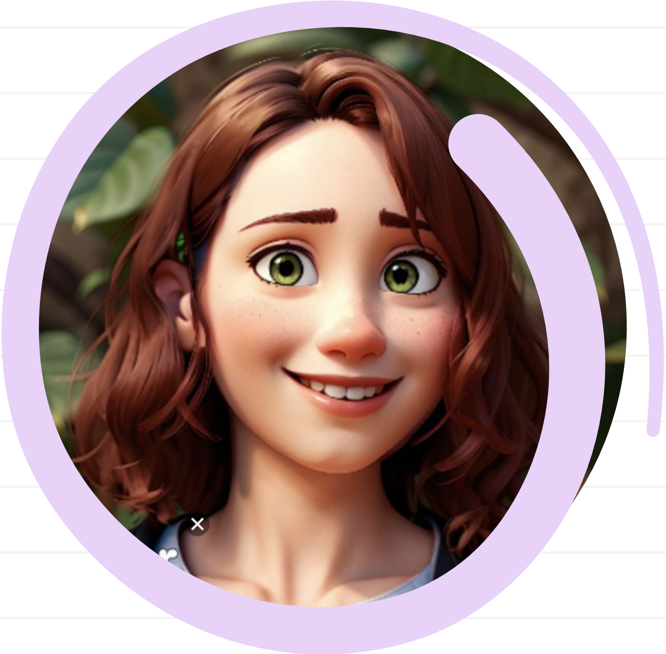
DSC 40B

Lecture 1 :

Intro/Review

Motivation

Mic!



Hello! I'm...

Marina Langlois.

I teach coding classes at HDSI :) and many of you have probably taken one, two, or even three classes!

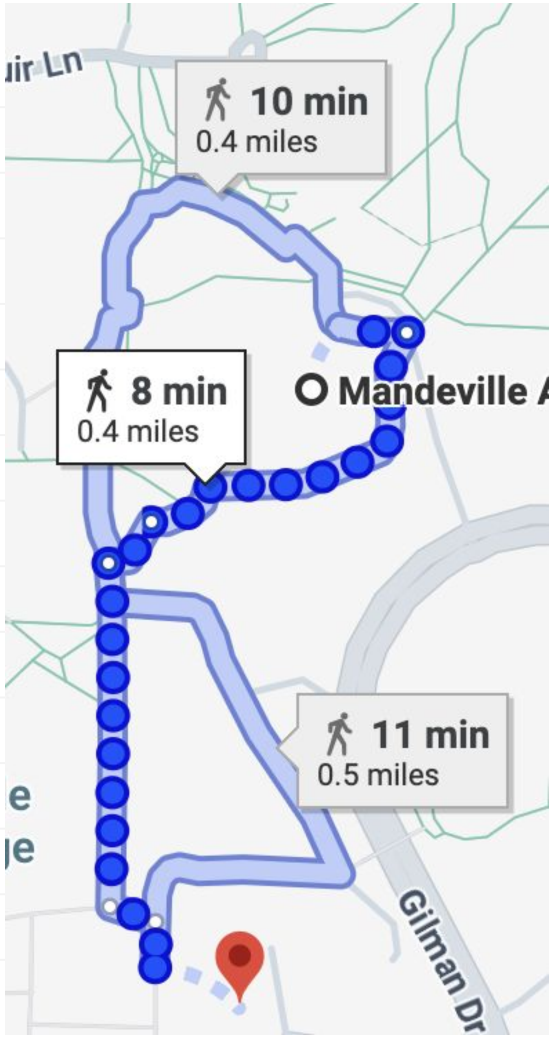
Fun fact: there was a student who took 6 (all different!) classes with me :)

Credits

- ***Most*** of the materials will be re-used from Justin Eldridge's offerings of this class.

I might be late for class because...

Section	Days	Time	Building & Room	
a Sci (4 Units)				P
A00	MWF	10:00a-10:50a	PETER	102
A01	M	4:00p-4:50p	WLH	2111
12/12/2025	F	8:00a-10:59a	TBA	TBA
(4 Units)				P
A00	MWF	12:00p-12:50p	MANDE	B-202
A01	M	4:00p-4:50p	MANDE	B-202
12/11/2025	Th	11:30a-2:29p	MANDE	B-202
nce (2 Units)				P
A00	M	11:00a-11:50a	YORK	4080A
A01		TBA		



Class Business

Syllabus

- All course materials, the syllabus, etc., can be found at dsc40b.com
  - 9 Labs, 8 homeworks (Due Monday + Wed)) + "super hw"
  - 2 exams (dates to be determined). Week 5 and 9.
  - Handwritten submissions, late policy, ChatGPT policy
  - One homework dropped, one lab dropped
  - Exam redemption

Participation

- This is one of the changes.
  - I need students to teach, not empty chairs :(
  - **Class Participation**: 2%
  - **Discussions**: 1%

https://webclicker.web.app/

ZNSOLY

Steps:

1. Go to a link above
2. Code: ZNSOLY
3. Make sure to use your UCSD email address (i.e., @ucsd.edu)
4. Use quest/public wifi please.
5. Answer the questions when I active the poll.
6. Do not worry if it does not work today. The first class does not count. We will figure it out eventually.

https://webclicker.web.app/ ZNSOLY

What is your DSC30 status? :)

A: Already took it

B: Took a similar class

C: Taking it this quarter

D: Still need to take it

E: Something else

Discussion on Monday?

Yes!


SOOOOO...
ANY QUESTIONS?

Let's jump back to DSC 40A...
Just for a bit

Big picture

- In what ways can we **define and represent** the process of **learning from data**?
    - **Learning from data**:
        - observing examples (like pictures of cats and dogs with labels, or past stock prices) and
        - figuring out a pattern or model that can make predictions about new, unseen examples.

Two questions

- In what ways can we **define and represent** the process of **learning** from data?
- How can we **translate** that representation into procedures a **computer** can execute?

Example 1: Minimize Absolute Error

- **Goal**: summarize a collection of numbers, x_1 , … , x_n :
- **Idea**: find number M minimizing the total absolute error:

$$\sum_{i=1}^{n} |M - x_i|$$

Example 1: Minimize Absolute Error

- **Goal**: summarize a collection of numbers, $x_1, \ldots, x_n$:
- **Idea**: find number M minimizing the total absolute error:

$$\sum_{i=1}^{n} |M - x_i|$$

What is M?

A: Range

B: Mean

C: Standard deviation

D: Median

Example 1: Minimize Absolute Error

Solution: The **median** of $x_1, \ldots, x_n$.

The End for 40A

The End?

Minimize Absolute Error

- How do we actually **compute** the median?

Minimize Absolute Error

- How do we actually **compute** the median?

Using **just** Python, no extra libraries. Imagine DSC20 Final :)

Please, talk to each other.

Minimize Absolute Error

- How do we actually **compute** the median?

1) Sort
2) Find the middle

Minimize Absolute Error

- How do we actually **compute** the median?

1) **Sort**
2) Find the middle

Minimize Absolute Error

- How do we actually **compute** the median?

1) **Sort**
2) Find the middle

Time complexity?

A: n

B: n log n

C: n^2

D: Did not take DSC30 yet

Minimize Absolute Error

- How do we actually **compute** the median?

1) **Sort**
2) Find the middle

Is this the best (fastest) you can do?

Time complexity?

A: n

B: n log n

C: n^2

D: Did not take DSC30 yet

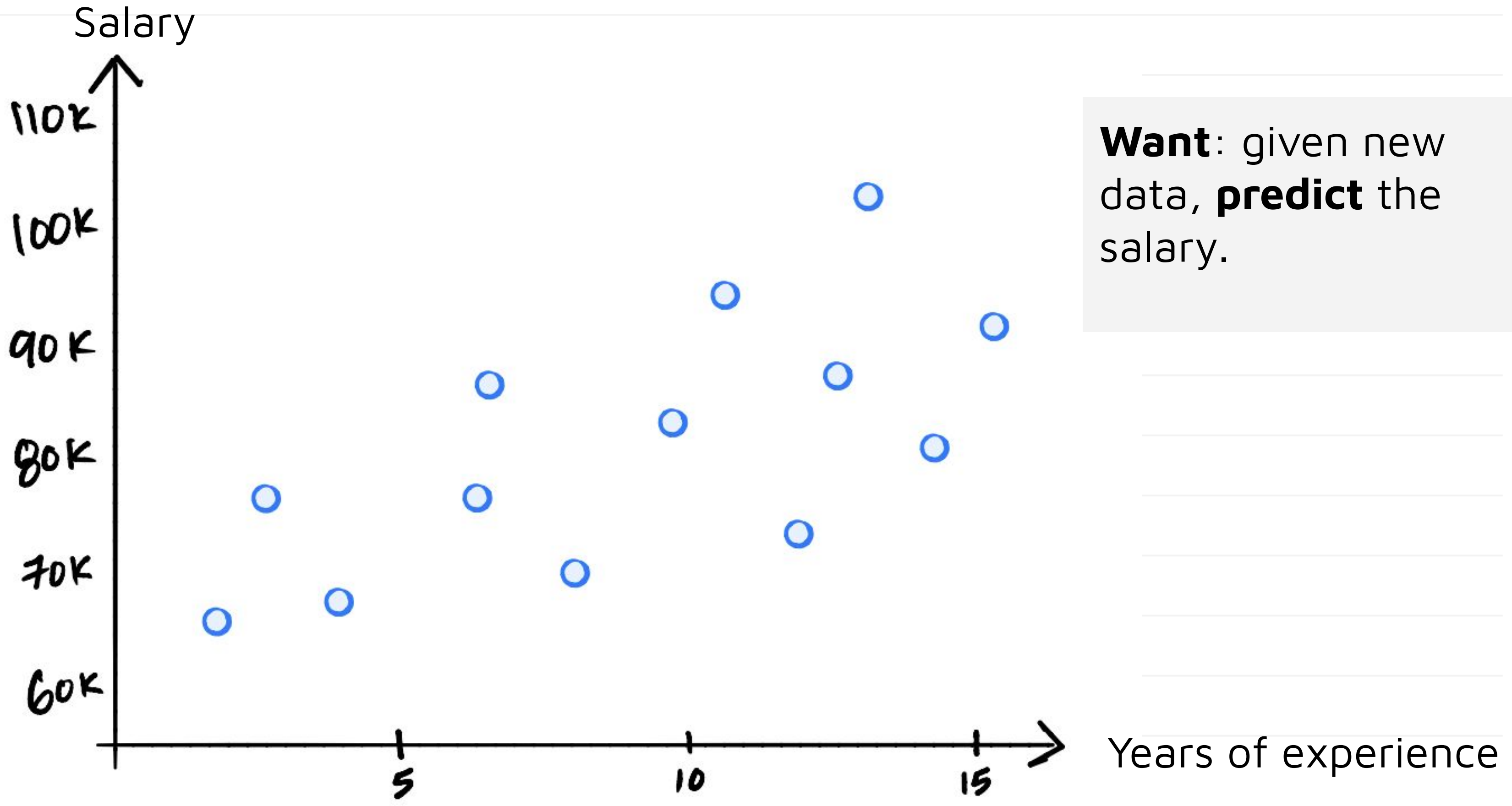
Key idea

- In this class, our work doesn't stop once we solve the math problem (like you did in DSC 40A).
- We still need to **compute** the answer.
- We need an **algorithm**.

Key idea

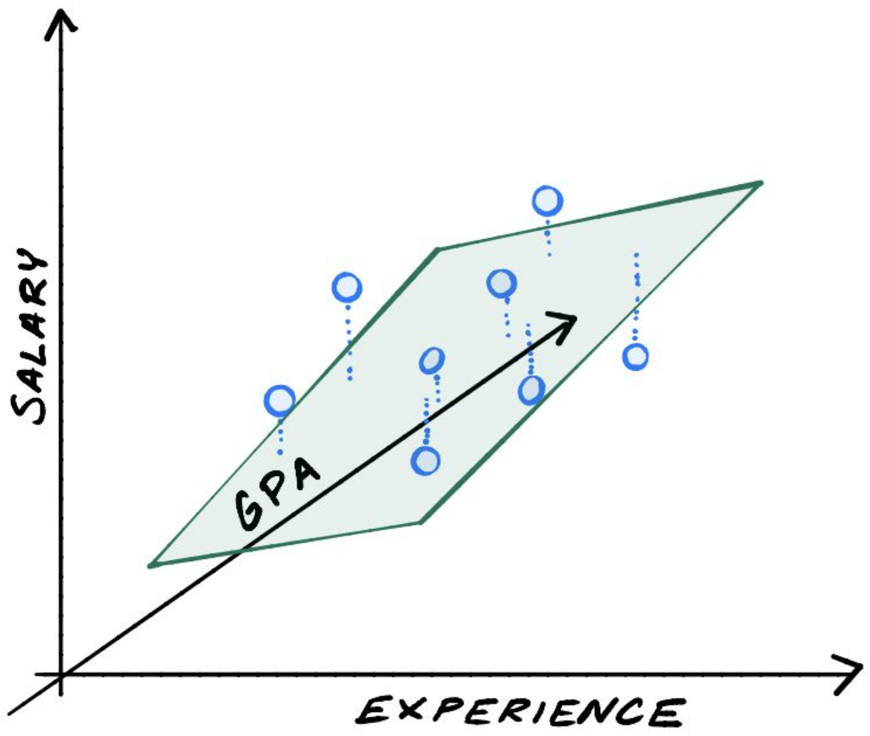
- In this class, our work doesn't stop once we solve the math problem (like you did in DSC 40A).
- We still need to **compute** the answer.
- We need an **algorithm**.
- More than that, we need an **implementation** of that algorithm (that is: **code**).

Example 2: Least Squares Regression



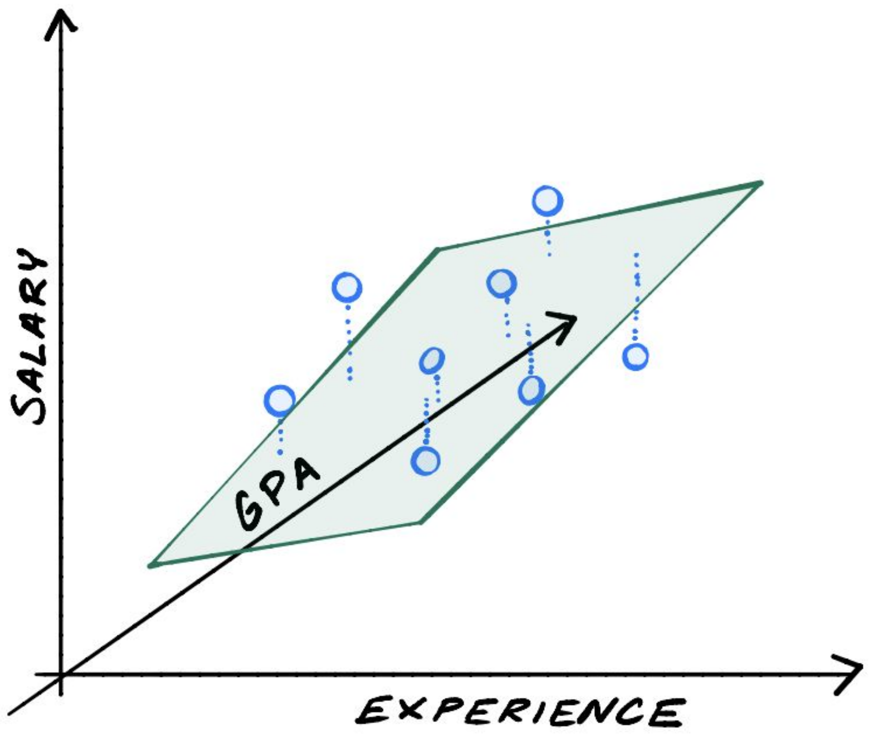
Want: given new data, **predict** the salary.

Example 2: Least Squares Regression



- Formulation (**linear regression**):
  - Find the best (hyper) plane *fitting* these points with **least total error** (sum-square distances).

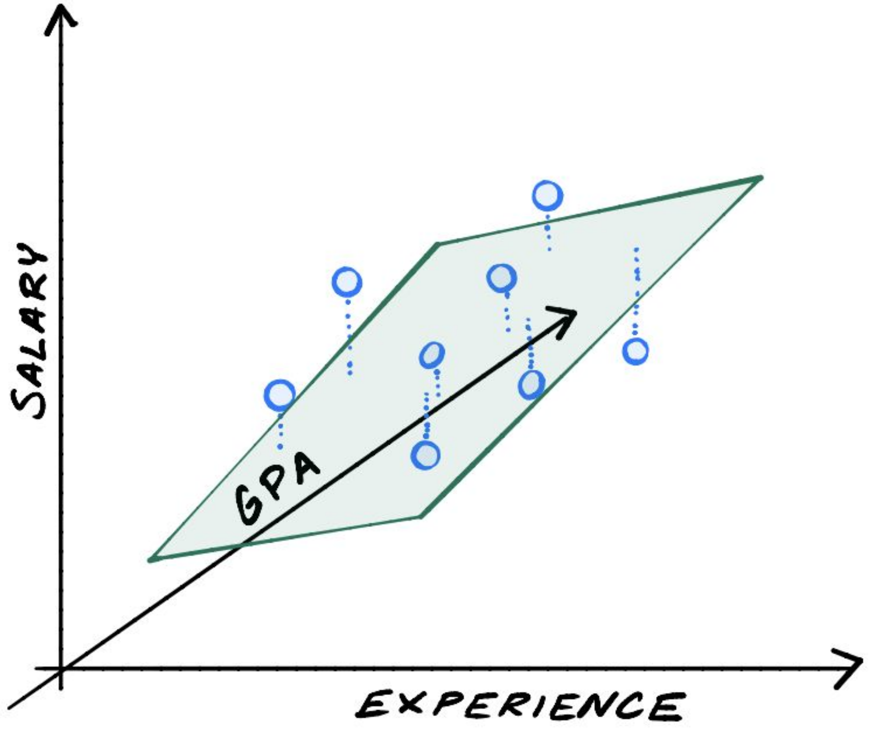
Example 2: Least Squares Regression



- Formulation (**linear regression**):
  - Find the best (hyper) plane *fitting* these points with **least total error** (sum-square distances)
- **Answer**:

$$(X^T X)\vec{w} = X^T \vec{b}$$

Example 2: Least Squares Regression



- Formulation (**linear regression**):
  - Find the best (hyper) plane *fitting* these points with **least total error** (sum-square distances)
- **Answer**:

$$(X^T X)\vec{w} = X^T \vec{b}$$

- **The END**

Wait...

- How do we **really** compute it?
- How do we ask a computer to **compute it for us**?
- We need an **algorithm**.

An Algorithm?

- Let's say we have `numpy` installed.
- It provides an implementation of an algorithm:
  - *Solves normal equations and does regression.*

```
>>> import numpy as np
>>> w = np.linalg.solve(X.T @ X, X.T @ b)
```

But ...is it even good?

- How fast does it run on 1,000 points?
- How does it scale on 1,000,000 points?
- What if the feature dimension increase to 100,000?
- Can we come up with the ***better*** algorithm for this problem?

Key idea

- Having an algorithm **isn't enough** – we need to know about its performance.
- Otherwise, it may be **useless** for our particular problem.

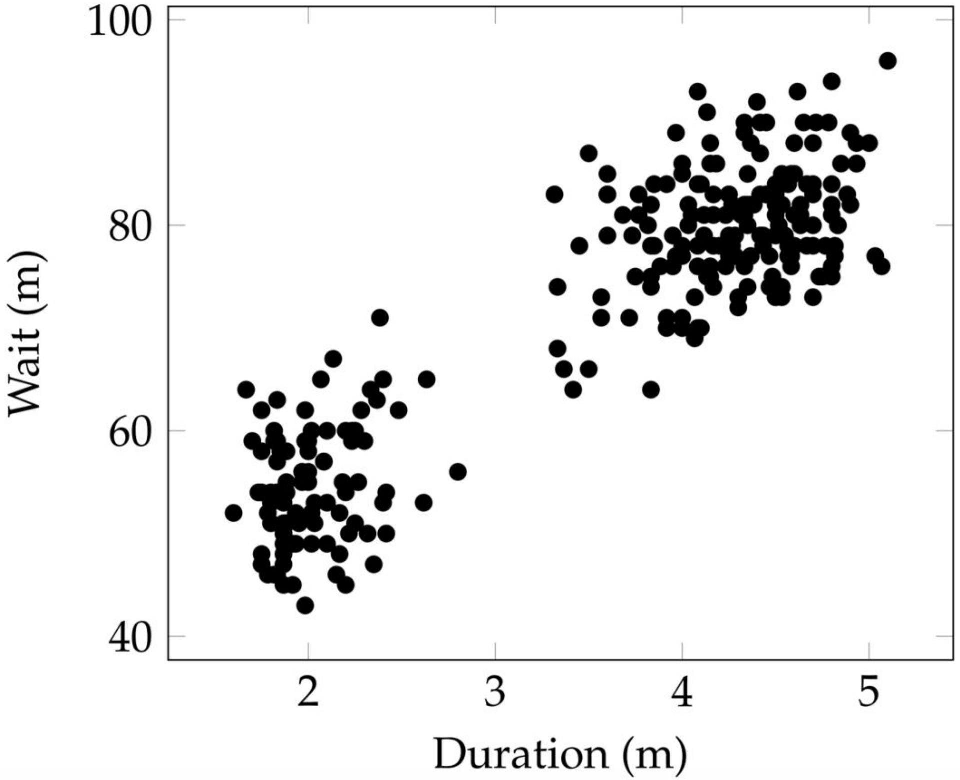
Not convinced? Another example: Clustering

- Given a pile of data, discover *similar* groups.
- **Examples**:
  - Find political groups within social network data.
  - Given data on COVID-19 symptoms, discover groups that are affected differently.
  - Find the similar regions of an image (segmentation).
- Most useful when data is high dimensional…

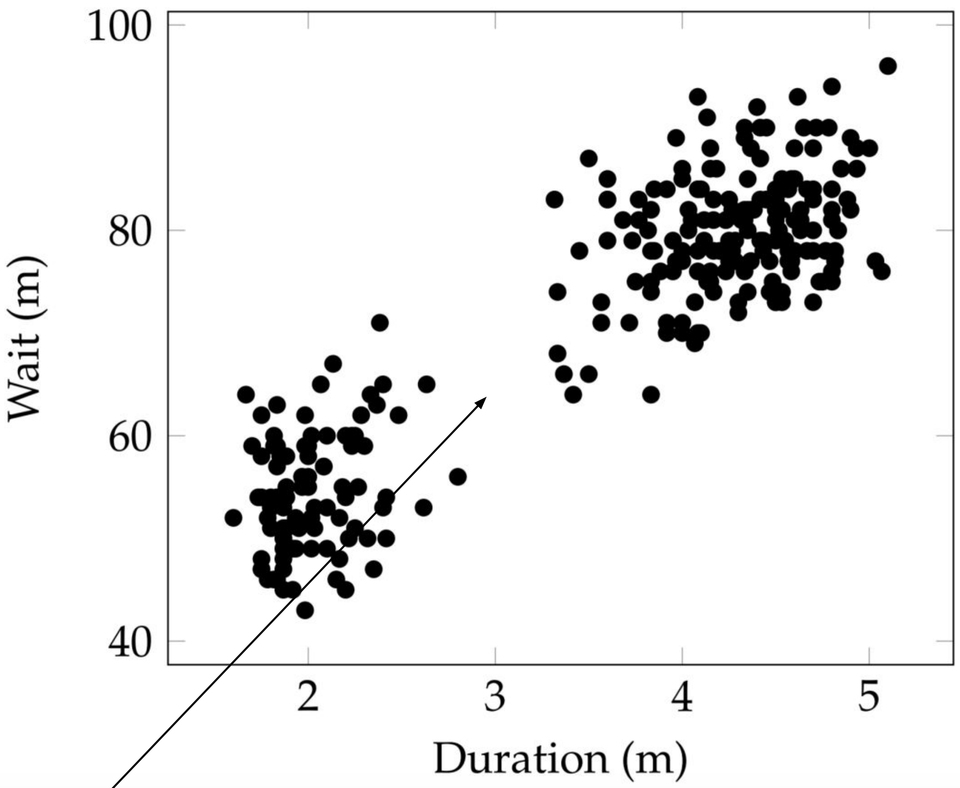
Example: Old Faithful geyser in Yellowstone

Example: Old Faithful in action

What is the pattern behind its eruption?



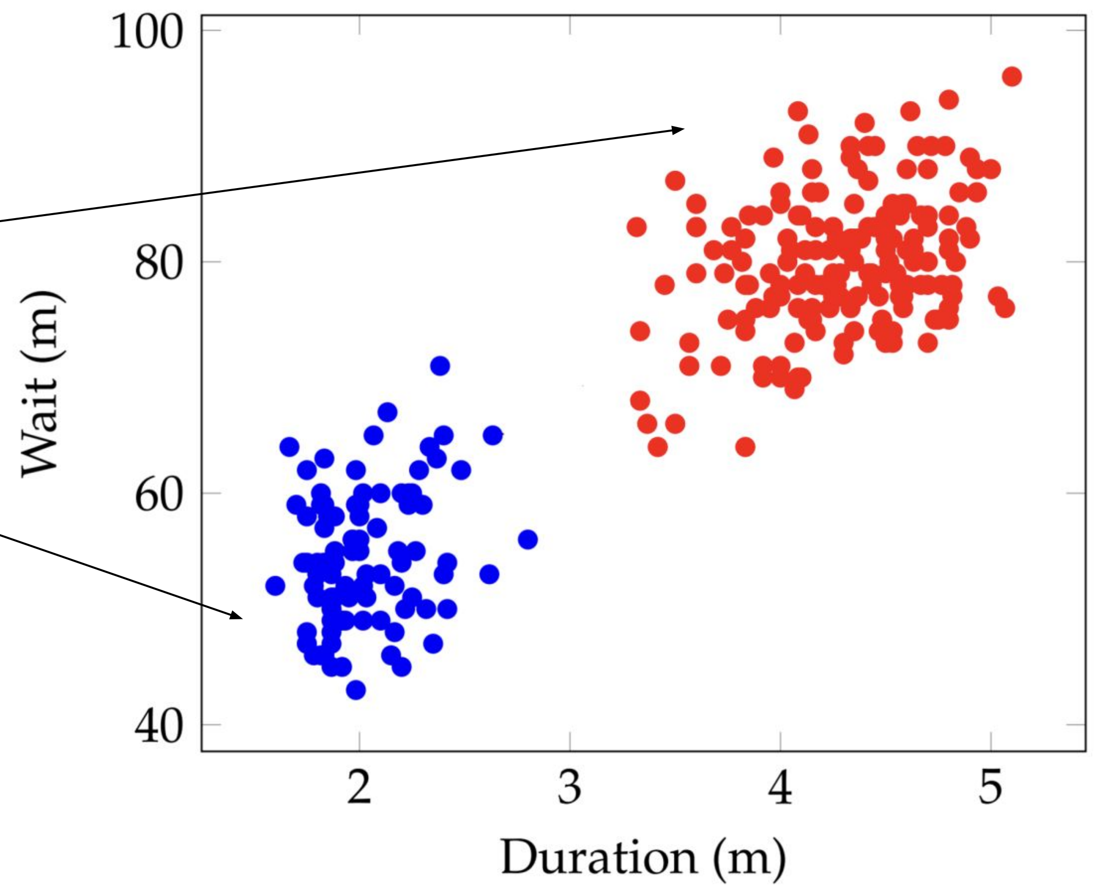
What is the pattern behind its eruption?



Interesting observation for geophysicists

Example: Old Faithful

Goal: Invent an algorithm that finds these clusters automatically.



Clustering

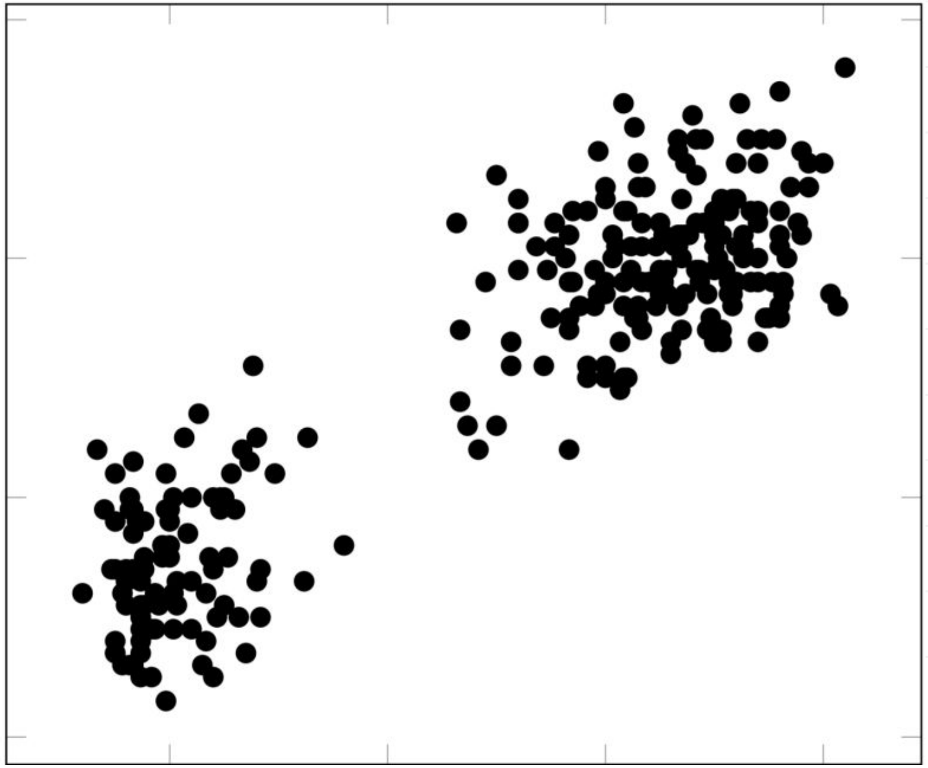
- **Goal**: for computer to identify the two groups in the data.
- **A clustering** is an assignment of a color to each data point.
- There are **many** possible clusterings.

Clustering

- How do we turn this into something a **computer** can do?
- DSC 40A says: “Turn it into an optimization problem”.
- **Idea**: **design** a way of quantifying the “goodness” of a clustering; find the **best**.
  - Design a **loss function**.
  - There are many possibilities, tradeoffs!

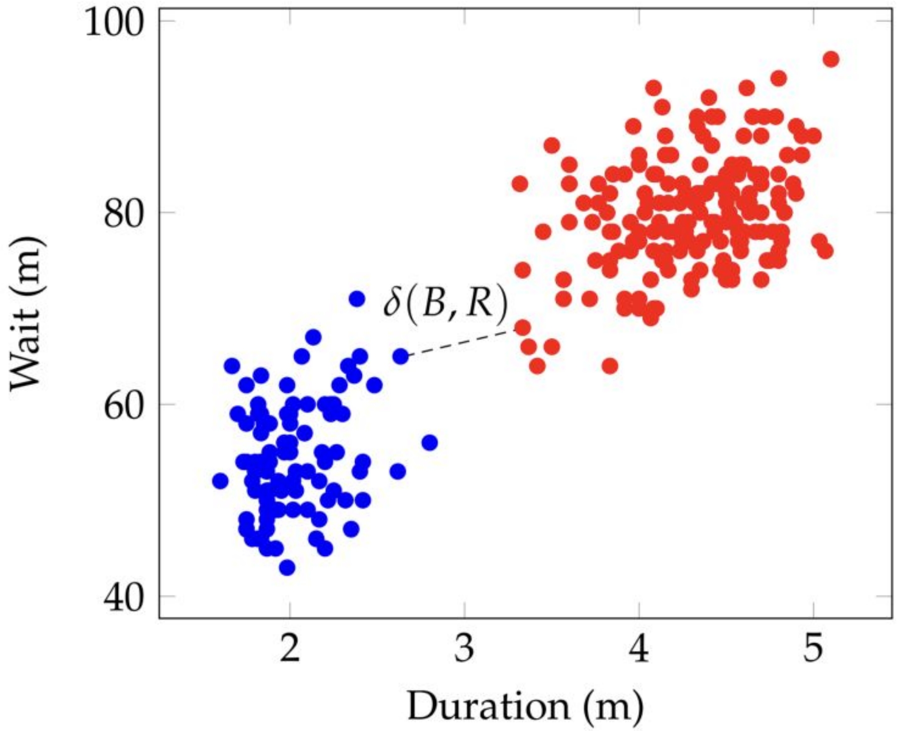
Think!

What's a **good** loss function for this problem? It should assign small loss to a **good** clustering.



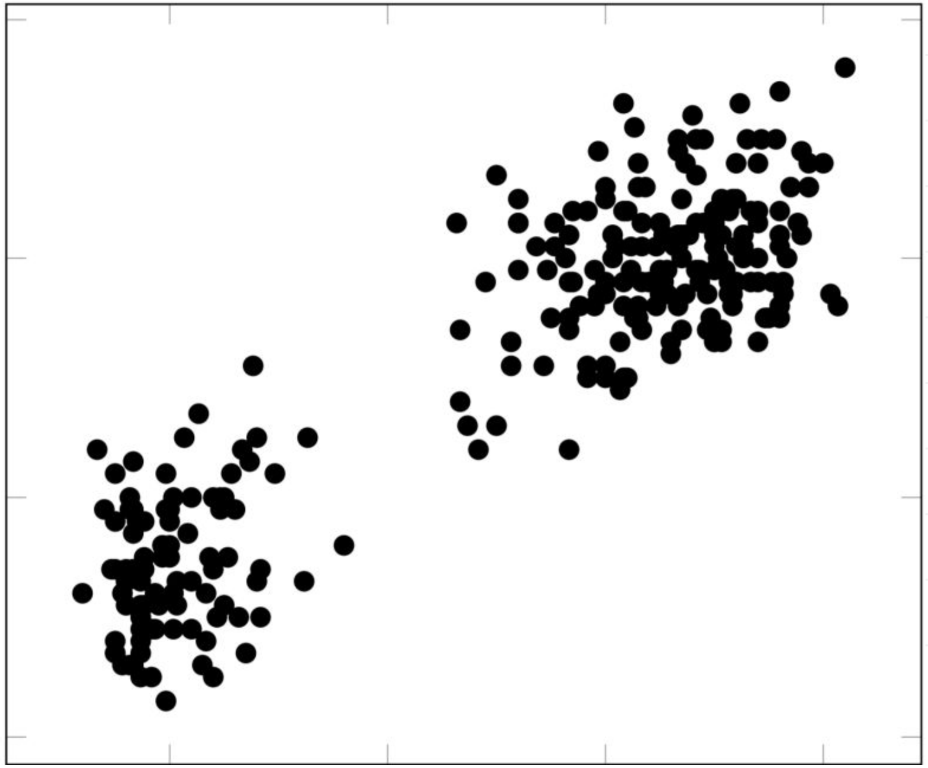
Quantifying Separation

Idea: Define the "separation" $\delta(B, R)$ to be the **smallest** distance between a blue point and red point.



Quantifying Separation

Idea: Define the "separation" $\delta(B, R)$ to be the **smallest** distance between a blue point and red point.



The Problem

- Given n points $\vec{x}^{(1)}, \ldots, \vec{x}^{(n)}$.

- **Find**: an assignment of points to clusters **R** and **B** so as to **maximize** $\delta(B, R)$.

DSC 40A: “The End”

DSC 40A: “The End”

DSC 40B: “The Beginning”

The "Brute Force" Algorithm

- There are finitely-many possible clusterings.
- **Algorithm**: Try **each possible** clustering, return that with largest separation, $\delta(\boldsymbol{B}, \boldsymbol{R})$.
- This is called a brute force algorithm.

Code

```
best_separation = -float('inf') # Python for "infinity"
best_clustering = None

for clustering in all_clusterings(data):

    sep = calculate_separation(clustering)

    if sep > best_separation:

        best_separation = sep

        best_clustering = clustering

print(best_clustering)
```

The Algorithm

- We have an algorithm!
- But how long will this take to run if there are n points?
- How many clusterings of n things are there?

Exercise

How many ways are there of assigning **R** or **B** to n points?

Solution

- **Two** choices for each object: 2 x 2 x … x 2 = 2^n
    - Small nitpick: actual color doesn't matter, 2^{n-1}

Time

- Suppose it takes at least 1 *nanosecond* to check a single clustering.
  - One *billionth* of a second.
  - Time it takes for light to travel 1 foot.
- If there are n points, it will take at *least* 2^n nanoseconds to check all clusterings.
- This is an *extremely* optimistic estimate. It's actually much slower, and scales with n.

Time Needed

n	Time
1	1 nanosecond

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond

1 millionth of a second

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond

1 thousandth of a second

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second
40	18 minutes

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second
40	18 minutes
50	

Check your intuition:

A: In minutes

B: In hours

C: In days

D: In weeks

E: In years

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second
40	18 minutes
50	13 days

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second
40	18 minutes
50	13 days
60	36 years

Time Needed

n	Time
1	1 nanosecond
10	1 microsecond
20	1 millisecond
30	1 second
40	18 minutes
50	13 days
60	36 years
70	37,000 years

Example: Old Faithful

- The Old Faithful data set has **270** points.
- Brute force algorithm will finish in **6×10^{64} years.**

Algorithm Design

- Often, most obvious algorithm is **unusably slow.**

Algorithm Design

- Often, most obvious algorithm is **unusably slow**.
- Does this mean our problem is too hard?
    - Direct result of our choice of loss function.

Algorithm Design

- Often, most obvious algorithm is **unusably slow**.
- Does this mean our problem is too hard?
  - Direct result of our choice of loss function.
- We'll see an efficient solution by the end of the quarter.

Main Idea

- Just having an algorithm isn’t enough – it must also be reasonably **efficient**. Otherwise, it might be **useless** for our particular problem.

DSC 40B is about:

- Assess the efficiency of algorithms.
- Understand why and how common algorithms work.
- Develop faster algorithms using design strategies and data structures.

Measuring Efficiency by Timing

Efficiency

- Speed matters, especially with large data sets.
- An algorithm is only useful if it runs **fast enough.**
  - That depends on the size of your data set.
- How do we measure the efficiency of code?
- How do we know if a method will be fast enough?

Scenario

- You’re building a least squares regression model to predict a patient’s blood oxygen level.
- You’ve trained it on 1,000 people.
- You have a full data set of 100,000 people.
- How long will it take? How does it **scale**?

Example: Scaling

- Your code takes 5 seconds on 1,000 points.
- How long will it take on 100,000 data points?
- 5 seconds × 100 = 500 seconds?
- More? Less?

Coming Up

- We'll answer this in coming lectures.
- Today: start with simpler algorithms for the mean, median.

Approach #1: Timing

- How do we measure the efficiency of code?
- Simple: time it!
- Useful Jupyter tools: `time` and `timeit`
    - Magic functions

[4]:
```
numbers = range(1000)
```

[5]:
```
%%time
sum (numbers)
```

```
CPU times: user 30 µs, sys: 0 ns, total: 30 µs
Wall time: 34.3 µs
```

[5]: 499500

[6]:
```
%%timeit
sum(numbers)
```

```
9.96 µs ± 3.79 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Disadvantages of Timing

1. Time depends on the computer.

Disadvantages of Timing

1. Time depends on the computer.

2. Depends on the particular input, too.

Disadvantages of Timing

1. Time depends on the computer.
2. Depends on the particular input, too.
3. One timing doesn’t tell us how algorithm **scales**.

Thank you!

Do you have any questions?

CampusWire!