3. Linear Regression

3.1 Overview

Many machine learning algorithms have their origin in the field of statistics, and linear regression is a prime example. In fact, the fundamentals of linear regression can be traced back to mathematicians in the early 1800s. We begin with linear regression because it is relatively simple, yet powerful, and introduces foundational concepts and terminology we will use for other algorithms throughout the book. In linear regression, our data consists of predictor values, x, and target values y. We wish to find the relationship between x and y. This linear relationship can be defined by parameters w and b, with w, the slope of the line, quantifying the amount that y changes with changes in x, and b serving as an intercept.

3.2 Linear Regression in R

Let's take a look at the `women` data set, one of the built-in data sets in R.

Code 3.2.1 — women data. Height in inches and weight in pounds for 15 women.

```
> str(women)
'data.frame': 15 obs. of  2 variables:
 $ height: num  58 59 60 61 62 63 64 65 66 67 ...
 $ weight: num  115 117 120 123 126 129 132 135 139 142 ...
> plot(women$weight~women$height)
```

This data set is from the mid-1970s, when Americans were considerably thinner than we are now. Figure 3.1 reveals a linear trend in the data: taller women weigh more than shorter women.

We will use the built-in lm() function to build a linear regression model. There are many parameters we can send to the lm() function, here we only send two: the formula, and the data. The formula *weight* ~ *height* says to model weight, our target, as a function on one predictor: height.

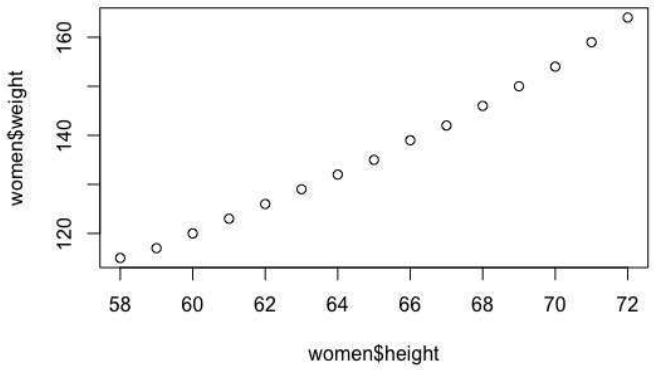

Figure 3.1: Women Data Set

Code 3.2.2 — Build a linear regression model. You can type `?lm` at the console to learn more about the function.

```
> lm1 <- lm(weight~height, data=women)
> lm1

Call:
lm(formula = weight ~ height, data = women)

Coefficients:
(Intercept)       height
     -87.52         3.45
```

When we built the model, we saved it to variable `lm1`. If we type the model name at the console as we see in the code above, we get basic information about the model. Our parameters (coefficients) are the w and b that the algorithm learned from the data. In this model, they are $w = 3.45$, and $b = -87.52$. The intercept parameter, b, is generally not of interest to us as it is just used to fit the data. The parameter w is of more interest. In linear regression, it tells us how much we can expect the y-value to change for every one-unit change in the x-value. So for every inch taller a woman is, we expect her to weight 3.45 pounds more. We can use these values for prediction. Let's say we have a woman who is 65 inches tall. Her weight should be: $65 * 3.45 - 87.52 = 136.7$. Check if that value makes sense given the data in Figure 3.1.

We have a model but we don't know if it is a good model. We can use the summary() function to get a glimpse of the goodness of fit achieved by this model. This is shown in the next code block. We will delve deeper into the summary() output later but now we will just point out two key elements of the output. Notice the three asterisks at the end of the line for height under Coefficients. This indicates that height was a good predictor. Notice also that the intercept received three asterisks, but again, we are not really interested in the intercept. The second thing to notice is the R-squared is about 0.99. The R-squared is a measure of goodness of fit that ranges from 0 to 1, the closer to 1 the better. This provides evidence that we have a good model for this data.

Code 3.2.3 — Model summary. Use the summary() function to learn more about the model.

```
> summary(lm1)

Call:
lm(formula = weight ~ height, data = women)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7333 -1.1333 -0.3833  0.7417  3.1167

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -87.51667    5.93694  -14.74 1.71e-09 ***
height        3.45000    0.09114   37.85 1.09e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.525 on 13 degrees of freedom
Multiple R-squared:  0.991,Adjusted R-squared:  0.9903
F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

3.3 Metrics

There are a lot of metrics associated with linear regression and we will start with metrics we can use in data analysis before we begin applying a machine learning algorithm.

3.3.1 Metrics for Data Analysis

In linear regression we often want to know if variables are correlated. We can use the cor() function or plot pairs() as shown in Section 2.4 earlier. For example, to see if x and y are correlated we can use this code: `cor(x, y)`.

The default method is Pearson's, which measures the linear correlation between two variables. It ranges from -1 to $+1$ where the former is a perfect negative correlation, the latter is a perfect positive correlation, and values close to 0 indicate little correlation. The formula for Pearson's correlation is:

$$\rho_{x,y} = Corr(x,y) = \frac{Cov(x,y)}{\sigma_x \sigma_y} \tag{3.1}$$

We see that correlation is actually covariance, scaled to $[-1,1]$. Covariance measures how changes in one variable are associated with changes in a second variable. The numbers can range wildly which is why the scaled correlation is often preferred. Here is the formula for covariance, where n is the number of data points:

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{n-1} \tag{3.2}$$

3.3.2 Linear Regression Metrics for Model Fit

The summary() output of a linear model gives a lot of useful metrics concerning the model fit. Looking back at the output of summary() for the linear model at the Coefficients section:

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -87.51667    5.93694  -14.74 1.71e-09 ***
height        3.45000    0.09114   37.85 1.09e-14 ***
```

The estimated coefficient for height and the intercept are given, along with standard error, t value and p value. The standard error gives us an estimate of variation in the coefficient estimate and can be used to predict a confidence interval for the coefficient. So the confidence interval for w_1 would be: $w_1 \pm 2SE(w_1)$. Standard errors are used for the hypothesis test on the coefficient, where the null hypothesis is that there is not relationship between the predictor variable and the target variable. In other words, the true $w = 0$. This is computed using the t-statistic:

$$t = \frac{\hat{w}_1 - 0}{SE(\hat{w}_1)} \tag{3.3}$$

which measures the number of standard deviations our estimate coefficient $\hat{w}_1$ is from 0. Notice we put the hat symbol, ^, over w to remind us that it is an estimate. The distribution of the t-statistic has a bell shape which makes it easy to compute the probability of observing a t-statistic larger in absolute value than what was computed, if the null hypothesis were true. This is the p-value. If the p-value is small we can reject the null hypothesis. Typical cut-off points for the p-value or 0.05 and 0.01. One caveat about p-values is that generally you will have more confidence in them if your data size is greater than 30. That is not the case for the women data because it has only 15 observations.

The final part of the summary() output of the women linear regression model is:

```
Residual standard error: 1.525 on 13 degrees of freedom
Multiple R-squared:  0.991,Adjusted R-squared:  0.9903
F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

Whereas the metrics for the coefficients indicate how well each coefficient modeled the true data, these statistics tell us how well the model as a whole fit the training data. The RSE, residual standard error, is computed from the RSS, residual sum of squares.

$$RSS = e_1^2 + e_2^2 + ... + e_n^2 = \sum_i (y_i - \hat{y}_i)^2 \tag{3.4}$$

The RSS is just the sum of squared errors. We square them because some will be errors in the positive direction and some will be in the negative direction. The RSE is computed from the RSS:

$$RSE = \sqrt{\frac{1}{n-2} RSS} \quad = \quad \sqrt{\frac{1}{n-2} \sum_i (y_i - \hat{y}_i)^2} \tag{3.5}$$

Why is RSS scaled by $\frac{1}{n-2}$? The value n is the number of observations and the 2 is because we have 2 estimated variables. This gives us $15 - 2 = 13$ degrees of freedom mentioned on the line

with the RSE. The RSE measures how off our model was from the data, the lack of fit of the model. It is measured in units of y so in this case our RSE of 1.525 is about 1.5 inches.

Since RSE is in terms of Y it can be hard to interpret. For this reason the R^2 statistic is also provided:

$$R^2 = 1 - \frac{RSS}{TSS} \tag{3.6}$$

where TSS, total sum of squares, is a measure of how far off y values tend to be from the mean: $TSS = \sum(y_i - \bar{y})^2$. The R^2 statistic will always be between 0 and 1, the closer to 1 the more variance in the model is explained by the predictors. The R^2 is the summary above was 0.991, which is very high. This means that almost all the variation in weight is predicted by height. For linear regression models such as this one where there is only one predictor, R^2 will be the same as the correlation between the the X and Y values.

The final statistic listed is the F-statistic:

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} \tag{3.7}$$

where n is the number of observations and p is the number of predictors. The F statistic takes into account all of the predictors to determine if they are significant predictors of Y. It provides evidence against the null hypothesis that the predictors are not really predictors. The advantage of the F-statistic over R^2 is that R^2 does not tell us whether it is statistically significant or not but the F-statistic does. It has an associated p-value. So we check for a F-statistic greater than 1 and a low p-value to indicate confidence in the model.

3.3.3 Metrics for Test Set Evaluation

Earlier we ran the women data set through the linear regression algorithm and looked at some metrics for judging the quality of the model. In practice, we will not run entire data sets through the algorithm, but divide the data into a training set and a test set. The model should be built on the training data and should not see the test data until evaluation time. Then the model is used to predict values for the test data, and we can use various metrics to see how far off the predictions were from the true values. We will do that in future examples. For now, we are going to just make up some test data out of the blue, making sure that the data does not fit well to the regression line so it shows up well on the graph. Again, we are only making up data for illustration purposes.

What are some metrics we can use to evaluate the prediction accuracy? For regression tasks, common metrics include mean-squared error and correlation. Imagine that we randomly selected a few data observations to hold out from the training data. These observations would not have been seen by the algorithm during training and so we can use them to test the model. It is very important that the algorithm be tested on unseen data, otherwise we don't know if the algorithm learned anything or just memorized data. Now using our test data to predict weight, given height, we can compute the correlation of the actual y values with the predicted y values with the R cor() function: $cor(predicted, actual)$. Correlation gives us a general idea about our model. A correlation close to +1 would mean that as height went up, weight went up as well. However, this would not tell us how much our estimates were off for the individual observations. These errors are called residuals:

```
residuals <- predicted - actual
```

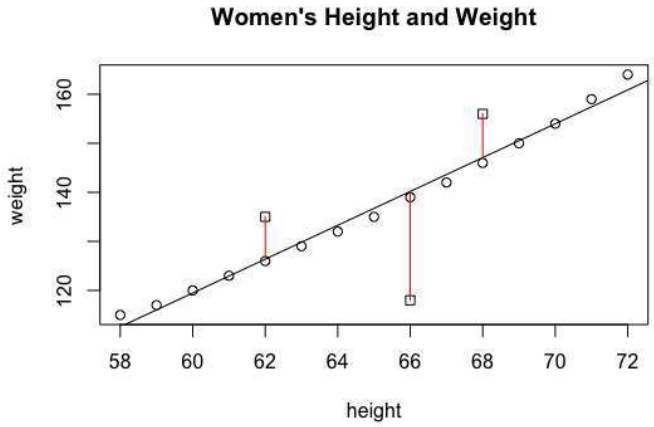

Figure 3.2: Test Set Errors

In Figure 3.2 these residuals are represented as vertical lines drawn from the data points to the regression line. Some errors may be in the positive direction and some may be in the negative direction. For this reason they are squared to find mean squared error:

$$mse = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.8}$$

This averages the squared difference between the actual values (y) and the predicted values (y-hat) over all elements in the test set. Sometimes the square root of the above is used, rmse (root mean squared error), since it will be in units of y. The metrics mse or rmse are useful in comparing two models built on the same training data.

Code 3.3.1 — Hallucinated Test Data. Women Height-Weight Model.

```
# fake test data
test <- women[c(5, 9, 11),]
test[1, 2] <- 135
test[2, 2] <- 118
test[3, 2] <- 156
# predictions on test data
pred <- predict(lm1, newdata=test)
# metrics
correlation <- cor(pred, test$weight)
print(paste("correlation: ", correlation))
mse <- mean((pred - test$weight)^2)
print(paste("mse: ", mse))
rmse <- sqrt(mse)
print(paste("rmse: ", rmse))
[1] "correlation:  0.38404402702441"
[1] "mse:  215.284722222223"
[1] "rmse:  14.6725840335717"
```

We see in Code 3.3.1 that our correlation is 0.38 which is not that good, unsurprisingly, since we purposely made up data that was far from the regression line. As you can see in Figure 3.2, the 3 test examples, shown as squares, are not that close to the regression line. Two are above the line and one is below. Red lines show the residuals, the errors. We quantify the amount of error with mse and rmse. The mse value can be hard to interpret in isolation; it is most useful in comparing models. The rmse however is in units of y. In this case, we see that our test data was off by 14.67 pounds on average.

3.4 The Algorithm

The example above is called *simple linear regression* because it had only one predictor variable, height, for the target variable weight. When we have more than one predictor variable as we will see below, this is called *multiple linear regression*. In general, the more predictors are added to a model, the better it will be but that does not mean you should just add all the predictors. We will learn techniques for determining which predictors should be included.

Recall that the coefficients for the simple linear regression model above were $w = 3.45$ and $b = -87.51667$. So looking at an observation where height is 64 inches, we would expect weight to be:

$3.45 * 64 - 87.511667 = 133.28$

and we see from Figure 3.2 that this is a reasonable estimate of weight for this height. These parameters w and b were learned from the training data, but how? We will dig deeper into the math below, but the general idea is that we want a line than minimizes the residuals, the errors, designated as e:

$$\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} e_1^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.9}$$

The above equations states that we want the parameters w, b that minimize the squared errors over all n examples in the training data. The estimated values of w and b are:

$$\hat{b} = \bar{y} - \hat{w}\bar{x} \qquad \hat{w} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{3.10}$$

As seen above, these equations rely on mean values of x and y to compute w and b. To prove to ourselves that these equations find the parameters w, b, let's find the mean of x and y in the women's data and plug them into the equations.

Code 3.4.1 — Verifying the equations. Manually Computing Coefficients.

```
x <- women$height
y <- women$weight
x_mean <- mean(women$height)
y_mean <- mean(women$weight)

w_hat <- sum((x-x_mean)*(y-y_mean)) / sum((x-x_mean)^2)
b_hat <- y_mean - w_hat * x_mean
print(paste("w and b estimates = ", w_hat, b_hat))

[1] "w and b estimates =  3.45 -87.5166666666667"
```

The algorithm described above for linear regression is called the **ordinary least squares (OLS) method**. Next we explore how these equations are derived.

3.5 Mathematical Foundations

Our goal in the OLS method is to reduce the errors over all the training data. These errors are quantified in the residual sum of errors, RSS:

$$RSS = e_1^2 + e_2^2 + ... + e_n^2 \tag{3.11}$$

Each error, e, in turn is the difference between the actual y value and the predicted value:

$$RSS = (y_1 - b - wx_1)^2 + (y_2 - b - wx_2)^2 + ... + (y_n - b - wx_n)^2 \tag{3.12}$$

The **loss function** describes how much accuracy we lose in our model. The first equation below specifies the loss for one year, the second averages the loss over all the years. By the way, you will also see this called a **cost function**. The terms loss function, cost function, and error function are used somewhat synonymously, but unfortunately inconsistently across the literature. In the equations below we see the loss function with subscript i to indicate the loss for one instance, and the loss function without the subscript indicates the loss averaged over all examples.

$$\mathcal{L}_i = (y_i - f(x_i))^2 \tag{3.13}$$

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{3.14}$$

Our goal is to find the parameters (coefficients) that minimize these errors on the training data:

$$\underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} e_1^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.15}$$

One method for finding the coefficients is to take the partial derivatives, set them to zero, and solve. This produces the normal equations given in the last section. Here we show the details of how they are derived. The equations above express the loss function in algrebraic notation. We could find the derivative with this notation but it is more concise if we use matrix notation. We are now going to consider parameter b to be one of possibly many parameters (in this case only two) in a vector. Specifically, we will refer to parameter b as w_0. Additionally we will express x as a vector, making the first element 1 so that it multiplies by w_0, the intercept.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$

$$f(x) = \mathbf{w}^T\mathbf{x} = w_0 + w_1 x_1$$

In the notation above, only one predictor is used. For multiple predictors the w and x vectors would expand accordingly. Let's visualize this matrix representation of f(x) for the women data set.

$$\mathbf{y}\begin{bmatrix}115\\117\\120\\123\\126\\...\\154\\159\\164\end{bmatrix} = \mathbf{w}\begin{bmatrix}-87.5167\\3.45\end{bmatrix}^T \quad \mathbf{X}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & ... & 1 & 1 & 1\\58 & 59 & 60 & 61 & 62 & ... & 70 & 71 & 72\end{bmatrix}$$

Our loss function expressed in matrix notation, with i indexing indivdual observations:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{n}(\mathbf{y}_i - \mathbf{w}^T\mathbf{x}_i)^2 \tag{3.16}$$

Rewriting the square term above into the form shown below uses the property that $(\mathbf{Xw})^T = \mathbf{w}^T\mathbf{X}^T$:

$$\mathcal{L} = \frac{1}{N}(\mathbf{y} - \mathbf{Xw})^T(\mathbf{y} - \mathbf{Xw}) \tag{3.17}$$

Next we go through a few steps to multiply out and collect terms. First, bring the transpose inside the parenthesis.

$$\mathcal{L} = \frac{1}{N}(\mathbf{y}^T - (\mathbf{Xw})^T)(\mathbf{y} - \mathbf{Xw}) \tag{3.18}$$

Multiply out:

$$\mathcal{L} = \frac{1}{N}\mathbf{y}^T\mathbf{y} - \frac{1}{N}\mathbf{Xwy}^T - \frac{1}{N}(\mathbf{Xw})^T\mathbf{y} + \frac{1}{N}(\mathbf{Xw})^T\mathbf{Xw} \tag{3.19}$$

We can combine the middle two terms because $w^T X^T y$ and $y^T Xw$ are transposes of one another and scalars.

$$\mathcal{L} = \frac{1}{N}\mathbf{y}^T\mathbf{y} - \frac{2}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{Xw} \tag{3.20}$$

Before we take the partial derivative wrt w, we can get rid of the first term since it doesn't involve w.

$$\mathcal{L} = -\frac{2}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{Xw} \tag{3.21}$$

Now use the rules for matrix partial derivatives to get:

$$\frac{\partial\mathcal{L}}{\partial w} = -\frac{2}{N}\mathbf{X}^T\mathbf{y} + \frac{2}{N}\mathbf{X}^T\mathbf{Xw} \tag{3.22}$$

Above we used the fact that the partial derivative of $w^T x = x$ for the first term and $w^T w = 2w$ for the second term. We simplify the equation to:

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} \tag{3.23}$$

And so our estimated parameters must be:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.24}$$

In practice, finding the inverse matrix above is computationally intense, $O(n^3)$, and therefore very slow for large data sets. For this reason, R uses other mathematical techniqes to find the parameters. One technique is gradient descent, described next.

3.5.1 Gradient descent

Unlike directly finding the inverse in the normal equation above, gradient descent will not get bogged down for large data sets. The algorithm starts with some value for the parameters **w** and keeps changing them in an iterative loop until they find a minimum. Figure 3.3 visualizes a convex function with a random starting point symbolized by the higher red dot. One iteration may move the red dot to the second location. This is one step.

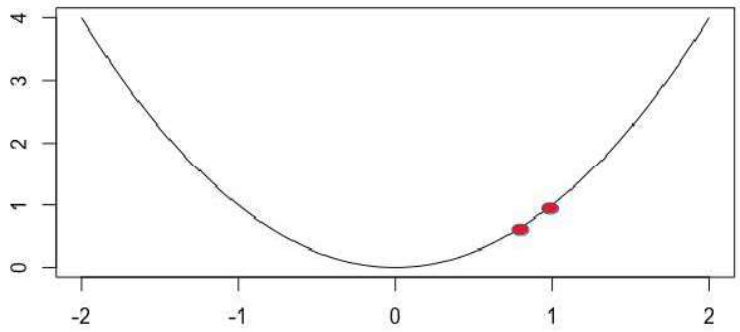

Figure 3.3: Searching for Error Minimum

The gradient descent algorithm repeats the parameter update until convergence:

$$w_j := w_j - \alpha\frac{\partial\mathcal{L}}{\partial w} \tag{3.25}$$

Note that all parameters in w are updated at every iteration. The derivative gives us the slope and the alpha determines our step size. If alpha is too small, the algorithm will be slow. On the other hand, if it is too large we could overshoot the minimum and fail to converge.

3.6 Multiple Linear Regression

Next we look at another example of linear regression in R. We will use the R build-in data set ChickWeight, which has 578 rows and 4 columns of data resulting from an experiment on the effect of different types of feed on chick weight. We will use weight as our target, with the following predictors:

- Time - number of days since birth

- Diet - a factor representing 4 different diets

We will ignore the Chick column which identifies the chicken. In **simple linear regression** we use only one predictor. In **multiple linear regression** we use more than one predictor. We will do both on this data set. First, let's make a couple of plots to visualize the data.

Code 3.6.1 — Plots. Use par() to set up a 1x2 grid for the plots.

```
par(mfrow=c(1,2))
plot(ChickWeight$Time, ChickWeight$weight,
     xlab="Time", ylab="Weight")
plot(ChickWeight$Diet, ChickWeight$weight,
     xlab="Diet", ylab="Weight")
```

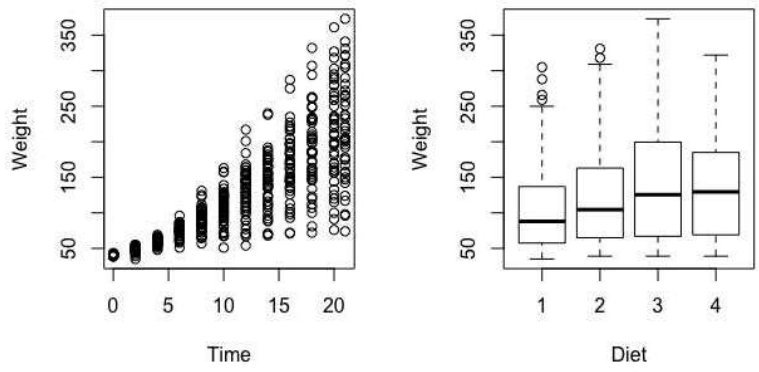

Figure 3.4: ChickWeight Weight Impacted by Time, Diet

In Figure 3.4 we see in the plot on the left that chicks gain wait over time. That is not surprising but what the plot also shows is that chicks start off at near identical weights and diverge over time. The plot on the right shows the impact of diet on weights. Box plots show the median value as the horizontal line through the box, the box itself shows the IQR (inter-quartile range from the first to the third quartile), and the horizontal lines at the end of the dashed lines show minimum and maximum values, not including suspected outliers which appear as dots beyond the horizontal lines. In this example it appears that diet 4 is slightly better than diet 3 and diet 1 appears to result in the lowest weights for chicks.

Next we divide the data into train and test sets by randomly sampling the rows. We set a seed so that each time we run this code we should get the same results. Vector i will contain the row numbers so we can subset the data frame with indices i to get a train set and *not i, or -i* to get the test set. After this 75/25 split, the train set has 433 observations and the test set has 145.

Code 3.6.2 — Divide data into train and test sets. Made a 75/25 split.

```
set.seed(1234)
i <- sample(1:nrow(ChickWeight), nrow(ChickWeight)*0.75, replace=FALSE)
train <- ChickWeight[i,]
test <- ChickWeight[-i,]
```

3.6.1 Interpreting summary() output for a linear model

Next we create a linear model on the train set, using only Time as a predictor. Let's look at the output of summary() in more detail. First it echoes back the code used to build lm1.

Next we see the distribution of the residuals, the errors. We want to see that the residuals are symmetrically distributed around the mean or median. There is a wide range here, from the minimum of -140 to the maximum of 158. That is not encouraging. The wide range of residuals confirms what we saw in Figure 3.4, that chick weight diverges widely as time goes on.

The Coefficients section lists for each predictor and the intercept, the estimated value, standard error, t value and p value, followed by significance codes. If you recall from an earlier statistics class, a t-value measures variation in the data, and the associated p-value estimates confidence in that value. A low p-value indicates evidence for rejecting the null hypothesis that the predictor does not influence the targer variable. We want to see low p-values and in this case we do. Time has a low p-value and correspondingly, 3 asterisks. The estimate for our one predictor, Time, is 8.952. This means we would expect chicks to gain an average of almost 9 gm a day. The standard error measures the average amount the coefficient estimate varies from the actual values. The t-value is a measure of how many standard deviations the coefficient estimate was from 0. The further it is from 0, the more confidence we have in rejecting the null hypothesis, in this case that Time has no effect on chick weight. The p-value gives the probability of observing a similar or larger t-value due to chance, given the data. A small p-value gives us confidence that there really is a relationship between our predictor(s) and the target variable. The chart for the significance codes is given at the bottom of this section.

The last section gives some statistics on the model. The residual standard error, RSE, is in units of y. In this case our RSE was 41.4, so the average error of the model was about 41 gm. This statistic was calculated on 431 degrees of freedom: we had 433 data points minus 2 predictors. Multiple R-squared is scaled from 0 to 1 and so is easier to interpret than RSE. The adjusted R-squared is 0.6863 which is not bad. This means that 68% of the variance in the model can be explain by our predictor. The adjusted R-squared takes into account the number of predictors. This is important because R-squared tends to increase as we add predictors, and the adjusted R-squared accounts for this. Finally the F-statistic also measures whether our predictors and target are related. We want to see the F-statistic be far away from zero and its associated p-value to be low. The more data observations we have, the lower the F-statistic can be to confirm the relationship. This is why the p-value is important.

Code 3.6.3 — Linear model 1. Using only Time as a predictor.

```
lm1 <- lm(weight~Time, data=train)
summary(lm1)
```

```
Call:
lm(formula = weight ~ Time, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-140.314  -16.648    0.778   14.682  158.686

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   26.318      3.743   7.031 8.06e-12 ***
Time           8.952      0.291  30.760  < 2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 41.4 on 431 degrees of freedom
Multiple R-squared:  0.687,Adjusted R-squared:  0.6863
F-statistic: 946.2 on 1 and 431 DF,  p-value: < 2.2e-16
```

3.6.2 Residuals

Plotting the residuals results in 4 plots, which we have arranged in a 2x2 grid in Figure 3.5. How do we interpret these plots? A comprehensive explanation is given here: `http://data.library.virginia.edu/diagnostic-plots/`. Below we provide a brief overview of the 4 plots:

Code 3.6.4 — Plot the residuals. The residuals give us information about how well the model fits the data.

```
par(mfrow=c(2,2))
plot(lm1)
```

Plotting the residuals results in 4 plots, which we have arranged in a 2x2 grid in Figure 3.5. How do we interpret these plots? A comprehensive explanation is given here: `http://data.library.virginia.edu/diagnostic-plots/`. Below we provide a brief overview of the 4 plots:

1. Plot 1 Residuals vs Fitted: This plots the residuals (errors) with a red trend line. You want to see a fairly horizontal red line. Otherwise, the plot is showing you some variation in the data that your model did not capture.
2. Plot 2 Normal Q-Q: If the residuals are normally distributed, you will see a fairly straight diagonal line following the dashed line.
3. Plot 3 Scale-Location: You want to see a fairly horizontal line with points distributed equally around it. If not, your data may not be homoscedastic (means "same variance").
4. Plot 4 Residuals vs Leverage: This plot will indicate leverage points which are influencing the regression line. They may or may not be outliers, but further investigation is warranted. An **outlier** is a data point with an unusual y value whereas a **leverage point** is a data point with an unusual x value.

Looking at Figure 3.5 we see some problems with our model. In the first graph, the red line is horizontal but notice that the residuals vary more as we go to the right. This confirms our very first observation of the data, that chicks vary greatly in how much weight they gain. Time and even diet cannot account for this. What is not accounted for in our model is the genetic contribution. We humans start off life at an average of about 7 pounds but end up as adults in a wide range of weights. We could consider genetics a hidden or unseen variable in this experiment. Also chick gender was not included in the data, so we have no way of knowing if this influenced the weight variation. The second graph indicates that most of the residuals are normally distributed except for those in the lower range. So there is variation in the data that our model does not capture.

Let's build another model using Time and Diet as predictors. The adjusted R-squared for lm2 is 0.7338 which is higher than the adjusted R-squared for lm1, which was 0.6863. Also, RSE has decreased to 38.13 from 41.1 in lm1. Adding Diet seems to have improved our model.

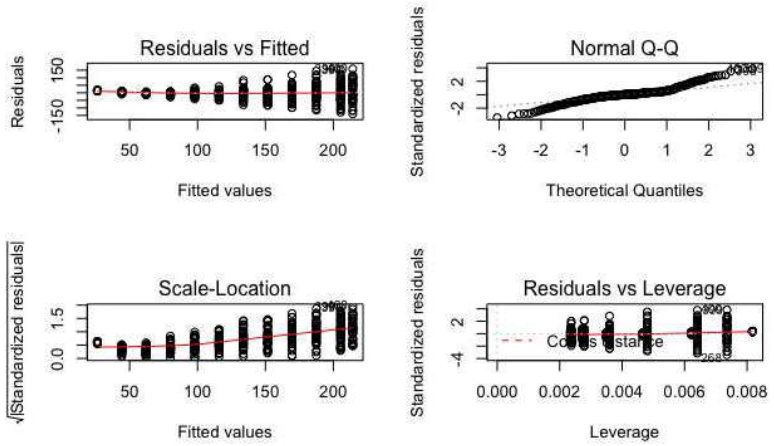

Figure 3.5: Residuals Plot

Code 3.6.5 — Linear model 2. Using Time and Diet as predictors.

```
lm2 <- lm(weight~Time+Diet, data=train)
summary(lm2)
```

```
Call:
lm(formula = weight ~ Time + Diet, data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-137.857  -20.492   -1.685   16.955  137.365

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   8.4109     4.1372   2.033 0.042670 *
Time          8.9086     0.2682  33.218  < 2e-16 ***
Diet2        16.3645     4.9235   3.324 0.000965 ***
Diet3        40.1424     4.8907   8.208 2.67e-15 ***
Diet4        32.1873     5.2503   6.131 1.99e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 38.13 on 428 degrees of freedom
Multiple R-squared:  0.7363,Adjusted R-squared:  0.7338
F-statistic: 298.8 on 4 and 428 DF,  p-value: < 2.2e-16
```

3.6.3 Dummy variables

Looking at the summary output for lm2, we see that Time and Diet were good predictors. Why do we have Diet2, Diet3, Diet4? Since Diet is a factor with 4 levels, R made 3 **dummy variables** for us. The base model represents Diet1, the dummy variables Diet2 - Diet 4 tell us how much each diet impacted the model compared to Diet 1. Recall from the boxplot above, that Diet1 resulted in the lowest weights and we see that confirmed in this summary, because Diet2 - Diet4 each have

positive coefficients. For each data observation, only only of the dummy variables will be active with the others being zero. So for a chick on Diet 1, the dummy variables for Diets 2 through 4 would be zero.

3.6.4 The anova0 function

We can compare the summary() statistics of models to guage their relative value. Another way to compare them is to run anova() on the two models. The anova() function lists each model and provides similar statistics as the summary() function for each model. We see that the RSS is lower for model 2, and model 2 is given a low p-value. This is confirmation that lm2 outperformed lm1.

Code 3.6.6 — The anova() function. Analysis of Variance.

```
anova(lm1, lm2)
```

```
Analysis of Variance Table

Model 1: weight ~ Time
Model 2: weight ~ Time + Diet
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1    431 738546
2    428 622323  3    116222 26.644 8.107e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Let’s try one more thing. Recall from Figure 3.5 that there was a funnel shape in the residuals in that they became more spread out from left to right. The log function damps down x values across the axis. Perhaps this could squish the chick weights closer about the linear regression line.

Code 3.6.7 — Linear model 3. Linear models are not always a straight line.

```
lm3 <- lm(log(weight)~Time+Diet, data=ChickWeight)
summary(lm3)
```

```
. . .
Residual standard error: 0.2281 on 573 degrees of freedom
Multiple R-squared:  0.8484,Adjusted R-squared:  0.8474
F-statistic:   802 on 4 and 573 DF,  p-value: < 2.2e-16
```

Above we show only the statistics portion of the summary() output. Now the R-squared has increased to 0.8474, indicating that lm3 may be better than lm1 and lm2. We cannot run anova on the 3 models because lm3 has a different response: log(weight) instead of weight. But we could look at the residuals plots to look for improvement.

3.7 Polynomial Linear Regression

To emphasize the point that linear regression is not always a straight line, we next look at polynomial linear regression, where the fitted line may be of higher degree than 1. We will perform polynomial regression on the cars dataset, included in R. The data set has 50 observations and 2 variables: speed and stopping distance. Using the range() function on the columns we see that speed ranges from 4 to 25 mph, and dist ranges from 2 to 120 feet. The data was collected in the 1920s. The code example is from the R documentation. The plot() call at the top of the code sets up the plot.

The seq() call sets up a sequence for the s values we want to plot across the horizontal axis. The for loop plots models of degree 1 through 4 in different colors. Colors 1-4 correspond to black, red, green3, blue. The poly() function is used to create orthogonal (not correlated) polynomials. Finally an anova() is run on the 4 models.

Code 3.7.1 — Polynomial Regression.

```
plot(cars, xlab = "Speed (mph)", ylab = "Stopping distance
    xlim = c(0, 25))
s <- seq(0, 25, length.out = 200)
for(degree in 1:4) {
  fm <- lm(dist ~ poly(speed, degree), data = cars)
  assign(paste("cars", degree, sep = "."), fm)
  lines(d, predict(fm, data.frame(speed = s)), col = degree)
}
anova(cars.1, cars.2, cars.3, cars.4)
```

```
Analysis of Variance Table

Model 1: dist ~ poly(speed, degree)
Model 2: dist ~ poly(speed, degree)
Model 3: dist ~ poly(speed, degree)
Model 4: dist ~ poly(speed, degree)
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1     48 11354
2     47 10825  1    528.81 2.3108 0.1355
3     46 10634  1    190.35 0.8318 0.3666
4     45 10298  1    336.55 1.4707 0.2316
```

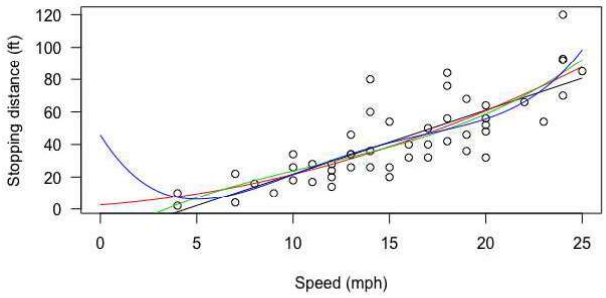

Figure 3.6: Cars with Polynomial Degree 1:4

3.8 Model Fitting and Assumptions

In this section we explore important concepts in machine learning that relate to how well a model fits the data. Overfitting is a common problem in many machine learning algorithms we will learn. And we will discuss the bias-variance tradeoff of various algorithms as we learn them.

3.8.1 Overfitting v. underfitting

The anova() results from the polynomial regression indicate the smallest RSS with the degree 4 model. However, none of the p-values are significant, and it is difficult to draw firm conclusions from such a small set of data points. However, the graph in Figure 3.6 gives us an opportunity to talk about underfitting versus overfitting. The linear degree=1 model probably underfits. In contrast the degree 3 and 4 models might be overfitting the data. When you underfit the data, your model does not have sufficient complexity to explain the data. That is what we see with the degree=1 model. The straight line is not capturing some of the complexity in the data. On the other hand, overfitting is when the model has too much complexity. The principle of **Occam's razon** tells us that when choosing between two likely explanations, choose the simpler one. In this case we might choose degree=2 since it did have a lower p-value. In a scenario where you have a train and test set, if the data performs well on the training data but poorly on the test data you may have overfit. Your model learned variation in the data that occured in the training set and this limited its ability to generalize to new data. On the other hand, if your model does poorly on the training set, you may have underfit. Figure 3.7 illustrates overfitting versus underfitting.

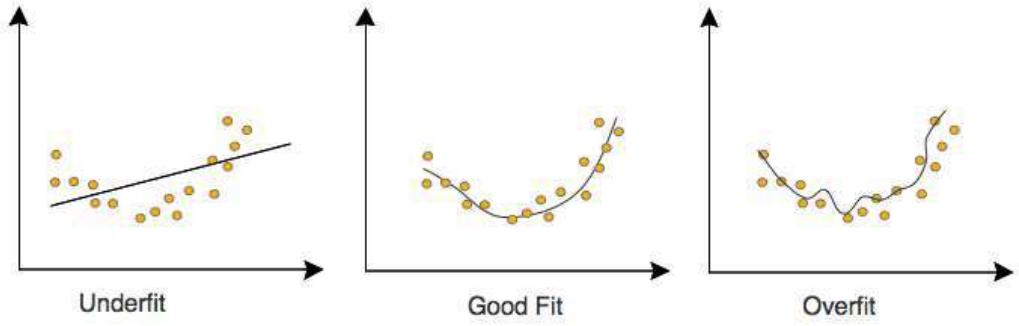

Figure 3.7: Fitting a model

3.8.2 Bias and variance

A common theme throughout this handbook will be the bias-variance tradeoff which is related to underfitting and overfitting. Each algorithm that we learn will have tendencies one way or the other. In the underfitting illustration in Figure 3.7, we see that the straight line underfit. It had a strong bias that the data was truly linear and is too simple a model for the data. On the other extreme, the overfitting example showed what can happen if variance is too high: the model learned random things from the data.

So bias-variance and underfitting-overfitting are correlated but it is important to distinguish their seperate meanings. A high bias, low variance model is likely to underfit and not capture the true shape of the data. This tends to happen more with simpler models like linear regression or logistic regression. In contrast, a low bias, high variance model captures too much complexity and noise in the data and will not generalize well to new data. This can happen with more complex models like decision trees, SVM, or neural networks.

So if you suspect your model has high bias, what can you do about it? We could try different algorithms. Right now, that's not helpful because we have only learned one algorithm. As we go we will learn more algorithms and learn their tendencies towards either bias or variance. We will see one tool in this chapter however that can help, and that is adding a regularization term that will help linear regression pull back it's tendency towards bias. Also in this chapter we learned that a linear model doesn't necessarily mean a straight line. It can be a polynomial line or any mathematical

transformation of the linear equation. One more technique we can try to reduce bias is to add more features, if available.

If you suspect your model is suffering from high variance, what can you do about it? More data should help. Algorithms that tend toward high variance are overly sensitive to noise in the data. Adding more data can quiet the effect of a few noisy observations. Another approach is to try fewer features if you are using a lot of features. Some features may be noiser than others so this could help.

3.8.3 Linear Model Assumptions

A linear model first and foremost assume some linear shape in the data. Beyond that, the linear model also has an **additive assumption**, that each predictor contributes to the model independently of the other predictors. In reality, some predictors may be correlated, in which case we might consider removing one of them, since it will be hard for the model to assess the effect of them independently and thus the coefficient estimates may be erroneous. Other predictors may have an **interaction effect**, a synergy between them. Yet another concern is **confounding** variables, which are variables that correlate with both the target and a predictor. How can we detect these situations? We can use the cor() and pairs() methods in R to quantify and visualize correlations in the data set.

3.9 Advanced Topic: Regularization

An extension of the least squares approach is to add a regularization term to the RSS, with its importance controlled by another parameter, lambda. This extra term penalizes large coefficients. When lambda=0 it is the same as the least squares estimate. As lambda gets larger, the coefficients will shrink. The intercept does not shrink because the goal is to reduce the coefficients associated with predictors. The notation $||w||^2$ denotes the *l2* norm, which is $\sqrt{(\sum_{j=1}^{p})w_j^2}$. Regularization can help prevent overfitting when you have relatively complex models on a small data set.

$$\mathbf{RSS} = \sum_{i=1}^{n}(\mathbf{y}_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda||\mathbf{w}||^2 \tag{3.26}$$

We can implement ridge regression with the R package glmnet. We will use the airquality data set. Since airquality has a lot of NAs we will omit observations with NAs in the columns we care about with the complete.cases() function. Before performing ridge regression, we build a multiple linear regression model as usual. indexregularization!ridge regression

Code 3.9.1 — Linear Regression. Multiple Linear Regression on Airquality

```
df <- airquality[complete.cases(airquality[, 1:5]),]
df <- df[,-6]

set.seed(1234)
i <- sample(1:nrow(df), .75*nrow(df), replace=FALSE)
train <- df[i,]
test <- df[-i,]
lm1 <- lm(Ozone~., data=train)
pred <- predict(lm1, newdata=test)
mse1 <- mean((pred-test$Ozone)^2)
print(paste("mse=", mse1))
```

The output above for the mse is 409.3799. Let's see if we can beat that with ridge regression. First we use the model.matrix() function to create a matrix of the predictors. Then we split into the same train and test observations as for lm1.

Code 3.9.2 — Regularization. Ridge Regression On Airquality

```
library(glmnet)
x <- model.matrix(Ozone~., df)[,-1]
y <- df$Ozone
train_x <- x[i,]
train_y <- y[i]
test_x <- x[-i,]
test_y <- y[-i]

# build a ridge regression model
rm <- glmnet(train_x, train_y, alpha=0)

# use cv to see which lambda is best
set.seed(1)
cv_results <- cv.glmnet(train_x, train_y, alpha=0)
l <- cv_results$lambda.min

# get data for best lambda, which is the 99th
pred2 <- predict(rm, s=l, newx=test_x)
mse2 <- mean((pred2-test_y)^2)
coef2 <- coef(rm)[,99]
```

Our mse for the ridge regression was 371.0138, which is about 10% lower than for the regular multiple regression. Let's confirm that the ridge regression shrunk our coefficients. It appears that all the coefficients shrunk a bit.

```
> lm1$coefficients
 (Intercept)      Solar.R         Wind         Temp        Month
-66.85709002   0.08314323  -3.75229006   1.98524049  -3.27749222

> coef2
 (Intercept)      Solar.R         Wind         Temp        Month
-60.80449134   0.08165752  -3.61256523   1.83183505  -2.60738344
```

3.10 Summary

In this chapter we learned the supervised regression technique of linear regression, where our target was a real number variable and our predictors could be any combination of quantitative or qualitative variables. Linear regression has a strong bias in that it assumes that the relationship between the target and the predictors is linear. Keep in mind that *linear* does not always mean a straight line as we saw in the examples.

When we run the linear regression algorithm on training data, we create a *model* of the data that can then be used for predictions on new data. Our model gives us the coefficients which quantify the effect of each predictor on the target variable. Linear regression strengths and weaknesses:

Strengths:

- Relatively simple algorithm with an intuitive explanation because the coefficients quantify the effect of predictors on the target variable.
- Works well when the data follows a linear pattern.
- Has low variance.

Weaknesses:

- High bias because it assumes a linear shape to the data.

3.10.1 Terminology

This chapter introduces a lot of terminology as we explored simple linear regression, multiple linear regression, and polynomial linear regression. There is a glossary at the end of the book but you might want to read back through the chapter again if you are unsure of the meaning of any of the following terms.

Terms related to the data:

- outlier
- leverage point
- dummy variables
- confounding variables

Terms related to the algorithm:

- coefficients
- residuals
- loss function, or cost function
- gradient descent
- additive assumption of linear regression
- interaction effect in linear predictors

Terms related to metrics:

- correlation
- covariance
- mse mean squared error
- rmse root mean squared error
- rss residual sum of squared errors
- rse residual standard error
- R^2 and adjusted R squared
- F-statistic
- p-value

Terms relevant to all machine learning algorithms:

- overfitting
- underfitting
- bias
- variance
- regularization

3.10.2 Quick Reference

Reference 3.10.1 Create Train and Test Sets

```
set.seed(...)
i <- sample(1:nrow(df), nrow(df)*0.8, replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

Reference 3.10.2 Build and Examine a Linear Regression Model

```
lmName <- lm(formula, data=train)
summary(lmName)
```

Reference 3.10.3 Predict on Test Data

```
pred <- predict(lmName, newdata=test)
mse1 <- mean((pred - test$y)^2)
cor1 <- cor(pred, test$y)
```

Reference 3.10.4 Comparing Models with anova()

```
anova(model1, model2, ..., modeln)
```

3.10.3 Practice

tbd

3.10.4 Exploring Concepts

tbd

3.10.5 Going Further

There are entire statistics courses devoted to linear regression and scores of books if you have time for a deep exploration. Here are some recommendations:

- Free online linear regression tutorial here: `https://onlinecourses.science.psu.edu/stat501/node/250`
- *Linear Models in Statistics* by Rencher and Schaalje. This book is available online or physical book at the UTD library.
- *Linear Models with R* by Faraway. This book is available online at the UTD library site.