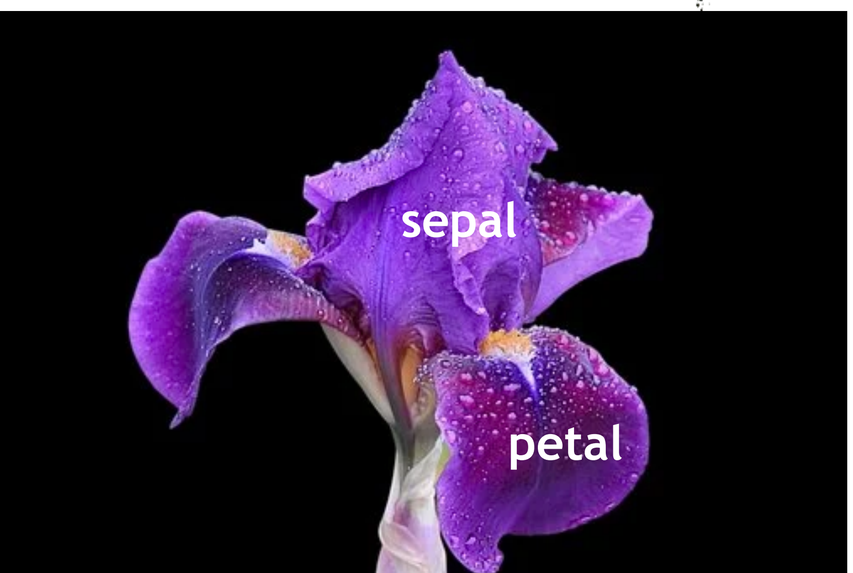
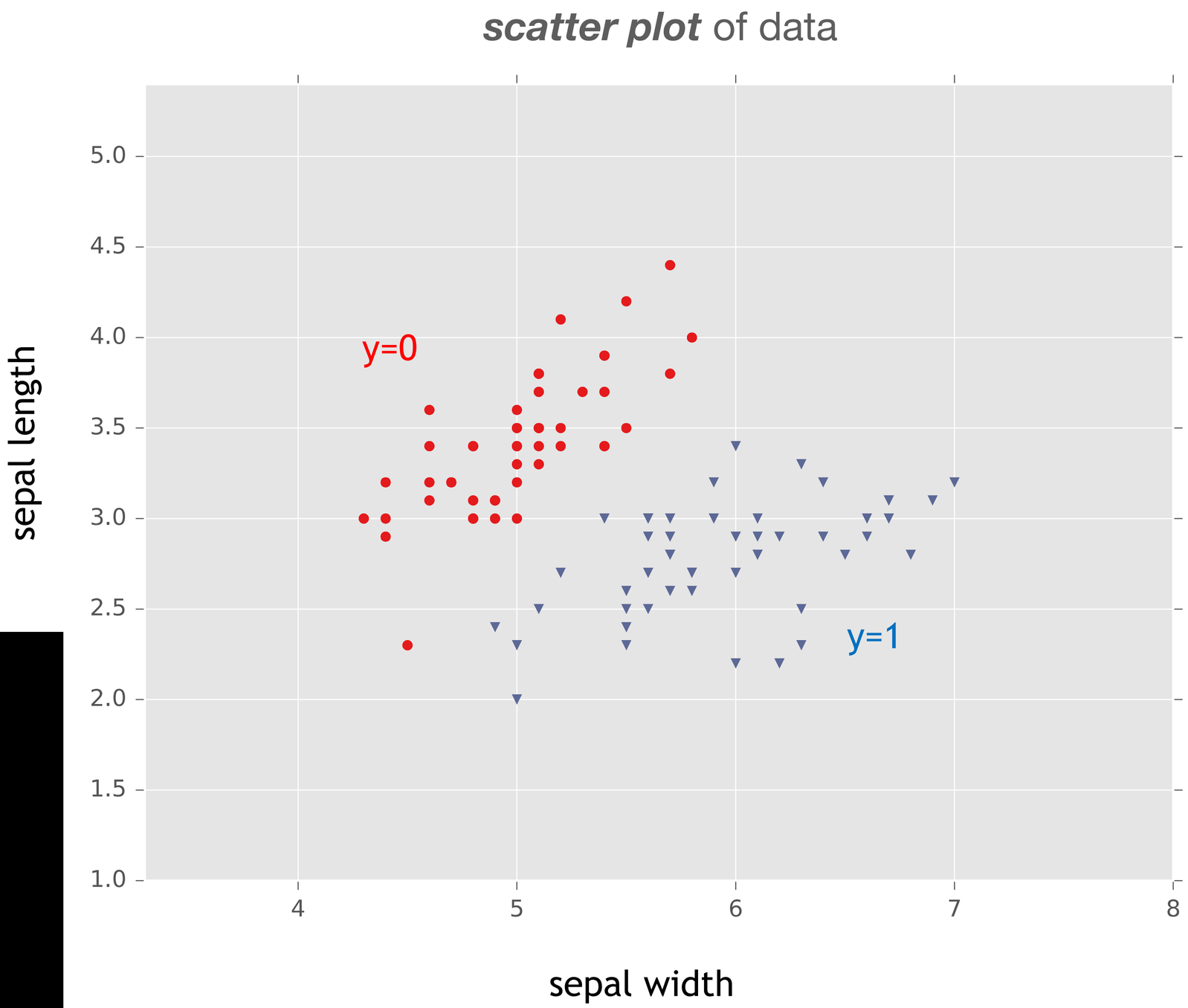
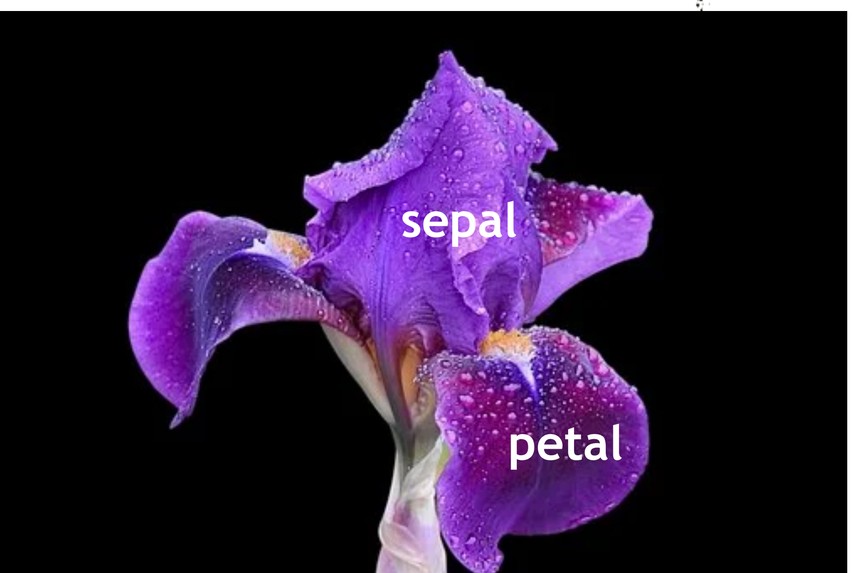
Continuous features

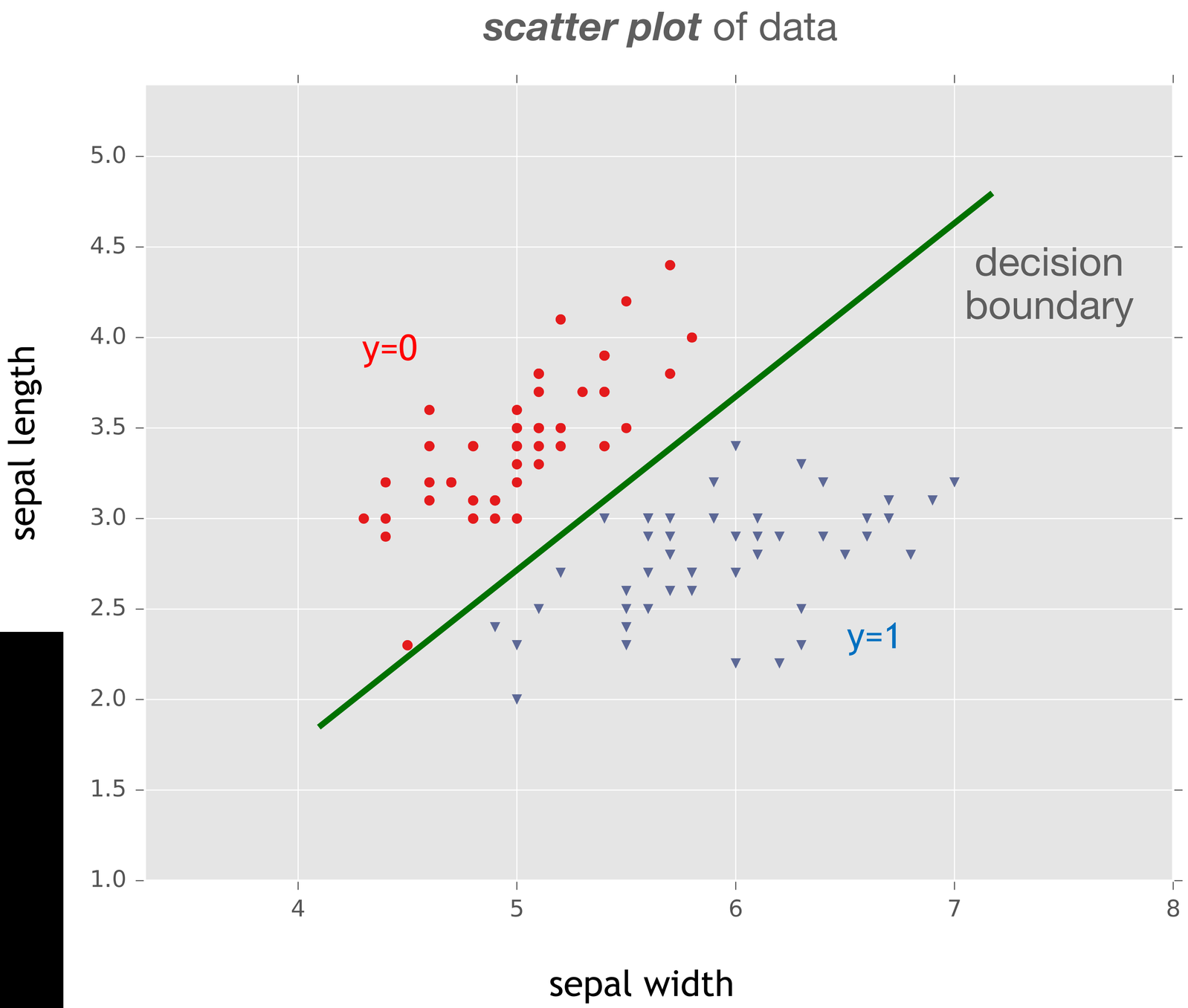




Geoff Gordon

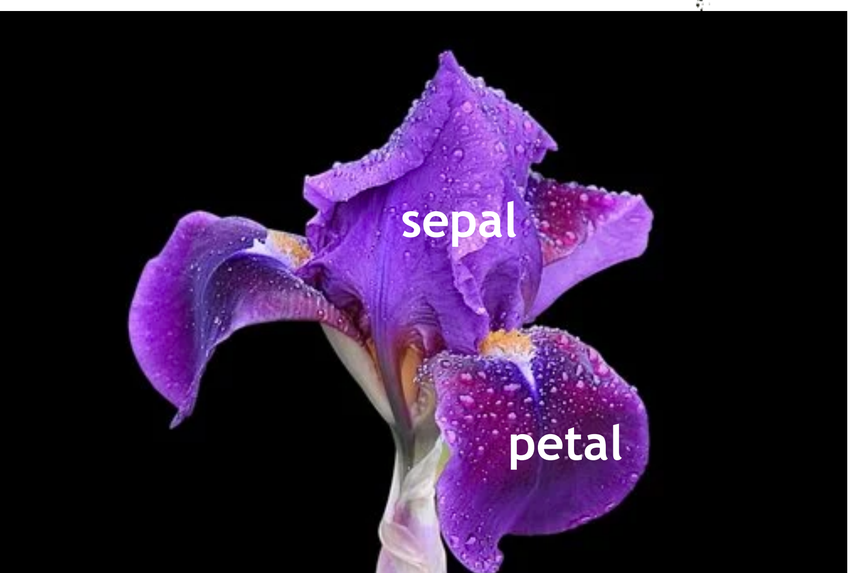
Continuous features

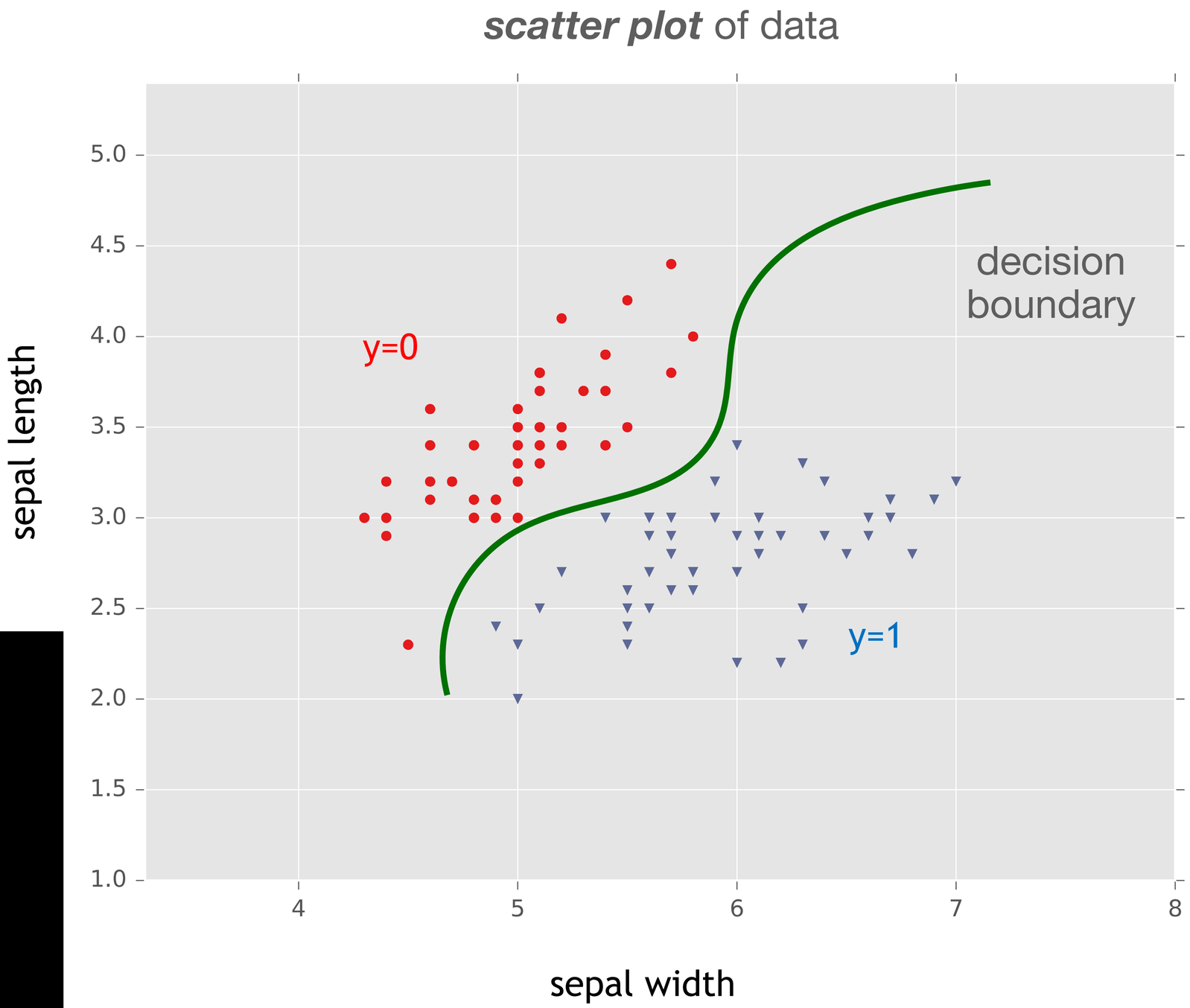




Geoff Gordon

Continuous features





Geoff Gordon

Overfitting

10-701 Introduction to Machine Learning
Geoff Gordon

Overfitting

- For *majority*, train error $\approx$ test error, while for *memorizer*, train error $\ll$ test error
  - we say that *memorizer* suffers from ***overfitting***
- Overfitting = test error – train error
  - more generally, $P - \hat{P}$ (for min) or $\hat{P} - P$ (for max)
  - lots of overfitting means the difference is large
  - nothing to do with P or $\hat{P}$ being large or small on its own
- Heuristic: “simpler” learning methods tend to overfit less
  - we’ll try later to measure simplicity — not simple to do!
  - e.g.: complexity = # nonzero weights in a linear classifier, or complexity = how long we train a deep net

Underfitting

- On the other hand, *majority* doesn't seem like a great learner — not very expressive, high error
- This is called ***underfitting***
  - overfitting is absent or minimal, and another method could achieve better $\hat{P}$ while still limiting overfitting
  - heuristically, we kept the hypothesis class *too* simple

Why does overfitting happen?

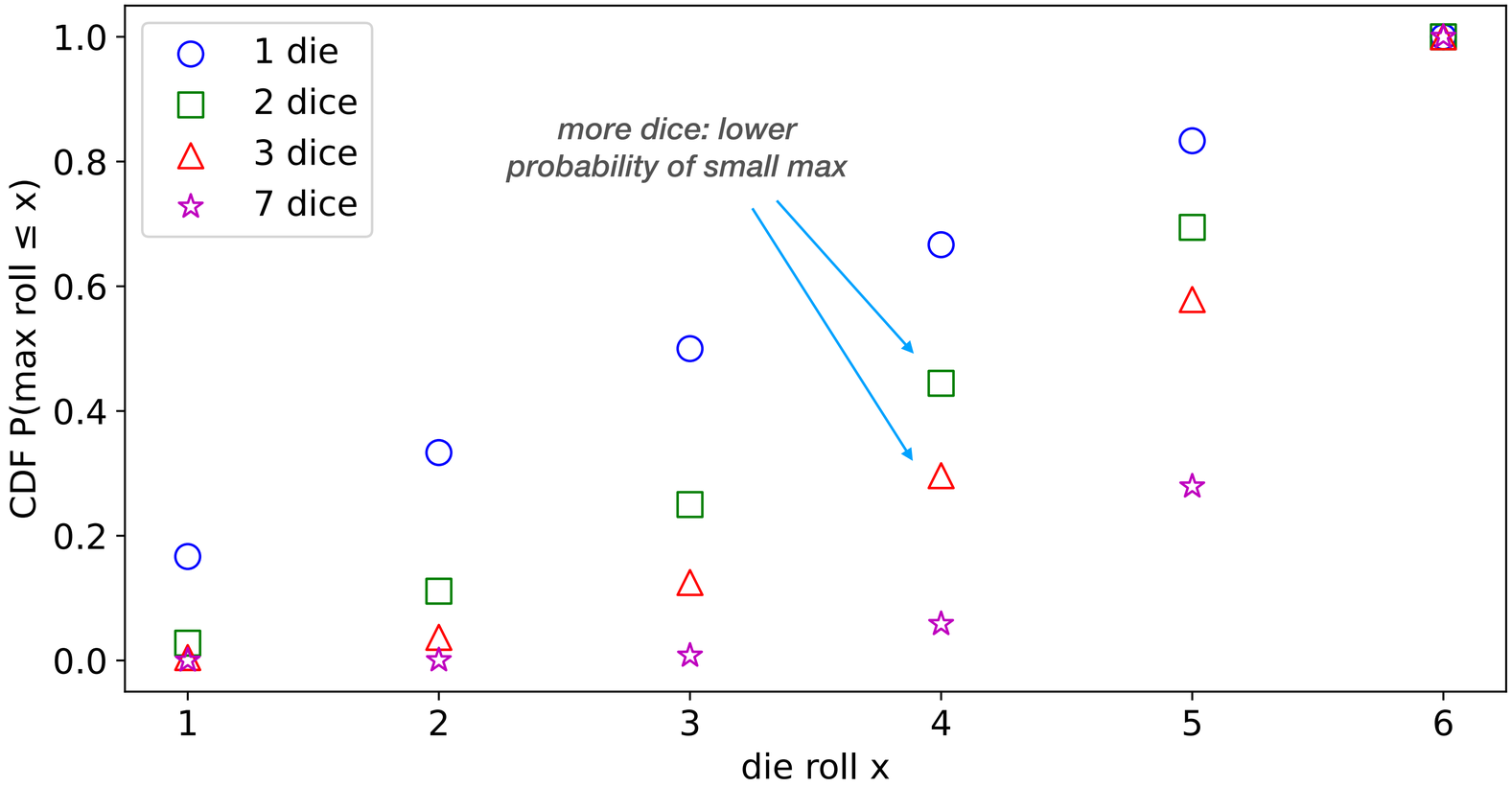
Expected value of each die = 3.5

Why does overfitting happen?

Expected value of highest die =
lower than 3.5?
the same: still 3.5?
higher than 3.5?

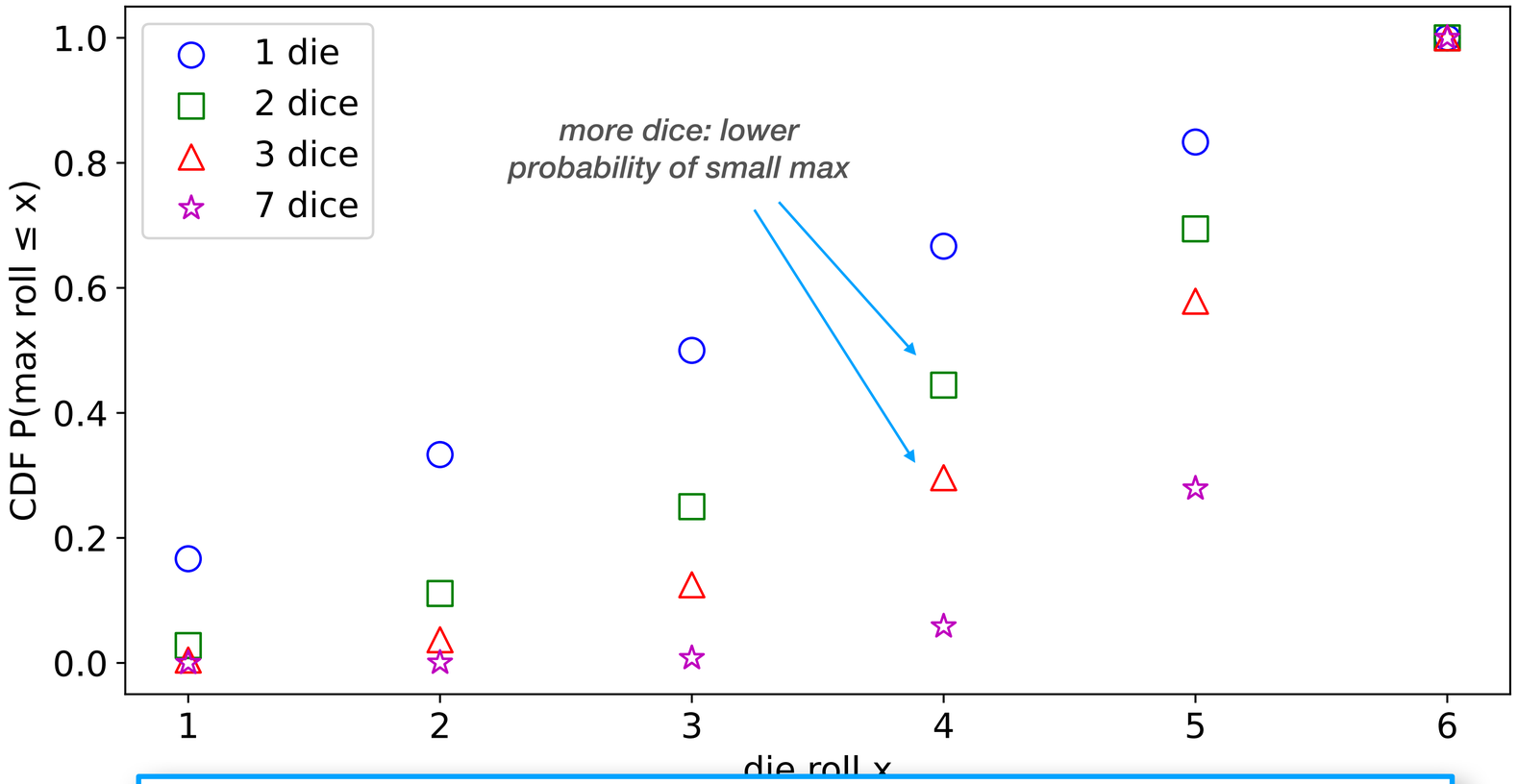
Expected value of each die = 3.5

Max of k dice



- CDF of one die: $i/6$
  - e.g., probability it's ≤ 3 is $\frac{1}{6} + \frac{1}{6} + \frac{1}{6}$
- CDF of max of k dice: $(i/6)^k$
  - probability of (all 3 ≤ i) is $P(\text{die 1} \leq i) \cdot P(\text{die 2} \leq i) \cdot P(\text{die 3} \leq i)$

Max of k dice



- CDF
  - e.g
- CDF
  - pro

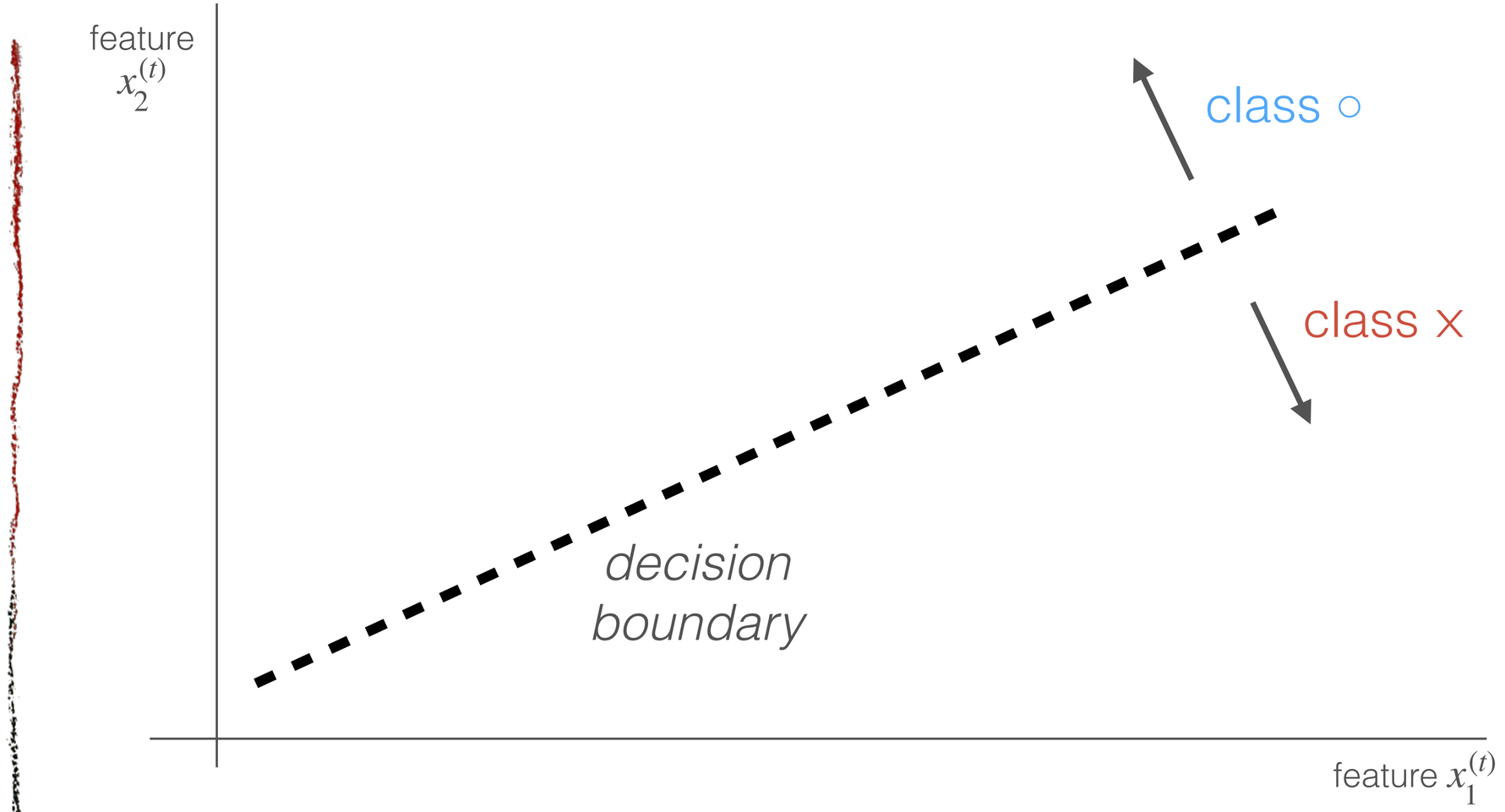
$\leq i)$

Number of dice	Expected value	Difference
1	3.5000	0.0000
2	4.4722	0.9722
3	4.9583	1.4583
4	5.2446	1.7446
5	5.4309	1.9309
6	5.5603	2.0603
7	5.6541	2.1541

What influences the max?

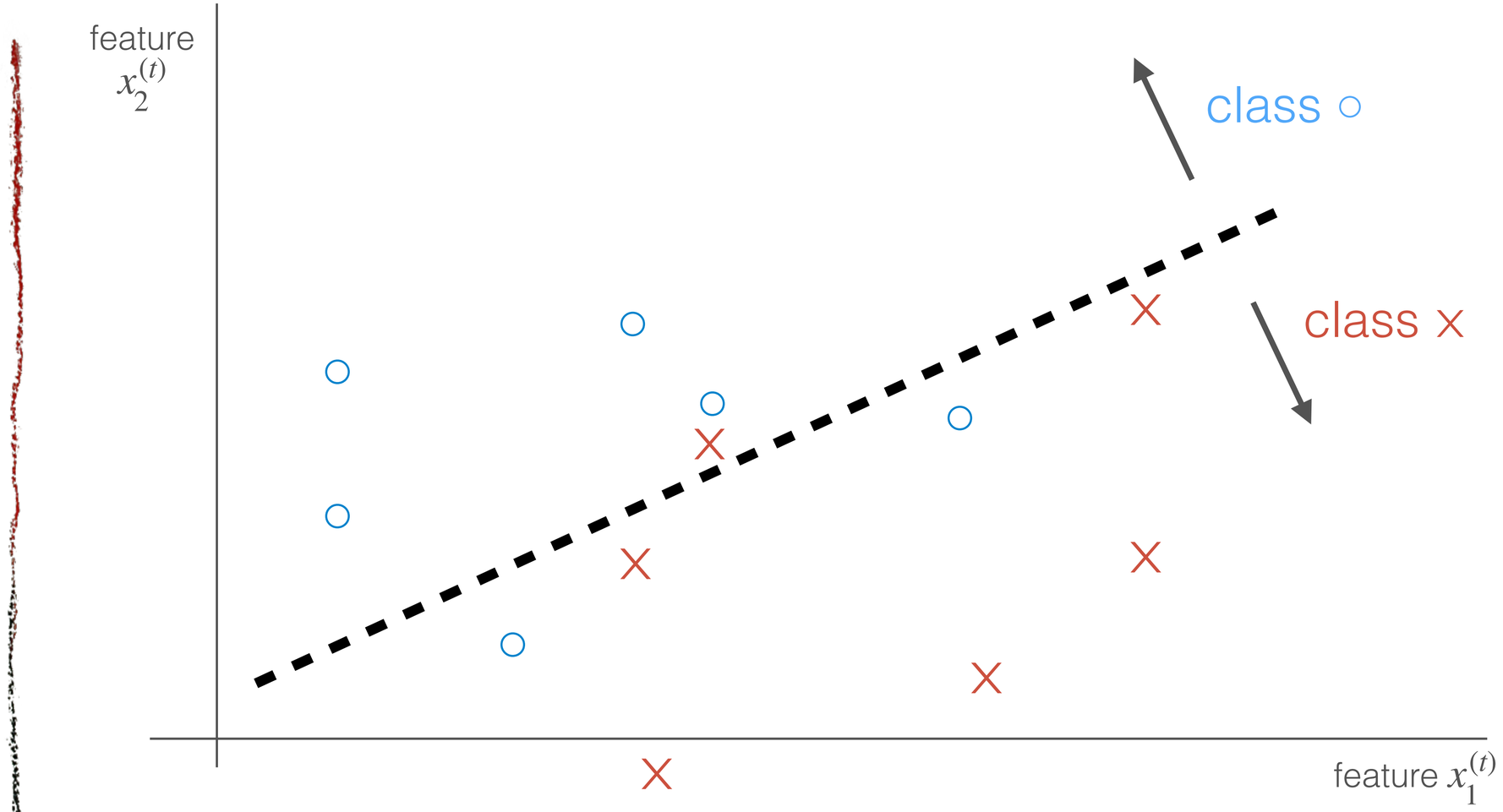
- Just saw: how many dice (more dice = higher)
- What if I add a die that's biased to roll 1 or 2 most of the time?
  -
- What if all the dice are biased so that they roll 1 and 6 much less often?
  - $\mathbb{E}(\text{1 die}) =$
  - $\mathbb{E}(\text{max of } k \text{ dice}) =$

An experiment



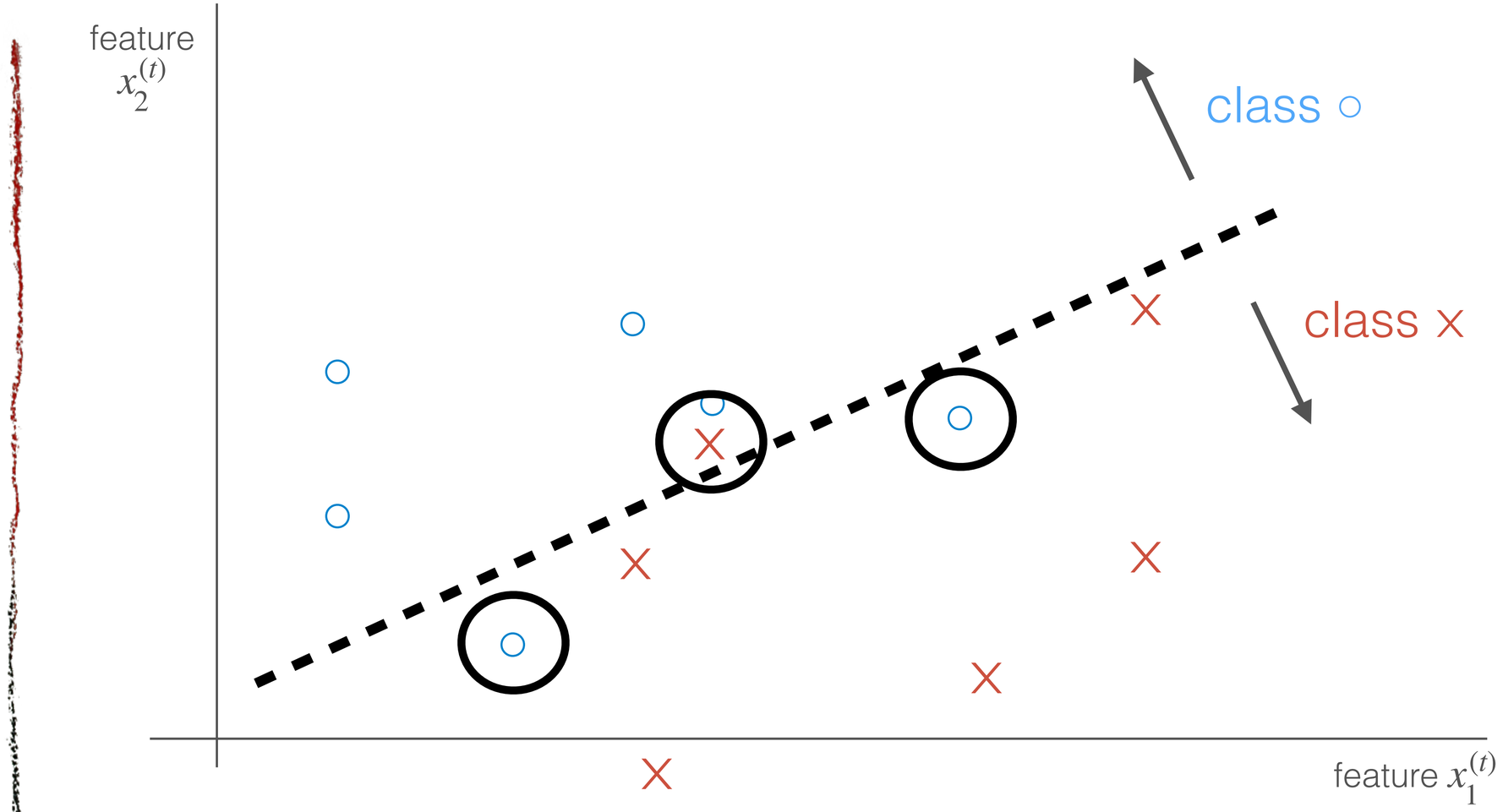
- Fix a model (here: a linear classifier)

An experiment



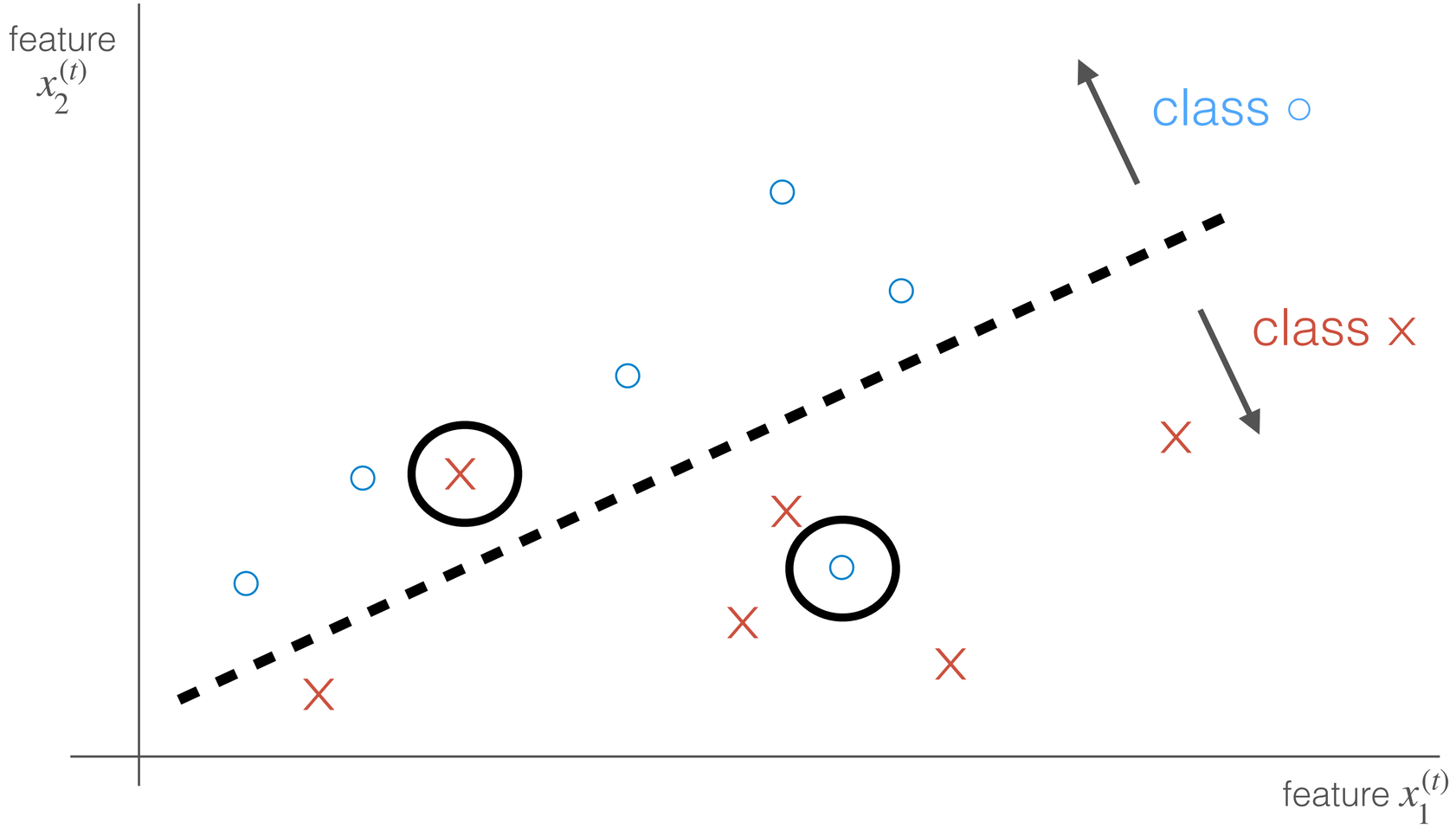
- Sample a dataset at random

An experiment



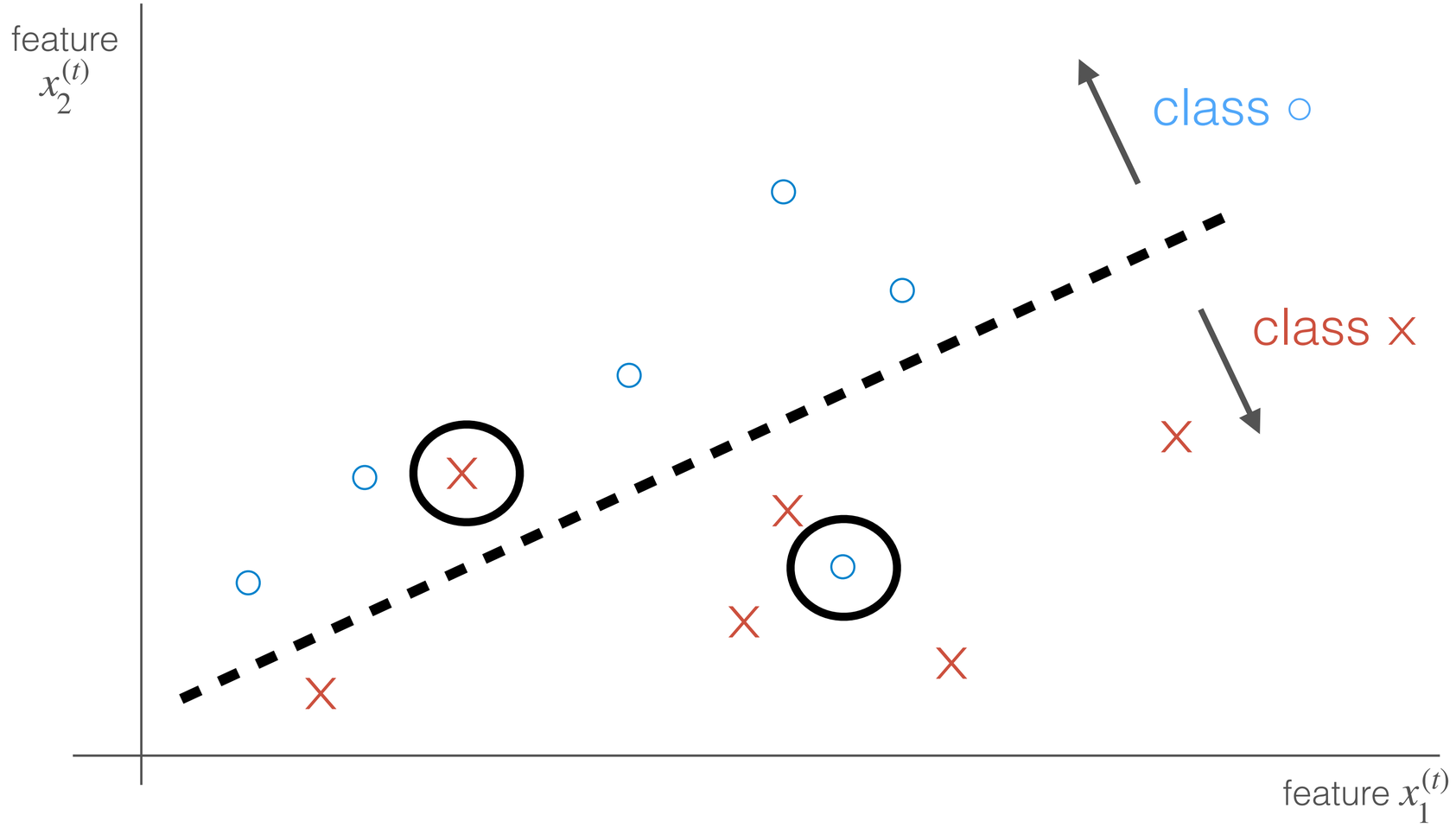
- Measure error rate (here: 3/12, or 25%)

An experiment



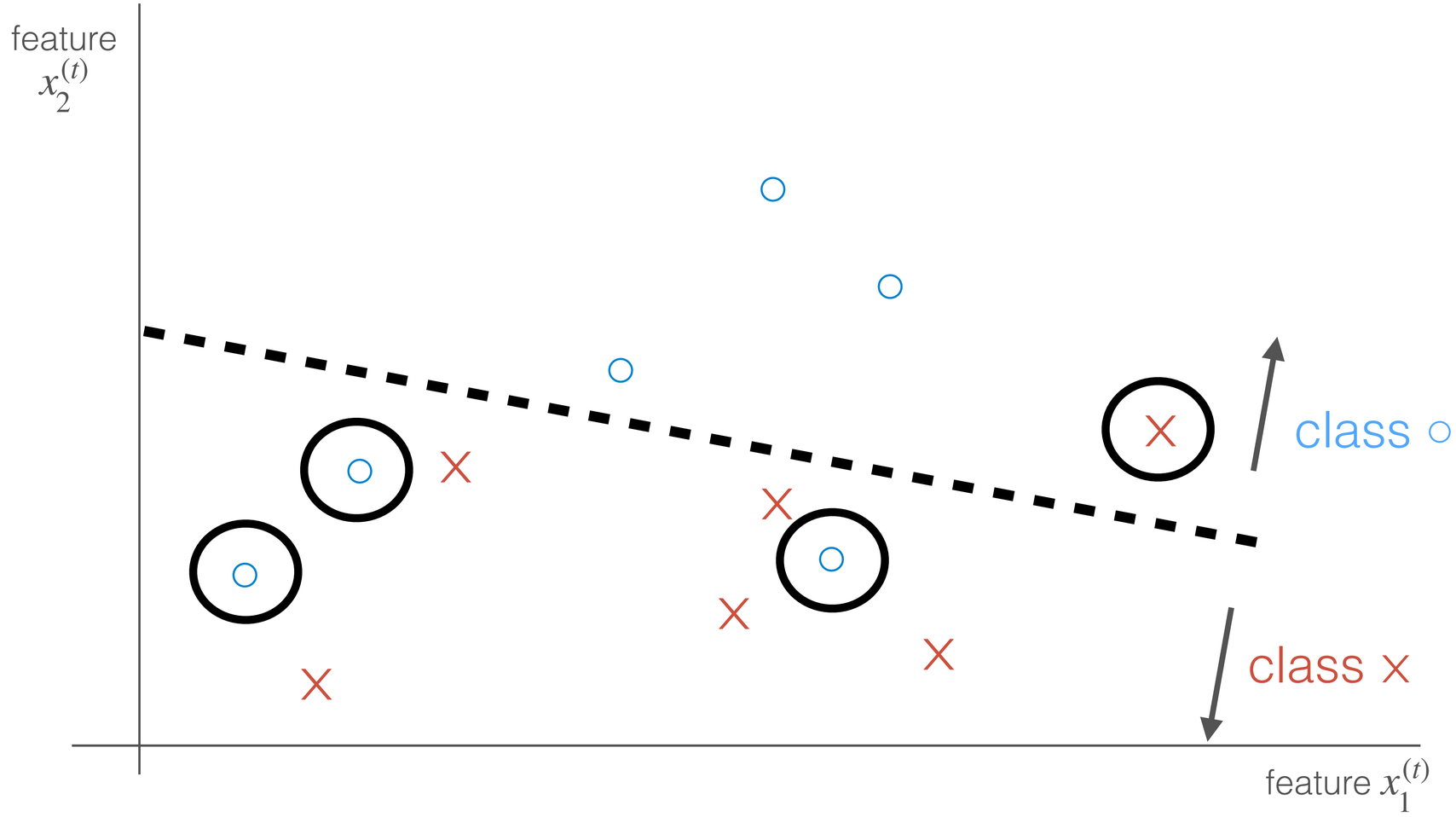
- Result of our experiment is a random variable: different samples can give different error rates (here 2/12 ≈ 17%)

An experiment



- $\mathbb{E}$(error) is called the ***true error rate*** of the classifier
- Variance of measured error depends on the size of the training set

Another experiment



- Evaluate a different classifier on the same dataset: 33%
- Now have two random variables: each time we sample a dataset we can measure error for both classifiers

Now: train a model

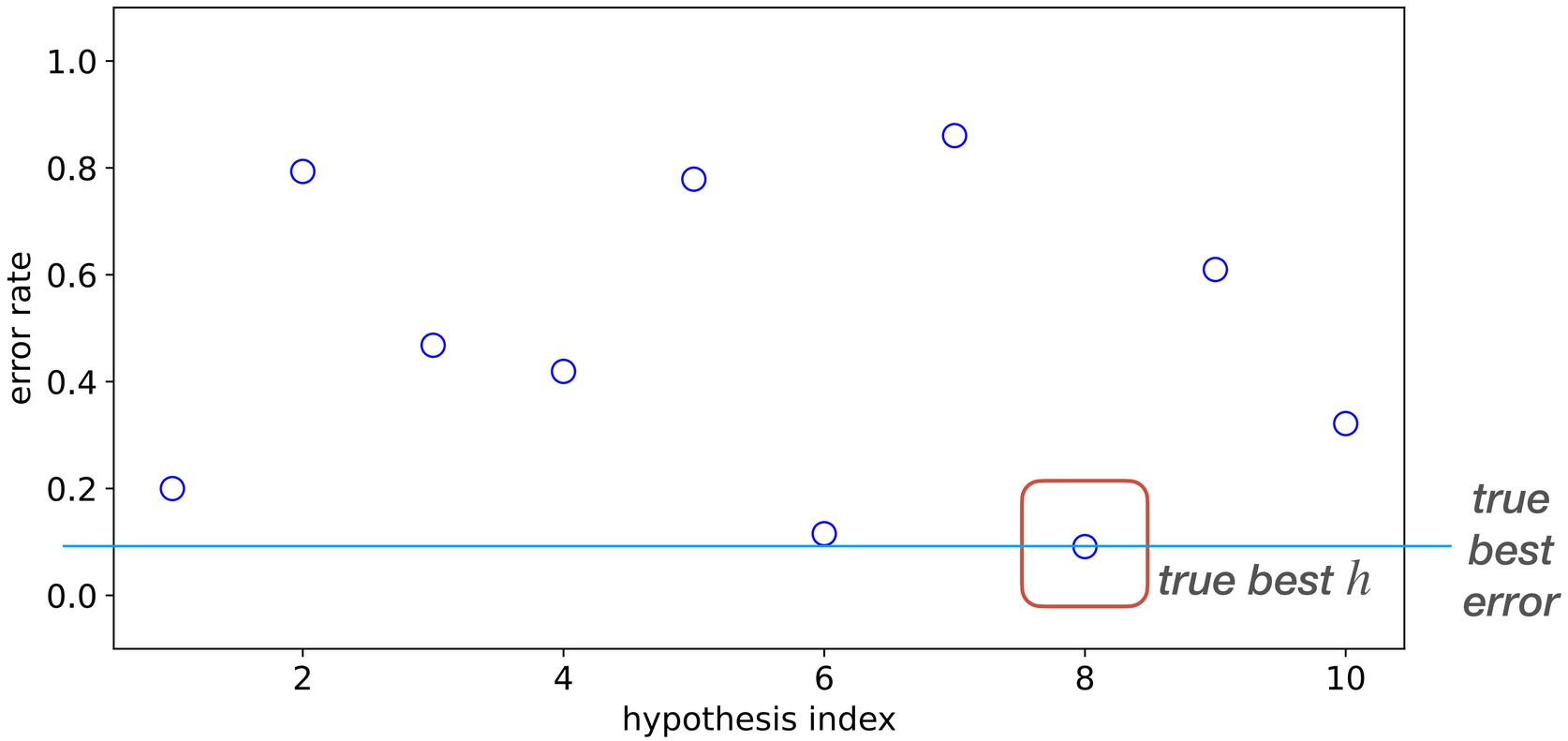
- Sample a dataset at random — call it our training set
- Evaluate many classifiers (every $h \in \mathscr{H}$) on this dataset
- Choose the one that looks best (lowest measured error)

- This should remind you of our previous experiment

Analogy

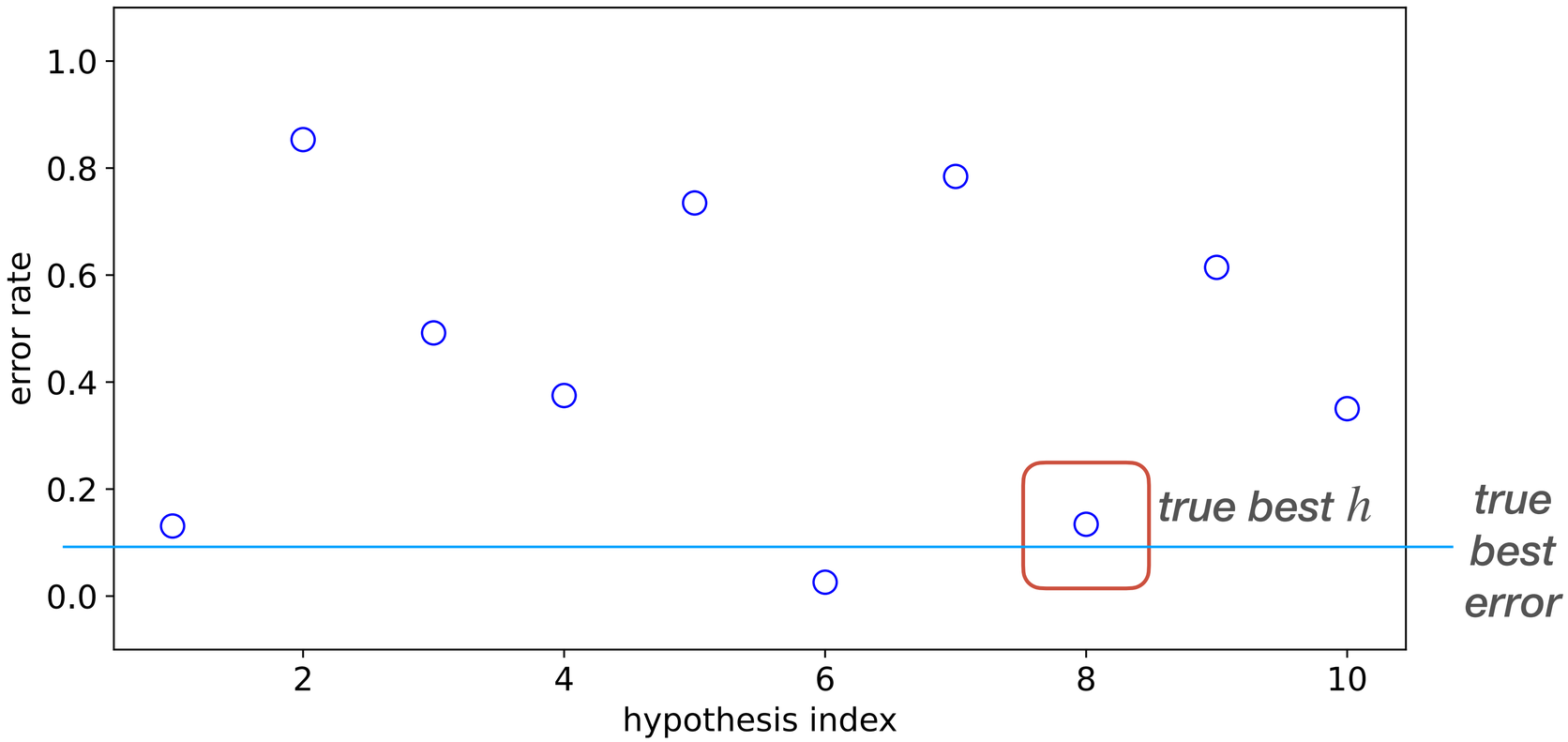
- Evaluating a single model on a random dataset is like rolling a die — a random number, depending on how lucky the model is (how often we select training points that the model happens to do well on)
- Evaluating many models is like rolling many dice: many random numbers, with some models being luckier and some models less lucky
- Training = choosing the apparently-best model from a hypothesis class
  - like taking the max or min of many dice

Visualize the training process



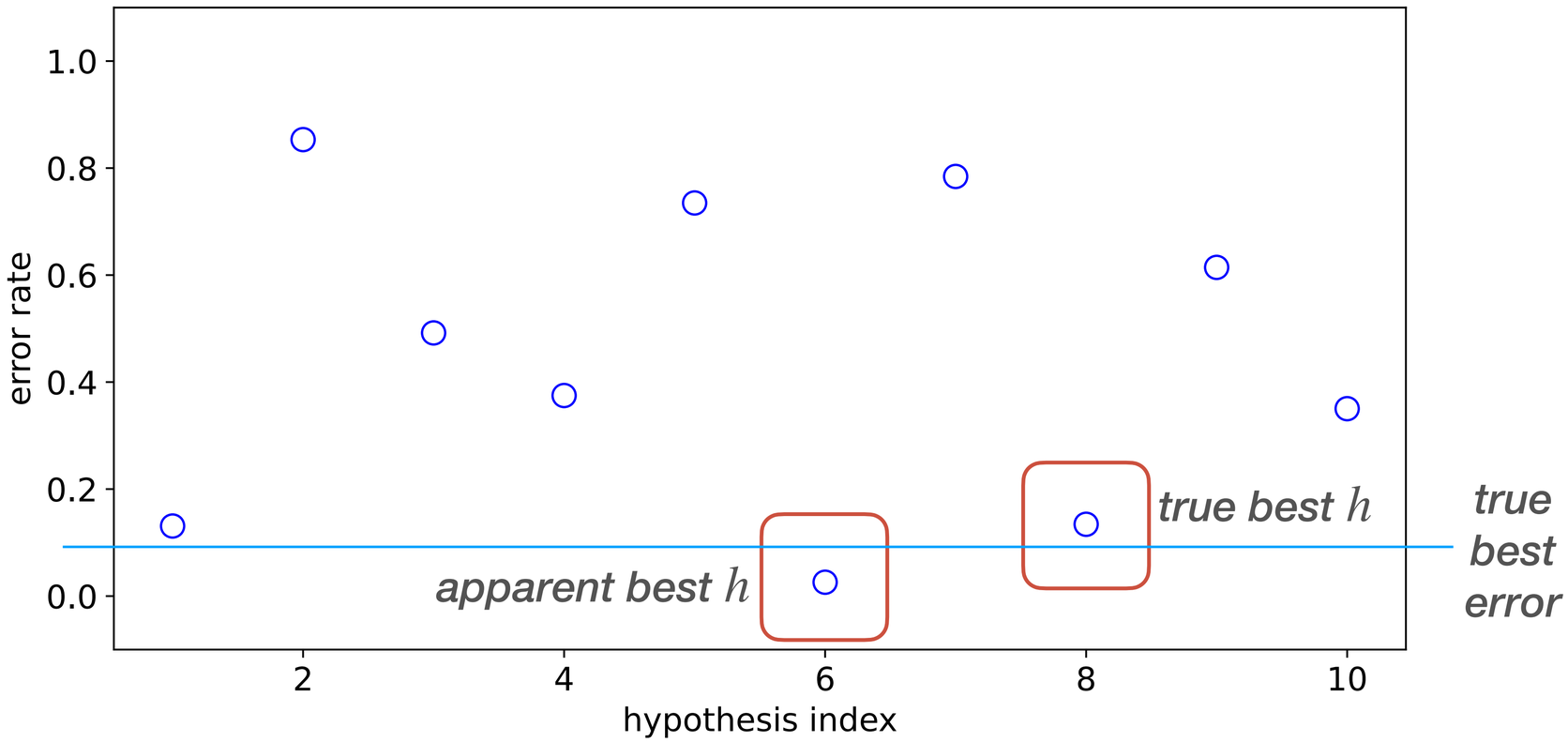
- Suppose we have 10 hypotheses ($|\mathcal{H}| = 10$), each with a different true (unobserved) error

Draw a random training set



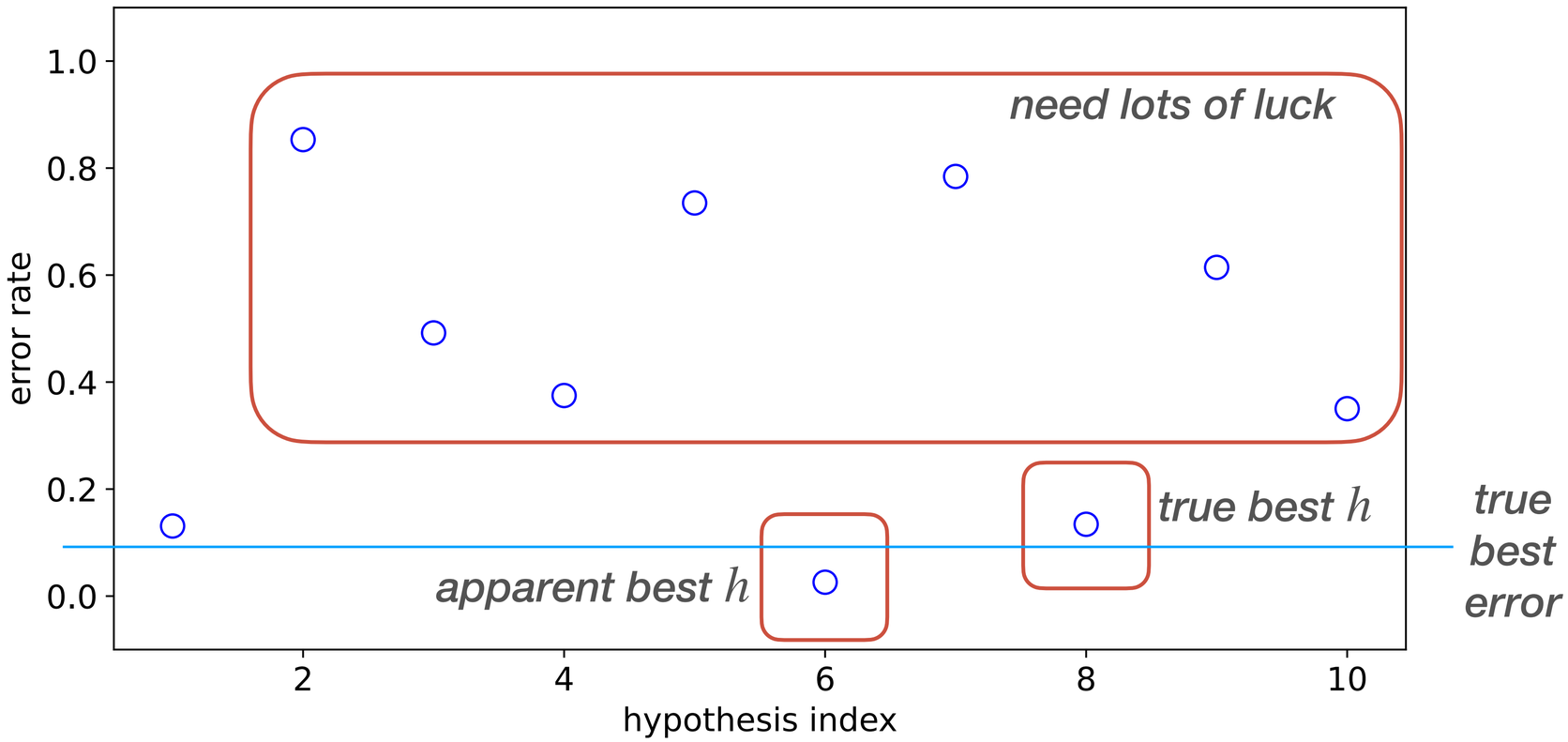
- When we pick a random training set, the measured error for each h is a bit different from its true error — here, best h appears a bit worse than truth

Pick the apparently-best model



- More importantly, training might get the wrong h!
  - and its estimated error is better (lower) than the true best error — overfitting!

Each model contributes differently to overfitting



- Models with bad true error are individually unlikely to be lucky enough to win — but there can be a lot of them, so even small chances can add up

The problem is max

- Any time we optimize on some examples (train a neural network, select hyperparameters, …)
- The apparent accuracy of the winner on those examples will be higher than its true accuracy
  - “winner’s curse,” “selection bias,” “overfitting”
- If we want to know the true accuracy, need fresh examples (a test set)
  - key is that we didn’t optimize on the fresh examples

What influences amount of overfitting?

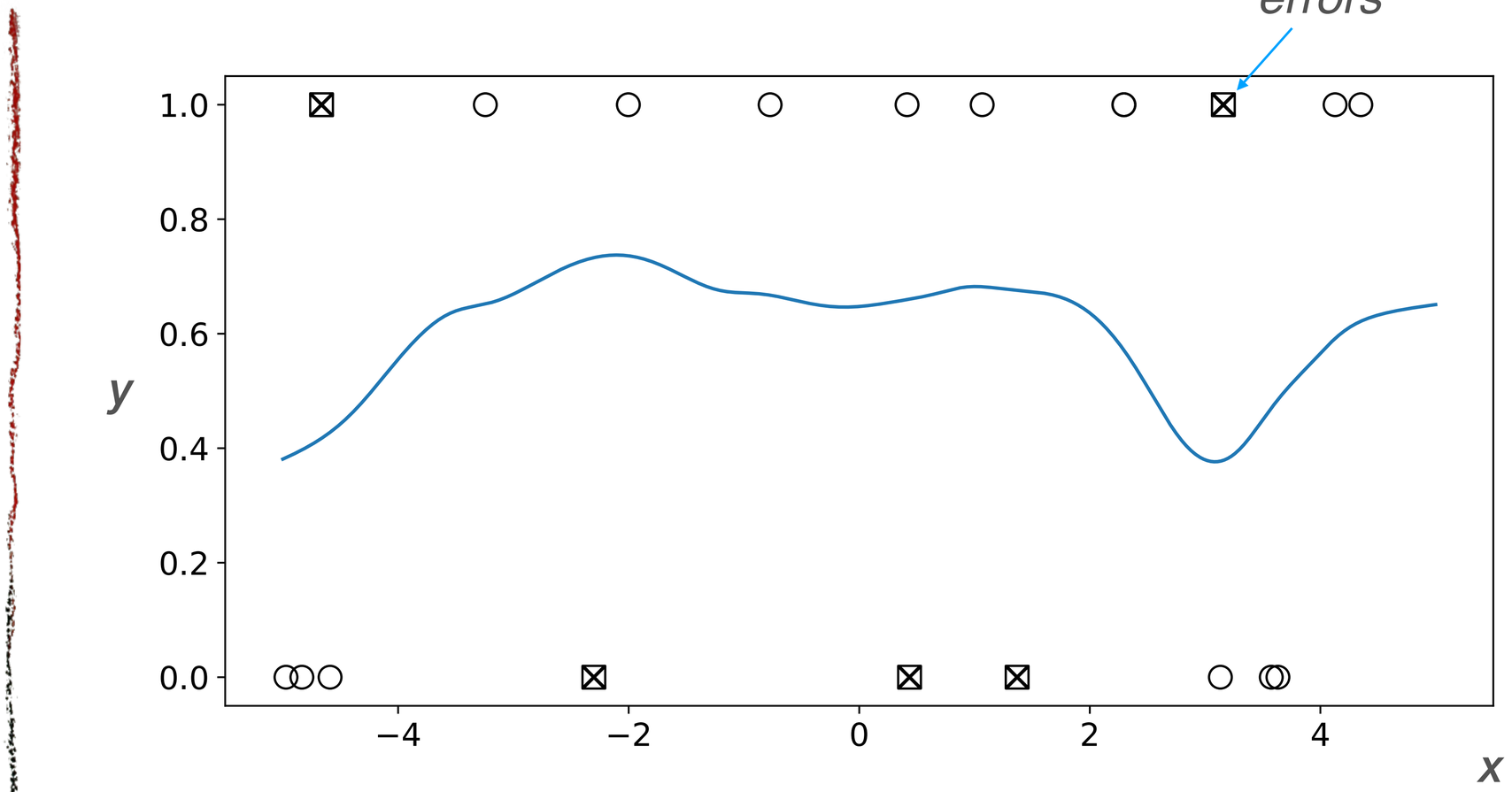
- We can answer by looking at our dice analogy
- We gave three factors that influence dice experiment:
  - number of dice
  - how many of them are close to the true optimum
  - how big is the variance of each die roll
- Similarly, for training a model
  - how many distinct models / complexity of $\mathscr{H}$
  - how many distinct models are near optimal
  - how much error in each model evaluation / how big is our training set, how much variance in each training label

distinct models = (near-)independent errors

Overfitting example

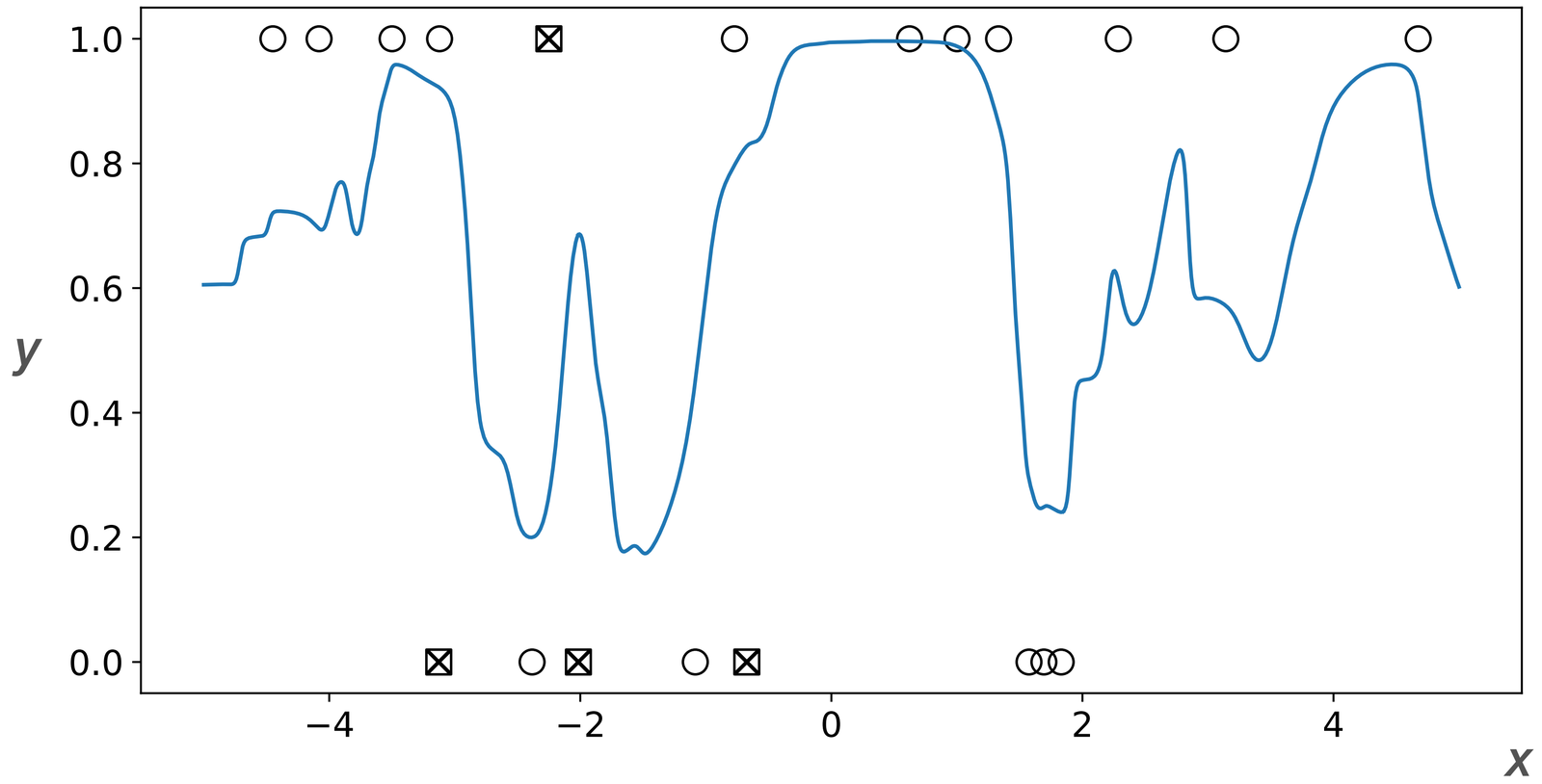
- Fix a training set: 1d classification, $N = 20$ examples, balanced classes (half are class 1, half are class 0)
  - true (unknown) h^*: $p(y^{(t)} = 1) = \frac{1}{2}$, independent of $x^{(t)}$
- Fix a hypothesis class: $H = 5000$ randomly-initialized neural networks: depth 3, width 32, tanh activation
- Train: choose the network in $\mathscr{H}$ with lowest training error (threshold output $p(y^{(t)} \mid x^{(t)}, w)$ at 0.5)
  - note: if we trained the usual way (optimized weights w/ PyTorch, without restricting to a finite set of networks) we would get a perfect fit (0% training error)
  - we are artificially limiting complexity of $\mathscr{H}$, both to limit overfitting and to better match our analysis above

Result



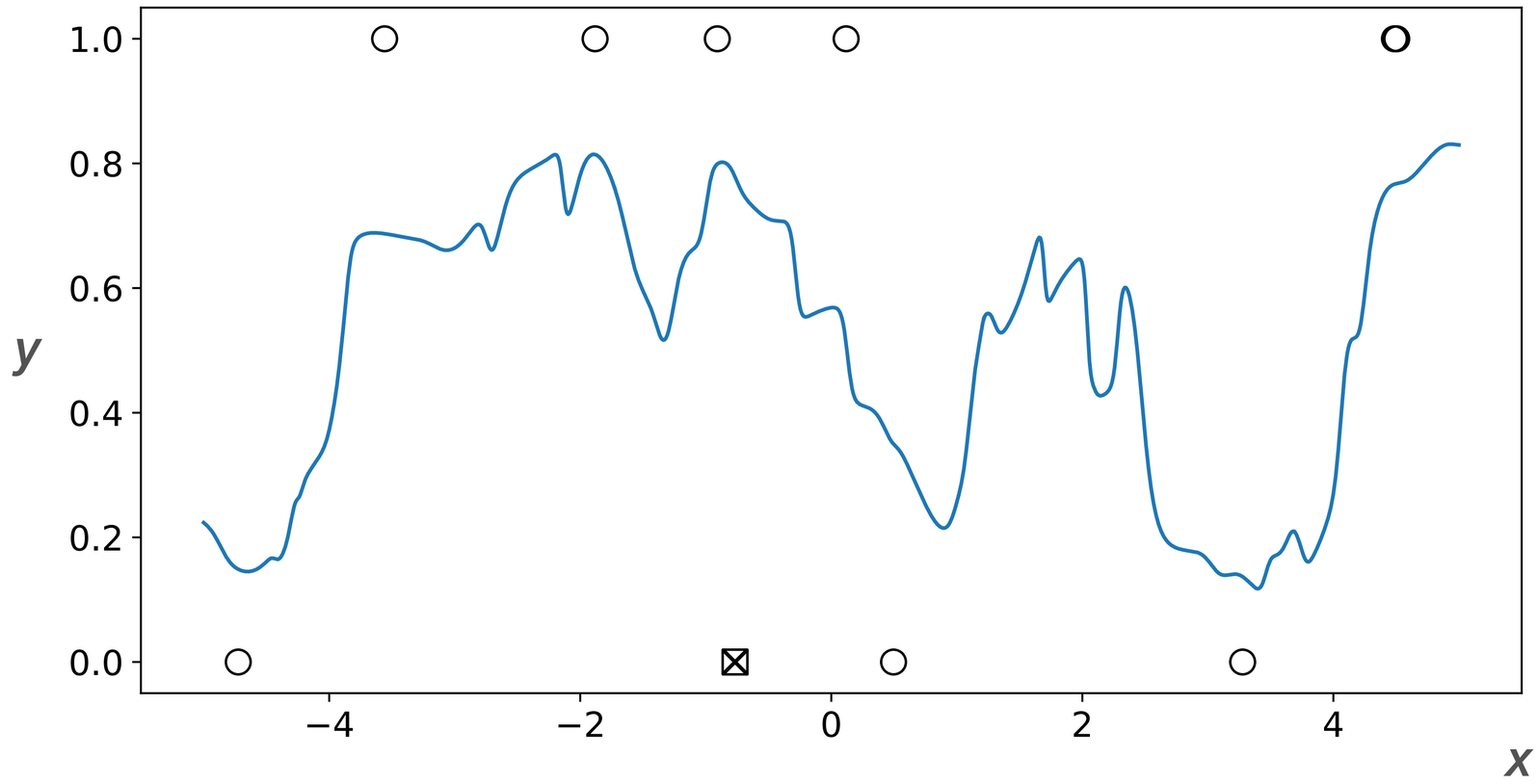
- Training error: 5/20 = 25%
  - cf. true error 50%: overfitting by 25 percentage points

Result



- If we change init parameters to decrease correlation between adjacent predictions, we get more overfitting (here 4/20 = 20% training error)

Result



- If we reduce size of training set (here $N = 10$) for same H, we get even more overfitting (error 1/20 = 5%)

How to limit overfitting?

- Get more training data
- Limit model complexity (effective count of models in $\mathscr{H}$)
  - limit $|\mathscr{H}|$ for a finite model class
  - limit number of nonzero weights in a linear model
  - limit precision of weights (find a group of similar models that all do OK)
  - limit bit-length of model description
  - regularize: limit norm of weight vector
- These don't always work for modern models!
  - for example, often see that increasing number of weights in a deep net will *decrease* overfitting
  - need new measures of simplicity: large margin, low sharpness, few training iterations — active research area

Learning objectives — overfitting

- What is overfitting? What is underfitting? How can we recognize them?
- What are training error and true error, and how do they differ?
- Why does overfitting happen?
- How does $\mathbb{E}(\max\{\text{some r.v.s}\})$ compare to $\max(\mathbb{E}(\text{some r.v.s}))$?
- What factors influence the amount of overfitting?
- What can we do to limit overfitting?

Information

10-701 Introduction to Machine Learning
Geoff Gordon

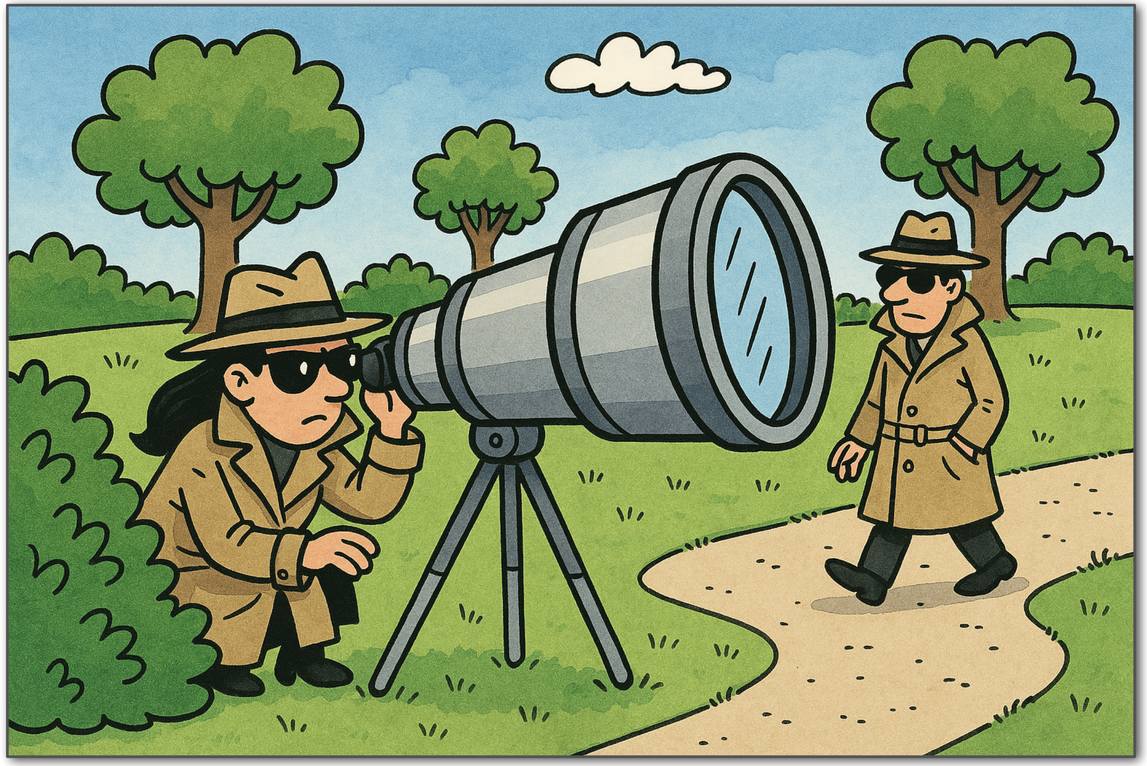
Complexity

$\approx$

information

- We want to limit complexity of our learner
- One general way to measure complexity is ***information***
  - a simple learner is one that only needs a small number of bits to specify
  - e.g., a linear classifier with only a few nonzero weights, and where each weight needs only low precision
- Of course we don't just want the learner to be simple — also want it to predict stuff
- So, can think of learning as ***compression****:* communicate lots of bits about our data (accurate labels for many examples) using only a few bits of hypothesis

Communication channel



- Every morning, Bob finds out a new secret fact: he discovers what the prime minister had for breakfast
- Immediately, he walks in the park, sits at a specific bench, and arranges the rocks next to it in one of four patterns
- Alice observes through a telescope and learns Bob’s message. She informs her superiors.

Code book

meaning	code word	
pancakes	00	
shredded wheat	10	
bagels	01	
caviar	11	

- This channel has a *rate* of 2 bits per day
  - we could speed it up by having more arrangements of rocks, having Bob take multiple walks per day, etc.
- Whatever the rate is, we want to use it efficiently: send as much information as possible per bit
  - for this purpose, A&B set up a *code book* ahead of time, like the one above for PM's breakfast items

Uniform code book

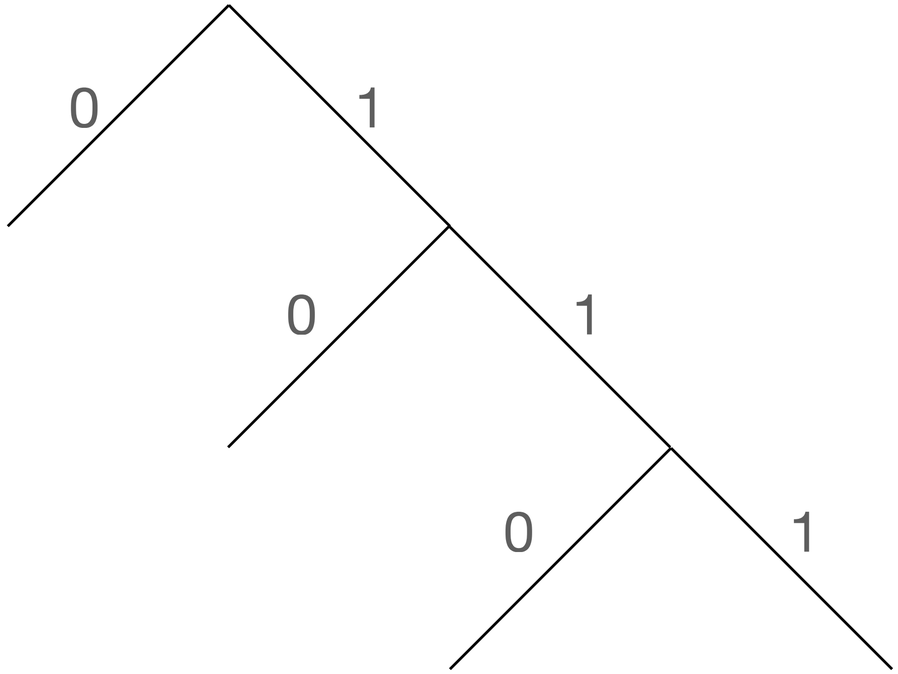
- In this book, all code words are the same length: if there are N possible breakfasts, each is $\lceil \log_2 N \rceil$ bits
- This is optimal in the worst case, which turns out to be when A&B expect all messages to occur equally often (uniform distribution)
- If some events are more common, we might want to send shorter messages for them

Nonuniform code book

meaning	code word	probability
pancakes	0	$\frac{1}{2}$
shredded wheat	10	$\frac{1}{4}$
bagels	110	$\frac{1}{8}$
caviar	111	$\frac{1}{8}$

- An example nonuniform code book (note *prefix property*)

Prefix property



Can guarantee no codeword is a prefix of another by making them paths to leaves in a binary tree

If prefix property failed (e.g., if 11 were a word), we'd be stuck after seeing 11: is this the end or do we have to wait for one more bit?

Nonuniform code book

meaning	code word	probability
pancakes	0	$\frac{1}{2}$
shredded wheat	10	$\frac{1}{4}$
bagels	110	$\frac{1}{8}$
caviar	111	$\frac{1}{8}$

- An example nonuniform code book (note *prefix property*)
- Expected message length = $p_1 b_1 + p_2 b_2 + \ldots + p_N b_N$: here $1\frac{3}{4}$
  - compression! this is what a good codebook buys us
- But if we were wrong and outcomes really were uniform, expected length would be $(1 + 2 + 3 + 3)/4 = 2\frac{1}{4}$

Shannon bound

e.g., if $p_i = \frac{1}{4}$, need 2 bits

- Lower bound (Shannon): best code uses $\geq \log_2 \frac{1}{p_i}$ bits on average to send a message whose probability is p_i
  - in fact we can achieve exactly this value in the long run, but have to be slightly cleverer than a fixed code book
  - or hope that all p_i are integer powers of 2
- Example of cleverer method: arithmetic coding

Entropy

- Sum over all possible messages: expected message length is $-\sum_i p_i \log_2 p_i$, the entropy of distribution p
  - this is binary entropy; if we send digits instead of bits we get base-10 entropy (using $\log_{10}$)
  - also common to report *nats* (using $\ln$)
  - differ by constant factor: more information per symbol $\leftrightarrow$ fewer symbols sent
  - e.g., digits vs. bits: factor $\log_2 10 \approx 3.32$ fewer symbols (just like writing a number in decimal vs. binary needs fewer symbols: 345 vs. 101011001, 3 digits vs. 9 bits)
- Entropy = how many bits to communicate one sample of a random variable

Wrong codebook

- Using a codebook based on q when true distribution is p costs $-\sum_i p_i \log_2 q_i$
  - messages are $-\log_2 q_i$ bits instead of $-\log_2 p_i$ bits
  - expectation over true distribution p
  - e.g., A&B set codebook before knowing true $p(\text{breakfast})$, can't meet up to fix it
- Called the ***cross entropy*** between p and q
  - cost of a codebook based on wrong distribution

Kullback-Leibler divergence

- Difference: an extra $\sum_i p_i \log_2 \frac{p_i}{q_i}$ bits
  - always ≥ 0, equality iff $q = p$
  - called $KL(p||q)$, the Kullback-Leibler divergence or relative entropy between p and q
- KL = price (# extra bits) for inaccurate information

Multiple random variables

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

X = breakfast,
Y = side of bed

- Alice and Bob might want to talk about more than one random variable — here breakfast & which side of bed
- Table shows ***joint*** distribution of X, Y

Multiple random variables — marginal

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

X = breakfast,
Y = side of bed

- We can extract ***marginal*** distribution of X or Y, like what we were working with before:
  - e.g., $P(Y = \text{right}) =$ $P(X = \text{pancakes}) =$

Multiple random variables — conditional

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$X =$ breakfast,
$Y =$ side of bed

- Or, if we find out something (e.g., $Y =$ left), can focus down to ***conditional*** distribution of remaining variables
  - e.g., $P(X =$ pancakes $\mid Y =$ left$) =$

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

X = breakfast,
Y = side of bed

- Suppose A&B both know the joint distribution above, and B wants to tell A what the PM had for breakfast
  - in expectation needs entropy(q) $\approx$ 0.918 bits where q $= \left[\frac{2}{3}, \frac{1}{3}\right]$ (the marginal of X)

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$\left.\vphantom{\frac{1}{2}}\right\}$ $p = \left[\frac{3}{4}, \frac{1}{4}\right]$

- What if B knows that PM got out of bed on left?
  - true distribution of breakfast is now p = conditional of X given left — but A doesn't know this, so B still has to use codebook for marginal q
  - pay an extra $D_{KL}(p||q) \approx 0.024$ bits on average
  - alternate way to say it: if Bob can tell Alice that PM got out of bed on left, she will gain 0.024 bits of information about breakfast (need 0.024-bit shorter message)

Information gain

X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

$\left.\vphantom{\frac{1}{6}}\right\}\ p' = \left[\frac{1}{2}, \frac{1}{2}\right]$

- What if B knows that PM got out of bed on right?
  - true distribution is now p'; pay an extra $D_{KL}(p' \| q) \approx 0.085$ bits from having to use q's codebook
  - alternate way to say it: if Bob tells Alice that PM got out of bed on right, she gains 0.085 bits of information

Mutual information

MI = how much does X tell us about Y (or vice versa)

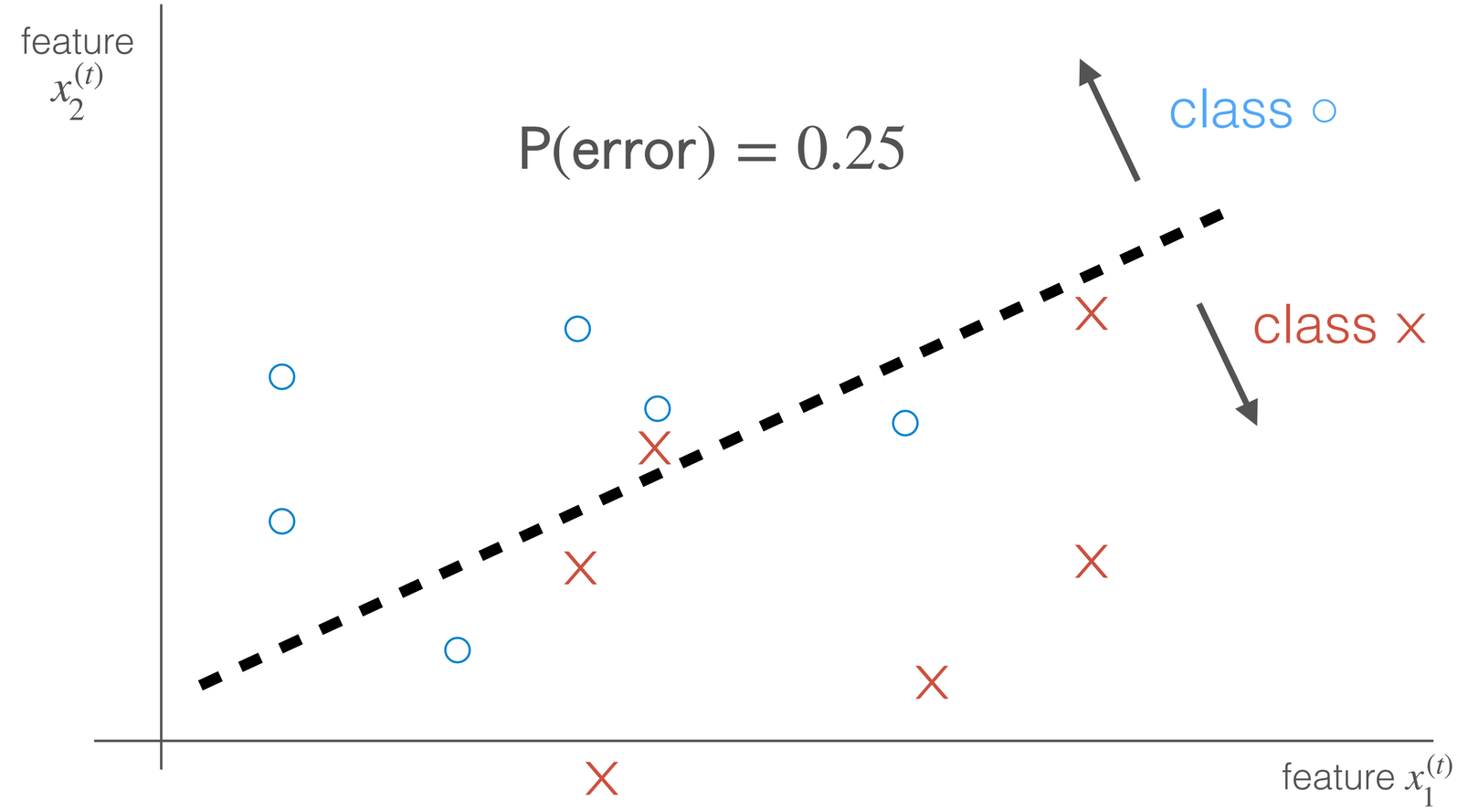
X	Y	p_{XY}
pancakes	left	$\frac{1}{2}$
shredded wheat	left	$\frac{1}{6}$
pancakes	right	$\frac{1}{6}$
shredded wheat	right	$\frac{1}{6}$

- On average, if B tells A left or right each day, A gains 0.024 bits on $\frac{2}{3}$ of days, and 0.085 bits on $\frac{1}{3}$ of days
- These 0.044 bits are the expected information gain from finding out side-of-bed — also called the *mutual information* between side-of-bed and breakfast
- Hard to see from this expression, but MI is symmetric between X and Y

Connect back to learning

- Setup:
  - both Alice and Bob know $x^{(t)}$ for N examples
  - but only Bob knows $y^{(t)}$ — wants to send labels to Alice
- Naive strategy:
  - Bob sends $y^{(t)}$ for all examples, using 1 bit per example
- Possibly better strategy:
  - Bob learns a model $h \in \mathscr{H}$ (e.g., weight w in $f_w(x^{(t)})$)
  - first sends h to Alice, then uses h to help send $y^{(t)}$
- A good hypothesis is one that communicates lots of bits about our data using only a few bits of hypothesis
  - pay once up front, save on each new example

Using hypothesis to help send labels



- Encode the errors instead of the labels
  - Bob sends training error probability ϵ as part of h
  - coding cost is entropy(ϵ) per label — can be much less than naive cost of 1 bit

Bound

preview of upcoming lectures on learning theory

- Can even make a bound: whatever compression we see for our training set, we're likely not to do much worse on unseen future test examples
- Whp, coding cost per example on training set satisfies

cost under true error rate	≤	observed cost	+	terms that shrink w/ N

- Observed cost per example uses observed training error rate, and includes (amortized) cost of sending h
- LHS grows w/ true error, so this bound rules out true error being too much larger than observed
- More sophisticated versions of this result actually give nontrivial generalization bounds for modern LLMs — still an active research area

Learning as compression

- Weird but fairly decent learning algorithm: `gzip`
- Training:
  - collect positive-class examples into `pos.txt`
  - collect negative-class examples into `neg.txt`
- Prediction for a new example `test_ex.txt`:
  - get file size of `cat pos.txt test_ex.txt | gzip` minus file size of `gzip pos.txt`
  - similarly, size of `cat neg.txt test_ex.txt | gzip` minus size of `gzip neg.txt`
  - pick the class with the ***smaller*** increment in file size
- Learned hypothesis = internal data structures of `gzip` after processing `pos.txt` and `neg.txt`

Learning objectives—information

- Why is measuring information important to ML?
- What are joint, marginal, and conditional distributions?
- What is the Shannon bound, and what does it tell us about codebooks / message length?
- What happens if we encode a message based on the wrong probability distribution?
- What are entropy, cross entropy, and relative entropy? What is mutual information?
- How does learning relate to communication and compression?