Contents

C H A P T E R 1

Regression

Classification tries to predict a class from a data item. Regression tries to predict a value. There are several reasons to do this. First, we might actually need to predict a value. For example, we know the zip code of a house, the square footage of its lot, the number of rooms and the square footage of the house, and we wish to predict its likely sale price. As another example, we know the cost and condition of a trading card for sale, and we wish to predict a likely profit in buying it and then reselling it. As yet another example, we have a picture with some missing pixels – perhaps there was text covering them, and we want to replace it – and we want to fill in the missing values. Predicting values is very useful, and so there are many examples like this.

Second, we might want to spot a trend in data. Doing so could make it clear what is really happening. Here is an example from Efron ("COMPUTER-INTENSIVE METHODS IN STATISTICAL REGRESSION", B. Efron, SIAM Review, 1988). Table 1 shows some data from medical devices, which sit in the body and release a hormone. Each is supposed to have the same behavior. The data describes devices from three production lots (A, B, and C). The important question is: Are the lots the same?

The scatter plot of figure 1.1 suggests what might be going on here; the longer a device has been in service, the less hormone it has. Now the question of whether the lots are different boils down to whether this relationship is different for each lot. To answer this question, we need to determine the relationship between time in service and hormone in more detail. This is what regression is for.

1.1 LEAST SQUARES AND LINEAR MODELS

Assume we have a dataset consisting of a set of N pairs $(\mathbf{x}_i, y_i)$. We think of y_i as the value of some function evaluated at $\mathbf{x}_i$, with some random component added. This means there might be two data items where the $\mathbf{x}_i$ are the same, and the y_i are different. We refer to the $\mathbf{x}_i$ as **explanatory variables** and the y_i is a **dependent variable**. We want to build a model of the dependence between y and $\mathbf{x}$. This model will be used to predict values of y for new values of $\mathbf{x}$, or to understand the relationships between the $\mathbf{x}$. The model needs to have some probabilistic component; we do not expect that y is a function of $\mathbf{x}$, and there is likely some error in evaluating y anyhow.

A good, simple model is to assume that

$$y = \mathbf{x}^T \beta + \xi$$

where ξ is a zero mean normal random variable with unknown variance (we will be able to estimate this later). Here β is a vector of weights. This model might look as though it will have trouble predicting cases where the y intercept is not zero (for

Number	A	B	C
1.	25.8	16.3	28.8
2.	20.5	11.6	22.0
3.	14.3	11.8	29.7
4.	23.2	32.5	28.9
5.	20.6	32.0	32.8
6.	31.1	18.0	32.5
7.	20.9	24.1	25.4
8.	20.9	26.5	31.7
9.	30.4	25.8	28.5

TABLE 1.1: *A table showing the amount of hormone remaining in devices from lot A, lot B and lot C. The numbering is arbitrary (i.e. there's no relationship between device 3 in lot A and device 3 in lot B). At first glance, lot C looks rather different from lots A and B.*

Number	A	B	C
1.	99	376	119
2.	152	385	188
3.	293	402	115
4.	155	29	88
5.	196	76	58
6.	53	296	49
7.	184	151	150
8.	171	177	107
9.	52	209	125

TABLE 1.2: *A table showing the time in service for devices from lot A, lot B and lot C. The numbering is arbitrary (i.e. there's no relationship between device 3 in lot A and device 3 in lot B), but corresponds to the numbering of table 1.2. The time may explain the differing amounts of hormone.*

example, $y = ax + b$ where b is not zero). It doesn't, because we can attach a one to the end of each $\mathbf{x}$, so that it looks like $(x_1, x_2, \ldots, x_n, 1)^T$.

We must first determine β. We must have that β minimizes

$$\sum_i (y_i - \mathbf{x}_i^T \beta)^2$$

which we can write more conveniently using vectors and matrices. Write $\mathbf{y}$ for the vector

$$\begin{pmatrix} y_1 \\ y_2 \\ \ldots \\ y_n \end{pmatrix}$$

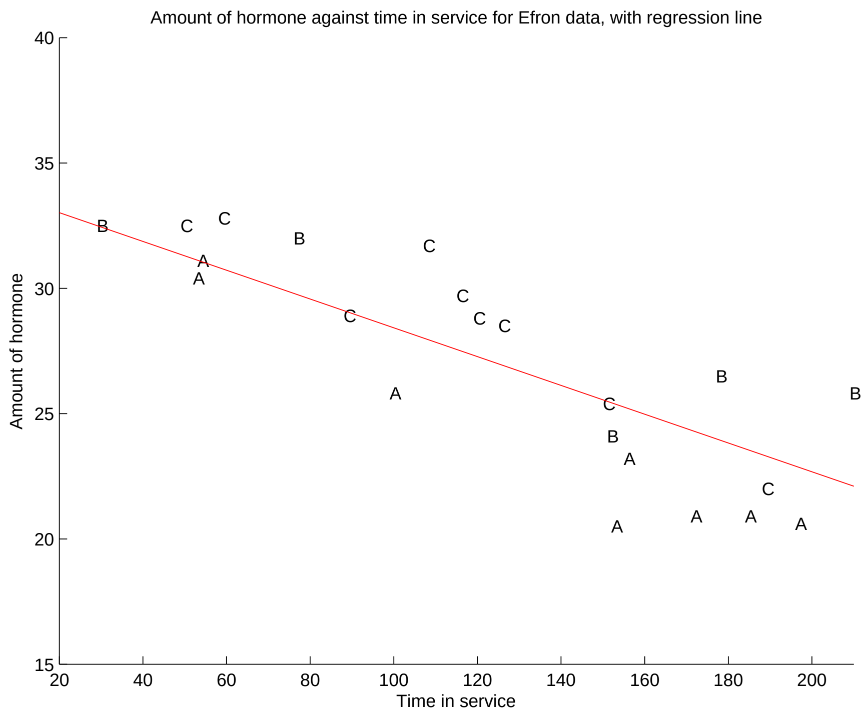

FIGURE 1.1: *A scatter plot of hormone against time for devices from tables 1.1 and 1.2. Notice that there is a pretty clear relationship between time and amount of hormone (the longer the device has been in service the less hormone there is). The issue now is to understand that relationship so that we can tell whether lots A, B and C are the same or different. The best fit line to all the data is shown as well, fitted using the methods of section 1.1.*

and $\mathcal{X}$ for the matrix

$$\begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \ldots \mathbf{x}_n^T \end{pmatrix}.$$

Then we want to minimize

$$(\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta)$$

which means that we must have

$$\mathcal{X}^T\mathcal{X}\beta - \mathcal{X}^T\mathbf{y} = 0.$$

For reasonable choices of features, we could expect that $\mathcal{X}^T\mathcal{X}$ — which should strike you as being a lot like a covariance matrix — has full rank. If it does, this equation is easy to solve. If it does not, there is more to do, which we will do below once we have done some examples.

1.1.1 Interpreting a Regression

For the data of tables 1.1 and 1.2, y is the hormone remaining in the device and $\mathbf{x} = (\text{time}, 1)$. This gives us a model of the hormone in the device as

$$y = \beta_1 \text{time} + \beta_2$$

which should look like the line in figure **??**. We get $\beta = (-0.0574, 34.2)$. Now we can ask whether some lots of device behave differently than others. One way to address this is to consider the **residual**,

$$y_i - \mathbf{x}_i^T \beta$$

which is the difference between the observed value and what the model predicts. Look at figure 1.4, which shows this residual plotted against the time. Notice that, for batch A, the model always over predicts, whereas batches B and C seem about the same; this suggests that there is something different about A — any effects caused by the hormone being taken up from the device have been accounted for by our model, and the residual shows the effects that are left.

You can regress anything against anything. For example, we will take the bodyfat dataset. The dependent variable is weight, and the explanatory variables are everything else (except the number of the example). Figure **??** shows the residual for this regression, plotted against example number. Figure **??** shows a regression of boston house prices against a variety of explanatory variables (the data set is quite well known; you can find it at the UCI repository, `http://archive.ics.uci.edu/ml/datasets/Housing`).

1.1.2 How Good is a Fit?

However, it is quite important to know whether the regression is helpful. For example, it seems highly unlikely that regressing the first digit of a telephone number against some numerological score of your name will work that well (or at all). There are some simple facts we can use to make this easier. Assume we ensure that $\mathcal{X}$ always has a column of ones (or twos, or threes, etc.) in it, so that the regression can have a non-zero x-intercept. We now fit a model

$$\mathbf{y} = \mathcal{X}\beta + \mathbf{e}$$

(where $\mathbf{e}$ is the vector of residual values) by choosing β such that $\mathbf{e}^T\mathbf{e}$ is minimized. The facts give some useful technical results.

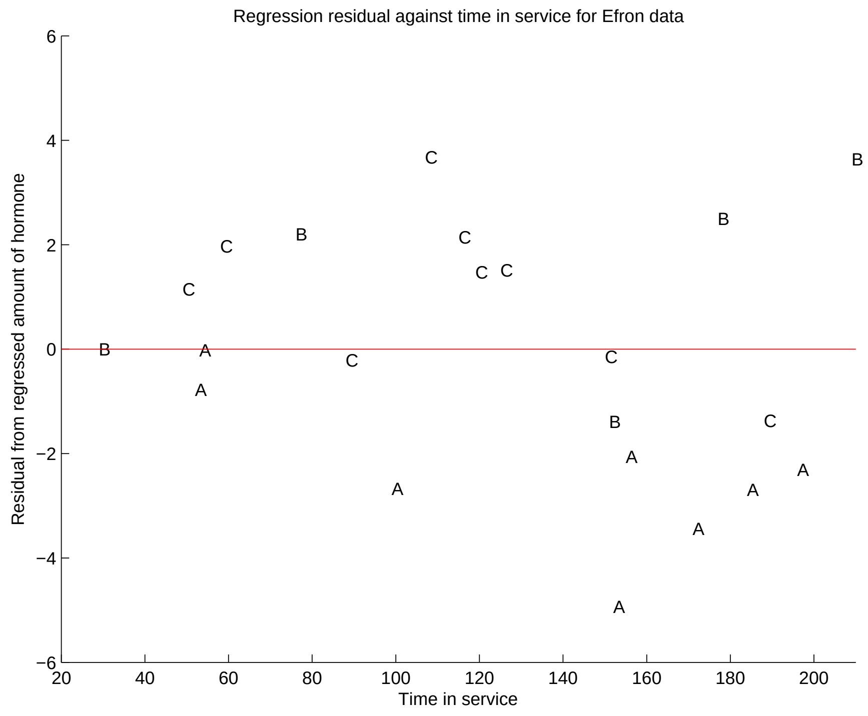

FIGURE 1.2: *This is a scatter plot of residual — the distance between each data point and the best fit line — against time for the devices from tables 1.1 and 1.2. Now you should notice a clear difference; some devices from lots B and C have positive and some negative residuals, but all lot A devices have negative residuals. This means that, when we account for loss of hormone over time, lot A devices still have less hormone in them. This is pretty good evidence that there is a problem with this lot.*

Useful facts: *Regression*

We write $\mathbf{y} = \mathcal{X}\beta + \mathbf{e}$, where $\mathbf{e}$ is the residual. Assume $\mathcal{X}$ has a column of ones, and β is chosen to minimize $\mathbf{e}^T\mathbf{e}$. Then we have

1. $\mathbf{e}^T\mathcal{X} = \mathbf{0}$, i.e. that $\mathbf{e}$ is orthogonal to any column of $\mathcal{X}$. This is because, if $\mathbf{e}$ is not orthogonal to some column of $\mathbf{e}$, we can increase or decrease the β term corresponding to that column to make the error smaller. Another way to see this is to notice that *beta* is chosen to minimize $\mathbf{e}^T\mathbf{e}$, which is $(\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta)$. Now because this is a minimum, the gradient with respect to β is zero, so $(\mathbf{y} - \mathcal{X}\beta)^T(-\mathcal{X}) = -\mathbf{e}^T\mathcal{X} = 0$.
2. $\mathbf{e}^T\mathbf{1} = 0$ (recall that $\mathcal{X}$ has a column of all ones, and apply the previous result).
3. $\mathbf{1}^T(\mathbf{y} - \mathcal{X}\beta) = 0$ (same as previous result).
4. $\mathbf{e}^T\mathcal{X}\beta = 0$ (first result means that this is true).

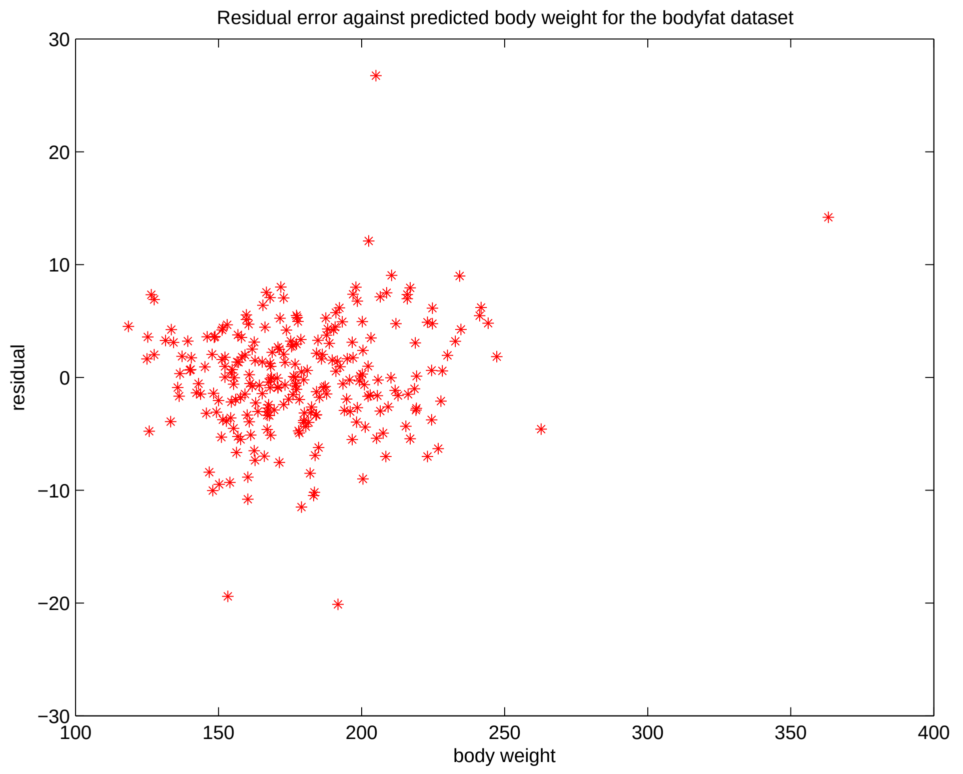

FIGURE 1.3: *This is a scatter plot of residual — the distance between each data point and the best fit line — against weight for the bodyfat dataset. You should notice that there is no particular pattern to the residuals — a good sign; if the residual had a pattern, then there is some structure in the data that the regression is not explaining. However, the residuals are moderately large. The mean weight in this dataset is 178.9, and the standard deviation of the weight is 29.4. The mean of the residual is zero — it has to be — and the standard deviation is 5. This means that the regression is explaining a lot, but not all, of the variance in the weight. This might be due to genuinely random errors, or it may be because there is some explanatory variable missing.*

Now $\mathbf{y}$ is a one dimensional dataset arranged into a vector, so $\mathsf{mean}(\{y_i\})$ is meaningful, as is $\mathsf{Var}[y_i]$. Similarly, $\mathcal{X}\beta$ is a one dimensional dataset arranged into a vector (its elements are $\mathbf{x}_i^T\beta$), as is $\mathbf{e}$, so we know the meaning of mean and variance for each. We have a particularly important result:

$$\mathsf{Var}[y_i] = \mathsf{Var}\left[\mathbf{x}_i^T\beta\right] + \mathsf{Var}[e_i].$$

This is quite easy to show, with a little more notation. Write $\overline{\mathbf{y}} = (1/N)(\mathbf{1}^T\mathbf{y})\mathbf{1}$ for

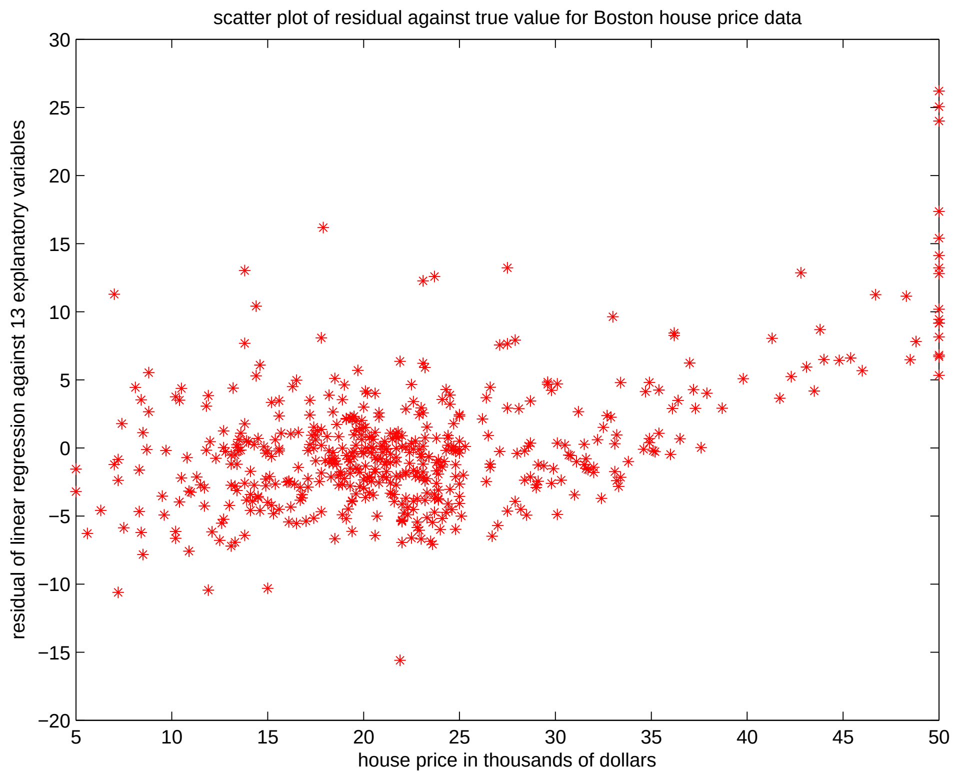

FIGURE 1.4: *This is a scatter plot of residual — the distance between each data point and the best fit line — against house price for a data set of Boston house prices. There are a total of 13 explanatory variables here. You should notice that there is no particular pattern to the residuals — a good sign; if the residual had a pattern, then there is some structure in the data that the regression is not explaining. However, the residuals are moderately large. The mean house price in this dataset is 22.5 (thousand's of dollars; this is quite old data) and the standard deviation is 9.2. The mean of the residual is zero — it has to be — and the standard deviation is 4.7. This means that the regression is explaining only about half of the variance in the house prices. This might be due to genuinely random errors, or — more likely — there is some explanatory variable missing.*

the vector whose entries are all $\mathsf{mean}\left(\{y_i\}\right)$; similarly for $\overline{e}$ and for $\overline{\mathcal{X}\beta}$. We have

$$\mathsf{Var}[y_i] = (1/N)(\mathbf{y}-\overline{\mathbf{y}})^T(\mathbf{y}-\overline{\mathbf{y}})$$

and so on for $\mathsf{Var}[e_i]$, etc. Notice from the facts that $\overline{\mathbf{y}} = \overline{\mathcal{X}\beta}$. Now

$$\begin{aligned}
\mathsf{Var}[y_i] &= (1/N)\left([\mathcal{X}\beta - \overline{\mathcal{X}\beta}] + [\mathbf{e} - \overline{\mathbf{e}}]\right)^T \left([\mathcal{X}\beta - \overline{\mathcal{X}\beta}] + [\mathbf{e} - \overline{\mathbf{e}}]\right) \\
&= (1/N)\left([\mathcal{X}\beta - \overline{\mathcal{X}\beta}]^T [\mathcal{X}\beta - \overline{\mathcal{X}\beta}] + 2\,[\mathbf{e} - \overline{\mathbf{e}}]^T [\mathcal{X}\beta - \overline{\mathcal{X}\beta}] + [\mathbf{e} - \overline{\mathbf{e}}]^T [\mathbf{e} - \overline{\mathbf{e}}]\right) \\
&= (1/N)\left([\mathcal{X}\beta - \overline{\mathcal{X}\beta}]^T [\mathcal{X}\beta - \overline{\mathcal{X}\beta}] + [\mathbf{e} - \overline{\mathbf{e}}]^T [\mathbf{e} - \overline{\mathbf{e}}]\right) \\
&\quad \text{because } \overline{\mathbf{e}} = 0 \text{ and } \mathbf{e}^T\mathcal{X}\beta = 0 \text{ and } \mathbf{e}^T\mathbf{1} = 0 \\
&= \mathsf{Var}\left[\mathbf{x}_i^T\beta\right] + \mathsf{Var}[e_i].
\end{aligned}$$

This is extremely important, because us allows us to think about a regression as explaining variance in $\mathbf{y}$. As we are better at explaining $\mathbf{y}$, $\mathsf{Var}[e_i]$ goes down. A natural measure of the goodness of a regression is what percentage of the variance of $\mathbf{y}$ it explains. This is known as R^2 (the r-squared measure). We have

$$R^2 = \frac{\mathsf{Var}\left[\mathbf{x}_i^T\beta\right]}{\mathsf{Var}[y_i]}$$

which gives some sense of how well the regression explains the training data.

One natural use of a regression is to try and find potential explanatory variables. It is quite natural to want to look at the entries in β, and wonder what they suggest about the real world. This is something that should be done with caution. For example, you might think that a β component with large absolute value is evidence that a variable is important; but it might just be evidence that that variable is poorly scaled. Similarly, you might want to try and add an explanatory variable to see if the R^2 of the regression improves; but you will find that it always does, because β is chosen to achieve the smallest possible value of R^2.

There is a procedure from hypothesis testing that will tell whether an improvement in a regression is significant (and so whether one model is better than another). Assume that we have two **nested models**. This means that model 2 sees the same explanatory variables as model one, and some extra explanatory variables as well. Write n_1 for the number of explanatory variables in model 1, n_2 for the number of explanatory variables in model 2, $\mathbf{e}_1$ for the residual under model 1, $\mathbf{e}_2$ for the residual under model 2, s_1 for $\mathbf{e}_1^T\mathbf{e}_1$, s_2 for $\mathbf{e}_2^T\mathbf{e}_2$, and N for the number of data points. If we compute the statistic

$$\frac{\frac{s_2 - s_1}{n_2 - n_1}}{\frac{s_2}{N - n_2}}$$

then, under the null hypothesis that model 2 is as good at explaining the data as model 1, this statistic will have an F-distribution with $(n2 - n1, N - n2)$ degrees of freedom. This means you do what we are used to, after chapter 1.1.2; you compute this number, look up the probability of getting this or a larger number under the null hypothesis in tables (which is where you use the degrees of freedom), and then use the significance level to decide whether to prefer model 2 to model 1 or not.

With this trick, it is natural to use regression as a tool to try and determine what effects are important. We can do this by repeatedly (a) inserting plausible explanatory variables then (b) checking whether the regression has improved or not.

We could obtain plausible explanatory variables by looking for new measurements (for example, does the acuity of color perception affect age at death?). We could take non-linear functions of existing explanatory variables. Natural choices include powers and logarithms.

Notice also that for some problems we can expect a power law — something like $y = x_1^{a_1} x_2^{a_2} \ldots x_n^{a_n}$. In this case, we have $\log y = a_1 \log x_1 + a_2 \log x_2 + \ldots a_n \log x_n$, so it makes sense to take logs and then apply a linear regression.

1.1.3 Regularizing Linear Regressions

One occasionally important difficulty is that the explanatory variables might be significantly correlated. If they are, then it will generally be easy to predict one explanatory variable from another. This means that $\mathcal{X}^T\mathcal{X}$ may have some very small eigenvalues (because there is a vector $\mathbf{u}$ so that $\mathcal{X}\mathbf{u}$ is small; this means that $\mathbf{u}^T\mathcal{X}^T\mathcal{X}\mathbf{u}$ must be small).

These small eigenvalues lead to bad predictions. If $\mathcal{X}^T\mathcal{X}$ has a small eigenvalue, then there is some vector $\mathbf{v}$ such that $\mathcal{X}^T\mathcal{X}\mathbf{v}$ is small, or, equivalently, that the matrix can turn large vectors into small ones; but that means that $(\mathcal{X}^T\mathcal{X})^{-1}$ will turn some small vectors into big ones. In turn, this means that small errors in $\mathbf{y}$ — which are likely inevitable — will result in big errors in β. This could cause trouble in two ways. If we are looking at β to tell which explanatory variables are important, then large errors in β will be a problem. And if we are trying to predict new y values, we expect that errors in β turn into errors in prediction.

An important and useful way to suppress these errors is to regularize, using the same trick we saw in the case of classification. Instead of choosing β to minimize

$$\sum_i (y_i - \mathbf{x}_i^T\beta)^2 = (\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta)$$

we minimize

$$\sum_i (y_i - \mathbf{x}_i^T\beta)^2 + \lambda\beta^T\beta = (\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta) + \lambda\beta^T\beta$$

where $\lambda > 0$ is a constant. We choose λ in the same way we used for classification; split the training set into a training piece and a validation piece, train for different values of λ, and test the resulting regressions on the validation piece. We choose the λ that yields the smallest validation error. Notice we could use multiple splits, and average over the splits.

This helps, because to solve for β we must solve the equation

$$(\mathcal{X}^T\mathcal{X} + \lambda\mathcal{I})\beta = \mathcal{X}^T\mathbf{y}$$

(obtained by differentiating with respect to β and setting to zero) and the smallest eigenvalue of the matrix $(\mathcal{X}^T\mathcal{X} + \lambda\mathcal{I})$ will be at least λ.

L_1 Regularization and the Lasso

If we are going to use linear regression to recover a set of explanatory variables, then we want some terms in β — terms corresponding to explanatory variables that aren't relevant — to be zero. This will not happen, unless we are very lucky, with our current schemes. Usually, numerical accidents and sampling effects mean that an explanatory variable will be slightly correlated with the dependent variable. Furthermore, our regularization does not help much, because the regularizer is squared. There is no particular advantage to driving a small β to zero, because the cost of it being non-zero is tiny.

There is a very clever method to encourage β to have zeros in it. We regularize with what is known as an L_1 norm. Rather than use $\beta^T\beta$, we use $\| \beta \|_1 = \sum_j \mid \beta_j \mid$. This means we must choose *beta* to minimize

$$\sum_i (y_i - \mathbf{x}_i^T\beta)^2 + \lambda \| \beta \|_1 = (\mathbf{y} - \mathcal{X}\beta)^T(\mathbf{y} - \mathcal{X}\beta) + \lambda \| \beta \|_1 .$$

This strategy is known as the **lasso**. It encourages β to have zeros, because small non-zero entries in β are relatively expensive. There are good numerical methods for obtaining β under this scheme, which are way beyond our scope. In practice, this method is very effective when $\mathbf{x}$ is large, and it is the method to be preferred. There are also strong mathematical results suggesting that the choice of explanatory variables one comes up with is quite good.

1.2 NON-PARAMETRIC REGRESSION AND TEXTURE SYNTHESIS

Linear regression predicts y for $\mathbf{x}$ with a simple model, which is quite straightforward to estimate. This model "fills in" or interpolates values between data points. The natural alternative is to collect a really large dataset, and predict using the nearest neighbors. There are cases where this strategy isn't really practical, but if a lot of data is available, it is a very good strategy. One impressive success is in texture synthesis.

Many different kinds of user want to remove things from images or from video. Art directors might like to remove unattractive telephone wires; restorers might want to remove scratches or marks; there's a long history of government officials removing people with embarrassing politics from publicity pictures (see the fascinating pictures in **?**); and home users might wish to remove a relative they dislike from a family picture. All these users must then find something to put in place of the pixels that were removed. Ideally, a program would create regions of texture that fit in and look convincing, using either other parts of the original image, or other images.

There are other important applications for such a program. One is to produce large quantities of texture for digital artists to apply to object models. We know that good textures make models look more realistic (it's worth thinking about why this should be true). Tiling small texture images tends to work poorly, because it can be hard to obtain images that tile well. The borders have to line up properly, and even when they do, the resulting periodic structure can be annoying.

1.2.1 Synthesis by Sampling Local Models

As **?** point out, an example texture can serve as a probability model for texture synthesis (Figure 1.5). Assume for the moment that we know every pixel in the synthesized image, except one. To obtain a model for the value of that pixel, we could match a neighborhood of the pixel to the example image. Every matching neighborhood in the example image has a possible value for the pixel of interest. This collection of values is a conditional histogram for the pixel of interest. By drawing a sample uniformly and at random from this collection, we obtain the value that is consistent with the example image.

We must now take some form of neighborhood around the pixel of interest, compare it to neighborhoods in the example image, and select some of these to form a set of example values. The size and shape of this neighborhood is significant, because it codes the range over which pixels can affect one another's values directly (see Figure 1.6). Efros *et al.* use a square neighborhood, centered at the pixel of interest.

```
Choose a small square of pixels at random from the example image
Insert this square of values into the image to be synthesized
Until each location in the image to be synthesized has a value
  For each unsynthesized location on
    the boundary of the block of synthesized values
    Match the neighborhood of this location to the
      example image, ignoring unsynthesized
      locations in computing the matching score
    Choose a value for this location uniformly and at random
      from the set of values of the corresponding locations in the
      matching neighborhoods
  end
end
```

Algorithm 1.1: *Non-parametric Texture Synthesis.*

The neighborhoods we select will be similar to the image example in some sense. A good measure of similarity between two image neighborhoods can be measured by forming the **sum of squared differences** (or **ssd**) of corresponding pixel values. We assume that the missing pixel is at the center of the patch to be synthesized, which we write $\mathcal{S}$. We assume the patch is square, and adjust the indexes of the patch to run from $-n$ to n in each direction. The sum of squared differences between this patch and an image patch $\mathcal{P}$ of the same size is given by

$$\sum_{(i,j)\in\text{patch},(i,j)\neq(0,0)} (\mathcal{A}_{ij} - \mathcal{B}_{ij})^2.$$

The notation implies that because we don't know the value of the pixel to be synthesized (which is at $(0, 0)$), we don't count it in the sum of squared differences. This similarity value is small when the neighborhoods are similar, and large when they are different (it is essentially the length of the difference vector). However, this measure places the same weight on pixels close to the unknown value as it does on distant pixels. Better results are usually obtained by weighting up nearby pixels

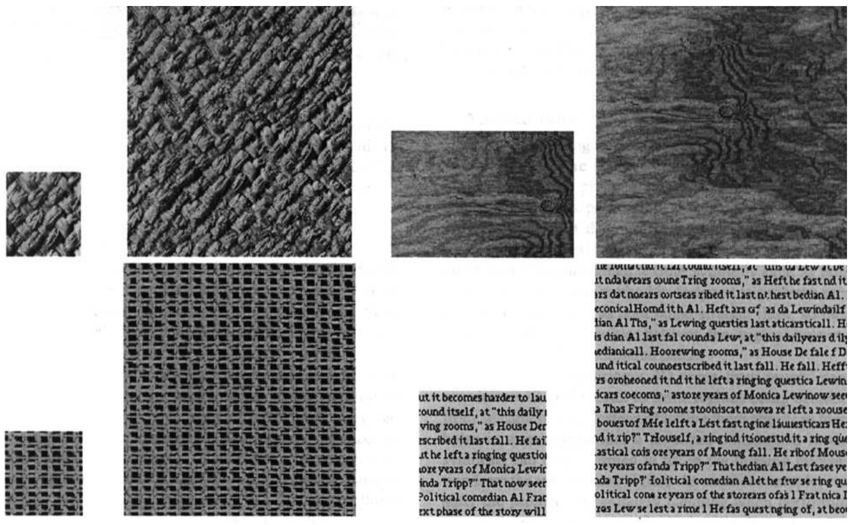

FIGURE 1.5: *? synthesize textures by matching neighborhoods of the image being synthesized to the example image, and then choosing at random amongst the possible values reported by matching neighborhoods (Algorithm 1.1). This means that the algorithm can reproduce complex spatial structures, as these examples indicate. The small block on the* **left** *is the example texture; the algorithm synthesizes the block on the* **right**. *Note that the synthesized text looks like text: it appears to be constructed of words of varying lengths that are spaced like text, and each word looks as though it is composed of letters (though this illusion fails as one looks closely).* This figure was originally published as Figure 3 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999 © IEEE, 1999.

and weighting down distant pixels. We can do so using Gaussian weights, yielding

$$\sum_{(i,j)\in\text{patch},(i,j)\neq(0,0)} (\mathcal{A}_{ij} - \mathcal{B}_{ij})^2 \exp\left(\frac{-(i^2+j^2)}{2\sigma^2}\right).$$

Now we know how to obtain the value of a single missing pixel: choose uniformly and at random amongst the values of pixels in the example image whose neighborhoods match the neighborhood of our pixel. We cannot choose those matching neighborhoods by just setting a threshold on the similarity function, because we might not have any matches. A better strategy to find matching neighborhoods is to select all whose similarity value is less than $(1+\epsilon)s_{min}$, where s_{min} is the similarity function of the closest neighborhood and ϵ is a parameter.

Generally, we need to synthesize more than just one pixel. Usually, the values of some pixels in the neighborhood of the pixel to be synthesized are not known; these pixels need to be synthesized too. One way to obtain a collection of examples for the pixel of interest is to count only the known values in computing the sum of squared differences, and scale the similarity to take into account the number of known pixels. Write $\mathcal{K}$ for the set of pixels around a point whose values are known,

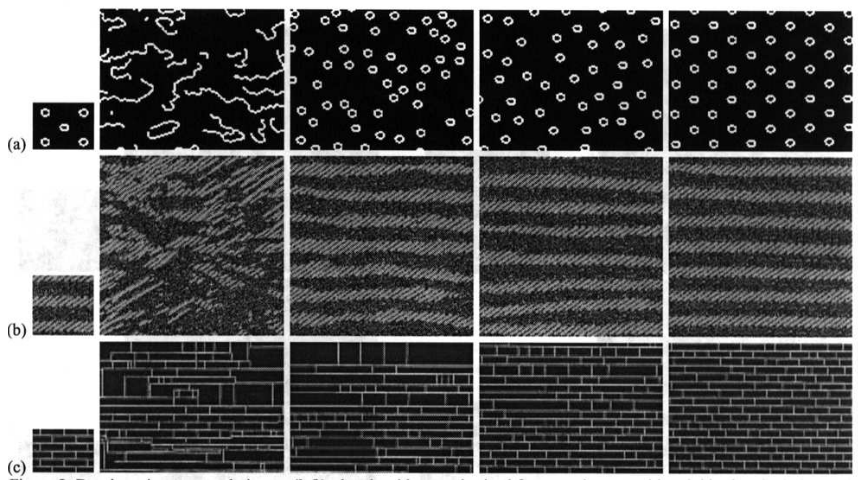

FIGURE 1.6: *The size of the image neighborhood to be matched makes a significant difference in Algorithm 1.1. In the figure, the textures at the right are synthesized from the small blocks on the* **left***, using neighborhoods that are increasingly large as one moves to the* **right***. If very small neighborhoods are matched, then the algorithm cannot capture large-scale effects easily. For example, in the case of the spotty texture, if the neighborhood is too small to capture the spot structure (and so sees only pieces of curve), the algorithm synthesizes a texture consisting of curve segments. As the neighborhood gets larger, the algorithm can capture the spot structure, but not the even spacing. With very large neighborhoods, the spacing is captured as well.* This figure was originally published as Figure 2 of "Texture Synthesis by Non-parametric Sampling," A. Efros and T.K. Leung, Proc. IEEE ICCV, 1999 © IEEE, 1999.

and $\sharp\mathcal{K}$ for the size of this set. We now have, for the similarity function,

$$\frac{1}{\sharp\mathcal{K}} \sum_{(i,j)\in\mathcal{K}} (\mathcal{A}_{ij} - \mathcal{B}_{ij})^2 \exp\left(\frac{-(i^2+j^2)}{2\sigma^2}\right).$$

The synthesis process can be started by choosing a block of pixels at random from the example image, yielding Algorithm 1.1.

Filling in Patches

Synthesizing a large texture in terms of individual pixels will be unnecessarily slow. Because textures repeat, we expect that whole blocks of pixels also should repeat. This suggests synthesizing a texture in terms of image patches, rather than just pixels. Most of the mechanics of the procedure follow those for pixels: to synthesize a texture patch at a location, we find patches likely to fit (because they have pixels that match the boundary at that location), then choose uniformly and at random from among them. However, when we place down the new patch, we must deal with the fact that some (ideally, many) of its pixels overlap with pixels that have already been synthesized. This problem is typically solved by image segmentation methods, and we defer that discussion to Chapter **??**.

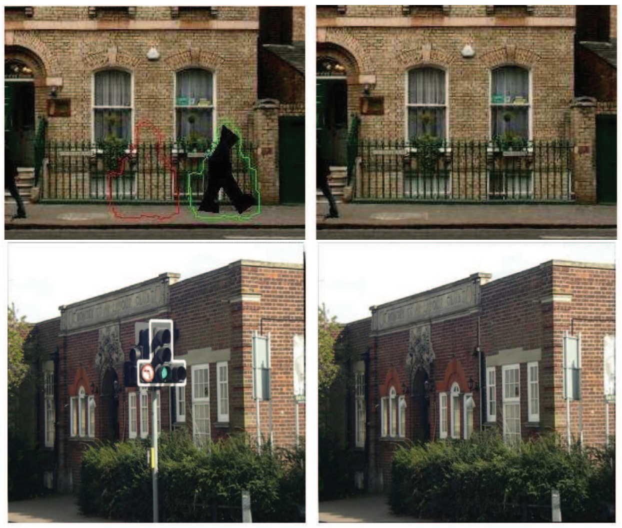

FIGURE 1.7: *If an image contains repeated structure, we have a good chance of finding examples to fill a hole by searching for patches that are compatible with its boundaries.* **Top left:** *An image with a hole in it (black pixels in a rough pedestrian shape). The pixels on the region outside the hole, but inside the boundary marked on the image, match pixels near the other curve, which represents a potentially good source of hole-filling pixels.* **Top right:** *The hole filled by placing the patch over the hole, then using a segmentation method (Chapter* **??***) to choose the right boundary between patch and image. This procedure can work for apparently unpromising images, such as the one on the* **bottom left***, an image of the facade of a house, seen at a significant slant. This slant means that distant parts of the facade are severely foreshortened. However, if we rectify the facade using methods from Section* **??***, then there are matching patches. On the* **bottom right***, the hole has been filled in using a patch from the rectified image, that is then slanted again.* This figure was originally published as Figures 3 and 6 of "Hole Filling through Photomontage," by M. Wilczkowiak, G. Brostow, B. Tordoff, and R. Cipolla, Proc. BMVC, 2005 and is reproduced by kind permission of the authors.

1.2.2 Filling in Holes in Images

There are four approaches we can use to fill a hole in an image. **Matching methods** find another image patch that looks a lot like the boundary of the hole, place that patch over the hole, and blend the patch and the image together. The patch might well be found in the image (for example, Figure 1.7). If we have a very large set of images, we could find a patch by looking for another image that matches the image with a hole in it. **?** show this strategy can be extremely successful. Blending is typically achieved using methods also used for image segmentation (Section **??**

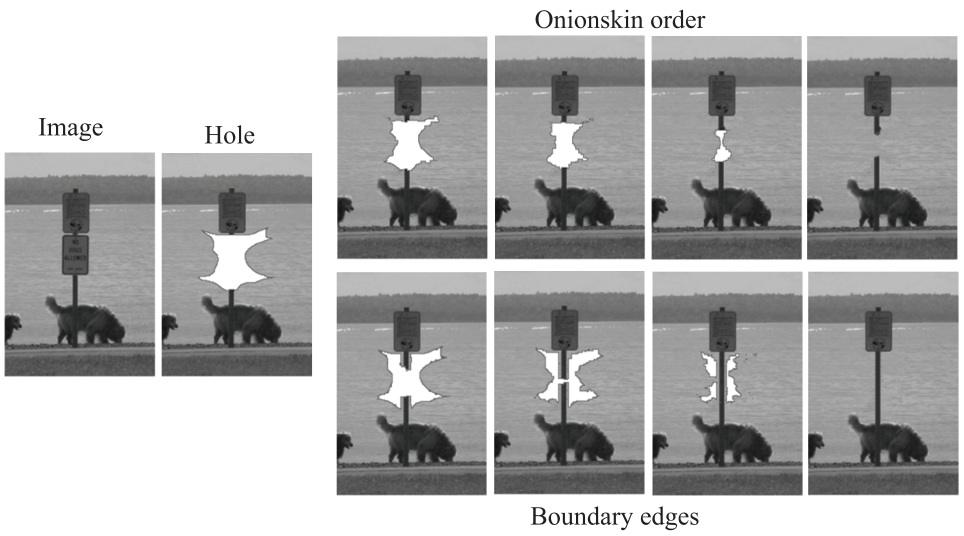

FIGURE 1.8: *Texture synthesis methods can fill in holes accurately, but the order in which pixels are synthesized is important. In this figure, we wish to remove the sign, while preserving the signpost. Generally, we want to fill in pixels where most of the neighbors are known first. This yields better matching patches. One way to do so is to fill in from the boundary. However, if we simply work our way inwards (onionskin filling), long scale image structures tend to disappear. It is better to fill in patches close to edges first.* This figure was originally published as Figure 11 of "Region Filling and Object Removal by Exemplar-Based Image Inpainting," by A. Criminisi, P. Perez, and K. Toyama, IEEE Transactions on Image Processing, 2004 © IEEE, 2004.

describes one method that can be used for blending).

As you would expect, matching methods work very well when a good match is available, and poorly otherwise. If the hole is in a region of relatively regular texture, then a good match should be easy to find. If the texture is less strongly structured, it might be hard to find a good match. In cases like this, it makes sense to try and synthesize the texture over the region of the hole, using the rest of the image as an example. Making such **texture synthesis methods** work well requires considerable care, because the order in which pixels are synthesized has a strong effect on the results. Texture synthesis tends to work better for patches when most of their neighbors are known, because the match is more constrained. As a result, one wants to synthesize patches at the boundary of the hole. It is also important to extend edges at the boundary of the hole into the interior (for example, see Figure 1.8); in practice, this means that it is important to synthesize patches at edges on the boundary before one fills in other patches. It is possible to capture both requirements in a priority function ((**?**)), which specifies where to synthesize next.

If we choose an image patch at (i, j) as an example to fill in location (u, v)

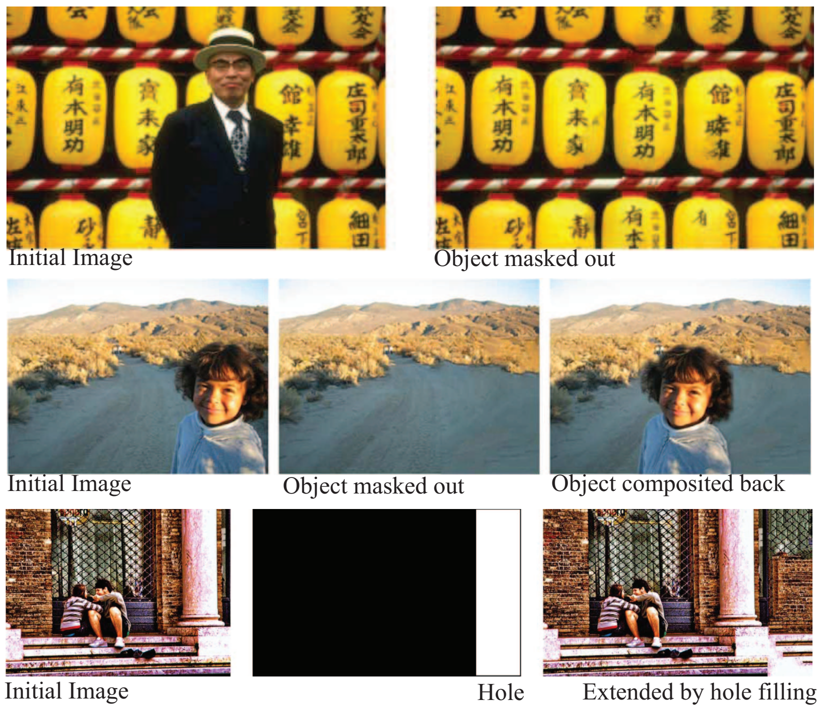

FIGURE 1.9: *Modern hole-filling methods get very good results using a combination of texture synthesis, coherence, and smoothing. Notice the complex, long-scale structure in the background texture for the example on the* **top** *row. The* **center** *row shows an example where a subject was removed from the image and replaced in a different place. Finally, the* **bottom** *row shows the use of hole-filling to resize an image. The white block in the center mask image is the "hole" (i.e., unknown pixels whose values are required to resize the image). This block is filled with a plausible texture.* This figure was originally published as Figures 9 and 15 of "A Comprehensive Framework for Image Inpainting," by A. Bugeau, M. Bertalmío, V. Caselles, and G. Sapiro, Proc. IEEE Transactions on Image Processing, 2010 © IEEE, 2010.

in the hole, then image patches near (i, j) are likely to be good for filling in points near (u, v). This observation is the core of **coherence methods**, which apply this constraint to texture synthesis. Finally, some holes in images are not really texture holes; for example, we might have a hole in a smoothly shaded region. Texture synthesis and matching methods tend to work poorly on such holes, because the intensity structure on the boundary is not that distinctive. As a result, we may find many matching patches, some of which have jarring interiors. **Variational methods** apply in these cases. Typically, we try to extend the level curves of the image into the hole in a smooth way. Modern hole-filling methods use a combination of these approaches, and can perform very well on quite demanding tasks (Figure 1.9).

1.3 MIXED STRATEGIES

Linear regression is extremely powerful, particularly for high dimensional data, because it involves a relatively simple model. However, it may be the case that the model is just too simple to get good results on a dataset. One alternative is to choose a non-parametric method, like that for texture synthesis above. But that method chose from the neighbors at random, which may not be appropriate in other applications. For example, imagine we wish to build a model of how a helicopter moves when we adjust cyclic and collective. We could collect a lot of data by flying (or, perhaps safer, simulating) the helicopter in a lot of states and applying experimental adjustments. However, the strategy we used above is not a particularly good choice. It would collect a set of states similar to the current state, then choose from them at random. This must produce some noise — we want something better.

There are two strategies: rather than using a linear regression, we could try to find more complex functions of more explanatory variables; or we could make the regression more sensitive to nearby data, and less sensitive to distant data. One strategy, that has proven useful for many datasets, is to build a linear regression for the nearby data, weighted by distance, and predict from that.