Social Data Science: Econometrics and Machine Learning

Week 2

Tree and kernel based methods

Social Data Science: Econometrics and Machine Learning

Week 2

~~Tree and kernel based methods~~

A tour of supervised machine learning (shallow learning)

Overview

1. Ensemble learning
2. Bagging, boosting
3. Decision trees
4. Random forests
5. K-nearest neighbor
6. Gradient boosting

Ensemble Learning

Ensemble Learning

Gist:

- Create and train many classification models
- Treat each model as a "voter"
- For each datapoint, classify it according to what models predicts it to be

Ensemble Learning

predictions of a single model can be highly sensitive to noise, but the average of many models is not

Gist:

- Create and train many classification models
- Treat each model as a "voter"
- For each datapoint, classify it according to what models predicts it to be

Pros:

1. Better generalization performance
2. Lowers overall error
3. Robust to overfitting

Cons:

1. Takes a little longer to train…

Bagging and boosting

Bagging and boosting

> Important concept: **bootstrapping**

Algorithm:

1. Given a list of length N, randomly select N elements *with* replacement

Bagging and boosting

> Important concept: **bootstrapping**

Algorithm:

1. Given a list of length N, randomly select N elements *with* replacement

```
In [10]: import numpy as np

         mylist = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
         np.random.choice(mylist, size=len(mylist))
```

executed in 6ms, finished 07:11:28 2020-02-14

```
Out[10]: array([3, 8, 9, 3, 9, 2, 3, 0, 0, 3])
```

Bagging

> A machine learning strategy for ensemble learning

Algorithm:

Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a **random sample with replacement** of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

Bagging

> A machine learning strategy for ensemble learning

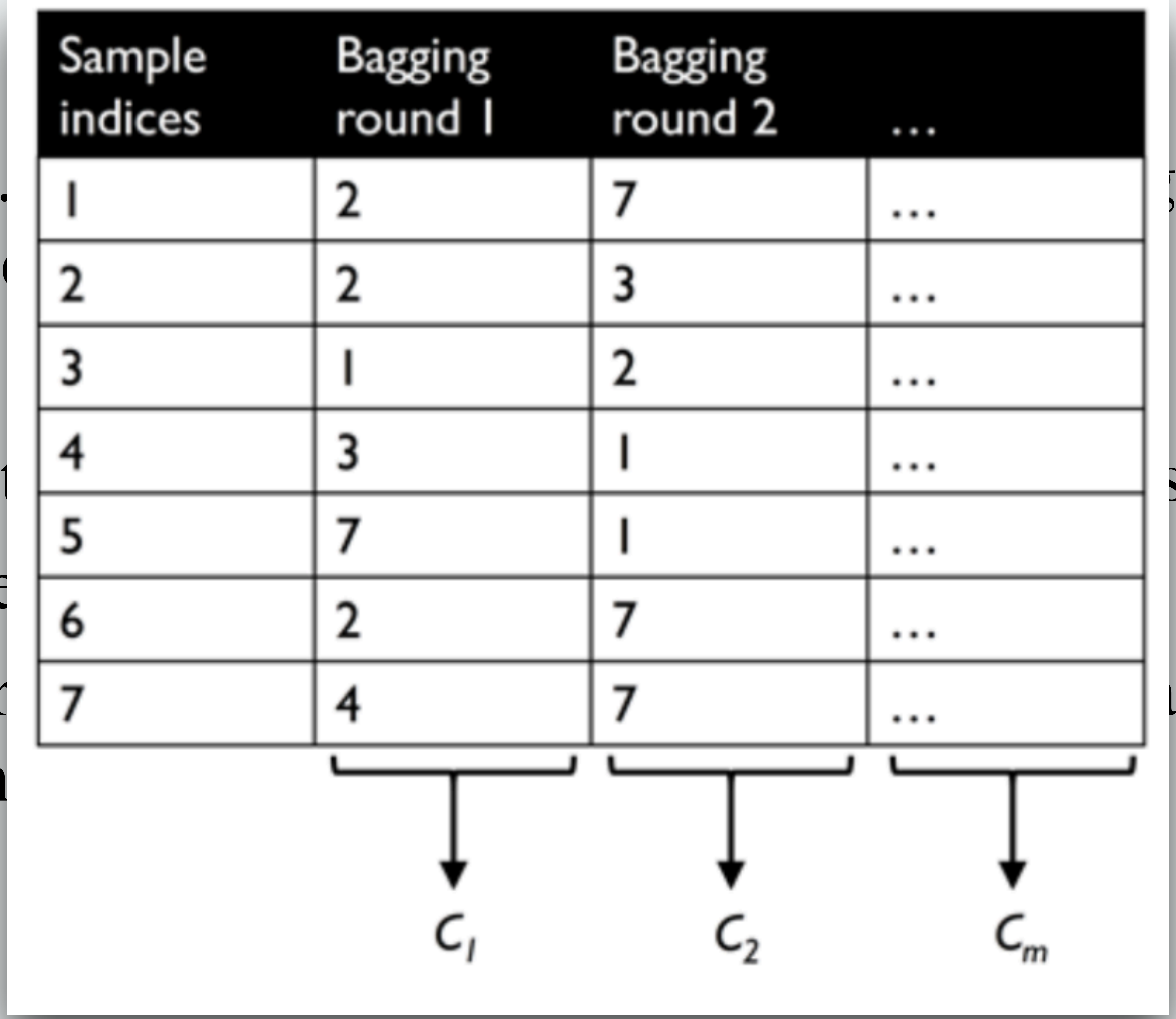
Algorithm:

Given a training set $X = x_1$, ... repeatedly (B times) selects a **random sample with replac** these samples:

For $b = 1, ..., B$:

1. Sample, with replacement se X_b, Y_b.
2. Train a classification or re

After training, predictions for aging the predictions from all the individual regression trees on



Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

Bagging

> A machine learning strategy for ensemble learning

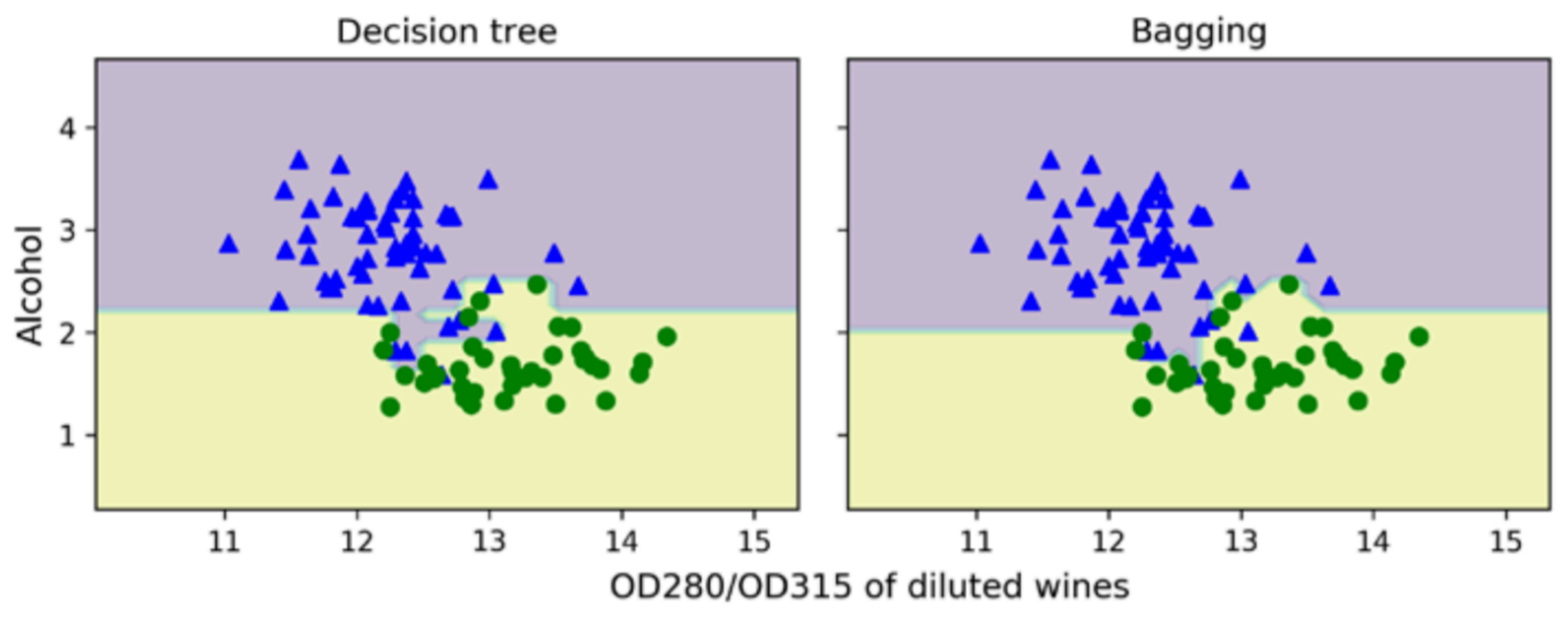
Algorithm:

Given a trai … es) selects a **random sa** …

For $b = 1, ..$

1. Sample …
2. Train a …

After trainin … ns from all the individual r …



$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

Boosting

> A(nother) machine learning strategy for ensemble learning

Intuition: Many weak models that learn from each other's mistakes, combine into one strong model

Algorithm (most general):

1. Create a weight vector $\mathbf{w}$ that encodes the *importance* of each training sample
2. For j out of m boosting iterations:
   a. Train a weighted weak classifier $C_j = \text{train}(X, y, w)$
   b. Predict class labels: $\hat{y} = \text{predict}(C_j, X)$
   c. Update $\mathbf{w}$ based on the errors that C_j makes (steps c-f in Raschka page 248)
3. To make predictions apply weighted voting, i.e. giving more prediction weight to less error prone classifiers

Boosting

> A(nother) machine learning strategy for ensemble learning

Intuitio

Algorit

1. Cr
2. Fo
   a.
   b.
   c.
3. To … less error prone classifiers

Sample indices	x	y	Weights	$\hat{y}(x <= 3.0)$?	Correct?	Updated weights
1	1.0	1	0.1	1	Yes	0.072
2	2.0	1	0.1	1	Yes	0.072
3	3.0	1	0.1	1	Yes	0.072
4	4.0	-1	0.1	-1	Yes	0.072
5	5.0	-1	0.1	-1	Yes	0.072
6	6.0	-1	0.1	-1	Yes	0.072
7	7.0	1	0.1	-1	No	0.167
8	8.0	1	0.1	-1	No	0.167
9	9.0	1	0.1	-1	No	0.167
10	10.0	-1	0.1	-1	Yes	0.072

AdaBoost example from Raschka

Boosting

> Mini exercise: discuss with your neighbor

Q1: What is the intuition behind weighting samples?

Q2: How is it practically done in AdaBoost (Raschka p. 248, c-f)

c. Compute weighted error rate: $\varepsilon = \boldsymbol{w} \cdot (\hat{\boldsymbol{y}} \neq \boldsymbol{y})$.

d. Compute coefficient: $\alpha_j = 0.5 \log \frac{1-\varepsilon}{\varepsilon}$.

e. Update weights: $\boldsymbol{w} := \boldsymbol{w} \times \exp(-\alpha_j \times \hat{\boldsymbol{y}} \times \boldsymbol{y})$.

f. Normalize weights to sum to 1: $\boldsymbol{w} := \boldsymbol{w} / \sum_i w_i$.

Q3: How are weights used during training?

$C_j = \text{train}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w})$.

Decision trees

Decision trees

Lays eggs	Cold blooded	Mammal
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
1	0	0
1	0	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0

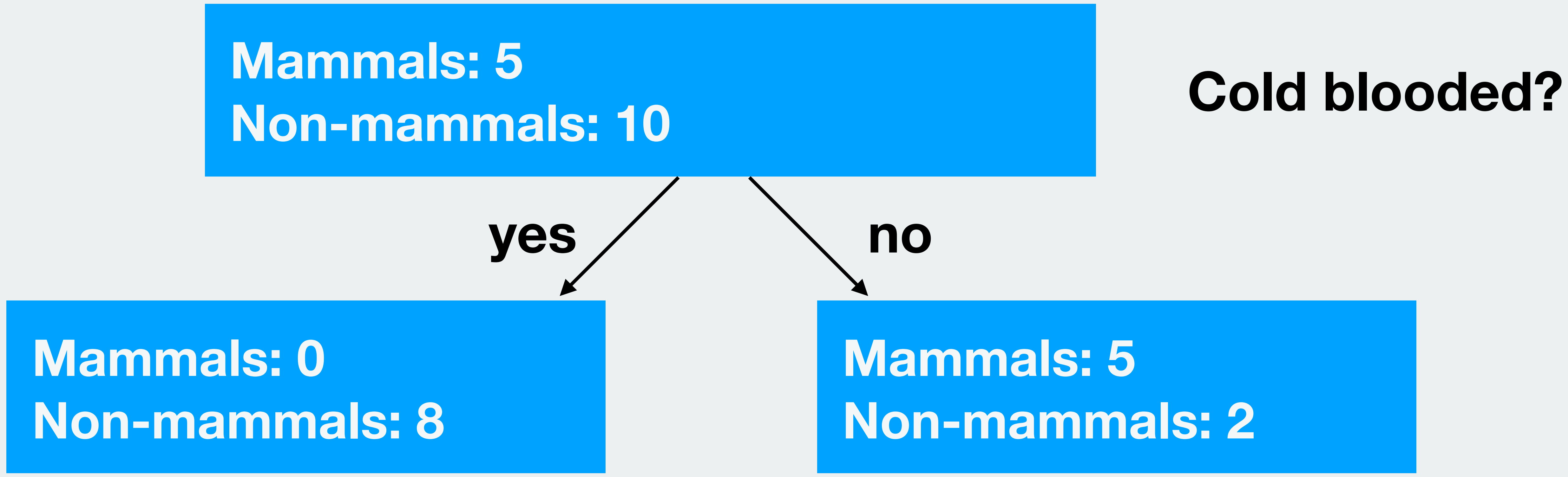
features **target**

Mammals: 5
Non-mammals: 10

Decision trees

Lays eggs	Cold blooded	Mammal
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
1	0	0
1	0	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0
1	1	0

features **target**

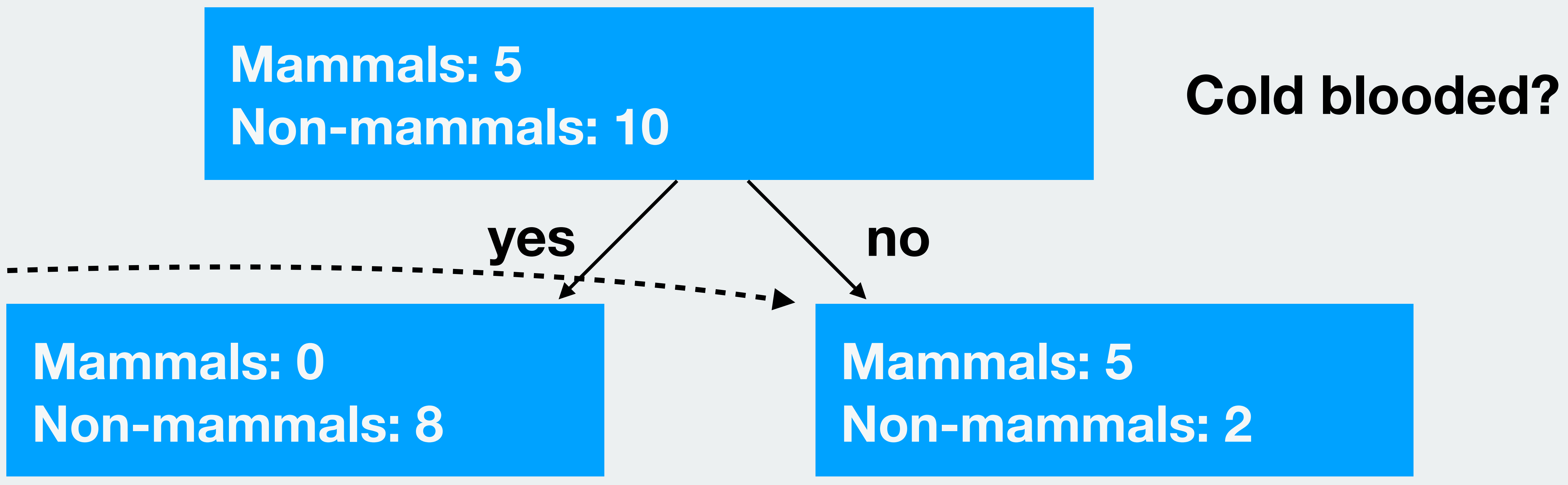


Decision trees

	Lays eggs	Cold blooded	Mammal
	0	0	1
	0	0	1
	0	0	1
no	0	0	1
	0	0	1
	1	0	0
	1	0	0
	1	1	0
	1	1	0
	1	1	0
yes	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

features

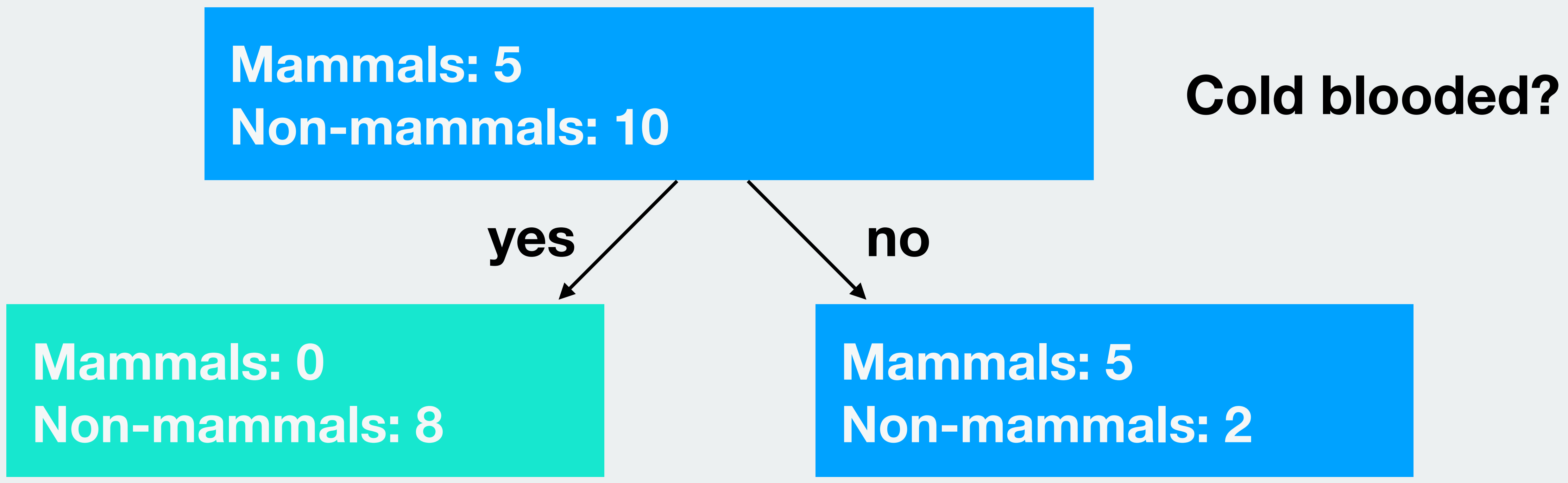
target



Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
	1	0	0
	1	0	0
yes	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

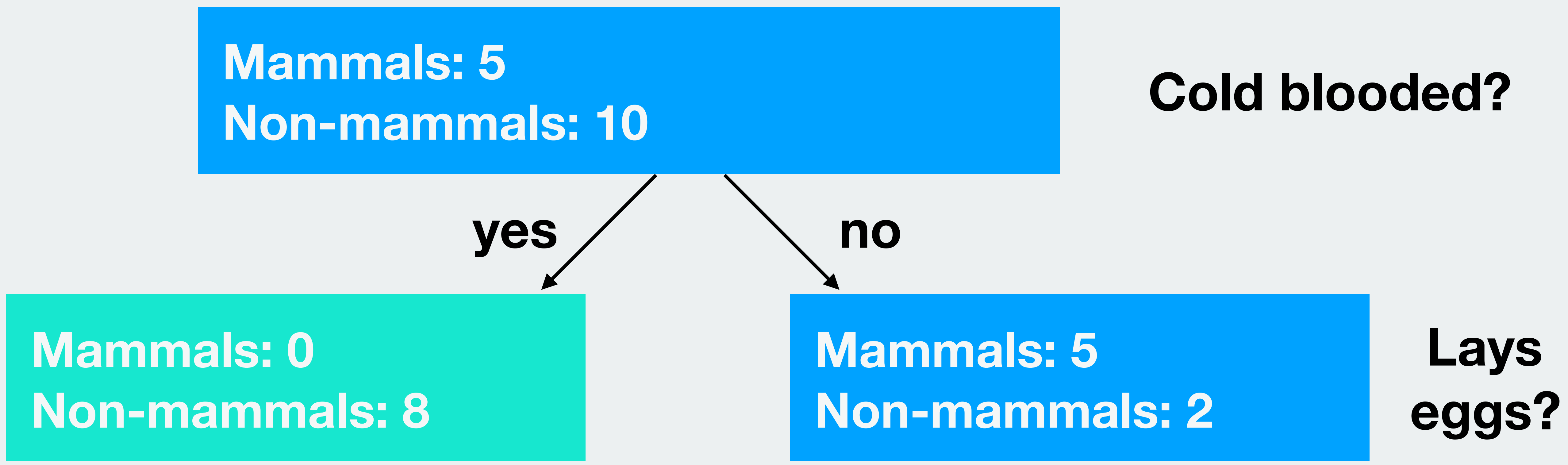
features | target



Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
	1	0	0
	1	0	0
yes	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

features (Lays eggs, Cold blooded) — target (Mammal)

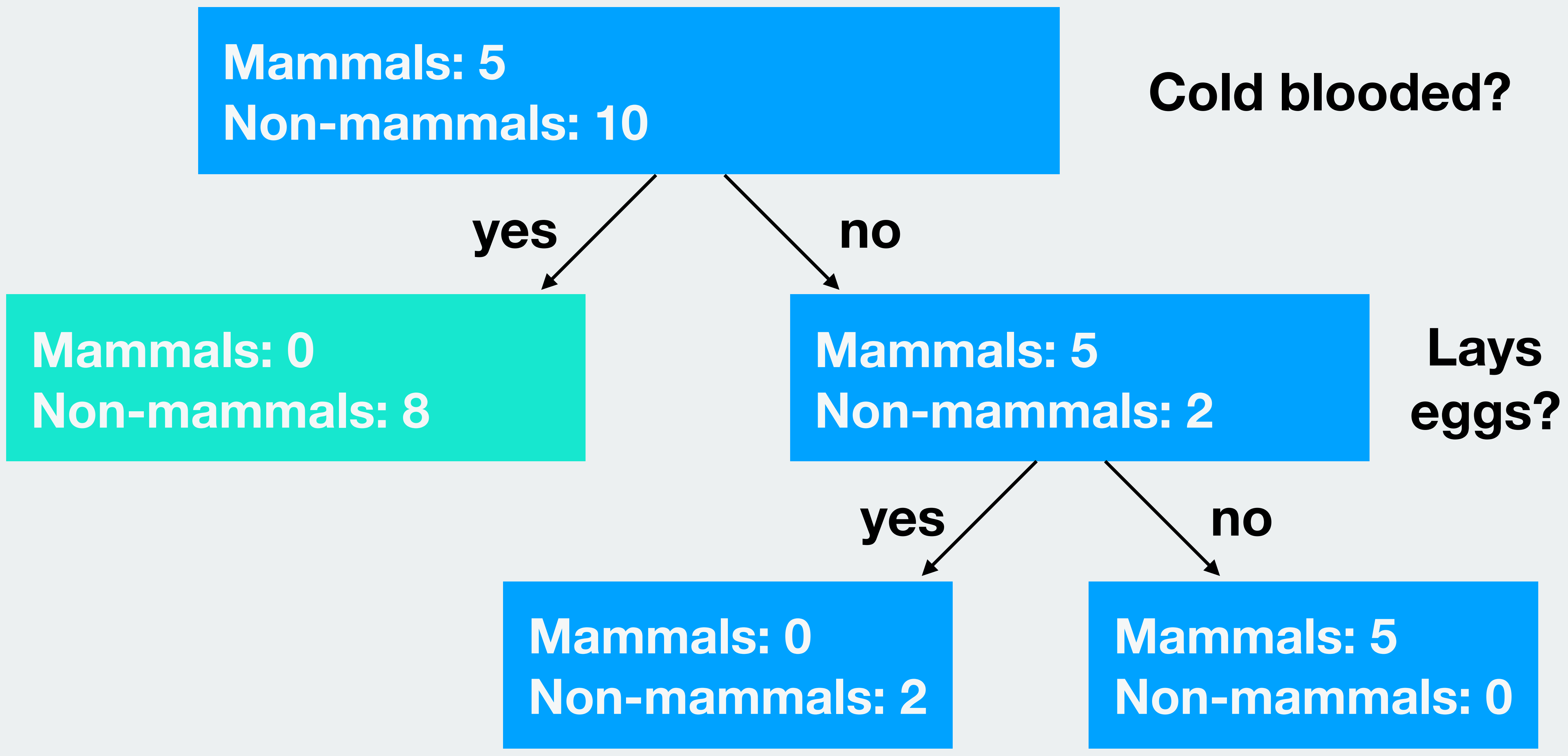


Decision trees

		Lays eggs	Cold blooded	Mammal
	no	0	0	1
		0	0	1
		0	0	1
		0	0	1
		0	0	1
	yes	1	0	0
		1	0	0
yes		1	1	0
		1	1	0
		1	1	0
		1	1	0
		1	1	0
		1	1	0
		1	1	0
		1	1	0

features

target

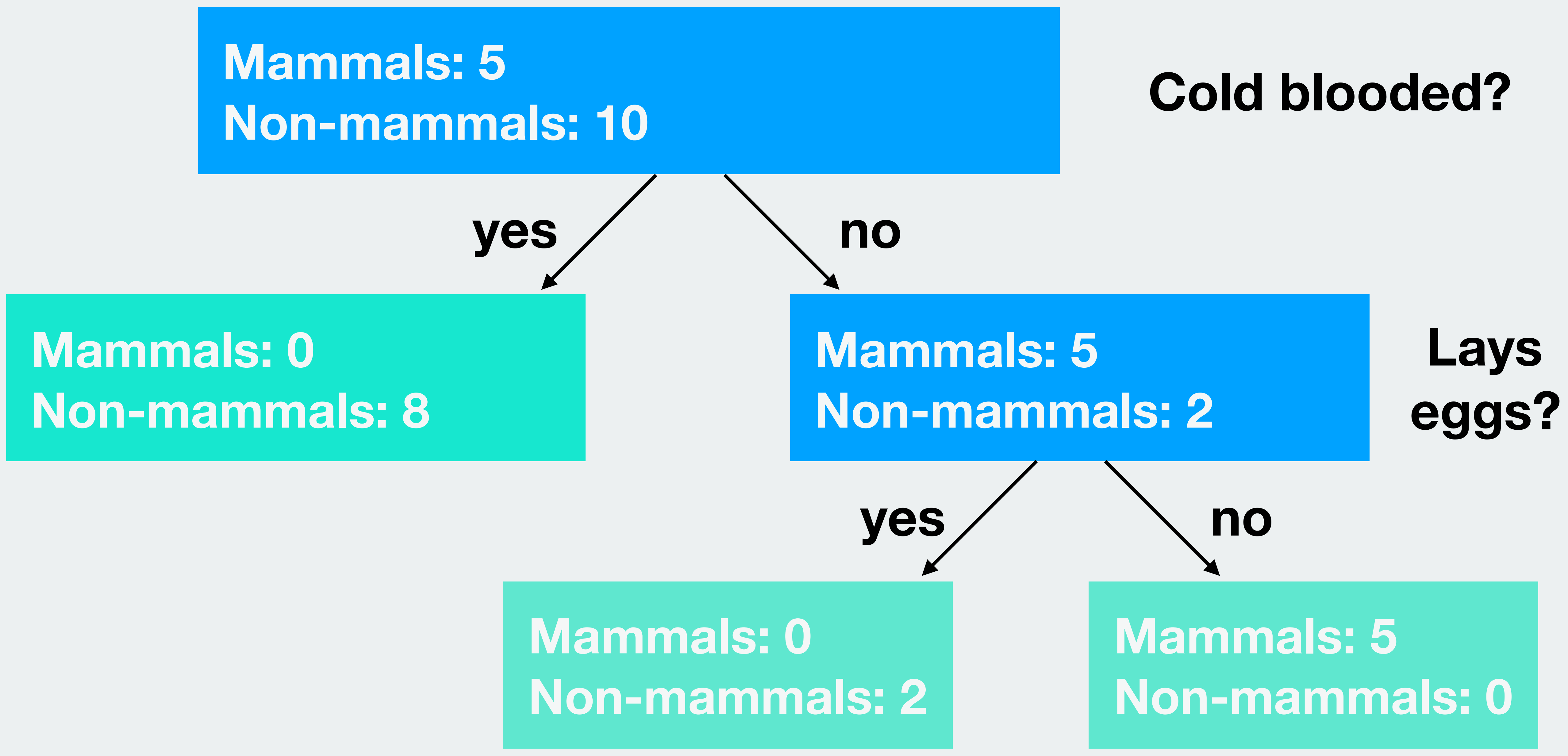


Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
yes	1	0	0
	1	0	0
yes	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

features

target



Decision trees

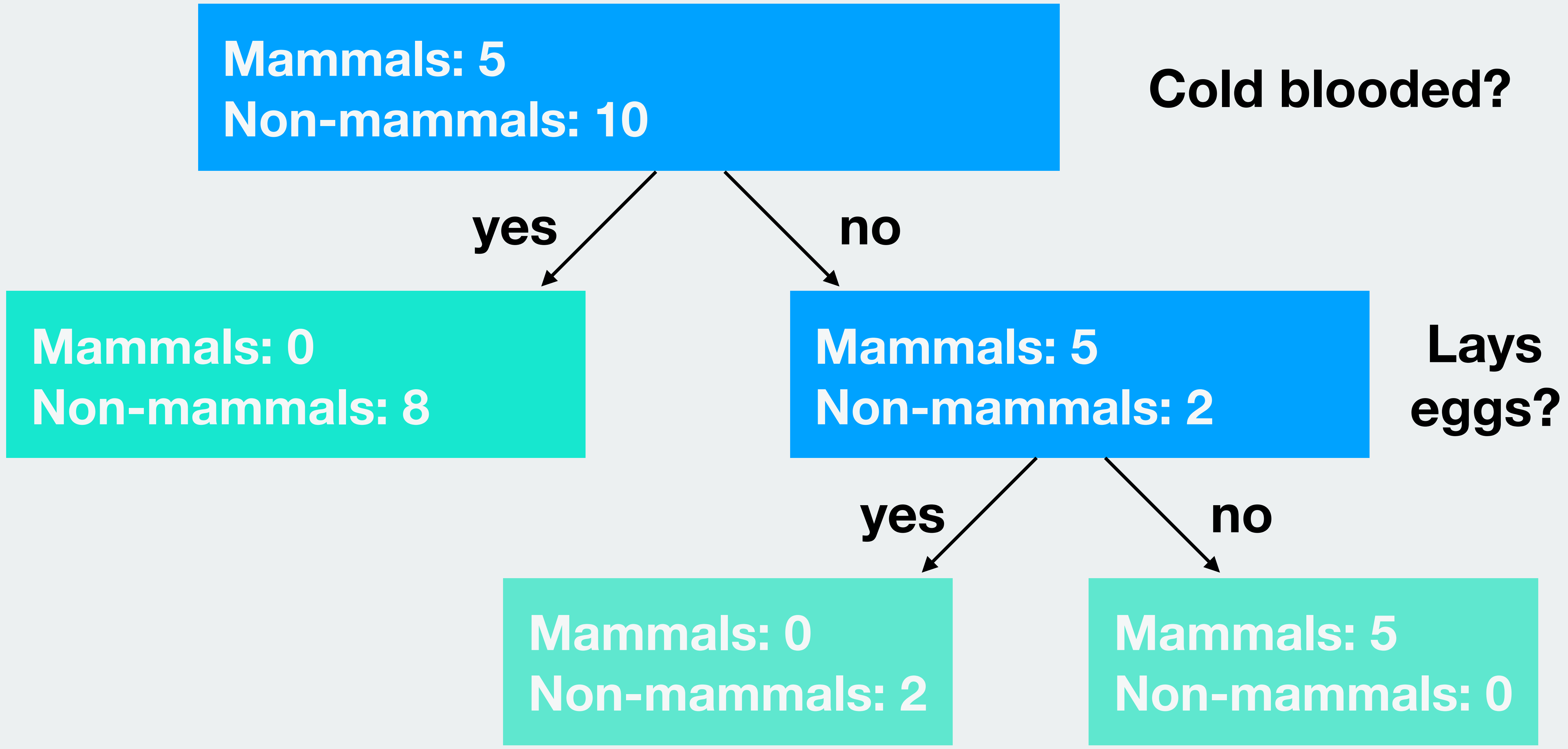
Could we have asked better questions?

Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
yes	1	0	0
	1	0	0
yes	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

features

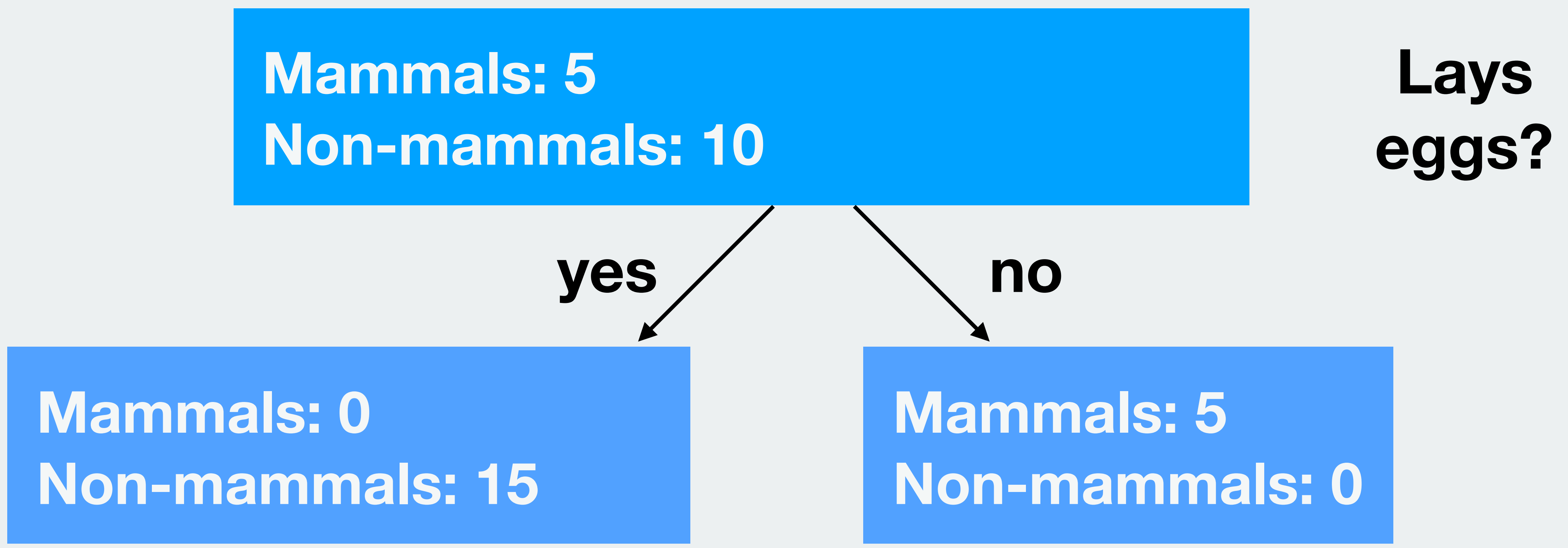
target



Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
yes	1	0	0
	1	0	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

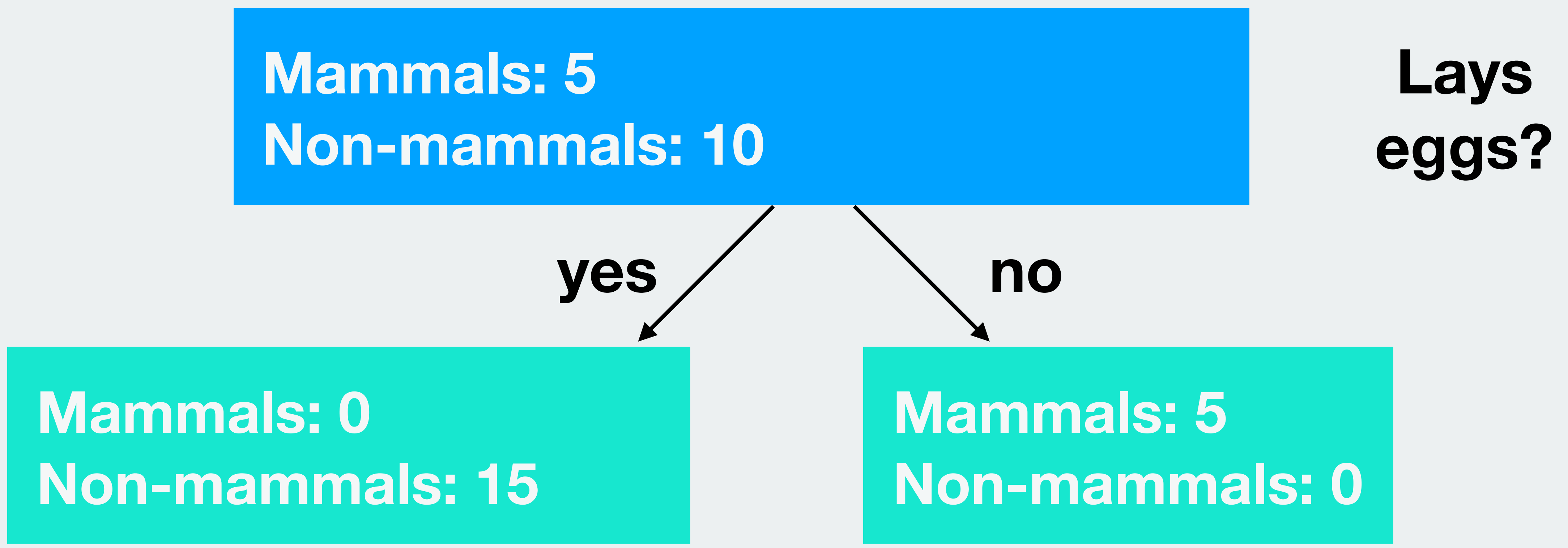
features **target**



Decision trees

	Lays eggs	Cold blooded	Mammal
no	0	0	1
	0	0	1
	0	0	1
	0	0	1
	0	0	1
yes	1	0	0
	1	0	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0
	1	1	0

features **target**



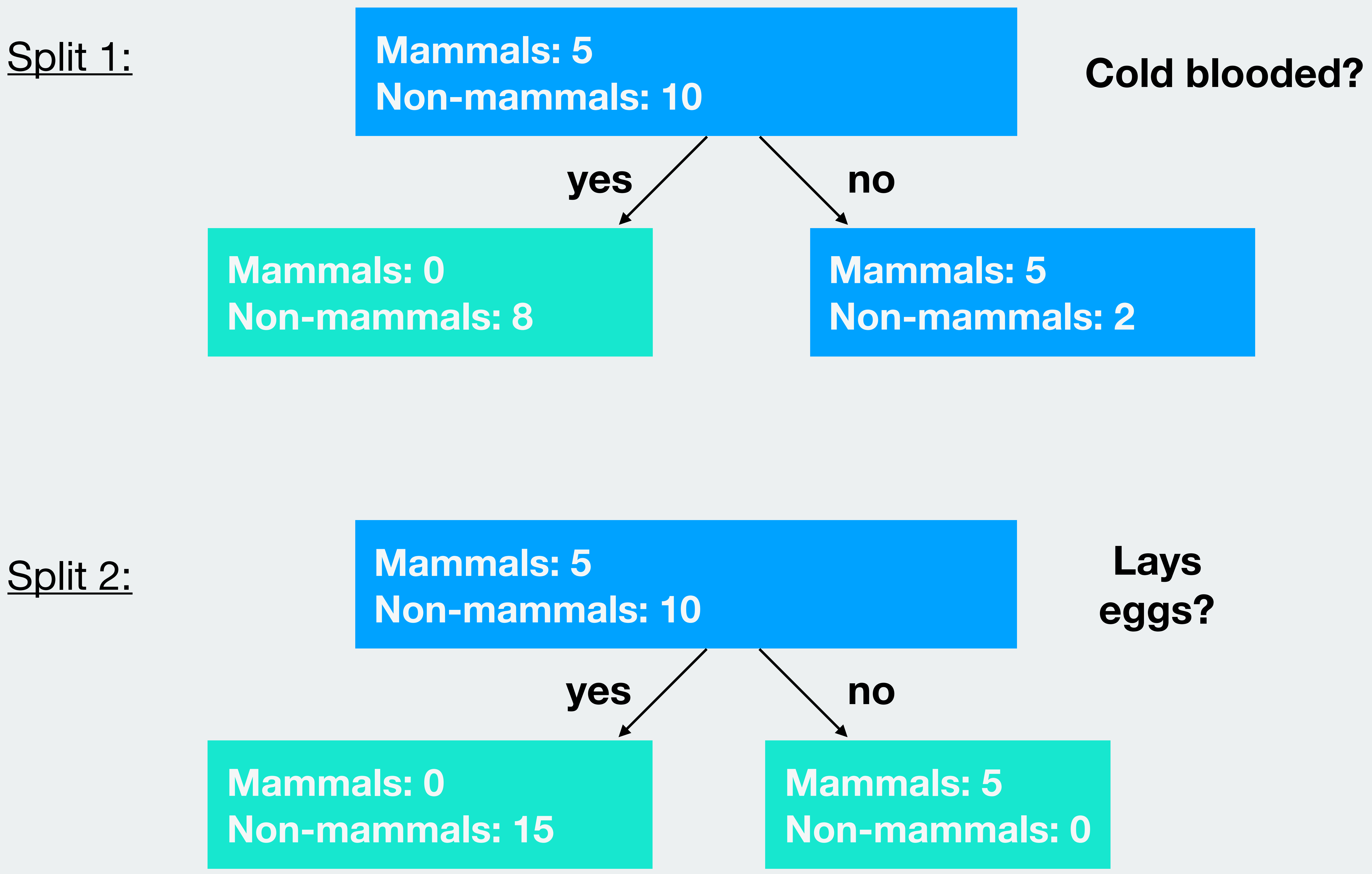
Decision trees

Can we somehow automize split selection?

Decision trees

	Pclass1	Pclass2	Pclass3	Sexfemale	Sexmale	Embarkednan	EmbarkedC	EmbarkedQ	EmbarkedS	CabinFalse	CabinTrue	PassengerId	Age	SibSp	Parch	Fare	Survived
0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1	22.0	1	0	7.2500	0
1	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	2	38.0	1	0	71.2833	1
2	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	3	26.0	0	0	7.9250	1
3	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	4	35.0	1	0	53.1000	1
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	5	35.0	0	0	8.0500	0
5	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	6	NaN	0	0	8.4583	0
6	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	7	54.0	0	0	51.8625	0
7	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	8	2.0	3	1	21.0750	0
8	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	9	27.0	0	2	11.1333	1
9	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	10	14.0	1	0	30.0708	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
881	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	882	33.0	0	0	7.8958	0
882	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	883	22.0	0	0	10.5167	0
883	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	884	28.0	0	0	10.5000	0
884	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	885	25.0	0	0	7.0500	0
885	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	886	39.0	0	5	29.1250	0
886	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	887	27.0	0	0	13.0000	0
887	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	888	19.0	0	0	30.0000	1
888	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	889	NaN	1	2	23.4500	0
889	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	890	26.0	0	0	30.0000	1
890	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	891	32.0	0	0	7.7500	0

Automatic split selection



Automatic split selection

(Shannon) $$Entropy = -\sum_i p(i) \log_2 p(i)$$

Input: Probability vector (a list of values between 0 and 1, which sums to 1)

Output: Entropy (a measure of how “spread out” the probability distribution is)

Automatic split selection

$$Entropy = -\sum_i p(i) \log_2 p(i)$$

Mammals: 0
Non-mammals: 8

p = [1, 0]

Automatic split selection

$$Entropy = -\sum_i p(i) \log_2 p(i)$$

Mammals: 0
Non-mammals: 8

p = [1, 0]

$Entropy = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$

Automatic split selection

$$Entropy = -\sum_i p(i)\log_2 p(i)$$

Mammals: 0
Non-mammals: 8

$p = [1, 0]$

$Entropy = -(1 \cdot \log_2(1) + 0 \cdot \log_2(0)) = 0$

Mammals: 5
Non-mammals: 2

$p = [2/7, 5/7]$

$Entropy = -(2/7 \cdot \log_2(2/7) + 5/7 \cdot \log_2(5/7)) = 0.86$

Automatic split selection

<u>Split 1:</u>

Mammals: 5
Non-mammals: 10

Cold blooded?

yes

Mammals: 0
Non-mammals: 8

no

Mammals: 5
Non-mammals: 2

split entropy $= 8/15 \cdot 0 + 7/15 \cdot 0.86 = 0.40$

Automatic split selection

Split 1:

Mammals: 5
Non-mammals: 10

Cold blooded?

yes → **Mammals: 0, Non-mammals: 8**

no → **Mammals: 5, Non-mammals: 2**

split entropy = $8/15 \cdot 0 + 7/15 \cdot 0.86 = 0.40$

Split 2:

Mammals: 5
Non-mammals: 10

Lays eggs?

yes → **Mammals: 0, Non-mammals: 15**

no → **Mammals: 5, Non-mammals: 0**

split entropy = $8/15 \cdot 0 + 7/15 \cdot 0 = 0$

Random forests

Random forests

1. Draw a random **bootstrap** sample of size n (randomly choose n samples from the training set with replacement).

In [7]:
```python
import numpy as np

mylist = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
np.random.choice(mylist, size=10)
```
executed in 5ms, finished 21:36:59 2020-02-13

Out[7]:
```
array([5, 8, 4, 9, 0, 8, 3, 9, 6, 0])
```

Random forests

1. Draw a r… m the training s…

	Pclass1	Pclass2	Pclass3	Sexfemale	Sexmale	Embarkednan	EmbarkedC	EmbarkedQ	EmbarkedS	CabinFalse	CabinTrue	PassengerId	Age	SibSp	Parch	Fare	Survived
0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1	22.0	1	0	7.2500	0
1	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	2	38.0	1	0	71.2833	1
2	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	3	26.0	0	0	7.9250	1
3	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	4	35.0	1	0	53.1000	1
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	5	35.0	0	0	8.0500	0
5	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	6	NaN	0	0	8.4583	0
6	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	7	54.0	0	0	51.8625	0
7	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	8	2.0	3	1	21.0750	0
8	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	9	27.0	0	2	11.1333	1
9	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	10	14.0	1	0	30.0708	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
881	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	882	33.0	0	0	7.8958	0
882	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	883	22.0	0	0	10.5167	0
883	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	884	28.0	0	0	10.5000	0
884	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	885	25.0	0	0	7.0500	0
885	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	886	39.0	0	5	29.1250	0
886	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	887	27.0	0	0	13.0000	0
887	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	888	19.0	0	0	30.0000	1
888	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	889	NaN	1	2	23.4500	0
889	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	890	26.0	0	0	30.0000	1
890	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	891	32.0	0	0	7.7500	0

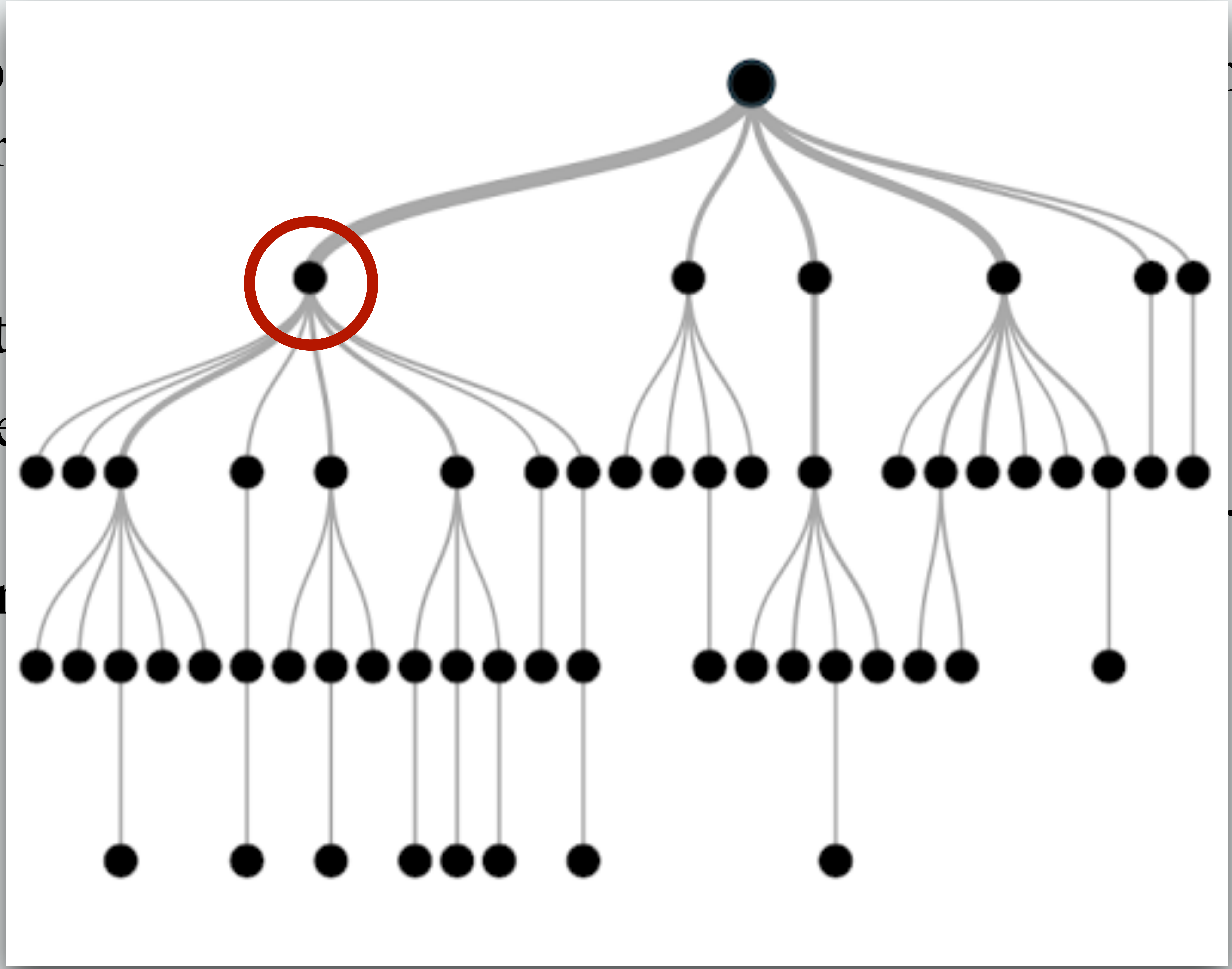
Random forests

1. Draw a random **bootstrap** sample of size n (randomly choose n samples from the training set with replacement).

2. Grow a decision tree from the bootstrap sample. At each node:
   - Randomly select d features without replacement.
   - Split the node using the feature that provides the best split according to the objective function, for instance, maximizing the information gain.

Random forests

1. Draw a random b
training set with r



mples from the

2. Grow a decision t

- Randomly sel
- Split the node
function, for i

rding to the objective

Random forests

1. Draw a random **bootstrap** sample of size n (randomly choose n samples from the training set with replacement).

2. Grow a decision tree from the bootstrap sample. At each node:
   - Randomly select d features without replacement.
   - Split the node using the feature that provides the best split according to the objective function, for instance, maximizing the information gain.

3. Repeat the steps 1-2 k times.

4. Aggregate the prediction by each tree to assign the class label by **majority vote**.

Random forests

1. Draw a random **bootstr** … n samples from the training set with replace …

2. Grow a decision tree fro …
   - Randomly select d fe …
   - Split the node using … according to the objective function, for instance …

3. Repeat the steps 1-2 k ti …

4. Aggregate the prediction by each tree to assign the class label by **majority vote**.

$$\left.\begin{array}{l} \text{model1(x) = 1} \\ \text{model2(x) = 1} \\ \text{model3(x) = 0} \\ \text{model4(x) = 1} \\ \text{model5(x) = 1} \\ \text{model6(x) = 1} \\ \text{model7(x) = 0} \\ \text{model8(x) = 1} \\ \quad \ldots \\ \text{modeln(x) = 1} \end{array}\right\} \text{ average = 0.84 } \approx \text{ 1}$$

Random forests

> Quick word on feature importance

- When fitting trees, we estimate the information gain for every feature at each split
- We can summarize the *importance* of a feature as its relative amount of information gain delivered during classification

```
In [18]:
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(X, y)
model.feature_importances_
```

executed in 50ms, finished 09:38:37 2020-02-14

```
Out[18]: array([0.15166831, 0.1353509 , 0.25520054, 0.27534356, 0.03120302,
                0.01501129, 0.01015842, 0.00808299, 0.01251548, 0.00859388,
                0.00075693, 0.        , 0.        , 0.01614362, 0.02831235,
                0.02773086, 0.02392785])
```

K-nearest neighbors

K-nearest neighbors

Algorithm:

1. Choose the number of k and a distance metric.
2. Find the k-nearest neighbors of the sample that we want to classify.
3. Assign the class label by majority vote.

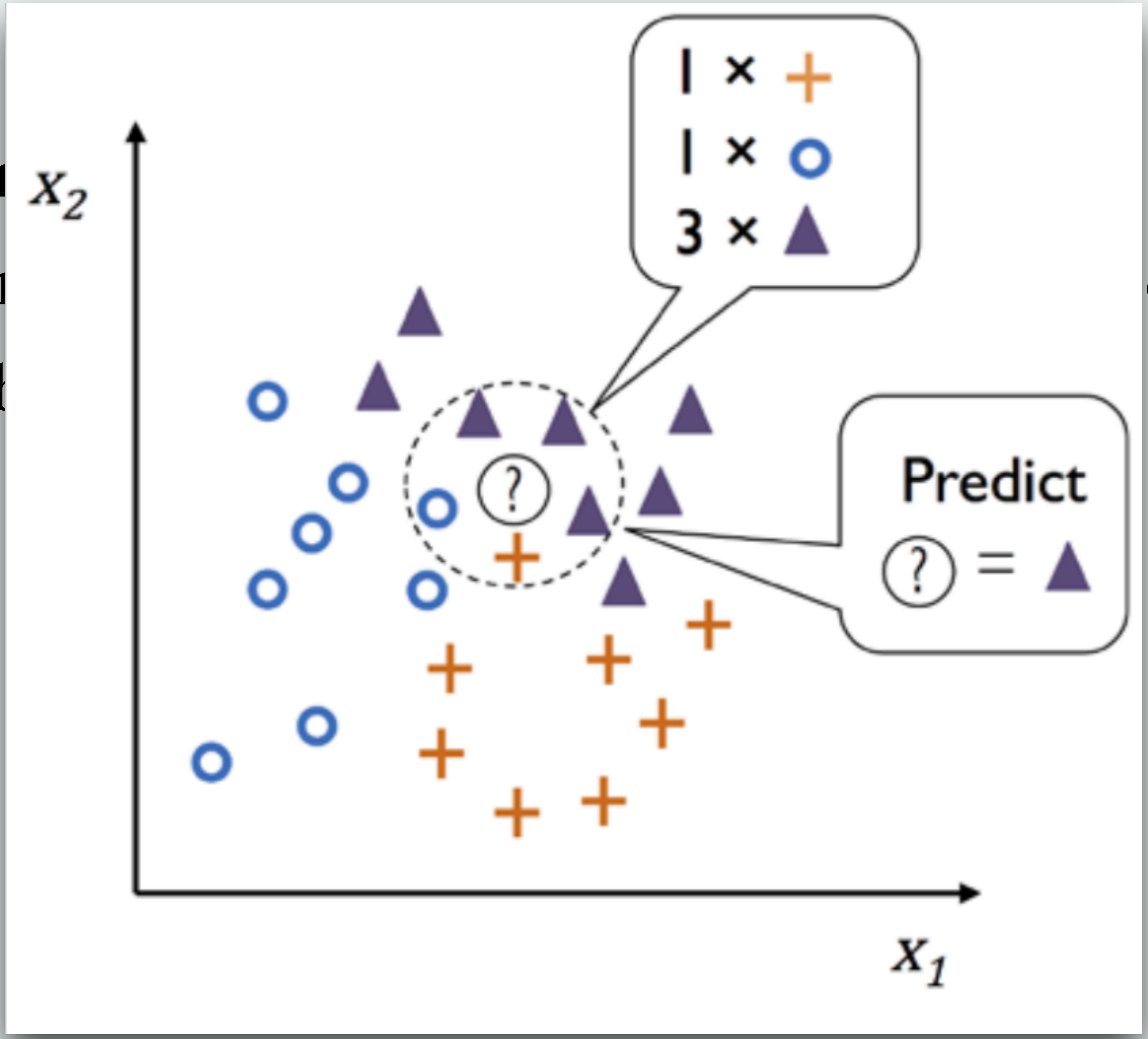
K-nearest neighbors

Algorithm:

1. Choose the numbe

2. Find the k-nearest classify.

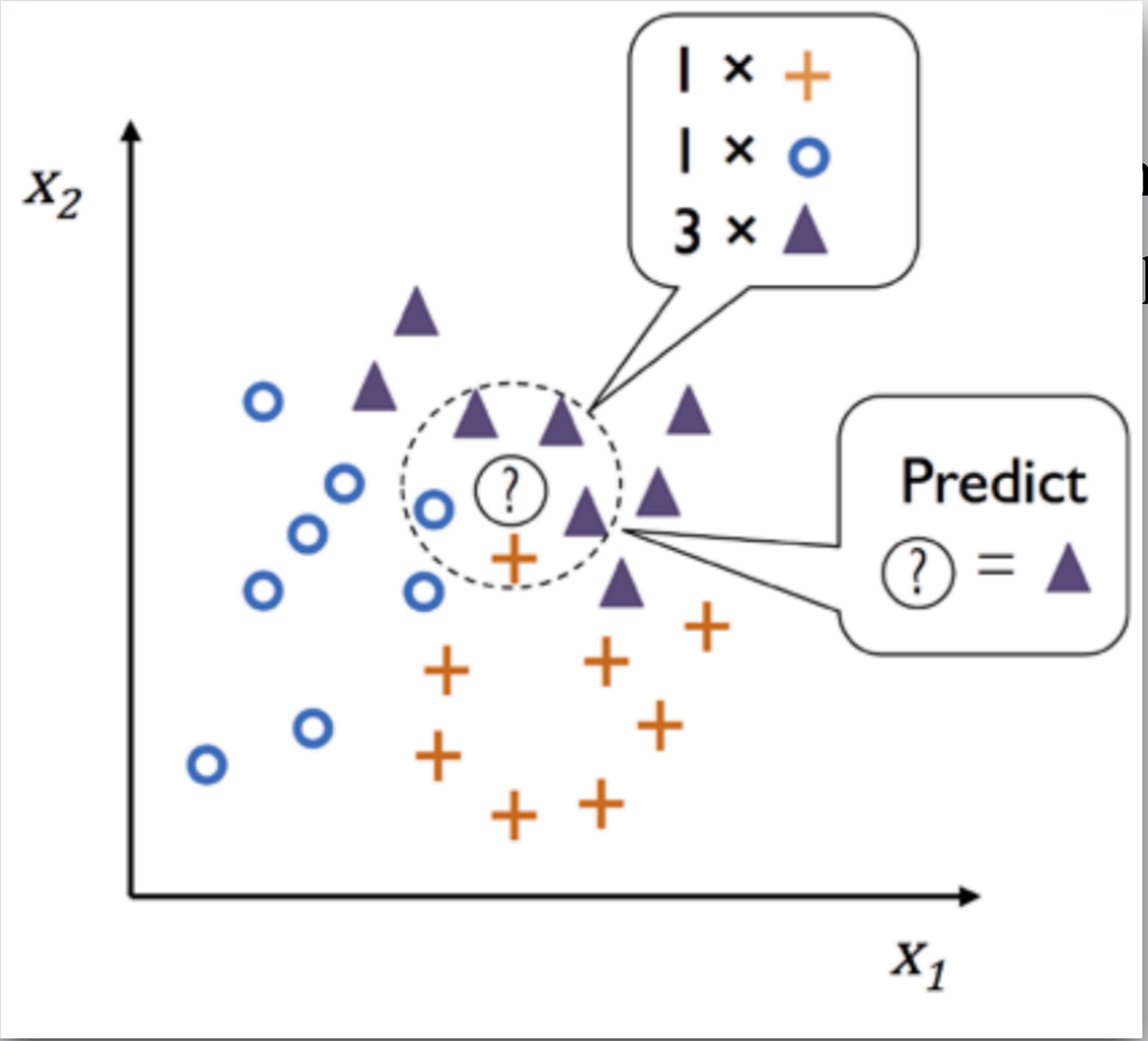
3. Assign the class la



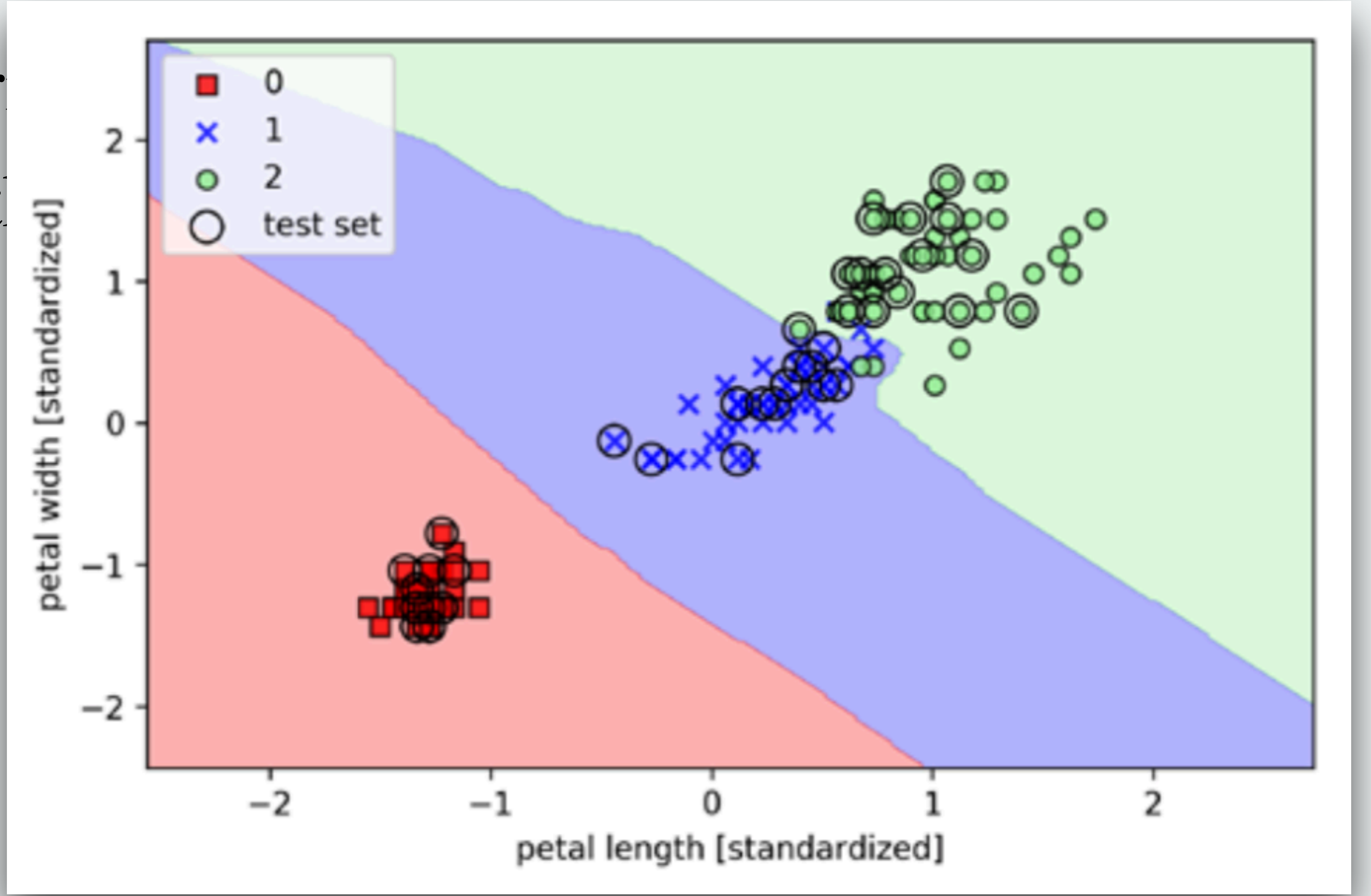
K-nearest neighbors

Algo

1. C
2. F
3. A



etr
le t



K-nearest neighbors

Pros:

Cons:

K-nearest neighbors

Pros:

1. No training needed!
2. Immediately adapts as we add more training data

Cons:

1. Search space grows linearly with amount of data
2. Choosing K requires hyperparameter tuning
3. Prone to overfitting due to ‘Curse of dimensionality’
4. For most distance metrics, data must be standardized

Gradient boosting

Gradient boosting

Algorithm (informally):

1. Create a very weak model F_0 that just predicts the average class, $\bar{y}$

2. Estimate the vector of residuals $\mathrm{y} - F_0(\mathrm{X})$

3. Fit another week model F_1 to predict the residuals $\mathrm{y} - F_0(\mathrm{X})$. A perfect F_1 would imply $F_{0+1} = F_0(\mathrm{X}) + F_1(\mathrm{X}) = \mathrm{y}$

4. But F_1 is weak, and highly imperfect. Therefore, continue adding more functions like this for many more iterations

Gradient boosting

Pros:

1. Very good performance with classification and regression trees
2. Easily extensible with arbitrary cost functions
3. State-of-the-art performance on *shallow* learning problems
4. The XGBoost library makes it easy to use

Cons:

1. Training is not super fast
2. Overfitting must be controlled with regularization

Further learning

Check out:

1. Scikit-learn user guide for a practical overview of methods
   https://scikit-learn.org/stable/user_guide.html
2. The XGBoost library for gradient boosting
   https://xgboost.readthedocs.io/en/latest/
3.