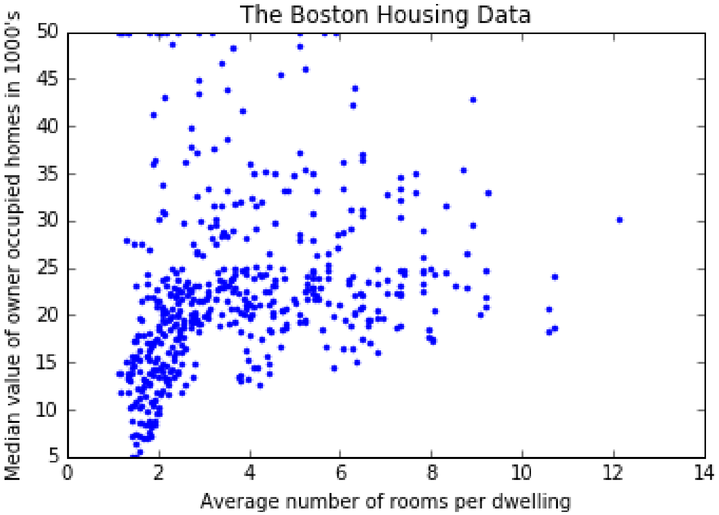
Linear Regression

Topics in Digital Modeling

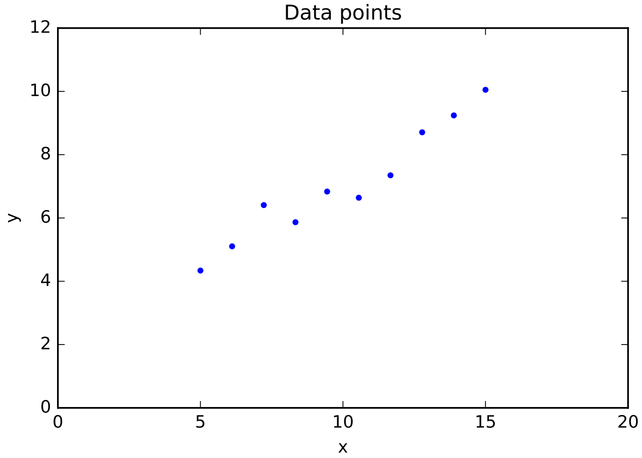
Faisal Qureshi

Regression


The Boston Housing Data
Median value of owner occupied homes in 1000's
Average number of rooms per dwelling
50
45
40
35
30
25
20
15
10
5
0
2
4
6
8
10
12
14

Regression

Consider data points $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})$. Our goal is to learn a function $f(x)$ that returns (predict) the value y given an x.



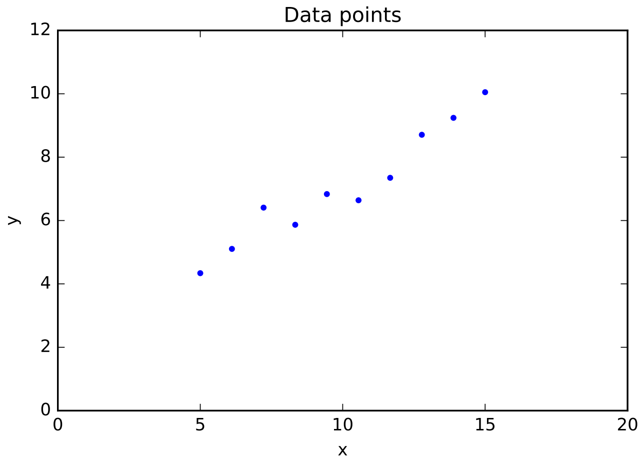
Regression

Given data $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})\}$, learn function $y = f(x)$.

- x is the input feature. In the example above, x is 1-dimensional, however, in practice x is often an M-dimensional vector.
- y is the target output. We assume that y is continuous. y is 1-dimensional (why?)

Linear regression

We assume that a linear model of the form $y = f(x) = \theta_0 + \theta_1 x$ best describe our data.



How do we determine the degree of “fit” of our model?

Least squares error

Loss-cost-objective function measures the degree of fit of a model to a given data.

A simple loss function is to sum the squared differences between the actual values $y^{(i)}$ and the predicted values $f(x^{(i)})$. This is called the *least squares error*.

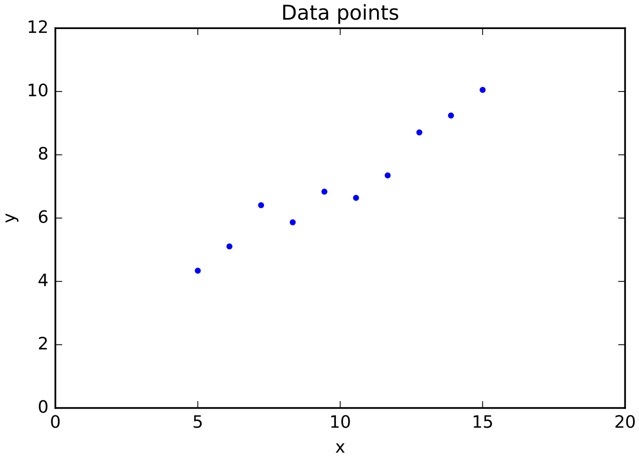
$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

Our task is to find values for θ_0 and θ_1 (model parameters) to minimize the cost.

We often refer to the predicted value as $\hat{y}$. Specifically, $\hat{y}^{(i)} = f(x^{(i)})$.

Least squares error

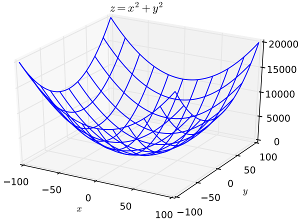
$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$



Least squares error

$$(\theta_0, \theta_1) = \underset{(\theta_0,\theta_1)}{\arg\min} \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

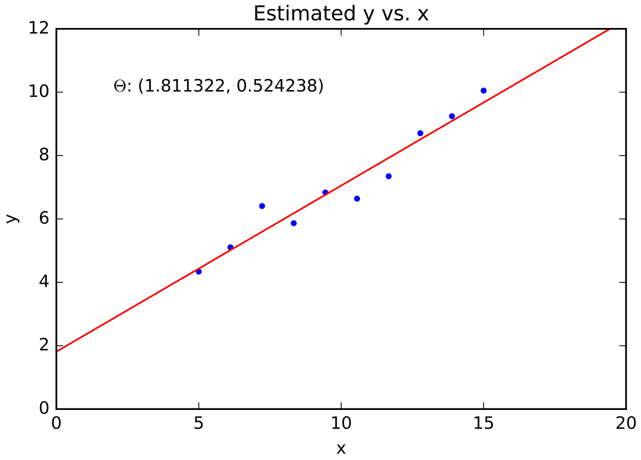
This is a convex function. We can solve for θ_0 and θ_1 by setting $\frac{\partial C}{\partial \theta_0} = 0$ and $\frac{\partial C}{\partial \theta_1} = 0$.



Least squares error

$$\theta_0 = \langle y \rangle - \theta_1 \langle x \rangle$$

$$\theta_1 = \left(\sum_{i=1}^{N} (y^{(i)} - \langle y \rangle) x^{(i)} \right) / \left(\sum_{i=1}^{N} (x^{(i)} - \langle x \rangle) x^{(i)} \right)$$



Linear least squares in higher dimensions

Input feature: $\mathbf{x}^{(i)} = \left(1, x_1^{(i)}, x_2^{(i)}, \cdots, x_M^{(i)}\right)^T$.

For this discussion, we assume $x_0^{(i)} = 1$ (just to simplify mathematical notation).

Target feature: $y^{(i)}$

Parameters: $\theta = (\theta_0, \theta_1, \cdots, \theta_M)^T \in \mathbb{R}^{(M+1)}$

Model: $f(\mathbf{x}) = \mathbf{x}^T\theta$

Linear least squares in higher dimensions

Loss:

$$C(\theta) = (\mathbf{Y} - \mathbf{X}\theta)^T(\mathbf{Y} - \mathbf{X}\theta)$$

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \mathbf{x}_2^T & - \\ & \vdots & \\ - & \mathbf{x}_N^T & - \end{bmatrix} \in \mathbb{R}^{N\times(M+1)}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^{N\times 1}$$

$\mathbf{X}$ is referred to as the *design matrix*.

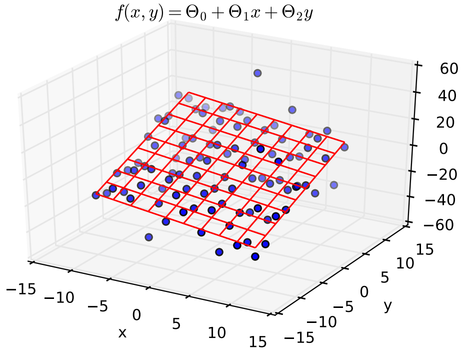
Linear least squares in higher dimensions

Loss: $C(\theta) = (\mathbf{Y} - \mathbf{X}\theta)^T(\mathbf{Y} - \mathbf{X}\theta)$

Solve θ by setting $\frac{\partial C}{\partial \theta} = 0$

Solution: $\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$

Linear least squares

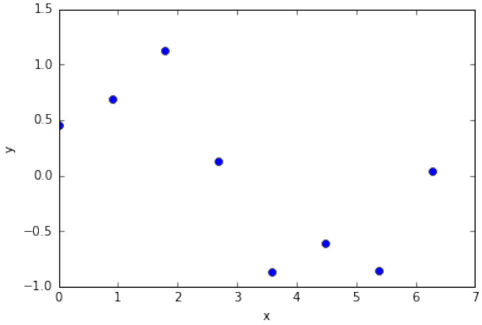


```
X = np.vstack([np.ones(x.shape), x, y]).T
Y = np.vstack([z]).T
XtX = np.dot(X.T, X)
XtY = np.dot(X.T, Y)
theta = np.dot(np.linalg.inv(XtX), XtY)
```

Exercise

Beyond lines and planes

How can we construct more complex models?



- It is possible to construct more complex models by defining input features that are some combinations of the components of $\mathbf{x}$.
- For example, in the 1D case, we can set an m order polynomial function as follows:

$$f(x) = \sum_{i=0}^{m} \theta_i x^i$$

Beyond lines and planes

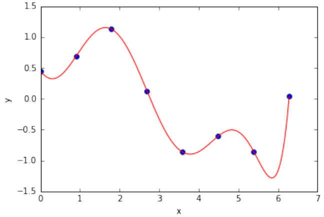
There are many ways to make linear models more powerful while retaining their nice mathematical properties:

- By using non-linear, non-adaptive basis functions, we can get generalized linear models that learn non-linear mappings from input to output but are linear in their parameters – only the linear part of the model learns. – By using kernel methods we can handle expansions of the raw data that use a huge number of non-linear, non-adaptive basis functions. simple case.
- Linear models do have fundamental limitations, and these can't be used to solve all our AI problems.

(*from R. Urtasun*)

Polynomial fitting

Note that the model is still linear in θ. We can still use the same least squares loss and use the same technique that we used for fitting a line in 1D to fit the polynomial.



How would you setup $\mathbf{X}$ *and* $\mathbf{Y}$*?*

Basis functions

The idea explored in the previous slide can be extended further. It is possible to introduce non-linearity in the system by using basis functions $\phi(.)$ as follows:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \theta$$

Example:

For a cubic polynomial (in 1D)

$\phi_0(x) = 1$
$\phi_1(x) = x$
$\phi_2(x) = x^2$
$\phi_3(x) = x^3$

Then

$$f(x) = \begin{bmatrix} \phi_0(x) & \phi_1(x) & \phi_2(x) & \phi_3(x) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Basis functions

Example continues from the previous slide

Using the basis functions, the loss can be written as

$$C(\theta) = \left(\mathbf{Y} - \Phi^T\theta\right)^T \left(\mathbf{Y} - \Phi^T\theta\right)$$

And the solution is

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{Y}$$

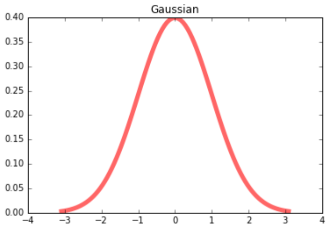
How would you set up matrix Φ?

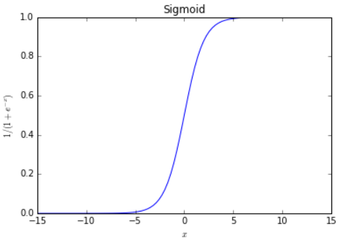
Basis functions

Example basis functions:

- Sigmoids
- Gaussians
- Polynomials (as seen before)

Similar basis functions are also used in neural networks, however, there is a key difference. Neural networks can also learn the "parameters" of the basis functions themselves. Linear regression only learns the parameters θ, i.e., the basis functions themselves are fixed.



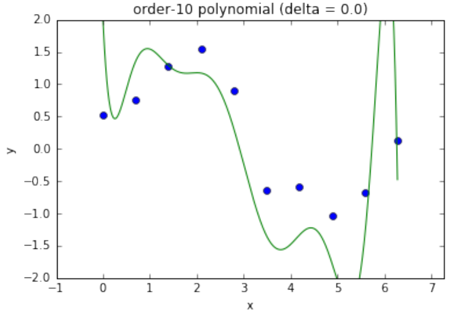


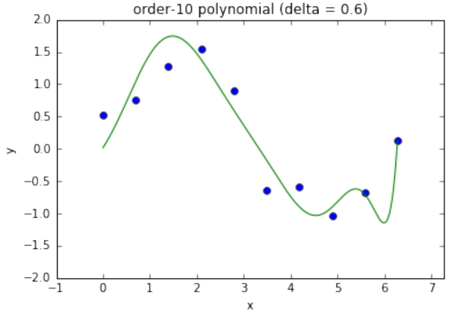
Regularization

- Increasing input features can increase model complexity
- We need an automatic way to select appropriate model complexity
- *Regularization* is the standard technique that is used to achieve this goal
- Use the following loss function that penalize squared parameters:

$$C(\theta) = (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \delta^2 \theta^T \theta$$

  - This is referred to as *ridge regression* in statistics.





Regularization

Solving for θ using

$$C(\theta) = (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \delta^2 \theta^T \theta$$

yields

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X} + \delta^2 \mathbf{I}_d)^{-1} \mathbf{X}^T \mathbf{Y}$$

So far, we have seen solutions having the following form:

$$\hat{\theta} = (\mathbf{X}^T\mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Inverting $\mathbf{X}^T\mathbf{X}$ can lead to problems, if the system of equations is ill conditioned. A solution is to add a small element to the diagonal of $\mathbf{X}^T\mathbf{X}$. Note that the above estimate (that we achieved using ridge regression is doing exactly that).

Regularization and basis function

When using basis functions, we define the loss function (for ridge regression) as follows

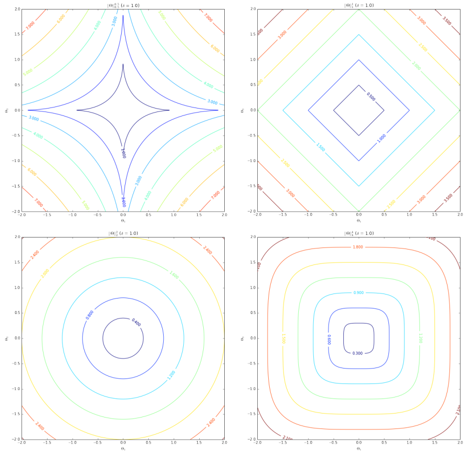
$$C(\theta) = (\mathbf{y} - \Phi\theta)^T (\mathbf{y} - \Phi\theta) + \delta^2 \theta^T \theta$$

And the solution is

$$\hat{\theta} = (\Phi^T \Phi + \delta^2 \mathbf{I}_d)^{-1} \Phi^T \mathbf{Y}$$

Other forms of regularizers

$$C(\theta) = (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) + \delta^2 \|\theta\|_q^q$$



Data whitening

If different components (dimensions) of the training data has different units (say one is measured in meters, while the other is measured in kilograms), then the squared penalty terms (that appear in our cost function) have very different weights, which can lead to erroneous solutions.

One scheme to avoid this is to “whiten the data”. Input components have:

- unit variance; and
- no covariance

$$\mathbf{X}^T_{\text{whitened}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-\frac{1}{2}}\mathbf{X}^T$$

But what if two components are perfectly correlated?

Regression

- What model should we choose?
- What may be the best way to parameterize this model?
- How do we decide if our model “fits” the data well?
- What confidence we have that our model also fits the *unseen* data, i.e., generalization.
    - This is important for prediction.

Fit error

- In general it is not possible (nor desirable, and more on this later) for a model to fit the data exactly.
- A model may not fit the data due to following reasons:
  - Imperfect data (noise present in the data)
  - Mislabeling (target errors)
  - Hidden attributes that may affect the target values, and which are not available to us during model fitting
  - Model may be too “simple”

How do we decide how well our model will fit the *unseen* data?

- Divide available data (input data + target values) into *training* and *testing* sets
- Only *training set* is available during the model fitting phase
- Evaluate the trained model (*hypothesis*) on the test set

Cross-Validation

1. Given training data $(x_{\text{train}}, y_{\text{train}})$, pick a value for δ^2, compute estimate $\hat{\theta}$
2. Compute predictions for training set $\hat{y}_{\text{train}}$.
3. Compute predictions for test set $\hat{y}_{\text{test}}$.

	Train error	Test error	Max	Min-Max (Case 1)	Average (Case 2)
0.1	100	2	100		
1	10	11	11	11	10.5
10	1	19	19		10
50	20	0	20		10
100	100	1000	1000		

(from Nando de Freitas)

Cross-Validation

Case 1

δ^2 selection via Min-Max is accounting for the worst-case scenario. This is appropropriate if, say, you are designing a safety critical system.

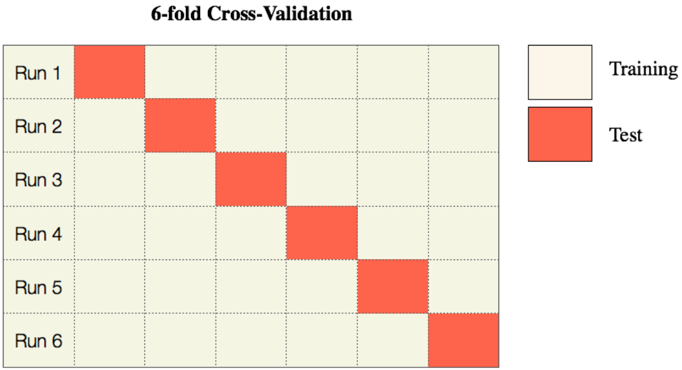
Case 2

δ^2 selection via picking the best average case is useful in cases when you want your system to work well on average, with the caveat that in some cases the system might fail miserably.

K-fold cross-validation

- Split the training data into K folds
- For each fold $k \in 1, \cdots, K$
  - Train the model on every fold **except** k
  - Test the model on fold k
  - Repeat in a round-robin fashion

Often K is set between 5 to 10



Leave-one-out Cross Validation (LOOCV)

- Set K equal to N, the number of data items.
- Train model on all data items except i.
- Use the left-out data item for test, and repeat in a round-robin fashion

Bias vs. Variance

- High bias leads to *underfitting*
  - The model has failed to capture the relevant features in the data. Perhaps the model is too simple!?
- High variance leads to *overfitting*
  - The model has latched on to the irrelevant features (say, noise) in the data.
  - Such models to not generalize well beyond the training data.

This is one of the reasons why we rely upon cross-validation to get a sense of how our model will perform on *previously unseen* data. This also suggests that unlike optimization where the sole purpose is to minimize the error, in training sometimes we accept larger training errors to achieve better generalization.

Summary

- 1-D linear regression is a useful case-study that illustrates many of issues that arise in regression in higher dimensions and in more complex models
- Model selection
  - Simple models are unable to capture all important variations in the data
  - Complex models overfit. Consequently, these do not generalize well.
- The quality of fit
  - Check whether or not the model generalizes, i.e., how does it perform on the test data that was not available to it during the training phase
- Minimizing loss (*optimization*)
  - Gradient descent (*to be discussed later*)
    - Batch update
    - Online or stochastic updates
  - Use analytical approaches when available

Summary

- More data can improve performance only if the model is of sufficient complexity