15-388/688 - Practical Data Science: Linear regression

J. Zico Kolter
Carnegie Mellon University
Fall 2016

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Implementation

Announcements

HW2 has been pushed back until tonight midnight, no late days after deadline

HW3 out, due next Wednesday (late days as normal)

Feedback on tutorial, additional office hours to discuss today from 2:30-3:30, GHC 8102 (we will have more on Wednesday)

688 -> 388 switches email me, need to fill out a form

Dijkstra's algorithm…

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Implementation

A simple example: predicting electricity use

What will peak power consumption be in Pittsburgh tomorrow?

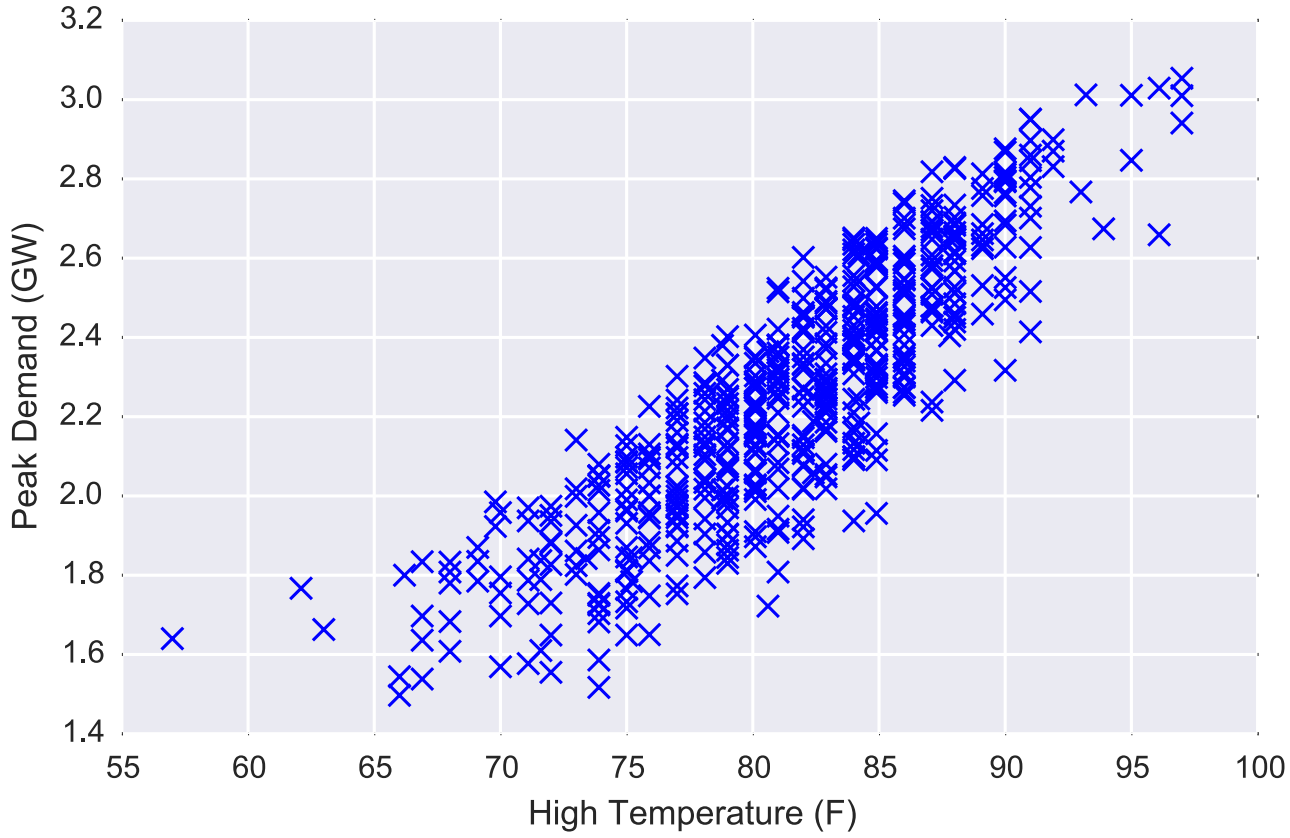
Difficult to build an “a priori” model from first principles to answer this question

But, relatively easy to record past days of consumption, plus additional features that affect consumption (i.e., weather)

Date	High Temperature (F)	Peak Demand (GW)
2011-06-01	84.0	2.651
2011-06-02	73.0	2.081
2011-06-03	75.2	1.844
2011-06-04	84.9	1.959
…	…	…

Plot of consumption vs. temperature

Plot of high temperature vs. peak demand for summer months (June – August) for past six years



Hypothesis: linear model

Let's suppose that the peak demand approximately fits a *linear model*

$$\text{Peak-Demand} \approx \theta_1 \cdot \text{High-Temperature} + \theta_2$$

Here θ_1 is the "slope" of the line, and θ_2 is the intercept

How do we find a "good" fit to the data?

Many possibilities, but natural objective is to minimize some difference between this line and the observed data, e.g. squared loss

$$\sum_{i \in \text{days}} (\theta_1 \cdot \text{High-Temperature}^{(i)} + \theta_2 \ - \text{Peak-Demand}^{(i)})^2$$

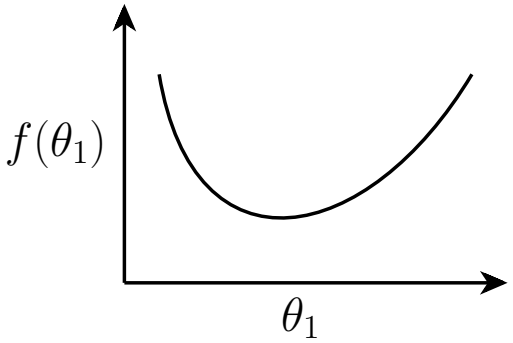
How do we find parameters?

How do we find the parameters θ_1, θ_2 that minimize the function

$$\sum_{i \in \text{days}} (\theta_1 \cdot \text{High-Temperature}^{(i)} + \theta_2 \ - \text{Peak-Demand}^{(i)})^2$$

$$\equiv \sum_{i \in \text{days}} (\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})^2$$

General idea: suppose we want to minimize some function $f(\theta_1)$



Minimum occurs at point where *derivative* of f with respect to θ_1 is zero

Solving for best θ

We can compute the derivatives of our loss w.r.t. θ_1 and θ_2, set both equal to zero:

$$\frac{\partial}{\partial \theta_1}\left(\sum_{i \in \text{days}} (\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})^2\right) = 2 \sum_{i \in \text{days}} x^{(i)} \cdot (\ \theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})$$

$$\frac{\partial}{\partial \theta_2}\left(\sum_{i \in \text{days}} (\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})^2\right) = 2 \sum_{i \in \text{days}} (\ \theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})$$

Setting $\frac{\partial}{\partial \theta_2} = 0$ gives:

$$\theta_2 = -\frac{1}{|\text{days}|} \sum_{i \in \text{days}} (\ \theta_1 \cdot x^{(i)} - y^{(i)}) = -\theta_1 \cdot \bar{x} + \bar{y}$$

where $\bar{x}$ and $\bar{y}$ denote means (mean of high temperature / peak demand)

Solving for best θ

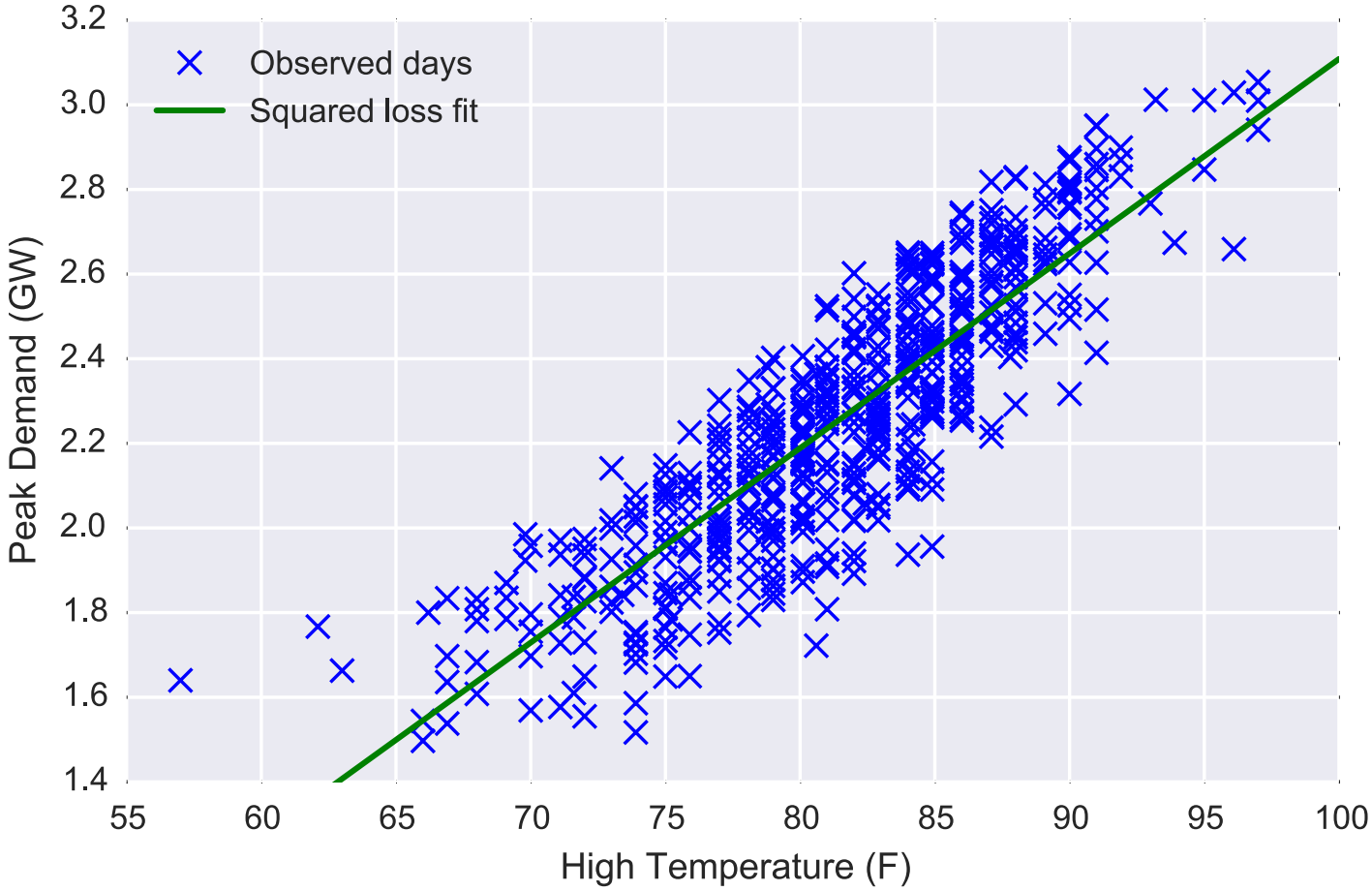
Plugging this in to first equation and solving gives

$$
\begin{aligned}
& 2 \sum_{i \in \text{days}} x^{(i)} \cdot \left(\theta_1 \cdot x^{(i)} - \theta_1 \cdot \bar{x} + \bar{y} - y^{(i)} \right) = 0 \\
& \Rightarrow \ \theta_1 \cdot \left(\sum_{i \in \text{days}} x^{(i)} \cdot \left(x^{(i)} - \bar{x} \right) \right) - \sum_{i \in \text{days}} x^{(i)} \cdot \left(y^{(i)} - \bar{y} \ \right) = 0 \\
& \Rightarrow \theta_1 = \frac{\sum_{i \in \text{days}} x^{(i)} \cdot \left(y^{(i)} - \bar{y} \right)}{\sum_{i \in \text{days}} x^{(i)} \cdot \left(x^{(i)} - \bar{x} \right)}
\end{aligned}
$$

For temperature / demand data we have:

$$\theta_1 = 0.046, \qquad \theta_2 = -1.489$$

Visualizing the fit



Making predictions

Importantly, our model also lets us make *predictions* about new days

What will the peak demand be tomorrow?

If we know the high temperature will be 72 degrees (ignoring for now that this is *also* a prediction), then we can predict peak demand to be:

$$\text{Predicted-demand} = \theta_1 \cdot 72 + \theta_2 = 1.821 \text{ GW}$$

Equivalent to just “finding the point on the line”

Extensions

What if we want to add additional features, e.g. day of week, instead of just temperature?

What if we want to use a different loss function instead of squared error (i.e., absolute error)?

What if we want to use a non-linear prediction instead of a linear one?

We can easily reason about all these things by adopting some additional notation…

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

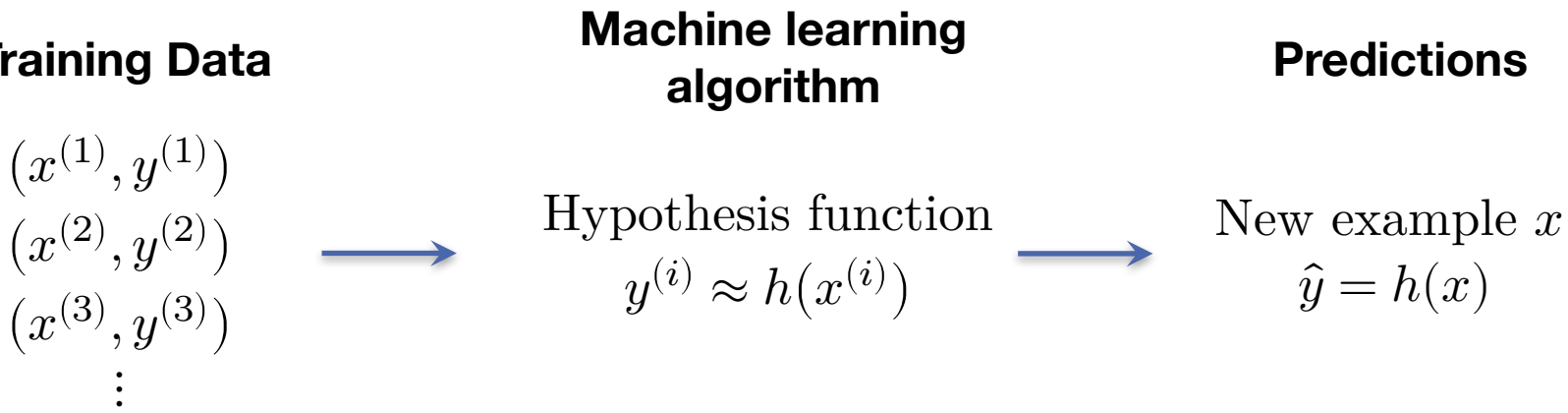
Implementation

Machine learning

This has been an example of a *machine learning algorithm*

Basic idea: in many domains, it is difficult to hand-build a predictive model, but easy to collect lots of data; machine learning provides a way to automatically infer the predictive model from data

The basic process (supervised learning):



Terminology

Input features: $x^{(i)} \in \mathbb{R}^n, i = 1, \ldots, m$

$$\text{E. g.: } x^{(i)} = \begin{bmatrix} \text{High-Temperature}^{(i)} \\ \text{Is-Weekday}^{(i)} \\ 1 \end{bmatrix}$$

Outputs: $y^{(i)} \in \mathcal{Y}, i = 1, \ldots, m$

$$\text{E. g.: } y^{(i)} \in \mathbb{R} = \text{ Peak-Demand}^{(i)}$$

Model parameters: $\theta \in \mathbb{R}^n$

Hypothesis function: $h_\theta : \mathbb{R}^n \to \mathcal{Y}$, predicts output given input

$$\text{E. g.: } h_\theta(x) = \theta^T x = \sum_{j=1}^{n} \theta_j \cdot x_j$$

Terminology

Loss function: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, measures the difference between a prediction and an actual output

$$\text{E. g.: } \ell(\hat{y}, y) = (\hat{y} - y)^2$$

The canonical machine learning optimization problem:

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

Virtually every machine learning algorithm has this form, just specify

1. What is the hypothesis function?
2. What is the loss function?
3. How do we solve the optimization problem?

Example machine learning algorithms

Note: we (machine learning researchers) have *not* been consistent in naming conventions, many machine learning algorithms actually only specify some of these three elements

Least squares: {linear hypothesis, squared loss, (usually) analytical solution}

Linear regression: {linear hypothesis, *, *}

Support vector machine: {linear or kernel hypothesis, hinge loss, *}

Neural network: {Composed non-linear function, *, (usually) gradient descent)

Decision tree: {Hierarchical axis-aligned halfplanes, *, greedy optimization}

Naïve Bayes: {Linear hypothesis, joint probability under certain independent assumptions, computing counts}

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Implementation

Least squares revisited

Using our new terminology, plus matrix notion, let's revisit how to solve linear regression with a squared error loss

Setup:

Linear hypothesis function: $h_\theta(x) = \theta^T x$

Squared error loss: $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

Resulting machine learning optimization problem:

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

Matrix notation

Can write the problem more compactly adopting the following notation:

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

Then our optimization problem can be written

$$\text{minimize}_\theta \ \frac{1}{2} \|X\theta - y\|_2^2$$

where recall that $\|x\|_2^2 = z^T z = \sum_i z_i^2$

The gradient

The condition for finding a minimum of a scalar-valued function , that the derivative must be equal to zero (actually just true for functions with a single local minimum), can be generalized to functions of vectors

We define the multi-variate analog of the derivative, called the gradient

For a function $f: \mathbb{R}^n \to \mathbb{R}$, the gradient is a vector of all partial derivatives

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

The gradient, continued

Minimizing a multi-variate function just involves finding a point where the entire gradient is zero

$$\nabla_\theta f(\theta) = 0 \text{ (the vector of zeros)}$$

To do this, we're going to use the following rules (without proof, but the analogue to the scalar case is hopefully clear)

Chain rule: $\nabla_\theta f(X\theta) = X^T \nabla_{X\theta} f(X\theta)$

Gradient of squared Euclidean norm: $\nabla_\theta \|\theta - z\|_2^2 = 2(\theta - z)$

Solving least squares

With this notation, it's "easy" to find an analytical solution to the least squares problem

$$\text{minimize}_\theta \ \frac{1}{2}\|X\theta - y\|_2^2$$

The gradient of the optimization objective is given by:

$$\nabla_\theta \frac{1}{2}\|X\theta - y\|_2^2 = X^T \nabla_{X\theta} \frac{1}{2}\|X\theta - y\|_2^2 = X^T(X\theta - y)$$

Setting this term equal to zeros gives the least-squares solution

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

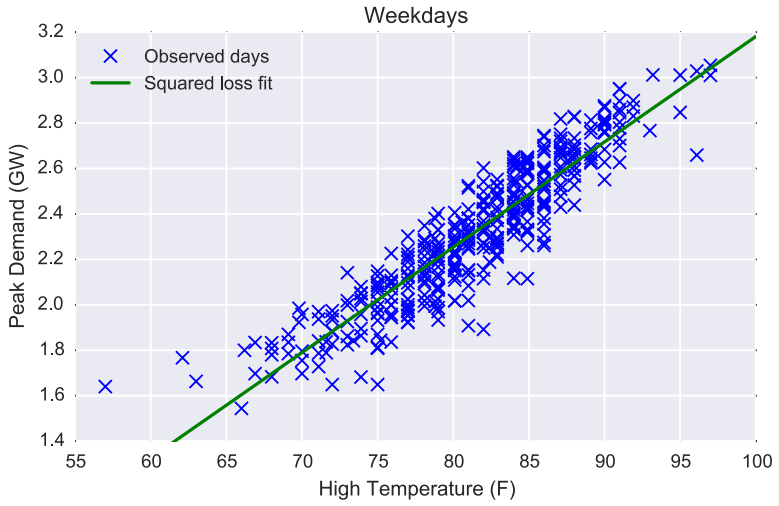
Note this solution works for *any* number of features

Example: electricity demand

Returning to our electricity demand example, this time with three features

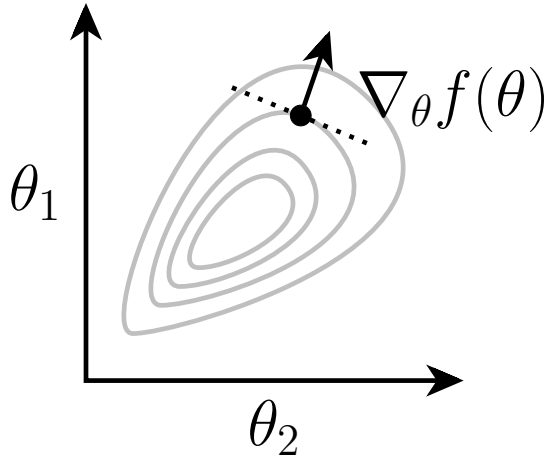
$$x^{(i)} = \begin{bmatrix} \text{High-Temperature}^{(i)} \\ \text{Is-Weekday}^{(i)} \\ 1 \end{bmatrix}$$

$$\theta = (X^T X)^{-1} X^T y = \begin{bmatrix} 0.046 \\ 0.227 \\ -1.683 \end{bmatrix}$$



An alternative solution method: gradient descent

The gradient provides more than a condition for optimality, it also gives the direction of "steepest increase" for the function



Provides an intuitive approach to minimizing $f(\theta)$: take steps in the direction of the negative gradient

Gradient descent algorithm

Generic gradient descent algorithm:

Given: hypothesis function h_θ, loss function ℓ,
input features $x^{(i)}$, outputs $y^{(i)}$, step size α
Initialization:
$\theta \leftarrow 0$
Repeat until convergence:
Compute gradient $g \leftarrow \sum_{i=1}^{m} \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
Update parameters $\theta \leftarrow \theta - \alpha \cdot g$

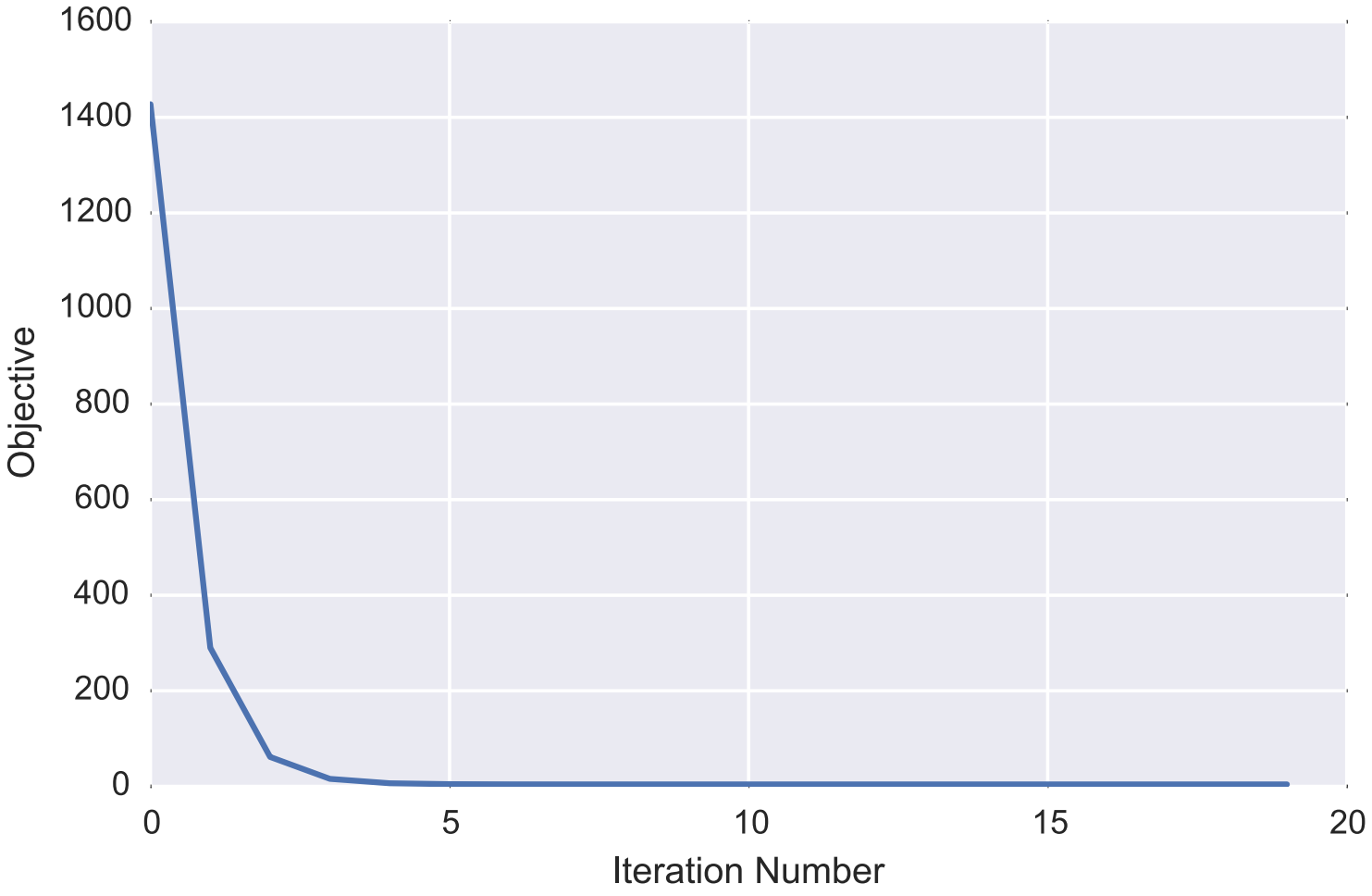
A workhorse of machine learning (also works for functions with local optima, are many data-efficient versions, etc)

Gradient descent for least squares

For linear hypothesis function and squared error, gradient descent takes the form:

Given: Feature matrix X, output vector y, step size α
Initialization:
    $\theta \leftarrow 0$
Repeat until convergence:
    Compute gradient $g \leftarrow X^T(X\theta - y)$
    Update parameters $\theta \leftarrow \theta - \alpha \cdot g$

Progress of gradient descent



Least absolute deviations

Why did we pick squared error $\ell(\hat{y}, y) = (\hat{y} - y)^2$?

An alternative to squared error is to use the *absolute* error loss function, which leads to the minimization problem:

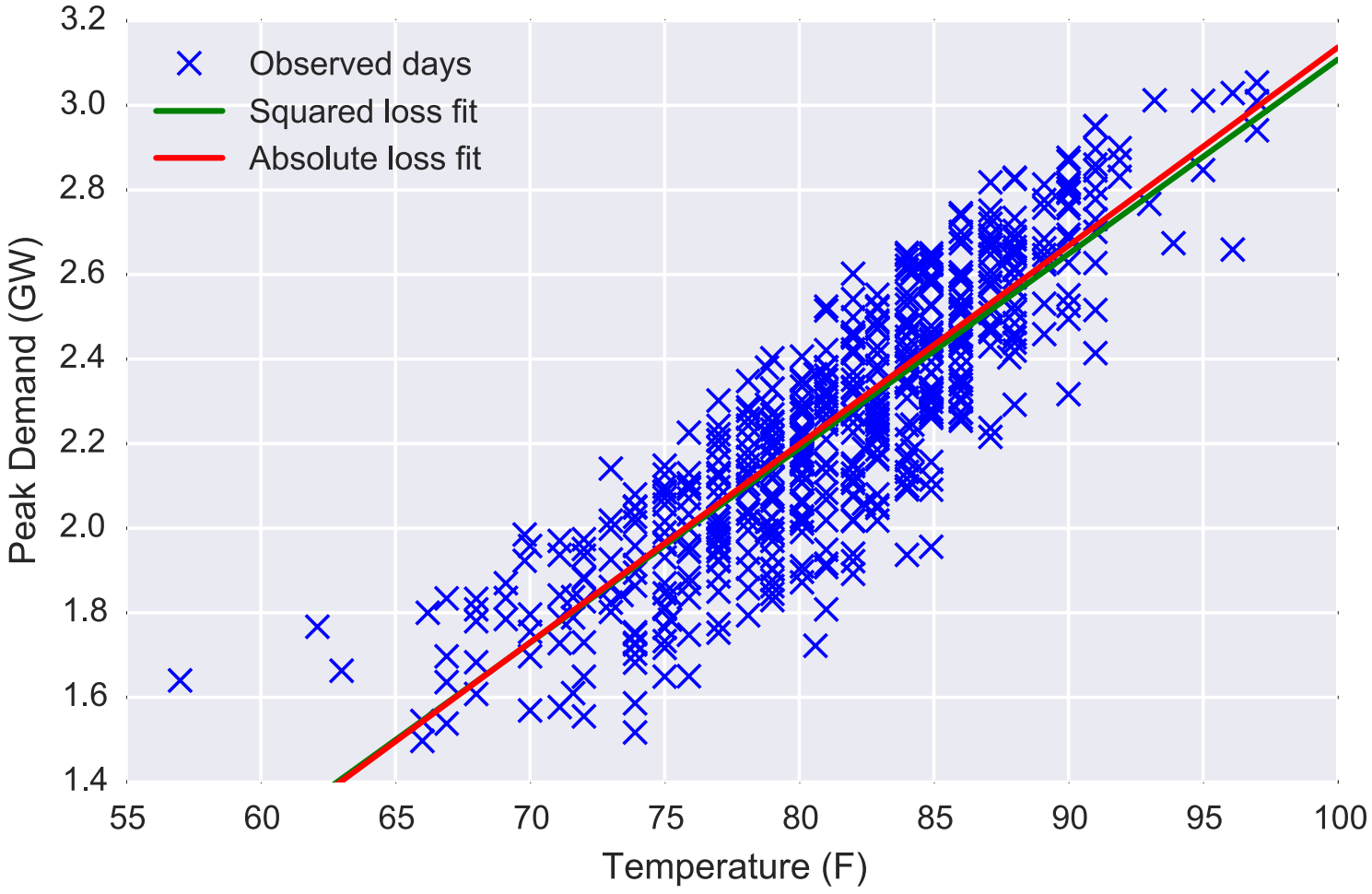
$$\text{minimize}_\theta \sum_{i=1}^{m} \left| \theta^T x^{(i)} - y^{(i)} \right| \equiv \text{minimize}_\theta \ \|X\theta - y\|_1$$

Unlike least squares, we cannot find a closed form solution to the zero gradient condition (because the function is not differentiable, it's actually called a *subgradient*, but we don't worry about such things here)

Can still be solved using gradient descent, and the gradient

$$\nabla_\theta \ \|X\theta - y\|_1 = X^T \text{sign}(X\theta - y)$$

Squared vs. absolute error



Regularization

We often want (for reasons that we'll discuss much more next lecture), to also penalize values of θ that are too large

To accomplish this, we'll actually include two terms in our canonical machine learning problem, the loss term and what is called a *regularization* term

$$\text{minimize}_\theta \quad \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{2}\|\theta\|_2^2$$

We'll talk about the meaning of regularization soon, but for now the important point is that when we add regularization, we are considering the *relative scales* of the θ_j terms

Feature normalization

The features themselves need to be scaled similarly, or parameter weights are not comparable, thus common to *normalize* features by

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \text{mean}(x_j)}{\text{std}(x_j)}$$

(not done for constant feature, x_n)

If we then solve for $\tilde{\theta}$ by minimizing loss, we need to transform θ similarly:

$$\theta_j = \frac{\tilde{\theta}_j}{\text{std}(x_j)}, \qquad \theta_n = \theta_n - \sum_{i=1}^{m} \theta_j \cdot \text{mean}(x_j)$$

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Implementation

Least squares in Python

There are libraries that will do this for you, but for ordinary least squares, it's my personal belief that you should always do it yourself

```
# Set up X and y numpy arrays
theta = np.linalg.solve(X.T.dot(X), X.T.dot(y))
```

Note: for numerical stability, you may have to add a little bit of *regularization* to the $X^T X$ matrix (we'll discuss this much more later)

```
# Set up X and y numpy arrays
theta = np.linalg.solve(X.T.dot(X) + 1e-4*np.eye(X.shape[1]), X.T.dot(y))
```

Gradient descent in Python

A simple implementation of gradient descent for least squares

```
def gradient_descent_squared_loss(X, y, T, alpha):
    m,n = X.shape
    theta = np.zeros(n)
    f = np.zeros(T)
    for i in range(T):
        f[i] = 0.5*np.linalg.norm(X.dot(theta) - y)**2
        g = X.T.dot(X.dot(theta) - y)
        theta = theta - alpha*g
    return theta, f
```

Absolute loss case is identical except for the lines:

```
        f[i] = np.linalg.norm(X.dot(theta) - y,1)
        g = X.T.dot(np.sign(X.dot(theta) - y))
```

Normalizing features

Normalize features before running gradient descent

```
# compute normalized features
X0 = X[:,:-1]
meanX = np.mean(X0,axis=0)
stdX = np.std(X0, axis=0)
X0 = np.hstack([(X0 - meanX)/stdX, np.ones((X.shape[0],1))])

theta, f0 = gradient_descent_squared_loss(X0,y,20,1e-3)
```

Convert theta back to original feature space

```
theta[:-1] = theta[:-1]/stdX
theta[-1] -= theta[:-1].dot(meanX)
```