MLSS 2014 – Introduction to Machine Learning Lecture 1

J. Zico Kolter

July 7, 2014

Outline

What is machine learning?

Supervised learning: regression

"Non-linear" regression, overfitting, and model selection

Supervised learning: classification

Outline

What is machine learning?

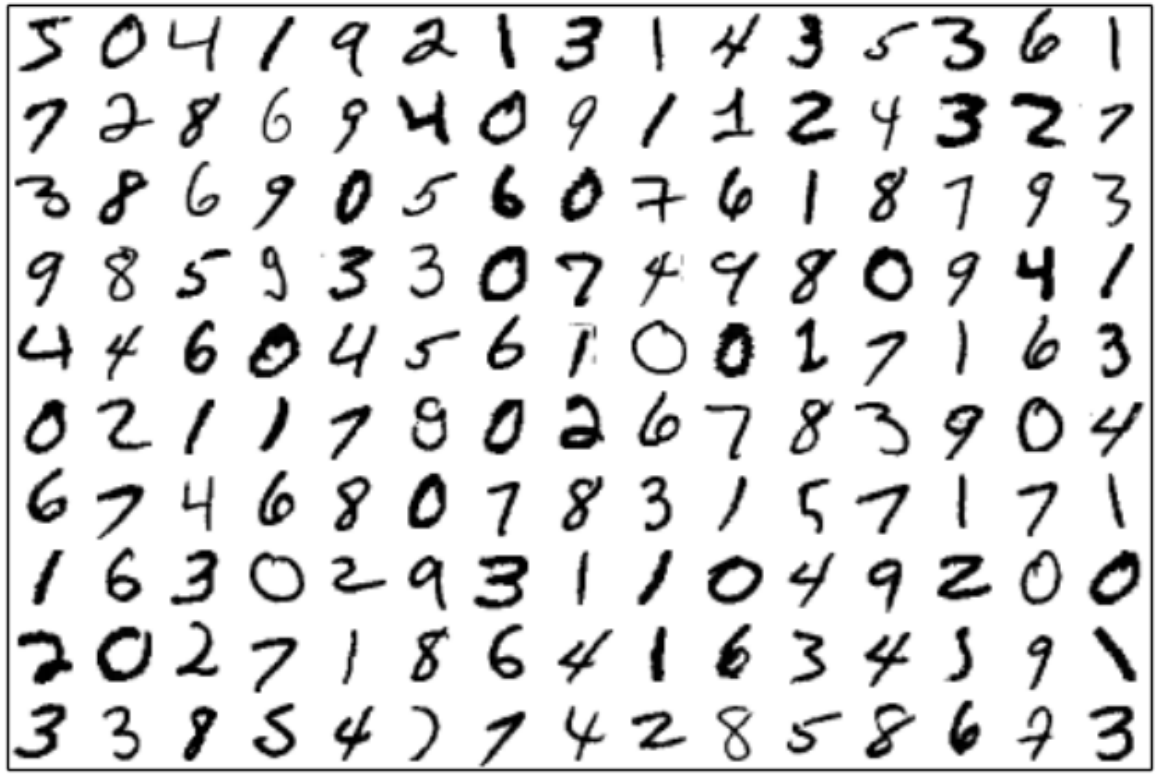
Supervised learning: regression

"Non-linear" regression, overfitting, and model selection

Supervised learning: classification

Introduction: digit classification

- The task: write a program that, given a 28x28 grayscale image of a digit, outputs the string representation



Digits from MNIST dataset
(http://yann.lecun.com/exdb/mnist/)

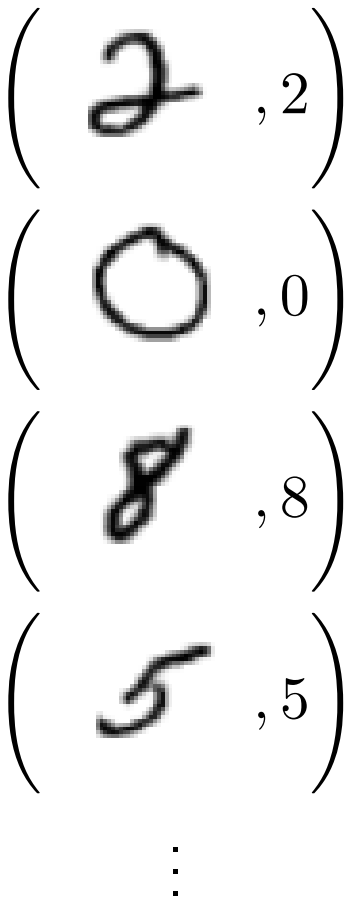
- One approach: try to write a program by hand that uses your a priori knowledge of digits to properly classify the images

- Alternative method (machine learning): collect a bunch of images and their corresponding digits, write a program that uses this data to build its own method for classifying images

- (More precisely, this is a subset of machine learning called *supervised learning*)

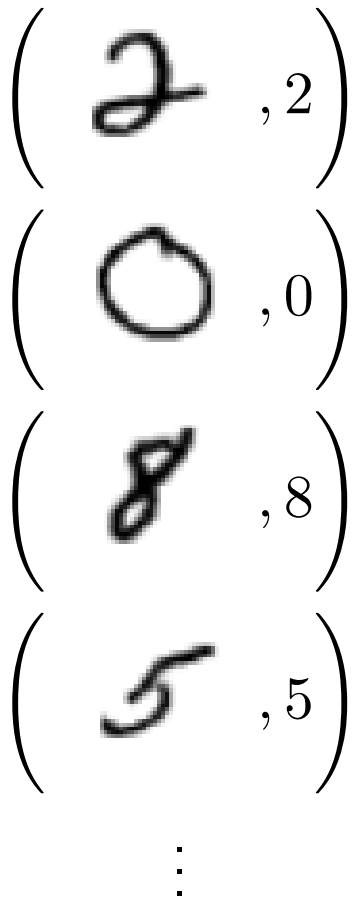
A supervised learning pipeline

Training Data



A supervised learning pipeline

Training Data

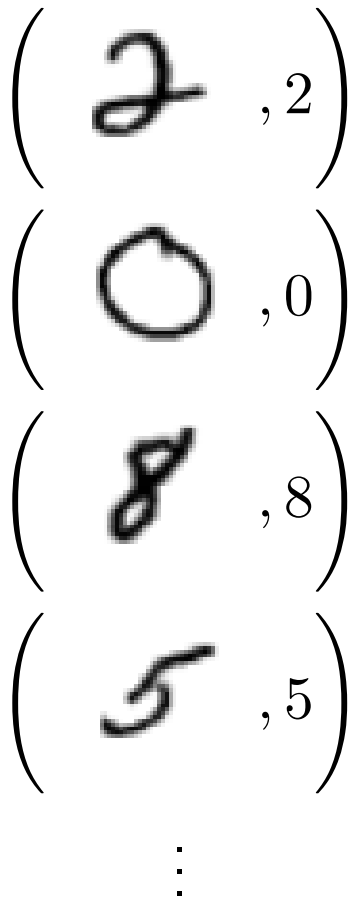


Machine Learning

$\longrightarrow$ Hypothesis function h_θ

A supervised learning pipeline

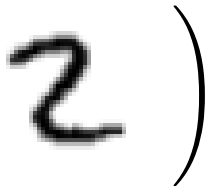
Training Data

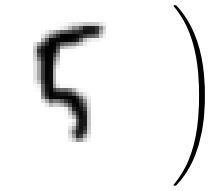


Machine Learning

$\longrightarrow$ Hypothesis function h_θ

Deployment

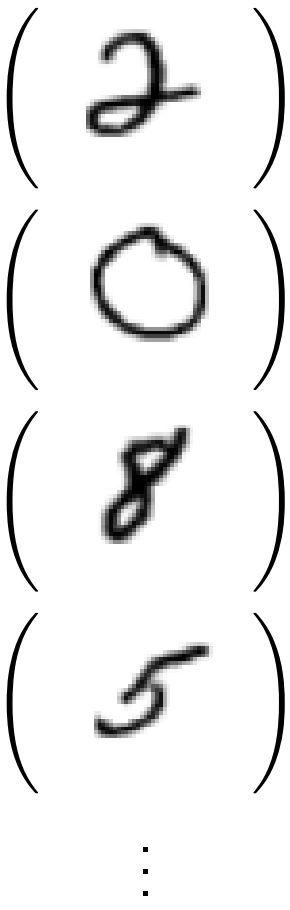
Prediction $= h_\theta$ 

Prediction $= h_\theta$ 

$\vdots$

Unsupervised learning

Training Data

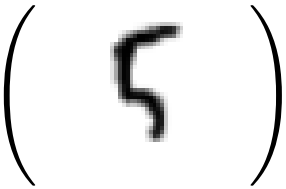


Machine Learning

$\longrightarrow$ Hypothesis function h_θ

Deployment

Prediction $= h_\theta$

Prediction $= h_\theta$ 

$\vdots$

Outline

What is machine learning?

Supervised learning: regression

"Non-linear" regression, overfitting, and model selection

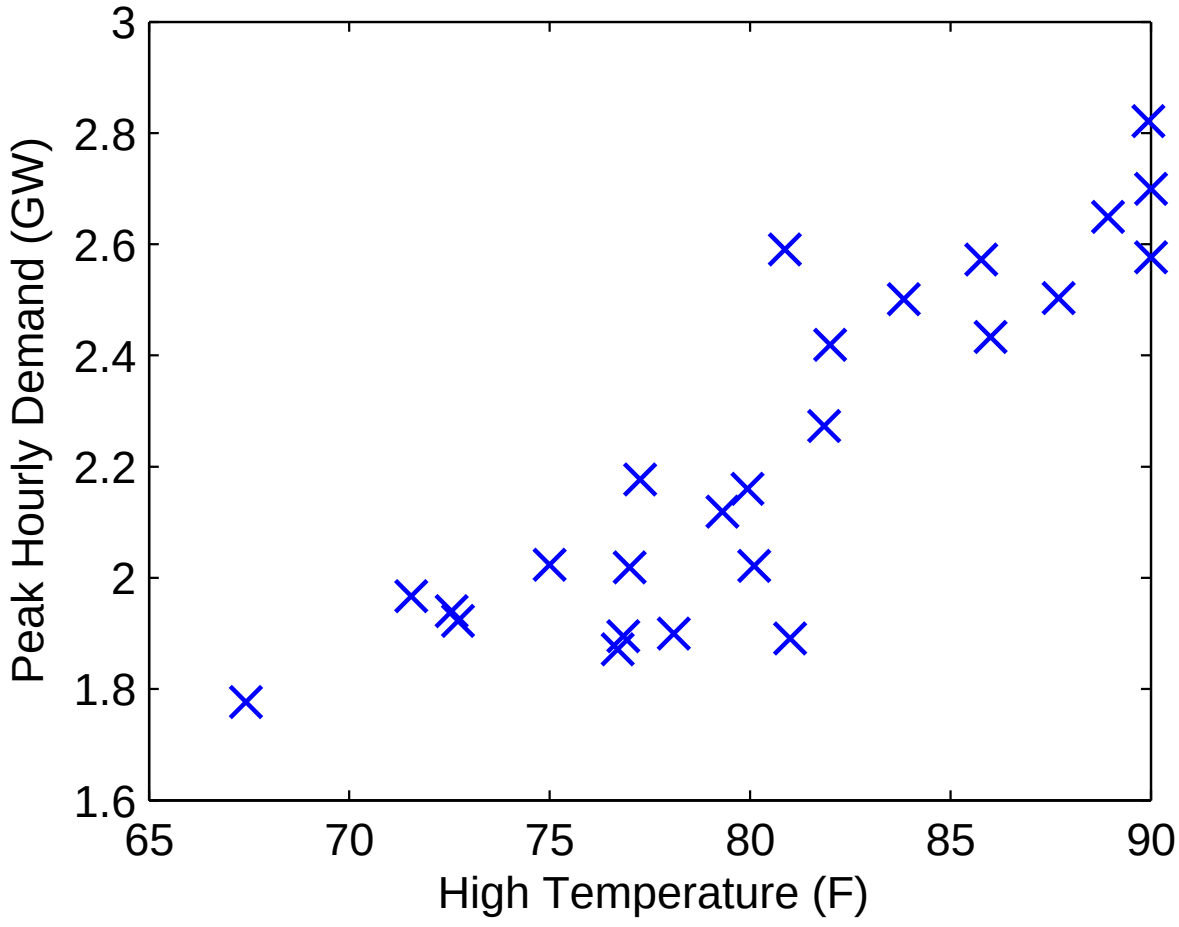
Supervised learning: classification

A simple example: predicting electricity use

- What will peak power consumption be in the Pittsburgh area tomorrow?

- Collect data of past high temperatures and peak demands

High Temperature (F)	Peak Demand (GW)
76.7	1.87
72.7	1.92
71.5	1.96
86.0	2.43
90.0	2.69
87.7	2.50
⋮	⋮



Several days of peak demand vs. high temperature in Pittsburgh

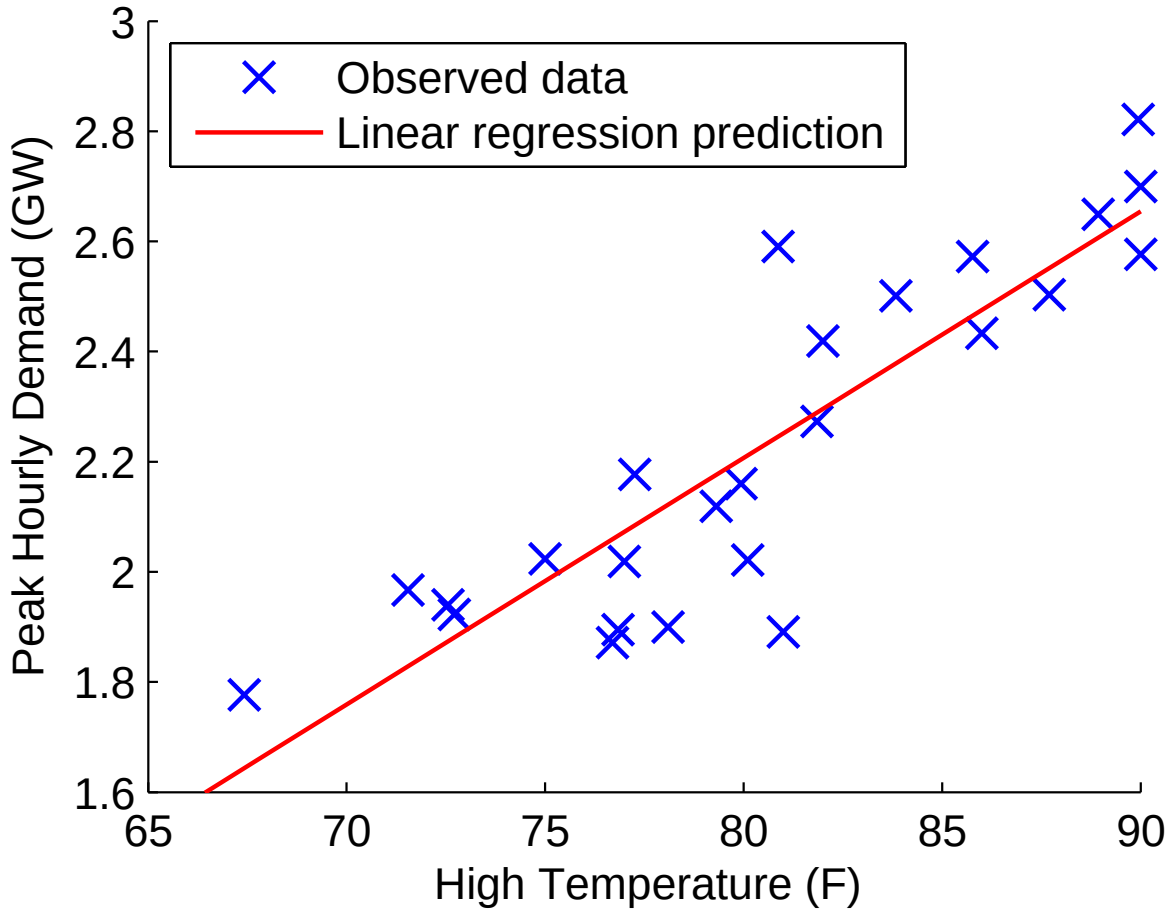
- Hypothesize model

$$\text{Peak demand} \approx \theta_1 \cdot (\text{High temperature}) + \theta_2$$

  for some numbers θ_1 and θ_2

- Then, given a forecast of tomorrow's high temperature, we can predict the likely peak demand by plugging it into our model

- Equivalent to "drawing a line through the data"



Notation

- **Input features**: $x^{(i)} \in \mathbb{R}^n, \quad i = 1, \ldots, m$
  - E.g.: $x^{(i)} \in \mathbb{R}^2 = \begin{bmatrix} \text{high temperature for day } i \\ 1 \end{bmatrix}$

- **Output**: $y^{(i)} \in \mathbb{R}$ (*regression* task)
  - E.g.: $y^{(i)} \in \mathbb{R} = \{\text{peak demand for day } i\}$

- **Model Parameters**: $\theta \in \mathbb{R}^n$

- **Hypothesis function**: $h_\theta(x) : \mathbb{R}^n \to \mathbb{R}$
  - Hypothesis function: $h_\theta(x)$ returns a *prediction* of the output y, e.g. *linear regression*

$$h_\theta(x) = \sum_{i=1}^{n} x_i \theta_i \; \left(\; \equiv x^T \theta, \quad \equiv \langle x, \theta \rangle \right)$$

Loss functions

- How do we measure how "good" a hypothesis is on the training data?

- Typically done by introducing a *loss function*

$$\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$$

- Intuitively, this function outputs a "small" value if $h_\theta(x)$ is "close" to y, a large value if it is "far" from y

- E.g., for regression, squared loss

$$\ell\left(h_\theta(x), y\right) = \left(h_\theta(x) - y\right)^2$$

The canonical machine learning problem

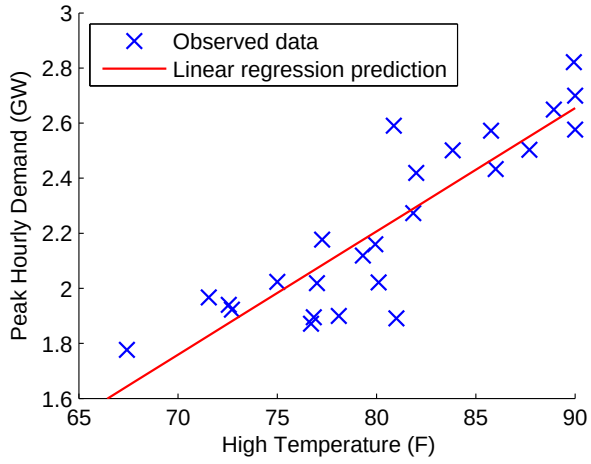
- Given a collection of input features and outputs $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, and a hypothesis function h_θ, find parameters θ that minimize the sum of losses

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

- Virtually all (supervised) learning algorithms can be described in this form, we just need to specify three things:

  1. The hypothsis class: h_θ
  2. The loss function: ℓ
  3. The algorithm for solving the optimization problem (often approximately)

Return to power demand forecasting



- Linear hypothesis class: $h_\theta(x) = x^T\theta$
- Squared loss function: $\ell(h_\theta(y), y) = (h_\theta(x) - y)^2$
- So how do we optimize:

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta(x^{(i)}), y^{(i)}\right) \equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \left(x^{(i)^T}\theta - y^{(i)}\right)^2 ?$$

Aside: optimization problems

- The problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

  is an example of an *optimization problem*; we want to find parameters θ that *minimize* the value of the function

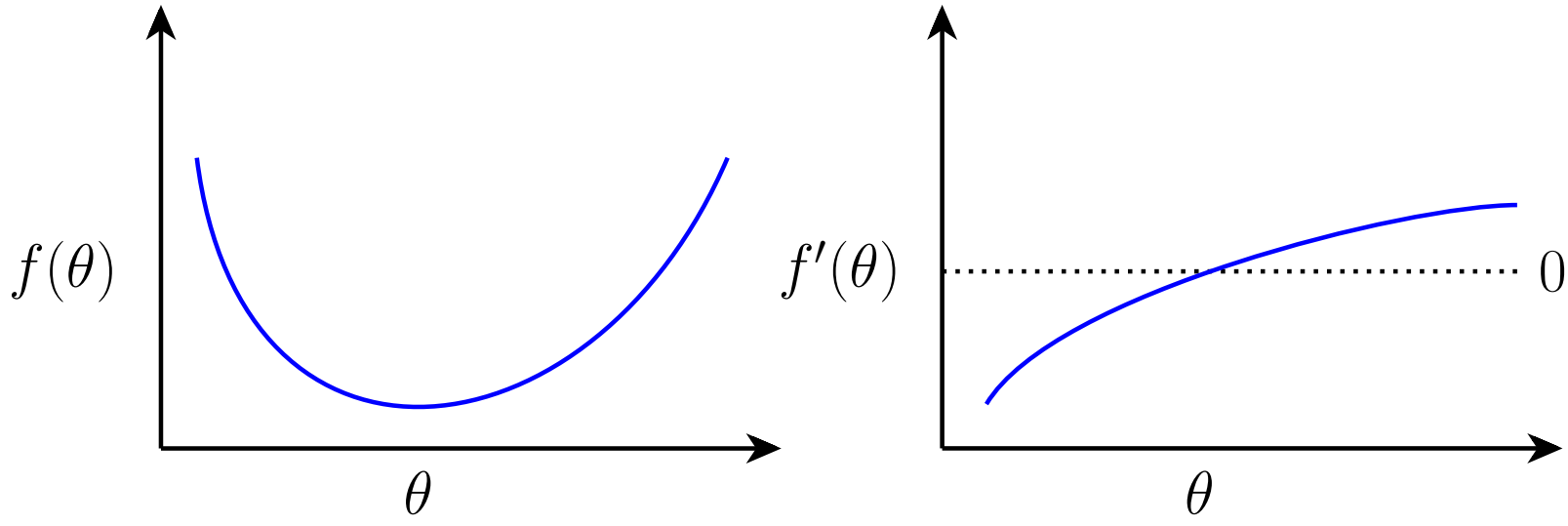
- In more abstract terms, let's consider the problem

$$\underset{\theta}{\text{minimize}} \; f(\theta)$$

  where $f : \mathbb{R}^n \to \mathbb{R}$ is a real-valued function

- How do we find a good value of θ?

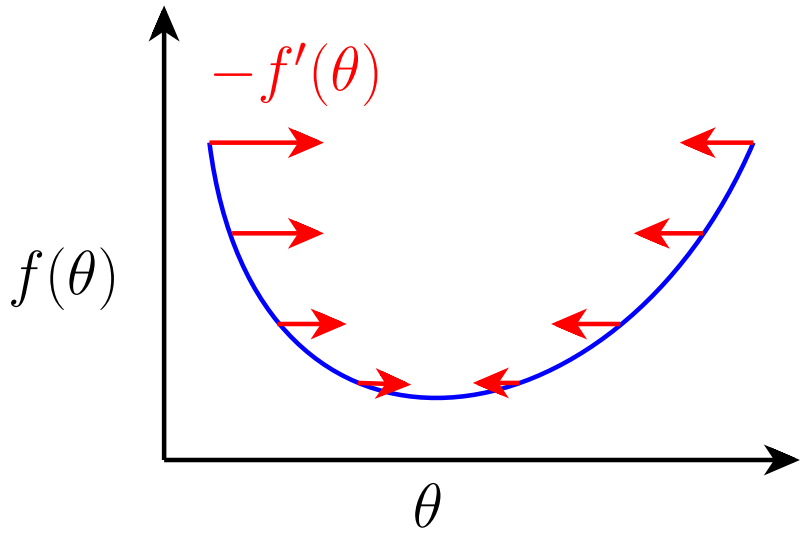
- An example for one-dimensional θ



- To find minimum point $\theta^\star$, we can look at the derivative of the function $f'(\theta)$: any location where $f'(\theta) = 0$ will be a "flat" point in the function

- For functions with certain properties (more on this in later lectures), this will be guaranteed to be a global minimum of the function

- The (negative) derivative has another useful property: it points in a "downhill" direction



- This motivates one of the most common optimization approaches, known as *gradient descent*

$$\text{Repeat:} \quad \theta := \theta - \alpha f'(\theta)$$

where α is some scaling factor (called a step size)

- For vector $\theta \in \mathbb{R}^n$, the analog of the derivative is called the *gradient*

$$\nabla_\theta f(\theta) \in \mathbb{R}^n = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_n} \end{bmatrix}$$

- The general gradient descent algorithm is the same as before, just using the gradient

$$\text{Repeat:} \quad \theta := \theta - \alpha \nabla_\theta f(\theta)$$

Return to linear regression

- Let's use these methods to solve our optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \left({x^{(i)}}^T \theta - y^{(i)} \right)^2$$

- After some fairly straightforward calculus

$$\begin{aligned} \nabla_\theta \sum_{i=1}^{m} \left({x^{(i)}}^T \theta - y^{(i)} \right)^2 &= \sum_{i=1}^{m} \nabla_\theta \left({x^{(i)}}^T \theta - y^{(i)} \right)^2 \\ &= \sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right) \end{aligned}$$

- Gradient descent, repeat: $\theta := \theta - \alpha \sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right)$

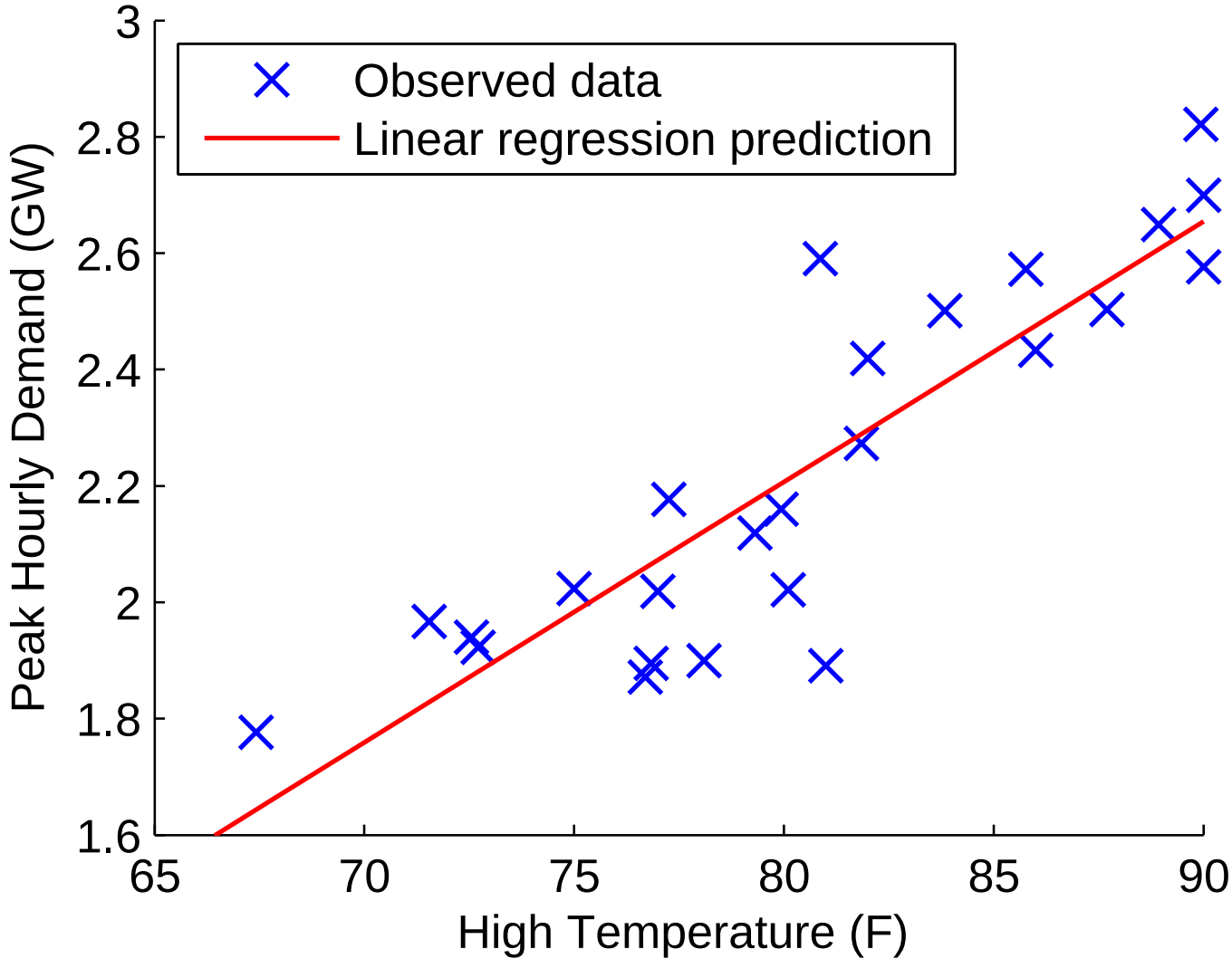
- In this case, we can also directly solve for $\nabla_\theta f(\theta) = 0$

$$\sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta^\star - y^{(i)} \right) = 0$$

$$\implies \left(\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T \right) \theta^\star = \sum_{i=1}^{m} x^{(i)} y^{(i)}$$

$$\implies \theta^\star = \left(\sum_{i=1}^{m} x^{(i)} {x^{(i)}}^T \right)^{-1} \left(\sum_{i=1}^{m} x^{(i)} y^{(i)} \right)$$

- Squared loss is one of the few cases that such directly solutions are possible, usually need to resort to gradient descent or other methods


Observed data
Linear regression prediction
Peak Hourly Demand (GW)
High Temperature (F)
3
2.8
2.6
2.4
2.2
2
1.8
1.6
65
70
75
80
85
90

Alternative loss functions

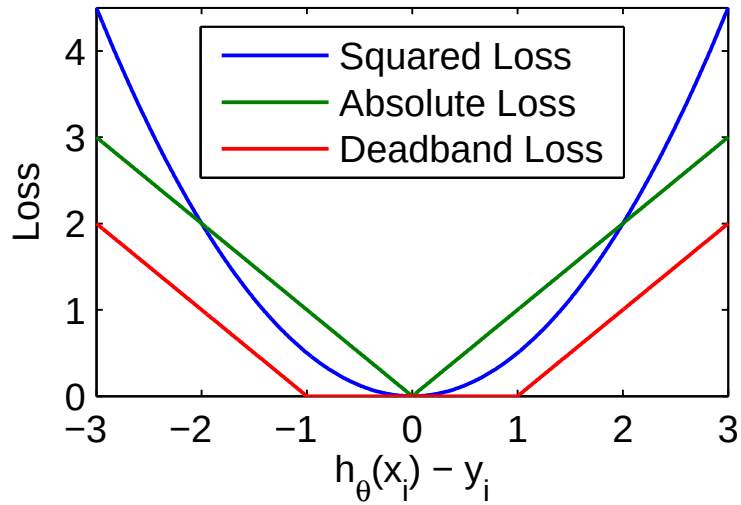
- Why did we choose the squared loss function

$$\ell\left(h_\theta(x), y\right) = \left(h_\theta(x) - y\right)^2?$$

- Some other alternatives

Absolute loss: $\ell(h_\theta(x), y) = |h_\theta(x) - y|$

Deadband loss: $\ell(h_\theta(x), y) = \max\{0, |h_\theta(x) - y| - \epsilon\},\ \epsilon \in \mathbb{R}_+$



- For these loss functions, no closed-form expression for $\theta^\star$, but gradient descent can still be very effective

- E.g., for absolute loss and linear hypothesis class

$$\text{Repeat : } \theta := \theta - \alpha \sum_{i=1}^{m} x^{(i)} \text{sign}\left({x^{(i)}}^T \theta - y^{(i)} \right)$$

- (Technically, absolute loss is not differentiable, so this is a method called *subgradient descent*)

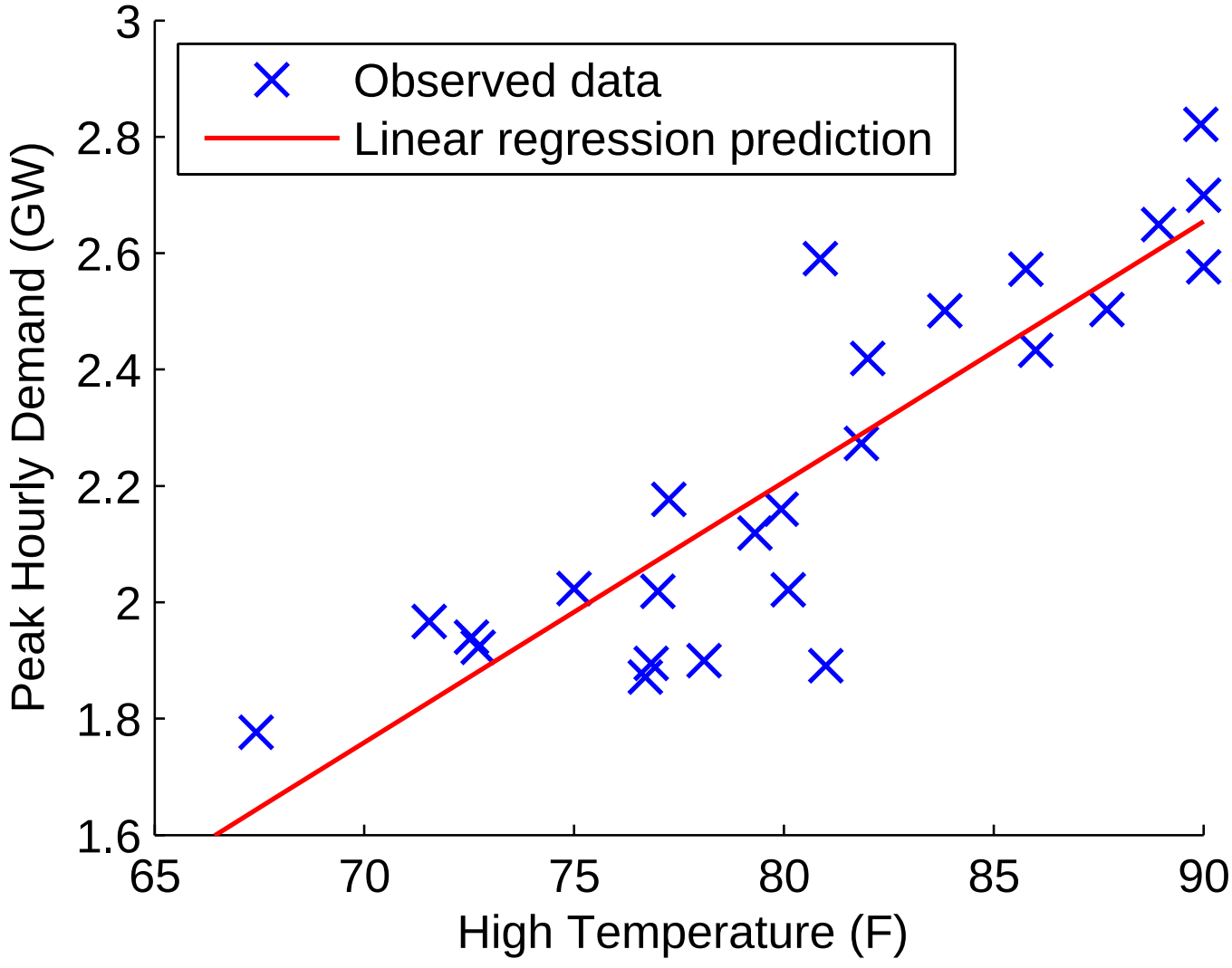
Outline

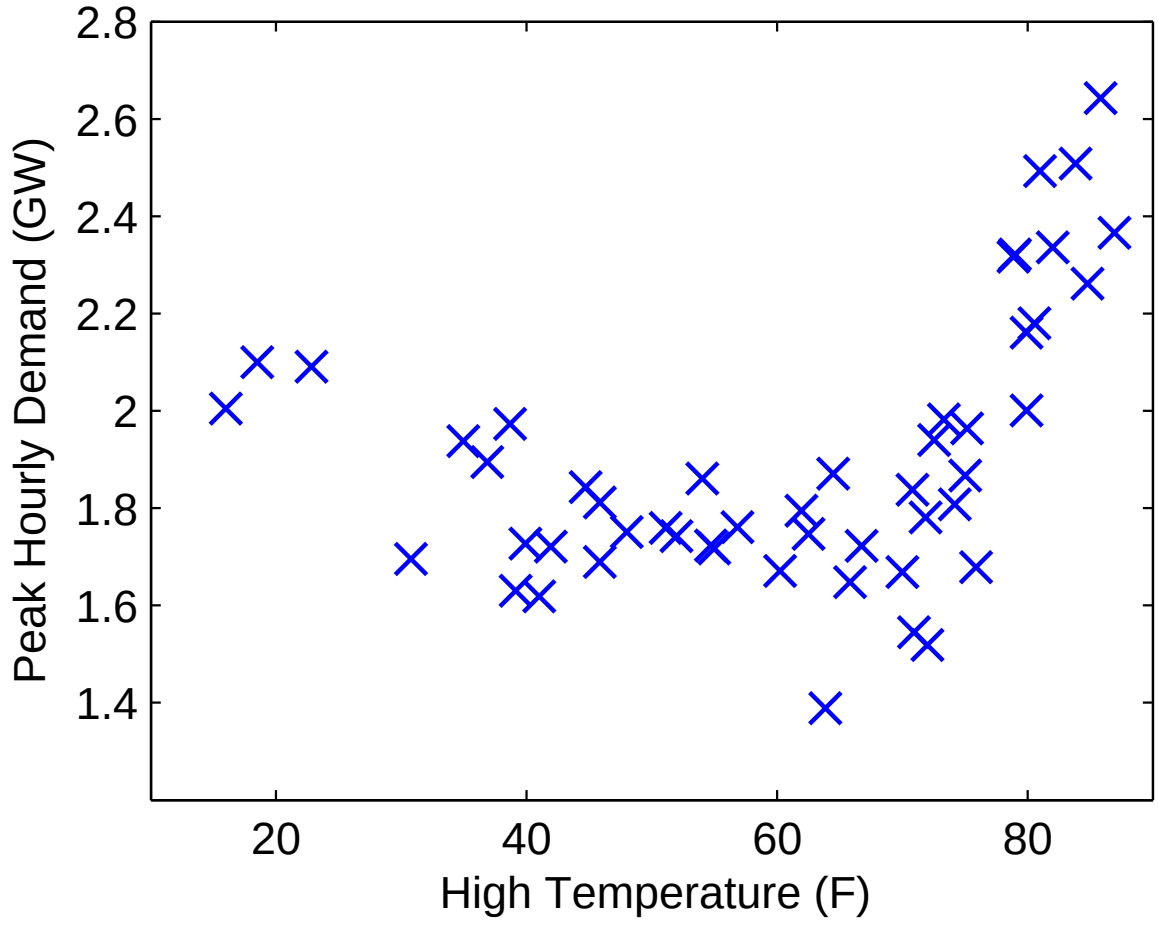
What is machine learning?

Supervised learning: regression

"Non-linear" regression, overfitting, and model selection

Supervised learning: classification


Observed data
Linear regression prediction
Peak Hourly Demand (GW)
High Temperature (F)
3
2.8
2.6
2.4
2.2
2
1.8
1.6
65
70
75
80
85
90



Several days of peak demand vs. high temperature in Pittsburgh over all months

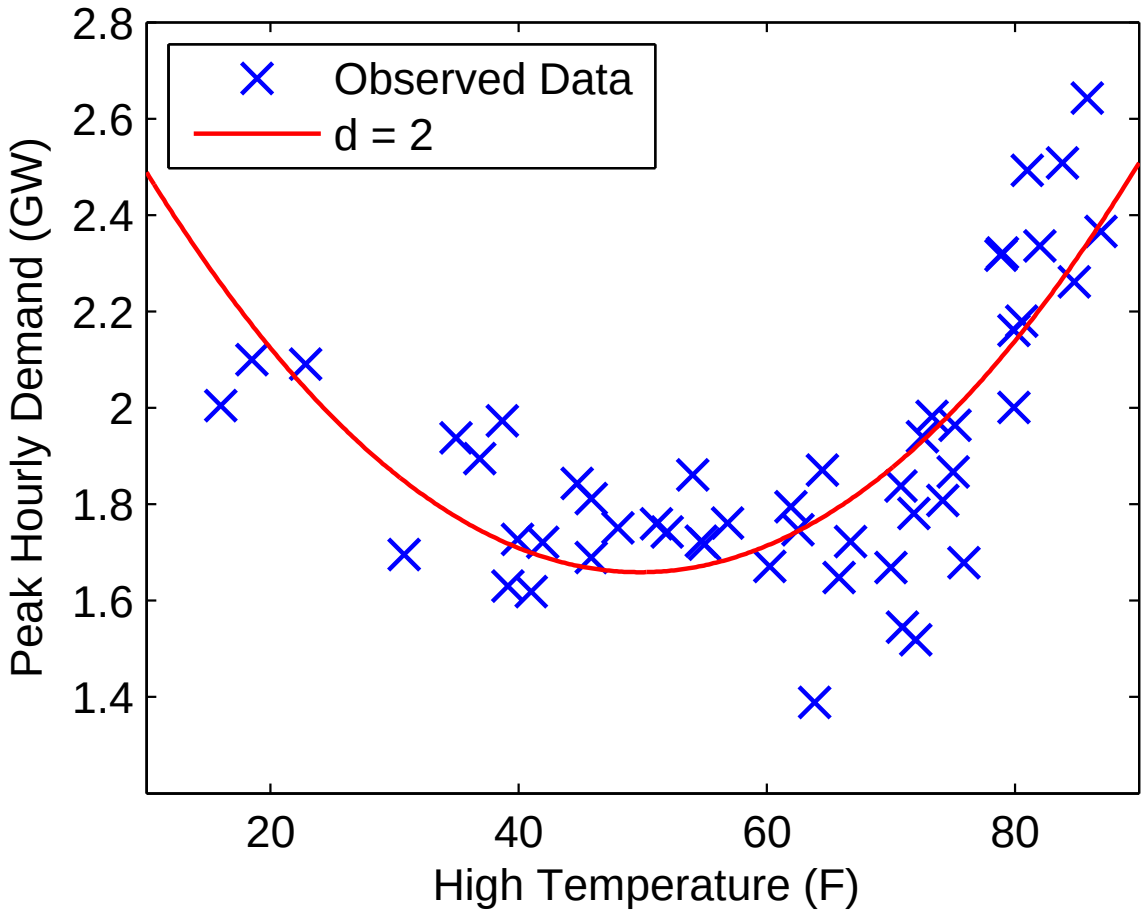
Overfitting

- Though they may seem limited, linear hypothesis classes are very powerful, since the input features can themselves include non-linear features of data

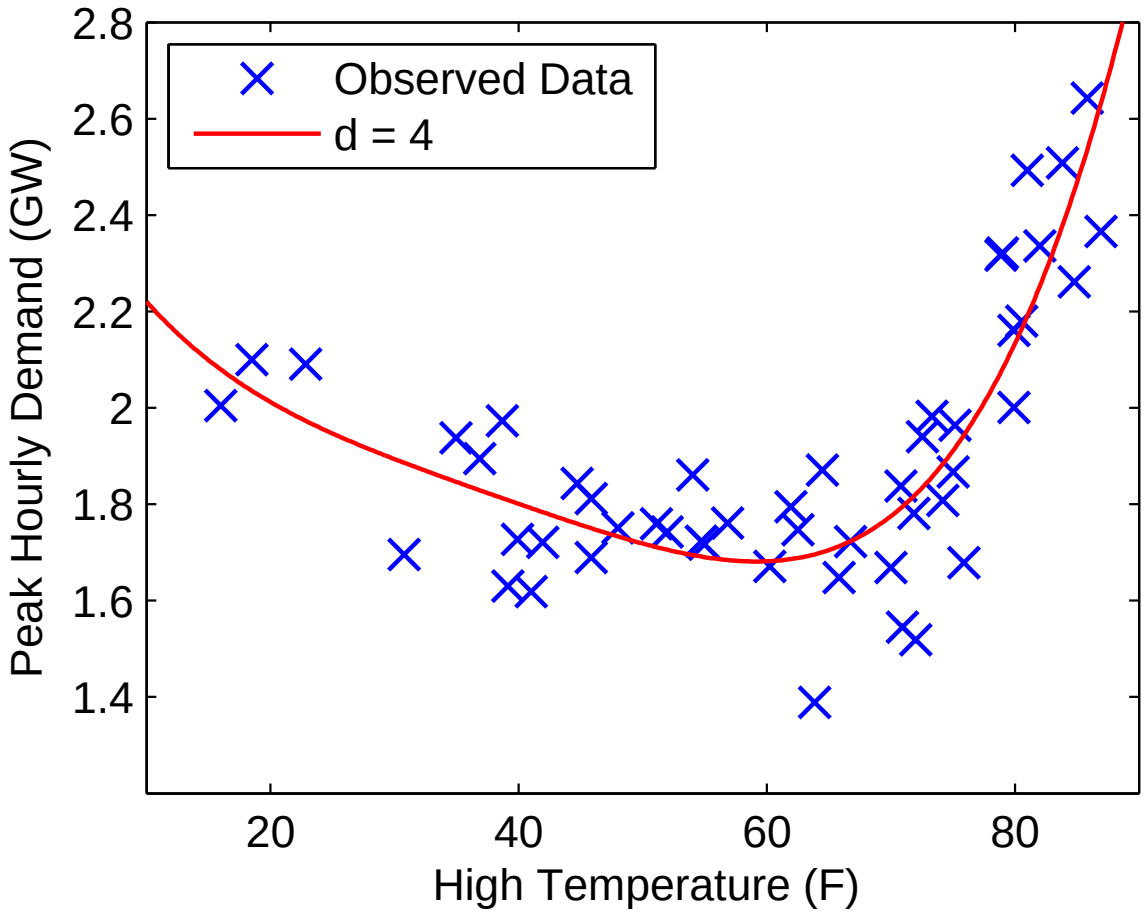
$$x^{(i)} \in \mathbb{R}^3 = \begin{bmatrix} (\text{high temperature for day } i)^2 \\ \text{high temperature for day } i \\ 1 \end{bmatrix}$$

- In this case, $h_\theta(x) = x^T\theta$ will be a non-linear function of "original" data (i.e., predicted peak power is a a non-linear function of high temperature)

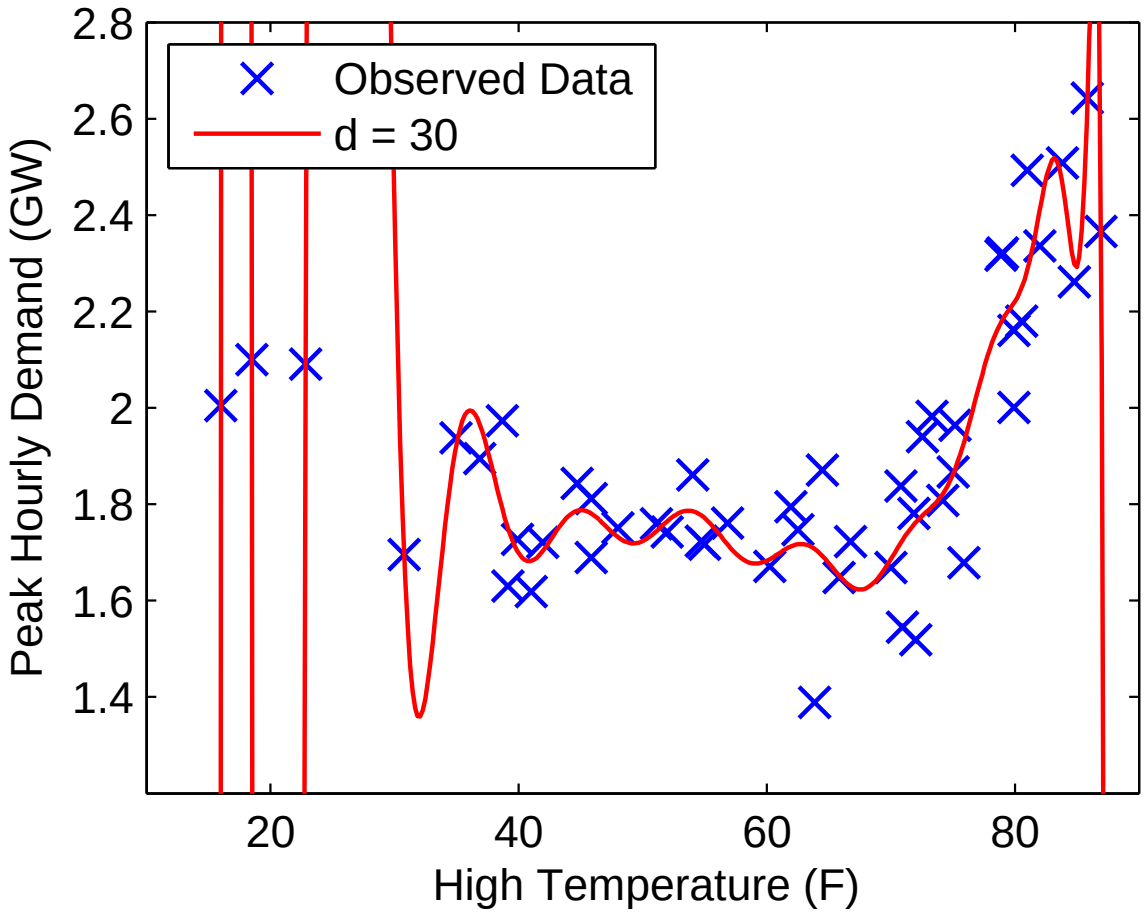
- Same solution method as before, gradient descent or (for squared loss) analytical solution



Linear regression with second degree polynomial features



Linear regression with fourth degree polynomial features



Linear regression with 30th degree polynomial features

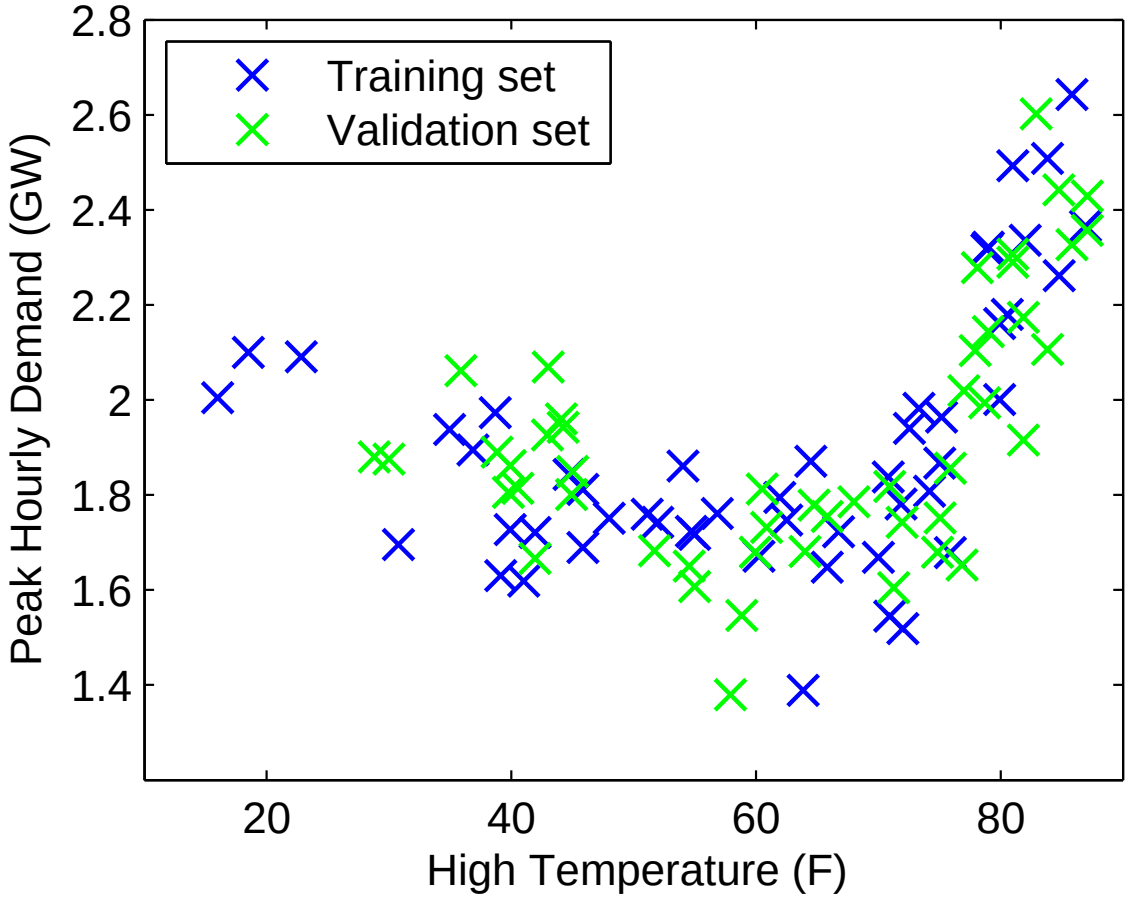
Training and validation loss

- Fundamental problem: we are looking for parameters that optimize

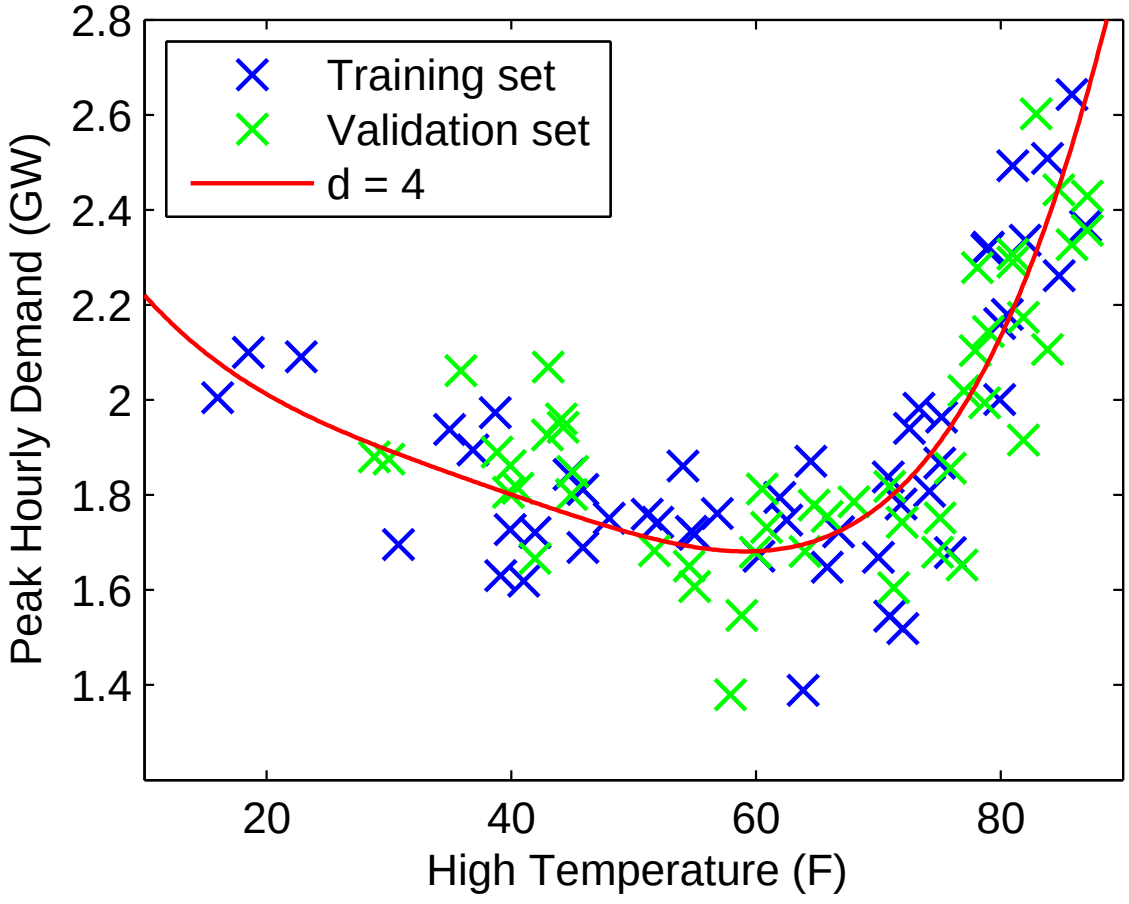
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

  but what we really care about is loss of prediction on *new* examples (x', y') (also called *generalization error*)

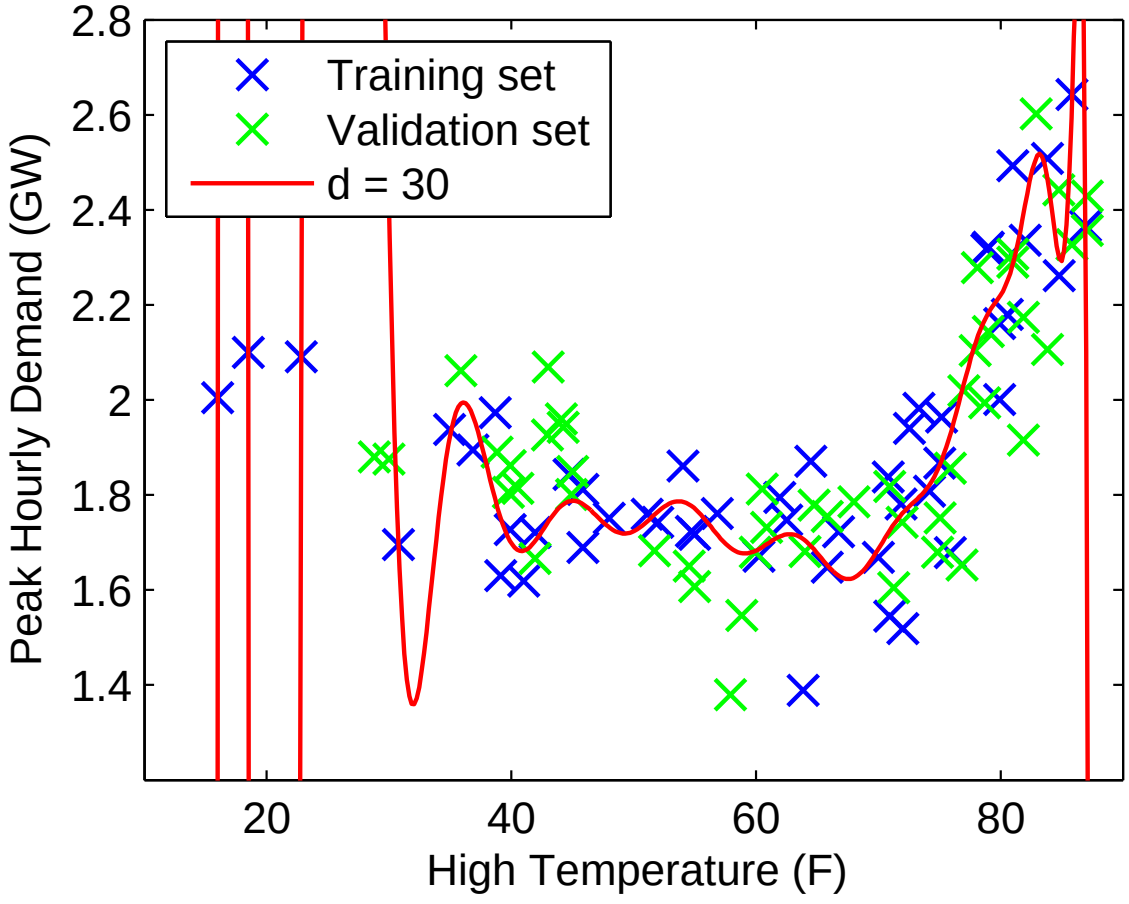
- Divide data into *training set* (used to find parameters for a fixed hypothesis class h_θ), and *validation set* (used to choose hypothesis class)
  - (Slightly abusing notation here, we're going to wrap the "degree" of the input features int the hypothesis class h_θ)



Training set and validation set

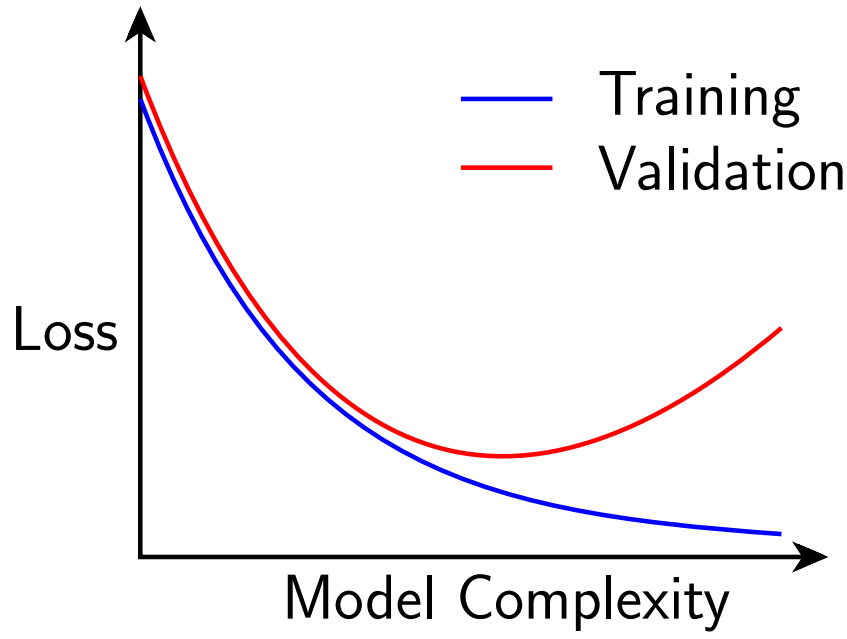


Training set and validation set, fourth degree polynomial

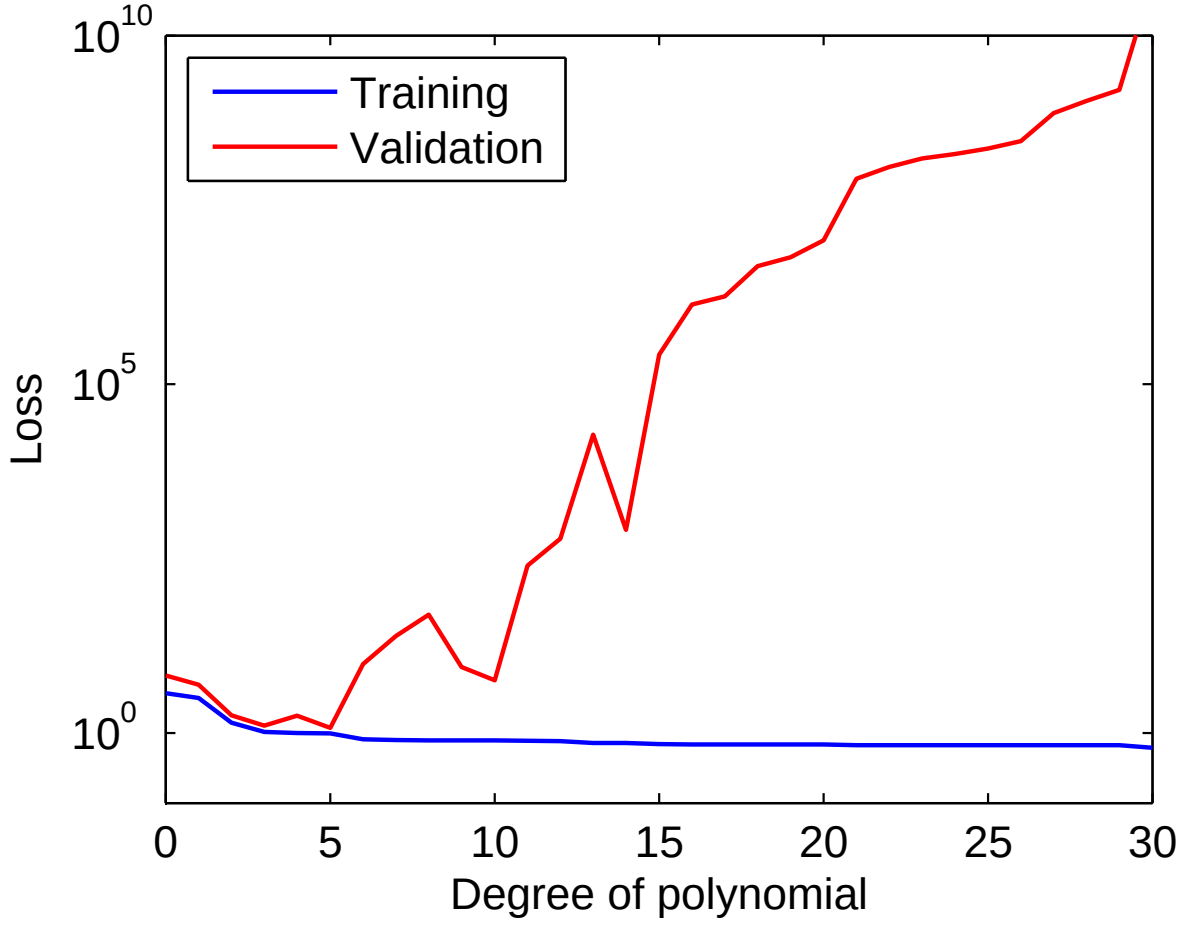


Training set and validation set, 30th degree polynomial

- General intuition for training and validation loss



- We would like to choose hypothesis class that is at the "sweet spot" of minimizing validation loss

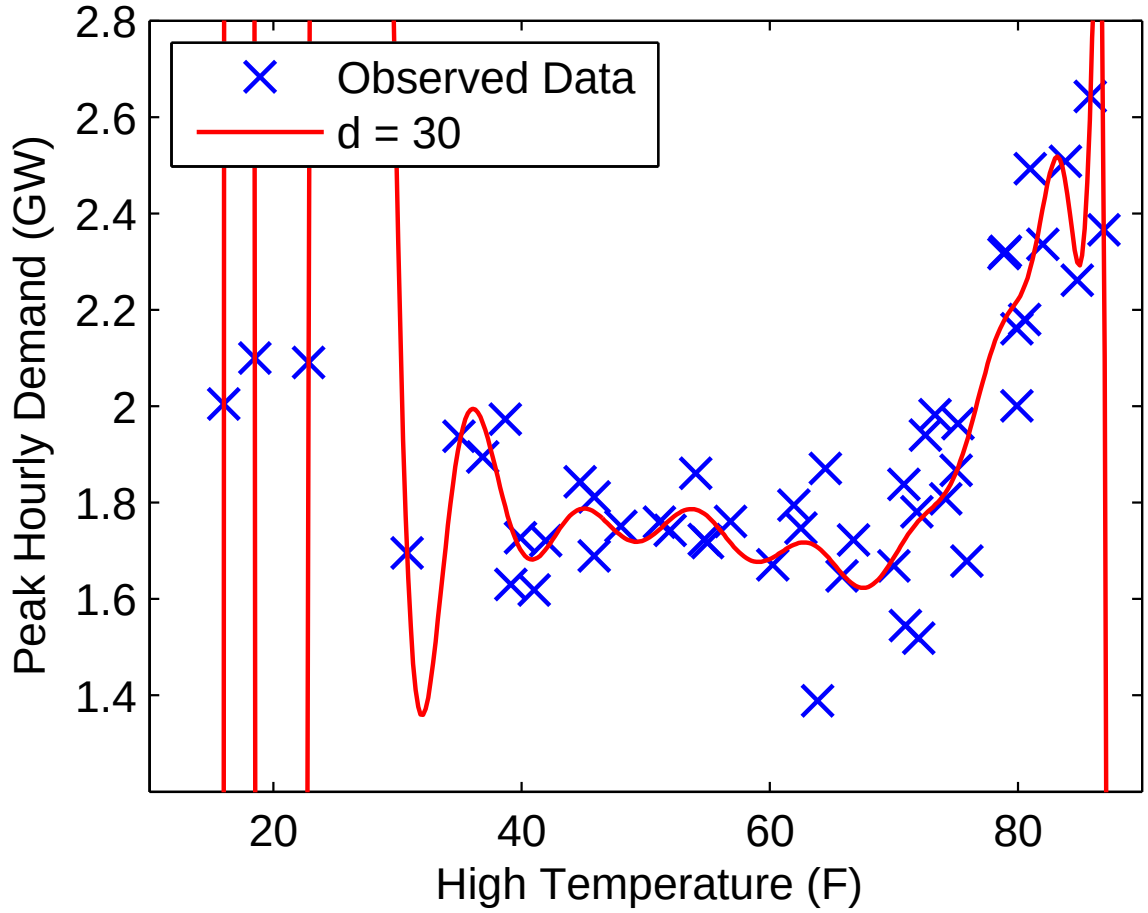


Training and validation loss on peak demand prediction

Model complexity and regularization

- A number of different ways to control "model complexity"
- An obvious one we have just seen: keep the number of features (number of parameters) low
- A less obvious method: keep the *magnitude* of the parameters small

- Intuition: a 30th degree polynomial that passes exactly through many of the data points requires very large entries in θ

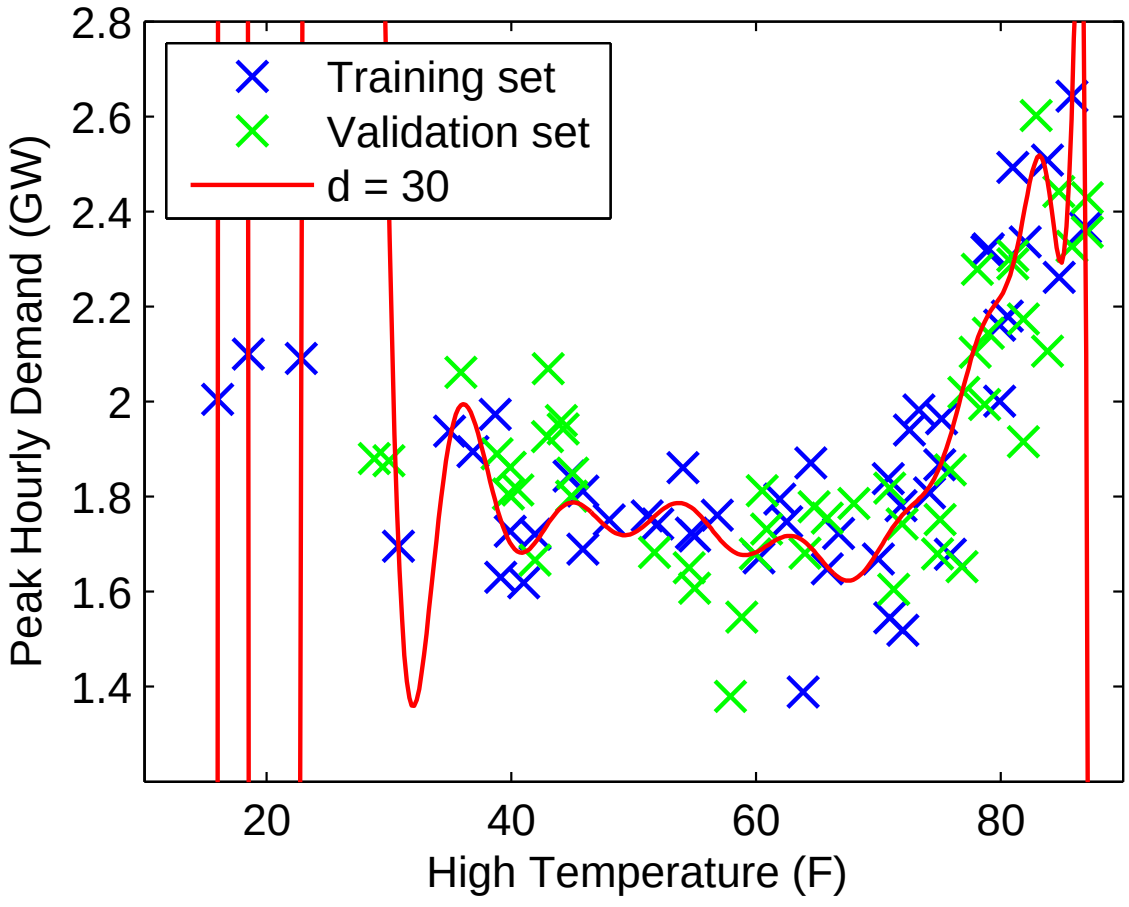


- We can directly prevent large entries in θ by penalizing the magnitude of its entries

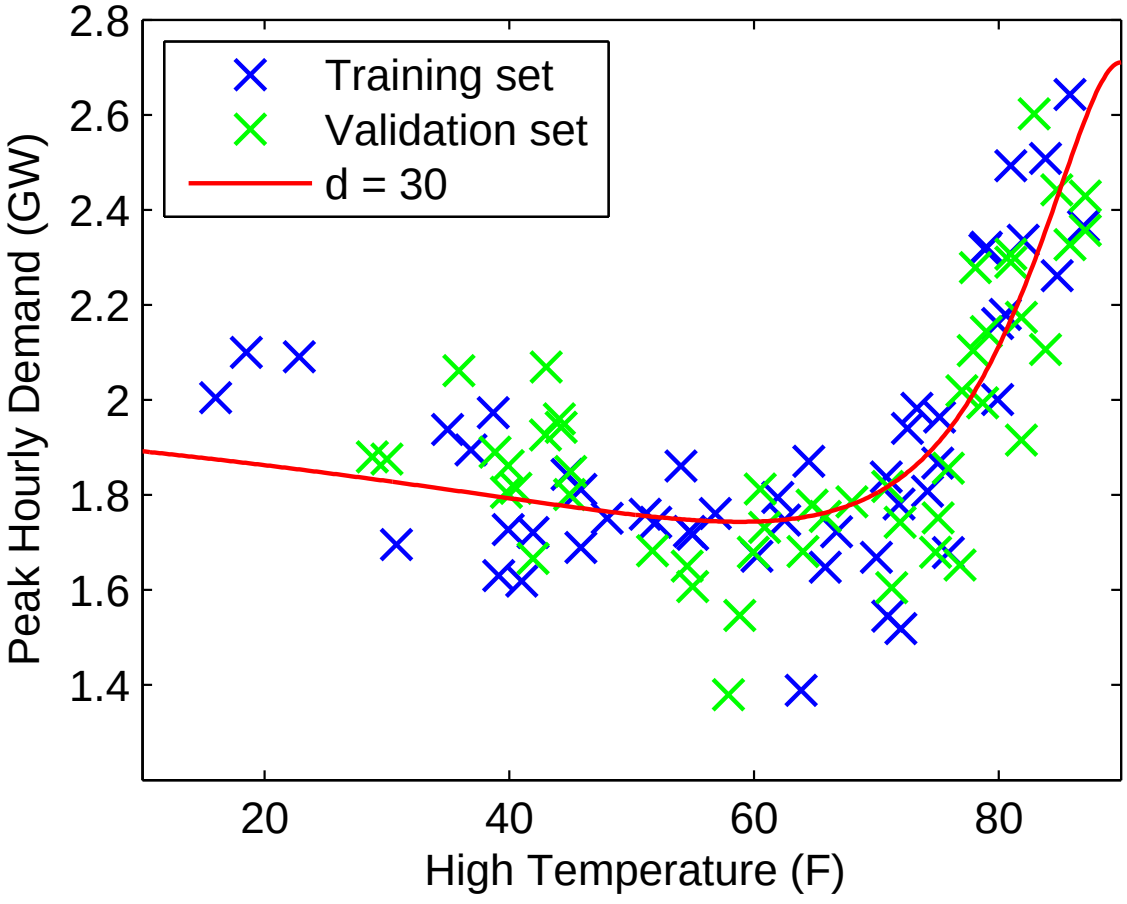
- Leads to *regularized loss minimization* problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell\left(h_\theta(x^{(i)}), y^{(i)}\right) + \lambda \sum_{i=1}^{n} \theta_i^2$$

  where $\lambda \in \mathbb{R}_+$ is a *regularization parameter* that weights the relative penalties of the size of θ and the loss



Degree 30 polynomial, with $\lambda = 0$ (unregularized)



Degree 30 polynomial, with $\lambda = 1$

Outline

What is machine learning?

Supervised learning: regression

"Non-linear" regression, overfitting, and model selection

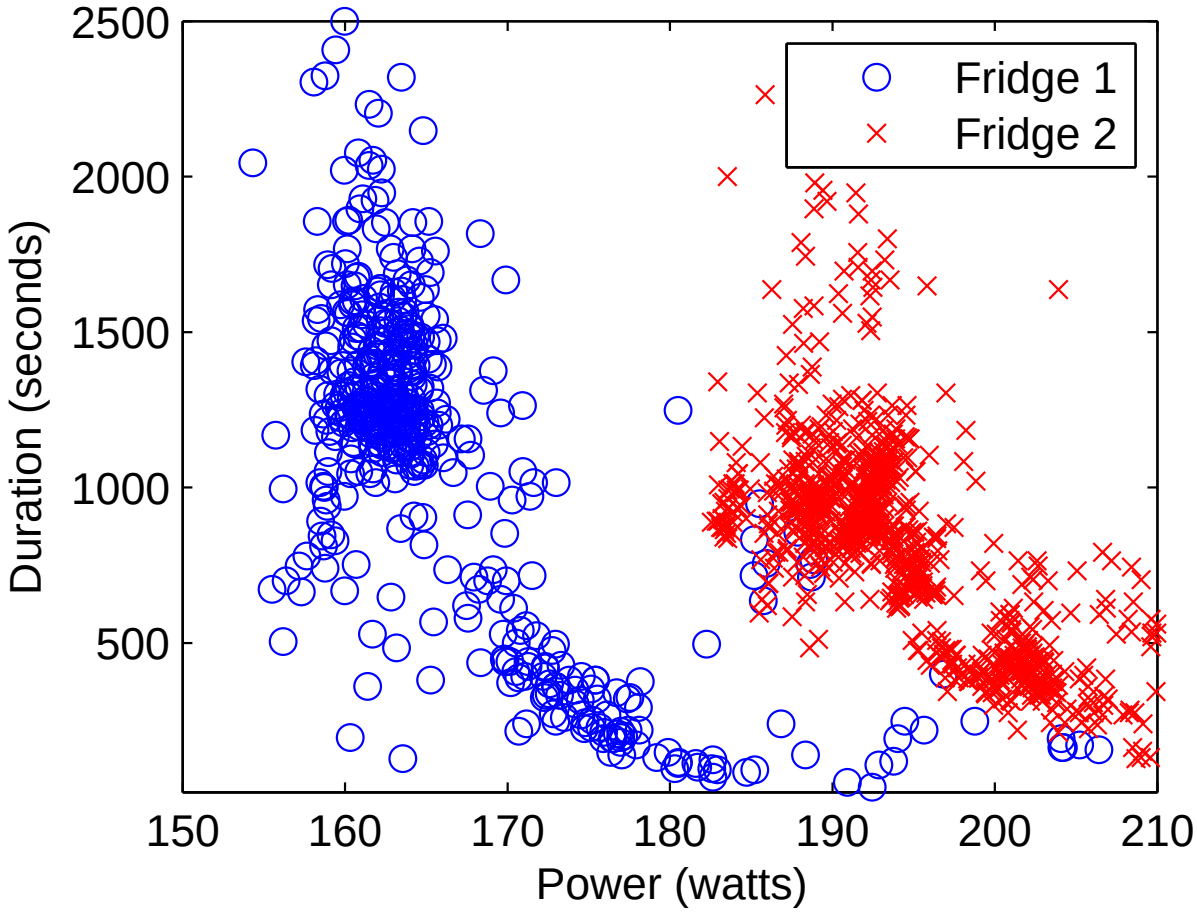
Supervised learning: classification

Classification problems

- Sometimes we want to predict discrete outputs rather than continuous
- Is the email spam or not? (YES/NO)
- What digit is in this image? (0/1/2/3/4/5/6/7/8/9)

Example: classifing household appliances

- Differentiate between two refrigerators using their power consumption signatures



Notation

- **Input features**: $x^{(i)} \in \mathbb{R}^n, \;\; i = 1, \ldots, m$
  - E.g.: $x^{(i)} \in \mathbb{R}^{784} = (\text{Duration } i, \text{Power } i, 1)$

- **Output**: $y^{(i)} \in \{-1, +1\}$ (binary classification task)
  - E.g.: $y^{(i)} =$ Is it fridge 1?

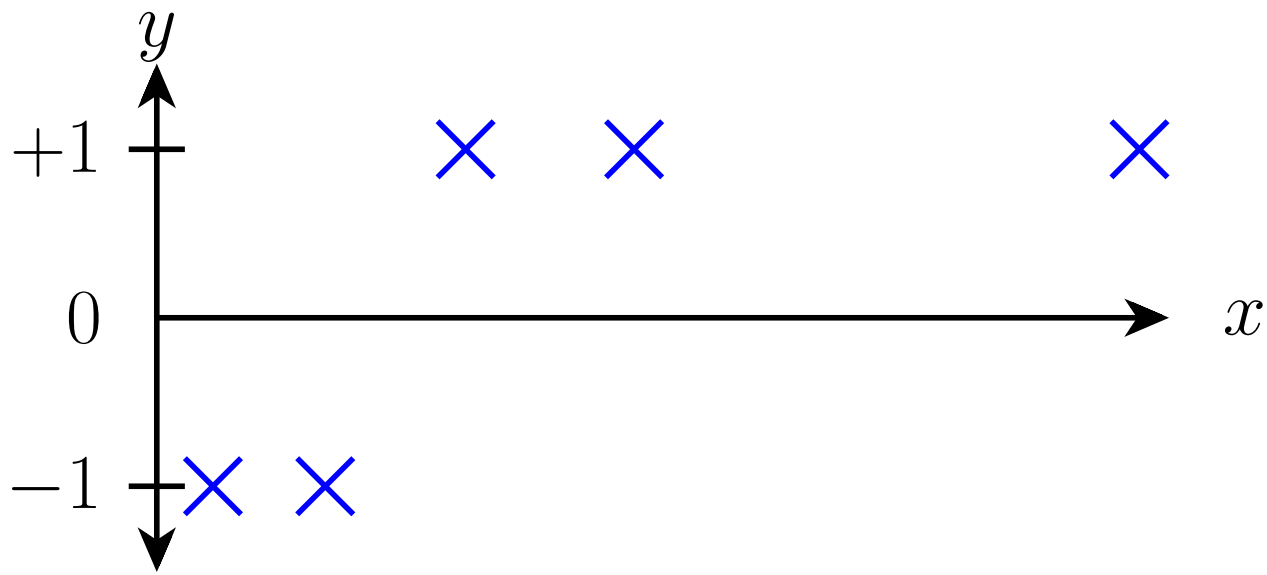
- **Model Parameters**: $\theta \in \mathbb{R}^n$

- **Hypothesis function**: $h_\theta(x) : \mathbb{R}^n \to \mathbb{R}$
  - Returns *continuous* prediction of the output y, where the value indicates how "confident" we are that the example is -1 or $+1$; $\text{sign}(h_\theta(x))$ is the actual binary prediction

  - Again, we will focus initially on *linear predictors* $h_\theta(x) = x^T\theta$

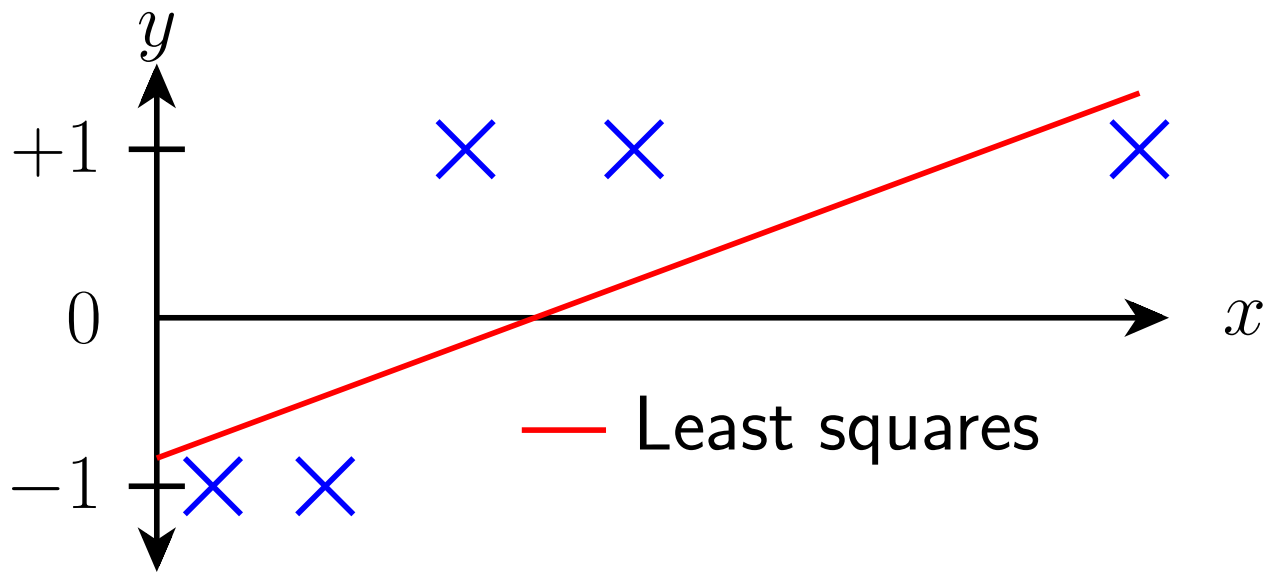
Loss functions

- Loss function $\ell : \mathbb{R} \times \{-1, +1\} \to \mathbb{R}_+$
- Do we need a different loss function?



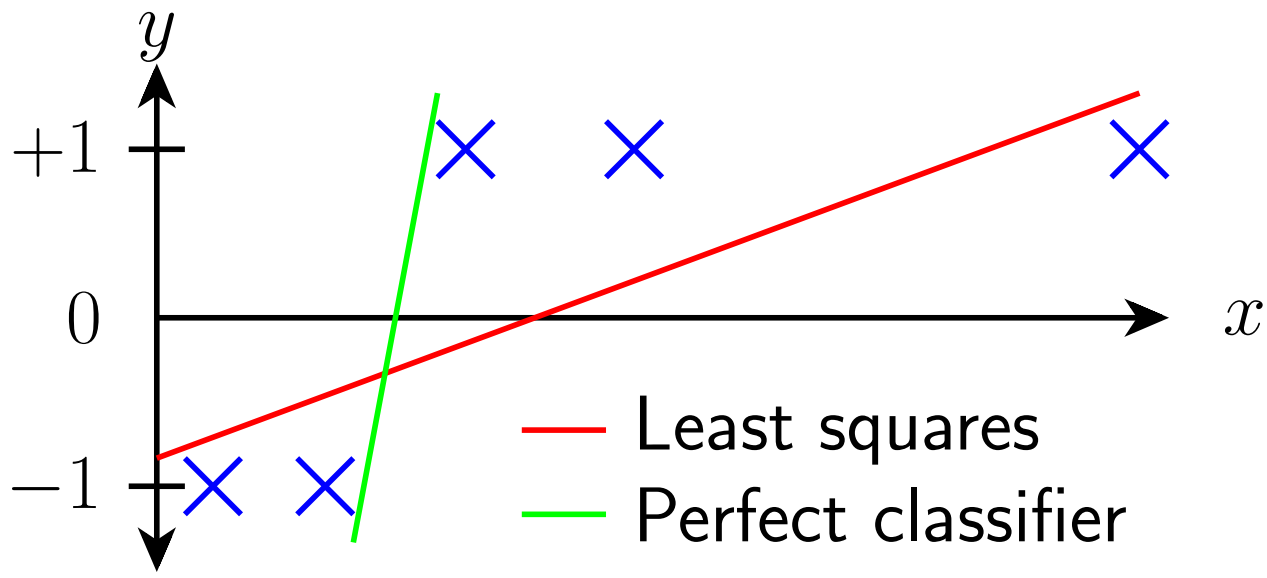
Loss functions

- Loss function $\ell : \mathbb{R} \times \{-1, +1\} \to \mathbb{R}_+$
- Do we need a different loss function?



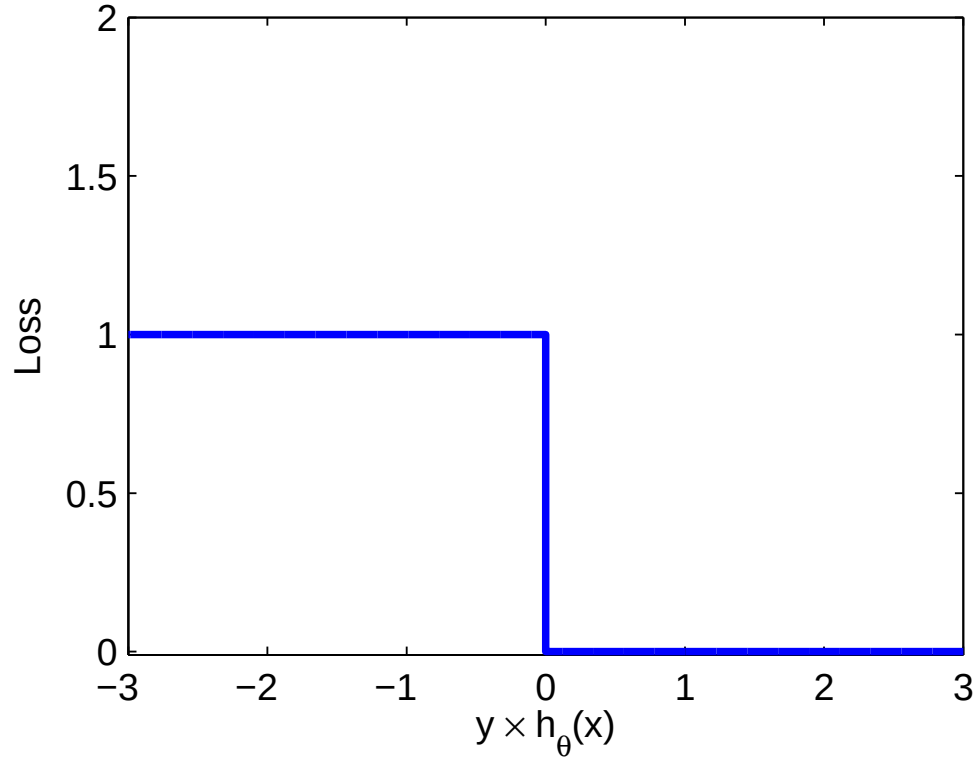
Loss functions

- Loss function $\ell : \mathbb{R} \times \{-1, +1\} \to \mathbb{R}_+$
- Do we need a different loss function?



- The simplest loss (0/1 loss, accuracy): count the number of mistakes we make

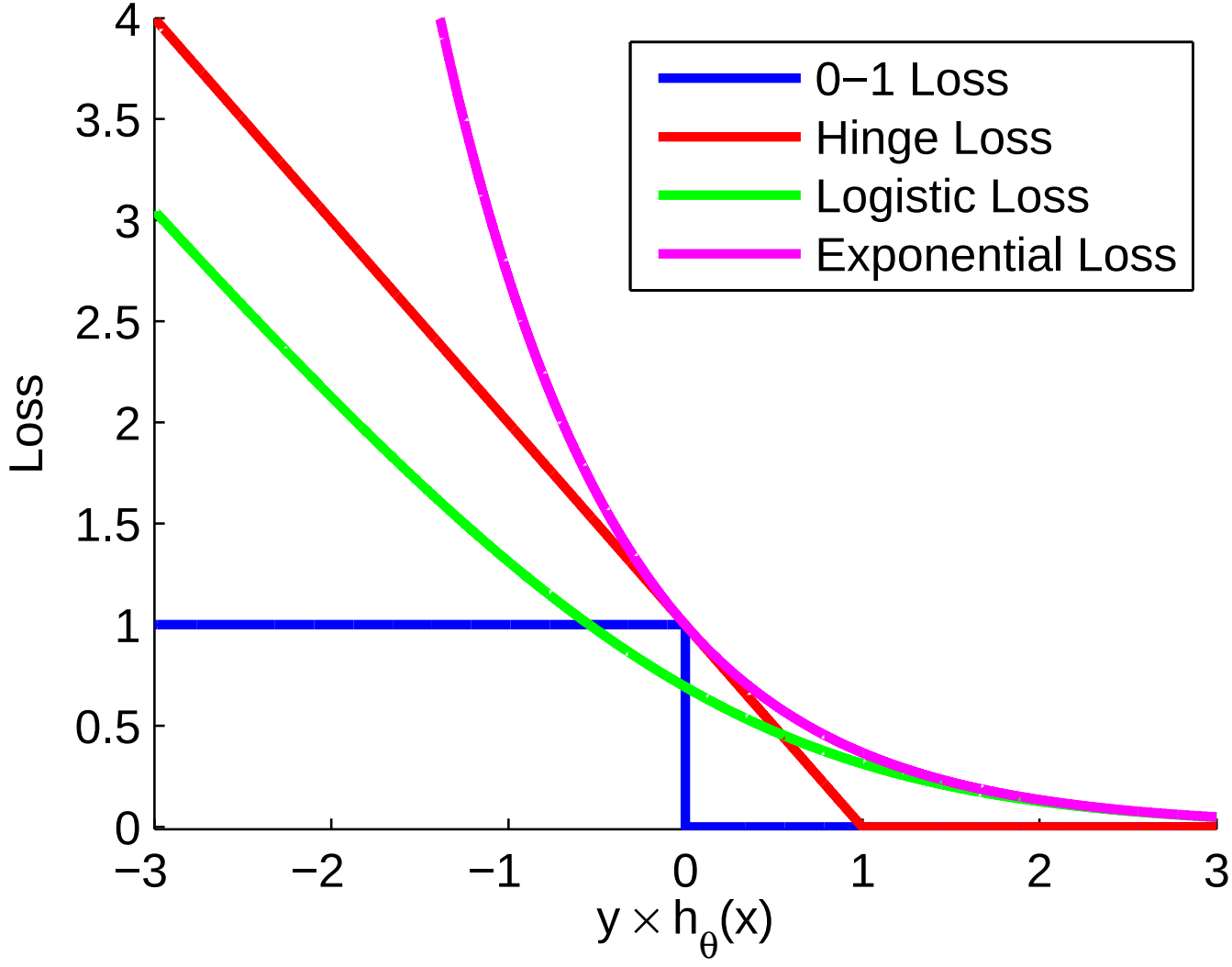
$$\ell(h_\theta(x), y) = \begin{cases} 1 & \text{if } y \neq \text{sign}(h_\theta(x)) \\ 0 & \text{otherwise} \end{cases}$$
$$= \mathbf{1}\{y \cdot h_\theta(x) \leq 0\}$$



- Unfortunately, minimizing sum of 0/1 losses leads to a hard optimization problem

- Because of this, a whole range of alternative "approximations" to 0/1 loss are used instead

$$\begin{aligned}
\text{Hinge loss:} \quad & \ell(h_\theta(x), y) = \max\{1 - y \cdot h_\theta(x), 0\} \\
\text{Squared hinge loss:} \quad & \ell(h_\theta(x), y) = \max\{1 - y \cdot h_\theta(x), 0\}^2 \\
\text{Logistic loss:} \quad & \ell(h_\theta(x), y) = \log(1 + e^{-y \cdot h_\theta(x)}) \\
\text{Exponential loss:} \quad & \ell(h_\theta(x), y) = e^{-y \cdot h_\theta(x)}
\end{aligned}$$



Common loss functions for classification

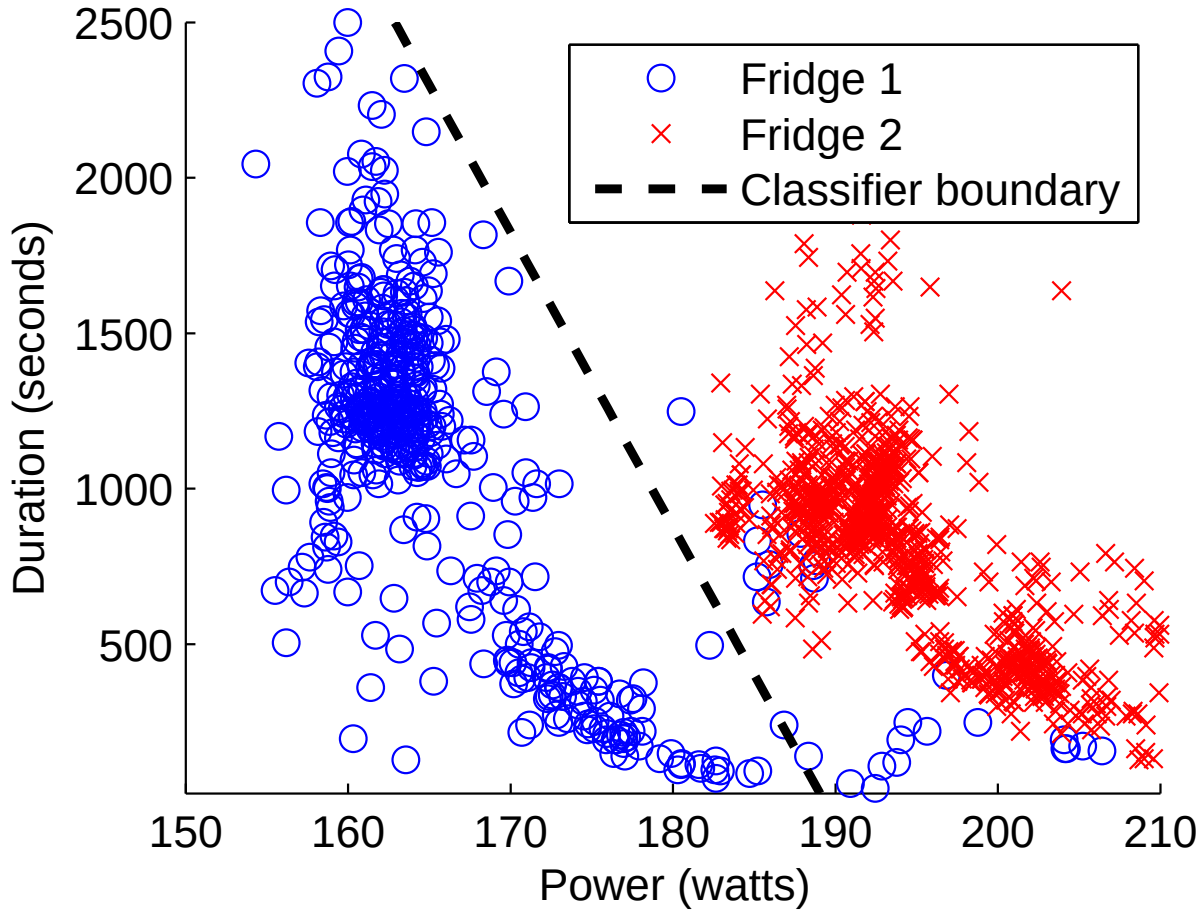
Support vector machines

- Support vector machine is just regularized hinge loss and linear prediction (caveat, also common to use "kernel" hypothesis function, more later)

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \max\{1 - y^{(i)} \cdot {x^{(i)}}^T \theta, 0\} + \lambda \sum_{i=1}^{n} \theta_i^2$$

- Gradient descent update, repeat:

$$\theta := \theta - \alpha \left(- \sum_{i=1}^{m} y^{(i)} x^{(i)} \mathbf{1}\{y^{(i)} \cdot {x^{(i)}}^T \theta < 1\} + 2\lambda \sum_{i=1}^{n} \theta_i \right)$$



Classification boundary of support vector machine

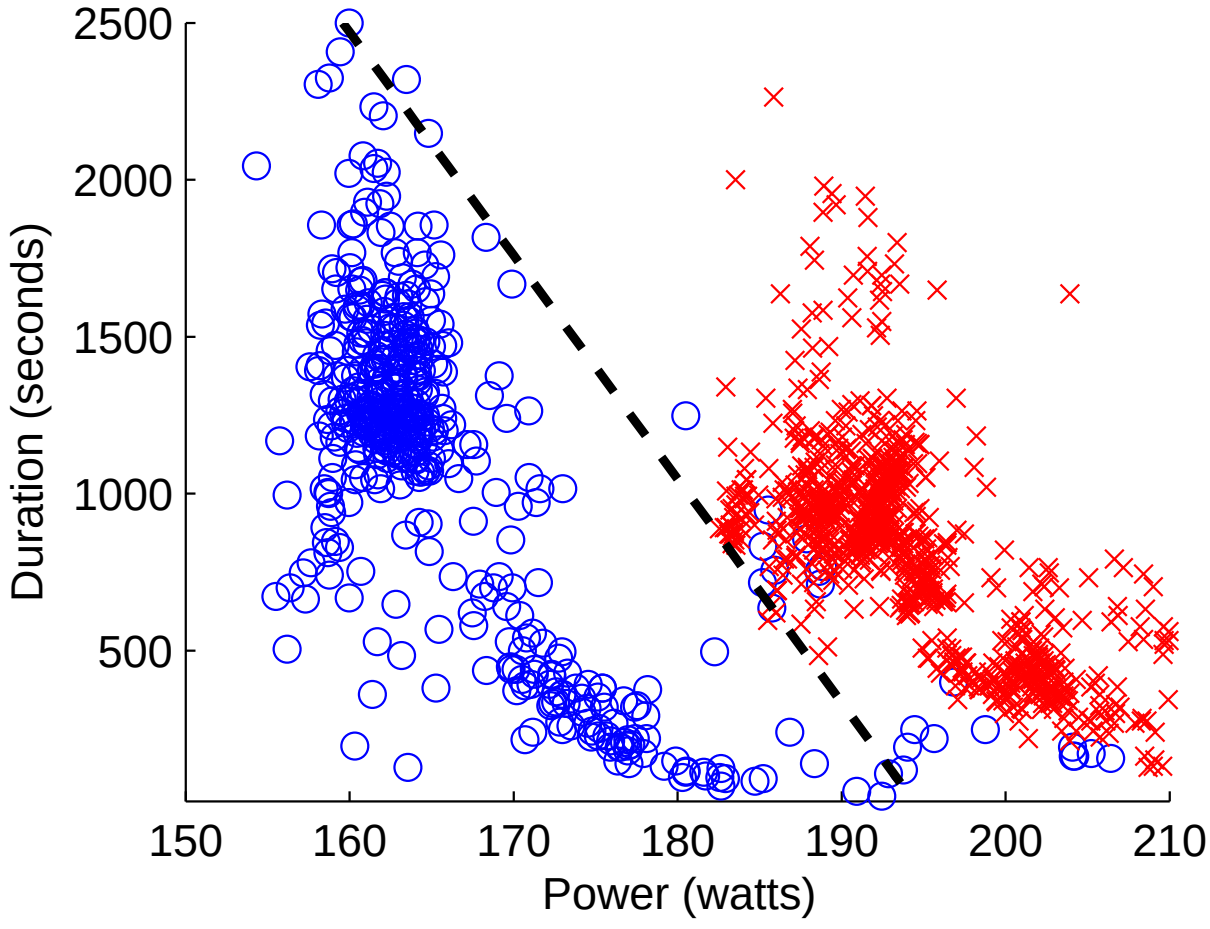
Logistic regression

- Logistic regression uses logistic loss

$$\underset{\theta}{\text{minimize}} \; + \sum_{i=1}^{m} \log(1 + e^{-y \cdot x^{(i)^T} \theta}) + \lambda \sum_{i=1}^{n} \theta_i^2$$

- Probabilistic interpretation: $p(y^{(i)} = +1 | x^{(i)}) = \frac{1}{1+\exp\{-x^{(i)^T}\theta\}}$

- Again, gradient descent is a reasonable algorithm (can you compute an equation for the gradient?)



Classification boundary of logistic regression

Multi-class classification

- When classification is not binary $y \in 0, 1, \ldots, k$ (i.e., classifying digit images), a common approach is "one-vs-all" method

- Create a new set of y's for the binary classification problem "is the label of this example equal to j"

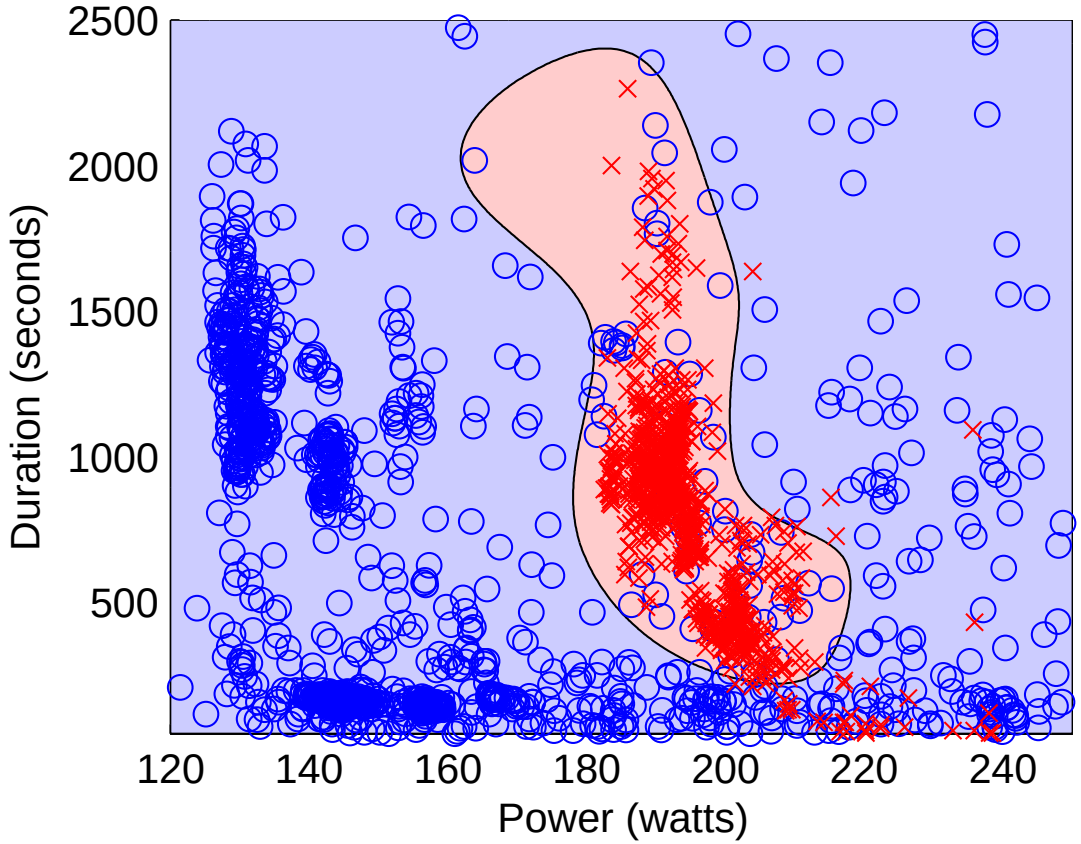
$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } y^{(i)} = j \\ -1 & \text{otherwise} \end{cases}$$

  and solve for the corresponding parameter θ^j

- For input x, classify according to the hypothesis with the highest confidence: $\operatorname{argmax}_j h_{\theta^j}(x)$

Non-linear classification

- Same exact approach as in the regression case: use non-linear features of input to capture non-linear decision boundaries



Classification boundary of support vector machine using non-linear features

Stochastic gradient descent

- One last point (since it underlies a lot of the big data methods we'll see in this course)

- Most of the losses and gradients we've seen so far, involve summing over *all* examples, not practical if there are billions of them

- A common starting point for "big data" algorithms is *stochastic gradient descent*, perform gradient updates for just one example at a time

$$\theta := \theta - \alpha \nabla_\theta \ell \left(h_\theta(x^{(i)}), y^{(i)} \right), \quad i = 1, \ldots, m$$