15-388/688 - Practical Data Science: Intro to Machine Learning & Linear Regression

J. Zico Kolter
Carnegie Mellon University

Spring 2018

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

A simple example: predicting electricity use

What will peak power consumption be in Pittsburgh tomorrow?

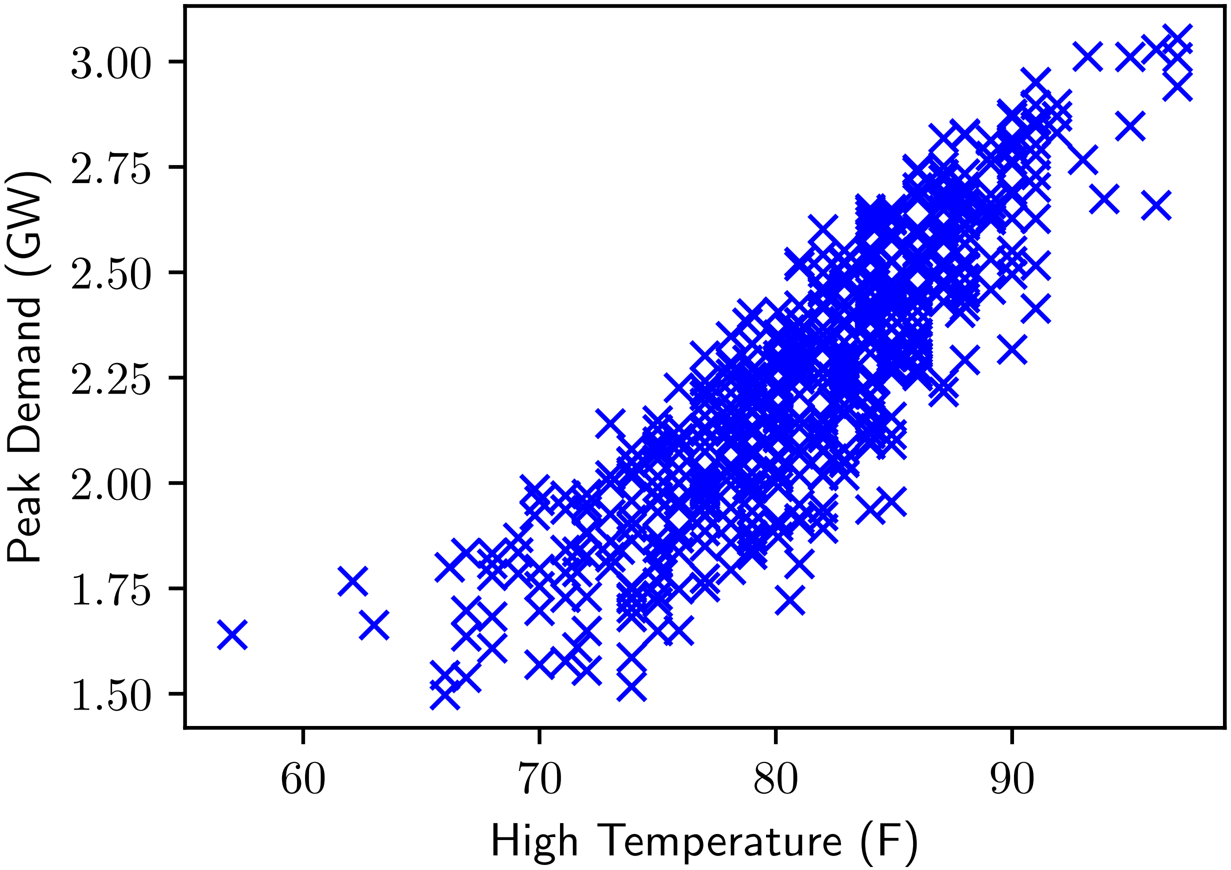
Difficult to build an “a priori” model from first principles to answer this question

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

$$E(\theta) = \sum_{i \in \text{days}} (\theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2 \; - \text{Peak\_Demand}^{(i)})^2$$

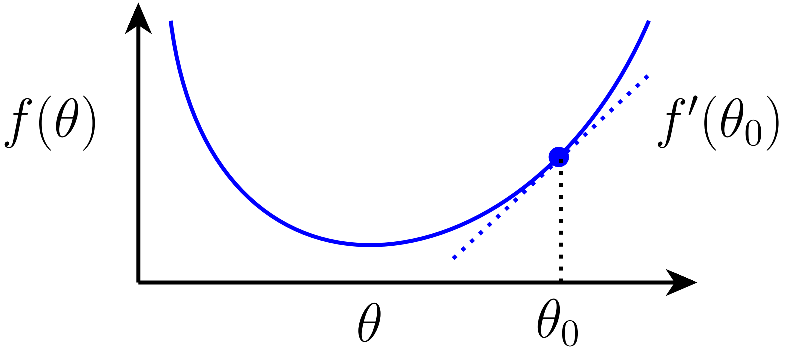
How do we find parameters?

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

$$E(\theta) = \sum_{i \in \text{days}} (\theta_1 \cdot \text{High\_Temperature}^{(i)} + \theta_2 \ - \text{Peak\_Demand}^{(i)})^2$$

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

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



Derivative is slope of the function, so negative derivative points “downhill”

Computing the derivatives

What are the derivatives of the error function with respect to each parameter θ_1 and θ_2?

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

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

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

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

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

Finding the best θ

To find a good value of θ, we can repeatedly take steps in the direction of the negative derivatives for each value

Repeat:

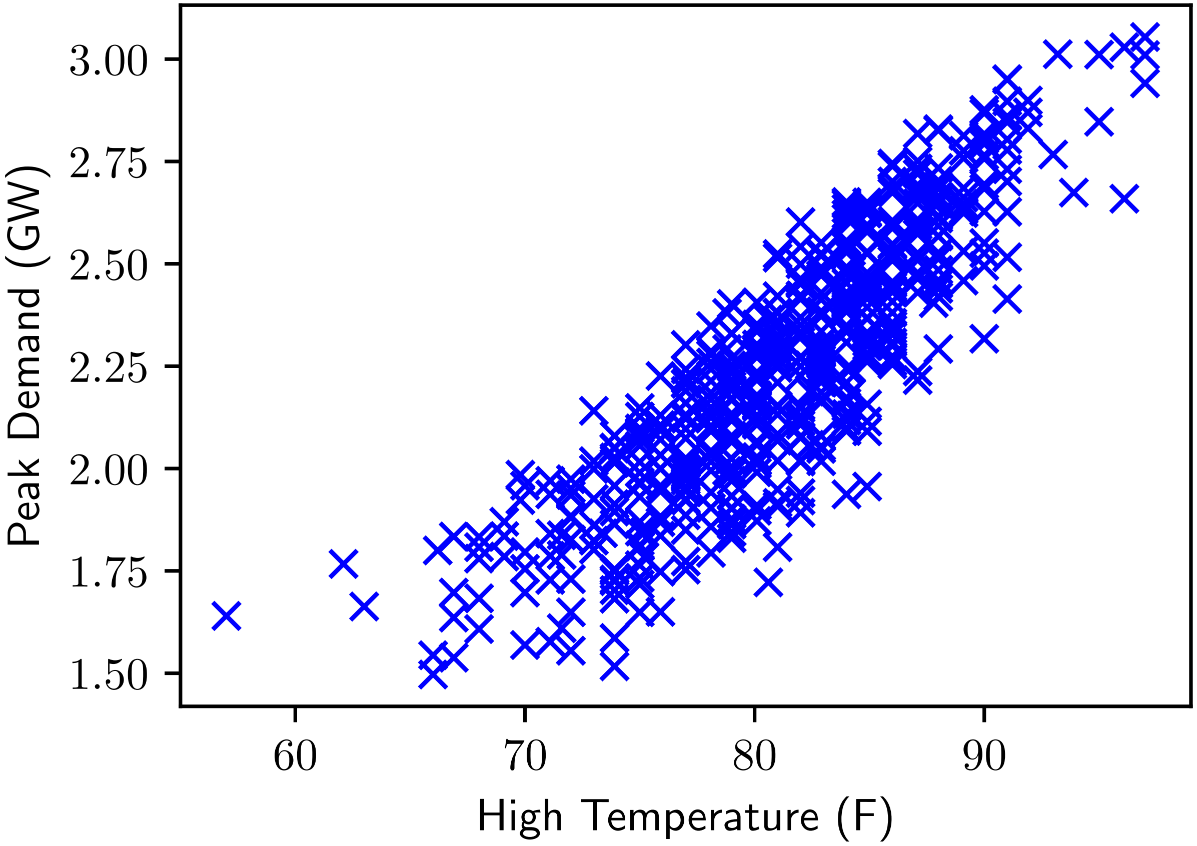
$$\theta_1 := \theta_1 - \alpha \sum_{i \in \mathrm{days}} (\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)}) \cdot x^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \sum_{i \in \mathrm{days}} (\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)})$$

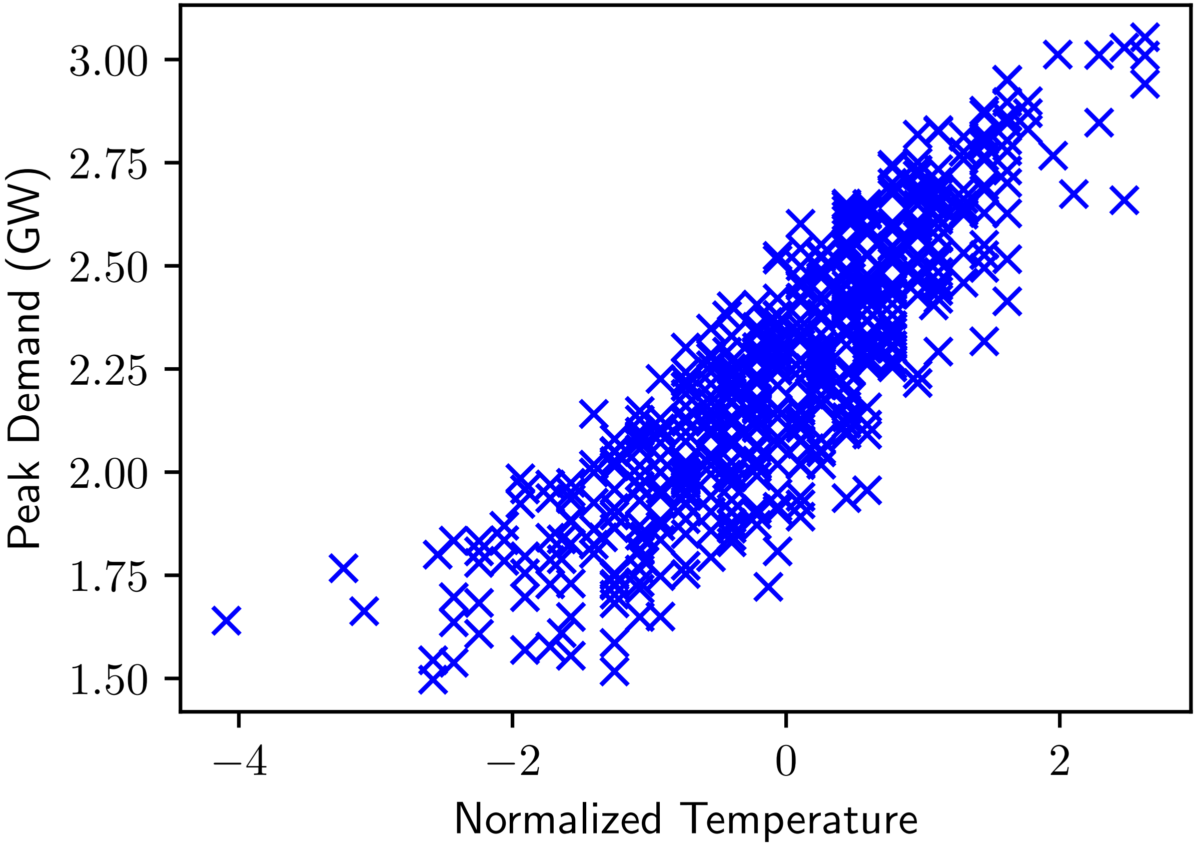
where α is some small positive number called the *step size*

This is the *gradient decent algorithm*, it is the workhorse of all modern machine learning

Gradient descent

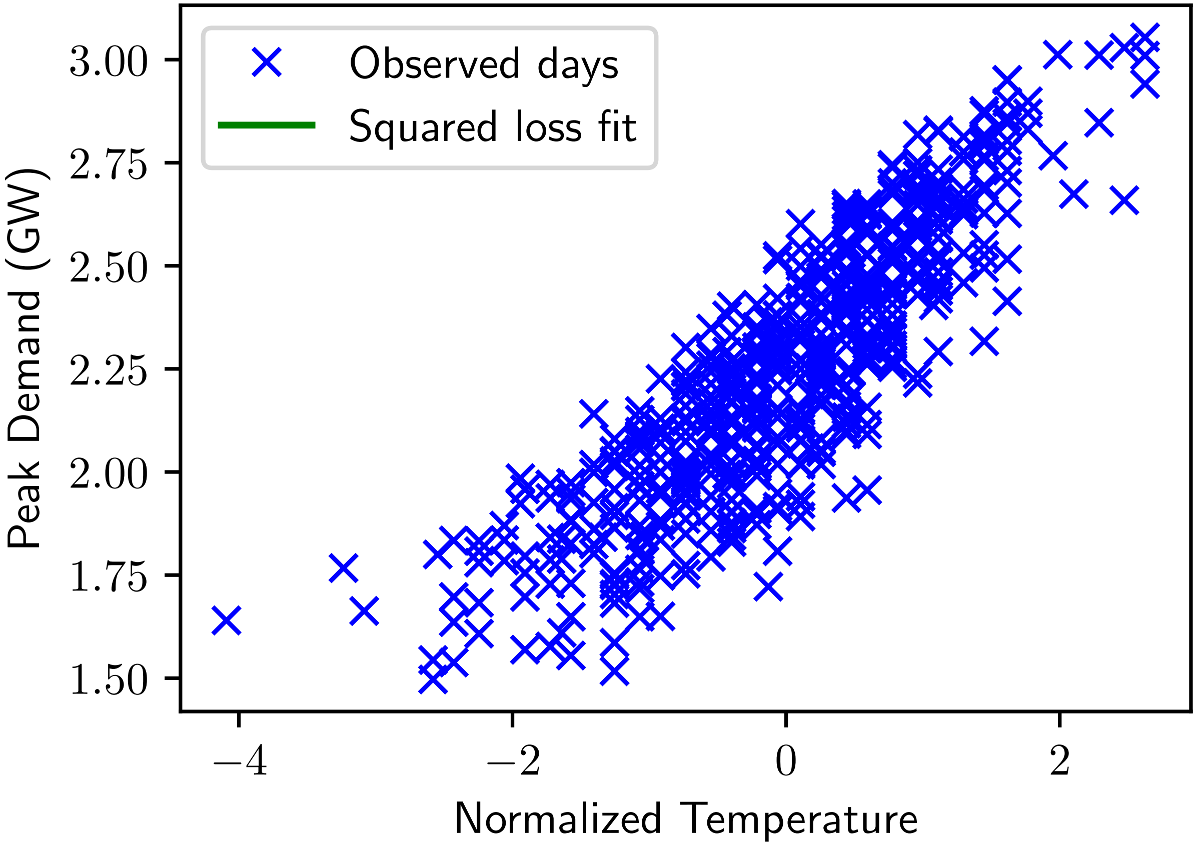


Gradient descent



Normalize input by subtracting the mean and dividing by the standard deviation

Gradient descent – Iteration 1

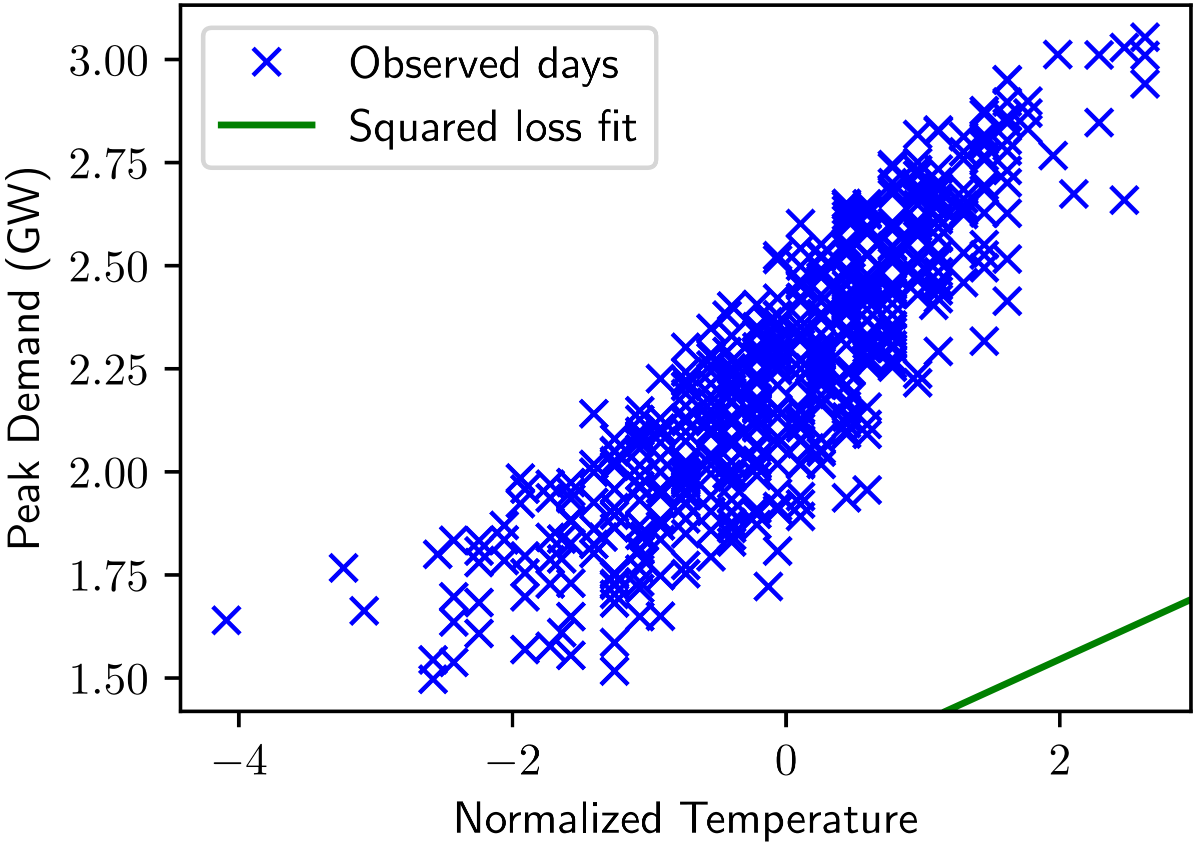


$\theta = (0.00, 0.00)$

$E(\theta) = 1427.53$

$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-151.20, -1243.10)$

Gradient descent – Iteration 2

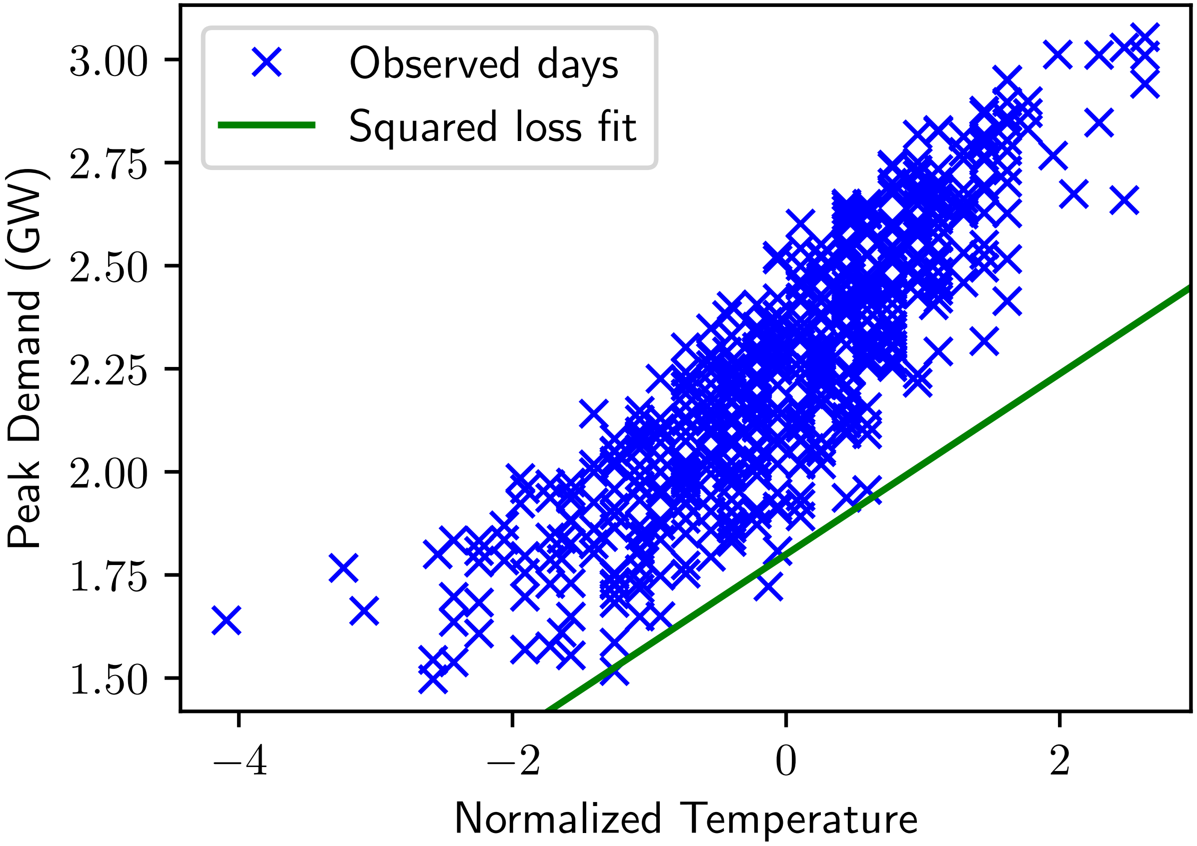


$\theta = (0.15, 1.24)$

$E(\theta) = 292.18$

$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-67.74, -556.91)$

Gradient descent – Iteration 3

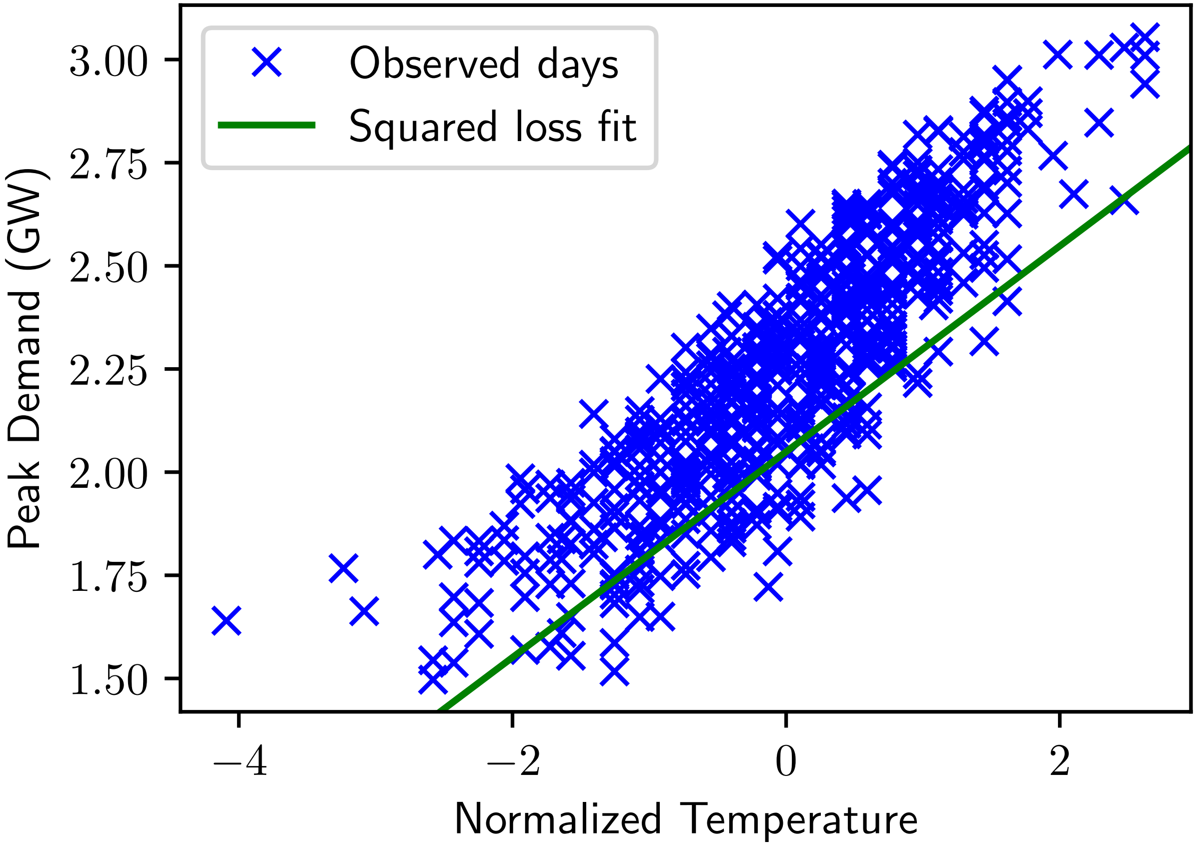


$\theta = (0.22, 1.80)$

$E(\theta) = 64.31$

$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-30.35, -249.50)$

Gradient descent – Iteration 4

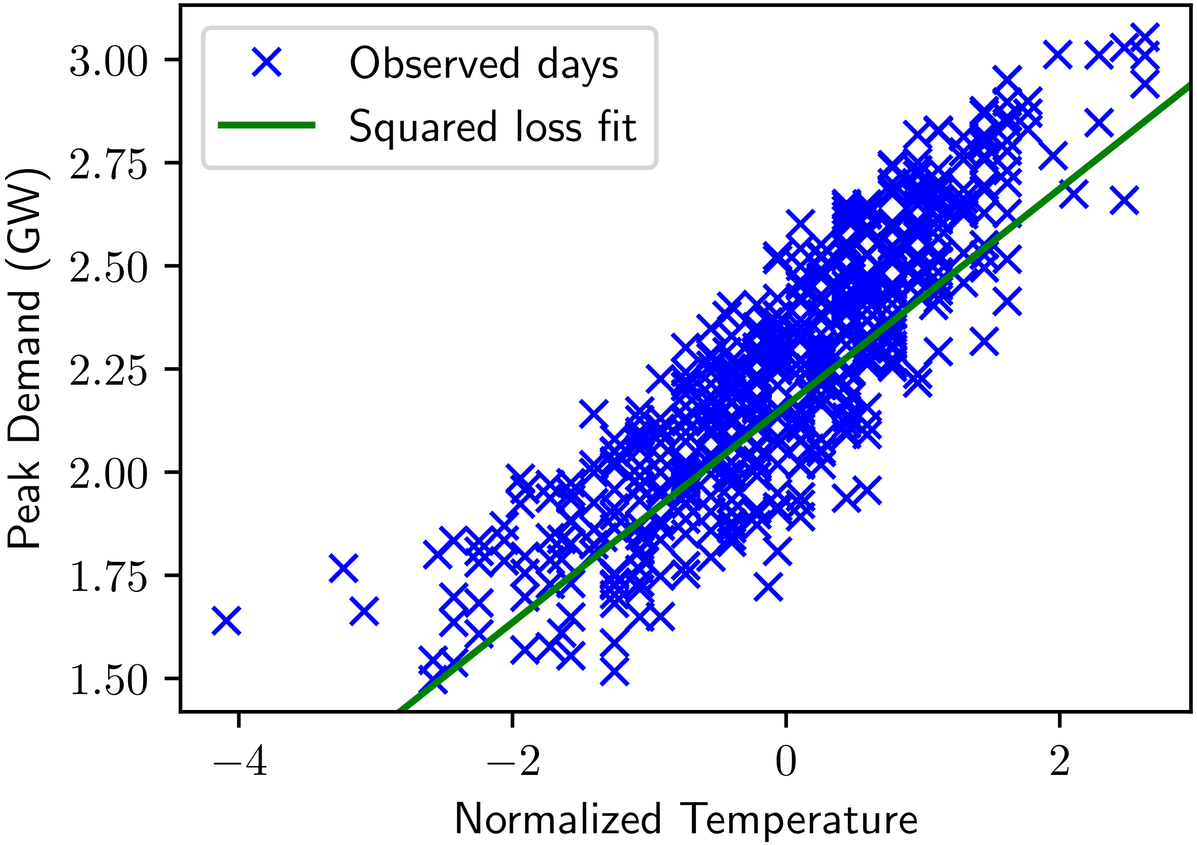


$\theta = (0.25, 2.05)$

$E(\theta) = 18.58$

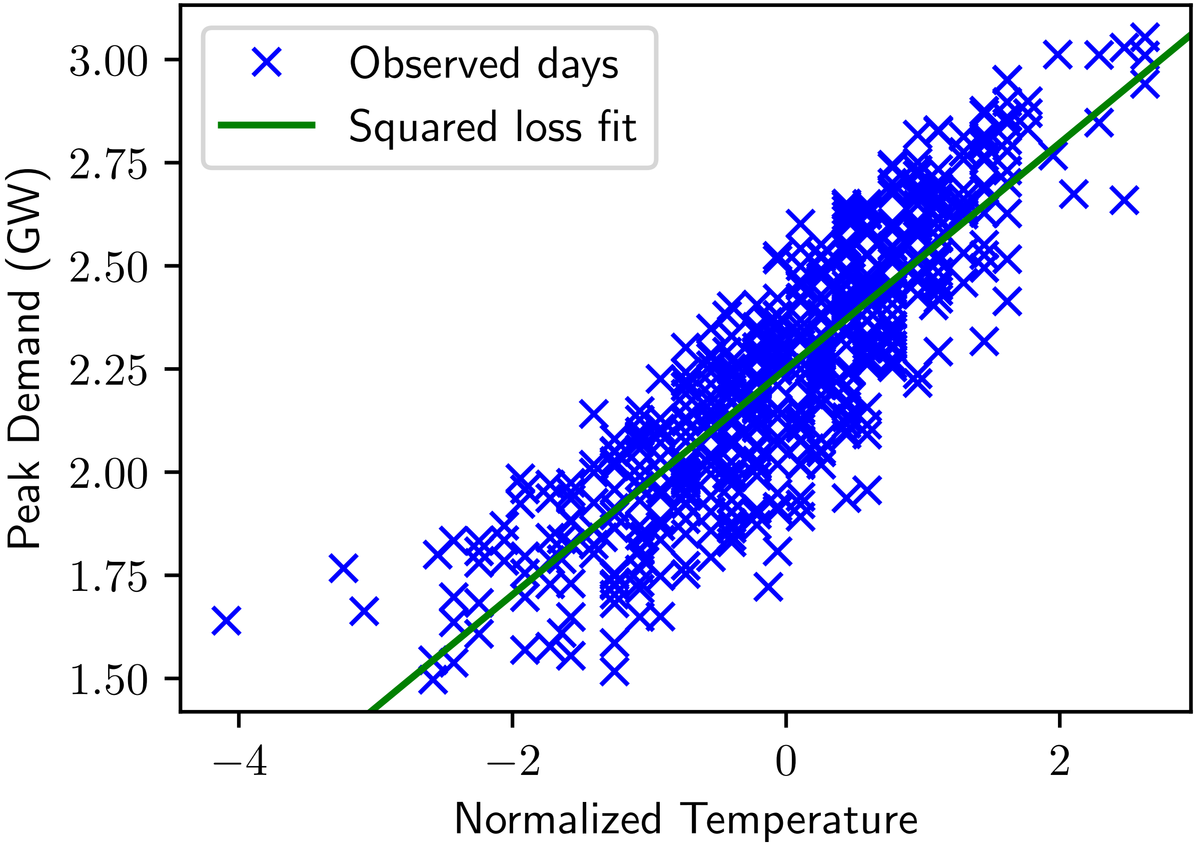
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-13.60, -111.77)$

Gradient descent – Iteration 5



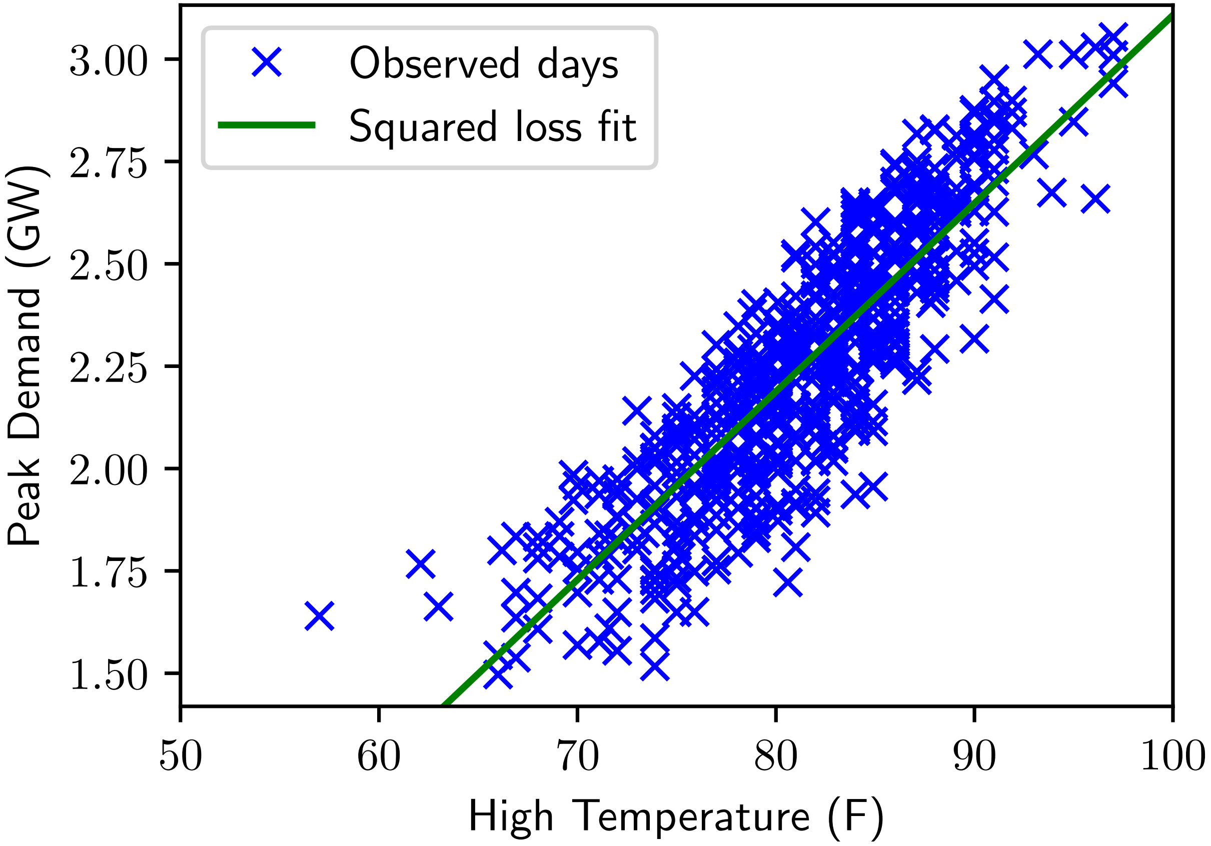
$\theta = (0.26, 2.16)$
$E(\theta) = 9.40$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-6.09, -50.07)$

Gradient descent – Iteration 10



$$\theta = (0.27, 2.25)$$
$$E(\theta) = 7.09$$
$$\left(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}\right) = (-0.11, -0.90)$$

Fitted line in “original” coordinates



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:

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

(requires that we rescale θ after solving to “normal” coordinates)

Equivalent to just “finding the point on the line”

Poll: Non-rescaled gradient descent

What do you expect would happen if we run gradient descent without first scaling the coordinates?

1. If the algorithm converges, it may converge to the incorrect solution

2. The algorithm might diverge (not reach any solution)

3. We will need a smaller step size α to ensure convergence

4. We will need a bigger step size α to ensure convergence

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

Matrix/vector notation and analytic solutions

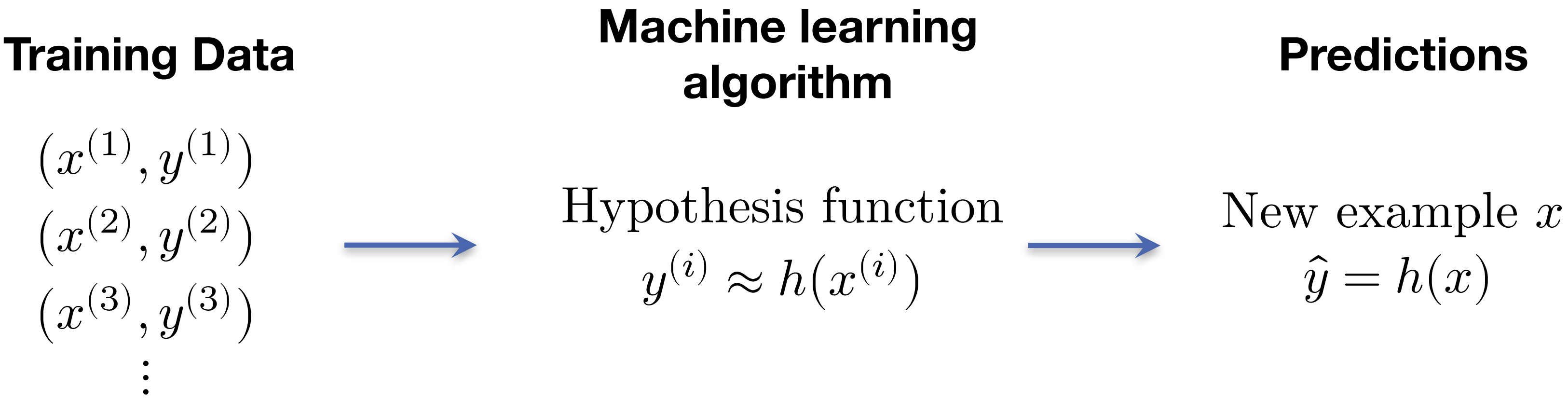
Implementing linear regression

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):



Training Data

$(x^{(1)}, y^{(1)})$

$(x^{(2)}, y^{(2)})$

$(x^{(3)}, y^{(3)})$

$\vdots$

Machine learning algorithm

Hypothesis function

$y^{(i)} \approx h(x^{(i)})$

Predictions

New example x

$\hat{y} = h(x)$

Terminology

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

$$\mathrm{E.g.:}\ \ x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ \text{Is\_Weekday}^{(i)} \\ 1 \end{bmatrix}$$

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

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

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

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

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

Terminology

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

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

The canonical machine learning optimization problem:

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

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 independence assumptions, analytical solution}

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

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: $h_\theta(x) = \sum_{j=1}^{n} \theta_j \cdot x_j$

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

Resulting machine learning optimization problem:

$$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2 \equiv \operatorname*{minimize}_{\theta} E(\theta)$$

Derivative of the least squares objective

We can compute the partial derivative with respect to an arbitrary model parameter θ_j

$$\begin{aligned}
\frac{\partial E(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2 \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right)^2 \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) \frac{\partial}{\partial \theta_j} \sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} \\
&= \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) x_j^{(i)}
\end{aligned}$$

Gradient descent algorithm

1. Initialize $\theta_j := 0,\ j = 1, \dots, n$

2. Repeat:

   - For $j = 1, \dots, n$:

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

Note: do not actually implement it like this, you'll want to use the matrix/vector notation we will over soon

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

The gradient

It is typically more convenient to work with a vector of all partial derivatives, called the ***gradient***

For a function $f \colon \mathbb{R}^n \to \mathbb{R}$, the gradient is a vector

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

Gradient in vector notation

We can actually *simplify* the gradient computation (both notationally and computationally) substantially using matrix/vector notation

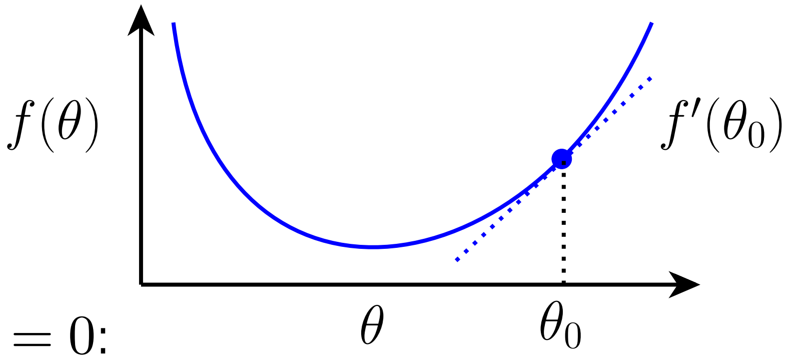
$$\frac{\partial E(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) x_j^{(i)}$$

$$\Longleftrightarrow \nabla_\theta E(\theta) = \sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right)$$

Putting things in this form also make it more clear how to analytically find the optimal solution for last squares

Solving least squares

Gradient also gives a condition for optimality: gradient must equal zero



Solving for $\nabla_\theta E(\theta) = 0$:

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

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

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

Matrix notation, one level deeper

Let's define the matrices

$$X = \begin{bmatrix} - & {x^{(1)}}^T & - \\ - & {x^{(2)}}^T & - \\ & \vdots & \\ - & {x^{(m)}}^T & - \end{bmatrix}, y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Then

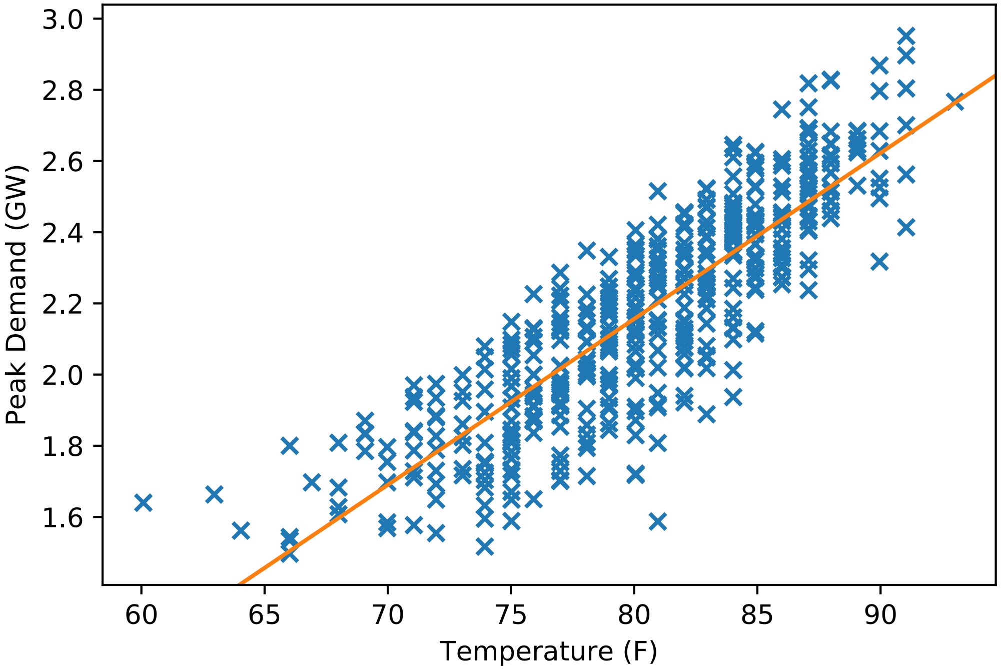
$$\nabla_\theta E(\theta) = \sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right) = X^T (X\theta - y)$$
$$\Longrightarrow \theta^\star = (X^T X)^{-1} X^T y$$

These are known as the *normal equations* an extremely convenient closed-form solution for least squares (without need for normalization)

Example: electricity demand

Returning to our electricity demand example:

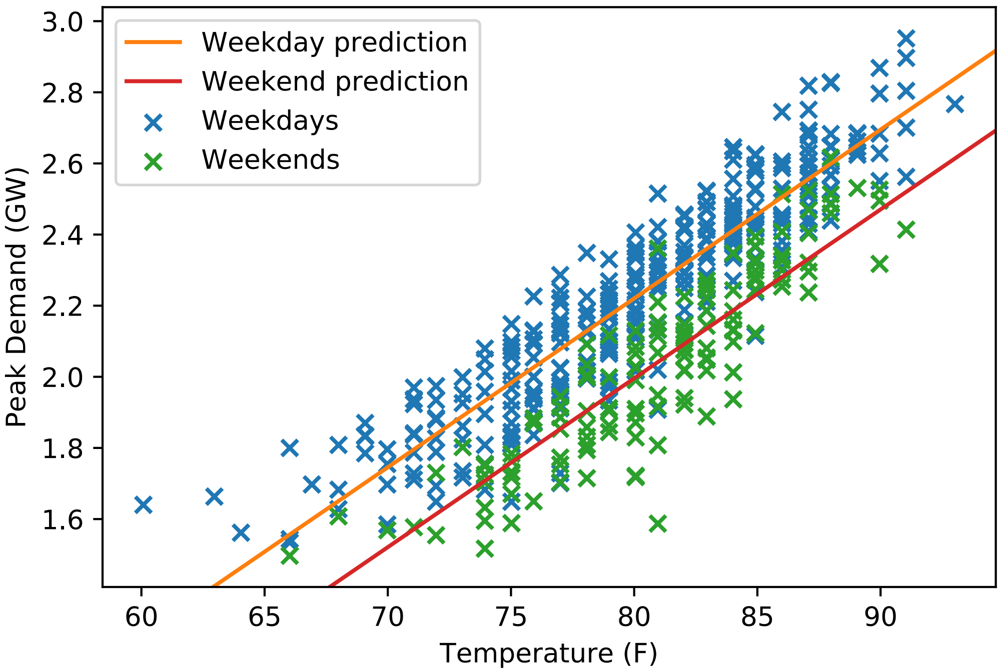
$$x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ 1 \end{bmatrix}, \qquad \theta^{\star}(X^T X)^{-1} X^T y = \begin{bmatrix} 0.046 \\ -1.574 \end{bmatrix}$$



Example: electricity demand

Returning to our electricity demand example:

$$x^{(i)} = \begin{bmatrix} \text{High\_Temperature}^{(i)} \\ \text{Is\_Weekday}^{(i)} \\ 1 \end{bmatrix},$$



Poll: linear regression models

In the previous example, we had the same slope for both weekend and weekday examples, just with a different intercept. It is possible to have a model with both different slopes and different intercepts?

1. The previous example already *did* have different slopes

2. This is not possible with linear regression

3. You need to build two models, one just one weekdays and one just on weekends

4. You can do it with a single model, just with different features

Outline

Least squares regression: a simple example

Machine learning notation

Linear regression revisited

Matrix/vector notation and analytic solutions

Implementing linear regression

Manual implementation of linear regression

Create data matrices:

```
# initialize X matrix and y vector
X = np.array([df["Temp"], df["IsWeekday"], np.ones(len(df))]).T
y = df_summer["Load"].values
```

Compute solution:

```
# solve least squares
theta = np.linalg.solve(X.T @ X, X.T @ y)
print(theta)
# [ 0.04747948  0.22462824 -1.80260016]
```

Make predictions:

```
# predict on new data
Xnew = np.array([[77, 1, 1], [80, 0, 1]])
ypred = Xnew @ theta
print(ypred)
# [ 2.07794778  1.99575797]
```

Scikit-learn

By far the most popular machine learning library in Python is the scikit-learn library (http://scikit-learn.org/)

Reasonable (usually) implementation of many different learning algorithms, usually fast enough for small/medium problems

Important: you *need* to understand the very basics of how these algorithms work in order to use them effectively

Sadly, a lot of data science in practice seems to be driven by the default parameters for scikit-learn classifiers…

Linear regression in scikit-learn

Fit a model and predict on new data

```
from sklearn.linear_model import LinearRegression

# don't include constant term in X
X = np.array([df_summer["Temp"], df_summer["IsWeekday"]]).T
model = LinearRegression(fit_intercept=True, normalize=False)
model.fit(X, y)

# predict on new data
Xnew = np.array([[77, 1], [80, 0]])
model.predict(Xnew)
# [ 2.07794778  1.99575797]
```

Inspect internal model coefficients

```
print(model.coef_, model.intercept_)
# [ 0.04747948  0.22462824]  -1.80260016
```

Scikit-learn-like model, manually

We can easily implement a class that contains a scikit-learn-like interface

```python
class MyLinearRegression:
    def __init__(self, fit_intercept=True):
        self.fit_intercept = fit_intercept

    def fit(self, X, y):
        if self.fit_intercept:
            X = np.hstack([X, np.ones((X.shape[0],1))])

        self.coef_ = np.linalg.solve(X.T @ X, X.T @ y)

        if self.fit_intercept:
            self.intercept_ = self.coef_[-1]
            self.coef_ = self.coef_[:-1]

    def predict(self, X):
        pred = X @ self.coef_
        if self.fit_intercept:
            pred += self.intercept_
        return pred
```