CSC 411: Lecture 02: Linear Regression

Raquel Urtasun & Rich Zemel

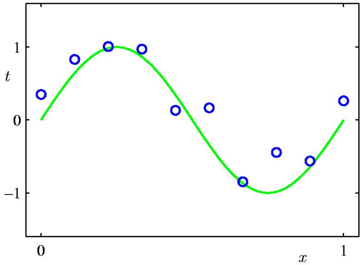
University of Toronto

Sep 16, 2015

Today

- Linear regression problem
  - continuous outputs
  - simple model
- Introduce key concepts:
  - loss functions
  - generalization
  - optimization
  - model complexity
  - regularization

Simple 1-D regression

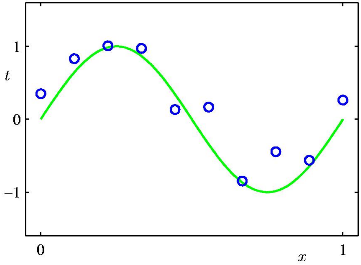


- Circles are data points (i.e., training examples) that are given to us
- The data points are uniform in x, but may be displaced in y

$$t(x) = f(x) + \epsilon$$

  with ϵ some noise
- In green is the "true" curve that we don't know
- Goal: We want to fit a curve to these points

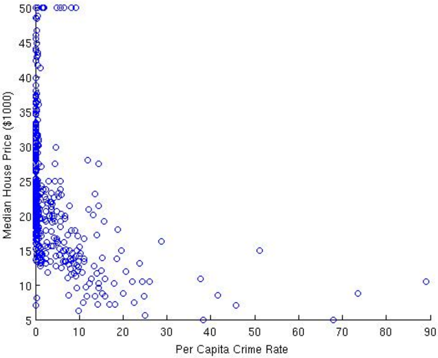
Simple 1-D regression



- Key Questions:
  - How do we parametrize the model?
  - What loss (objective) function should we use to judge the fit?
  - How do we optimize fit to unseen test data (generalization)?

Example: Boston Housing data

- Estimate median house price in a neighborhood based on neighborhood statistics
- Look at first (of 13) attributes: per capita crime rate



- Use this to predict house prices in other neighborhoods
- Is this a good input (attribute) to predict house prices?

Represent the Data

- Data is describe as pairs $\mathcal{D} = \{(x^{(1)}, t^{(1)}), \cdots, (x^{(N)}, t^{(N)})\}$
  - x is the input feature (per capita crime rate)
  - t is the target output (median house price)
- Here t is continuous, so this is a regression problem
- Model outputs y, an estimate of t

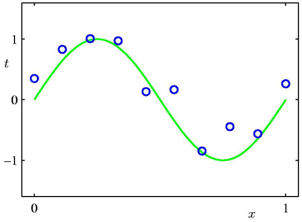
$$y(x) = w_0 + w_1 x$$

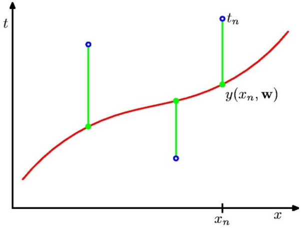
- What type of model did we choose?
- Divide the dataset into training and testing examples
  - Use the training examples to construct hypothesis, or function approximator, that maps x to predicted y
  - Evaluate hypothesis on test set

Noise

- A simple model typically does not exactly fit the data – lack of fit can be considered noise
- Sources of noise:
  - Imprecision in data attributes (input noise)
  - Errors in data targets (mis-labeling)
  - Additional attributes not taken into account by data attributes, affect target values (latent variables)
  - Model may be too simple to account for data targets

Least-squares Regression





- Define a model

$$y(x) = w_0 + w_1 x$$

- Standard loss/cost/objective function measures the squared error between y and the true value t

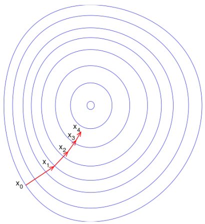
$$\ell(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2$$

- The loss for the red hypothesis is the sum of the squared vertical errors.
- How do we obtain the weights $\mathbf{w} = (w_0, w_1)$?

Optimizing the Objective

- One straightforward method: gradient descent
  - initialize $\mathbf{w}$ (e.g., randomly)
  - repeatedly update $\mathbf{w}$ based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$



- λ is the learning rate
- For a single training case, this gives the LMS update rule:

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda(t^{(n)} - y(x^{(n)}))x^{(n)}$$

- Note: As error approaches zero, so does the update

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. Batch updates: sum or average updates across every example n, then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)}))x^{(n)}$$

  2. Stochastic/online updates: update the parameters for each training case in turn, according to its own gradients

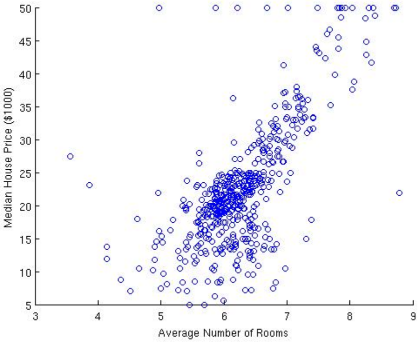
  - Underlying assumption: sample is independent and identically distributed (i.i.d.)

Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

$$y(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

- In the Boston housing example, we can look at the number of rooms



- We can use gradient descent to solve for each coefficient, or use linear algebra – solve system of equations

Linear Regression

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n, with observations indexed by j:

$$\mathbf{x}^{(n)} = \left(\mathbf{x}_1^{(n)}, \cdots, \mathbf{x}_d^{(n)}\right)$$

- We can incorporate the bias w_0 into $\mathbf{w}$, by using $x_0 = 1$, then

$$y = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

- We can then solve for $\mathbf{w} = (w_0, w_1, \cdots, w_d)$. How?
- What if our linear model is not good? How can we create a more complicated model?

Fitting a Polynomial

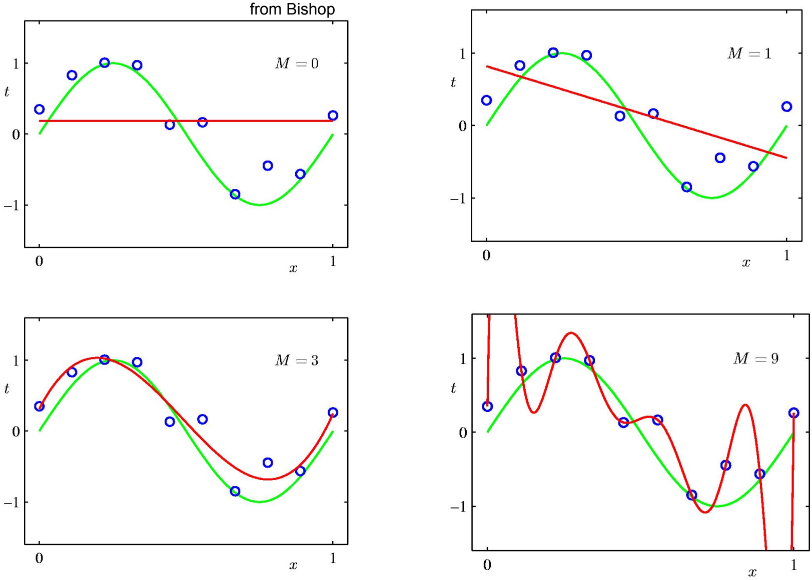
- We can create a more complicated model by defining input variables that are combinations of components of $\mathbf{x}$
- Example: an M-th order polynomial function

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

  where x^j is the j-th power of x

- We can use the same approach to optimize the values of the weights on each coefficient
- How do we do that?

Which fit is best?



Regularized least squares

- Increasing the input features this way can complicate the model considerably
- Goal: select the appropriate model complexity automatically
- Standard approach: regularization

$$\tilde{\ell}(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 + \alpha \mathbf{w}^T \mathbf{w}$$

- The penalty on the squared weights is known as ridge regression in statistics
- Leads to modified update rule

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda[\sum_{n=1}^{N} (t^{(n)} - y(x^{(n)}))x^{(n)} - \alpha \mathbf{w}]$$

1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit
  - More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model
- One method of assessing fit: test generalization = model's ability to predict the held out data
- Optimization is essential: stochastic and batch iterative approaches; analytic when available