3. Classification & Logistic Regression

Now we are now switching from regression problems to *classification problems*. In particular, we introduce a *Logistic regression* methodology to **classify data into discrete outcomes**. For example, we might use logistic regression to classify:

- *Email*: spam / not spam
- *Online Transactions*: Fraud? (yes/no)
- *Tumors*: Malignant / Benign

In each case, we have a **binary classification** problem. This is a *Supervised Learning* problem in which the goal is to predict which of two classes the input data falls into. Or, more specifically, the output, y will take on only two values, 0 and 1. Although this is focused on *binary classification*, most of what follows extends to multiple-class cases.

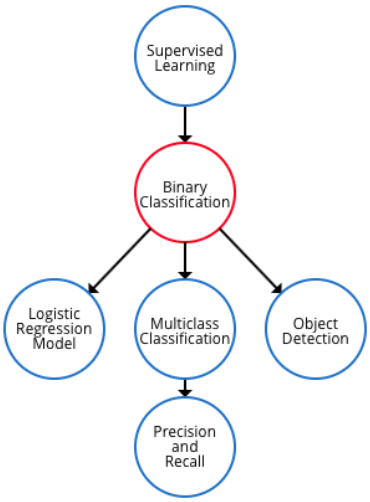


Figure 1. Binary Classification concept graph

3.1. Binary Classification.

In the examples above, we can think of a variable y that can take on two discrete values:

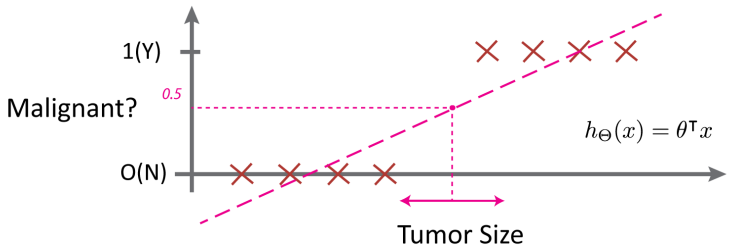
$$y \in \{0, 1\} \quad \begin{cases} 0: \text{ 'Negative Class'} \\ 1: \text{ 'Positive Class'} \end{cases}$$

The assignment of 0 or 1 is arbitrary, but *'Negative'* generally refers to the absence of something.

Example 3.1 : *Classification Model*

Now, wow do we develop a classification problem? Recall the cancer tumor example from linear regression. In this figure we have a set of tumors that are either *Malignant* (yes) or *Benign* (no).

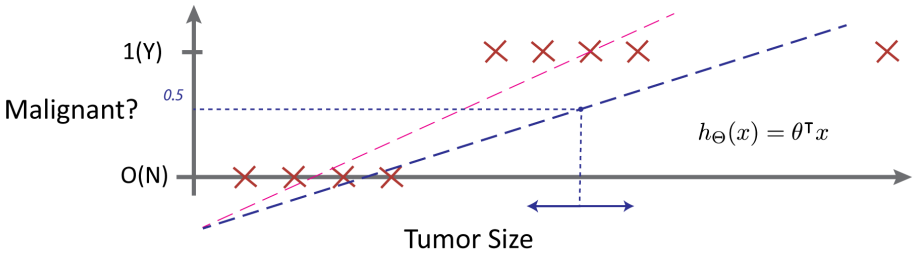
If we apply a linear regression: $h_\theta(x) = \theta^\intercal x$



Then, we can threshold the classifier output: $h_\theta(x)$ at 0.5:

$$\text{if } h_\theta(x) \geq 0.5, \text{ predict: } y = 1$$
$$\text{if } h_\theta(x) < 0.5, \text{ predict: } y = 0$$

In this particular example, it appears the data will be classified properly. However, if we change the problem by expanding horizontal axis a little bit – and adding a few training examples – we have a new problem:



Now, thresholding the hypothesis at 0.5, the division line is now shifted excessively to the right. So, applying linear regression to a classification problem often isn't a great idea.

Note: This hypothesis function, $h_\theta(x)$ can be greater than 1 or less than zero. Even though, for our classification problem, our *labels* should be $y = 0$ or 1.

To address this problem, we will develop an algorithm called *logistic regression* to bound our hypothesis function: $0 \leq h_\theta(x) \leq 1$

3.2. Logistic Regression Model.

Previously, in *linear regression*, we used the following hypothesis function:

$$h_\theta(x) = \theta^\intercal x$$

Now, we modify this representation as follows:

$$h_\theta(x) = g(\theta^\intercal x)$$

where:

$$g(z) = \frac{1}{1 + e^z} \qquad \textit{(Sigmoid Function)}$$

This function is also known as the *'logistic function'*, hence the name of the regression model. Combining the above, we have:

$$h_\theta(x) = \frac{1}{1 + e^{\theta^\intercal x}}$$

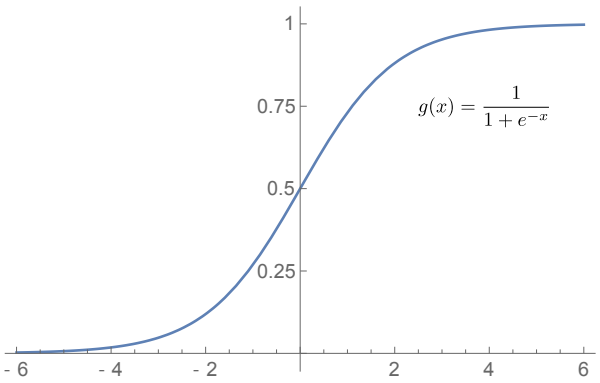


Figure 2. Sigmoid Function

Notice that the sigmoid function above, approaches 0 or 1 as x, or θ approaches ∞.

3.2.1. *Interpretation of Model.*

The output of the hypothesis function is to be interpreted as the estimated *probability* that $y = 1$ on the input x. That is:

$$h_\theta(x) = \text{ Estimated probability that } y = 1, \text{ on } x$$

Example 3.2 : *Interpretation*

Let's assume that we have:

$$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{Tumor Size} \end{bmatrix}$$

$$h_\theta(x) = 0.7$$

$\rightarrow$ *Based on hypothesis output: There is a* 70% *chance of tumor being malignant* $(y = 1)$.

Or, written mathematically:

$$h_\theta(x) = P(y = 1|x; \theta)$$

Which states: *probability that* $y = 1$*, given* x*, parameterized by* θ.

Now, since $y \in \{0, 1\}$, and we know:

$$P(y{=}0|x; \theta) + P(y{=}1|x; \theta) = 1$$

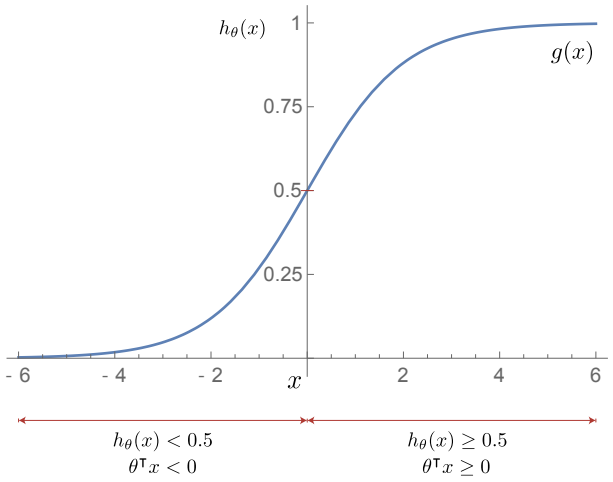
then we can also compute the probability for $y = 0$:

$$P(y{=}0|x; \theta) = 1 - P(y{=}1|x; \theta)$$

3.2.2. *Decision Boundary.*

To get a better sense of what the *logistic regression* hypothesis function is computing, let's take a closer look as to when this *hypothesis* will make predictions of $y = 1$ vs. $y = 0$. We recall that our hypothesis is represented as:

$$h_\theta(x) = g(\theta^\intercal x)$$
$$g(z) = \frac{1}{1+e^z}$$



From the figure above, suppose we predict:

$$(\mathbf{y = 1}) \quad \textit{if} \quad h_\theta(x) \geq 0.5$$

this will be true when:

$$\begin{aligned} g(x) &\geq 0.5 \\ h_\theta(x) = g(\theta^\intercal x) &\geq 0.5 \\ \theta^\intercal x &\geq 0 \end{aligned}$$

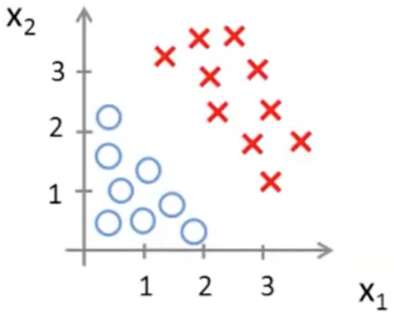
Or, in summary:

$$\begin{aligned} (\mathbf{y = 1}) \quad \textit{if} \quad & h_\theta(x) \geq 0.5 \\ & \theta^\intercal x \geq 0 \\ \\ (\mathbf{y = 0}) \quad \textit{if} \quad & h_\theta(x) < 0.5 \\ & \theta^\intercal x < 0 \end{aligned}$$

Example 3.3 : *Decision Boundary:*

Given the following hypothesis function and training set:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 \theta_2 x_2)$$
$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$



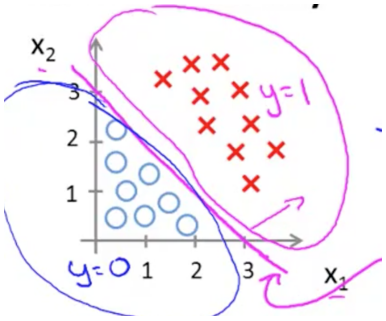
Now, given this choice of parameters, lets figure out where this hypothesis will predict the value of $y = 1$ & $y = 0$.

Predict:

$$(\mathbf{y = 1}) \quad \textit{if} \quad \underbrace{-3 + x_1 + x_2}_{\theta^T x} \geq 0$$

$$(\mathbf{y = 1}) \quad \textit{if} \quad x_1 + x_2 \geq 3$$

This is essentially the equation for a straight line that passes through 3 on the x_1 & x_2 axes. Plotting this:



Here, the middle line is the *Decision Boundary*.

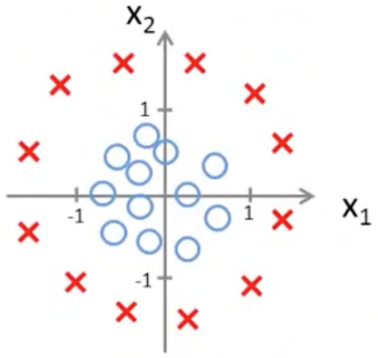
Example 3.4 : *Decision Boundary:* *Non-linearity*

Now, lets take a look at a *non-linear decision boundary. How can logistic regression fit this dataset?* Assume we have the following:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

with:

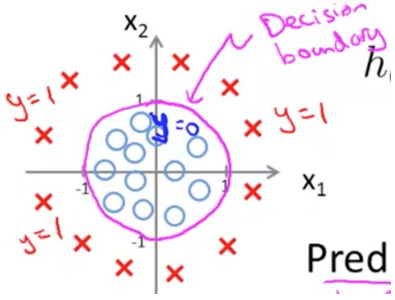
$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



Predict:

$$(\mathbf{y = 1}) \quad \textit{if} \quad \underbrace{-1 + x_1^2 + x_2^2}_{x_1^2 + x_2^2 \geq 1} \geq 0$$

The decision boundary for this data set is:

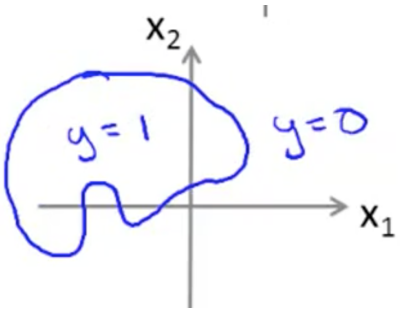


Note: The hypothesis, h_θ, and parameters, θ, define the decision boundary, not the training set. However, later on we will use the training set to fit the parameters, but not at this moment.

Example 3.5 : *Decision Boundary:* *Higher order polynomial*

Here is a simple example of a higher-order polynomial that is possible:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta x_1^3 + \dots)$$



3.3. Cost Function.

In this section we will discuss how to fit the parameters, θ, for logistic regression and develop a cost function.

Given the following training set with m *examples*, in which each example is represented by a *feature vector* $x \in \mathbb{R}^{n+1}$.

$$\textit{training set:} \qquad \{(x^{(1)}, y^{(1)}), (x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$\textit{features:} \qquad x \in \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad x_0 = 1,\ y \in \{0, 1\}$$

with the logistic hypothesis function:

$$h_\theta(x) = \frac{1}{1 + e^{\theta^\intercal x}}$$

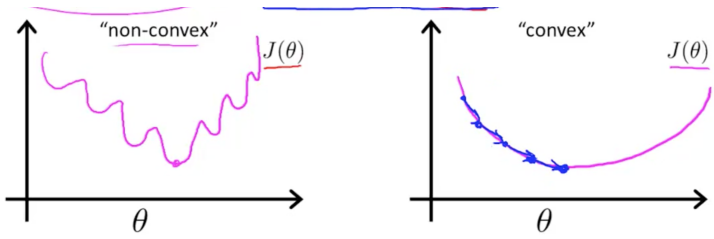
How do we choose the parameters, θ? Recall, from linear regression we used:

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

Simplifying this, we set:

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \Big(h_\theta(x^{(i)} - y^{(i)} \Big)^2$$

Unfortunately, it turns out that if we use this particular cost function, this would be a non-convex function of the parameter's data.

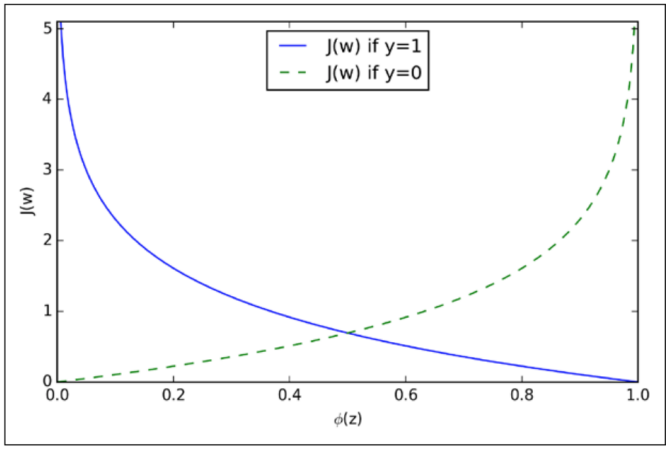


3.3.1. *Logistic Regression Cost Function.*

Now, we introduce the cost function that we will actually use for logistic regression:

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Plotting this function for $y = 1$ & $y = 0$, we have:



Note:

$$\begin{aligned} \textbf{\textit{Cost = 0}} \quad & \text{if} \quad y = 1,\ h_\theta(x) = 1 \\ \textbf{\textit{Cost} = } \infty \quad & \text{if} \quad h_\theta(x) \to 0 \end{aligned}$$

This captures the intuition that if $h_\theta(x) = 0$ (predict: $P(y = 1|x; \theta) = 0$ which is impossible), but $y = 1$, then the learning algorithm is penalized by a very large cost.

Or, a better analogy: it's like saying the probability that you have a malignant tumor, the probability that $y = 1$, *is zero*. So, it's absolutely *impossible* that your tumor is malignant.

But if it turns out that the tumor actually IS malignant (if $y = 1$) even after we told them that the probability of it happening is zero, then we penalize the learning algorithm by a very, very large cost. And that's captured by having this cost go towards infinity if $y = 1$ and $h_\theta(x) \to 0$.

3.3.2. *Simplified Cost Function.*

Since y is always 0 or 1, we can simplify the cost function as follows:

Definition 1. ***Logistic Regression***

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

Then, to fit the parameters θ:

$$\min_\theta J(\theta)$$

Then, if we're given a new set of data to make a prediction from, we use:

$$h_\theta(x) = \frac{1}{1 + e^{\theta^\intercal x}}$$

Which is interpreted as:

$$h_\theta(x) = P(y = 1|x; \theta)$$

Which is the output of the hypothesis, interpreted as the *probability of* $y = 1$*, parameterized by* θ*.*

3.3.3. *Gradient Descent.*

All that remains is to choose a method for minimizing $J(\theta)$, so that we can actually fit the parameters to our training set. The way we're going to minimize the cost function is using *gradient descent*. The gradient descent algorithm is:

Repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J\big(\theta\big) \quad \text{(Simultaneously update for all } \theta_j)$$

}

Computing the derivative term:

$$\frac{\partial}{\partial \theta_j} J\big(\theta\big) = \frac{1}{m} \sum_{j=1}^{n} \big(h_\theta(x^{(i)}) - y^{(i)}\big) \cdot x_j^{(i)}$$

and plugging back in:

Repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{j=1}^{n} \big(h_\theta(x^{(i)}) - y^{(i)}\big) \cdot x_j^{(i)}$$

(Simultaneously update for all θ_j)

}

A vectorized implementation may be written as:

$$\theta_j := \theta_j - \frac{\alpha}{m} X^\intercal \big(g(X\theta) - y\big)$$

Q: This looks identical to linear regression - *so what has changed?*

- Now, the hypothesis function has taken a new form:
  - *Linear Regression:* $h_\theta(x) = \theta^\intercal x$
  - *Logistic Regression:* $h_\theta(x) = \frac{1}{1+e^{-\theta^\intercal x}}$

3.4. Advanced Optimization.

Previously we used *gradient descent* for minimizing the cost function $J(\theta)$ for logistic regression. Now, we move on to some more advanced optimization algorithms that scale much better.

In general, we need:

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$

In order to compute the next step. Besides *gradient descent* there are several algorithms available:

Optimization Algorithms

- Conjugate gradient
- BFGS
- L-BFGS

Advantages

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages

- More complex

These methods use a line search algorithm that optimized the learning rate α.

Example 3.6 : *Matlab Implementation*

Assume we're given the following:

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

In which:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \qquad \frac{\partial}{\partial \theta_1} = 2(\theta_1 - 5)$$
$$\frac{\partial}{\partial \theta_2} = 2(\theta_2 - 5)$$

If we wish to minimize this function:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

We can code this in `Matlab` as follows:

```
% Define Cost Function
function [jVal, gradient] = costFunction(theta)
  jVal = (theta(1)-5)^2 + (theta(2)-5)^2;

  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5)
```

Now call `fminunc`, or *function, minimize, unconstrained*:

```
% set options for optimization routine
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);

% run fminunc
[optTheta, functionVal, exitFlag] = fminunc(\@costFunction, initialTheta, options);
```

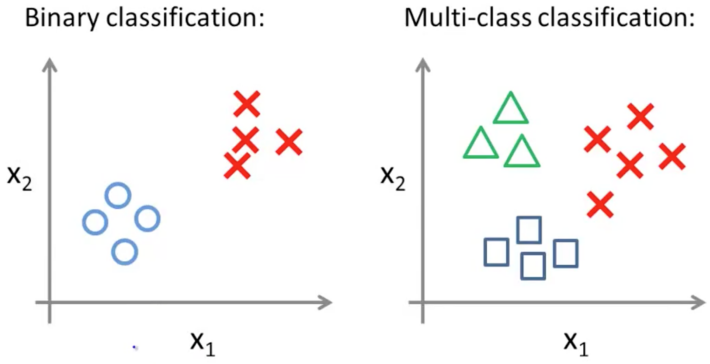
3.5. Multi-Class Classification.

Now we look at applying logistic regression for *multi-class classification* problems. In particular, the *one-versus-all classification* algorithm.

What is multi-class classification? Same as before, but now we approach the classification of data when we have more than two categories. For example:

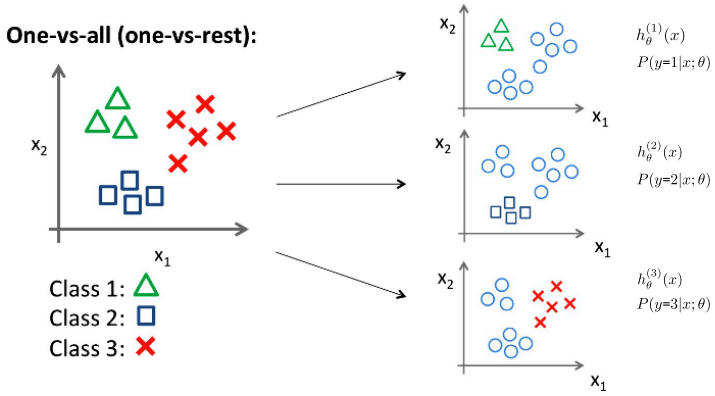
- ***Email tagging:*** *Work, Friends, Family, Hobby*
- ***Medical diagrams:*** *Healthy, Cold, Flu*
- ***Weather:*** *Sunny, Cloudy, Rain, Snow*

In each case, each category is assigned a discrete number, $y \in \{0, 1, \ldots, n\}$, instead of $y = \{0, 1\}$.



One-vs-All: *Basic Idea*

Essentially, we convert the multi-class classification into multiple binary classification problems. This is done by dividing our problem into $n+1$ binary classification problems, then predicting the probability that 'y' is a member of one of our classes.



For the first instance, the classifier is learning to recognize the triangles. More specifically, it's thinking of the triangles as a positive class, or trying to estimate the probability of $y = 1$.

Similarly, the second instance views the squares as a positive class and so it's trying to estimate the probability that $y = 2$.

Hence, this can be summarized as:

$$h_\theta^{(i)} = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$

Summary: *One-vs-all:*

- Train a logistic regression classifier, $h_\theta^{(i)}(x)$ for each class, i, to predict the probability that $y = i$.
- On a new input x, to make a prediction, pick the class i that maximizes:

$$\max_i h_\theta^{(i)}(x)$$

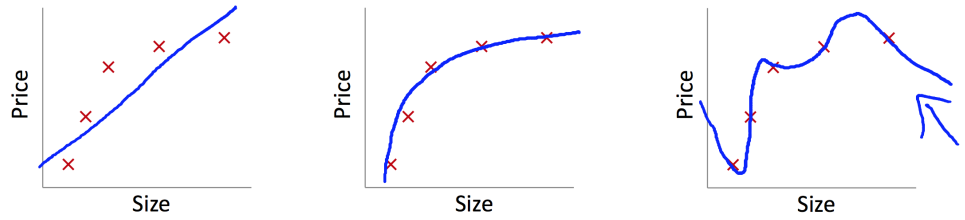
3.6. Regularization.

By now, we've seen a couple different learning algorithms, *linear regression* and *logistic regression*. They work well for many problems, but when you apply them to certain machine learning applications, they can run into a problem called overfitting that can cause them to perform very poorly.

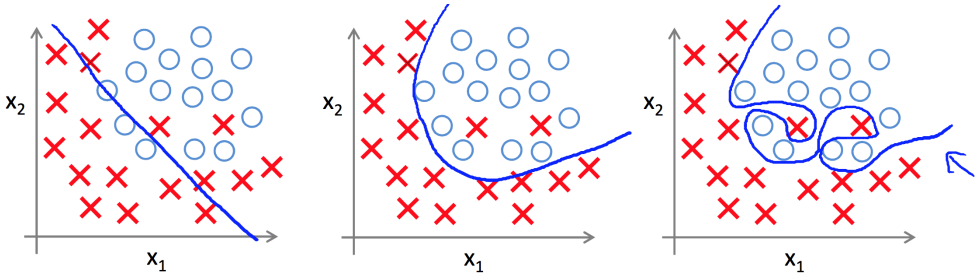
Definition 2. ***Overfitting:*** If we have too many features, the learned hypothesis may fit the training set very well $\left(J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \approx 0\right)$, but fail to generalize to new examples (predict prices on new examples).

In the figure below, we see several curve-fitting examples for polynomials of increasing order:

- ***Linear:*** $h_\theta(x) = \theta_0 + \theta_1 x$
- ***Quad:*** $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
- ***Cubic:*** $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$



The same phenomena can be see with *logistic regression:*

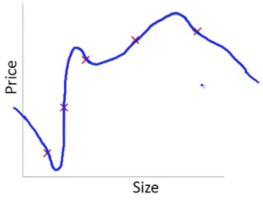


3.6.1. *Addressing Overfitting.*

More often, there are many features, which makes visualization very difficult. In addition, if there are many features and not very many examples, overfitting may be expected.

Addressing overfitting:

x_1 = size of house
x_2 = no. of bedrooms
x_3 = no. of floors
x_4 = age of house
x_5 = average income in neighborhood
x_6 = kitchen size
$\vdots$
x_{100}



In order to address overfitting, we generally have two options:

(1) ***Reduce number of features.***

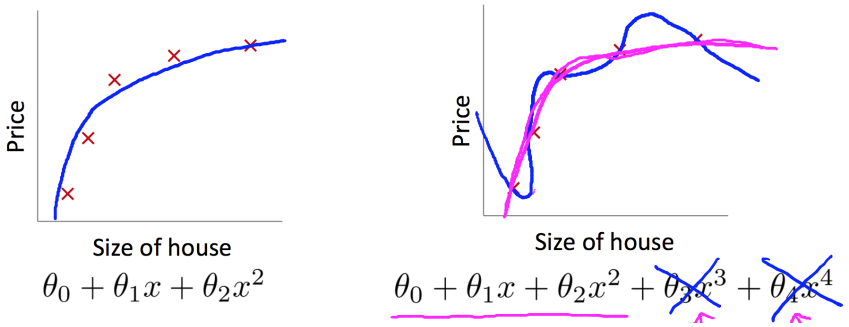
- Manually select which features to keep
- Model selection algorithm

(2) ***Regularization.***

- Keep all features, but reduce magnitude of parameters θ_j
- Works well when we have a lot of features, each of which contributes a bit to predicting y

3.6.2. *Cost Function.*

Previously, we saw that overly high order degree polynomials end up with a curve that may fit the training set very well, but does not generalize well. Suppose we were to penalize two of the parameters, θ_3 & θ_4.



This may be accomplished by adding a few terms to our cost function:

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2$$

Then, once the parameters, θ are solved for, $\theta_3 \approx \theta_4 \approx 0$. In this way, we can reduce their influence.

In addition, this will lead to smaller values for the parameters, and subsequently:

- 'Simpler' hypothesis (and often smoother)
- Less prone to overfitting

Example 3.7 : *Regularization:* *Housing*

For this example, say we have the following

- *Features:* $x_1, x_2, \ldots, x_{100}$
- *Parameters:* $\theta_0, \theta_1, \theta_2, \ldots, \theta_{100}$

Since there are so many features, we cannot determine which are most important. Hence, we take our general cost function and add an extra regularization term:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{i=1}^{n} \theta_j^2$$

In which λ is the ***regularization parameter***. This term controls the trade-off between keeping the parameters small and representing the training set well.

- If λ is very large, then all parameters $\theta \rightsquigarrow 0$ *(underfitting)*

Note: By convention, we do not regularize θ_0, this term sums over the number of parameters, n.

3.6.3. *Regularized Linear Regression.*

Now, including the regularization term, we can rewrite the algorithm as follows:

```
Repeat {
```

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_0^{(i)}$$

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{j=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}$$

```
}
```

Here, we know that $(1 - \alpha\frac{\lambda}{m}) \leq 1$. Hence, at every iteration, we are shrinking the θ_j value and then performing an update operation.

Normal Equation:

We can also add a regularization term to the normal equations.

$$X = \begin{bmatrix} — & x^{(1)\intercal} & — \\ — & x^{(2)\intercal} & — \\ & \vdots & \\ — & x^{(m)\intercal} & — \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}, \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m,$$

Then, we simply add the following

$$\theta = \left(X^\intercal X + \lambda \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}\right)^{-1} X^\intercal y$$

Note on Singularity:

Suppose we have: $m \leq n$, *(examples)* $\times$ *(features)*.

Then $A = X^\intercal X$ will be singular / non-invertible. However, if $\lambda > 0$, then it can be shown that the matrix will be invertible.

3.6.4. *Regularized Logistic Regression.*

As shown above, logistic regression is also susceptible to overfitting as linear regression. We can add a regularization term as follows:

$$J(\theta) = -\frac{1}{m} \sum_{j=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{j=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]$$

}

3.7. Implementation. This section provides key aspects of the MATLAB implementation.

3.7.1. ***Regularized linear regression cost function.***

Recall that regularized linear regression has the following cost function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{i=1}^{n} \theta_j^2$$

To compute the cost function and gradient (needed by advanced algorithms):

```
function [J, grad] = linearRegCostFunction(X, y, theta, lambda)
%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear
%regression with multiple variables
%   [J, grad] = LINEARREGCOSTFUNCTION(X, y, theta, lambda) computes the
%   cost of using theta as the parameter for linear regression to fit the
%   data points in X and y. Returns the cost in J and the gradient in grad

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));

% set hypthosis fcn
h = X*theta;

% set J
J = (1/(2*m))*sum((h-y).^2) + (lambda/(2*m))*sum(theta(2:end).^2);

% compute gradient - note that the first term is different.
grad(1) = (1/m)*sum((h-y).*X(:,1));
grad(2:end) = ((1/m)*sum((h-y).*X(:,2:end)))' + (lambda/m)*theta(2:end);

grad = grad(:);

end
```