15-388/688 - Practical Data Science: Introduction

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Some possible definitions

Data science is the application of computational and statistical techniques to address or gain insight into some problem in the real world

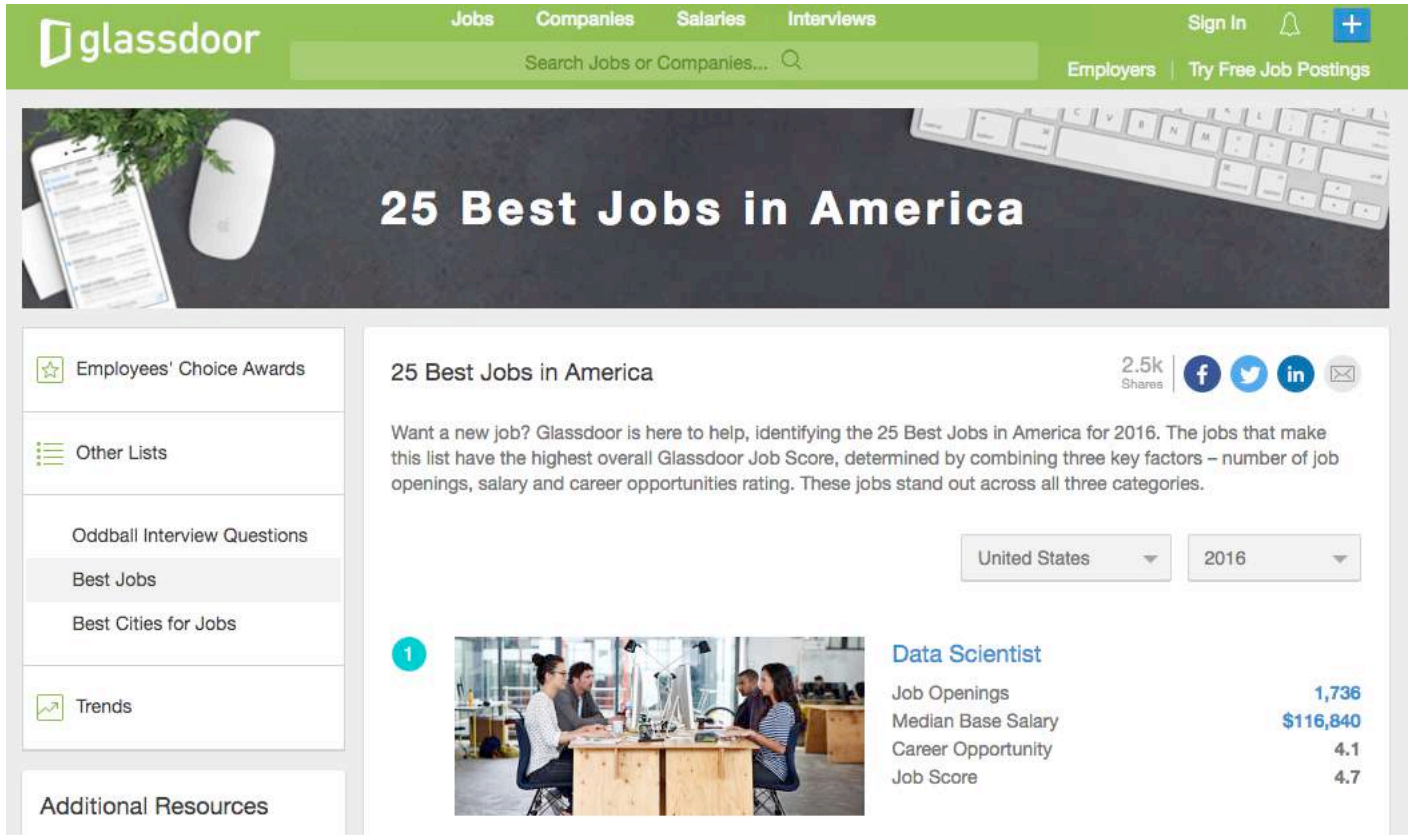
Some possible definitions

Data science is the application of **computational** and **statistical** techniques to address or gain insight into some problem in the **real world**

Some possible definitions

Data science = statistics +
data processing +
machine learning +
scientific inquiry +
visualization +
business analytics +
big data + …

Data science is the best job in America



Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Data science is not machine learning

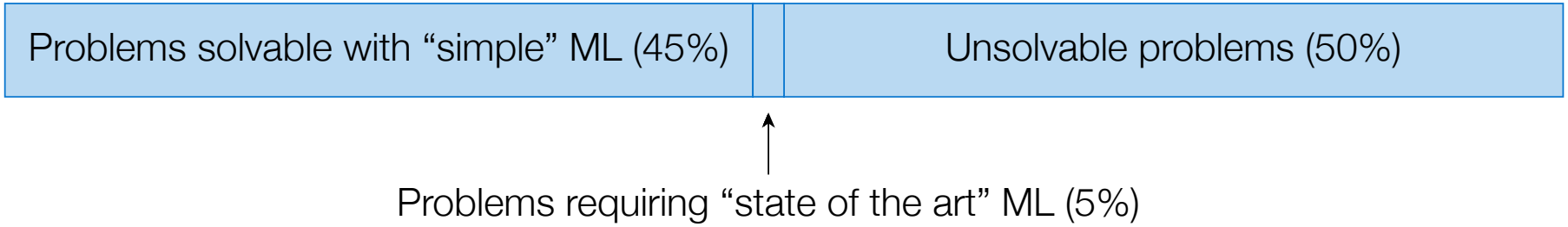
Machine learning involves computation and statistics, but has not (traditionally) been very concerned about answering *scientific questions*

Machine learning has a heavy focus on fancy algorithms…

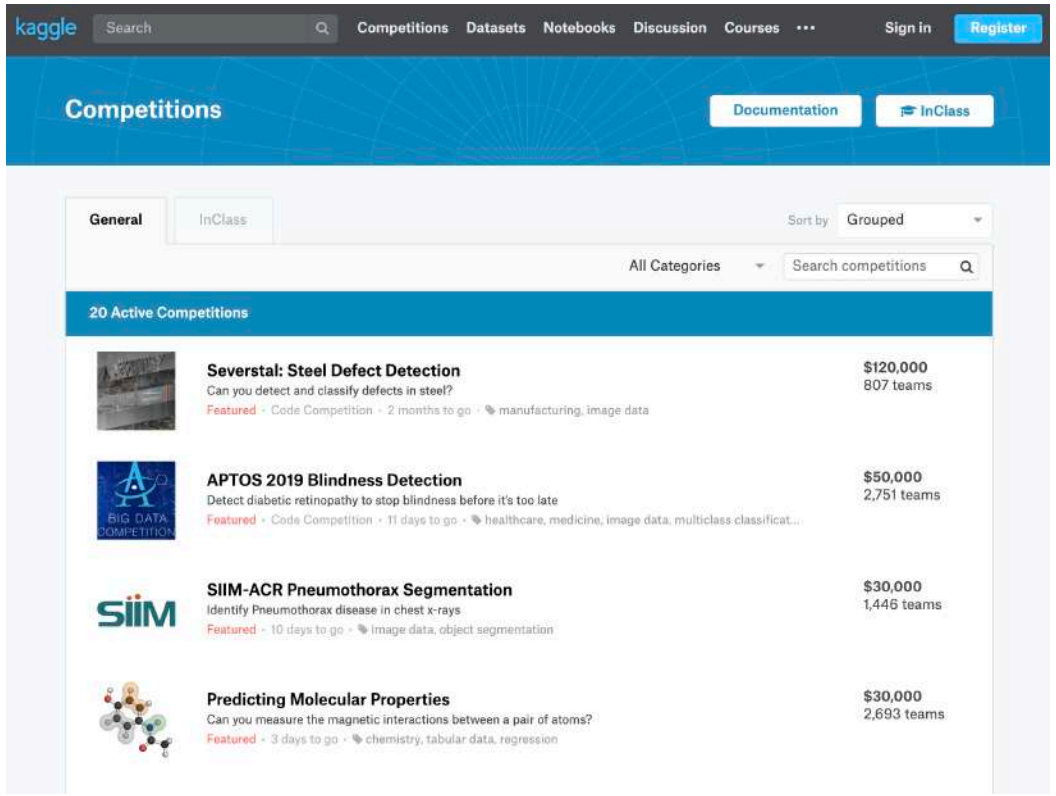
... but sometimes the best way to solve a problem is just by visualizing the data, for instance

Data science is not machine learning

Universe of machine learning problems



Data science is not machine learning competitions



Data science competitions like Kaggle ask you to optimize a metric on a fixed data set

This may or may not ultimately solve the desired business/scientific problem

Data science is the iterative cycle of designing a concrete problem, building an algorithm to solve it (or determining that this is not possible), and evaluating what insights this provides for the real underlying question

Data science is not statistics

“Analyzing data computationally, to understand some phenomenon in the real world, you say? … that sounds an awful lot like statistics”

Statistics (at least the academic type) has evolved a lot more along the mathematical/theoretical frontier

Not many statistics courses have a lecture on e.g. web scraping, or a lot of data processing more generally

Plus, statisticians use R, while data scientists use Python ... clearly these are completely different fields

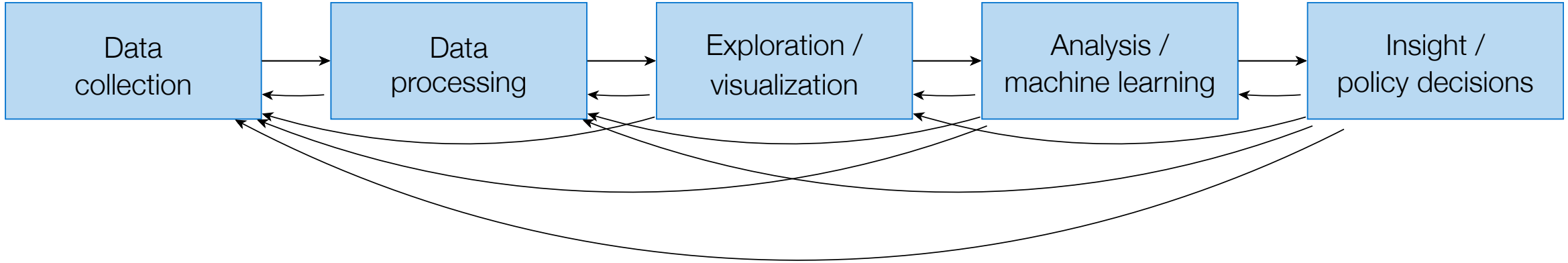
Data science is not big data

Sometimes, in order to truly understand and answer your question, you need massive amounts of data…

…But sometimes you don’t

Don’t create more work for yourself than you need to

Back to what data science is



Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Gendered language in professor reviews

Gendered Language in Teacher Reviews

This interactive chart lets you explore the words used to describe male and female teachers in about 14 million reviews from RateMyProfessor.com.

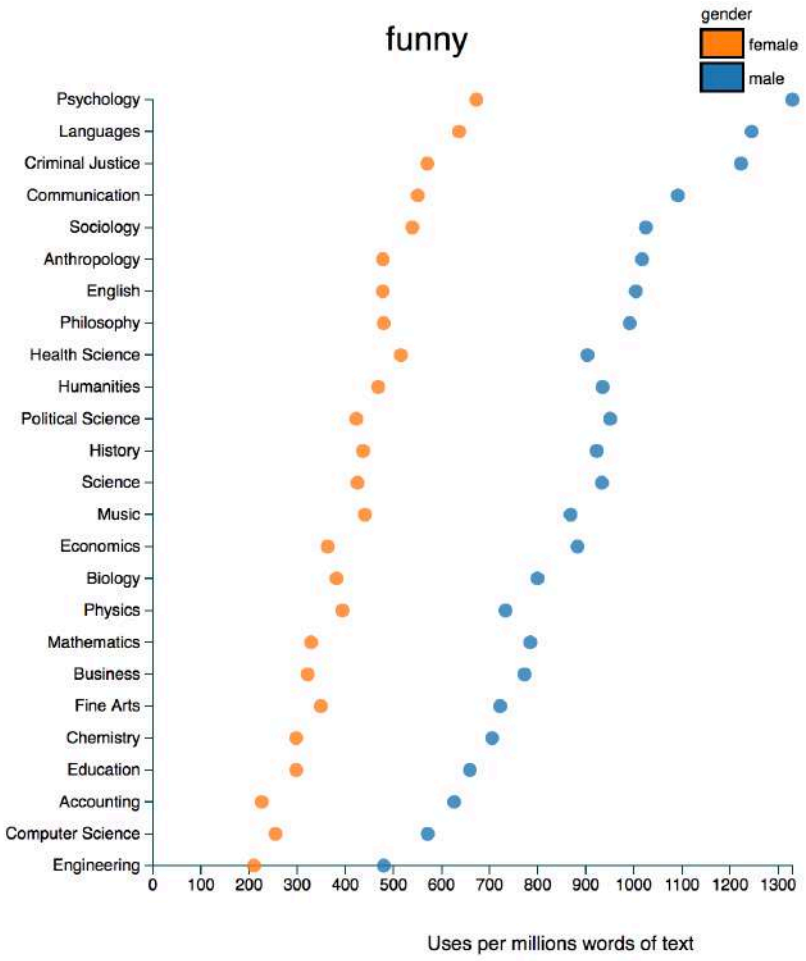
You can enter any other word (or two-word phrase) into the box below to see how it is split across gender and discipline: the x-axis gives how many times your term is used per million words of text (normalized against gender and field). You can also limit to just negative or positive reviews (based on the numeric ratings on the site). For some more background, see here.

Not all words have gender splits, but a surprising number do. Even things like pronouns are used quite differently by gender.

Search term(s) (case-insensitive): use commas to aggregate multiple terms

funny

All ratings | Only positive | Only negative



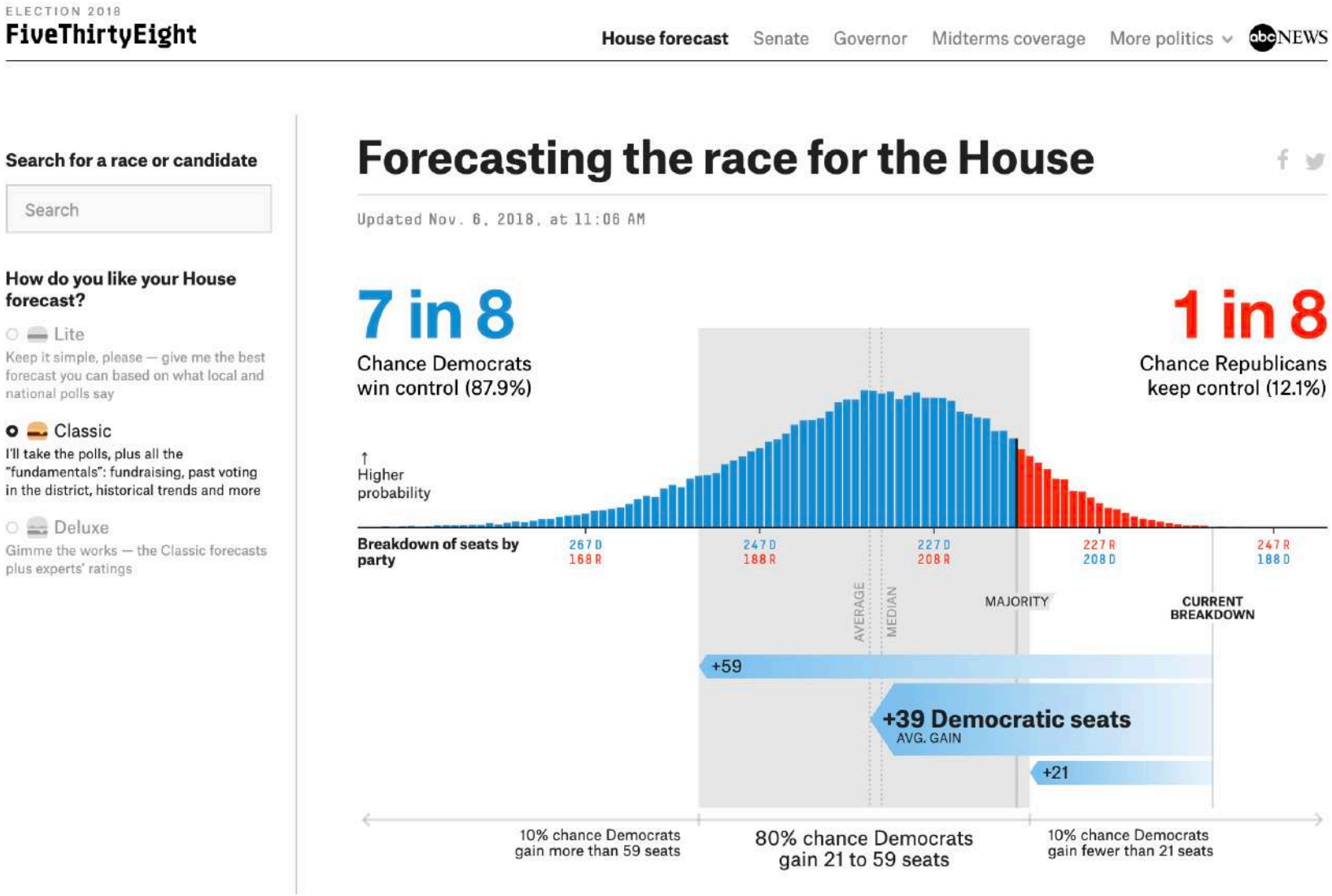
http://benschmidt.org/profGender/

Obligatory quote

The greatest value of a picture is when it forces us to notice what we never expected to see.

-John Tukey

FiveThirtyEight



https://projects.fivethirtyeight.com/2018-midterm-election-forecast/house/

Poverty Mapping

Figure 2: Example of metal roof in center of satellite image.

Figure 3: Example of thatched roof in center of satellite image.

Abelson, Varshney, and Sun. “Targeting Direct Cash Transfers to the Extremely Poor,” 2012

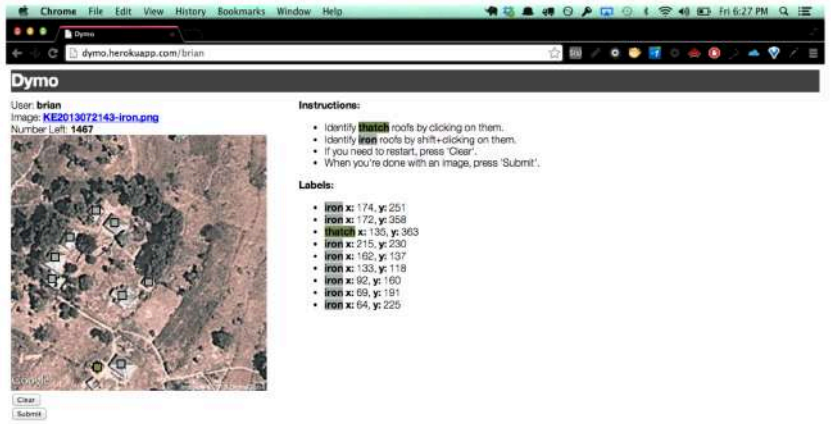


Figure 6: Screen shot of application deployed for crowdsourced labeling of roofs in satellite images.

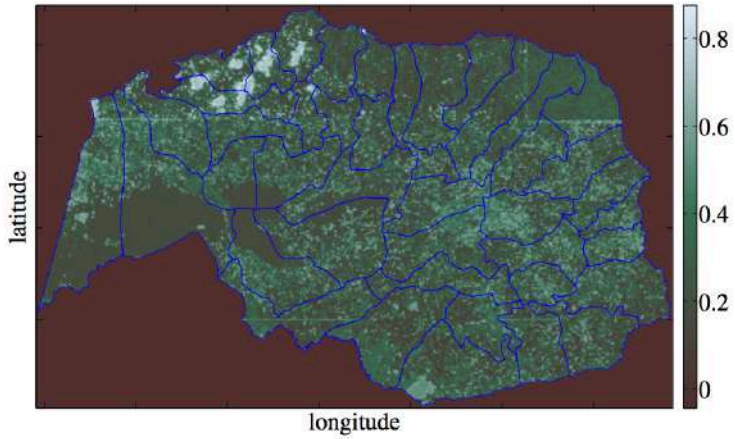


Figure 11: Heat map of proportion of roofs that are metal in the region of interest.

Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Learning objectives of this course

After taking this course, you should…

… understand the full data science pipeline, and be familiar with programming tools to accomplish the different portions

... be able to collect data from unstructured sources and store it using appropriate structure such as relational databases, graphs, matrices, etc

... know to explore and visualize your data

... be able to analyze your data rigorously using a variety of statistical and machine learning approaches

Topics covered (subject to change)

Data collection and management: relational data, matrices and vectors, graphs and networks, free text processing, geographical data

Statistical modeling and machine learning: linear and nonlinear classification and regression, regularization, data cleaning, hypothesis testing, kernel methods and SVMs, boosting, clustering, dimensionality reduction, recommender systems, deep learning, probabilistic models, scalable ML

Visualization: basic visualization and data exploration, data presentation and interactivity

Philosophy: tools and deeper understand

Most of the techniques we will teach in this course have mature tools that you will likely use in practice

But, the philosophy of this course is that you will use these tools most effectively when you understand what is going on under the hood

This course will teach you some of the more common tools, but (especially in 15-688 problem sets), you will also need to implement some of the underlying methods

Example: we'll teach you how to run machine learning algorithms using scikit-learn library, but you'll also need to implement some of the algorithms yourself

Differences between 15-388/688 and XX

There are many courses that cover similar or related material (10-601, 10-701, 11-663, 05-839, 36-402, etc)

In general, this course puts a high emphasis on exploring and analyzing real (unprepared) data, managing the entire data science pipeline

Compared to other machine learning or statistics courses, there is relatively little theory, higher emphasis on implementation and use on practical data sets

Recommended background

The only formal prerequisite for this course is an intro to programming (if you have taken one at another university, this is fine)

We strongly recommend that students have **experience with Python**, ideally some background in **probability and statistics, and linear algebra**

If you don't have background in these areas, you may still sign up, but be aware that you will probably need to learn some of these items as the class goes on (we will be providing pointers to references)

General rule of thumb: If the homework seems hard, but you have ideas about how to proceed, you probably have the right level of background; if the homework seems hard and you have no idea how to proceed, this may be the wrong course

Outline

What is data science?

What is data science not?

(A few) data science examples

Course objectives and topics

Course logistics

Course materials and discussion

All course material (slides, notes, lecture videos, assignments) is available on the course webpage:

http://www.datasciencecourse.org

Slides posted before class, videos up ~2-3 hours after, notes posted asynchronously, typically well before lecture

All forums/discussion and homeworks will be submitted online via Diderot, signup instructions on the course page, under “Assignments”

http://diderot.one

15-388 vs. 15-688

Two versions of the course: 15-388 (undergrad, 9 unit), 15-688 (graduate, 12 unit)

Courses are identical (same lectures, assignments, etc) except that
15-688 problem sets have an additional question per assignment, usually
requiring that students implement some advanced technique

Undergraduates **may take 15-688** for 12 units, but please wait until enrollment
shakes out (for now, just start doing the 15-688 questions on the homeworks)

Course waitlist and DNM section

We currently have many more students enrolled than available space

To allows in as many people as possible, we added Section B, a DNM (does not meet) section to 15-688, courses are identical except that lectures are online

The reality is that by the first few weeks of the semester, there will be room in the course, even if you are in Section B

Will I get off the waitlist?

- 15-388: Probably yes
- 15-688-A: Probably not
- 15-688-B: Yes

Auditing?

Auditing is permitted, but *only* within the B section of 15-688 (i.e., non-auditors will have preference for in-class version of the course)

The requirements to pass an audit are to receive at least **50% of the points on 4 out of the 5 assignments** (out of the whole assignment, so both the two 388 questions and the one additional 688 question)

No tutorial or final project are required for audit

We discourage final projects consisting of some full-credit participants and some auditors, unless you have a very good reason

Course videos

All lectures will be recorded, made available on the course website (a permanent link to all the videos will also be posted)

Attendance still required for the Section A students (more on this in a moment)

Videos are being made **publicly available** this semester, so be aware of this if you sit nearby the camera

Note that even if you ask a question in class, the video likely will not pick up your voice (I need to repeat questions after they are asked)

Grading

Grading breakdown is posted on the web site (updated):

50% homework
15% tutorial
25% class project
10% class participation

Final grades are assigned on a curve (separate for 15-388 and 15-688 versions)

Homeworks

One homework assignment every two weeks: released on Thursdays by midnight, due the Thursday two weeks later at midnight (though first homework is already released, **due 9/12**)

We may miss this deadline sometimes (we are sorry in advance, we will of course also extend the due date)

Work will be largely (solely?) about **writing code** to solve problems

Homeworks are are in the form of Jupyter notebooks, **solutions autograded by Diderot:** http://diderot.one

Autograding

The meta-goal for this course is to have a *scalable* introduction to data science

We believe that the current best way to achieve scalability is through heavy use of autograding

This presents additional problem for data science, where part of the process is developing scientific conclusions from the data (this is what the class project is for)

Note: tutorial and class project will be graded manually (by myself)

Late days

Assignments are due at 11:59pm (midnight) on Thursdays

You have **5 late days** to use over the course of the semester

Each assignment can use a maximum of **2 late days** (midnight Saturday)

You cannot use late days for final project submission

Class participation

For 15-388/688A (in-class sections), class attendance is required: class participation grade will come from **participating in in-class Diderot polls** (you don't need to submit the right answer, just an answer)

For 15-688B (online section), you will need to watch all the videos lectures (Panopto system tracks this), and **answer a short quiz, within one week** of the lecture

If you are in Section A and miss a class, you should watch the video and take the corresponding quiz; if you are in the B section and attend class (and answer poll), you don't need to watch the video or answer the quiz

Additional extra credit participation for *answering* student questions on Diderot

Tutorial

The best way to learn a subject is to teach it

In lieu of a midterm, students will design a mini-tutorial, in the form of a Jupyter notebook, on a subject of their choice (though we will also provide suggestions)

Your tutorial will be read by the instructors, but also by other students, and peer grading will factor in to your final grade on the tutorial

Class project

A major component of the class: goal is to take a real-world domain that you are interested in, and apply data science methodologies to gain insight into the domain

Work to be done in groups of 2-3 students

Final report will be a Jupyter Notebook working through the analysis of your data, including code and visual results

Also presented in a video presentation (in lieu of final)

Class projects *must* be focused on some real data problem (ideally one that you collect yourself), not an already-curated data set

Academic integrity and homeworks

All submitted content (code and prose for homeworks, tutorials, and and final project) must be your own original content

You can discuss ideas and methodology for the homeworks or tutorial with other students in the course, but **you must write your solutions completely independently**

We will be running automated code-checking tools to assess similar submissions or submissions that use code from other sources

You may use snippets of code from sources like Stack Overflow, as long as you cite these properly (put a comment above and below whatever portion of code is copied), but be reasonable

Student well-being

CMU and courses like this one are stressful environments

In my experience, most academic integrity violations are the product of these environments and decisions made out of desperation

Please don’t let it get to this point (or potentially much worse)

Don’t sacrifice quality of life for this course: still make time to sleep, eat well, exercise

Up next

Next class: web scraping and data collection

First homework released today, use it as a gauge (after a few of the next lectures) to determine if the course is right for you

15-388/688 - Practical Data Science: Recommender systems

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Recommender systems

Collaborative filtering

User-user and item-item approaches

Matrix factorization

Outline

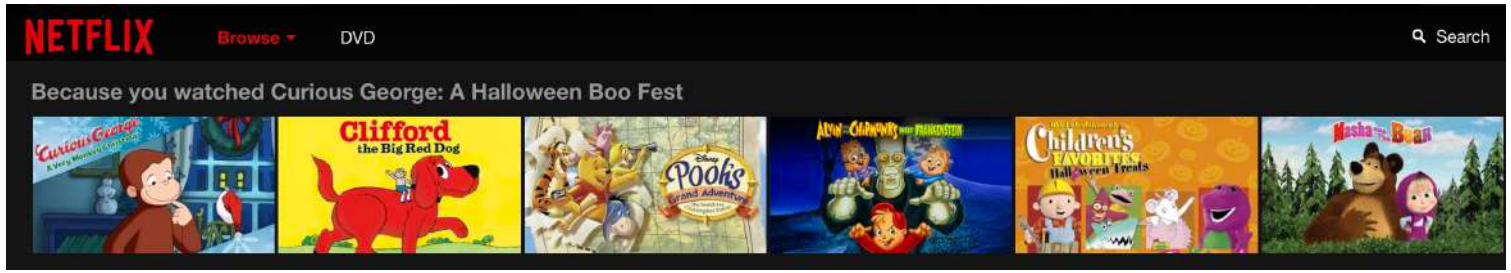
Recommender systems

Collaborative filtering

User-user and item-item approaches

Matrix factorization

Recommender systems



Information we can use to make predictions

“Pure” user information:

- Age
- Location
- Profession

“Pure” item information:

- Movie budget
- Main actors
- (Whether it is a Netflix release)

User-item information:

- Which items are most similar to those I have bought before?
- What items have users most similar to me bought?

Supervised or unsupervised?

Do recommender systems fit more within the “supervised” or “unsupervised” setting?

Like supervised learning, there are known outputs (items that the uses purchases), but like unsupervised learning, we want to find structure/similarity between users/items

We won’t worry about classifying this as just one or the other, but we will again formulate the problem within the three elements of a machine learning algorithm: 1) hypothesis function, 2) loss function, 3) optimization

Challenges in recommender systems

There are many challenges beyond what we will consider here in recommender systems:

1. Lack of user ratings / only “presence” data

2. Balancing personalization with generic “good” items

3. Privacy concerns

Historical note: Netflix Prize

Public competition ran from 2006 to 2009, goal was to produce a recommender system with 10% improvement in RMSE over existing Netflix system (based upon item-item Pearson correlation plus linear regression), $1M prize

Sparked a great deal of research in collaborative filtering, especially matrix factorization techniques

Larger impacts: put “data science competitions” in the public eye, emphasized practical importance of ensemble methods (though winning solution was never fielded)

Outline

Recommender systems

Collaborative filtering

User-user and item-item approaches

Matrix factorization

Collaborative filtering

Collaborative filtering refers to recommender systems that make recommendations *based solely upon the preferences that other users have indicated for these item (e.g., past ratings)*

The mathematical setting to have in mind in that of a matrix with mostly unknown entries

$$X = \begin{bmatrix} 1 & & & 3 \\ & 2 & 5 & \\ & 3 & & 5 \\ 4 & & 4 & \end{bmatrix}$$

rows correspond to different users

columns correspond to different items

entries correspond to *known* (given by user) scores for that user, for that items

Matrix view of collaborative filtering

Collaborative filtering X matrix is *sparse*, but unknown entries do not correspond to zero, are just missing

Goal is to “fill in” the missing entries of the matrix

$$X = \begin{bmatrix} 1 & ? & ? & 3 \\ ? & 2 & 5 & ? \\ ? & 3 & ? & 5 \\ 4 & ? & 4 & ? \end{bmatrix}$$

Approaches to collaborative filtering

User – user approaches: find the users that are most similar to myself (based upon only those items that are rated for *both* of us), and predict scores for other items based upon the average

Item – item approaches: find the items most similar to a given item (based upon all users rated both items), and predict scores for other users based upon the average

Matrix factorization approaches: find some low-rank decomposition of the X matrix that agrees at observed values

Outline

Recommender systems

Collaborative filtering

User-user and item-item approaches

Matrix factorization

User-user and item-item approaches

Basic intuition of user-user approach: find other users who are similar to me, e.g. by correlation coefficient or cosine similarity, look at how they ranked other items that I did not rank

One difference: correlation coefficient, etc, are only defined for vectors of the same size, so we only typically compute correlation across items that both users ranked

$$X = \begin{bmatrix} 1 & ? & ? & 3 \\ ? & 2 & 5 & ? \\ ? & 3 & ? & 5 \\ 4 & ? & 4 & ? \end{bmatrix}$$

Item-item approaches do the same thing but by column instead of row

User-user approach: formally

To match with our previous notation as much as possible, we will our prediction of X_{ij} as $\hat{X}_{ij}$ (we will later also refer to this as $h_\theta(i,j)$, our hypothesis evaluated on point i,j)

User-user methods typically make predictions:

$$\hat{X}_{ij} = \bar{x}_i + \frac{\sum_{k:X_{kj}\neq 0} w_{ik}\left(X_{kj} - \bar{x}_k\right)}{\sum_{k:X_{kj}\neq 0} w_{ik}}$$

- $\bar{x}_i$ - mean of user i's ratings
- w_{ik} - similarity function between users i and k

Common modification: restrict sum to only K users "most similar" to i

Similarity measures

How do we measure similarity between two users?

Two example approaches:

1. Pearson correlation ($\mathcal{J}_{ik}$ denotes items ranked by users i and k):

$$w_{ik} = \frac{\sum_{j \in \mathcal{J}_{ik}} (X_{ij} - \bar{x}_i)(X_{kj} - \bar{x}_k)}{\left(\sum_{j \in \mathcal{J}_{ik}} (X_{ij} - \bar{x}_i)^2 \cdot \sum_{j \in \mathcal{J}_{ik}} (X_{kj} - \bar{x}_k)^2\right)^{1/2}}$$

2. Raw cosine similarity (treating missing as zero):

$$w_{ik} = \frac{\sum_j X_{ij} \cdot X_{kj}}{\left(\sum_j X_{ij}^2 \cdot \sum_j X_{kj}^2\right)^{1/2}}$$

Item-item approaches

Item-item approaches just do the same process flipping rows/columns

Make predictions:

$$\hat{X}_{ij} = \bar{x}_j + \frac{\sum_{k:X_{ik}\neq 0} w_{jk}\,(X_{ik} - \bar{x}_k)}{\sum_{k:X_{ik}\neq 0} w_{jk}}$$

Similarity function, e.g.:

$$w_{jk} = \frac{\sum_{i\in\mathcal{J}_{jk}} (X_{ij} - \bar{x}_j)(X_{ik} - \bar{x}_k)}{\left(\sum_{i\in\mathcal{J}_{jk}} (X_{ij} - \bar{x}_j)^2 \cdot \sum_{i\in\mathcal{J}_{jk}} (X_{ik} - \bar{x}_k)^2\right)^{1/2}}$$

Poll: efficiency of user and item based method

Suppose we have many more users than items. Assuming we use dense matrix operations for everything, which method would be more efficient for computing *all* the predictions $\hat{X}_{ij}$ for all missing elements?

1. The user-user approach will be more efficient

2. The item-item approach will be more efficient

3. They will both have the same complexity

Outline

Recommender systems

Collaborative filtering

User-user and item-item approaches

Matrix factorization

Matrix factorization approach

Approximate the i, j entry of $X \in \mathbb{R}^{m \times n}$ as $\hat{X}_{ij} = u_i^T v_j$ where $u_i \in \mathbb{R}^k$ denotes user-specific weights and $v_j \in \mathbb{R}^k$ denotes item-specific weights

1. Hypothesis function

$$\hat{X}_{ij} = h_\theta(i, j) = u_i^T v_j, \qquad \theta = \{u_{1:m}, v_{1:n}\}$$

2. Loss function: squared error (on observed entries)

$$\ell(h_\theta(i, j), X_{ij}) = (h_\theta(i, j) - X_{ij})^2$$

leads to optimization problem (S denotes set of observed entries)

$$\underset{\theta}{\text{minimize}} \sum_{i,j \in S} \ell\left(h_\theta(i, j), X_{ij}\right)$$

Optimization approaches

3. How do we optimize the matrix factorization objective? (Like k-means, EM, possibility of local optima)

Consider the objective with respect to a *single* u_i term:

$$\underset{u_i}{\text{minimize}} \sum_{j:(i,j)\in S} (v_j^T u_i - X_{ij})^2$$

This is just a least-squares problem, can solve analytically:

$$u_i = \left(\sum_{j:(i,j)\in S} v_j v_j^T \right)^{-1} \left(\sum_{j:(i,j)\in S} v_j X_{ij} \right)$$

Alternating minimization algorithm: Repeatedly solve for all u_i for each user, v_j for each item (may not give global optimum)

Matrix factorization interpretation

What we are effectively doing here is factorizing X as a low rank matrix

$$X \approx UV, \qquad U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times n}$$

where

$$U = \begin{bmatrix} - & u_1^T & - \\ & \vdots & \\ - & u_m^T & - \end{bmatrix}, \qquad V = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix}$$

However, we are *only* requiring the X match the factorization at the observed entries of X

Relationship to PCA

PCA also performs a factorization of $X \approx UV$ (if you want to follow the precise notation of the PCA slides, it would actually be $X^T = UV$ where V contains the columns $Wx^{(i)}$)

But unlike collaborative filtering, in PCA, *all* the entries of X are observed

Though we won't get into the details: this difference is what lets us solve PCA exactly, while we can only solve matrix factorization for collaborative filtering locally

15-388/688 - Practical Data Science: Deep learning

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Outline

Recent history in machine learning

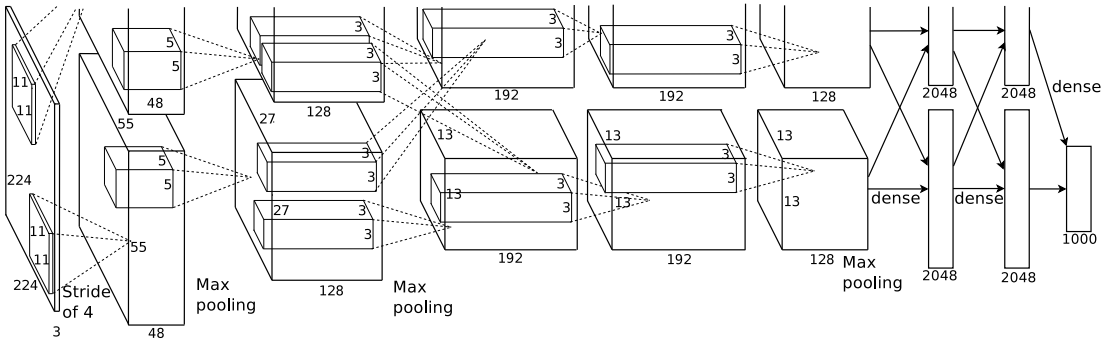
Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

AlexNet

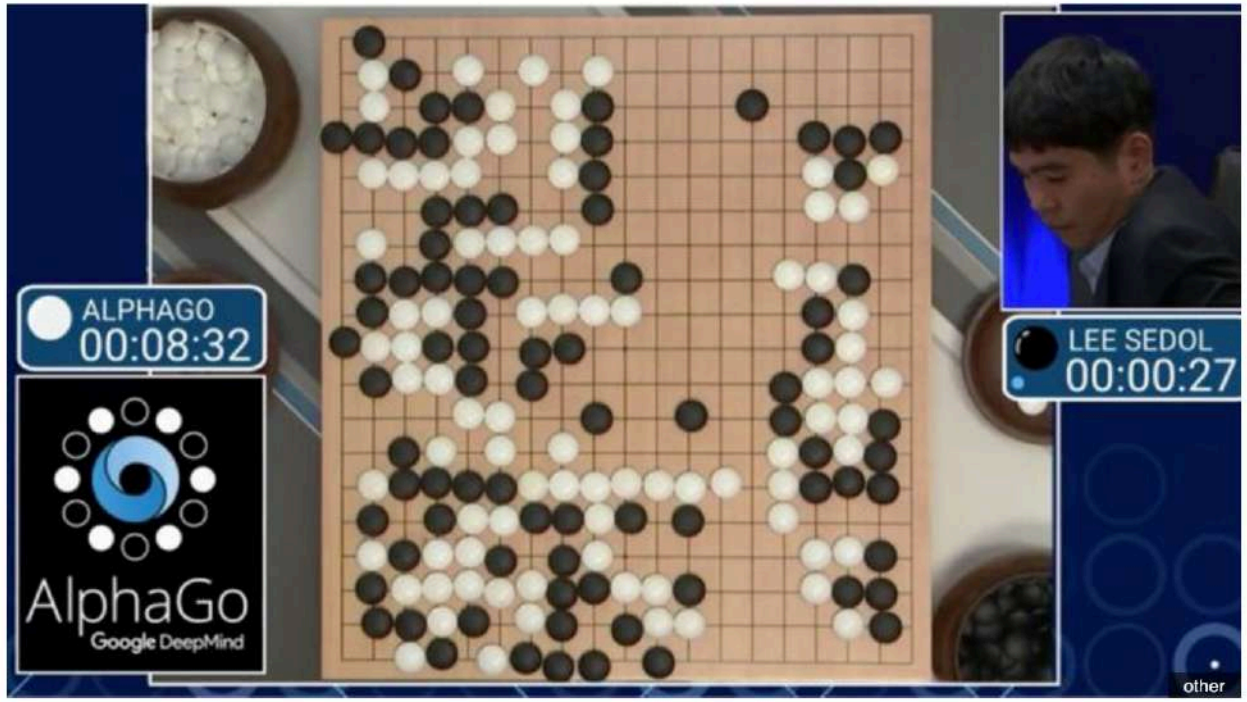


"AlexNet" (Krizhevsky et al., 2012), winning entry of ImageNet 2012 competition with a Top-5 error rate of 15.3% (next best system with highly engineered features based got 26.1% error)

AlphaGo

Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol

12 March 2016 | Technology



Google Translate

In November 2016, Google transitioned it's translation service to a deep-learning-based system, dramatically improved translation quality in many settings

Kilimanjaro is 19,710 feet of the mountain covered with snow, and it is said that the highest mountain in Africa. Top of the west, "Ngaje Ngai" in the Maasai language, has been referred to as the house of God. The top close to the west, there is a dry, frozen carcass of a leopard. Whether the leopard had what the demand at that altitude, there is no that nobody explained.

Kilimanjaro is a mountain of 19,710 feet covered with snow and is said to be the highest mountain in Africa. The summit of the west is called "Ngaje Ngai" in Masai, the house of God. Near the top of the west there is a dry and frozen dead body of leopard. No one has ever explained what leopard wanted at that altitude.

https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Neural networks for machine learning

The term “neural network” largely refers to the hypothesis class part of a machine learning algorithm:

1. **Hypothesis:** non-linear hypothesis function, which involve compositions of multiple linear operators (e.g. matrix multiplications) and elementwise non-linear functions

2. **Loss:** “Typical” loss functions for classification and regression: logistic, softmax (multiclass logistic), hinge, squared error, absolute error

3. **Optimization:** Gradient descent, or more specifically, a variant called stochastic gradient descent we will discuss shortly

Linear hypotheses and feature learning

Until now, we have (mostly) considered machine learning algorithms that linear hypothesis class

$$h_\theta(x) = \theta^T \phi(x)$$

where $\phi: \mathbb{R}^n \to \mathbb{R}^k$ denotes some set of typically non-linear features

Example: polynomials, radial basis functions, custom features like TFIDF (in many domains every 10 years or so there would be new feature types)

The performance of these algorithms depends crucially on coming up with good features

Key question: can we come up with an algorithm that will automatically *learn* the features themselves?

Feature learning, take one

Instead of a simple linear classifier, let's consider a two-stage hypothesis class where one linear function creates the features and another produces the final hypothesis

$$h_\theta(x) = W_2\phi(x) + b_2 = W_2(W_1 x + b_1) + b_2,$$
$$\theta = \{W_1 \in \mathbb{R}^{k\times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1\times k}, b_2 \in \mathbb{R}\}$$

But there is a problem:

$$h_\theta(x) = W_2(W_1 x + b_1) + b_2 = \widetilde{W}x + \tilde{b}$$

i.e., we are still just using a linear classifier (the apparent added complexity is actually not changing the underlying hypothesis function)

Neural networks

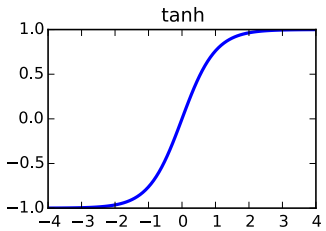
Neural networks are a simple extension of this idea, where we additionally apply a non-linear function after each linear transformation

$$h_\theta(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

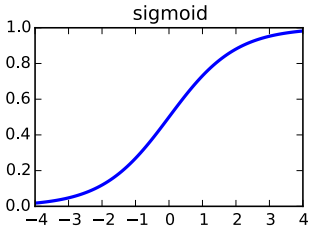
where $f_1, f_2 \colon \mathbb{R} \to \mathbb{R}$ are a non-linear function (applied elementwise)

Common choices of f_i:

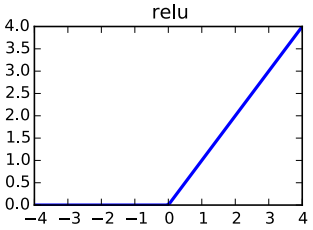
Hyperbolic tangent: $f(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$



Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

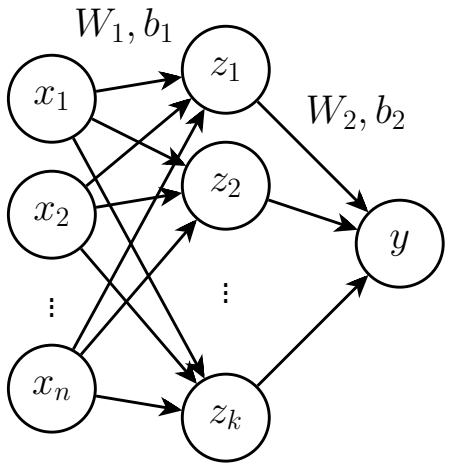


Rectified linear unit (ReLU): $f(x) = \max\{x, 0\}$



Illustrating neural networks

We can illustrate the form of neural networks using figures like the following

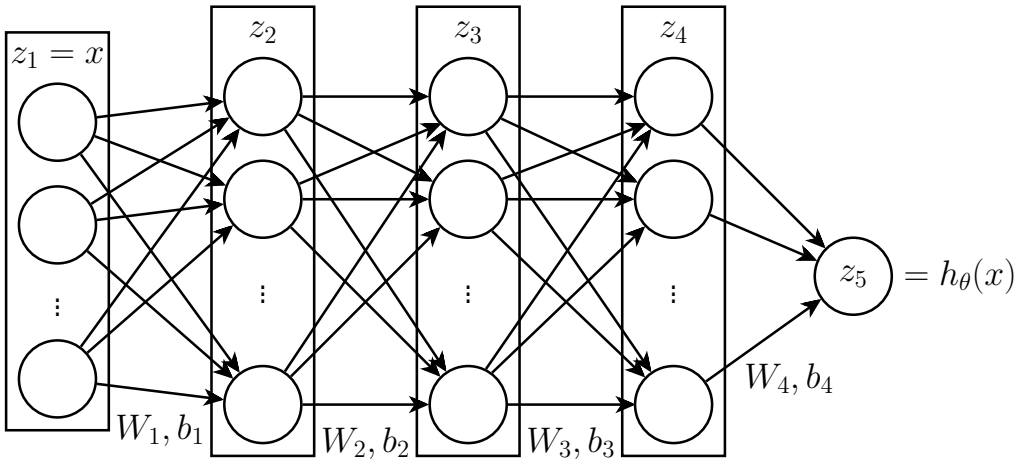


Middle layer z is referred to as the *hidden layer* or *activations*

These are the learned features, nothing in the data prescribed what values they should take, left up to algorithm to decide

Deep learning

"Deep learning" refers (almost always) to machine learning using neural network models with multiple hidden layers



Hypothesis function for k-layer network

$$z_{i+1} = f_i(W_i z_i + b_i), \qquad z_1 = x, \qquad h_\theta(x) = z_k$$

(note the z_i here refers to a vector, not an entry into vector)

Properties of neural networks

A neural network will a single hidden layers (and enough hidden units) is a *universal function approximator*, can approximate *any* function over inputs

In practice, not *that* relevant (similar to how polynomials can fit any function), and the more important aspect is that they appear to work very well in practice for many domains

The hypothesis $h_\theta(x)$ is not a convex function of parameters $\theta = \{W_i, b_i\}$, so we have possibility of local optima

Architectural choices (how many layers, how they are connected, etc), become important algorithmic design choices (i.e. hyperparameters)

Why use deep networks

Motivation from circuit theory: many function can be represented more efficiently using deep networks (e.g., parity function requires $O(2^n)$ hidden units with single hidden layer, $O(n)$ with $O(\log n)$ layers

- But not clear if deep learning really learns these types of network

Motivation from biology: brain appears to use multiple levels of interconnected neurons

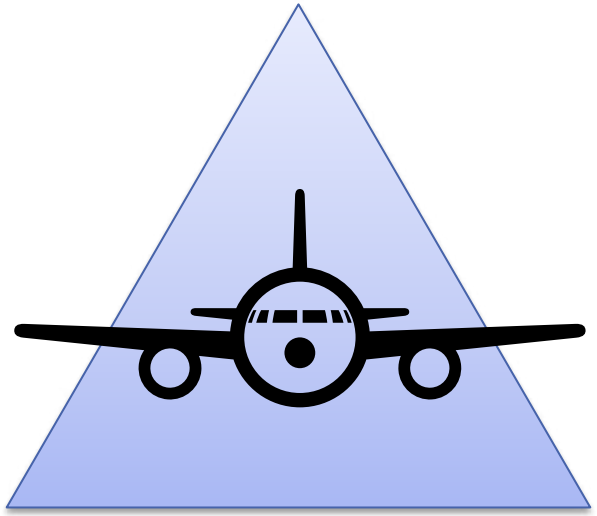
- But despite the name, the connection between neural networks and biology is extremely weak

Motivation from practice: works much better for many domains

- Hard to argue with results

Why now?

Better models and algorithms



Lots of computing power

Lots of data

Poll: Benefits of deep networks

What advantages would you expect of applying a deep network to some machine learning problem versus a (pure) linear classifier?

1. Less chance of overfitting data
2. Can capture more complex prediction functions
3. Better test set performance when the number of data points is small
4. Better training set performance when number of data points is small
5. Better test set performance when number of data points in large

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Neural networks for machine learning

Hypothesis function: neural network

Loss function: “traditional” loss, e.g. logistic loss for binary classification:

$$\ell(h_\theta(x), y) = \log(1 + \exp(-y \cdot h_\theta(x)))$$

Optimization: How do we solve the optimization problem

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

Just use gradient descent as normal (or rather, a version called stochastic gradient descent)

Stochastic gradient descent

Key challenge for neural networks: often have very large number of samples, computing gradients can be computationally intensive.

Traditional gradient descent computes the gradient with respect to the sum over *all examples,* then adjusts the parameters in this direction

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

Alternative approach, *stochastic gradient descent* (SGD): adjust parameters based upon just *one* sample

$$\theta := \theta - \alpha \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

and then repeat these updates for all samples

Gradient descent vs. SGD

Gradient descent, repeat:

- For $i = 1, \ldots, m$:

$$g^{(i)} \leftarrow \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

- Update parameters:

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^{m} g^{(i)}$$

Stochastic gradient descent, repeat:

- For $i = 1, \ldots, m$:

$$\theta \leftarrow \theta - \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$$

In practice, stochastic gradient descent uses a small collection of samples, not just one, called a *minibatch*

Computing gradients: backpropagation

So, how do we compute the gradient $\nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$?

Remember θ here denotes a set of parameters, so we're really computing gradients with respect to all elements of that set

This is accomplished via the *backpropagation* algorithm

We won't cover the algorithm in detail, but backpropagation is just an application of the (multivariate) chain rule from calculus, plus "caching" intermediate terms that, for instance, occur in the gradient of both W_1 and W_2

Training neural networks in practice

The other good news is also that you will rarely need to implement backpropagation yourself

Many libraries provides methods for you to just specify the neural network "forward" pass, and automatically compute the necessary gradients

Examples: Tensorflow, PyTorch

You'll use one of these a bit on the homework

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

Specialized neural network architectures

Deep learning in data science

Specialized architectures

Very little of the current wave of enthusiasm for deep learning has actually come from the simple “fully connected” neural network model we have seen so far

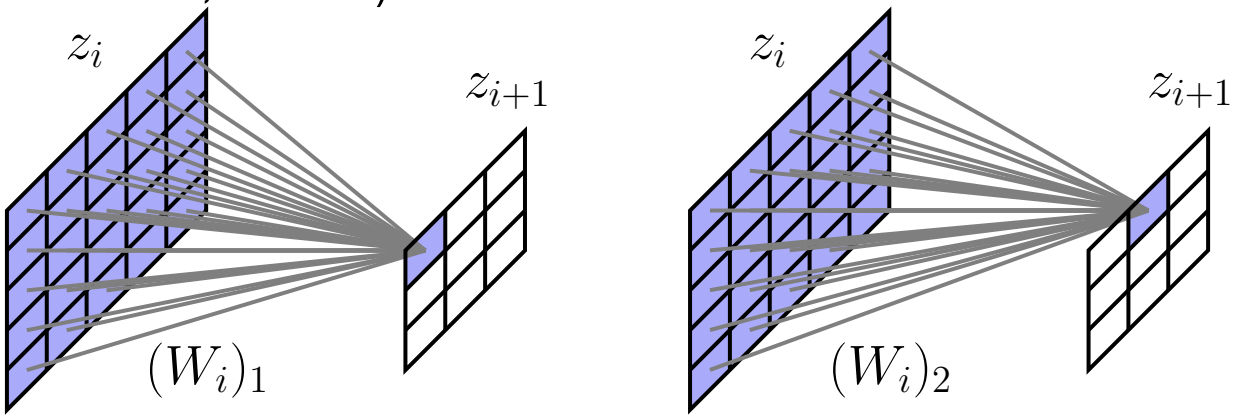
Instead, most of the excitement has come from two more specialized architectures: **convolutional neural networks**, and **recurrent neural networks**

The problem with fully-connected networks

A 256x256 (RGB) image means ~200,000 dimensional input

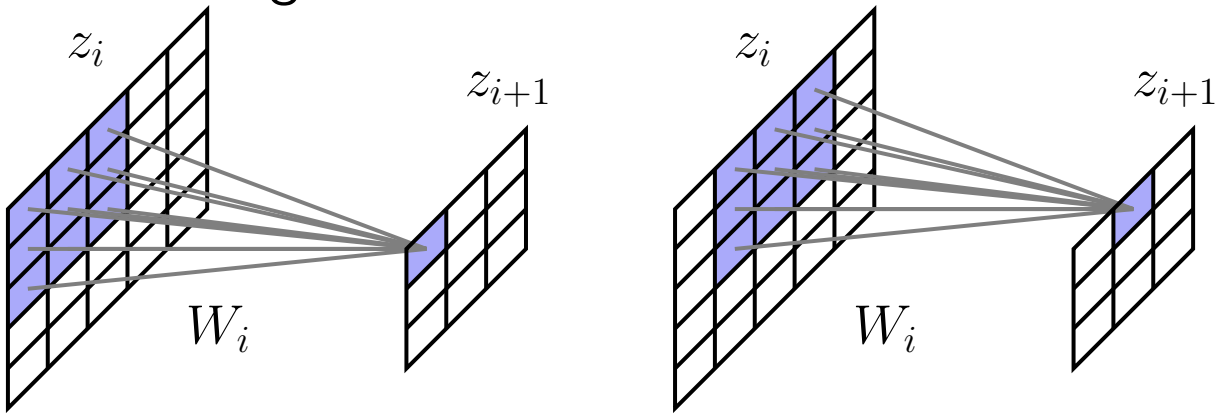
Fully connected deep network would require a huge number of parameters, very likely to overfit to data

A generic deep network also doesn't capture of the the "natural" *invariances* we expect in images (location, scale)

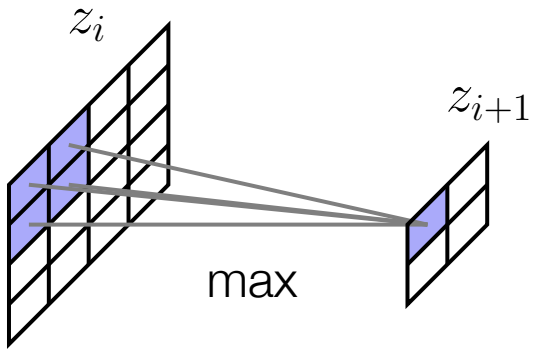


Convolutional neural networks

Constrain weights: require that activations in following layer be a “local” function of previous layer, and share weights across all locations

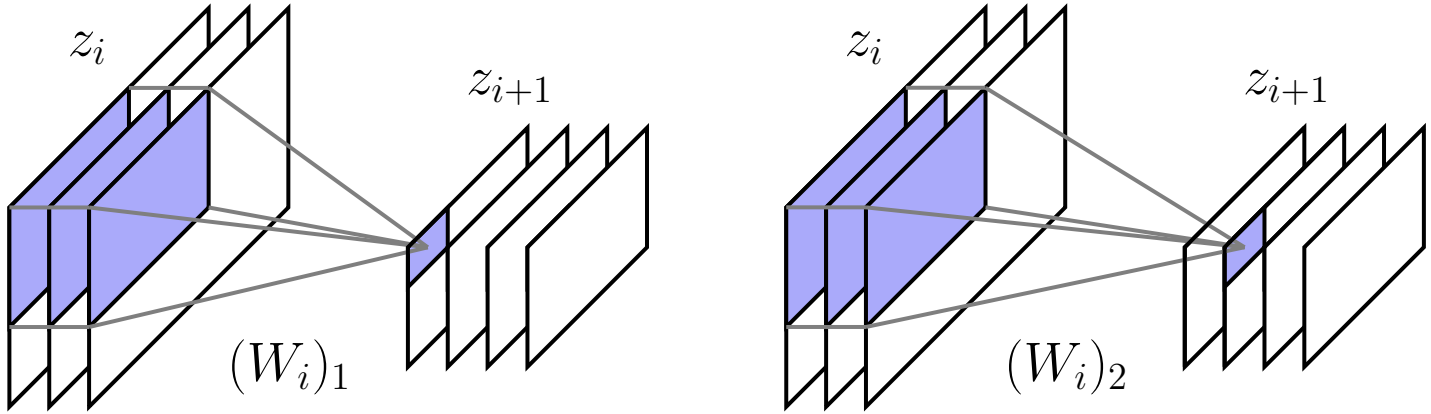


Also common to use *max-pooling* layers that take maximum over region



Convolutional networks in practice

Actually common to use “3D” convolutions to combine multiple channels, and use multiple convolutions at each layer to create different features



Convolutions are still linear operations, and we can take gradients using backpropagation in much the same manner

Predicting sequential data

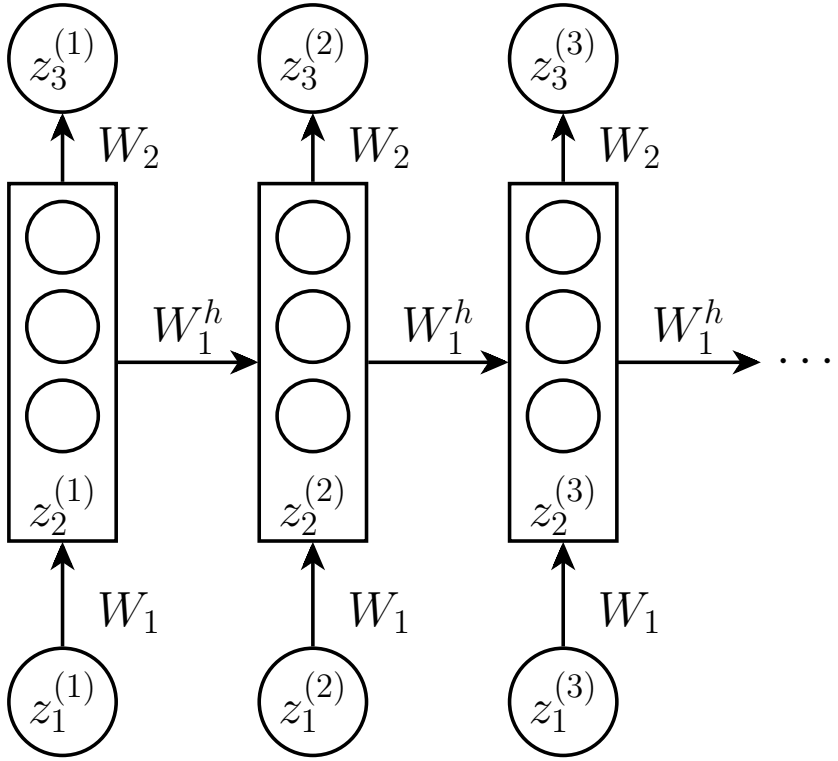
In practice, we often want to predict a *sequence* of outputs given a *sequence* of inputs

Just predicting each output independently would miss crucial information

Many examples: time series forecasting, sentence labeling, part of speech tagging, etc

Recurrent neural networks

Maintain state over time, activations are a function of current input and *previous* activations



$$z_{i+1}^{(t)} = f_i(W_i x^{(t)} + W_i^h z^{(t-1)} + b_i)$$
$$h_\theta(x^{(t)}) = z_k^{(t)}$$

Recurrent neural networks in practice

Traditional RNNs have trouble capturing long-term dependencies

More typical to use a more complex hidden unit and activations, called a long short term memory (LSTM) network

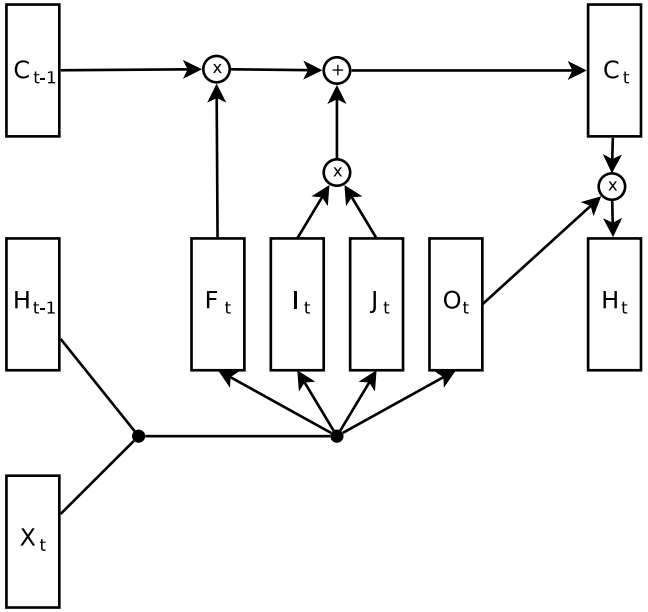


Figure from (Jozefowicz et al., 2015)

Outline

Recent history in machine learning

Machine learning with neural networks

Training neural networks

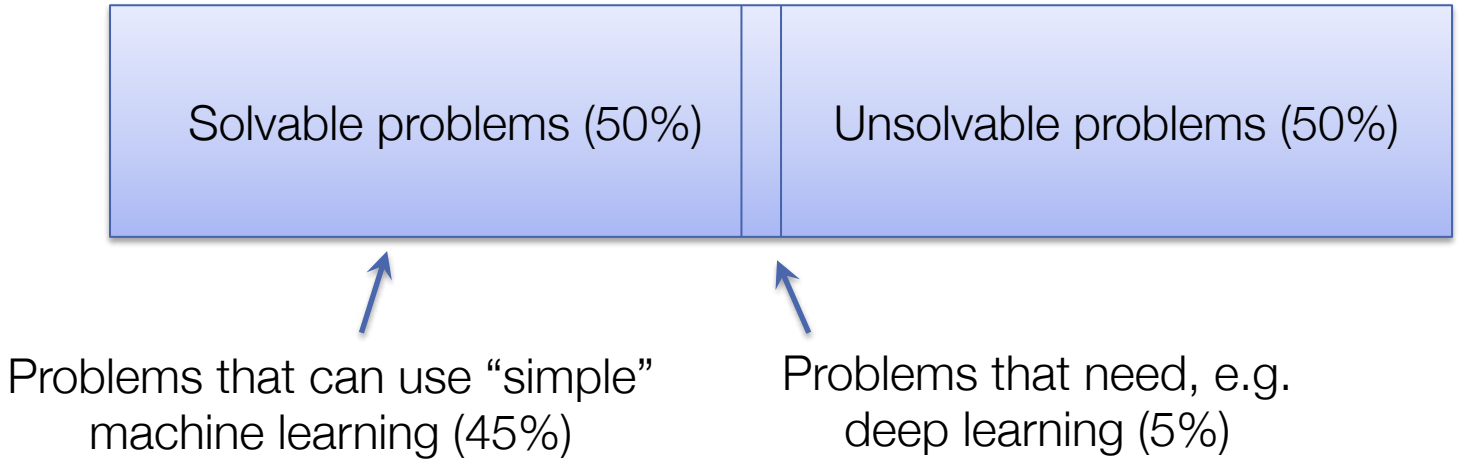
Specialized neural network architectures

Deep learning in data science

Deep learning in data science

What role does deep learning have to play in data science?

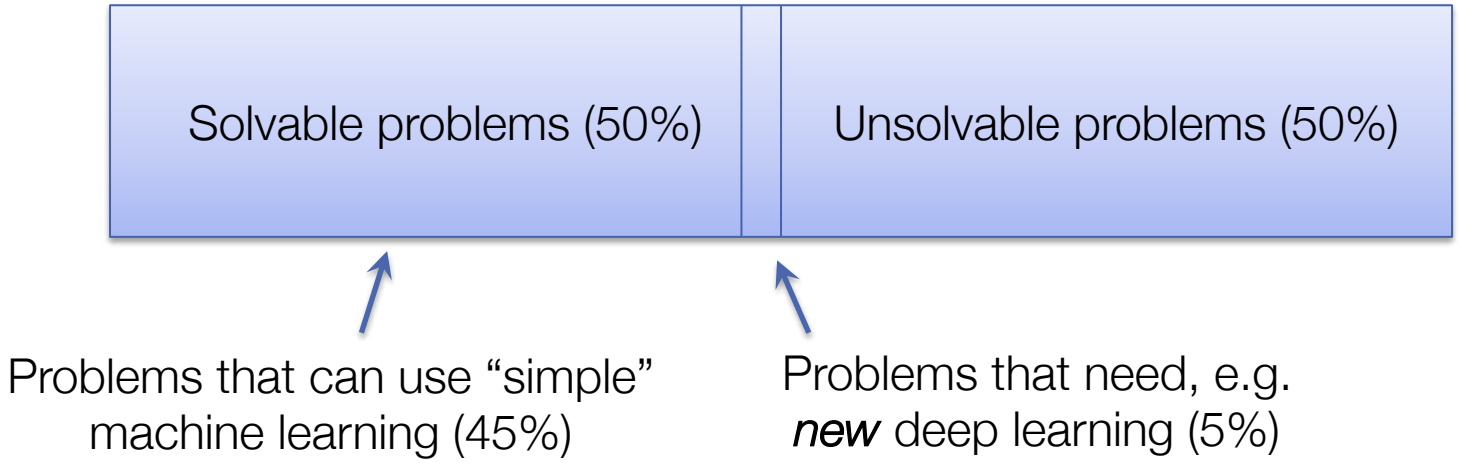
Data problems we would like to solve



Deep learning in data science

What role does deep learning have to play in data science?

Data problems we would like to solve



Solving data science problems with deep learning

When you come up against some machine learning problem with “traditional” features (i.e., human-interpretable characteristics of the data) do *not* try to solve it by applying deep learning methods first

Use linear regression/classification, linear regression/classification with non-linear features, or gradient boosting methods instead

If these still don’t solve your problem *and* you can visualize the data in a way that lets you solve it “manually”, *or* if you really want to squeeze out a 1-2% improvement in performance, *then* you can apply deep learning

The exceptions

However, it's also undeniable that deep learning has made remarkable progress for structured data like images, audio, or text

For these types of data, you can use an *already trained* network as a *feature extractor* (i.e., a way of mapping the data to some alternatively, probably lower dimensional representation)

Example: Image processing with VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure from Simonyan and Zisserman, 2015

VGG network (Simonyan and Zisserman, 2015), trained on ImageNet 1000-way classification of images

Given a new image classification problem, take pre-trained VGG network, take the last layer of weights, and use them as features

Can also “finetune” last few layers of a network to specialize to a new task

Example: text processing with word2vec

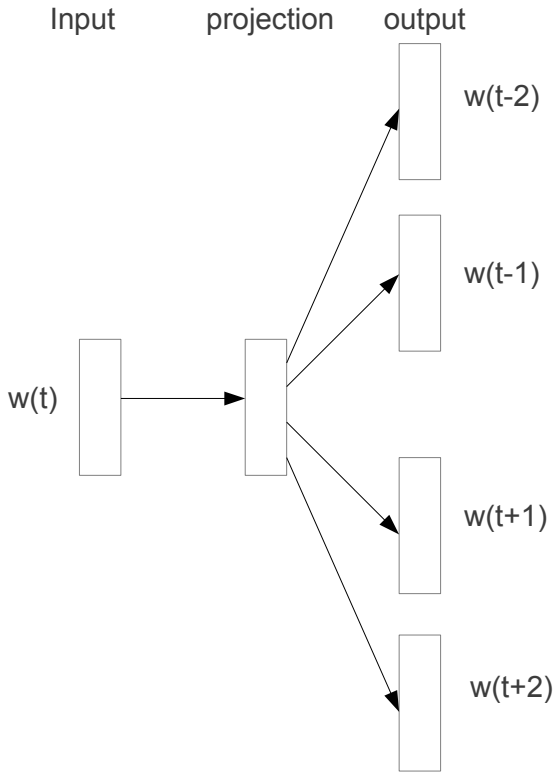


Figure from Mikolov, et al., 2013

word2vec (Mikolov, et al., 2013) is a method developed for predicting surrounding words from a given word

To do so, it creates an “embedding” for every word that acts as a good surrogate for the things this word can mean, pre-trained versions available

Bottom line: instead of using bag of words, use word2vec to get a vector representation of each word in a corpus

Example: text processing with BERT

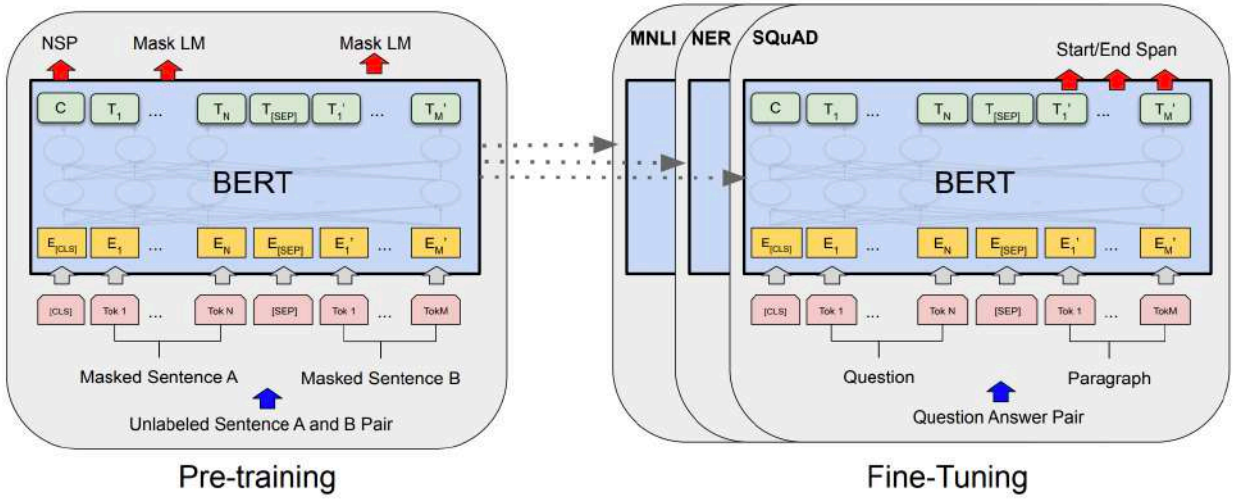


Figure from Devlin, et al., 2018

BERT (Bidirectional Encoder Representations from Transformers), (Devlin et al., 2018) trains a language model to predict missing elements of a sentence and predict one sentence from another for two sentence pairs

At application time, can fine-tune this generic model to many other possible tasks such as question answering, sentence classification, etc

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

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Announcements

HW 3 released, due 10/24 (not 10/17)

Feedback on tutorial sent to everyone who submitted by deadline, will send to remaining people by tomorrow evening

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?

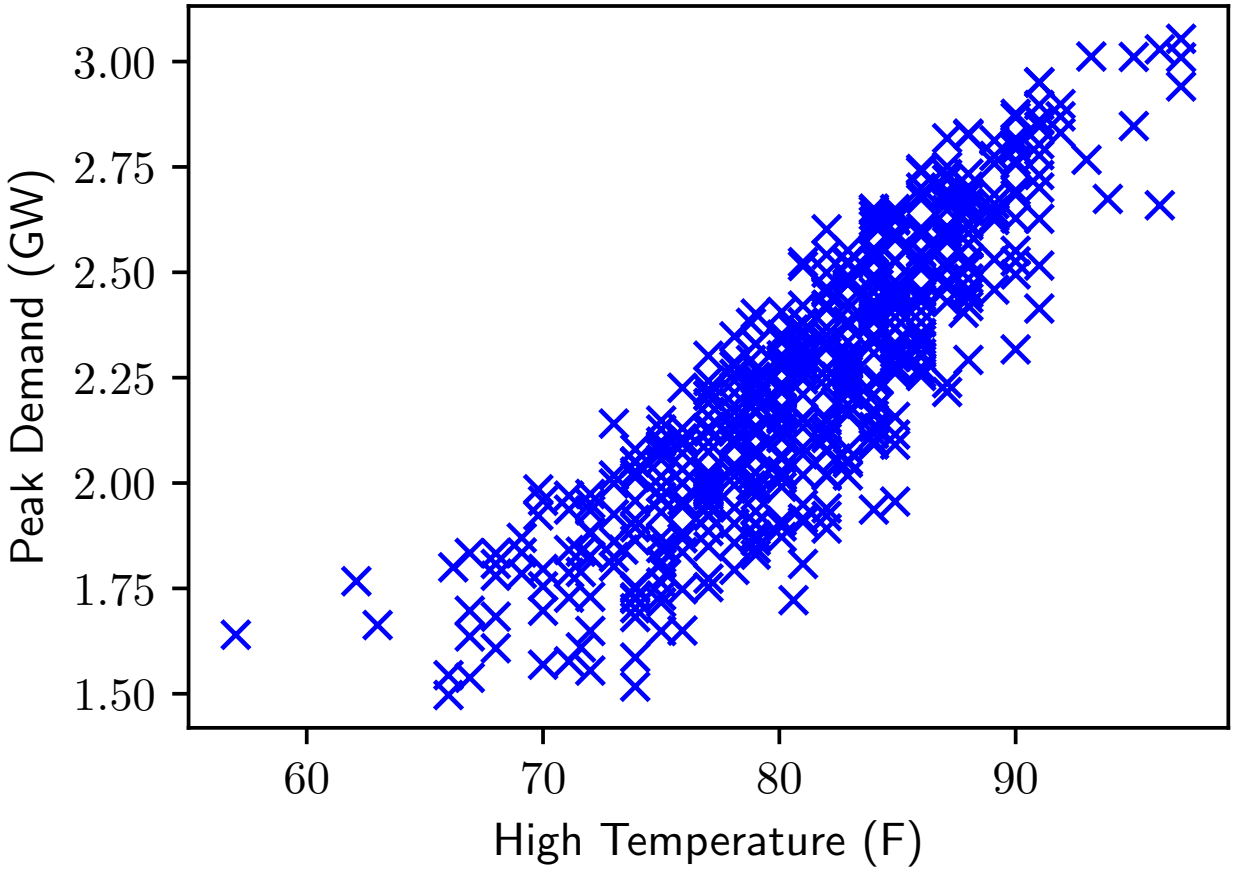
Difficult to build an “a priori” model from first principles to answer this question

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

$$E(\theta) = \sum_{i \in \text{days}} (\theta_1 \cdot \text{High_Temperature}^{(i)} + \theta_2 \ - \text{Peak_Demand}^{(i)})^2$$

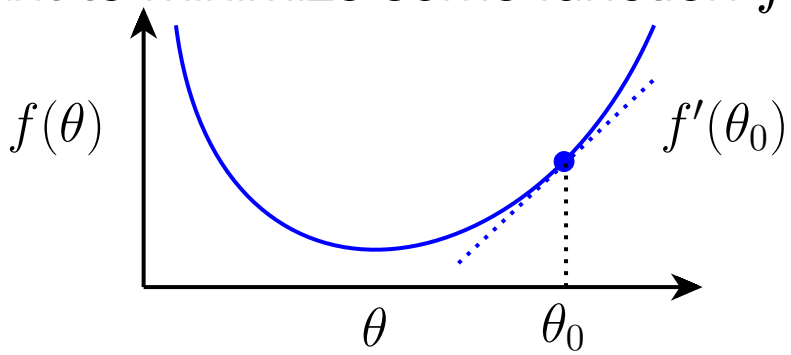
How do we find parameters?

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?

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

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:

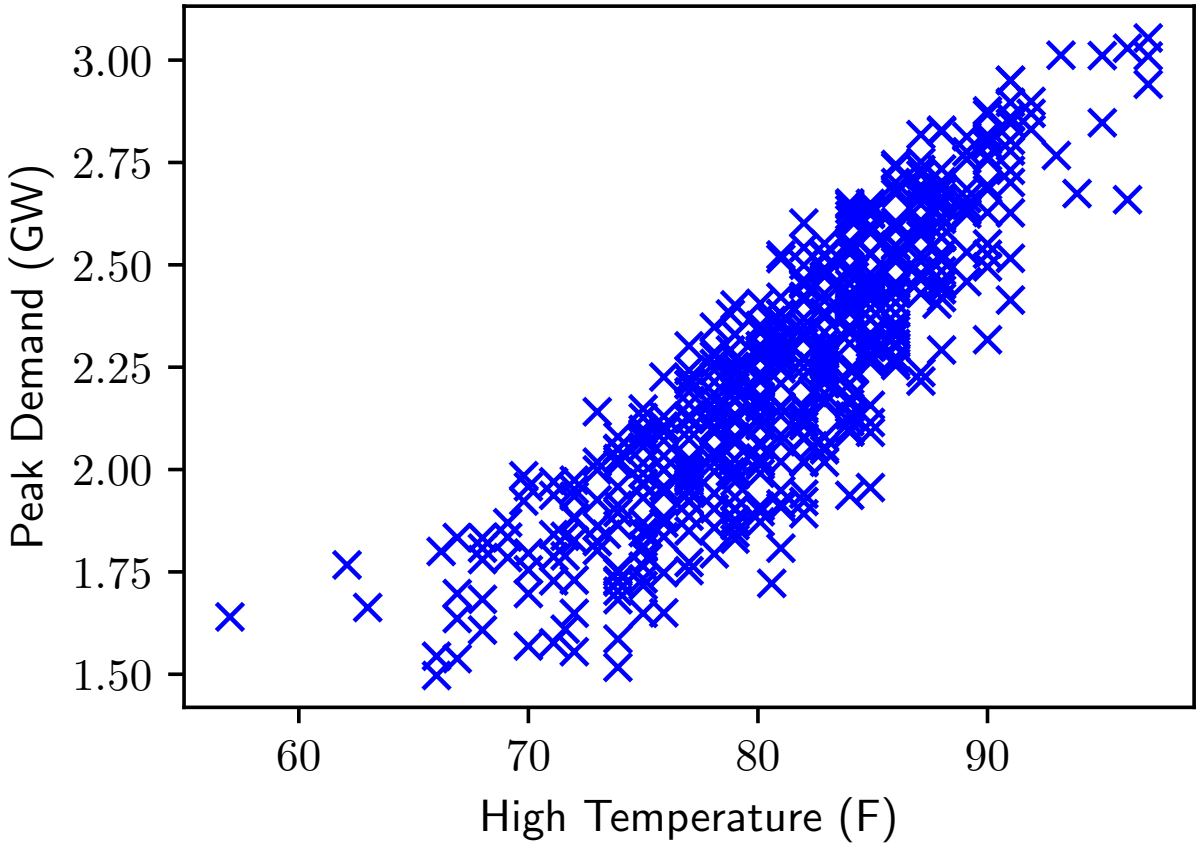
$$\theta_1 := \theta_1 - \alpha \sum_{i \in \text{days}} 2(\theta_1 \cdot x^{(i)} + \theta_2 \ - y^{(i)}) \cdot x^{(i)}$$

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

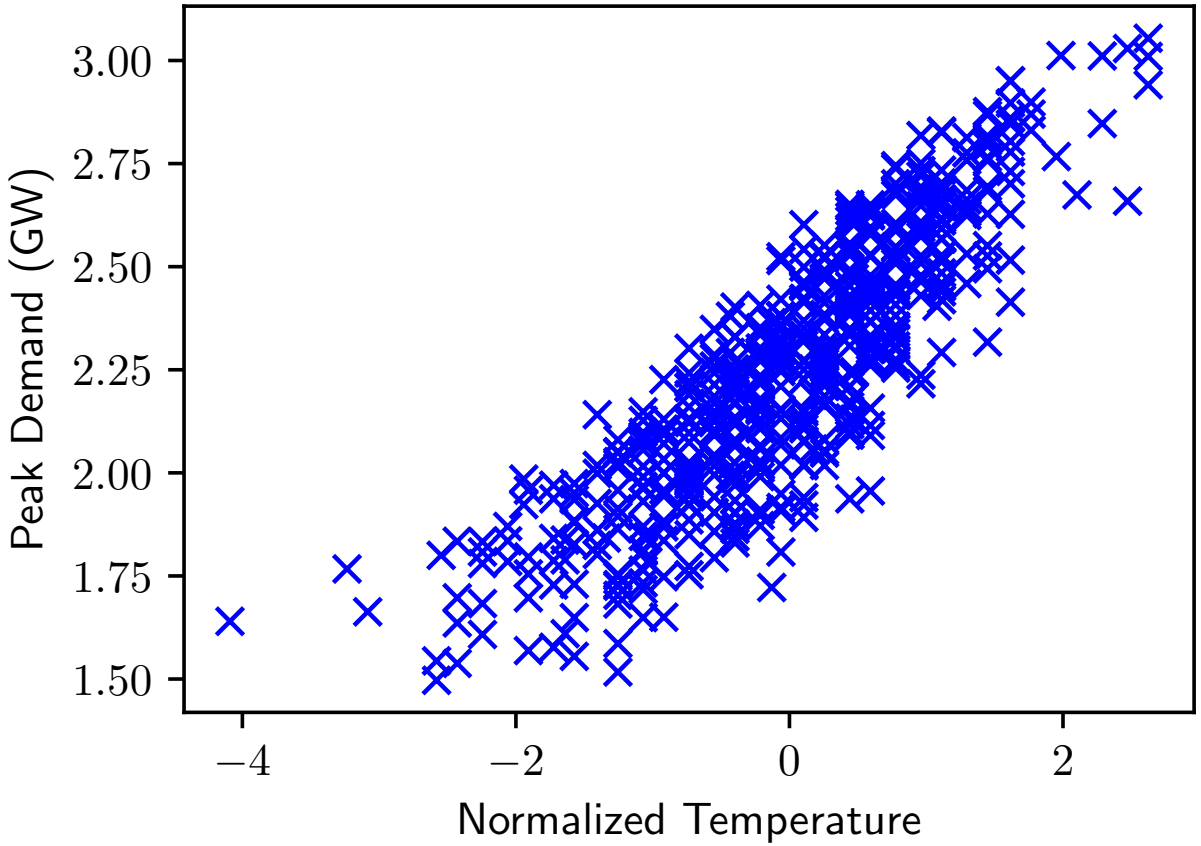
where α is some small positive number called the *step size*

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

Gradient descent

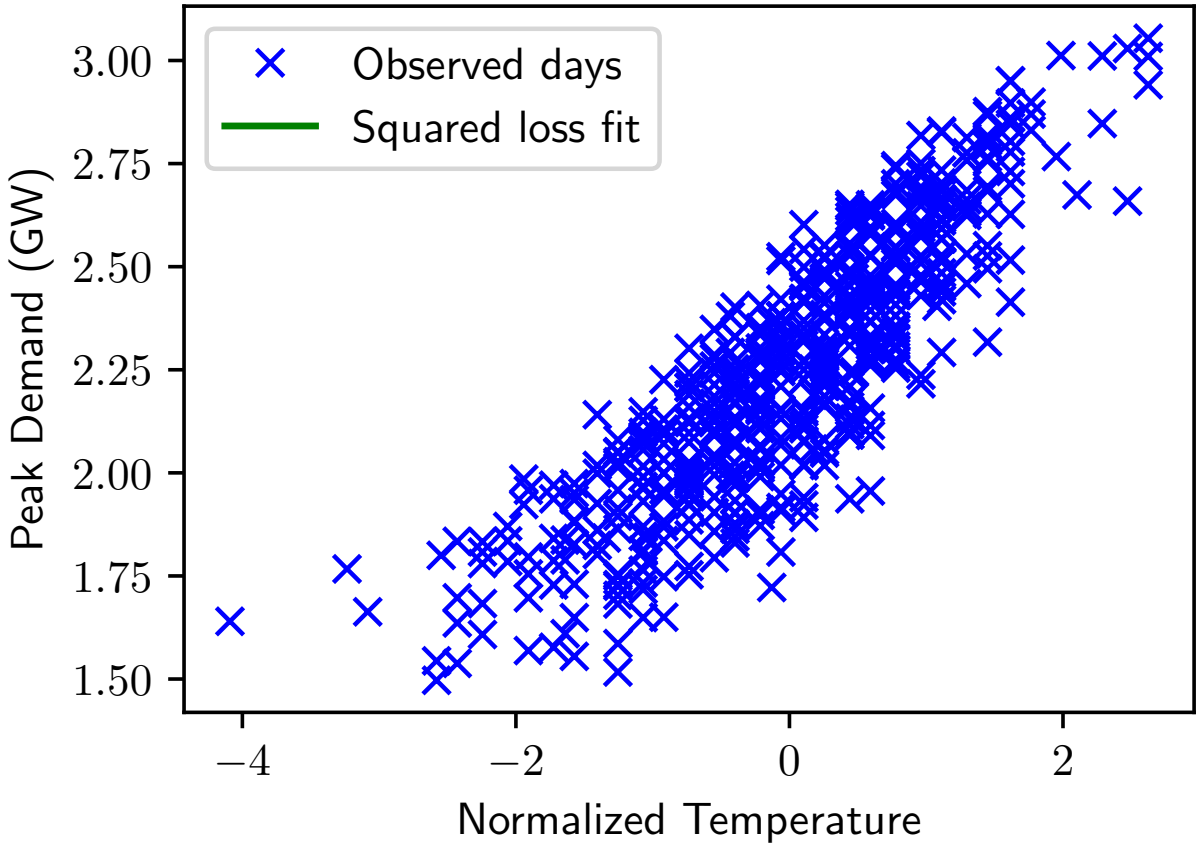


Gradient descent



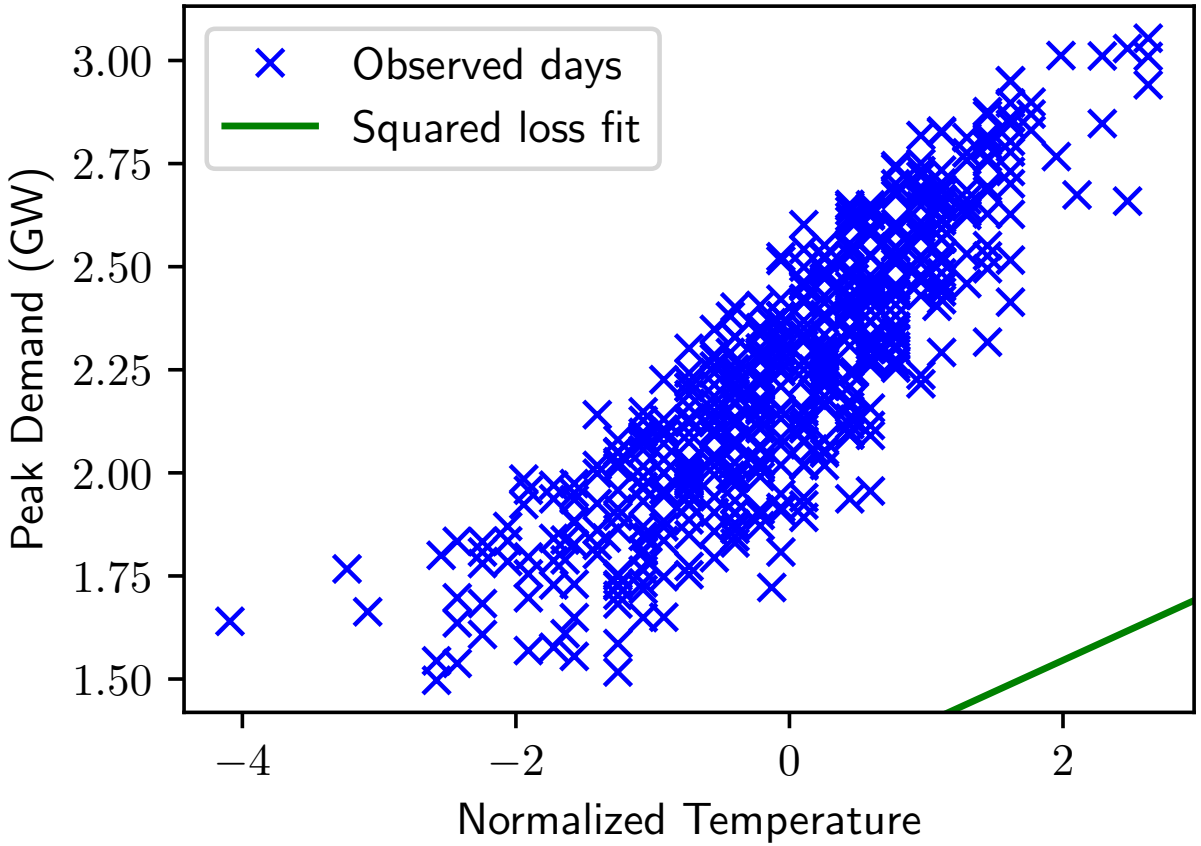
Normalize input by subtracting the mean and dividing by the standard deviation

Gradient descent – Iteration 1



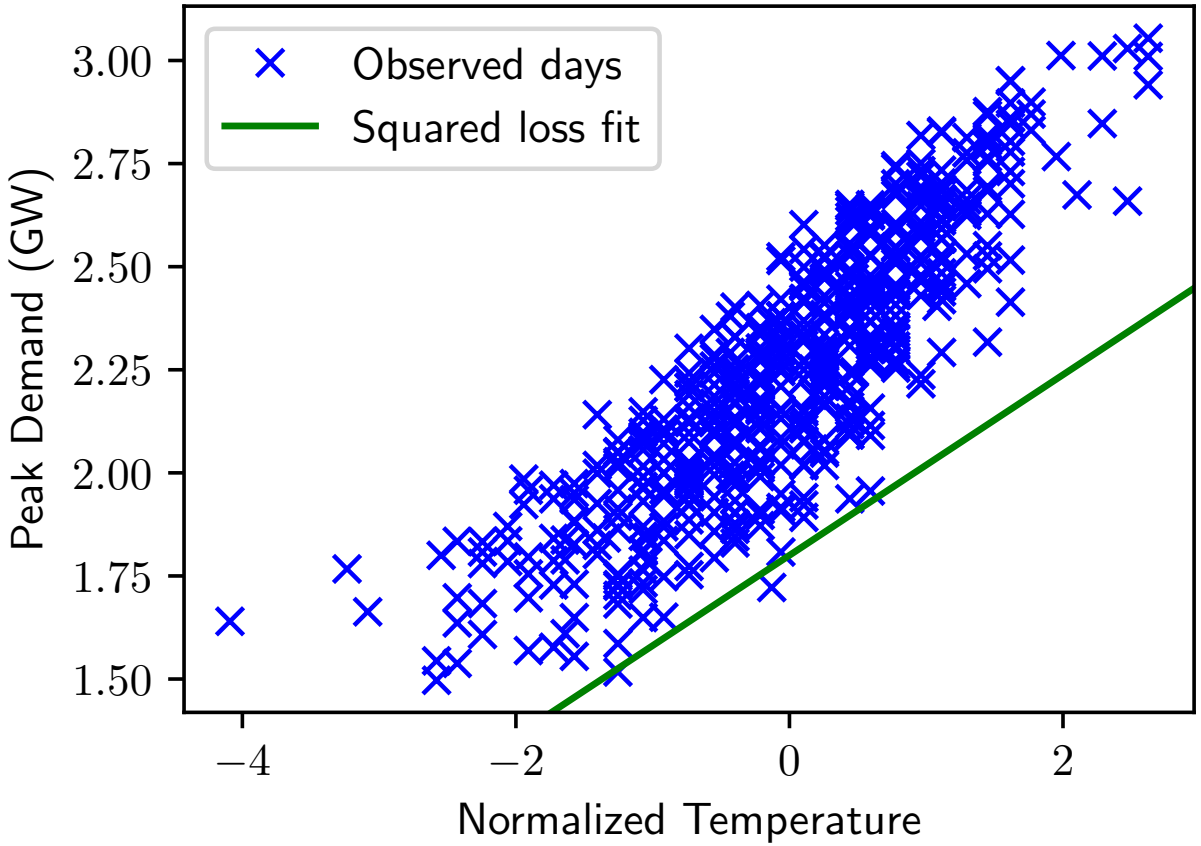
$\theta = (0.00, 0.00)$
$E(\theta) = 1427.53$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-151.20, -1243.10)$

Gradient descent – Iteration 2



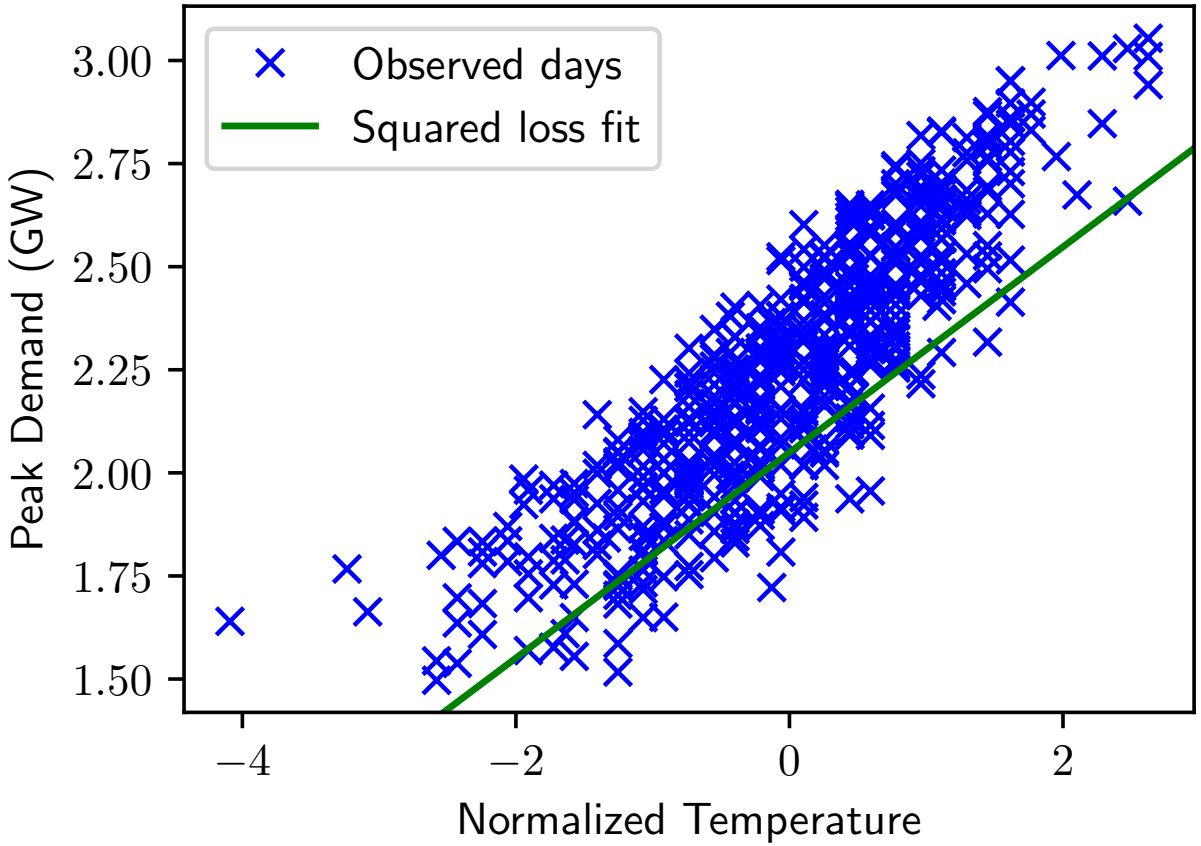
$\theta = (0.15, 1.24)$
$E(\theta) = 292.18$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-67.74, -556.91)$

Gradient descent – Iteration 3



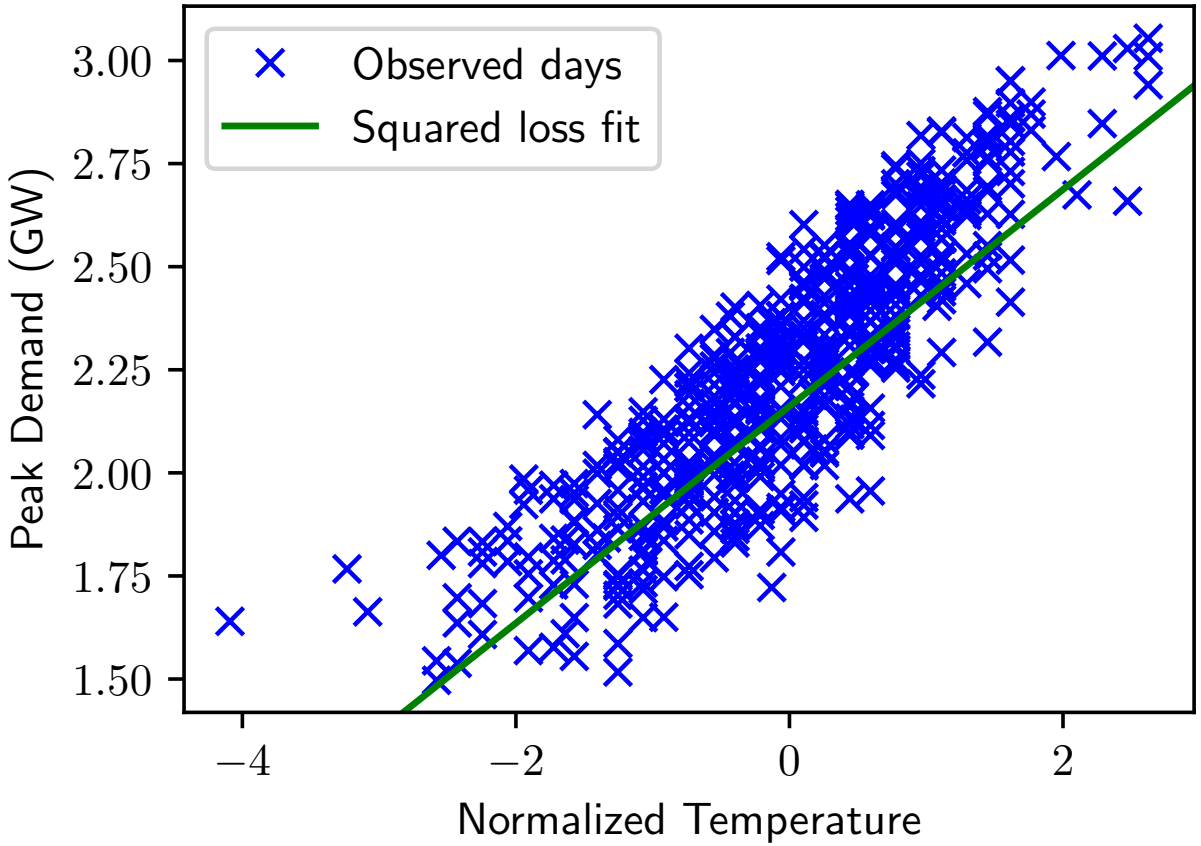
$\theta = (0.22, 1.80)$
$E(\theta) = 64.31$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-30.35, -249.50)$

Gradient descent – Iteration 4



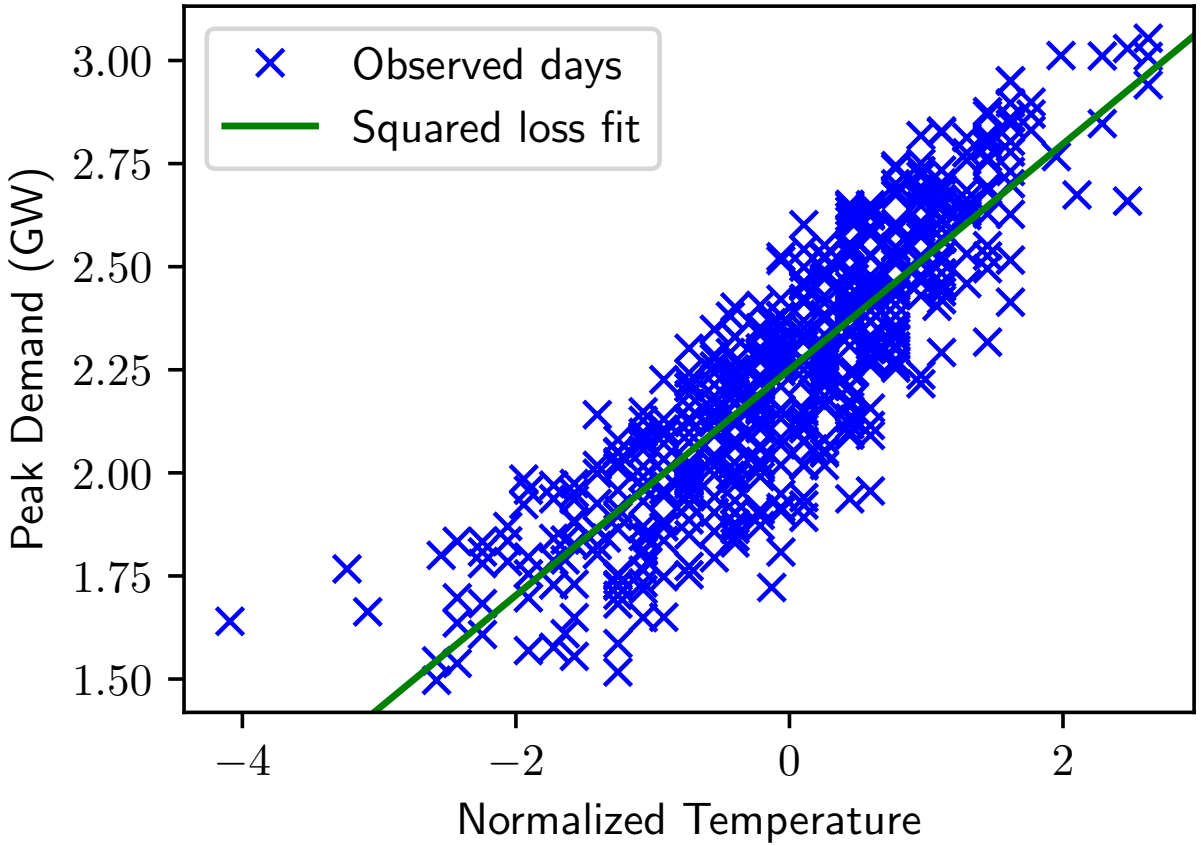
$\theta = (0.25, 2.05)$
$E(\theta) = 18.58$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-13.60, -111.77)$

Gradient descent – Iteration 5



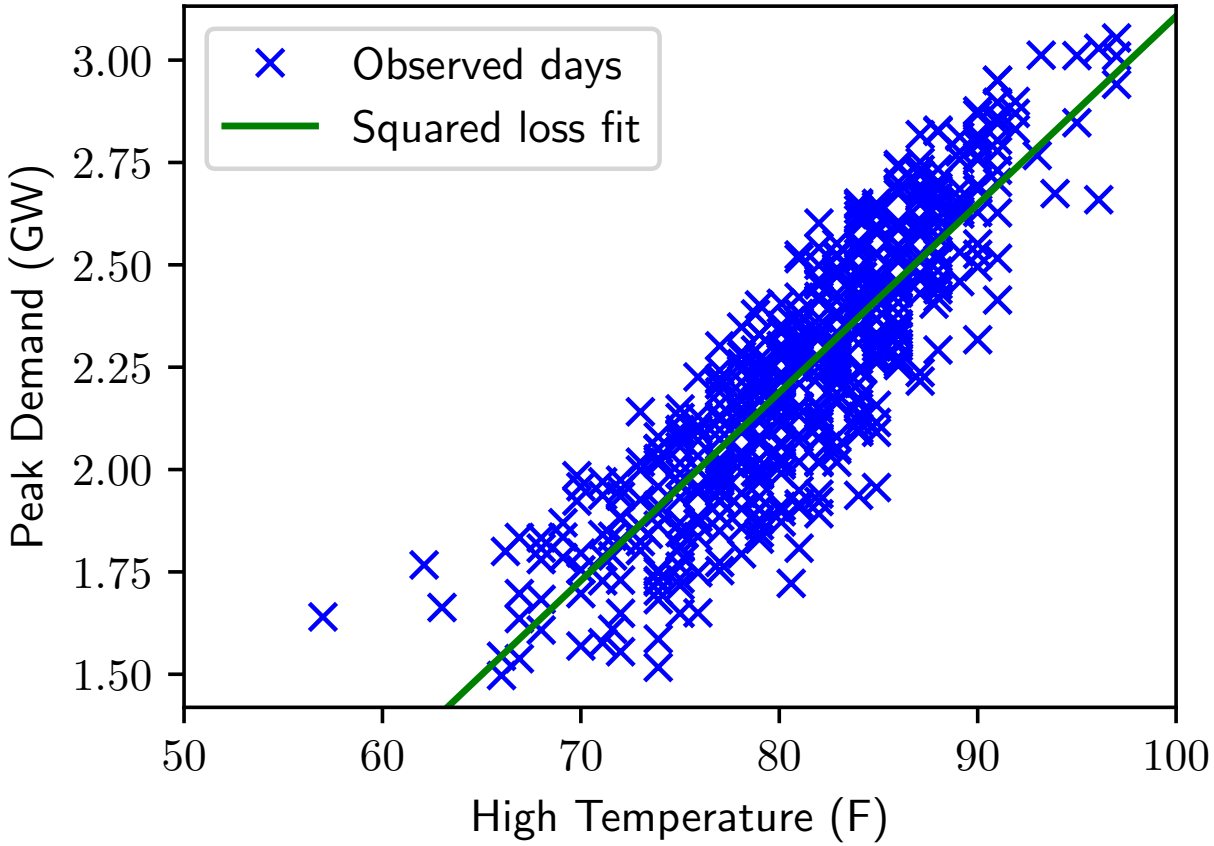
$\theta = (0.26, 2.16)$
$E(\theta) = 9.40$
$(\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}) = (-6.09, -50.07)$

Gradient descent – Iteration 10



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

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"

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		**Machine learning algorithm**		**Predictions**
$(x^{(1)}, y^{(1)})$ $(x^{(2)}, y^{(2)})$ $(x^{(3)}, y^{(3)})$ $\vdots$	$\longrightarrow$	Hypothesis function $y^{(i)} \approx h(x^{(i)})$	$\longrightarrow$	New example x $\hat{y} = h(x)$

Terminology

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

$$\text{E.g.: } x^{(i)} = \begin{bmatrix} \text{High_Temperature}^{(i)} \\ \text{Is_Weekday}^{(i)} \\ 1 \end{bmatrix}$$

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

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

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

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

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

Terminology

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

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

The canonical machine learning optimization problem:

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

Virtually every machine learning algorithm has this form, just specify

- What is the hypothesis function?
- What is the loss function?
- 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: $\ell(\hat{y}, y) = (\hat{y} - y)^2$
- Resulting machine learning optimization problem:

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

Derivative of the least squares objective

Compute the partial derivative with respect to an arbitrary model parameter θ_j

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

Gradient descent algorithm

1. Initialize $\theta_k := 0,\ k = 1, \dots, n$

2. Repeat:
   - For $k = 1, \dots, n$:

$$\theta_k := \theta_k - \alpha \sum_{i=1}^{m} 2 \left(\sum_{j=1}^{n} \theta_j \cdot x_j^{(i)} - y^{(i)} \right) x_k^{(i)}$$

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

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

Gradient in vector notation

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

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

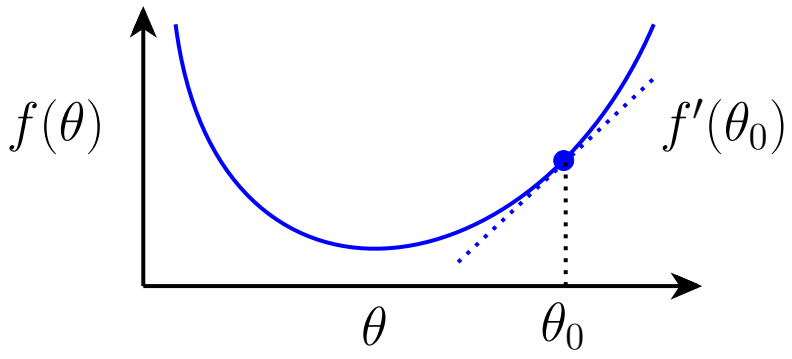
$$\iff \nabla_\theta E(\theta) = 2 \sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right)$$

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

$$2\sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)} \right) = 0$$

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

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

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

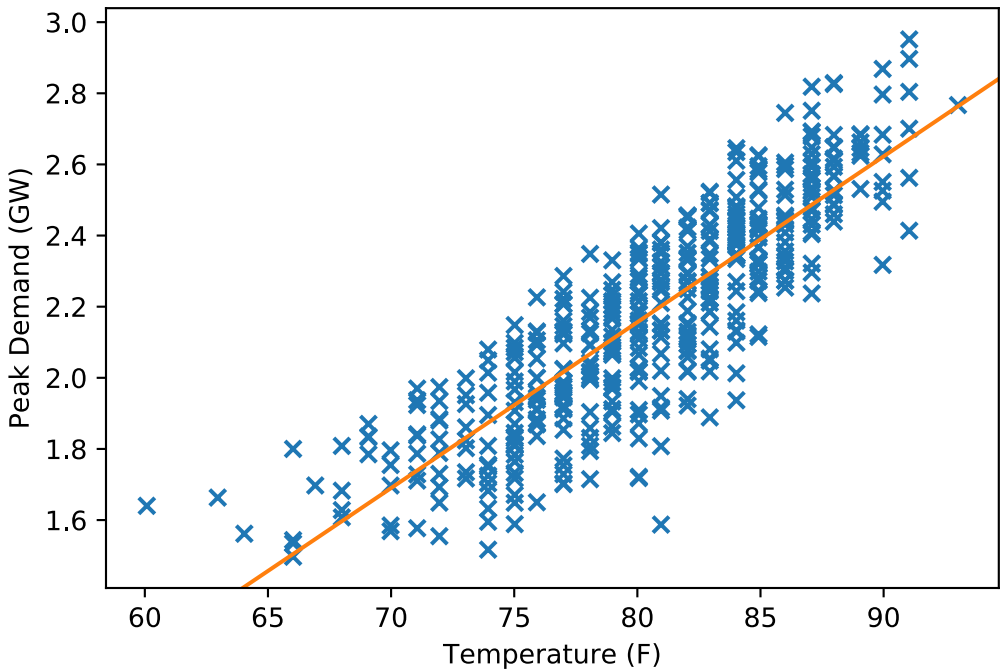
$$\nabla_\theta E(\theta) = 2\sum_{i=1}^{m} x^{(i)} \left({x^{(i)}}^T \theta - y^{(i)}\right) = 2X^T(X\theta - y)$$
$$\implies \theta^\star = (X^T X)^{-1} X^T y$$

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:

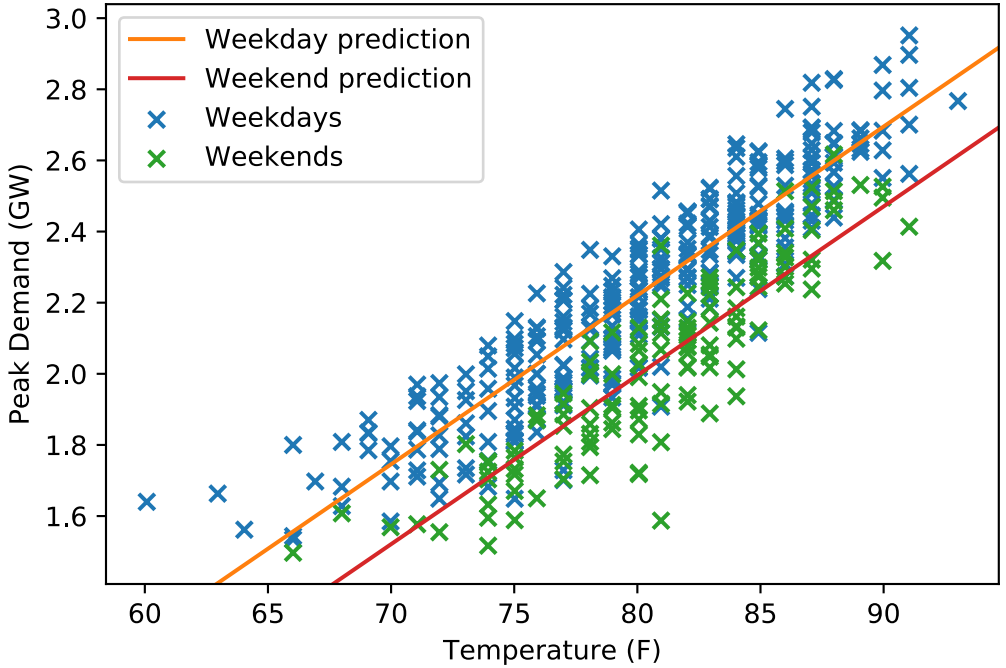
$$x^{(i)} = \begin{bmatrix} \text{High_Temperature}^{(i)} \\ 1 \end{bmatrix}, \qquad \theta^\star = (X^T X)^{-1} X^T y = \begin{bmatrix} 0.046 \\ -1.574 \end{bmatrix}$$



Example: electricity demand

Returning to our electricity demand example:

$$x^{(i)} = \begin{bmatrix} \text{High_Temperature}^{(i)} \\ \text{Is_Weekday}^{(i)} \\ 1 \end{bmatrix}, \qquad \theta^\star = (X^T X)^{-1} X^T y = \begin{bmatrix} 0.047 \\ 0.225 \\ -1.803 \end{bmatrix}$$



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

```
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
```

15-388/688 - Practical Data Science: Big data and MapReduce

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Big data

Some context in distributed computing

map + reduce

MapReduce

MapReduce in Python (very briefly)

Outline

Big data

Some context in distributed computing

map + reduce

MapReduce

MapReduce in Python (very briefly)

“Big data”

My laptop
8GB RAM
500GB Disk

Big data?
No

?

Google Data Center
??? RAM/Disk
(>> PBs)

Big data?
Yes

Some notable inflection points

1. Your data fits in RAM on a single machine
2. Your data fits on disk on a single machine
3. Your data fits in RAM/disk on a “small” cluster of machines (you don’t need to worry about machines dying)
4. Your data fits in RAM/disk on a “large” cluster of machine (you need to worry about machines dying)

Probably reasonable to refer to 3+ as “big data”, but many would only consider 4

Do you have big data?

If your data fits on a single machine (even on disk), then it's almost always better to think about how you can design an efficient single-machine solution, unless you have extremely good reasons for doing otherwise

scalable system	cores	twitter	uk-2007-05
GraphChi [10]	2	3160s	6972s
Stratosphere [6]	16	2250s	-
X-Stream [17]	16	1488s	-
Spark [8]	128	857s	1759s
Giraph [8]	128	596s	1235s
GraphLab [8]	128	249s	833s
GraphX [8]	128	419s	462s
Single thread (SSD)	1	300s	651s
Single thread (RAM)	1	275s	-

Table 2: Reported elapsed times for 20 PageRank iterations, compared with measured times for single-threaded implementations from SSD and from RAM. GraphChi and X-Stream report times for 5 PageRank iterations, which we multiplied by four.

scalable system	cores	twitter	uk-2007-05
Stratosphere [6]	16	950s	-
X-Stream [17]	16	1159s	-
Spark [8]	128	1784s	$\geq$ 8000s
Giraph [8]	128	200s	$\geq$ 8000s
GraphLab [8]	128	242s	714s
GraphX [8]	128	251s	800s
Single thread (SSD)	1	153s	417s

Table 3: Reported elapsed times for label propagation, compared with measured times for single-threaded label propagation from SSD.

Tables from [McSherry et al., 2015 "Scalability! But at what COST"]

Outline

Big data

Some context in distributed computing

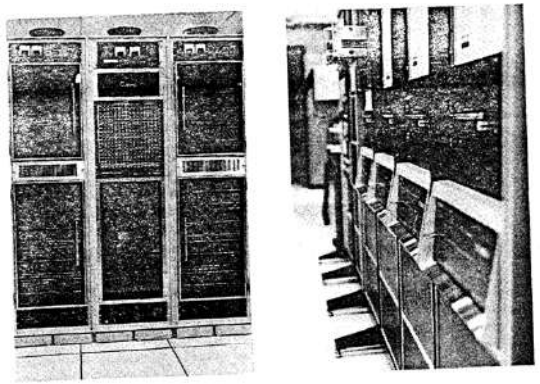
map + reduce

MapReduce

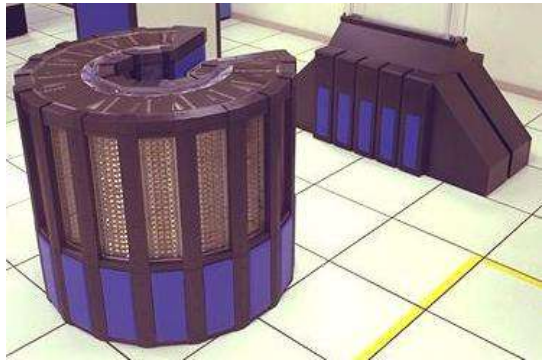
MapReduce in Python (very briefly)

Distributed computing

Distributed computing rose to prominence in the 70s/80s, often built around "supercomputing," for scientific computing applications



1971 – CMU C.mmp
(16 PDP-11 processors)



1984 – Cray-2
(4 vector processors)

Message passing interface

In mid-90s, researchers built a common interface for distributed computing called the message passing interface (MPI)

MPI provided a set of tools to run multiple processes (on a single machine or across many machines), that could communicate, send data between each other (all of “scattering”, “gathering”, “broadcasting”), and synchronize execution

Still common in scientific computing applications and HPC (high performance computing)

Downsides to MPI

MPI is extremely powerful but has some notable limitations

1. MPI is complicated: programs need to explicitly manage data, synchronize threads, etc

2. MPI is brittle: if machines die suddenly, can be difficult to recover (unless explicitly handled by the program, making them more complicated)

A new paradigm for data processing

When Google was building their first data centers, they used clusters of off-the-shelf commodity hardware; machines had different speeds and failures were common given cluster sizes

Data itself was distributed (redundantly) over many machines, as much as possible wanted to do the computation on the machine where the data is stored

Led to the development of the MapReduce framework at Google [Dean and Ghemawat, 2004], later made extremely popular through the Apache Hadoop open source implementation, now more dominantly through Apache Spark

MapReduce

A simple paradigm for distributed computation where users write just two functions: a *mapper* and a *reducer*

Work can be automatically farmed out to a large collection of machines

As much as possible, computation is done on the machine where the data lives

Node failures or “stragglers” (nodes that are slow for some reason) are automatically handled

Big data since MapReduce

MapReduce is a wonderful, but many disadvantages (discussed shortly)

Since ~2010s, big data community has been “slowly” trying to re-integrate some of the ideas from the HPC community

Aside: GPUs are really the natural descendants of the HPC line of work, which are doing pretty well in data science these days…

Remember:

Speed(network) < Speed(disk) < Speed(RAM) < Speed(Cache)

(use the fastest data storage mechanism possible)

Outline

Big data

Some context in distributed computing

map + reduce

MapReduce

MapReduce in Python (very briefly)

Primer: map and reduce functions

We can get some intuition on MapReduce by inspiration from the map and reduce functions in Python (but MapReduce $\neq$ map + reduce)

The map call takes a function and a list (iterable) and generates a new list of the function applied to each element:

```
map(f, [a, b, c, ...]) -> [f(a), f(b), f(c), ...]
```

The reduce call takes a function and a list (iterable) and iteratively applies the function to two elements (next item in the list and result of previous function)

```
reduce(g, [a, b, c, ...]) -> g(g(g(a,b),c), ...)
```

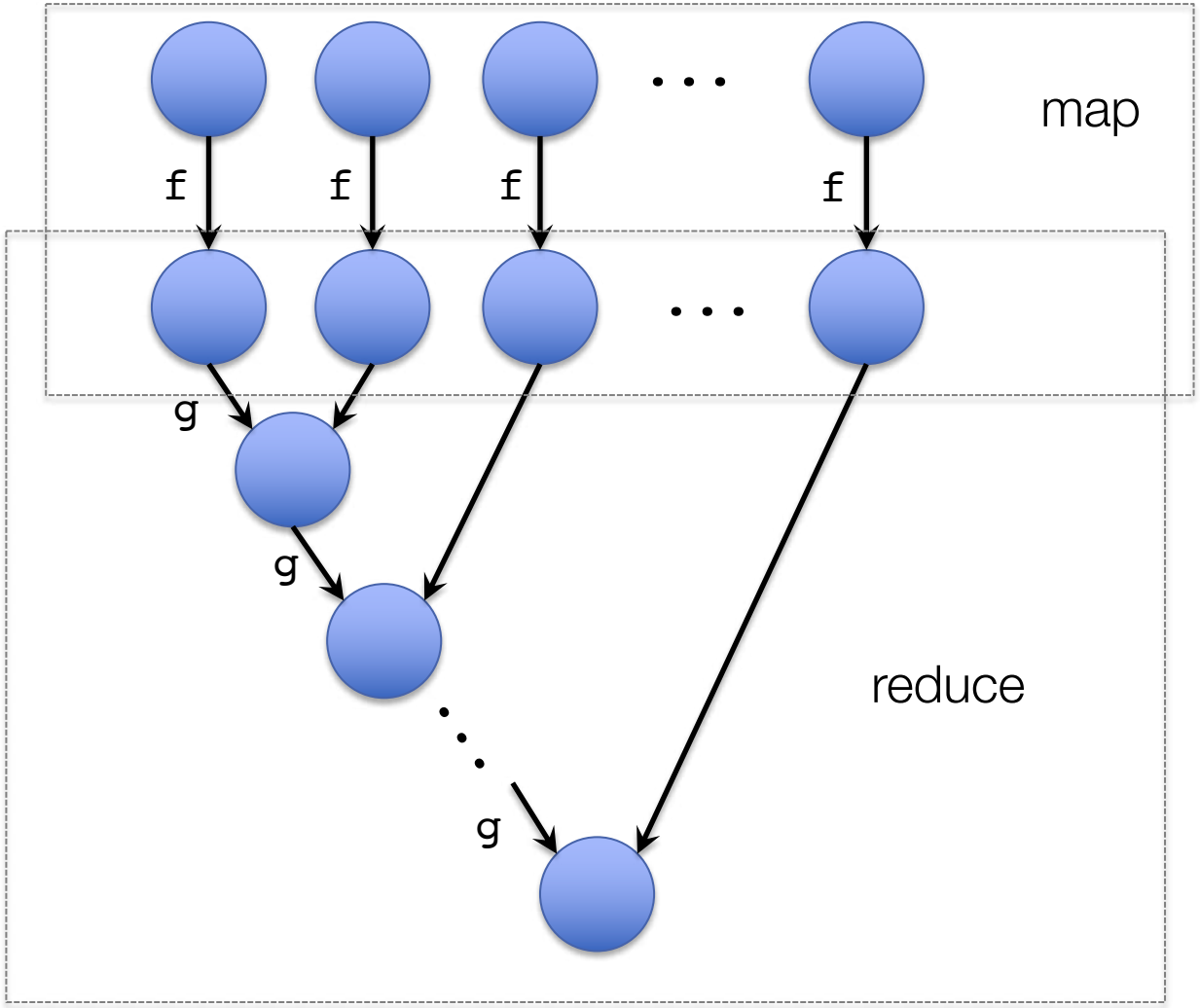
Example: Sum of squared elements

We could take a list, square each element, and add these squared terms together using the following code

```
from functools import reduce

data = [1,2,3,4]
values = map(lambda x : x*x, data)
# values = [1, 4, 9, 16]
output = reduce(lambda x,y: x+y, values)
# output = 30
```

Map and reduce graphically



Mappers, reducers, and execution engines

We'll specifically refer to the **mapper** function, the **reducer** function and the **execution engine** (the supporting code that actually calls the map and reduce functions)

```
def map_reduce_execute(data, mapper, reducer):
    values = map(mapper, data)
    output = reduce(reducer, values)
    return output

def mapper_square(x):
    return x**2

def reducer_sum(x,y):
    return x+y

map_reduce_execute([1,2,3,4], mapper_square, reducer_sum)
```

Abstracting map + reduce

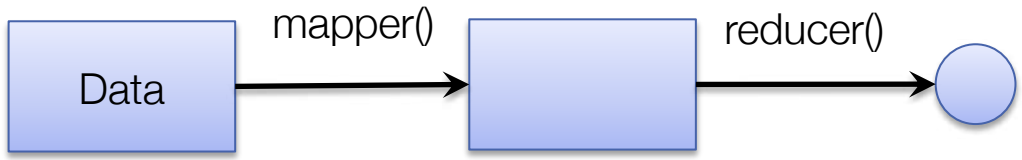
Key point: to use this framework, the programmer only needs to implement the mapper and reducer function, and the execution engine can use whatever method it wants to actually compute the result

For instance, the application of the mapper functions is inherently parallel, can be carried out in separate threads/machines

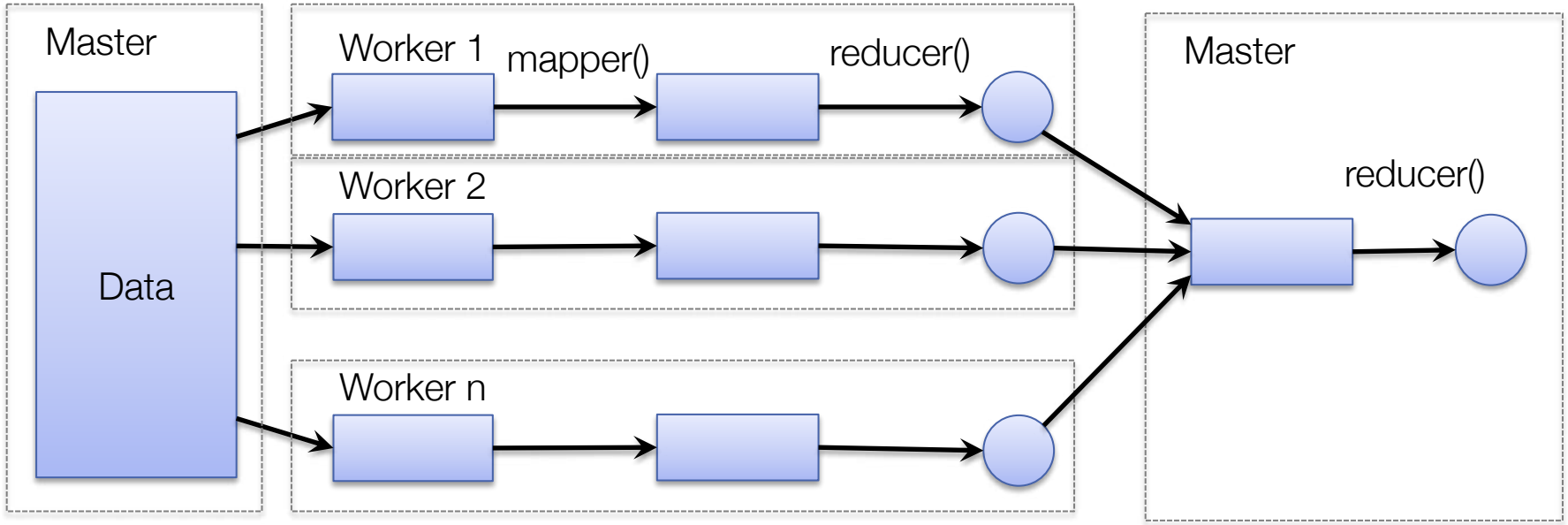
In many cases, the reduce step can also be carried out incrementally

Distributed map + reduce

Single machine execution engine:



Distributed execution engine:



Poll: properties for distributed map + reduce

Suppose we run a distributed map + reduce execution engine as in the previous slide, which distributes data randomly to each worker before applying the mapper, reducer, collecting the data, and applying the reducer again.

Which properties of the mapper f and reducer g are required to ensure the results are deterministic?

1. g is *commutative* $g(a, b) = g(b, a)$

2. g is associative $g(g(a, b), c) = g\big(a, g(b, c)\big)$

3. f and g are distributive $f\big(g(a, b)\big) = g(f(a), f(b))$

Outline

Big data

Some context in distributed computing

map + reduce

MapReduce

MapReduce in Python (very briefly)

MapReduce ($\neq$ map + reduce)

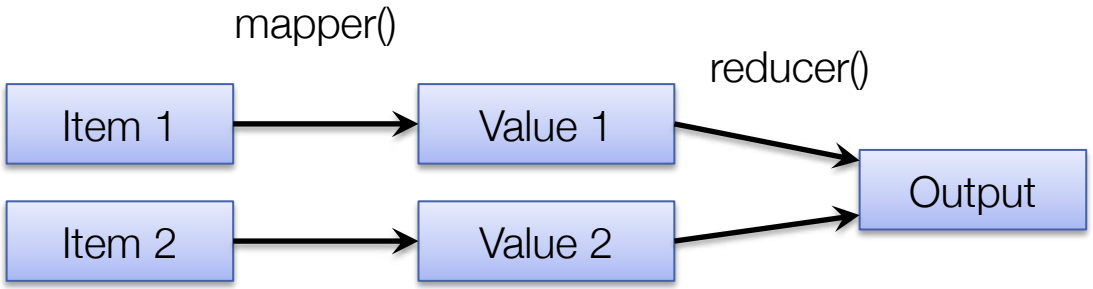
You can think of MapReduce as map + reduce **“by key”**

Mapper function doesn’t just return a single value, but a *list* of key-value pairs (with potentially multiple instances of the same key)

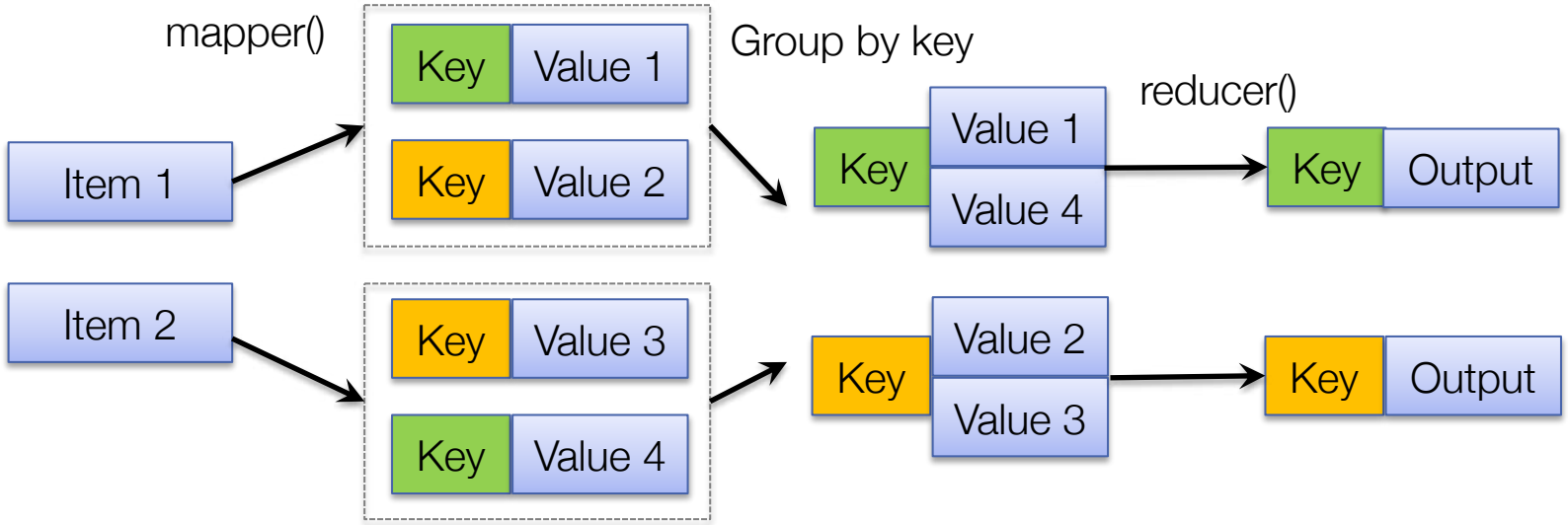
Before calling the reducer, the execution engine groups all results by key

map + reduce vs. MapReduce

map + reduce



MapReduce



Example: word count

the wheels on the bus
go round and round
round and round
round and round
the wheels on the bus
go round and round
all through the town

mapper()

[(the,1) (wheels,1) (on,1) (the,1) (bus,1)]
[(go,1) (round,1) (and,1) (round,1)]
[(round,1) (and,1) (round,1)]
[(round,1) (and,1) (round,1)]
[(the,1) (wheels,1) (on,1) (the,1) (bus,1)]
[(go,1) (round,1) (and,1) (round,1)]
[(all,1) (through,1) (the,1) (town,1)]

group by key

(and, [1,1,1,1])
(on, [1,1])
(all, [1]),
(bus, [1,1]),
(round, [1,1,1,1,1,1,1,1]),
(town, [1]),
(through, [1]),
(go, [1, 1]),
(the, [1, 1, 1, 1, 1]),
(wheels, [1,1])

reducer()

(and, 4)
(on, 2)
(all, 1),
(bus, 2),
(round, 8),
(town, 1),
(through, 1),
(go, 2),
(the, 5),
(wheels, 2)

MapReduce execution engine

A simple MapReduce execution engine (no parallelism, so not particularly useful), can be written as follows

```
def mapreduce_execute(data, mapper, reducer):
    values = map(mapper, data)

    groups = {}
    for items in values:
        for k,v in items:
            if k not in groups:
                groups[k] = [v]
            else:
                groups[k].append(v)
    output = [reducer(k,v) for k,v in groups.items()]
    return output
```

MapReduce word occurrence count example

In this engine, we can run our word occurrence counter by specifying the following mapper and reducer

```
def mapper_word_occurrence(line):
    return [(word, 1) for word in line.split(" ")]

def reducer_sum(key, val):
    return (key, sum(val))

lines = ["the wheels on the bus",
         "go round and round",
         "round and round",
         "round and round",
         "the wheels on the bus",
         "go round and round",
         "all through the town"]

mapreduce_execute(lines, mapper_word_occurrence, reducer_sum)
```

More advanced usage

In original paper, and most implementations, input data is *also* in key/value form, so the mapper also is provided with a key value pair

Many real applications require chaining together multiple map/reduce steps

“Combiners” are local reducers that run after each map to potentially reduce network overhead

Optional ability for functions to all share some additional context (i.e., shared read-only memory between multiple mappers / reducers)

Advantages of MapReduce

MapReduce isn't popular because of what it can do, it's popular because of what it can't do (i.e., what you don't need to do)

End user just needs to implement two functions: mapper and reducer

No exposure of interprocess communication, data splitting, data locality, redundancy mechanisms (can all be handled by underlying system)

Disadvantages of MapReduce

Can be extremely slow: in traditional MapReduce, resilience is attained by reading/writing data from/to disk between each stage of processing

Sometimes you really do want communication between processes

Distributed data systems beyond MapReduce: Spark, GraphLab, parameter servers, many others

All of them will frequently be slower than a single machine, if your data fits on the disk of a single machine

Outline

Big data

Some context in distributed computing

map + reduce

MapReduce

MapReduce in Python (very briefly)

Practical MapReduce

(Obviously) you don't want to write you own MapReduce execution engine, use one of the many engines available

Python `mrjob` library: write simple mappers/reducers in Python, and execute on Hadoop systems, Amazon Elastic MapReduce, Google Cloud

Word occurrence count example:

```
from mrjob.job import MRJob

class WordOccurrenceCount(MRJob):
    def mapper(self, _, line):
        for word in line.split(" "):
            yield word, 1

    def reducer(self, key, values):
        yield key, sum(values)
```

15-388/688 - Practical Data Science: Nonlinear modeling, cross-validation, and regularization

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Announcements

Tutorial “proposal” sentence due tonight

I will send feedback on topics by next week, you *may* change topics after feedback, but don’t submit with the intention of doing this

Piazza note about linear regression in HW 3

TA Office Hours calendar posted to course webpage, under “Instructors”

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

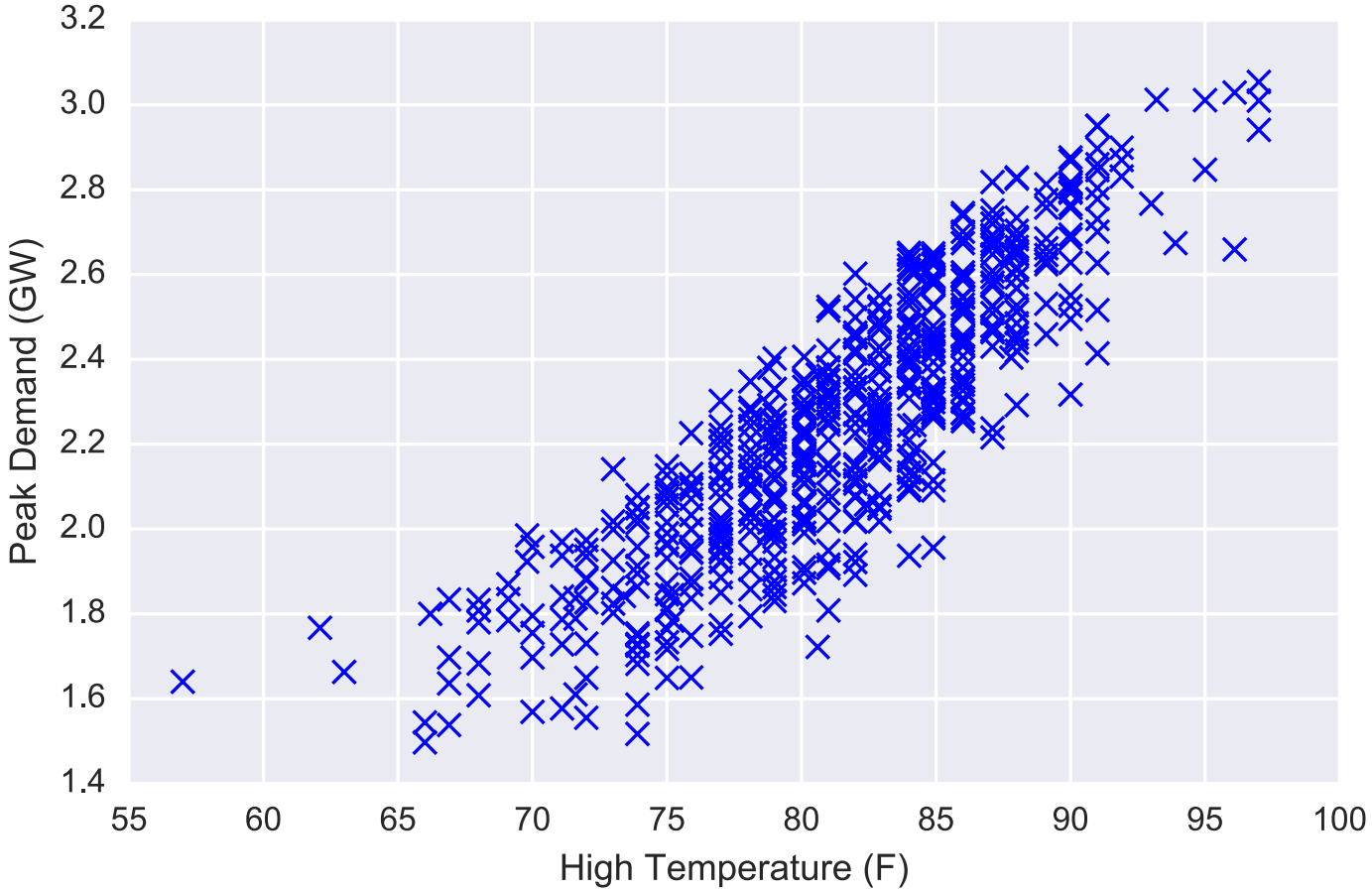
Regularization

General nonlinear features

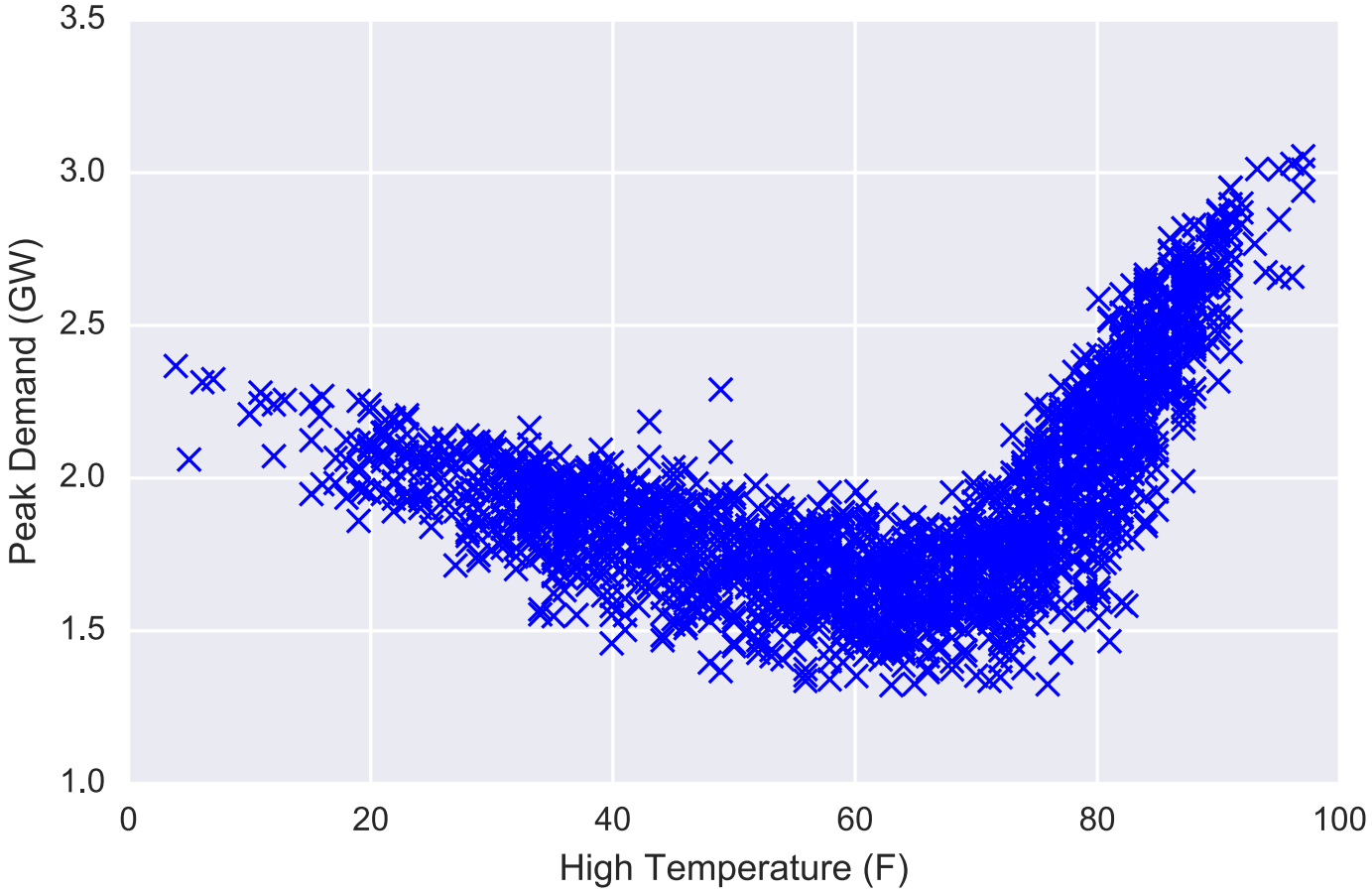
Kernels

Nonlinear classification

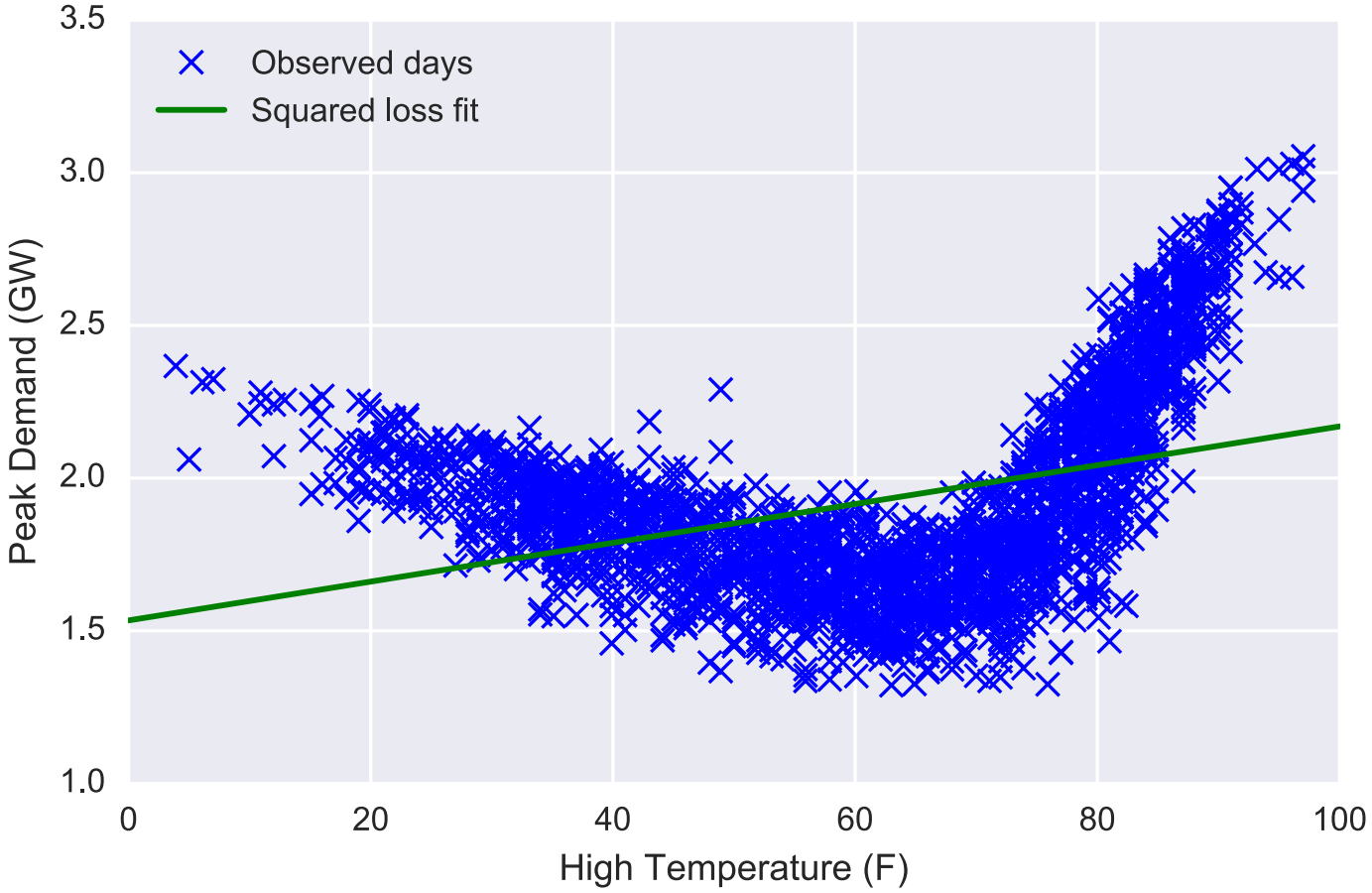
Peak demand vs. temperature (summer months)



Peak demand vs. temperature (all months)



Linear regression fit



“Non-linear” regression

Thus far, we have illustrated linear regression as “drawing a line through through the data”, but this was really a function of our input features

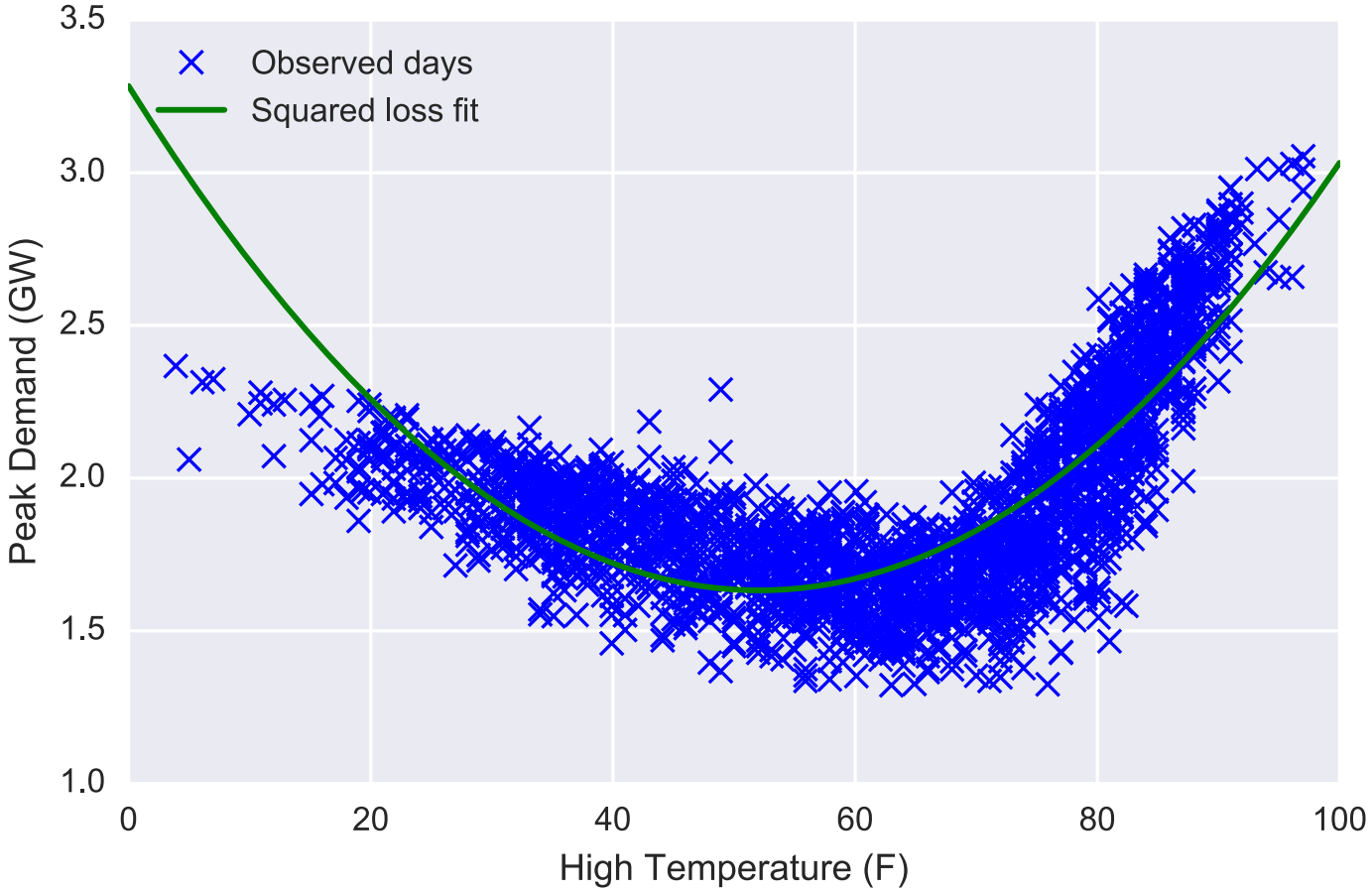
Though it may seem limited, linear regression algorithms are quite powerful when applied to *non-linear features* of the input data, e.g.

$$x^{(i)} = \begin{bmatrix} (\text{High_Temperature}^{(i)})^2 \\ \text{High_Temperature}^{(i)} \\ 1 \end{bmatrix}$$

Same hypothesis class as before $h_\theta(x) = \theta^T x$, but now prediction will be a non-linear function of base input (e.g. a quadratic function)

Same least-squares solution $\theta = (X^T X)^{-1} X^T y$

Polynomial features of degree 2



Code for fitting polynomial

The only element we need to add to write this non-linear regression is the creation of the non-linear features

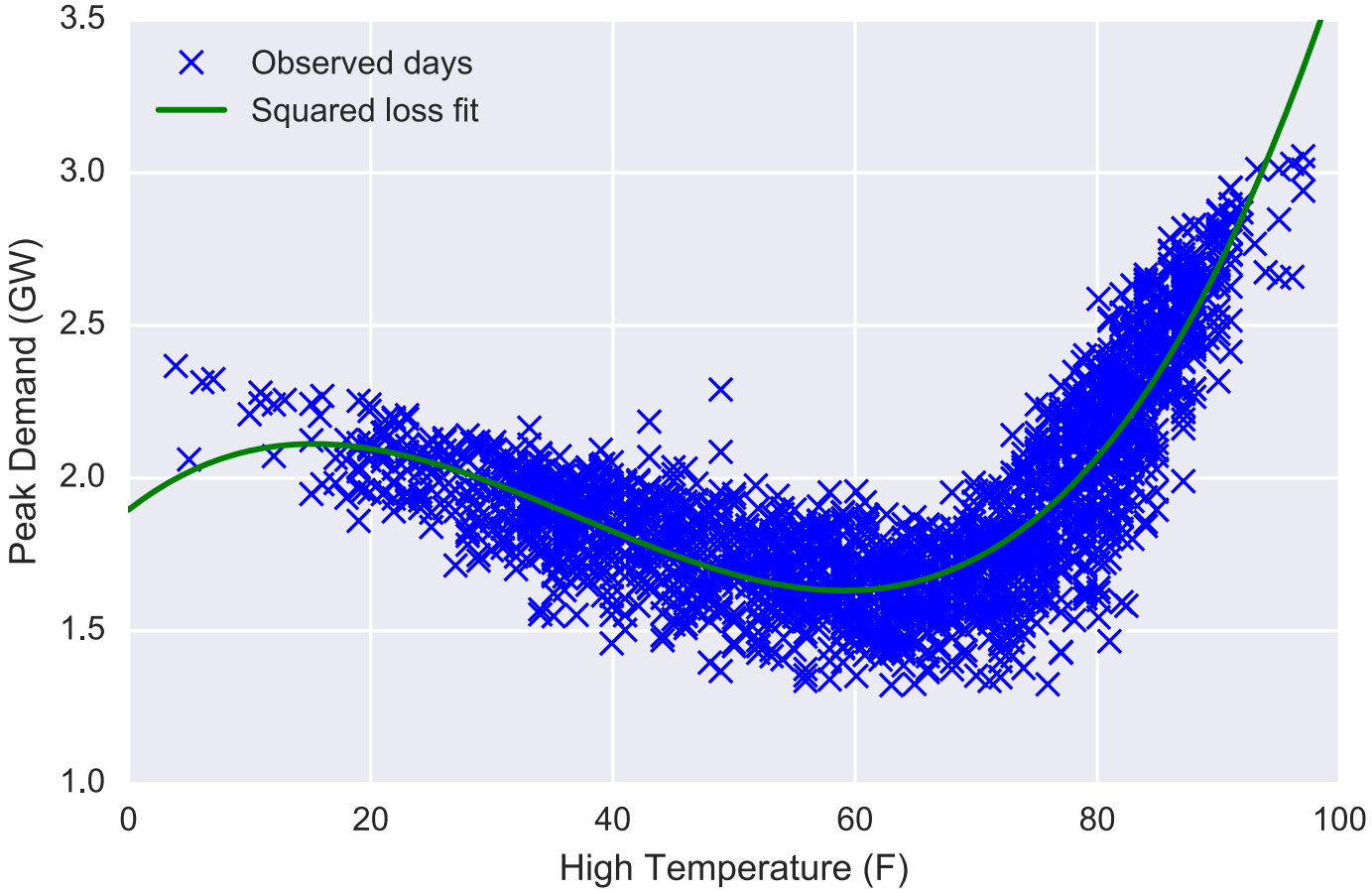
```
x = df_daily.loc[:,"Temperature"]
min_x, rng_x = (np.min(x), np.max(x) - np.min(x))
x = 2*(x - min_x)/rng_x - 1.0
y = df_daily.loc[:,"Load"]

X = np.vstack([x**i for i in range(poly_degree,-1,-1)]).T
theta = np.linalg.solve(X.T.dot(X), X.T.dot(y))
```

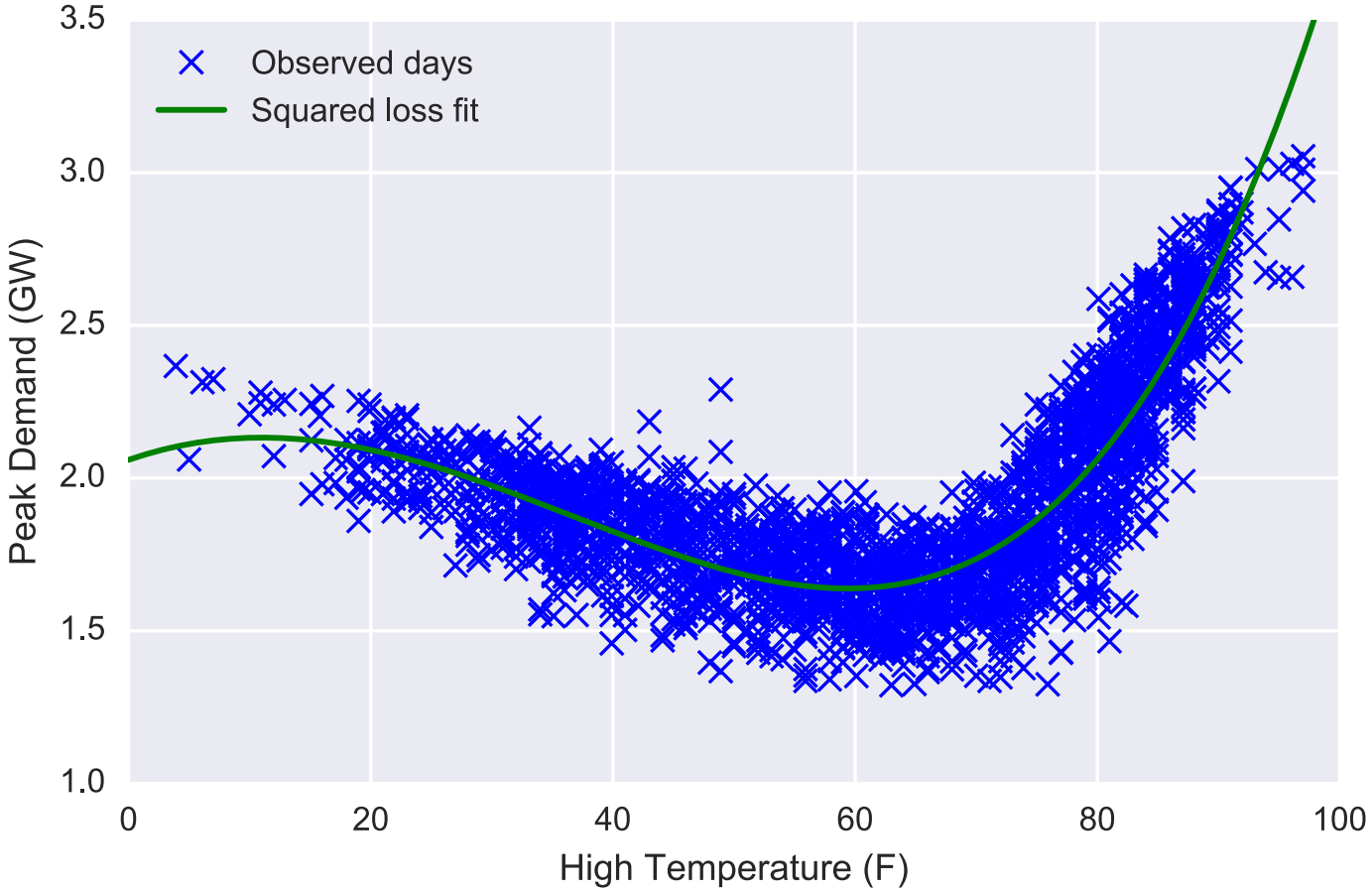
Output lea

```
x0 = 2*(np.linspace(xlim[0], xlim[1],1000) - min_x)/rng_x - 1.0
X0 = np.vstack([x0**i for i in range(poly_degree,-1,-1)]).T
y0 = X0.dot(theta)
```

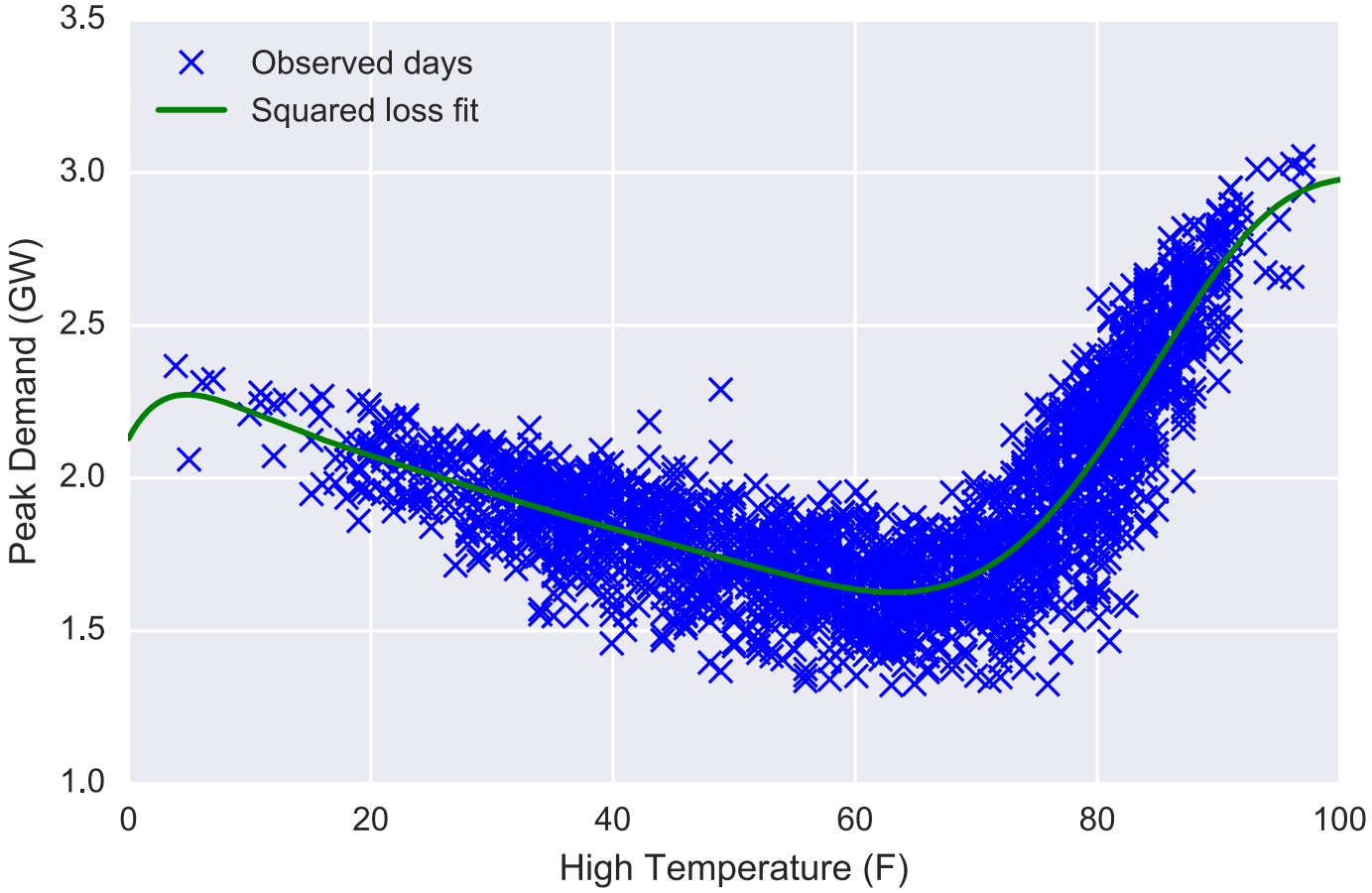
Polynomial features of degree 3



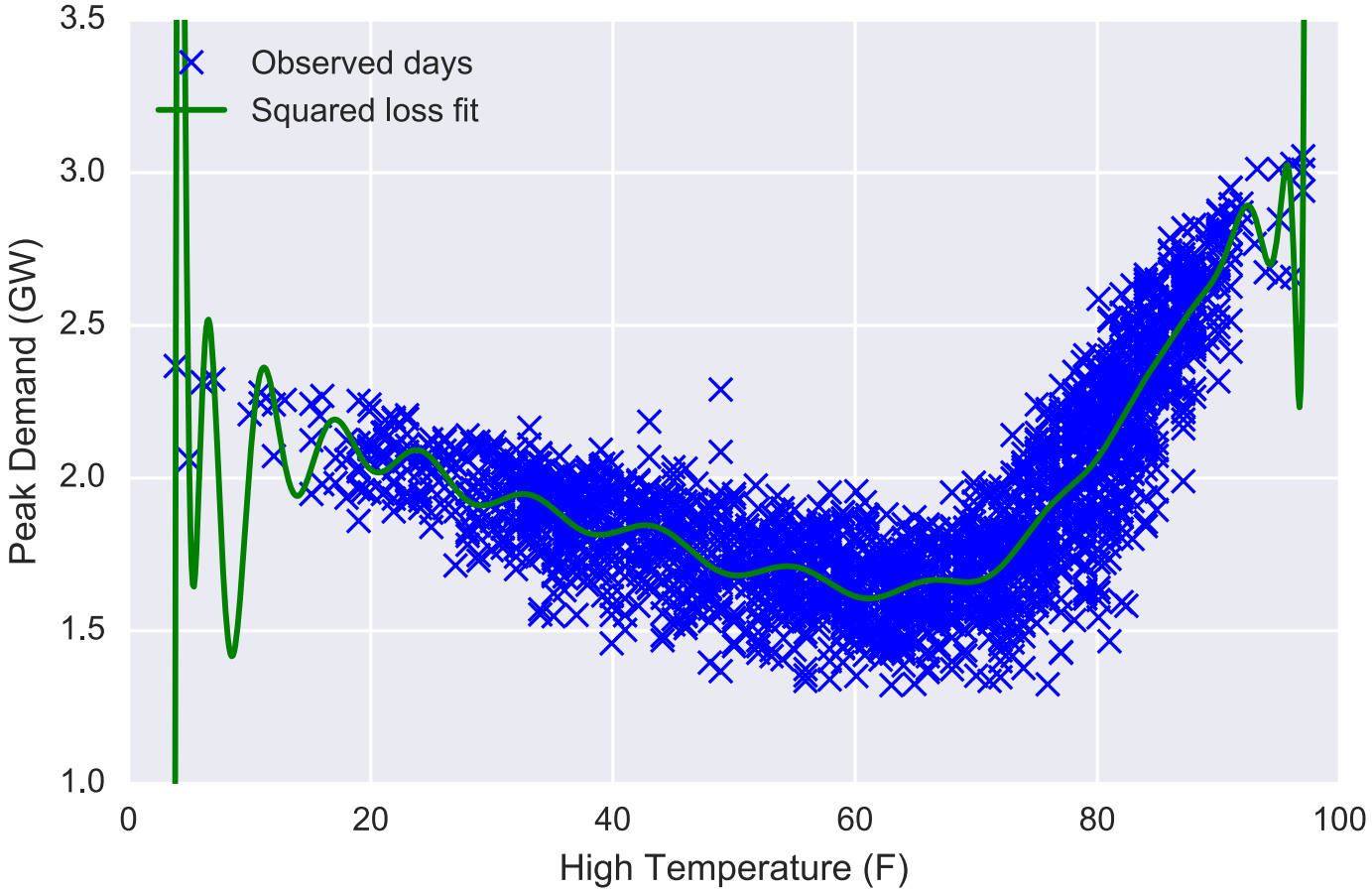
Polynomial features of degree 4



Polynomial features of degree 10



Polynomial features of degree 50



Linear regression with many features

Suppose we have m examples in our data set and $n = m$ features (plus assumption that features are linearly independent, though we'll always assume this)

Then $X \in \mathbb{R}^{m \times n}$ is a square matrix, and least squares solution is:

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

and we therefore have $X\theta = y$ (i.e., we fit data exactly)

Note that we can *only* perform the above operations when X is square, though if we have *more* features than examples, we can still get an exact fit by simply discarding features

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Generalization error

The problem we the canonical machine learning problem is that we don't *really* care about minimizing this objective on the given data set

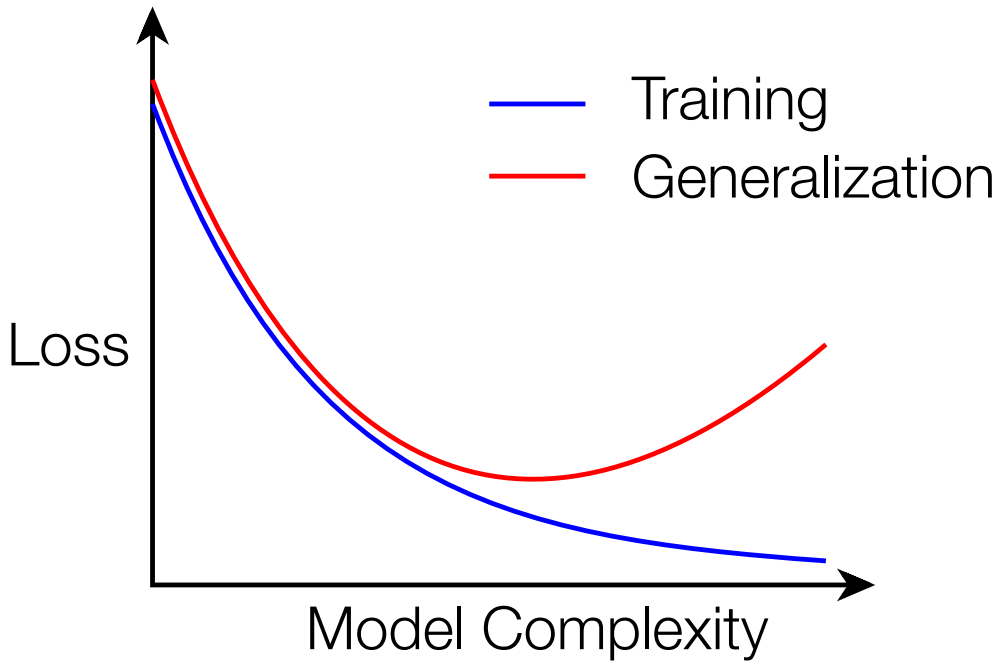
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$

What we really care about is how well our function will generalize to *new examples* that we *didn't* use to train the system (but which are drawn from the "same distribution" as the examples we used for training)

The higher degree polynomials exhibited *overfitting*: they actually have very *low* loss on the training data, but create functions we don't expect to generalize well

Cartoon version of overfitting

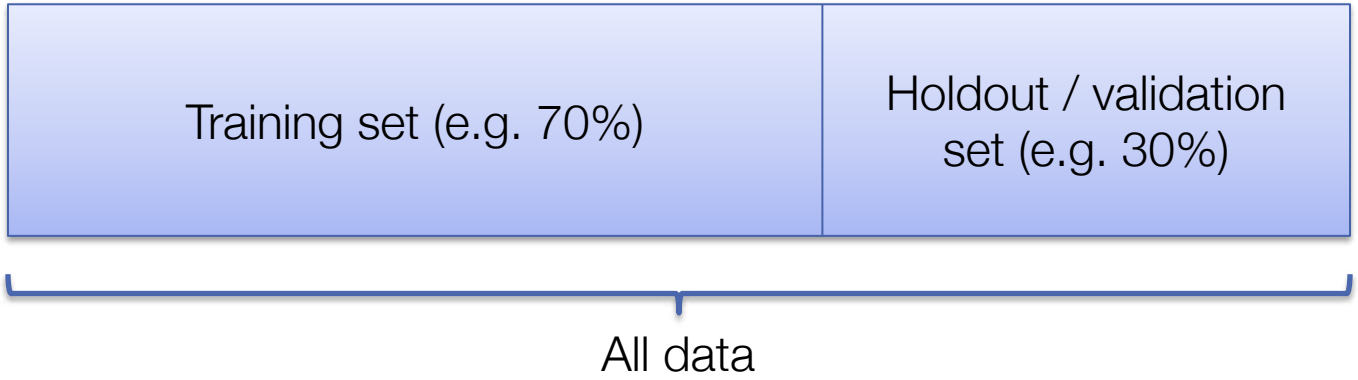
As model becomes more complex, training loss always decreases; generalization loss decreases to a point, then starts to increase



Cross-validation

Although it is difficult to quantify the true generalization error (i.e., the error of these algorithms over the *complete* distribution of possible examples), we can approximate it by ***holdout cross-validation***

Basic idea is to split the data set into a training set and a holdout set



Train the algorithm on the training set and evaluate on the holdout set

Cross-validation in code

A simple example of holdout cross-validation:

```
# compute a random split of the data
np.random.seed(0)
perm = np.random.permutation(len(df_daily))
idx_train = perm[:int(len(perm)*0.7)]
idx_cv = perm[int(len(perm)*0.7):]

# scale features for each split based upon training
xt = df_daily.iloc[idx_train,0]
min_xt, rng_xt = (np.min(xt), np.max(xt) - np.min(xt))
xt = 2*(xt - min_xt)/rng_xt - 1.0
xcv = 2*(df_daily.iloc[idx_cv,0] - min_xt)/rng_xt -1
yt = df_daily.iloc[idx_train,1]
ycv = df_daily.iloc[idx_cv,1]

# compute least squares solution and error on holdout and training
X = np.vstack([xt**i for i in range(poly_degree,-1,-1)]).T
theta = np.linalg.solve(X.T.dot(X), X.T.dot(yt))
err_train = 0.5*np.linalg.norm(X.dot(theta) - yt)**2/len(idx_train)
err_cv = 0.5*np.linalg.norm(Xcv.dot(theta) - ycv)**2/len(idx_cv)
```

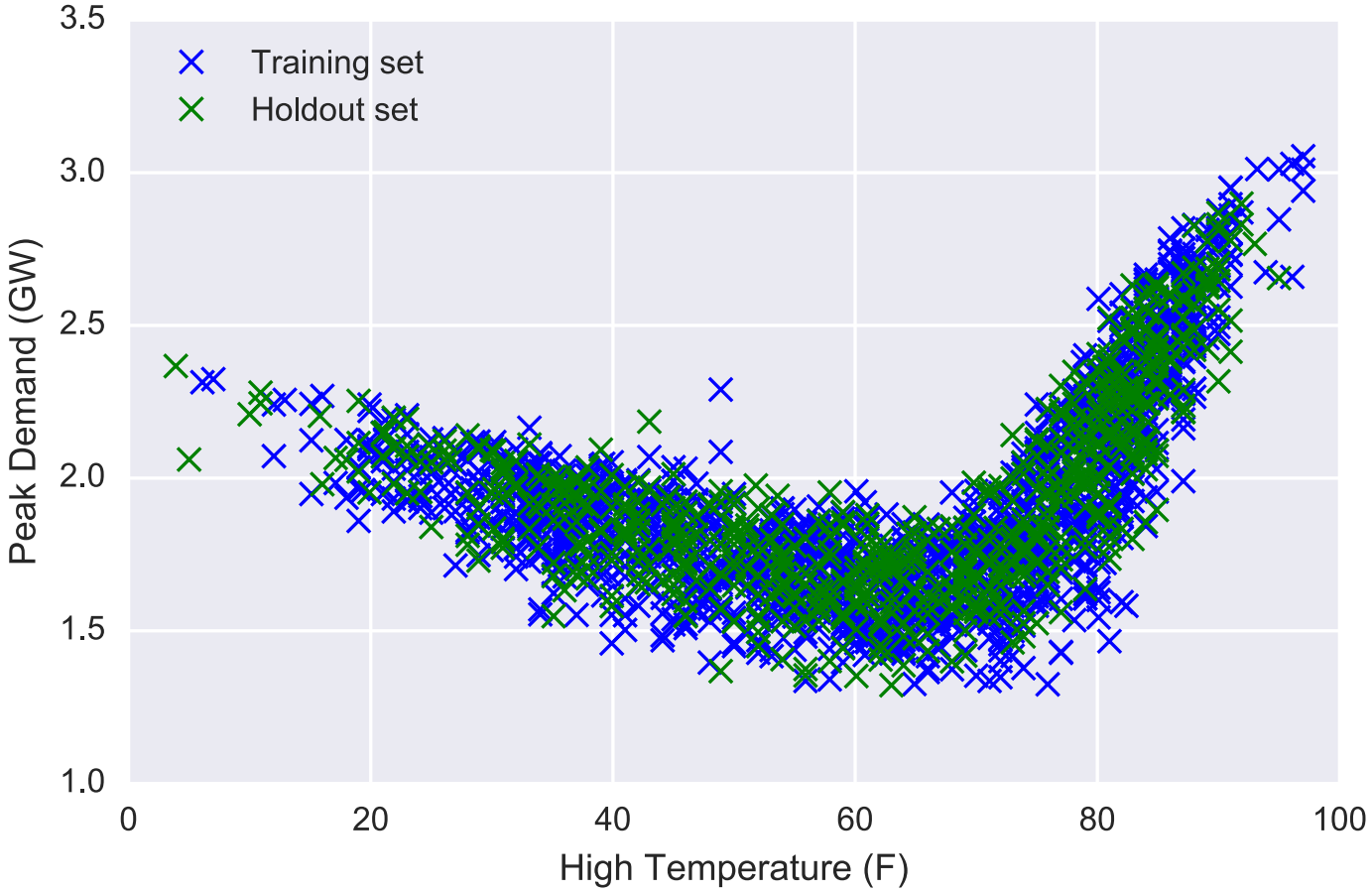
Parameters and hyperparameters

We refer to the θ variables as the *parameters* of the machine learning algorithm

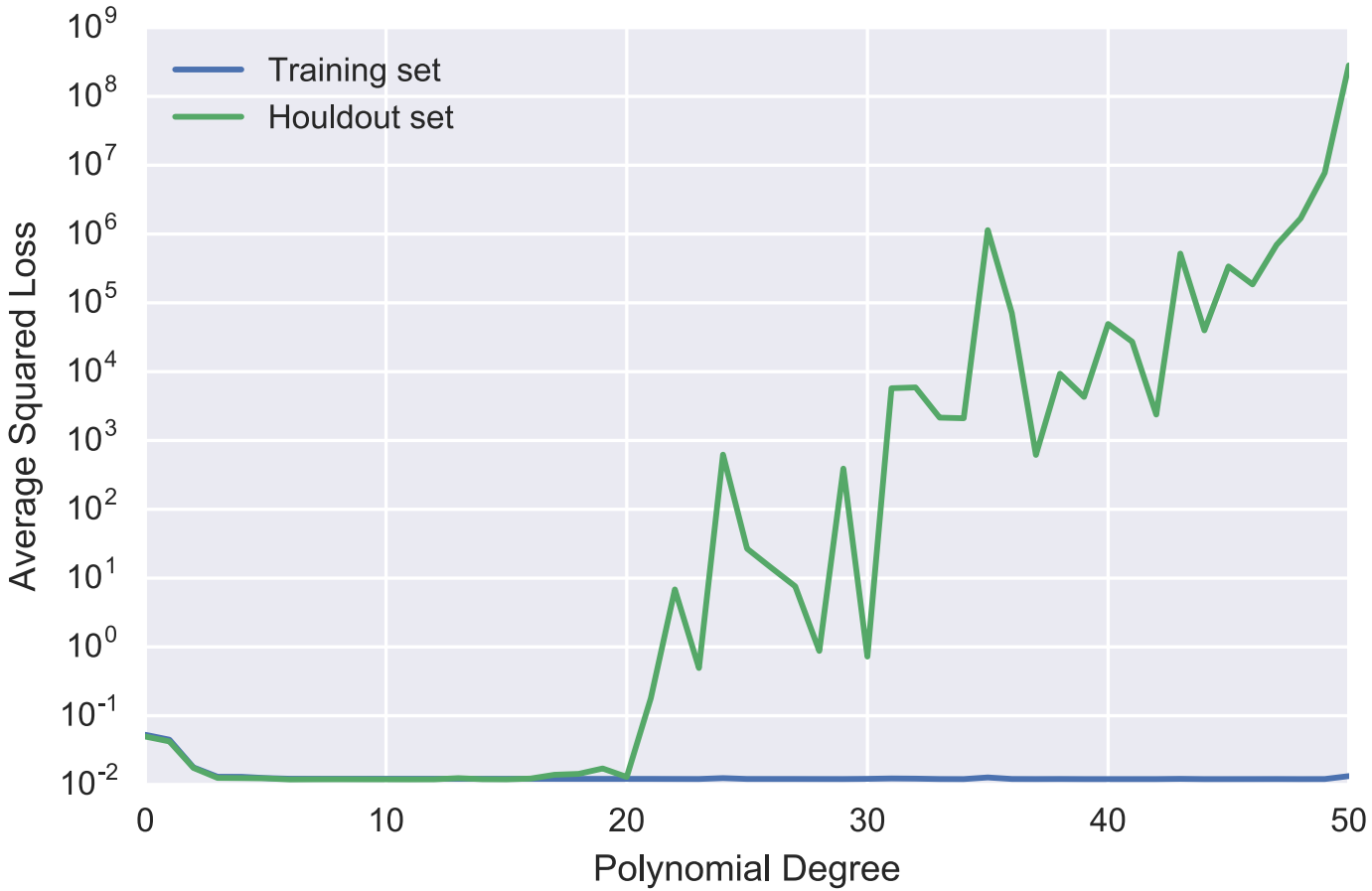
But there are other quantities that also affect the classifier: degree of polynomial, amount of regularization, etc; these are collectively referred to as the *hyperparameters* of the algorithm

Basic idea of cross-validation: use training set to determine the parameters, use holdout set to determine the hyperparameters

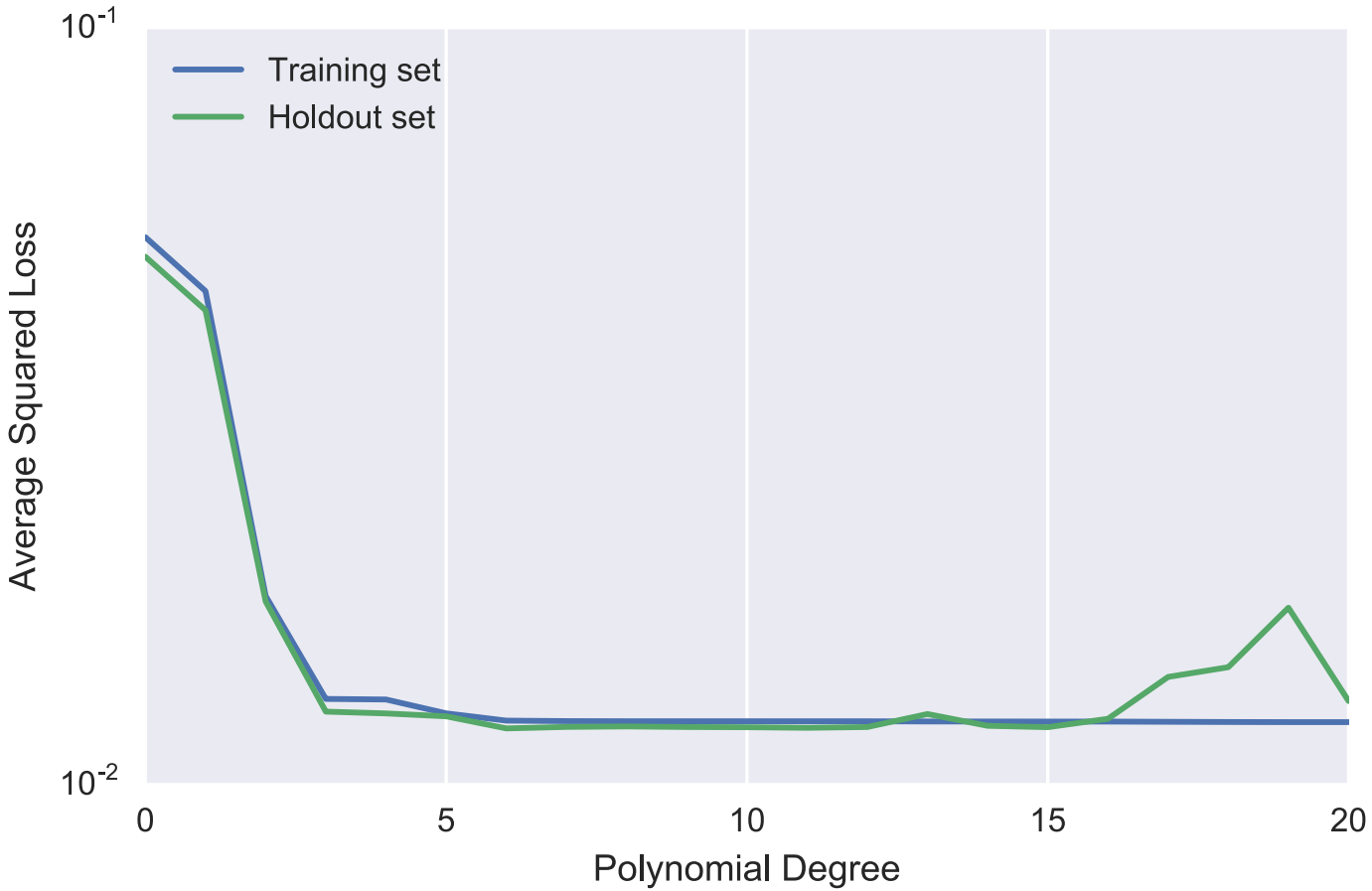
Illustrating cross-validation



Training and cross-validation loss by degree



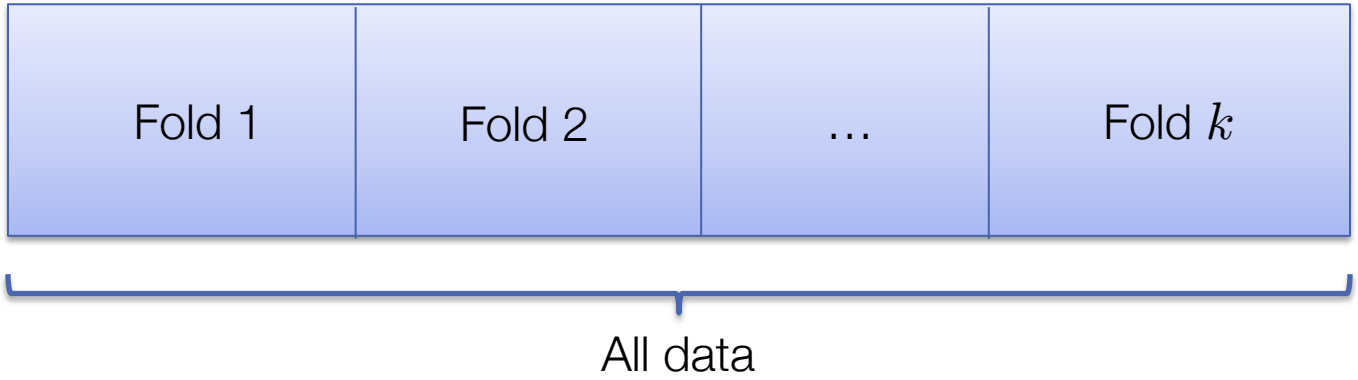
Training and cross-validation loss by degree



K-fold cross-validation

A more involved (but actually slightly more common) version of cross validation

Split data set into k disjoint subsets (folds); train on $k-1$ and evaluate on remaining fold; repeat k times, holding out each fold once



Report average error over all held out folds

Variants

Leave-one-out cross-validation: the limit of k-fold cross-validation, where each fold is only a single example (so we are training on all other examples, testing on that one example)

> [Somewhat surprisingly, for least squares this can be computed *more* efficiently than k-fold cross validation, same complexity solving for the optimal θ using matrix equation]

Stratified cross-validation: keep an approximately equal percentage of positive/negative examples (or any other feature), in each fold

Warning: k-fold cross validation is *not* always better (e.g., in time series prediction, you would want to have holdout set all occur after training set)

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Regularization

We have seen that the degree of the polynomial acts as a natural measure of the "complexity" of the model, higher degree polynomials are more complex (taken to the limit, we fit any finite data set exactly)

But fitting these models also requires extremely *large* coefficients on these polynomials

For 50 degree polynomial, the first few coefficients are

$$\theta = -3.88\times10^6, 7.60\times10^6, 3.94\times10^6, -2.60\times10^7, ...$$

This suggests an alternative way to control model complexity: keep the *weights small* (**regularization**)

Regularized loss minimization

This leads us back to the regularized loss minimization problem we saw before, but with a bit more context now:

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

This formulation trades off loss on the *training* set with a penalty on high values of the parameters

By varying λ from zero (no regularization) to infinity (infinite regularization, meaning parameters will all be zero), we can sweep out different sets of model complexity

Regularized least squares

For least squares, there is a simple solution to the regularized loss minimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2$$

Taking gradients by the same rules as before gives:

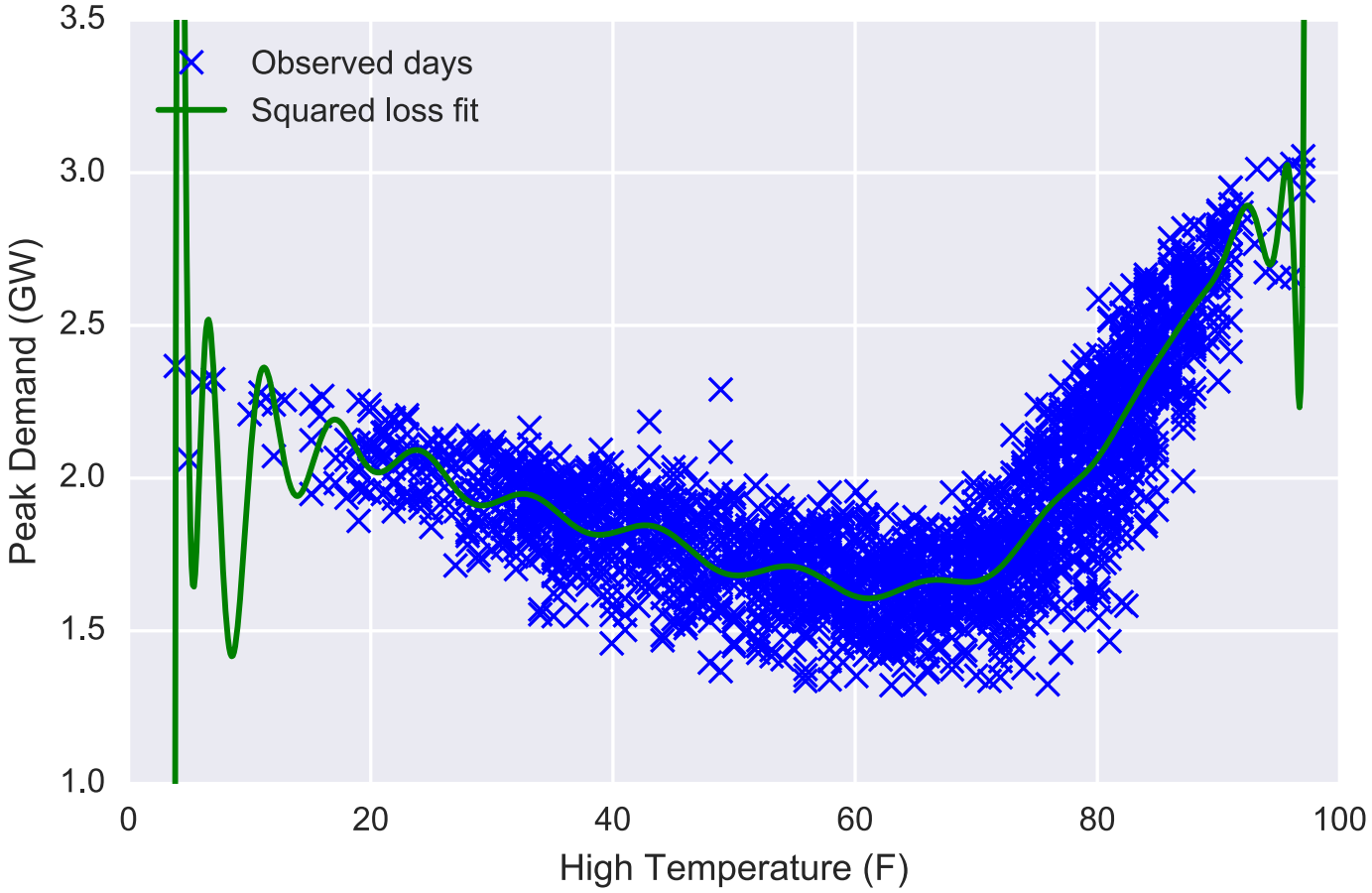
$$\nabla_\theta \left(\sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2 \right) = 2X^T(X\theta - y) + 2\lambda\theta$$

Setting gradient equal to zero leads to the solution

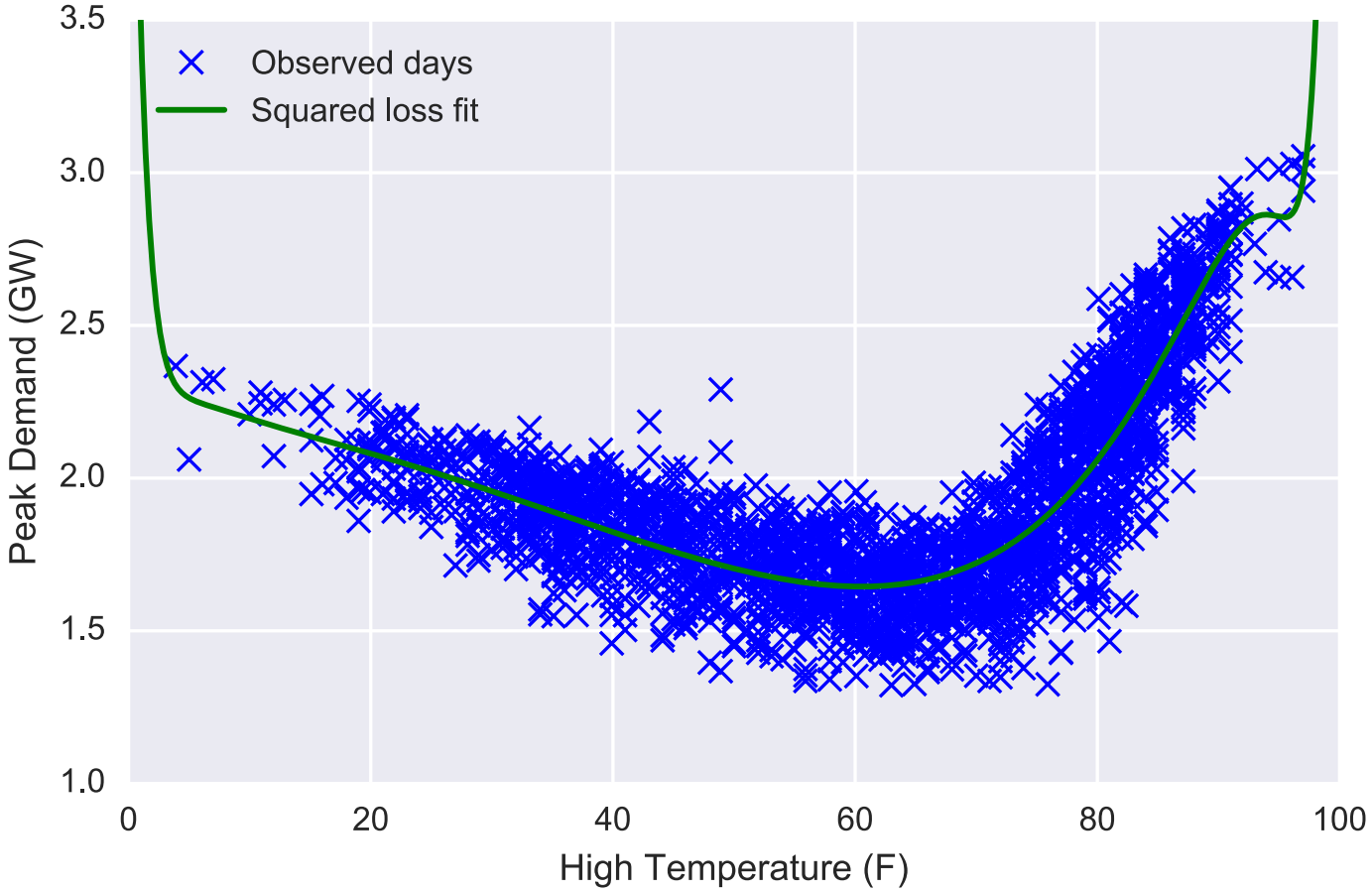
$$2X^T X\theta + 2\lambda\theta = 2X^T y \implies \theta = (X^T X + \lambda I)^{-1} X^T y$$

Looks just like the normal equations but with an additional λI term

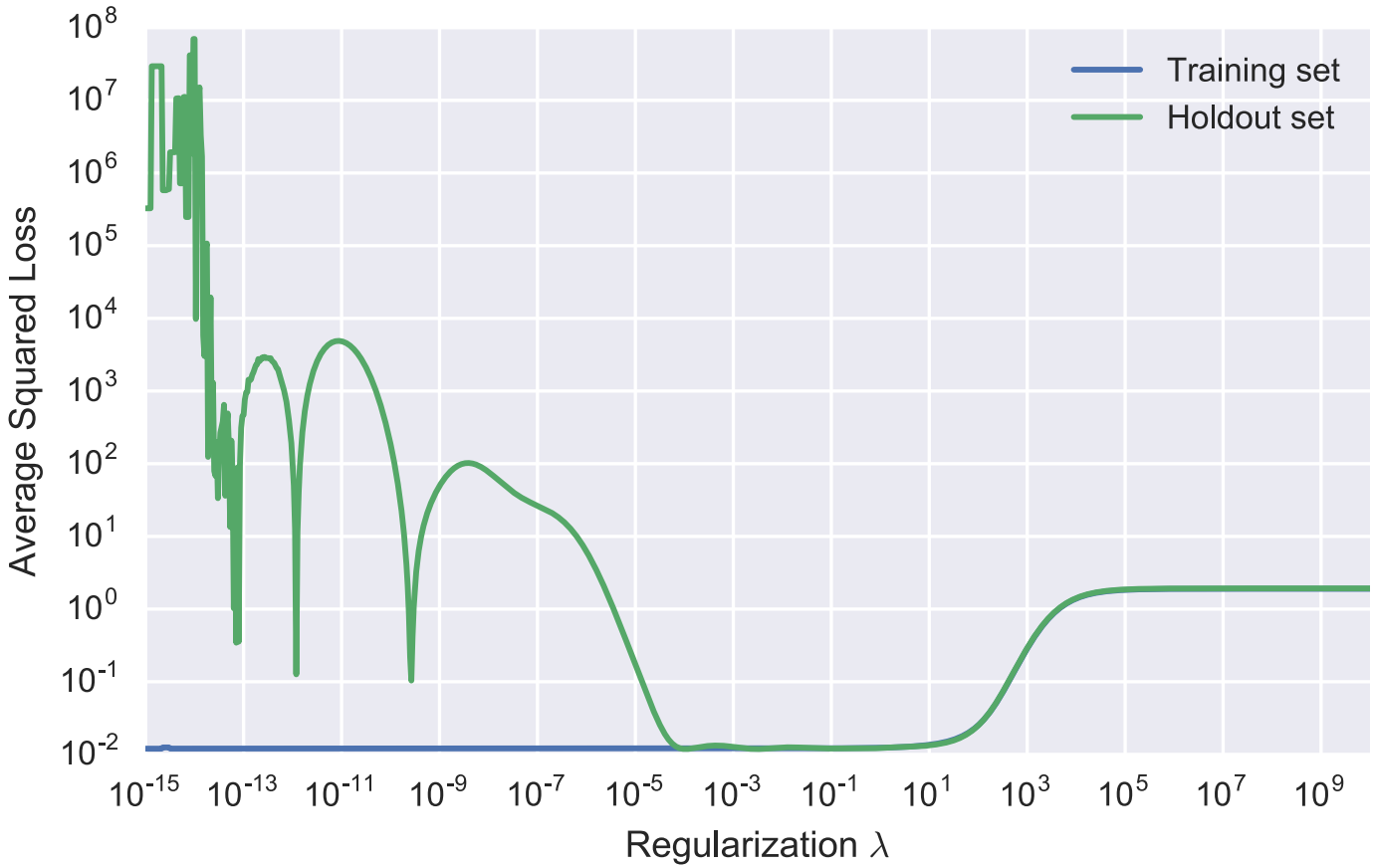
50 degree polynomial fit



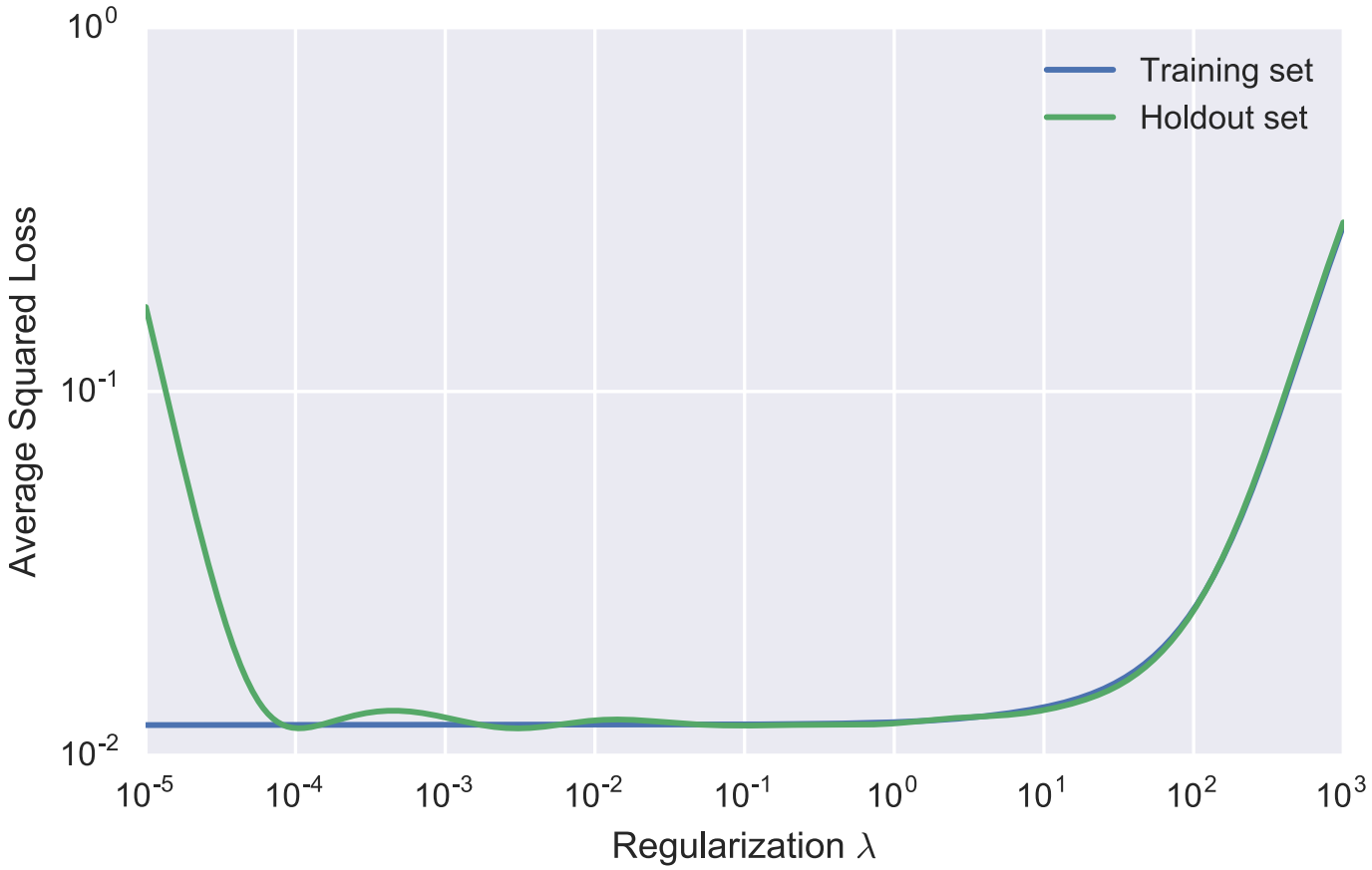
50 degree polynomial fit – $\lambda = 1$



Training/cross-validation loss by regularization



Training/cross-validation loss by regularization



Poll: features and regularization

Suppose you run linear regression with polynomial features and some initial guess for d and λ. You find that your validation loss is much higher than you training loss. Which actions might be beneficial to take?

1. Decrease λ

2. Increase λ

3. Decrease d

4. Increase d

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Notation for more general features

We previously described polynomial features for a *single* raw input, but if our raw input is itself multi-variate, how do we define polynomial features?

Deviating a bit from past notion, for precision here we're going to use $x^{(i)} \in \mathbb{R}^k$ to denote the *raw* inputs, and $\phi^{(i)} \in \mathbb{R}^n$ to denote the input features we construct (also common to use the notation $\phi(x^{(i)})$)

We'll also drop (i) superscripts, but important to understand we're transforming *each* feature this way

E.g., for the high temperature:

$$x = [\text{High_Temperature}], \qquad \phi = \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix}$$

Polynomial features in general

One possibility for higher degree polynomials is to just use an independent polynomial over each dimension (here of degree d)

$$x \in \mathbb{R}^k \implies \phi = \begin{bmatrix} x_1^d \\ \vdots \\ x_1 \\ \vdots \\ x_k^d \\ \vdots \\ x_k \\ 1 \end{bmatrix} \in \mathbb{R}^{kd+1}$$

But this ignores cross terms between different features, i.e., terms like $x_1 x_2^2 x_k$

Polynomial features in general

A better generalization of polynomials is to include *all* polynomial terms between raw inputs up to degree d

$$x \in \mathbb{R}^k \implies \phi = \left\{ \prod_{i=1}^{k} x_i^{b_i} \; : \; \sum_{i=1}^{n} b_i \leq d \right\} \in \mathbb{R}^{\binom{k+d}{k}}$$

Code to generate all polynomial features with degree exactly d:

```
from itertools import combinations_with_replacement
[np.prod(a) for a in combinations_with_replacement(x, d)]
```

Code to generate all polynomial features with degree up to d

```
[np.prod(a) for i in range(d+1) for a in combinations_with_replacement(x,i)]
```

Code for general polynomials

The following code efficiently (relatively) generates all polynomials up to degree d for an entire data matrix X

```
def poly(X,d):
    return np.array([reduce(operator.mul, a, np.ones(X.shape[0]))
                     for i in range(1,d+1)
                     for a in combinations_with_replacement(X.T, i)]).T
```

It is using the same logic as above, but applying it to entire columns of the data at a time, and thus only needs one call to `combinations_with_replacement`

Radial basis functions (RBFs)

For $x \in \mathbb{R}^k$, select some set of p *centers*, $\mu^{(1)}, \dots, \mu^{(p)}$ (we'll discuss shortly how to select these), and create features

$$\phi = \left\{ \exp\left(-\frac{\|x - \mu^{(i)}\|_2^2}{2\sigma^2} \right) : i = 1, \dots, p \right\} \bigcup \{1\} \in \mathbb{R}^{p+1}$$

Very important: need to normalize columns of X (i.e., different features), to all be the same range, or distances wont be meaningful

(Hyper)parameters of the features include the choice of the p centers, and the choice of the *bandwidth* σ

Choose centers, i.e., to be a uniform grid over input space, can choose σ e.g. using cross validation (don't do this, though, more on this shortly)

Example radial basis function

Example:

$$x = [\text{High_Temperature}],$$

$$\mu^{(1)} = [20], \mu^{(2)} = [25], \ldots, \mu^{(16)} = [95], \sigma = 10$$

Leads to features:

$$\phi = \begin{bmatrix} \exp(-(\text{High_Temperature} - 20)^2/200) \\ \vdots \\ \exp(-(\text{High_Temperature} - 95)^2/200) \\ 1 \end{bmatrix}$$

Code for generating RBFs

The following code generates a complete set of RBF features for an entire data matrix $X \in \mathbb{R}^{m \times k}$ and matrix of centers $\mu \in \mathbb{R}^{p \times k}$

```
def rbf(X,mu,sig):
    sqdist = -2*X@mu.T + (X**2).sum(axis=1)[:,None] + (mu**2).sum(axis=1)
    return np.exp(-sqdist/(2*sig**2))
```

Important “trick” is to efficiently compute distances between *all* data points and all centers

Poll: complexity of computing features

For n dimensional input, m examples, p centers, what is the complexity of computing the RBF feature ϕ for *every* training example $x^{(i)}$?

1. $O(mnp)$
2. $O(mp)$
3. $O(mn^2p)$
4. $O(mn^2p^2)$

Difficulties with general features

The challenge with these general non-linear features is that the number of potential features grows very quickly in the dimensionality of the raw input

Polynomials: k-dimensional raw input $\implies \binom{k+d}{k} = O(d^k)$ total features (for fixed d)

RBFs: k-dimensional raw input, uniform grid with d centers over each dimension $\implies d^k$ total features

These quickly become impractical for large feature raw input spaces

Practical polynomials

Don't use the full set of all polynomials, for anything but very low dimensional input data (say $k \leq 4$)

Instead, form polynomials only of features where you know that the relationship may be important:

- E.g. $\mathrm{Temperature}^2 \cdot \mathrm{Weekday}$, but not $\mathrm{Temperature} \cdot \mathrm{Humidity}$

For binary raw inputs, no point in taking every power ($x_i^2 = x_i$)

These elements do all require some insight into the problem

Practical RBFs

Don't create RBF centers in a grid over your raw input space (your data will *never* cover an entire high-dimensional space, but will lie on a subset)

Instead, pick centers by randomly choosing p data points in the training set (a bit fancier, run k-means to find centers, which we'll describe later)

Don't pick σ using cross validation

Instead, choose the following (called the *median trick)*

$$\sigma = \mathrm{median}(\{\|\mu^{(i)} - \mu^{(j)}\|_2, i, j = 1, \dots, p\})$$

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Kernels

One of the most prominent advances in machine learning in the past 20 years (recently fallen out of favor relative to neural networks, but still can be the best-performing approach for many “medium-sized” problems)

Kernels fundamentally are about specific hypothesis function

$$h_\theta(x) = \sum_{i=1}^{m} \theta_i K(x, x^{(i)})$$

where $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a kernel function

Kernels can *implicitly* represent high dimensional feature vectors *without* the need to form them explicitly (we won’t prove this here, but will provide a short description in the notes over break)

Kernels as high dimensional features

1. **Polynomial Kernel**

$$K(x, z) = (1 + x^T z)^d$$

is equivalent to using full degree d polynomial ($\binom{n+d}{d}$-dimension) features in the raw inputs

2**. RBF Kernel**

$$K(x, z) = \exp\left(-\frac{\|x - z\|_2^2}{2\sigma^2}\right)$$

is equivalent to an *infinite dimensional* RBF feature with centers at *every point in space*

Kernels: what is the "catch"

What is the downside of using kernels?

Recall hypothesis function

$$h_\theta(x) = \sum_{i=1}^{m} \theta_i K(x, x^{(i)})$$

Note that we need a parameter for every training example (complexity increases with the size of the training set)

Called a *non-parametric method* (number of parameters increase with the number of data points)

Typically, complexity of resulting ML algorithm is $O(m^2)$ (or larger), leads to impractical algorithms on large data sets

Poll: complexity of gradient descent with kernels

Given by kernel hypothesis function

$$h_\theta(x) = \sum_{i=1}^{m} \theta_i K(x, x^{(i)})$$

and the RBF kernel, what is the complexity of computing the gradient of the machine learning objective

$$\nabla_\theta \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})\,?$$

1. $O(mn)$
2. $O(mn^2)$
3. $O(m^2n)$
4. $O(m^2n^2)$

Outline

Example: return to peak demand prediction

Overfitting, generalization, and cross validation

Regularization

General nonlinear features

Kernels

Nonlinear classification

Nonlinear classification

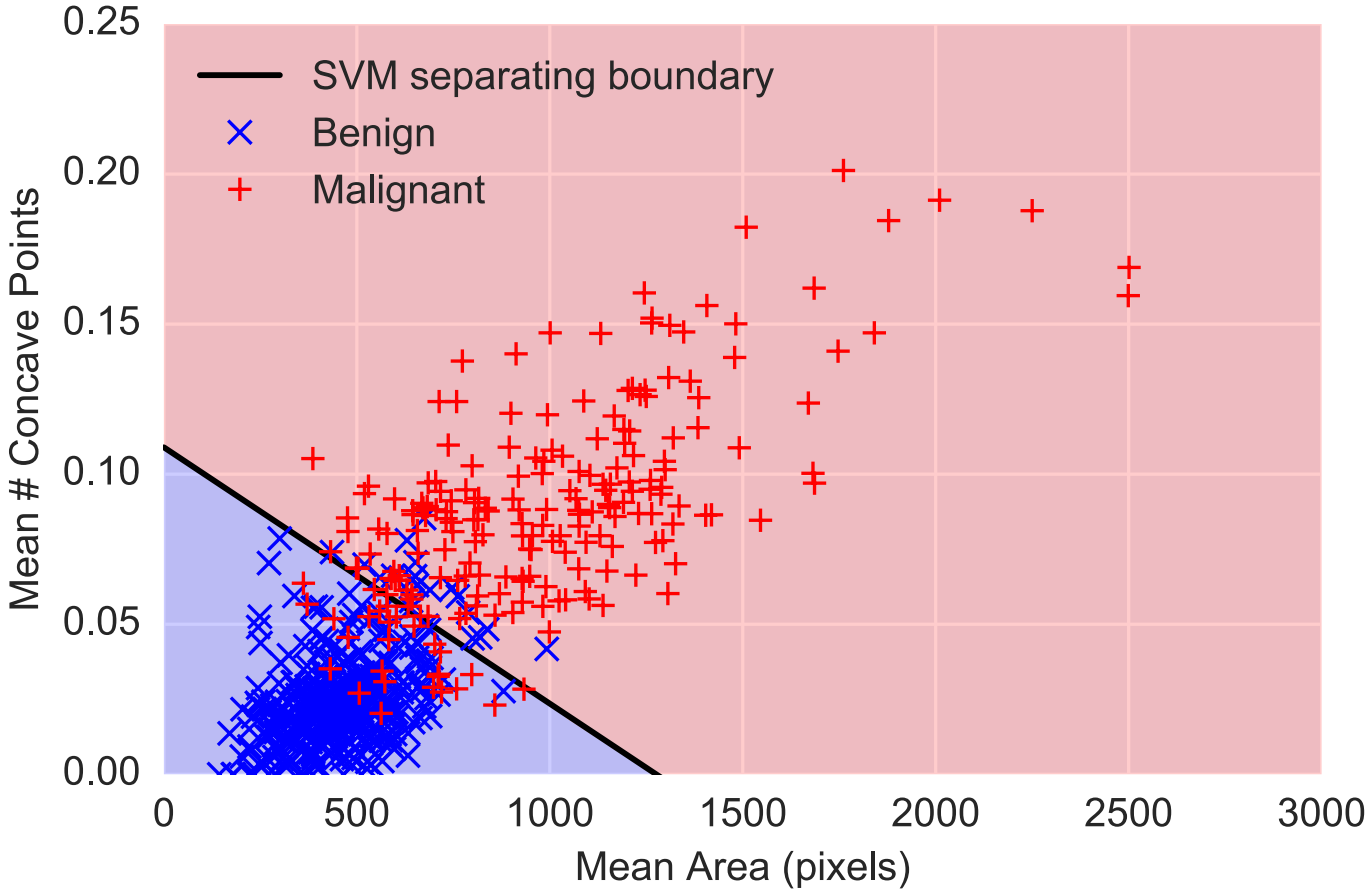
Just like linear regression, the nice thing about using nonlinear features for classification is that our algorithms remain exactly the same as before

I.e., for an SVM, we just solve (using gradient descent)

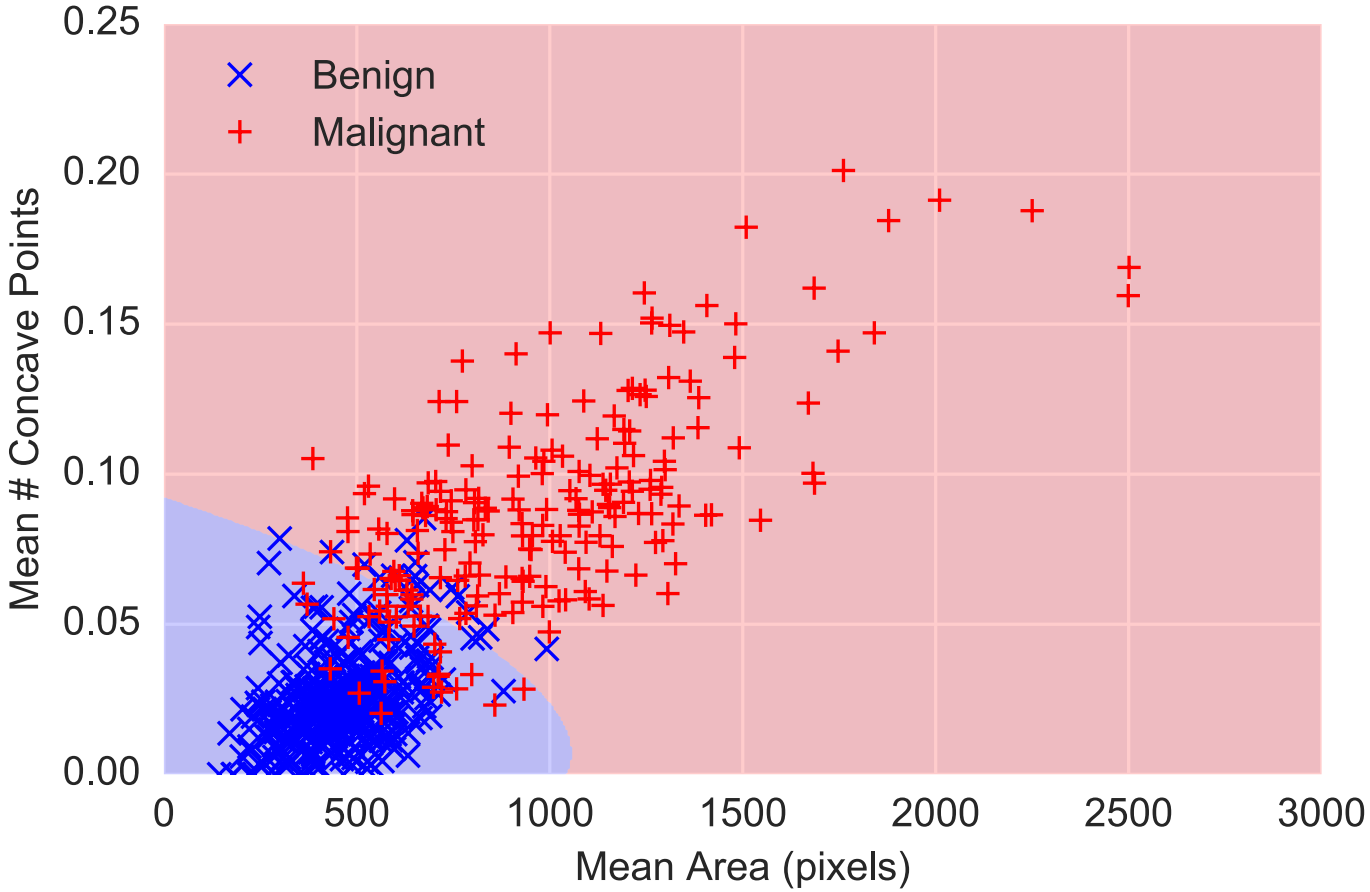
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \max\{1 - y^{(i)} \cdot \theta^T x^{(i)}, 0\} + \frac{\lambda}{2} \|\theta\|_2^2$$

Only difference is that $x^{(i)}$ now contains non-linear functions of the input data

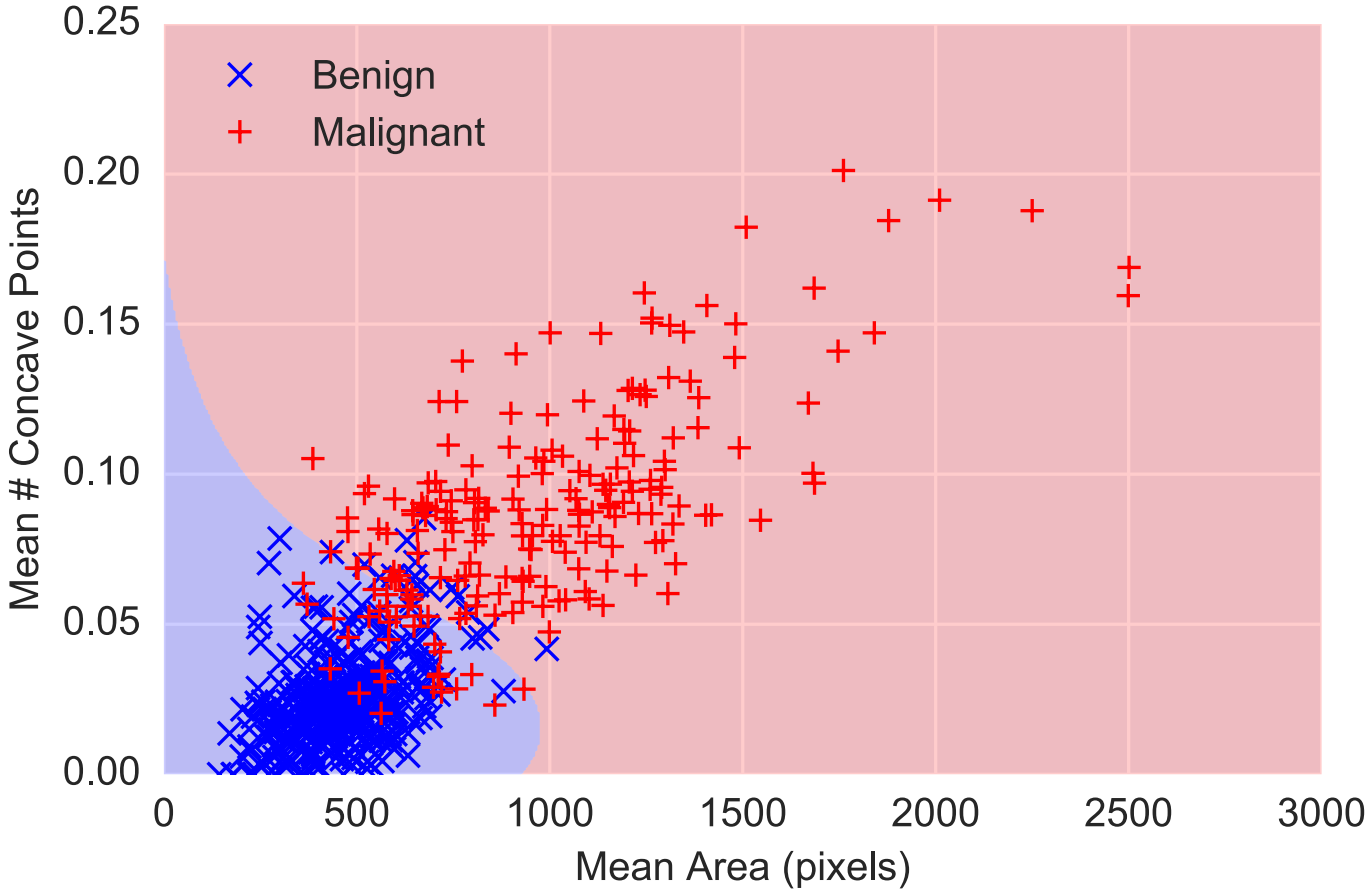
Linear SVM on cancer data set



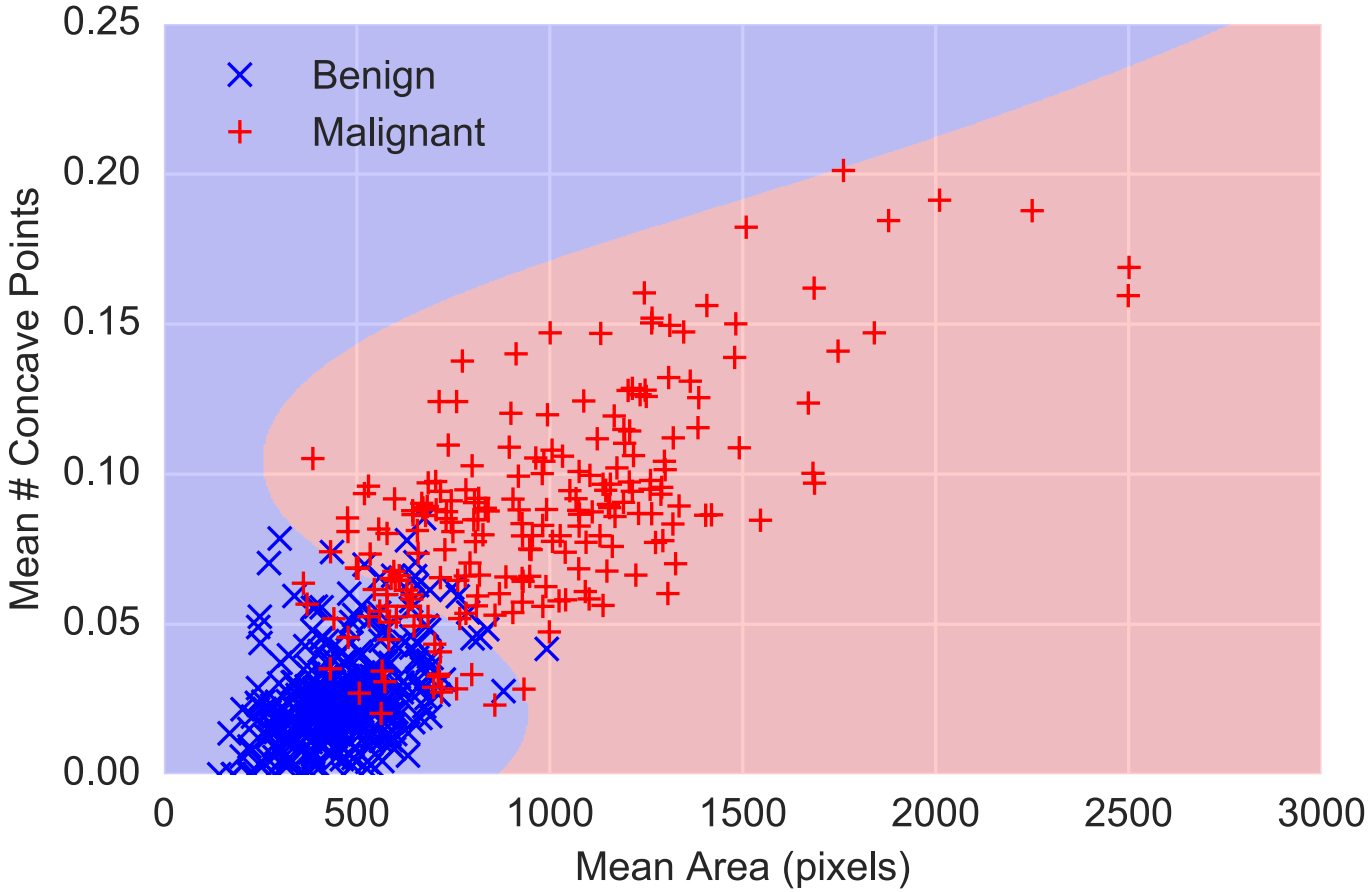
Polynomial features $d = 2$



Polynomial features $d = 3$



Polynomial features $d = 10$



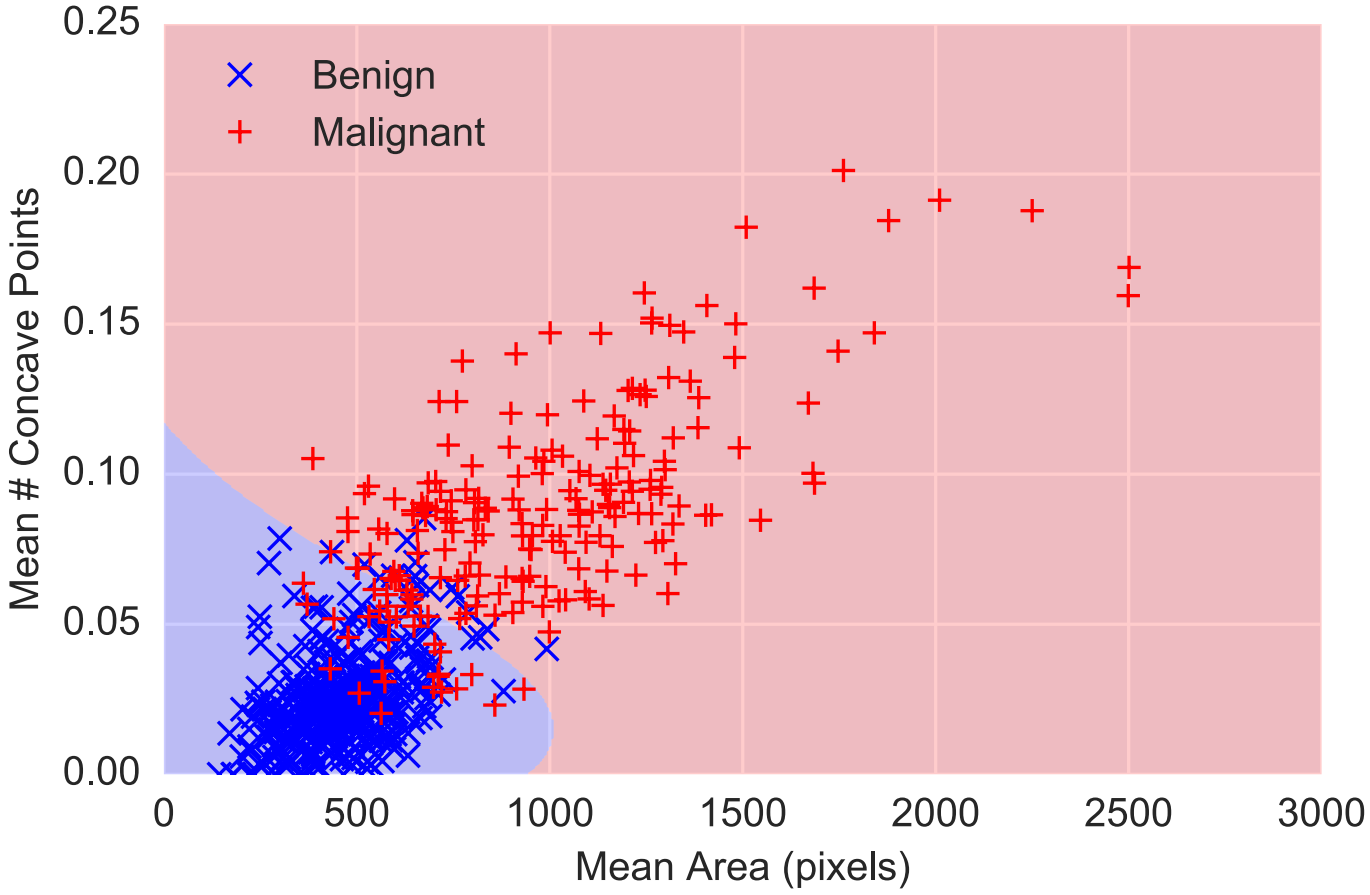
RBF features

Below, we assume that X has been normalized so that each feature lies between $[-1, +1]$ (same as we did for polynomial features)

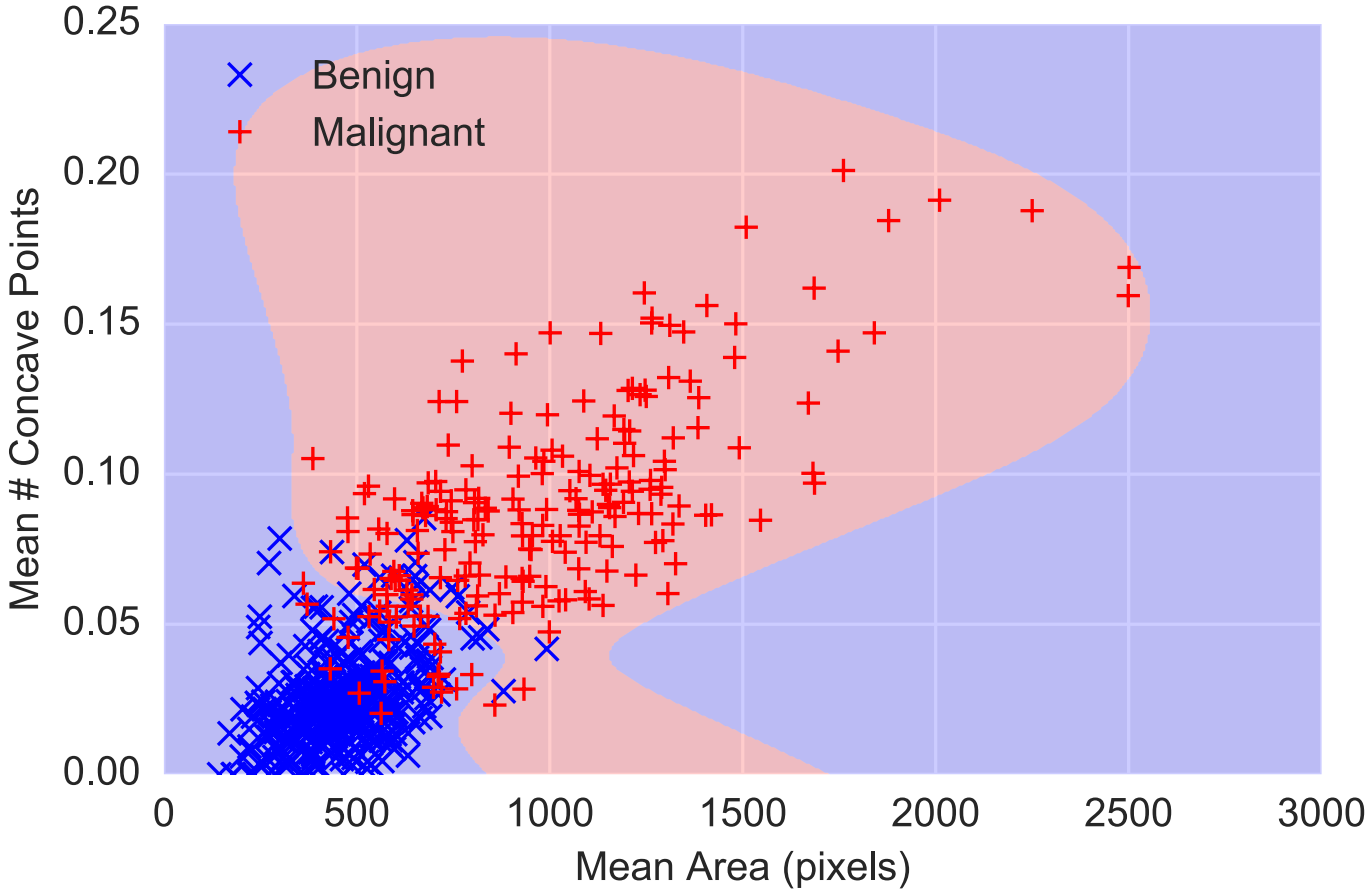
We're consider to observe how the classifier changes as we change different parameters of the RBFs

p will refer to total number of centers, d will refer the number of centers along each dimensions, assuming centers form a regular grid (so since we have two raw inputs, $p = d^2$)

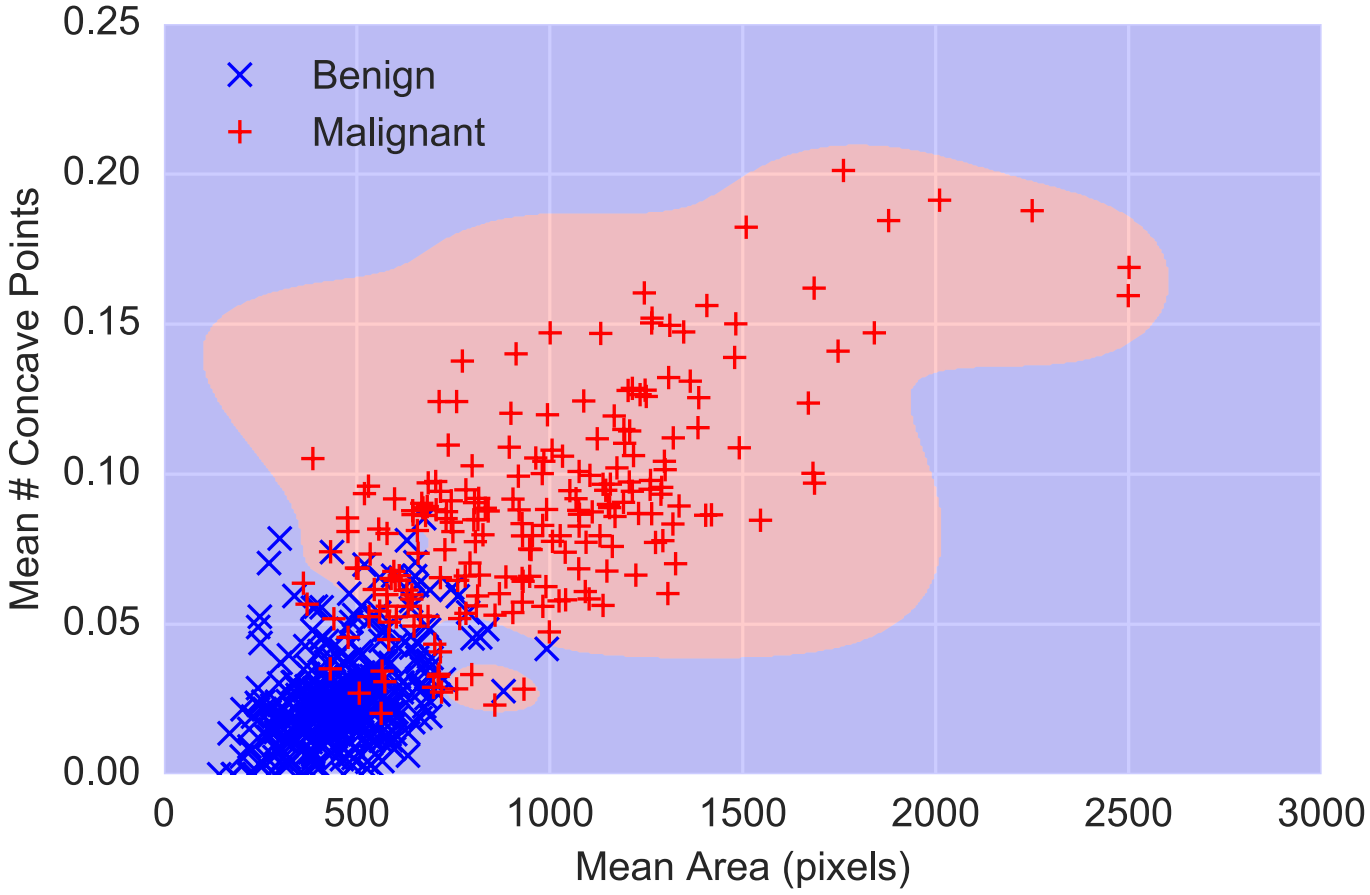
RBF features, $d = 3, \sigma = 2/d$



RBF features, $d = 10, \sigma = 2/d$



RBF features, $d = 20, \sigma = 2/d$

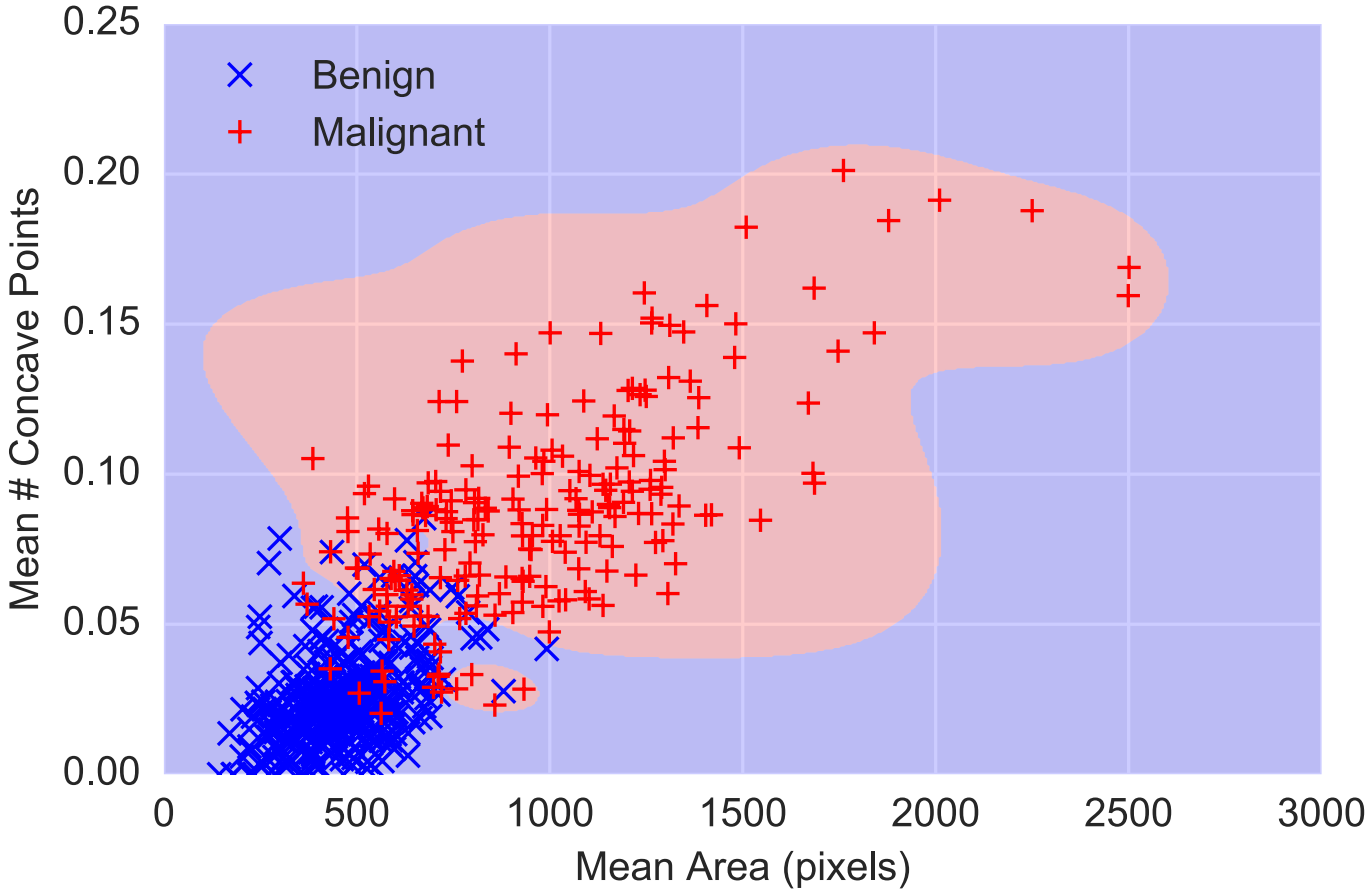


Model complexity and bandwidth

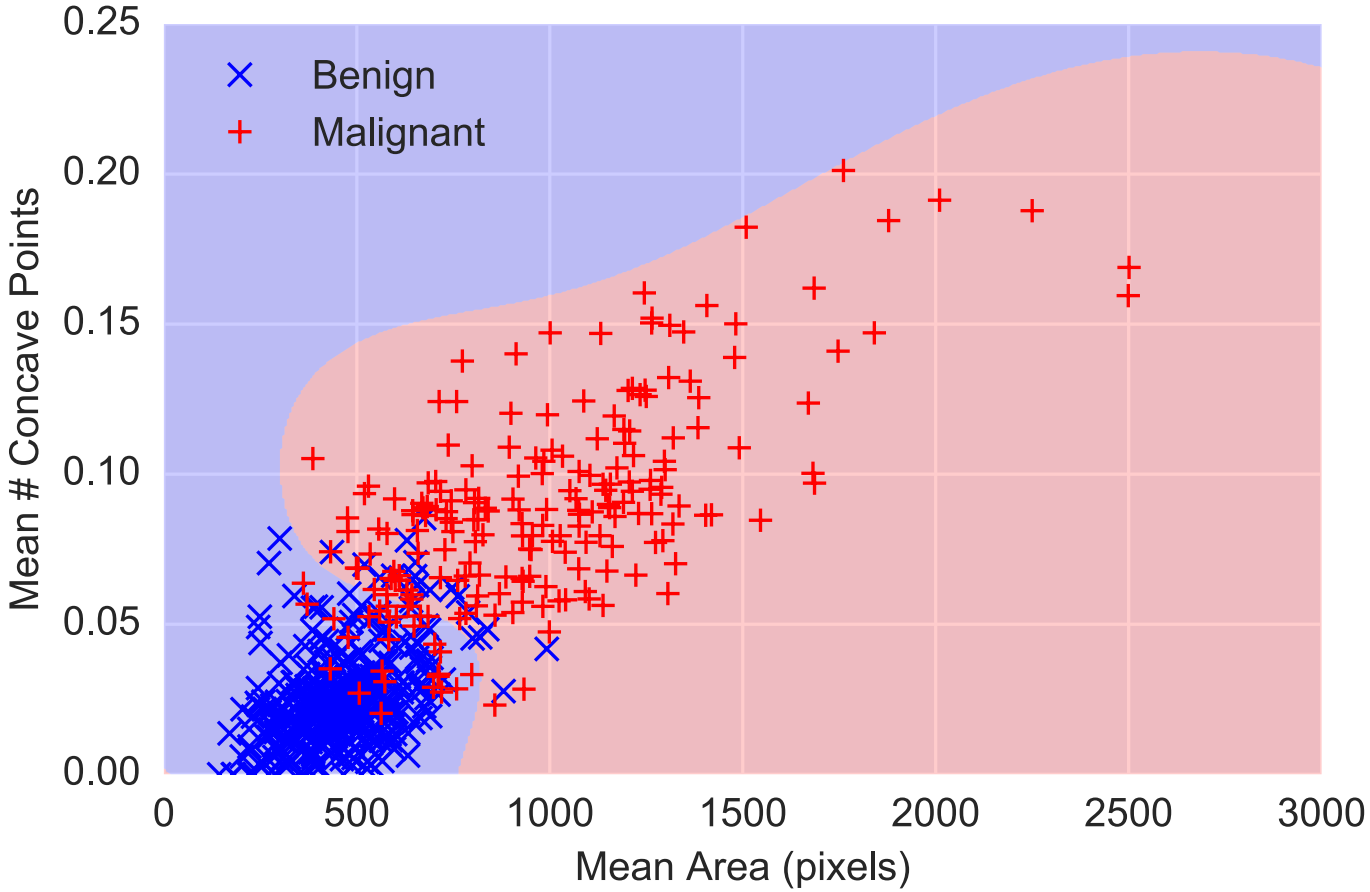
We can control model complexity with RBFs in three ways: two of which we have already seen

1. Choose number of RBF centers
2. Increase/decrease regularization parameter
3. Increase/decrease bandwidth

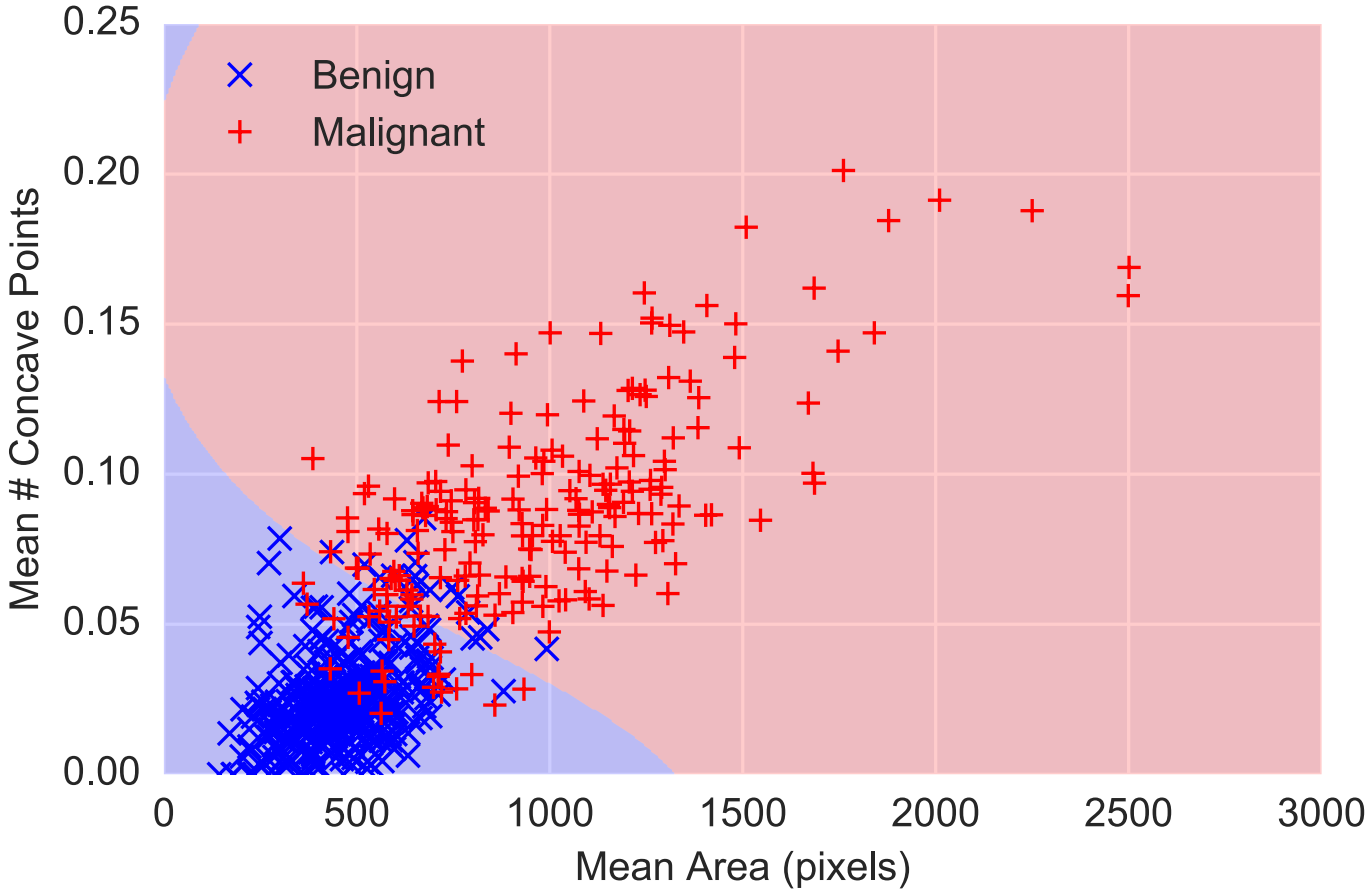
RBF features, $d = 20, \sigma = 0.1$



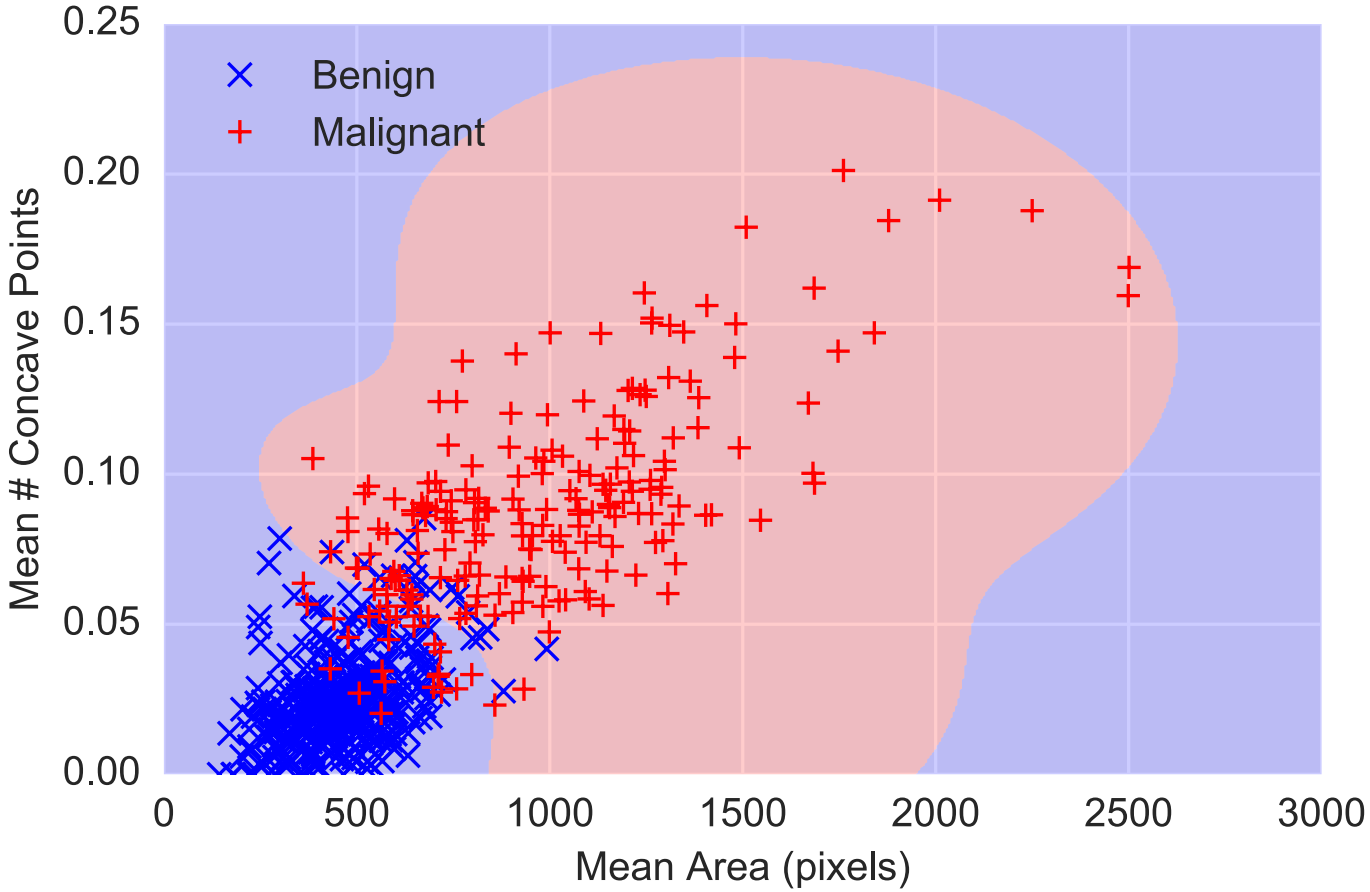
RBF features, $d = 20, \sigma = 0.5$



RBF features, $d = 20, \sigma = 1.07$ (median trick)



RBFs from data, $p = 50, \sigma = \text{median_trick}$



15-388/688 - Practical Data Science: Anomaly detection and mixture of Gaussians

J. Zico Kolter
Carnegie Mellon University
Spring 2018

Outline

Anomalies and outliers

Multivariate Gaussian

Mixture of Gaussians

Outline

Anomalies and outliers

Multivariate Gaussian

Mixture of Gaussians

What is an “anomaly”

Two views of anomaly detection

Supervised view: anomalies are what some user labels as anomalies

Unsupervised view: anomalies are outliers (points of low probability) in the data

In reality, you want a combination of both these viewpoints: not all outliers are anomalies, but all anomalies should be outliers

This lecture is going to focus on the unsupervised view, but this is only part of the full equation

What is an outlier?

Outliers are points of low probability

Given a collection of data points $x^{(1)}, \ldots, x^{(m)}$, describe the points using some distribution, then find points with lowest $p\left(x^{(i)}\right)$

Since we are considering points with no labels, this is an *unsupervised* learning algorithm (could formulate in terms of hypothesis, loss, optimization, but instead for this lecture we'll be focusing on the probabilistic notation)

Outline

Anomalies and outliers

Multivariate Gaussian

Mixture of Gaussians

Multivariate Gaussian distributions

We have seen Gaussian distributions previously, but mainly focused on distributions over scalar-valued data $x^{(i)} \in \mathbb{R}$

$$p(x; \mu, \sigma^2) = (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Gaussian distributions generalize nicely to distributions over vector-valued random variables X taking values in $\mathbb{R}^n$

$$\begin{aligned} p(x; \mu, \Sigma) &= |2\pi\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \\ &\equiv \mathcal{N}(x; \mu, \Sigma) \end{aligned}$$

with parameters $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$, and were $|\cdot|$ denotes the determinant of a matrix (also written $X \sim \mathcal{N}(\mu, \Sigma)$)

Properties of multivariate Gaussians

Mean and variance

$$\mathbf{E}[X] = \int_{\mathbb{R}^n} x\mathcal{N}(x;\mu,\Sigma)dx = \mu$$

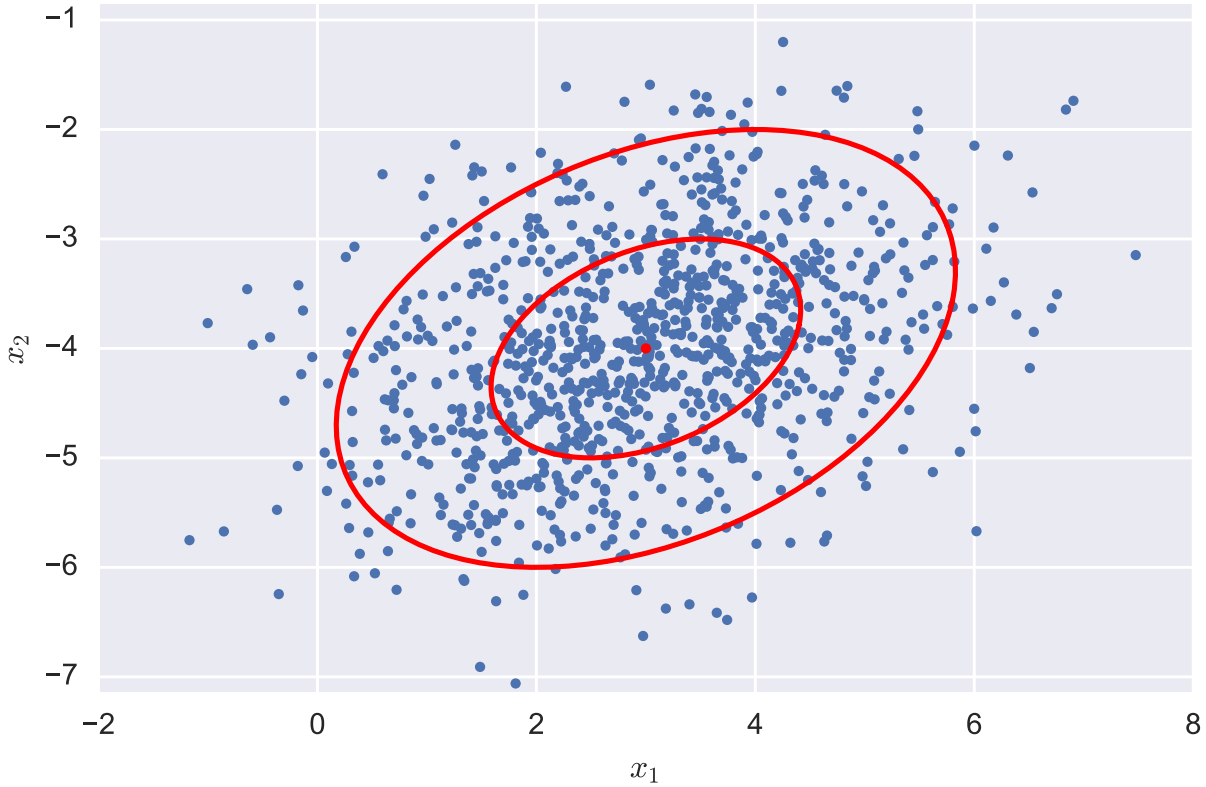
$$\mathbf{Cov}[X] = \int_{\mathbb{R}^n} (x-\mu)(x-\mu)^T\mathcal{N}(x;\mu,\Sigma)dx = \Sigma$$

(these are *not obvious)*

Creation from univariate Gaussians: for $x \in \mathbb{R}$, if $p(x_i) = \mathcal{N}(x;0,1)$ (i.e., each element x_i is an independent univariate Gaussian, then $y = Ax + b$ is also normal, with distribution

$$Y \sim \mathcal{N}(\mu = b, \Sigma = AA^T)$$

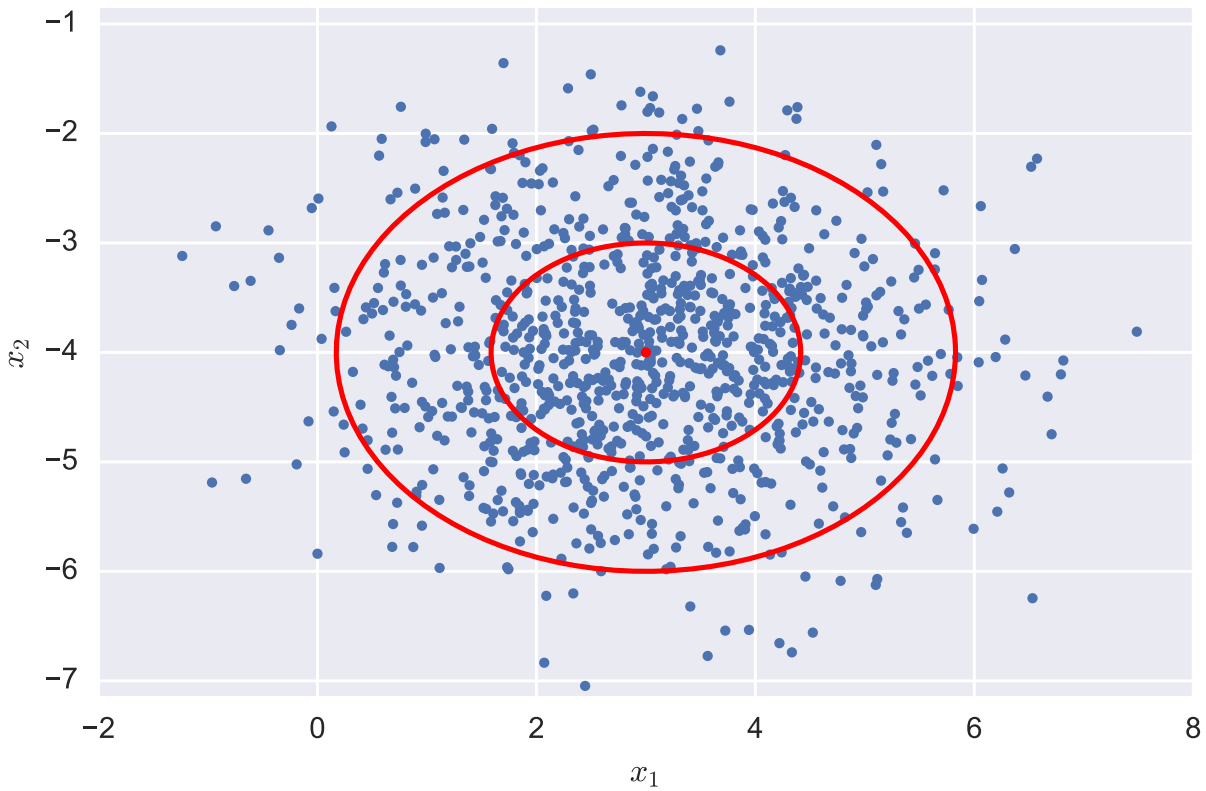
Multivariate Gaussians, graphically



$$\mu = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$$

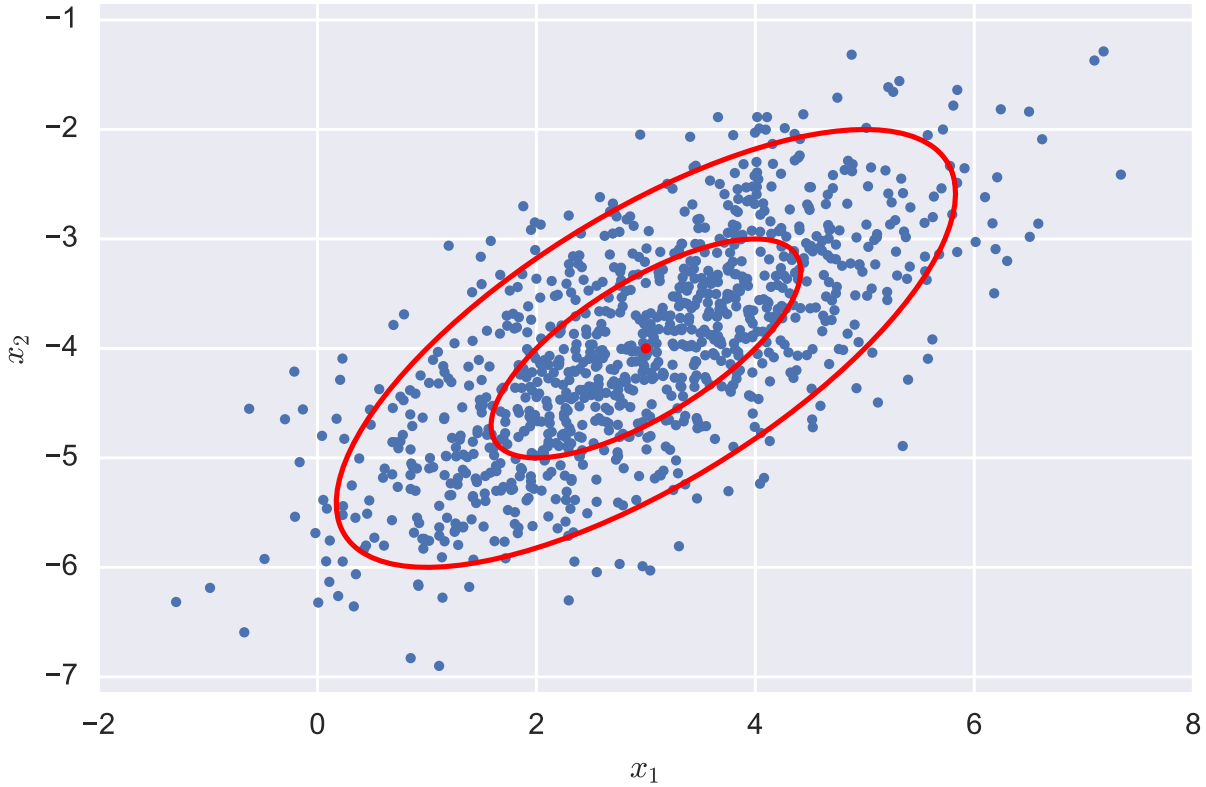
Multivariate Gaussians, graphically



$$\mu = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.0 & 0 \\ 0 & 1.0 \end{bmatrix}$$

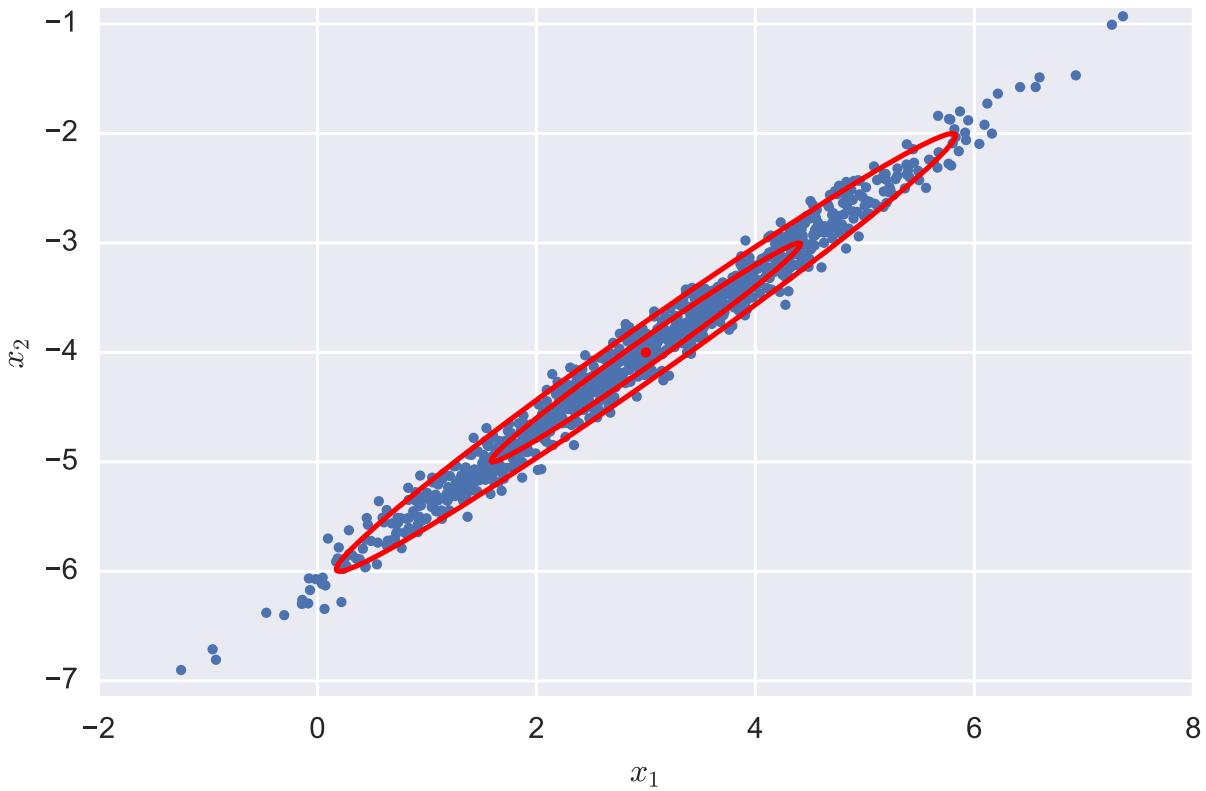
Multivariate Gaussians, graphically



$$\mu = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$

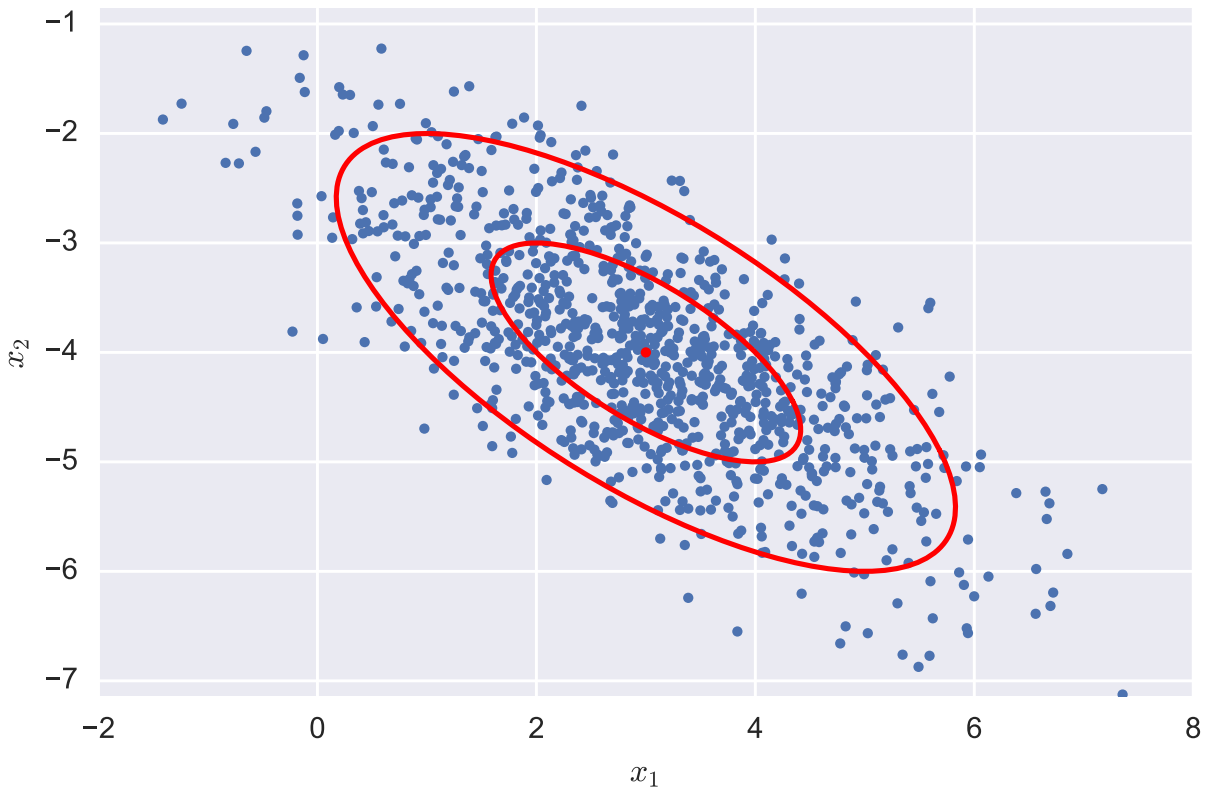
Multivariate Gaussians, graphically



$$\mu = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.0 & 1.4 \\ 1.4 & 1.0 \end{bmatrix}$$

Multivariate Gaussians, graphically



$$\mu = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.0 & -1.0 \\ -1.0 & 1.0 \end{bmatrix}$$

Maximum likelihood estimation

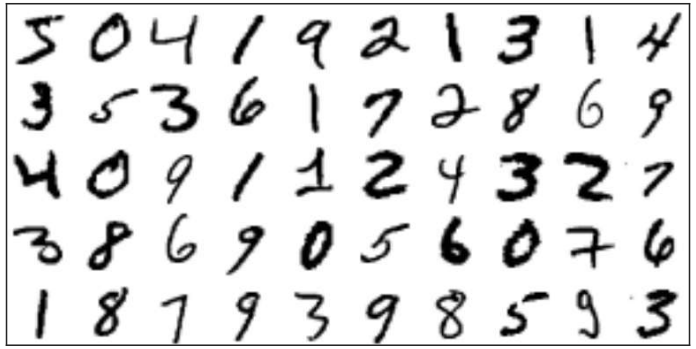
The maximum likelihood estimate of μ, Σ are what you would "expect", but derivation is non-obvious

$$\begin{aligned}\underset{\mu,\Sigma}{\text{minimize}} \;\; \ell(\mu,\Sigma) &= \sum_{i=1}^{m} \log p(x^{(i)};\mu,\Sigma) \\ &= \sum_{i=1}^{m} \left(-\frac{1}{2}\log|2\pi\Sigma| - \frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu) \right)\end{aligned}$$

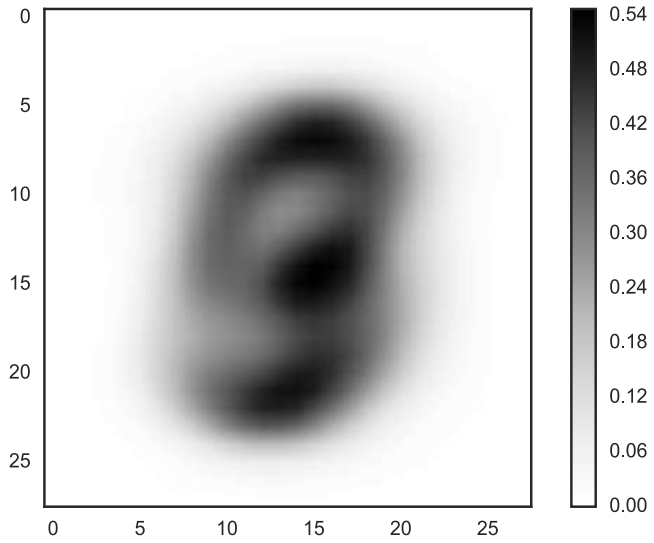
Taking gradients with respect to μ and Σ and setting equal to zero give the closed-form solutions

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$$

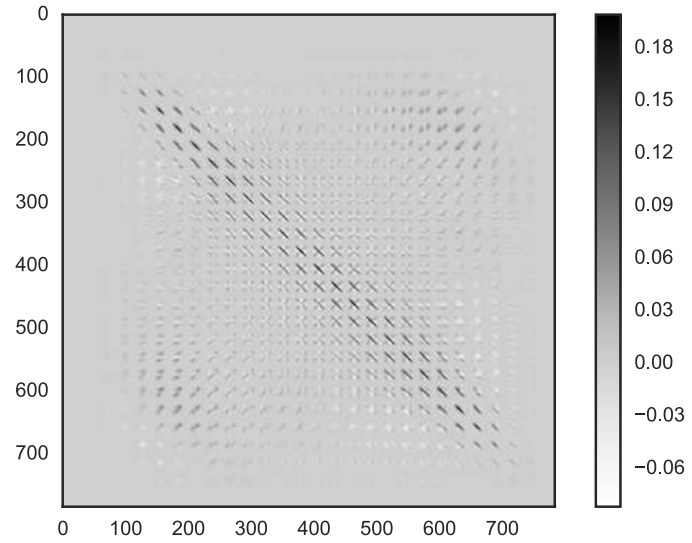
Fitting Gaussian to MNIST



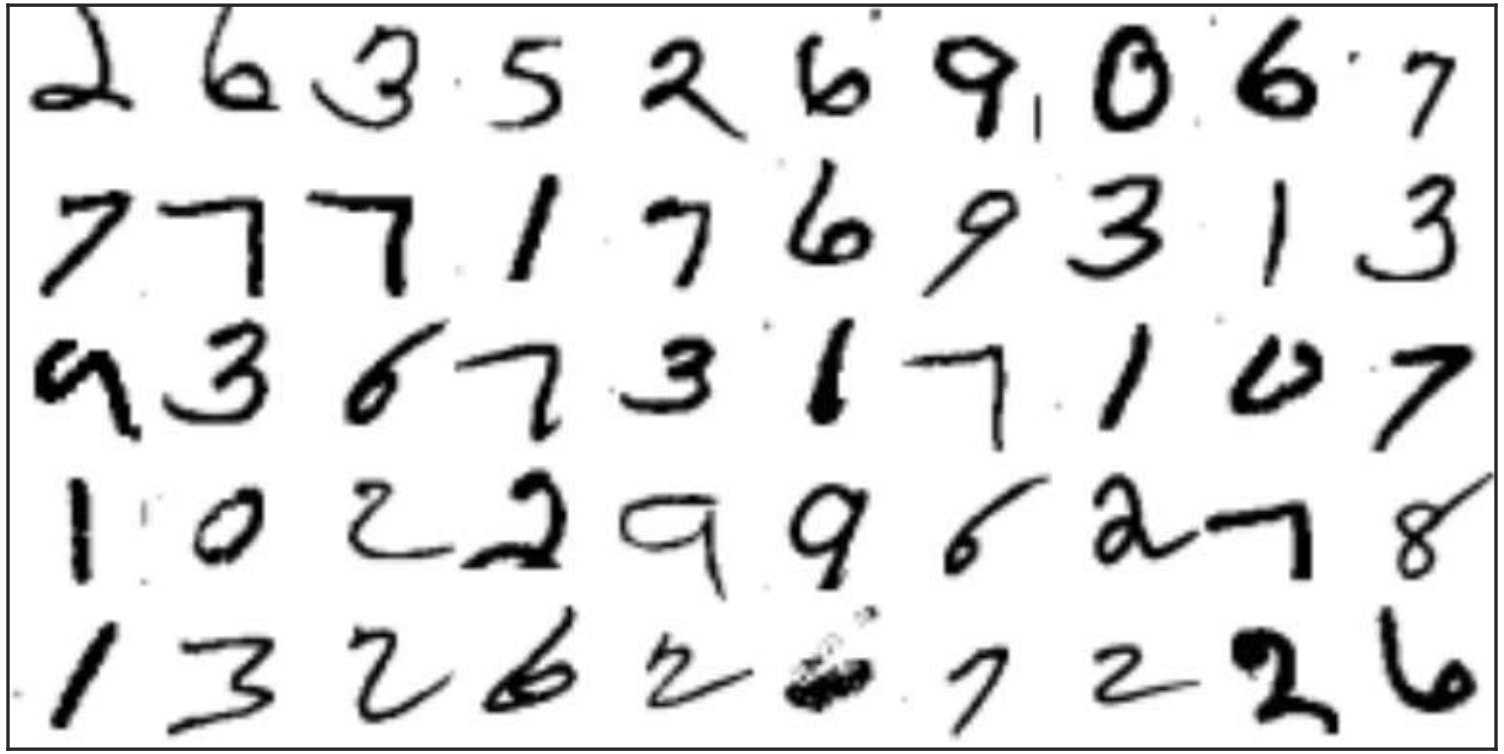
$\mu =$



$\Sigma =$



MNIST Outliers



Outline

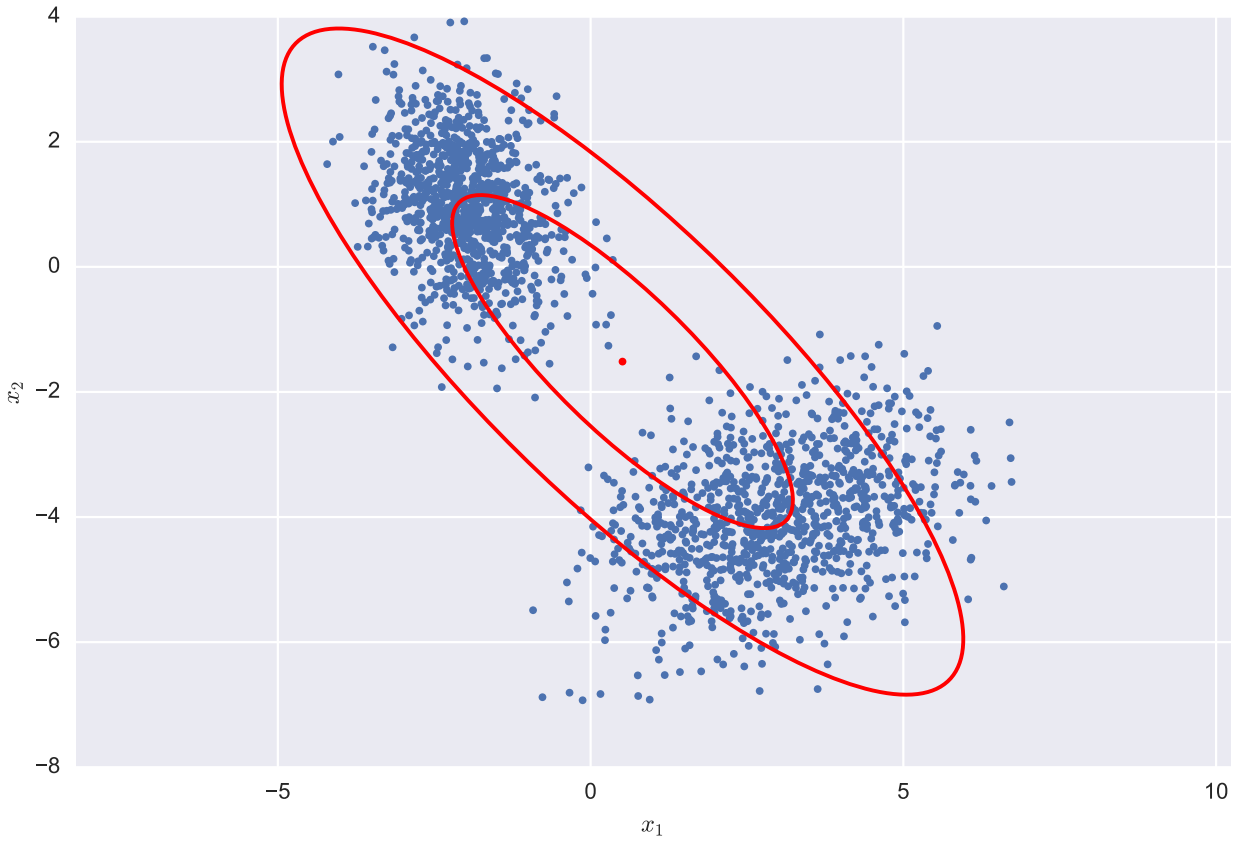
Anomalies and outliers

Multivariate Gaussian

Mixture of Gaussians

Limits of Gaussians

Though useful, multivariate Gaussians are limited in the types of distributions they can represent



Mixture models

A more powerful model to consider is a *mixture* of Gaussian distributions, a distribution where we first consider a categorical variable

$$Z \sim \text{Categorical}(\phi), \qquad \phi \in [0,1]^k, \sum_i \phi_i = 1$$

i.e., z takes on values $\{1, \dots, k\}$

For each potential value of Z, we consider a *separate* Gaussian distribution:

$$X|Z = z \sim \mathcal{N}(\mu^{(z)}, \Sigma^{(z)}), \qquad \mu^{(z)} \in \mathbb{R}^n, \Sigma^{(z)} \in \mathbb{R}^{n \times n}$$

Can write the distribution of X using marginalization

$$p(X) = \sum_z p(X|Z = z)p(Z = z) = \sum_z \mathcal{N}(x; \mu^{(z)}, \Sigma^{(z)})\phi_z$$

Learning mixture models

To estimate parameters, suppose first that we can observe both X and Z, i.e., our data set is of the form $(x^{(i)}, z^{(i)}), i = 1, \ldots, m$

In this case, we can maximize the log-likelihood of the parameters:

$$\ell(\mu, \Sigma, \phi) = \sum_{i=1}^{m} \log p(x^{(i)}, z^{(i)}; \mu, \Sigma, \phi)$$

Without getting into the full details, it hopefully should not be too surprising that the solutions here are given by:

$$\phi_z = \frac{\sum_{i=1}^{m} \mathbf{1}\{z^{(i)} = z\}}{m}, \qquad \mu^{(z)} = \frac{\sum_{i=1}^{m} \mathbf{1}\{z^{(i)} = z\} x^{(i)}}{\sum_{i=1}^{m} \mathbf{1}\{z^{(i)} = z\}},$$

$$\Sigma^{(z)} = \frac{\sum_{i=1}^{m} \mathbf{1}\{z^{(i)} = z\}(x^{(i)} - \mu^{(z)})(x^{(i)} - \mu^{(z)})^T}{\sum_{i=1}^{m} \mathbf{1}\{z^{(i)} = z\}}$$

Latent variables and expectation maximization

In the unsupervised setting, $z^{(i)}$ terms will not be known, these are referred to as *hidden* or *latent* random variables

This means that to estimate the parameters, we can't use the function $1\{z^{(i)} = z\}$ anymore

Expectation maximization (EM) algorithm (at a high level): replace indicators $1\{z^{(i)} = z\}$ with probability estimates $p(z^{(i)} = z | x^{(i)}; \mu, \Sigma, \phi)$

When we re-estimate these parameter, probabilities change, so repeat:

E (expectation) step: compute $p(z^{(i)} = z | x^{(i)}; \mu, \Sigma, \phi), \forall i, z$

M (maximization) step: re-estimate μ, Σ, ϕ

EM for Gaussian mixture models

E step: using Bayes' rule, compute probabilities

$$\hat{p}_z^{(i)} = p\big(z^{(i)} = z \big| x^{(i)}; \mu, \Sigma, \phi\big) = \frac{p\big(x^{(i)} \big| z^{(i)} = z; \mu, \Sigma\big) p\big(z^{(i)} = z; \phi\big)}{\sum_{z'} p\big(x^{(i)} \big| z^{(i)} = z'; \mu, \Sigma\big) p\big(z^{(i)} = z'; \phi\big)}$$

$$= \frac{\mathcal{N}\big(x^{(i)}; \mu^{(z)}, \Sigma^{(z)}\big)\phi_z}{\sum_{z'} \mathcal{N}\big(x^{(i)}; \mu^{(z')}, \Sigma^{(z')}\big)\phi_{z'}}$$

M step: re-estimate parameters using these probabilities

$$\phi_z \leftarrow \frac{\sum_{i=1}^m \hat{p}_z^{(i)}}{m}, \ \mu^{(z)} \leftarrow \frac{\sum_{i=1}^m \hat{p}_z^{(i)} x^{(i)}}{\sum_{i=1}^m \hat{p}_{i,z}}, \ \Sigma^{(z)} \leftarrow \frac{\sum_{i=1}^m \hat{p}_z^{(i)} (x^{(i)} - \mu^{(z)})(x^{(i)} - \mu^{(z)})^T}{\sum_{i=1}^m \hat{p}_z^{(i)}}$$

Local optima

Like k-means, EM is effectively optimizating a *non-convex* problem

Very real possibility of local optima (seemingly moreso than k-means, in practice)

Same heuristics work as for k-means (in fact, common to initialize EM with clusters from k-means)

Illustration of EM algorithm

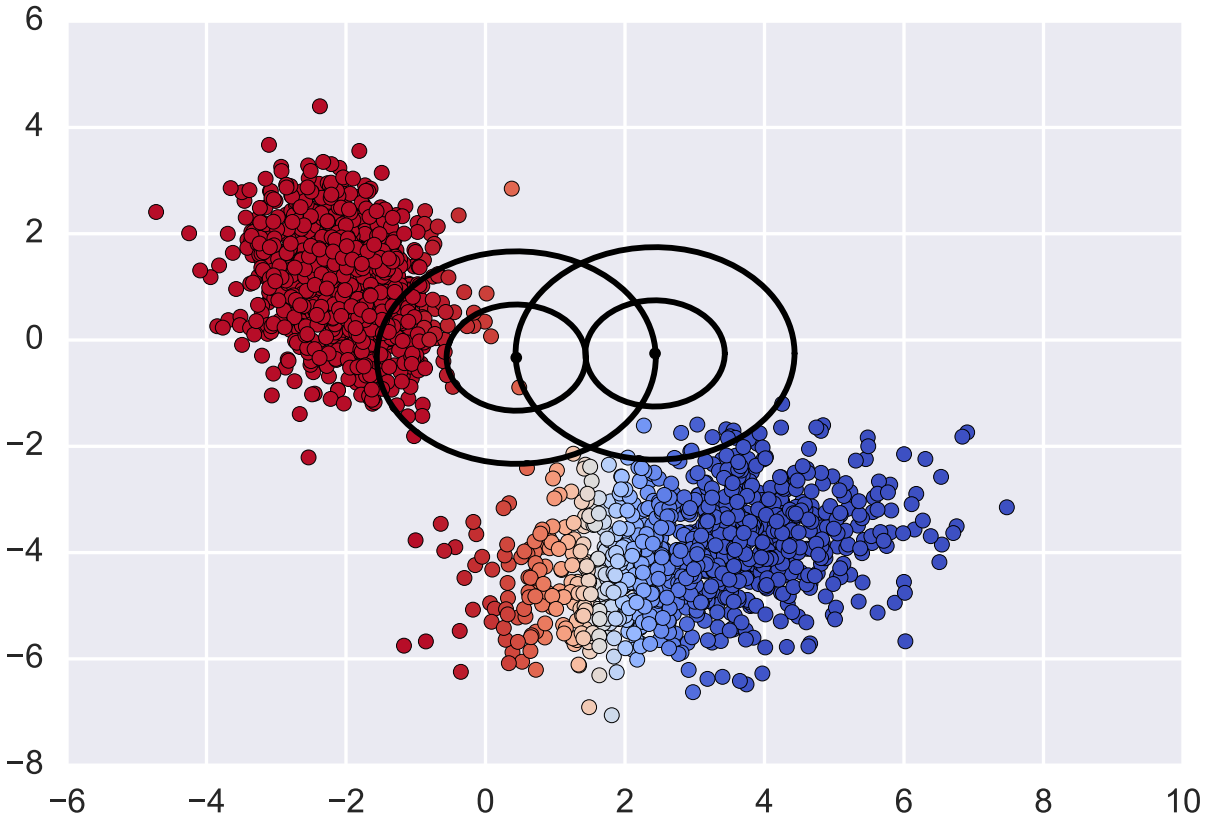


Illustration of EM algorithm

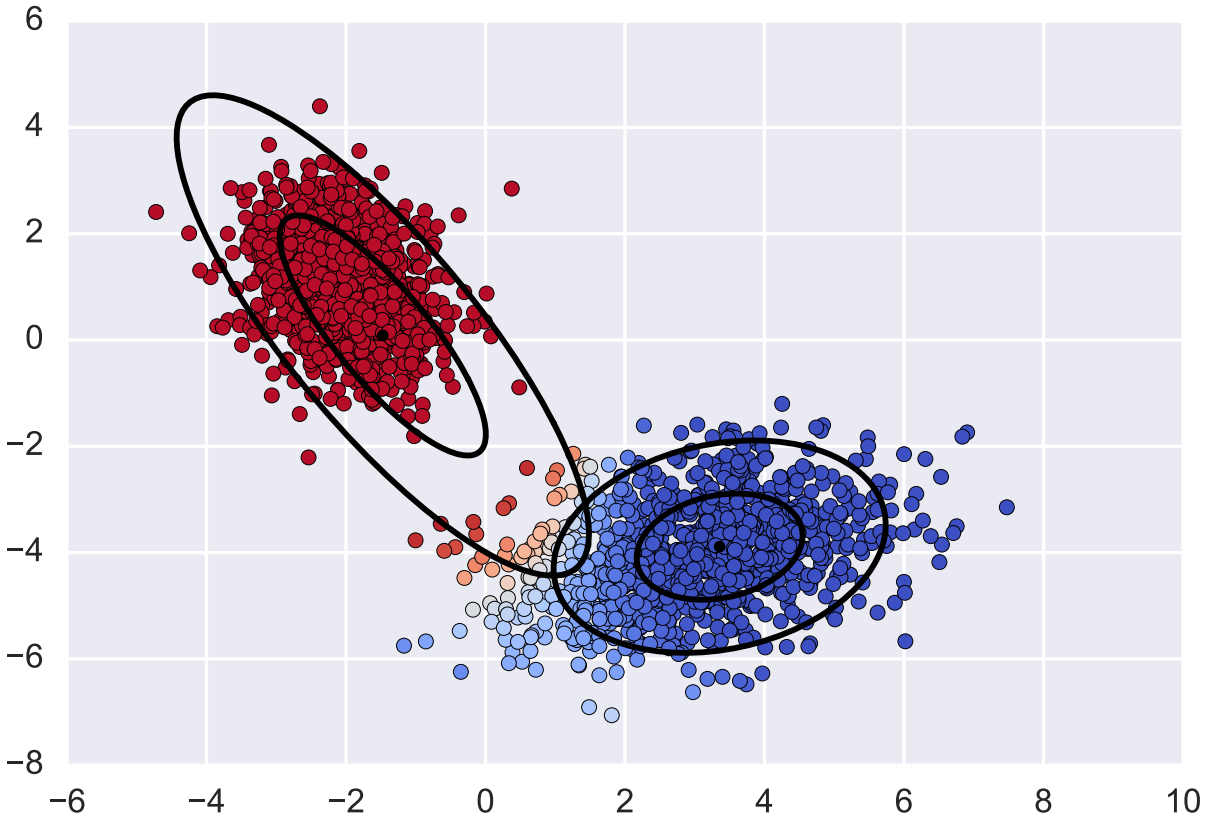


Illustration of EM algorithm

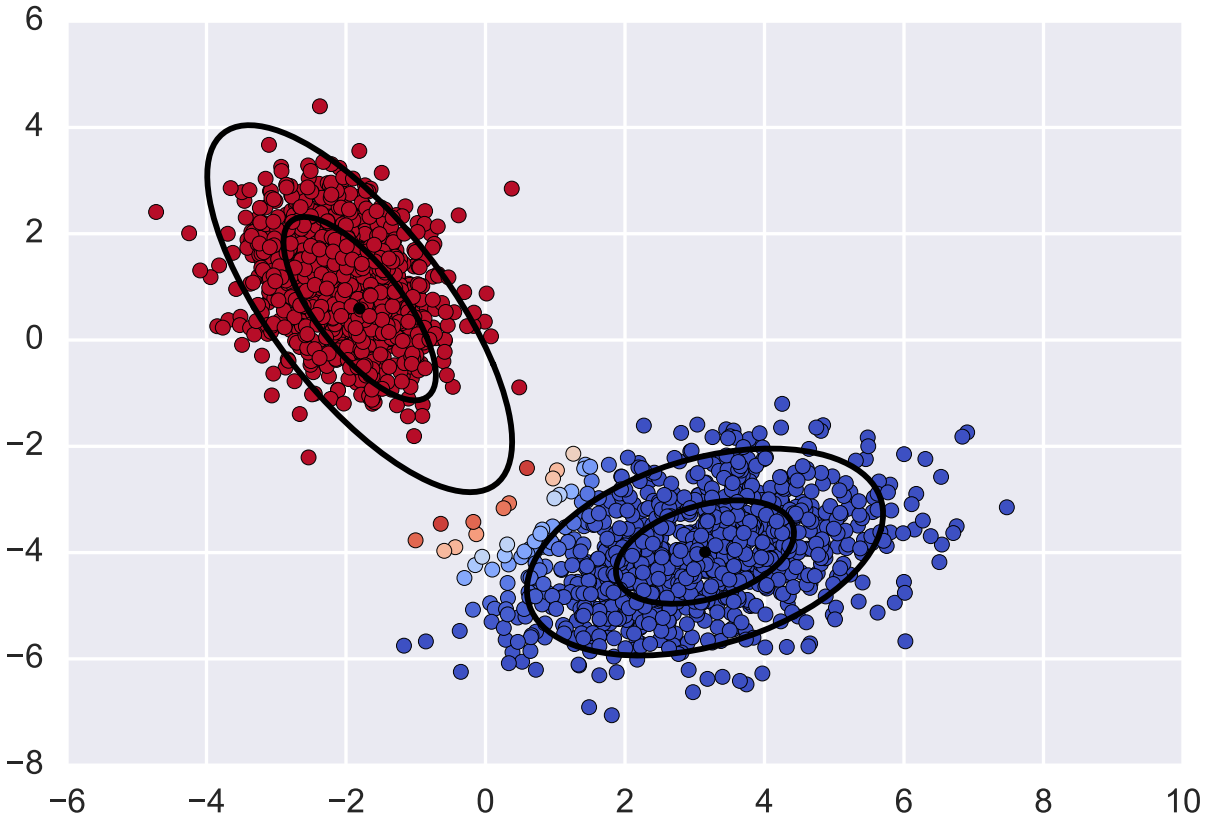


Illustration of EM algorithm

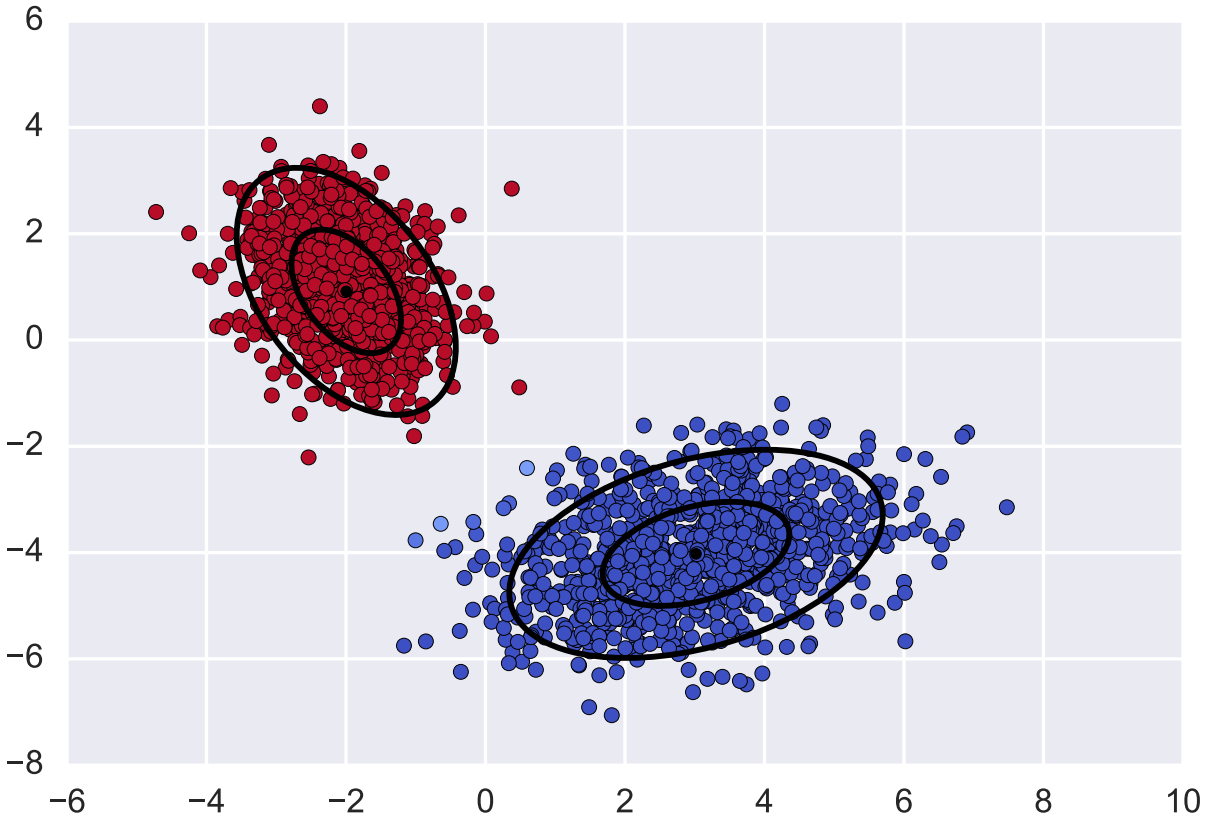
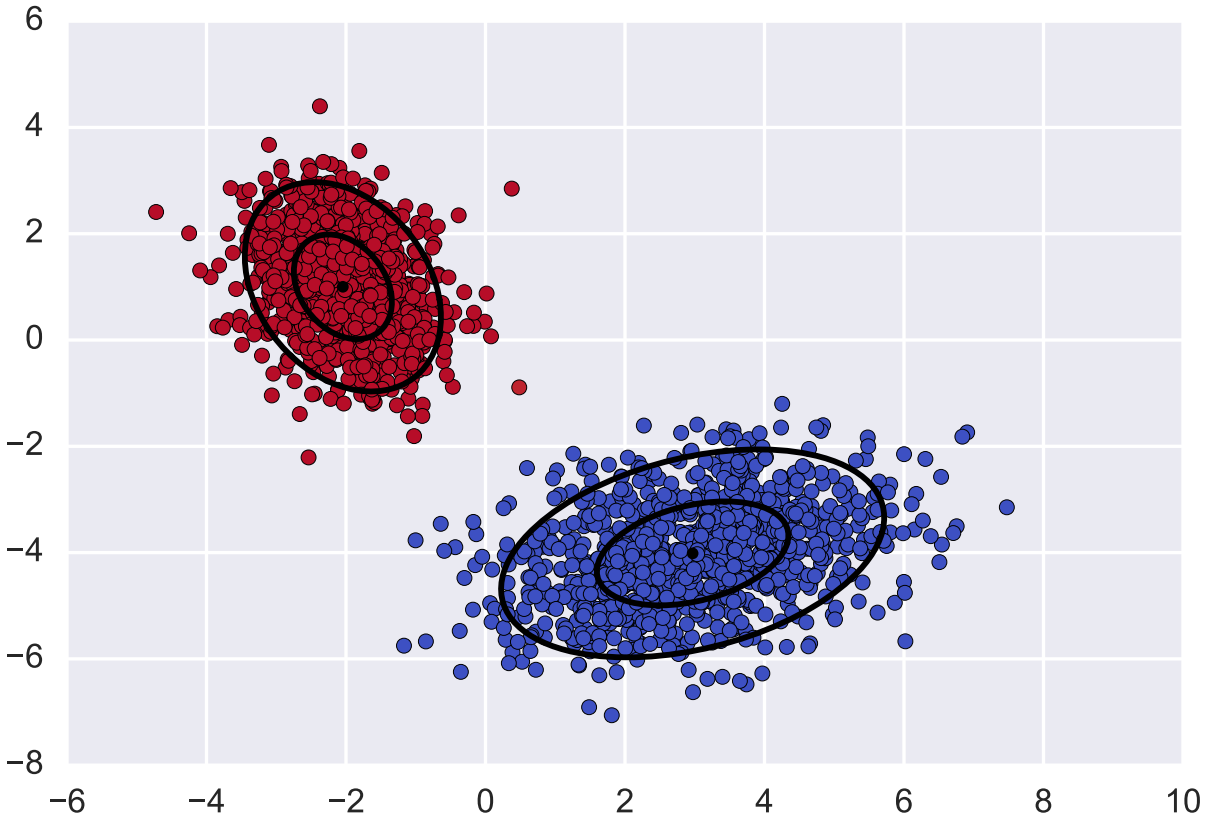
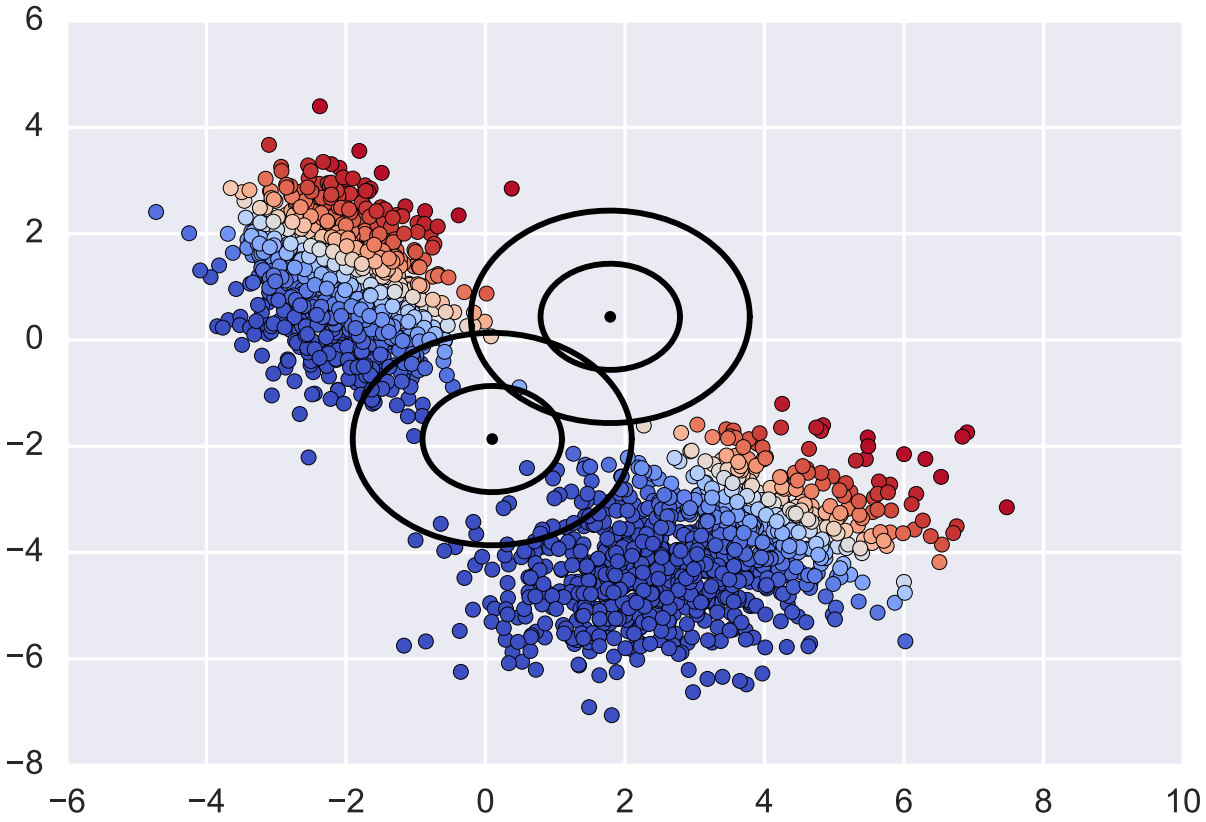


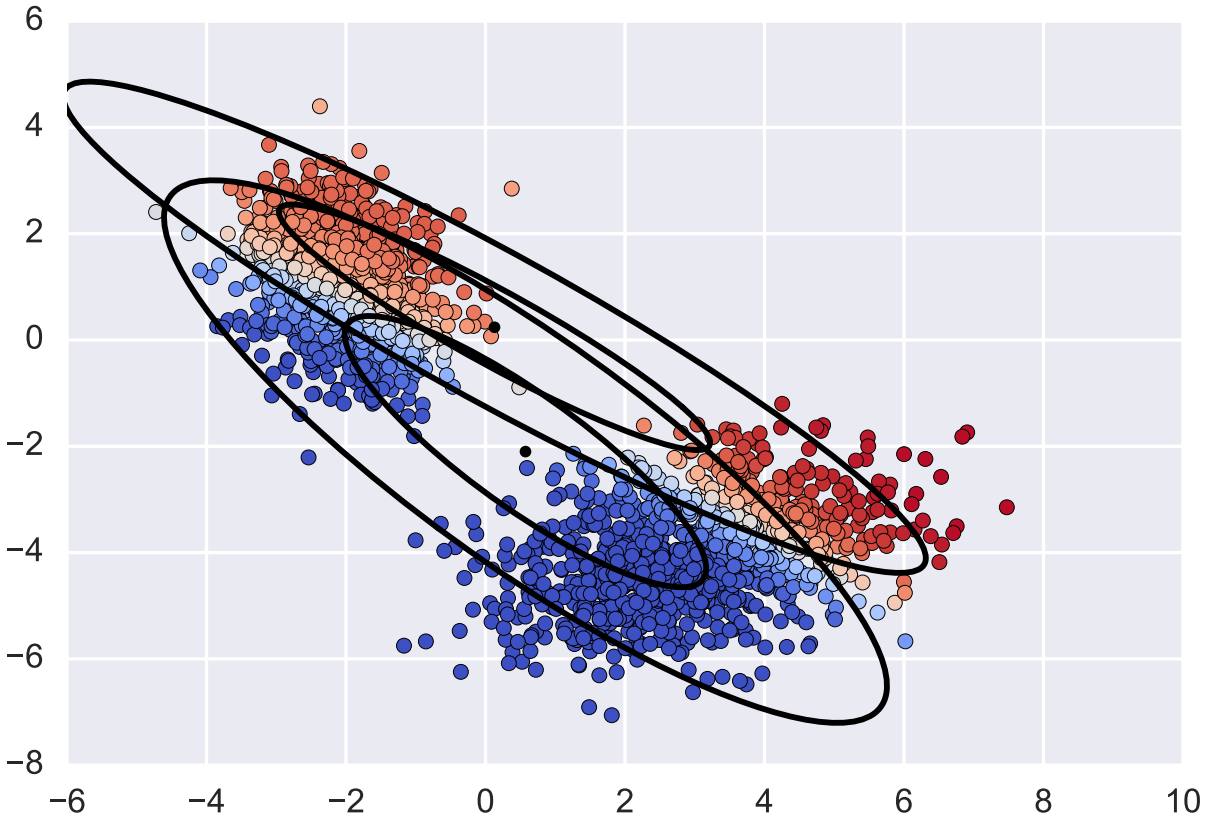
Illustration of EM algorithm



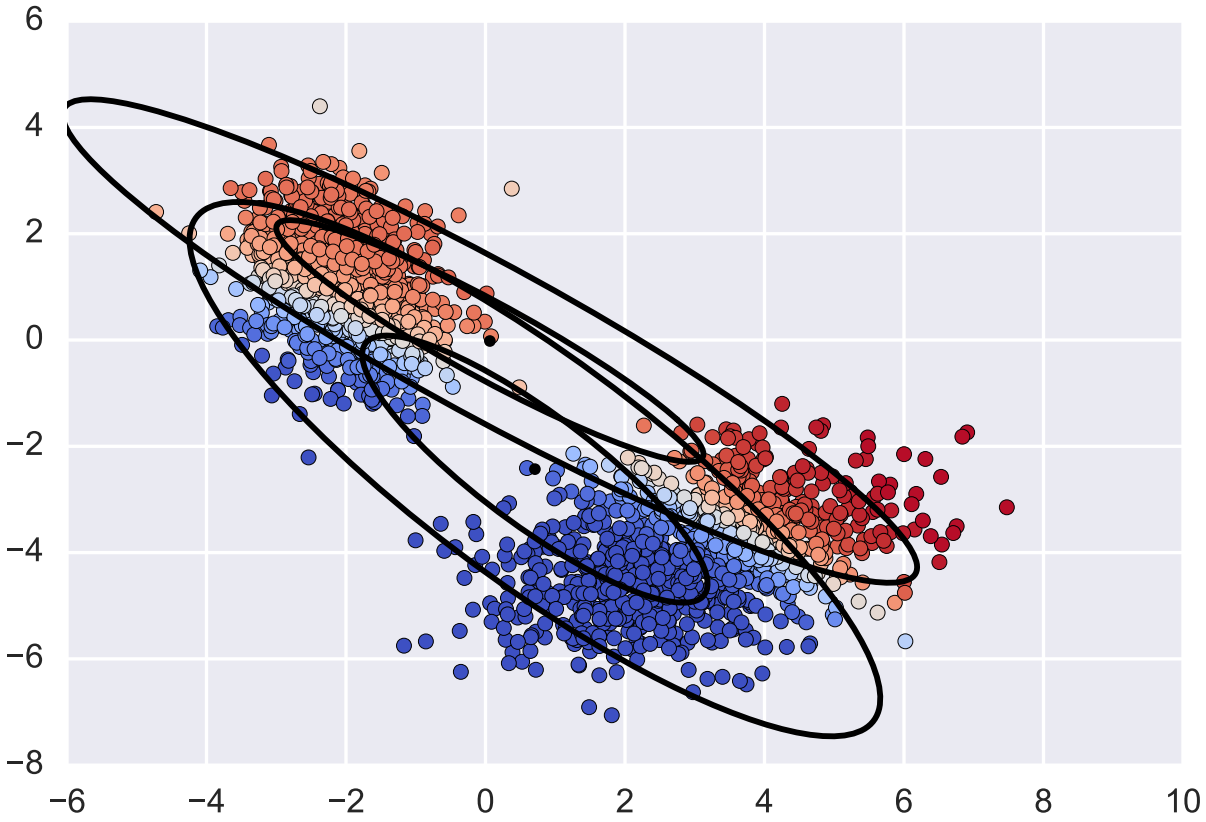
Possibility of local optima



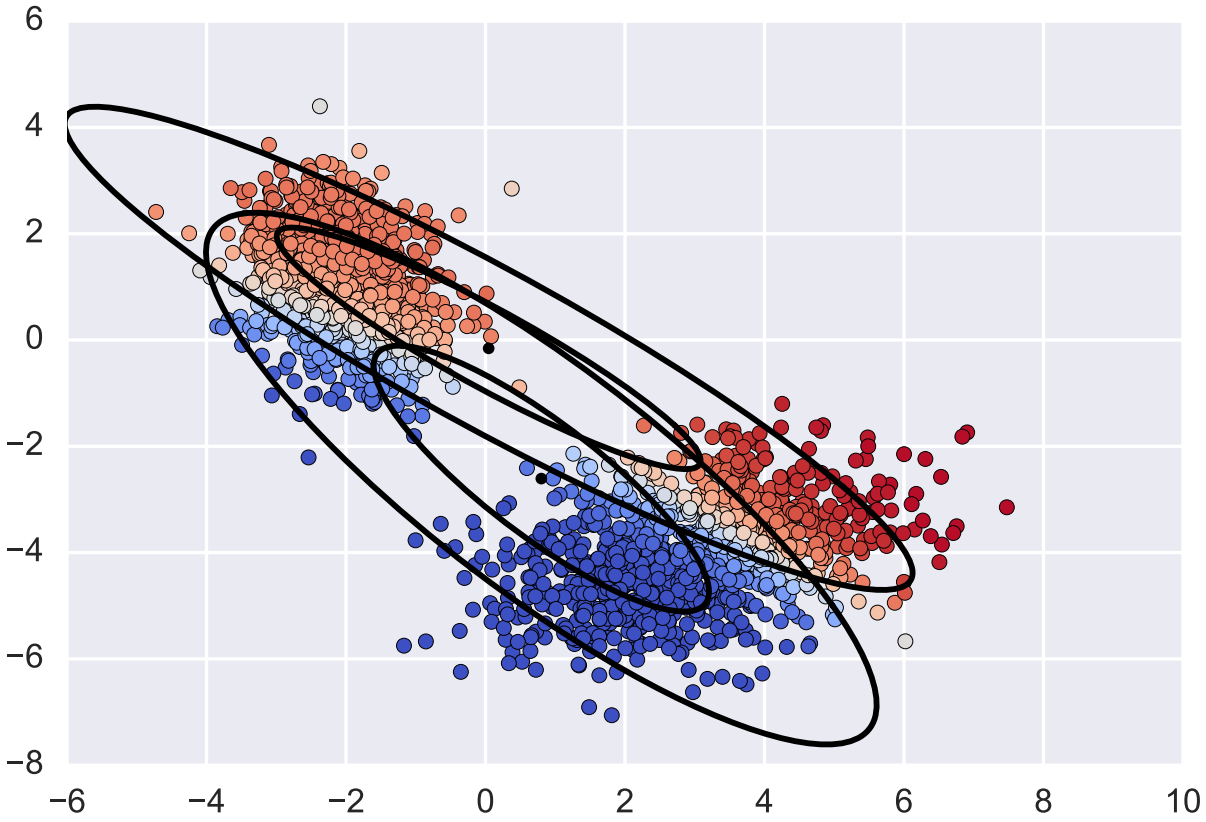
Possibility of local optima



Possibility of local optima



Possibility of local optima



Poll: outliers in mixture of Gaussians

Consider the following cartoon dataset:

If we fit a mixture of two Gaussians to this data via the EM algorithm, which group of points is likely to contain more “outliers” (points with the lowest $p(x)$)?

1. Left group
2. Right group
3. Equal chance of each, depending on initialization

EM and k-means

As you may have noticed, EM for mixture of Gaussians and k-means seem to be doing very similar things

Primary differences: EM is computing “distances” based upon the inverse covariance matrix, allows for “soft” assignments instead of hard assignments

15-388/688 - Practical Data Science: Matrices, vectors, and linear algebra

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Matrices and vectors

Basics of linear algebra

Libraries for matrices and vectors

Sparse matrices

Announcements

Tutorial instructions released today, (one-sentence) proposal due **9/27**

Homework 2 recitation tomorrow, 9/17, at 6pm in Hammerschlag Hall B103

Outline

Matrices and vectors

Basics of linear algebra

Libraries for matrices and vectors

Sparse matrices

Vectors

A vector is a 1D array of values

We use the notation $x \in \mathbb{R}^n$ to denote that x is an n-dimensional vector with real-valued entries

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We use the notation x_i to denote the *i*th entry of x

By default, we consider vectors to represent *column* vectors, if we want to consider a row vector, we use the notation x^T

Matrices

A matrix is a 2D array of values

We use the notation $A \in \mathbb{R}^{m \times n}$ to denote a real-valued matrix with m rows and n columns

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}$$

We use A_{ij} to denote the entry in row i and column j

Use the notation $A_{i:}$ to refer to row i, $A_{:j}$ to refer to column j (sometimes we'll use other notation, but we will define before doing so)

Matrices and linear algebra

Matrices are:

1. The “obvious” way to store tabular data (particularly numerical entries, though categorical data can be encoded too) in an efficient manner

2. The foundation of linear algebra, how we write down and operate upon (multi-variate) systems of linear equations

Understanding both these perspectives is critical for virtually all data science analysis algorithms

Matrices as tabular data

Given the “Grades” table from our relation data lecture

Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80
100	100	100

Natural to represent this data (ignoring primary key) as a matrix

$$A \in \mathbb{R}^{3\times 2} = \begin{bmatrix} 100 & 80 \\ 60 & 80 \\ 100 & 100 \end{bmatrix}$$

Row and column ordering

Matrices can be laid out in memory by row or by column

$$A = \begin{bmatrix} 100 & 80 \\ 60 & 80 \\ 100 & 100 \end{bmatrix}$$

Row major ordering: $100, 80, 60, 80, 100, 100$

Column major ordering: $100, 60, 100, 80, 80, 100$

Row major ordering is default for C 2D arrays (and default for Numpy), column major is default for FORTRAN (since a lot of numerical methods are written in FORTRAN, also the standard for most numerical code)

Higher dimensional matrices

From a data storage standpoint, it is easy to generalize 1D vector and 2D matrices to higher dimensional ND storage

“Higher dimensional matrices” are called *tensors*

There is also an extension or linear algebra to tensors, but be aware: most tensor use cases you see are *not* really talking about true tensors in the linear algebra sense

Outline

Matrices and vectors

Basics of linear algebra

Libraries for matrices and vectors

Sparse matrices

Systems of linear equations

Matrices and vectors also provide a way to express and analyze systems of linear equations

Consider two linear equations, two unknowns

$$\begin{array}{rcrcr} 4x_1 & - & 5x_2 & = & -13 \\ -2x_1 & + & 3x_2 & = & 9 \end{array}$$

We can write this using matrix notation as

$$Ax = b$$

$$A = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}, \qquad b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Basic matrix operations

For $A, B \in \mathbb{R}^{m \times n}$, matrix addition/subtraction is just the elementwise addition or subtraction of entries

$$C \in \mathbb{R}^{m \times n} = A + B \iff C_{ij} = A_{ij} + B_{ij}$$

For $A \in \mathbb{R}^{m \times n}$, transpose is an operator that "flips" rows and columns

$$C \in \mathbb{R}^{n \times m} = A^T \iff C_{ji} = A_{ij}$$

For $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}$ matrix multiplication is defined as

$$C \in \mathbb{R}^{m \times p} = AB \iff C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

- Matrix multiplication is associative ($A(BC) = (AB)C$), distributive ($A(B + C) = AB + AC$), *not commutative* ($AB \neq BA$)

Matrix inverse

The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones on diagonal and zeros elsewhere, has property that for $A \in \mathbb{R}^{m \times n}$

$$AI = IA = A \text{ (for different sized } I\text{)}$$

For a square matrix $A \in \mathbb{R}^{n \times n}$, matrix inverse $A^{-1} \in \mathbb{R}^{n \times n}$ is the matrix such that

$$AA^{-1} = I = A^{-1}A$$

Recall our previous system of linear equations $Ax = b$, solution is easily written using the inverse

$$x = A^{-1}b$$

Inverse need not exist for all matrices (conditions on linear independence of rows/columns of A), we will consider such possibilities later

Some miscellaneous definitions/properties

Transpose of matrix multiplication, $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times p}$

$$(AB)^T = B^T A^T$$

Inverse of product, $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times n}$ *both square and invertible*

$$(AB)^{-1} = B^{-1} A^{-1}$$

Inner product: for $x, y \in \mathbb{R}^n$, special case of matrix multiplication

$$x^T y \in \mathbb{R} = \sum_{i=1}^{n} x_i y_i$$

Vector norms: for $x \in \mathbb{R}^n$, we use $\|x\|_2$ to denote Euclidean norm

$$\|x\|_2 = (x^T x)^{\frac{1}{2}}$$

Poll: Valid linear algebra expressions

Assume $A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times m}, C \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n$ with $m > n$. Which of the following are valid linear algebra expressions?

1. $A + B$
2. $A + BC$
3. $(AB)^{-1}$
4. $(ABC)^{-1}$
5. CBx
6. $Ax + Cx$

Outline

Matrices and vectors

Basics of linear algebra

Libraries for matrices and vectors

Sparse matrices

Software for linear algebra

Linear algebra computations underlie virtually *all* machine learning and statistical algorithms

There have been *massive* efforts to write extremely fast linear algebra code: don't try to write it yourself!

Example: matrix multiply, for large matrices, specialized code will be ~10x faster than this "obvious" algorithm

```
void matmul(double **A, double **B, double **C, int m, int n, int p) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            C[i][j] = 0.0;
            for (int k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Numpy

In Python, the standard library for matrices, vectors, and linear algebra is Numpy

Numpy provides *both* a framework for storing tabular data as multidimensional arrays *and* linear algebra routines

Important note: numpy ndarrays are multi-dimensional arrays, *not* matrices and vectors (there are just routines that support them acting like matrices or vectors)

Specialized libraries

BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage) provide general interfaces for basic matrix multiplication (BLAS) and fancier linear algebra methods (LAPACK)

Highly optimized version of these libraries: ATLAS, OpenBLAS, Intel MKL

Anaconda typically uses a reasonably optimized version of Numpy that uses one of these libraries on the back end, but you should check

```
import numpy as np
print(np.__config__.show()) # print information on underlying
libraries
```

Creating Numpy arrays

Creating 1D and 2D arrays in Numpy

```
b = np.array([-13,9])              # 1D array construction
A = np.array([[4,-5], [-2,3]])     # 2D array contruction

b = np.ones(4)             # 1D array of ones
b = np.zeros(4)            # 1D array of zeros
b = np.random.randn(4)     # 1D array of random normal entries

A = np.ones((5,4))         # 2D array of all ones
A = np.zeros((5,4))        # 2D array of zeros
A = np.random.randn(5,4)   # 2D array with random normal entries

I = np.eye(5)                 # 2D identity matrix (2D array)
D = np.diag(np.random(5))     # 2D diagonal matrix (2D array)
```

Indexing into Numpy arrays

Arrays can be indexed by integers (to access specific element, row), or by slices, integer arrays, or Boolean arrays (to return subset of array)

```
A[0,0]   # select single entry
A[0,:]   # select entire column
A[0:3,1] # slice indexing

# integer indexing
idx_int = np.array([0,1,2])
A[idx_int,3]

# boolean indexing
idx_bool = np.array([True, True, True, False, False])
A[idx_bool,3]

# fancy indexing on two dimensions
idx_bool2 = np.array([True, False, True, True])
A[idx_bool, idx_bool2]     # not what you want
A[idx_bool,:][:,idx_bool2] # what you want
```

Basic operations on arrays

Arrays can be added/subtracted, multiply/divided, and transposed, but these are *not* the same as matrix operations

```
A = np.random.randn(5,4)
B = np.random.randn(5,4)
x = np.random.randn(4)
y = np.random.randn(5)

A+B            # matrix addition
A-B            # matrix subtraction

A*B            # ELEMENTWISE multiplication
A/B            # ELEMENTWISE division
A*x            # multiply columns by x
A*y[:,None]    # multiply rows by y (look this one up)

A.T            # transpose (just changes row/column ordering)
x.T            # does nothing (can't transpose 1D array)
```

Basic matrix operations

Matrix multiplication done using the .dot() function or @ operator, special meaning for multiplying 1D-1D, 1D-2D, 2D-1D, 2D-2D arrays

```
A = np.random.randn(5,4)
C = np.random.randn(4,3)
x = np.random.randn(4)
y = np.random.randn(5)
z = np.random.randn(4)

A @ C        # matrix-matrix multiply (returns 2D array)
A @ x        # matrix-vector multiply (returns 1D array)
x @ z        # inner product (scalar)

A.T @ y      # matrix-vector multiply
y.T @ A      # same as above
y @ A        # same as above
#A @ y       # would throw error
```

There is also an np.matrix class … don't use it

Solving linear systems

Methods for inverting a matrix, solving linear systems

```
b = np.array([-13,9])
A = np.array([[4,-5], [-2,3]])

np.linalg.inv(A)       # explicitly form inverse
np.linalg.solve(A, b) # A^(-1)*b, more efficient and numerically stable
```

Important, always prefer to *solve a linear system* over directly forming the inverse and multiplying (more stable and cheaper computationally)

Details: solution methods use a factorization (e.g., LU factorization), which is cheaper than forming inverse

Complexity of operations

Assume $A, B \in \mathbb{R}^{n \times n}, x, y \in \mathbb{R}^n$

Matrix-matrix product AB: $O(n^3)$

Matrix-vector product Ax: $O(n^2)$

Vector-vector inner product $x^T y$: $O(n)$

Matrix inverse/solve: A^{-1}, $A^{-1}y$: $O(n^3)$

Important: Be careful about order of operations, $(AB)x = A(Bx)$ but the left one is $O(n^3)$ right is $O(n^2)$

Outline

Matrices and vectors

Basics of linear algebra

Libraries for matrices and vectors

Sparse matrices

Sparse matrices

Many matrices are sparse (contain mostly zero entries, with only a few non-zero entries)

Examples: matrices formed by real-world graphs, document-word count matrices (more on both of these later)

Storing all these zeros in a standard matrix format can be a huge waste of computation and memory

Sparse matrix libraries provide an efficient means for handling these sparse matrices, storing and operating only on non-zero entries

- Note: this is important from the first (storage-based) perspective of matrices, the linear algebra is the same (mostly)

Coordinate format

There are several different ways of storing sparse matrices, each optimized for different operations

Coordinate (COO) format: store each entry as a tuple

$$(\text{row_index}, \text{col_index}, \text{value})$$

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{aligned} \text{data} &= [2\ 4\ 1\ 3\ 1\ 1] \\ \text{row_indices} &= [1\ 3\ 2\ 0\ 3\ 1] \\ \text{col_indices} &= [0\ 0\ 1\ 2\ 2\ 3] \end{aligned}$$

Important: these could be placed in any order

A good format for constructing sparse matrices

Compressed sparse column format

Compressed sparse column (CSC) format

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{data} = [2\ 4\ 1\ 3\ 1\ 1]$$
$$\text{row_indices} = [1\ 3\ 2\ 0\ 3\ 1]$$
$$\cancel{\text{col_indices} = [0\ 0\ 1\ 2\ 2\ 3]}$$
$$\Downarrow$$
$$\text{col_indices} = [0\ 2\ 3\ 5\ 6]$$

Ordering is important (always column-major ordering)

Faster for matrix multiplication, easier to access individual columns

Very bad for modifying a matrix, to add one entry need to shift all data

Sparse matrix libraries

Need specialized libraries for handling matrix operations (multiplication/solving equations) for sparse matrices

General rule of thumb (very adhoc): if your data is 80% sparse or more, it's probably worthwhile to use sparse matrices for multiplication, if it's 95% sparse or more, probably worthwhile for solving linear systems)

The `scipy.sparse` module provides routines for constructing sparse matrices in different formats, converting between them, and matrix operations

```
import scipy.sparse as sp
A = sp.coo_matrix((data, (row_idx, col_idx)), size)
B = A.tocsc()
C = A.todense()
```

15-388/688 - Practical Data Science: Unsupervised learning

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Unsupervised learning

K-means

Pricinpal Component Analysis

Outline

Unsupervised learning

K-means

Pricinpal Component Analysis

Supervised learning paradigm

Training Data

$$(x^{(1)}, y^{(1)})$$
$$(x^{(2)}, y^{(2)})$$
$$(x^{(3)}, y^{(3)})$$
$$\vdots$$

$\longrightarrow$

Machine learning algorithm

Hypothesis function

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

$\longrightarrow$

Predictions

New example x

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

Unsupervised learning paradigm

Training Data

$$\left(x^{(1)}\right)$$
$$\left(x^{(2)}\right)$$
$$\left(x^{(3)}\right)$$
$$\vdots$$

$\longrightarrow$

Machine learning algorithm

Hypothesis function

$$? \approx h\left(x^{(i)}\right)$$

$\longrightarrow$

Predictions

New example x

$$? = h(x)$$

Three elements of unsupervised learning

It turns out the virtually all unsupervised learning algorithms can be considered in the same manner as supervised learning:

1. Define hypothesis function

2. Define loss function

3. Define how to optimize the loss function

But, what do a hypothesis function and loss function signify in the unsupervised setting?

Unsupervised learning framework

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

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

Hypothesis function: $h_\theta \colon \mathbb{R}^n \to \mathbb{R}^n$, approximates *input* given input, i.e. we want $x^{(i)} \approx h_\theta(x^{(i)})$

Loss function: $\ell \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$, measures the difference between a hypothesis and actual input, e.g.: $\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$

Similar canonical machine learning optimization as before:

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

Hypothesis and loss functions

The framework seems odd, what does it mean to have a hypothesis function approximate the input?

Can't we just pick $h_\theta(x) = x$?

The goal of unsupervised learning is to pick some *restricted* class of hypothesis functions that extract some kind of structure from the data (i.e., one that does not include the identity mapping above)

In this lecture, we'll consider two different algorithms that both fit the framework: k-means and principal component analysis

Outline

Unsupervised learning

K-means

Pricinpal Component Analysis

K-means graphically

The k-means algorithm is easy to visualize: given some collection of data points we want to find k centers such that all points are close to at least one center

K-means in unsupervised framework

Parameters of k-means are the choice of centers $\theta = \{\mu^{(1)}, \dots \mu^{(k)}\}$, with $\mu^{(i)} \in \mathbb{R}^n$

Hypothesis function outputs the center closest to a point x

$$h_\theta(x) = \underset{\mu \in \{\mu^{(1)}, \dots \mu^{(k)}\}}{\operatorname{argmin}} \|\mu - x\|_2^2$$

Loss function is squared error between input and hypothesis

$$\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

Optimization problem is thus

$$\underset{\mu^{(1)}, \dots \mu^{(k)}}{\text{minimize}} \sum_{i=1}^{m} \left\| h_\theta(x^{(i)}) - x^{(i)} \right\|_2^2$$

Optimizing k-means objective

The k-means objective is non-convex (possibility of local optima), and does not have a closed form solution, so we resort to an approximate method, by repeating the following (Lloyd's algorithm, or just "k-means")

1. Assign points to nearest cluster
2. Compute cluster center as mean of all points assigned to it

Given: Data set $\left(x^{(i)}\right)_{i=1,\ldots,m}$, # clusters k

Initialize:

$$\mu^{(j)} \leftarrow \mathrm{Random}\left(x^{(i)}\right),\ j = 1, \ldots, k$$

Repeat until convergence:

Compute cluster assignment:

$$y^{(i)} = \underset{j}{\mathrm{argmin}} \left\|\mu^{(j)} - x^{(i)}\right\|_2^2,\ i = 1, \ldots, m$$

Re-compute means:

$$\mu^{(j)} \leftarrow \mathrm{Mean}\left(\left\{x^{(i)} | y^{(i)} = j\right\}\right), j = 1, \ldots, k$$

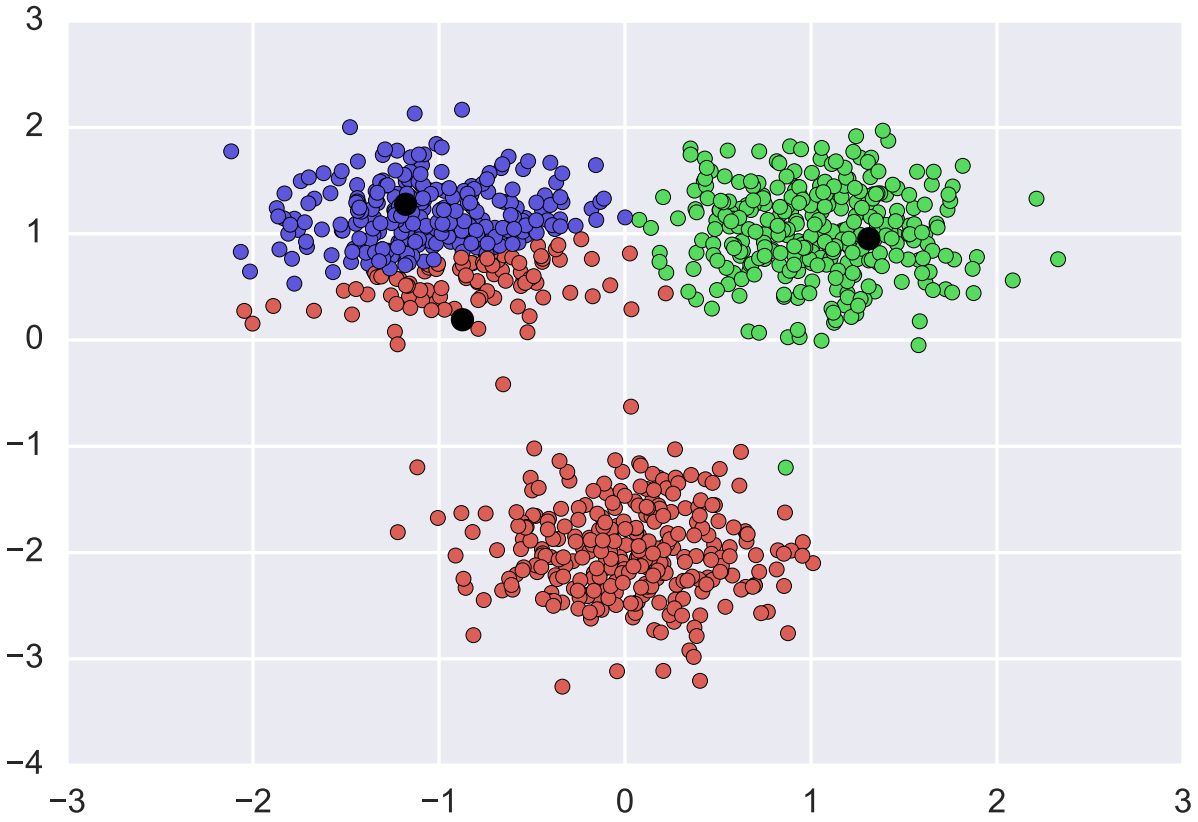
K-means in a few lines of code

Scikit-learn, etc, contains k-means implementations, but again these are pretty easy to write

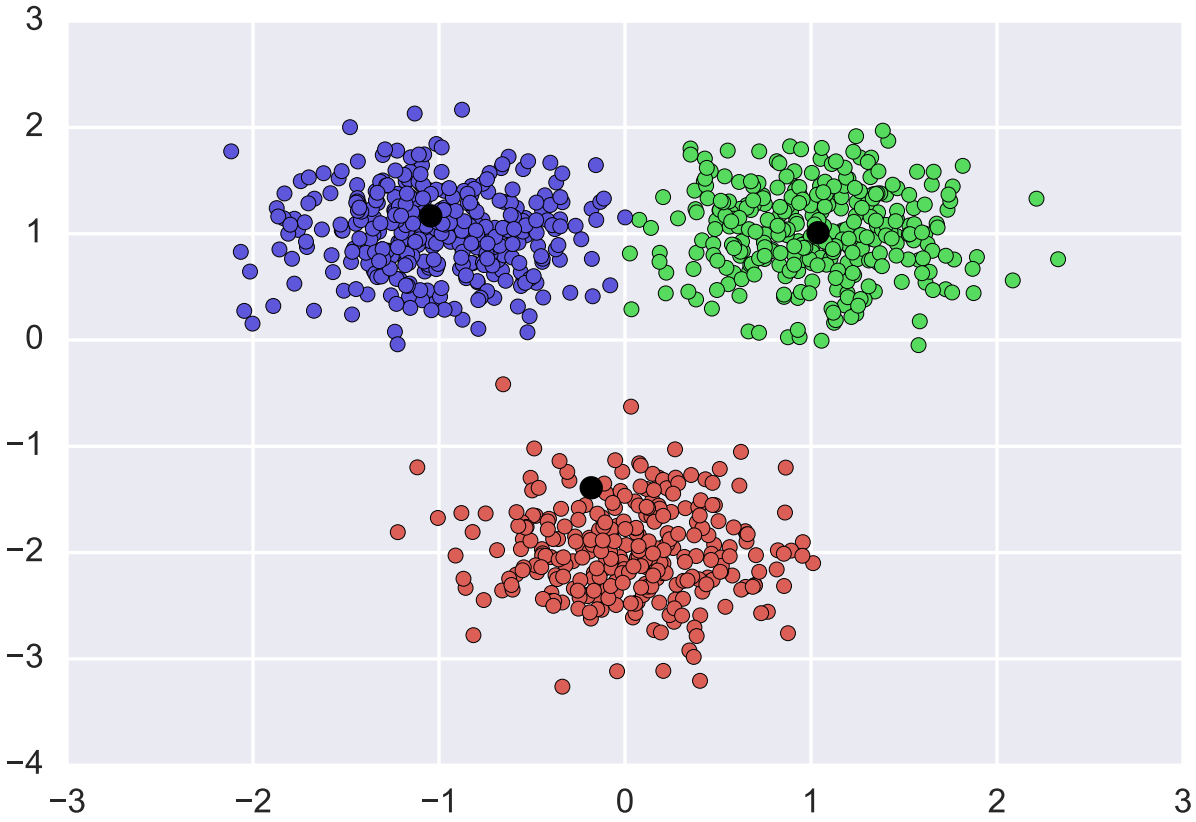
For better implementation, want to check for convergence as well as max number of iterations

```
def kmeans(X, k, max_iter=10):
    Mu = X[np.random.choice(X.shape[0],k),:]
    for i in range(max_iter):
        D = -2*X@Mu.T + (X**2).sum(axis=1)[:,None] + (Mu**2).sum(axis=1)
        C = np.eye(k)[np.argmin(D,axis=1),:]
        Mu = (C.T @ X)/np.sum(C,axis=0)[:,None]
    loss = np.linalg.norm(X - Mu[np.argmin(D,axis=1),:])**2
    return Mu, C, loss
```

Convergence of k-means



Convergence of k-means



Convergence of k-means

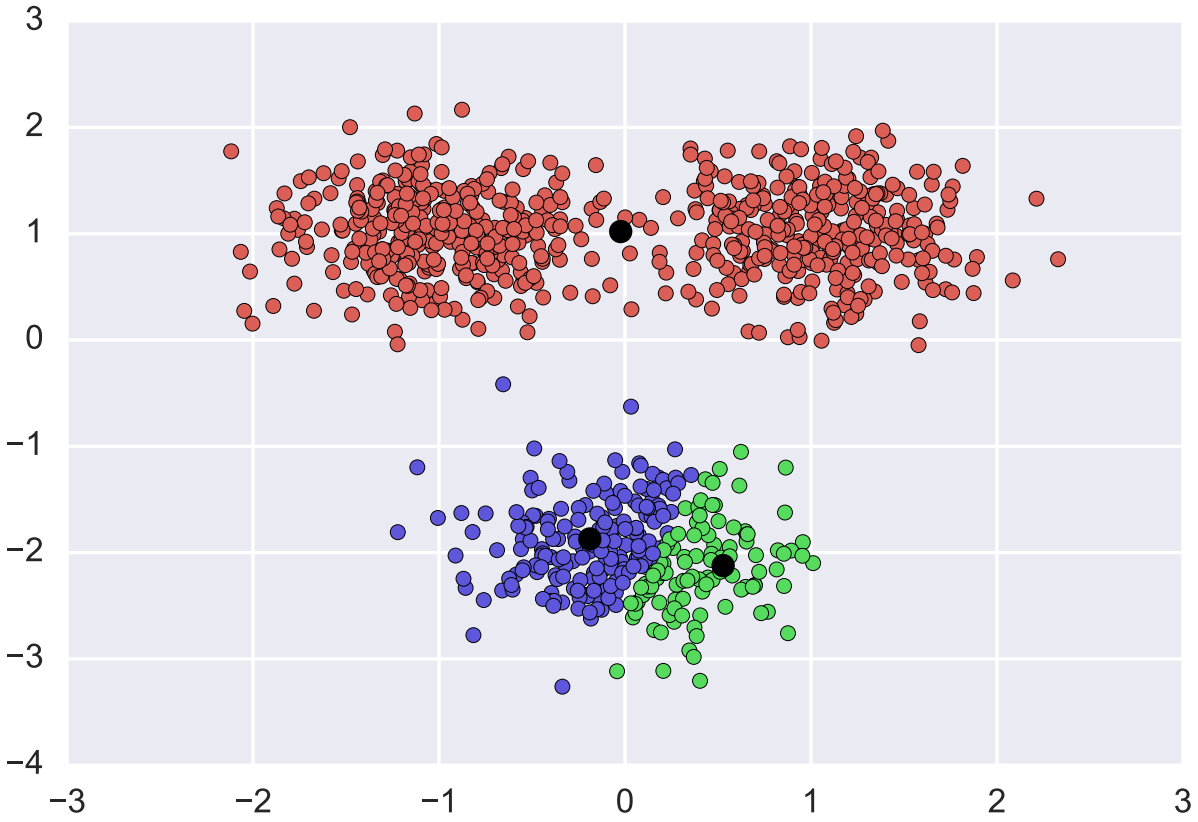
Possibility of local optima

Since the k-means objective function has local optima, there is the chance that we convert to a less-than-ideal local optima

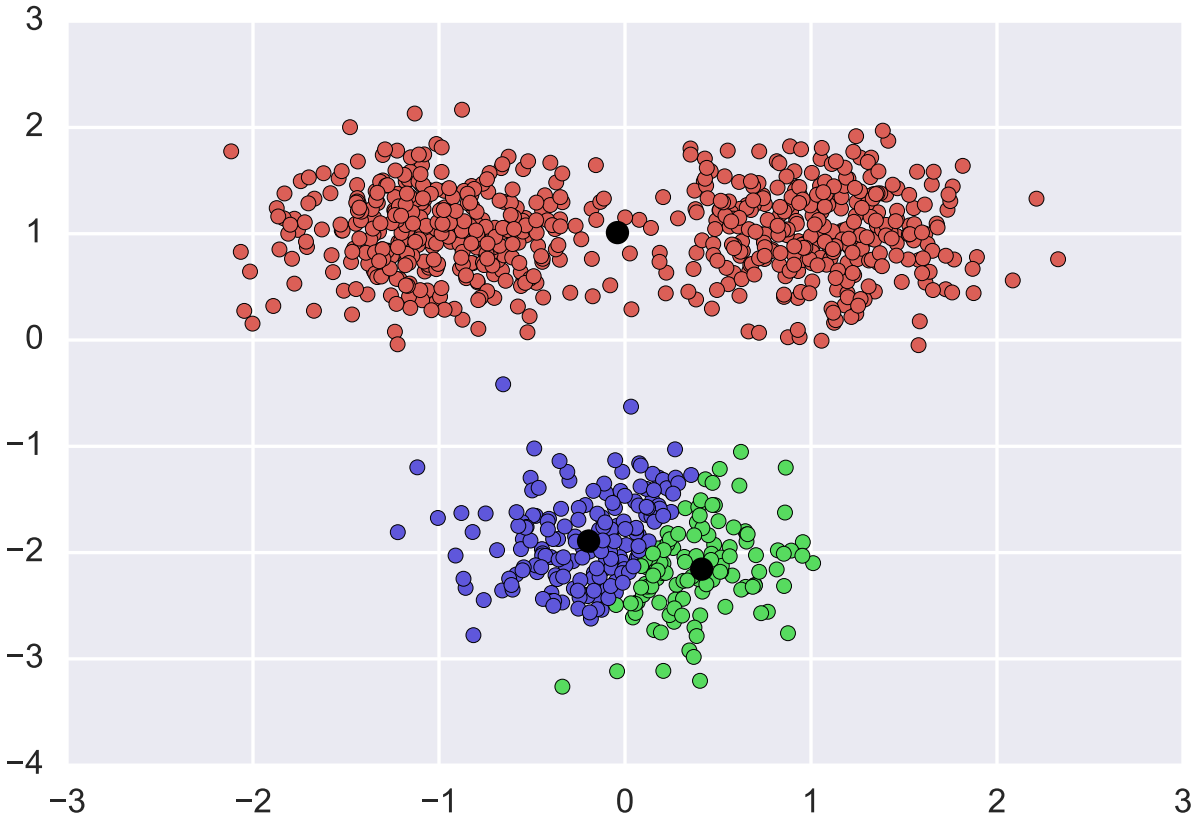
Especially for large/high-dimensional datasets, this is not hypothetical: k-means will usually converge to a different local optima depending on its starting point

Convergence of k-means (bad)

Convergence of k-means (bad)



Convergence of k-means (bad)



Addressing poor clusters

Many approaches to address potential poor clustering: e.g. randomly initialize many times, take clustering with lowest loss

A common heuristic, k-means++: when initializing means, don't select $\mu^{(i)}$ randomly from all clusters, instead choose $\mu^{(i)}$ sequentially, sampled with probability proportion to the minimum squared distance to all other centroids

After these centers are initialized, run k-means as normal

K-means++

Given: Data set $\left(x^{(i)}\right)_{i=1,\ldots,m}$, # clusters k

Initialize:

$$\mu^{(1)} \leftarrow \mathrm{Random}\left(x^{(1:m)}\right)$$

For $j = 2, \ldots, k$:

Select new cluster:

$$\mu^{(j)} \leftarrow \mathrm{Random}\left(x^{(1:m)}, p^{(1:m)}\right)$$

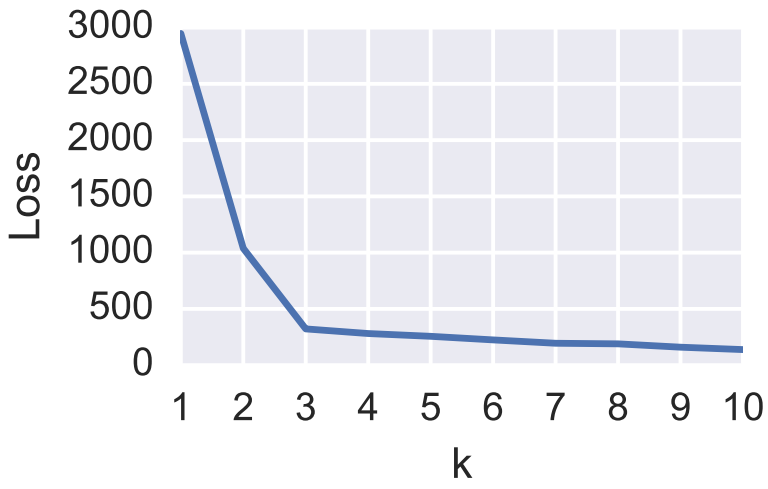
where probabilities $p^{(i)}$ given by

$$p^{(i)} \propto \min_{j'<j} \left\|\mu^{(j')} - x^{(i)}\right\|_2^2$$

How to select k?

There's no "right" way to select k (number of clusters): larger k virtually always will have lower loss than smaller k, even on a hold out set

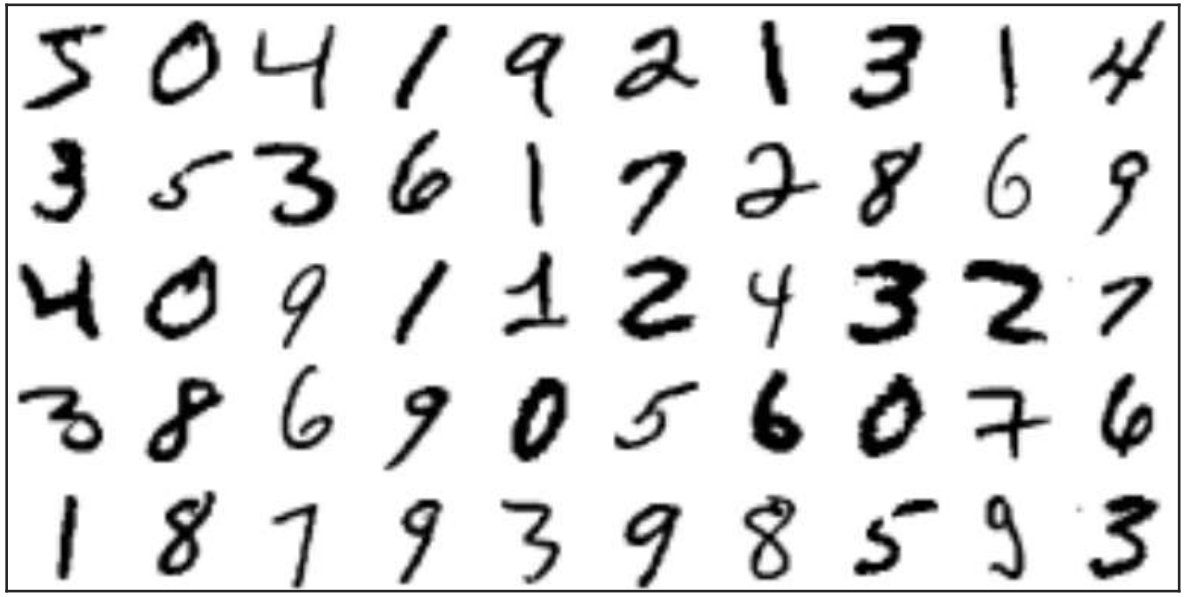
Instead, it's common to look at the loss function as a function of increasing k, and stop when things look "good" (lots of other heuristics, but they don't convincingly outperform this)



Example on real data

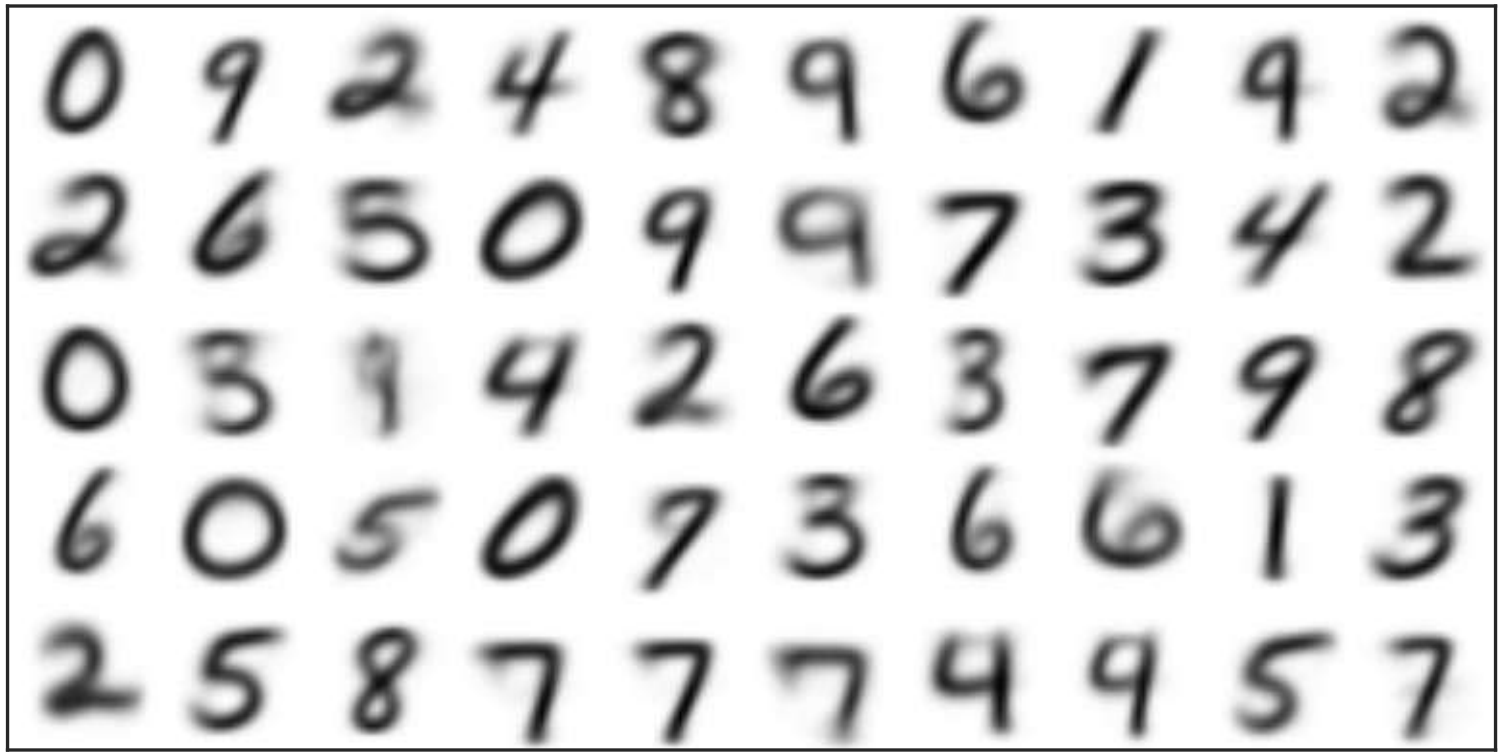
MNIST digit classification data set (used in question for 688 HW4)

60,000 images of digits, each 28x28



K-means run on MNIST

Means for k-means run with k=50 on MNIST data



Outline

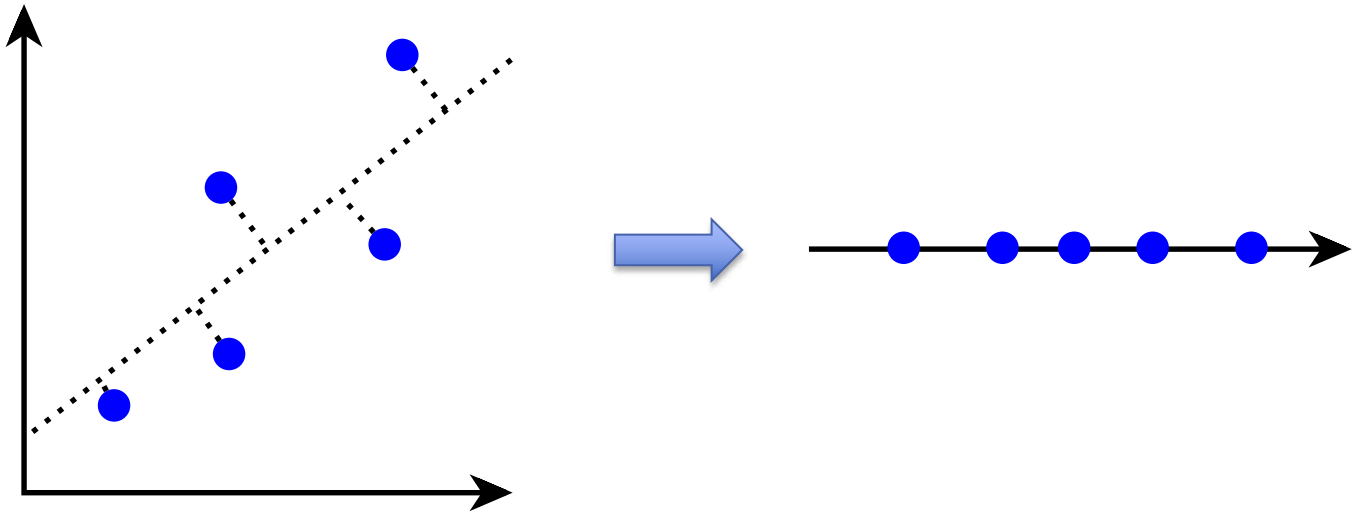
Unsupervised learning

K-means

Pricinpal Component Analysis

Principal component analysis graphically

Principal component analysis (PCA) looks at “simplifying” the data in another manner, by preserving the axes of major variation in the data



PCA in unsupervised setting

We'll assume our data is normalized (each feature has zero mean, unit variance, otherwise normalize it)

Hypothesis function:

$$h_\theta(x) = UWx, \theta = \{U \in \mathbb{R}^{n \times k}, W \in \mathbb{R}^{k \times n}\}$$

i.e., we are "compressing" input by multiplying by a *low rank* matrix

Loss function: same as for k-means, squared distance

$$\ell(h_\theta(x), x) = \|h_\theta(x) - x\|_2^2$$

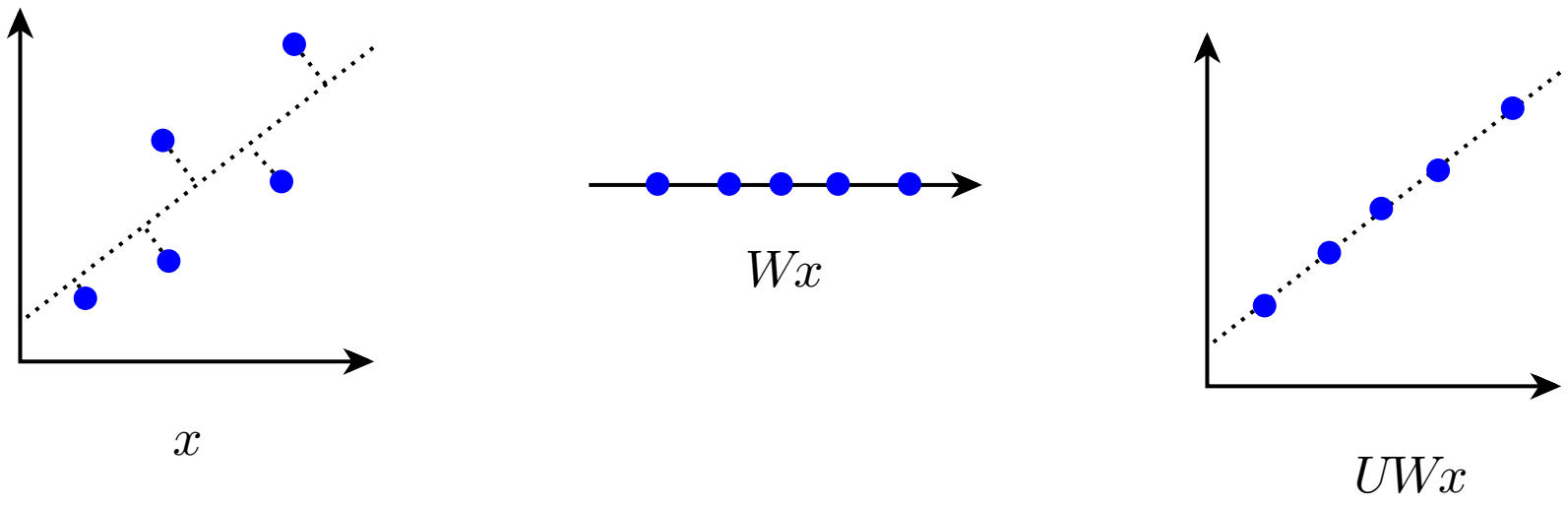
Optimization problem:

$$\underset{U,W}{\text{minimize}} \sum_{i=1}^{m} \|UWx^{(i)} - x^{(i)}\|_2^2$$

Dimensionality reduction with PCA

One of the standard uses for PCA is to reduce the dimension of the input data (indeed, we motivated it this way)

If $h_\theta(x) = UWx$, then $Wx \in \mathbb{R}^k$ is a "reduced" representation of x



Solving PCA optimization problem

The PCA optimization problem is also not convex, subject to local optima if we use e.g. gradient descent

However, amazingly, we *can* solve this problem exactly using what is called a singular value decomposition (all stated without proof)

Given: normalized data matrix X, # of components k

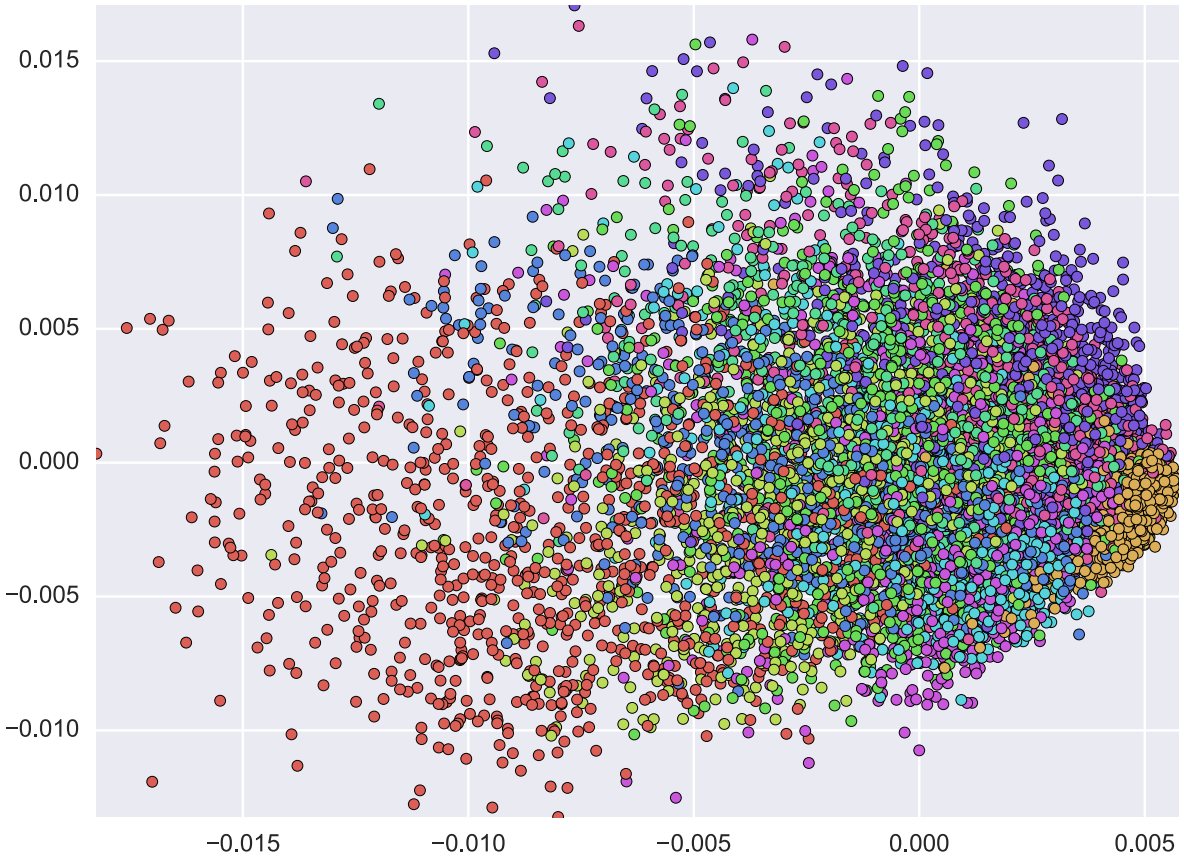
1. Compute singular value decomposition $USV^T = X$ where U, V is orthogonal and S is diagonal matrix of singular values
2. Return $U = V_{:,1:k} S^{-1}_{1:k,1:k}$, $W = V^T_{:,1:k}$
3. Loss given by $\sum_{i=k+1}^{n} S_{ii}^2$

Code to solve PCA

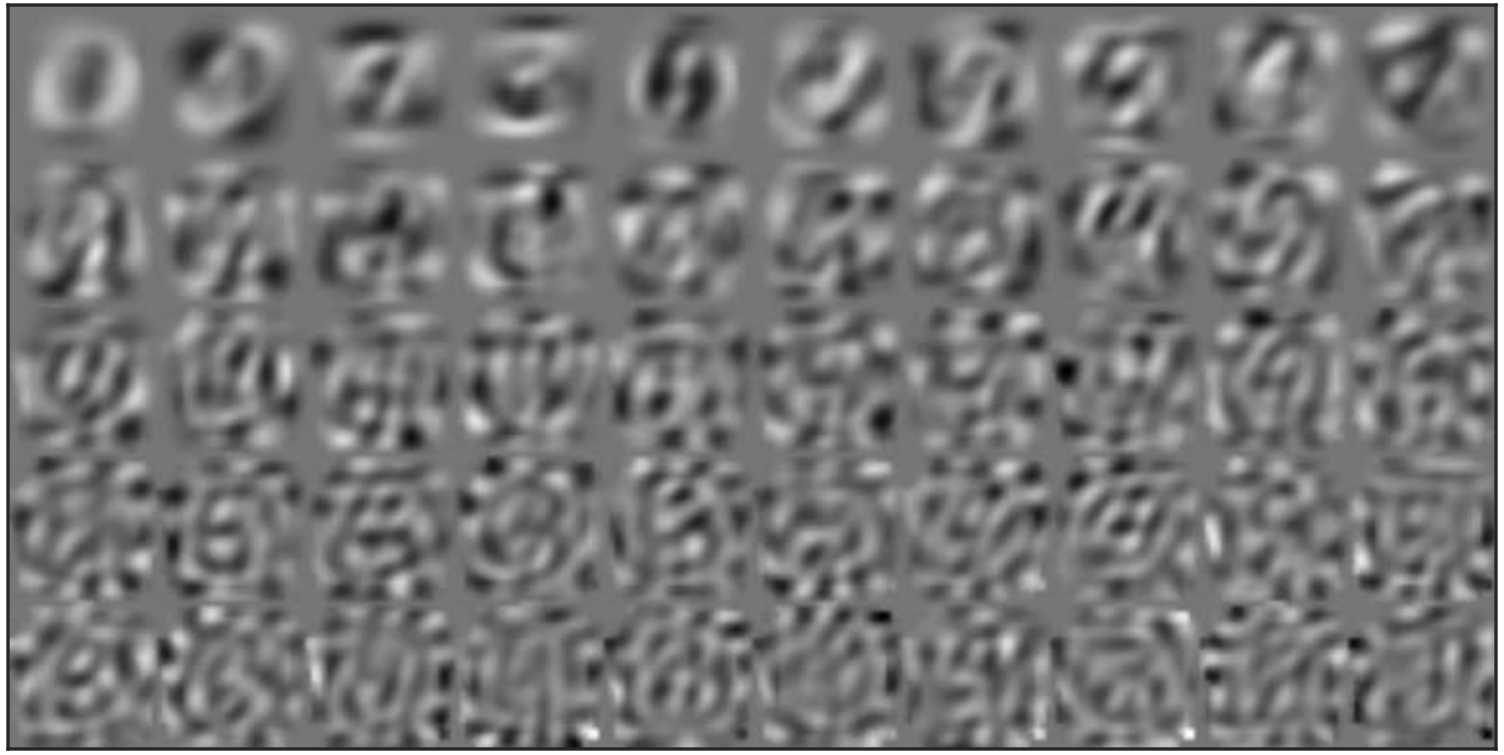
PCA is just a few lines of code, but all the actual interesting elements are the SVD call, if you’re not familiar with this (which is fine), then there won’t be too much insight

```
def pca(X,k):
    X0 = (X - np.mean(X, axis=0)) / np.std(X,axis=0) + 1e-8)
    U,s,VT = np.linalg.svd(X0, compute_uv=True, full_matrices=False)
    loss = np.sum(s[k:]**2)
    return VT.T[:,:k]/s[:k], VT.T[:,:k], loss
```

Dimensionality reduction on MNIST

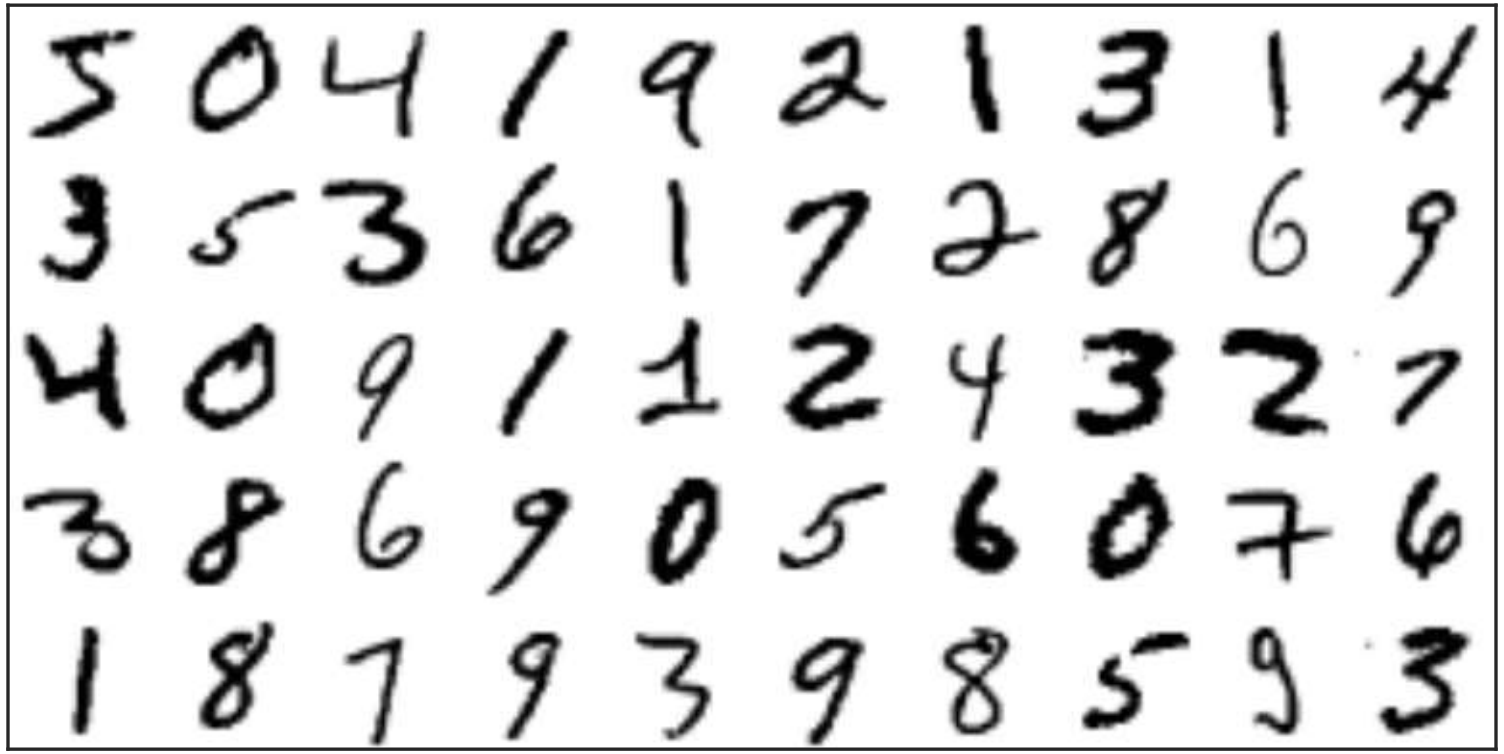


Top 50 principal components for MNIST

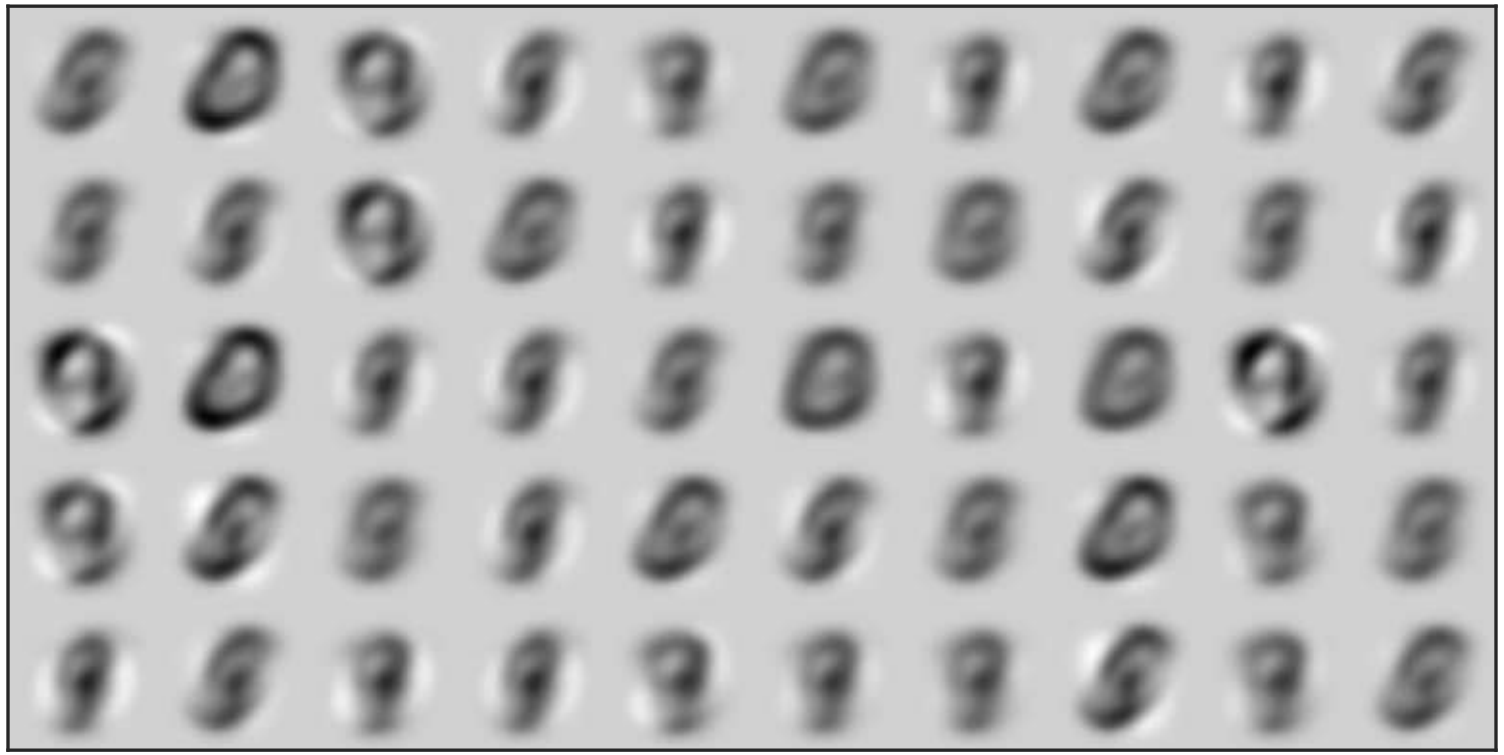


Images are reconstructed as linear combination of principal components

Reconstructed images, original



Reconstructed images, k=2



Reconstructed images, k=10

Reconstructed images, k=100

K-means and PCA in data preparation

Although useful in their own right as unsupervised algorithms, K-means and PCA are also useful in data preparation for supervised learning

Dimensionality reduction with PCA:

- Run PCA, get W matrix
- Transform inputs to be $\tilde{x}^{(i)} = Wx^{(i)}$

Radial basis functions with k-means

- Run k-means to extract k centers, $\mu^{(1)}, \ldots, \mu^{(k)}$
- Create radial basis function features $\phi_j^{(i)} = \exp\left(-\frac{\|x^{(i)} - \mu^{(j)}\|_2^2}{2\sigma^2}\right)$

15-388/688 - Practical Data Science: Relational Data

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Overview of relational data

Entity relationships

Pandas and SQLite

Joins

Outline

Overview of relational data

Entity relationships

Pandas and SQLite

Joins

The basic relation (i.e. the table)

The term technical term “relation” can be interchanged with the standard notion we have of “tabular data,” say an instance of a “Person” relation

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Manek	Gaurav	TA
3	Rice	Leslie	TA
4	Peres	Filipe	TA
5	Gates	William	Student
6	Musk	Elon	Student

The basic relation (i.e. the table)

The term technical term “relation” can be interchanged with the standard notion we have of “tabular data,” say an instance of a “Person” relation

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Manek	Gaurav	TA
3	Rice	Leslie	TA
4	Peres	Filipe	TA
5	Gates	William	Student
6	Musk	Elon	Student

Rows are called *tuples (or records),* represent a single instance of this relation, and *must be unique*

The basic relation (i.e. the table)

The term technical term “relation” can be interchanged with the standard notion we have of “tabular data,” say an instance of a “Person” relation

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Manek	Gaurav	TA
3	Rice	Leslie	TA
4	Peres	Filipe	TA
5	Gates	William	Student
6	Musk	Elon	Student

Columns are called *attributes*, specify some element contained by each of the tuples

Multiple tables and relations

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

Primary keys

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

Primary key: *unique* ID for every tuple in a relation (i.e. every row in the table), each relation must have exactly one primary key

Foreign keys

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

A **foreign key** is an attribute that points to the primary key of *another* relation

If you delete a primary key, need to delete all foreign keys pointing to it

Indexes (not indices)

Indexes are created as ways to “quickly” access elements of a table

For example, consider finding people with last name “Gates”: no option but just scan through the whole dataset: $O(n)$ operation

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Indexes

Person

Location	ID	Last Name	First Name	Role ID
0	1	Kolter	Zico	1
100	2	Manek	Gaurav	2
200	3	Rice	Leslie	2
300	4	Peres	Filipe	2
400	5	Gates	William	3
500	6	Musk	Elon	3

Last Name Index

Last Name	Location
Gates	400
Kolter	0
Manek	100
Musk	500
Peres	300
Rice	200

Think of an index as a separate *sorted* table containing the indexed column and the tuple location: searching for value takes $O(\log n)$ time

In practice, use data structure like a B-tree or several others

Indexes

The primary key always has an index associated with it (so you can think of primary keys themselves as always being a fast way to access data)

Indexes don’t have to be on a single column, can have an index over multiple columns (with some ordering)

Outline

Overview of relational data

Entity relationships

Pandas and SQLite

Joins

Entity relationships

Several types of inter-table relationships

1. One-to-one
2. (One-to-zero/one)
3. One-to-many (and many-to-one)
4. Many-to-many

These relate one (or more) rows in a table with one (or more) rows in another table, via a foreign key

Note that these relationships are really between the “entities” that the tables represent, but we won’t formalize this beyond the basic intuition

One-to-many relationship

We have already seen a one-to-many relationship: one **role** can be shared by many **people**, denoted as follows

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

One-to-one relationships

In a true one-to-one relationship spanning multiple tables, each row in a table has *exactly* one row in another table

Not very common to break these across multiple tables, as you may as well just add another attribute to an existing table, but it is possible

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2

⋮

Andrew ID

Person ID	Andrew ID
1	zkolter
2	gmanek

⋮

One-to-zero/one relationships

More common in databases is to find “one-to-zero/one” relationships broken across multiple tables

Consider adding a “Grades” table to our database: each person can have at most one tuple in the grades table

Grades

Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80

Bars and circles denote “mandatory” versus “option” relationships (we won’t worry about these, just know that there is notation for them)

Many-to-many relationships

Creating a grades table as done before is a bit cumbersome, because we need to keep adding columns to the table, null entries if someone doesn't do the homework

Alternatively, consider adding two tables, a "homework" table that represents information about each homework, and an associative table that links homeworks to people

Homework

ID	Name	388 Points	688 Points
1	HW 1	65	35
2	HW 2	75	25

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Associative tables

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

What is the primary key of this table? What are foreign keys?

Which indexes would you want to create on this table?

Poll: Primary key of associative table

What should the primary key be for this table?

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

1. Person ID
2. HW ID
3. (Person ID, HW ID)
4. (Person ID, HW ID, Score)

Many-to-many relationships

Setups like this encode many-to-many relationships: each person can have multiple homeworks, and each homework can be done by multiple people

We could also write this in terms of relationships specified by the associative table, but this is not really correct, as it is mixing up the underlying relationships with how they are stored in a database

Outline

Overview of relational data

Entity relationships

Pandas and SQLite

Joins

Pandas

Pandas is a “Data Frame” library in Python, meant for manipulating in-memory data with row and column labels (as opposed to, e.g., matrices, that have no row or column labels)

Pandas is *not* a relational database system, but it contains functions that mirror some functionality of relational databases

We’re going to cover Pandas in more detail in other portions of the class, but just discuss basic functionality for now

Pandas examples

Create a DataFrame with our Person example:

```
import pandas as pd

df = pd.DataFrame([(1, 'Kolter', 'Zico'),
                   (2, 'Manek', 'Gaurav'),
                   (3, 'Rice', 'Leslie'),
                   (4, 'Peres', 'Filipe'),
                   (5, 'Gates', 'Bill'),
                   (6, 'Musk', 'Elon')],
                  columns=["Person ID", "Last Name", "First Name"])
df.set_index("Person ID", inplace=True)
```

```
In [4]: df
```

Out[4]:

Person ID	Last Name	First Name
1	Kolter	Zico
2	Manek	Gaurav
3	Rice	Leslie
4	Peres	Filipe
5	Gates	Bill
6	Musk	Elon

Some important notes

As mentioned, Pandas is *not* a relational data system, in particular it has no notion of primary keys (but it does have indexes)

Operations in Pandas are typically *not* in place (that is, they return a new modified DataFrame, rather than modifying an existing one)

- Use the “inplace” flag to make them done in place

If you select a single row or column in a Pandas DataFrame, this will return a “Series” object, which is like a one-dimensional DataFrame (it has only an index and corresponding values, not multiple columns)

Some common Pandas commands

```
# read CSV file into DataFrame
df = pd.read_csv(filename)

# get first five rows of DataFrame
df.head()

# index into a dataframe
# df.loc[rows, columns] and df.iloc[row numbers, column numbers]
df.loc[:, "Last Name"]              # Series of all last names
df.loc[:, ["Last Name"]]            # DataFrame with one column
df.loc[[1,2], :]                    # DataFrame with only ids 1,2
df.loc[1,"Last Name"] = "Kilter"    # Set an entry in a DataFrame
df.loc[7,:] = ("Moore", "Andrew")   # Add a new entry with index=7
df.iloc[0,0]                        # Index rows and columns by zero-index
```

We're going to cover more next lecture in conjunction with visualization

SQLite

An actual relational database management system (RDBMS)

Unlike most systems, it is a *serverless* model, applications directly connect to a file

Allows for simultaneous connections from many applications to the same database file (but not quite as much concurrency as client-server systems)

All operations in SQLite will use SQL (Structured Query Language) command issued to the database object

You can enforce foreign keys in SQLite, but we won't bother

Creating a database and table

You can create a database and connect using this boilerplate code:

```
import sqlite3
conn = sqlite3.connect("people.db")
cursor = conn.cursor()

# do your stuff

conn.close()
```

Create a new table:

```
cursor.execute("""
CREATE TABLE role (
    id INTEGER PRIMARY KEY,
    name TEXT
)""")
```

Creating a new table and inserting data

Insert data into the table:

```
cursor.execute("INSERT INTO role VALUES (1, 'Instructor')")
cursor.execute("INSERT INTO role VALUES (2, 'TA')")
cursor.execute("INSERT INTO role VALUES (3, 'Student')")
conn.commit()
```

Delete items from a table:

```
cursor.execute("DELETE FROM role WHERE id == 3")
conn.commit()
```

Note: if you don't call commit, you can undo with `conn.rollback()`

Querying all data from a table

Read all the rows from a table:

```
for row in cursor.execute('SELECT * FROM role'):
    print row
```

Read table directly into a Pandas DataFrame:

```
pd.read_sql_query("SELECT * FROM role", conn, index_col="id")
```

	name
id	
1	Instructor
2	TA
3	Student

Outline

Overview of relational data

Entity relationships

Pandas and SQLite

Joins

Joins

Join operations merge multiple tables into a single relation (can be then saved as a new table or just directly used)

Four typical types of joins:

1. Inner
2. Left
3. Right
4. Outer

You join two tables *on* columns from each table, where these columns specify which rows are kept

Example: joining Person and Grades

Consider joining two tables, Person and Grades, on ID / Person ID

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Grades

Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80
100	100	100

Inner join (usually what you want)

Join two tables where we only return the rows where the two joined columns contain the *same value*

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80

Only these two rows have an entry in “Person” *and* an entry in “Grades”

Inner join in Pandas/SQLite

```
# Pandas way
df_person = pd.read_sql_query("SELECT * FROM person", conn)
df_grades = pd.read_sql_query("SELECT * FROM grades", conn)
df_person.merge(df_grades, how="inner",
                left_on="id", right_on="person_id")

# SQLite way
cursor.execute("SELECT * FROM person, grades WHERE
                person.id == grades.person_id")
```

In Pandas, you can also join on index using `right_index/left_index` parameters

There is also the join call in Pandas, which is a bit more limited (always assumes right is joined on index, left not on index)

Left joins

Keep all rows of the left table, add entries from right table that match the corresponding columns

Example: left join Person and Grades on ID, Person ID

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
1	Kolter	Zico	1	NULL	NULL
2	Manek	Gaurav	2	NULL	NULL
3	Rice	Leslie	2	NULL	NULL
4	Peres	Filipe	2	NULL	NULL
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80

Left join in Pandas and SQLite

```
# Pandas way
df_person.merge(df_grades, how="left",
                left_on="id", right_on="person_id")

# SQLite way
cursor.execute("SELECT * FROM person LEFT JOIN grades ON
                person.id == grades.person_id")
```

	id	last_name	first_name	person_id	hw1_grade	hw2_grade
0	1	Kolter	Zico	NaN	NaN	NaN
1	2	Manek	Gaurav	NaN	NaN	NaN
2	3	Rice	Leslie	NaN	NaN	NaN
3	4	Peres	Filipe	NaN	NaN	NaN
4	5	Gates	Bill	5.0	85.0	95.0
5	6	Musk	Elon	6.0	80.0	60.0

Right join

Like a left join but with the roles of the tables reversed

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80
100	NULL	NULL	NULL	100	100

```
# Pandas way
df_person.merge(df_grades, how="right",
                left_on="id", right_on="person_id")

# Not supported in SQLite
```

Outer join

Return all rows from both left and right join

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
1	Kolter	Zico	1	NULL	NULL
2	Manek	Gaurav	2	NULL	NULL
3	Rice	Leslie	2	NULL	NULL
4	Peres	Filipe	2	NULL	NULL
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80
100	NULL	NULL	NULL	100	100

```
# Pandas way
df_person.merge(df_grades, how="outer",
                left_on="id", right_on="person_id")
```

15-388/688 - Practical Data Science: Free text and natural language processing

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Announcements

There will be **no lecture** next Monday, 9/30, and we will record a video lecture for this class instead

We will announce the time for the video lecture once it is finalized, and you are welcome to attend in person rather than watch the video online

There will be a few other instances of this during this semester, and we will post these well in advance for future lectures

One-sentence tutorial topic proposals due **Friday**

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

Free text in data science vs. NLP

A large amount of data in many real-world data sets comes in the form of free text (user comments, but also any “unstructured” field)

(Computational) natural language processing: write computer programs that can understand natural language

This lecture: try to get some meaningful information out of unstructured text data

Understanding language is hard

Multiple potential parse trees:

> "While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know." – Groucho Marx

Winograd schemas:

> "The city councilmen refused the demonstrators a permit because they [feared/advocated] violence."

Basic point: We use an incredible amount of context to understand what natural language sentences mean

But is it always hard?

Two reviews for a movie (Star Wars Episode 7)

1. “… truly, a stunning exercise in large-scale filmmaking; a beautifully-assembled picture in which Abrams combines a magnificent cast with a marvelous flair for big-screen, sci-fi storytelling.”

2. “It's loud and full of vim -- but a little hollow and heartless.”

Which one is positive?

We can often very easily tell the “overall gist” of natural language text without understanding the sentences at all

But is it always hard?

Two reviews for a movie (Star Wars Episode 7):

1. "… truly, a **stunning** exercise in large-scale filmmaking; a beautifully-assembled picture in which Abrams combines a **magnificent** cast with a **marvelous** flair for big-screen, sci-fi storytelling."

2. "It's loud and full of vim -- but a little **hollow** and **heartless**."

Which one is positive?

We can often very easily tell the "overall gist" of natural language text without understanding the sentences at all

Natural language processing for data science

In many data science problems, we don't need to truly understand the text in order to accomplish our ultimate goals (e.g., use the text in forming another prediction)

In this lecture we will discuss two simple but very useful techniques that can be used to infer some meaning from text *without* deep understanding

1. Bag of words approaches and TFIDF matrices
2. N-gram language models

Note: This is finally the year that these methods are no longer sufficient for text processing in data science, due to advent of word embedding techniques; in later lectures we will cover deep learning methods for text (word embedding)

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

Brief note on terminology

In this lecture, we will talk about “documents”, which mean individual groups of free text

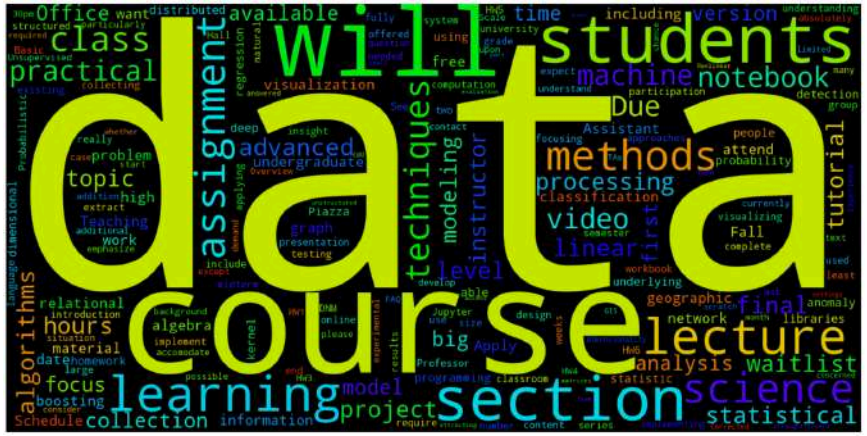
(Could be actual documents, or e.g. separate text entries in a table)

“Words” or “terms” refer to individual words (tokens separated by whitespace) and often also punctuation

“Corpus” refers to a collection of documents

Bag of words

AKA, the word cloud view of documents



Word cloud of class webpage

Represent each document as a vector of word frequencies

Order of words is irrelevant, only matters how often words occur

Bag of words example

"The goal of this lecture is to explain the basics of free text processing"

"The bag of words model is one such approach"

"Text processing via bag of words"

$$
\begin{array}{cl}
\begin{array}{cccccccccc}
\text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & \text{approach}
\end{array} & \\
X = \begin{bmatrix}
2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \dots & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0
\end{bmatrix} & \begin{array}{l} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{array}
\end{array}
$$

Term frequency

"Term frequency" just refers to the counts of each word in a document

Denoted $\mathrm{tf}_{i,j}$ = frequency of word j in document i (sometimes indices are reversed, we use these for consistency with matrix above)

Often (as in the previous slide), this just means the raw count, but there are also other possibilities

1. $\mathrm{tf}_{i,j} \in \{0,1\}$ – does word occur in document or not
2. $\log(1 + \mathrm{tf}_{i,j})$ – log scaling of counts
3. $\mathrm{tf}_{i,j} / \max_j \ \mathrm{tf}_{i,j}$ – scale by document's most frequent word

Inverse document frequency

Term frequencies tend to be “overloaded” with very common words (“the”, “is”, “of”, etc)

Idea if *inverse document frequency* weight words negatively in proportion to how often they occur in the entire set of documents

$$\mathrm{idf}_j = \log\left(\frac{\#\ \text{documents}}{\#\ \text{documents with word } j}\right)$$

As with term frequency, there are other version as well with different scalings, but the log scaling above is most common

Note that inverse document frequency is just defined for *words* not for word-document pairs, like term frequency

Inverse document frequency examples

$$
\begin{array}{cc}
 & \begin{array}{cccccccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & \text{approach} \end{array} \\
X = & \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{bmatrix} \begin{array}{l} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{array}
\end{array}
$$

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

TFIDF

Term frequency inverse document frequency = $\mathrm{tf}_{i,j} \times \mathrm{idf}_j$

Just replace the entries in the X matrix with their TFIDF score instead of their raw counts (also common to remove “stop words” beforehand)

This seems to work much better than using raw scores for e.g. computing similarity between documents or building machine learning classifiers on the documents

$$X = \begin{array}{c} \begin{array}{cccc} \text{the} & \text{is} & \text{of} & \text{goal} \end{array} \\ \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & \end{bmatrix} \end{array}$$

Cosine similarity

A fancy name for “normalized inner product”

Given two documents x, y represented by TFIDF vectors (or just term frequency vectors), cosine similarity is just

$$\text{Cosine_Similarity}(x, y) = \frac{x^T y}{\|x\|_2 \cdot \|y\|_2}$$

Between zero and one, higher numbers mean documents more similar

Equivalent to the (1 minus) the squared distance between the two normalized document vectors

$$\frac{1}{2}\|\tilde{x} - \tilde{y}\|_2^2 = 1 - \text{Cosine_Similarity}(x, y), \qquad \text{where } \tilde{x} = \frac{x}{\|x\|_2}, \tilde{y} = \frac{y}{\|y\|_2}$$

Cosine similarity example

“The goal of this lecture is to explain the basics of free text processing”

“The bag of words model is one such approach”

“Text processing via bag of words”

$$M = \begin{bmatrix} 1 & 0.068 & 0.078 \\ 0.068 & 1 & 0.103 \\ 0.078 & 0.103 & 1 \end{bmatrix}$$

Poll: Cosine similarity

What would you expect to happen if the cosine similarity used term frequency vectors instead of TFIDF vectors?

1. Average cosine similarity between all documents would go up

2. Average cosine similarity between all documents would go down

3. Average cosine similarity between all documents would roughly stay the same

Term frequencies as vectors

You you think of individual words in a term-frequencies model as being "one-hot" vectors in an $\#\text{words}$ dimensional space (here $\#\text{words}$ is total number of unique words in corpus)

$$\text{“pittsburgh”} \equiv e_{\text{pittsburgh}} \in \mathbb{R}^{\#\text{words}} = \begin{array}{c} \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \end{array} \begin{array}{l} \\ \text{pitted} \\ \text{pittsburgh} \\ \text{pivot} \\ \\ \end{array}$$

Document vectors are sums of their word vectors

$$x_{\text{doc}} = \sum_{\text{word} \in \text{doc}} e_{\text{word}}$$

“Distances” between words

No notion of similarity in term frequency vector space:

$$\|e_{\text{pittsburgh}} - e_{\text{boston}}\|_2 = \|e_{\text{pittsburgh}} - e_{\text{banana}}\|_2 = 1$$

But, some words are inherently more related that others

- “Pittsburgh has some excellent new restaurants”
- “Boston is a city with great cuisine”
- “PostgreSQL is a relational database management system”

Under TFIDF cosine similarity (if we don’t remove stop words), then the second two sentences are more similar than the first and second

- Preview of *word embeddings*, to be discussed in later lecture

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

Language models

While the bag of words model is surprisingly effective, it is clearly throwing away a lot of information about the text

The terms “boring movie and not great” is not the same in a movie review as “great movie and not boring”, but they have the exact same bag of words representations

To move beyond this, we would like to build a more accurate model of how words really relate to each other: language model

Probabilistic language models

We haven't covered probability much yet, but with apologies for some forward references, a (probabilistic) language model aims at providing a probability distribution over every word, given all the words before it

$$P(\text{word}_i | \text{word}_1, \ldots, \text{word}_{i-1})$$

E.g., you probably have a pretty good sense of what the next word should be:

- "Data science is the study and practice of how we can extract insight and knowledge from large amounts of"

$$P(\text{word}_i = \text{“data”} | \text{word}_1, \ldots, \text{word}_{i-1}) = ?$$
$$P(\text{word}_i = \text{“hotdogs”} | \text{word}_1, \ldots, \text{word}_{i-1}) = ?$$

Building language models

Building a language model that captures the true probabilities of natural language is still a distant goal

Instead, we make simplifying assumptions to build approximate tractable models

N-gram model: the probability of a word depends only on the $n-1$ words preceding it

$$P(\text{word}_i | \text{word}_1, \ldots, \text{word}_{i-1}) \approx P(\text{word}_i | \text{word}_{i-n+1}, \ldots, \text{word}_{i-1})$$

This puts a hard limit on the *context* that we can use to make a prediction, but also makes the modeling more tractable

"large amounts of data" vs. "large amounts of hotdogs"

Estimating probabilities

A simple way (but *not* the only way) to estimate the conditional probabilities is simply by counting

$$P(\text{word}_i | \text{word}_{i-n+1}, \ldots, \text{word}_{i-1}) = \frac{\#(\text{word}_{i-n+1}, \ldots, \text{word}_\text{i})}{\#(\text{word}_{i-n+1}, \ldots, \text{word}_{i-1})}$$

E.g.:

$$P(\text{“data”} | \text{“large amounts of”}) = \frac{\#(\text{“large amounts of data”})}{\#(\text{“large amounts of”})}$$

Example of estimating probabilities

Very short corpus:

“The goal of this lecture is to explain the basics of free text processing”

Using an 2-gram model

$$P(\text{word}_i | \text{word}_{i-1} = \text{“of”}) = ?$$

Laplace smoothing

Estimating language models with raw counts tends to estimate a lot of zero probabilities (especially if estimating the probability of some new text that was not used to build the model)

Simple solution: allow for any word to appear with some small probability

$$P(\mathrm{word}_i | \mathrm{word}_{i-n+1}, \dots, \mathrm{word}_{i-1}) = \frac{\#(\mathrm{word}_{i-n+1}, \dots, \mathrm{word_i}) + \alpha}{\#(\mathrm{word}_{i-n+1}, \dots, \mathrm{word}_{i-1}) + \alpha D}$$

where α is some number and D is total size of dictionary

Also possible to have “backoffs” that use a lower degree n-gram when the probability is zero

How do we pick n?

Lower n: less context, but more samples of each possible n-gram

Higher n: more context, but less samples

“Correct” choice is to use some measure of held-out cross-validation

In practice: use $n = 3$ for large datasets (i.e., triplets) , $n = 2$ for small ones

Examples

Random samples from language model trained on Shakespeare:

n=1: “in as , stands gods revenge ! france pitch good in fair hoist an what fair shallow-rooted , . that with wherefore it what a as your . , powers course which thee dalliance all”

n=2: “look you may i have given them to the dank here to the jaws of tune of great difference of ladies . o that did contemn what of ear is shorter time ; yet seems to”

n=3: “believe , they all confess that you withhold his levied host , having brought the fatal bowels of the pope ! ' and that this distemper'd messenger of heaven , since thou deniest the gentle desdemona ,”

More examples

n=7: “so express'd : but what of that ? 'twere good you do so much for charity . i cannot find it ; 'tis not in the bond . you , merchant , have you any thing to say ? but little”

This is starting to look a lot like Shakespeare, because it is Shakespeare

As we have higher order n-grams, the previous (n-1) words have only appeared very few times in the corpus, so we will always just sample the next word that occurred

Evaluating language models

How do we know how well a language model performs

Common strategy is to estimate the probability of some held out portion of data, and evaluate *perplexity*

$$\text{Perplexity} = 2^{-\frac{\log_2 P(\text{word}_1, \ldots \text{word}_N)}{N}} = \left(\frac{1}{P(\text{word}_1, \ldots \text{word}_N)}\right)^{\frac{1}{N}}$$

where we can evaluate the probability using

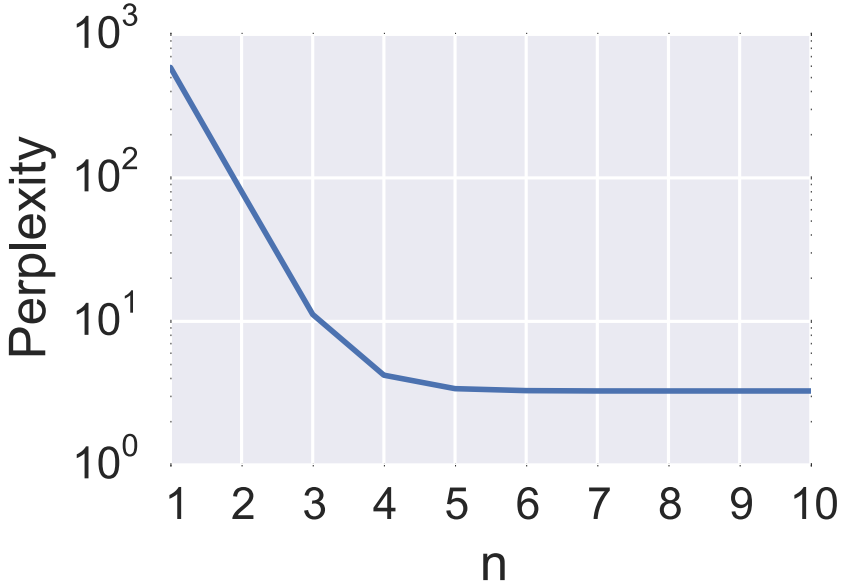
$$P(\text{word}_1, \ldots \text{word}_n) = \prod_{i=n}^{N} P(\text{word}_i | \text{word}_{i-n+1}, \ldots, \text{word}_{i-1})$$

(note that you can compute the log of this quantity directly)

Evaluating perplexity

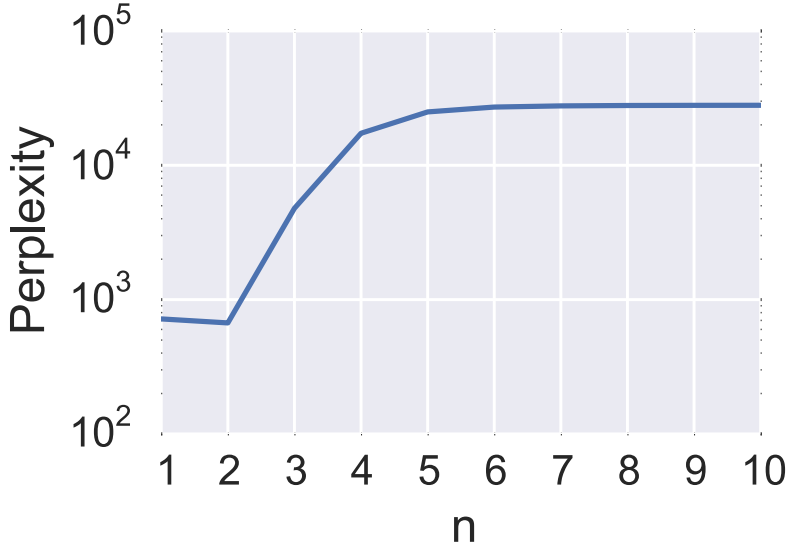
Perplexity on the corpus used to build the model will always decrease using higher n (fewer choices of what comes next means higher probability of observed data)

Note: this is only strictly true when $\alpha = 0$



Evaluating perplexity

What really matters is how well the model captures text from the “same” distribution that was *not* used to train the model



This is a preview of overfitting/model selection, which we will talk about a lot in the machine learning section

Outline

Free text in data science

Bag of words and TFIDF

Language models and N-grams

Libraries for handling free text

NLTK library

The NLTK (natural language toolkit) library (http://www.nltk.org) is a standard Python library for handling text and natural language data

Note: NLTK is a massive library, and is a bit more geared towards things like tagging, parsing, and more complex processes instead of the techniques described previously

Additionally, it actually doesn't contain much of what we want to do (no TFIDF creation, there was an n-gram language model but it was removed due to bugs)

You may want to look at some other options: spacy, CoreNLP

Reading and tagging documents

Load nltk and download necessary files:

```
import nltk
import nltk.corpus
#nltk.download() # just run this once
```

Tokenize a document

```
sentence = "The goal of this lecture isn't to explain complex free text
processing"
tokens = nltk.word_tokenize(sentence)
# ['The', 'goal', 'of', 'this', 'lecture', 'is', "n't", 'to', 'explain',
'complex', 'free', 'text', 'processing']
```

Tag parts of speech

```
pos = nltk.pos_tag(tokens)
# [('The', 'DT'), ('goal', 'NN'), ('of', 'IN'), ('this', 'DT'),
('lecture', 'NN'), ('is', 'VBZ'), ("n't", 'RB'), ('to', 'TO'),
('explain', 'VB'), ('complex', 'JJ'), ('free', 'JJ'), ('text', 'NN'),
('processing', 'NN')]
```

Stop words and n-grams

Get list of English stop words (common words)

```
stopwords = nltk.corpus.stopwords.words("English")
print [a for a in tokens if a.lower() not in stopwords]
# ['goal', 'lecture', "n't", 'explain', 'complex', 'free', 'text',
'processing']
```

Generate n-grams from document

```
list(nltk.ngrams(tokens, 3))
# [('The', 'goal', 'of'), ('goal', 'of', 'this'), ('of', 'this',
'lecture'), ('this', 'lecture', 'is'), ('lecture', 'is', "n't"), ('is',
"n't", 'to'), ("n't", 'to', 'explain'), ('to', 'explain', 'complex'),
('explain', 'complex', 'free'), ('complex', 'free', 'text'), ('free',
'text', 'processing')]

# code below does the same thing, without nltk
zip(*[tokens[i:] for i in range(3)])
```

15-388/688 - Practical Data Science: Data collection and scraping

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

The data collection process

Common data formats and handling

Regular expressions and parsing

Outline

The data collection process

Common data formats and handling

Regular expressions and parsing

The first step of data science

The first step in data science …

... is to get some data

You will typically get data in one of four ways:

1. Directly download a data file (or files) manually – not much to say
2. Query data from a database – to be covered in later lecture
3. Query an API (usually web-based, these days) } covered today
4. Scrap data from a webpage

Issuing HTTP queries

The vast majority of automated data queries you will run will use HTTP requests (it's become the dominant protocol for much more than just querying web pages)

I know we promised to teach you know things work under the hood … but we are *not* going to make you implement an HTTP client

Do this instead (requests library, http://docs.python-requests.org/):

```
import requests
response = requests.get("http://www.datasciencecourse.org")

# some relevant fields
response.status_code
response.content # or response.text
response.headers
response.headers['Content-Type']
```

HTTP Request Basics

You've seen URLs like these:
`https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&cad=rja&uact=8…`

The weird statements after the url are *parameters,* you would provide them using the requests library like this:

```
params = {"sa":"t", "rct":"j", "q":"", "esrc":"s",
          "source":"web", "cd":"9", "cad":"rja", "uact":"8"}
response = requests.get("http://www.google.com/url", params=params)
```

HTTP GET is the most common method, but there are also PUT, POST, DELETE methods that *change* some state on the server

```
response = requests.put(...)
response = requests.post(...)
response = requests.delete(...)
```

RESTful APIs

If you move beyond just querying web pages to web APIs, you'll most likely encounter REST APIs (Representational State Transfer)

REST is more a design architecture, but a few key points:

1. Uses standard HTTP interface and methods (GET, PUT, POST, DELETE)
2. Stateless – the server doesn't remember what you were doing

Rule of thumb: if you're sending the your account key along with each API call, you're probably using a REST API

Querying a RESTful API

You query a REST API similar to standard HTTP requests, but will almost always need to include parameters

```
token = "" # not going to tell you mine
response = requests.get("https://api.github.com/user",
                        params={"access_token":token})
print(response.content)
#{"login":"zkolter","id":2465474,"avatar_url":"https://avatars.githubu…
```

Get your own access token at https://github.com/settings/tokens/new

GitHub API uses GET/PUT/DELETE to let you query or update elements in your GitHub account automatically

Example of REST: server doesn't remember your last queries, for instance you always need to include your access token if using it this way

Authentication

Basic authentication has traditionally been the most common approach to access control for web pages

```
# this won't work anymore
response = requests.get("https://api.github.com/user",
                        auth=('zkolter', 'passwd'))
```

Most APIs have replaced this with some form of OAuth (you'll get familiar with OAuth in the homework)

Outline

The data collection process

Common data formats and handling

Regular expressions and parsing

Data formats

The three most common formats (judging by my completely subjective experience):

1. CSV (comma separate value) files

2. JSON (Javascript object notation) files and strings

3. HTML/XML (hypertext markup language / extensible markup language) files and strings

CSV Files

Refers to any delimited text file (not always separated by commas)

```
"Semester","Course","Section","Lecture","Mini","Last Name","Preferred/First
Name","MI","Andrew ID","Email","College","Department","Class","Units","Grade
Option","QPA Scale","Mid-Semester Grade","Final Grade","Default Grade","Added
By","Added On","Confirmed","Waitlist Position","Waitlist Rank","Waitlisted
By","Waitlisted On","Dropped By","Dropped On","Roster As Of Date”
"F16","15688","B","Y","N",”Kolter","Zico","","zkolter","zkolter@andrew.cmu.edu","S
CS","CS","50","12.0","L","4+"," "," ","","reg","1 Jun
2016","Y","","","","","","","30 Aug 2016 4:34"
```

If values themselves contain commas, you can enclose them in quotes (our registrar apparently always does this, just to be safe)

```
import pandas as pd
dataframe = pd.read_csv("CourseRoster_F16_15688_B_08.30.2016.csv",
                        delimiter=',', quotechar='"')
```

We’ll talk about the pandas library a lot more in later lectures

JSON files / string

JSON originated as a way of encapsulating Javascript objects

A number of different data types can be represented

- Number: `1.0` (always assumed to be floating point)
- String: `"string"`
- Boolean: `true` or `false`
- List (Array): `[item1, item2, item3,…]`
- Dictionary (Object in Javascript): `{"key":value}`

Lists and Dictionaries can be embedded within each other:

```
[{"key":[value1, [value2, value3]]}]
```

Example JSON data

JSON from Github API


```
{
 "login":"zkolter",
 "id":2465474,
 "avatar_url":"https://avatars.githubusercontent.com/u/2465474?v=3",
 "gravatar_id":"",
 "url":"https://api.github.com/users/zkolter",
 "html_url":"https://github.com/zkolter",
 "followers_url":"https://api.github.com/users/zkolter/followers",
 "following_url":"https://api.github.com/users/zkolter/following{/other_user}",
 "gists_url":"https://api.github.com/users/zkolter/gists{/gist_id}",
 "starred_url":"https://api.github.com/users/zkolter/starred{/owner}{/repo}",
 "subscriptions_url":"https://api.github.com/users/zkolter/subscriptions",
 "organizations_url":"https://api.github.com/users/zkolter/orgs",
 "repos_url":"https://api.github.com/users/zkolter/repos",
 "events_url":"https://api.github.com/users/zkolter/events{/privacy}",
 "received_events_url":"https://api.github.com/users/zkolter/received_events",
 "type":"User",
 "site_admin":false,
 "name":"Zico Kolter”
 ...
```

Parsing JSON in Python

Built-in library to read/write Python objects from/to JSON files

```
import json

# load json from a REST API call
response = requests.get("https://api.github.com/user",
                        params={"access_token":token})
data = json.loads(response.content)

json.load(file) # load json from file
json.dumps(obj) # return json string
json.dump(obj, file) # write json to file
```

XML / HTML files

The main format for the web (though XML seems to be loosing a bit of popularity to JSON for use in APIs / file formats)

XML files contain hiearchical content delineated by tags

```
<tag attribute="value">
    <subtag>
        Some content for the subtag
    </subtag>
    <openclosetag attribute="value2”/>
</tag>
```

HTML is syntactically like XML but horrible (e.g., open tags are not always closed), more fundamentally, HTML is mean to describe appearance

Parsing XML/HTML in Python

There are a number of XML/HTML parsers for Python, but a nice one for data science is the BeautifulSoup library (specifically focused on getting data out of XML/HTML files)

```
# get all the links within the data science course schedule
from bs4 import BeautifulSoup
import requests
response = requests.get("http://www.datasciencecourse.org/2016")

root = BeautifulSoup(response.content)
root.find("section",id="schedule")\
    .find("table").find("tbody").findAll("a")
```

You'll play some with BeautifulSoup in the first homework

Outline

The data collection process

Common data formats and handling

Regular expressions and parsing

Regular expressions

Once you have loaded data (or if you need to build a parser to load some other data format), you will often need to search for specific elements within the data

E.g., find the first occurrence of the string “data science”

```
import re
text = "This course will introduce the basics of data science"
match = re.search(r"data science", text)
print(match.start())
# 49
```

Regular expressions in Python

A few common methods to call regular expressions in Python:

```
match = re.match(r"data science", text) # check if start of text matches
match = re.search(r"data science", text) # find first match or None
for match in re.finditer("data science", text):
    # iterate over all matches in the text
    ...
all_matches = re.findall(r"data science", text) # return all matches
```

You can also use “compiled” version of regular expressions

```
regex = re.compile(r"data science")
regex.match(text, [startpos, [endpos]])
regex.search(...)
regex.finditer(...)
regex.findall(...)
```

Matching multiple potential characters

The real power of regular expressions comes in the ability to match multiple possible sequence of characters

Special characters in regular expressions: .^$*+?{}\[]|() (if you want to match these characters exactly, you need to escape them: \$)

Match sets of characters:

- Match the character ‘a’: a
- Match the character ‘a’, ‘b’, or ‘c’: [abc]
- Many any character except ‘a’, ‘b’, or ‘c’: [^abc]
- Match any digit: \d (= [0-9])
- Match any alpha-numeric: \w (= [a-zA-z0-9_])
- Match whitespace: \s (= [\t\n\r\f\v])
- Match any character:. (including newline with re.DOTALL)

Matching repeated characters

Can match one or more instances of a character (or set of characters)

Some common modifiers:

- Match character ‘a’ exactly once: a
- Match character ‘a’ zero or one time: a?
- Match character ‘a’ zero or more times: a*
- Match character ‘a’ one or more times: a+
- Match character ‘a’ exactly n times: a{n}

Can combine these with multiple character matching:

- Match all instances of “<something> science” where <something> is an alphanumeric string with at least one character
- \w+\s+science

Poll: regular expressions

Which strings would be matched (i.e, calling re.match()) by the regular expression?

`\w+\s+science`

1. “life science”
2. “life sciences”
3. “life. Science”
4. “this data science problem”

Grouping

We often want to obtain more information that just whether we found a match or not (for instance, we may want to know what text matched)

Grouping: enclose portions of the regular expression in quotes to “remember” these portions of the match

`(\w+)\s([Ss]cience)`

```
match = re.search(r"(\w+)\s([Ss]cience)", text)
print(match.start(), match.groups())
# 49 ('data', 'science')
```

Why the ‘r’ before the string? Avoids need to double escape strings

Substitutions

Regular expressions provide a power mechanism for replacing some text with outer text

```
better_text = re.sub(r"data science", r"schmada science", text)
```

To include text that was remembered in the matching using groups, use the escaped sequences \1, \2, … in the substitution text

```
better_text = re.sub(r"(\w+)\s([Ss])cience", r"\1 \2hmience", text)
```

(You can also use backreferences within a single regular expression)

Ordering and greedy matching

There is an order of operations in regular expressions

- `abc|def` matches the strings “abc” or “def”, not “ab(c or d)ef”
- You can get around this using parenthesis e.g. `a(bc|de)f`
- This also creates a group, use `a(?:bc|de)f` if you don’t want to capture it

By default, regular expressions try to capture as much text as possible (greedy matching)

- `<(.*)>` applied to `<a>text</a>` will match the entire expression
- If you want to capture the *least* amount of text possible use `<(.*?)>` this will just match the `<a>` term

Additional features

We left out a lot of elements here to keep this brief: start/end lines, lookaheads, named groups, etc

Don't worry if you can't remember all this notation (I had to look some things up while preparing this lecture too)

Use the docs: https://docs.python.org/3/howto/regex.html, https://docs.python.org/3/library/re.html

Try out test expressions to see what happens

15-388/688 - Practical Data Science: Graph and network processing

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

Networks vs. graphs?

Our terminology (fairly standard, though some use them differently):

Networks are the systems of interrelated objects (in the real world)

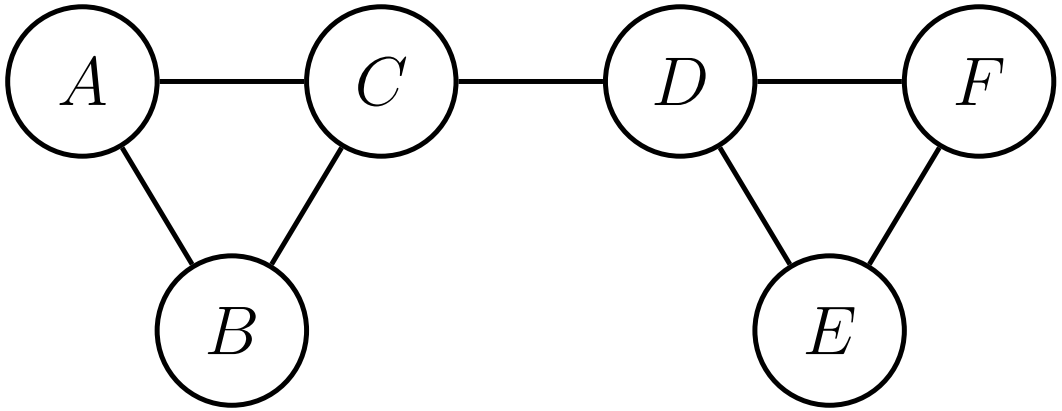
Graphs are the mathematical model for representing networks

This lecture is largely about representations and algorithms for graphs

But of course, in data science we use these algorithms to answer questions about networks

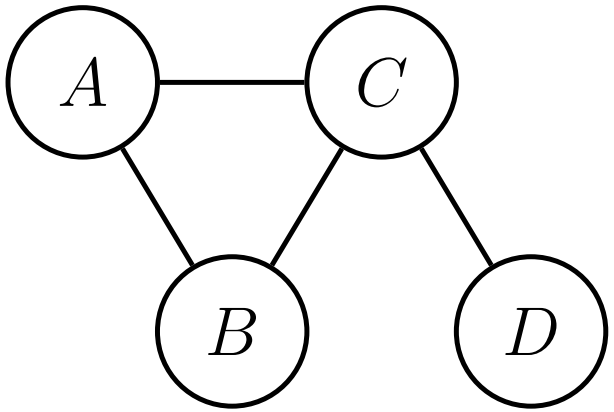
Graphs models

A graph is a collection of vertices (nodes) and edges $G = (V, E)$

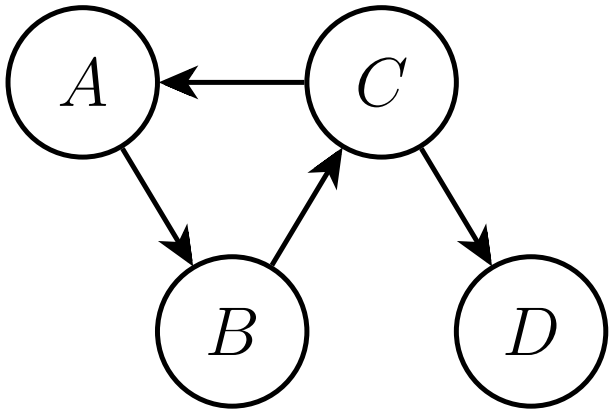


$$V = \{A, B, C, D, E, F\}$$
$$E = \{(A, B), (A, C), (B, C), (C, D), (D, E), (D, F), (E, F)\}$$

Directed vs. undirected graphs

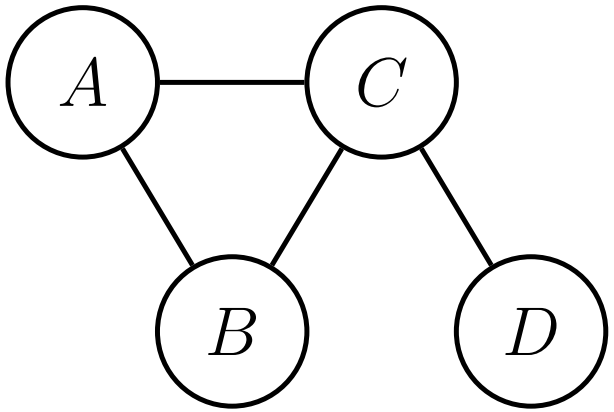


Undirected
E.g. paper co-authorship

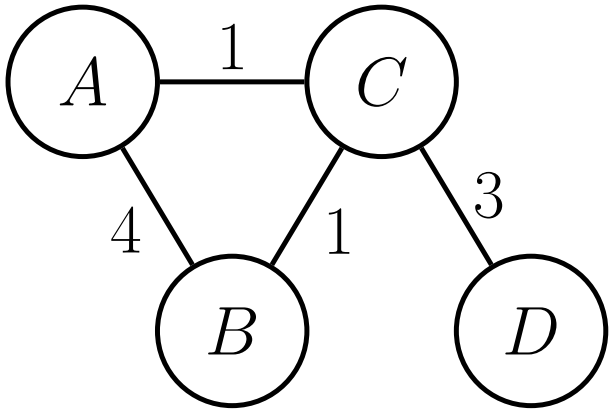


Directed
E.g. web links

Weighted vs. unweighted graphs

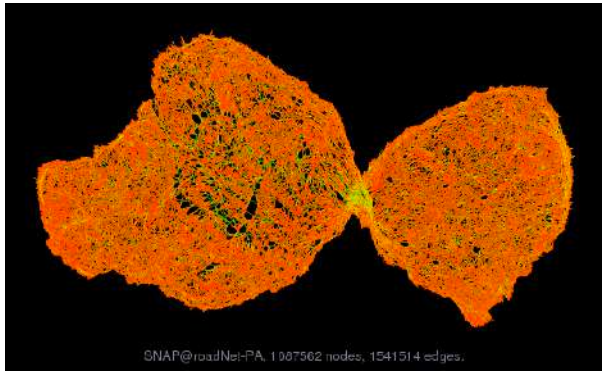


Unweighted
E.g. friends on social network

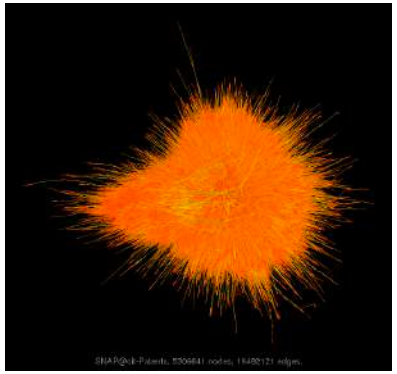


Weighted
E.g. travel distance between cities

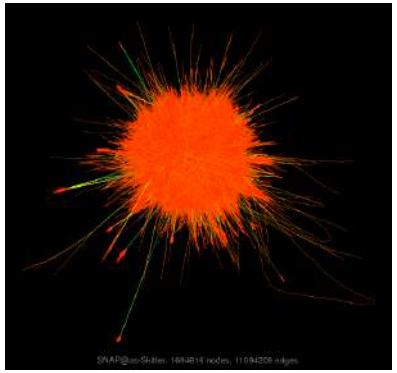
Some example graphs



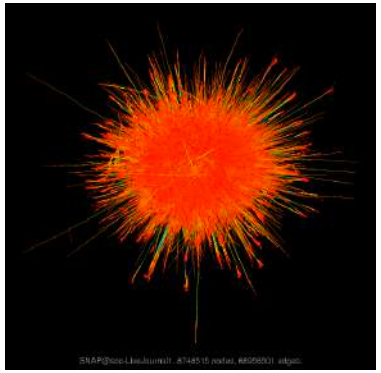
PA road network:
1M nodes, 3M edges



Patent citations:
3.7M nodes, 16.5M edges



Internet topology (in 2005)
1.6M nodes, 11M edges



LiveJournal social network
4.8M nodes, 69M edges

Graphs from http://snap.stanford.edu, visualizations from http://www.cise.ufl.edu/research/sparse/matrices/SNAP/

Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

Representations of graphs

There are a few different ways that graphs can be represented in a program, which one you choose depends on your use case

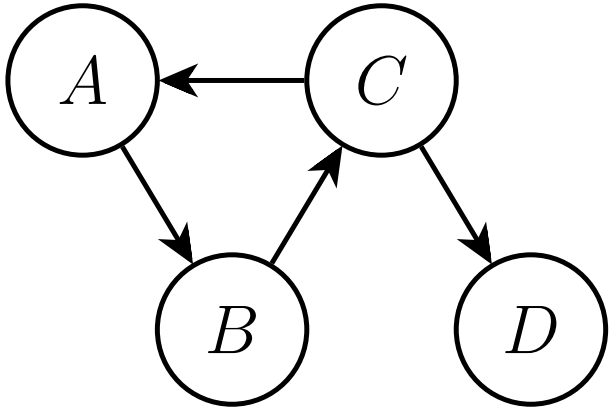
E.g., are you going to be modifying the graph dynamically (adding/removing nodes/edges), just analyzing a static graph, etc?

Three main types we will consider:

1. Adjacency list
2. Adjacency dictionary
3. Adjacency matrix

Adjacency list

For each node, store an array of the nodes that it connects to



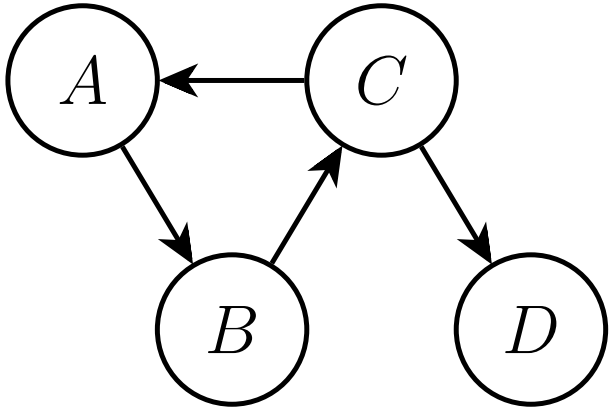
Node	Edges
A	[B]
B	[C]
C	[A,D]
D	[]

Pros: easy to get all outgoing links from a given node, fast to add new edges (without checking for duplicates)

Cons: deleting edges or checking existence of an edge requires scan through given node's full adjacency array

Adjacency dictionary

For each node, store a dictionary of the nodes that it connects to



Node (key)	Edges
A	{B:1.0}
B	{C:1.0}
C	{A:1.0,D:1.0}
D	{}

Pros: easy to add/remove/query edges (requires two dictionary lookups, so a $O(1)$ operation)

Cons: overhead of using a dictionary over array

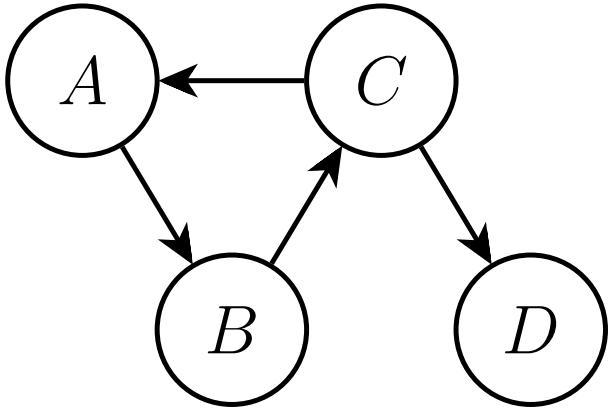
Poll: complexity of graph operations

Suppose we have a directed graph with n nodes where each node has fewer than some constant $k \ll n$ ingoing or outgoing edges. In the adjacency dictionary representation, which of the following operations are constant time ($O(1) \equiv O(k)$)?

1. Checking if there is a link between two nodes $A \to B$
2. Finding all the outgoing edges of a node A
3. Finding all the incoming edges of a node A
4. Deleting all outgoing and incoming edges of a node A
5. Deleting the link between two nodes $A \to B$
6. Adding a new node Z to the graph and adding links $A \to Z$, $Z \to B$

Adjacency matrix

Store the connectivity of the graph as a matrix



$$A = \begin{array}{c} \text{(From)} \\ \begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{array}{c} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \end{array} \text{(To)} \end{array}$$

In virtually all cases, you will want to store this as a sparse matrix

Pros/cons depend on which sparse matrix format you use, but most operations on a static graph will but much faster using the right format

Connection between adjacency list and sparse CSC format

Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

Graph algorithms

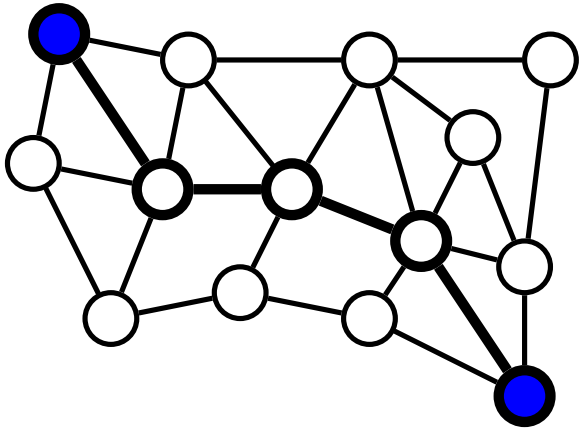
Algorithms for graphs could be (in fact, is) an entire course on its own

We're going to briefly highlight just three algorithms that address different problem classes in graphs

1. Finding shortest paths in a graph – Dijkstra's algorithm
2. Finding important nodes in a graph – PageRank
3. Finding communities in a graph – Girvan-Newman

Shortest path problem

Classical graph problem: find the shortest path between two nodes



Some important distinctions or modifications

- Weighted vs. unweighted, directed vs. undirected, negative weights
- Single-source shortest path (we'll do this one)
- All-pairs shortest path

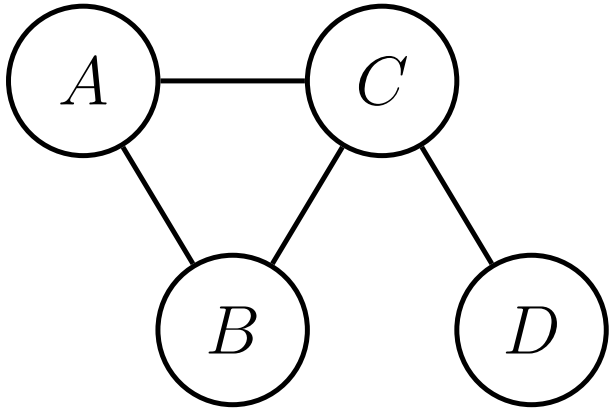
Dijkstra's algorithm

Algorithm for single-source shortest path

Basic idea: dynamic programming algorithm, at each node maintain an *upper bound* on distance to source, iteratively expand node with smallest upper bound (updating bounds of its neighbors)

Given: Graph $G = (V, E)$, Source s
Initialize:
$D[s] \leftarrow 0,\ D[i \neq s] \leftarrow \infty$
$Q \leftarrow V$
Repeat until Q empty:
$i \leftarrow$ Remove element from Q with smallest D
For all j such that $(i, j) \in E$:
$D[j] = \min(D[j], D[i] + 1)$

Dijkstra's algorithm example



Initialization: source A

$$D = [0, \infty, \infty, \infty]$$
$$Q = \{A, B, C, D\}$$

Step 1: Pop node A

$$Q = \{B, C, D\}$$
$$D = [0,1,1,\infty]$$

Step 2: Pop node B

$$Q = \{C, D\}$$
$$D = [0,1,1,\infty]$$

Step 3: Pop node C

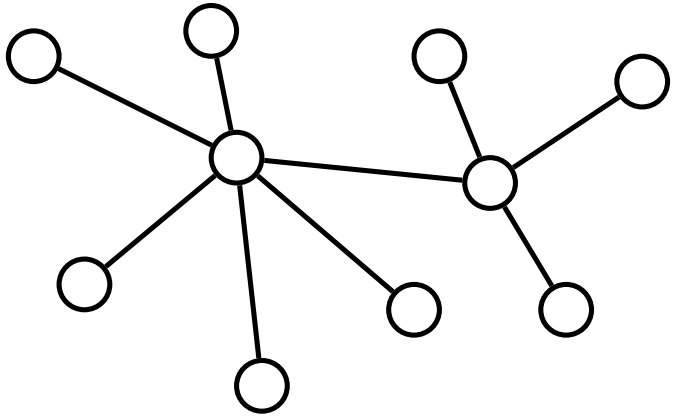
$$Q = \{D\}$$
$$D = [0,1,1,2]$$

Step 4: Pop node D

$$Q = \{\ \}$$
$$D = [0,1,1,2]$$

“Important” nodes

What are the important nodes in the following network?



Unlike shortest path, there is not correct answer here, depends on how you define importance

PageRank algorithm

The algorithm that started Google

Perspective on importance: consider a *random walk* on the graph

- We start at a random node
- We repeatedly jump to a random neighboring node
- If the node has no outgoing edges (in directed graph), jump to a random node
- (Optionally) also jump to a random node with probability d

Node importance is the probability that we will be at a given node when following the above procedure

PageRank algorithm

Given: Graph $G = (V, E)$, restart probability d, iteration count T
Initialize:

$A \leftarrow \text{Adjacency_Matrix}(G)$

$P \leftarrow$ replace zero columns of A with 1, and normalize columns

$\hat{P} \leftarrow (1-d)P + \frac{d}{|V|}11^T$

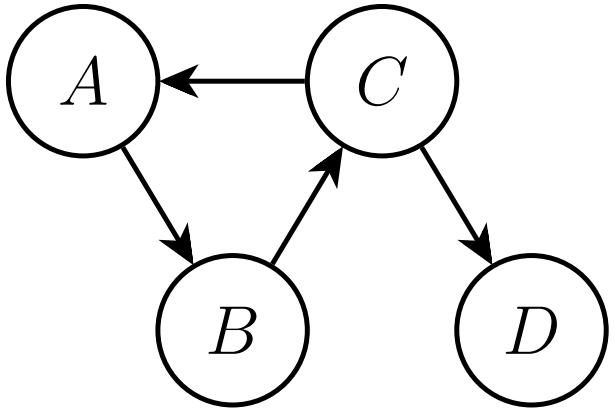
$x \leftarrow \frac{1}{|V|}1$

Repeat T times:

$x \leftarrow \hat{P}x$

For those who have heard these terms, this algorithm is creating a Markov chain over the graph, and finding the stationary distribution (largest eigenvector) of this Markov chain

PageRank example



$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad d = 0.1$$

$$P = \begin{bmatrix} 0 & 0 & 0.5 & 0.25 \\ 1 & 0 & 0 & 0.25 \\ 0 & 1 & 0 & 0.25 \\ 0 & 0 & 0.5 & 0.25 \end{bmatrix} \quad \hat{P} = \begin{bmatrix} 0.025 & 0.025 & 0.475 & 0.25 \\ 0.925 & 0.025 & 0.025 & 0.25 \\ 0.025 & 0.925 & 0.025 & 0.25 \\ 0.025 & 0.025 & 0.475 & 0.25 \end{bmatrix} \quad x \rightarrow \begin{bmatrix} 0.21 \\ 0.26 \\ 0.31 \\ 0.21 \end{bmatrix}$$

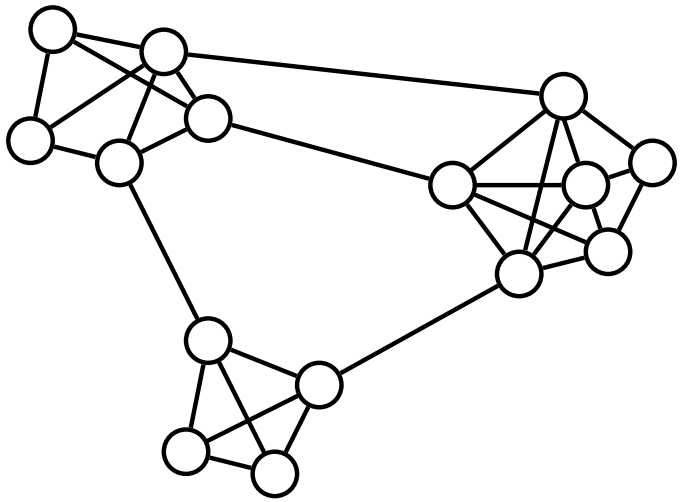
Poll: PageRank

What would happen if we did not replace the all-zero columns in A with all-ones?

1. The algorithm would take longer to run before it converged
2. Pages with no outgoing edges would increase in importance
3. Pages with no outgoing edges would decrease in importance
4. The sum of all probabilities (x) would no longer sum to one

Community detection

Community: subgraphs where nodes are densely connected to each other, but sparsely connected to other nodes



A “soft” version of a clique (a fully connected subgraph)

A fundamental concept in e.g. social networks

Girvan-Newman Algorithm

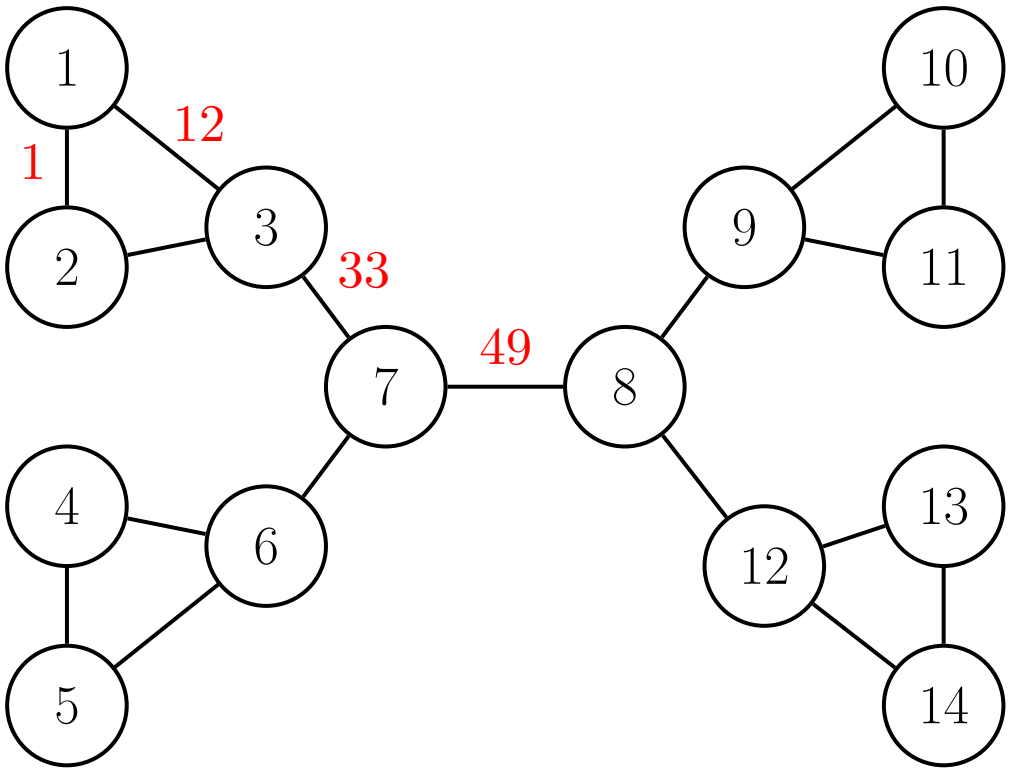
Published in 2002 (Girvan and Newman, 2002), one of the first methods of "modern" community detection

Basic idea: Recursively partition the network by removing edges, groups that are last to be partitioned are "communities"

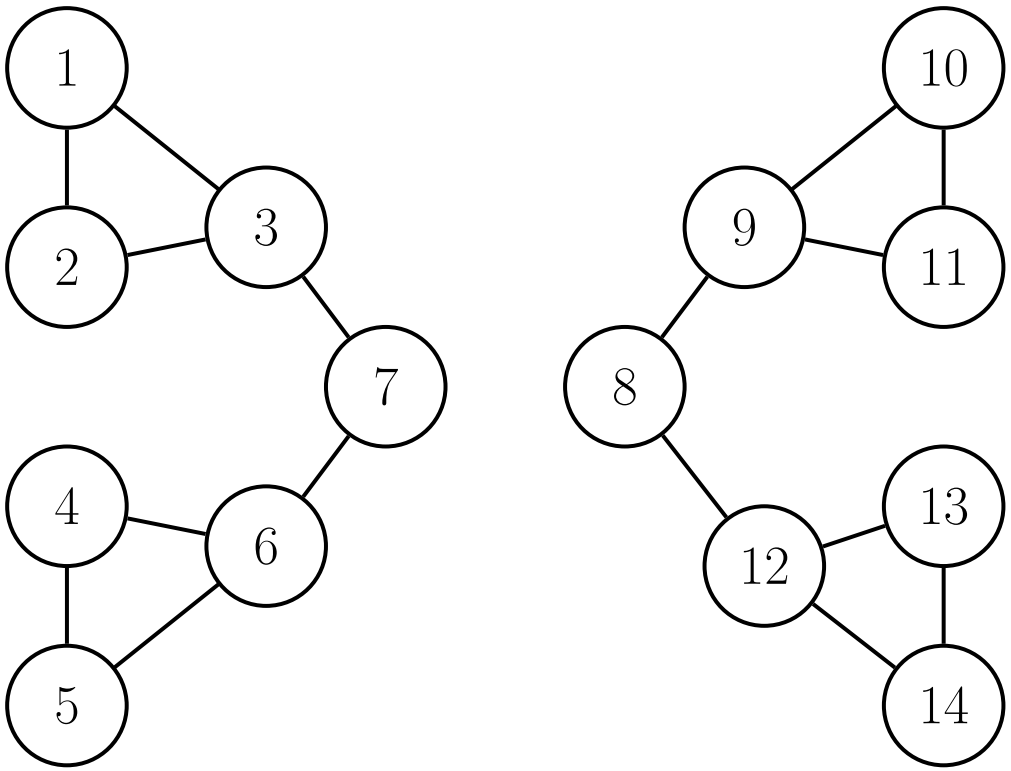
1. Compute "betweenness" of edges in the network = number of shortest paths that pass through each edge
2. Remove edge(s) with highest betweenness, if this breaks the graph into subgraphs, recursively partition each one
3. Result is a hierarchical partitioning of the graph

Challenge is efficiently computing betweenness as we partition graph (we will not cover this)

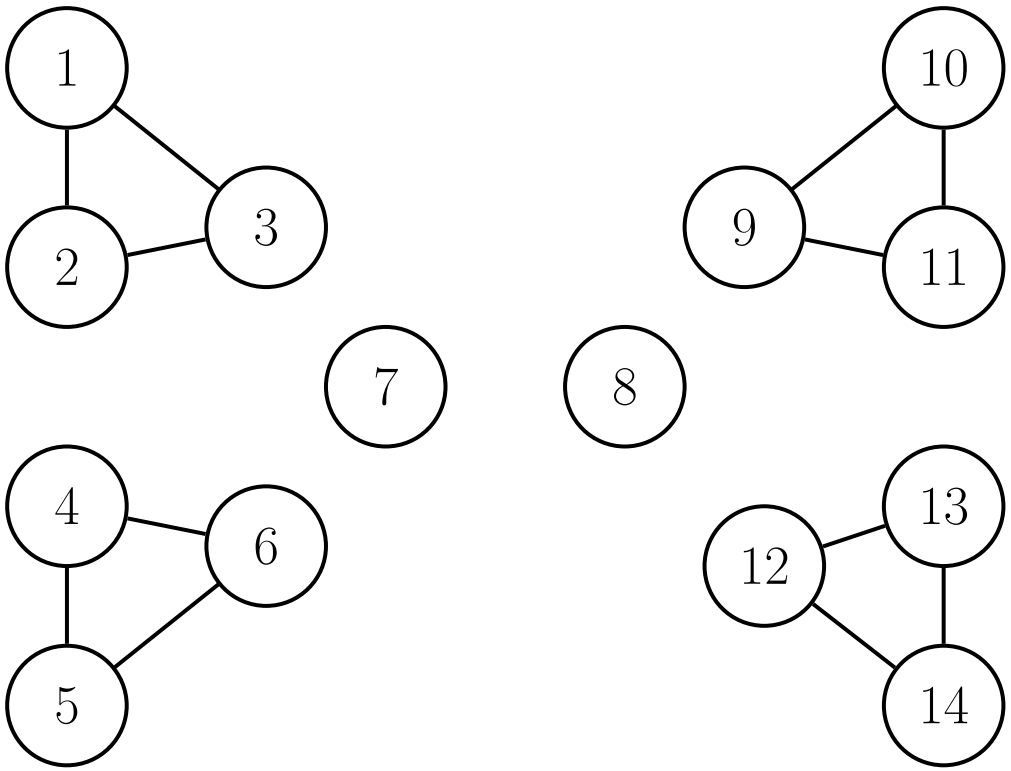
Algorithmic illustration



Algorithmic illustration



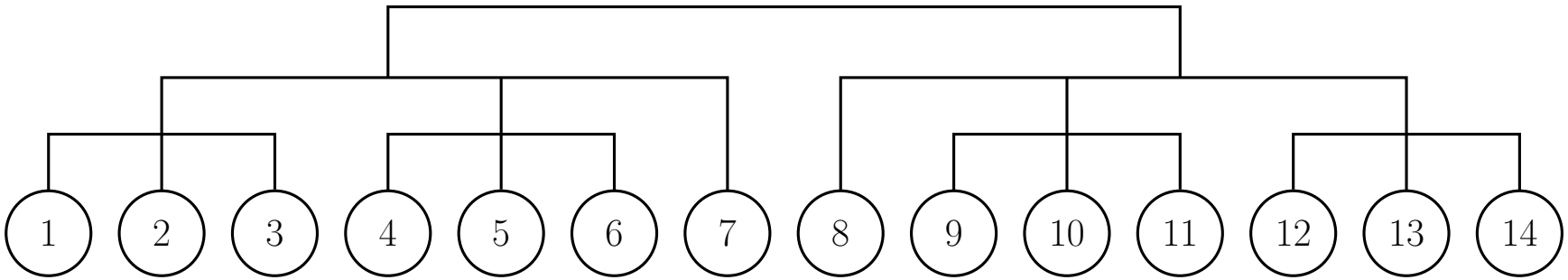
Algorithmic illustration



Resulting hierarchy (dendrogram)

Communities can be extracted by looking at the grouping at different levels of the tree

May want to threshold on things like community size, etc



Outline

Networks and graph

Representing graphs

Graph algorithms

Graph libraries

NetworkX

NetworkX: Python library for dealing with (medium-sized) graphs

https://networkx.github.io/

Simple Python interface for constructing graph, querying information about the graph, and running a large suite of algorithms

Not suitable for very large graphs (all native Python, using adjacency dictionary representation)

Creating graphs

Create an undirected or directed graph

```
import networkx as nx

G = nx.Graph()    # undirected graph
G = nx.DiGraph() # directed graph
```

Add and remove nodes/edges

```
# add and remove edges
G.add_edges_from([("A","B"), ("B","C"), ("C","A"), ("C","D")])
G.remove_edge("A","B")
G.add_edge("A","B")
G.remove_edges_from([("A","B"), ("B","C")])
G.add_edges_from([("A","B"), ("B","C")])
# also add_node(), remove_node(), add_nodes_form(), remove_nodes_from()
```

Nodes/edges and properties

NetworkX uses adjacency dictionary format internally

```
print G["C"]
# {'A': {}, 'D': {}}
```

Iterate over nodes and edges

```
for i in G.nodes():     # loop over nodes
    print i
for i,j in G.edges():   # loop over edges
    print i,j
```

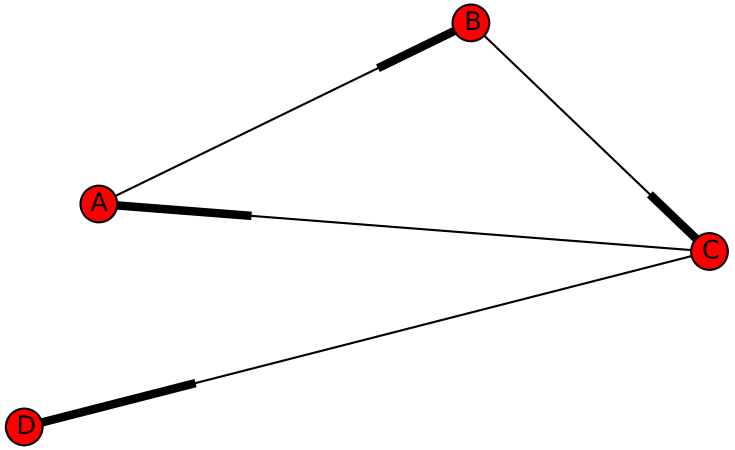
Get and set node/edge properties

```
G.node["A"]["node_property"] = "node_value"
G.edge["A"]["B"]["edge_property"] = "edge_value"
G.nodes(data=True) # iterator over nodes returning properties
G.edges(data=True) # iterator over edges returning properties
```

Drawing and node properties

Draw a graph using matplotlib (not the best visualization)

```
import matplotlib.pyplot as plt
%matplotlib inline
nx.draw(G,with_labels=True)
plt.savefig("mpl_graph.pdf")
```



Algorithms

Almost all the (medium scale) algorithms you could want

```
nx.shortest_path_length(G,source="A") # iterater over path lengths

nx.pagerank(G,alpha=0.9) # dictionary of node ranks

# NOTE: this requires Networkx 2.0
nx.girvan_newman(G) # iterator over partitions at
                      different hierarchy levels
```

15-388/688 - Practical Data Science: Visualization and Data Exploration

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Annoucements

HW1 due tomorrow

HW2 released tomorrow, **due 10/1**

Pinned thread on Diderot for common questions on HW1

Very firm on deadlines for HW (submit well before midnight, any additional time will count as a late day)

Outline

Basics of visualization

Data types and visualization types

Software plotting libraries

Outline

Basics of visualization

Data types and visualization types

Software plotting libraries

Two types of visualization

Data exploration visualization: figuring out what is true

Data presentation visualization: convincing other people it is true

This lecture will mostly be focused on the first, some later lectures will touch on the second

“Data exploration” is much broader than just visualization (most of the analysis techniques we will cover fit into it)

Importance of visualization

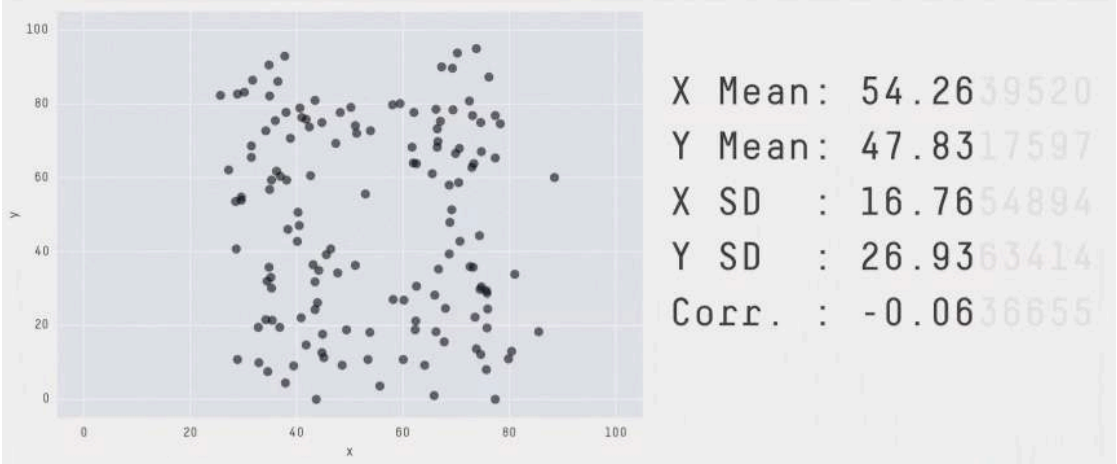
Before you run any analysis, build any machine learning system, etc, always visualize your data

If you can't identify a trend or make a prediction for your dataset, neither will an automated algorithm

- This is especially important to keep in mind as you hear stories of "superhuman" performance of AI methods (it is possible, but takes a long time, and is not the norm)

Visualization vs. statistics

Visualization almost always presents a more informative (though less quantitative) view of your data than statistics (the noun, not the field)



[Source: https://twitter.com/JustinMatejka/status/770682771656368128 Credit: @JustinMatejka, @albertocairo]

This is a mathematical property: n data points and m equations to satisfy, with $n > m$

Outline

Basics of visualization

Data types and visualization types

Software plotting libraries

Data types

Nominal: categorical data, no ordering

Example – Pet: {dog, cat, rabbit, …}

Operations: $=, \neq$

Ordinal: categorical data, with ordering

Example – Rating: {1,2,3,4,5}

Operations: $=, \neq, \geq, \leq, >, <$

Interval: numerical data, zero has no fixed meaning

Example – Temperature Fahrenheit

Operations: $=, \neq, \geq, \leq, >, <, +, -$

Ratio: numerical data, zero has special meaning

Example – Temperature Kelvin

Operations: $=, \neq, \geq, \leq, >, <, +, -, \div$

Poll: Nominal and ordinal values

Which of the following questions that may be asked on a survey would be considered *ordinal*? (unchecked ones are *nominal*)

1. Gender: {male, female, other, prefer not to disclose}

2. Yearly income: {<$18k, $18-40k, $40-75k, >$75k}

3. Reaction to question: {Strongly disagree, slightly disagree, neutral, slightly agree, strongly agree}

4. May we add you to our mailing list: {No, Yes}

Poll: Interval and ratio values

Which of the following quantities would be considered *ratio*? (unchecked values are *interval*)

1. Length (meters)
2. Length (feet)
3. Velocity (meters/second)
4. IQ Score

Visualization Types

Most discussion of visualization types emphasizes what elements the chart is trying to convey

Instead, we are going to focus on the type and dimensionality of the underlying data

Visualization types (not an exhaustive list):

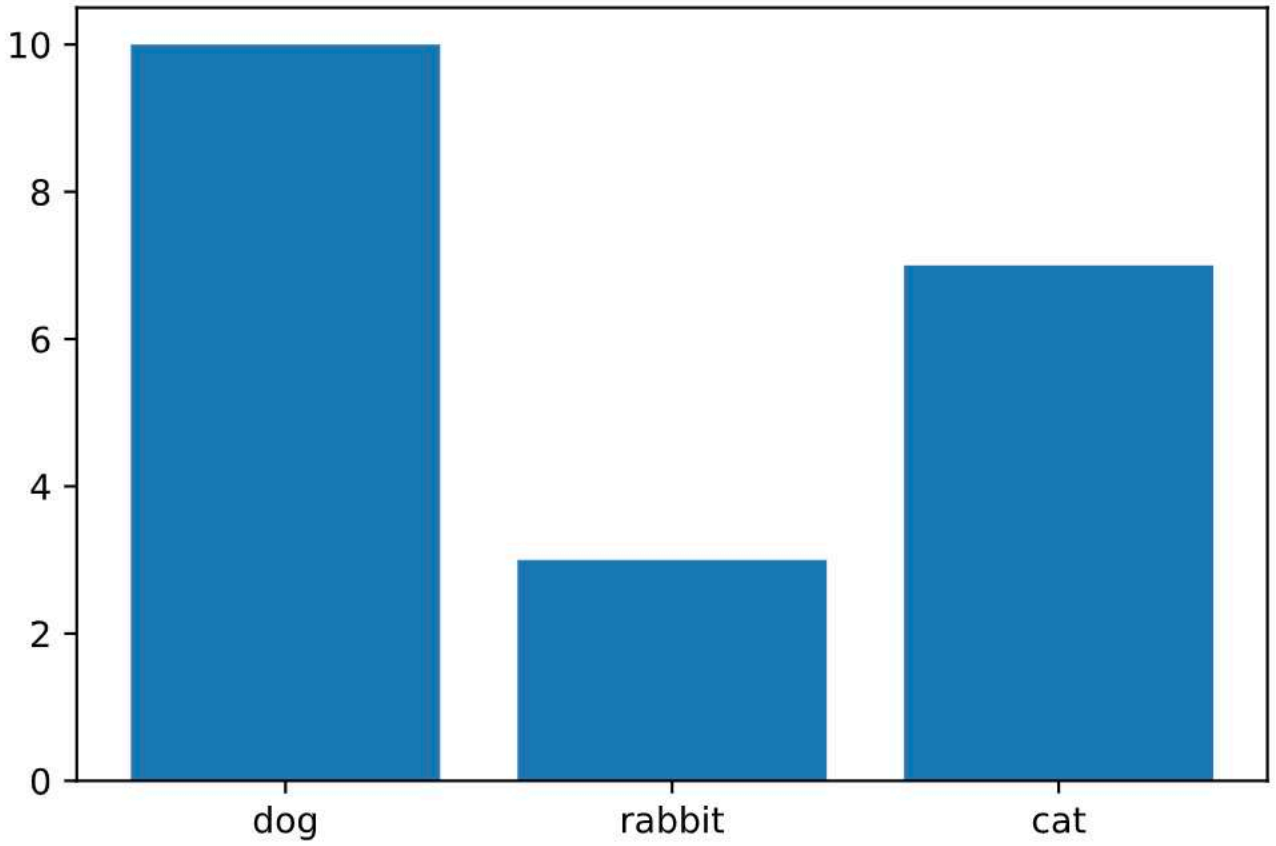
- 1D: bar chart, pie chart, histogram
- 2D: scatter plot, line plot, box and whisker plot, heatmap
- 3D+: scatter matrix, bubble chart

1D DATA

Bar chart

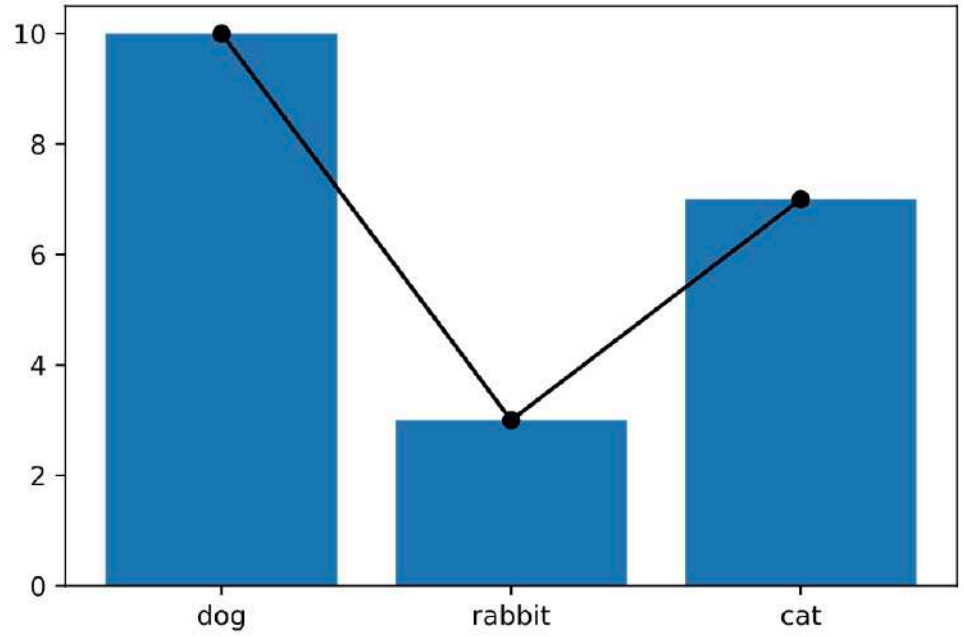
	Data
Nominal	
Ordinal	
Interval	✗
Ratio	✗

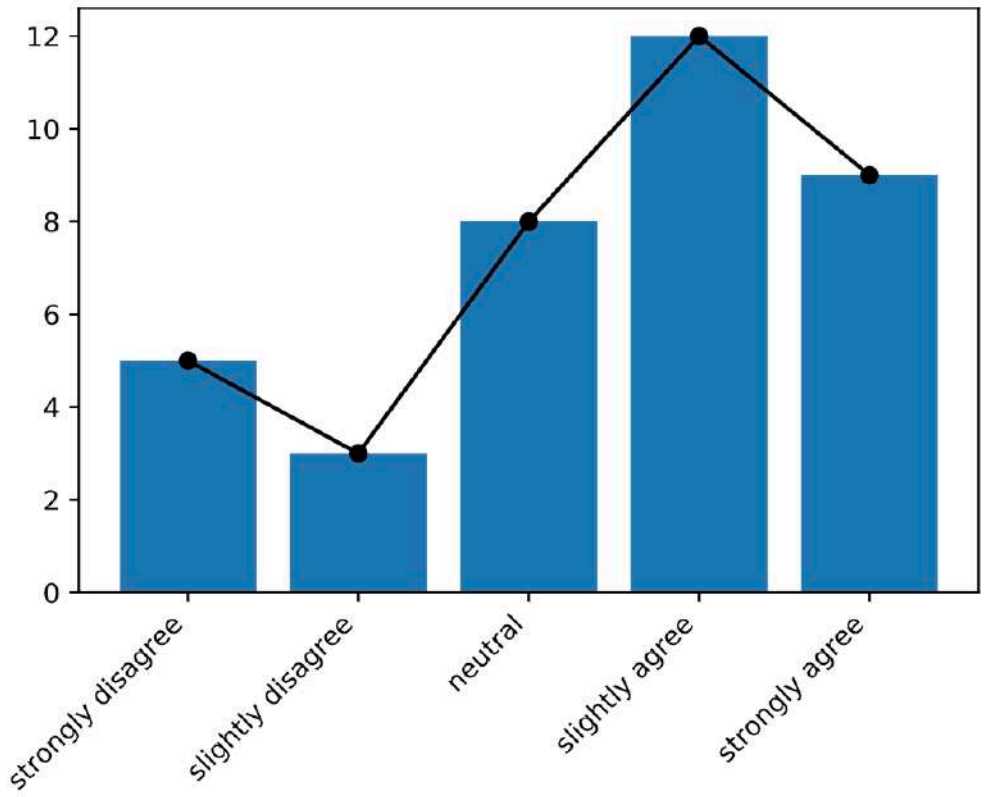
Suggestions, not rules



Bar chart (bad)

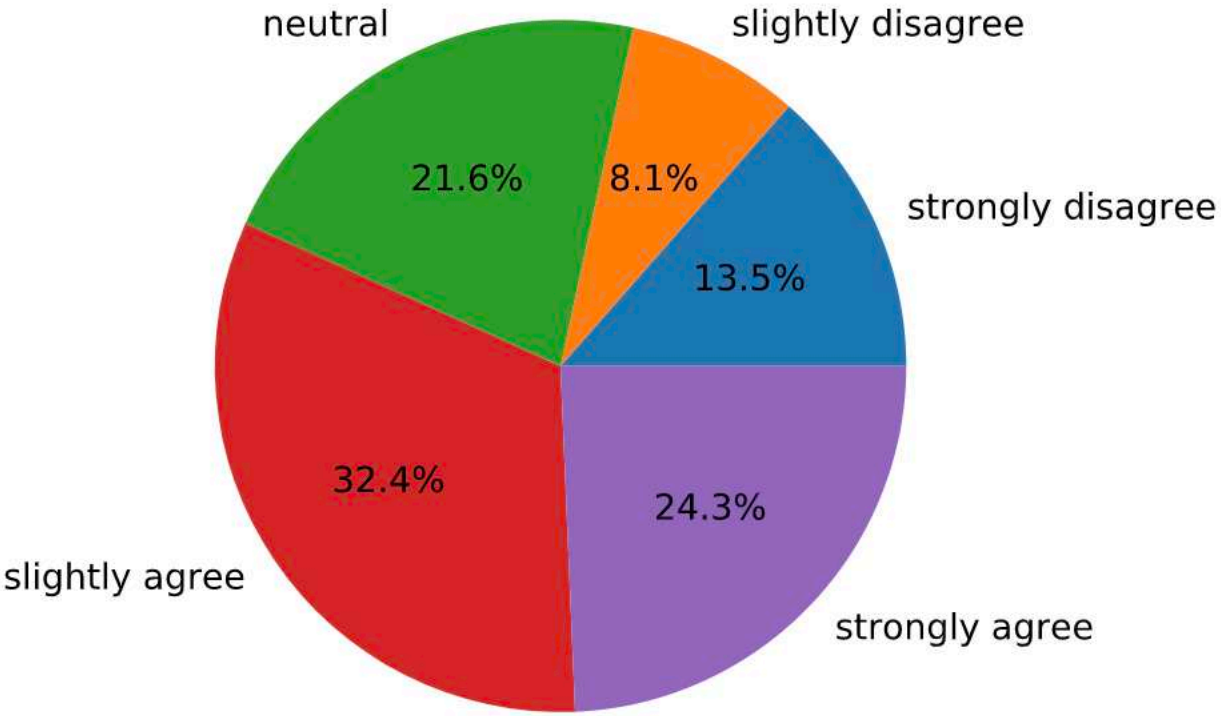
Don't use lines within a bar chart for categorial or ordinal features!





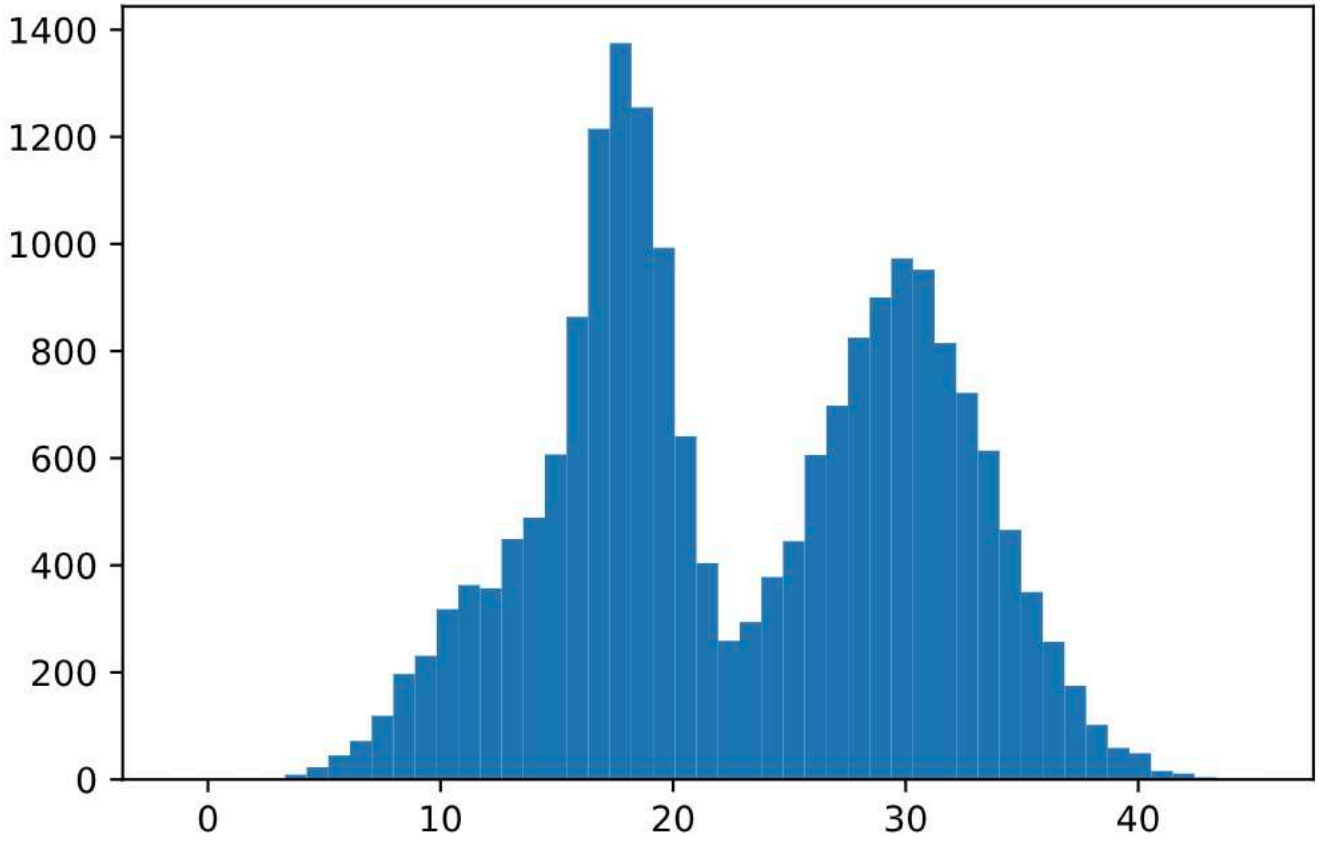
Pie chart

	Data
Nominal	✗
Ordinal	✗
Interval	✗
Ratio	✗



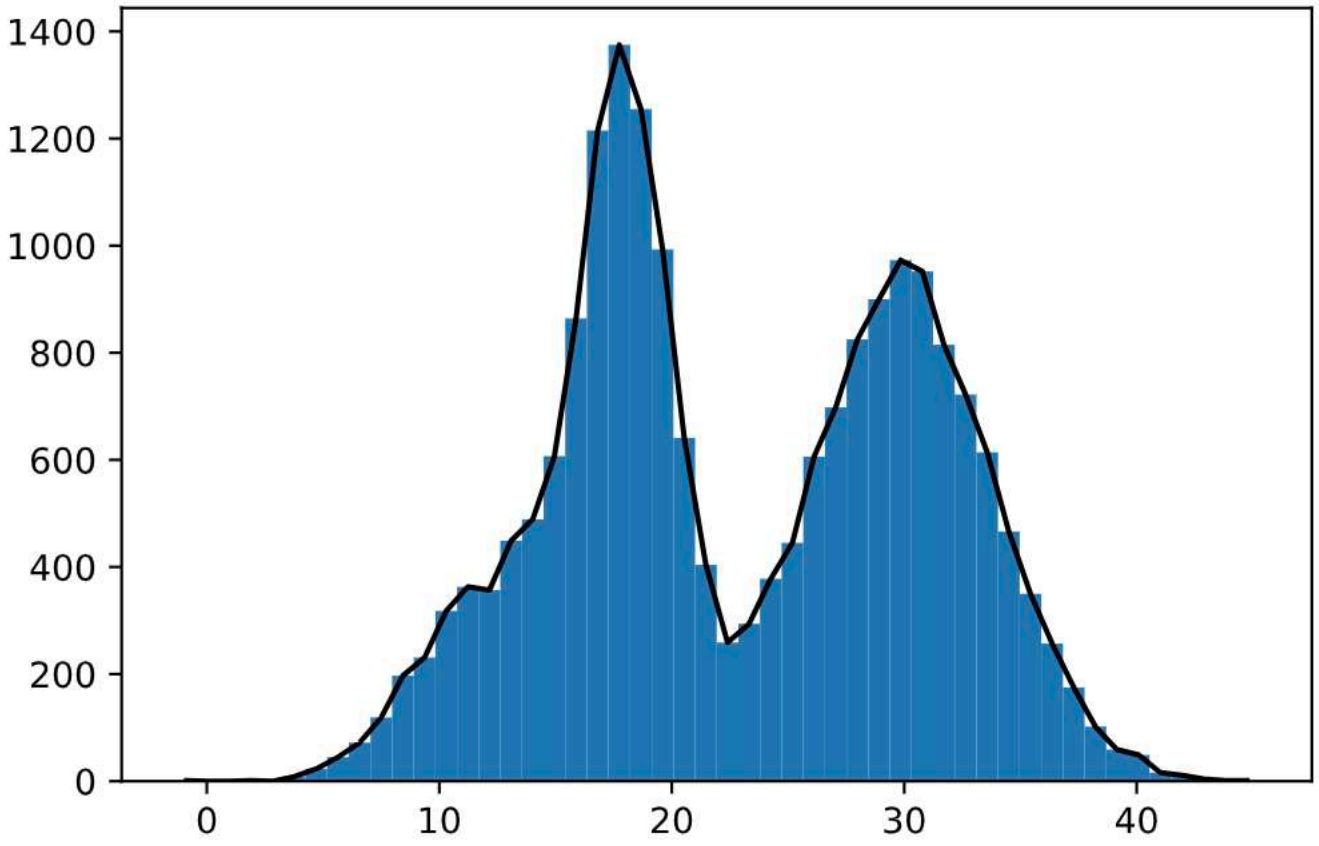
Histogram

	Data
Nominal	✗
Ordinal	✗
Interval	
Ratio	



Histogram

OK to use lines within a histogram (but not very informative)

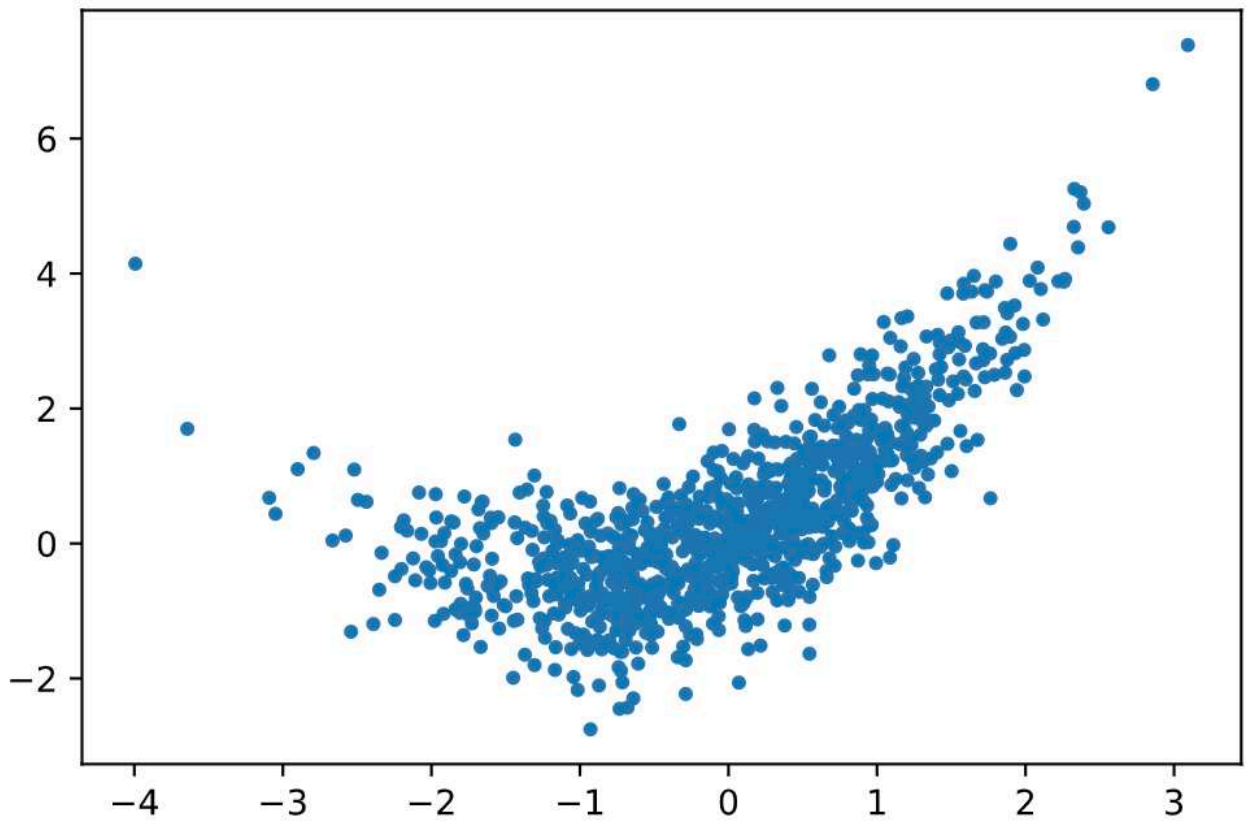


2D DATA

Scatter plot

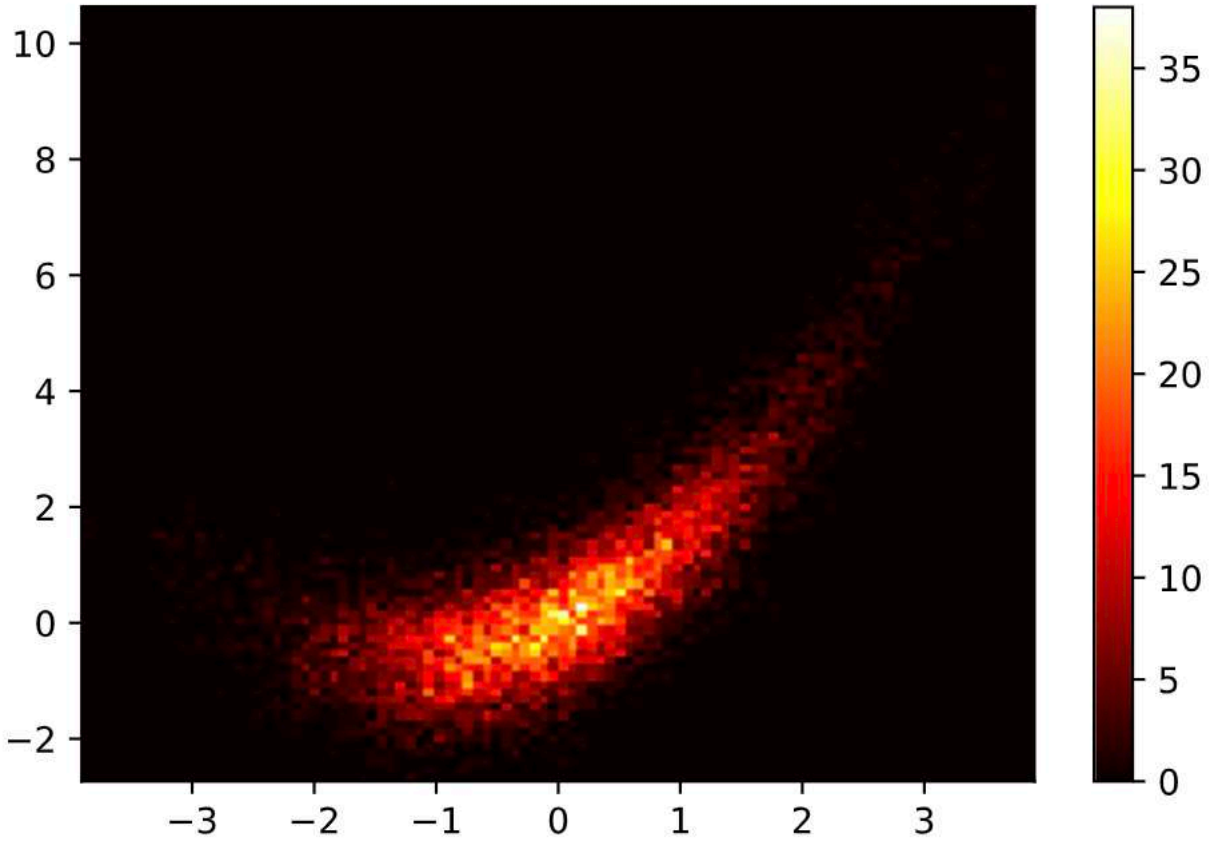
	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval		
Ratio		

Why not ordinal data in first dimension?



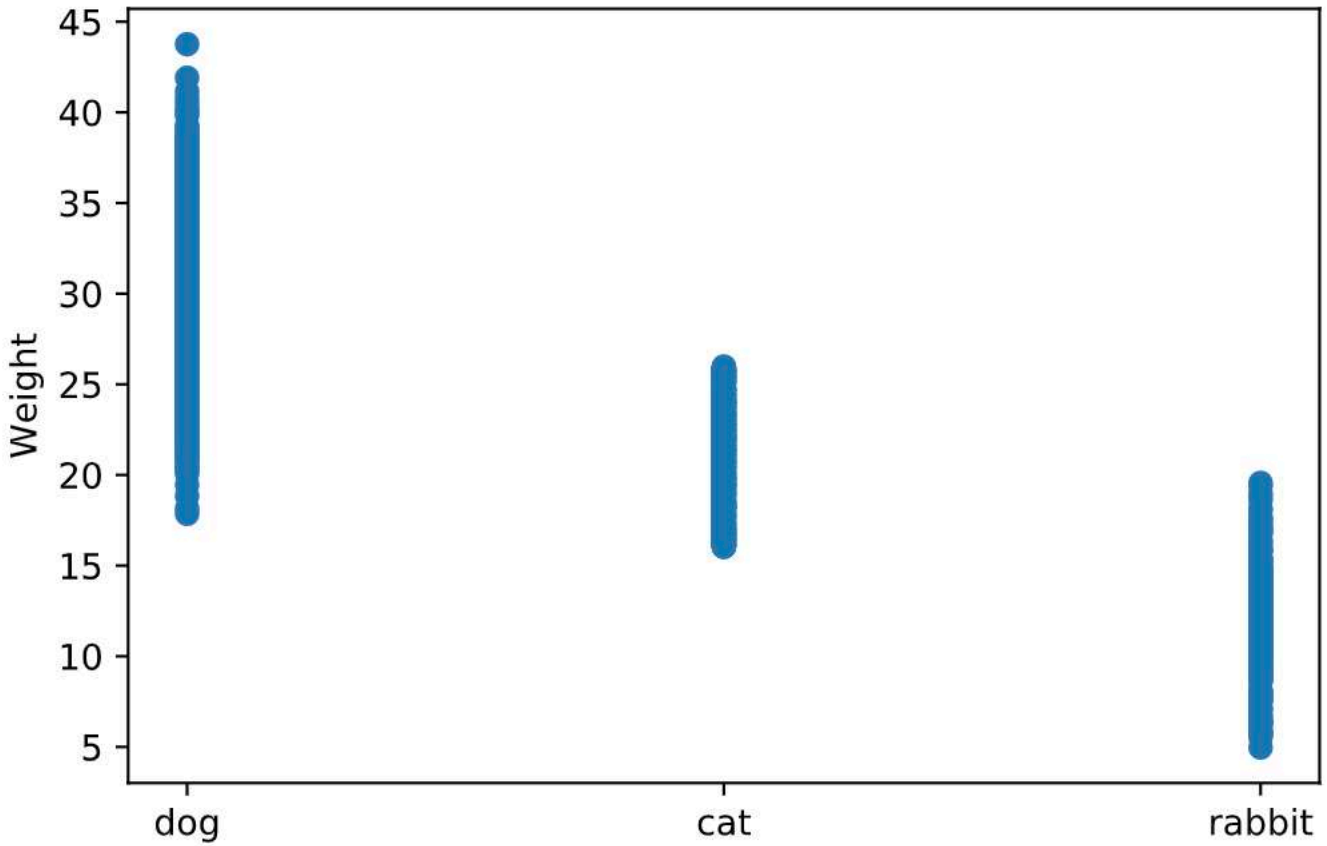
Heatmap (density, or 2D histogram)

	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval		
Ratio		



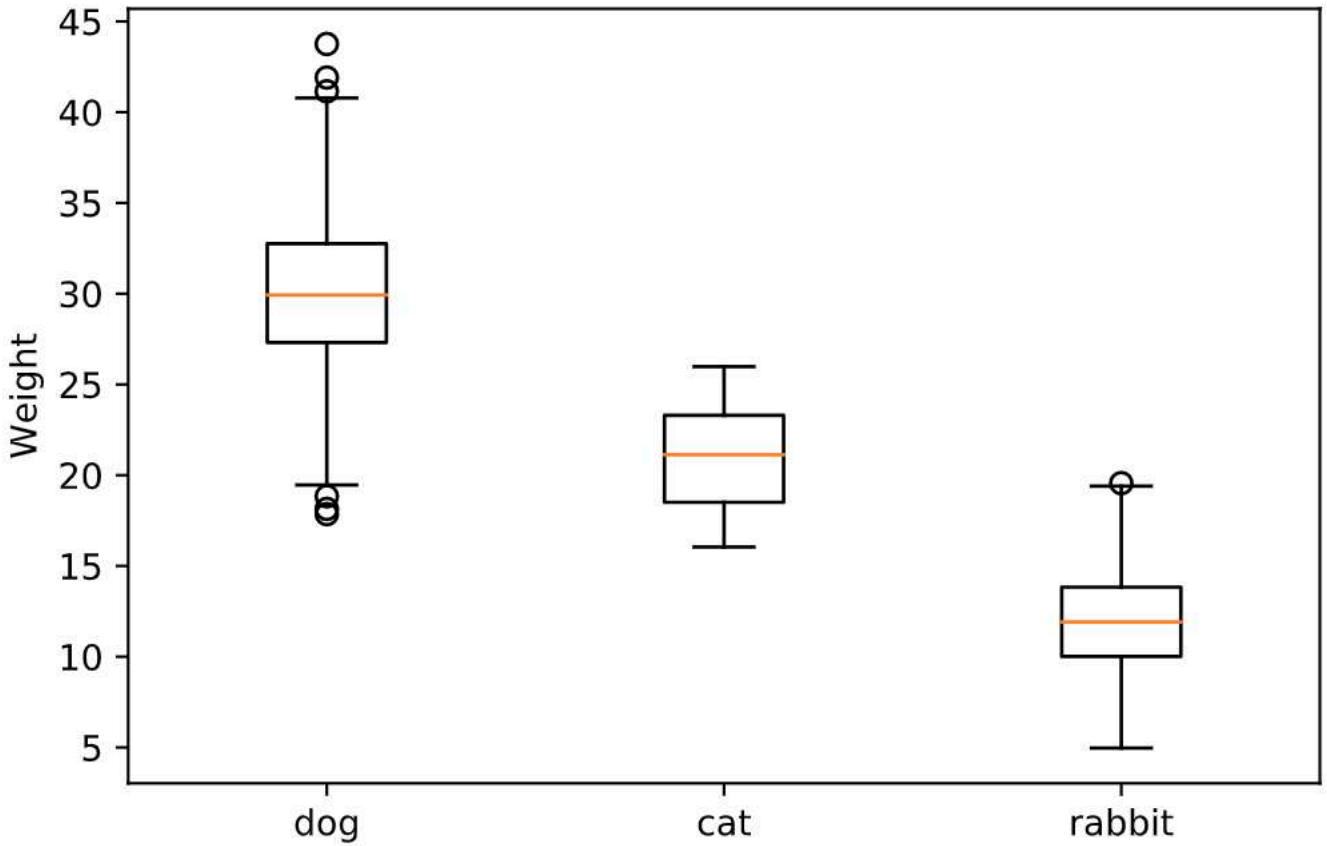
Scatter plot (bad)

	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval		
Ratio		



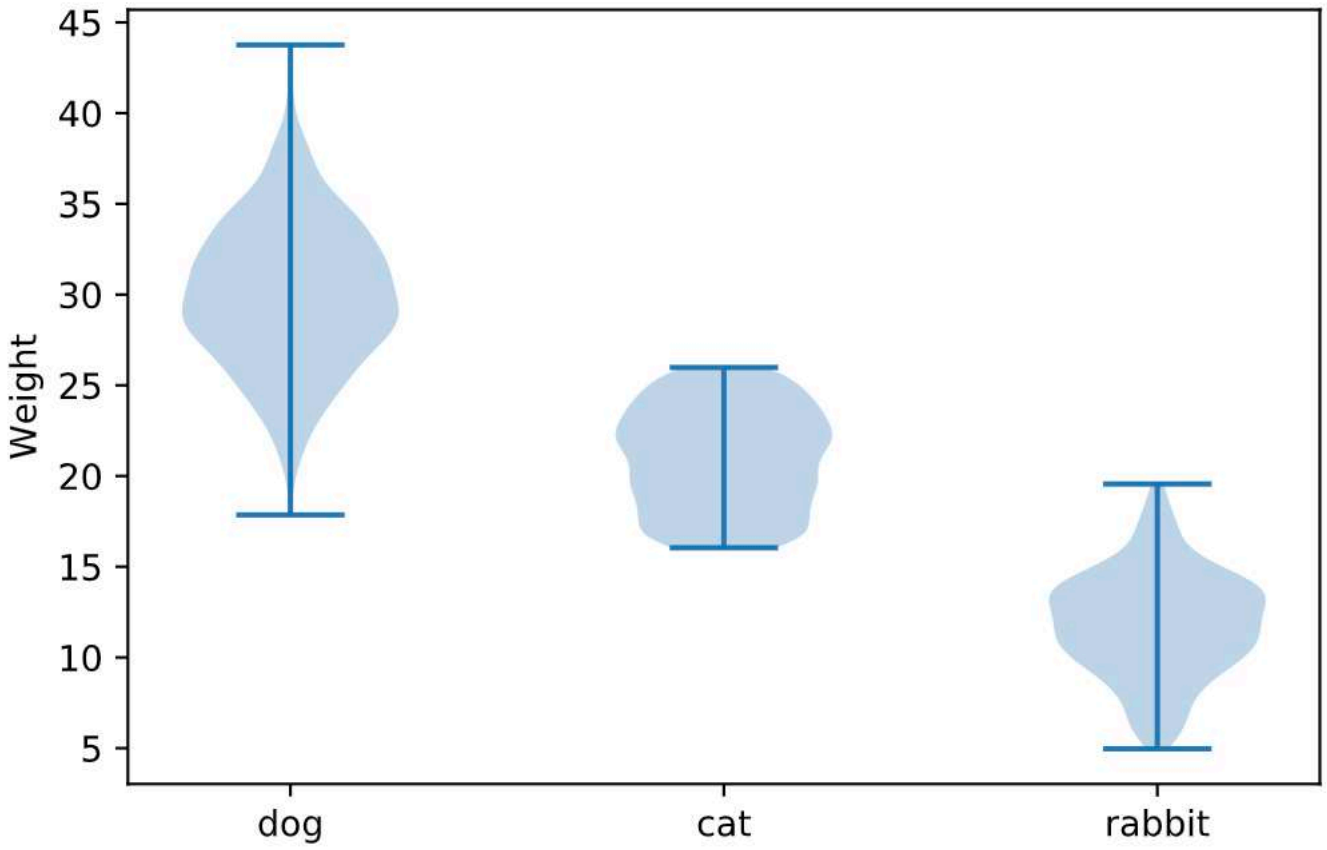
Box and whiskers

	Dim 1	Dim 2
Nominal		✗
Ordinal		✗
Interval	✗	
Ratio	✗	



Violin plot

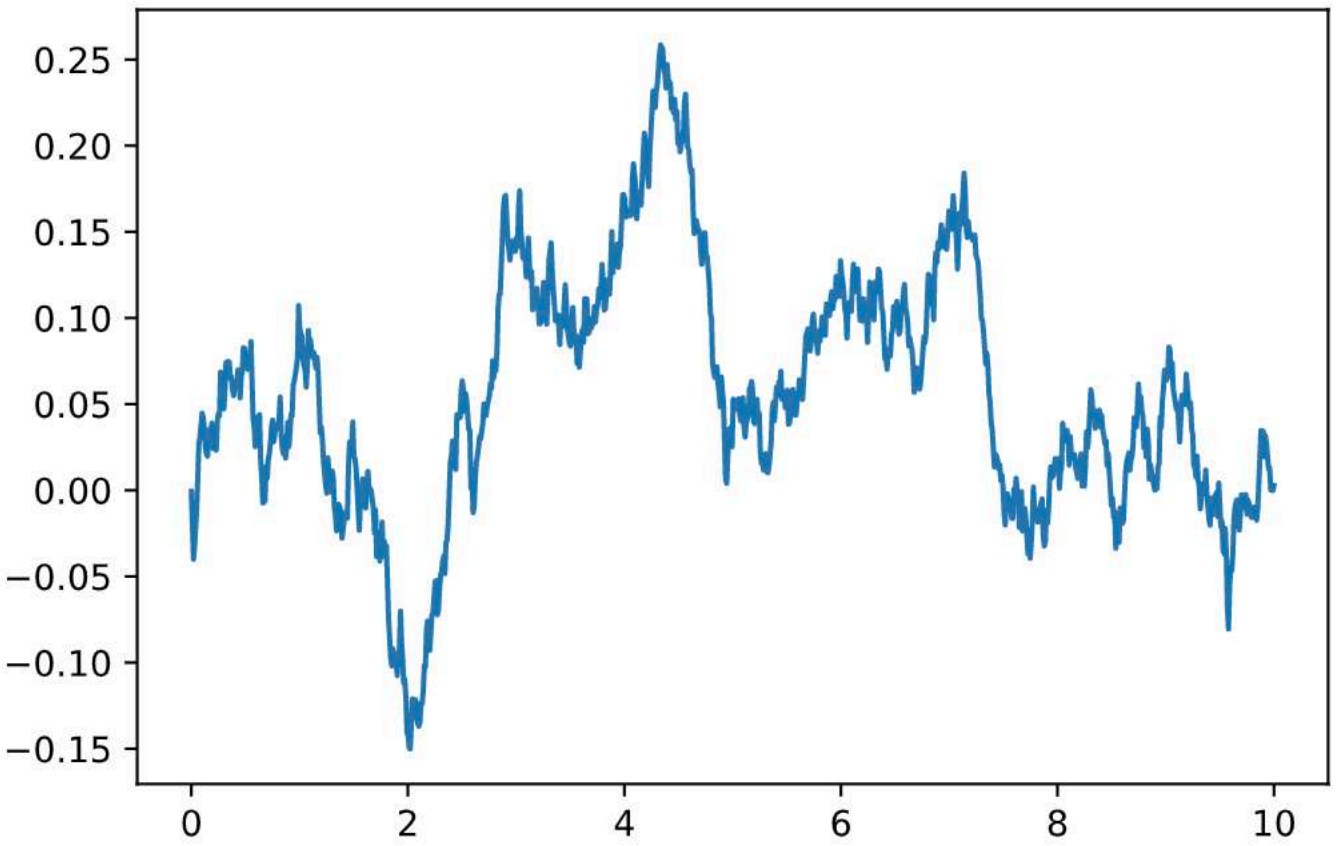
	Dim 1	Dim 2
Nominal		✗
Ordinal		✗
Interval	✗	
Ratio	✗	



Line plot

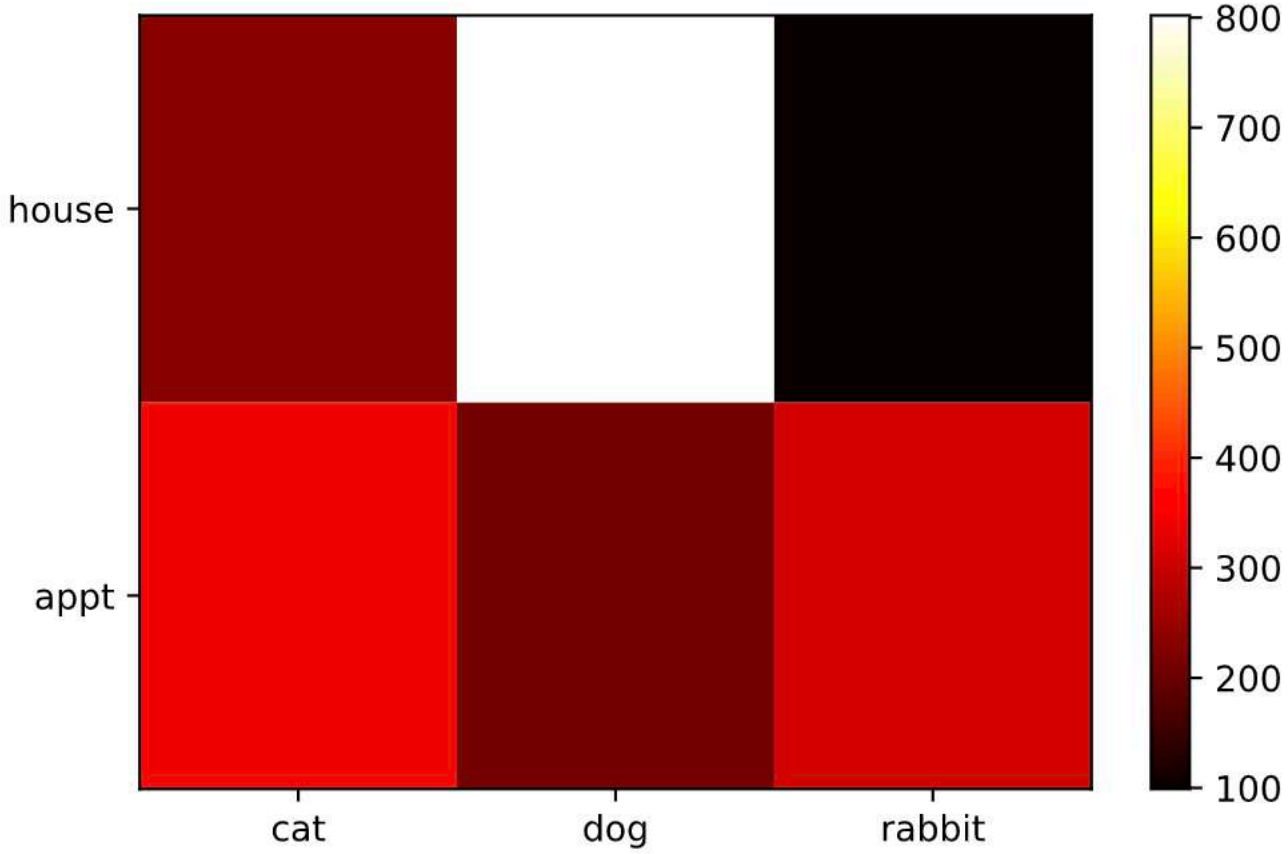
	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval		
Ratio		

Why not ordinal data in first dimension?



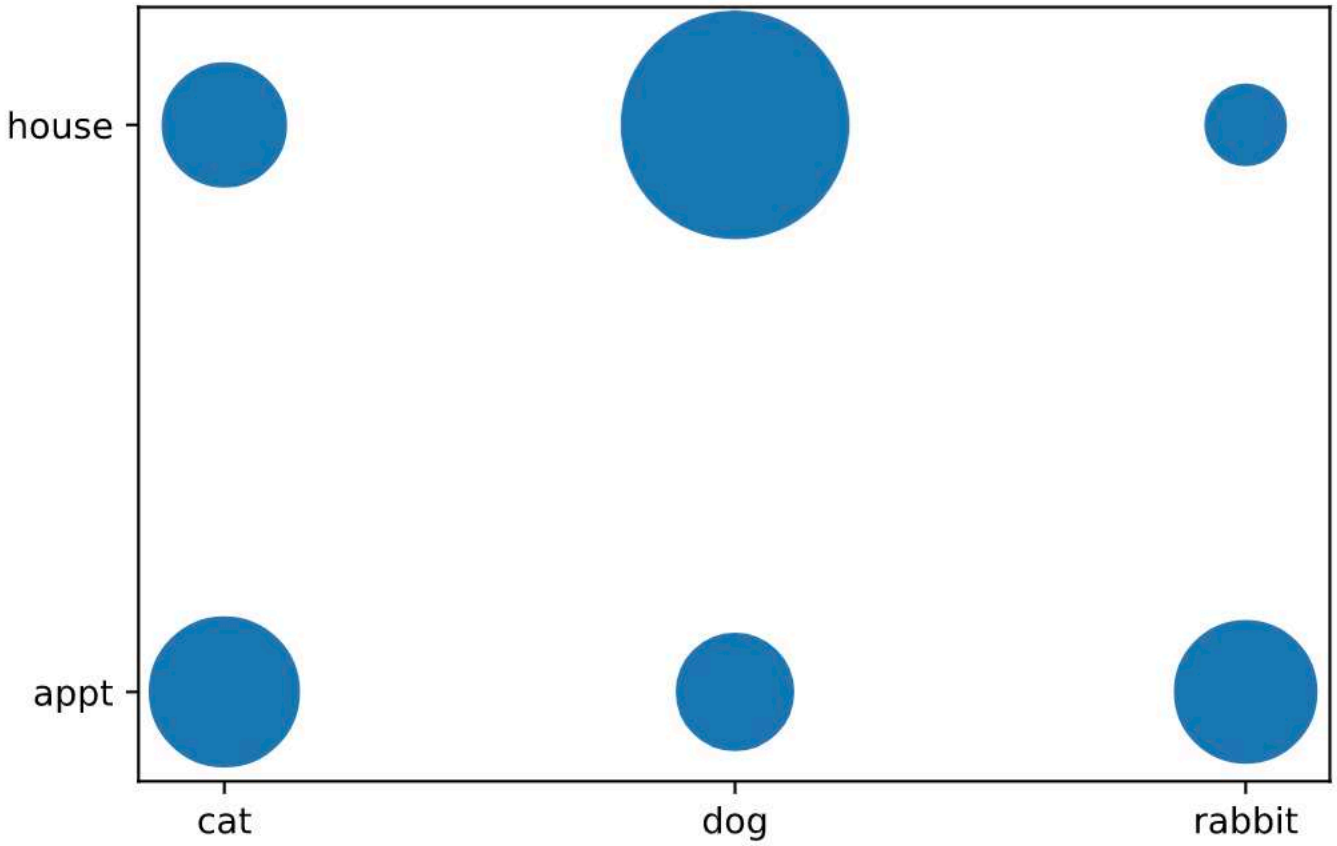
Heatmap (matrix)

	Dim 1	Dim 2
Nominal		
Ordinal		
Interval	X	X
Ratio	X	X



Bubble plot

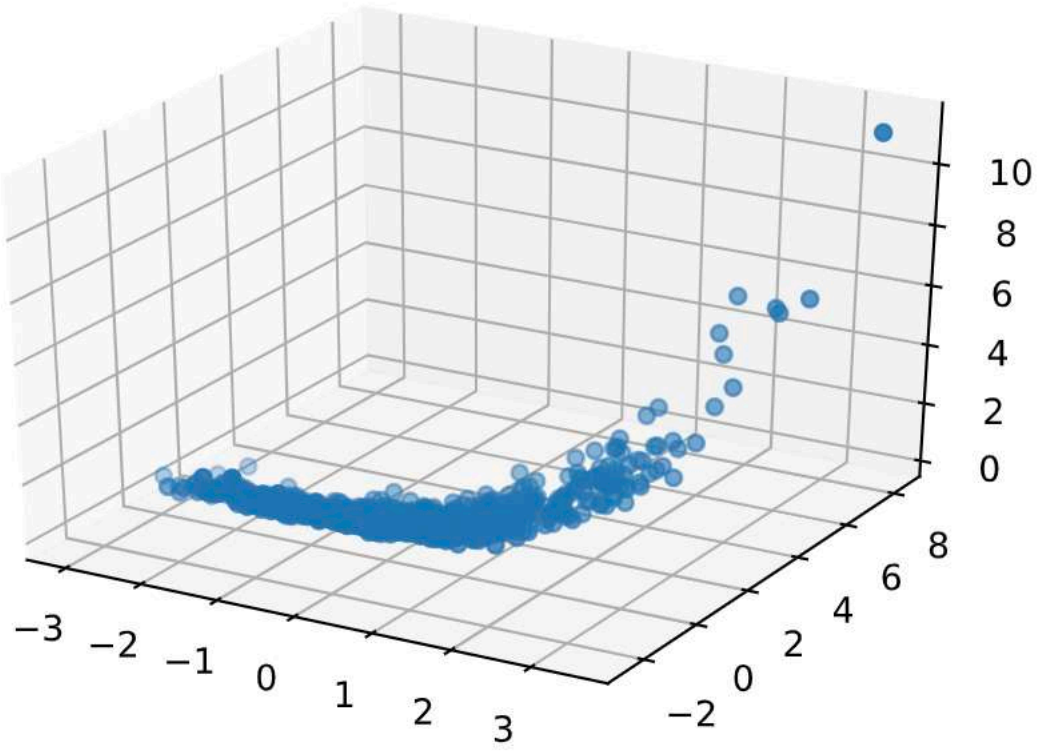
	Dim 1	Dim 2
Nominal		
Ordinal		
Interval	X	X
Ratio	X	X



3D+ DATA

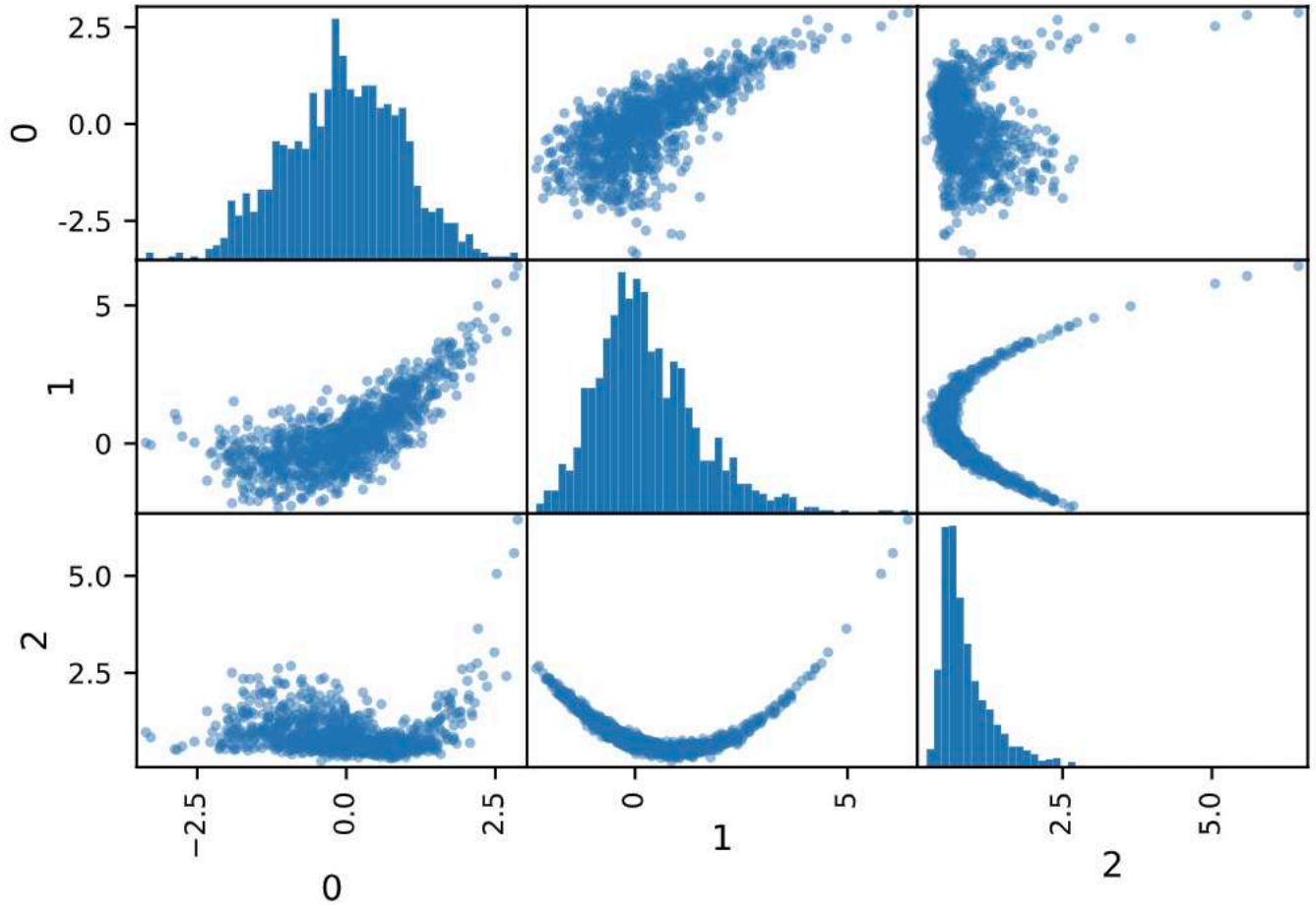
3D scatter plot

	Dim 1	Dim 2	Dim 3
Nominal	✗	✗	✗
Ordinal	✗	✗	✗
Interval	✗	✗	✗
Ratio	✗	✗	✗



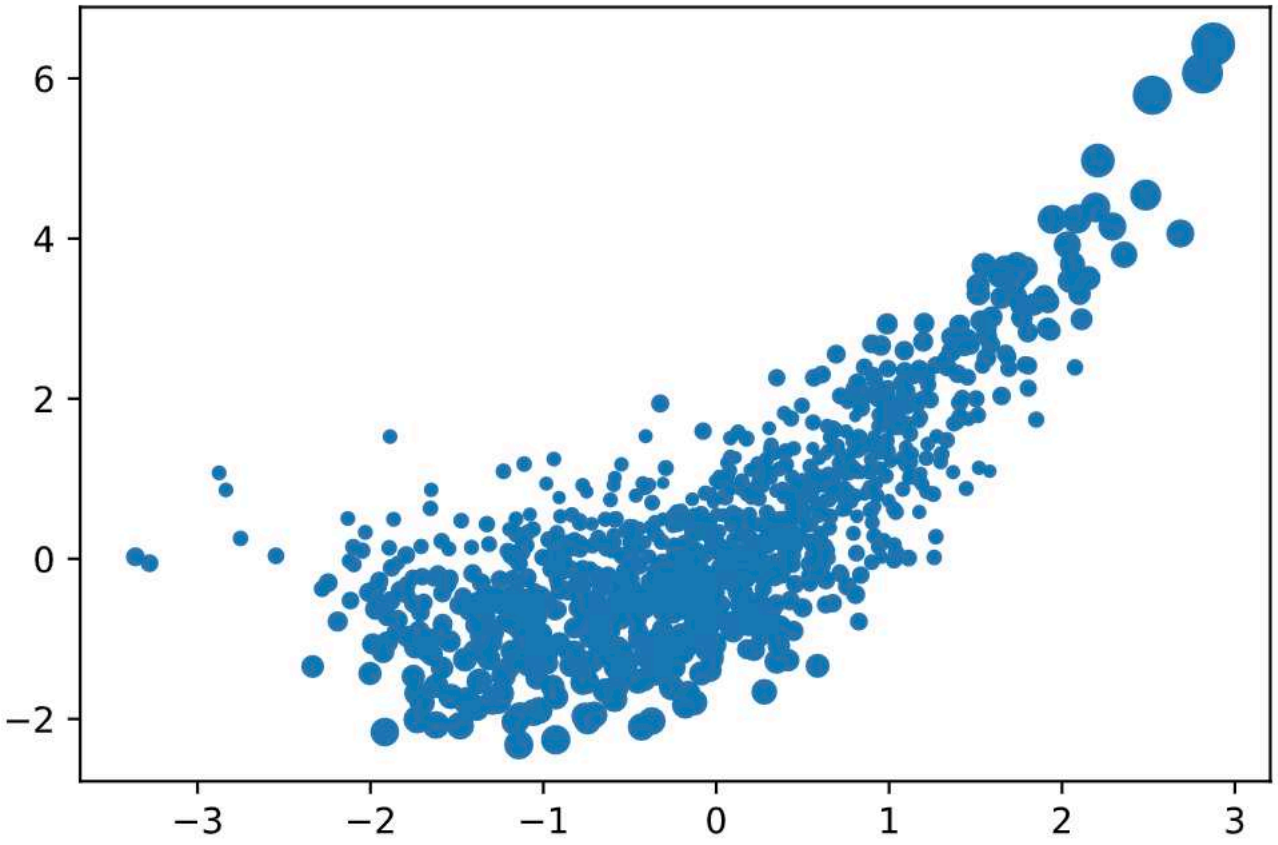
Scatter plot matrix

	Dim 1	Dim 2	Dim 3
Nominal	X	X	X
Ordinal	X	X	X
Interval			
Ratio			



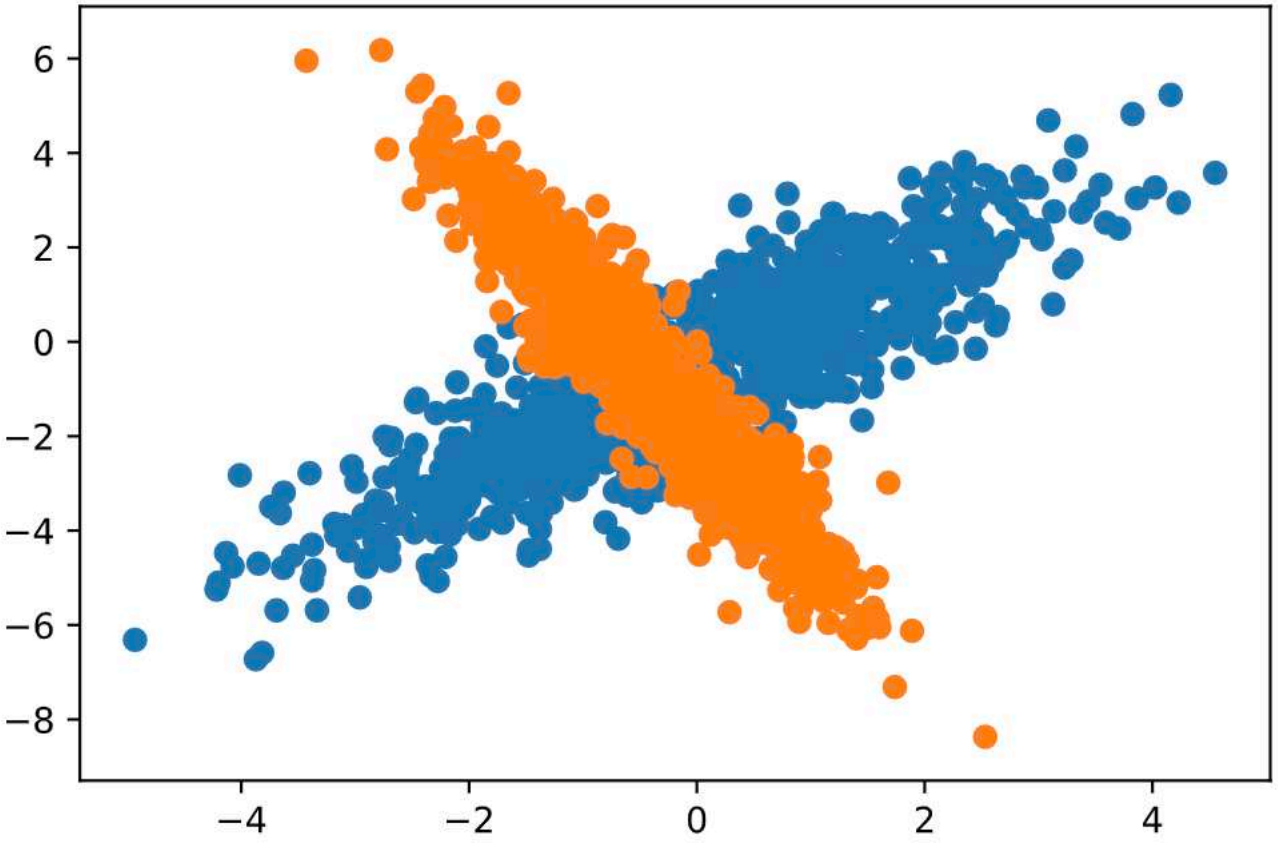
Bubble plot

	Dim 1	Dim 2	Dim 3
Nominal	X	X	X
Ordinal	X	X	X
Interval			
Ratio			



Color scatter plot

	Dim 1	Dim 2	Dim 3
Nominal	X	X	
Ordinal	X	X	
Interval			X
Ratio			X



Outline

Basics of visualization

Data types and visualization types

Software plotting libraries

Matplotlib

Matplotlib is the standard for plotting in Python / Jupyter Notebook

Matplotlib used to generate fairly ugly plots by default, but in recent versions this is no longer the case, so minimal need for additional libraries

It is aimed at generating static plots, not very good for interacting with data (with a few exceptions)

A number of additional libraries provide some level of interactive plot (and static plots), but matplotlib is enough of a standard that we'll use it here

15-388/688 - Practical Data Science: Basic probability

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Probability in data science

Basic rules of probability

Some common distributions

Outline

Probability in data science

Basic rules of probability

Some common distributions

Basic probability and statistics

Thus far, in our discussion of machine learning, we have largely avoided any talk of probability

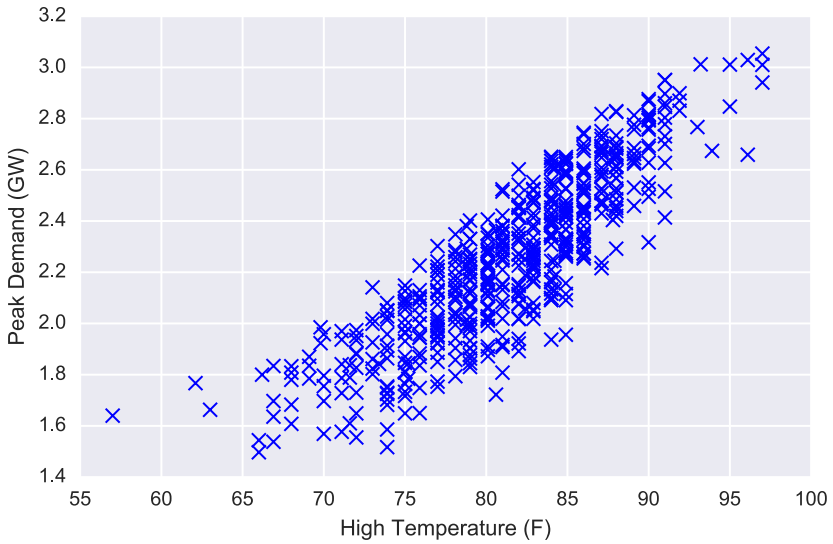
This won't be the case any longer, understanding and modeling probabilities is a crucial component of data science (and machine learning)

For the purposes of this course: statistics = probability + data

Probability and uncertainty in data science

In many prediction tasks, we never expect to be able to achieve perfect accuracy (there is some inherent randomness at the level we can observe the data)

In these situations, it is important to understand the uncertainty associated with our predictions



Outline

Probability in data science

Basic rules of probability

Some common distributions

Random variables

A random variable (informally) is a variable whose value is not initial known

Instead, these variables can take on different values (including a possibly infinite number), and must take on exactly one of these values, each with an associated probability, which all together sum to one

$$
\begin{aligned}
&\text{“Weather” takes values } \{\text{sunny}, \text{rainy}, \text{cloudy}, \text{snowy}\} \\
&p(\text{Weather} = \text{ sunny}) = 0.3 \\
&p(\text{Weather} = \text{rainy}) = 0.2 \\
&\ldots
\end{aligned}
$$

Slightly different notation for continuous random variables, which we will discuss shortly

Notation for random variables

In this lecture, we use upper case letters, X to denote random variables

For a random variable X taking values $\{1,2,3\}$

$$p(X) = \begin{cases} 1: & 0.1 \\ 2: & 0.5 \\ 3: & 0.4 \end{cases}$$

represents a mapping from values to probabilities numbers that sum to one (odd notation, would be better to use p_X, but this is not common)

Conversely, we will use lower case x to denote a specific *value* of X (i.e., for above example $x \in \{1,2,3\}$), and $p(X = x)$ or just $p(x)$ refers to a *number* (the corresponding entry of $p(X)$)

Examples of probability notation

Given two random variables: X_1 with values in $\{1,2,3\}$ and X_2 with values in $\{1,2\}$:

- $p(X_1, X_2)$ refers to the joint distribution, i.e., a set of 6 possible values for each setting of variables, i.e. a dictionary mapping $(1,1), (1,2), (2,1), \ldots$ to corresponding probabilities)
- $p(x_1, x_2)$ is a number: probability that $X_1 = x_1$ and $X_2 = x_2$
- $p(X_1, x_2)$ is a set of 3 values, the probabilities for all values of X_1 for the given value $X_2 = x_2$, i.e., it is a dictionary mapping $0,1,2$ to numbers (note: not probability distribution, it will not sum to one)

We generally call all of these terms factors (dictionaries mapping values to numbers, even if they do not sum to one)

Example: weather and cavity

Let $\mathrm{Weather}$ denote a random variable taking on values in $\{\mathrm{sunny}, \mathrm{rainy}, \mathrm{cloudy}\}$ and Cavity a random variables taking on values in $\{\mathrm{yes}, \mathrm{no}\}$

$$P(\mathrm{Weather}, \mathrm{Cavity}) = \begin{cases} \mathrm{sunny}, \mathrm{yes} & 0.07 \\ \mathrm{sunny}, \mathrm{no} & 0.63 \\ \mathrm{rainy}, \mathrm{yes} & 0.02 \\ \mathrm{rainy}, \mathrm{no} & 0.18 \\ \mathrm{cloudy}, \mathrm{yes} & 0.01 \\ \mathrm{cloudy}, \mathrm{no} & 0.09 \end{cases}$$

$$p(\mathrm{sunny}, \mathrm{yes}) = 0.07$$

$$p(\mathrm{Weather}, \mathrm{yes}) = \begin{cases} \mathrm{sunny} & 0.07 \\ \mathrm{rainy} & 0.02 \\ \mathrm{cloudy} & 0.01 \end{cases}$$

Operations on probabilities/factors

We can perform operations on probabilities/factors by performing the operation on every corresponding value in the probabilities/factors

For example, given three random variables X_1, X_2, X_3:

$$p(X_1, X_2) \ \langle \text{op} \rangle \ p(X_2, X_3)$$

denotes a factor over X_1, X_2, X_3 (i.e., a dictionary over all possible combinations of values these three random variables can take), where the value for x_1, x_2, x_3 is given by

$$p(x_1, x_2) \ \langle \text{op} \rangle \ p(x_2, x_3)$$

Conditional probability

The **conditional probability** $p(X_1|X_2)$ (the conditional probability of X_1 given X_2) is defined as

$$p(X_1|X_2) = \frac{p(X_1, X_2)}{p(X_2)}$$

Can also be written $p(X_1, X_2) = p(X_1|X_2)p(X_2)$

Marginalization

For random variables X_1, X_2 with joint distribution $p(X_1, X_2)$

$$p(X_1) = \sum_{x_2} p(X_1, x_2) = \sum_{x_2} p(X_1 | x_2) p(x_2)$$

Generalizes to joint distributions over multiple random variables

$$p(X_1, \dots, X_i) = \sum_{x_{i+1}, \dots, x_n} p(X_1, \dots, X_i, x_{i+1}, \dots, x_n)$$

For p to be a probability distribution, the marginalization over *all* variables must be one

$$\sum_{x_1, \dots, x_n} p(x_1, \dots, x_n) = 1$$

Bayes' rule

A straightforward manipulation of probabilities:

$$p(X_1|X_2) = \frac{p(X_1, X_2)}{p(X_2)} = \frac{p(X_2|X_1)p(X_1)}{p(X_2)} = \frac{p(X_2|X_1)p(X_1)}{\sum_{x_1} p(X_2|x_1)\, p(x_1)}$$

Poll: I want to know if I have come with with a rate strain of flu (occurring in only 1/10,000 people). There is an "accurate" test for the flu (if I have the flu, it will tell me I have 99% of the time, and if I do not have it, it will tell me I do not have it 99% of the time). I go to the doctor and test positive. What is the probability I have the this flu?

Bayes' rule

BBC Sign in News Sport Weather Shop Earth Travel

NEWS

Home Video World US & Canada UK Business Tech Science Stories

Enjoy up to 40% off base rates and up to 2,750 miles.
Terms apply.
Book with this offer
Hertz MileagePlus UNITED

Magazine

Do doctors understand test results?

By William Kremer
BBC World Service

7 July 2014 Share

In one session, almost half the group of 160 gynaecologists responded that the woman's chance of having cancer was nine in 10. Only 21% said that the figure was one in 10 - which is the correct answer. That's a worse result than if the doctors had been answering at random.

The fact that 90% of women with breast cancer get a positive result from a mammogram doesn't mean that 90% of women with positive results have breast cancer. The high false alarm rate, combined with the disease's prevalence of 1%, means that roughly nine out of 10 women with a worrying mammogram don't actually have breast cancer.

Independence

We say that random variables X_1 and X_2 are **(marginally) independent** if their joint distribution is the product of their marginals

$$p(X_1, X_2) = p(X_1)p(X_2)$$

Equivalently, can also be stated as the condition that

$$p(X_1|X_2) \ \left(= \frac{p(X_1, X_2)}{p(X_2)} = \frac{p(X_1)p(X_2)}{p(X_2)}\right) = p(X_1)$$

$$(\text{and similarly}) \ \ p(X_2|X_1) = p(X_2)$$

Poll: Weather and cavity

Are the weather and cavity random variables independent?

$$P(\text{Weather}, \text{Cavity}) = \begin{cases} \text{sunny}, \text{yes} & 0.07 \\ \text{sunny}, \text{no} & 0.63 \\ \text{rainy}, \text{yes} & 0.02 \\ \text{rainy}, \text{no} & 0.18 \\ \text{cloudy}, \text{yes} & 0.01 \\ \text{cloudy}, \text{no} & 0.09 \end{cases}$$

Conditional independence

We say that random variables X_1 and X_2 are **conditionally independent given** X_3, if

$$p(X_1, X_2|X_3) = p(X_1|X_3)p(X_2|X_3)$$

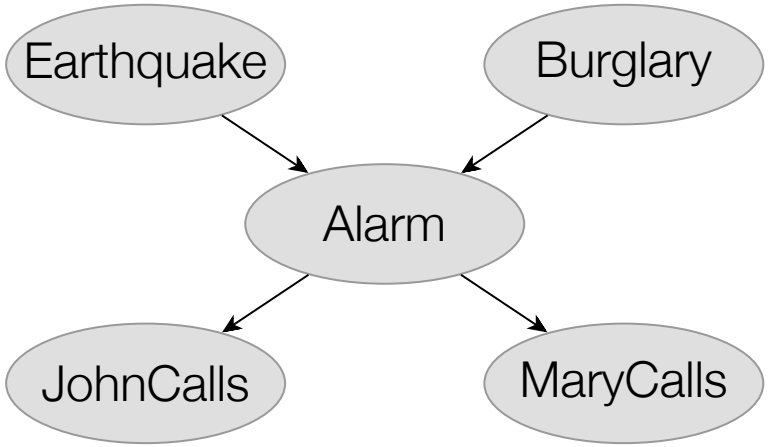
Again, can be equivalently written:

$$p(X_1|X_2, \mathrm{X}_3) \left(= \frac{p(X_1, X_2|X_3)}{p(X_2|X_3)} = \frac{p(X_1|X_3)p(X_2|X_3)}{p(X_2|X_3)} \right) = p(X_1|X_3)$$

And similarly $p(X_2|X_1, X_3) = p(X_2|X_3)$

Marginal and conditional independence

Important: Marginal independence does not imply conditional independence or vice versa



$$P(\text{Earthquake}|\text{Burglary}) = P(\text{Earthquake}) \text{ but}$$
$$P(\text{Earthquake}|\text{Burglary}, \text{Alarm}) \neq P(\text{Earthquake}|\text{Alarm})$$

$$P(\text{JohnCalls}|\text{MaryCalls}, \text{Alarm}) = P(\text{JohnCalls}|\text{Alarm}) \text{ but}$$
$$P(\text{JohnCalls}|\text{MaryCalls}) \neq P(\text{JohnCalls})$$

Expectation

The expectation of a random variable is denoted:

$$\mathbf{E}[X] = \sum_{x} x \cdot p(x)$$

where we use upper case X to emphasize that this is a function of the entire random variable (but unlike $p(X)$ is a number)

Note that this only makes sense when the values that the random variable takes on are *numerical* (i.e., We can't ask for the expectation of the random variable "Weather")

Also generalizes to *conditional expectation:*

$$\mathbf{E}[X_1|x_2] = \sum_{x_1} x_1 \cdot p(x_1|x_2)$$

Rules of expectation

Expectation of sum is always equal to sum of expectations (even when variables are not independent):

$$
\begin{aligned}
\mathbf{E}[X_1 + X_2] &= \sum_{x_1, x_2} (x_1 + x_2) p(x_1, x_2) \\
&= \sum_{x_1} x_1 \sum_{x_2} p(x_1, x_2) + \sum_{x_2} x_2 \sum_{x_1} p(x_1, x_2) \\
&= \sum_{x_1} x_1 p(x_1) + \sum_{x_2} x_2 p(x_2) \\
&= \mathbf{E}[X_1] + \mathbf{E}[X_2]
\end{aligned}
$$

Rules of expectation

If x_1, x_2 independent, expectation of products is product of expectations

$$\begin{aligned}
\mathbf{E}[X_1 X_2] &= \sum_{x_1, x_2} x_1 x_2 \, p(x_1, x_2) \\
&= \sum_{x_1, x_2} x_1 x_2 \, p(x_1) p(x_2) \\
&= \sum_{x_1} x_1 p(x_1) \sum_{x_2} x_2 p(x_2) \\
&= \mathbf{E}[X_1]\mathbf{E}[X_2]
\end{aligned}$$

Variance

Variance of a random variable is the expectation of the variable minus its expectation, squared

$$\begin{aligned}\mathbf{Var}[X] &= \mathbf{E}[(X - \mathbf{E}[X])^2] \left(= \sum_x (x - \mathbf{E}[x])^2 p(x) \right) \\ &= \mathbf{E}[X^2 - 2X\mathbf{E}[X] + \mathbf{E}[X]^2] \\ &= \mathbf{E}[X^2] - \mathbf{E}[X]^2\end{aligned}$$

Generalizes to covariance between two random variables

$$\begin{aligned}\mathbf{Cov}[X_1, X_2] &= \mathbf{E}[(X_1 - \mathbf{E}[X_1])(X_2 - \mathbf{E}[X_2])] \\ &= \mathbf{E}[X_1 X_2] - \mathbf{E}[X_1]\mathbf{E}[X_2]\end{aligned}$$

Infinite random variables

All the math above works the same for discrete random variables that can take on an infinite number of values (for those with some math background, I'm talking about *countably infinite* values here)

The only difference is that $p(X)$ (obviously) cannot be specified by an explicit dictionary mapping variable values to probabilities, need to specify a *function* that produces probabilities

To be a probability, we still must have $\sum_x p(x) = 1$

Example:

$$P(X = k) = \left(\frac{1}{2}\right)^k, \qquad k = 1, \ldots, \infty$$

Continuous random variables

For random variables taking on *continuous* values (we'll only consider real-valued distributions), we need some slightly different mechanisms

As with infinite discrete variables, the distribution $p(X)$ needs to be specified as a function: here is referred to as a **probability density function** (PDF) and it must *integrate* to one $\int_{\mathbb{R}} p(x)dx = 1$

For any interval (a, b), we have that $p(a \leq x \leq b) = \int_a^b p(x)dx$ (with similar generalization to multi-dimensional random variables)

Can also be specified by their **cumulative distribution function** (CDF), $F(a) = p(x \leq a) = \int_{\infty}^{a} p(x)$

Outline

Probability in data science

Basic rules of probability

Some common distributions

Bernoulli distribution

A simple distribution over binary $\{0,1\}$ random variables

$$p(X = 1; \phi) = \phi, \qquad P(X = 0; \phi) = 1 - \phi$$

where $\phi \in [0,1]$ is the parameter that governs the distribution

Expectation is just $\mathbf{E}[x] = \phi$ (but not very common to refer to it this way, since this would imply that the $\{0,1\}$ terms are actual real-valued numbers)

Categorical distribution

This is the discrete distribution we've mainly considered so far, a distribute over finite discrete elements with each probability specified

Written generically as:

$$p(X = i; \phi) = \phi_i$$

where $\phi_1, \ldots \phi_k \in [0,1]$ are the parameters of the distribution (the probability of each random variable, must sum to one)

Note: we could actually parameterize just using $\phi_1, \ldots \phi_{k-1}$, since this would determine the last elements

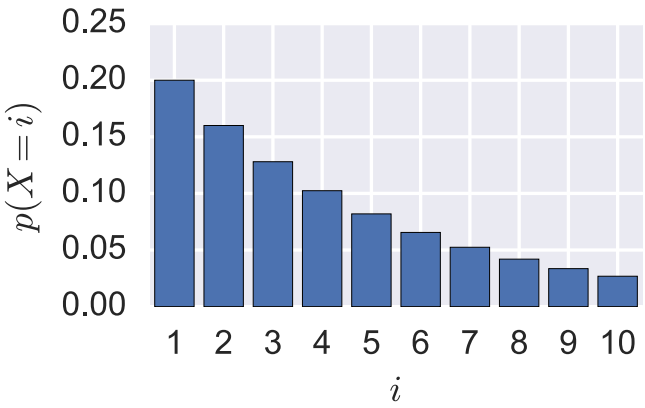
Unless the actual numerical value of the i's are relevant, it doesn't make sense to take expectations of a categorical random variable

Geometric distribution

The geometric distribution is an distribution over the positive integers, can be viewed as the number of Bernoulli trials needed before we get a "1"

$$p(X = i; \phi) = (1 - \phi)^{i-1}\phi, \qquad i = 1, \ldots, \infty$$

where $\phi \in [0,1]$ is parameter governing distribution (also $\mathbf{E}[X] = 1/\phi$)



$\phi = 0.2$

Note: easy to check that

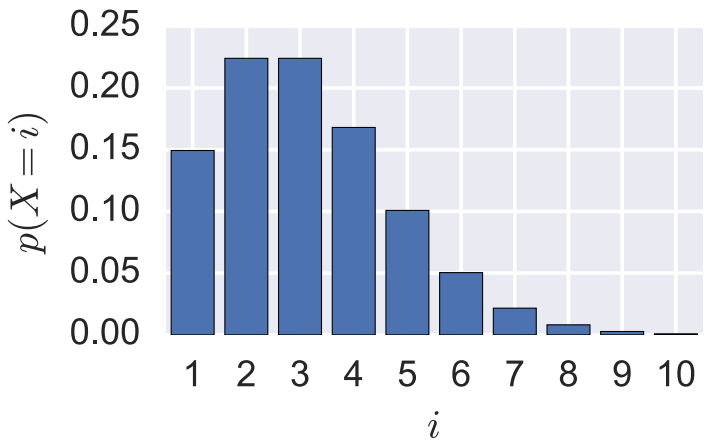
$$\sum_{i=1}^{\infty} p(X = i) = \phi \sum_{i=1}^{\infty} (1 - \phi)^{i-1} = \phi \cdot \frac{1}{1 - (1 - \phi)} = 1$$

Poisson distribution

Distribution over non-negative integers, popular for modeling number of times an event occurs within some interval

$$P(X = i; \lambda) = \frac{\lambda^i e^{-\lambda}}{i!}, \qquad i = 0, \ldots, \infty$$

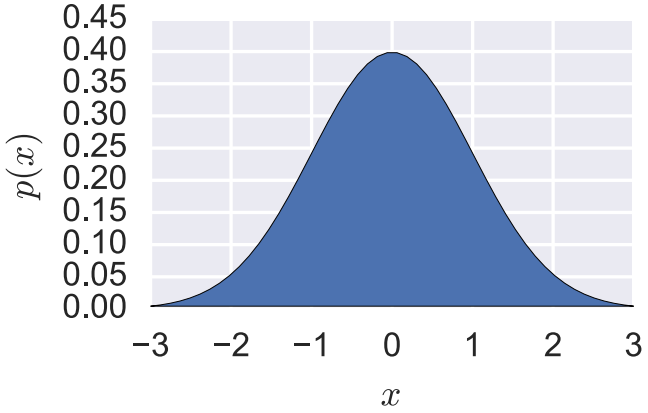
where $\lambda \in \mathbb{R}$ is parameter governing distribution (also $\mathbf{E}[X] = \lambda$)



$\lambda = 3$

Gaussian distribution

Distribution over real-valued numbers, empirically the most common distribution in all of data science (not in data itself, necessarily, but for people applying data science), the standard “bell curve”:



$\mu = 0$
$\sigma^2 = 1$

Probability density function:

$$p(x; \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \equiv \mathcal{N}(x; \mu, \sigma^2)$$

with parameters $\mu \in \mathbb{R}$ (mean) and $\sigma^2 \in \mathbb{R}_+$ (variance)

Multivariate Gaussians

The Gaussian distribution is one of the few distributions that generalizes nicely to higher dimensions

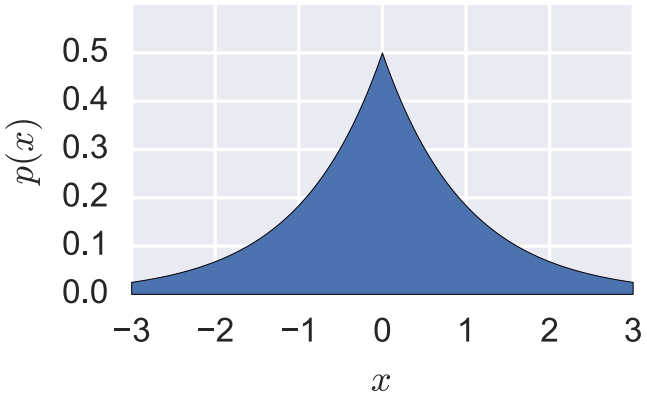
We'll discuss this in much more detail when we talk about anomaly detection and the mixture of Gaussians model, but for now, just know that we can also write a distribution over random *vectors* $x \in \mathbb{R}^n$

$$p(x; \mu, \Sigma) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\Big(-(x-\mu)^T \Sigma^{-1} (x-\mu)\Big)$$

where $\mu \in \mathbb{R}^n$ is mean and $\Sigma \in \mathbb{R}^{n \times n}$ is *covariance matrix,* and $|\cdot|$ denotes the determinant of a matrix

Laplace distribution

Like a Gaussian but with absolute instead of squared difference, gives the distribution (relatively) “heavy tails”



$\mu = 0$
$b = 1$

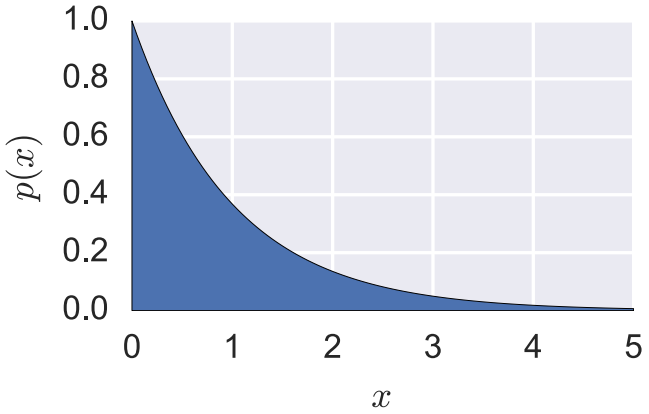
Probability density function:

$$p(x; \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right)$$

with parameters μ (mean), b (variance is $2b^2$)

Exponential distribution

A one-sided Laplace distribution, often used to model arrival times



$\lambda = 1$

Probability density function:

$$p(x; \lambda) = \lambda \exp(-\lambda x)$$

with parameter $\lambda \in \mathbb{R}_+$ (mean/variance $\mathbf{E}[X] = 1/\lambda, \mathbf{Var}[x] = 1/\lambda^2$)

Some additional examples

Student's t distribution – distribution governing estimation of normal distribution from finite samples, commonly used in hypothesis testing

χ^2 (chi-squared) distribution – distribution of Gaussian variable squared, also used in hypothesis testing

Cauchy distribution – very heavy tailed distribution, to the point that variables have undefined expectation (the associated integral is undefined)

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

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Classification tasks

Regression tasks: predicting real-valued quantity $y \in \mathbb{R}$

Classification tasks: predicting *discrete-valued* quantity y

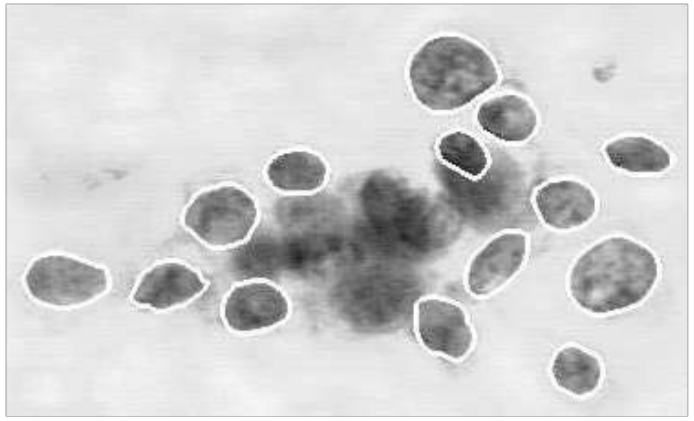
Binary classification: $y \in \{-1, +1\}$

Multiclass classification: $y \in \{1, 2, \ldots, k\}$

Example: breast cancer classification

Well-known classification example: using machine learning to diagnose whether a breast tumor is benign or malignant [Street et al., 1992]

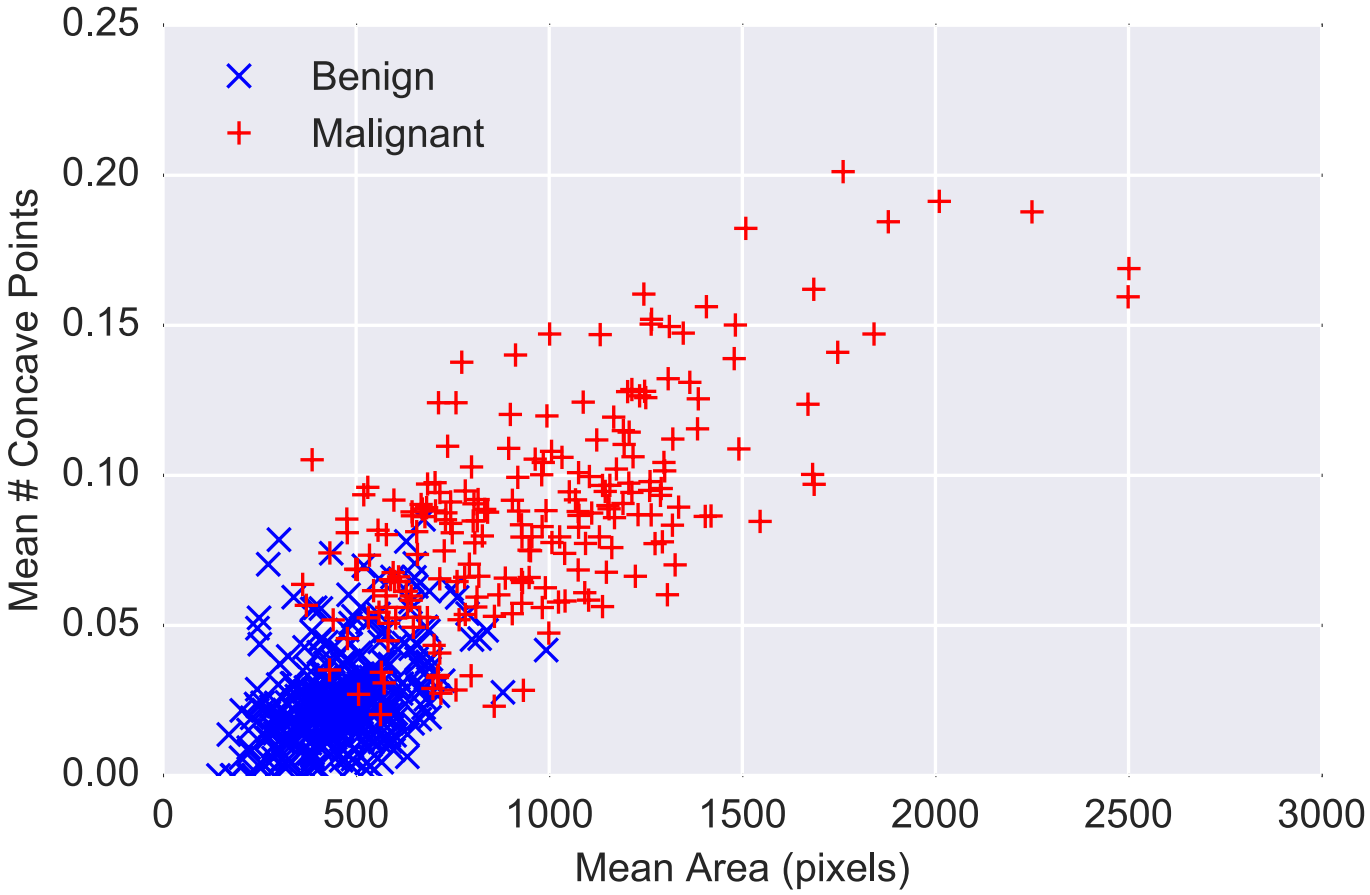
Setting: doctor extracts a sample of fluid from tumor, stains cells, then outlines several of the cells (image processing refines outline)



System computes features for each cell such as area, perimeter, concavity, texture (10 total); computes mean/std/max for all features

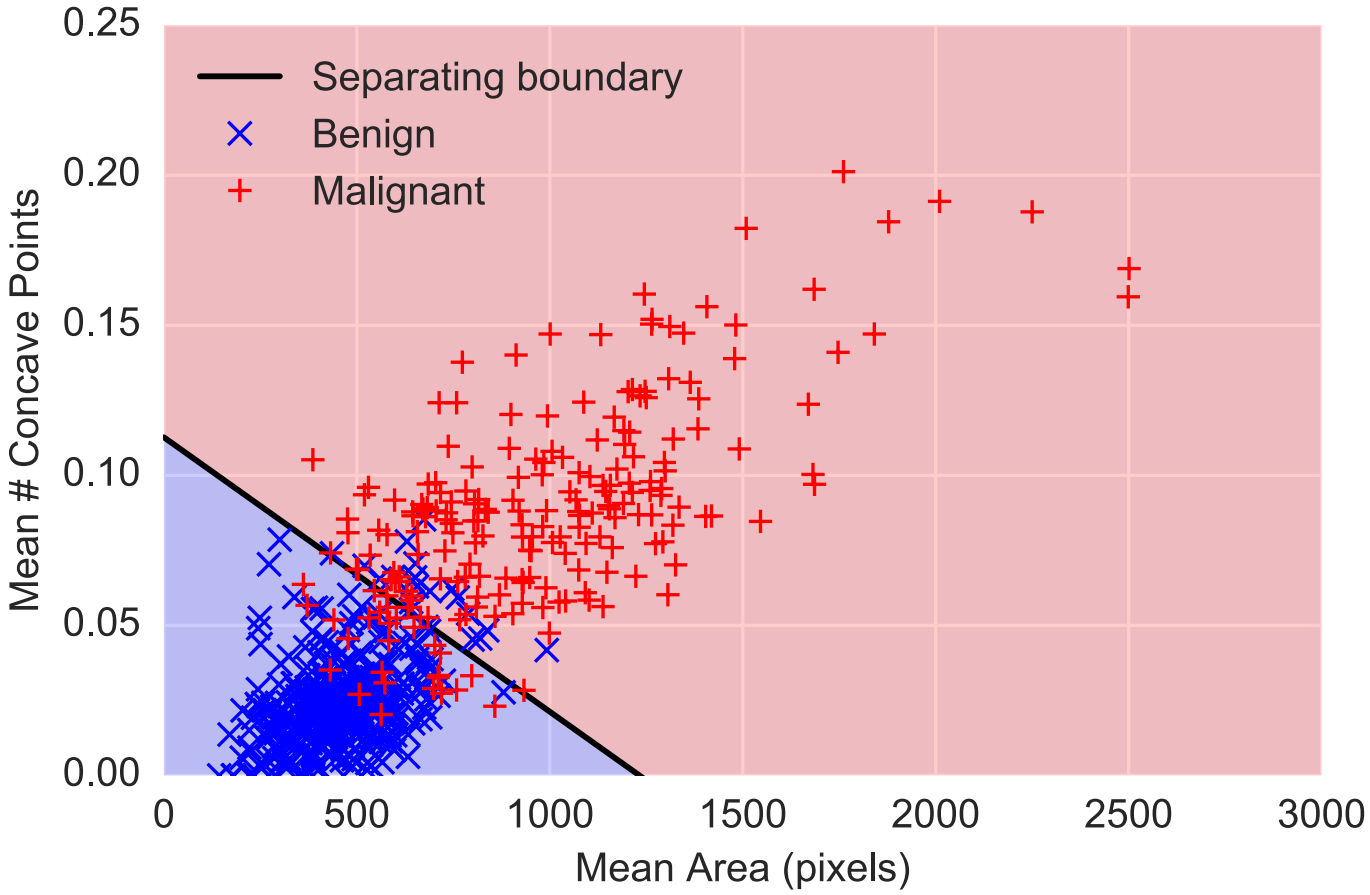
Example: breast cancer classification

Plot of two features: mean area vs. mean concave points, for two classes



Linear classification example

Linear classification ≡ “drawing line separating classes”



Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Formal setting

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

$$\text{E.g.:}\ x^{(i)} = \begin{bmatrix} \text{Mean_Area}^{(i)} \\ \text{Mean_Concave_Points}^{(i)} \\ 1 \end{bmatrix}$$

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

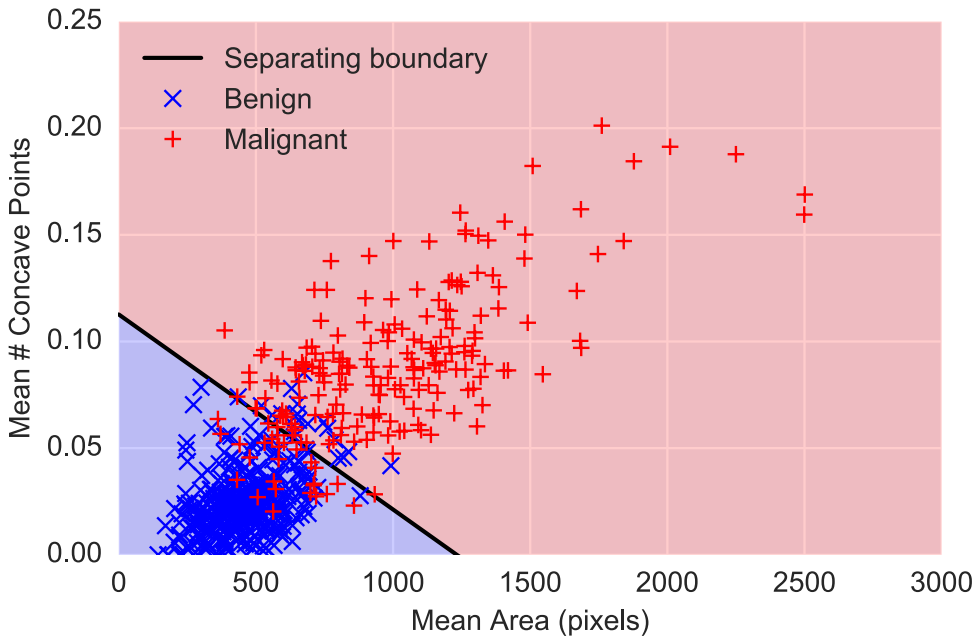
$$\text{E.g.:}\ y^{(i)} \in \{-1\ (\text{benign}), +1\ (\text{malignant})\}$$

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

Hypothesis function: $h_\theta \colon \mathbb{R}^n \to \mathbb{R}$, aims for same *sign* as the output (informally, a measure of *confidence* in our prediction)

$$\text{E.g.:}\ h_\theta(x) = \theta^T x, \qquad \hat{y} = \text{sign}(h_\theta(x))$$

Understanding linear classification diagrams



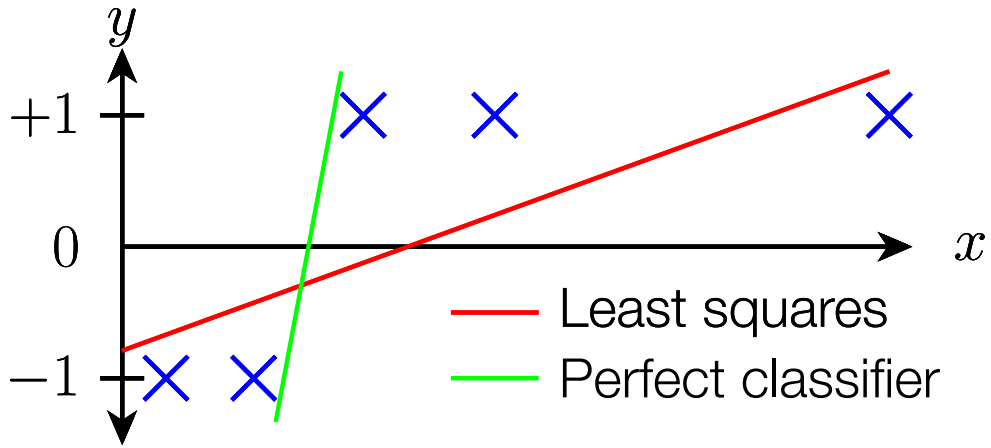
Color shows regions where the $h_\theta(x)$ is positive

Separating boundary is given by the equation $h_\theta(x) = 0$

Loss functions for classification

How do we define a loss function $\ell\colon \mathbb{R}\times\{-1,+1\} \to \mathbb{R}_+$?

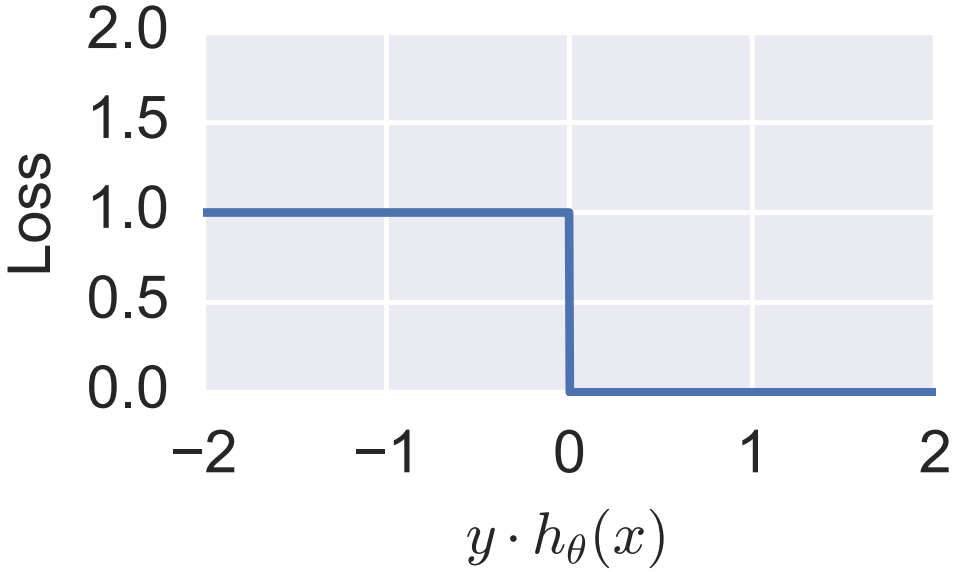
What about just using squared loss?



0/1 loss (i.e. error)

The loss we would like to minimize (0/1 loss, or just "error"):

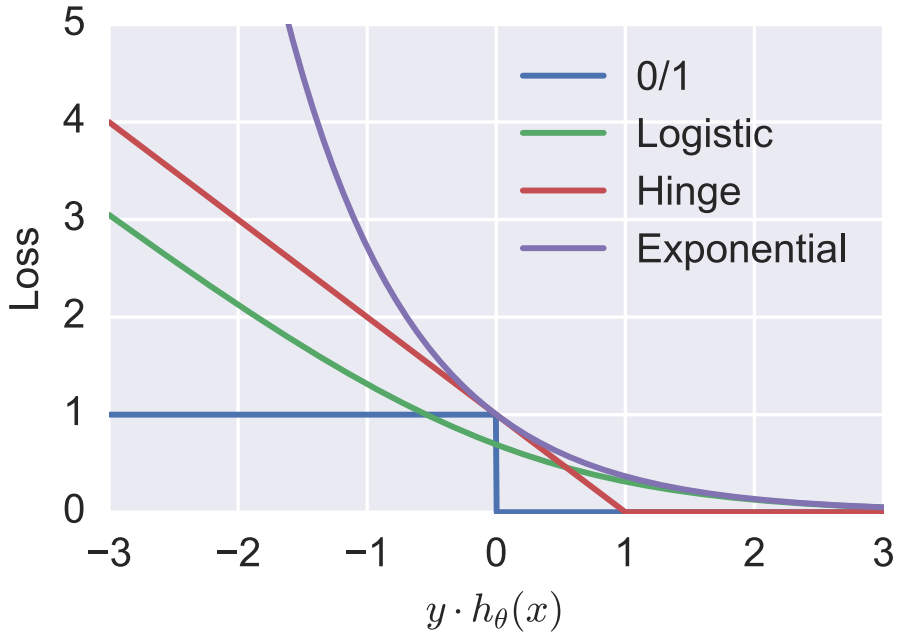
$$\ell_{0/1}(h_\theta(x), y) = \begin{cases} 0 & \text{if } \operatorname{sign}\big(h_\theta(x)\big) = y \\ 1 & \text{otherwise} \end{cases}$$
$$= \mathbf{1}\{y \cdot h_\theta(x) \leq 0\}$$



Alternative losses

Unfortunately 0/1 loss is hard to optimize (NP-hard to find classifier with minimum 0/1 loss, relates to a property called convexity of the function)

A number of alternative losses for classification are typically used instead



$$\ell_{0/1} = 1\{y \cdot h_\theta(x) \leq 0\}$$
$$\ell_{\text{logistic}} = \log(1 + \exp(-y \cdot h_\theta(x)))$$
$$\ell_{\text{hinge}} = \max\{1 - y \cdot h_\theta(x), 0\}$$
$$\ell_{\text{exp}} = \exp(-y \cdot h_\theta(x))$$

Poll: sensitivity to outliers

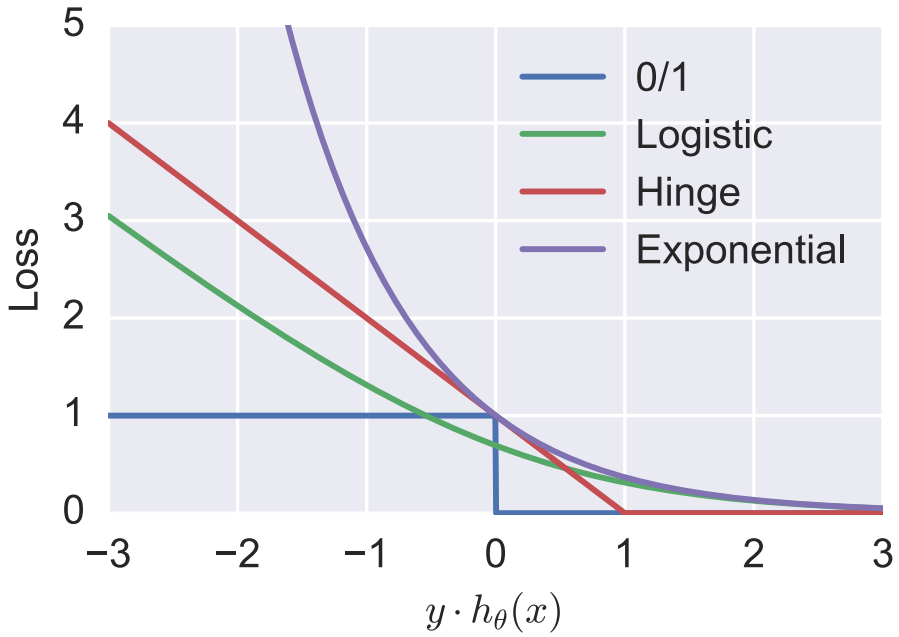
How sensitive would you estimate each of the following losses would be to outliers (i.e., points typically heavily misclassified)?

1. 0/1 < Exp < {Hinge,Logitistic}

2. Exp < Hinge < Logistic < 0/1

3. Hinge < 0/1 < Logistic < Exp

4. 0/1 < {Hinge,Logistic} < Exp



5. Outliers don't exist in the classification because output space is bounded

Machine learning optimization

With this notation, the "canonical" machine learning problem is written in the exact same way

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

Unlike least squares, there is not an analytical solution to the zero gradient condition for most classification losses

Instead, we solve these optimization problems using gradient descent (or a alternative optimization method, but we'll only consider gradient descent here)

$$\text{Repeat: } \theta := \theta - \alpha \sum_{i=1}^{m} \nabla_\theta \ell(\, h_\theta(x^{(i)}), y^{(i)})$$

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Libraries for machine learning

Support vector machine

A (linear) support vector machine (SVM) just solves the canonical machine learning optimization problem using hinge loss and linear hypothesis, plus an additional regularization term, more on this next lecture

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \max\{1 - y^{(i)} \cdot \theta^T x^{(i)}, 0\} + \frac{\lambda}{2} \|\theta\|_2^2$$

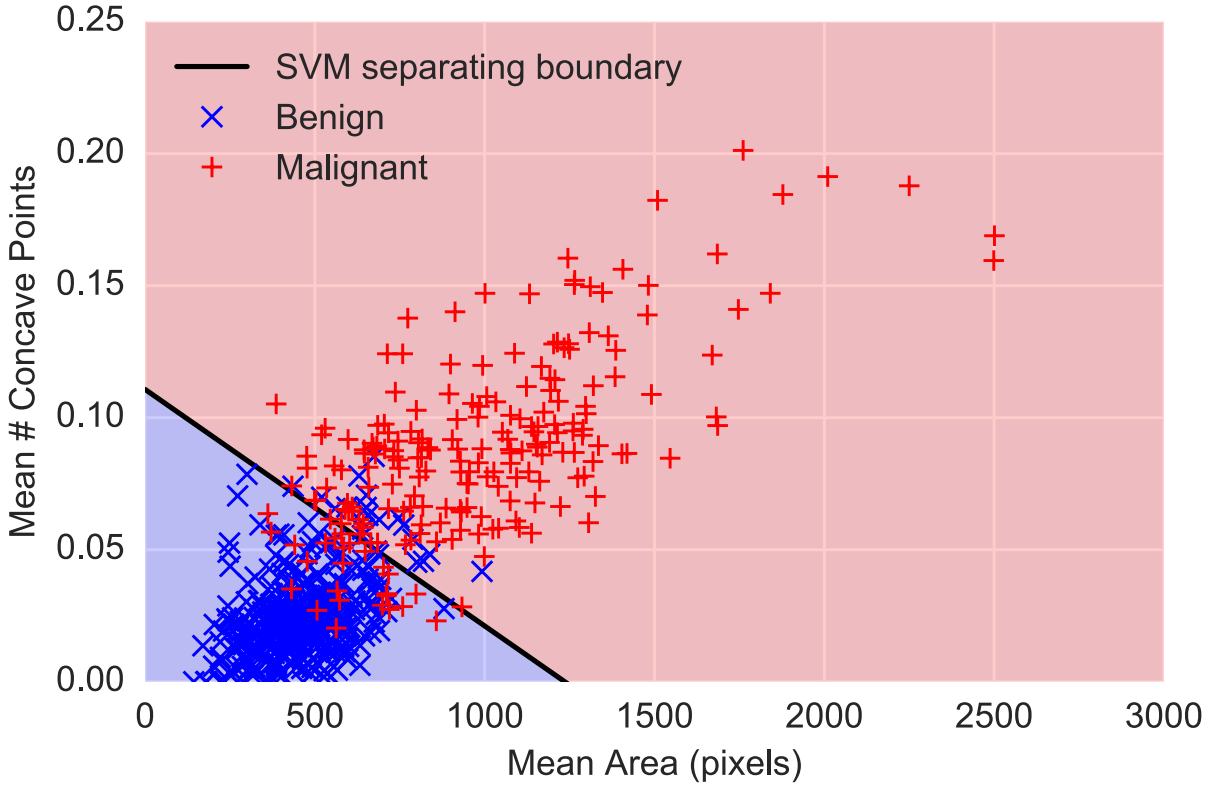
Even more precisely, the "standard" SVM doesn't actually regularize the θ_i corresponding to the constant feature, but we'll ignore this here

Updates using gradient descent:

$$\theta := \theta \ - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} 1\{ y^{(i)} \cdot \theta^T x^{(i)} \leq 1\} - \alpha \lambda \theta$$

Support vector machine example

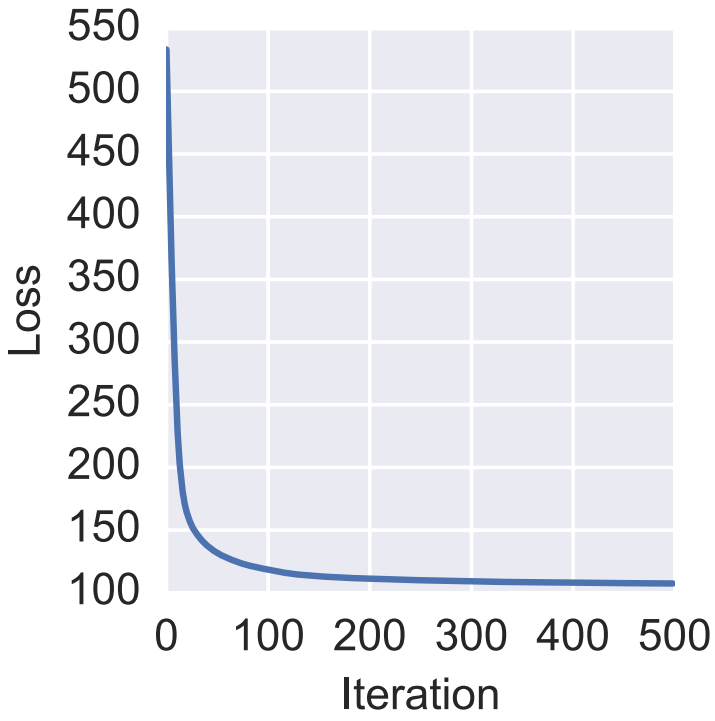
Running support vector machine on cancer dataset, with small regularization parameter (effectively zero)

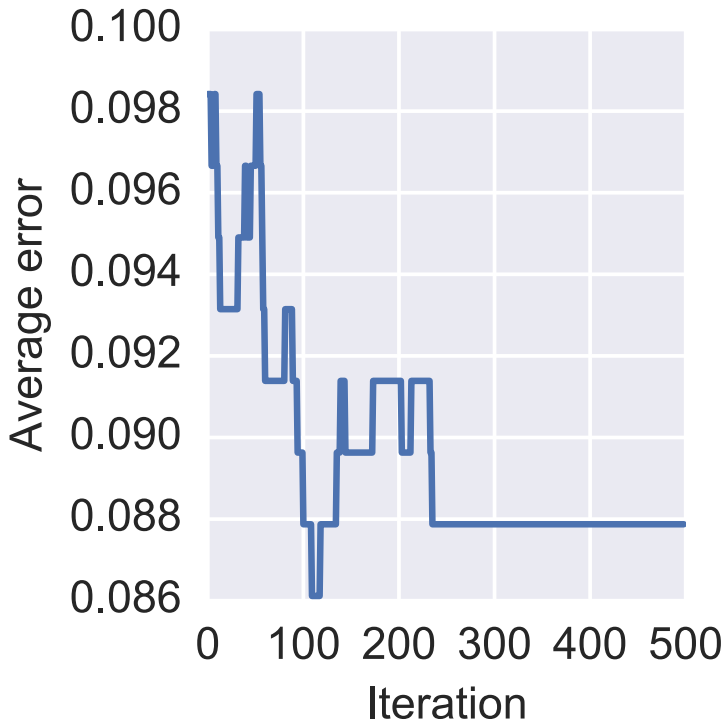


$$\theta = \begin{bmatrix} 1.456 \\ 1.848 \\ -0.189 \end{bmatrix}$$

SVM optimization progress

Optimization objective and error versus gradient descent iteration number





Logistic regression

Logistic regression just solves this problem using logistic loss and linear hypothesis function

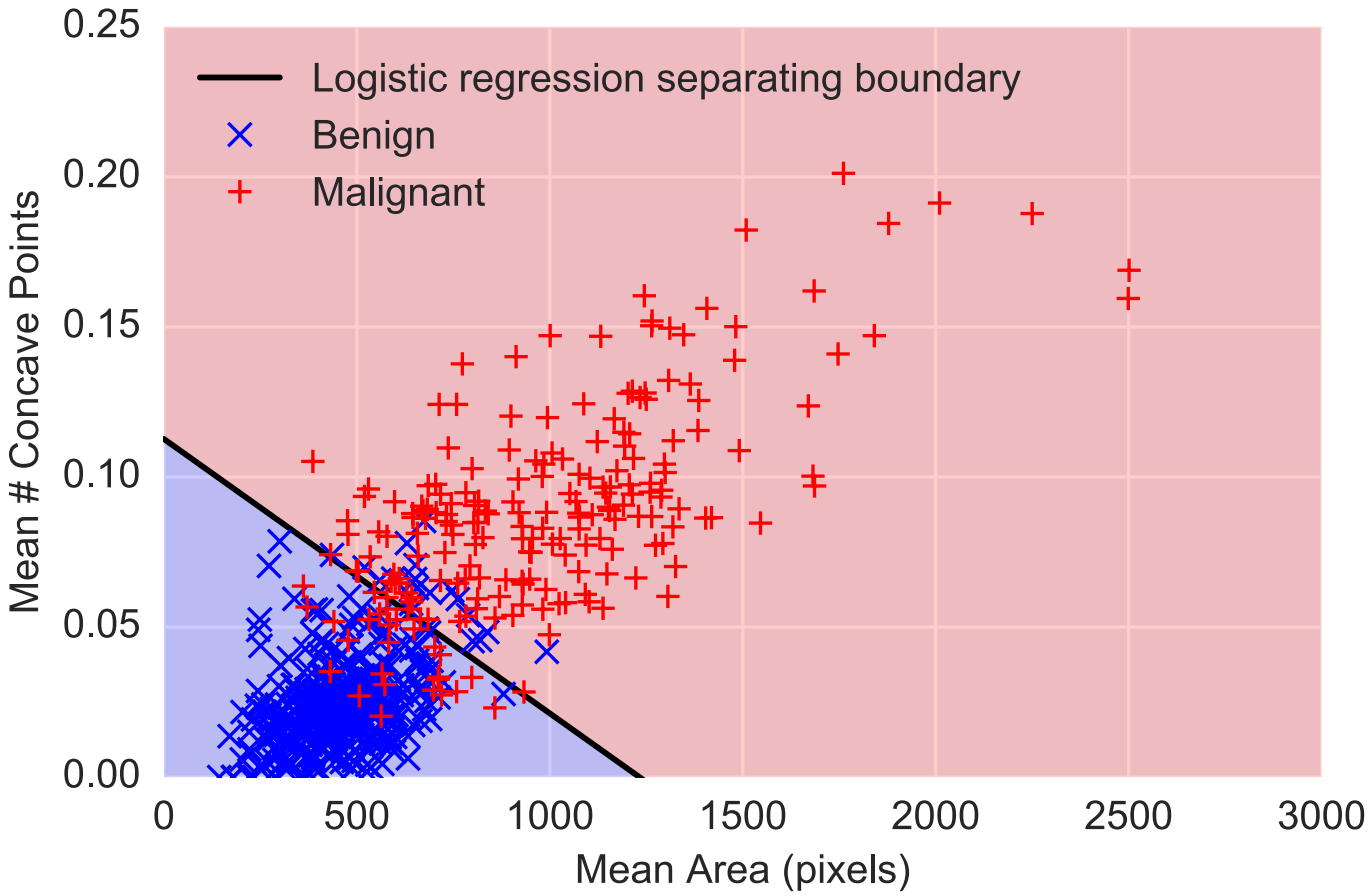
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \log(1 + \exp(-y^{(i)} \cdot \theta^T x^{(i)}))$$

Gradient descent updates (can you derive these?):

$$\theta := \theta - \alpha \sum_{i=1}^{m} -y^{(i)} x^{(i)} \frac{1}{1 + \exp(y^{(i)} \cdot \theta^T x^{(i)})}$$

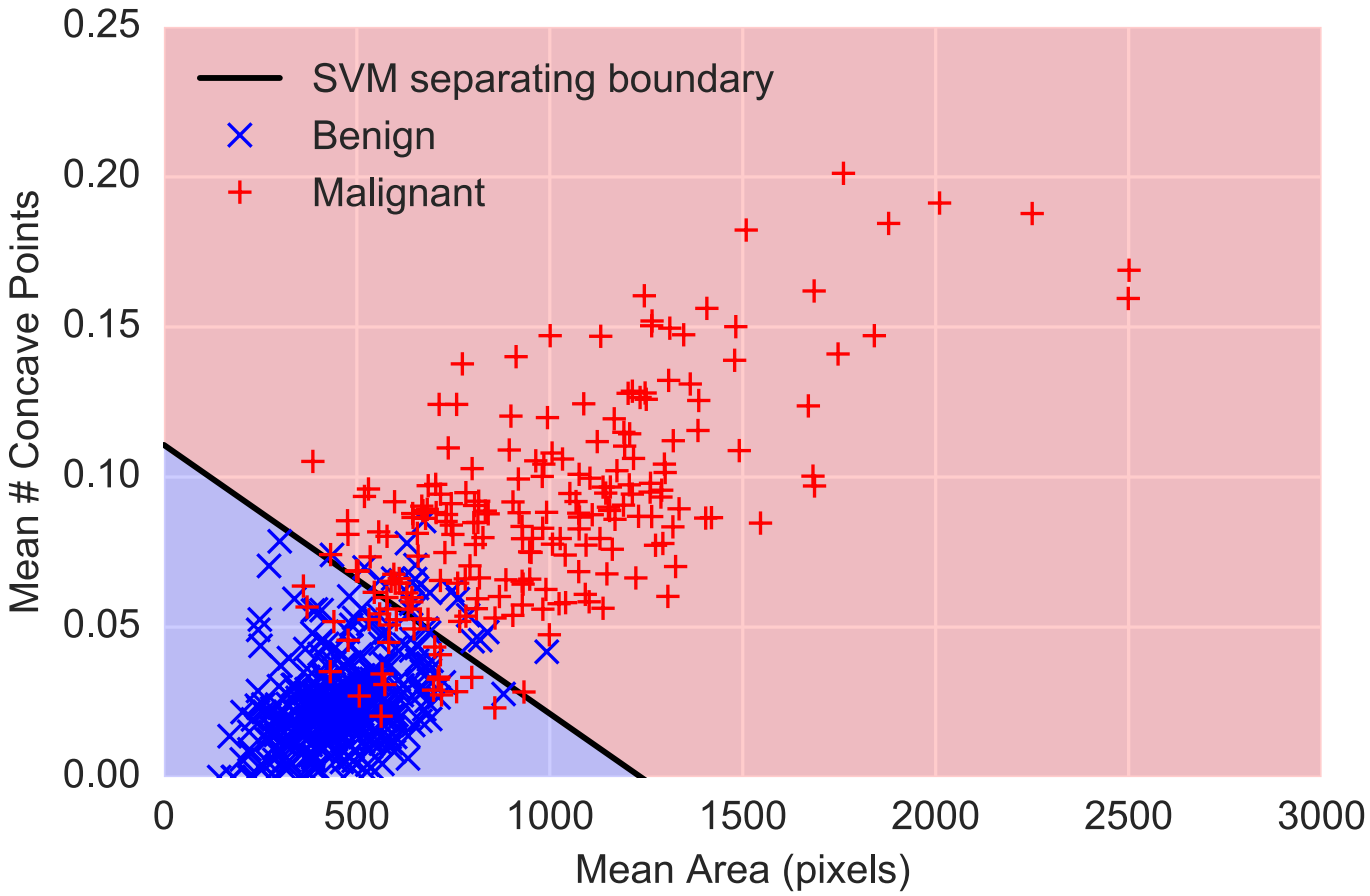
Logistic regression example

Running logistic regression on cancer data set, small regularization



Logistic regression example

Running logistic regression on cancer data set, small regularization



Multiclass classification

When output is in $\{1, \dots, k\}$ (e.g., digit classification), a few different approaches

Approach 1: Build k different binary classifiers h_{θ_i} with the goal of predicting class i vs all others, output predictions as

$$\hat{y} = \operatorname*{argmax}_i h_{\theta_i}(x)$$

Approach 2: Use a hypothesis function $h_\theta \colon \mathbb{R}^n \to \mathbb{R}^k$, define an alternative loss function $\ell \colon \mathbb{R}^k \times \{1, \dots, k\} \to \mathbb{R}_+$

E.g., softmax loss (also called cross entropy loss):

$$\ell(h_\theta(x), y) = \log \sum_{j=1}^{k} \exp(h_\theta(x)_j) - h_\theta(x)_y$$

Outline

Example: classifying tumors

Classification in machine learning

Example classification algorithms

Classification with Python libraries

Support vector machine in scikit-learn

Train a support vector machine:

```
from sklearn.svm import LinearSVC, SVC

clf = SVC(C=1e4, kernel='linear') # or
clf = LinearSVC(C=1e4, loss='hinge', max_iter=1e5)
clf.fit(X, y) # don't include constant features in X
```

Make predictions:

```
y_pred = clf.predict(X)
```

Note: Scikit-learn in solving the problem (inverted regularization term):

$$\text{minimize}_\theta \quad C \sum_{i=1}^{m} \max\{1 - y^{(i)} \cdot \theta^T x^{(i)}, 0\} + \frac{1}{2}\|\theta\|_2^2$$

Native Python SVM

It's pretty easy to write a gradient-descent-based SVM too

```
def svm_gd(X, y, lam=1e-5, alpha=1e-4, max_iter=5000):
    m,n = X.shape
    theta = np.zeros(n)
    Xy = X*y[:,None]
    for i in range(max_iter):
        theta -= alpha*(-Xy.T.dot(Xy.dot(theta) <= 1) + lam*theta)
    return theta
```

For the most part, ML algorithms are very simple, you can easily write them yourself, but it's fine to use libraries to quickly try many algorithms

But watch out for idiosyncratic differences (e.g., C vs λ, the fact that I'm using $y \in \{-1, +1\}$, not $y \in \{0,1\}$, etc)

Logistic regression in scikit-learn

Admittedly very nice element of scikit-learn is that we can easily try out other algorithms

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(C=10000.0)
clf.fit(X, y)
```

For both this example and SVM, you can access resulting parameters using the fields

```
clf.coef_ # parameters other than weight on constant feature
clf.intercept_ # weight on constant feature
```

15-388/688 - Practical Data Science: Hypothesis testing and experimental design

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Outline

Motivation

Background: sample statistics and central limit theorem

Basic hypothesis testing

Experimental design

Outline

Motivation

Background: sample statistics and central limit theorem

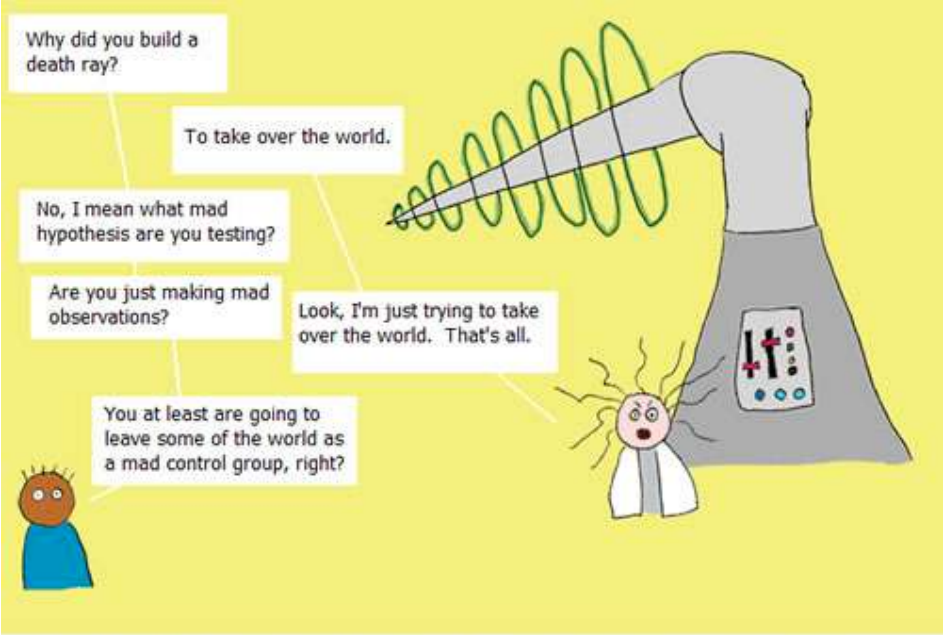
Basic hypothesis testing

Experimental design

Motivating setting

For a data science course, there has been very little “science” thus far…

“Science” as I’m using it roughly refers to “determining truth about the real world”



Sad truth: Most "mad scientists" are actually just mad engineers

Asking scientific questions

Suppose you work for a company that is considering a redesign of their website; does their new design (design B) offer any statistical advantage to their current design (design A)?

In linear regression, does a certain variable impact the response? (E.g. does energy consumption depend on whether or not a day is a weekday or weekend?)

In both settings, we are concerned with making actual statements about the nature of the world

Outline

Motivation

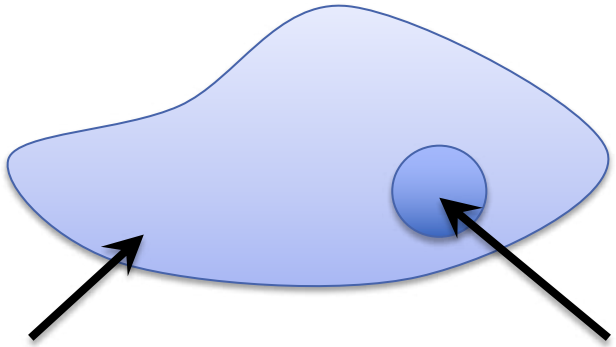
Background: sample statistics and central limit theorem

Basic hypothesis testing

Experimental design

Sample statistics

To be a bit more consistent with standard statistics notation, we'll introduce the notion of a *population* and a *sample*



	Population	**Sample**
Mean	$\mu = \mathbf{E}[X]$	$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
Variance	$\sigma^2 = \mathbf{E}[(X-\mu)^2]$	$s^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x^{(i)} - \bar{x})^2$

Sample mean as random variable

The same mean is an empirical average over m independent samples from the distribution; it can also be considered as a random variable

This new random variable has the mean and variance

$$\mathbf{E}[\bar{x}] = \mathbf{E}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] = \frac{1}{m}\sum_{i=1}^{m}\mathbf{E}[X] = \mathbf{E}[X] = \mu$$

$$\mathbf{Var}[\bar{x}] = \mathbf{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right] = \frac{1}{m^2}\sum_{i=1}^{m}\mathbf{Var}[X] = \frac{\sigma^2}{m}$$

where we used the fact that for *independent* random variables X_1, X_2

$$\mathbf{Var}[X_1 + X_2] = \mathbf{Var}[X_1] + \mathbf{Var}[X_2]$$

When estimating variance of sample, we use s^2/m (the square root of this term is called the **standard error**)

Central limit theorem

Central limit theorem states further that $\bar{x}$ (for "reasonably sized" samples, in practice $m \geq 30$) actually has a Gaussian distribution *regardless of the distribution of* X

$$\bar{x} \to \mathcal{N}\left(\mu, \frac{\sigma^2}{m}\right) \text{ (or equivalently) } \frac{\bar{x} - \mu}{\sigma / m^{1/2}} \to \mathcal{N}(0,1)$$

In practice, for $m < 30$ and for estimating σ^2 using sample variance, we use a Student's t-distribution with $m - 1$ degrees of freedom

$$\frac{\bar{x} - \mu}{s / m^{1/2}} \to T_{m-1}, \qquad p(x; \nu) \propto \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

Aside: why the $m-1$ scaling?

We scale the sample variance by $m-1$ so that it is an *unbiased* estimate of the population variance

$$
\begin{aligned}
&\mathbf{E}\left[\sum_{i=1}^{m}(x^{(i)}-\bar{x})^2\right] = \mathbf{E}\left[\sum_{i=1}^{m}\left((x^{(i)}-\mu)-(\bar{x}-\mu)\right)^2\right] \\
&= \mathbf{E}\left[\sum_{i=1}^{m}(x^{(i)}-\mu)^2 - 2(\bar{x}-\mu)\sum_{i=1}^{m}(x^{(i)}-\mu) + m(\bar{x}-\mu)^2\right] \\
&= \mathbf{E}\left[\sum_{i=1}^{m}(x^{(i)}-\mu)^2\right] - m\mathbf{E}\left[\sum_{i=1}^{m}(\bar{x}-\mu)^2\right] \\
&= m\mathbf{Var}[X] - \frac{m\mathbf{Var}[X]}{m} = (m-1)\sigma^2
\end{aligned}
$$

Outline

Motivation

Background: sample statistics and central limit theorem

Basic hypothesis testing

Experimental design

Hypothesis testing

Using these basic statistical techniques, we can devise some tests to determine whether certain data gives evidence that some effect “really” occurs in the real world

Fundamentally, this is evaluating whether things are (likely to be) true about the population (all the data) given a sample

Lots of caveats about the precise meaning of these terms, to the point that many people debate the usefulness of hypothesis testing at all

But, still incredibly common in practice, and important to understand

Hypothesis testing basics

Posit a *null hypothesis* H_0 and an alternative hypothesis H_1 (usually just that "H_0 is not true"

Given some data x, we want to accept or reject the null hypothesis in favor of the alternative hypothesis

	H_0 **true**	H_1 **true**
Accept H_0	Correct	Type II error (false negative)
Reject H_0	Type I error (false positive)	Correct

$p(\text{reject } H_0 | H_0 \text{ true})$ = "significance of test"

$p(\text{reject } H_0 | H_1 \text{ true})$ = "power of test"

Basic approach to hypothesis testing

Basic approach: compute the probability of observing the data *under the null hypothesis* (this is the p-value of the statistical test)

$$p = p(\text{data}|H_0 \text{ is true})$$

Reject the null hypothesis if the p-value is below the desired significance level (alternatively, just report the p-value itself, which is the lowest significance level we could use to reject hypothesis)

Important: p-value is $p(\text{data}|H_0 \text{ is true})$ not $p(H_0 \text{ is true } |\text{data})$

Poll: p-value hacking

Suppose you adopt the following procedure. You test 100 patients to see if a drug has a statistically significant effect. If so, you stop the test and publish your current p-value. If not, you collect 100 additional patients, test the drug again, and publish that p-value (statistically significant or not). Is this a valid experimental design?

1. Yes

2. No

3. Depends on what p value you achieve

Canonical example: t-test

Given a sample $x^{(1)}, \ldots, x^{(m)} \in \mathbb{R}$

$$H_0 \colon \mu = 0 \text{ (for population)}$$
$$H_1 \colon \mu \neq 0$$

By central limit theorem, we know that $(\bar{x} - \mu)/(s/m^{\frac{1}{2}}) \sim T_{m-1}$ (Student's t-distribution with $m - 1$ degrees of freedom)

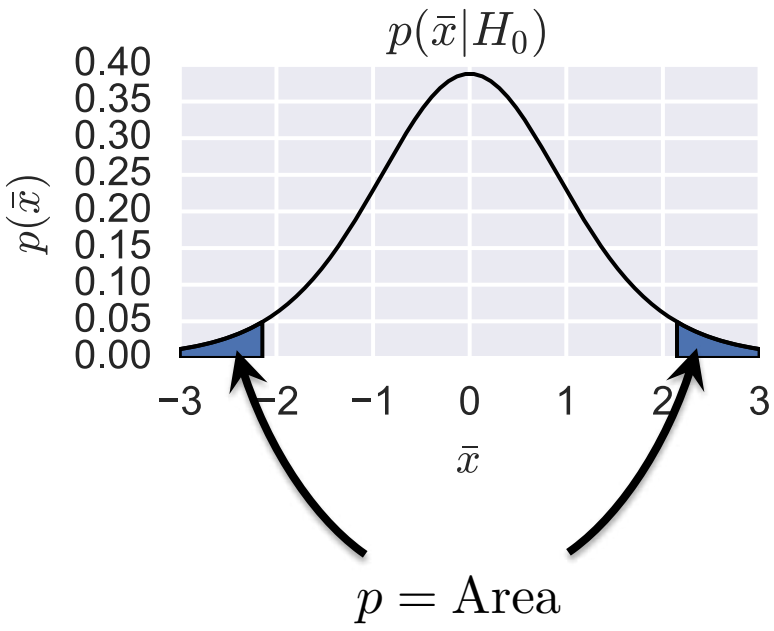
So we just compute $t = \bar{x}/\left(s/m^{\frac{1}{2}}\right)$ (called *test statistic*), then compute

$$p = p(x > |t|) + p(x < -|t|) = F(-|t|) + 1 - F(|t|) = 2F(-|t|)$$

(where F is cumulative distribution function of Student's t-distribution)

Visual example

What we are doing fundamentally is modeling the distribution $p(\bar{x}|H_0)$ and then determining the probability of the observed $\bar{x}$ or a more extreme value



Code in Python

Compute t statistic and p value from data

```
import numpy as np
import scipy.stats as st
x = np.random.randn(m)

# compute t statistic and p value
xbar = np.mean(x)
s2 = np.sum((x - xbar)**2)/(m-1)
std_err = np.sqrt(s2/m)
t = xbar/std_err

t_dist = st.t(m-1)
p = 2*td.cdf(-np.abs(t))

# with scipy alone
t,p = st.ttest_1samp(x, 0)
```

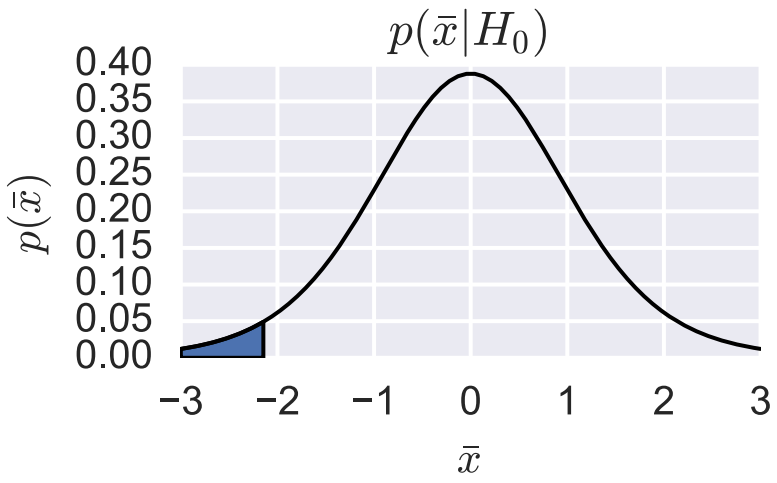
Two-sided vs. one-sided tests

The previous test considered deviation from the null hypothesis in both directions (two-sided test), also possible to consider a one-sided test

$$H_0 \colon \mu \geq 0 \text{ (for population)}$$
$$H_1 \colon \mu < 0$$

Same t statistic as before, but we only compute the area under the left side of the curve

$$p = p(x < t) = F(t)$$



Confidence intervals

We can also use the t statistic to create *confidence intervals* for the mean

Because $\bar{x}$ has mean μ and variance s^2/m, we know that $1-\alpha$ of its probability mass must lie within the range

$$\bar{x} = \mu \pm \frac{s}{m^{1/2}} \cdot F^{-1}\left(1 - \frac{\alpha}{2}\right) \equiv \mu + CI(s, m, \alpha)$$
$$\iff \mu = \bar{x} \pm CI(s, m, \alpha)$$

where F^{-1} denotes the inverse CDF function of t-distribution with $m-1$ degrees of freedom

```
# simple confidence interval compuation
CI = lambda s,m,a : s / np.sqrt(m) * st.t(m-1).ppf(1-a/2)
```

Outline

Motivation

Background: sample statistics and central limit theorem

Basic hypothesis testing

Experimental design

Experimental design: A/B testing

Up until now, we have assumed that the null hypothesis is given by some *known* mean, but in reality, we may not know the mean that we want to compare to

Example: we want to tell if some additional feature on our website makes user stay longer, so we need to estimate both how long users stay on the current site and how long they stay on redesigned site

Standard approach is A/B testing: create a *control group* (mean μ_1) and a *treatment group* (mean μ_2)

$$H_0: \mu_1 = \mu_2 \ (\text{or e.g. } \mu_1 \geq \mu_2)$$
$$H_1: \mu_1 \neq \mu_2 \ (\text{or e.g. } \mu_1 < \mu_2)$$

Independent t-test (Welch's t-test)

Collect samples (possibly different numbers) from *both* populations

$$x_1^{(1)}, \dots, x_1^{(m_1)}, \qquad x_2^{(1)}, \dots, x_2^{(m_2)}$$

compute sample mean $\bar{x}_1, \bar{x}_2$ and sample variance s_1^2, s_2^2 for each group

Compute test statistic

$$t = \frac{\bar{x}_1 - \bar{x}_2}{(s_1^2/m_1 + s_2^2/m_2)^{1/2}}$$

And evaluate using a t distribution with degrees of freedom given by

$$\frac{(s_1^2/m_1 + s_2^2/m_2)^2}{\frac{(s_1^2/m_1)^2}{m_1 - 1} + \frac{(s_2^2/m_2)^2}{m_2 - 1}}$$

Starting seem a bit ad-hoc?

There are a huge number of different tests for different situations

You probably won't need to remember these, and can just look up whatever test is most appropriate for your given situation

But the basic idea in call cases is the same: you're trying to find the distribution of your test statistic under the hull hypothesis, and then you are computing the probability of the observed test statistic or something more extreme

All the different tests are really just about different distributions based upon your problem setup

Hypothesis testing in linear regression

One last example (because it's useful in practice): consider the linear regression $y \approx \theta^T x$, and suppose we want to perform a hypothesis test on the coefficients of θ

Example: suppose that instead of just two website, you have a website with multiple features that can be turned on/off, and your sample data includes a wide variety of different samples

We would like to ask the question: is the ith variable relevant for predicting the output?

We've already seen ways we can do this (i.e., evaluate cross-validation error, but it's a bit difficult to understand what this means)

Formula for sample variance in linear regression

There is an analogous formula for sample variance on the *errors* that a linear regression model makes

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

Use this to determine sample covariance of *coefficients*

$$\mathbf{Cov}[\theta] = s^2(X^T X)^{-1}$$

Can then evaluate null hypothesis $H_0 \colon \theta_i = 0$, using t statistic

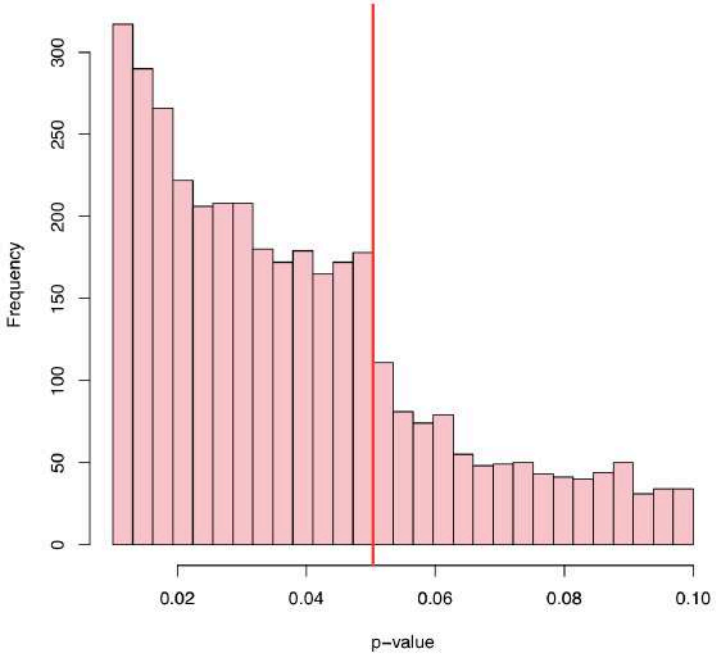
$$t = \theta_i / \mathbf{Cov}[\theta]_{i,i}^{1/2}$$

Similar procedure to get confidence intervals of coefficients

P-values considered harmful

A basic problem is that $p(\text{data}|H_0) \neq p(H_0|\text{data})$ (despite being frequently interpreted as such)

People treat $p < 0.05$ with *way* too much importance



Histogram of p values from ~3,500 published journal papers (from E. J. Masicampo and Daniel Lalande, *A peculiar prevalence of p values just below .05*, 2012)

15-388/688 - Practical Data Science: Decision trees and interpretable models

J. Zico Kolter
Carnegie Mellon University
Spring 2018

Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

Boosting

Examples

Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

Boosting

Examples

Overview

Decision trees and boosted decision trees are some of the most ubiquitous algorithms in data science

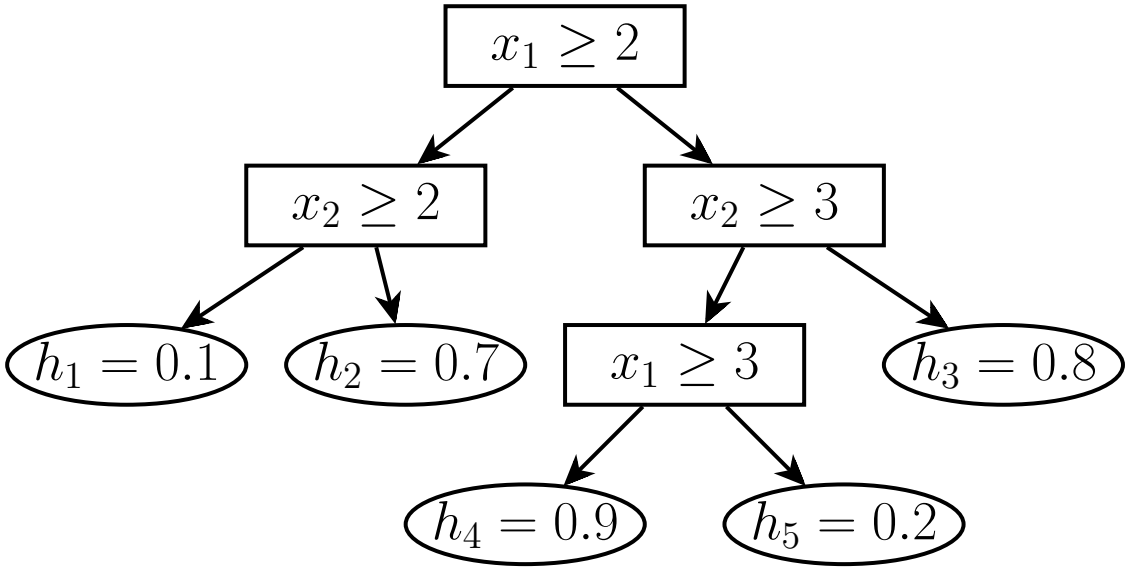
Boosted decision trees typically perform very well without much tuning (the majority of Kaggle contests, for instance, are won with boosting methods)

Decision trees, while not as powerful from a pure ML standpoint, are still one of the canonical examples of an “understandable” ML algorithm

Decision trees

Decision trees were one of the first machine learning algorithms

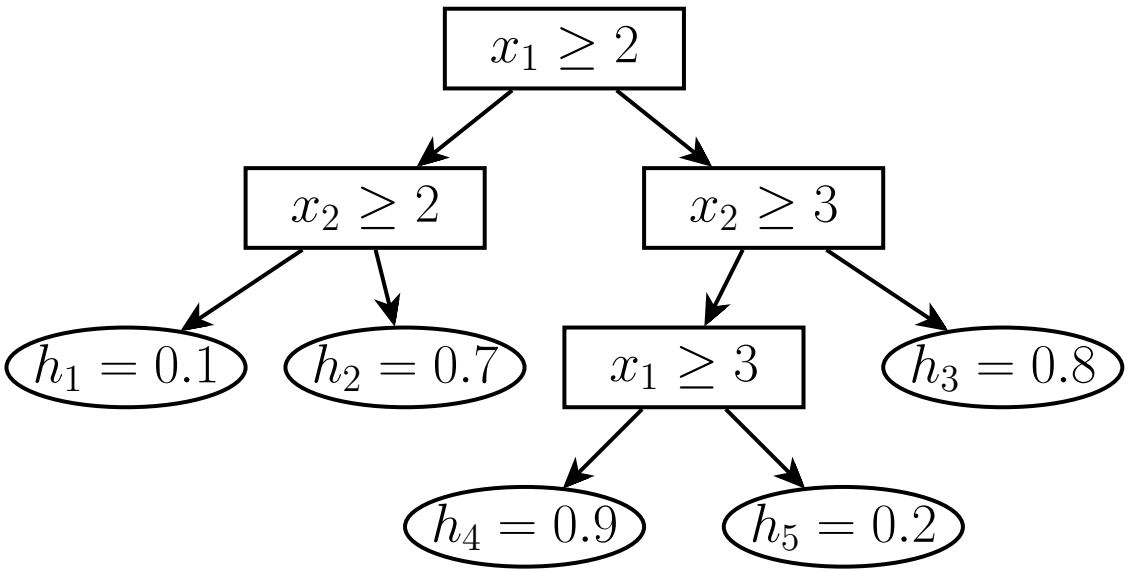
Basic idea: make classification/regression predictions by tracing through rules in a tree, with a constant prediction at each leaf node

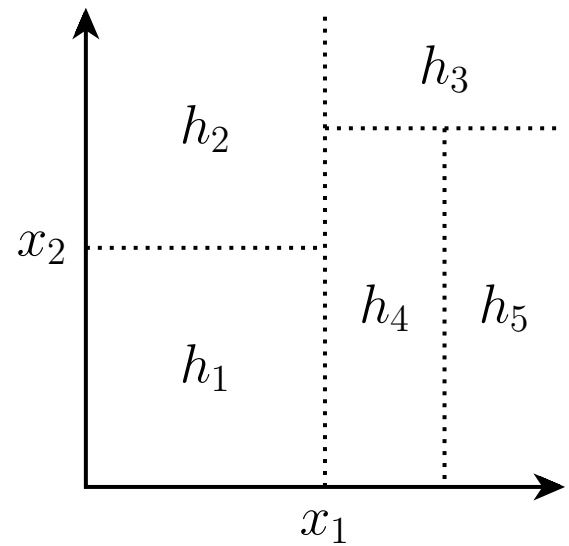


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

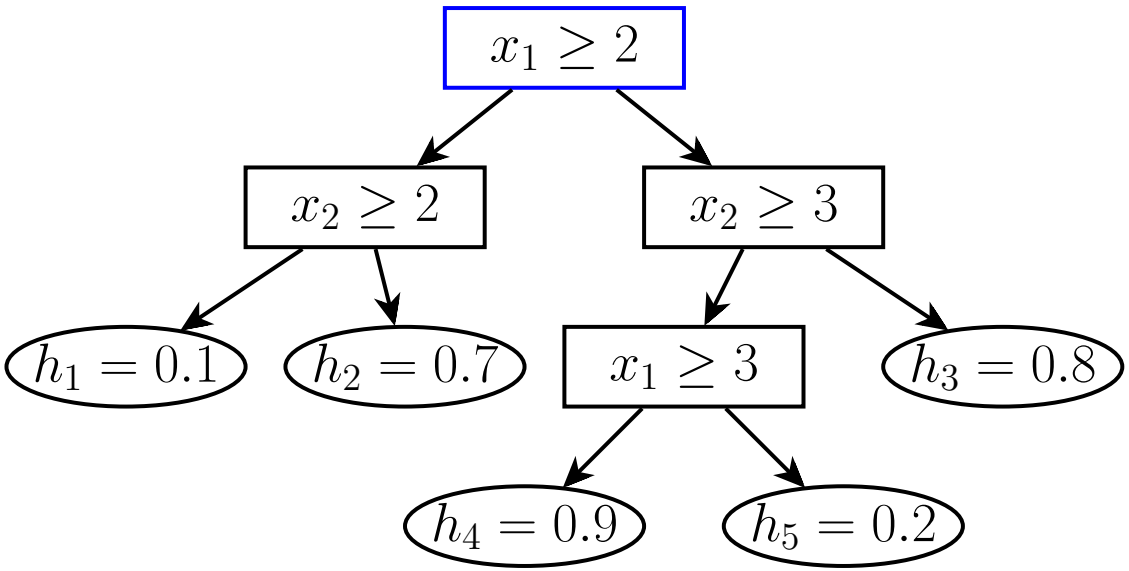


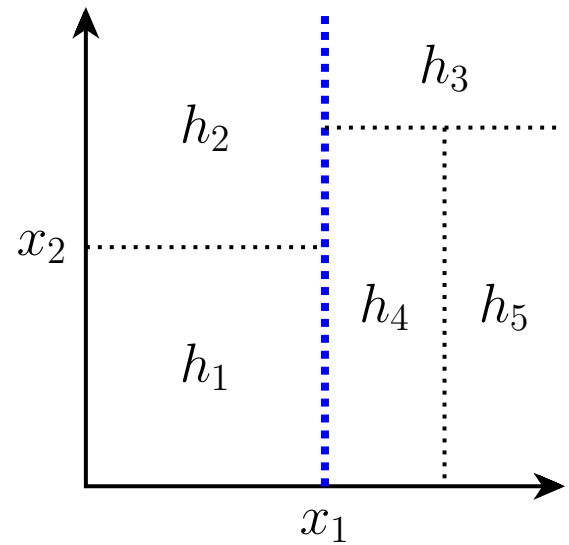


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

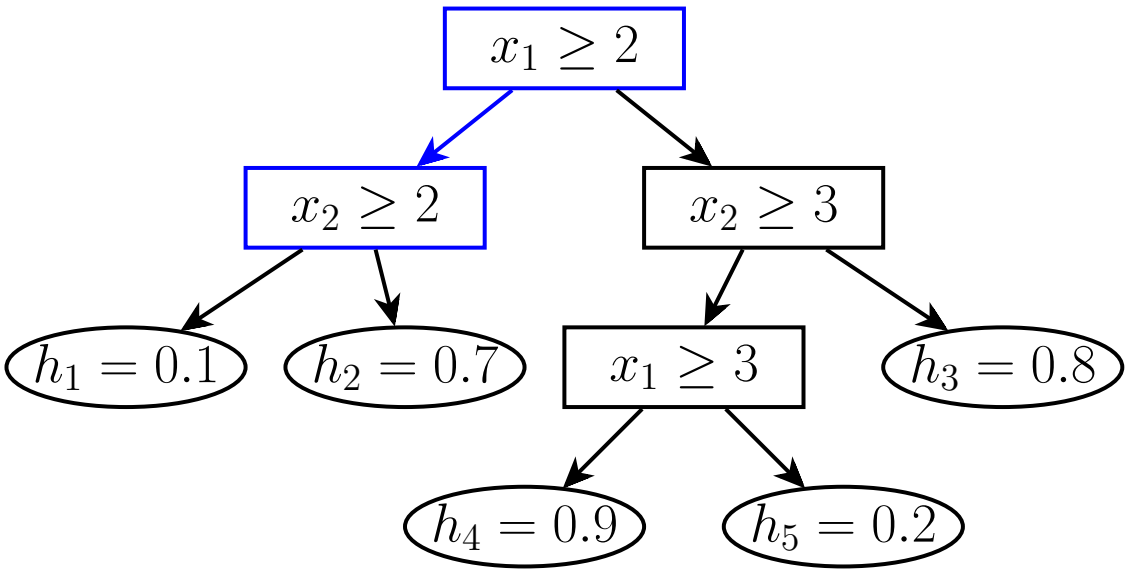


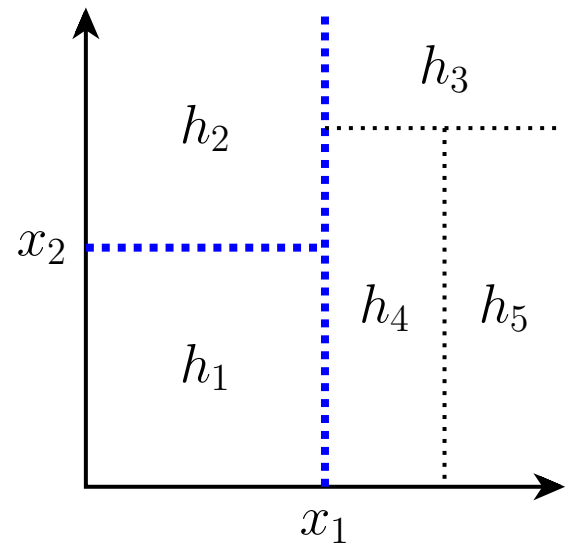


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value

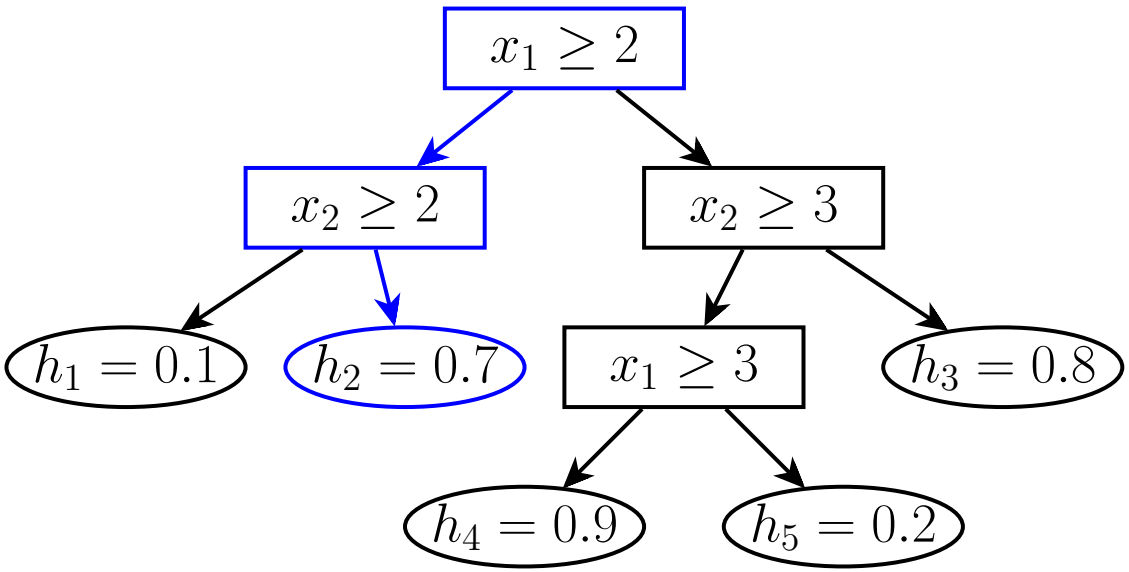


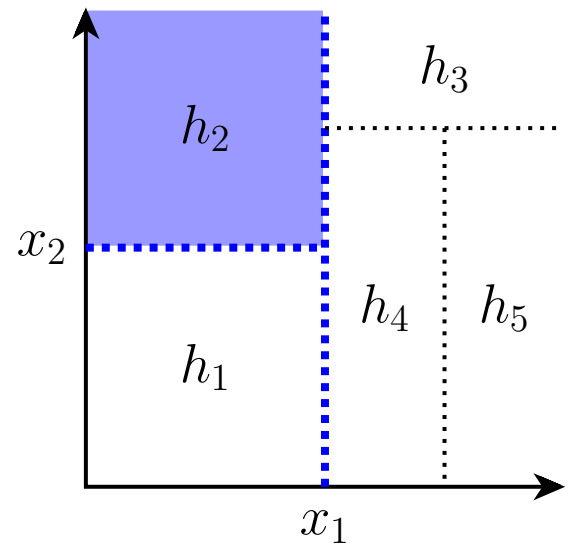


Partitioning the input space

You can think of the hypothesis function of decision trees as partitioning the input space with axis-aligned boundaries

In each partition, predict a constant value





Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

Boosting

Examples

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. What is the loss function $\ell(h_\theta(x), y)$?

3. How do we minimize the loss function?

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

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. **What is the hypothesis function $\boldsymbol{h_\theta(x)}$?**
2. What is the loss function $\ell(h_\theta(x), y)$?
3. How do we minimize the loss function?

…a decision tree (θ is shorthand for all the parameters that define the tree: tree structure, values to split on, leaf predictions, etc)

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

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. **What is the loss function $\ell(\boldsymbol{h_\theta}(\boldsymbol{x}), \boldsymbol{y})$?**

3. How do we minimize the loss function?

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

Loss functions in decision trees

Let's assume the output is binary for now (classification task, we will deal with regression shortly), and assume $y \in \{0,1\}$

The typical decision tree algorithm using a probabilistic loss function that considers y to be a Bernoulli random variable with probability $h_\theta(x)$

$$p(y|h_\theta(x)) = h_\theta(x)^y \big(1 - h_\theta(x)\big)^{1-y}$$

The loss function is just the negative log probability of the output (like in maximum likelihood estimation)

$$\begin{aligned}\ell(h_\theta(x), y) &= -\log p(y|h_\theta(x)) \\ &= -y \log h_\theta(x) - (1-y) \log\big(1 - h_\theta(x)\big)\end{aligned}$$

Decision trees as ML algorithms

To specify the decision trees from a machine learning standpoint, we need to specify

1. What is the hypothesis function $h_\theta(x)$?

2. What is the loss function $\ell(h_\theta(x), y)$?

3. **How do we minimize the loss function?**

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

Optimizing decision trees

Key challenge: unlike models we have considered previously, discrete tree structure means there are no gradients

Additionally, even if we assume binary inputs i.e., $x \in \{0,1\}^n$, there are 2^{2^n} possible decision trees: $n = 7$ means 3.4×10^{38} possible trees

Instead, we're going to use *greedy* methods to incrementally build the tree (i.e., minimize the loss function) one node at a time

Optimizing a single leaf

Consider a single leaf in a decision tree (could be root of initial tree)

Let $\mathcal{X}$ denote the examples at this leaf (e.g., in this partition), where $\mathcal{X}^+$ denotes the positive examples and $\mathcal{X}^-$ denotes negative (zero) examples

What should we choose as the (constant) prediction h at this leaf?

$$\underset{h}{\text{minimize}} \sum_{x,y \in \mathcal{X}} \ell(h, y) \;\; = -\frac{|\mathcal{X}^+|}{|\mathcal{X}|} \log h - \frac{|\mathcal{X}^-|}{|\mathcal{X}|} \log(1-h)$$

$$\implies h = \frac{|\mathcal{X}^+|}{|\mathcal{X}|},$$

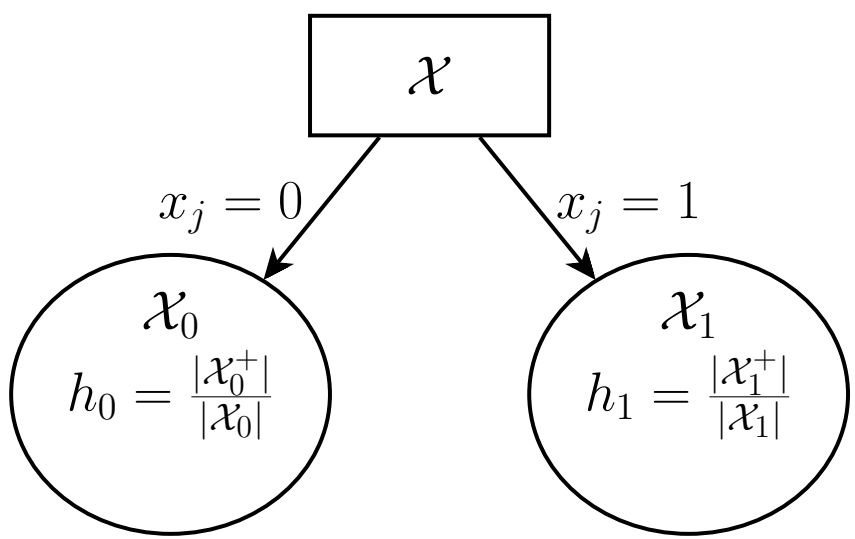
Which achieves loss:

$$\ell = -h \log h - (1-h) \log(1-h)$$

Optimizing splits

Now suppose we want to split this leaf into two leaves, assuming for the time being that $x \in \{0,1\}^n$ is binary

If we split on a given feature j, this will separate $\mathcal{X}$ into two sets: $\mathcal{X}_0$ and $\mathcal{X}_1$ (with $\mathcal{X}_{0/1}^{+/-}$ and defined similarly to before), and we would choose optimal prediction $h_0 = |\mathcal{X}_0^+|/|\mathcal{X}_0|, h_1 = |\mathcal{X}_1^+|/|\mathcal{X}_1|$



Loss of split

The new leafs will each now suffer loss

$$\ell_0 = -h_0 \log h_0 - (1 - h_0) \log(1 - h_0)$$
$$\ell_1 = -h_1 \log h_1 - (1 - h_1) \log(1 - h_1)$$

Thus, if we split the original leaf on feature j, we no longer suffer our original loss ℓ, but we do suffer losses $\ell_1 + \ell_2$, i.e., we have decreased the overall loss function by $\ell - \ell_0 - \ell_1$ (this quantity is called ***information gain***)

Greedy decision tree learning – repeat:

- For all leaf nodes, evaluate information gain (i.e., decrease in loss) when splitting on each feature j
- Split the node/feature that minimizes loss the most
- (Run cross-validation to determine when to stop, or after N nodes)

Poll: Decision tree training

Which of the following are true about training decision trees?

1. Once a feature has been selected as a split, it will never be selected again as a split in the tree

2. If a feature has been selected as a split, it will never be selected as a split in the next level in the tree

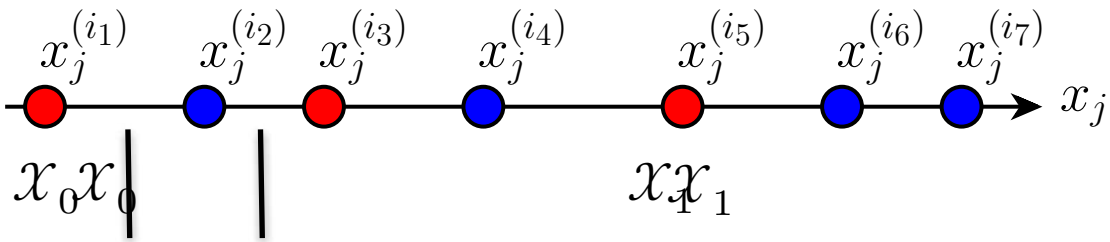
3. Assuming no training points are identical, decision trees can always obtain zero error if they are trained deep enough

4. The loss will never increase after a split

Continuous features

What if x_j's are continuous?

Solution: sort the examples by their x_j values, compute information gain at each possible split point



Regression trees

Regression trees are the same, except that the hypothesis h are real-valued instead of probabilities, and we use squared loss

$$\ell(h, y) = (h - y)^2$$

This means that the loss a node is given by

$$\underset{h}{\text{minimize}} \ \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} (h - y)^2 \implies h = \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} y \ \ (\text{i. e. mean})$$

and suffers loss

$$\ell = \frac{1}{|\mathcal{X}|} \sum_{x,y \in \mathcal{X}} (y - h)^2 \quad (\text{i. e. variance})$$

Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

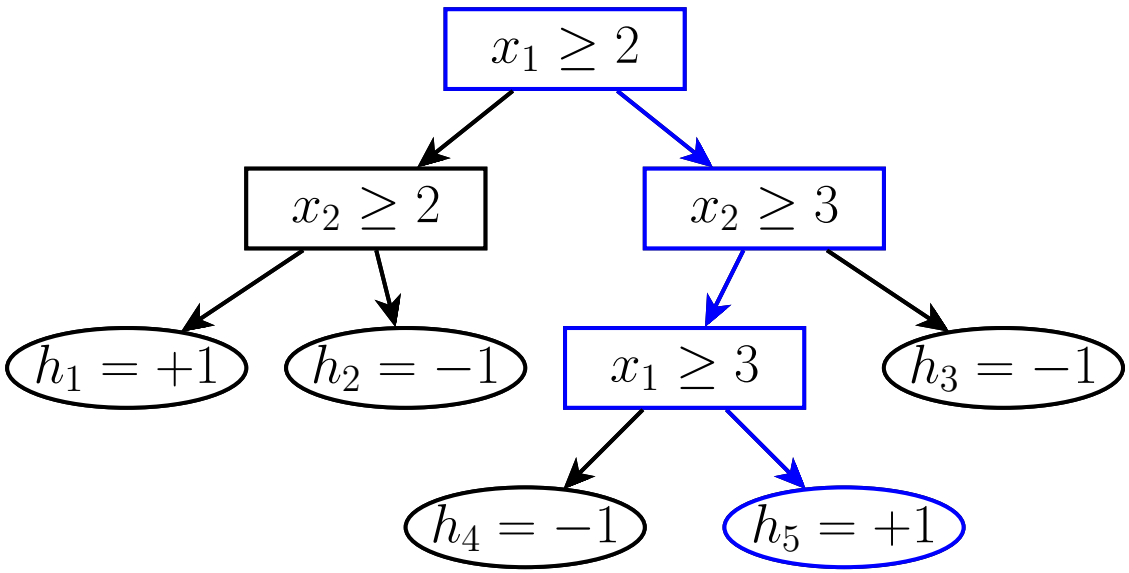
Boosting

Examples

Interpretable models

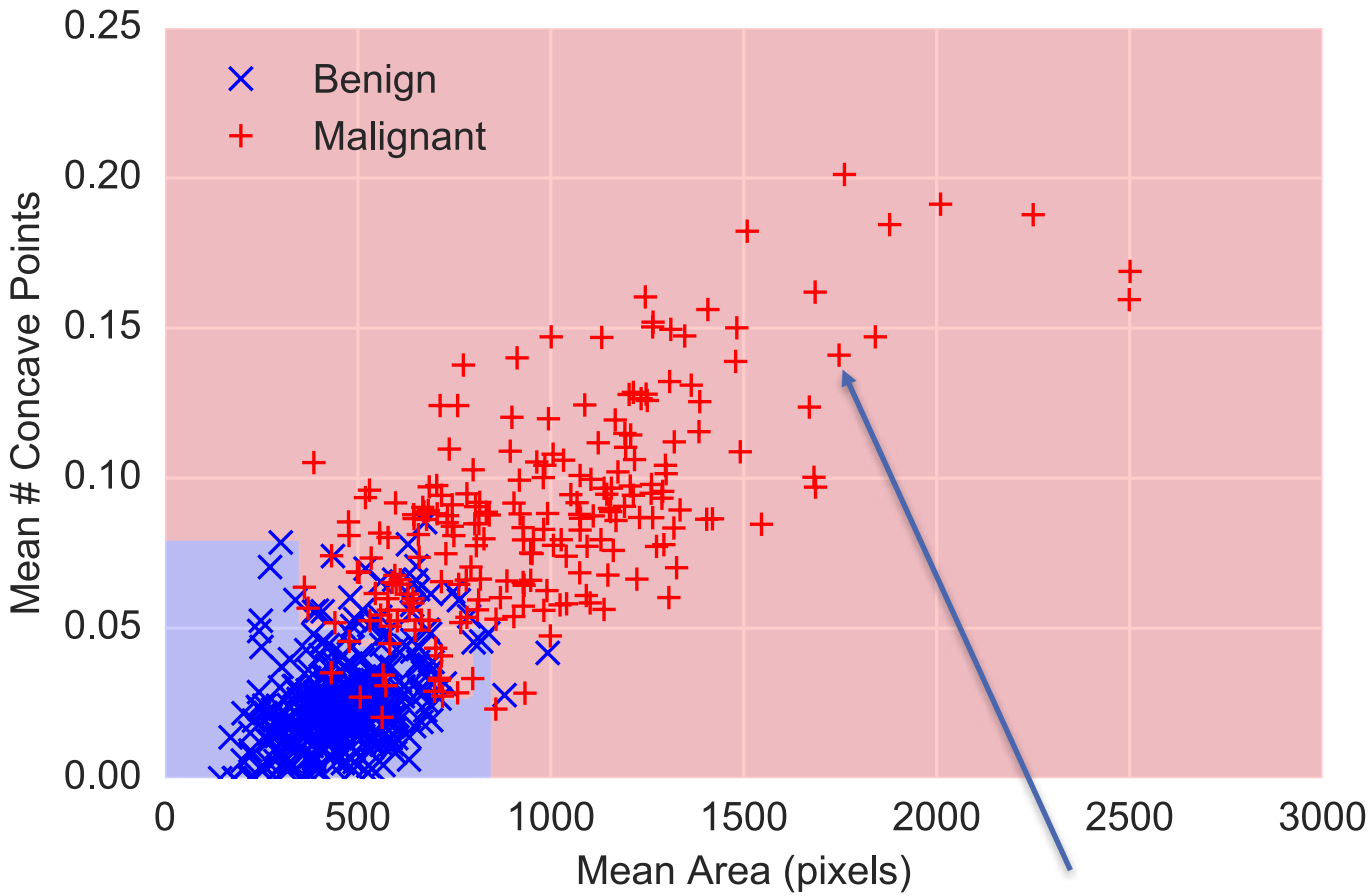
Decision trees are the canonical example of an interpretable model

Why did we we predict +1?



…because $x_1 \geq 2$, $x_2 \geq 3$, $x_1 \geq 3$

Decision tree surface for cancer prediction



…because mean concave points > 0.05, mean area > 791

Explanations in higher dimensions

Explanatory power works “well” even for data with high dimension

Example from full breast cancer dataset with 30 features, “example classified as positive because”

$$\text{max_perimeter} > 117.41, \text{max_concave_points} > 0.087$$

Compare to linear classifier, “exampled classified positive because”

$$2.142 * \text{mean_radius} + 0.119 * \text{mean_texture} + \ldots > 0$$

Do we care about interpretability?

A philosophical question underlying a lot of work in machine learning: should we seek to make predictions that can be "explained"

My own take:

- **Data scientist 1:** I'd rather have a doctor that was 90% accurate than one who was 80% accurate, even if the 80% one gave me explanations.
- **Data scientist 2:** But if the classifier isn't interpretable it may often make "foolish" mistakes because of idiosyncrasies in data.
- **Data scientist 1:** That is not a question of interpretability, that is a question of training set / testing set mismatch.
- **Data scientist 2:** Want to know a good way of telling if you have training set / testing set mismatch? Train an interpretable classifier.

Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

Boosting

Examples

Ensembles of trees

Decision trees have notable advantages (they are relatively easy to interpret, usually fast to train, insensitive to scale of input features)

But, they are also quite limited in their representation power (require axis-aligned splits, don't model probabilities very smoothly)

Basic idea of tree *ensemble* methods is to combine *multiple* tree models together to form a better predictor

Two of the most popular ensemble methods: random forests and boosting

(These models no longer interpretable in the sense we have discussed)

Boosting

Boosting originated as an idea in theoretical machine learning, for "boosting" the performance of weak classifiers (i.e., combining many classifiers that each had modest accuracy to one that had high accuracy)

After some initial success in theory, during the 90s there came to be several practical applications of boosting methods

There are *many* interpretations of boosting (experts still disagree on the "right" one!), and I'm going to highlight one:

We focus on the Gradient Boosted Regression Trees (GBRT) algorithm

Machine learning with general predictions

Let's consider the basic machine learning optimization problem (could be any loss function, classification or regression)

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

where $\hat{y}^{(i)}$ denotes our prediction for the ith example

The gradient with respect to these *predictions* themselves to determine the best way to adjust our predictions (ignoring whether we have any hypothesis function that could actually change the predictions in this way)

$$\frac{\partial}{\partial \hat{y}^{(i)}} \ell(\hat{y}^{(i)}, y^{(i)})$$

Basic idea of GBRTs

Basic idea: GBRTs are effectively performing *gradient descent* on our loss function, using *regression trees* to approximate the gradient

Given: Data set $\left(x^{(i)}, y^{(i)}\right)_{i=1,\dots,m}$, # trees T, loss ℓ, step size α

Initialize:

$$\hat{y}^{(i)} \leftarrow 0, \quad \forall\ i = 1, \dots, m$$

For $t = 1, \dots, T$:

Compute gradients:

$$g_t^{(i)} \leftarrow \frac{\partial}{\partial \hat{y}} \ell\left(\hat{y}^{(i)}, y^{(i)}\right), \quad \forall\ i = 1, \dots, m$$

$$h_t \leftarrow \ \mathrm{Train_Regression_Tree}\left(x^{(1,\dots,m)}, g_t^{(1,\dots,m)}\right)$$

Update predictions:

$$\hat{y}^{(i)} \leftarrow \hat{y}^{(i)} - \alpha h_t\left(x^{(i)}\right)$$

For new data point x:

Predict: $\hat{y} = -\alpha \sum_{i=1}^{T} h_t\left(x^{(i)}\right)$

GBRTs a bit more practically

In practice, fitting the trees is slow, so we actually do a line search to determine how large of a gradient step to take

Can take different gradient steps at different leaves in tree

Here you probably want to use a library (you *could* write your own implementation, but it would be about ~100 lines of code, not ~5 like for an SVM)

Outline

Decision trees

Training (classification) decision trees

Interpreting predictions

Boosting

Examples

Decision trees and GBRTs in scikit-learn

Interface for decision trees and GBRTs in scikit-learn is just like any other classifier

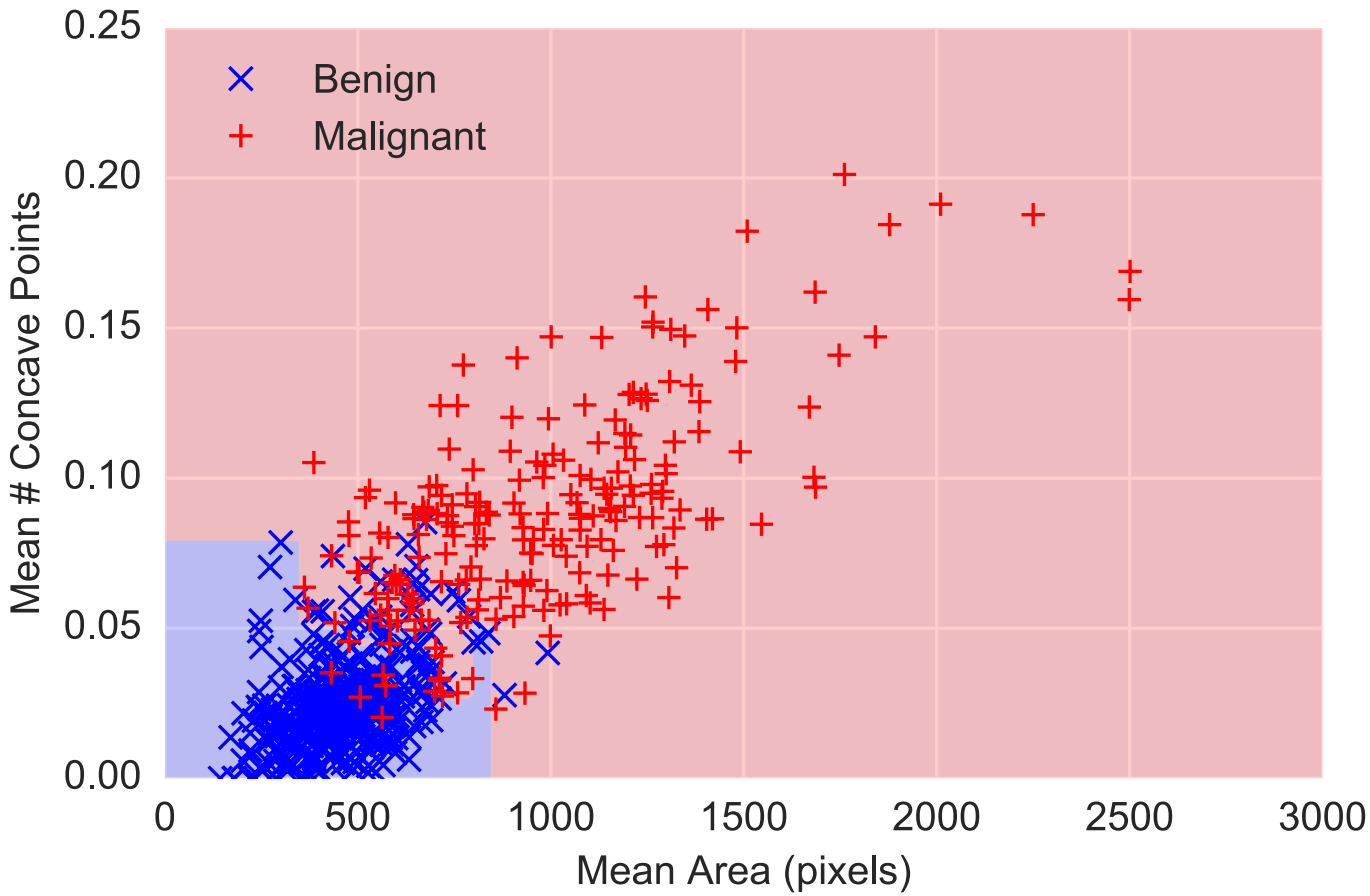
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion='entropy', max_depth=5)
clf.fit(X, y)
```

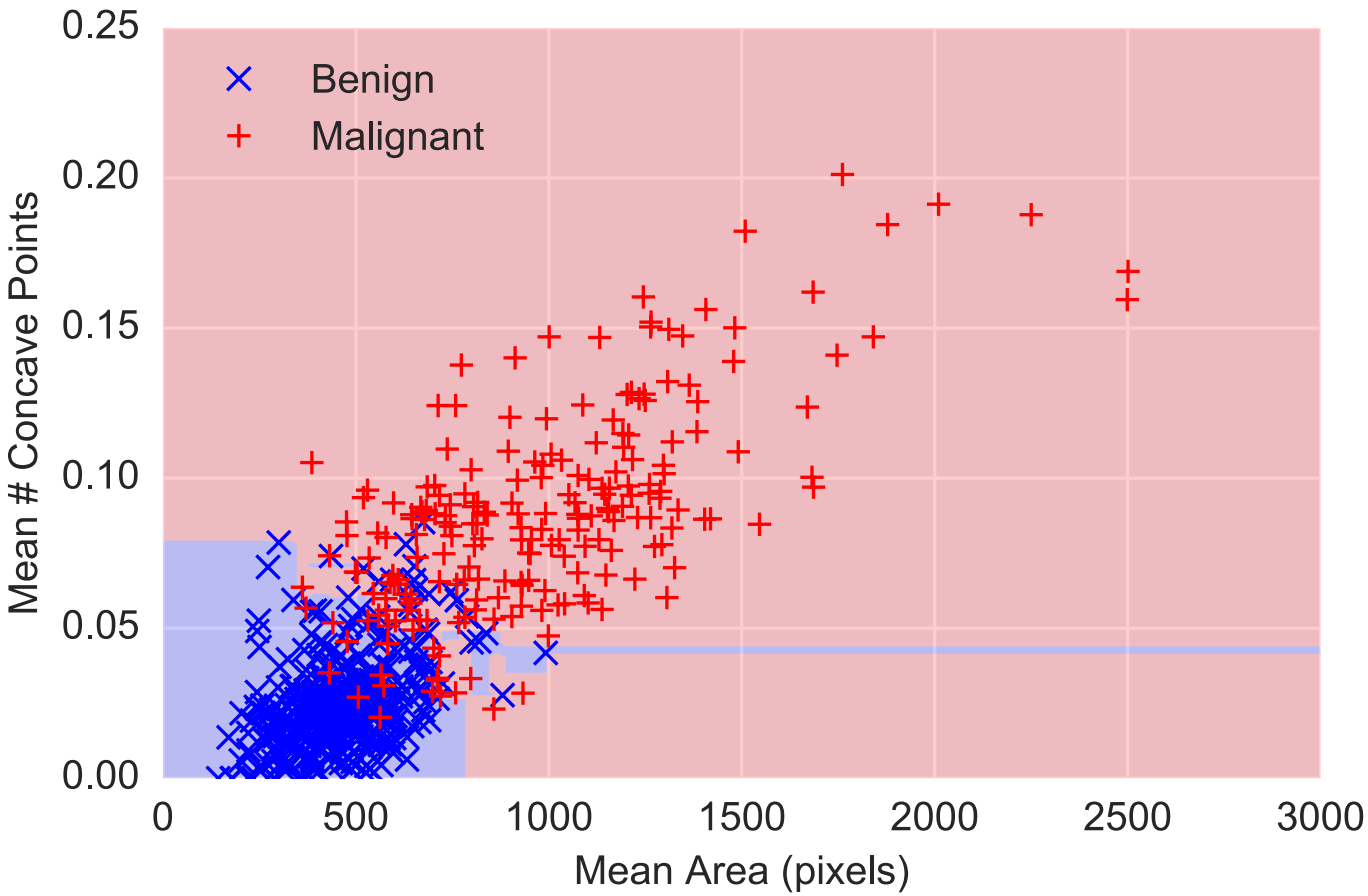
```
from sklearn.ensemble import GradientBoostingClassifier

clf = GradientBoostingClassifier(loss='deviance',
                                 max_depth=3, n_estimators=100)
clf.fit(X, y)
```

Decision tree surface for cancer prediction



GBRT surface for cancer prediction



15-388/688 - Practical Data Science: Jupyter notebook lab

J. Zico Kolter
Carnegie Mellon University
Fall 2019

Announcements

Recitation tomorrow (Thursday, **9/5**) from **6-8pm** in Doherty Hall 2210 (this room)

Waitlist will cleared continually throughout the day and tomorrow (there is room in the course)

Homework advice: make sure your code passes included local tests (and ideally, write more tests, and look up the relevant ones on Diderot) **before** submitting to Diderot

Outline

Python and Jupyter Notebook

Jupyter lab

Outline

Python and Jupyter Notebook

Jupyter lab

Python

"The language of data science"

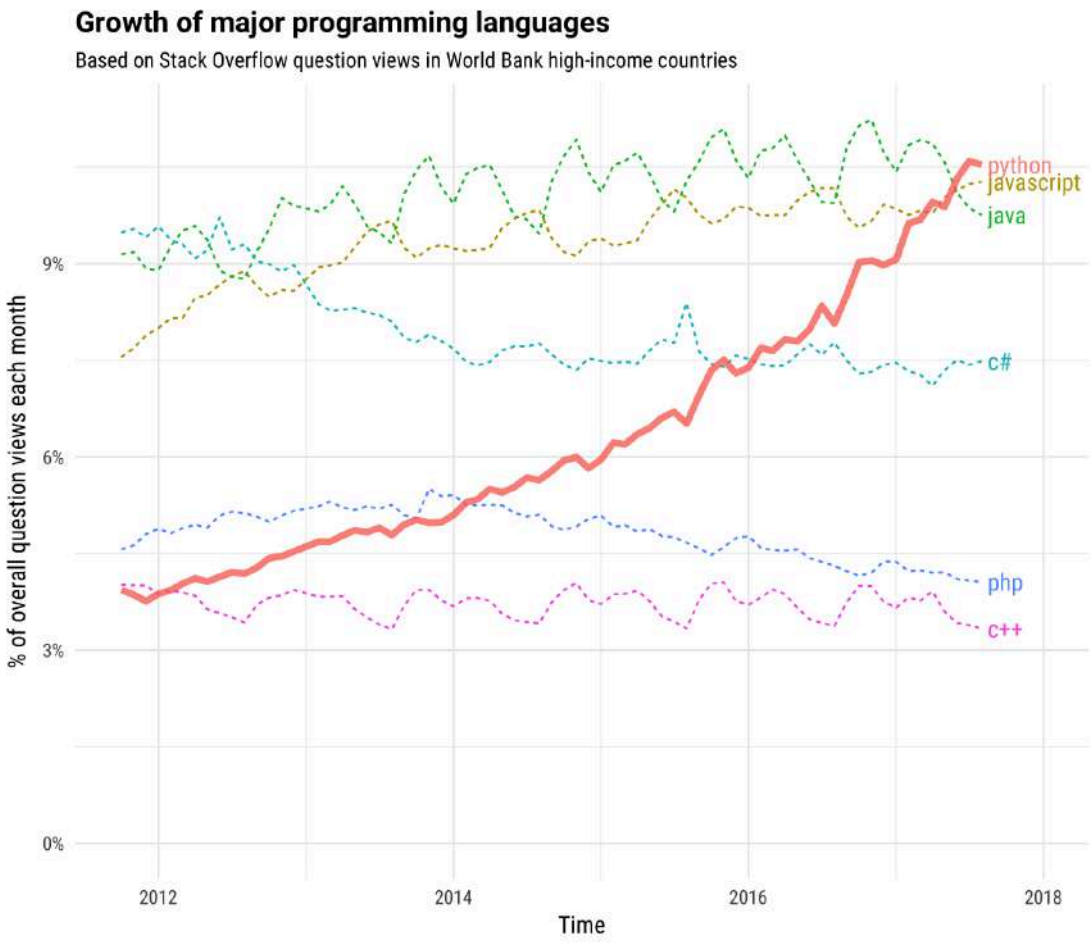
- Especially true if the data science tasks involve lots of data processing and/or machine learning
- Less true if the tasks are more "purely statistical" (then R is more standard)

Python 2->3 debacle

The most visible changes to the language in Python 3 (honestly) are:

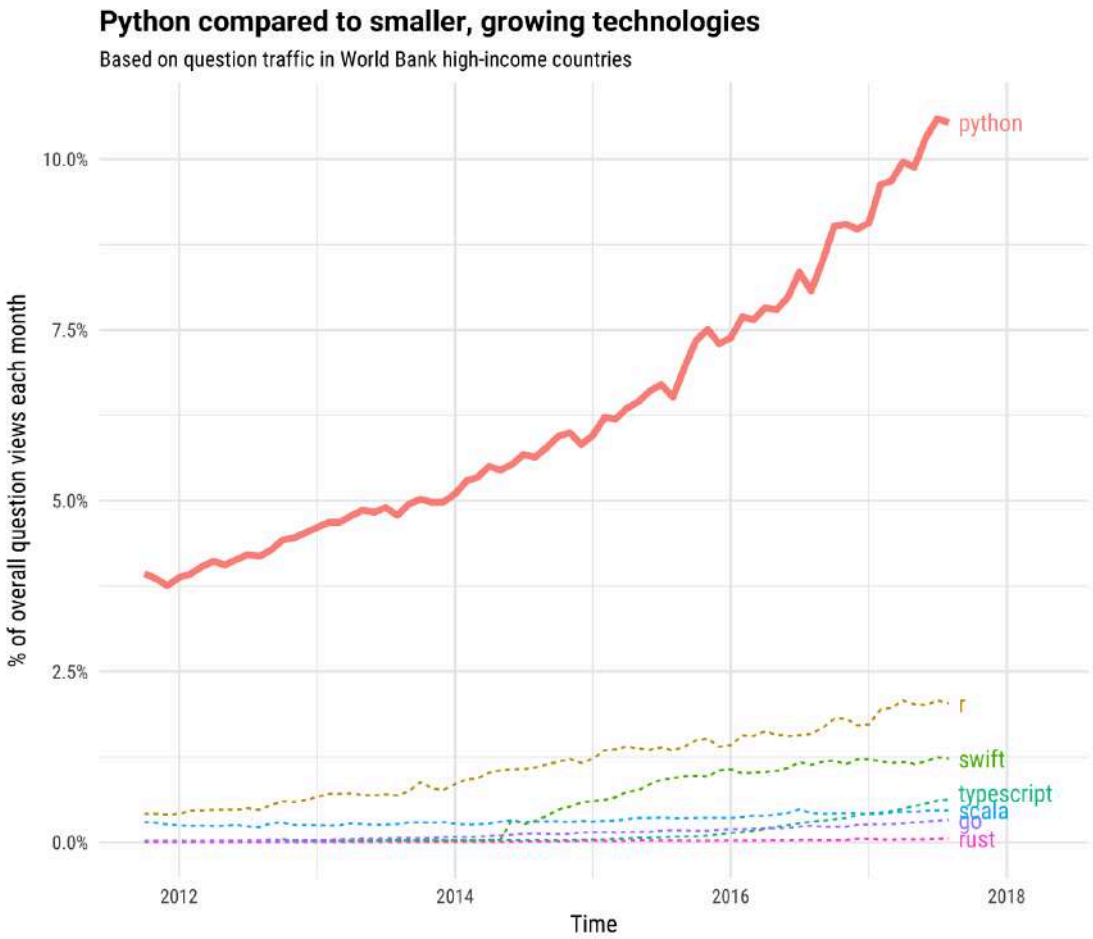
1. `print` is a command, not a statement (so you need parentheses)
2. `1/2` returns 0.5 (floating point), not 0 (integer); to get 0, you use the operation `1//2`

Python growth



Source: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Python growth



Source: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Anaconda

For this class, we strongly recommend you use Anaconda, a common distribution of Python, which includes several common libraries and tools including the Jupyter notebook and a package manager, available at:

https://www.anaconda.com/download/

You can verify you are using the Anaconda distribution by running Python and making sure you see something like the following:

```
Zicos-MacBook-Pro-2:jupyter zkolter$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct  6 2017, 12:04:38)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Installing additional packages

Several of the homework assignments will require that you have additional libraries

There are two typical ways to install these, via the conda package manager (part of Anaconda), and via pip:

- conda install beautifulsoup4 – install BeautifulSoup4
- conda search beautiful – search conda packages for any that includes the string “beautiful”
- pip install beautifulsoup4 – install BeautifulSoup4
- pip search beautiful – search pip packages for any that include the string “beautiful”

Rule of thumb: use conda when you can (plays nicer with Anaconda installation), but some packages can only be installed via pip

Jupyter notebook

All course assignments (and even the notes) are distributed as Jupyter notebooks

Jupyter notebooks are a browser-based environment for writing code, interspersing code and Markdown, and displaying figures, all contained in “cells”

- More info about Jupyter here: http://www.jupyter.org

Launch jupyter via the command:

- jupyter notebook
- Then navigate to http://localhost:8888 (or possibly a later port number, if you have multiple notebooks open)

Tips for homework

Carefully follow problem specifications to match the output required by Diderot

Test your code locally on the provided test cases *and* additional test cases you create, to ensure it gives the expected output for all inputs you can come up with

You “should” be able to exactly know your Diderot score before you even submit, because the code passes all local tests (or at least most of the tests)

Outline

Python and Jupyter Notebook

Jupyter lab

Jupyter lab

(Continued in live notebook)

Poll: Jupyter notebook

```
In [21]: a = 1.0

In [17]: a = 2.0

In [ ]: a = 3.0
```

What is the current value of the variable `a` in this notebook (assuming that no other cells exist)?

1. 1.0
2. 2.0
3. 3.0

Poll: Jupyter notebook

```
In [5]: def function(a):
            return b*a**2
```

```
In [4]: a = 2.0
        b = 2.0
```

```
In [ ]: function(4)
```

What will be the output of the selected cell?

1. 8.0
2. 16.0
3. 32.0
4. Error: “b” is undefined

15-388/688 - Practical Data Science: Debugging data science

J. Zico Kolter
School of Computer Science
Fall 2019

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Data science debugging

Imagine this: you go about defining a data science problem, define input/output pairs for a prediction task, only to find that when you run some machine learning algorithm, it doesn't work

This happens to everyone

"What differentiates experts in data science from others is not what you do first, it's what you do second when that first thing doesn't work."

- Kolter's Law

Traditional debugging

Traditional debugging of programs is relatively straightforward

You have some desired input/output pairs

You have a mental model (or maybe something more formal) of how each step in the algorithm “should” work

You trace through the execution of the program (either through a debugger or with print statement), to see where the state diverges from your mental model (or to discover your mental model is wrong)

Data science debugging

You have some desired input/output pairs

Your mental model is that an ML algorithm should work because … math? ... magic?

What can you trace through to see why it may not be working? Not very useful to step through an implementation of logistic regression…

Debugging data science vs. machine learning

Many of the topics here overlap with material on “debugging machine learning”

We are indeed going to focus largely on debugging data science prediction tasks (debugging web scraping, etc, is much more like traditional debugging)

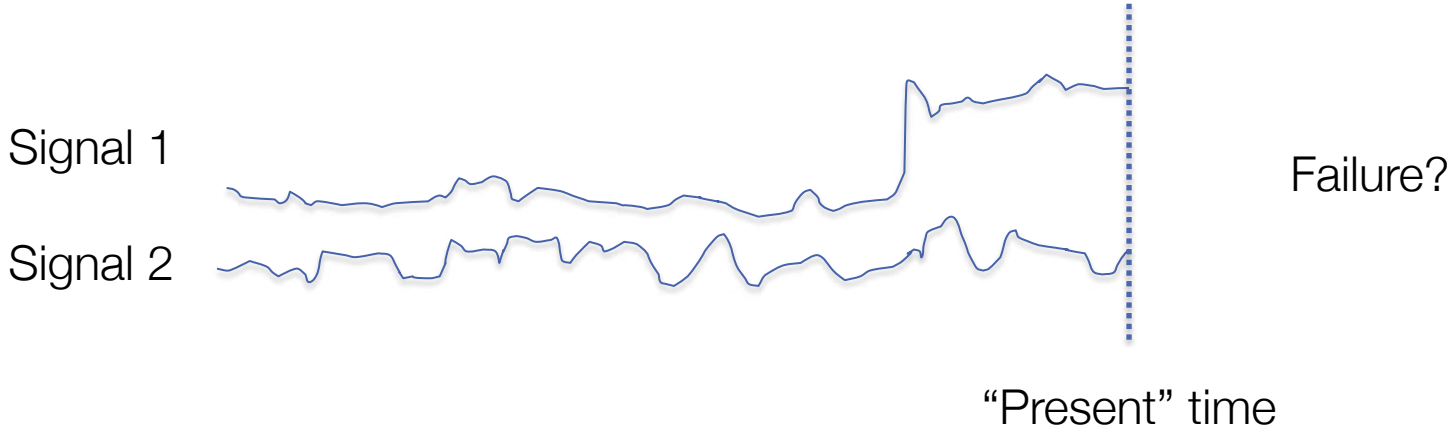
But, there is also a more abstract concept here of debugging the *problem* instead of debugging the *algorithm*

An example: predictive maintenance

An example task: you run a large factory and what to predict whether any given machine will fail within the next 90 days

You're given signals monitoring the state of this device

You want to predict binary response of whether the machine will fail



Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

The first step of data science debugging

Step 1: determine if your problem is impossible

There are plenty of tasks that would be really nice to be able to predict, and absolutely no evidence that there the necessary signals to predict them (see e.g., predicting stock market from Twitter)

But, hope springs eternal, and it's hard to prove a negative…

A good proxy for impossibility

Step 1: ~~determine if your problem is impossible~~ see if *you* can solve your problem manually

Create an interface where you play the role of the prediction algorithm, you need to make the predictions of the outputs given the available inputs

To do this, you'll need to provide some intuitive way of visualizing what a complete set of input features looks like: tabular data for a few features, raw images, raw text, etc

Just like a machine learning algorithm, you can refer to training data (where you know the labels), but you can't peak at the answer on your test/validation set

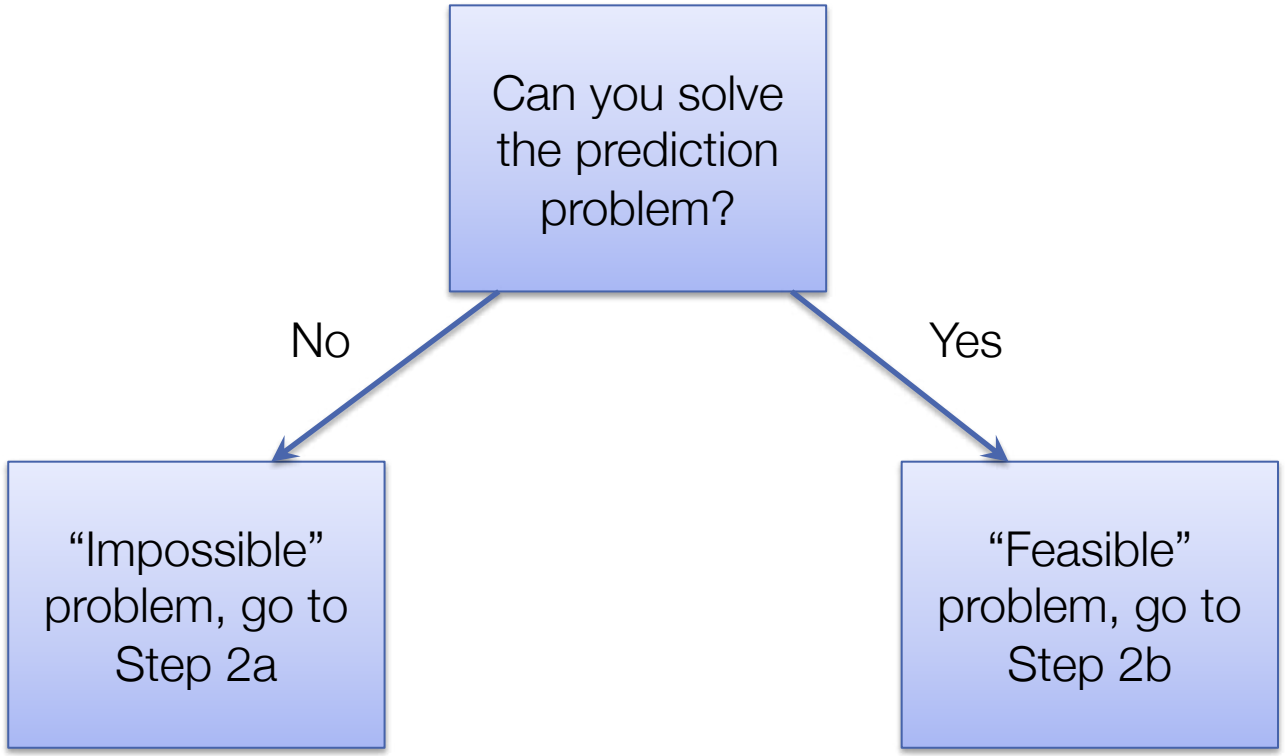
What about “superhuman” machine learning

It’s a common misconception that machine learning will *outperform* human experts on most tasks

In reality, the benefit from machine learning often doesn’t come from superhuman performance in most cases, it comes from the ability to scale up expert-level performance extremely quickly

If you can’t make good predictions, neither will a machine learning algorithm (at least the first time through, and probably always)

Decision diagram



Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Dealing with “impossible” problems

So you’ve built a tool to manually classify examples, run through many cases (or had a domain expert run through them), and you get poor performance

What do you do?

You do *not* try to throw more, bigger, badder, machine learning algorithms at the problem

Instead you need to change the problem by: 1) changing the input (i.e., the features), 2) changing the output (i.e., the problem definition)

Changing the input (i.e., adding features)

The fact that we can always add more features is what makes these problems "impossible" (with quotes) instead of impossible (no quotes)

You can always hold out hope that you just one data source away from finding the "magical" feature that will make your problem easy

But you probably aren't… adding more data is good, but:

1. Do spot checks (visually) to see if this new features can help *you* differentiate between what you were previously unable to predict

2. Get advice from domain experts, see what sorts of data source they use in practice (if people are already solving the problem)

Changing the output (i.e., changing the problem)

Just make the problem easier! (well, still need to preserve the character of the data science problem)

A very useful procedure: instead of trying to predict the future, try to predict what an expert would predict given the features you have available

E.g., for predictive maintenance this shifts the question from: “would this machine fail?” to “would an expert choose to do maintenance on this machine?”

With this strategy we already have an existence proof that it’s feasible

Changing the output #2

Move from a question of getting “good” prediction to a question of characterizing the uncertainty of your predictions

Seems like a cop-out, but many tasks are *inherently* stochastic, the best you can do is try to quantify the likely uncertainty in output given the input

E.g.: if 10% of all machines fail within 90 days, it can still be really valuable to predict if whether a machine will fail with 30% probability

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

Dealing with feasible problems

Good news! Your prediction problem seems to be solvable (because you can solve it)

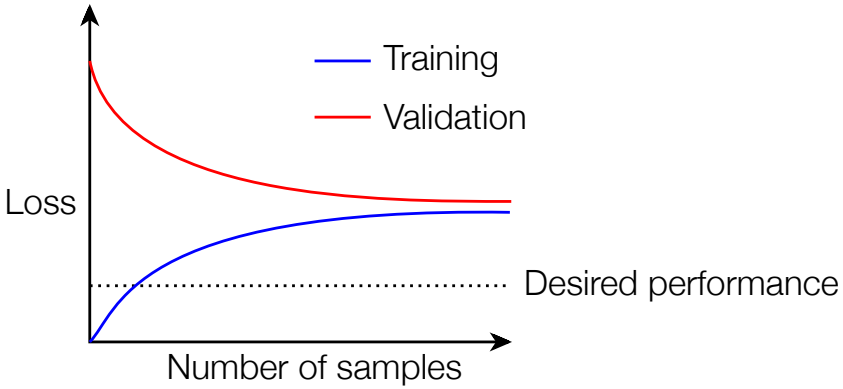
You run your machine learning algorithm, and find that it doesn't work (performs worse than you do)

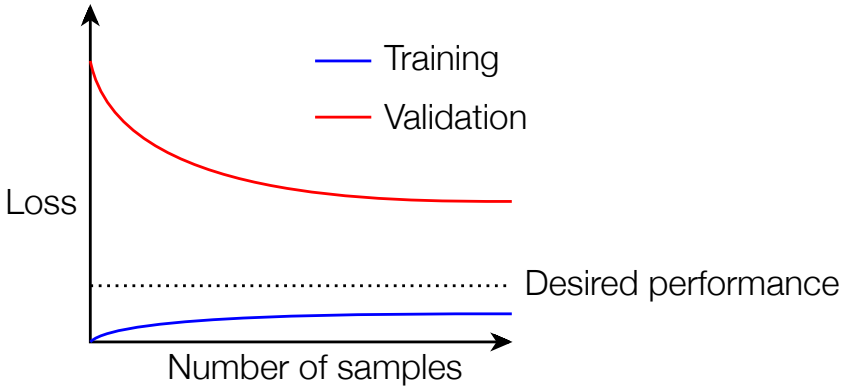
Again, you can try just throwing more algorithms, data, features, etc, at the problem, but this is unlikely to succeed

Instead you want to build diagnostics that can check what the problem may be

Characterizing bias vs. variance

Consider the training and testing loss of your algorithm (often plotting over different numbers of samples), to determine if you problem is one of high bias or high variance



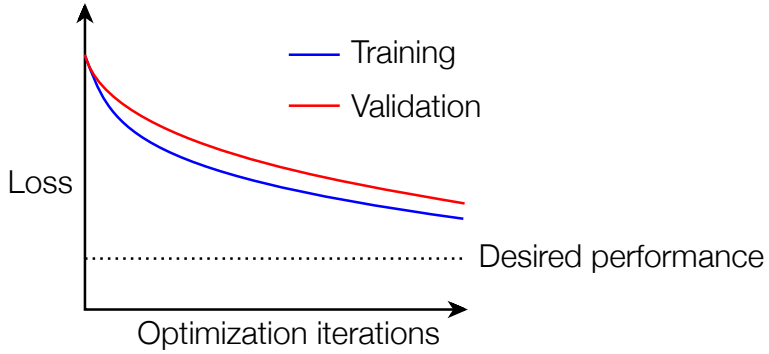


For high bias, add features based upon your own intuition of how you solved the problem

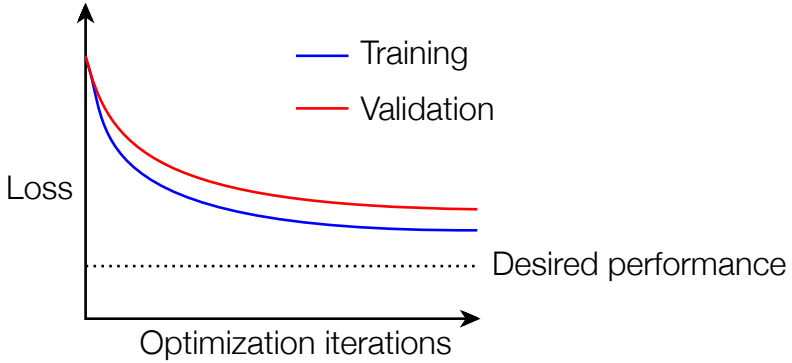
For high variance, add data or remove features (keeping features based upon your intuition)

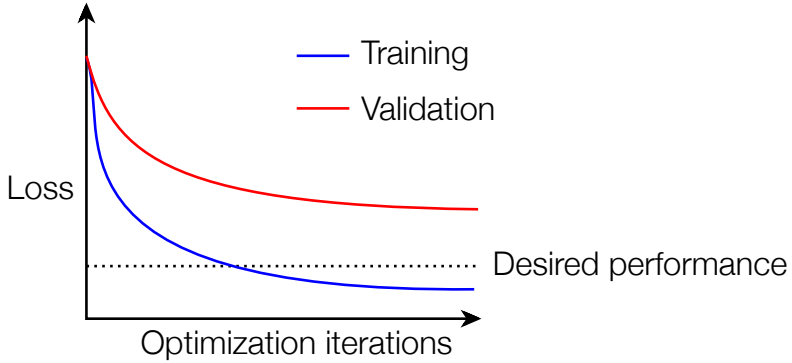
Characterizing optimization performance

It is a much less common problem, but you may want to look at training/testing loss versus algorithm iteration, may look like this:



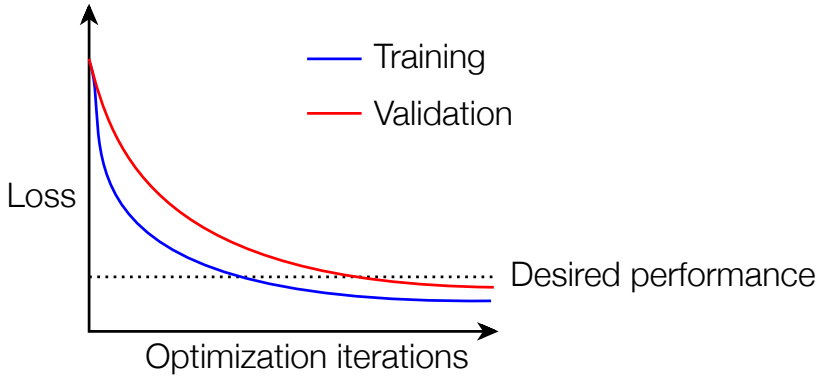
But it probably looks like this:

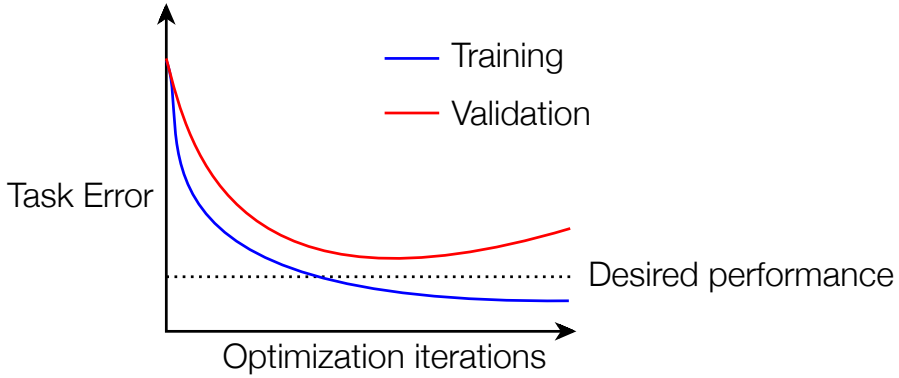




Consider loss vs. task error

Remember that machine learning algorithms try to minimize some loss, which may be different from the task error you actually want to optimize





This is common when dealing e.g. with imbalanced data sets for which cost of different classifications is very different

Get a Ph.D.

Your problem may genuinely be “AI Hard”, you can solve it, but a computer cannot

These are typically the most “interesting” questions from a research standpoint

But, presuming you don’t want to interrupt your career for a life in academics (though I can’t fathom why not), you will want to go back to adjusting the problem itself to be feasible given a more limited set of features

Outline

Data science debugging vs. traditional debugging

Step 1: determine if your problem is impossible

Step 2a: What to do about "impossible" problems?

Step 2b: What to do about feasible problems?

Step 1b: Impossibly good performance

One more possibility

You run your machine learning algorithm right off the bat (because I know you won't actually write the diagnostics first), and it works great

Be skeptical, unless it's "obvious" how to solve the problem

Still try to solve the problem manually, and if you can't, be *extremely skeptical*

The reason: it's very easy to accidentally set up a problem that lets the algorithm "cheat", use a feature that is virtually a deterministic one-to-one function of desired output

I have seen companies "fix" their historical data in a way that makes it easy to predict upcoming failure…

Poll: A very bad data science product

(This is based upon, sadly, a true story). You are interviewing with a company that claims to have built a ML-based lie detector test, which can determine whether a person is lying from a video of their conversation. Where would the data science debugging process presumably break down for this problem?

1. Step 1: Determining if your problem is impossible

2. Step 2a: Debugging and correcting impossible problems

3. Step 2b: Debugging machine learning performance

4. Step 1b: Getting impossibly good performance

Summary

	ML predicts well	ML predicts poorly
You predict well	Congrats, but still be skeptical and do a bit of analysis	“Feasible” problem, debug your machine learning algorithm
You predict poorly	Be skeptical, make sure you aren’t cheating	“Impossible” problem, debug the problem

15-388/688 - Practical Data Science: Maximum likelihood estimation, naïve Bayes

J. Zico Kolter
Carnegie Mellon University
Spring 2018

Outline

Maximum likelihood estimation

Naive Bayes

Machine learning and maximum likelihood

Outline

Maximum likelihood estimation

Naive Bayes

Machine learning and maximum likelihood

Estimating the parameters of distributions

We're moving now from probability to statistics

The basic question: given some data $x^{(1)}, \ldots, x^{(m)}$, how do I find a distribution that captures this data "well"?

In general (if we can pick from the space of all distributions), this is a hard question, but if we pick from a particular *parameterized family* of distributions $p(X; \theta)$, the question is (at least a little bit) easier

Question becomes: how do I find parameters θ of this distribution that fit the data?

Maximum likelihood estimation

Given a distribution $p(X;\theta)$, and a collection of observed (independent) data points $x^{(1)}, \dots, x^{(m)}$, the probability of observing this data is simply

$$p(x^{(1)}, \dots, x^{(m)}; \theta) = \prod_{i=1}^{m} p(x^{(i)}; \theta)$$

Basic idea of maximum likelihood estimation (MLE): find the parameters that maximize the probability of the observed data

$$\underset{\theta}{\text{maximize}} \ \prod_{i=1}^{m} p(x^{(i)}; \theta) \ \equiv \ \underset{\theta}{\text{maximize}} \ \ell(\theta) = \sum_{i=1}^{m} \log p(x^{(i)}; \theta)$$

where $\ell(\theta)$ is called the **log likelihood** of the data

Seems “obvious”, but there are many other ways of fitting parameters

Parameter estimation for Bernoulli

Simple example: Bernoulli distribution

$$p(X = 1; \phi) = \phi, \qquad p(X = 0; \phi) = 1 - \phi$$

Given observed data $x^{(1)}, \ldots, x^{(m)}$, the "obvious" answer is:

$$\hat{\phi} = \frac{\#1\text{'s}}{\#\ \text{Total}} = \frac{\sum_{i=1}^{m} x^{(i)}}{m}$$

But why is this the case?

Maybe there are other estimates that are just as good, i.e.?

$$\phi = \frac{\sum_{i=1}^{m} x^{(i)} + 1}{m + 2}$$

MLE for Bernoulli

Maximum likelihood solution for Bernoulli given by

$$\underset{\phi}{\text{maximize}} \prod_{i=1}^{m} p(x^{(i)}; \phi) = \underset{\phi}{\text{maximize}} \prod_{i=1}^{m} \phi^{x^{(i)}} (1-\phi)^{1-x^{(i)}}$$

Taking the negative log of the optimization objective (just to be consistent with our usual notation of optimization as minimization)

$$\underset{\phi}{\text{maximize}}\ \ell(\phi) = \sum_{i=1}^{m} \left(x^{(i)} \log \phi + (1 - x^{(i)}) \log(1-\phi) \right)$$

Derivative with respect to ϕ is given by

$$\frac{d}{d\phi} \ell(\phi) = \sum_{i=1}^{m} \left(\frac{x^{(i)}}{\phi} - \frac{1 - x^{(i)}}{1-\phi} \right) = \frac{\sum_{i=1}^{m} x^{(i)}}{\phi} - \frac{\sum_{i=1}^{m} (1 - x^{(i)})}{1-\phi}$$

MLE for Bernoulli, continued

Setting derivative to zero gives:

$$\frac{\sum_{i=1}^{m} x^{(i)}}{\phi} - \frac{\sum_{i=1}^{m}(1 - x^{(i)})}{1-\phi} \equiv \frac{a}{\phi} - \frac{b}{1-\phi} = 0$$
$$\implies (1-\phi)a = \phi b$$
$$\implies \phi = \frac{a}{a+b} = \frac{\sum_{i=1}^{m} x^{(i)}}{m}$$

So, we have shown that the “natural” estimate of ϕ actually corresponds to the maximum likelihood estimate

Poll: Bernoulli maximum likelihood

Suppose we observe binary data $x^{(1)}, \ldots, x^{(m)}$ with $x^{(i)} \in \{0,1\}$ with some $x^{(i)} = 0$ and some $x^{(j)} = 1$, and we compute the Bernoulli MLE

$$\phi = \frac{\sum_{i=1}^{m} x^{(i)}}{m}$$

Which of following statements is necessarily true? (may be more than one)

1. For any $\phi' \neq \phi$, $p(x^{(i)}; \phi') \leq p(x^{(i)}; \phi\ )$ for all $i = 1, \ldots, n$

2. For any $\phi' \neq \phi$, $\prod_{i=1}^{m} p(x^{(i)}; \phi') \leq \prod_{i=1}^{m} p(x^{(i)}; \phi\ )$

3. We always have $p(x^{(i)}; \phi') \geq p(x^{(i)}; \phi\ )$ for at least one i

MLE for Gaussian, briefly

For Gaussian distribution

$$p(x; \mu, \sigma^2) = (2\pi\sigma^2)^{-1/2} \exp(-(1/2)(x-\mu)^2/\sigma^2)$$

Log likelihood given by:

$$\ell(\mu, \sigma^2) = -m\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\sum_{i=1}^{m}\frac{(x^{(i)}-\mu)^2}{\sigma^2}$$

Derivatives (see if you can derive these fully):

$$\frac{d}{d\mu}\ell(\mu, \sigma^2) = -\frac{1}{2}\sum_{i=1}^{m}\frac{x^{(i)}-\mu}{\sigma^2} = 0 \Longrightarrow \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\frac{d}{d\sigma^2}\ell(\mu, \sigma^2) = -\frac{m}{2\sigma^2} + \frac{1}{2}\sum_{i=1}^{m}\frac{(x^{(i)}-\mu)^2}{(\sigma^2)^2} = 0 \Longrightarrow \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)^2$$

Outline

Maximum likelihood estimation

Naive Bayes

Machine learning and maximum likelihood

Naive Bayes modeling

Naive Bayes is a machine learning algorithm that rests relies heavily on probabilistic modeling

But, it is also interpretable according to the three ingredients of a machine learning algorithm (hypothesis function, loss, optimization), more on this later

Basic idea is that we model input and output as random variables $X = (X_1, X_2, \ldots, X_n)$ (several Bernoulli, categorical, or Gaussian random variables), and Y (one Bernoulli or categorical random variable), goal is to find $p(Y|X)$

Naive Bayes assumptions

We're going to find $p(Y|X)$ via Bayes' rule

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)} = \frac{p(X|Y)p(Y)}{\sum_y p(X|y)\,p(y)}$$

The denominator is just the sum over all values of Y of the distribution specified by the numeration, so we're just going to focus on the $p(X|Y)p(Y)$ term

Modeling full distribution $p(X|Y)$ for high-dimensional X is not practical, so we're going to make the **naive Bayes assumption**, that the elements X_i are conditionally independent given Y

$$p(X|Y) = \prod_{i=1}^{n} p(X_i|Y)$$

Modeling individual distributions

We're going to explicitly model the distribution of each $p(X_i|Y)$ as well as $p(Y)$

We do this by specifying a distribution for $p(Y)$ and a *separate* distribution and for each $p(X_i|Y=y)$

So assuming, for instance, that Y_i and X_i are binary (Bernoulli random variables), then we would represent the distributions

$$p(Y;\phi_0), \qquad p(X_i|Y=0;\phi_i^0), \qquad p(X_i|Y=1;\phi_i^1)$$

We then estimate the parameters of these distributions using MLE, i.e.

$$\phi_0 = \frac{\sum_{j=1}^m y^{(j)}}{m}, \qquad \phi_i^y = \frac{\sum_{j=1}^m x_i^{(j)} \cdot 1\{y^{(j)}=y\}}{\sum_{j=1}^m 1\{y^{(j)}=y\}}$$

Making predictions

Given some new data point x, we can now compute the probability of each class

$$p(Y = y|x) \propto p(Y = y) \prod_{i=1}^{m} p(x_i|Y = y) = \phi_0 \prod_{i=1}^{m} (\phi_i^y)^{x_i} (1 - \phi_1^y)^{1-x_i}$$

After you have computed the right hand side, just normalize (divide by the sum over all y) to get the desired probability

Alternatively, if you just want to know the most likely Y, just compute each right hand side and take the maximum

Example

Y	X_1	X_2
0	0	0
1	1	0
0	0	1
1	1	1
1	1	0
0	1	0
1	0	1
?	1	0

$$p(Y=1) = \ \phi_0 =$$

$$p(X_1 = 1|Y=0) = \phi_1^0 =$$

$$p(X_1 = 1|Y=1) = \phi_1^1 =$$

$$p(X_2 = 1|Y=0) = \phi_2^0 =$$

$$p(X_2 = 1|Y=0) = \phi_2^1 =$$

$$p(Y|X_1 = 1, X_2 = 0) =$$

Potential issues

Problem #1: when computing probability, the product $\mathrm{p}(y)\prod_{i=1}^{n} p(x_i|y)$ quickly goes to zero to numerical precision

Solution: compute log of the probabilities instead

$$\log p(y) + \sum_{i=1}^{n} \log p(x_i|y)$$

Problem #2: If we have never seen either $X_i = 1$ or $X_i = 0$ for a given y, then the corresponding probabilities computed by MLE will be zero

Solution: Laplace smoothing, "hallucinate" one $X_i = 0/1$ for each class

$$\phi_i^y = \frac{\sum_{j=1}^{m} x_i^{(j)} \cdot 1\{y^{(j)} = y\} + 1}{\sum_{j=1}^{m} 1\{y^{(j)} = y\} + 2}$$

Other distributions

Though naive Bayes is often presented as "just" counting, the value of the maximum likelihood interpretation is that it's clear how to model $p(X_i|Y)$ for non-categorical random variables

Example: if x_i is real-valued, we can model $p(X_i|Y=y)$ as a Gaussian

$$p(x_i|y;\mu^y,\sigma_y^2) = \mathcal{N}(x_i;\mu^y,\sigma_y^2)$$

with maximum likelihood estimates

$$\mu^y = \frac{\sum_{j=1}^m x_i^{(j)} \cdot 1\{y^{(j)}=y\}}{\sum_{j=1}^m 1\{y^{(j)}=y\}},\ \sigma_y^2 = \frac{\sum_{j=1}^m (x_i^{(j)} - \mu^y)\hat{}2 \cdot 1\{y^{(j)}=y\}}{\sum_{j=1}^m 1\{y^{(j)}=y\}}$$

All probability computations are exactly the same as before (it doesn't matter that some of the terms are probability densities)

Outline

Maximum likelihood estimation

Naive Bayes

Machine learning and maximum likelihood

Machine learning via maximum likelihood

Many machine learning algorithms (specifically the loss function component) can be interpreted probabilistically, as maximum likelihood estimation

Recall logistic regression:

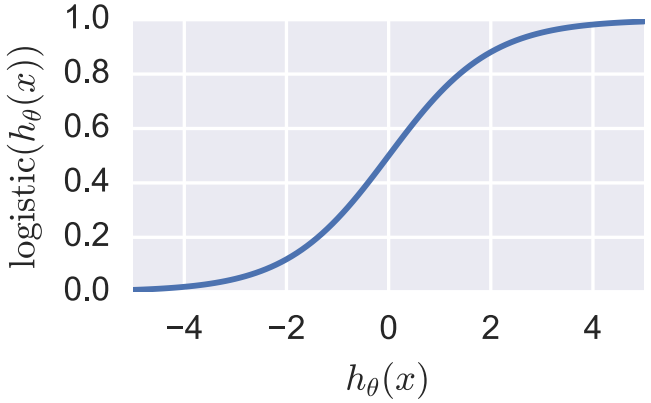
$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell_{\text{logistic}}(h_\theta(x^{(i)}), y^{(i)})$$

$$\ell_{\text{logistic}}(h_\theta(x), y) = \log(1 + \exp(-y \cdot h_\theta(x)))$$

Logistic probability model

Consider the model (where Y is binary taking on $\{-1, +1\}$ values)

$$p(y|x;\theta) = \text{logistic}\big(y \cdot h_\theta(x)\big) = \frac{1}{1 + \exp(-y \cdot h_\theta(x))}$$



Under this model, the maximum likelihood estimate is

$$\underset{\theta}{\text{maximize}} \sum_{i=1}^{m} \log p(y^{(i)} | x^{(i)}; \theta) \equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell_{\text{logistic}}(h_\theta(x^{(i)}), y^{(i)})$$

Least squares

In linear regression, assume

$$y = \theta^T x + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$
$$\iff p(y|x;\theta) = \mathcal{N}(\theta^T x, \sigma^2)$$

Then the maximum likelihood estimate is given by

$$\underset{\theta}{\text{maximize}} \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\theta) \equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

i.e., the least-squares loss function can be viewed as MLE under Gaussian errors

Other approaches possible too: absolute loss function can be viewed as MLE under Laplace errors