Carnegie Mellon University
Heinz College

Unstructured Data Analysis

Lecture 9: Topic modeling (cont'd); intro to predictive data analytics

George Chen

An Alternative Feature Vector Representation for Text: TF-IDF

Intuition: words that appear in more documents are likely less useful (same intuition as stop words!) — let's *downweight* these words!

Word

1 2 … ***d***

Document

1

2

⋮

n

Term frequency (TF)

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

multiply TF by $\log \dfrac{1}{\mathbb{P}(\text{document has word } j)} = \log \dfrac{1}{\left(\frac{\#\text{ documents that have word } j}{n}\right)}$

$= \log \boxed{\dfrac{n}{\#\text{ documents that have word } j}}$

Hack: add 1 to numerator & 1 to denominator

An Alternative Feature Vector Representation for Text: TF-IDF

Intuition: words that appear in more documents are likely less useful (same intuition as stop words!) — let's *downweight* these words!

There are many TF-IDF variants! (Lots of hacks!)

Word

1 2 . . . d

Document

1
2
$\vdots$
n

Term frequency (TF)

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

$$\times \left[1 + \log \frac{1 + n}{1 + \text{\# documents that have word } j} \right]$$

Default TF-IDF weighting in sklearn

sklearn's default behavior further normalizes each row to have Euclidean norm 1

TF-IDF is in your HW2

(usage is similar to `CountVectorizer` from the demo)

How to choose the number of topics *k*?

Look at *within topic variability* and *between topic variability*

Within Topic Variability

Let's look at top-20 word lists (the ones from the demo)

```
[Topic 0]
good : 0.01592254908129389
like : 0.01584763067117222
just : 0.01571497459780987 4
think : 0.014658035148150044
don : 0.01336602502776772
time : 0.012159230893303024
year : 0.011442050656933937
new : 0.008768217977593912
years : 0.00843922077825026
game : 0.008416482579473757
make : 0.008318270139852606
ve : 0.00805613381872604
know : 0.00786552901690738
going : 0.007357414502894818
better : 0.007305177940555176
really : 0.007282768897233162
got : 0.00710024216618747 5
way : 0.00702025822161851 9
team : 0.006901091494924322
car : 0.006860678090522195
```

```
[Topic 1]
drive : 0.025114459755967225
card : 0.019045045227142 93
scsi : 0.0157480734630964 5
disk : 0.015086151949241311
u[illegible]e : 0.01311205775591249
outpu[illegible] : 0.012487568705565076
file : 0.[illegible]1147497481922729 8
bit : 0.0114[illegible]0491727323115
hard : 0.01042[illegible]35918865882
entry : 0.0099623[illegible]704950415
memory : 0.00989293[illegible]703385204
mac : 0.009531449582[illegible]7765
v[illegible]deo : 0.009451338641933656
drives [illegible]
pc : 0.0090703286112168
windows : 0.[illegible]
16 : 0.0079[illegible]823814975238
bus : 0.007927283819698584
contro[illegible]
program : 0.00784268458596016
```

```
[Topic 2]
10 : 0.0320292203
00 : 0.0269643305
25 : 0.0218296912
15 : 0.0206063577
11 : 0.0206043503
20 : 0.0204957609
12 : 0.0203766844
14 : 0.0180554708
16 : 0.0164602656
13 : 0.0160230124
17 : 0.0160189031
18 : 0.0159314160
30 : 0.0134871298
[illegible]30831
24 : 0.0131269045
[illegible]125205615
55 : 0.0125002331
21 : 0.0122642479
40 [illegible]1525
22 : 0.0112072317
```

If we see the word "good", how likely are we to see the word "years"?

P(see word "years" | see word "good")

Focus on a single topic at a time

If this probability is high for every pair of words in the top-20 list, then in some sense the topic is more "coherent"

Between Topic Variability

Let's look at top-20 word lists (the ones from the demo)

```
[Topic 0]
good : 0.01592254908129389
like : 0.01584763067117222
just : 0.015714974597809874
think : 0.014658035148150044
don : 0.01336602502776772
time : 0.012159230893303024
year : 0.011442050656933937
new : 0.008768217977593912
years : 0.00843922077825026
game : 0.008416482579473757
make : 0.008318270139852606
ve : 0.00805613381872604
know : 0.00786552901690738
going : 0.007357414502894818
better : 0.007305177940555176
really : 0.007282768897233162
got : 0.007100242166187475
way : 0.007020258221618519
team : 0.006901091494924322
car : 0.006860678090522195
```

```
[Topic 1]
drive : 0.025114459755967225
card : 0.01904504522714293
scsi : 0.01574807346309645
disk : 0.015086151949241311
use : 0.01311205775591249
output : 0.012487568705565076
file : 0.011474974819227298
bit : 0.011450491727323115
hard : 0.010426435918865882
entry : 0.009962381704950415
memory : 0.009892936703385204
mac : 0.009531449582937765
video : 0.009451338641933656
drives : 0.009074000962777757
pc : 0.0090703286112168
windows : 0.008135023862197355
16 : 0.00798823814975238
bus : 0.007927283819698584
controller : 0.007902057876189581
program : 0.00784268458596016
```

```
[Topic 2]
10 : 0.0320292203
00 : 0.0269643305
25 : 0.0218296912
15 : 0.0206063577
11 : 0.0206043503
20 : 0.0204957609
12 : 0.0203766844
14 : 0.0180554708
16 : 0.0164602656
13 : 0.0160230124
17 : 0.0160189031
18 : 0.0159314160
30 : 0.0134871298
50 : 0.0133230831
24 : 0.0131269045
19 : 0.0125205615
55 : 0.0125002331
21 : 0.0122642479
40 : 0.0119281525
22 : 0.0112072317
```

If "good" only shows up in the top-20 word list for topic 0, then it is considered a **unique top word** for topic 0

Each topic has a **number of unique top words**

How to Choose Number of Topics k?

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

Coherence (within topic variability):

$$\sum_{\substack{\text{top words } v,w \\ \text{that are not the same}}} \log \frac{\text{\# documents that contain both } v \text{ and } w \; + 0.1}{\text{\# documents that contain } w}$$

log of P(see word v | see word w)

avoid numerical issues

Number of unique words (between topic variability):

Can average each of these across the topics

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

Can plot average coherence vs k, and average # unique words vs k (for values of k you are willing to try)

Unlike for CH index, no clear way to trade off between avg. coherence and avg. # unique words (they aren't even in the same units!!!)

Topic Modeling: Last Remarks

- There are actually *many* topic models, not just LDA
  - Hierarchical Dirichlet Process, correlated topic models, SAGE, anchor word topic models, ProdLDA, embedded topic model, …
- Dynamic topic models: track how topics change *over time*
- Trivial to add supervision to topic models! Can have topics learned help with prediction tasks!
- Reminder: learning topic models can be very sensitive to random initialization

95-865

Part I: Exploratory data analysis

Identify structure present in "unstructured" data

- Frequency and co-occurrence analysis
- Visualizing high-dimensional data/dimensionality reduction
- Clustering
- Topic modeling

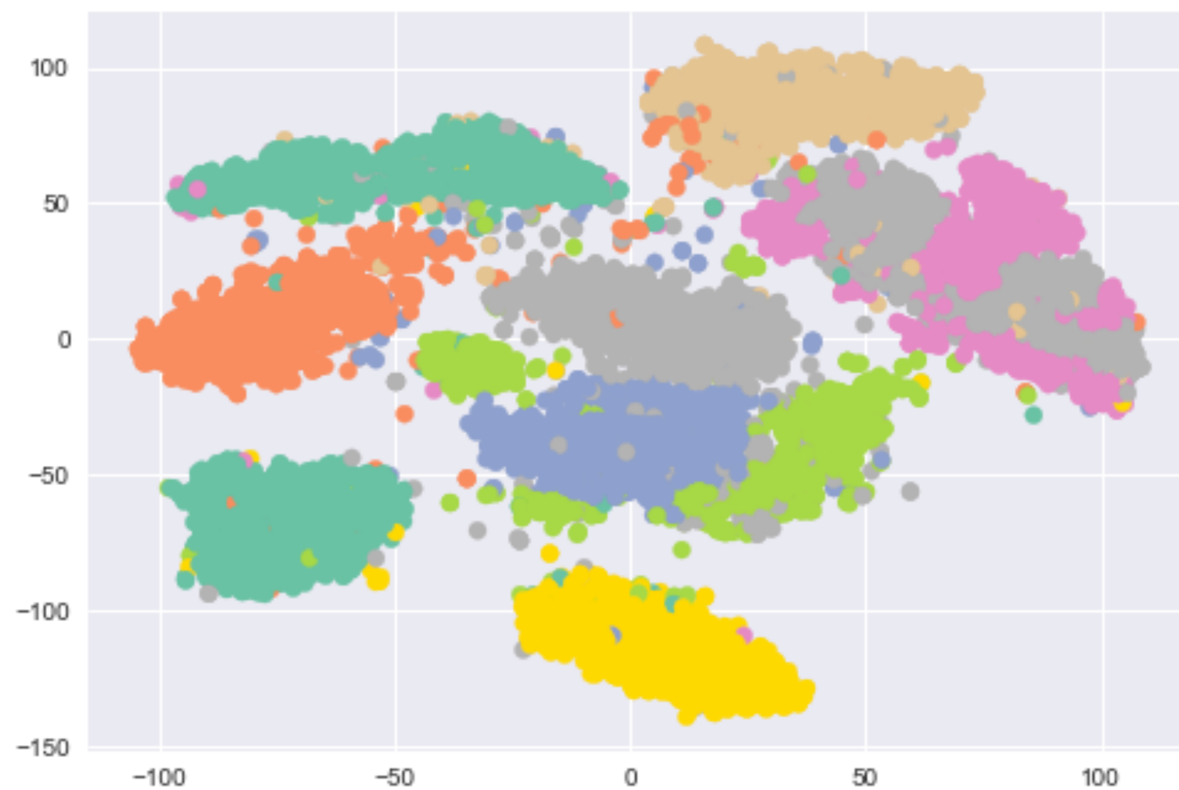
Part II: Predictive data analysis

Make predictions using known structure in data

- Basic concepts and how to assess quality of prediction models
- Neural nets and deep learning for analyzing images and text

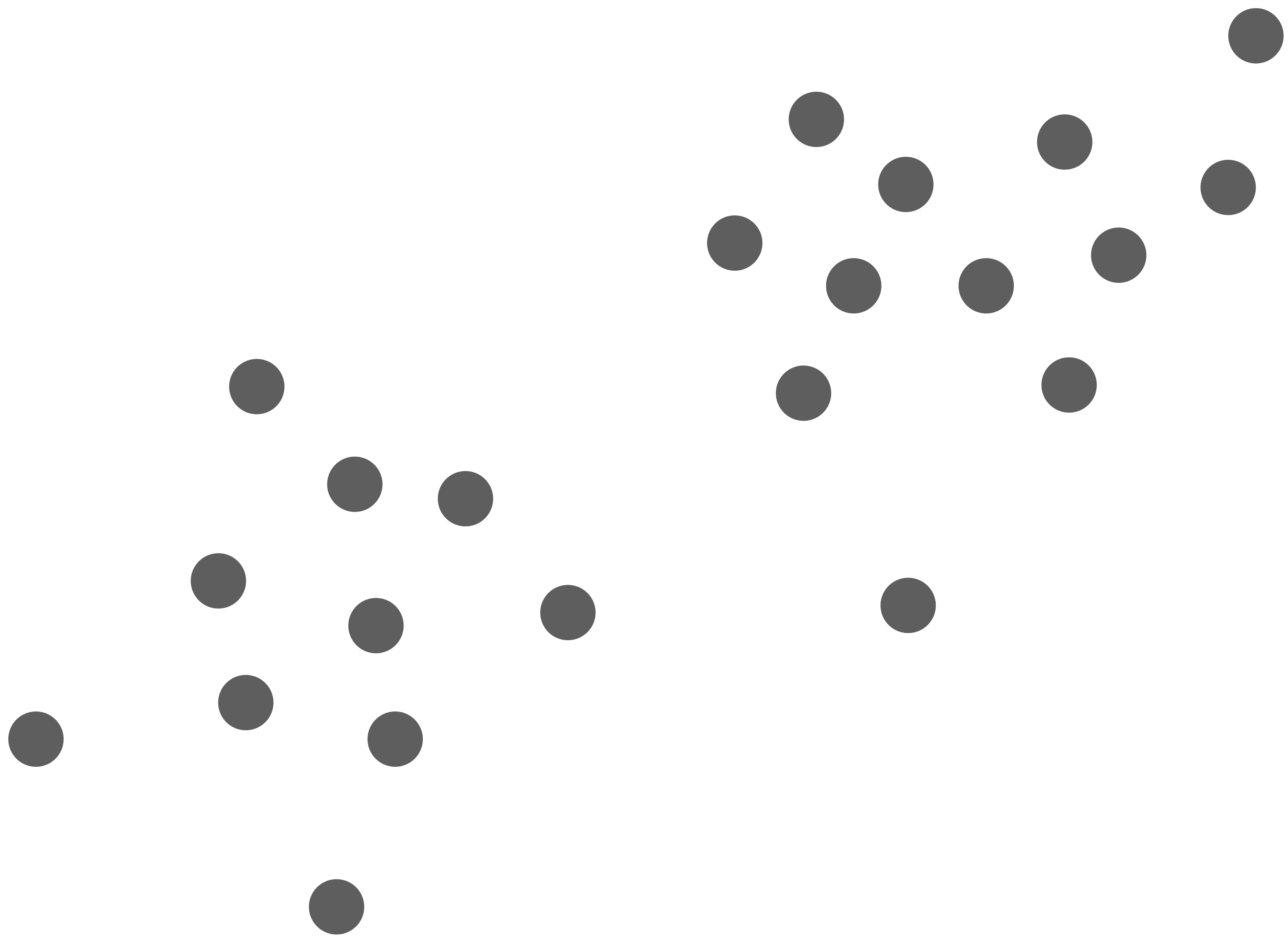
What if we have labels?

Disclaimer: unfortunately "***k***" means many things



Example: MNIST handwritten digits have known labels

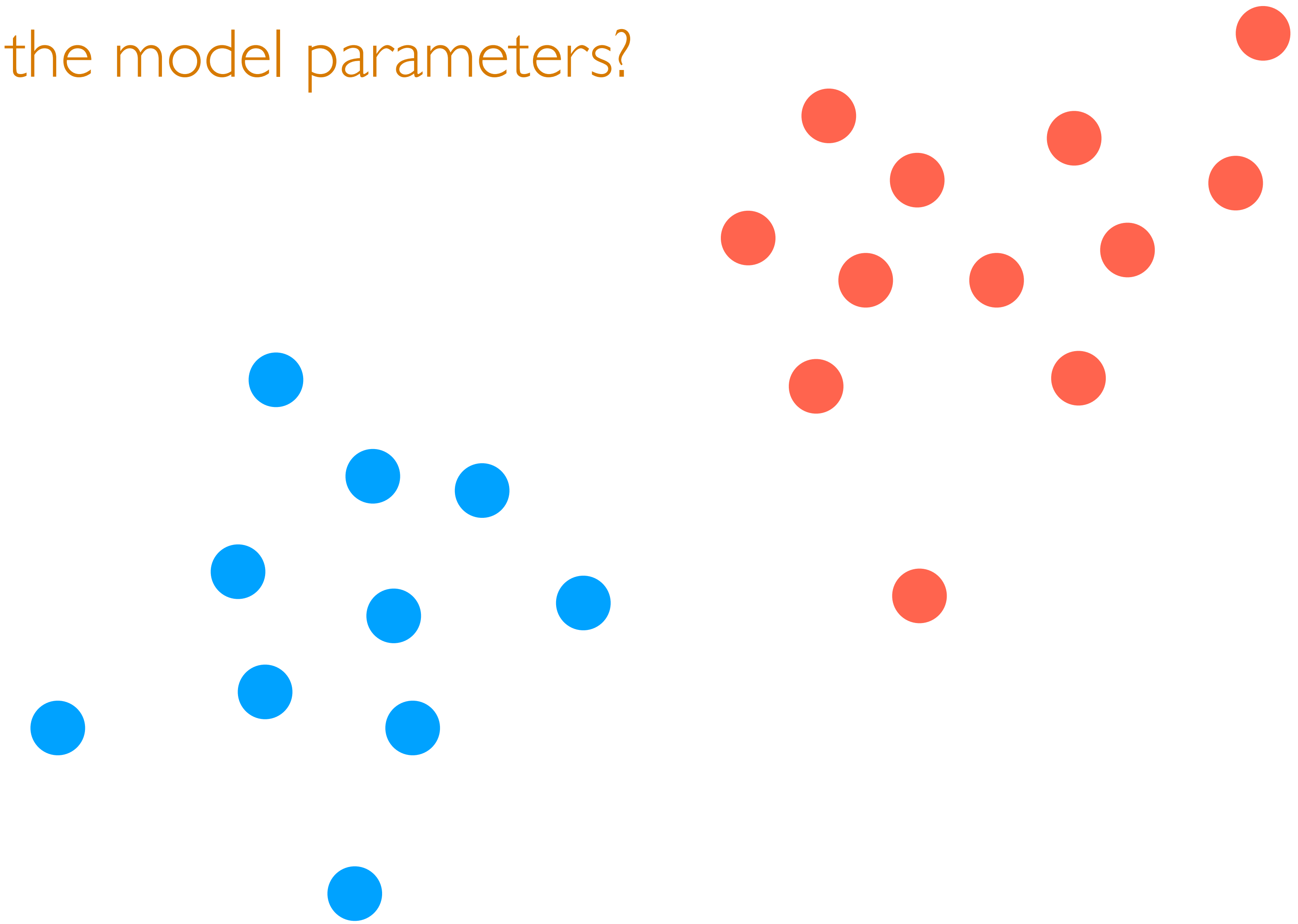
If the labels are known…



If the labels are known…

And we assume data generated by GMM…

What are the model parameters?



(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 probabilities, means, and covariances**

(often done using ***k***-means)

Repeat until convergence:

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

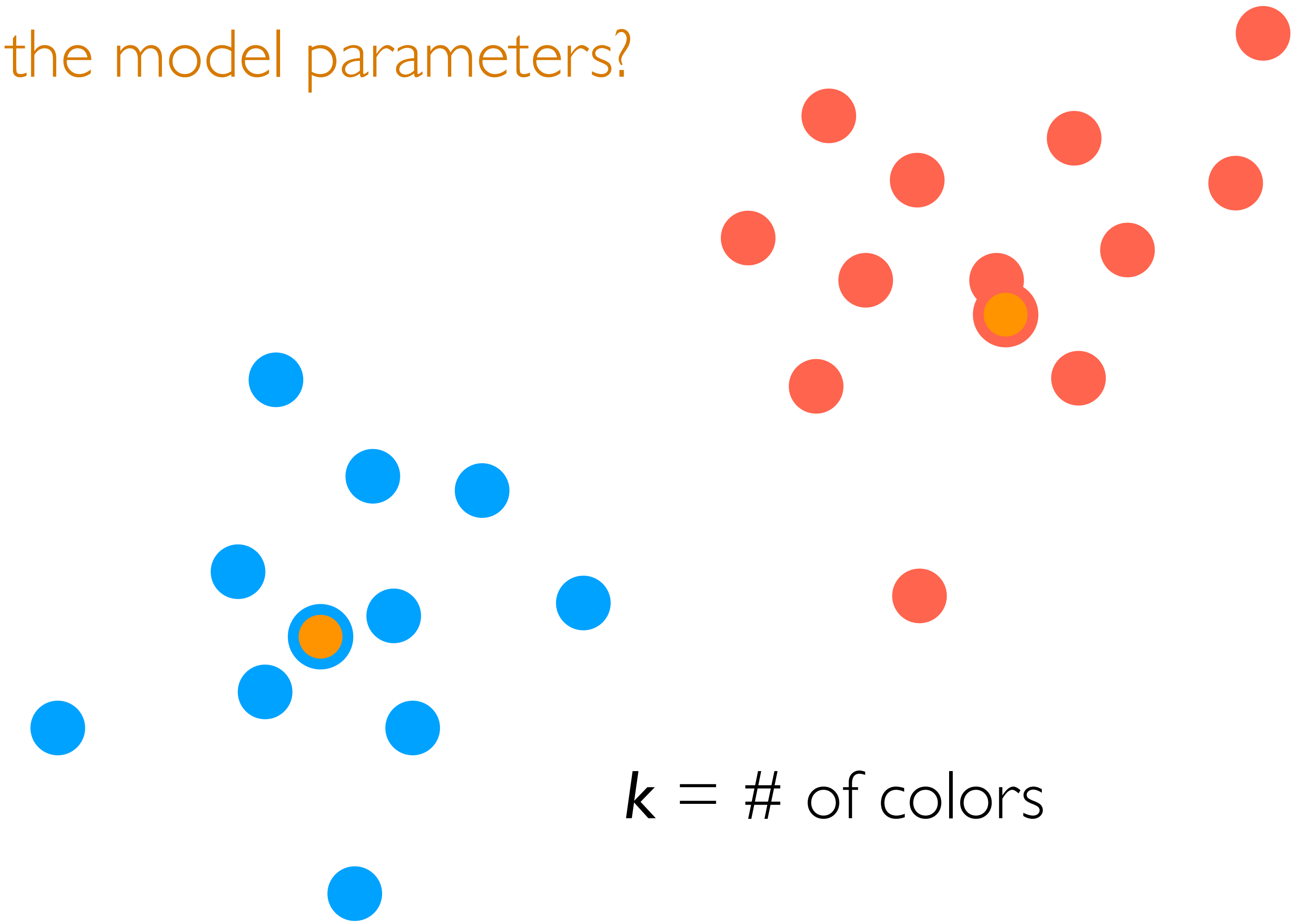
Step 3: Update **cluster probabilities, means, and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

We don't need to repeat until convergence

If the labels are known…

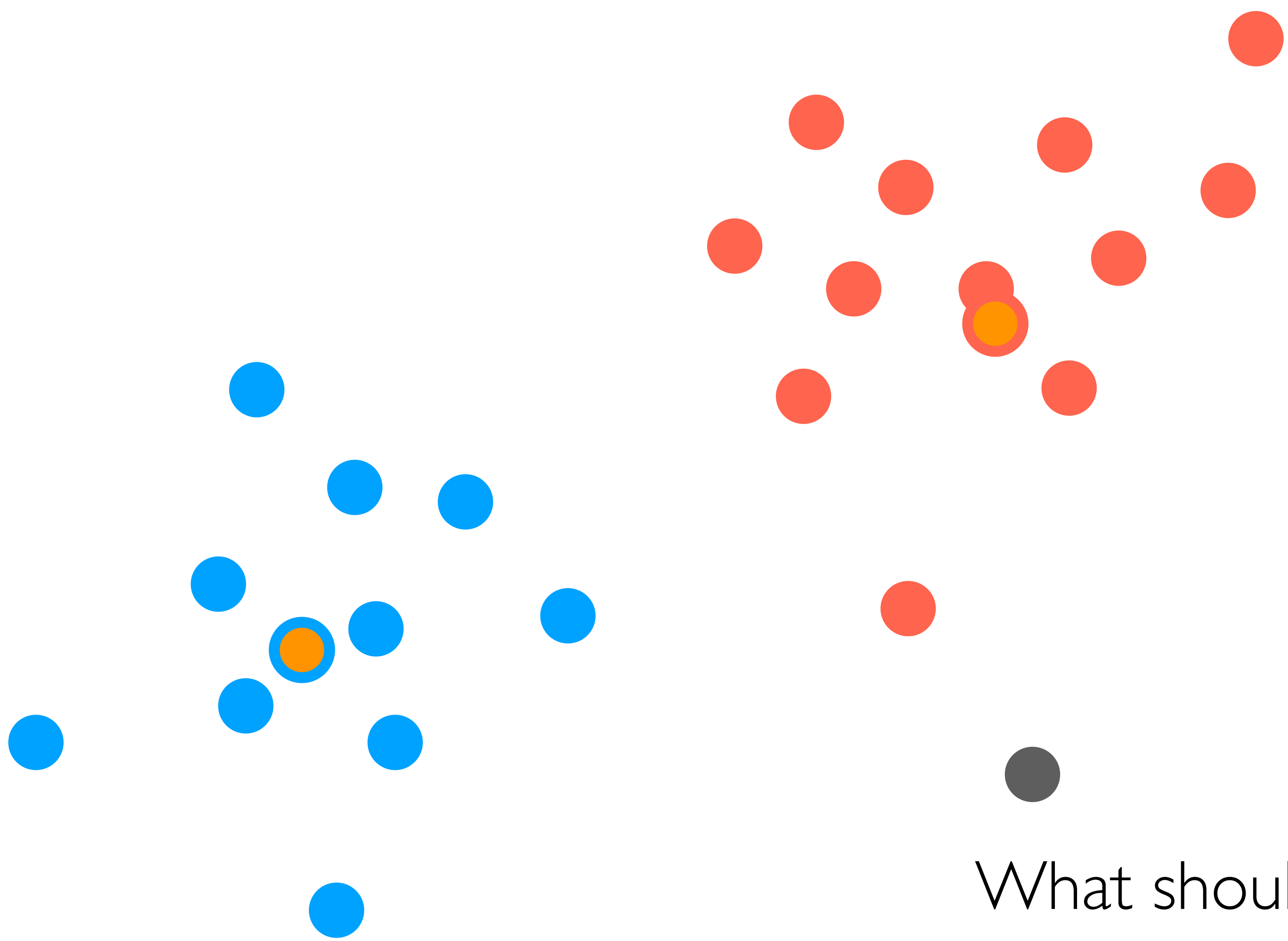
And we assume data generated by GMM…

What are the model parameters?



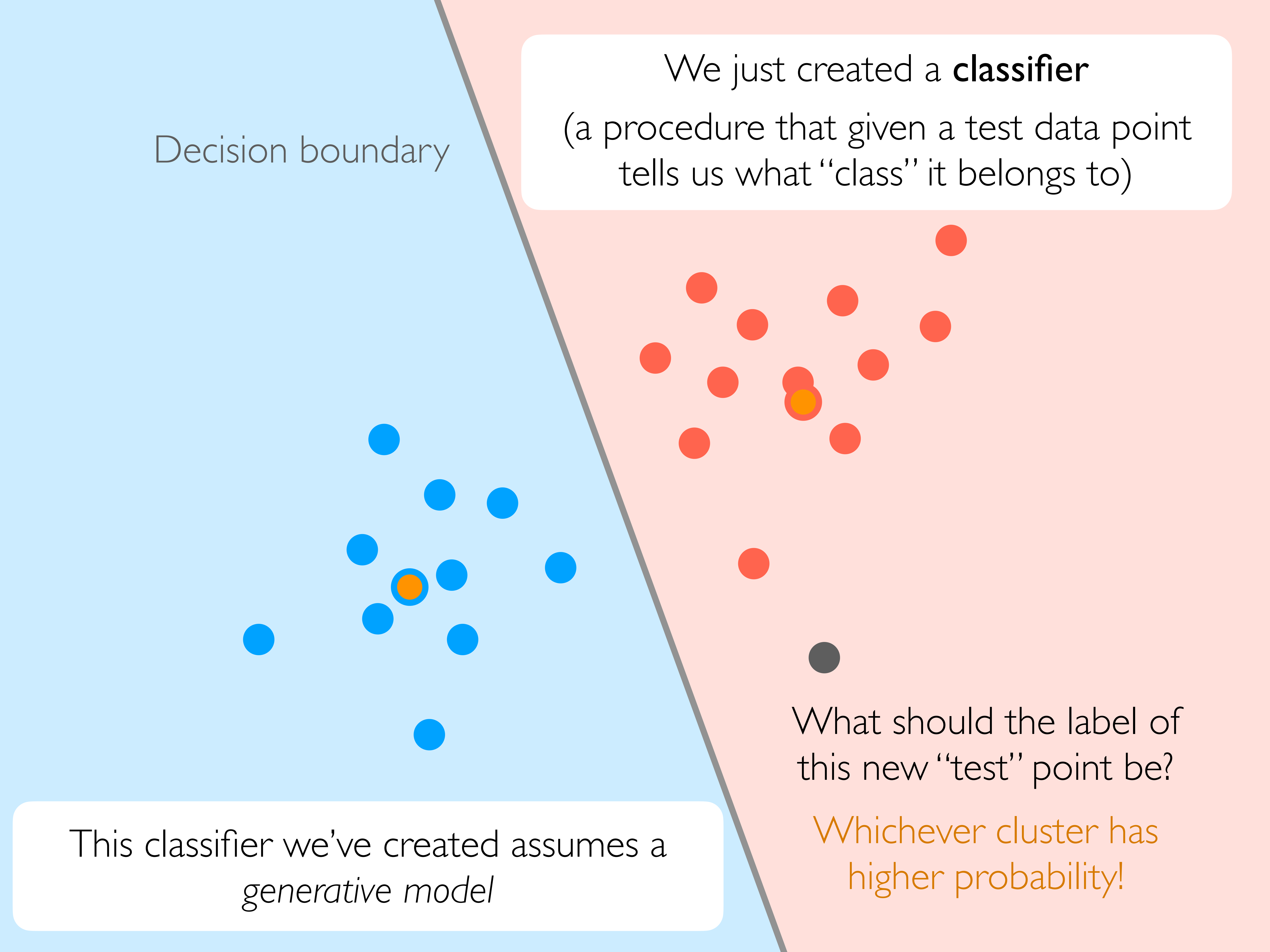
k = # of colors

We can directly estimate cluster means, covariances



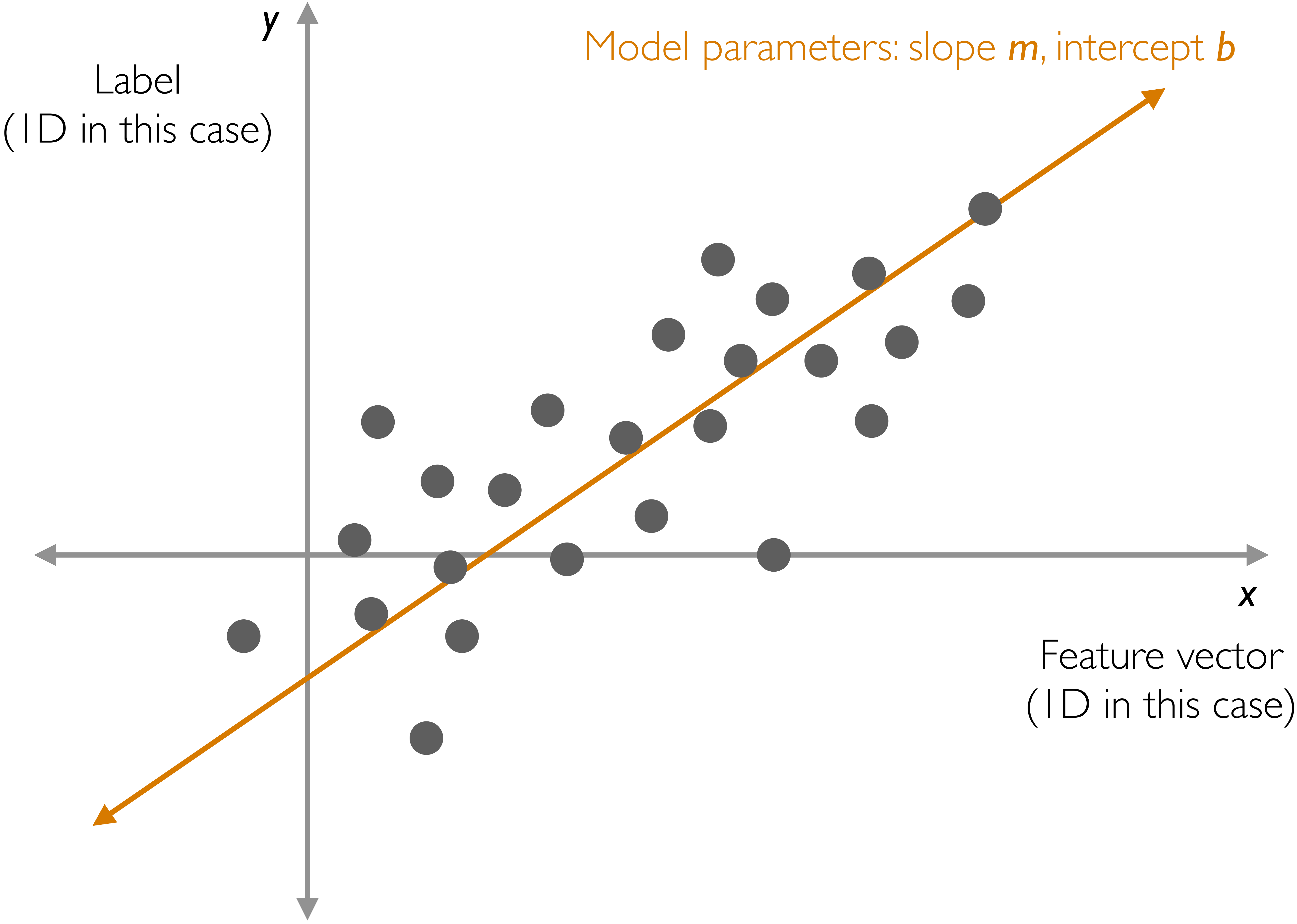
What should the label of this new "test" point be?

Whichever cluster has higher probability!

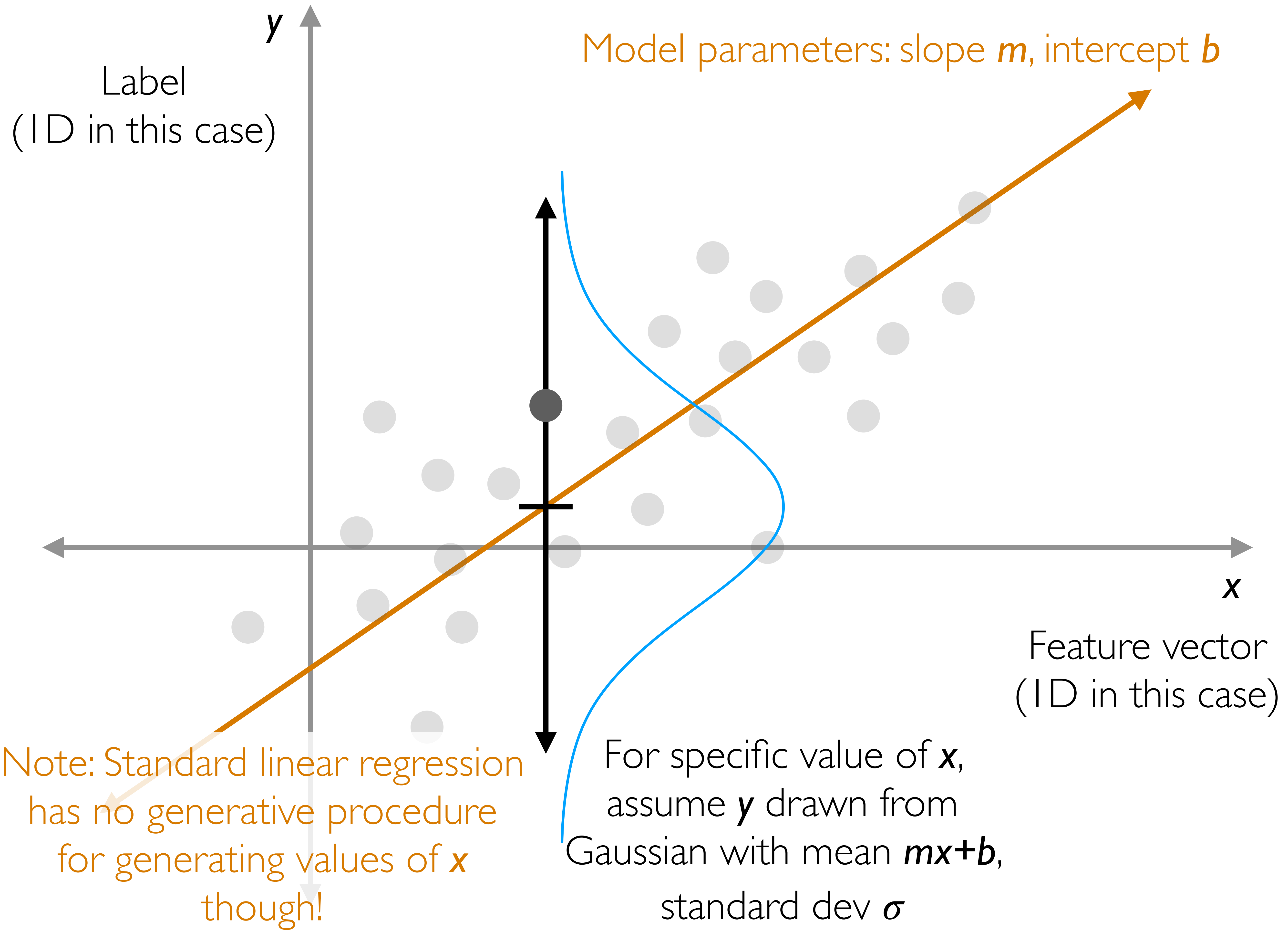

We just created a **classifier**
(a procedure that given a test data point tells us what "class" it belongs to)
Decision boundary
What should the label of this new "test" point be?
Whichever cluster has higher probability!
This classifier we've created assumes a *generative model*

You’ve seen a prediction model that is partly a generative model

Linear regression!



y
Label
(1D in this case)
Model parameters: slope m, intercept b
x
Feature vector
(1D in this case)


y
Label
(1D in this case)
Model parameters: slope m, intercept b
x
Feature vector
(1D in this case)
Note: Standard linear regression has no generative procedure for generating values of x though!
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 test feature vector $\boldsymbol{x}$, predict label $\boldsymbol{y}$

- $\boldsymbol{y}$ is discrete (such as colors red and blue)
  ➔ prediction is referred to as **classification**

- $\boldsymbol{y}$ is continuous (such as a real number)
  ➔ prediction is referred to as **regression**

We could have many such test feature vectors, which we collectively refer to as *test data*

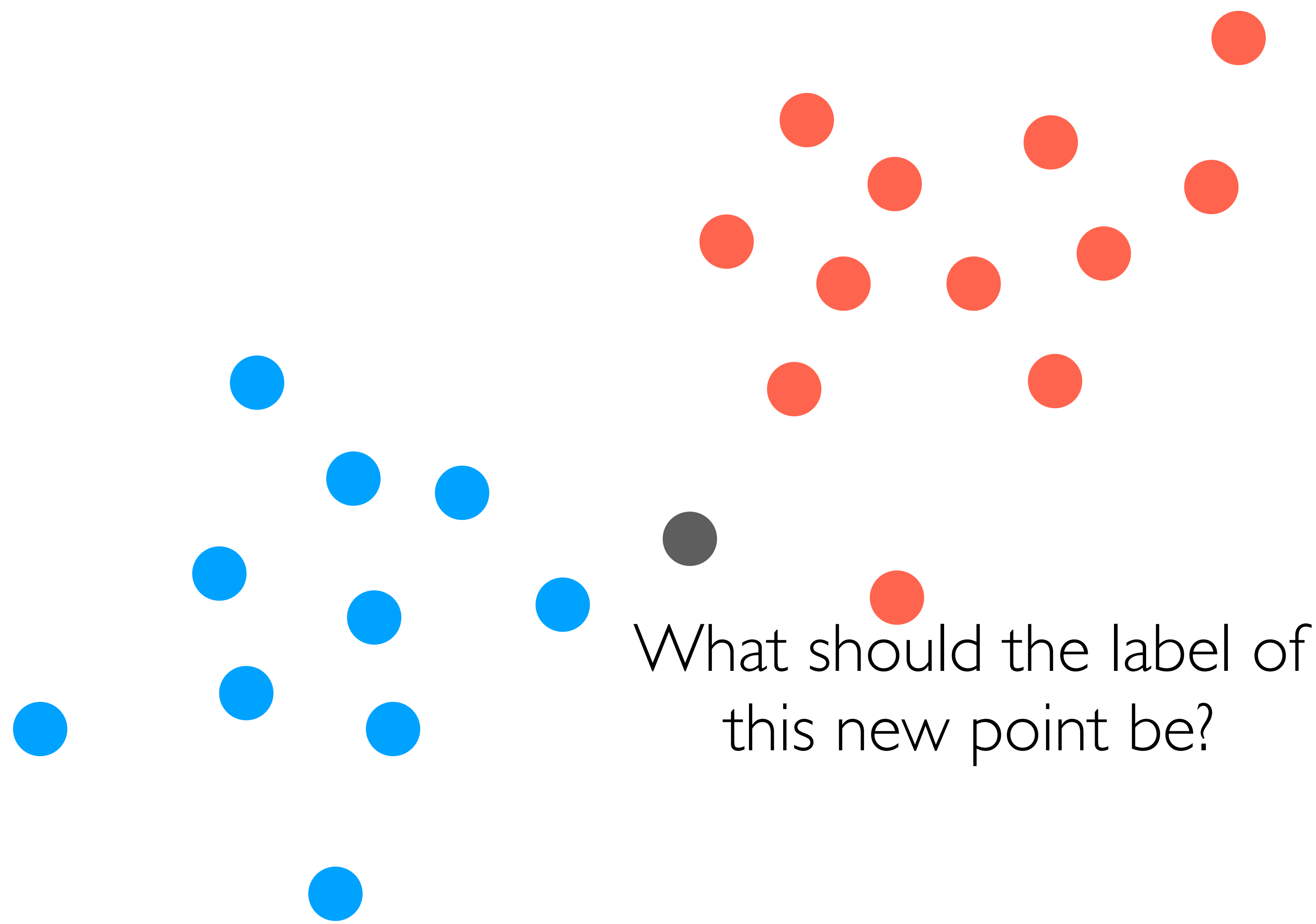
A giant zoo of methods

- Generative models (like what we just described)

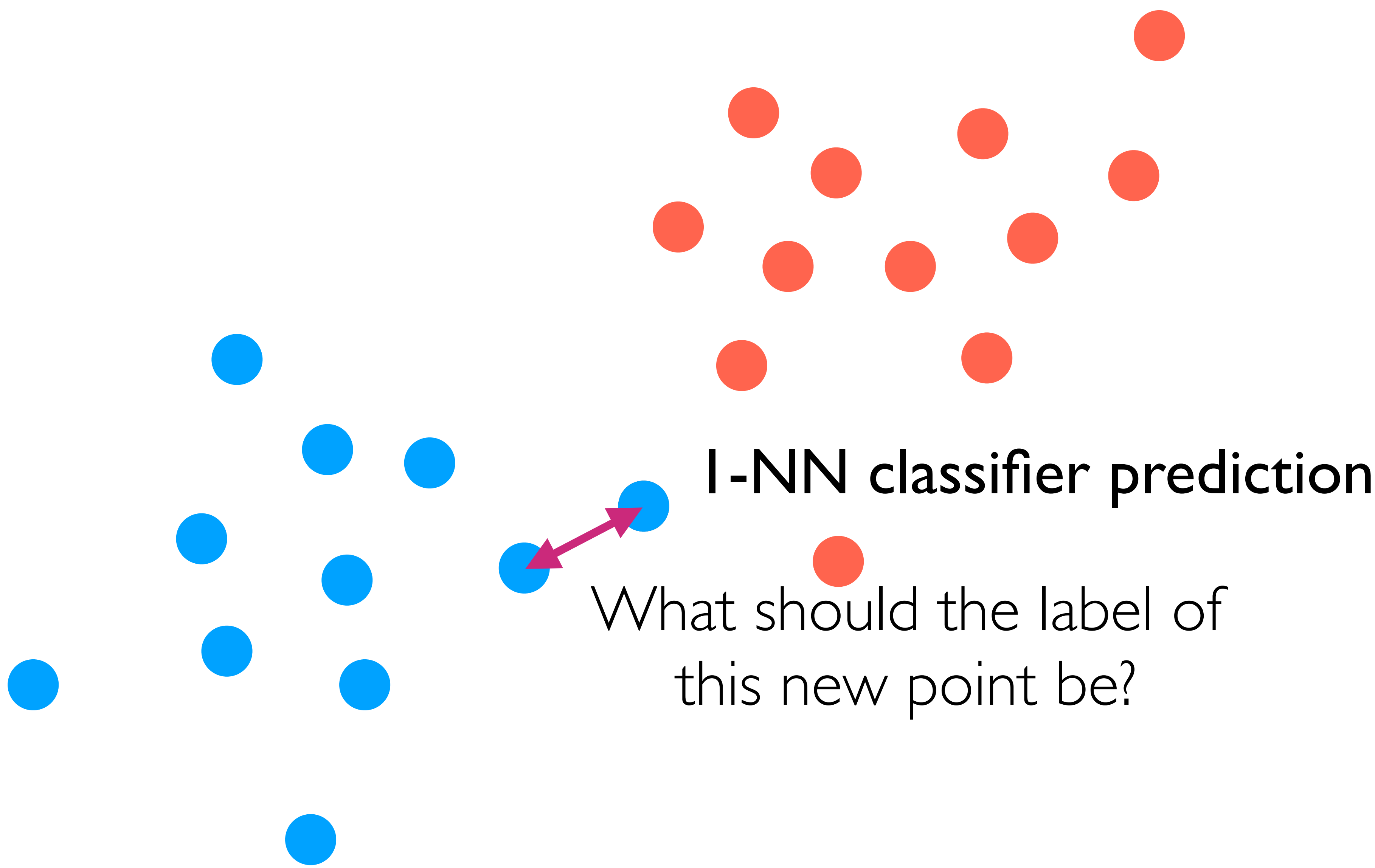
- Discriminative models (just care about learning prediction rule; after training model, we don't have a way to generate data)

Example of a Discriminative Method: *k*-NN Classification

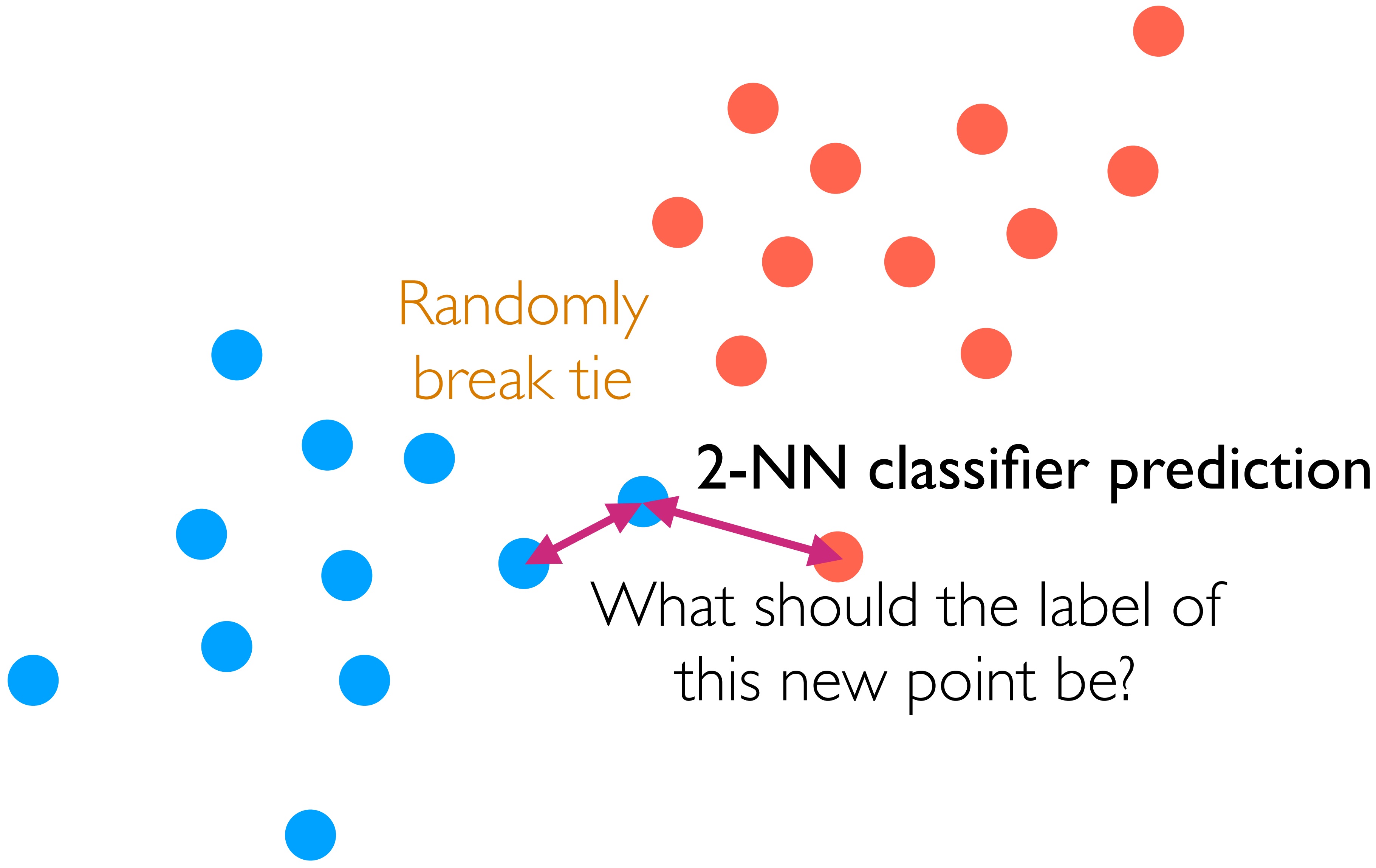
Example: *k*-NN Classification



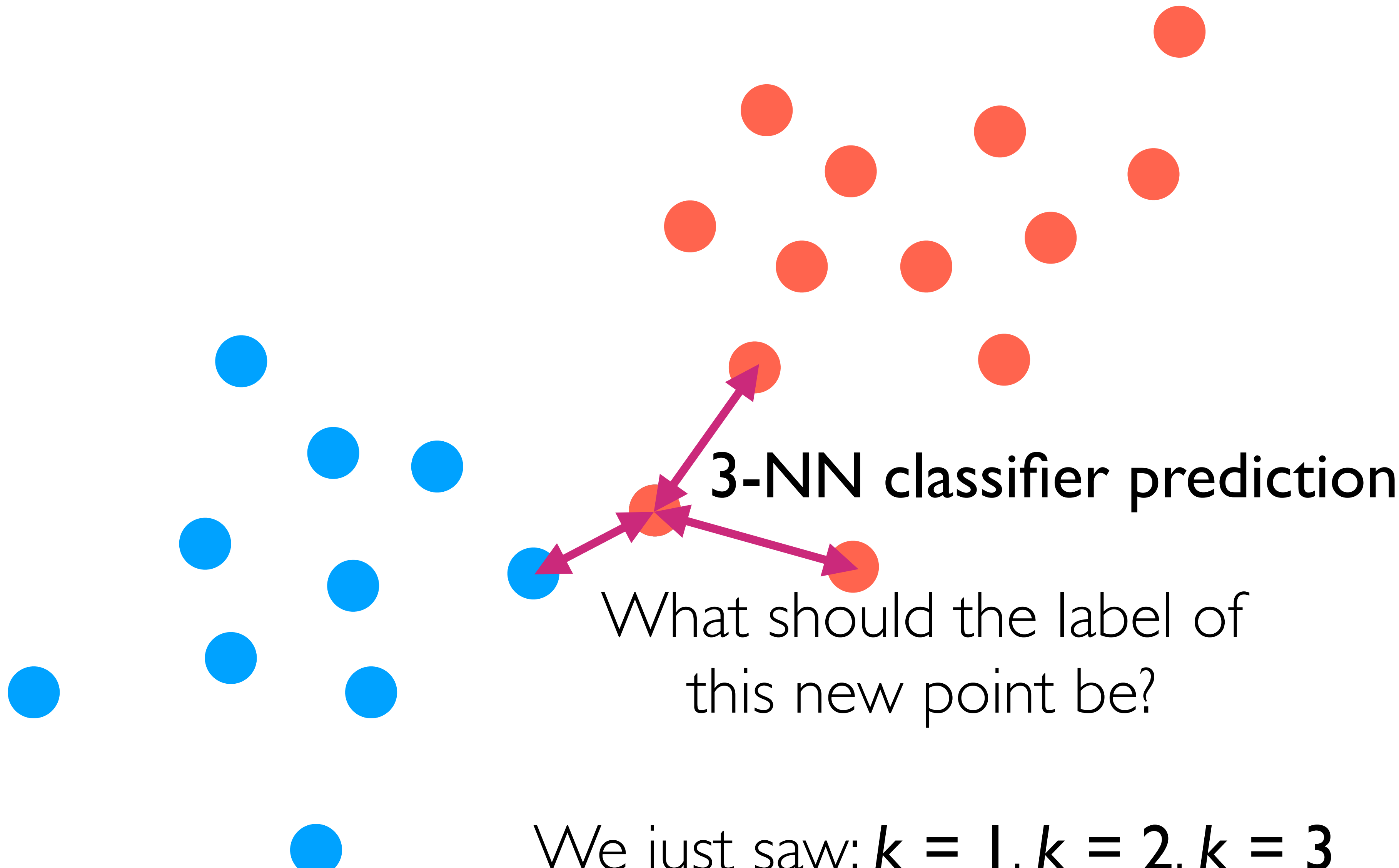
Example: *k*-NN Classification



Example: *k*-NN Classification



Example: *k*-NN Classification



We just saw: k = 1, k = 2, k = 3

What happens if $k = n$?

How do we choose ***k***?

What I'll describe next can be used to select hyperparameter(s) for any prediction method

Fundamental question:
How do we assess how good a prediction method is?

Hyperparameters vs. Parameters

- We fit a model's parameters to training data (terminology: we "learn" the parameters)

- We pick values of hyperparameters and they do *not* get fit to training data

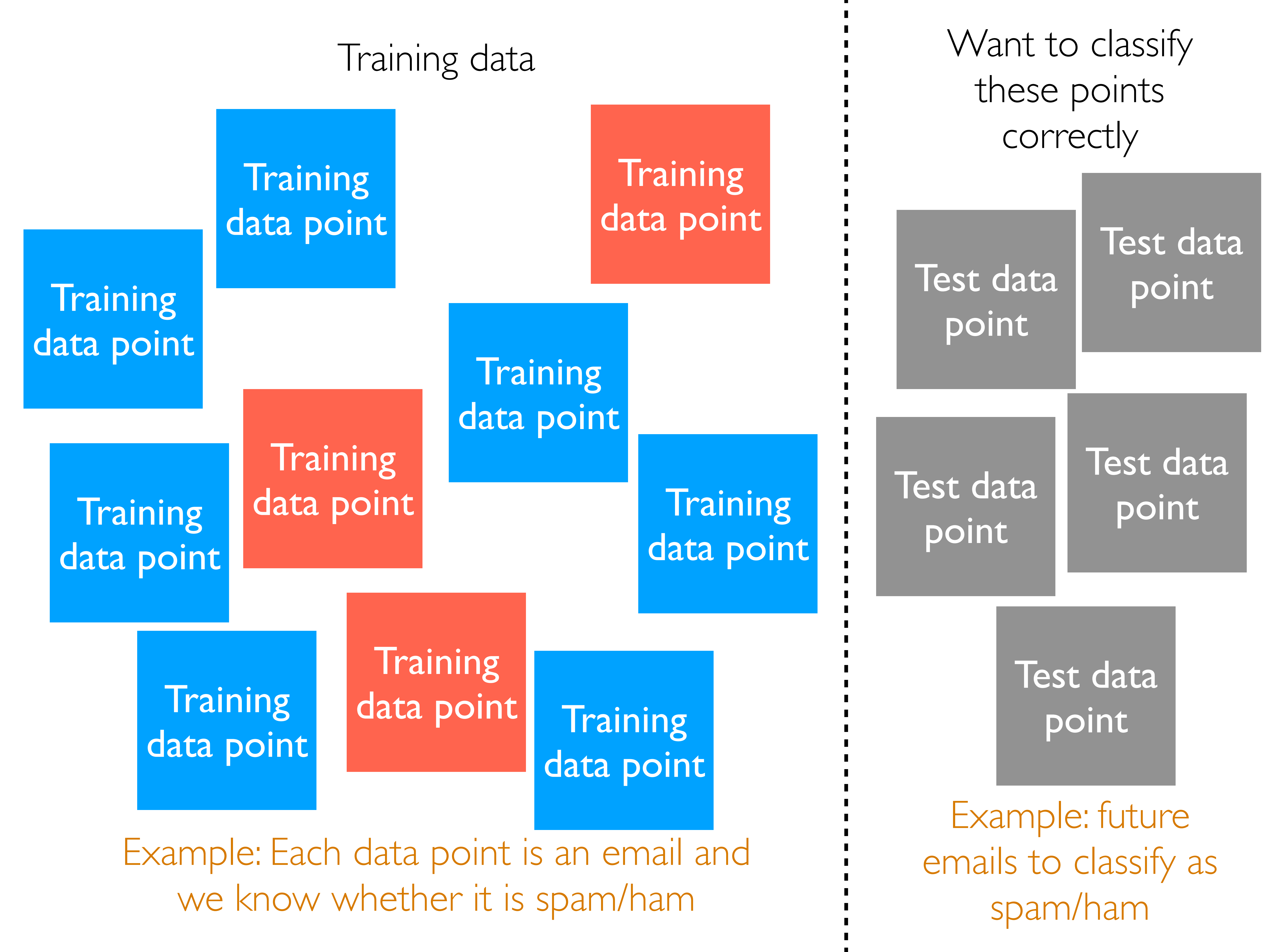
- Example: Gaussian mixture model
  - Hyperparameter: number of clusters ***k***
  - Parameters: cluster probabilities, means, covariances

- Example: ***k***-NN classification
  - Hyperparameter: number of nearest neighbors ***k***
  - Parameters: N/A

⚠️ Major assumption:
the training and test data "look similar"
(technically: training and test data are i.i.d. sampled from the same underlying distribution)

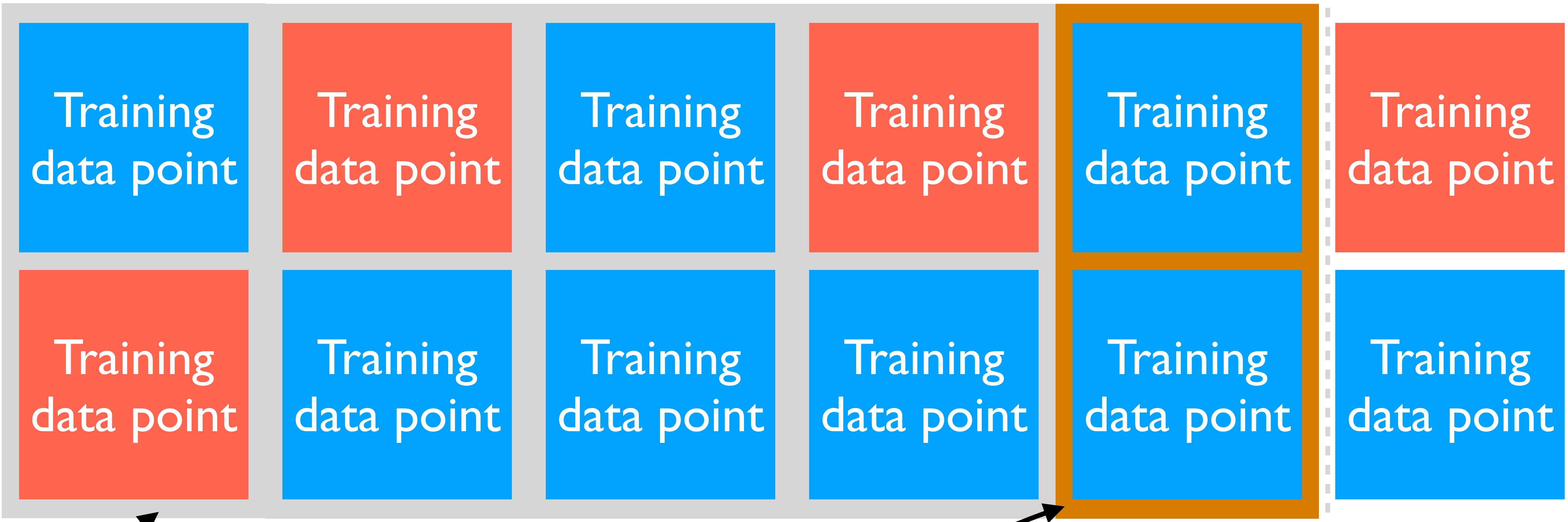
In other words, we assume that there is an *unknown* generative process that produces every pair (x_i, y_i) from the exact same distribution

Prediction becomes harder when training and test data appear quite different!


Training data
Training data point
Training data point
Training data point
Training data point
Training data point
Training data point
Training data point
Training data point
Training data point
Training data point
Example: Each data point is an email and we know whether it is spam/ham
Want to classify these points correctly
Test data point
Test data point
Test data point
Test data point
Test data point
Example: future emails to classify as spam/ham

(this shuffling makes sense since we assume data are i.i.d.)

Predicted labels



Train method on data in gray

Predict on data in orange

Terminology for this class:
“**Proper training data**”
(the gray box)

“**Validation data**”
(the orange box)

Compute prediction error

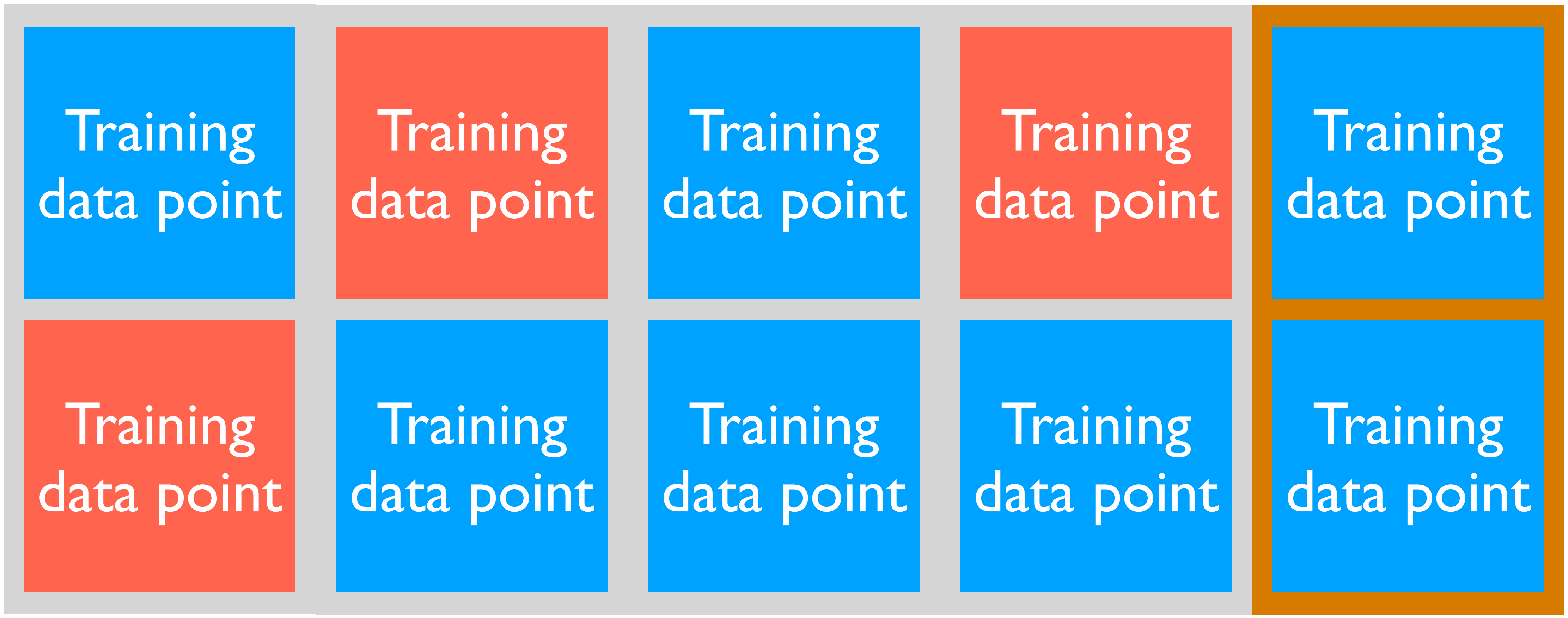
50%

This is called data splitting/“train-validation split”

In this example: we did a 80%-20% split

Some people, including sklearn, call this “train-test split” but in this class, we will use “test data” to refer to true test data that the training procedure does not see

But we could have chosen different proper training/validation data!



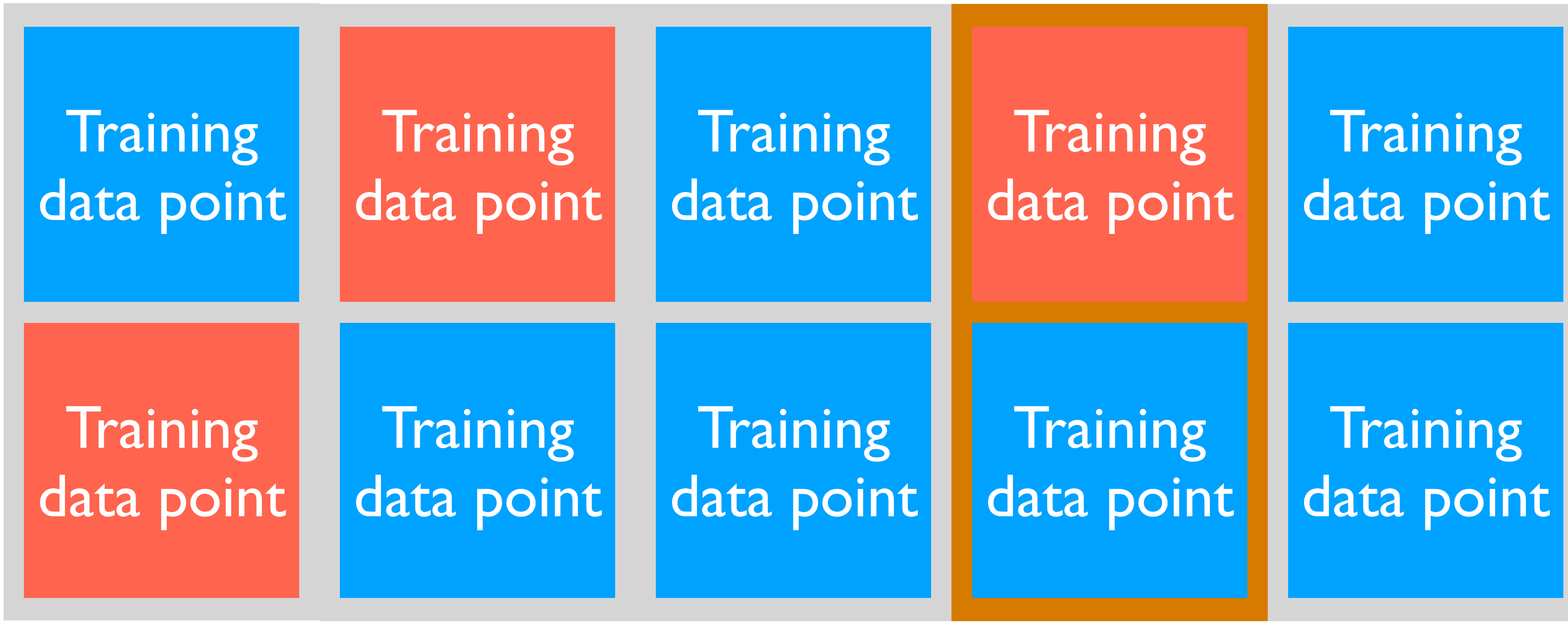
Train method on data in gray

Predict on data in orange

Compute prediction error

50%

But we could have chosen different proper training/validation data!



Train method on data in gray

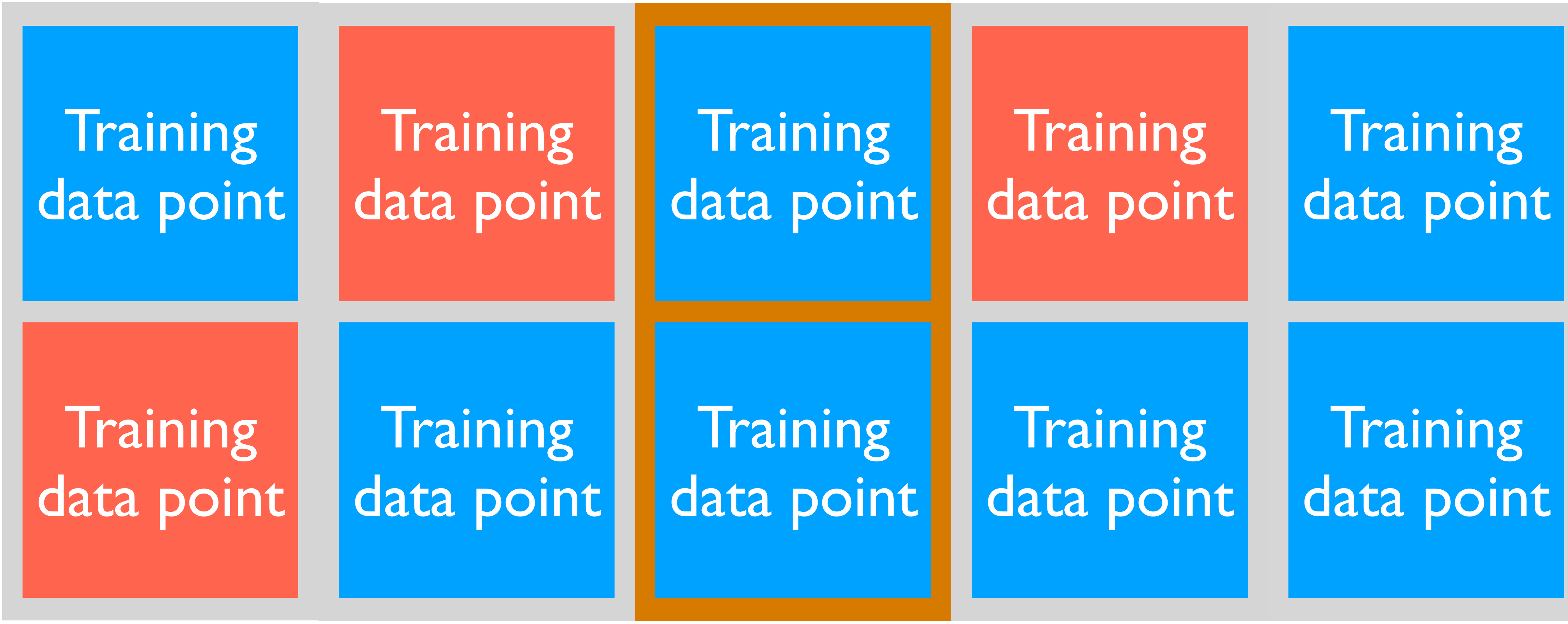
Predict on data in orange

Compute prediction error

0%

50%

But we could have chosen different proper training/validation data!



Train method on data in gray

Predict on data in orange

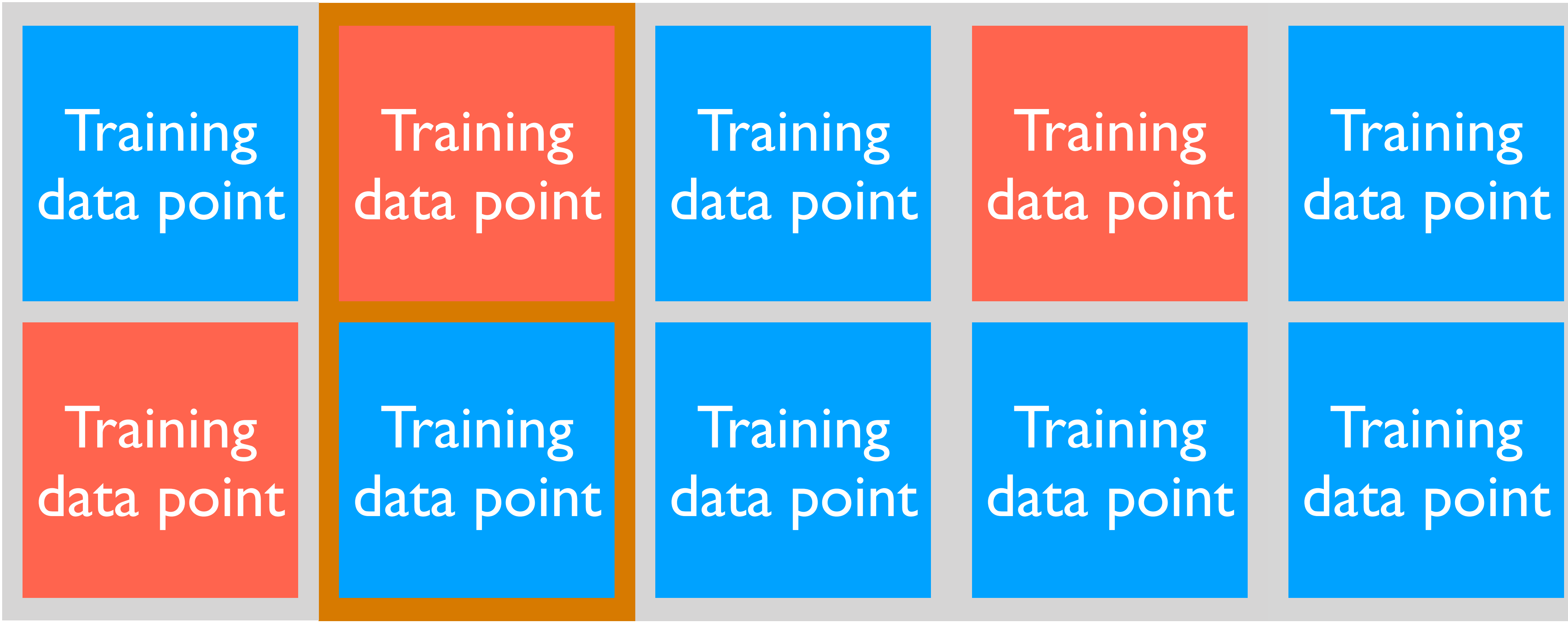
Compute prediction error

50%

0%

50%

But we could have chosen different proper training/validation data!



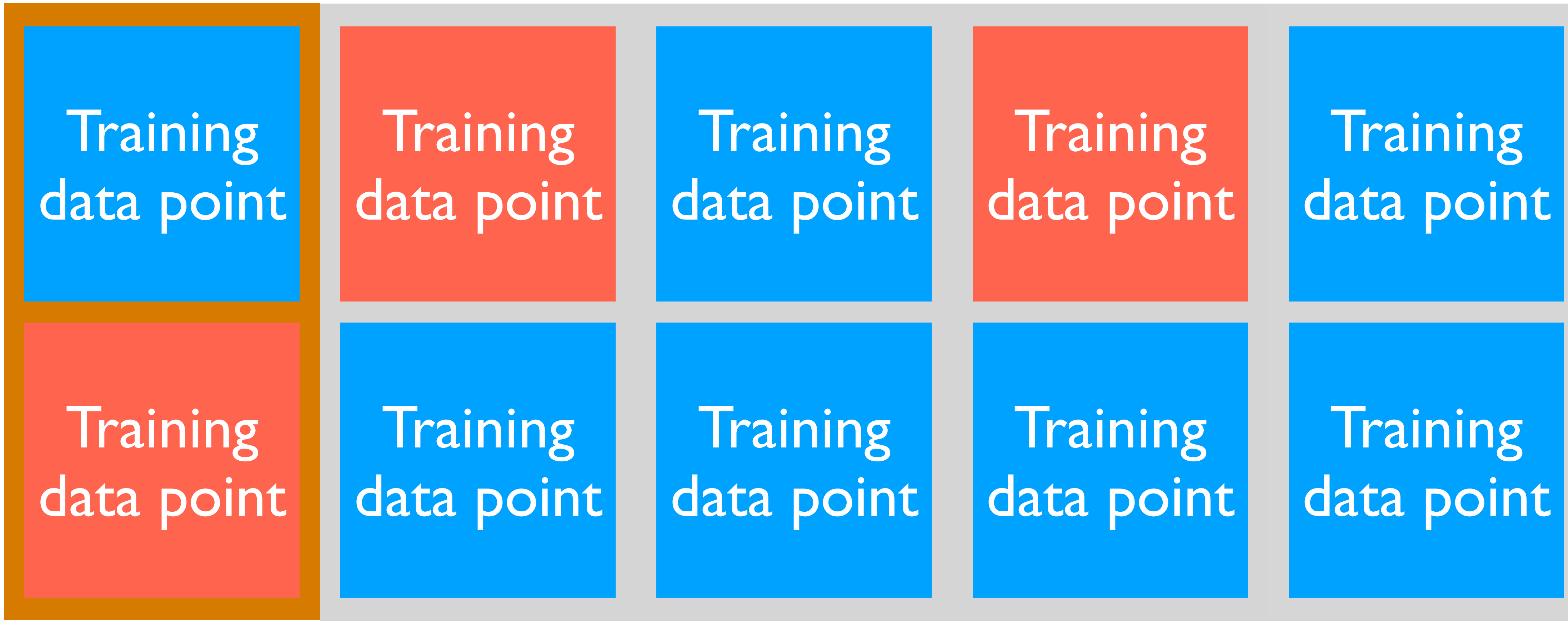
Train method on data in gray

Predict on data in orange

Compute prediction error

0%   50%   0%   50%

But we could have chosen different proper training/validation data!



Train method on data in gray

Predict on data in orange

Compute prediction error

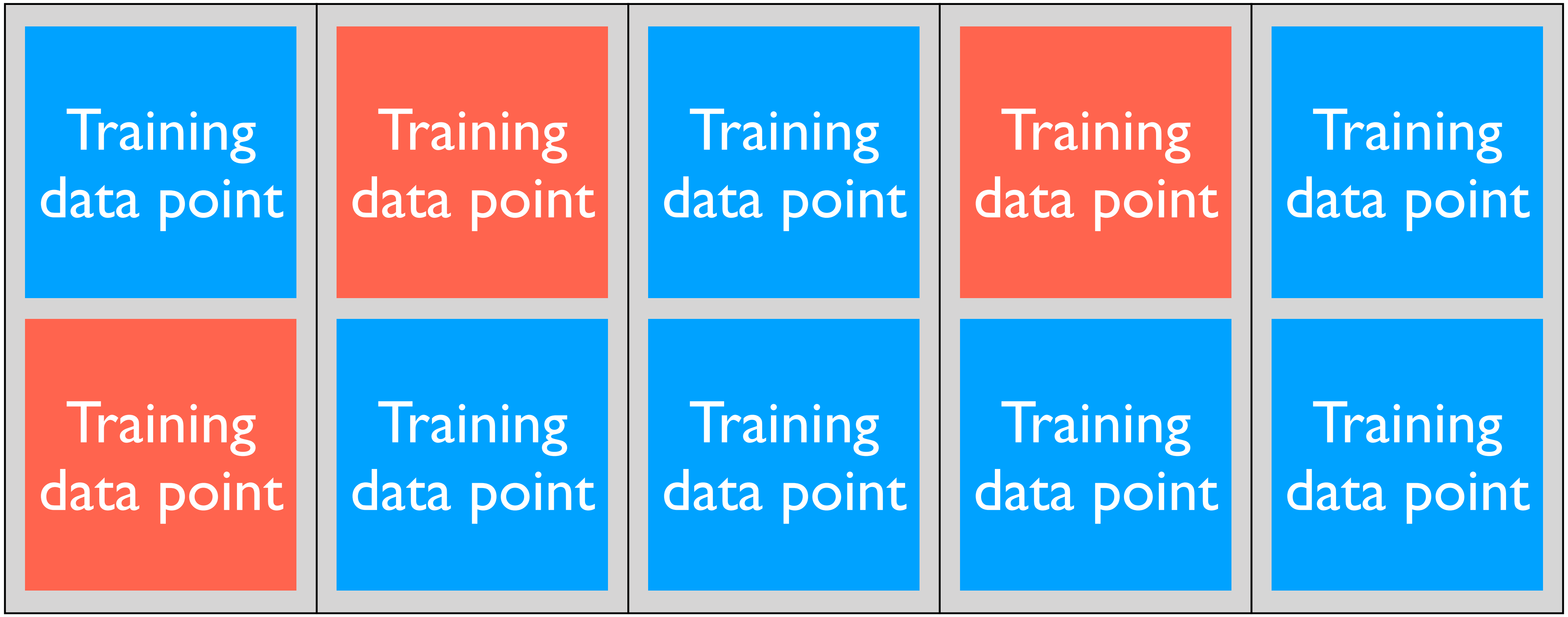
We get 5 different prediction errors…
which is more accurate? 🧐

0% 0% 50% 0% 50%

Unclear which is best, so let's just average: (0+0+50+0+50)/5 = 20%

not the same ***k*** as in ***k***-means or ***k***-NN classification

k-fold Cross-Validation



1. Shuffle data and split them into 5 (roughly) equal size portions

   $k = 5$

2. For each of the equal sized portions:

   (a) Treat the current portion has the validation data and the rest as proper training data

   (b) Train on the proper training data, predict on the validation data

   (c) Compute prediction error

   You need to specify how to measure prediction error!

3. Compute average prediction error

   "cross validation score"

Choosing *k* in *k*-NN Classification

For each $\boldsymbol{k} = 1, 2, 3, \ldots,$ the maximum ***k*** you are willing to try:

Compute 5-fold cross validation score using ***k***-NN classifier as prediction method

Use whichever ***k*** has the best cross validation score