Week 1

Regression

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Recap on mathematical notation

Learning objectives

In this lecture we will...

- Revise the mathematical notation necessary to cover the basics of machine learning algorithms

Vectors and matrices

x : **vector** – usually a vector of **features**

x_i : a vector component, usually a single feature

y : a vector of **labels**

θ : a vector of **parameters**

X : **matrix** – usually a **feature matrix**

X_i : feature vector for a specific datapoint

X^T: **transpose** operator

$\|\theta\|_2^2 = \sum_i \theta_i^2$: vector **norm**

(in general $\|\theta\|_p = (\sum_i |\theta_i|^p)^{\frac{1}{p}}$

Linearity

We will frequently talk about **linear** models

Precisely speaking, a function *f(x)* is **linear** if

$$f(x+y) = f(x) + f(y) \text{ (additivity)}$$
$$f(\alpha x) = \alpha f(x) \text{ (homogeneity)}$$

For the purposes of this class, we care about functions of the form:

$$\theta \cdot x$$

which is linear in theta

Probability and statistics

$p(y)$: **probability** of some event

$p(y|x)$: **conditional probability** of some event

$\bar{x} = \frac{1}{|x|} \sum_i x_i$: **mean** of a vector

$var(x)$: **variance** of a vector

$\sigma(x) = \frac{1}{1+e^{-x}}$: **sigmoid** function

Summary of concepts

- Covered basic notation of vectors, matrices, probability, and statistics

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Introduction to supervised learning

Learning objectives

In this lecture we will

- Introduce the concept of **supervised learning**
- Contrast **supervised learning** and **unsupervised learning**

Concept: What is supervised learning?

Supervised learning is the process of trying to infer from **labeled data** the underlying function that produced the labels associated with the data

What is supervised learning?

Given **labeled training data** of the form

$$\{(\text{data}_1, \text{label}_1), \ldots, (\text{data}_n, \text{label}_n)\}$$

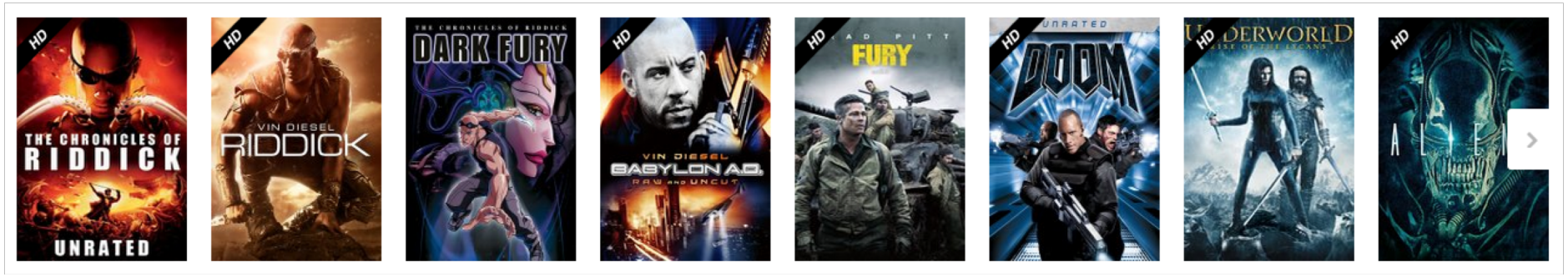
Infer the function

$$f(\text{data}) \xrightarrow{?} \text{labels}$$

Example: movie recommendation

Suppose we want to build a movie recommender

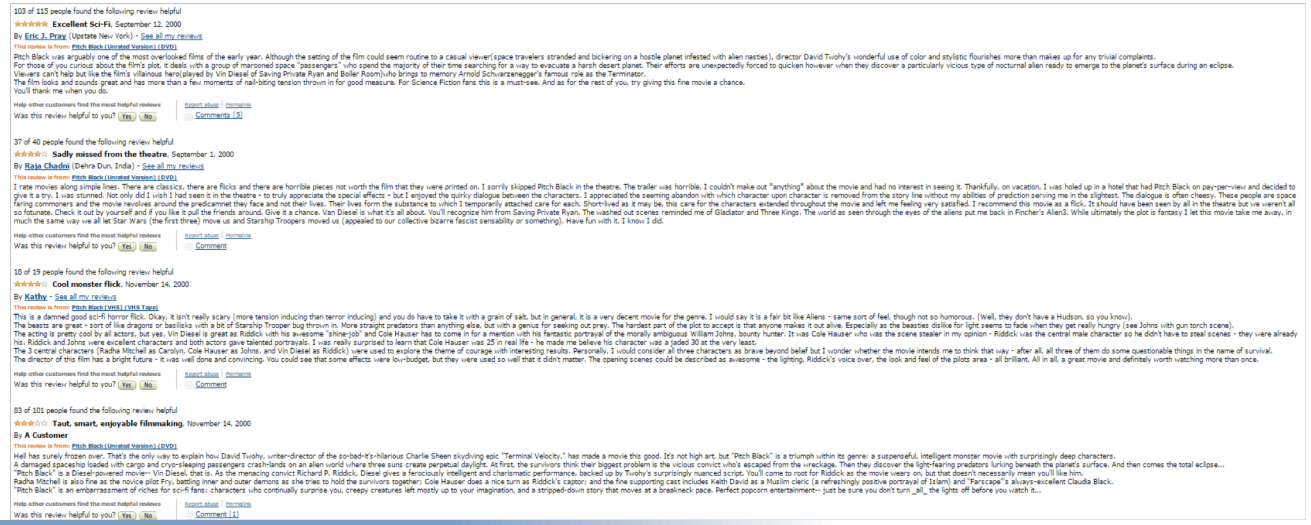
e.g. which of these films will I rate highest?



Example: movie recommendation

Q: What are the **labels?**

A: ratings that others have given to each movie, and that I have given to other movies



103 of 115 people found the following review helpful

★★★★★ **Excellent Sci-Fi**, September 12, 2000

By **Eric J. Pray** (Upstate New York) - See all my reviews

This review is from: Pitch Black (Unrated Version) (DVD)

Pitch Black was arguably one of the most overlooked films of the early year. Although the setting of the film could seem routine to a casual viewer(space travelers stranded and bickering on a hostile planet infested with alien nasties), director David Twohy's wonderful use of color and stylistic flourishes more than makes up for any trivial complaints.
For those of you curious about the film's plot, it deals with a group of marooned space "passengers" who spend the majority of their time searching for a way to evacuate a harsh desert planet. Their efforts are unexpectedly forced to quicken however when they discover a particularly vicious type of nocturnal alien ready to emerge to the planet's surface during an eclipse.
Viewers can't help but like the film's villainous hero(played by Vin Diesel of Saving Private Ryan and Boiler Room)who brings to memory Arnold Schwarzenegger's famous role as the Terminator.
The film looks and sounds great and has more than a few moments of nail-biting tension thrown in for good measure. For Science Fiction fans this is a must-see. And as for the rest of you, try giving this fine movie a chance.
You'll thank me when you do.

Help other customers find the most helpful reviews | Report abuse | Permalink
Was this review helpful to you? Yes No | Comments [5]

37 of 40 people found the following review helpful

★★★★☆ **Sadly missed from the theatre**, September 1, 2000

By **Raja Chadni** (Dehra Dun, India) - See all my reviews

This review is from: Pitch Black (Unrated Version) (DVD)

I rate movies along simple lines. There are classics, there are flicks and there are horrible pieces not worth the film that they were printed on. I sorrily skipped Pitch Black in the theatre. The trailer was horrible. I couldn't make out "anything" about the movie and had no interest in seeing it. Thankfully, on vacation, I was holed up in a hotel that had Pitch Black on pay-per-view and decided to give it a try. I was stunned. Not only did I wish I had seen it in the theatre - to truly appreciate the special effects - but I enjoyed the quirky dialogue between the characters. I appreciated the seeming abandon with which character upon character is removed from the story line without my abilities of prediction serving me in the slightest. The dialogue is often cheesy. These people are space faring commoners and the movie revolves around the predicament they face and not their lives. Their lives form the substance to which I temporarily attached care for each. Short-lived as it may be, this care for the characters extended throughout the movie and left me feeling very satisfied. I recommend this movie as a flick. It should have been seen by all in the theatre but we weren't all so fotunate. Check it out by yourself and if you like it pull the friends around. Give it a chance. Van Diesel is what it's all about. You'll recognize him from Saving Private Ryan. The washed out scenes reminded me of Gladiator and Three Kings. The world as seen through the eyes of the aliens put me back in Fincher's Alien3. While ultimately the plot is fantasy I let this movie take me away, in much the same way we all let Star Wars (the first three) move us and Starship Troopers moved us (appealed to our collective bizarre fascist sensability or something). Have fun with it. I know I did.

Help other customers find the most helpful reviews | Report abuse | Permalink
Was this review helpful to you? Yes No | Comment

18 of 19 people found the following review helpful

★★★★☆ **Cool monster flick**, November 14, 2000

By **Kathy** - See all my reviews

This review is from: Pitch Black [VHS] (VHS Tape)

This is a damned good sci-fi horror flick. Okay, it isn't really scary (more tension inducing than terror inducing) and you do have to take it with a grain of salt, but in general, it is a very decent movie for the genre. I would say it is a fair bit like Aliens - same sort of feel, though not so humorous. (Well, they don't have a Hudson, so you know).
The beasts are great - sort of like dragons or basilisks with a bit of Starship Trooper bug thrown in. More straight predators than anything else, but with a genius for seeking out prey. The hardest part of the plot to accept is that anyone makes it out alive. Especially as the beasties dislike for light seems to fade when they get really hungry (see Johns with gun torch scene).
The acting is pretty cool by all actors, but yes, Vin Diesel is great as Riddick with his awesome "shine-job" and Cole Hauser has to come in for a mention with his fantastic portrayal of the morally ambiguous William Johns, bounty hunter. It was Cole Hauser who was the scene stealer in my opinion - Riddick was the central male character so he didn't have to steal scenes - they were already his. Riddick and Johns were excellent characters and both actors gave talented portrayals. I was really surprised to learn that Cole Hauser was 25 in real life - he made me believe his character was a jaded 30 at the very least.
The 3 central characters (Radha Mitchell as Carolyn, Cole Hauser as Johns, and Vin Diesel as Riddick) were used to explore the theme of courage with interesting results. Personally, I would consider all three characters as brave beyond belief but I wonder whether the movie intends me to think that way - after all, all three of them do some questionable things in the name of survival.
The director of this film has a bright future - it was well done and convincing. You could see that some effects were low-budget, but they were used so well that it didn't matter. The opening scenes could be described as awesome - the lighting, Riddick's voice over, the look and feel of the plots area - all brilliant. All in all, a great movie and definitely worth watching more than once.

Help other customers find the most helpful reviews | Report abuse | Permalink
Was this review helpful to you? Yes No | Comment

83 of 101 people found the following review helpful

★★★☆☆ **Taut, smart, enjoyable filmmaking**, November 14, 2000

By **A Customer**

This review is from: Pitch Black (Unrated Version) (DVD)

Hell has surely frozen over. That's the only way to explain how David Twohy, writer-director of the so-bad-it's-hilarious Charlie Sheen skydiving epic "Terminal Velocity," has made a movie this good. It's not high art, but "Pitch Black" is a triumph within its genre: a suspenseful, intelligent monster movie with surprisingly deep characters.
A damaged spaceship loaded with cargo and cryo-sleeping passengers crash-lands on an alien world where three suns create perpetual daylight. At first, the survivors think their biggest problem is the vicious convict who's escaped from the wreckage. Then they discover the light-fearing predators lurking beneath the planet's surface. And then comes the total eclipse...
"Pitch Black" is a Diesel-powered movie-- Vin Diesel, that is. As the menacing convict Richard P. Riddick, Diesel gives a ferociously intelligent and charismatic performance, backed up by Twohy's surprisingly nuanced script. You'll come to root for Riddick as the movie wears on, but that doesn't necessarily mean you'll like him.
Radha Mitchell is also fine as the novice pilot Fry, battling inner and outer demons as she tries to hold the survivors together; Cole Hauser does a nice turn as Riddick's captor; and the fine supporting cast includes Keith David as a Muslim cleric (a refreshingly positive portrayal of Islam) and "Farscape"'s always-excellent Claudia Black.
"Pitch Black" is an embarrassment of riches for sci-fi fans: characters who continually surprise you, creepy creatures left mostly up to your imagination, and a stripped-down story that moves at a breakneck pace. Perfect popcorn entertainment-- just be sure you don't turn _all_ the lights off before you watch it...

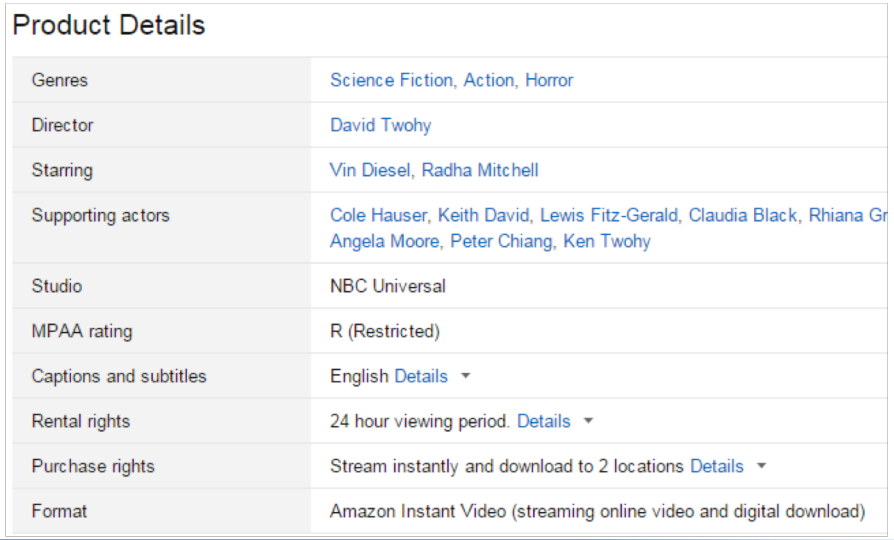
Help other customers find the most helpful reviews | Report abuse | Permalink
Was this review helpful to you? Yes No | Comment [1]

Example: movie recommendation

Q: What is the **data?**

A: features about the movie and the users who evaluated it

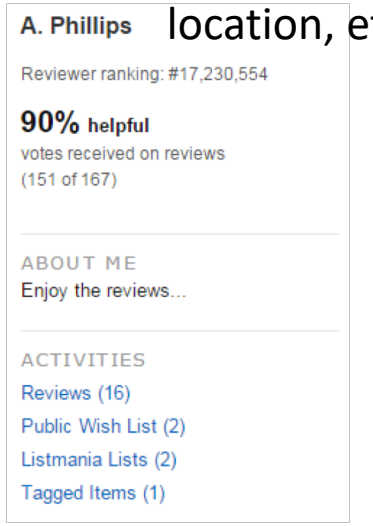
Movie features: genre, actors, rating, length, etc.



Product Details

Genres	Science Fiction, Action, Horror
Director	David Twohy
Starring	Vin Diesel, Radha Mitchell
Supporting actors	Cole Hauser, Keith David, Lewis Fitz-Gerald, Claudia Black, Rhiana Gr Angela Moore, Peter Chiang, Ken Twohy
Studio	NBC Universal
MPAA rating	R (Restricted)
Captions and subtitles	English Details
Rental rights	24 hour viewing period. Details
Purchase rights	Stream instantly and download to 2 locations Details
Format	Amazon Instant Video (streaming online video and digital download)

User features: age, gender, location, etc.



A. Phillips

Reviewer ranking: #17,230,554

90% helpful

votes received on reviews (151 of 167)

ABOUT ME

Enjoy the reviews...

ACTIVITIES

Reviews (16)

Public Wish List (2)

Listmania Lists (2)

Tagged Items (1)

Example: movie recommendation

Movie recommendation:

$$f(\text{data}) \overset{?}{\rightarrow} \text{labels}$$

=

$$f(\text{user features}, \text{movie features}) \overset{?}{\rightarrow} \text{star rating}$$

Solution 1

Solution: Design a system based on **prior knowledge**, e.g.

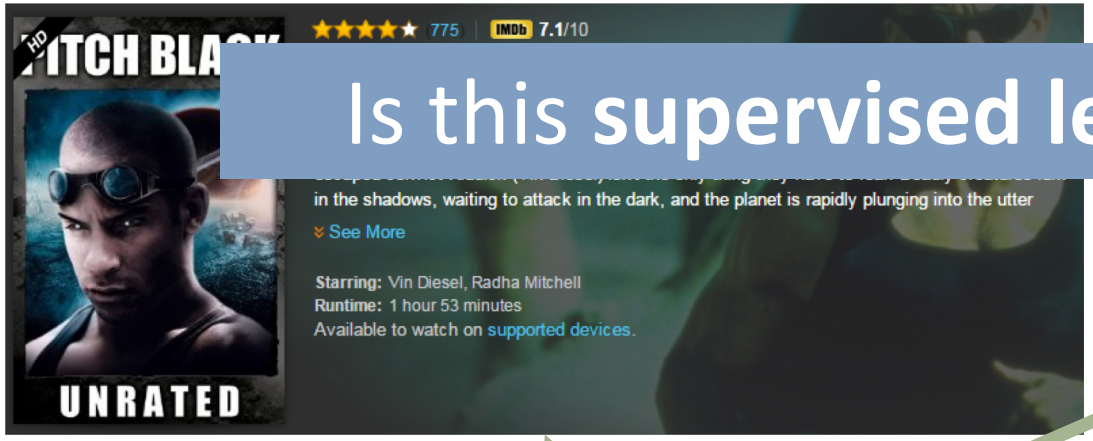
```
def prediction(user, movie):
    if (user['age'] <= 14):
        if (movie['mpaa_rating']) == "G"):
            return 5.0
        else:
            return 1.0
    else if (user['age'] <= 18):
        if (movie['mpaa_rating']) == "PG"):
            return 5.0
….. Etc.
```

Is this **supervised learning?**

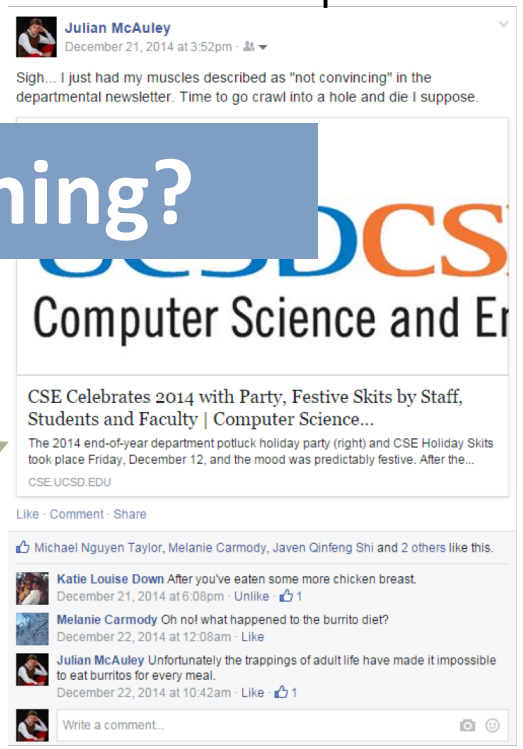
Solution 2

Solution: Identify words that I frequently mention in my social media posts, and recommend movies whose plot synopses use **similar** types of language

Plot synopsis



Social media posts



Is this **supervised learning?**

argmax similarity(synopsis, post)

Solution 3

Solution: Identify which attributes (e.g. actors, genres) are associated with positive ratings. Recommend movies that exhibit those attributes.

Is this **supervised learning?**

Solution 1

(design a system based on prior knowledge)

Disadvantages:

- Depends on possibly false **assumptions** about how users relate to items
- Cannot adapt to new data/information

Advantages:

- Requires no data!

Solution 2

(identify similarity between wall posts and synopses)

Disadvantages:

- Depends on possibly false **assumptions** about how users relate to items
- May not be adaptable to new settings

Advantages:

- Requires data, but does not require **labeled** data

Solution 3

(identify attributes that are associated with positive ratings)

Disadvantages:

- Requires a (possibly large) dataset of movies with labeled ratings

Advantages:

- Directly optimizes a measure we care about (predicting ratings)
- Easy to adapt to new settings and data

Concept: Supervised versus unsupervised learning

Learning approaches attempt to **model data** in order to solve a problem

Unsupervised learning approaches find patterns/relationships/structure in data, but **are not** optimized to solve a particular predictive task

Supervised learning aims to directly model the relationship between input and output variables, so that the output variables can be predicted accurately given the input

Summary of concepts

- Introduced the concept of **supervised learning**
- Contrasted **supervised learning, unsupervised learning,** (and also non-learning-based approaches)
- Described the relative merits of these approaches

On your own...

- Consider a related problem (e.g. predicting a movie's box-office gross), and propose ways to solve it based on **unsupervised learning, supervised learning,** (and without learning)

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Supervised Learning: Regression

Learning objectives

In this lecture we will...

- Demonstrate the idea behind **linear regression**
- Understand the role of **parameters** in a model

Regression

Regression is one of the simplest supervised learning approaches to learn relationships between input variables (features) and output variables (predictions)

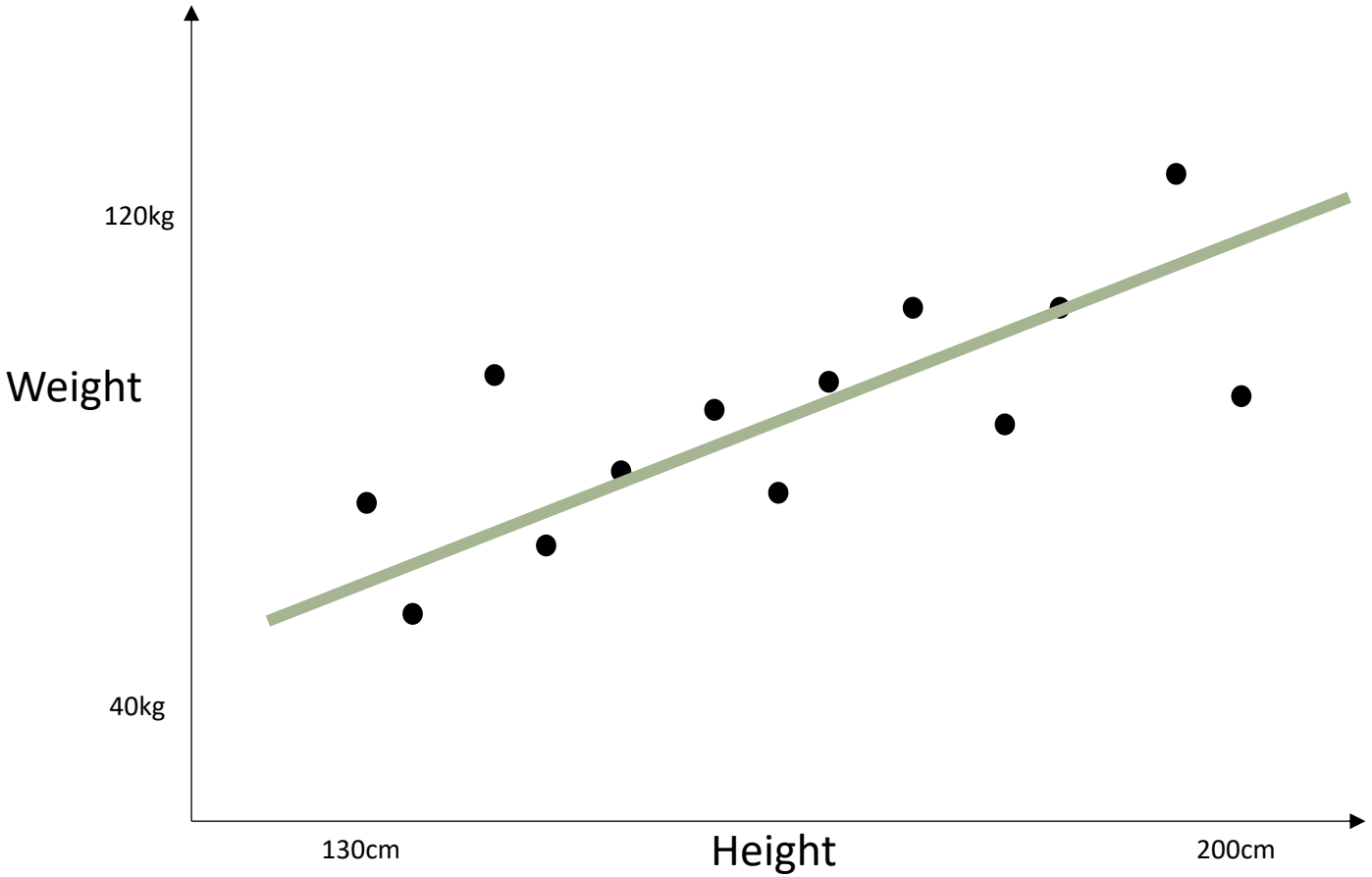
Motivation: height vs. weight

Suppose we wanted to understand the relationship between height and weight...

…or equivalently, can we **predict** a person's weight from their height?

Motivation: height vs. weight

Q: Can we find a line that (approximately) fits the data?



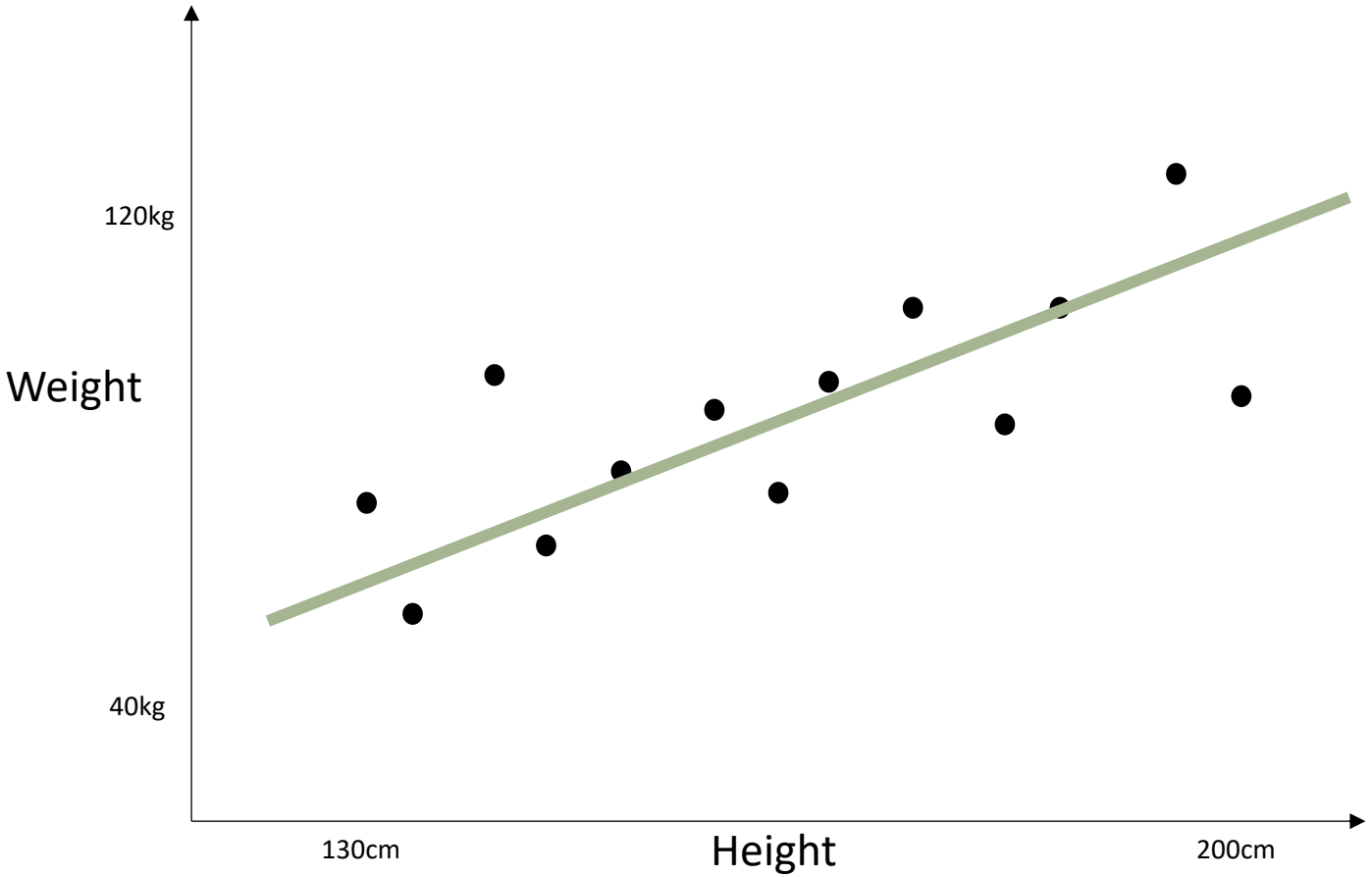
Motivation: height vs. weight

Q: Can we find a line that (approximately) fits the data?

- If we can find such a line, we can use it to make **predictions** (i.e., estimate a person's weight given their height)
  - How do we **formulate** the problem of finding a line?
  - If no line will fit the data exactly, how to **approximate?**
    - What is the "best" line?

Recap: equation for a line

What is the formula describing the line?

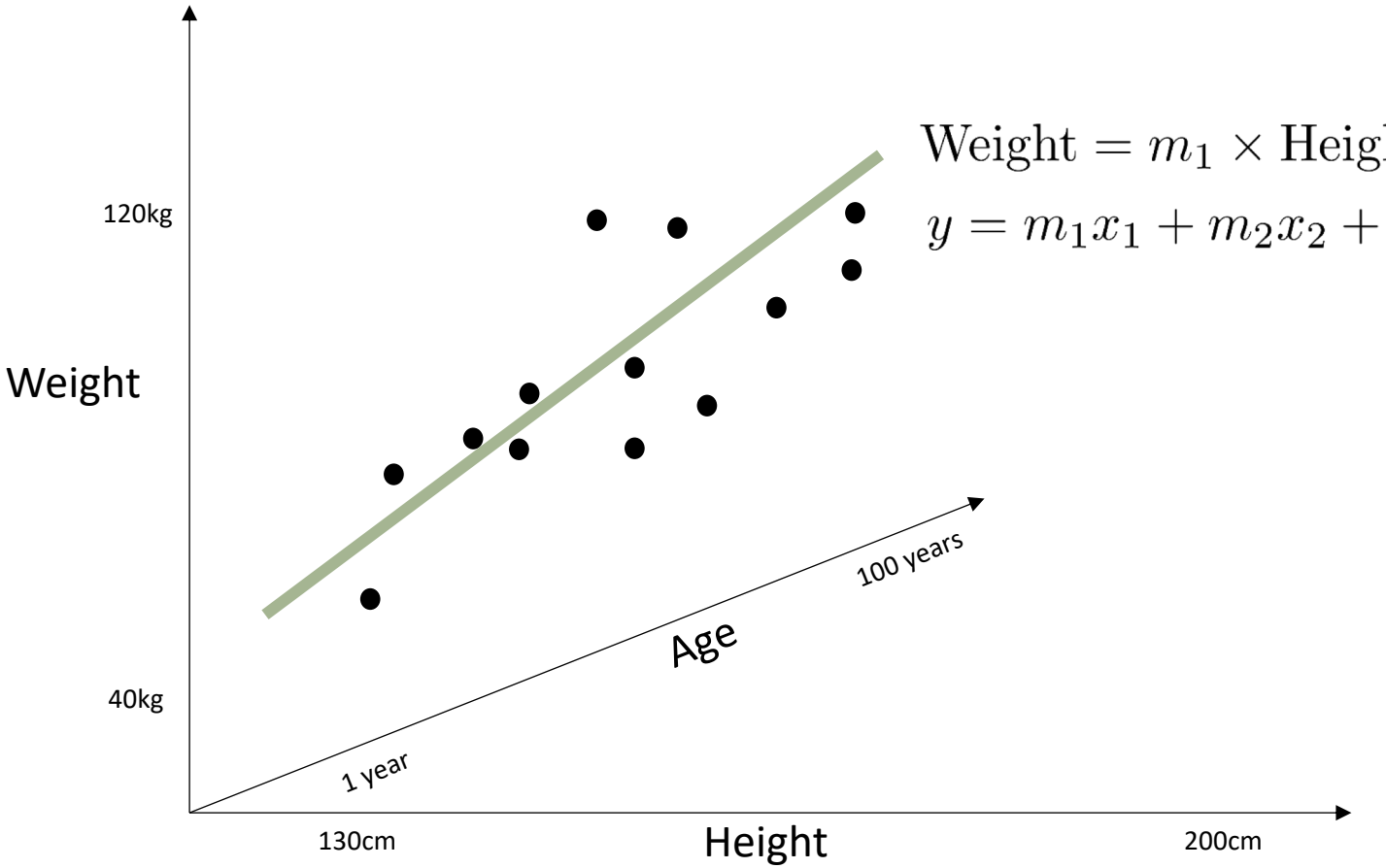


$$y = mx + b$$

$$\text{Weight} = m \times \text{Height} + b$$

Recap: equation for a line

What about in more dimensions?

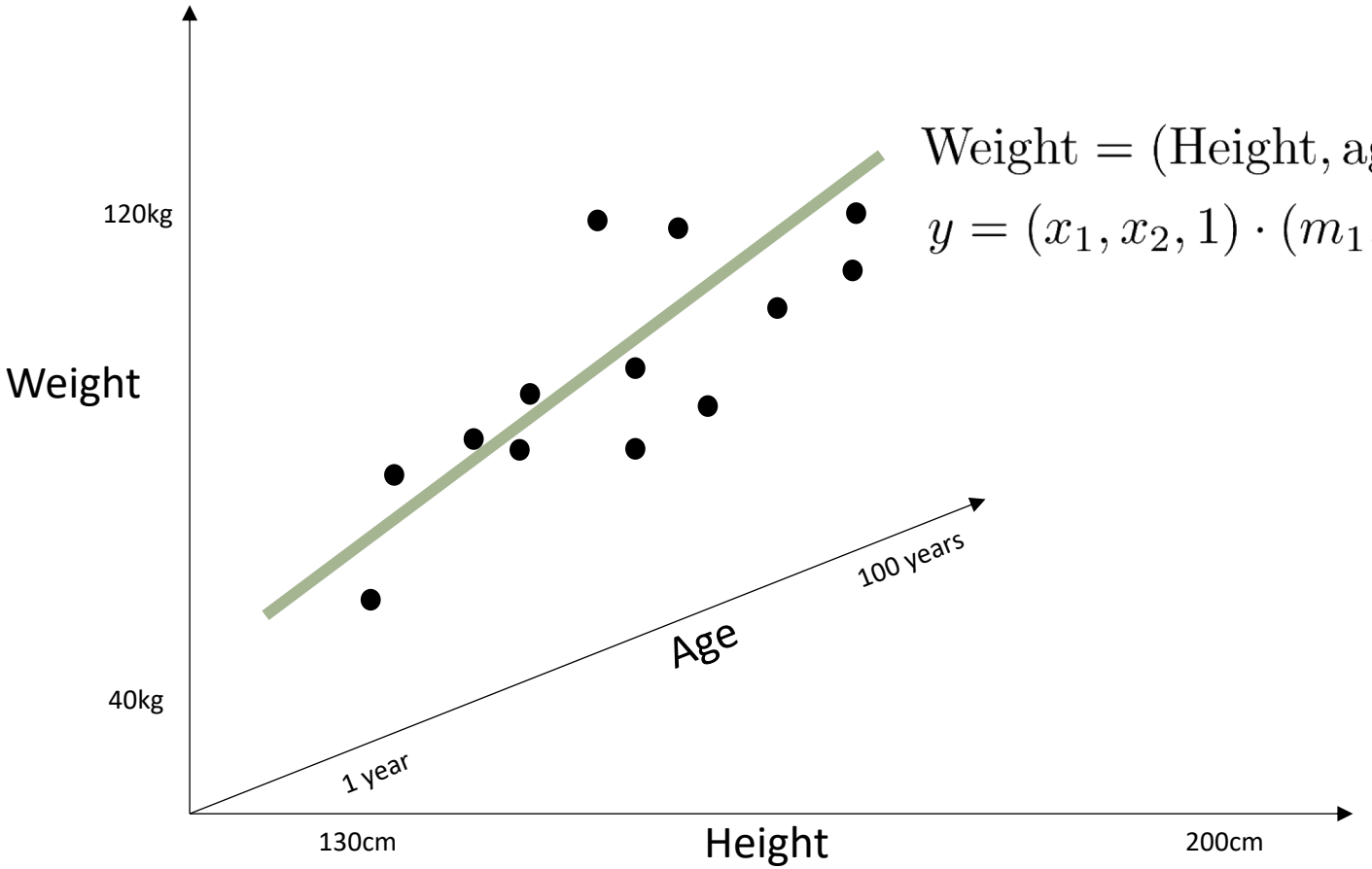


$$\text{Weight} = m_1 \times \text{Height} + m_2 \times \text{age} + b$$

$$y = m_1 x_1 + m_2 x_2 + b$$

Recap: equation for a line as an inner product

What about in more dimensions?



$$\text{Weight} = (\text{Height}, \text{age}, 1) \cdot (m_1, m_2, b)$$

$$y = (x_1, x_2, 1) \cdot (m_1, m_2, b)$$

Linear regression

In general:

$$y = x \cdot \theta$$

What we want to predict **(label)**

What we can use to predict it **(features)**

Parameters that we can tune to make the prediction

Linear regression

But here we have **many** observations, so we can rewrite using a matrix:

$$y_i = X_i \cdot \theta$$

$$y = X\theta$$

Vector of parameters

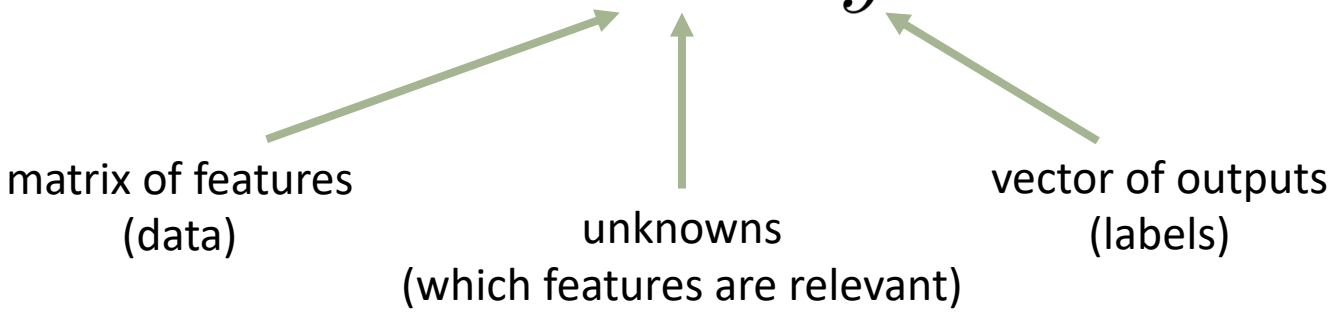
Vector of labels

Matrix of features

Concept: Linear regression

Linear regression assumes a predictor of the form

$$X\theta = y$$



(or $Ax = b$ if you prefer)

Linear regression

Linear regression assumes a predictor of the form

$$X\theta = y$$

Q: Solve for theta

A:

Linear regression

Linear regression assumes a predictor of the form

$$X\theta = y$$

Q: Solve for theta

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

Summary of concepts

- **Regression** is one of the simplest forms of supervised learning
- **Linear regression** is essentially equivalent to finding a line that best fits the data
- Can express this as solving a system of matrix equations

On your own...

- Try writing down the regression equation (equation for a line) for a similar problem, e.g. predicting a person's income from various features

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Regression in Python

Learning objectives

In this lecture we will...

- Explore how to express linear regression equations in terms of Python data structures
- Work through a (simple) **real-world regression example**
- Compare a "manual" implementation of linear regression to a library function

Example – Air quality prediction

We'll look at the problem of predicting **air quality,** using an index called pm2.5, measured in Beijing

- This is a "simpler" dataset than some of the others we've been working with, as the relevant features are all real-valued
- It's also useful in our following lecture (on time-series prediction), since the data is in the form of a time series

Example – UCI Dataset Repository

Beijing PM2.5 Data Data Set

Download: Data Folder, Data Set Description

Abstract: This hourly data set contains the PM2.5 data of US Embassy in Beijing. Meanwhile, meteorological data from Beijing Capital International Airport are also included.

Data Set Characteristics:	Multivariate, Time-Series	**Number of Instances:**	43824	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	2017-01-19
Associated Tasks:	Regression	**Missing Values?**	Yes	**Number of Web Hits:**	70637

Source:

Song Xi Chen, csx '@' gsm.pku.edu.cn, Guanghua School of Management, Center for Statistical Science, Peking University.

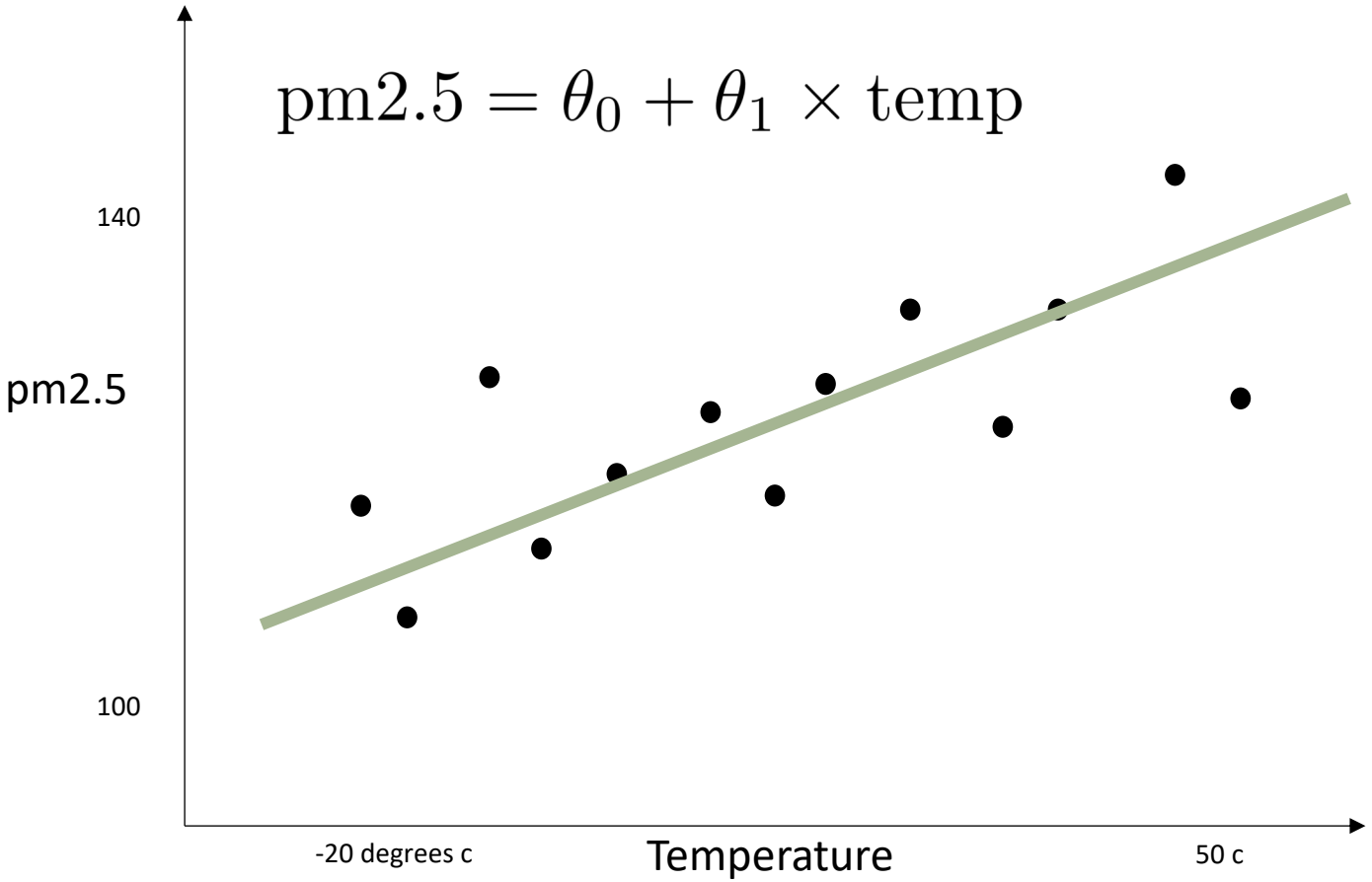
Data Set Information:

The dataâ€™s time period is between Jan 1st, 2010 to Dec 31st, 2014. Missing data are denoted as â€œNAâ€.

https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

What are we trying to predict?

E.g. pm2.5 vs. Temperature:



Code: Reading the file

In [2]:
```
path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')
```

In [3]:
```
dataset = []
header = f.readline().strip().split(',')
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [4]:
```
N = len(dataset)
N
```

Out[4]:
```
43824
```

In [5]:
```
header
```

Out[5]:
```
['No',
 'year',
 'month',
 'day',
 'hour',
 'pm2.5',
 'DEWP',
 'TEMP',
 'PRES',
 'cbwd',
 'Iws',
```

Label that we want to predict

Feature we want to use for prediction

Code: Extracting features and labels

```
In [6]: header.index('pm2.5')
Out[6]: 5

In [7]: header.index('TEMP')
Out[7]: 7

In [8]: y = [float(d[5]) for d in dataset]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-caefd9ca5537> in <module>()
----> 1 y = [float(d[5]) for d in dataset]

<ipython-input-8-caefd9ca5537> in <listcomp>(.0)
----> 1 y = [float(d[5]) for d in dataset]

ValueError: could not convert string to float: 'NA'
```

```
In [9]: dataset = [d for d in dataset if d[5] != 'NA']
```

Code: Let's try again...

In [10]:
```
y = [float(d[5]) for d in dataset]
```

In [11]:
```
def feature(datum):
    feat = [1, float(datum[7])]
    return feat
```

In [12]:
```
X = [feature(d) for d in dataset]
```

In [13]:
```
y[:10]
```

Out[13]:
```
[129.0, 148.0, 159.0, 181.0, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0]
```

In [14]:
```
X[:10]
```

Out[14]:
```
[[1, -4.0],
 [1, -4.0],
 [1, -5.0],
 [1, -5.0],
 [1, -5.0],
 [1, -6.0],
 [1, -6.0],
 [1, -5.0],
 [1, -6.0],
 [1, -5.0]]
```

Reminder: Constant feature

Why did we implement our feature function like this?

In [11]:
```
def feature(datum):
    feat = [1, float(datum[7])]
    return feat
```

Code: Finding the parameters

```
In [15]: theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [16]: theta
Out[16]: array([107.10183392,  -0.68447989])
```

pm2.5 = 107.1 - 0.68 * temp

Code: Adding more features

```
In [17]: def feature(datum):
             feat = [1, float(datum[7]), float(datum[8]), float(datum[10])]
             return feat

In [18]: X = [feature(d) for d in dataset]

In [19]: theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [20]: theta

Out[20]: array([ 3.26373064e+03, -3.10933772e+00, -3.06517728e+00, -4.60017221e-01])
```

Note: pressure

Note: wind speed

pm2.5 = 3263.7
- 3.109 * temp
- 3.065 * pressure
- 0.460 * wind speed

Code: Doing the same thing manually

```
In [20]: theta
```

```
Out[20]: array([ 3.26373064e+03, -3.10933772e+00, -3.06517728e+00, -4.60017221e-01])
```

```
In [21]: X = numpy.matrix(X)
         y = numpy.matrix(y)
         numpy.linalg.inv(X.T * X) * X.T * y.T
```

```
Out[21]: matrix([[ 3.26373064e+03],
                 [-3.10933772e+00],
                 [-3.06517728e+00],
                 [-4.60017221e-01]])
```

Summary of concepts

- Demonstrated how to perform simple linear regression in Python
- Performed linear regression on an "air quality" example from the UCI Machine Learning Repository
- Introduced the numpy "least squares" function for linear regression

On your own...

- Try extending the code provided here to use different features and feature combinations

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Time-series Regression (or "Autoregression")

Learning objectives

In this lecture we will...

- Explore options to apply regression to time-series data
- Consider the merits of alternative approaches
- Introduce the "autoregression" framework for time-series regression

Time-series regression

Here, we'd like to predict sequences of **real-valued** events as accurately as possible

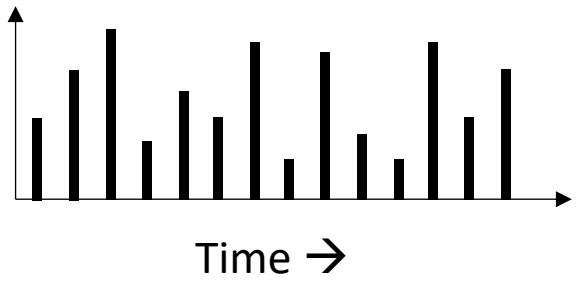
Given: a time series:

$$(x_1, \ldots, x_N) \in \mathbb{R}^N$$

Suppose we'd like to predict the next values in the sequence as accurately as possible

Time-series regression

Method 1: maintain a “moving average” using a window of some fixed length



Time-series regression

Method 1: maintain a "moving average" using a window of some fixed length

$$f(x_1, \ldots, x_m) = \frac{1}{K} \sum_{k=0}^{K-1} x_{m-k}$$

This can be computed efficiently via dynamic programming:

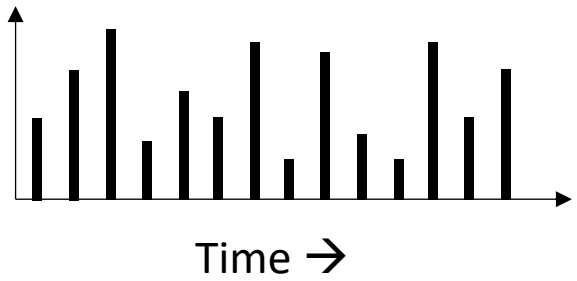
$$f(x_1, \ldots, x_{m+1}) = \frac{1}{K}(K \cdot f(x_1, \ldots, x_m) - x_{m-k} + x_{m+1})$$

"peel-off" the oldest point

add the newest point

Time-series regression

Method 2: weight the points in the moving average by age



Time-series regression

Method 2: weight the points in the moving average by age

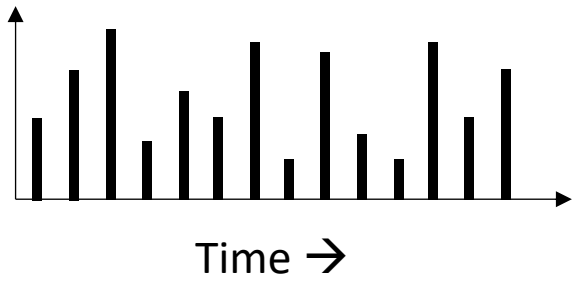
newest points have the highest weight

weight decays to zero after K points

$$f(x_1, \ldots, x_m) = \frac{\sum_{k=0}^{K-1} (K-k) x_{m-k}}{\binom{K}{2}}$$

Time-series regression

Method 3: weight the most recent points exponentially higher



Time-series regression

Method 3: weight the most recent points exponentially higher

$$f(x_1) = x_1$$

$$f(x_1, \ldots, x_m) = \alpha \cdot x_m + (1 - \alpha) f(x_1, \ldots, x_{m-1})$$

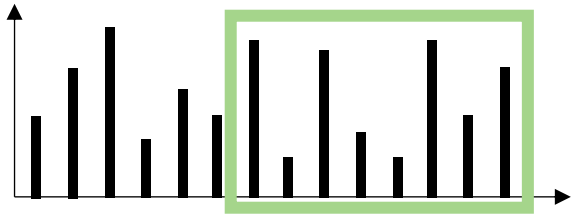
most recent point has weight \alpha

previous prediction has weight 1 - \alpha

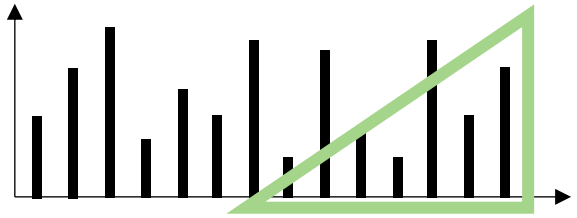
Methods 1, 2, 3

Method 1: Sliding window
Method 2: Linear decay
Method 3: Exponential decay

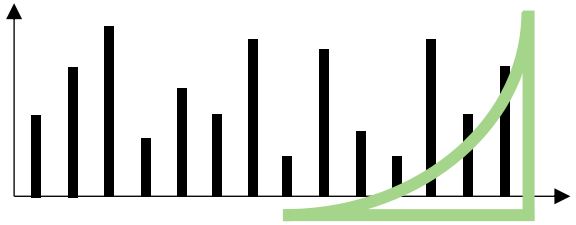
Method 1:



Method 2:



Method 3:



Time-series regression

Method 4: all of these models are assigning **weights** to previous values using some predefined scheme, why not just **learn** the weights?

$$f(x_1, \ldots, x_m) = \alpha + \sum_{k=0}^{K-1} \theta_k \cdot x_{m-k}$$

$$= \alpha + \langle \theta, (x_{m-(K-1)}, \ldots, x_m) \rangle$$

- We can now fit this model using least-squares
- This procedure is known as **autoregression**
- Using this model, we can capture **periodic** effects, e.g. that the traffic of a website is most similar to its traffic 7 days ago

Summary of concepts

- Introduced the problems of time-series prediction and "autoregression"
- Introduced and compared four techniques to solve the time-series prediction problem

On your own...

- By considering different types of data, consider in which scenarios (a) sliding windows, (b) linear decay, and (c) exponential decay would work better than each of the others

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Autoregression in Python

Learning objectives

In this lecture we will...

- Show a real-world example of autoregression in Python
- Adapt the regression code we previously developed to handle autoregressive problems

Time-series regression

We'll quickly adapt our regression code to use the autoregressive framework

- The air quality data (which we used previously) is made up of sequential (hourly) predictions
- So, we can predict the next air quality measurement from the previous ones

Code: Reading the data

In [1]:
```
import numpy
```

In [2]:
```
path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')
```

In [3]:
```
dataset = []
header = f.readline().strip().split(',')
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [4]:
```
dataset = [d for d in dataset if d[5] != 'NA']
```

(see "regression in python" lecture for further explanation)

Code: Extracting autoregressive features

```
In [5]: def feature(dataset, ind, windowSize):
            feat = [1]
            previousValues = [float(d[5]) for d in dataset[ind-windowSize:ind]]
            return feat + previousValues
```

Vector of previous (windowSize) observations

- Feature vector is made up of the previous pm2.5 observations
- The number of previous observations is provided as a configurable parameter

Code: Extracting autoregressive features

```
In [6]: windowSize = 10
        N = len(dataset)

In [7]: X = [feature(dataset, ind, windowSize) for ind in range(windowSize,N)]

In [8]: X[:10]

Out[8]: [[1, 129.0, 148.0, 159.0, 181.0, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0],
         [1, 148.0, 159.0, 181.0, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0, 140.0],
         [1, 159.0, 181.0, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0, 140.0, 152.0],
         [1, 181.0, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0, 140.0, 152.0, 148.0],
         [1, 138.0, 109.0, 105.0, 124.0, 120.0, 132.0, 140.0, 152.0, 148.0, 164.0],
         [1, 109.0, 105.0, 124.0, 120.0, 132.0, 140.0, 152.0, 148.0, 164.0, 158.0],
         [1, 105.0, 124.0, 120.0, 132.0, 140.0, 152.0, 148.0, 164.0, 158.0, 154.0],
         [1, 124.0, 120.0, 132.0, 140.0, 152.0, 148.0, 164.0, 158.0, 154.0, 159.0],
         [1, 120.0, 132.0, 140.0, 152.0, 148.0, 164.0, 158.0, 154.0, 159.0, 164.0],
         [1, 132.0, 140.0, 152.0, 148.0, 164.0, 158.0, 154.0, 159.0, 164.0, 170.0]]

In [9]: y = [float(d[5]) for d in dataset[windowSize:]]
```

Vector of previous (windowSize) observations

Code: Extracting autoregressive features

```
In [10]: theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [11]: theta

Out[11]: array([ 3.92354307,  0.01296079,  0.00414674, -0.00925187,  0.00794837,
                -0.01458271, -0.0164066 ,  0.00626613,  0.04355583, -0.22994129,
                 1.15548897])
```

Note: Weight associated with **most recent** observation

Code: Extracting autoregressive features

```
In [12]: def feature(dataset, ind, windowSize):
             feat = [1, float(dataset[ind][7]), float(dataset[ind][8]), float(dataset[ind][10])]
             previousValues = [float(d[5]) for d in dataset[ind-windowSize:ind]]
             return feat + previousValues
```

Note: Features for temperature, pressure, and wind-speed

```
In [13]: X = [feature(dataset, ind, windowSize) for ind in range(windowSize,N)]

In [14]: theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [15]: theta

Out[15]: array([ 1.66261451e+02, -1.71855736e-01, -1.56441424e-01, -2.50471317e-02,
                 1.50765140e-02,  3.66638827e-03, -9.37757357e-03,  7.43748773e-03,
                -1.50878276e-02, -1.67895315e-02,  5.71176519e-03,  4.27640539e-02,
                -2.29558932e-01,  1.15031328e+00])
```

- Note that we don't need to use autoregression **or** regular regression exclusively
- We can include both types of features simultaneously!

Summary of concepts

- Demonstrated how to perform autoregression in Python

On your own...

- Experiment with different sliding window sizes and their impact on performance
- Try alternative approaches (e.g. sliding windows) and compare them to autoregression

Week 2

Feature Engineering

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Features from categorical data

Learning objectives

In this lecture we will...

- Demonstrate how to incorporate binary and categorical features into regressors
- Compare the benefits of various feature representation strategies

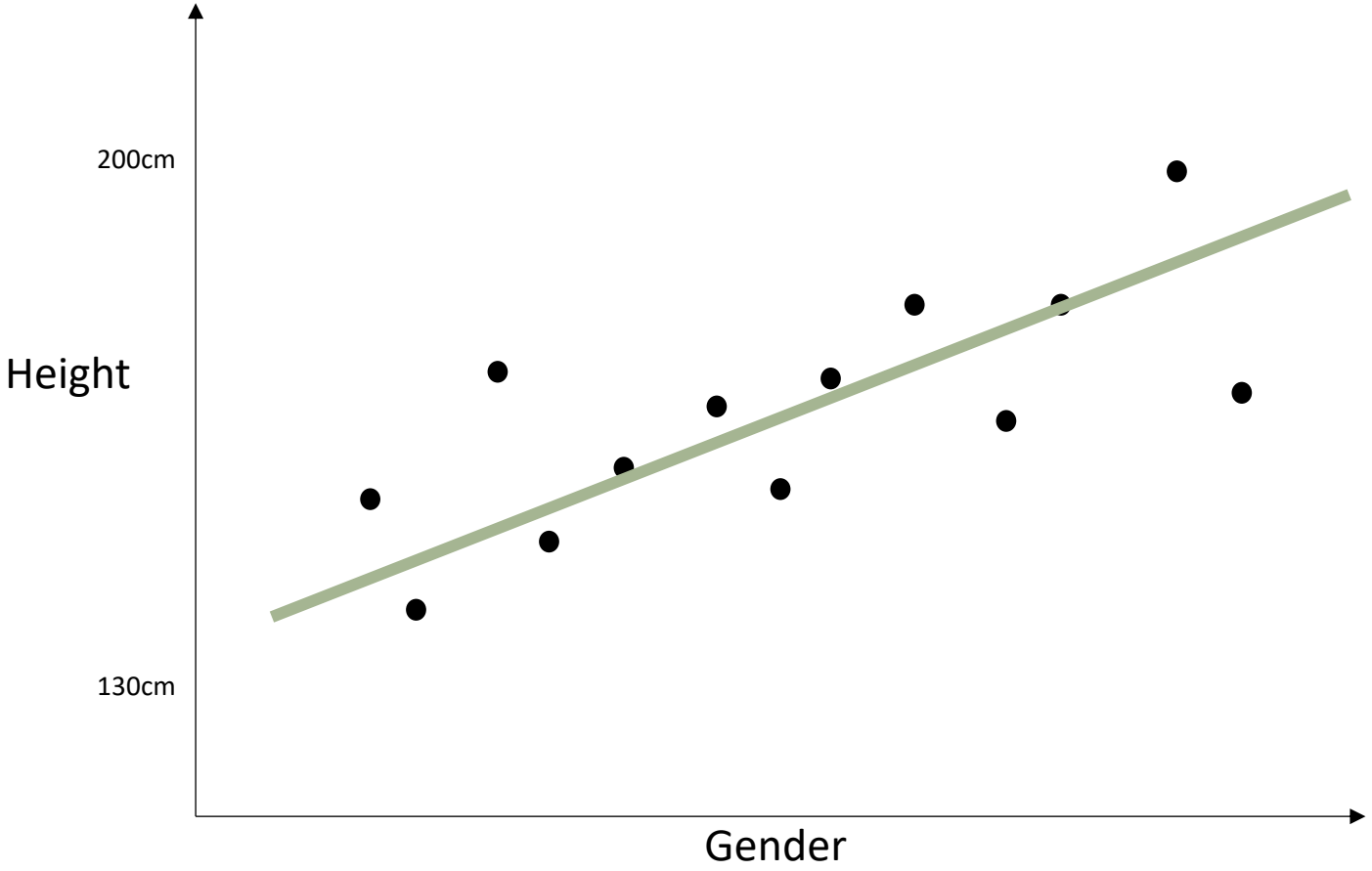
Motivating examples

How would we build regression models that incorporate features like:

- How does height vary with **gender?**
- How do preferences vary with **geographical region?**
- How does product demand change during different **seasons?**

Motivating examples

E.g. How does height vary with **gender?**



Motivating examples

E.g. How does height vary with **gender?**

- Previous picture doesn't quite make sense: we're unlikely to have a dataset including a continuum of gender values, so fitting a "line" doesn't seem to fit
- So how can we deal with this type of data using a linear regression framework?

Motivating examples

E.g. How does height vary with **gender?**

- Presumably our gender values might look more like
  {"male", "female", "other", "not specified"}
- **Let's first start with a binary problem where we just have {"male", "female"}**

Motivating examples

What should our **model equation** look like?

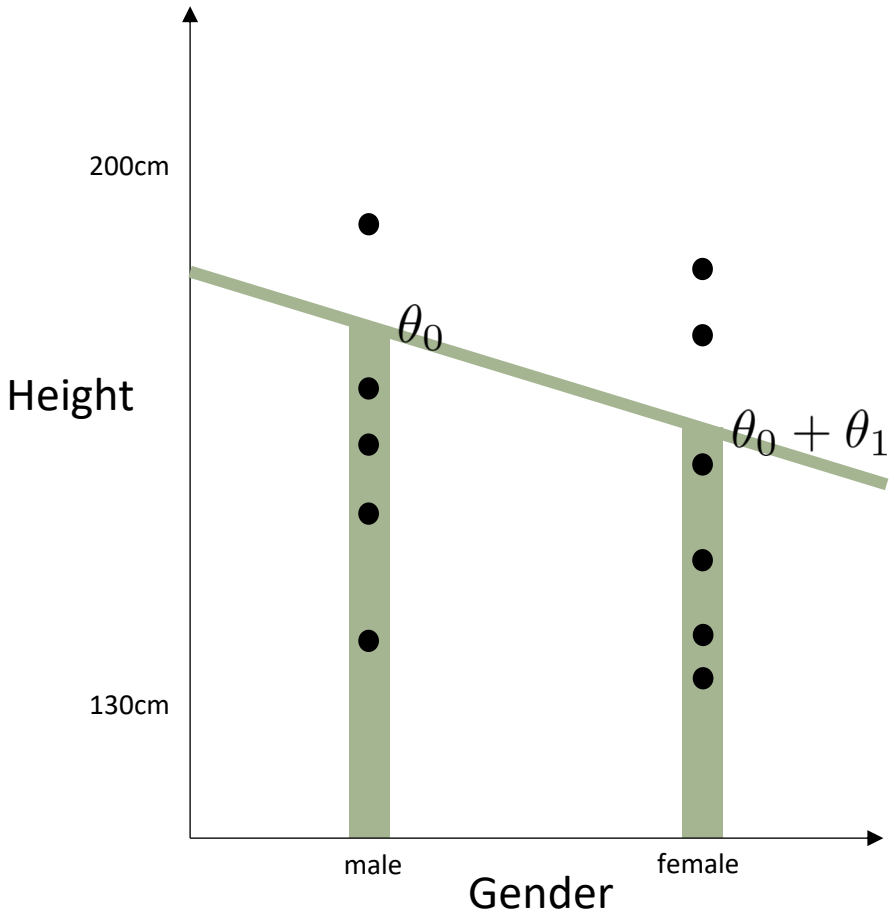
$$\text{Height} = \theta_0 + \theta_1 \times \text{gender}$$

gender = **0 if male, 1 if female**

$$\text{Height} = \theta_0 \quad \textbf{if male}$$

$$\text{Height} = \theta_0 + \theta_1 \quad \textbf{if female}$$

Motivating examples



θ_0 is the (predicted/average) height for males

θ_1 is the **how much taller** females are than males (in this case a negative number)

We're really still fitting a line though!

Motivating examples

What if we had more than two values?
(e.g {“male”, “female”, “other”, “not specified”})

Could we apply the same approach?

$$\text{Height} = \theta_0 + \theta_1 \times \text{gender}$$

gender = **0 if “male”, 1 if “female”, 2 if “other”, 3 if “not specified”**

$\text{Height} = \theta_0$ **if male**

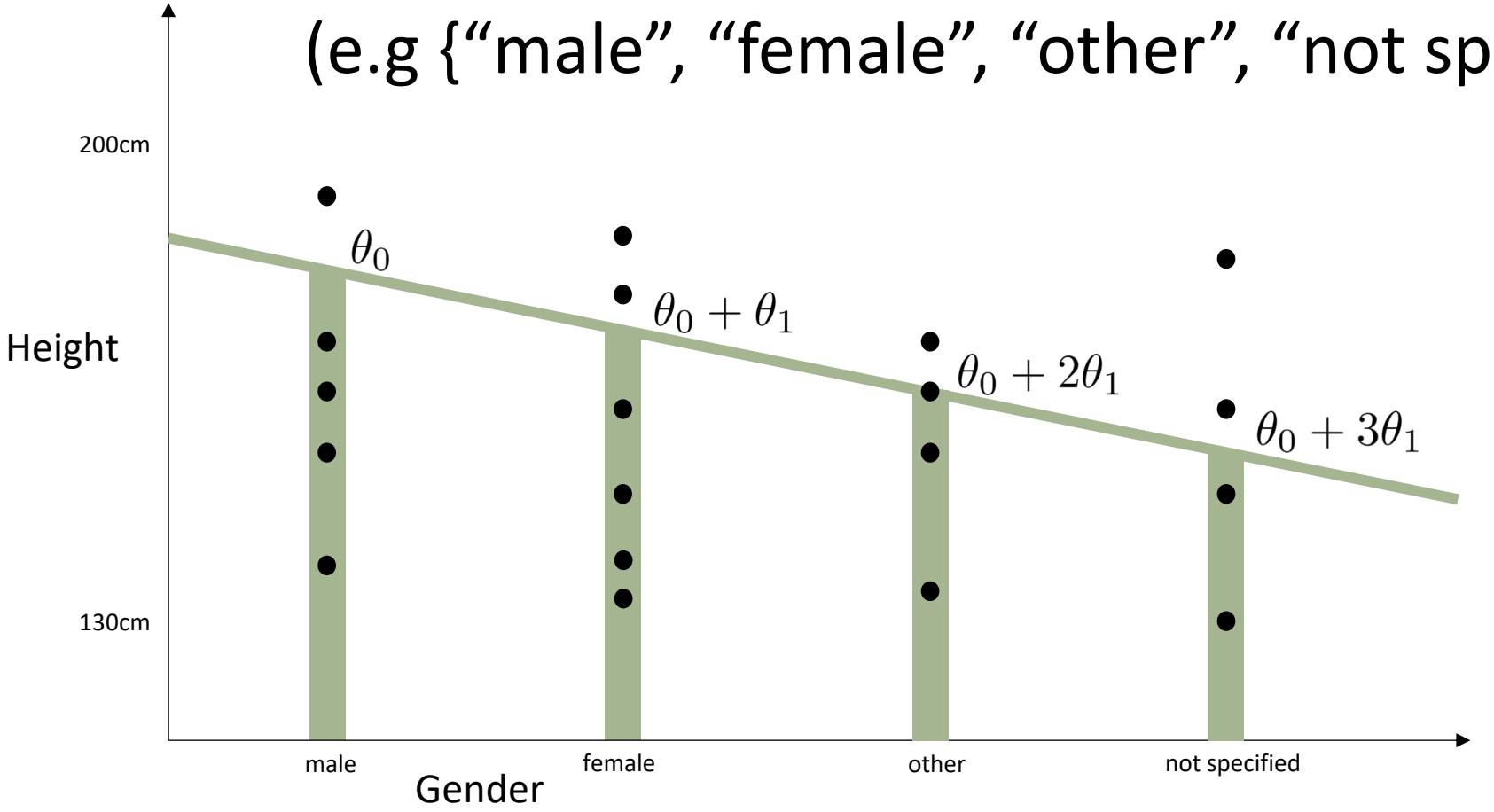
$\text{Height} = \theta_0 + \theta_1$ **if female**

$\text{Height} = \theta_0 + 2\theta_1$ **if other**

$\text{Height} = \theta_0 + 3\theta_1$ **if not specified**

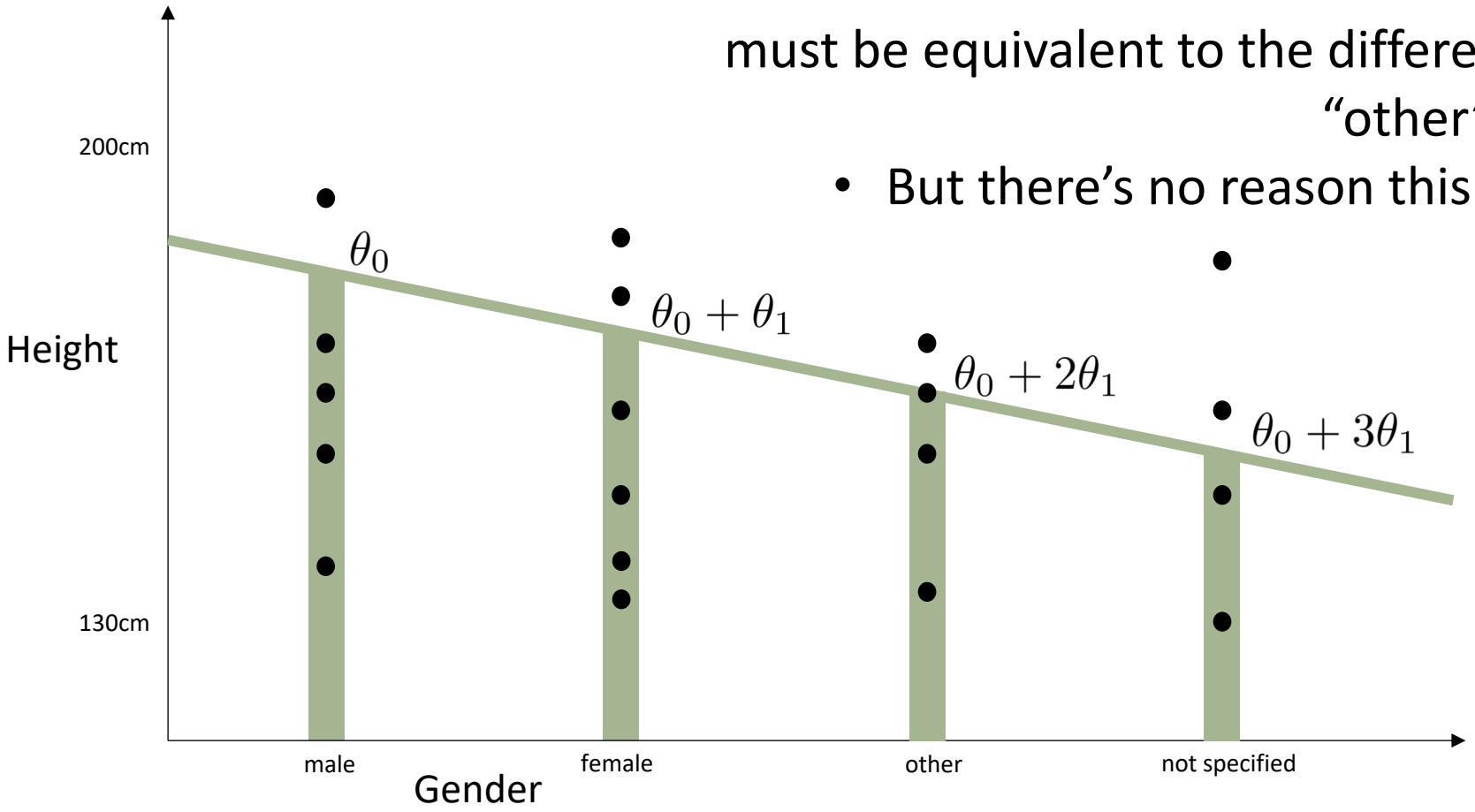
Motivating examples

What if we had more than two values?
(e.g {“male”, “female”, “other”, “not specified”})



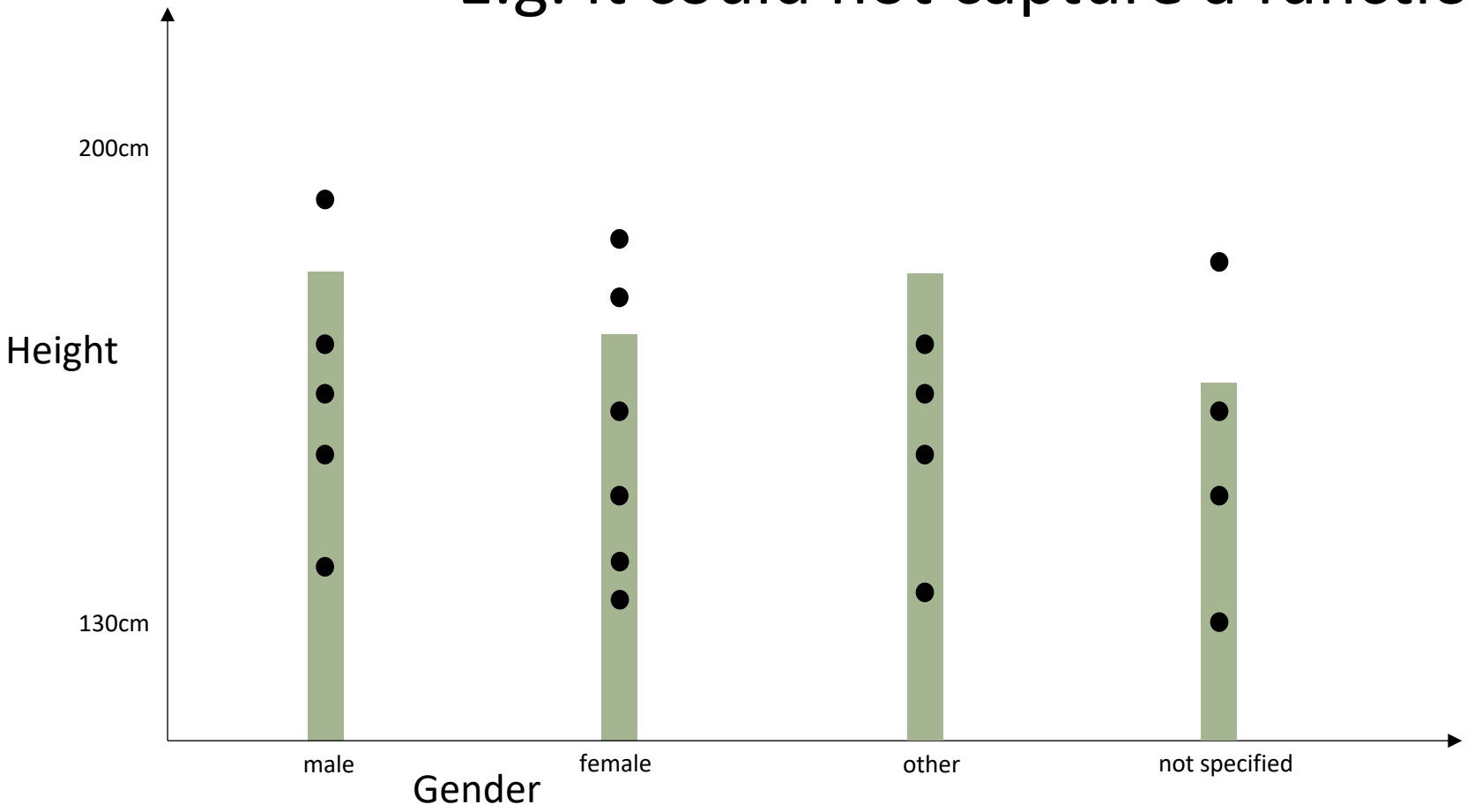
Motivating examples

- This model is **valid,** but won't be very **effective**
- It assumes that the difference between "male" and "female" must be equivalent to the difference between "female" and "other"
- But there's no reason this should be the case!



Motivating examples

E.g. it could not capture a function like:



Motivating examples

Instead we need something like:

$\text{Height} = \theta_0$ **if male**

$\text{Height} = \theta_0 + \theta_1$ **if female**

$\text{Height} = \theta_0 + \theta_2$ **if other**

$\text{Height} = \theta_0 + \theta_3$ **if not specified**

Motivating examples

This is equivalent to:

$$(\theta_0, \theta_1, \theta_2, \theta_3) \cdot (1; \text{feature})$$

where feature = [1, 0, 0] for “female”
feature = [0, 1, 0] for “other”
feature = [0, 0, 1] for “not specified”

Concept: One-hot encodings

feature = [1, 0, 0] for “female”
feature = [0, 1, 0] for “other”
feature = [0, 0, 1] for “not specified”

- This type of encoding is called a **one-hot encoding** (because we have a feature vector with only a single “1” entry
- Note that to capture 4 possible categories, we only need three dimensions (a dimension for “male” would be redundant)
- This approach can be used to capture a variety of categorical feature types, as well as objects that belong to multiple categories

Summary of concepts

- Described how to capture binary and categorical features within linear regression models
- Introduced the concept of a “one-hot” encoding

On your own...

- Think how you would encode different categorical features, e.g. the set of categories a business belongs to, or the set of a user’s friends on a social network

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Features from temporal data

Learning objectives

In this lecture we will...

- Investigate different strategies for extracting features from temporal (or seasonal) data
- Extend the concept of one-hot-encodings to represent temporal information

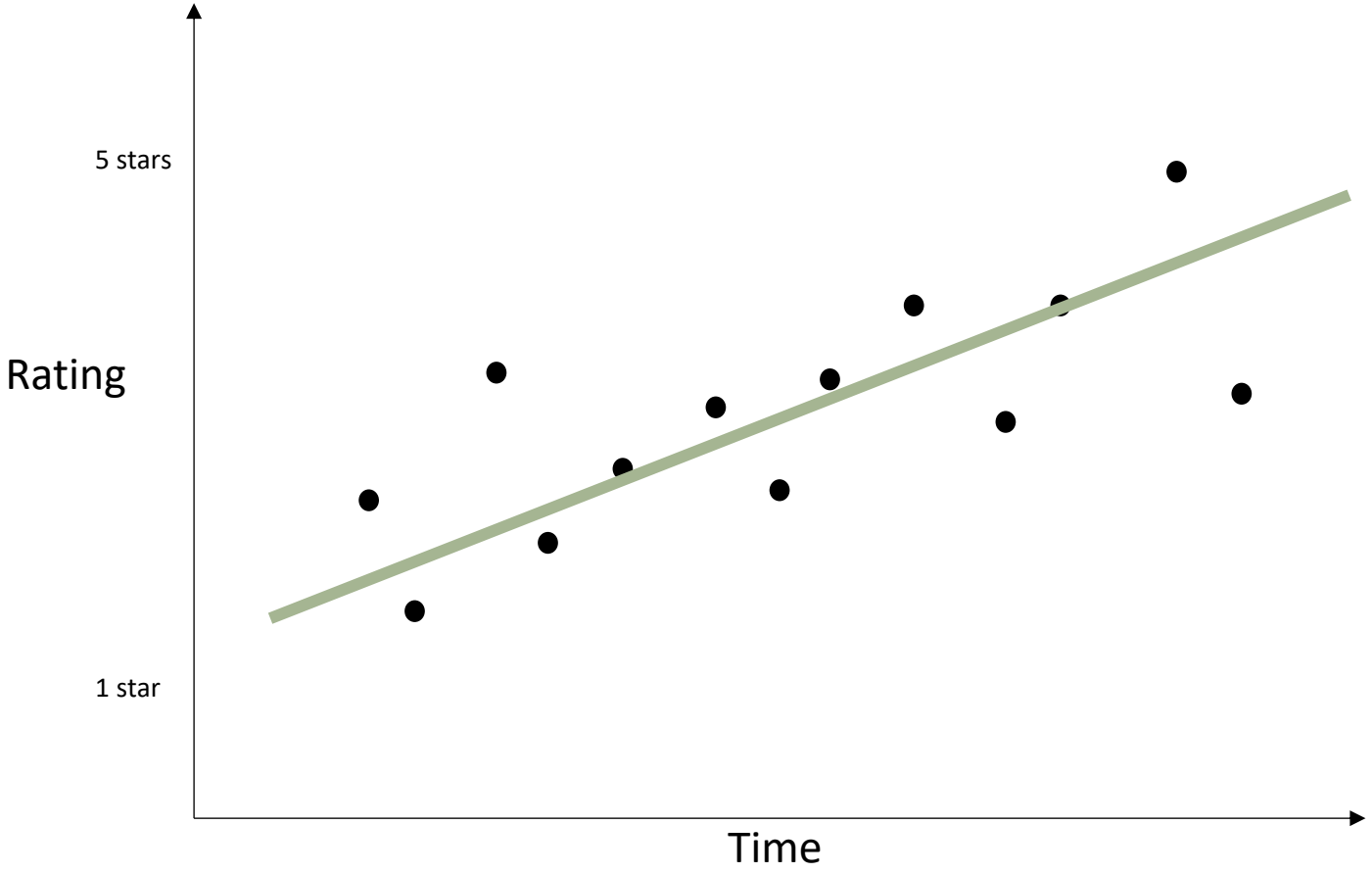
Motivating examples

How would we build regression models that incorporate features like:

- How do sales (or preferences) vary over time?
- What are the **long term** trends of sales?
- What are the **short term** trends (e.g. day of the week, season, etc.)

Motivating examples

E.g. How do **ratings** vary with **time**?



Motivating examples

E.g. How do **ratings** vary with **time**?

- In principle this picture looks okay (compared our previous lecture on categorical features) – we're predicting a **real valued** quantity from **real valued** data (assuming we convert the date string to a number)
- So, what would happen if (e.g. we tried to train a predictor based on the month of the year)?

Motivating examples

E.g. How do **ratings** vary with **time**?

- Let's start with a simple feature representation, e.g. map the month name to a month number:

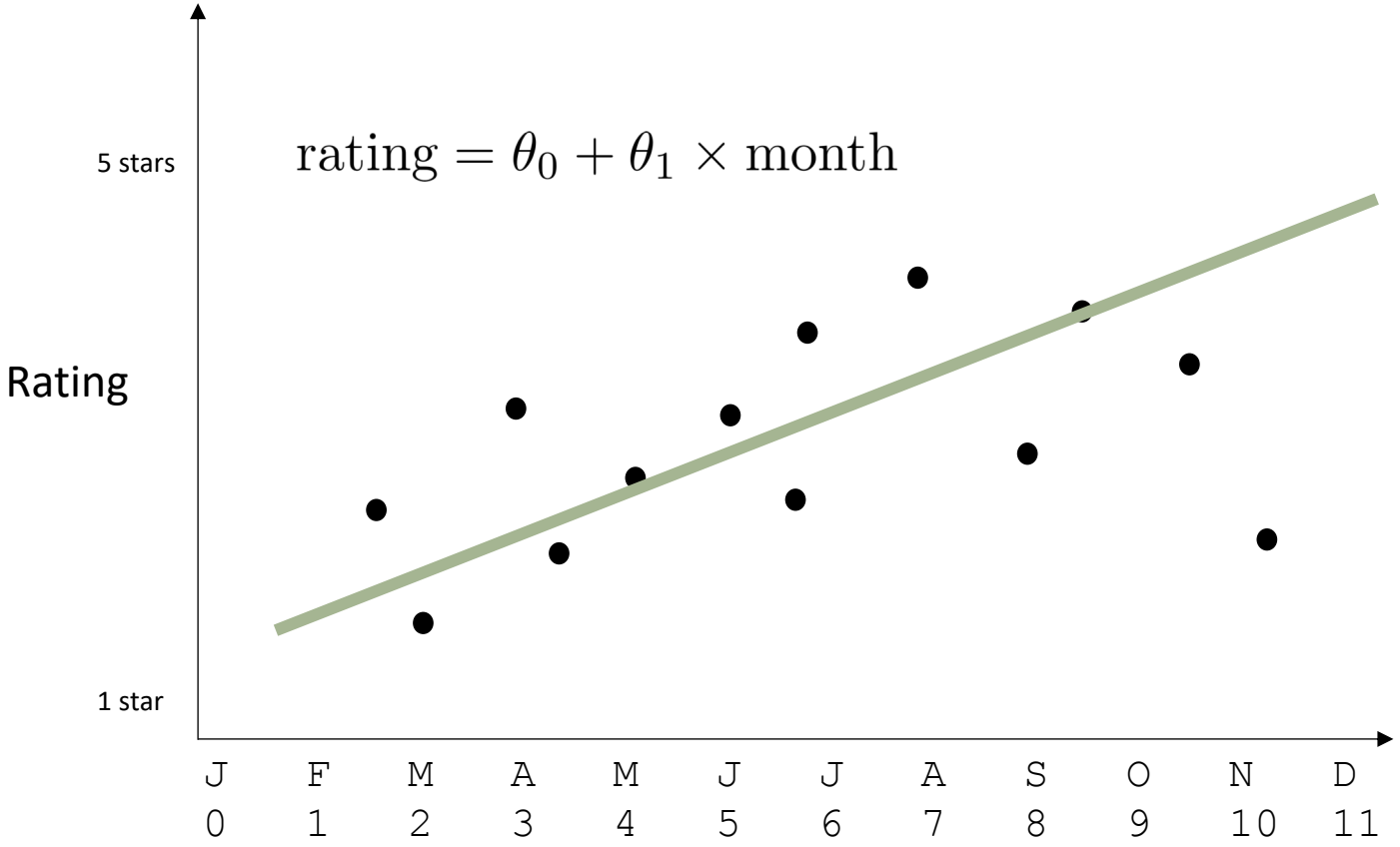
$$\text{rating} = \theta_0 + \theta_1 \times \text{month}$$

where

```
Jan = [0]
Feb = [1]
Mar = [2]
etc.
```

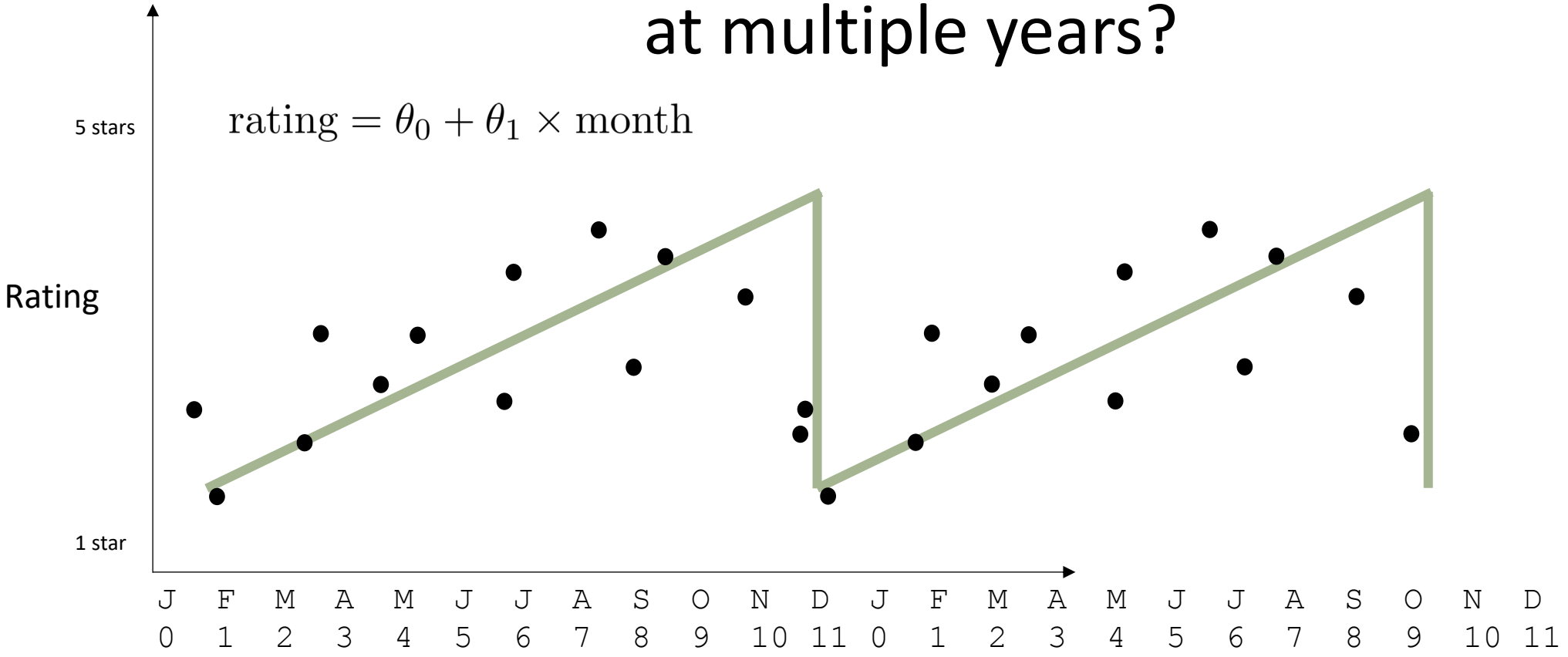
Motivating examples

The model we'd learn might look something like:



Motivating examples

This seems fine, but what happens if we look at multiple years?



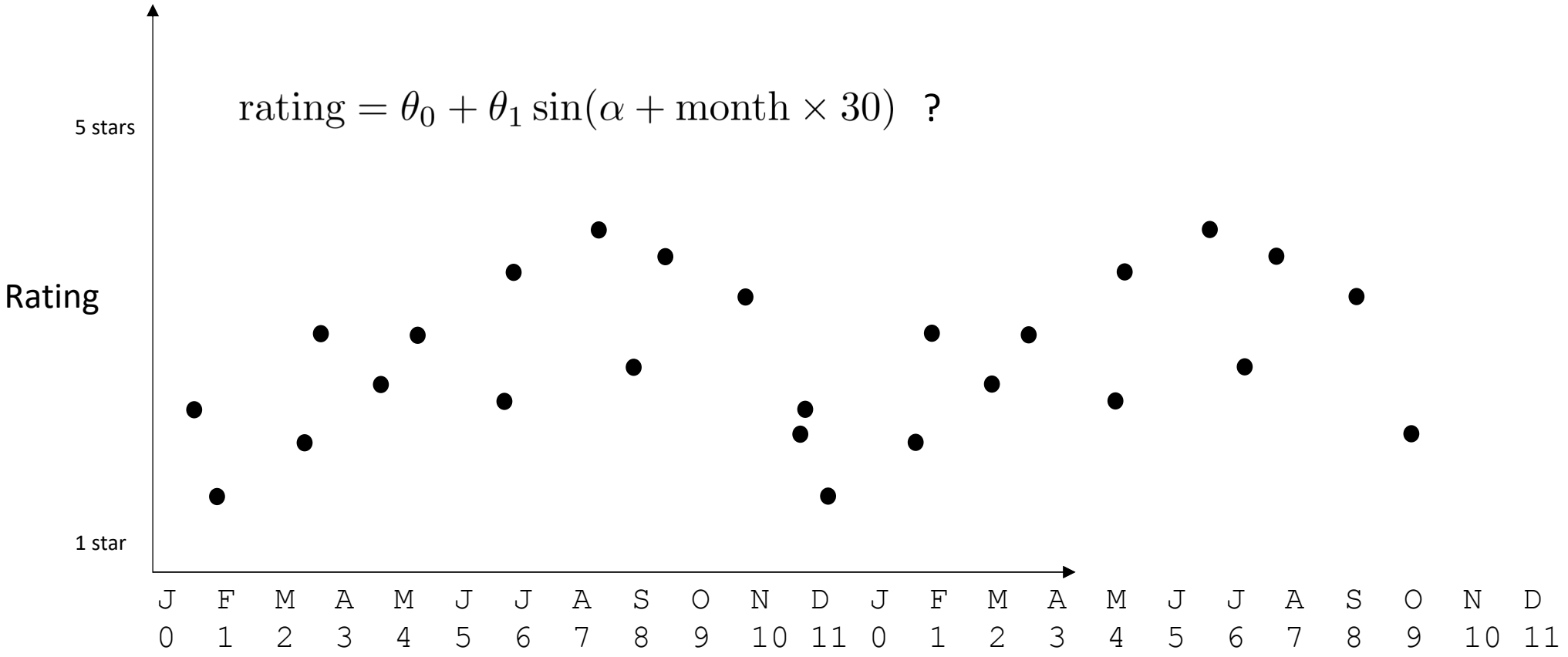
Modeling temporal data

This seems fine, but what happens if we look at multiple years?

- This representation implies that the model would "wrap around" on December 31 to its January 1st value.
- This type of "sawtooth" pattern probably isn't very realistic

Modeling temporal data

What might be a more realistic shape?



Modeling temporal data

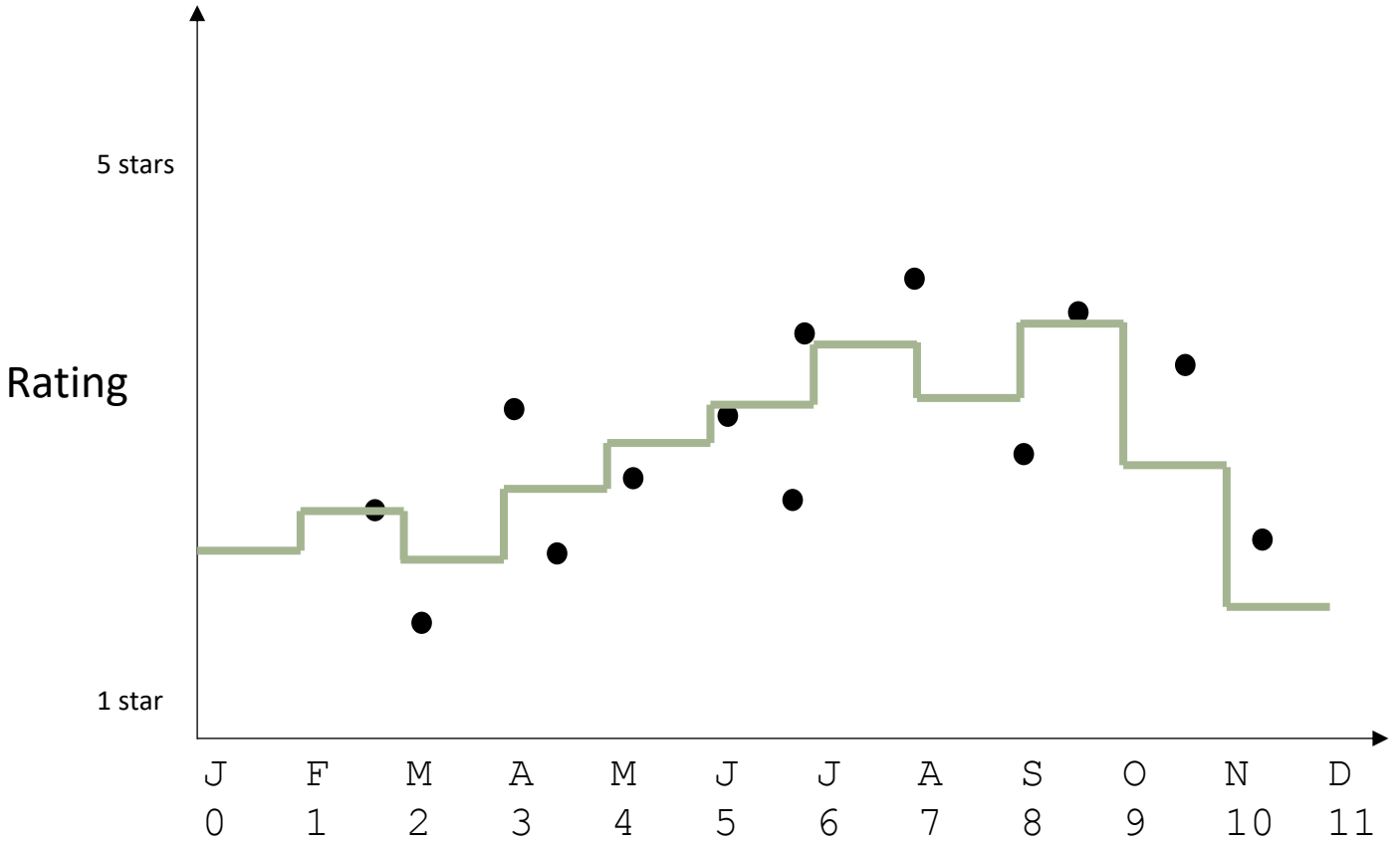
Fitting some periodic function like a sin wave would be a valid solution, but is difficult to get right, and fairly inflexible

- Also, it's not a **linear model**

- **Q:** What's a class of functions that we can use to capture a more flexible variety of shapes?
- **A:** Piecewise functions!

Concept: Fitting piecewise functions

We'd like to fit a function like the following:



Fitting piecewise functions

In fact this is very easy, even for a linear model! This function looks like:

$$\text{rating} = \theta_0 + \theta_1 \times \delta(\text{is Feb}) + \theta_2 \times \delta(\text{is Mar}) + \theta_3 \times \delta(\text{is Apr}) \cdots$$

1 if it's Feb, 0 otherwise

- Note that we don't need a feature for January
- i.e., theta_0 captures the January value, theta_0 captures the *difference* between February and January, etc.

Fitting piecewise functions

Or equivalently we'd have features as follows:

$$\text{rating} = \theta \cdot x \quad \text{where}$$

```
x = [1,1,0,0,0,0,0,0,0,0,0,0] if February
    [1,0,1,0,0,0,0,0,0,0,0,0] if March
    [1,0,0,1,0,0,0,0,0,0,0,0] if April
    ...
    [1,0,0,0,0,0,0,0,0,0,0,1] if December
```

Fitting piecewise functions

Note that this is still a form of **one-hot** encoding, just like we saw in the “categorical features” lecture

- This type of feature is very flexible, as it can handle complex shapes, periodicity, etc.
- We could easily increase (or decrease) the resolution to a week, or an entire season, rather than a month, depending on how fine-grained our data was

Concept: Combining one-hot encodings

We can also extend this by combining several one-hot encodings together:

$$\text{rating} = \theta \cdot x = \theta \cdot [x_1; x_2]$$ where

```
x1 = [1,1,0,0,0,0,0,0,0,0,0,0] if February
     [1,0,1,0,0,0,0,0,0,0,0,0] if March
     [1,0,0,1,0,0,0,0,0,0,0,0] if April
     ...
     [1,0,0,0,0,0,0,0,0,0,0,1] if December
```

```
x2 = [1,0,0,0,0,0] if Tuesday
     [0,1,0,0,0,0] if Wednesday
     [0,0,1,0,0,0] if Thursday
     ...
```

Summary of concepts

- Motivated the use of piecewise functions to model temporal data
- Described how one-hot encodings can be used to model piecewise functions

On your own...

- Think about what piecewise functions you might use to model demand on Amazon
  - Is the day of the week important?
  - Or the day of the month?
- How would you incorporate significant holidays (which may influence demand) into this model?

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Feature transformations

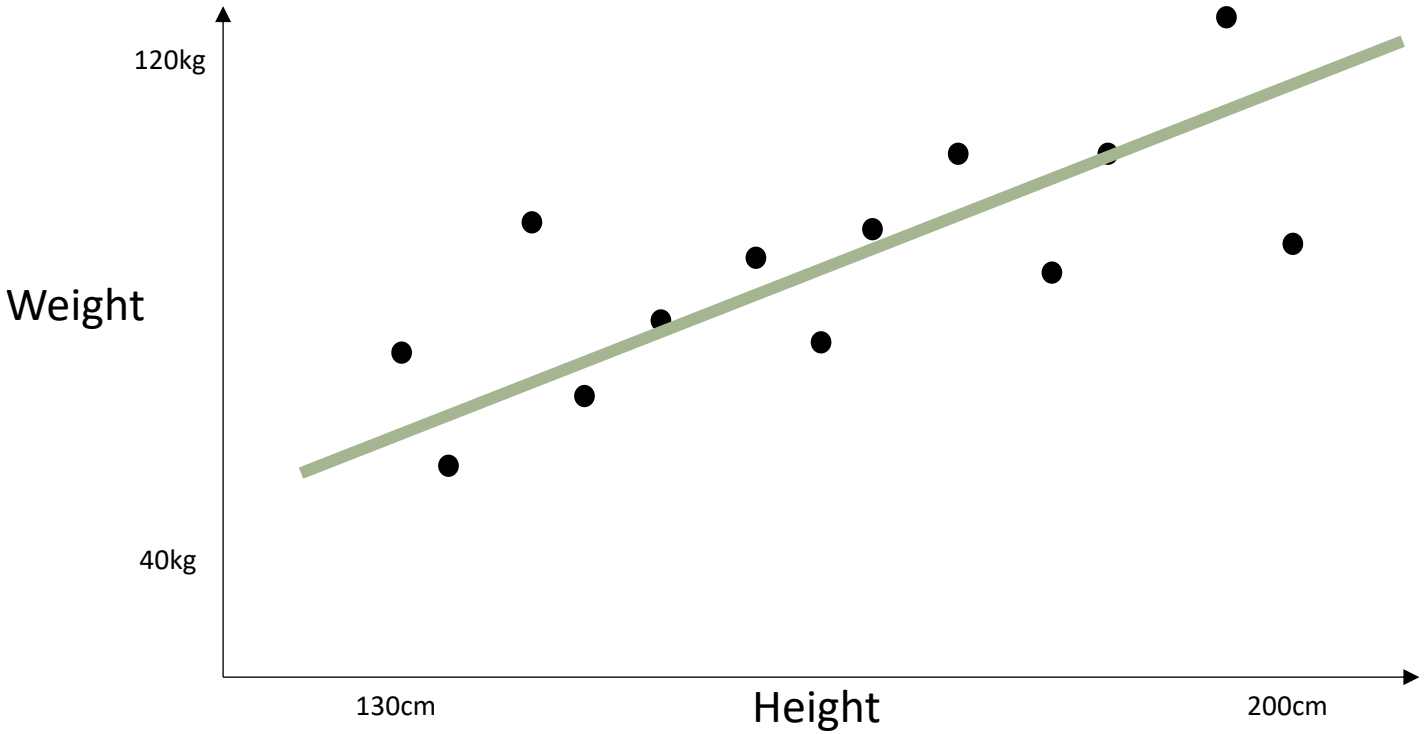
Learning objectives

In this lecture we will...

- Demonstrate the use of **transformations** to incorporate non-linear functions into linear models

Motivating example

We've previously seen simple examples of regression models such as Weight vs. Height:



Motivating example

We've previously seen simple examples of regression models such as Weight vs. Height:

- A linear relationship is probably *okay* for modeling this data, and in practice we'd often get away with using this type of model
- But, it certainly makes some assumptions that aren't totally justified
- How can we fit more suitable, or more general functions

Motivating example

How *should* the right model look for weight vs. height?

- A linear function?
- A quadratic or polynomial function?
- An asymptotic function?
- etc.

Fitting complex functions

Let's start with a polynomial function (e.g. a cubic function):

$$\text{weight} = \theta_0 + \theta_1 \times \text{height} + \theta_2 \times \text{height}^2 + \theta_3 \times \text{height}^3$$

- Note that this is perfectly straightforward: the model still takes the form

$$\text{weight} = \theta \cdot x$$

- We just need to use the feature vector

```
x = [1, height, height^2, height^3]
```

Fitting complex functions

Note that we can use the same approach to fit arbitrary functions of the features! E.g.:

$$\text{weight} = \theta_0 + \theta_1 \times \text{height} + \theta_2 \times \text{height}^2 + \theta_3 \exp(\text{height}) + \theta_4 \sin(\text{height})$$

- We can perform arbitrary combinations of the **features** and the model will still be linear in the **parameters** (theta):

$$\text{weight} = \theta \cdot x$$

Fitting complex functions

The same approach would **not** work if we wanted to transform the parameters:

$$\text{weight} = \theta_0 + \theta_1 \times \text{height} + \theta_2^2 \times \text{height} + \sigma(\theta_3) \times \text{height}$$

- The **linear** models we've seen so far do not support these types of transformations (i.e., they need to be linear in their parameters)
- There *are* alternative models that support non-linear transformations of parameters, e.g. neural networks

Summary of concepts

- Showed how to apply arbitrary transformations to features in a linear model
- Further explained the restrictions and assumptions of **linear models**

On your own...

- Extend our previous code (on pm2.5 levels vs. air temperature) to handle simple polynomial functions

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Missing values

Learning objectives

In this lecture we will...

- Introduce the issues around datasets with missing values
- Investigate different strategies for dealing with missing values in datasets

Motivation

- Even the simple PM2.5 dataset we introduced had missing values (indicated by "NA")
- So far we dealt with them simply by **discarding** those instances:

```
In [4]: dataset = [d for d in dataset if d[5] != 'NA']
```

- This was an okay strategy when dealing with a single feature where missing data was rare, but how would in generalize?
- In particular, this approach wouldn't work if **many** features might be missing

Concept: Strategies for dealing with missing values

In this lecture we'll look at three strategies for dealing with missing data:

- **Filtering** (I.e., discarding missing values), as we discussed on the previous slide
- **Missing data imputation:** filling in the missing values with "reasonable" estimates
- **Modeling:** changing our regression/classification algorithms to handle missing data explicitly

Missing data imputation

Even in cases where only a small amount of data is missing, simply discarding instances may not be an option. What else can we do?

Missing data imputation seeks to replace missing values by reasonable estimates

Missing data imputation

A simple scheme would be to replace every missing value with the **average** for that feature.

What are the consequences of such a scheme?

- The average may be sensitive to outlying values (though this could be addressed by using the **median** instead)
- The imputed value may or may not be "reasonable" (e.g. consider our "gender = male" feature)

Missing data imputation

Alternately we could consider more sophisticated data imputation schemes

- Rather than imputing using the mean, does it make more sense to compute the mean **of a certain subgroup** (e.g. if "height" is missing, can we impute using the average height of users with the same gender?)
- We could also train a separate **predictor** to impute the missing values (though this is complex if there are missing values for many different features)

Modeling missing data

How can we directly model the missing values within a regression or classification algorithm?

- One simple scheme: add an **additional feature** indicating that a value is missing
- e.g.:

  feature = [1, 0, 0] for “female”
  feature = [0, 1, 0] for “male”
  feature = [0, 0, 1] for “feature missing”

Modeling missing data

What predictions does the model make under this scheme?

feature = [1, 0, 0] for “female”
feature = [0, 1, 0] for “male”
feature = [0, 0, 1] for “feature missing”

$$\theta \cdot \text{feature} = \theta_0 \text{ for female}$$
$$= \theta_1 \text{ for male}$$
$$= \theta_2 \text{ for “feature missing”}$$

Note that θ_2 **learns** what value should be predicted when this feature is missing

Summary of concepts

- Discussed some simple schemes for dealing with missing data
- Introduced the ideas of **data imputation** and **modeling missing data**

On your own...

- Extend our previous code (on pm2.5 levels vs. air temperature) to handle missing features (other than the pm2.5 measurement itself)
- Experiment with different missing data imputation schemes and note their effect on performance

Week 3

Classification

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Supervised Learning: Classification

Learning objectives

In this lecture we will...

- Introduce the idea of **classification**
- Compare classification and regression models
- Explain what types of problems can be solved using classification

Regression vs. classification

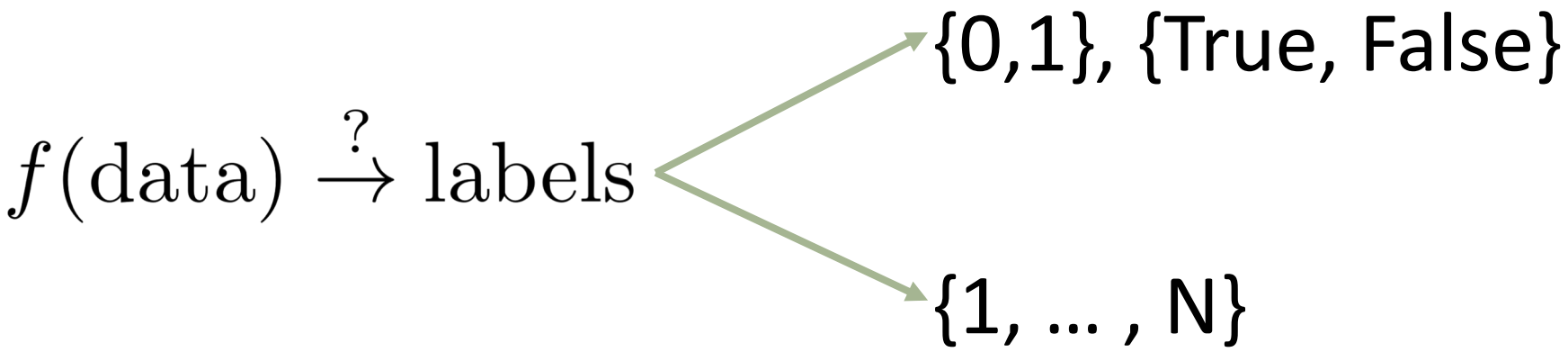
So far, we've studied **regression** problems that allow us to make predictions of the form

$$y = x \cdot \theta$$

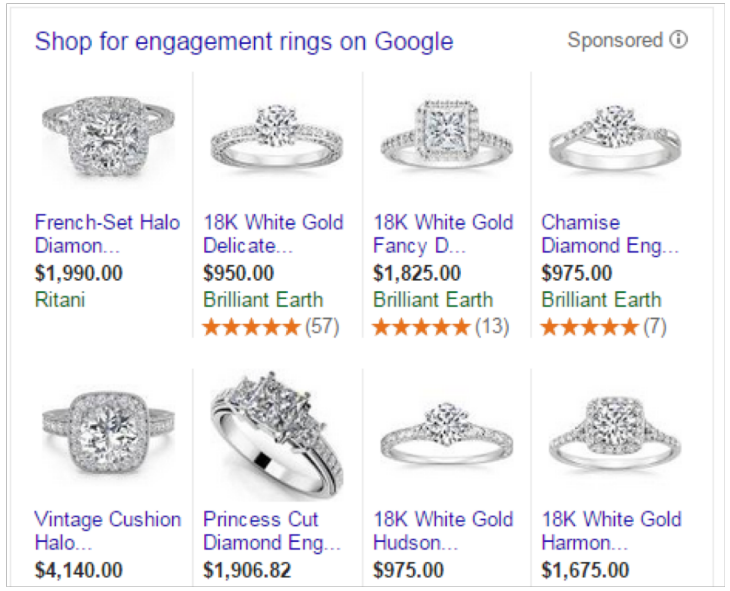
That is, we've assumed **real-valued** (or numerical) outputs

Classification

How can we predict **binary** or **categorical** variables?



Why classification?



Will I **purchase** this product?

(yes)

Will I **click on** this ad?

(no)

Why classification?

What animal appears in this image?

(mandarin duck)

Why classification?

What are the **categories** of the item being described?

(book, fiction, philosophical fiction)

From Booklist

Houellebecq's deeply philosophical novel is about an alienated young man searching for happiness in the computer age. Bored with the world and too weary to try to adapt to the foibles of friends and coworkers, he retreats into himself, descending into depression while attempting to analyze the passions of the people around him. Houellebecq uses his nameless narrator as a vehicle for extended exploration into the meanings and manifestations of love and desire in human interactions. Ironically, as the narrator attempts to define love in increasingly abstract terms, he becomes less and less capable of experiencing that which he is so desperate to understand. Intelligent and well written, the short novel is a thought-provoking inspection of a generation's confusion about all things sexual. Houellebecq captures precisely the cynical disillusionment of disaffected youth. *Bonnie Johnston --This text refers to an out of print or unavailable edition of this title.*

Concept: Linear Classification

We'll attempt to build **classifiers** that make decisions according to rules of the form

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Concept: Linear Classification

This is called **linear classification,** since we're still making decisions according to a linear function, $X_{i\cdot} \cdot \theta$

Classification

We'll look at a few different types of classifiers in detail, and briefly discuss others:

- Nearest neighbors – a simple (non-learning-based) classification scheme
- Logistic regression – a generalization of the techniques we've already seen
- (briefly) Support Vector Machines and other approaches
- Python libraries for classification

Summary of concepts

- Introduced the idea of **classification**
- Compared and contrasted classification with regression

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Classification: Nearest Neighbor

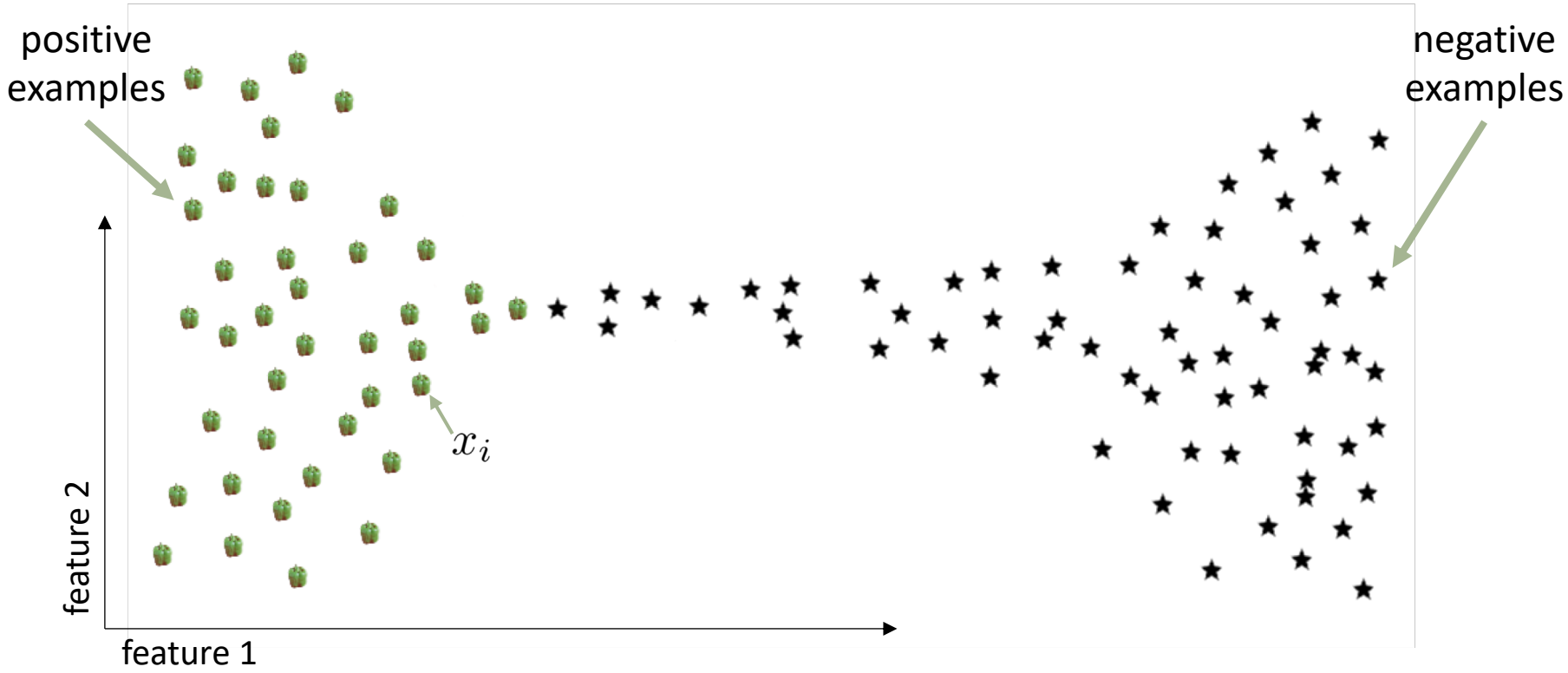
Learning objectives

In this lecture we will...

- Introduce a simple classification algorithm, before we proceed to more complex alternatives in later lectures
- Demonstrate a “non-learning” solution to classification problems

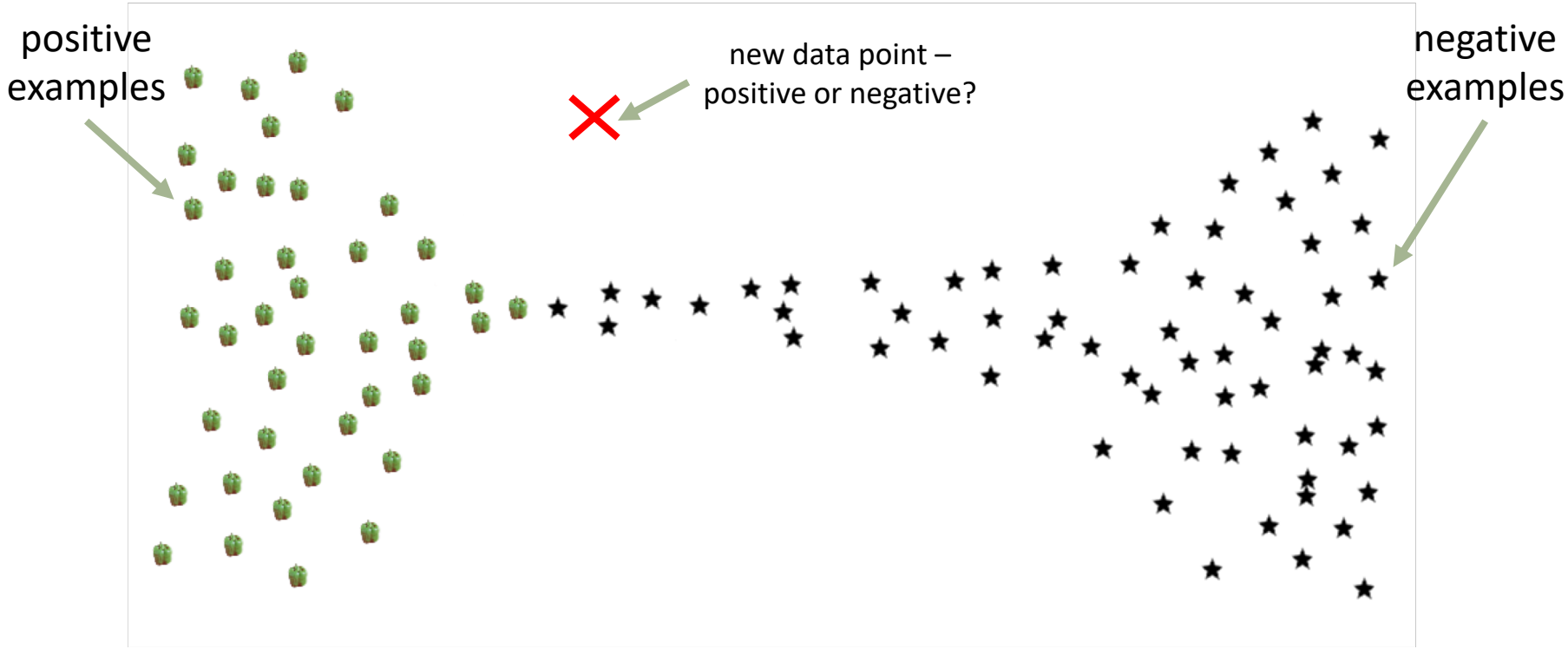
Classification

Suppose we have some data we wish to classify, belonging to one of two classes (positive or negative)



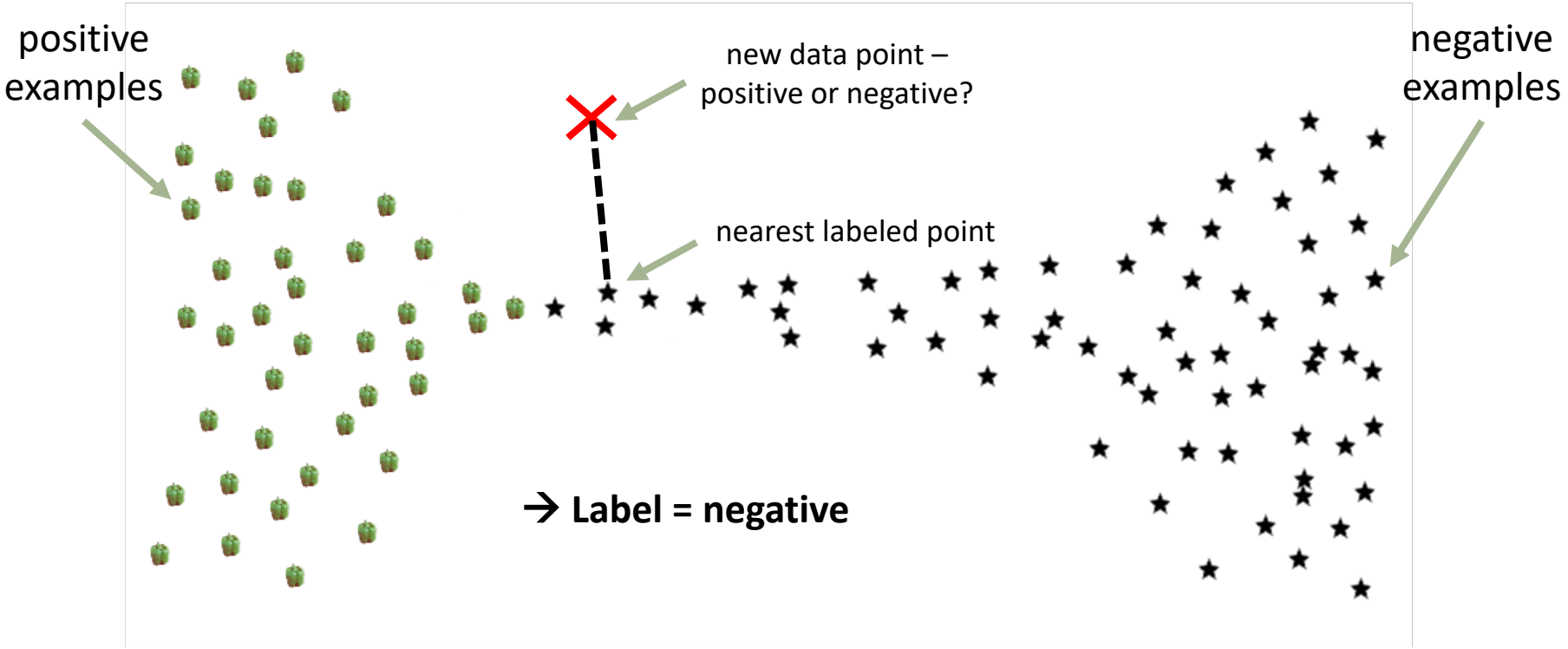
Classification

What is the simplest algorithm we could come up with to classify a new data point?



Concept: Nearest Neighbor Classification

The **nearest neighbor** classification algorithm assigns the point the **label of the nearest point**



Nearest Neighbor Classification

Precisely speaking, if we have a collection of points ***X*** (really a collection of *feature vectors*) and labels ***y,***
and we see a new point (that we wish to label) ***z,*** then:

$$\text{label}(z) = y_{\arg\min_i \|z - X_i\|}$$

- label assigned to the new point
- label of nearest point
- nearest point
- distance between z and the *i*th point

Summary of concepts

- Introduced **nearest neighbor** classification
- Introduced the notation used do describe classifiers for the rest of this course

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: gradient descent

Learning objectives

In this lecture we will...

- Introduce the notion of gradient descent, a general-purpose approach for model fitting

Closed form solutions

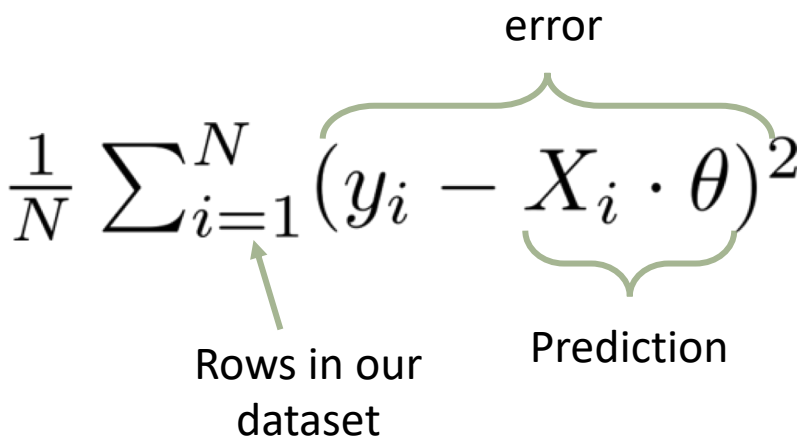
When we introduced regression, we optimized its parameters by solving a system of matrix equations:

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

- But in general we don't have nice **closed-form** solutions to choosing our models
- What can we do in cases where a closed-form isn't available?
  - E.g. how would we solve *this* problem if we couldn't find a closed form?

Optimization with gradient descent

Precisely, the problem we're trying to solve looks like



- How do we **solve this for theta**?
  - i.e., how do we choose

$$\arg\min_\theta \sum_i (x_i \cdot \theta - y_i)^2$$

Concept: Gradient Descent

Gradient Descent is a general-purpose optimization approach to solve **continuous minimization** problems that don't have a closed form

- Normally, to solve a continuous minimization problem, we would
  1. Compute the gradient w.r.t. theta
  2. Find the points where the gradient is equal to zero (i.e., the *minima* of the function)
- If we can't do (2) above, gradient descent helps us to find *local minima*

Concept: Gradient Descent

With gradient descent, rather than "solving" the problem by finding its zeros, we instead start with an initial guess, and iteratively update our solution *in the direction of the gradient*

- In this way, we gradually find solutions that come progressively closer to being zeros of the gradient equation – even though we couldn't solve it in closed-form
- The points we find are called *local minima* of the original function

The gradient descent algorithm

In essence gradient descent (to minimize a function $f(\theta)$) works as follows:

1. Initialize θ at random
2. While (not converged) do
   $\theta := \theta - \alpha f'(\theta)$

All sorts of annoying issues:

- How to initialize theta?
- How to determine when the process has converged?
- How to set the step size alpha

(these aren't really the point of this course though)

The gradient descent algorithm

So what exactly is going on here?

The gradient descent algorithm

And how would we use it to solve our regularization objective?

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

$$\frac{\partial f}{\partial \theta_k} \ ?$$

Summary of concepts

- Introduced the **gradient descent** algorithm
- Showed how this algorithm can be applied to solve the types of regression problems we've seen so far

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Classification: Logistic Regression

Learning objectives

In this lecture we will...

- Introduce **logistic regression,** a modification of regression algorithms to handle classification problems
- Show how logistic regression can be solved using gradient descent approaches

Logistic regression

Previously: regression

$$y_i = X_i \cdot \theta$$

This lecture: logistic regression

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

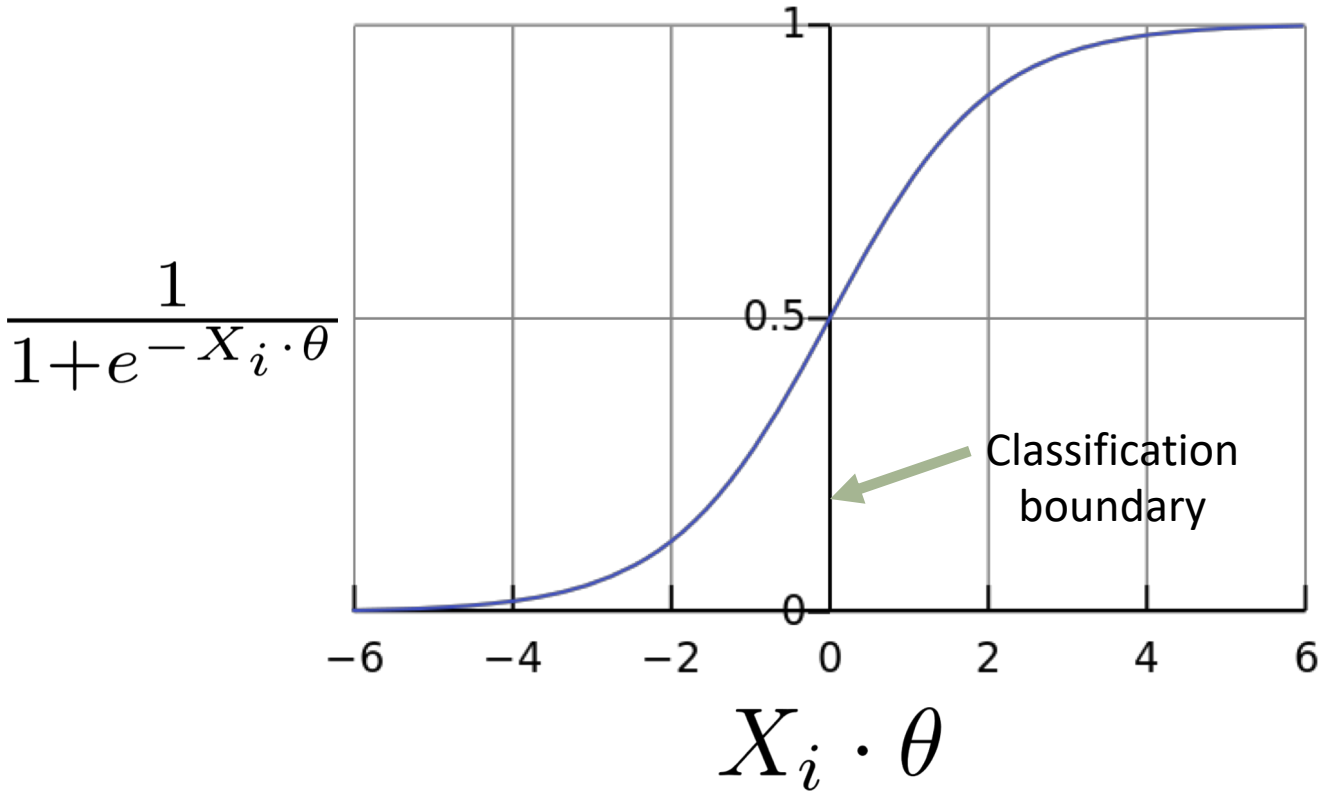
Logistic regression

Q: How to convert a real-valued expression ($X_i \cdot \theta \in \mathbb{R}$)

Into a probability ($P_\theta(y_i|X_i) \in [0, 1]$)

Logistic regression

A: sigmoid function: $\sigma(t) = \frac{1}{1+e^{-t}}$



Logistic regression

Training:

$X_i \cdot \theta$ should be maximized when y_i is positive and minimized when y_i is negative

$$\arg\max_\theta \prod_i \delta(y_i = 1) p_\theta(y_i|X_i) + \delta(y_i = 0)(1 - p_\theta(y_i|X_i))$$

$\delta(\arg)$ = 1 if the argument is true, = 0 otherwise

Logistic regression

How to optimize?

$$L_\theta(y|X) = \prod_{y_i=1} p_\theta(y_i|X_i) \prod_{y_i=0} (1 - p_\theta(y_i|X_i))$$

- Take logarithm
- Compute gradient
- Solve using gradient **ascent**

Logistic regression

$$L_\theta(y|X) = \prod_{y_i=1} p_\theta(y_i|X_i) \prod_{y_i=0} (1 - p_\theta(y_i|X_i))$$

Logistic regression

$$l_\theta(y|X) = \sum_i -\log(1 + e^{-X_i \cdot \theta}) + \sum_{y_i=0} -X_i \cdot \theta$$

$$\frac{\partial l}{\partial \theta_k} =$$

Logistic regression

Log-likelihood:

$$l_\theta(y|X) = \sum_i -\log(1 + e^{-X_i \cdot \theta}) + \sum_{y_i=0} -X_i \cdot \theta$$

Derivative:

$$\frac{\partial l}{\partial \theta_k} = \sum_i X_{ik}(1 - \sigma(X_i \cdot \theta)) + \sum_{y_i=0} -X_{ik}$$

Summary of concepts

- Introduced **logistic regression**
- Showed how to solve the logistic regression objective using gradient ascent

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Introduction to Support Vector Machines

Learning objectives

In this lecture we will...

- Briefly introduce the Support Vector Machine (SVM) classifier
- Discuss some of the relative merits of different classifiers, and reasons for choosing one classifier over another

So far we've seen...

- Last lecture we looked at **logistic regression,** which is a classification model of the form:

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

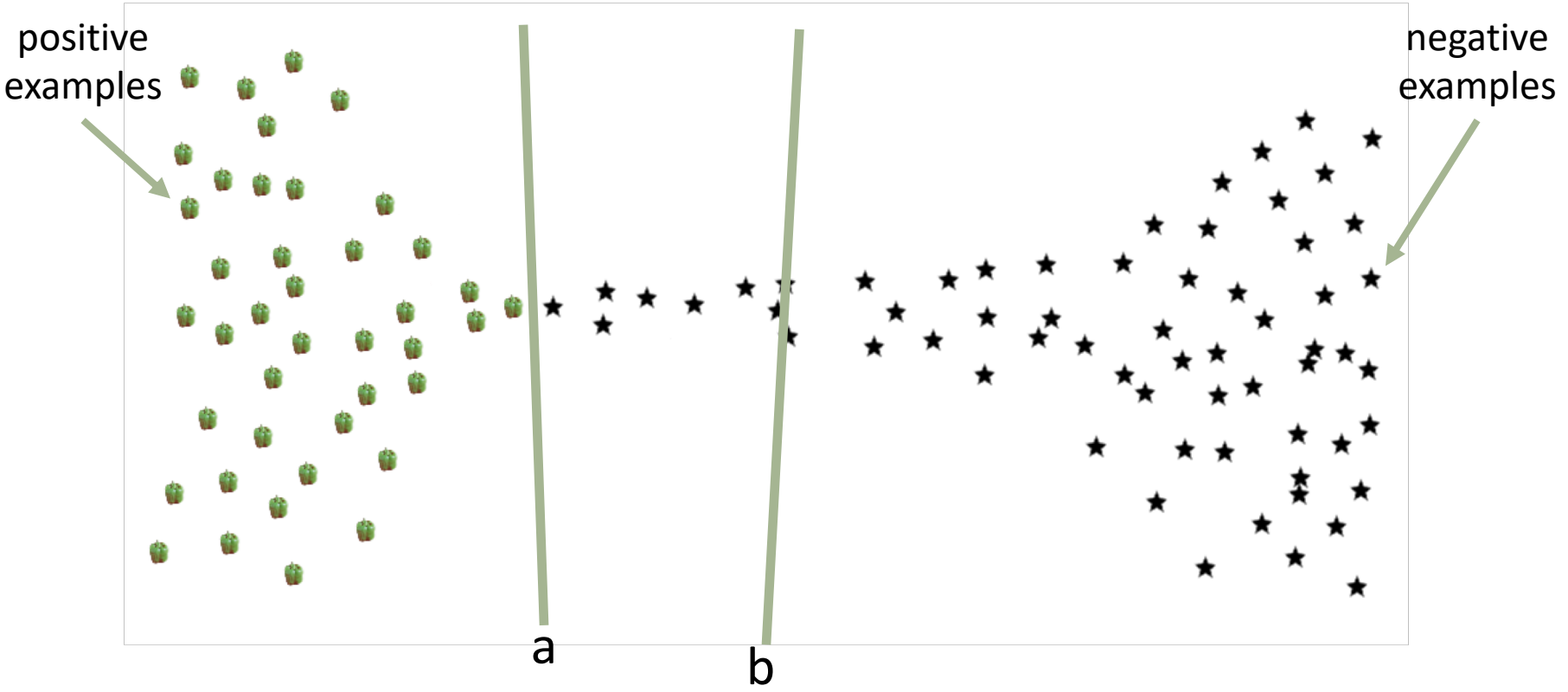
- In order to do so, we made certain **modeling assumptions,** but there are many different models that rely on different assumptions
  - In this lecture we'll look at another such model

Concept: Support vector machines

- **Support Vector Machines** are an alternative linear classification scheme that aims to **minimize the number of classification errors**
- Note that this is a different goal (as we'll see next) than the goal of logistic regression, which aims to maximize a probabilistic expression

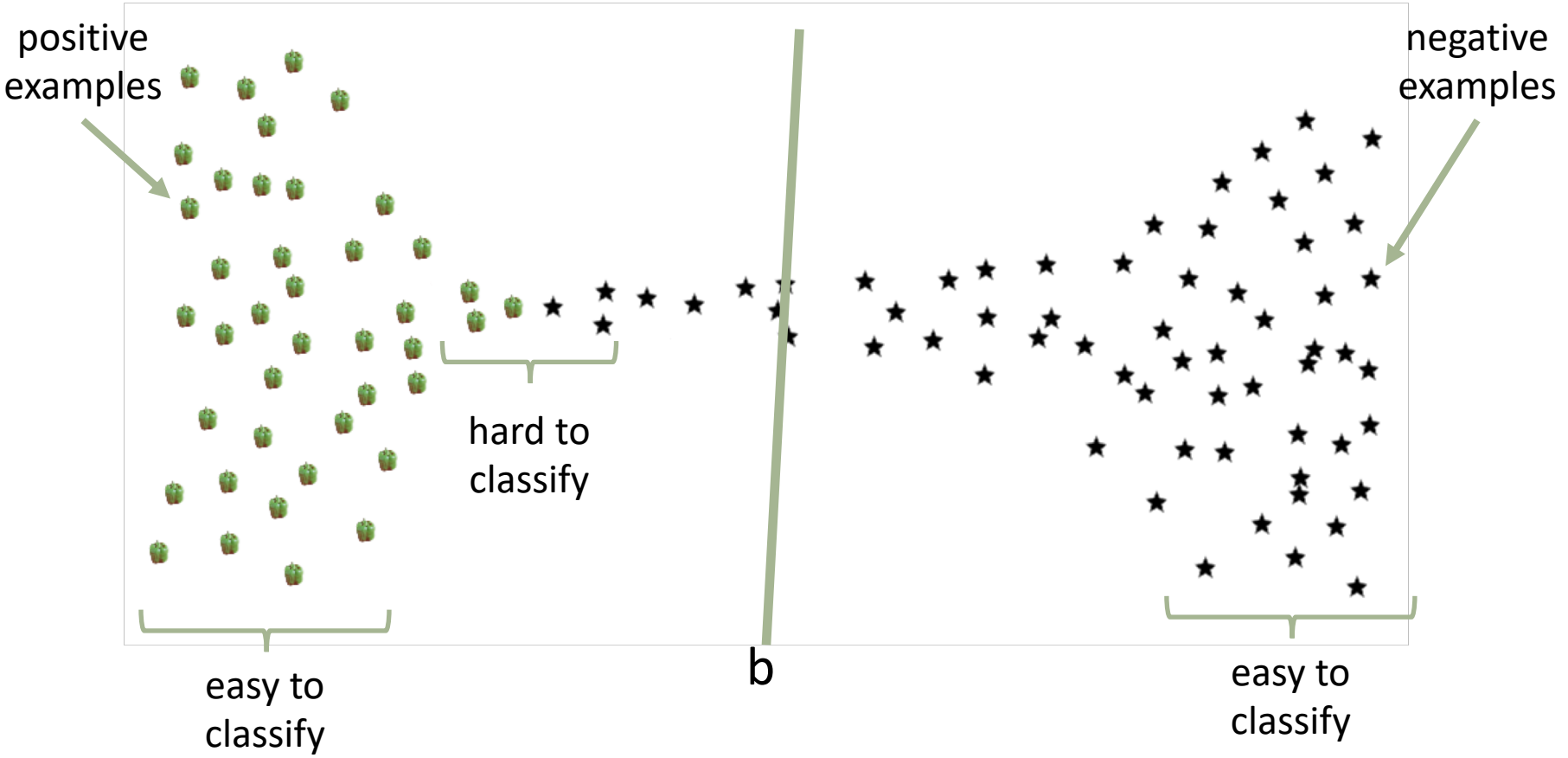
Motivation: SVMs vs Logistic regression

Q: Where would a logistic regressor place the decision boundary for these features?



SVMs vs Logistic regression

Q: Where would a logistic regressor place the decision boundary for these features?

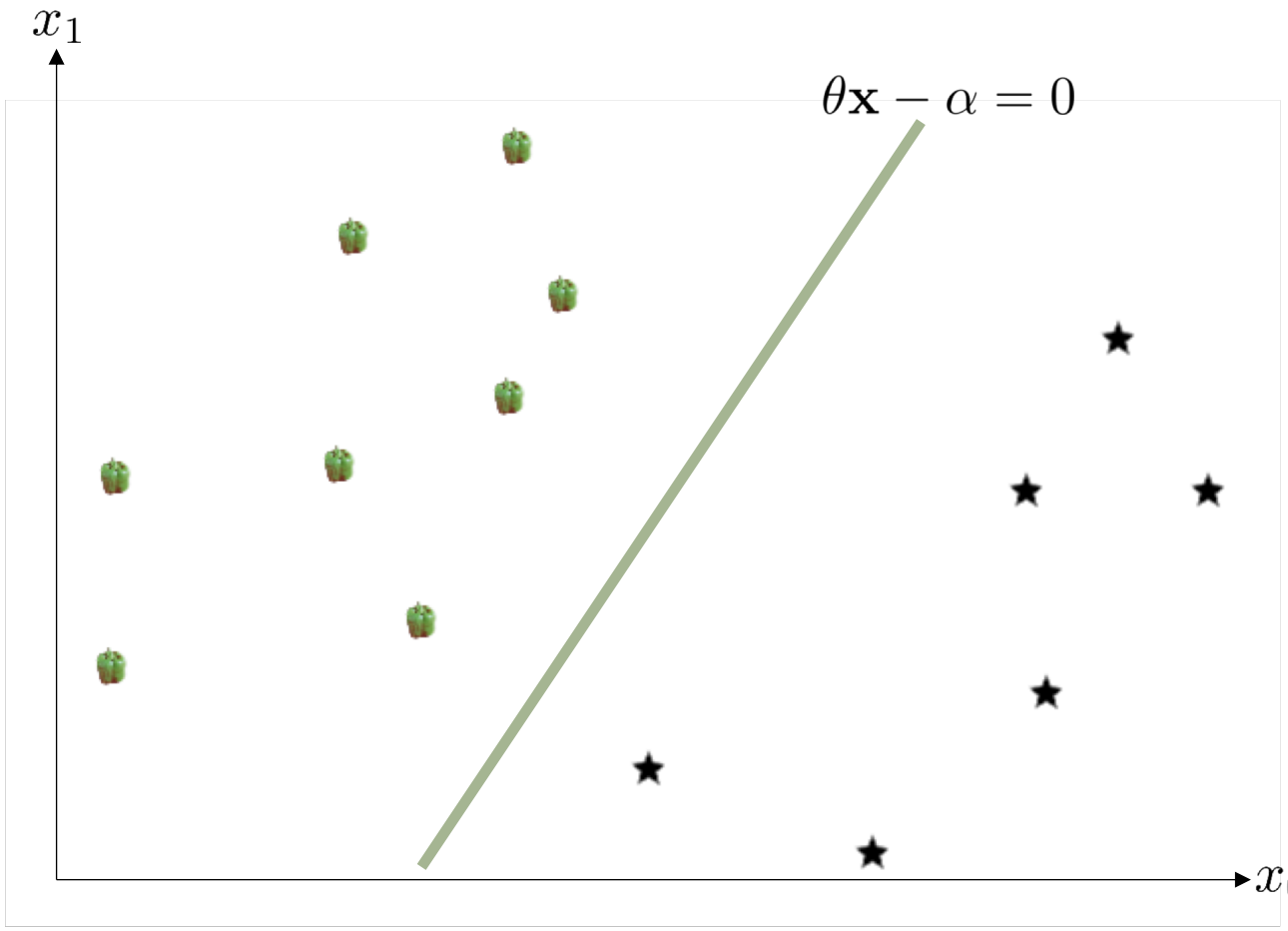


SVMs vs Logistic regression

- Logistic regressors don't optimize the number of "mistakes"
- No special attention is paid to the "difficult" instances – every instance influences the model
- But "easy" instances can affect the model (and in a bad way!)
- How can we develop a classifier that optimizes the number of mislabeled examples?

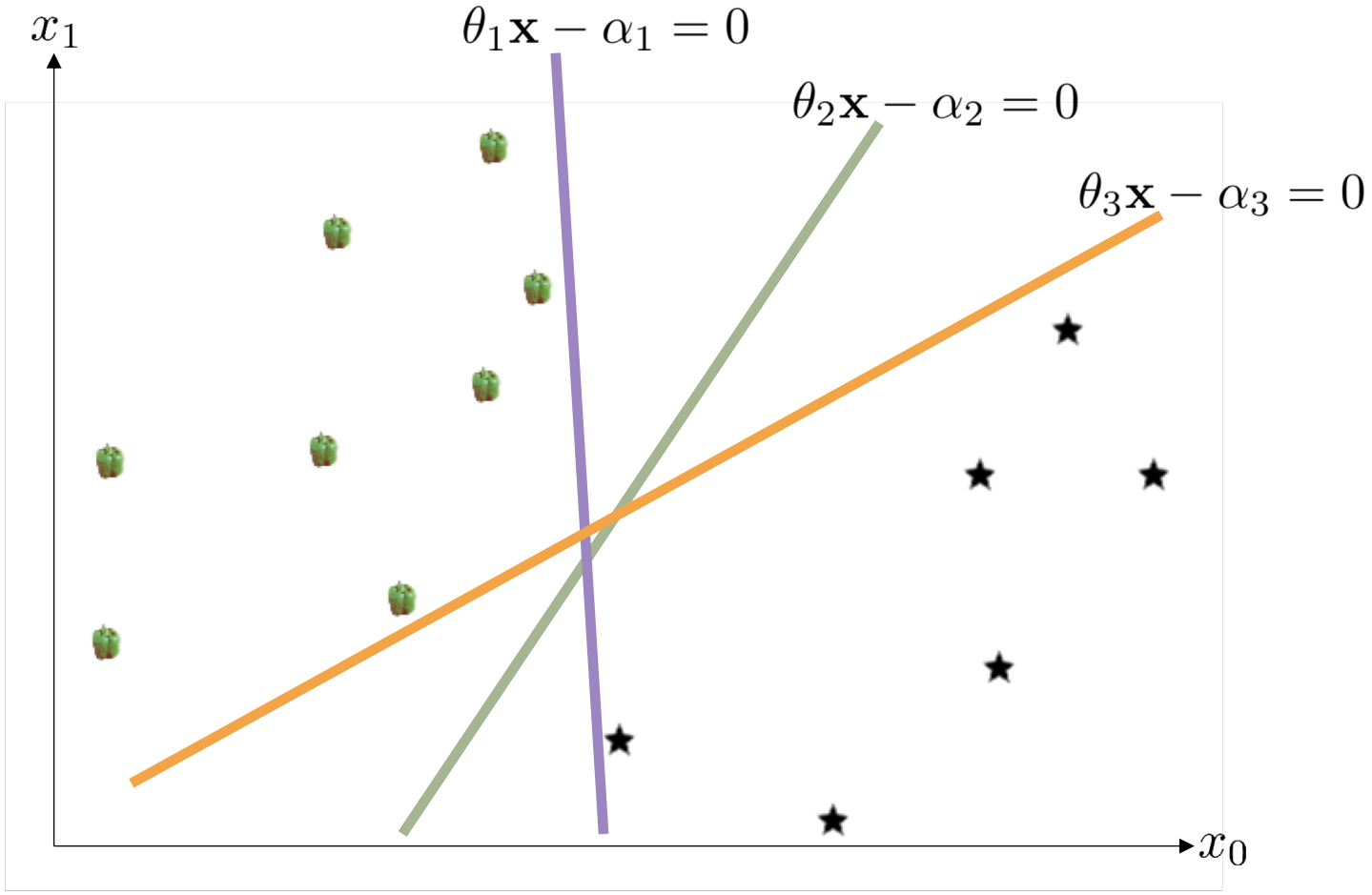
Support Vector Machines: Basic idea

A classifier can be defined by the hyperplane (line) $\theta\mathbf{x} - \alpha = 0$

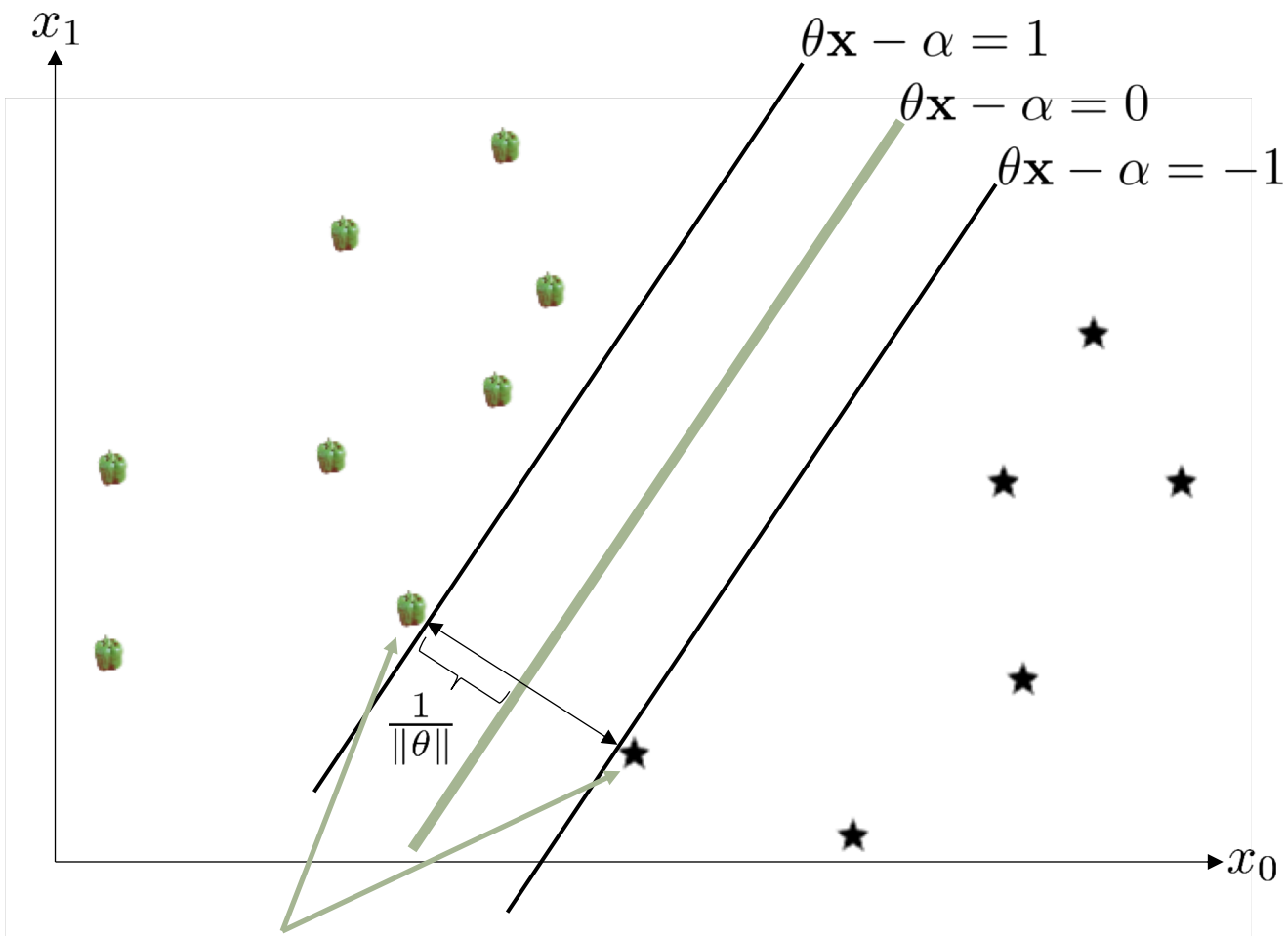


Support Vector Machines: Basic idea

Observation: Not all classifiers are equally good



Support Vector Machines



"support vectors"

- An SVM seeks the classifier (in this case a line) that is **furthest from the nearest points**
- This can be written in terms of a specific optimization problem:

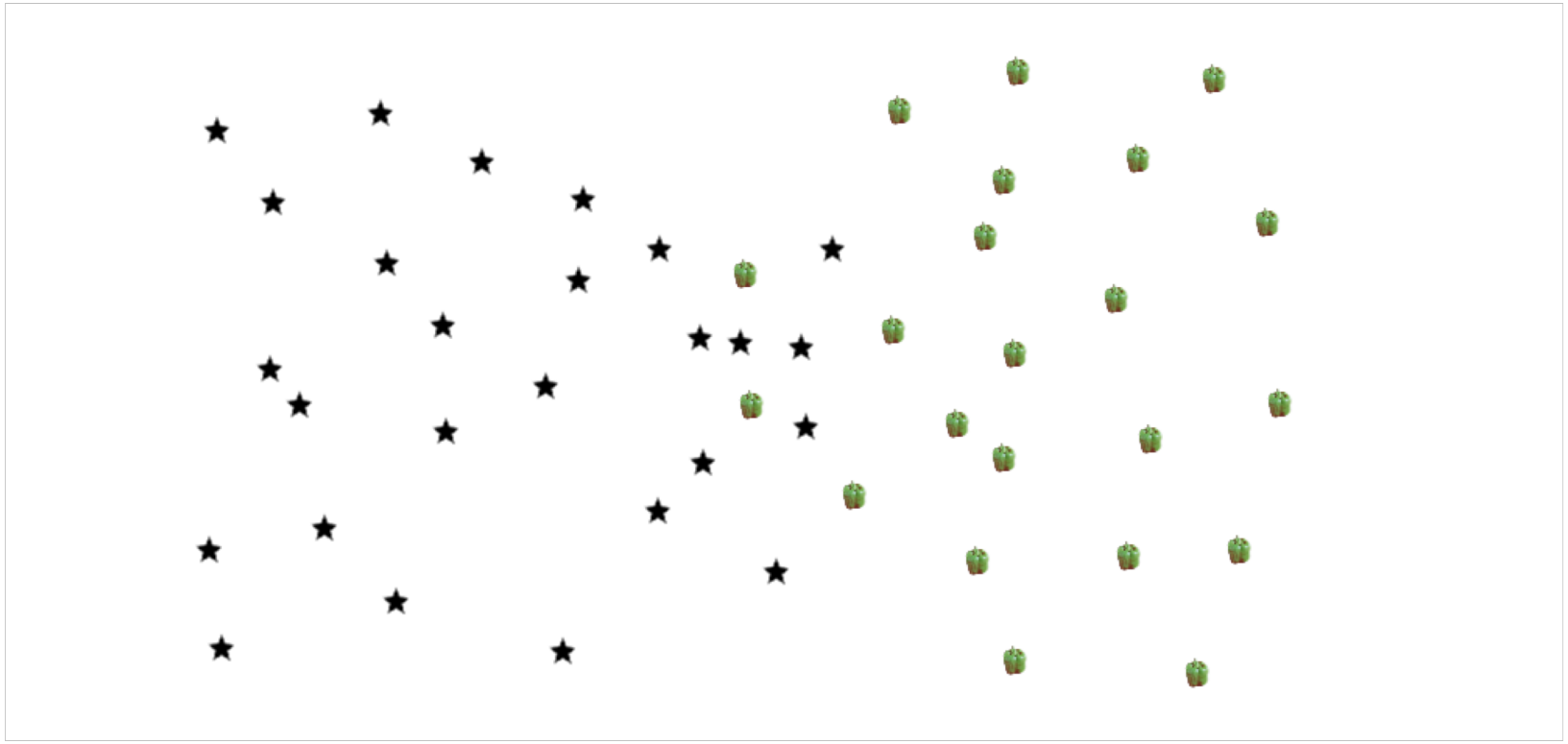
$$\arg\min_{\theta,\alpha} \frac{1}{2}\|\theta\|_2^2$$

such that

$$\forall_i y_i(\theta \cdot X_i - \alpha) \geq 1$$

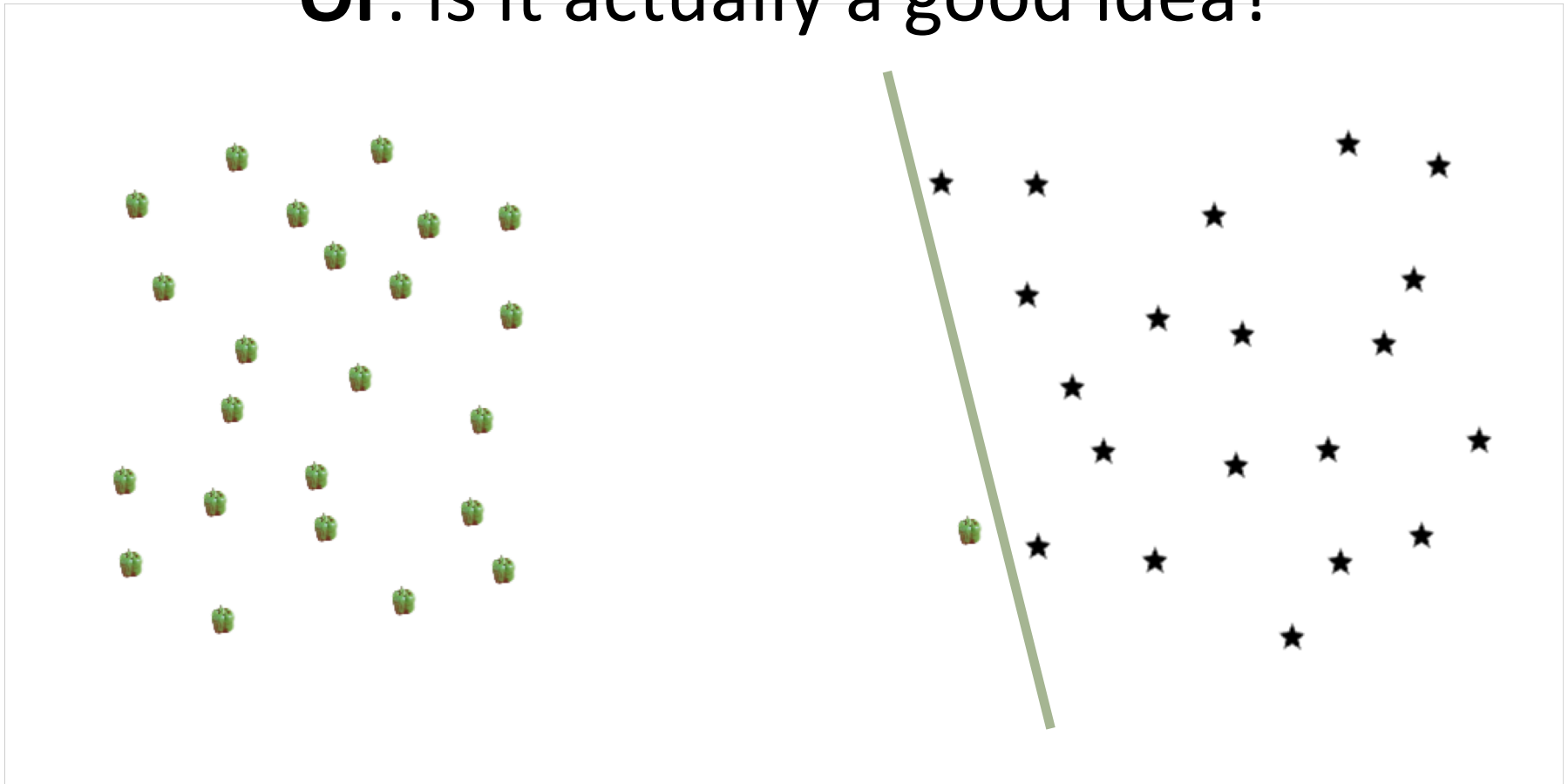
Support Vector Machines

But: is finding such a separating hyperplane even possible?



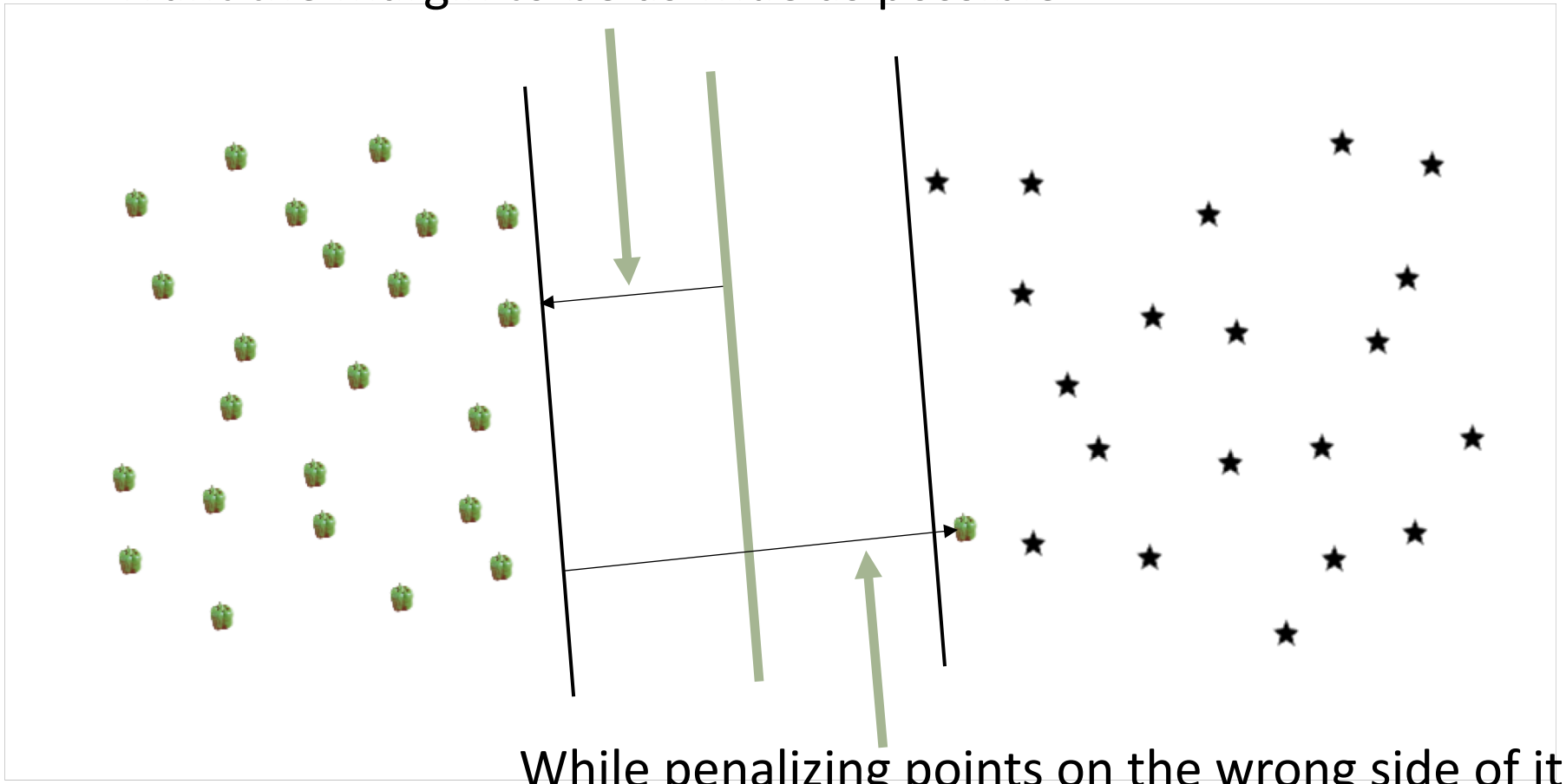
Support Vector Machines

Or: is it actually a good idea?



Support Vector Machines

Want the margin to be as wide as possible



While penalizing points on the wrong side of it

Support Vector Machines

Soft-margin formulation:

$$\arg\min_{\theta,\alpha,\xi>0} \frac{1}{2}\|\theta\|_2^2 + C\sum_i \xi_i$$

such that

$$\forall_i y_i(\theta \cdot X_i - \alpha) \geq 1 - \xi_i$$

Summary of Support Vector Machines

- SVMs seek to find a hyperplane (in two dimensions, a line) that optimally separates two classes of points
- The “best” classifier is the one that classifies all points correctly, such that the nearest points are **as far as possible** from the boundary
- If not all points can be correctly classified, a penalty is incurred that is proportional to **how badly the points are misclassified** (i.e., their distance from this hyperplane)

Summary of concepts

- Introduced **Support Vector Machines**
- Demonstrated some advantages/disadvantages among different types of classification objectives

Python Data Products

Course 2: Design thinking and predictive pipelines

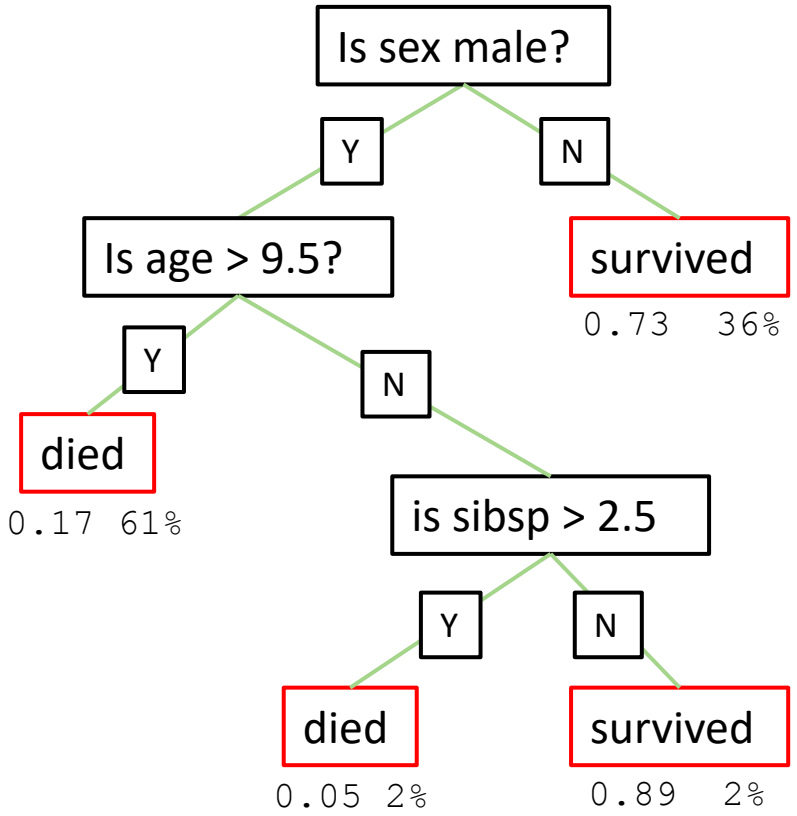
Lecture: Other classification schemes

Learning objectives

In this lecture we will...

- Introduce a few other popular classification schemes

Concept: Decision trees



(example from "Titanic Survival Data", see https://www.kaggle.com/c/titanic)

- Decision trees are a "rule based" classifier
- Each rule corresponds to a branch in the tree based on one of its features
- This is an example of a **non-linear** classifier, where different combinations of rules (and a sufficiently deep tree) can allow the data to be separated in arbitrary ways

Concept: Neural Network-based classification

This lecture sucks I don’t think I’ll bother keeping it

Summary of concepts

- Briefly introduced other popular classification schemes

Week 4

Libraries and tools for regression and classification

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: classification in Python

Learning objectives

In this lecture we will...

- Demonstrate how to set up classification problems in Python
- Introduce the **LogisticRegression** model from the **sklearn** library

Dataset – Polish bankruptcies

We'll look at a simple dataset from the UCI repository:
https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data

- This dataset is concerned with which (Polish) companies go bankrupt

Polish companies bankruptcy data Data Set

Download: Data Folder, Data Set Description

Abstract: The dataset is about bankruptcy prediction of Polish companies.The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.

Data Set Characteristics:	Multivariate	**Number of Instances:**	10503	**Area:**	Business
Attribute Characteristics:	Real	**Number of Attributes:**	64	**Date Donated**	2016-04-11
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	44898

Source:

Creator: Sebastian Tomczak
-- Department of Operations Research, WrocÅ‚aw University of Science and Technology, wybrzeÅ¼e WyspiaÅ„skiego 27, 50-370, WrocÅ‚aw, Poland

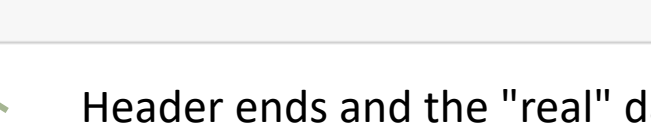
Donor: Sebastian Tomczak (sebastian.tomczak '@' pwr.edu.pl), Maciej Zieba (maciej.zieba '@' pwr.edu.pl), Jakub M. Tomczak (jakub.tomczak '@' pwr.edu.pl), Tel. (+48) 71 320 44 53

Reading the data

- Data is in CSV format, but first contains a header that we need to skip

In [1]:
```
f = open("datasets/bankruptcy/5year.arff", 'r')
```

In [2]:
```
while not '@data' in f.readline():
    pass
```



Header ends and the "real" data begins after we see the "@data" tag

- Next we read the CSV data. We (a) skip rows with missing entries; (b) convert all fields to floats; and (c) convert the label to a bool

In [3]:
```
dataset = []
for l in f:
    if '?' in l:
        continue
    l = l.split(',')
    values = [1] + [float(x) for x in l]
    values[-1] = values[-1] > 0 # Convert to bool
    dataset.append(values)
```

Processing the data

- Next let's look at some simple statistics about our data

```
In [4]: len(dataset)
Out[4]: 3031
```

Number of samples (after discarding missing values)

```
In [5]: sum([x[-1] for x in dataset])
Out[5]: 102
```

Number of **positive** samples

- Next we extract our features (X) and labels (y), much as we would do for a regression problem

```
In [6]: X = [values[:-1] for values in dataset]
```

```
In [7]: y = [values[-1] for values in dataset]
```

True/False labels

Concept: The **sklearn** library

The **sklearn** library contains a number of different regression and classification models

For example:

- linear_model.LinearRegression() - linear regression
- linear_model.LogisticRegression() - logistic regression
- svm.SVC - Support Vector Classifier

- In this lecture we'll use the **LogisticRegression** module

Fitting the logistic regression model

- First we import the library and create an instance of the model, before fitting it to data

```
In [8]: from sklearn import linear_model

In [9]: model = linear_model.LogisticRegression()

In [10]: model.fit(X, y)

Out[10]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

- Note that this function doesn't produce any output, rather it just updates the class instance to store the model

Making predictions

- Make predictions from the data:

```
In [11]: predictions = model.predict(X)

In [12]: predictions
Out[12]: array([False, False, False, ..., False, False, False])
```

- Check whether they match the labels

```
In [13]: correctPredictions = predictions == y

In [14]: correctPredictions
Out[14]: array([ True,  True,  True, ..., False, False, False])
```

- And compute the error

```
In [15]: sum(correctPredictions) / len(correctPredictions)
Out[15]: 0.9663477400197954
```

Training vs. Testing?

We achieved fairly high accuracy using a simple classifier "off the shelf"

- But note that we're evaluating our classifier on the same data that was used to train it
  - How can we be sure that our classifier will work well on **unseen data?**
- This is something we'll cover in the next course, when we look at **training, testing, and validation**

Other classification algorithms in sklearn

This example showed how to use **logistic regression,** but other classifiers are available in sklearn and have a similar interface:

- sklearn.svm.SVC: **Support Vector Classifier**
- sklearn.tree.DesicionTreeClassifier: **Decision trees**
- sklearn.naive_bayes: **Naïve Bayes**
- sklearn.neighbors.KNeighborsClassifier: **Nearest Neighbors**

- see http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html for additional comparisons

Summary of concepts

- Introduced the sklearn library
- Showed how to set up a simple classification problem in Python

On your own...

- Try to set up a similar classification problem using another of the UCI datasets – look for classification datasets that have numerical attributes (i.e., datasets similar to the one used for this exercise)

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: Introduction to Training and Testing

Learning objectives

In this lecture we will...

- Introduce the concepts of **training versus testing**
- Discuss the importance of evaluating models on **unseen** data
- Show how to adjust our Python code to make use of these ideas

Training and testing?

In the previous lecture we saw that we can obtain good performance with a simple classifier, but highlight a possible issue:

If we **evaluate** a system on the same data used to **train** the system, we may overestimate its performance

Really, we want to know how well a method is likely to work on **unseen data**

Training and testing?

To estimate how well a system is likely to perform on new data, we can split our dataset in to two components:

- A **training set** to train the machine learning model
- A **test set** used to estimate the performance on new data

We'll investigate both of these ideas more in **Course 3**, but for the moment, let's quickly explore how we can adapt our previous code to incorporate these two components

Code: Training and testing

First we read the dataset, exactly as we did in previous lectures:

In [1]:
```
f = open("/home/jmcauley/datasets/mooc/5year.arff", 'r')
```

In [2]:
```
while not '@data' in f.readline():
    pass
```

In [3]:
```
dataset = []
for l in f:
    if '?' in l:
        continue
    l = l.split(',')
    values = [1] + [float(x) for x in l]
    values[-1] = values[-1] > 0 # Convert to bool
    dataset.append(values)
```

Code: Training and testing

The first thing we do differently is to **shuffle** the data:

```
In [4]: import random

In [5]: random.shuffle(dataset)

In [6]: X = [values[:-1] for values in dataset]

In [7]: y = [values[-1] for values in dataset]
```

We do this because we we want the training and test set to be **random samples** of the data – if we didn't use random samples, different subsets of the data could have distinct characteristics that could cause the model to under- (or over) perform on one of them

Code: Training and testing

Next we split the data into a **train** and a **test** portion

In [8]:
```
N = len(X)
X_train = X[:N//2]
X_test = X[N//2:]
y_train = y[:N//2]
y_test = y[N//2:]
```

In [9]:
```
len(X), len(X_train), len(X_test)
```

Out[9]:
```
(3031, 1515, 1516)
```

Note that we split both the data (*X*) and the labels (*y*) in the same way

Code: Training and testing

Next we train our model as before, but we use **only the training data and labels**

```
In [10]: from sklearn import linear_model

In [11]: model = linear_model.LogisticRegression()

In [12]: model.fit(X_train, y_train)

Out[12]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Code: Training and testing

Finally we can compute the accuracy of the model, but this time we do so separately for the training and test portions

```
In [13]: predictionsTrain = model.predict(X_train)
         predictionsTest = model.predict(X_test)

In [14]: correctPredictionsTrain = predictionsTrain == y_train
         correctPredictionsTest = predictionsTest == y_test

In [15]: sum(correctPredictionsTrain) / len(correctPredictionsTrain) # Training accuracy
Out[15]: 0.9630363036303631

In [16]: sum(correctPredictionsTest) / len(correctPredictionsTest) # Test accuracy
Out[16]: 0.9637203166226913
```

The latter quantity measures **how well the model is likely to perform on new data**

Summary

This lecture presented a very brief introduction to training and testing. Though we'll cover more in Course 3, the basic concepts are:

- Simply training on a dataset doesn't give us a sense of how a model will **generalize to new data**
- This generazilation ability can be estimated using a test set
- Training and test sets should be **non-overlapping, random** splits of our data

Summary of concepts

- Introduced the concepts of training and testing sets
- Briefly described the difference between training performance versus generalization ability
- Showed how to adapt our classification code to measure performance on the training set

On your own...

- Try repeating this exercise for our **regression** example from the previous lecture, i.e., split the data into training/testing portions and measure its training and testing performance

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: gradient descent in Python

Learning objectives

In this lecture we will...

- Show how gradient descent can be implemented in Python
- Introduce the relationship between equations/mathematical objectives (theory) and their implementation (practice)

Goal: Regression objective

$$\arg\min_\theta \sum_i (x_i \cdot \theta - y_i)^2$$

$$\frac{\partial f}{\partial \theta_k} = \sum_i 2X_{ik}(X_i \cdot \theta - y_i)$$

Let's look at implementing this on the same PM2.5 dataset from our previous lecture on regression

Code: Reading the data

Reading the data from CSV, and discarding missing entries:

In [1]:
```
path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')
```

In [2]:
```
dataset = []
header = f.readline().strip().split(',')
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [3]:
```
header.index('pm2.5')
```

Out[3]: 5

In [4]:
```
dataset = [d for d in dataset if d[5] != 'NA']
```

Code: Extracting features from the data

Extract features from the dataset:

In [5]:
```
def feature(datum):
    feat = [1, float(datum[7])] # Temperature
    return feat
```

In [6]:
```
X = [feature(d) for d in dataset]
y = [float(d[5]) for d in dataset]
```

Offset and temperature

In [7]:
```
X[0]
```

Out[7]:
```
[1, -4.0]
```

In [8]:
```
K = len(X[0])
K
```

Out[8]:
```
2
```

K = number of feature dimensions

Code: Initialization

Initialize parameters (and include some utility functions)

In [9]:
```
theta = [0.0]*K
```

In [10]:
```
theta[0] = sum(y) / len(y)
```

In [11]:
```
def inner(x,y):
    return sum([a*b for (a,b) in zip(x,y)])
```

In [12]:
```
def norm(x):
    return sum([a*a for a in x]) # equivalently, inner(x,x)
```

- Initializing theta_0 (the offset parameter) to the mean value will help the model to converge faster
- Generally speaking, initializing gradient descent algorithms with a "good guess" can help them to converge more quickly

Code: Derivative

Compute partial derivatives for each dimension:

In [13]:
```
def derivative(X, y, theta):
    dtheta = [0.0]*len(theta)
    K = len(theta)
    N = len(X)
    MSE = 0
    for i in range(N):
        error = inner(X[i],theta) - y[i]
        for k in range(K):
            dtheta[k] += 2*X[i][k]*error/N
        MSE += error*error/N
    return dtheta, MSE
```

Derivative:

$$\frac{\partial f}{\partial \theta_k} = \sum_i 2X_{ik}(X_i \cdot \theta - y_i)$$

Also compute MSE, just for utility

Code: Derivative

Compute partial derivatives for each dimension:

```
In [14]: learningRate = 0.003

In [15]: while (True):
             dtheta,MSE = derivative(X, y, theta)
             m = norm(dtheta)
             print("norm(dtheta) = " + str(m) + " MSE = " + str(MSE))
             for k in range(K):
                 theta[k] -= learningRate * dtheta[k]
             if m < 0.01: break
```

Update in direction of derivative

Stopping condition

```
norm(dtheta) = 0.011085715419421865 MSE = 8403.627794070962
norm(dtheta) = 0.011020632851413479 MSE = 8403.627760862651
norm(dtheta) = 0.01095593237337664 MSE = 8403.627727849314
norm(dtheta) = 0.010891611742123273 MSE = 8403.627695029725
norm(dtheta) = 0.010827668727610302 MSE = 8403.627662403022
norm(dtheta) = 0.010764101112955294 MSE = 8403.627629967714
norm(dtheta) = 0.01070090669415397 MSE = 8403.627597722905
norm(dtheta) = 0.01063808328031018 MSE = 8403.627565667332
norm(dtheta) = 0.010575628693268708 MSE = 8403.627533799903
norm(dtheta) = 0.010513540767733419 MSE = 8403.627502119632
norm(dtheta) = 0.010451817351068865 MSE = 8403.627470625426
norm(dtheta) = 0.010390456303283141 MSE = 8403.627439316158
norm(dtheta) = 0.010329455497002886 MSE = 8403.627408190638
```

Code: Derivative

Read output

```
In [16]: theta
Out[16]: [107.00031826701057, -0.6803048266097109]
```

- (Almost) identical to the result we got when using the regression library in the previous lecture

Summary

Although a crude (and fairly slow) implementation, this type of approach can be extended to handle quite general and complex objectives. However it has several difficult issues to deal with:

- How to initialize?
- How to set parameters like the learning rate and convergence criteria?
- Manually computing derivatives is time-consuming – and difficult to debug

Summary of concepts

- Briefly introduced a crude implementation of gradient descent in Python
- Later, we'll see how the same operations can be supported via libraries

Python Data Products

Course 2: Design thinking and predictive pipelines

Lecture: gradient descent in tensorflow

Learning objectives

In this lecture we will...

- Introduce the **tensorflow** library

Tensorflow

Tensorflow, though often associated with deep learning, is really just a library that simplifies gradient descent and optimization problems, like those we saw in the previous lecture

In this lecture we'll reimplement the previous lecture's example in Tensorflow

Code: Gradient Descent in Tensorflow

Reading the data is much the same as before (except that we first import the tensorflow library)

In [1]:
```
import tensorflow as tf
```

In [2]:
```
path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')
```

In [3]:
```
dataset = []
header = f.readline().strip().split(',')
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [4]:
```
header.index('pm2.5')
```

Out[4]: 5

In [5]:
```
dataset = [d for d in dataset if d[5] != 'NA']
```

Code: Gradient Descent in Tensorflow

Next we extract features from the data

In [6]:
```
def feature(datum):
    feat = [1, float(datum[7]), float(datum[8]), float(datum[10])] # Temperature, pressure, and wind speed
    return feat
```

In [7]:
```
X = [feature(d) for d in dataset]
y = [float(d[5]) for d in dataset]
```

In [8]:
```
y = tf.constant(y, shape=[len(y),1])
```

In [9]:
```
K = len(X[0])
```

Note that we convert *y* to a native tensorflow vector. In particular we convert it to **column** vector. We have to be careful about getting our matrix dimensions correct or we may (accidentally) apply the wrong matrix operations.

Code: Gradient Descent in Tensorflow

Next we write down the objective – note that we use native tensorflow operations to do so

```
In [10]: def MSE(X, y, theta):
           return tf.reduce_mean((tf.matmul(X,theta) - y)**2)
```

Next we setup the variables we want to optimize – note that we explicitly indicate that these are **variables** to be optimized (rather than constants)

```
In [11]: theta = tf.Variable(tf.constant([0.0]*K, shape=[K,1]))
```

Initialized to zero

```
In [12]: optimizer = tf.train.AdamOptimizer(0.01)
```

Stochastic gradient descent optimizer with learning rate of 0.01

```
In [13]: objective = MSE(X,y,theta)
```

Specify the objective we want to optimize – note that no computation is performed (yet) when we run this function

Code: Gradient Descent in Tensorflow

Boilerplate for initializing the optimizer...

```
In [14]: train = optimizer.minimize(objective)
```

We want to **minimize** the objective

```
In [15]: init = tf.global_variables_initializer()
```

```
In [16]: sess = tf.Session()
         sess.run(init)
```

Code: Gradient Descent in Tensorflow

Run 1,000 iterations of gradient descent:

```
In [17]: for iteration in range(1000):
           cvalues = sess.run([train, objective])
           print("objective = " + str(cvalues[1]))
```

```
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5103
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
```

Code: Gradient Descent in Tensorflow

Print out the results:

In [18]:
```
with sess.as_default():
  print(MSE(X, y, theta).eval())
  print(theta.eval())
```

```
7836.5093
[[ 0.23223479]
 [-0.89481604]
 [ 0.11925128]
 [-0.4959688 ]]
```

Summary

Note that in contrast to our "manual" implementation of gradient descent, many of the most difficult issues were taken care of for us:

- No need to compute the gradients – tensorflow does this for us!
- Easy to experiment with different models
- Very fast to run 1,000 iterations, especially with GPU acceleration!

Other libraries

Tensorflow is just one example of a library that can be used for this type of optimization. Alternatives include:

- Theano - http://deeplearning.net/software/theano/
- Keras - https://keras.io/
- Torch - http://torch.ch/
- Etc.

Each has fairly similar functionality, but some differences in interface

Summary of concepts

- Introduced the Tensorflow library for optimization