Overview

Linear Regression with One Variable

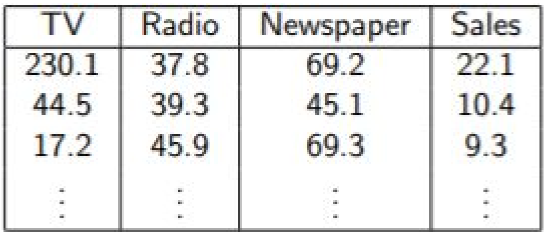
Gradient Descent

Linear Regression with Multiple Variables

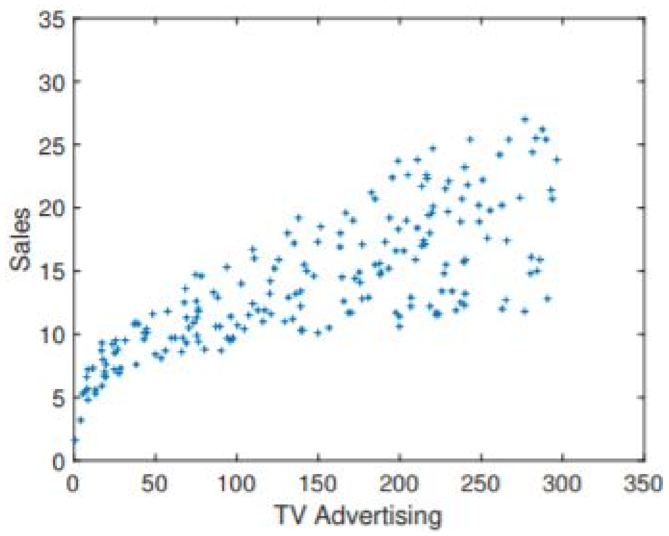
Gradient Descent with Multiple Variables

Example

Data consists of the advertising budget for three media (TV, radio, and newspapers) and the overall sales in 200 different markets.



TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
⋮	⋮	⋮	⋮



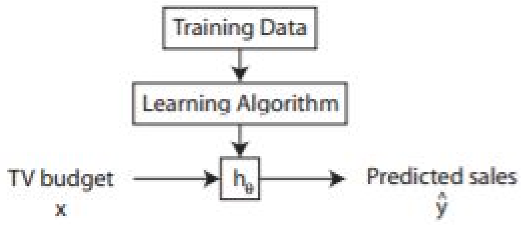
So how would we predict sales in a new area? Or sales with the TV budget increased to 350?

… Draw a line to fit the data points

Notation:

m=number of training instances
x = input variables/features
y = output variable/target
(x^i,y^i) is the ith instance

Our prediction: $\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$. θ_0, θ_1 are unknown parameters.



Our model:

How should we choose model parameters θ?

- Idea: Choose θ_0, θ_1 so that $h_\theta(x^{(i)})$ is as close to $y^{(i)}$ for each training instance as possible
- Least squares case: select the values for θ_0, θ_1 which minimise the cost function…

$$J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- Residuals are the difference between the value predicted by the fit and the observed value
    - **Do they look random or do they have some structure? Is our model satisfactory?**
    - We can estimate a confidence interval for the prediction made by our linear fit, using these residuals.
- **Cross-validation/bootstrapping could be used to estimate our confidence in the fit itself**

Gradient Descent

- A smarter minimisation approach than brute forcing it
    - **Start with some θ_0, θ_1**
    - **Repeat: Update θ_0, θ_1 with a new value which makes J(θ_0, θ_1) smaller**
    - This will eventually find the minimum if the curve is "bowl shaped" or convex
    - When a curve has several minima then we can't be sure which we will converge to…
        - **Might converge to a local minimum, not global**
- **One option:** carry out local search of θ_0, θ_1 to find one that decreases J.
- **Another option: Gradient Descent:**

$$temp0 := \theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1)$$
$$temp1 := \theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1)$$
$$\theta_0 := temp0,\ \theta_1 := temp1$$

-

- $\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) \approx \frac{J(\theta_0+\delta,\theta_1)-J(\theta_0,\theta_1)}{\delta}$ for δ sufficiently small.
  $J(\theta_0+\delta,\theta_1) \approx J(\theta_0,\theta_1)+\delta\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$
  When $\delta = -\alpha\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)$ then
  $J(\theta_0+\delta,\theta_1) \approx$
  $J(\theta_0,\theta_1)-\alpha\left(\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1)\right)^2$
    - α is the step size. Too small a step size will prolong the time taken to find a minimum
        - Too large a value can lead to us overshooting the minimum
    - We need to adjust the step size to converge in a reasonable amount of time
- For $J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$ with $h_\theta(x)=\theta_0+\theta_1 x$:
  - $\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{2}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})$
  - $\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{2}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)}$

  So gradient descent algorithm is:
  - repeat:
```
    $temp0 := \theta_0 - \frac{2\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})$
    $temp1 := \theta_1 - \frac{2\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)}$
    $\theta_0 := temp0,\ \theta_1 := temp1$
```

So back to the Advertising example…

- n=number of features (3)
- **Linear Regression with multiple variables**
    - **Hypothesis:** $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$
    - Create some dummy variable: $x_0 = 1$ to make multiplication more straight forward (same number of features as parameter weights)
    - **So we can redefine our hypothesis:** $h_\theta(x) = \theta^T x$
- **Our goals and cost functions are the same as they were with a single variable**
- Can brute force it, or…
- **Gradient descent:**
    - **Starting with some** θ
    -
```
for $j$=0 to $n$ {$tempj := \theta_j - \frac{2\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}$}
for $j$=0 to $n$ {$\theta_j := tempj$}
```

Fitting nonlinear curves: feature selection

- Hypothesis: $h_\theta(x) = \theta_0 + \theta_1\ x^2$
- Define a feature vector z with z=x^2 , this new z can be computed given x so it is known.
- We can now rewrite the hypothesis: $h_\theta(z) = \theta_0 + \theta_1 z$
- **So we can apply what we already know**

Gradient Descent in Practice

Feature Scaling and Mean Normalisation

- For better numerical behaviour
    - Try to make all features exist on a similar scale, ideally $-1 \le x_j \le 1$
    - Replace x_j with $x_j - \mu_j$ to achieve a zero mean
        - $\mu_j = \frac{1}{m} \sum_{i=1}^{n} x_j^{(i)}$
    - Generally normalize using $x_1 = \frac{x_1 - \mu_1}{max(x_1) - min(x_1)}$
- $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$
    - “Debugging”: How do we ensure that gradient descent is working properly? **J(θ) should decrease with every iteration**
        - **Stop when the decrease is <10^{-3}**
    - **Automated approaches to adjust the learning rate**

      ation. E.g. using **line search**:

      ```
      Repeat {
          Choose descent direction, e.g.
          for j=0 to n {$\delta_j := \frac{2}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$}
          Select $\alpha$ that makes $J(\theta + \alpha\delta)$ smallest
          $\theta := \theta - \alpha\delta$
      }
      ```

        - **More computationally expensive than a fixed learning rate**

Scalability

- **Batch Gradient Descent**

  ```
  Repeat {
      for j=0 to n {$tempj := \theta_j - \frac{2\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$}
      for j=0 to n {$\theta_j := tempj$}
  }
  ```

    - **At each iteration (i) uses all m training data, (ii) updates all n+1 parameters**

- **Alternatives:**
    - **Co-ordinate Gradient Descent**

```
$j = 0$
Repeat {
    j:=(j+1)mod (n+1)
    $\theta_j := \theta_j - \frac{2\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
}
```

        -
        - **At each iteration only update a single parameter**
    - **Stochastic Gradient Descent**

```
Repeat {
for $i$=1 to $m$ {
    for $j$=0 to $n$ {
        $tempj := \theta_j - \frac{2\alpha}{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
    }
    for $j$=0 to $n$ {$\theta_j := tempj$}
}
}
```

        -
        - **Repeatedly runs through training set, each time updating the parameters w.r.t a single training instance**
- **Closed-Form Solution**
    - **We can find the θ that minimises J(θ) in closed form**
        - Example: suppose we can choose one feature x_1 and one parameter θ_1
        - **Goal is to select the θ_1 to minimise** $J(\theta_1) = \sum_{i=1}^{m} (\theta_1 x_1^{(i)} - y^{(i)})^2$
        - **Compute the derivative w.r.t θ_1 :**

$$\frac{dJ}{d\theta_1} = \frac{1}{m} \sum_{i=1}^{m} (\theta_1 x_1^{(i)} - y^{(i)}))x_1^{(i)} = \frac{1}{m} \left(\theta_1 \sum_{i=1}^{m} (x_1^{(i)})^2 - \sum_{i=1}^{m} y^{(i)} x_1^{(i)} \right)$$

            -
            - Set the derivative equal to 0 and solve for θ_1 :

$$\theta_1 \sum_{i=1}^{m} (x_1^{(i)})^2 - \sum_{i=1}^{m} y^{(i)} x_1^{(i)} = 0$$

$$\text{i.e. } \theta_1 = \sum_{i=1}^{m} y^{(i)} x_1^{(i)} / \sum_{i=1}^{m} (x_1^{(i)})^2$$

            -
            - **In vector-matrix notation….**

$$\theta_1 = (X^T X)^{-1} X^T y$$

$$\text{where } y = \begin{bmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}, X = \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(m)} \end{bmatrix}$$

        -
        - **With multiple Features the minimising vector θ is:**
            - $\theta = (X^T X)^{-1} X^T y$

$$\text{where } y = \begin{bmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}, X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & & & & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

- $(X^T X)^{-1}$ is the inverse of matrix $X^T X$
- It satisfies $(X^T X)^{-1}(X^T X) = I$ where I is the identity matrix.

        -
    - **Comparison of Solution approaches (given m training examples, n features)**
        - **Gradient Descent**
            - Need to choose learning rate (α)
            - May need many iterations
            - Works well even when n is large
            - Stochastic Gradient Descent works well even when working with big data
        - **Closed-Form Solution**
            - No learning rate
            - Don't need to iterate
            - Need to compute $(X^T X)^{-1}$, n by n matrix and O(n^3)
                - Slow when n exceeds ~1000
            - Matrix X is m by n so won't scale well to big data

Regularisation and Model Selection

- **Bias-Variance trade-off**
    - Trying to find the right balance of overfitting and underfitting
- More data can help (noise becomes insignificant)
    - But can still overfit to data present and fail for any values outside of the set
- **Cross-validation for feature/model selection**
    - Draw uniformly at random and with replacement a set of p < m points from the set of m training data e.g. use p = 0.7m
    - For hypothesis of interest, find the parameter values θ that fit the set of p points with minimal square error
        - $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2.$

- Repeat to obtain multiple values for the square error J(θ) e.g 1000 times
- Repeat for each hypothesis of interest (e.g. each order of polynomial)
- **Note: as data was randomly sampled - these values themselves are noisy**

Regularisation

- Another tool in our armoury is **regularisation** i.e. add constraints/penalties on the parameters θ .
- Gives us a new J(θ): $$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + R(\theta)$$
    - **Where R(θ) is a penalty function.**
- **The two most common regularisation penalties:**
    - **Quadratic/L2 Penalty:** $R(\theta) = \theta^T\theta = \sum_{j=1}^{n}\theta_j^2$
        - **Ridge Regression**
        - **Encourages small values of** θ
        - **Multiply Penalty by 1/** λ
            - **Small lambda suggests we are certain that** θ **= 0**
            - **Large lambda says we don't know much about** θ
            - **Cross validation helps choose** λ
    - **L1 Penalty:** $R(\theta) = \sum_{i=1}^{n}|\theta_j|$
        - **LASSO regression**
        - **Encourages sparsity of solution i.e. few non-zero elements in** θ