Machine learning notes: week 2

Lars Yencken

Linear regression with multiple variables

Notation for multiple features

- Suppose we have multiple numeric input features
- $x_1 \dots x_n$ are the input features, where n is the number of features
- The j^{th} feature of the i^{th} training example is $x_j^{(i)}$
- Convention: $x_0^{(i)} = 1$ for every training example
- Hypothesis:
  - $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$
  - $h_\theta(x) = \theta^T x$ using matrix notation
- Call the parameter vector $\theta_0, \dots, \theta_n$ just θ for short

Gradient descent

- We can now generalise our update rule
- $\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$ for $j = 0, \dots, n$
- Thanks to our $x_0^{(i)} = 1$ convention, this lines up neatly with our earlier one-variable rule

Making it converge faster

- *Feature scaling*
  - Idea: ensure features are of a similar scale
  - Gradient descent works faster on a function with a symmetric bowl shape rather than a distored elipsis bowl shape
  - Harder to pick a good value of α if feature scale varies widely
  - Ideal case: every feature in to a $-1 \leq x_i \leq 1$, or a range of similar magnitude
- *Mean normalisation*
  - Replace x_i with $\frac{x_i - \mu_i}{\max(x_i) - \min(x_i)}$
  - Dividing by the standard deviation σ_i is fine too

Debugging gradient descent

- Plot $J(\theta)$ against the number of iterations
- It's working properly if
  - It decreases every iteration
  - It decreases reasonably quickly
  - It plateaus at the end
- If it's bowl-shaped or bouncing, α is too big
- If it doesn't plateau at the bottom, α may be too small or more iterations may be needed
- Try values of α with several different orders of magnitude
  - E.g. $\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$

Features and polynomial regression

- Sometimes combined features can be more useful than features in isolation
- We can fit polynomial features (e.g. x, x^2, x^3) by just adding them as if they were new features
- Since these features will vary widely by scale, you should use feature scaling
- Other non-polynomial functions (e.g. *sqrt*) might be useful feature transformations too

Normal equation

- A way to solve for θ directly, without iteration
- Intution: if $J(\theta)$ was quadratic, we could set the derivative to zero and solve it directly
- Solving the multivariate case:
  1. Build a matrix X of the data (including x_0 column set to 1) and y vector
  2. Then $\theta = (X^T X)^{-1} X^T y$
- In Octave: `pinv(X'*X)*X'*y`
- No need to do feature scaling when using the normal method
- Favour gradient descent when n is large (e.g. > 1000)
  - Inverting a matrix is an $O(n^3)$ operation, slowing the normal equation down
- Favour the normal equation when n is relatively small (e.g. ≤ 1000)
  - Don't need to pick α
  - Don't need to monitor convergence
- (Advanced) what if $X^T X$ is non-invertible ("singular")
  - Octave handles this case with the `pinv` function, but not if you use `inv` instead
  - This only happens if you have redundant features (e.g. $x_2 = 3 * x_1$), or too many features for your training set size

- If you have a lot of parameters, you can use *regularization* to handle this case better (covered later)