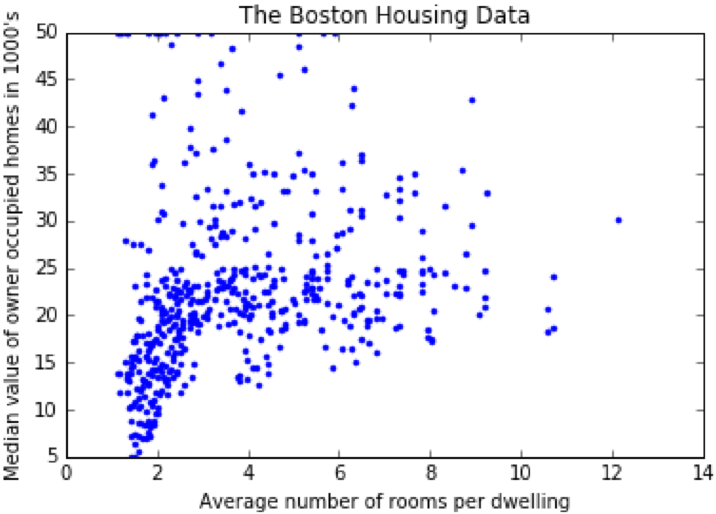
Linear regression

Machine Learning (CSCI 5770G)

Faisal Z. Qureshi

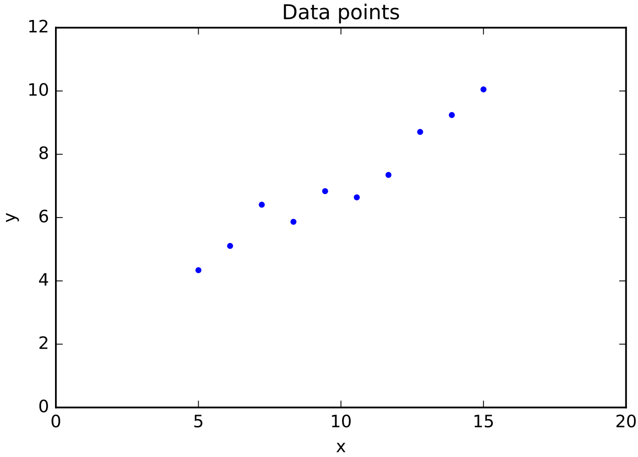
http://vclab.science.ontariotechu.ca

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.



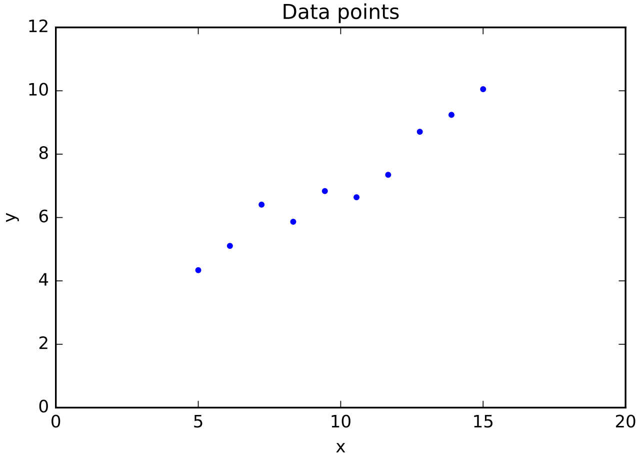
Regression

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*.

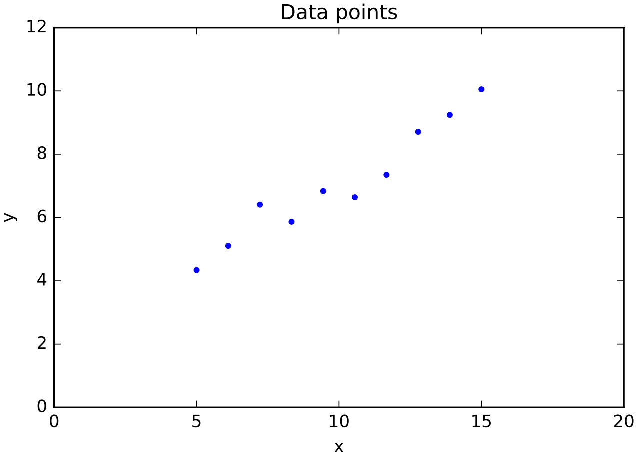
$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

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

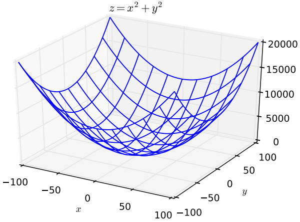
$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$



Model fitting

$$
\begin{aligned}
(\theta_0, \theta_1) &= \arg\min_{(\theta_0,\theta_1)} C(\theta_0, \theta_1) \\
&= \arg\min_{(\theta_0,\theta_1)} \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2 \\
&= \arg\min_{(\theta_0,\theta_1)} \sum_{i=1}^{N} \left(y^{(i)} - (\theta_0 + \theta_1 x^{(i)}) \right)^2
\end{aligned}
$$

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



Solving for θ_1 and θ_0

From calculus

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

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

Setting these to 0, we get

$$\sum_{i=1}^{N} 2x^{(i)}\left(y^{(i)} - \theta_0 - \theta_1 x^{(i)}\right) = 0 \tag{1}$$

$$\sum_{i=1}^{N} 2\left(y^{(i)} - \theta_0 - \theta_1 x^{(i)}\right) = 0 \tag{2}$$

Solving for θ_1 and θ_0

Let

$$\langle y \rangle = \frac{1}{N} \sum_{i=1}^{N} y^{(i)}$$

$$\langle x \rangle = \frac{1}{N} \sum_{i=1}^{N} x^{(i)}$$

Re-writing Eq. 2

$$2 \sum_{i=1}^{N} y^{(i)} - 2N\theta_0 - 2\theta_1 \sum_{i=1}^{N} x^{(i)} = 0$$

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

Solving for θ_1 and θ_0

Re-writing Eq. 1

$$\begin{aligned}
\sum_{i=1}^{N} x^{(i)} y^{(i)} - \theta_0 \sum_{i=1}^{N} x^{(i)} - \theta_1 \sum_{i=1}^{N} {x^{(i)}}^2 &= 0 \\
\Rightarrow \frac{1}{N} \sum_{i=1}^{N} x^{(i)} y^{(i)} - \frac{\theta_0}{N} \sum_{i=1}^{N} x^{(i)} - \frac{\theta_1}{N} \sum_{i=1}^{N} {x^{(i)}}^2 &= 0
\end{aligned}$$

yields

$$\langle xy \rangle - \theta_1 \langle x^2 \rangle = \theta_0 \langle x \rangle$$

where

$$\langle xy \rangle = \frac{1}{N} \sum_{i=1}^{N} x^{(i)} y^{(i)}$$

$$\langle x^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} {x^{(i)}}^2 .$$

Solving for θ_1 and θ_0

Substitute the value of θ_0

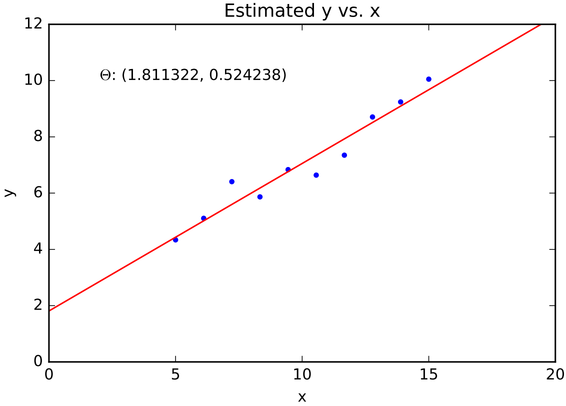
$$
\begin{aligned}
&\langle xy \rangle - \theta_1 \langle x^2 \rangle = \langle x \rangle \langle y \rangle - \theta_1 \langle x \rangle^2 \\
\Rightarrow &\theta_1 \left(\langle x \rangle^2 - \langle x^2 \rangle \right) = \langle x \rangle \langle y \rangle - \langle xy \rangle \\
\Rightarrow &\theta_1 = \frac{\langle x \rangle \langle y \rangle - \langle xy \rangle}{\langle x \rangle^2 - \langle x^2 \rangle}
\end{aligned}
$$

Use this value of θ_1 to solve for θ_0.

Solving for θ_1 and θ_0

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

$$\theta_1 = \frac{\langle x \rangle \langle y \rangle - \langle xy \rangle}{\langle x \rangle^2 - \langle x^2 \rangle}$$



Linear least squares in higher dimensions

Input features

Assume $x_0^{(i)} = 1$ (bias)

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_M^{(i)} \end{bmatrix}$$

Targets

$$y^{(i)}$$

Model

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

where model parameters are

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_M \end{bmatrix}$$

Linear least squares in higher dimensions

Design matrix

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

Targets vector

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

Loss

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

Model fitting (solving for θ)

As before solve θ by setting $\frac{\partial C}{\partial \theta} = 0$

$$\begin{aligned} C(\theta) &= (\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta) \\ &= \left(\mathbf{y}^T - (\mathbf{X}\theta)^T\right)(\mathbf{y} - \mathbf{X}\theta) \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\theta - (\mathbf{X}\theta)^T\mathbf{y} + (\mathbf{X}\theta)^T\mathbf{X}\theta \\ &= \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\theta)^T\mathbf{y} + (\mathbf{X}\theta)^T\mathbf{X}\theta \end{aligned}$$

We know that

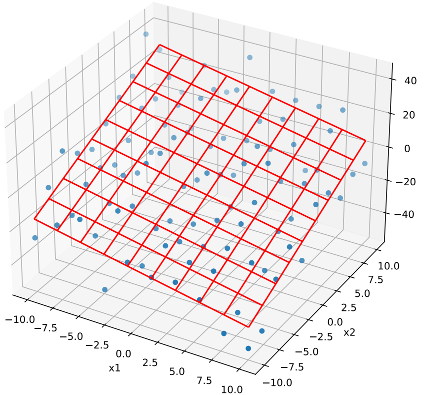
$$\frac{\partial C}{\partial \theta} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\theta$$

Setting $\frac{\partial C}{\partial \theta} = 0$, we get solution

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

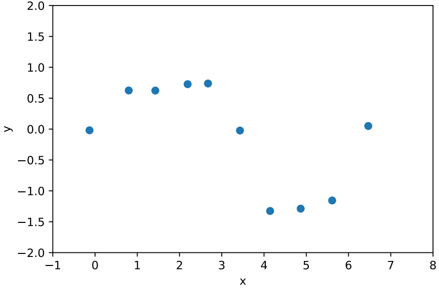
Linear least squares in 2D

Plane fitting



Beyond lines and planes

- Sometimes we need more complex models



- How do we construct more complex models?
  - Suitable complexity
  - Tractable
  - Expressive
  - Compute efficient

Beyond lines and planes

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

- Use non-linear, non-adaptive basis functions to construct *generalized* linear models that learn non-linear mappings from inputs to outputs
- Use kernel methods that expands raw data using a large number of non-linear, non-adaptive basis functions
- These models are still linear in their parameters
- Only linear part of the model learns

Polynomial fitting

- Construct more complicated linear models by defining input features that are some combination of the components of $\mathbf{x}$

Degree 3 polynomial

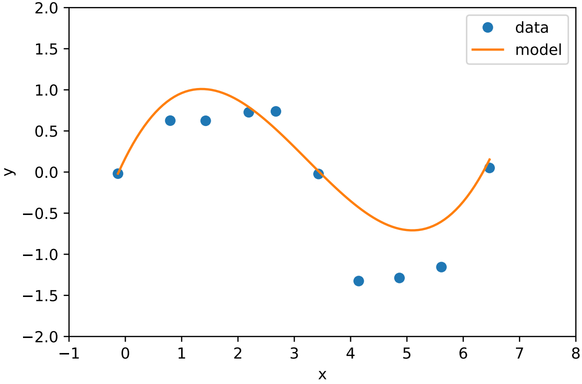
Set input feature as:

$$\mathbf{x}^{(i)} = \left(1, x^{(i)}, {x^{(i)}}^2, {x^{(i)}}^3\right)$$

Then we can write the model as

$$f(\mathbf{x}^{(i)}) = \begin{bmatrix} 1 & x^{(i)} & {x^{(i)}}^2 & {x^{(i)}}^3 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Polynomial fitting



- The model is still linear in θ
- We can still use the same least squares loss and the same technique that we used for fitting a line

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:

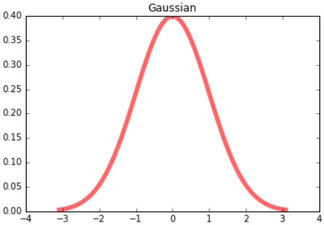
$$f(\mathbf{x}^{(i)}) = \phi(\mathbf{x}^{(i)})^T \theta$$

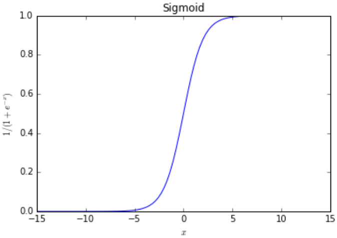
- $\phi(.)$ is a vector-valued function
- These models are sometimes referred to as linear basis function models

Basis functions

Example basis functions:

- Polynomials
- Gaussians
- Sigmoids





Example: polynomial basis functions linear models

Using basis functions to setup a cubic polynomial in 1D

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

Then we can write the model as follows:

$$f(x) = \left[\begin{array}{cccc} \phi_0(x) & \phi_1(x) & \phi_2(x) & \phi_3(x) \end{array} \right] \left[\begin{array}{c} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{array} \right]$$

Example: Gaussian basis functions in linear models

Gaussian basis function is

$$\phi_i(\mathbf{x}) = \exp\left(-\gamma_i \|\mu_i - \mathbf{x}\|_2^2\right)$$

We can use it to setup Φ

$$\Phi = \begin{bmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \cdots & \phi_M(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \cdots & \phi_M(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \cdots & \phi_M(\mathbf{x}^{(1)}) \end{bmatrix}$$

where

$$\phi_0(.) = 1$$

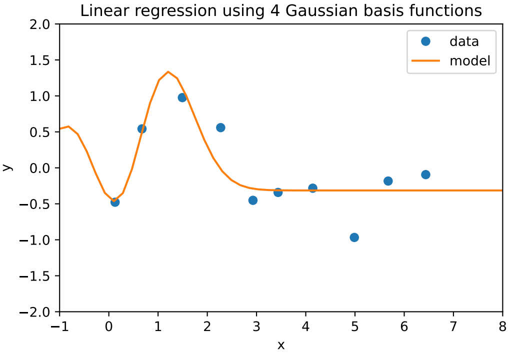
Basis functions

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

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



Linear models and basis functions

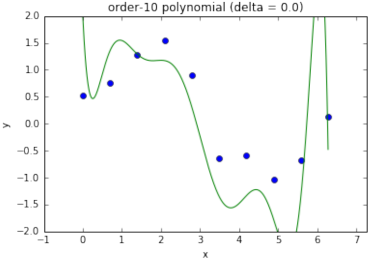
- Linear models are fundamentally limited in terms of the kind of problems they can solve.
  - These cannot be used to solve all of our AI problems
- Neural networks, which are non-linear, also use basis functions; however, there is an important difference
  - Neural networks can also learn the "parameters" of the basis functions themselves.
  - Linear regression only learns the parameters θ, i.e., the basis functions themselves are fixed.

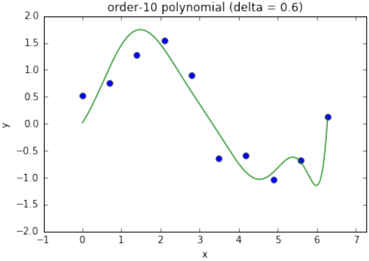
Regularization

- 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
  - Eg., 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

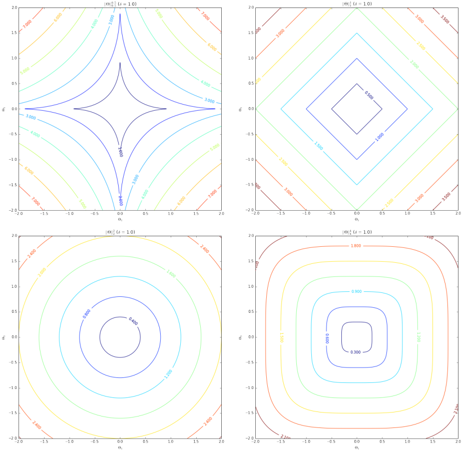
$$C(\theta) = (\mathbf{y} - \Phi\theta)^T (\mathbf{y} - \Phi\theta) + \delta^2 \theta^T \theta$$

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

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

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

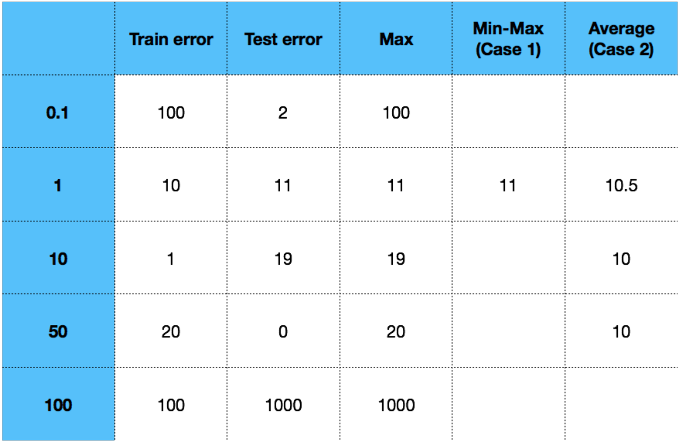
- In general it is not possible (nor desirable, and more on this later) for a model to fit the data exactly.
- A model may not fit the data due to following reasons:
  - Imperfect data (noise present in the data)
  - Mislabeling (target errors)
  - Hidden attributes that may affect the target values, 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

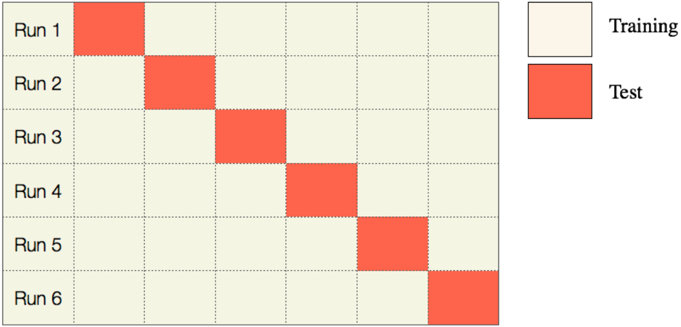
δ^2 selection via picking the best average case is useful in cases when you want your system to work well on average, with the caveat that in some cases the system might fail miserably.

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

6-fold Cross-Validation



Run 1	🟥	⬜	⬜	⬜	⬜	⬜
Run 2	⬜	🟥	⬜	⬜	⬜	⬜
Run 3	⬜	⬜	🟥	⬜	⬜	⬜
Run 4	⬜	⬜	⬜	🟥	⬜	⬜
Run 5	⬜	⬜	⬜	⬜	🟥	⬜
Run 6	⬜	⬜	⬜	⬜	⬜	🟥

⬜ Training

🟥 Test

Leave-one-out Cross Validation (LOOCV)

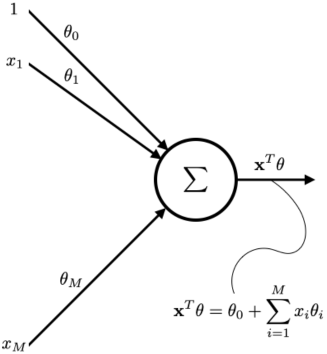
- Set K equal to N, the number of data items.
- Train model on all data items except i.
- Use the left-out data item for test, and repeat in a round-robin fashion

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.

Network view of linear regression

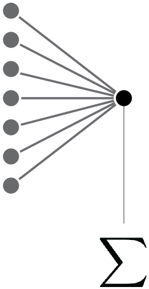


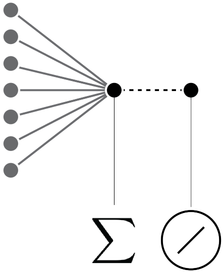
Graphical representation of linear regression

Both figures model the following function.

$$f(\mathbf{x}) = \sum_j \mathbf{x}_j \theta_j$$

The figure on the right also shows an identify $a(x) = x$ activation function. Each connection denotes a parameter/weight that is multiplied to the corresponding input value. Gray circles denote input, and black circles denote sum operation





What does it remind us of?

Key Takeaway 1

- Model
    - Parameters, and optionally hyper-parameters
- Mechanism to determine model fitness
    - Loss
- Model fitting
    - Searching model parameters that results in the "best" model fitness or the "lowest" loss
    - Even for simple problems, it is not always possible to fit the model using an analytical solution
    - For neural networks, we often use some variant of gradient descent
        - The ability to compute gradient of loss w.r.t. model paramters (and inputs)

Key Takeaway 2

- It is possible to use basis functions to develop models capable of dealing with non-linearities present in the data
- Neural networks can be seen as using basis functions as well; however, there is a key difference, neural networks can also learn the "parameters" of the basis functions themselves.
  - Linear regression models discussed above only learns the parameters θ, i.e., the basis functions themselves are fixed.

Key Takeaway 3

- Regularization is necessary to reduce generalization error without effecting training error.
- Without regularization a complex model will most likely overfit training data, leading to poor performance on test data.
  - Extra constraints and penalties
  - Prior knowledge

What about neural networks? How do we deal with model complexity in neural networks?

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

Summary

- Minimizing loss (*optimization*)
  - Gradient descent (*to be discussed later*)
    - Batch update
    - Online or stochastic updates
  - Use analytical approaches when available
- More data can improve performance only if the model is of sufficient complexity

Copyright and License