Cross Validation, Regularization and Overfitting, Assessing model effectiveness

Jay Urbain, PhD

Credits: Hastie and Tibshirani, Andrew Ng, Stanford;

Data has inherent variance that does not have predictive value

$$
\begin{aligned}
E(Y - \hat{Y})^2 &= E[f(X) + \epsilon - \hat{f}(X)]^2 \\
&= \underbrace{[f(X) - \hat{f}(X)]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}} ,
\end{aligned}
$$

Necessitates the need for training, validation, and test sets.

- Training set – Learn model
- Validation set – tune model
- Test set – evaluate tuned model

Assessing the accuracy of regression model coefficients

Linear regression with residual term. Represents what we can't explain with our model.

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

RSS measures the amount of variability that is left unexplained after performing the regression

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

TSS (Total sum of squares) measures the total variance when measuring the response y.

$$\text{TSS} = \sum (y_i - \bar{y})^2$$

R^2 amount of variance explained by our model

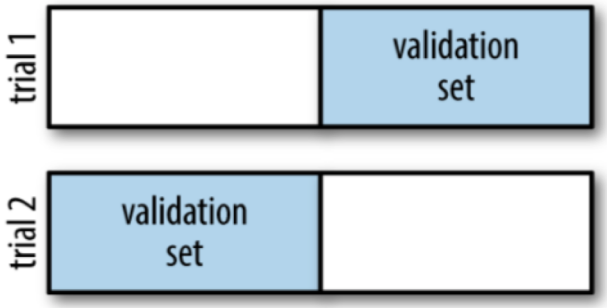
$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

The RSE is an estimate of the standard deviation of ε. It is basically the average amount that the response will deviate from the true regression line.

$$\text{RSE} = \sqrt{\frac{1}{n-2}\text{RSS}} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}.$$

Model validation via cross-validation

- One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training.
- One way to address this is to use *cross-validation*—i.e., to do a sequence of fits where each subset of the data is used both as a training set and as a validation set.

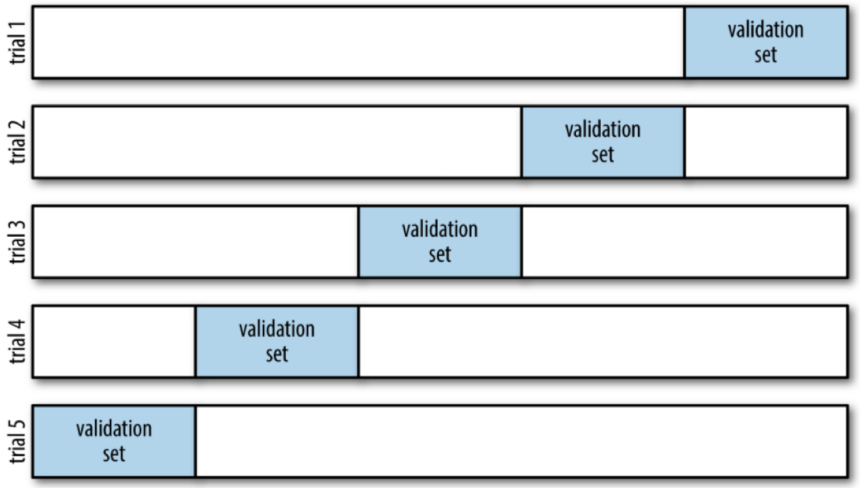



```
In[6]: y2_model = model.fit(X1, y1).predict(X2)
       y1_model = model.fit(X2, y2).predict(X1)
       accuracy_score(y1, y1_model), accuracy_score(y2, y2_model)

Out[6]: (0.9599999999999996, 0.9066666666666662)
```

Combine with average

5-Fold Cross Validation




```
from sklearn.cross_validation import LeaveOneOut
scores = cross_val_score(model, X, y, cv=LeaveOneOut(len(X)))
```

Combine with average

Selecting the best model

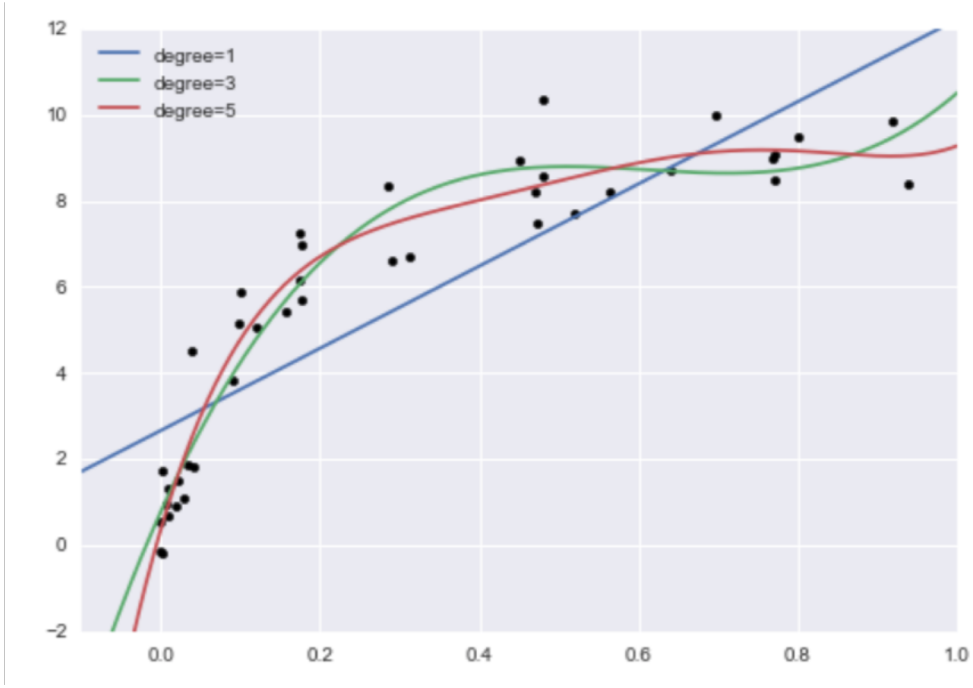
If our estimator is underperforming, how should we move forward? There are several possible answers:

- Use a more complicated/more flexible model
- Use a less complicated/less flexible model
- Gather more training samples
- Gather more data to add features to each sample

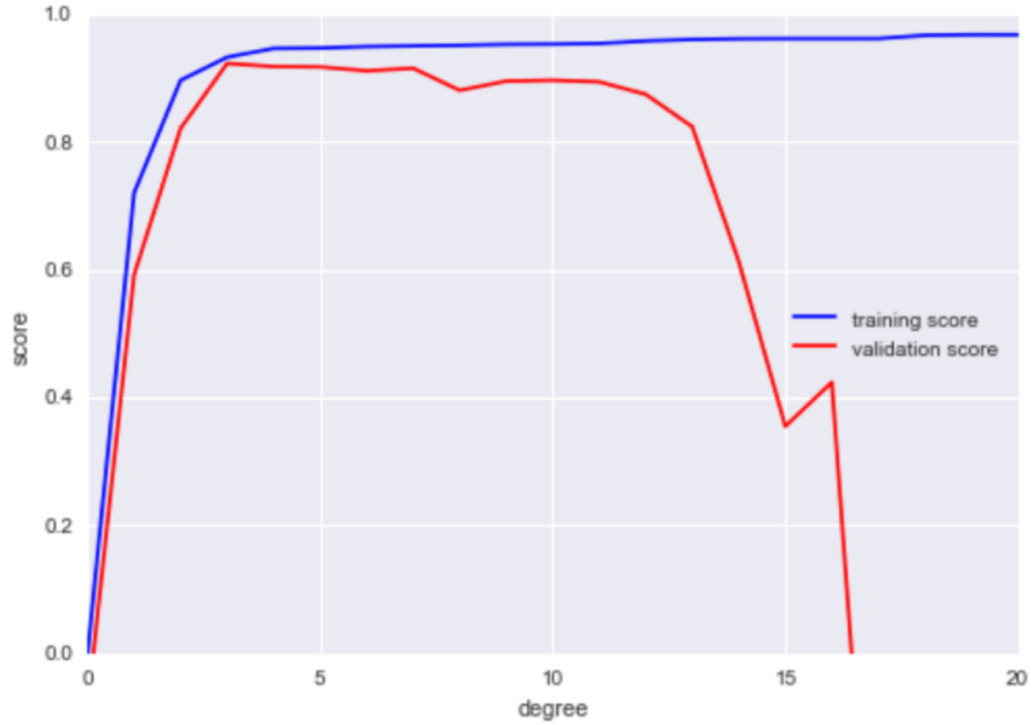
The answer to this question is often counterintuitive.

- Sometimes using a more complicated model will give worse results, and adding more training samples may not improve your results!
- The ability to determine what steps will improve your model is what separates the successful machine learning practitioners from the unsuccessful.

3 Different Polynomial Models
Which is the best model?

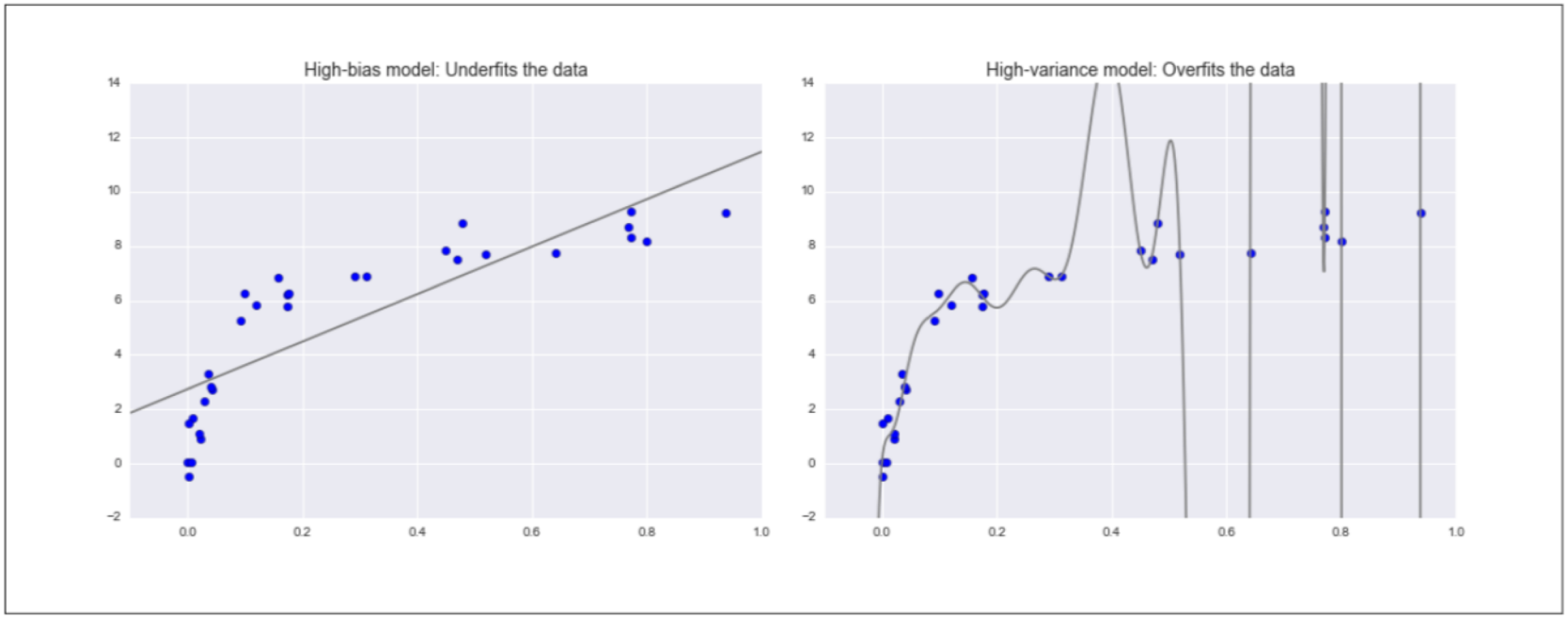


Training vs. validation Scores

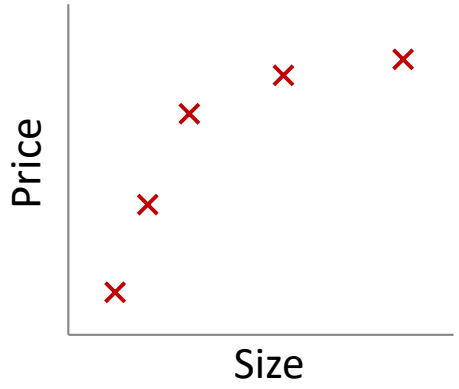


The bias–variance trade-off

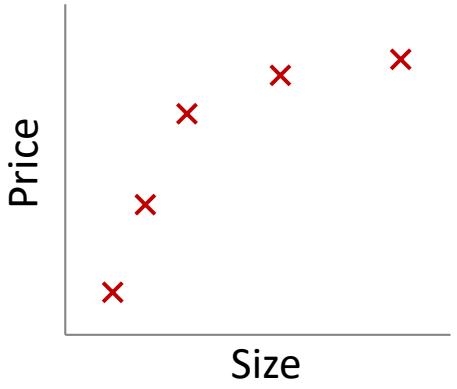
- Fundamentally, the question of “the best model” is about finding a sweet spot in the trade-off between *bias* and *variance*.



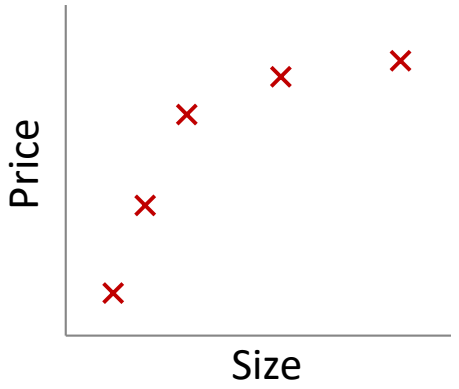
Example: Linear regression (housing prices)



$\theta_0 + \theta_1 x$



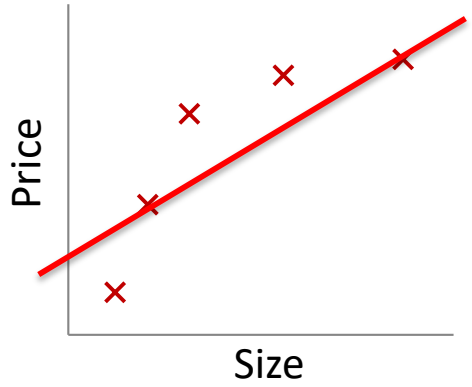
$\theta_0 + \theta_1 x + \theta_2 x^2$



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

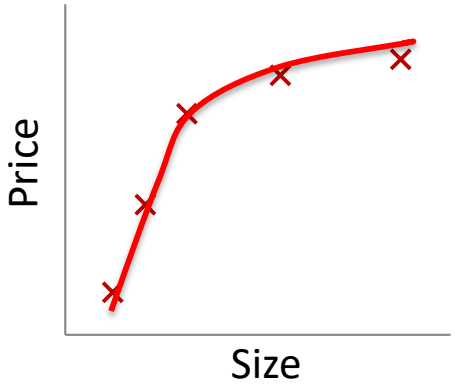
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Example: Linear regression (housing prices)

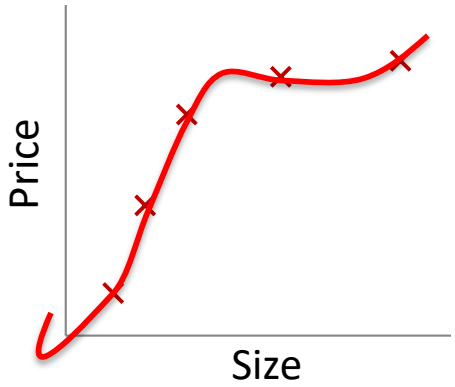


$\theta_0 + \theta_1 x$

Underfit, high bias



$\theta_0 + \theta_1 x + \theta_2 x^2$

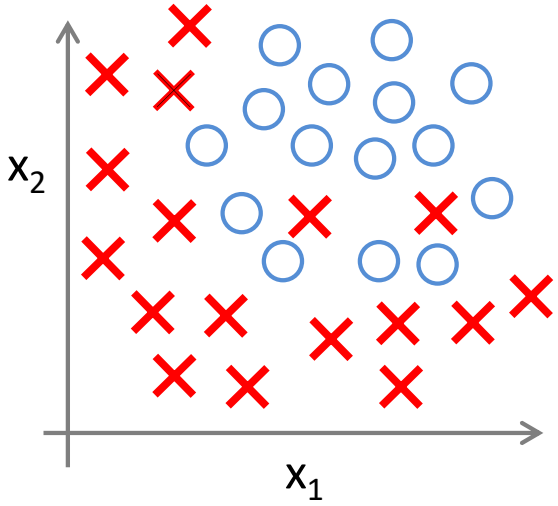


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfit, high variance

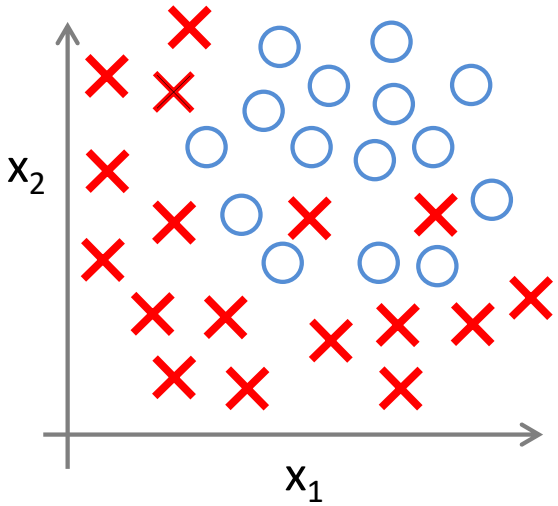
Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Example: Logistic regression

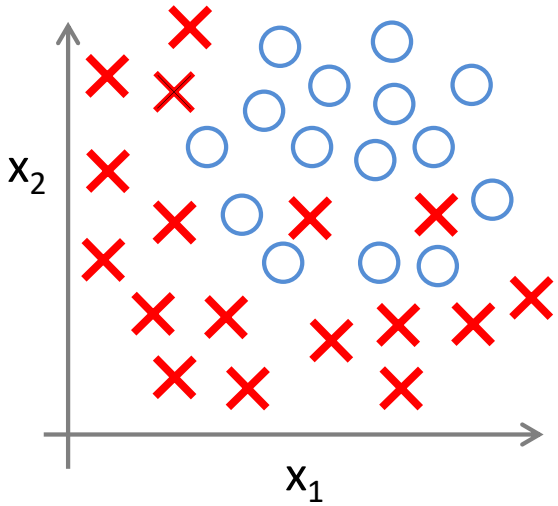


$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

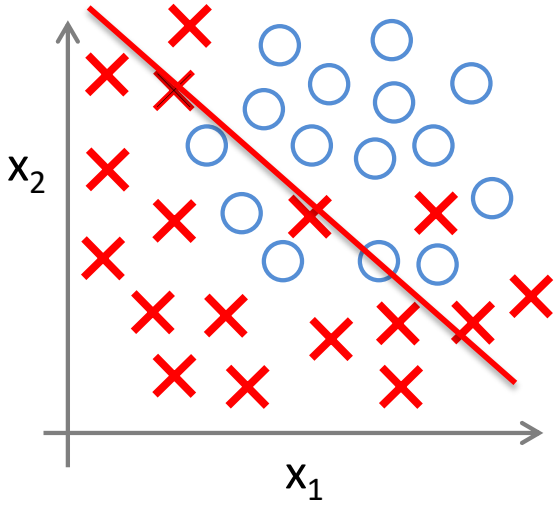


$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

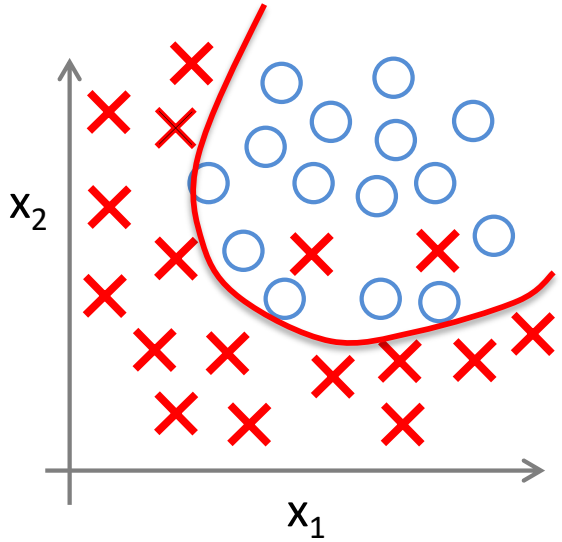
Example: Logistic regression



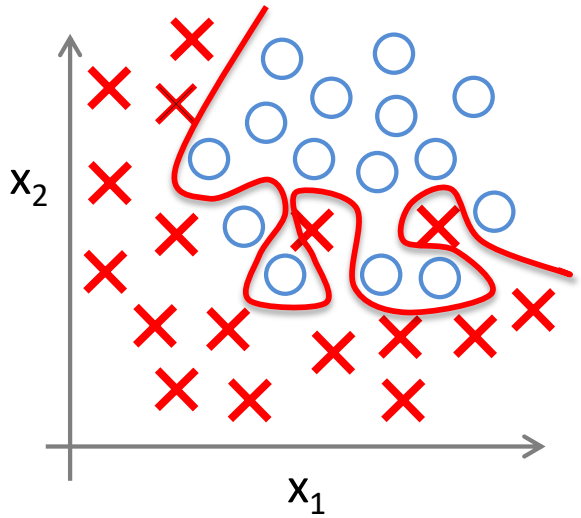
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

Underfit, high bias



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$



$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

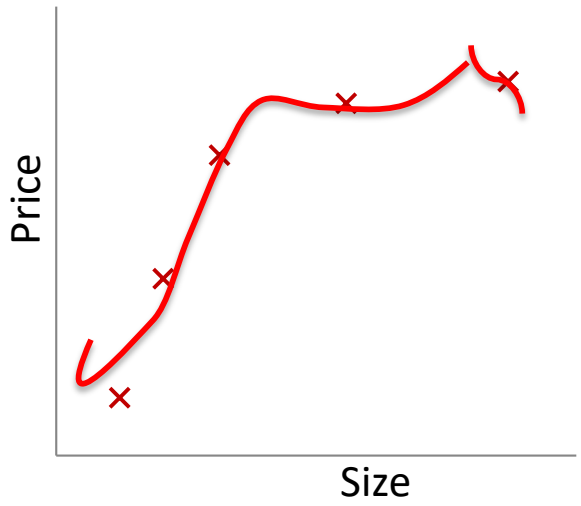
Overfit, high variance

Low model complexity ←

→ High model complexity

Addressing overfitting:

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

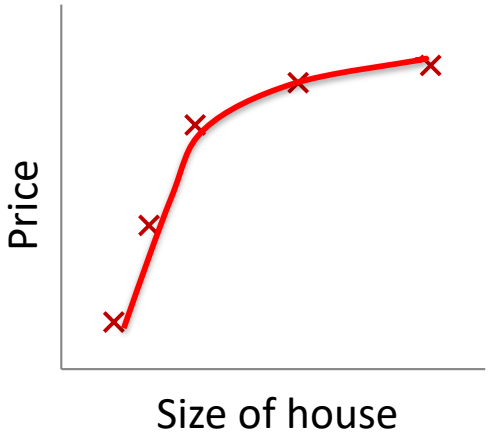


Addressing overfitting:

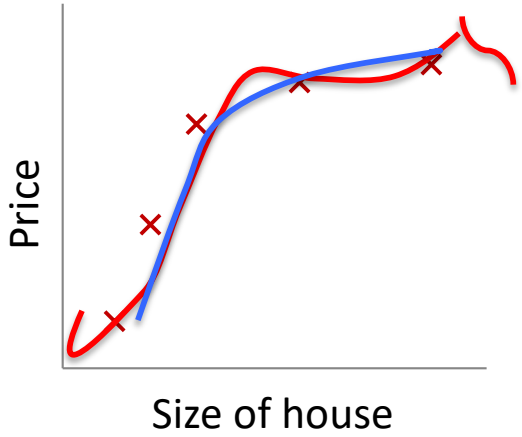
Options:

1. Reduce number of features.
   - Manually select which features to keep.
   - Model selection algorithm.
2. Regularization.
   - Keep all the features, but reduce magnitude/values of parameters θ_j.
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.

Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



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

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \quad + 1000 * \Theta_3 + 1000 * \Theta_4$$

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

Housing:

- Features: $x_1, x_2, \ldots, x_{100}$
- Parameters: $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

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

$$\min_\theta J(\theta)$$

Do not penalize bias term

Regularized linear regression.

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

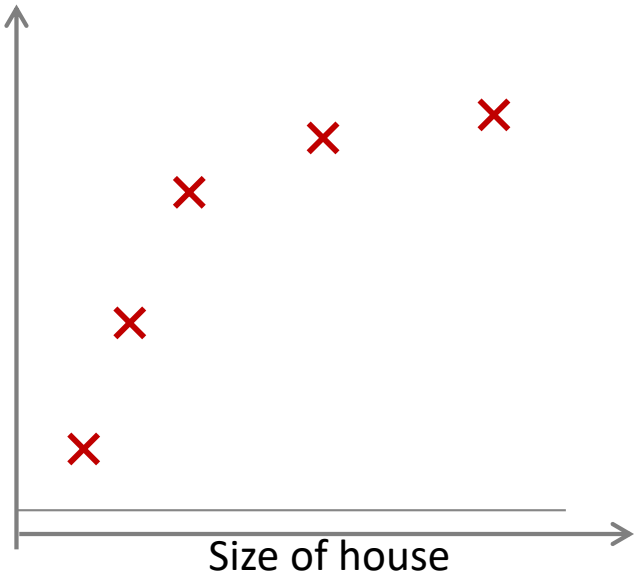
$$\min_\theta J(\theta)$$

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

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

$(1 - \alpha\frac{\lambda}{m})$ Usually < 1



In regularized linear regression, we choose θ to minimize

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

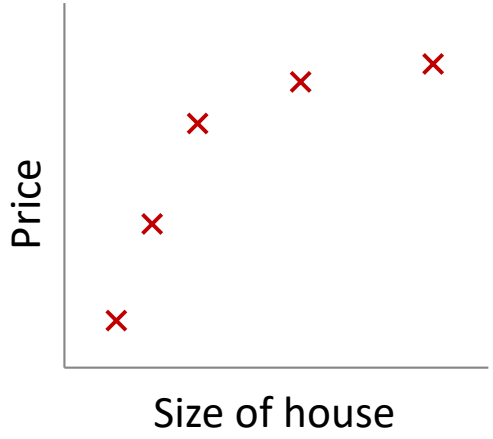
What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm fails to eliminate *overfitting*.
- Algorithm results in *underfitting*. (Fails to fit even training data well).
- Gradient descent will fail to converge.

In regularized linear regression, we choose θ to minimize

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

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



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

Gradient descent

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

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

$$(j = 0, 1, 2, 3, \ldots, n)$$

$\}$

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

Non-invertibility for Normal Equation

Suppose $m \leq n$,

(#examples) (#features)

$$\theta = (X^T X)^{-1} X^T y$$

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

L1 and L2 Regularization Methods

- A regression model that uses **L1** regularization technique is called ***Lasso Regression*** and model which uses **L2** is called ***Ridge Regression***.

Ridge regression

- **Ridge regression** adds "*squared magnitude*" of coefficient as penalty term to the loss function. Here the *highlighted* part represents **L2** regularization element.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

- if *lambda* is zero you are back to OLS.
- If *lambda* is very large then it will add too much weight and it will lead to under-fitting. So it's important how *lambda* is chosen. This technique works very well to avoid over-fitting issue.

Lasso Regression

- **Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds "*absolute value of magnitude*" of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

If *lambda* is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.

Ridge vs. Lasso Regression

- **Key difference:** Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features.

L2 loss function	L1 loss function
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions

$$P = \alpha \Sigma_{n=1}^{N} \theta_n^2 \qquad P = \alpha \Sigma_{n=1}^{N} |\theta_n|$$

No regularization (below)
L2 (upper right)
L1 (lower right)

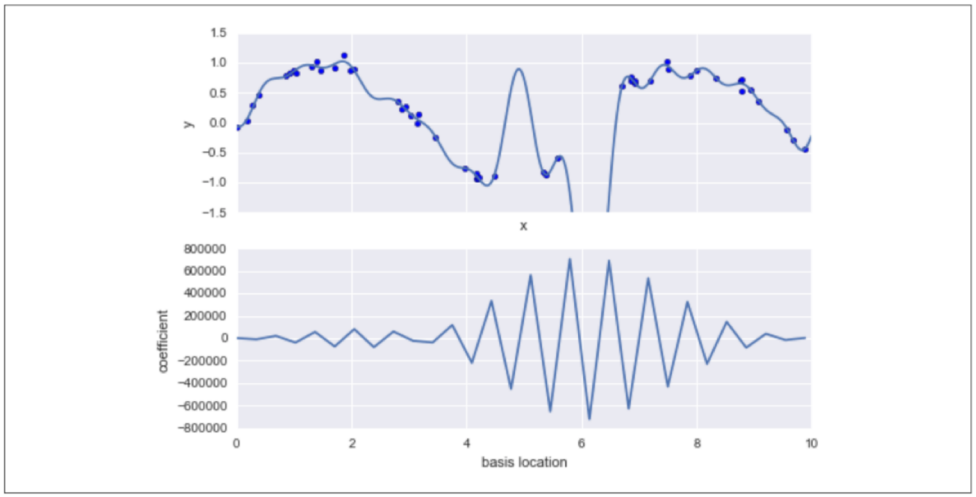


Figure 5-48. The coefficients of the Gaussian bases in the overly complex model

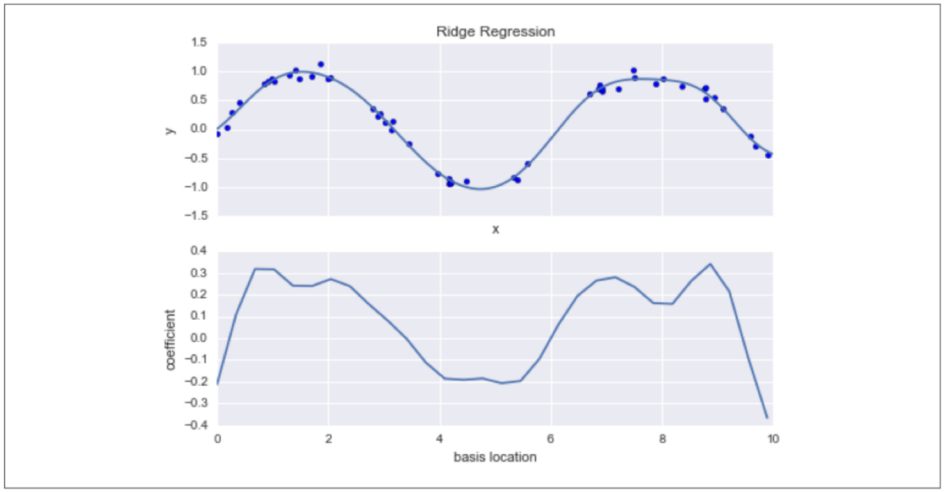


Figure 5-49. Ridge (L_2) regularization applied to the overly complex model (compare to

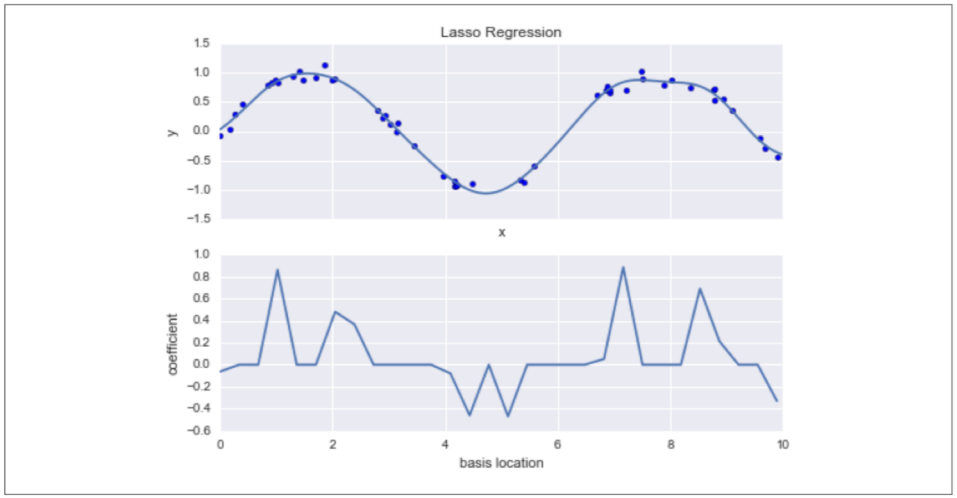
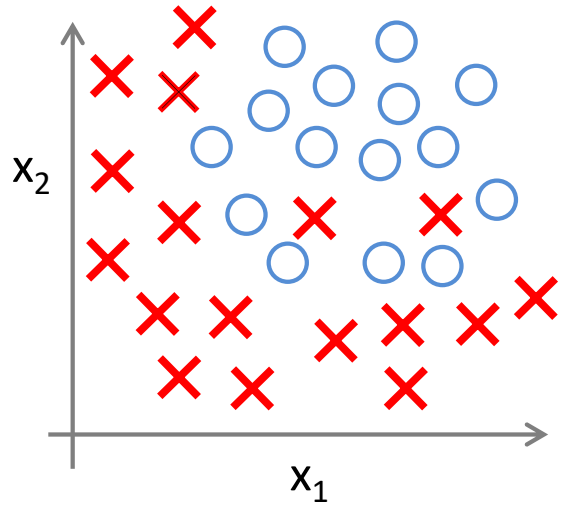


Figure 5-50. Lasso (L_1) regularization applied to the overly complex model (compare to Figure 5-48)

Regularized logistic regression.



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

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

Gradient descent

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

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

$$(j = 0, 1, 2, 3, \ldots, n)$$

$\}$

Assessing the performance of classification

		Predicted Class: Yes	Predicted Class: No
Actual Class	Yes	TP	FN
Actual Class	No	FP	TN

Accuracy measures the overall correctness of classification.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity (also called the true positive rate, or recall) measures the proportion of positives that are correctly identified as such. E.g., people who have cancer.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Specificity (also called the true negative rate) measures the proportion of negatives that are correctly identified as such. E.g., people who do not have cancer.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Precision = TP/(TP + FN)
Recall = TP/(TP + FN)
F1 = 2*Precision*Recall/(Precision + Recall