5 Regularisation

Contents

Abstract

Regularisation is designed to address the problem of overfitting. **High bias** or **underfitting** is when the form of our hypothesis maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features (e.g. using a linear regression hypothesis when clearly the data appear quadratic).

At the other extreme, **overfitting** or **high variance** is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.

This terminology is applied to both linear and logistic regression.

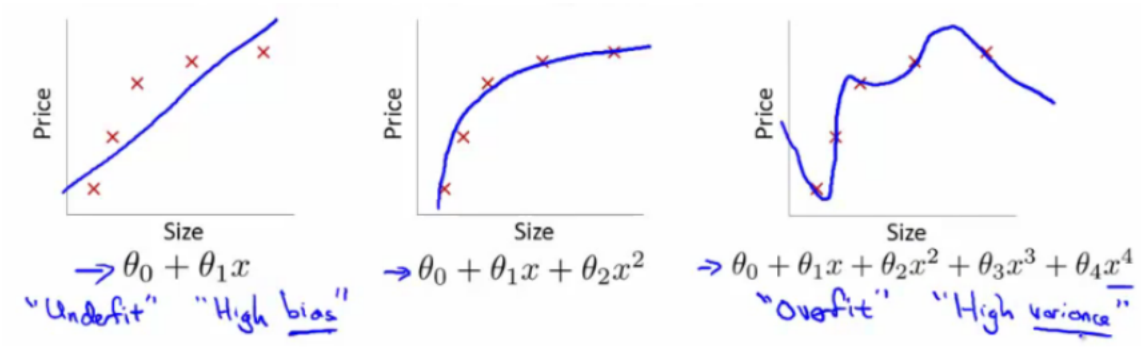


Figure 5.1: Underfitting vs. "a good fit" vs. Overfitting

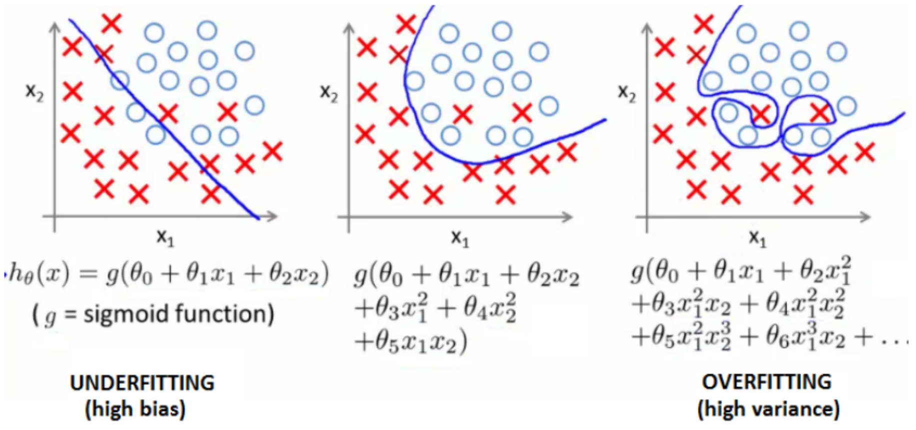


Figure 5.2: Logistic regression: Underfitting vs. "a good fit" vs. Overfitting

5.1 Addressing the issue of overfitting

There are 2 main options for addressing the issue of overfitting:

1. ***Reduce the number of features.*** This can be achieved by:

   (a) M*anually* selecting which features to keep.

(b) Using a model selection algorithm (studied later in the course).

Of course, reducing the number of features ultimately results in losing some information (we ideally select those features which minimize data loss).

2. ***Regularization***

   (a) Keep all the features, but reduce the parameters θ_j.

   (b) This works well when we have a lot of features, each of which contributes a bit to predicting y.

Regularisation works well when we have a lot of slightly useful features.

5.2 Cost function modification for regularisation

To reduce the weights (or parameters) so that the following hypothesis function is more quadratic:

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

we can eliminate the influence of $\theta_3 x^3$ and $\theta_4 x^4$, without actually getting rid of these features or changing the form of our hypothesis. We can instead modify our cost function:

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 + 1000\theta_3^2 + 1000\theta_4^2$$

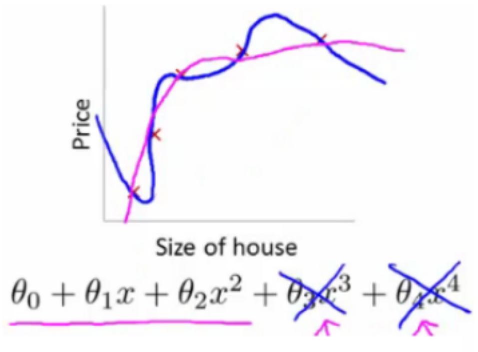


Figure 5.3: This modification of our cost function help to penalise $\theta_3 x^3$ and $\theta_4 x^4$, so here we end up with θ_3 and θ_4 being close to zero (because the constants are massive) - so we're basically left with a quadratic function.

More generally, for regularisation we can modify the cost function:

Definition 5.1. The regularised objective function is defined as

$$\min_\theta \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

By convention we don't penalize θ_0 - minimisation is from θ_1 (but in practice including θ_0 has negligible effect).

Definition 5.2. λ is the **regularisation parameter**, which determines how much the costs of our theta parameters are inflated.

Remark. λ controls a trade off between our two goals:

1. Fitting the training set well; we can smooth/make less curvy the output of our hypothesis function to reduce overfitting.

2. Keeping parameters small; WARNING: if $\lambda \gg 1$ it may smooth out the function too much (i.e. effectively removing all terms) and cause underfitting.

We look at some automatic ways to select λ later in the course.

5.3 Optimisation: Gradient descent for regularisation

Because the θ_0 term is excluded from regularisation, the algorithm is modified in the following (again using calculus):

```
Algorithm 1. Gradient Descent for regularisation
repeat until convergence {
$\theta_0 := \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}$
$\theta_j := \theta_j - \frac{\alpha}{m} \left[\sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} + \lambda\theta_j\right]$, for $j = 1, 2, ..., n$
}
for $\alpha$ some (positive) learning rate (aka. step size) and some initial guess for $\boldsymbol{\theta}$.
```

Remark. Grouping the θ_j terms, we see that

$$\theta_j = \theta_j \left(1 - \alpha\frac{\lambda}{m}\right) - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

$\frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$ is exactly the same as the original gradient descent; the term $\frac{\lambda\theta_j}{m}$ performs our regularisation. Noting that α is small and m is large, we see that typically $\left(1 - \alpha\frac{\lambda}{m}\right) < 1$, in particular $0.95 \leq \left(1 - \alpha\frac{\lambda}{m}\right) \leq 0.99$. Thus, this in effect means θ_j gets multiplied by 0.99 (reducing the value of θ_j by some amount on every update).

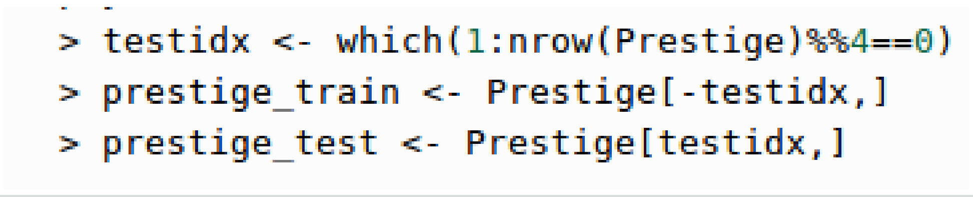
5.4 Optimisation: Normal equations for regularisation

To add in regularisation to our normal equations, we use:

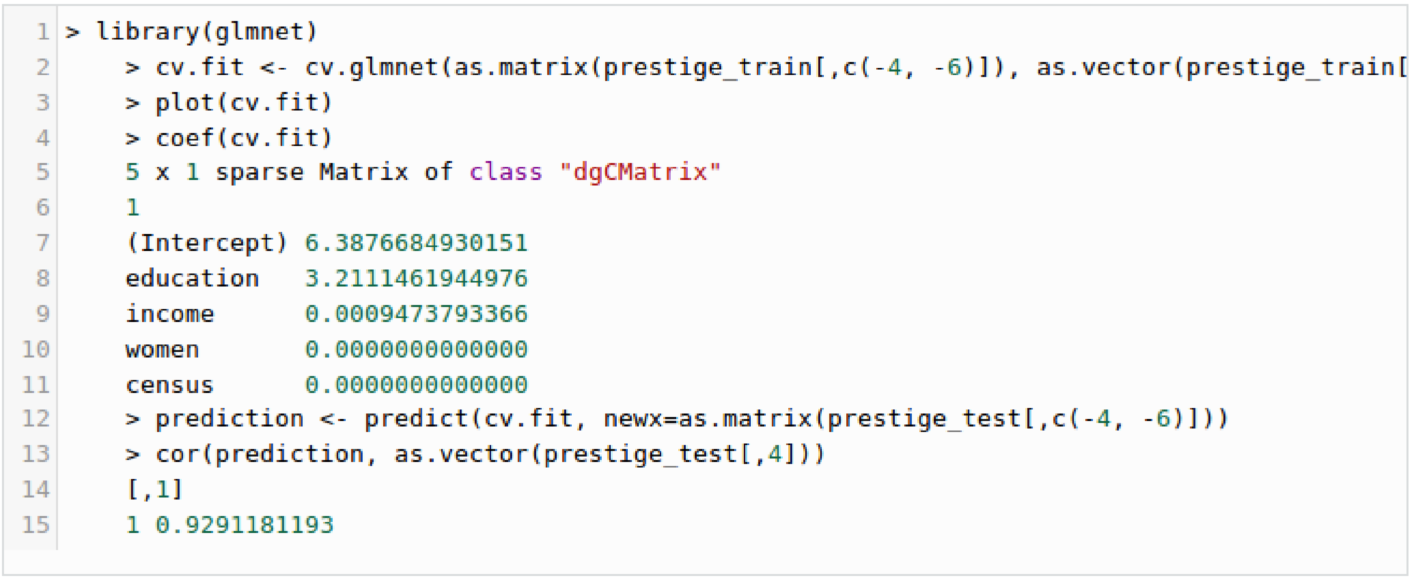
$$\boldsymbol{\theta} = \left(X^T X + \lambda L\right)^{-1} X^T \mathbf{y}, \text{ where } L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

Remark. Intuitively, L is essentially the identity matrix multiplied by a single real number λ (not including x_0).

5.5 R Example



```
> testidx <- which(1:nrow(Prestige)%%4==0)
> prestige_train <- Prestige[-testidx,]
> prestige_test <- Prestige[testidx,]
```



```
 1 > library(glmnet)
 2     > cv.fit <- cv.glmnet(as.matrix(prestige_train[,c(-4, -6)]), as.vector(prestige_train[
 3     > plot(cv.fit)
 4     > coef(cv.fit)
 5     5 x 1 sparse Matrix of class "dgCMatrix"
 6     1
 7     (Intercept) 6.3876684930151
 8     education   3.2111461944976
 9     income      0.0009473793366
10     women       0.0000000000000
11     census      0.0000000000000
12     > prediction <- predict(cv.fit, newx=as.matrix(prestige_test[,c(-4, -6)]))
13     > cor(prediction, as.vector(prestige_test[,4]))
14     [,1]
15     1 0.9291181193
```

Figure 5.4: Regularisation in **R**. Note the second line reads `> cv.fit <- cv.glmnet(as.matrix(prestige_train[,c(-4, -6)]), as.vector(prestige_train[,4]), nlambda=100, alpha=0.7, family="gaussian")`

The `glmnet` library provides a cross-validation test to automatically choose the better λ:

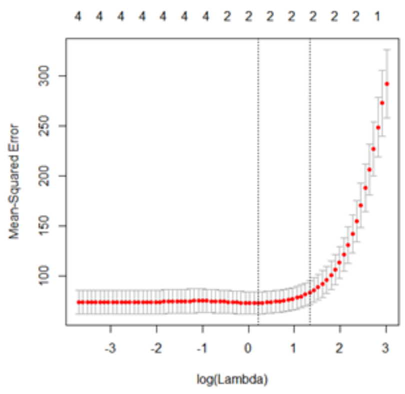


Figure 5.5: Cross-validation plot for λ. It shows the best λ with minimal-root, mean-square error.