inclusion of qualitative or categorical predictors. For example, a model with two predictors X_1 and X_2 assumes the true form

$$
Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon
$$

and fits a plane to a set of points in three dimensional (X_1, X_2, Y) space, as shown in Figure 4.



Source: ISLR2 Figure 3.4

Figure 4: Example linear regression with two predictors

Qualitative predictors (called *factors* in statistics) wich multiple, exclusive *levels* or categories can be included using *dummy variables*. A categorical variable that can take on k values requires $k - 1$ binary dummy variables. For example, a factor x that has four levels ”a”, ”b”, ”c”, and ”d” might be encoded using three dummy variables as follows:

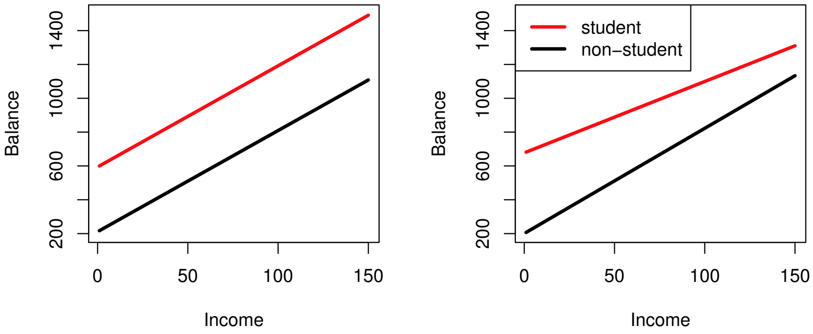
$$x_{i1} = \begin{cases} 1 & \text{level ”a”} \\ 0 & \text{else} \end{cases}$$

$$x_{i2} = \begin{cases} 1 & \text{level ”b”} \\ 0 & \text{else} \end{cases}$$

$$x_{i3} = \begin{cases} 1 & \text{level ”c”} \\ 0 & \text{else} \end{cases}$$

Note that $x_{i1} = x_{i2} = x_{i3} = 0$ represents level ”d”. Such *contrasts* determine how factor levels are coded using dummy variables.

Earlier, it was noted that a linear model is linear in its parameters, not necessarily in its input variables. Hence, it is possible to include transformations of input variables in a linear regression model, such as polynomials, products, or other functional transformations like logarithms, square roots, etc. For example, the model



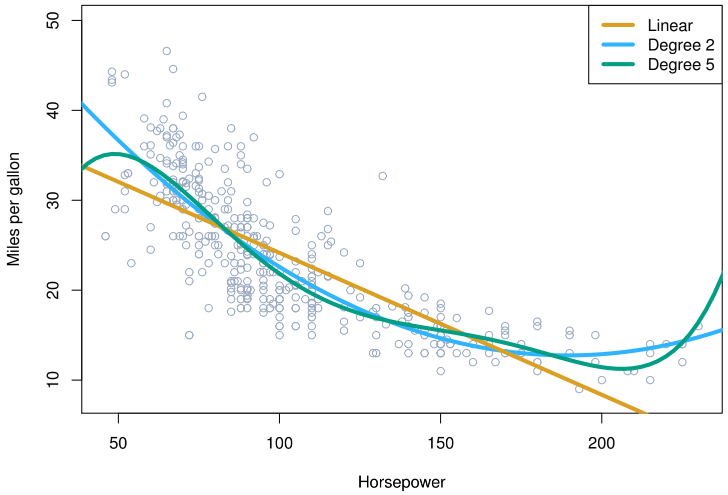
Source: ISLR2 Figure 3.7

Figure 5: Example interaction effect in linear regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_2 + \beta_4 X_1 X_2 + \epsilon$$

is still linear in its parameters β_i. This model also shows the difference between *input variables* and *predictors* or *features* that was alluded to in the previous chater. This model has two input variables X_1 and X_2 but has four predictors or features, X_1, X_1^2, X_2, and X_1X_2. In statistics terminology, the coefficients of single input variables, here β_1 and β_3, are said to represent *main effects*, while the parameters of products of two or more input variables, here β_4 are said to represent *interaction effects*Interaction effect. Interaction effects can include the product of a numerical variable and a dummy binary variable. This results in different regression slopes for different categories or factor levels, as shown in the example in Figure 5 where the left panel shows no interaction effect, that is, the lines are parallel with the same slopes. The right panel shows an interaction effect where the slopes for different categories are different.

An example of polynomial predictors like X^2, X^3 or higher degrees is shown in Figure 6. As the figure shows, and as is true in general, increasing the number of predictors increases the flexibility of the model to fit the data. This results in smaller model bias but at the expense of model variance.



Source: ISLR2 Figure 3.8

Figure 6: Regression example with polynomial predictors

3 Linear Regression in R

Linear regression is part of the basic R system, no libraries need to be installed. The function `lm()` requires a formula representing the regression model and a data frame, and returns a linear model.

The following example uses the `Boston` data set that contains housing prices (”median value”, `medv`) as well as demographic and socioeconomic information for suburbs in the city of Boston. The data is available as part of the `ISRL2` package[1].

First, examine the data, get summary statistics, examine the first few rows of the data frame, and create scatterplots to identify the form of a linear model:

```r
# Data set from the textbook 'Introduction to
# Statistical Learning with Applications in R'
library(ISLR2)

# Get a description of the data
?Boston

# Get a summary and examine first few rows
summary(Boston)
head(Boston)

# Bivariate scatterplots
plot(Boston)
```

[1] The R code for this example is based on material in Section 3.6 of ISLR2

The formula interface to `lm()` is the easiest to use and resembles the way one would write the regression equation. The $\sim$ sign separates the output on the left side from the predictors or features on the right side of the formula. A `1` in the formula represents the intercept (β_0 in the formula above). The intercept is normally added automatically but can be explicitly added or removed (using `-1`) as needed.

The following R code fits a simple model to predict the median house value, shows the model summary, produces some plots, and illustrates the use of the `predict()` function to predict values:

```r
# Fit a model with intercept only
fitted.model <- lm(medv ~ 1, data=Boston)
summary(fitted.model)

# Fit a model with predictor lstat
fitted.model <- lm(medv ~ lstat, data=Boston)
summary(fitted.model)

# Plot the data and the regression line
plot(medv ~ lstat, data=Boston)
abline(fitted.model, lwd=3, col='red')

# Plot the residuals against predicted values
plot(predict(fitted.model), residuals(fitted.model))

# Predict three new observations of lstat
predict(fitted.model, data.frame(lstat=c(5, 10, 15)),
   interval='confidence')
```

The following R code fragment adds a second input variable (age) to the model. It also demonstrates some special notation in the R formula interface, such as the `.` to include all main effects, the `:` to specify interaction effects, and the ⋆ to include both main and interaction effects.

```r
# Add another predictor
fitted.model <- lm(medv ~ lstat + age, data=Boston)

# Add all main effects
fitted.model <- lm(medv ~ ., data=Boston)

# Add interaction terms
fitted.model <- lm(medv ~ lstat + age + lstat:age, data=Boston)

# Shorter and equivalent
fitted.model <- lm(medv ~ lstat*age, data=Boston)
summary(fitted.model)
```

The next example adds polynomial terms to the regression. It uses the `I(.)` function in R that can also be used with other transformations of the inputs, such as `I(sqrt(.))`

or `I(log(.))`. To make it easier to include all polynomials up to a particular degree, one can us the `poly(.)` function:

```
# Add a polynomial term; use the I(.) function
# for any data transformations, such as log(),
# or exp() or sqrt() as well as polynomials
fitted.model <- lm(medv ~ lstat + I(lstat^2), data=Boston)
summary(fitted.model)

# Add all polynomial terms up to degree 5
fitted.model <- lm(medv ~ poly(lstat, 5), data=Boston)
# Note the coefficients for the polynomials in the summary
summary(fitted.model)
```

The use of categorical input variables is illustrated in the following example that uses the `Carseats` data, also from the `ISLR2` package. The data frame represents sales data of car seats for different stores and a range of store characteristics in other variables, many of which are categorical.

The following R code demonstrates functions for dealing with categorical variables, called ”factors” in R. The example then fits a regression model to predict Sales from the the main effects of all input variables in the data frame.

```
?Carseats

# Identify factor/categorical variables and their levels:
is.factor(Carseats$ShelveLoc)
levels(Carseats$ShelveLoc)
levels(Carseats$Urban)
levels(Carseats$US)

# Contrasts show the dummy variables created (columns) and
# the values they take for different  factor levels (row)
contrasts(Carseats$ShelveLoc)
contrasts(Carseats$US)

# Fit the model
summary(lm(Sales ~ . , data=Carseats))
```

Hands-On Exercise

Use the `Auto` data set from the ISLR2 library with `mpg` as the target.

1. Perform a linear regression with `horsepower` as predictor
2. Is there a relationship between the predictor and target? What form and how strong?
3. What is the predicted `mpg` value for a `horsepower` of 98?
4. Plot the response and predictor. Use the `abline()` function to add the regression line
5. Produce a scatterplot of all variables
6. Perform a linear regression of all main effects (except for the variable `name`), then remove non-significant predictors
7. Use the `*` and `:` symbols to add interaction effects. Retain only significant ones
8. Add transformations of the predictors (using the `I(.)` function) such as $\log(X)$, $\sqrt{X}$, X^2.

Source: ISLR2 Section 3.7

Hands-On Exercise

Use the `Carseats` data set from the ISLR2 library with `Sales` as the target.

1. Perform a linear regression with `Price`, `Urban` and `US` as predictors
2. Interpret the coefficients. Tip: Some variables are categorical
3. Remove non-significant predictors
4. How well do the two models fit the data in terms of their R^2?

Source: ISLR2 Section 3.7

4 Cross-Validation in R

The previous chapter already introduced the concepts of cross-validation. Recall that the appropriate way to judge the predictive performance of a model is not to evaluate it on its training data, but to evaluate it on unseen test data. This section illustrates three different cross-validation approaches in R, beginning with the validation set or holdout sample approach, which splits the data set randomly into training and test data.

The following example uses the `Auto` data set from the `ISLR2` package, which contains information on different vehicle models, their fuel economy (”miles per gallon”, mpg) and vehicle characteristics[2].

Randomly splitting the data set is accomplished using the `sample()` function in R which returns a boolean vector with values `TRUE` or `FALSE` that can be used to select the appropriate data from the data frame.

[2] The R code for this example is based on material in Section 5.3 of ISLR2

To make the example repeatable, the initial seed of the pseudo-random number generator (RNG) is set to a fixed value with `set.seed()`. Computers cannot generate truly random number; they are deterministic machines. The pseudo-random numbers they generate are computed by a deterministic algorithm, the ”random number generator” (RNG), based on a given start value. With the same start value, the algorithm produces the same sequence of numbers. A good RNG produces numbers that are effectively indistinguishable from true trandom numbers.

```r
# Set the seed for the pseudo-random number generator (RNG)
set.seed(1)

# Randomly use half the Auto data as training sample
train.idx <- sample(nrow(Auto), nrow(Auto)/2)
train.data <- Auto[train.idx,]
test.data <- Auto[-train.idx,]

# Fit model to (train model on) a subset
fitted.model <- lm(mpg ~ horsepower, data=train.data)

# Calculate the test data MSE by predicting from the test data set
mean((test.data$mpg-predict(fitted.model,test.data))^2)

# Calculate the training MSE by predicting from the training data set
mean((train.data$mpg-predict(fitted.model,train.data))^2)

# The MSE can also be calculated from the squared residuals
mean(summary(fitted.model)$residuals^2)
```

The next R code block illustrates Leave-One-Out Cross-Validation (LOOCV), where one observation is used as test observation, and the remainder form the training data. This is repeated so that each observation becomes the test observation in turn. The errors are then averaged.

While this can be done manually in R using an iteration over the data frame, the library `boot` provides easy-to-use cross-validation functions for models fitted using the `glm` function (generalized linear models). Note that LOOCV is just k-fold cross-validation where k, the number of test data folds, is equal to the number of observations.

```r
library(boot)

# Fit a model with glm and show its summary
fitted.model <- glm(mpg ~ horsepower, data=Auto)
summary(fitted.model)

# LOOCV is k-fold CV where k equals the number of observations
cv.err <- cv.glm(Auto, fitted.model, K=nrow(Auto))
cv.err$delta[1]
```

The same functions can be used for k-fold CV with a typical value for k:

```
cv.err <- cv.glm(Auto, glm.fit, K=10)
cv.err$delta[1]
```

Cross-validation is useful for comparing different models, as shown in the following example that fits linear regression models with different degrees of polynomials to a data frame and computes their cross-validation errors using a `for` loop in R:

```
set.seed(1)
cv.err <- rep(0, 5)
for(i in 1:10) {
  fitted.model <- glm(mpg ~ poly(horsepower,i), data=Auto)
  cv.err[i] <- cv.glm(Auto, fitted.model, K=10)$delta[1]
}
print(cv.err)
```

Hands-On Exercise

1. Fit a regression model to the `Boston` data set with `medv` as target, and `age`, `lstat`, and `ptratio` as predictors
2. Using the holdout approach, compute the test error of this model. Perform the following steps
   (a) Split the data set using 75% for training and 25% for testing
   (b) Fit the model to training data
   (c) Predict the target for the testing data
   (d) Compute the test error
3. Repeat the previous step 2 times, using different splits. How do the results change?
4. Calculate the mean and the variance of the test errors of the four splits.
5. Calculate the test error estimate using LOOCV. Compare your result to the mean that you computed in step 4.
6. Calculate the test error estimate using 10-fold cross-validation. Compare the estimate to the mean that you computed in step 4

5 Shrinkage Methods

Shrinkage methods are so-called because their aim is shrink the magnitude of the regression parameter values $\hat{\beta}_i$. This is primarily to avoid overfitting a model to one specific data set. In other words, they are a kind of *regularization*. Shrinkage methods for linear regression model works by *penalizing* the model for large regression parameter values, that is, by performing *penalized regression*. There are three frequently used types of penalized regression or linear regression regularization: L1 regularization penalizes large absolute parameter values (that is, the L1 norm of the vector of parameters $\hat{\beta}$), L2 regularization penalizes large squared parameter values (that is, the L2 norm of the vector of parameters $\hat{\beta}$, and Elastic Net regularization is a combination of both. L1 regularization is also known as the *LASSO* (”least absolute shrinkage and selection operator”) while L2 regularization is known as *ridge regression* or Tikhonov regularization.

5.1 Ridge Regression

The loss function or minimization objective of ridge regression is the usual RSS that now also includes a term that penalizes large values of β.

$$\text{Minimize} \quad RSS + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda ||\beta||_2^2$$

where $|| \cdot ||_2$ indicates the *L2 vector norm*, that is, the sum of squared entries of the vector β.

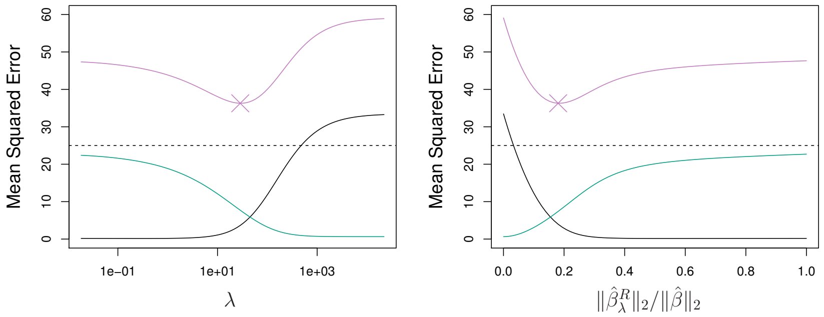
Because the scale, that is, the standard deviation or variance, of different predictors affects the size of their associated β parameter values, *all predictors should be rescaled or standardized to have the same standard deviation prior to performing a ridge regression*.

The degree of penalization or the amount of shrinkage is controlled by the parameter λ. Larger values for λ increase the penalty, thus generally leading to larger model bias but smaller variance. This effect is shown in Figure 7 where the left panel shows the bias (black line), the variance (turquoise line) and the total MSE (pink line) as a function of λ and the right panel shows the bias, the variance, and the total MSE as a function of the proportion to which the L2 norm of the β is restricted or penalized compared to the unrestricted estimates, i.e. $||\hat{\beta}_\lambda^R||_2 / ||\hat{\beta}||_2$ where $|| \cdot ||_2$ indicates the *L2 vector norm*, i.e. the sum of the squared elements of the vector.

Figure 8 shows this effect for a polynomial of degree 14 that is fitted to data set of 21 observations. The different panels show different degrees of penalization or shrinkage as indciated by different values of λ. The bottom right panel in Figure 8 shows the MSE on the training data (blue) and test data (red) for multiple values of λ. It is clear that the unpenalized model in panel (a) of Figure 8 overfits, while models that are penalized too heavily underfit and have large bias, such as the model in panel (c) of Figure 8.

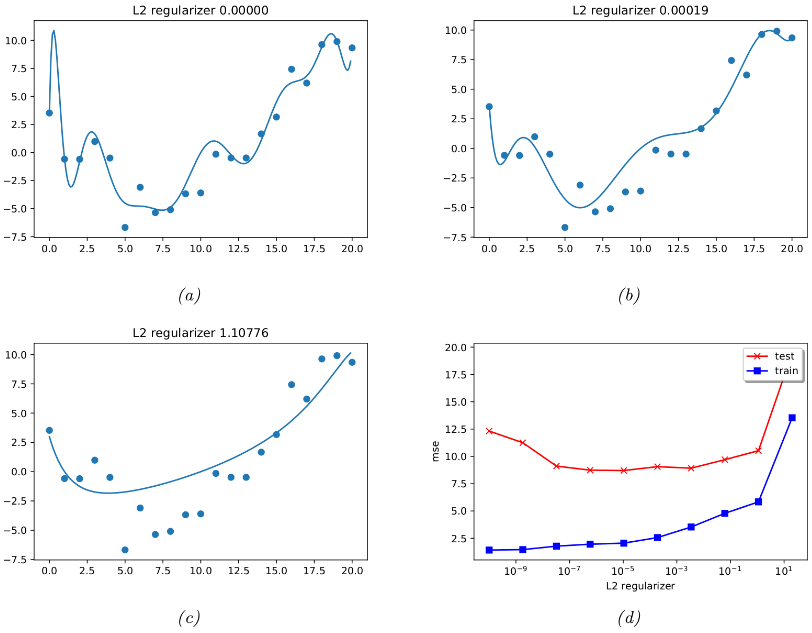
5.2 LASSO

In contrast to the ridge regression, where the model parameter values are shrunk towards zero, but are never forced to equal zero, the LASSO does just that. In this form of penalized regression, as the degree of penalty is increased, model parameter values are set to 0, effectively making the LASSO a model or *predictor selection method* as well, that is, it can be used to select only important predictors. This is reflected in



Source: ISLR2 Figure 6.5

Figure 7: Bias (black), variance (turquoise), and MSE (pink) in ridge regression



(a)

(b)

(c)

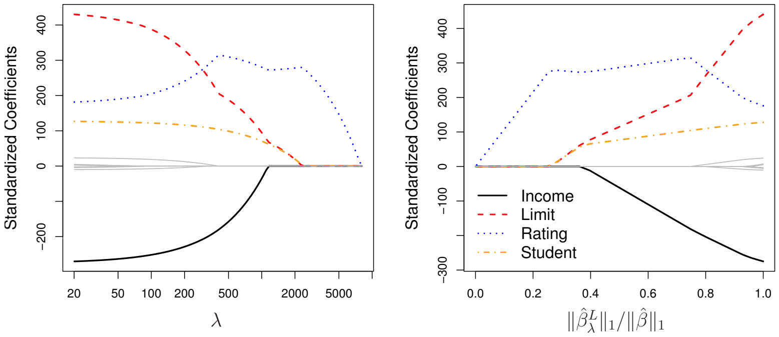
(d)

Source: Murphy Figure 4.5

Figure 8: Fitting a degree 14 polynomial with ridge regression

its name: ”Least Absolute Shrinkage and Selection Operator”. The advantage over ridge regression is that this results in more parsminious, that is, smaller, models that are easier to interpret.

The loss function or minimization objective of the LASSO is the RSS but penalized for the L1 norm of the vector of parameters β. As in ridge regression, the amount of shrinkage is controlled by a parameter λ.



Source: ISLR2 Figure 6.6

Figure 9: Preditor selection in the LASSO

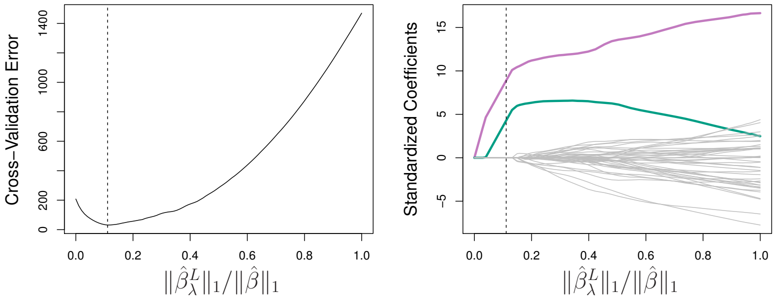
$$\text{Minimize} \quad RSS + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda ||\beta||_1$$

where $||\dot{}||_1$ indicates the *L1 vector norm*, that is the sum of the absolute values of the elements of the vector β.

The model selection property of the LASSO are shown in Figure reffig:lasso1. The left panel shows the value of different model parameters as a function of the penalization parameter λ. As λ increases, fewer and fewer parameters are allowed to retain absolute values larger than 0. The same is shown in the right panel of Figure 9, but now as a function of the relative size of the restricted L1 norm of β compared to the L1 norm of the unconstrained β.

The LASSO and ridge regression shrinkage or penalty parameter λ is typically chosen through cross-validation to minimize the cross-validation or test error. For cross-validation, a ”grid” or range of possible values of λ is defined. A model is fitted for each value of λ and its cross-validation or test error is calculated. The final model is then fitted using the optimal value of λ, that is, the value that results in the lowest cross-validation or test error.

This is illustrated in Figure 10 for a LASSO, where the left panel shows the cross-validation error as a function of the relative shrinkage. Towards the left of the graph in that panel, where shrinkage is maximal, the model underfits, resulting in a relative large error due to a large bias, whereas to the right of the graph, where shrinkage is minimal, the model overfits, leading to a large cross-validation error due to large variance. The right panel of Figure 10 shows the size of model parameter values; at the optimal λ only two model parameters are different from zero, that is, only two predictors are selected for the model.



Source: ISLR2 Figure 6.13

Figure 10: Cross-validation error in the LASSO

5.3 Elastic Net

The Elactic Net is a combination of rigde regression and LASSO, controlled by the parameter α. The Elastic Net penalty is defined as

$$\lambda\left(\alpha||\beta||_1 + (1-\alpha)||\beta||_2^2\right)$$

When $\alpha = 0$ the Elastic Net reduces to a ridge regression, and for $\alpha = 1$ the Elastic Net reduces to the LASSO. The optimal combination of α and λ is found through cross-validation as described above.

6 Shrinkage Methods in R

The `glmnet` library for R implements the Elastic Net which can be used for both ridge regression and LASSO as well, simply by choosing the appropriate value for the parameter α.

The following R code examples use the `Hitters` data set containing information on baseball players. The data set is part of the `ISLR2` library. The code examples model a player's Salary as the output or prediction target and use a number of other variables as inputs[3].

```
library(ISLR2)
library(glmnet)

# Remove missing values
Hitters <- na.omit(ISLR2::Hitters)
```

[3] The R code for this example is based on material in Section 6.5.2 of ISLR2

The `glmnet(.)` function requires separate x (predicots) and y (target) values, instead of providing a formula interface like the `lm(.)` and `glm(.)` functions. To create dummy variables for categorical variables, use the `model.matrix` function:

```
# Create dummy variables for categorical variables
# and remove the intercept (first column) from the model
x <- model.matrix(Salary ~ ., Hitters)[, -1]
y <- Hitters$Salary
```

To illustrate the concepts of ridge regression, the following R code sets up a grid of 100 different λ values, fits ridge regression models to each of them, and shows information about two models for different λ values:

```
grid <- 10^seq(from=10, to=-2, length=100)
print(grid)
ridge.model <- glmnet(x, y, alpha=0, lambda=grid)

# Select the 50th lambda value
ridge.model$lambda[50]
coef(ridge.model)[, 50]
L2.norm = sqrt(sum(coef(ridge.model)[-1, 50]^2))

# Select the 60th lambda value
ridge.model$lambda[60]
coef(ridge.model)[, 60]
L2.norm = sqrt(sum(coef(ridge.model)[-1, 60]^2))
```

The optimal value for λ is chosen through cross-validation. For this, a holdout test data set is created to evaluate the final model. The training data set portion is used with cross-validation, so that the final model evaluation is done on an independent data set from model selection.

```
# Randomly split the Hitters data
train.idx <- sample(nrow(Hitters), nrow(Hitters)/2)
x.train <- x[ train.idx,]
x.test  <- x[-train.idx,]
y.train <- y[ train.idx]
y.test  <- y[-train.idx]
```

The `glmnet` library provides the function `cv.glmnet` as an easy-to-use way to combine Elastic Net model fitting with k-fold cross-validation. The following example uses 5-fold CV on the training portion of the data set:

```
# 5-fold cross-valiation, use MSE as metric
cv.out <- cv.glmnet(x.train, y.train, alpha=0,
   nfolds=5, type.measure='mse')
```

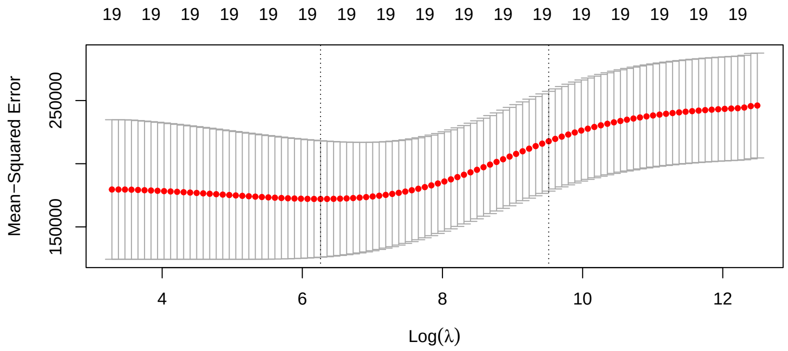


Figure 11: Cross-validation MSE in ridge regression

The next R code block shows the optimal λ and plots the MSE for different values of λ. The generated plot is shown in Figure 11. It includes the standard errors of the MSE (as determined by cross-validation) and the number of non-zero coefficients (indicated above the graph). The left vertical line indicates the optimal λ, while the right vertical line indicates that largest λ for which the error is within one standard error of the minimum error, that is, of the error of the optimal λ.

```
print(cv.out)
plot(cv.out)
lambda.opt <- cv.out$lambda.min
```

To evaluate the final model, the holdout test data is fitted to a ridge regression model with the optimal λ. Then, the model parameter values of the ridge regression with the optimal λ are compared to the parameter values of an unrestricted linear regression model. Note the use of the `type='coefficients'` parameter to `predict(.)` that asks for the model parameter values, rather than the predicted target values in the following R code block.

```
# Fit test data:
ridge.test <- glmnet(x.test, y.test, alpha=0)
# Show the coefficients at optimal lambda
predict(ridge.test, type='coefficients', s=lambda.opt)
# Compare to unpenalized least-squares fit
coef(lm.fit(x.test, y.test))
```

Finally, the target values for the holdout test data set are predicted. Note the use of the `type='response'` parameter to `predict(.)` that asks for the predicted target values, rather than the model parameter values, in the following R code block:

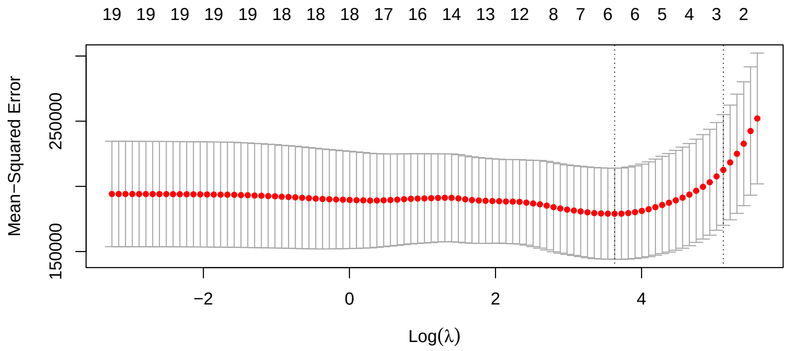


Figure 12: Cross-validation MSE in the LASSO

```
predictions <- predict(ridge.test, type='response',
                       s=lambda.opt, newx=x.test)
# Calculate test MSE to compare to the CV optimal MSE above:
mean((predictions - y.test)^2)
```

Because it uses the same `glmnet(.)` and `cv.glmnet(.)` functions, the LASSO in R is very similar to ridge regression. The following example shows cross-validation on the training data set to determine the optimal value for λ.

```
# Set alpha to 1 for lasso
cv.out <- cv.glmnet(x.train, y.train, alpha=1,
   nfolds=5, type.measure='mse')
print(cv.out)
plot(cv.out)
lambda.opt <- cv.out$lambda.min
```

The MSE values for different values of λ are shown in Figure 12. Note that now the number of non-zero model parameters, indicated above the graph, decreases. At the optimal λ value, there are only six non-zero parameters, that is, only six predictors are selected to remain in the model.

Hands-On Exercise

Predict the number of applications received using the other variables in the `College` dataset

1. Split the data set into a training and a test set
2. Fit an unpenalized linear model on the training set. Report the test error.
3. Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error.
4. Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error.
5. Compare and conrast the results

Source: ISLR2, Section 6.6

Hands-On Exercise

Predict the per-capita crime rate in the `Boston` data set using the other variables.

1. Split the data set into a training and a test set
2. Fit an unpenalized linear model on the training set. Report the test error.
3. Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error.
4. Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error.
5. Compare and conrast the results

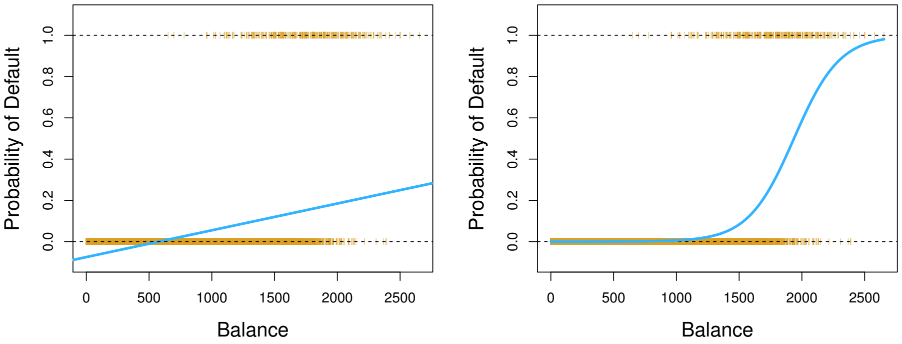
Source: ISLR2, Section 6.6

7 Classification

In classification, the output or target value is categorical. In particular, in binary classification, the target may take on one of two values. Classification predicts class membership for new observations by estimating the probability of membership in each class for that observation.

7.1 Logistic Regression

Linear models, as used in regression, are not suitable for classification without modification, because probabilities are bounded between 0 and 1, inclusive, while the regression output can take on any real value. The solution to this problem is to use a *link function* that transforms the output of a linear regression model and bounds it between 0 and 1. This is shown in Figure 13 where the probability of credit card default are to be predicted from the credit card balance. The yellow points are the training observations, classified as either defaulters (0) or non-defaulters (1). The left panel shows that a linear regression (blue line) yields negative probabilities for small balances, and



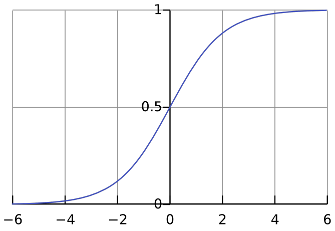
Source: ISLR2 Figure 4.2

Figure 13: Transforming linear regression output for binary classification

will yield probabilities larger than one for very large credit card balances. The right panel shows the transformed linear regression output, now bounded between 0 and 1, and interpretable as probabilities from which class membership can be predicted.

There are many transformation or link functions that may be used, but a popular one is the *logistic function*, a type of *sigmoid function* (”s-shaped”) (often the two terms are equated), defined as follows:

$$\begin{aligned}\sigma(x) &= \frac{1}{1+e^{-x}}\\ &= \frac{e^x}{1+e^x}\\ &= 1-\sigma(-x)\end{aligned}$$



https://commons.wikimedia.org/wiki/File:Logistic-curve.svg

For the binary case, this leads to *binary logistic regression*, expressed by the following equalitites:

$$\begin{aligned}p(X) &= \sigma(\beta_0+\beta_1 X)\\ &= \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} && (1)\\ \Rightarrow \quad \frac{p(X)}{1-p(X)} &= e^{\beta_0+\beta_1 X} \qquad \text{"Odds"} && (2)\\ \Rightarrow \quad \log\left(\frac{p(X)}{1-p(X)}\right) &= \beta_0+\beta_1 X \qquad \text{"Log-Odds", "Logits"} && (3)\end{aligned}$$

Equation 1 defines the probability of an observation being ”true” as just the logistic transformation of the linear combination of predictors. Dividing and a little algebraic rearrangement yields equation 2 which represents the *odds* as the exponential of the linear combination of predictors. Taking the natural logarithm yields equation 3 which shows that the linear combination of predictors are equal to the *log-odds*, also called *logits*.

Once estimates of the model parameters β_0 and β_1 have been calculated, predictions of the class probability can be made using the logistic link function:

$$\hat{p}(X) = \sigma(\hat{\beta}_0 + \hat{\beta}_1 X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 X}}$$

From there, a *threshold value* can be used to predict class membership, for example based on $\hat{p}(X) > 0.5$, but other decision rules and threshold values may be used, depending on the desired proportion or cost of false positives and true positives.

The binary logistic regression can be extended to *mutinomial logistic regression*, where there are more than two classes for the output or target. The form of the equations is similar for K classes, starting from the log-odds. In this derivation, the last class K plays the role of reference class. As classes are not ordered, this choice is arbitrary.

$$\log\left(\frac{\Pr(Y=k|X=x)}{\Pr(Y=K|X=x)}\right) = \beta_{k0} + \beta_{k1}x_1 + \cdots + \beta_{kp}x_p \text{ }, k < K$$

Exponentiating and multiplying:

$$\Pr(Y=k|X=x) = \Pr(Y=K|X=x)e^{\beta_{k0} + \beta_{k1}x_1 + \cdots + \beta_{kp}x_p} \text{ }, k < K \tag{4}$$

Because probabilities must sum to 1:

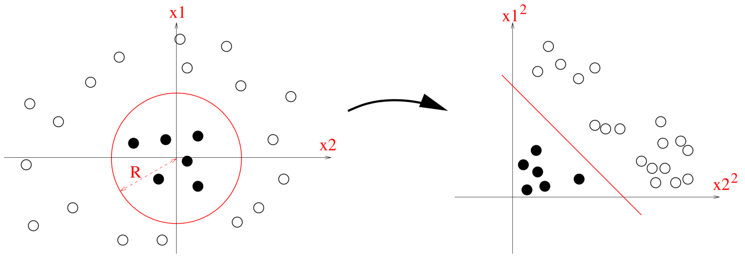
$$\Pr(Y=K|X=x) = 1 - \sum_{l=1}^{K-1} \Pr(Y=l|X=x)$$

Substiting Eq. 4 into the right-hand side:

$$= 1 - \sum_{l=1}^{K-1} \Pr(Y=K|X=x)e^{\beta_{l_0} + \beta_{l_1}x_1 + \cdots + \beta_{l_p}x_p}$$

Moving $\Pr(Y=K|X=x)$ out of the sum, dividing by it, and rearranging:

$$\Rightarrow \Pr(Y=K|X=x) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l_0} + \beta_{l_1}x_1 + \cdots + \beta_{l_p}x_p}} \tag{5}$$



Source: Murphy Figure 10.3

Figure 14: Transforming non-linear decision boundaries using polynomials

Substituting Eq. 5 into Eq. 4:

$$\Rightarrow \Pr(Y = k|X = x) = \frac{e^{\beta_{k_0}+\beta_{k_1}x_1+\cdots+\beta_{k_p}x_p}}{1+\sum_{l=1}^{K-1} e^{\beta_{l_0}+\beta_{l_1}x_1+\cdots+\beta_{l_p}x_p}}\ ,\ k < K \tag{6}$$

Equations 5 and 6 give the class probabilities for the reference class K and for any other class $k < K$. They are formally similar to the equations for the binary logistic regression above; for example, compare equation 6 to equation 1.

Just as polynomials of the input variables may be useful as predictors in linear regression, they also have applications in classification. In particular, they transform non-linear decision boundaries into linear ones that can be modeled using linear logistic regression. For example, the nonlinear boundary in the left panel of Figure 14 can be transformed into a linear boundary by squaring both predictors, as shown in the right panel of Figure 14.

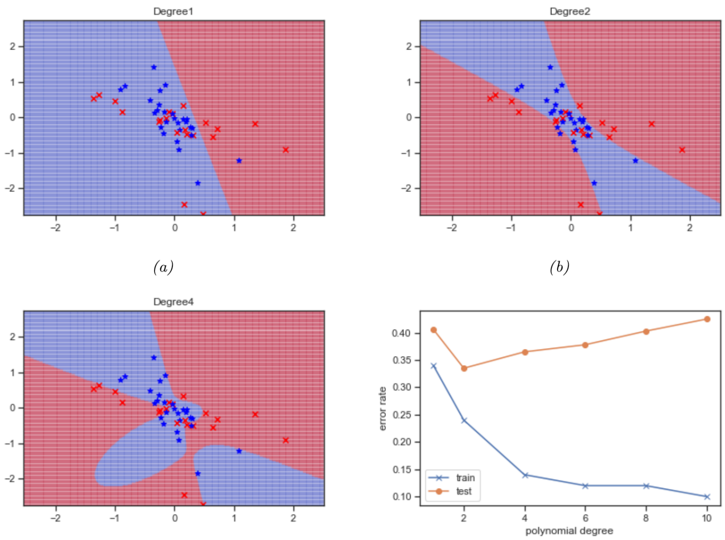
Another example is shown in Figure 15 where the linear decision boundary dividing the top left panel into a blue and red region clearly does not fit the observations, shown as blue and red points. Fitting logistic regressions with different degrees of polynomials shows that the decision boundary can be transformed to better fit the observed data. The bottom right panel in Figure 15 shows the the train and test error rate for different degrees of polynomials.

7.2 Logistic Regression in R

The following illustration of logistic regression in R uses the `Smarket` data set of the `ISLR2` library. The data set contains stock market information and is used in this example to predict the direction of the movement of the market, either ”up” or ”down”, based on previous day’s (”lagged”) data and other variables[4].

Logistic regression can be performed using the same `glm(.)` function as for linear regression; it is one specific form of the generalized linear model. The `family` argument is used to indicate the type of regression and the link function:

[4] The R code for this example is based on material in Section 4.7.2 of ISLR2



(a) *(b)*

(c) *(d)*

Source: Murphy Figure 10.4

Figure 15: Transforming linear decision boundaries using polynomials

```
library(ISLR2)
?Smarket

# Contrasts show how factor levels are encoded using dummy variables:
contrasts(Smarket$Direction)

# Fit a logistic regression model
logreg.fitted <-
   glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume, data=Smarket,
           family=binomial(link='logit'))
summary(logreg.fitted)
```

The `predict(.)` function for the fitted model can be used to predict either the logits or the class probabilities:

```
# Predict logits for training data
logreg.logits <- predict(logreg.fitted, newdata = Smarket)

# Predict probabilities for training test
logreg.probabilities <- predict(logreg.fitted, newdata = Smarket,
                                   type='response')
```

A decision rule is necesary to assign observations to classes using the calculated probabilities. The following example classifies them into the ”Up” class, if its class-membership probability is greater than .5:

```
# Predict 'up' or 'down' based on probabilities and a threshold
pred.direction <- rep('Down', nrow(Smarket))
pred.direction[logreg.probabilities > .5] <- 'Up'
```

The confusion matrix can be produced by using the `table(.)` function on the predicted class and the observed class. Accuracy is simply the average number of observations for which predicted class and observed class are identical.

```
# Compute confusion matrix
logreg.cm <- table(pred.direction, Smarket$Direction)
print(logreg.cm)

# Compute accuracy
mean(pred.direction == Smarket$Direction)
```

The next R code blocks illustrate the use of a holdout sample or validation set approach to evaluate the performance of the logistic regression classifier.

Because the data set is a time series, a random split is not appropriate because it would mean that the training set would contain information later in time that informs the model parameter estimates which are used to predict earlier observations in the test set. Instead, time series data must be split non-randomly at some point in time:

```
train.data <- Smarket[Smarket$Year < 2005,]
test.data <- Smarket[!(Smarket$Year < 2005),]
```

Next, the model is fitted to the training data set, class-membership probabilities for observations in the test data set are then predicted, and the observations are classified using a decision rule:

```
logreg.fitted <-
   glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume, data=train.data,
            family=binomial(link='logit'))

# Predict probabilities for test data and classify:
logreg.probabilities <- predict(logreg.fitted, newdata = test.data,
                                 type='response')
pred.direction <- rep('Down', nrow(test.data))
pred.direction[logreg.probabilities > .5] <- 'Up'
```

While the confusion matrix is useful to understand how a classifier behaves, it is only a first step in evaluating the classifier performance. The `ROCR` library provides the `performance(.)` function to evaluate the predictive performance of a classifier. It

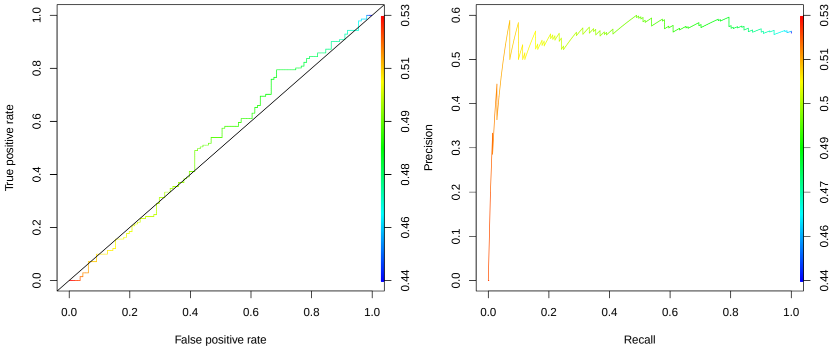


Figure 16: ROC and precision/recall curves for a logistic regression classifier

provides metrics such as accuracy, precision, recall, and can generate ROC curves. The last two plots generated by the R code block below are shown in Figure 16. The left panel shows the ROC curve, the right panel shows the precision/recall plot. It is clear from the results that predicting the stock market from the inputs in the given data set does not work well.

```
library(ROCR)

# A prediction object contains probabilities and true labels
pred.obj <- prediction(logreg.probabilities, test.data$Direction)

# Get some classifier performance metrics, ROCR varies the threshold.
plot(performance(pred.obj, 'acc'))
plot(performance(pred.obj, 'prec'))
plot(performance(pred.obj, 'rec'))
plot(performance(pred.obj, 'f'))

# ROC curve: True positive rate versus false positive rate
plot(performance(pred.obj, 'tpr', 'fpr'), colorize=T)
abline(0, 1)
# Precision/Recall plot
plot(performance(pred.obj, 'prec', 'rec'), colorize=T)
# Calculate the AUC
performance(pred.obj, 'auc')@y.values[[1]]
```

7.3 Naive Bayes Classifier

The naive Bayes classifier is based on *Bayes' theorem* of conditional probabilities. The probability that an observation described by a vector of inputs X is a member of class c can be described as the probability of observing inputs X given that the class is c, multiplied by the unconditional probability of an observation being in class c, divided by the unconditional observation of observing inputs X. Formally:

$$\Pr(Y = c\,|X) = \frac{p(X|Y = c)\,p(Y = c)}{p(X)}$$

The overall probability of observing vector X is the sum of the probabilities of observing X in each class l, multiplied by the probability of an observation being member of class l:

$$= \frac{p(X|Y = c)\,p(Y = c)}{\sum_{l=1}^{K} p(X|Y = l)\,p(Y = l)} \tag{7}$$

The *naive Bayes assumption* is that within each class c, the D different input features that make up X are independently distributed of each other. With this assumption, one can write:

$$\begin{aligned} p(X|Y = c) &= p(x_1|Y = c) \times p(x_2|Y = c) \times \cdots \times p(x_D|Y = c) \\ &= \prod_{d=1}^{D} p(x_d|Y = c) \end{aligned} \tag{8}$$

Substituting Equation 8 into Equation 7 yields the *posterior probability* of class membership:

$$p(Y = c|X) = \frac{\left(\prod_{d=1}^{D} p(x_d|Y = c)\right)\, p(Y = c)}{\left(\sum_{l=1}^{K} \prod_{d=1}^{D} p(x_d|Y = l)\right)\, p(Y = l)}$$

The probabilities in the product of Equation 8 can be trivially estimated from the data, simply as the proportion of each x_d for each class c. However, the assumption of independence is violated when features are correlated. In other words, the naive Bayes classifier can be expected to perform best for independent features.

7.4 Naive Bayes Classifier in R

The `e1071` library for R provides the `naiveBayes(.)` function that implements the naive Bayes classifier. The following illustration uses the same `Smarket` data from the `ISLR2` library that was used in the above example on logistic regression[5]:

[5] The R code for this example is based on material in Section 4.7.5 of ISLR2

```
library(e1071)
library(ISLR2)
train.data <- Smarket[Smarket$Year < 2005,]
test.data <- Smarket[!(Smarket$Year < 2005),]

# Fit using same syntax as glm
nb.fitted <- naiveBayes(Direction ~ Lag1 + Lag2, data=train.data)
# Output contains prior and conditional probabilities (and their SD)
print(nb.fitted)
```

A `predict(.)` method is available to predict class membership, given a fitted naive Bayes classifier and a data set of observations. A confusion matrix can be constructed by comparing predicted classes to observed classes. The following R code predicts class memberships and computes the confusion matrix for the test data.

```
nb.predictions <- predict(nb.fitted, test.data)
nb.cm <- table(nb.predictions, test.data$Direction)
print(nb.cm)
```

Evaluating the classifier is similar to evaluating the logistic regression classifier and again uses the `ROCR` library. The ROC curve created by the following R code block is shown in Figure 17. Comparing this ROC curve to the one in Figure 16 shows that the naive Bayes classifier performs slightly better than the logistic regression classifier.

```
library(ROCR)

# Predict probabilities (for use with ROCR)
nb.probabilities <- predict(nb.fitted, test.data, type='raw')
# Create an ROCR prediction object
nb.pred.obj <- prediction(nb.probabilities[,'Up'], test.data$Direction)

# Generate an ROC plot
plot(performance(nb.pred.obj, 'tpr', 'fpr'), colorize=T)
abline(0, 1)
# Compute the AUC value
performance(nb.pred.obj, 'auc')@y.values[[1]]
```

7.5 KNN Classification

K-Nearest Neighbour classification is a non-parametric classification technique. It identifies the k nearest neighbours of a new observation, typically using the Euclidean distance or the L2-norm of the vector differences. It then estimates the class membership probabilities as the class membership proportions among the k nearest neighbours of the new observations. Details can be found in the previous chapter.

One important requirement in KNN classification is *scaling of inputs* to have similar standard deviations or variance to avoid the distance metric being dominated by the

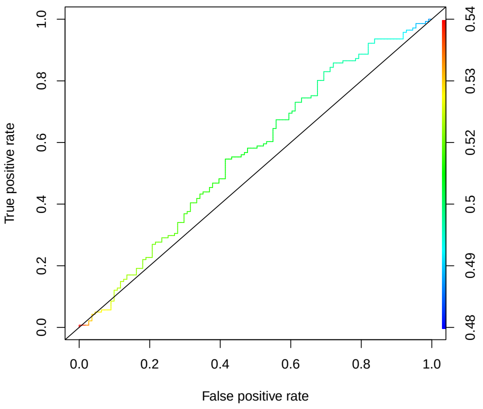


Figure 17: ROC curve of a naive Bayes classifier

input on the largest scale. For example, if one input ranges between 1 and 10 million, while another input ranges between 1 and 10, the first input is clearly most important in determining the distance between two observations.

7.6 KNN Classification in R

The `class` library for R provides the `knn(.)` function, as illustrated in the following example R code block that uses the same `Smarket` stock market data set as the above examples for logistic regression and naive Bayes[6].

```
library(class)
library(ISLR2)
train.data <- Smarket[Smarket$Year < 2005,]
test.data <- Smarket[!(Smarket$Year < 2005),]

# Split the data into test and train sets
train.x <- cbind(train.data$Lag1, train.data$Lag2)
test.x <- cbind(test.data$Lag1, test.data$Lag2)
train.y <- train.data$Direction
test.y <- test.data$Direction
```

Next, the `knn(.)` function is used to make predictions for the test data set, given the training data inputs and training data outputs. The `knn(.)` function uses the Euclidean distance metric. The following R code example considers $k = 3$ nearest neighbours and returns the class membership probabilities in addition to the class memberships of the test data set observations:

[6] The R code for this example is based on material in Section 4.7.6 of ISLR2

```r
knn.pred <- knn(train.x, test.x, train.y, k=3, prob=T)
```

A confusion matrix can be computed using the `table(.)` function and the accuracy is calculated as the mean number (proportion) of observations for which the knn prediction is the same as the observed class membership.

```r
# Confusion matrix
table(knn.pred, test.y)
# Accuracy
mean(knn.pred == test.y)
```

The class membership probabilities returned in the `knn(.)` function result are those of the majority class, in this example the class ”Down”. Because this is a binary classification, the class membership probabilities for the ”Up” class can be trivially calculated, as shown in the following R code block:

```r
knn.probs <- attributes(knn.pred)$prob

# Compute class probabilities of the minority class:
knn.class.probs <- knn.probs
knn.class.probs[knn.pred=='Down'] <- 1-knn.probs[knn.pred=='Down']
```

With the class membership probabilities for both classes, the `ROCR` library functions can be used to evaluate the classifier by plotting the ROC curve and computing the AUC. The ROC curve produced by the R code block below is shown in Figure 18.

```r
knn.pred.obj <- prediction(knn.class.probs, test.data$Direction)
plot(performance(knn.pred.obj, 'tpr', 'fpr'), colorize=T)
abline(0, 1)
performance(knn.pred.obj, 'auc')@y.values[[1]]
```

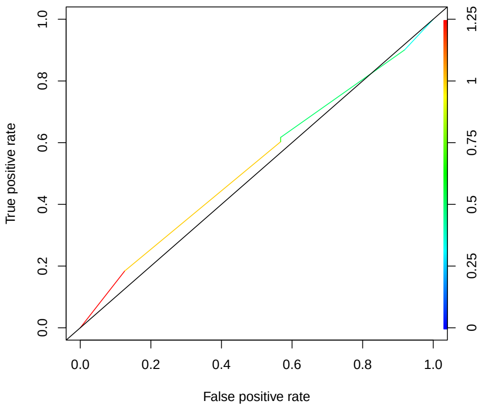


Figure 18: ROC curve of a k-NN classifier

Hands-On Exercise

Use the `Weekly` data set in the `ISLR2` package.

1. Use the full data set to perform a logisttic regression with `Direction` as target. Which predictors are statistically significant?
2. Compute the confusion matrix and accuracy.
3. Use the 1990 to 2008 data for a training set and the 2009/2010 for a test set. Fit a logistic regression model with `Lag2` as the only predictor.
4. Repeat (3) using Naive Bayes
5. Repeat (3) using KNN with $K = 1$
6. Which model provides the best results on this data?

Source: ISLR2 Section 4.8

Hands-On Exercise

Use the `Auto` data set in the `ISLR2` package.

1. Create a binary variable, `mpg01` that contains a 1 if `mpg` is above its median, 0 otherwise. *Tip*: Use the `median()` function. Add the new variable to the data frame.
2. Split the data set into training and test set
3. Perform a logistic regression on the training data to predict `mpg01` from the other features. What is the test error of this model?
4. Repeat (3) using Naive Bayes
5. Repeat (3) using KNN with different values of K. What value of K performs best?

Source: ISLR2 Section 4.8

Hands-On Exercise

Using the `Boston` data set in the `ISLR2` library, fit classification models to predict whether a given census tract has a crime rate above or below the median.

1. Create a new binary variable `crime01` that is 1 is `crime` is above its median, and 0 otherwise. Combine this variable with the data frame. *Tip*: Use the `median()` function for this.
2. Split your data set into a training and test data set
3. Fit logistic regression, Naive Bayes, and KNN (with different K)
4. Describe your findings in terms of prediction error, precision, recall, F1 and AUC

Source: ISLR2 Section 4.8

8 Review Questions

Linear Regression

1. Explain the primary objective of linear regression and how it is implemented in a statistical model.
2. Discuss the importance of visual inspection of data before choosing a regression model.
3. Define *fitted values* or *predicted values* in the context of linear regression. How are they computed?
4. Discuss why a model with a term like $\beta_2 X^2$ is still considered a linear regression model.
5. How does the residual sum of squares (RSS) relate to mean squared error (MSE) in linear regression analysis?
6. Explain the importance and use of the *standard errors* of the estimates in linear

regression.

7. Describe what a *t-test* in regression analysis involves, and how it is used to test hypotheses about model parameters.
8. What does the R^2 statistic tell us about a linear regression model? What are its limitations?
9. Explain the term *interaction effects* using an example, and describe how they can be identified in a regression model.
10. What role do dummy variables play when incorporating categorical predictors into a regression model? Give an example.
11. How can the inclusion of more predictors into a linear regression model affect the model's bias and variance?

Random Numbers

12. Define a pseudo-random number generator (RNG). How does it differ from a true random number generator?
13. Discuss the role of the seed in the generation of pseudo-random numbers. What happens if the seed is not set before generating random numbers in a program?
14. Describe a scenario where using the same seed value might be advantageous in computational analyses.

Shrinkage Methods

15. Explain what is meant by ”shrinkage methods” in the context of regression analysis. Why is it necessary to shrink the magnitude of regression coefficients?
16. What is the difference between L1 and L2 regularization? Provide examples where each might be preferable.
17. Explain the rationale behind using ridge regression and LASSO as alternatives to standard linear regression. What problem do they address?
18. Discuss why it is important to standardize predictors before applying ridge regression. What could happen if the predictors are on different scales?
19. Explain the concept of the LASSO as a form of penalized regression. How does it differ from ridge regression in terms of the impact on model parameters?
20. Discuss the method of selecting the penalty parameter λ in shrinkage methods like ridge regression and LASSO.
21. Discuss how the bias-variance trade-off is managed in ridge regression through the adjustment of λ. What are the signs that λ is set too high or too low?
22. Explain how the Elastic Net method balances the properties of L1 and L2 penalties. What role does the parameter α play in this balance?

Logistic regression

23. Explain the concept of the sigmoid or logistic function as a solution for bounding the output of a regression model between 0 and 1.
24. What is a link function in logistic regression? Describe its purpose and how it modifies the output of a linear model.

25. Define the logistic function and explain how it is used in logistic regression to estimate probabilities.
26. In logistic regression, what does the logit (or log-odds) function represent? How does it relate to the probabilities of class memberships?
27. Discuss the significance of the threshold value in logistic regression. How is it used to determine class membership?
28. Explain the concept of multinomial logistic regression and how it extends the binary logistic regression model to multiple classes.
29. Discuss how incorporating polynomial terms of input variables into a logistic regression model can help in transforming non-linear decision boundaries into linear ones.

Naive Bayes classification

30. What is Bayes' theorem and how is it applied in the naive Bayes classifier to compute class probabilities?
31. Explain how the probability $p(X|Y = c)$ is calculated under the naive Bayes assumption.
32. Discuss the implications of the independence assumption among features in the naive Bayes classifier. What are the potential limitations of this assumption in real-world scenarios?
33. Explain the role of prior probabilities $p(Y = c)$ in the naive Bayes classifier. How do these influence the final classification?
34. What is the effect of having highly correlated features on the performance of the naive Bayes classifier?

KNN classification

35. Define K-Nearest Neighbor (KNN) classification. Why is it classified as a non-parametric technique?
36. Explain the concept of "nearest neighbors" in the context of KNN. What metrics can be used to determine proximity in feature space?
37. Discuss the role of the number k in KNN classification. How does the choice of k influence the classifier's performance?
38. Describe how KNN estimates the class membership probabilities for a new observation.
39. Discuss the impact of feature scaling on the performance of KNN classification. Why is it important to scale features?
40. Discuss the trade-offs between choosing a larger versus smaller value of k in KNN classification.