Carnegie Mellon University
Heinz College

Clustering Part IV: summary, topic modeling with Latent Dirichlet Allocation (LDA)

George Chen

CMU 95-865 Spring 2018

Going from Similarities to Clusters

There’s a whole zoo of clustering methods

Two main categories we'll talk about:

Generative models

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

Hierarchical clustering

Top-down: Start with everything in 1 cluster and decide on how to recursively split

Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

Going from Similarities to Clusters

Generative models

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

The most popular models effectively assume Euclidean distance…

You learn a model
➔ can predict cluster assignments for points not seen in training

Hierarchical clustering

Top-down: Start with everything in 1 cluster and decide on how to recursively split

Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

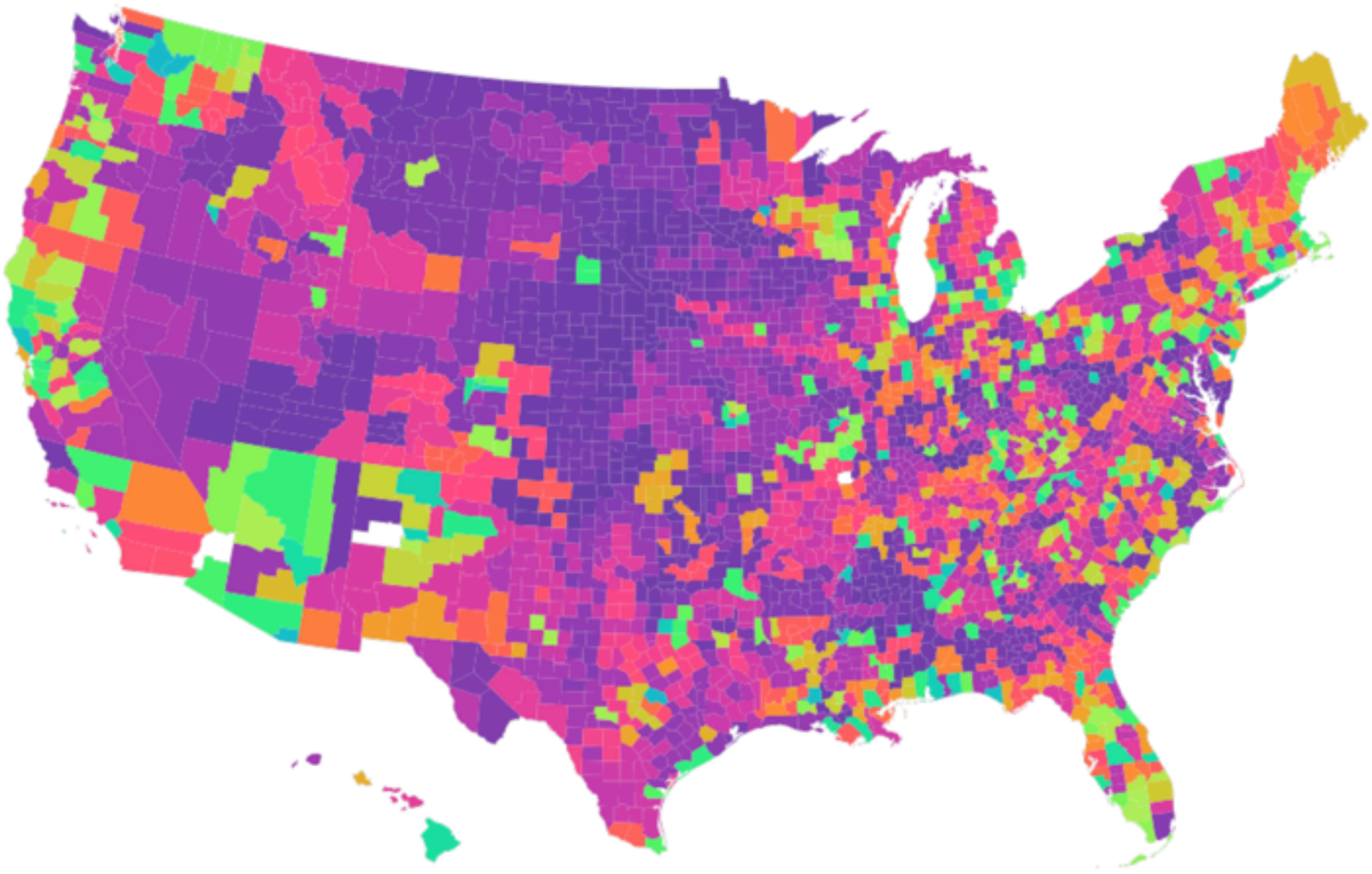
Easily works with different distances (not just Euclidean)

Great for problems that don't need to predict clusters for future points

Different split/merge criteria lead to clusters that look specific ways (e.g., chaining, crowding)

Example: Clustering on U.S. Counties

(using opioid death rate data across 37 years)



No need to predict which cluster new counties should belong to, since we're already looking at all U.S. counties!

Image source: Amanda Coston

Clustering

Generative models

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

Hierarchical clustering

Top-down: Start with everything in 1 cluster and decide on how to recursively split

Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

***Many* more methods we didn't cover**

- `sklearn` has a whole bunch more (*not* close to exhaustive)
- Also: remember the recommendation system setup?
  - **Co-clustering** is the problem of clustering both users and items at the same time (`sklearn` has a few methods)

How to Choose a Clustering Method?

In general: not easy!

Some questions to think about:

- What features to even cluster on?
- Does Euclidean distance make sense for your application, or do you use some custom distance function?
- Do you care about figuring out which cluster new points belong to?
- After you run the clustering algorithm, look at what data points ended up in the same cluster and make visualizations (e.g., histogram of various feature values)
  - Do the clusters seem interpretable to you?
  - Compare the cluster centers: do two clusters seem a bit too close and should be merged?
- Can you come up with some heuristic score function to say how good a cluster assignment is?

Clustering Last Remarks

- It's possible that several clustering methods give similar results (*which is great!* — it means that there are some reasonably "stable" clusters in your data)
  - Example: *tons* of clustering methods can figure out from senate voting data who Democrats and Republicans are (of course, *without* knowing each senator's political party)
- Ultimately, *you* have to decide on which clustering method and number of clusters make sense for your data
  - Do not just blindly rely on numerical metrics (e.g., CH index)
  - Interpret the clustering results in the context of the application you are looking at

If you can set up a prediction task, then you can use the prediction task to guide the clustering

Is Clustering Structure Enough?

Items

1 2 3 4 5 ⋯ *m*

User clusters

1

2

⋮

k

Is Clustering Structure Enough?

Items

1 2 3 4 5 ⋯ *m*

User clusters

1

2

⋮

k

Is Clustering Structure Enough?

Items

1 2 3 4 5 ⋯ *m*

User clusters

1

2

⋮

k

What if these two users shared a Netflix account (and used the same user profile)?

Is Clustering Structure Enough?

Items

1 2 3 4 5 ⋯ *m*

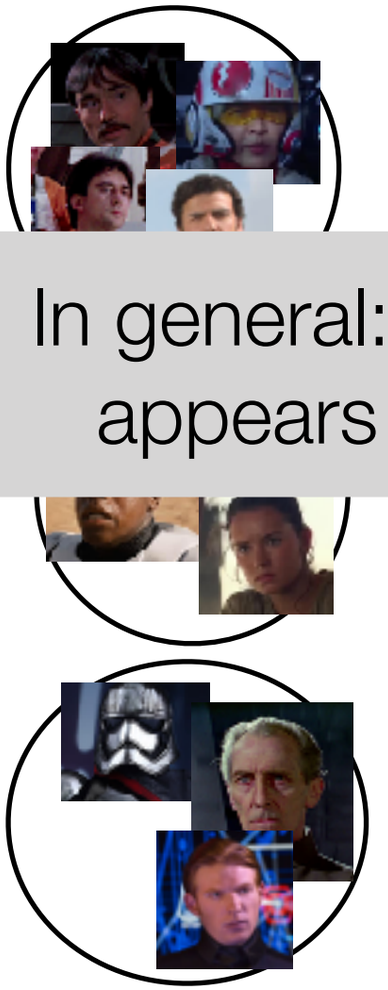
User clusters

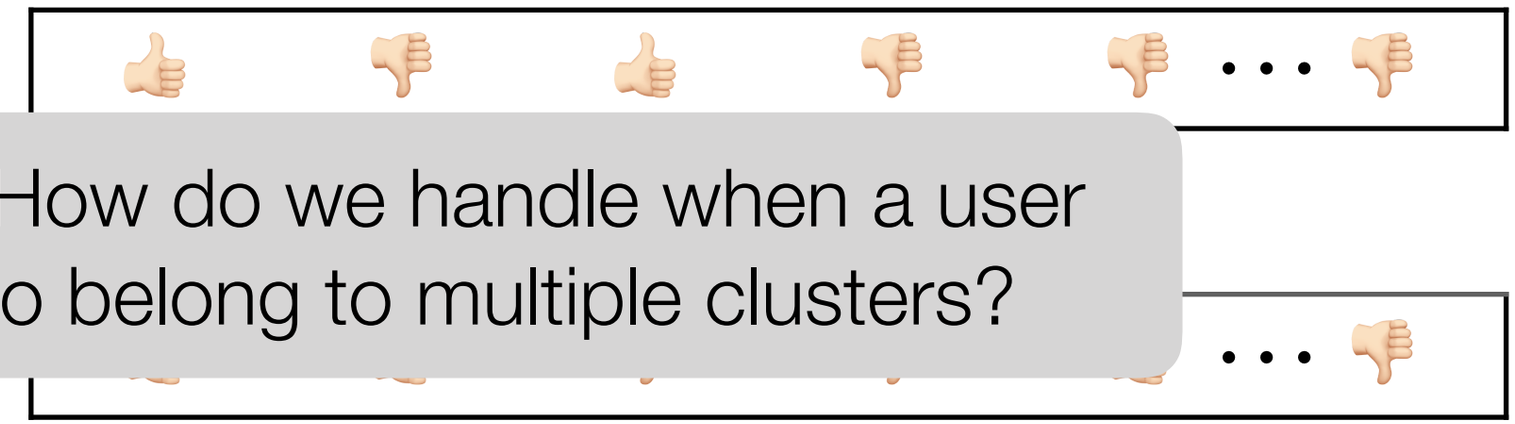
1

2

⋮

k





In general: How do we handle when a user appears to belong to multiple clusters?

What if these two users shared a Netflix account (and used the same user profile)?

Topic Modeling

Movie recommendation

Each user is part of multiple “clusters”/topics

Each cluster/topic consists of a bunch of movies
(example clusters: “sci-fi epics”, “cheesy rom-coms”)

Text

Each document is part of multiple topics

Each topic consists of a bunch of regularly co-occurring words
(example topics: “sports”, “medicine”, “movies”, “finance”)

Health care

Each patient’s health records explained by multiple “topics”

Each topic consists of co-occurring “events”
(example topics: “heart condition”, “severe pancreatitis”)

Topic Modeling

Movie recommendation

Each user is part of multiple “clusters”/topics

Each cluster/topic consists of a bunch of movies
(example clusters: “sci-fi epics”, “cheesy rom-coms”)

Each to words
(exar nce”)

In all of these examples:

- Each data point (a feature vector) is part of multiple topics
- Each topic corresponds to specific feature values in the feature vector likely appearing

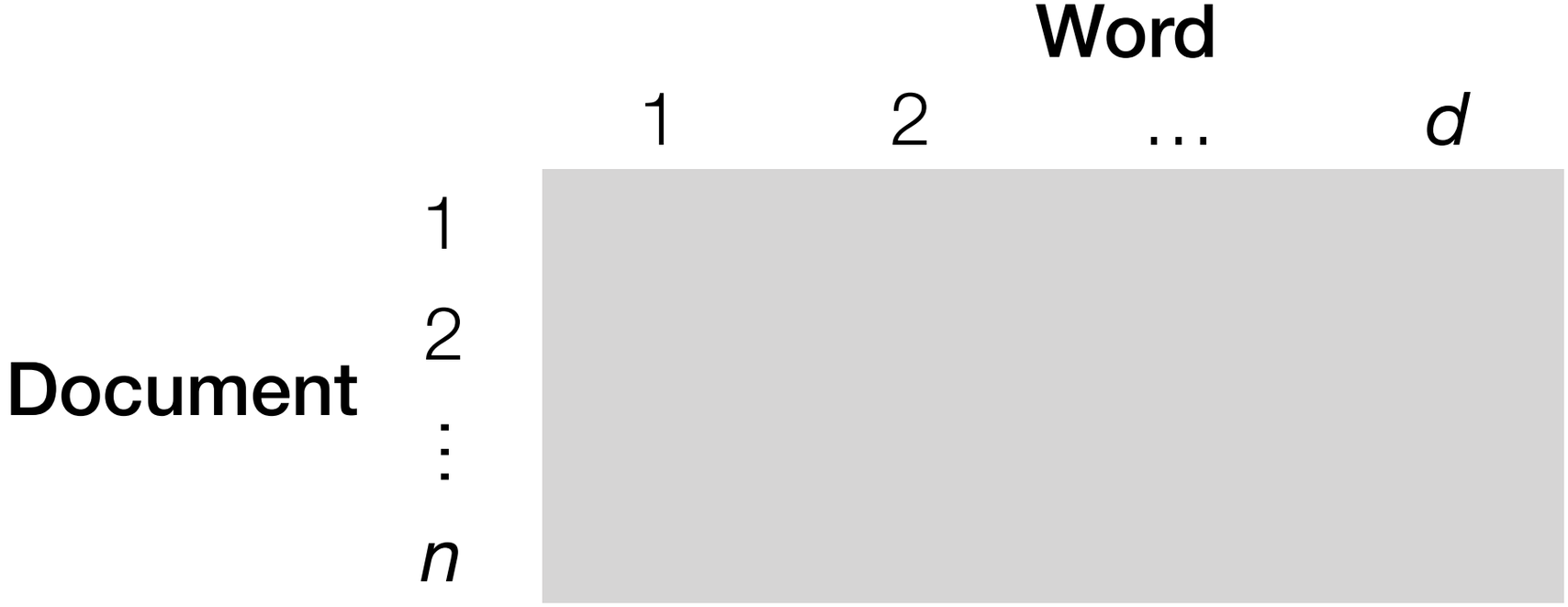
Health care

Each patient’s health records explained by multiple “topics”

Each topic consists of co-occurring “events”
(example topics: “heart condition”, “severe pancreatitis”)

Latent Dirichlet Allocation (LDA)

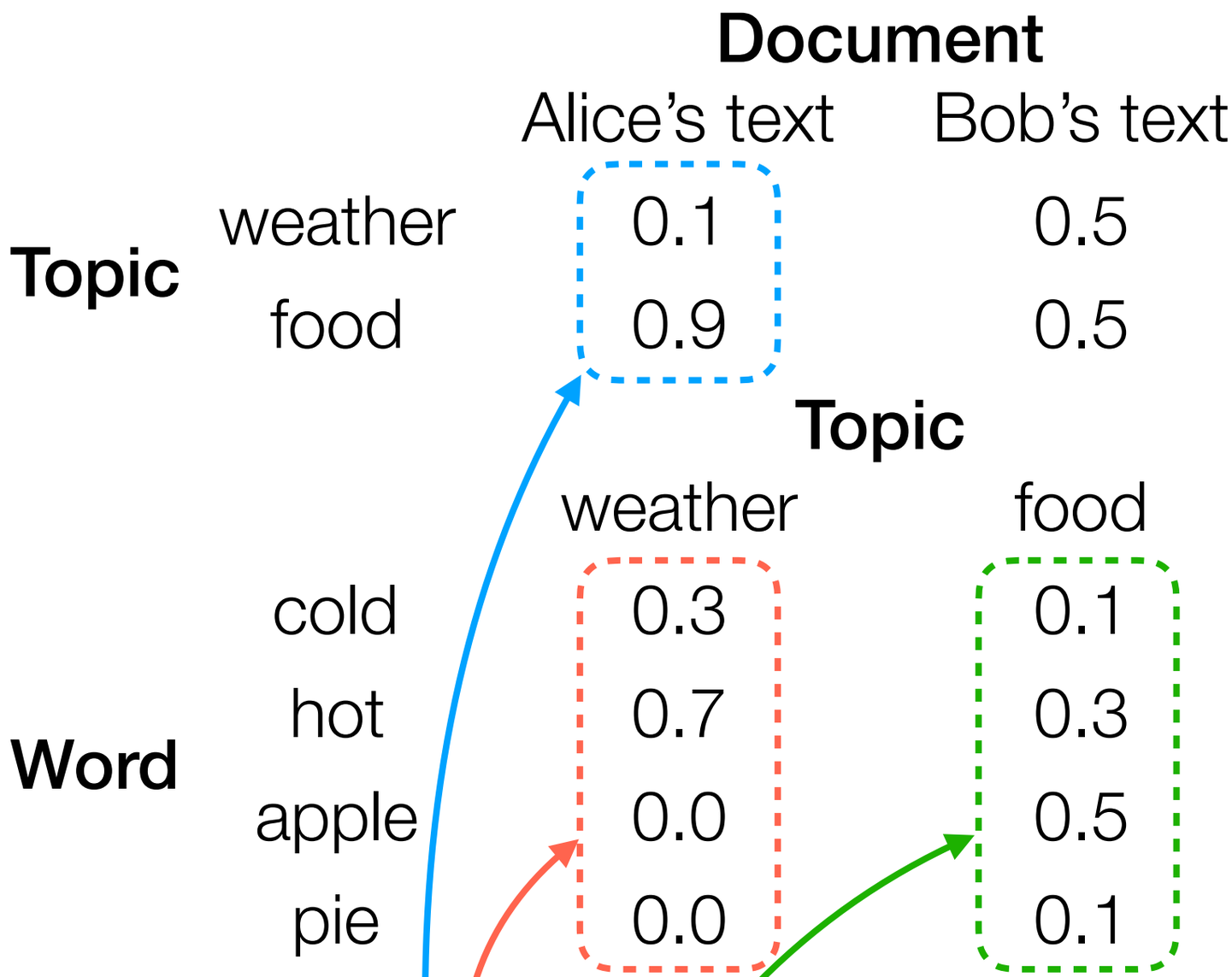
- Easy to describe in terms of text (but works for not just text)
- Input: “document-word” matrix, and pre-specified # topics *k*



i-th row, *j*-th column: # times word *j* appears in doc *i*

- Output: what the *k* topics are (details on this shortly)

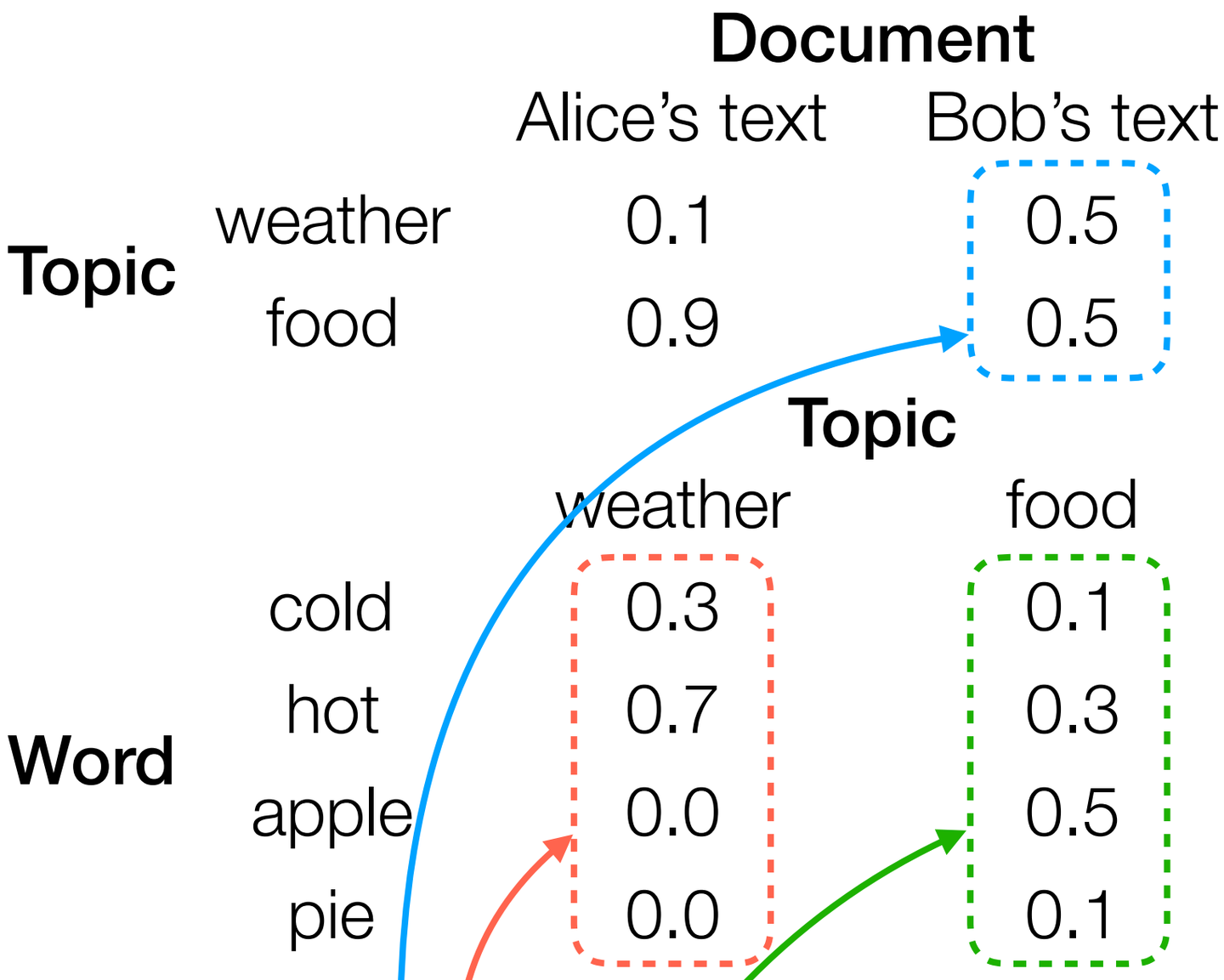
LDA Example



Each word in Alice's text is generated by:

1. Flip 2-sided coin for Alice
2. If weather: flip 4-sided coin for weather
   If food: flip 4-sided coin for food

LDA Example



Each word in Bob's text is generated by:

1. Flip 2-sided coin for Bob
2. If weather: flip 4-sided coin for weather
   If food: flip 4-sided coin for food

LDA Example

Document

Topic	Alice's text	Bob's text
weather	0.1	0.5
food	0.9	0.5

Topic

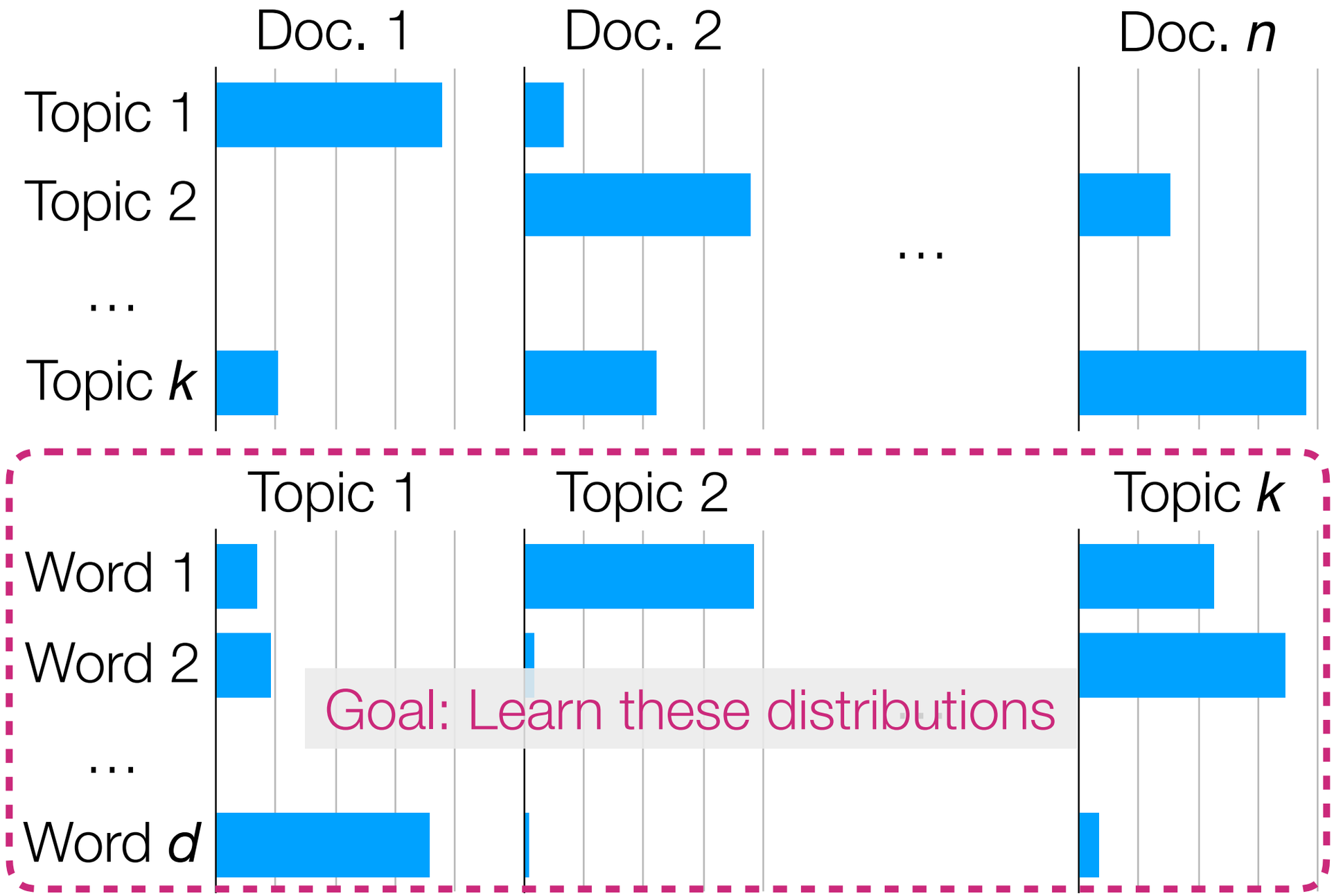
Word	weather	food
cold	0.3	0.1
hot	0.7	0.3
apple	0.0	0.5
pie	0.0	0.1

Each word in doc. *i* is generated by:

1. Flip 2-sided coin for doc. *i*
2. If weather: flip 4-sided coin for weather
   If food: flip 4-sided coin for food

"Learning the topics" means figuring out these 4-sided coin probabilities

LDA

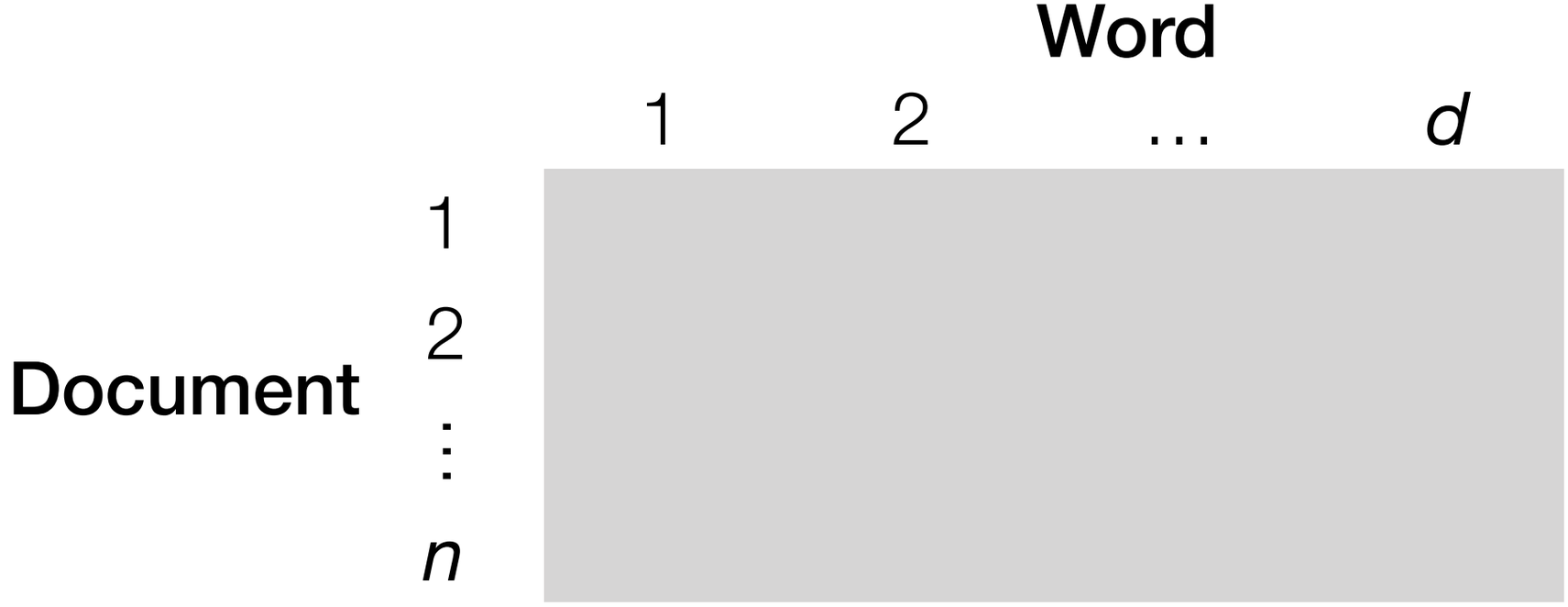


LDA models each word in document *i* to be generated as:

- Randomly choose a topic *Z* (use topic distribution for doc *i*)
- Randomly choose a word (use word distribution for topic *Z*)

LDA

- Easy to describe in terms of text (but works for not just text)
- Input: “document-word” matrix, and pre-specified # topics ***k***



i-th row, *j*-th column: # times word *j* appears in doc *i*

- Output: the ***k*** topics’ distribution of words

LDA

Demo

How to Choose Number of Topics *k*?

Bayesian nonparametric variant of LDA:
Hierarchical Dirichlet Process (HDP)

(similar to how we went from GMM to DP-GMM)

Something like CH index is also possible:

For a specific topic, look at the *m* most probable words ("top words")

Coherence (within cluster/topic variability):

$$\sum_{\substack{\text{top words } v,w \\ \text{that are not the same}}} \log \frac{(\#\text{ documents with at least one appearance of } v \text{ and } w) + \varepsilon}{\#\text{ documents with at least one appearance of } w}$$

choose something small like 0.01

Inter-topic similarity (between cluster/topic variability):

Count # top words that do not appear in any of the other topics' *m* top words

(number of "unique words")

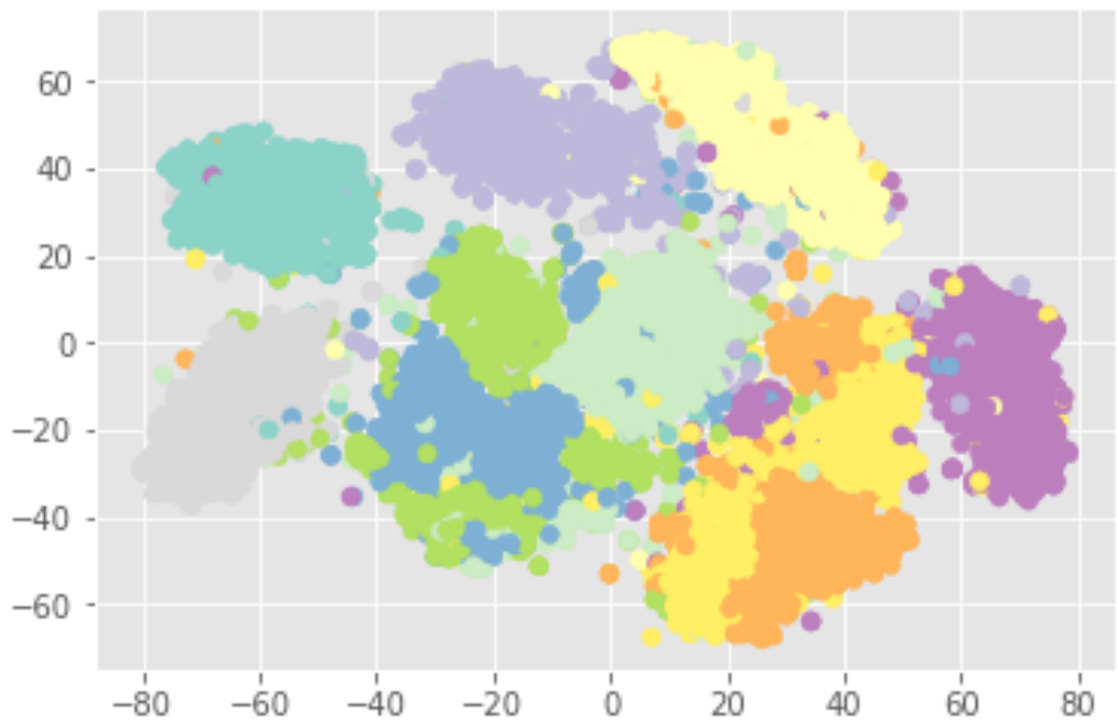
Can average each of these across the topics

Topic Modeling

- There are actually *many* topic models, not just LDA & HDP
  - Correlated topic models, Pachinko allocation, biterm topic models, anchor word topic models, …

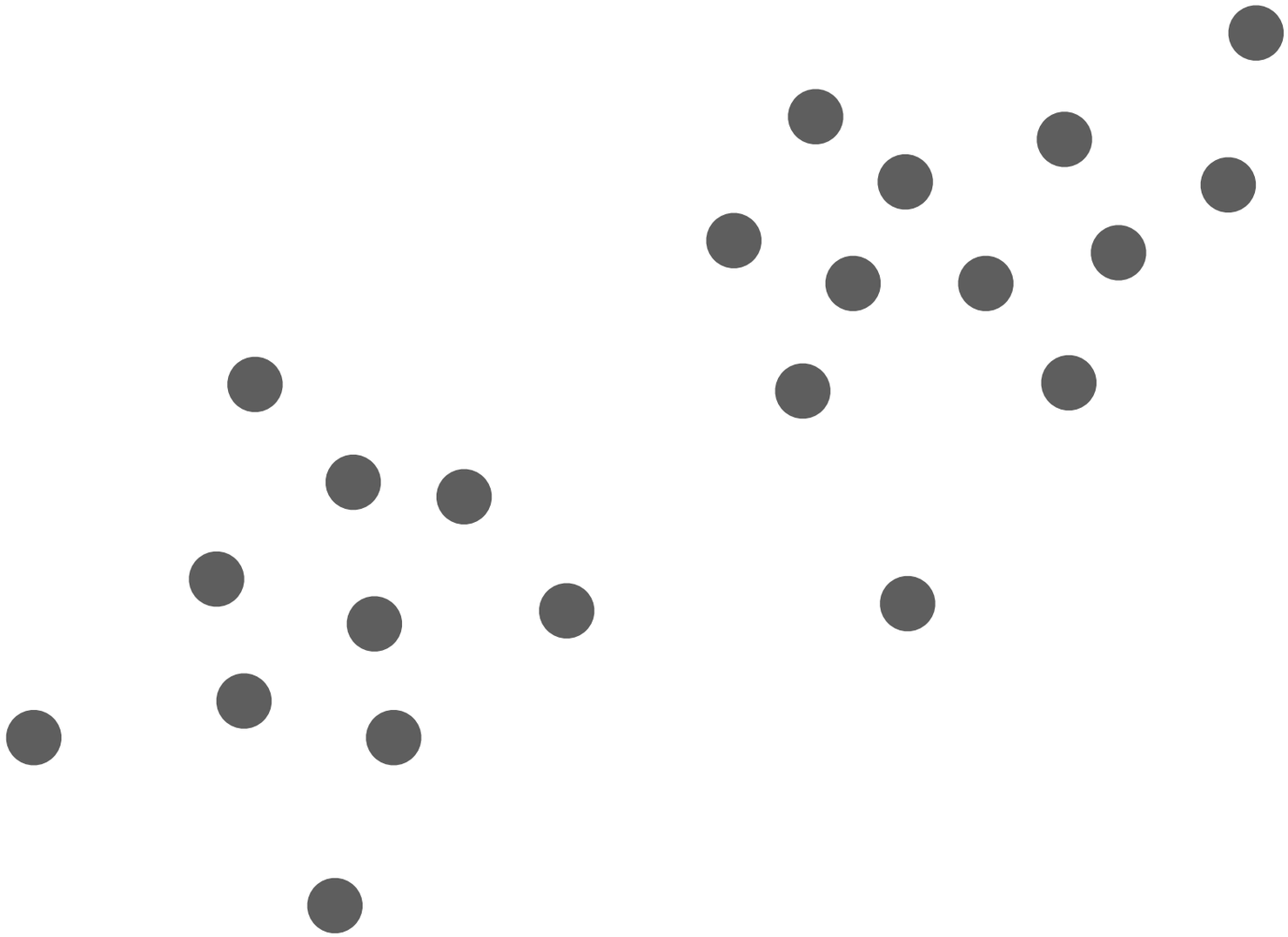
- Dynamic topic models: tracks how topics change *over time*
  - This sort of idea could be used to figure out how user tastes change over time in a recommendation system
    - Could try to see if there are existing patterns for how certain topics become really popular

What if we have labels?



Example: MNIST handwritten digits have known labels

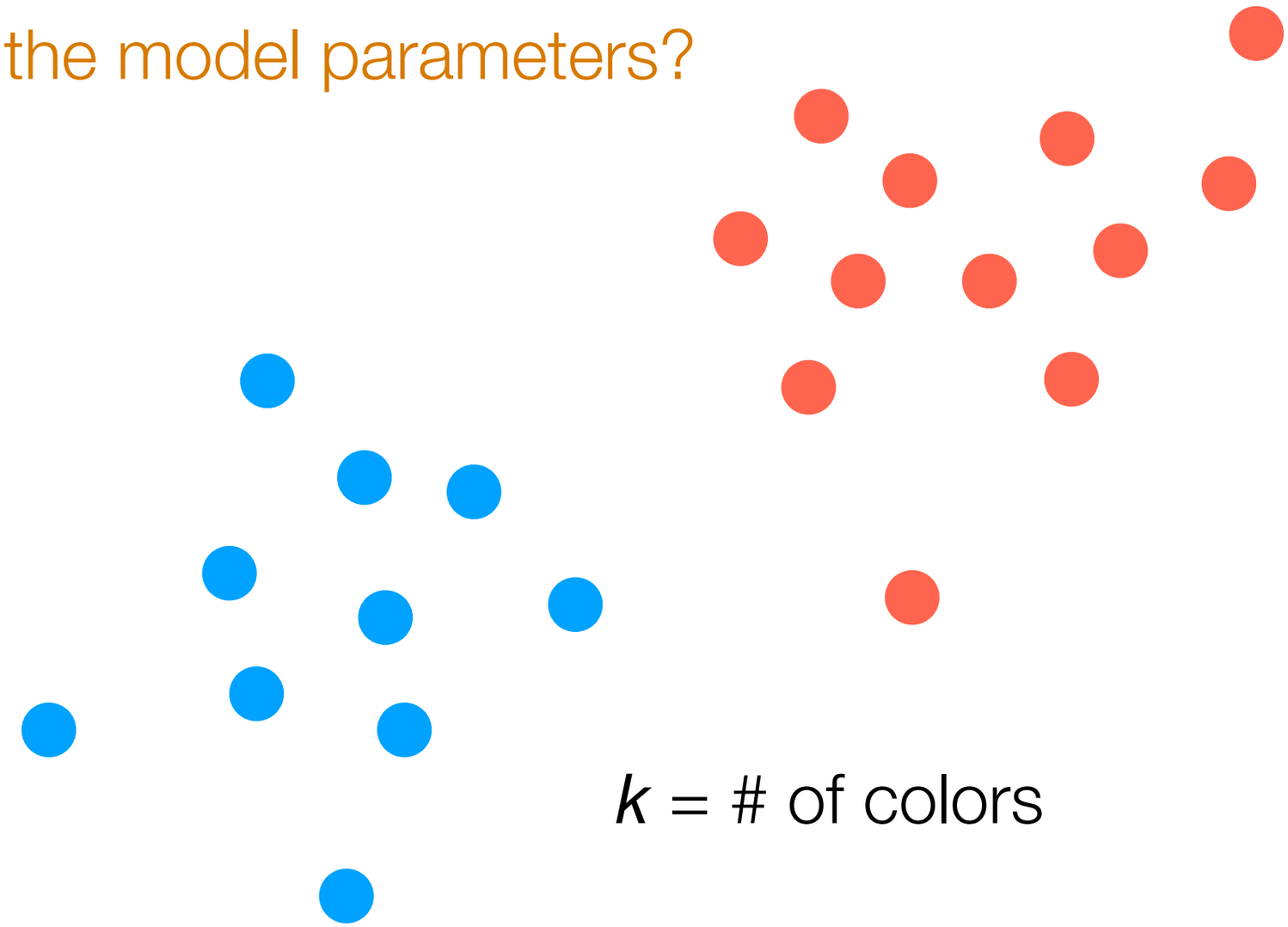
If the labels are known…



If the labels are known…

And we assume data generated by GMM…

What are the model parameters?



k = # of colors

We can directly estimate cluster means, covariances

Flashback: Learning a GMM

Don’t need this top part if we know the labels!

Step 0: Pick *k*

Step 1: Pick guesses for **cluster means and covariances**

Repeat until convergence:

Step 2: Compute probability of each point belonging to each of the *k* clusters

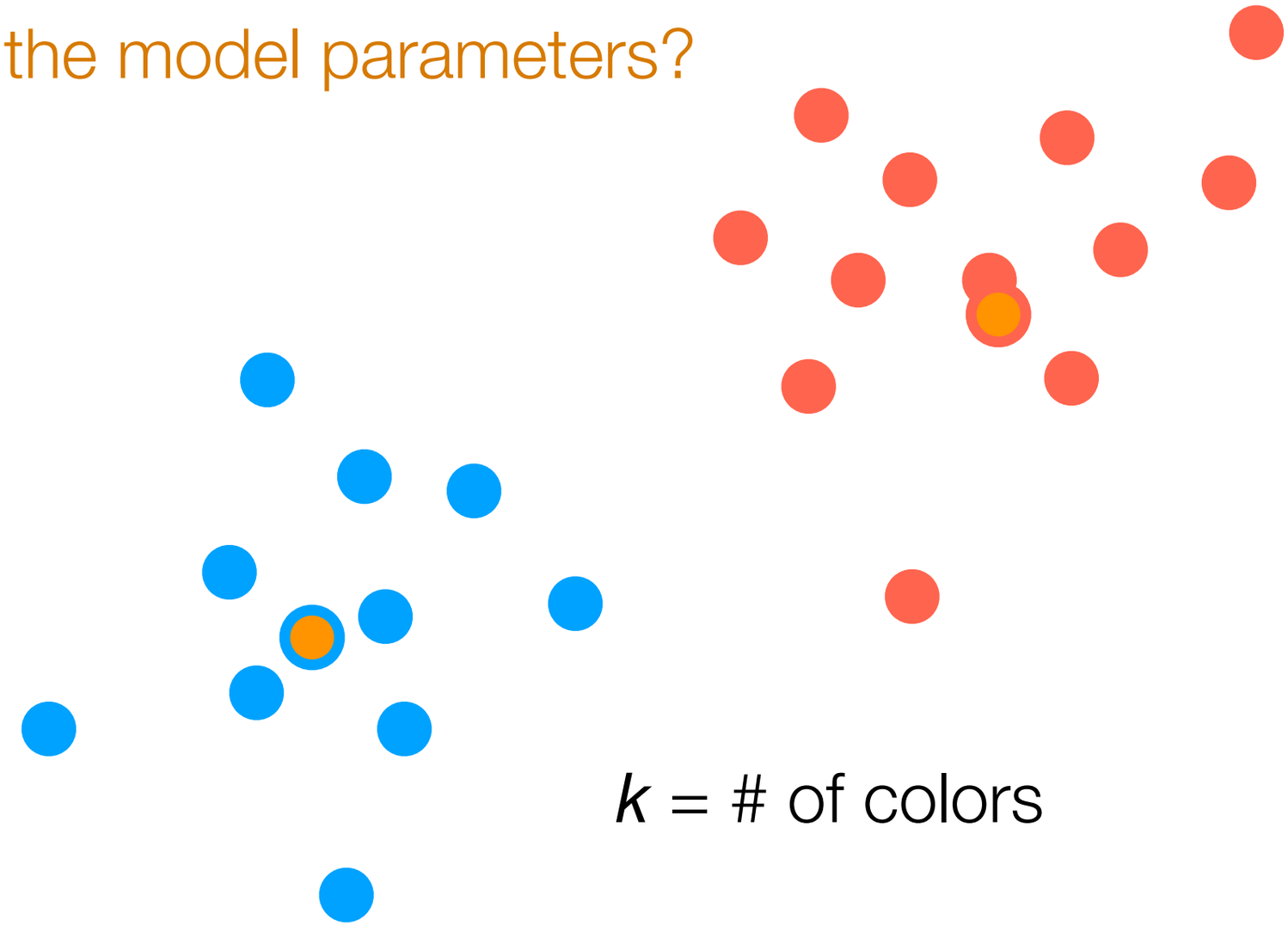
Step 3: Update **cluster means and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

We don’t need to repeat until convergence

If the labels are known…

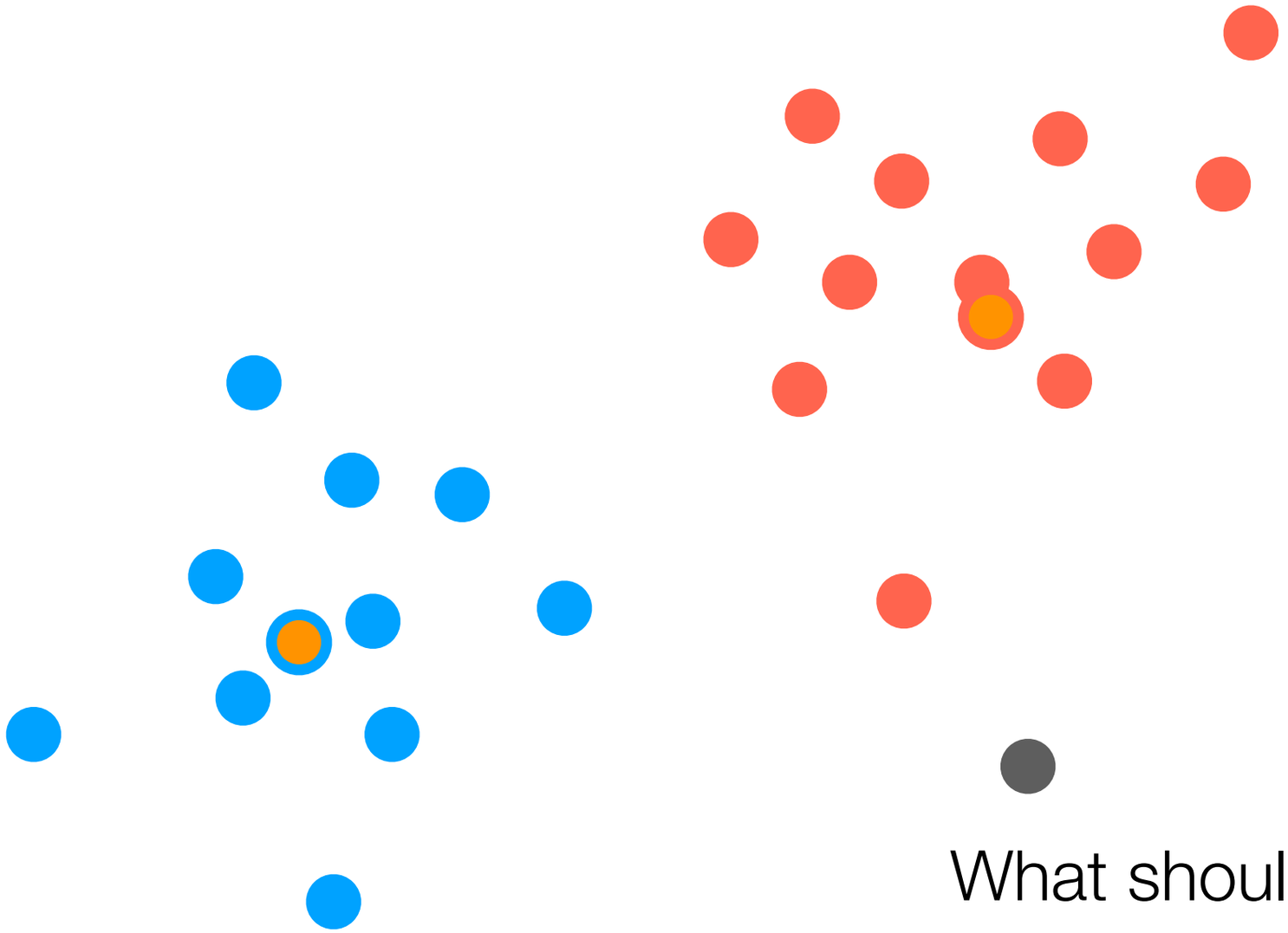
And we assume data generated by GMM…

What are the model parameters?



k = # of colors

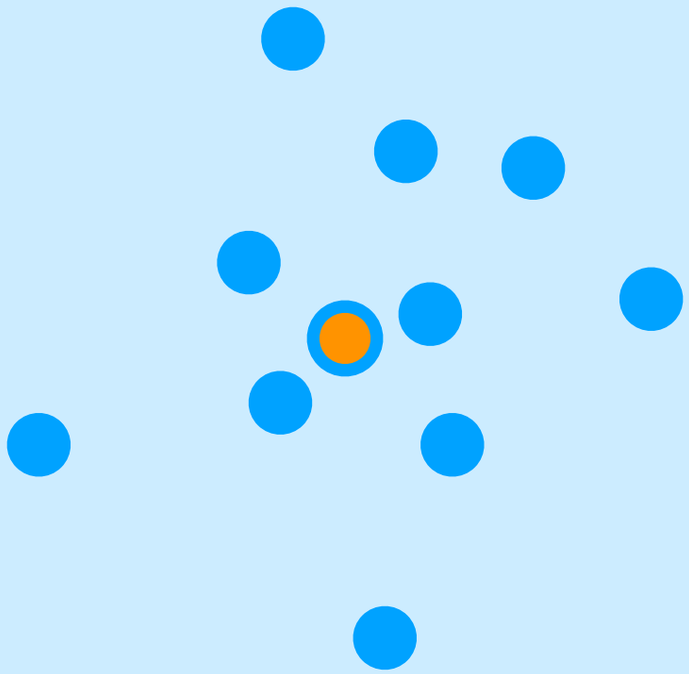
We can directly estimate cluster means, covariances



What should the label of this new point be?

Whichever cluster has higher probability!

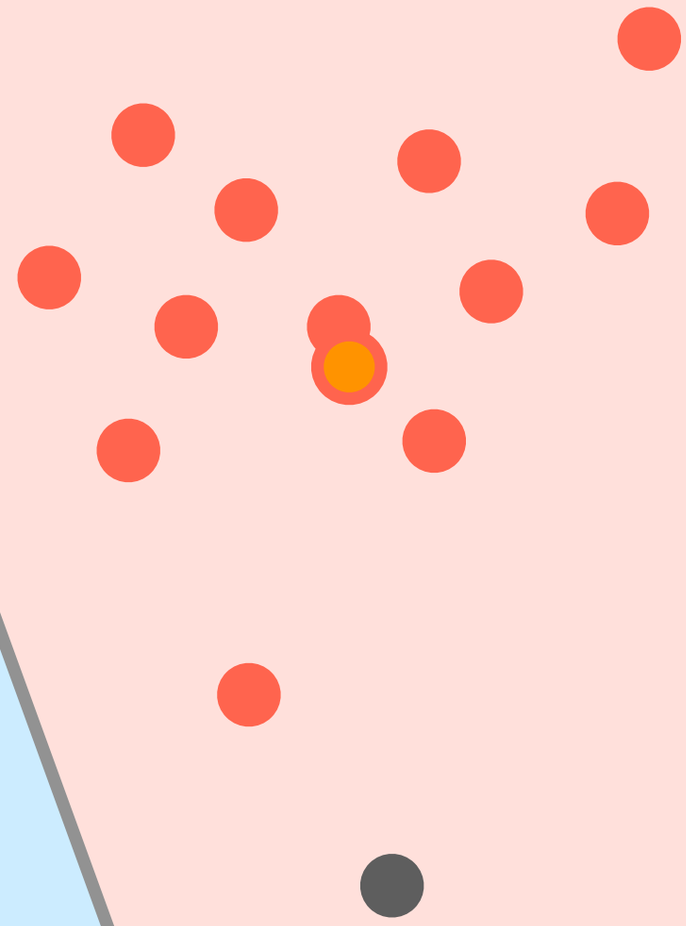
Decision boundary



This classifier we've created assumes a *generative model*

We just created a **classifier**

(a procedure that given a new data point tells us what "class" it belongs to)



What should the label of this new point be?

Whichever cluster has higher probability!

You've seen generative models before for prediction

Linear regression!

Model parameters: slope *m*, intercept *b*

y

Label
(1D in this case)

x

Feature vector
(1D in this case)

Label
(1D in this case)

y

Model parameters: slope *m*, intercept *b*

x

Feature vector
(1D in this case)

For specific value of *x*,
assume *y* drawn from
Gaussian with mean
mx+*b*, standard dev σ

Predictive Data Analysis

Training data

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

Goal: Given new feature vector x, predict label y

- y is discrete (such as colors red and blue)
  ➔ prediction method is called a **classifier**

- y is continuous (such as a real number)
  ➔ prediction method is called a **regressor**

A giant zoo of methods