instead maximize the **log likelihood** $\ell(\theta)$:

$$\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2.
\end{aligned}$$

Hence, maximizing $\ell(\theta)$ gives the same answer as minimizing

$$\frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2,$$

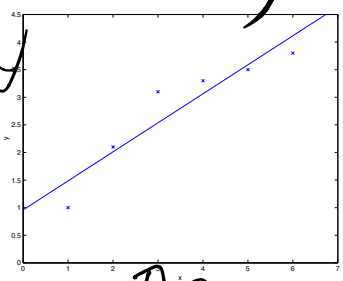
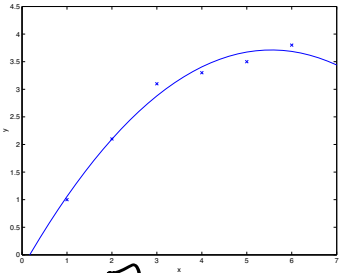
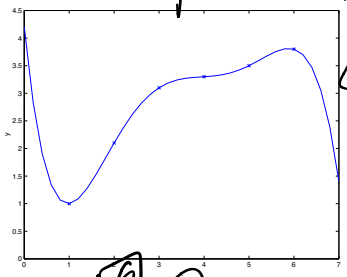
which we recognize to be $J(\theta)$, our original least-squares cost function.

To summarize: Under the previous probabilistic assumptions on the data, least-squares regression corresponds to finding the maximum likelihood estimate of θ. This is thus one set of assumptions under which least-squares regression can be justified as a very natural method that's just doing maximum likelihood estimation. (Note however that the probabilistic assumptions are by no means *necessary* for least-squares to be a perfectly good and rational procedure, and there may—and indeed there are—other natural assumptions that can also be used to justify it.)

Note also that, in our previous discussion, our final choice of θ did not depend on what was σ^2, and indeed we'd have arrived at the same result even if σ^2 were unknown. We will use this fact again later, when we talk about the exponential family and generalized linear models.

4 Locally weighted linear regression

Consider the problem of predicting y from $x \in \mathbb{R}$. The leftmost figure below shows the result of fitting a $y = \theta_0 + \theta_1 x$ to a dataset. We see that the data doesn't really lie on straight line, and so the fit is not very good.

Instead, if we had added an extra feature x^2, and fit $y = \theta_0 + \theta_1 x + \theta_2 x^2$, then we obtain a slightly better fit to the data. (See middle figure) Naively, it might seem that the more features we add, the better. However, there is also a danger in adding too many features: The rightmost figure is the result of fitting a 5-th order polynomial $y = \sum_{j=0}^{5} \theta_j x^j$. We see that even though the fitted curve passes through the data perfectly, we would not expect this to be a very good predictor of, say, housing prices (y) for different living areas (x). Without formally defining what these terms mean, we'll say the figure on the left shows an instance of **underfitting**—in which the data clearly shows structure not captured by the model—and the figure on the right is an example of **overfitting**. (Later in this class, when we talk about learning theory we'll formalize some of these notions, and also define more carefully just what it means for a hypothesis to be good or bad.)

As discussed previously, and as shown in the example above, the choice of features is important to ensuring good performance of a learning algorithm. (When we talk about model selection, we'll also see algorithms for automatically choosing a good set of features.) In this section, let us talk briefly talk about the locally weighted linear regression (LWR) algorithm which, assuming there is sufficient training data, makes the choice of features less critical. This treatment will be brief, since you'll get a chance to explore some of the properties of the LWR algorithm yourself in the homework.

In the original linear regression algorithm, to make a prediction at a query point x (i.e., to evaluate $h(x)$), we would:

1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

In contrast, the locally weighted linear regression algorithm does the following:

1. Fit θ to minimize $\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$.
2. Output $\theta^T x$.

Here, the $w^{(i)}$'s are non-negative valued **weights**. Intuitively, if $w^{(i)}$ is large for a particular value of i, then in picking θ, we'll try hard to make $(y^{(i)} - \theta^T x^{(i)})^2$ small. If $w^{(i)}$ is small, then the $(y^{(i)} - \theta^T x^{(i)})^2$ error term will be pretty much ignored in the fit.

A fairly standard choice for the weights is[4]

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

Note that the weights depend on the particular point x at which we're trying to evaluate x. Moreover, if $|x^{(i)} - x|$ is small, then $w^{(i)}$ is close to 1; and if $|x^{(i)} - x|$ is large, then $w^{(i)}$ is small. Hence, θ is chosen giving a much higher "weight" to the (errors on) training examples close to the query point x. (Note also that while the formula for the weights takes a form that is cosmetically similar to the density of a Gaussian distribution, the $w^{(i)}$'s do not directly have anything to do with Gaussians, and in particular the $w^{(i)}$ are not random variables, normally distributed or otherwise.) The parameter τ controls how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x; τ is called the **bandwidth** parameter, and is also something that you'll get to experiment with in your homework.

Locally weighted linear regression is the first example we're seeing of a **non-parametric** algorithm. The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm, because it has a fixed, finite number of parameters (the θ_i's), which are fit to the data. Once we've fit the θ_i's and stored them away, we no longer need to keep the training data around to make future predictions. In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around. The term "non-parametric" (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis h grows linearly with the size of the training set.

[4] If x is vector-valued, this is generalized to be $w^{(i)} = \exp(-(x^{(i)} - x)^T(x^{(i)} - x)/(2\tau^2))$, or $w^{(i)} = \exp(-(x^{(i)} - x)^T\Sigma^{-1}(x^{(i)} - x)/2)$, for an appropriate choice of τ or Σ.

Part II
Classification and logistic regression

Let's now talk about the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** problem in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then $x^{(i)}$ may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the **negative class**, and 1 the **positive class**, and they are sometimes also denoted by the symbols "-" and "+." Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** for the training example.

5 Logistic regression

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$.

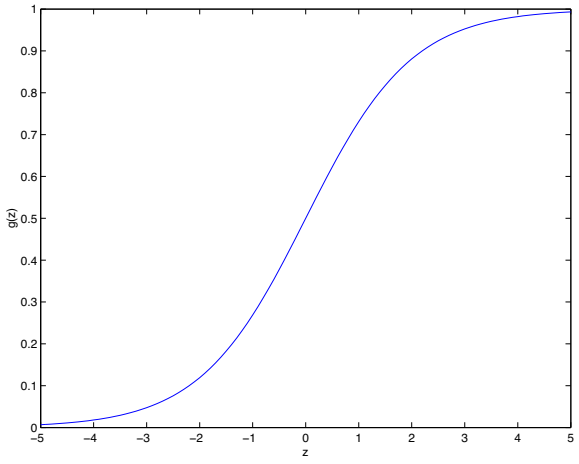
To fix this, let's change the form for our hypotheses $h_\theta(x)$. We will choose

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is called the **logistic function** or the **sigmoid function**. Here is a plot showing $g(z)$:



Notice that $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. Moreover, g(z), and hence also $h(x)$, is always bounded between 0 and 1. As before, we are keeping the convention of letting $x_0 = 1$, so that $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$.

For now, let's take the choice of g as given. Other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons that we'll see later (when we talk about GLMs, and when we talk about generative learning algorithms), the choice of the logistic function is a fairly natural one. Before moving on, here's a useful property of the derivative of the sigmoid function, which we write as g':

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

基础求导

So, given the logistic regression model, how do we fit θ for it? Following how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let's endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood.

极大似然估计

Let us assume that

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Note that this can be written more compactly as

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Assuming that the m training examples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

As before, it will be easier to maximize the log likelihood:

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

How do we maximize the likelihood? Similar to our derivation in the case of linear regression, we can use gradient ascent. Written in vectorial notation, our updates will therefore be given by $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$. (Note the positive rather than negative sign in the update formula, since we're maximizing, rather than minimizing, a function now.) Let's start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient ascent rule:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\ &= (y - h_\theta(x)) \, x_j \end{aligned}$$