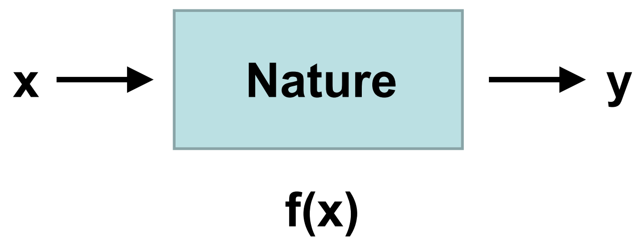
Today

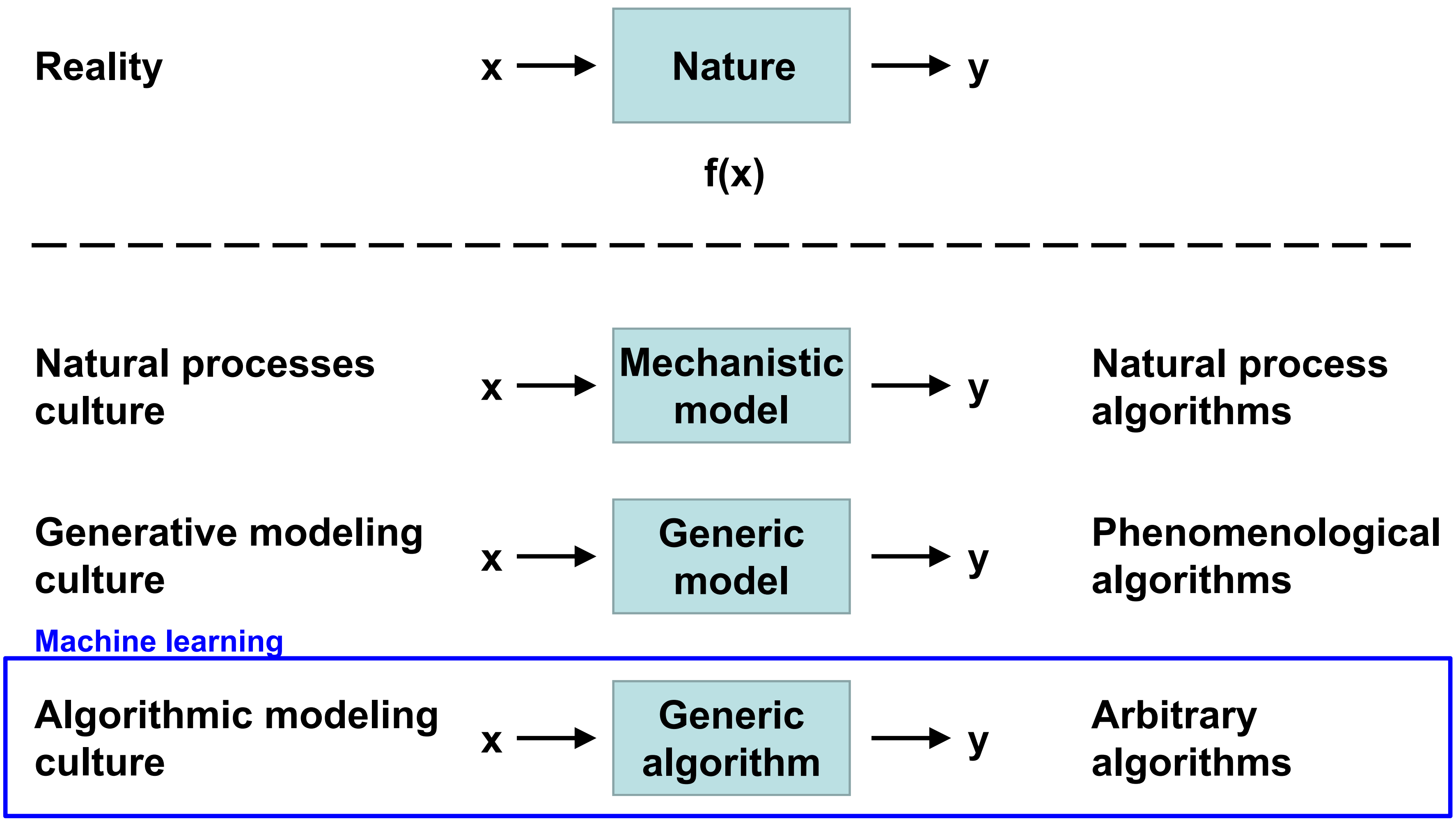
- Machine learning in a nutshell
- All of it, all at once!

Trying to learn a function f

Reality



Trying to learn a function f



f can mean different things in different cultures

f(x) for prediction

Reality $\boldsymbol{Z_1, Z_2, \ldots, Z_\Omega}$ $\boldsymbol{Y = \mathrm{nature}(Z)}$

Some set of causally-connected variables

Data $\boldsymbol{X_1, X_2, \ldots, X_p}$

A set of potential predictor variables

$$\boldsymbol{Y = f(X) + \epsilon}$$

Systematic component Error

Prediction $\boldsymbol{\hat{Y} = \hat{f}(X)}$ Hats indicate predicted Y and estimated f

Goal of prediction

Use data to find a function $\hat{f}$ that has good predictive performance given X

That is, $\hat{f}$ is accurate on new observations

Goal of machine learning

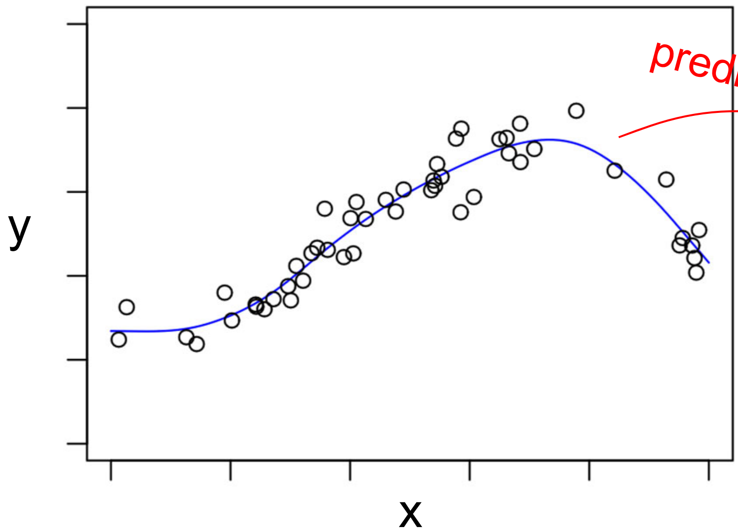
To predict accurately!

- Species distribution
  - map
  - predict accurately for places we won’t visit
- Climate change forecast
  - predict accurately for the future
- Antelopes in camera trap images
  - hand over the identification task to a machine so we don’t have to look at images!
  - predict accurately for images that we’ll never look at

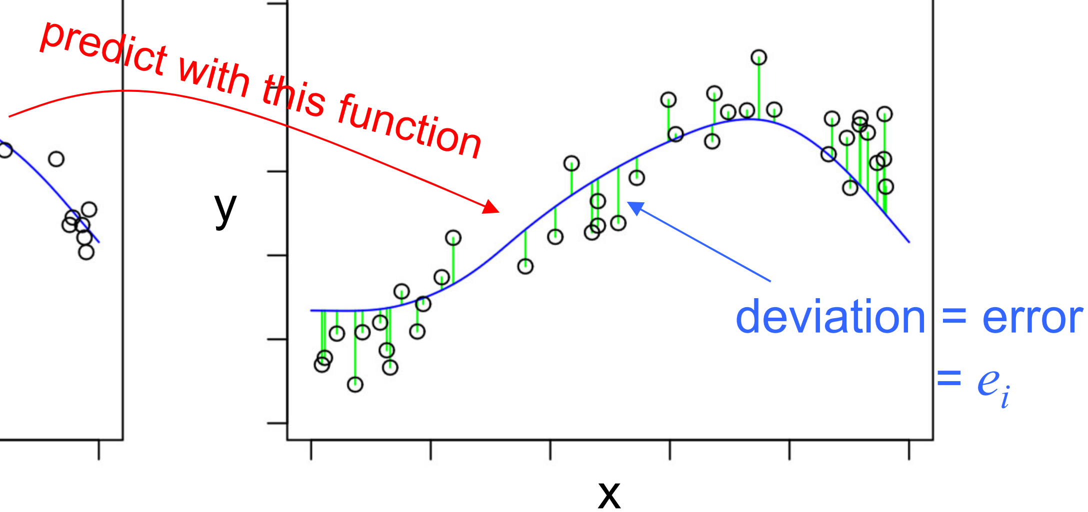
Predictive skill

Basic idea: out-of-sample accuracy

$\hat{f}$ (trained on some data)



$\hat{f}$ predicting new data



e.g. mean squared error (MSE) $\frac{1}{n}\sum_{i}^{n} e_i^2$

Machine learning workflow

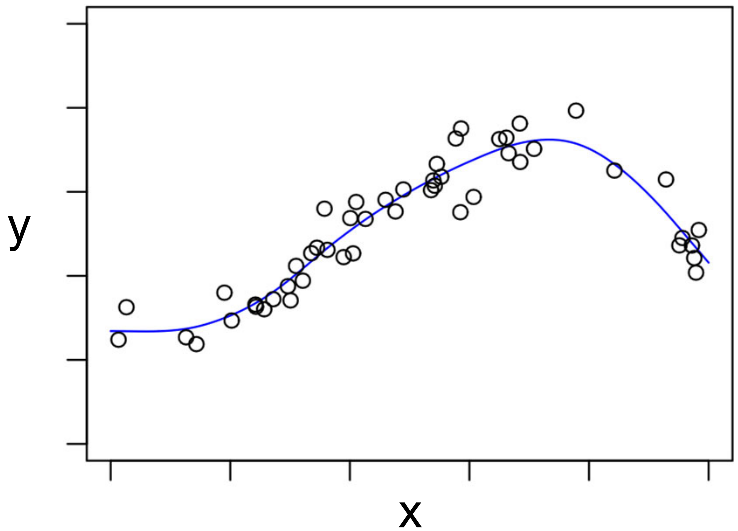
Overall algorithm:

1. Create model algorithm(s) for $\hat{f}(x)$
2. Use a training algorithm to find parameter values of $\hat{f}(x)$
3. Use an inference algorithm to measure error and compare predictive skill among models (model families, tuning parameters, x sets, etc).

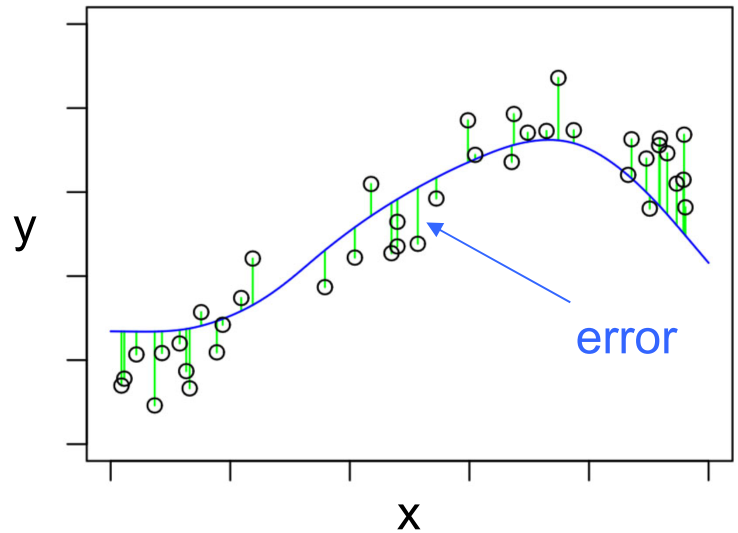
Inference algorithm

Basic algorithm: out-of-sample validation

1. Train model on training dataset

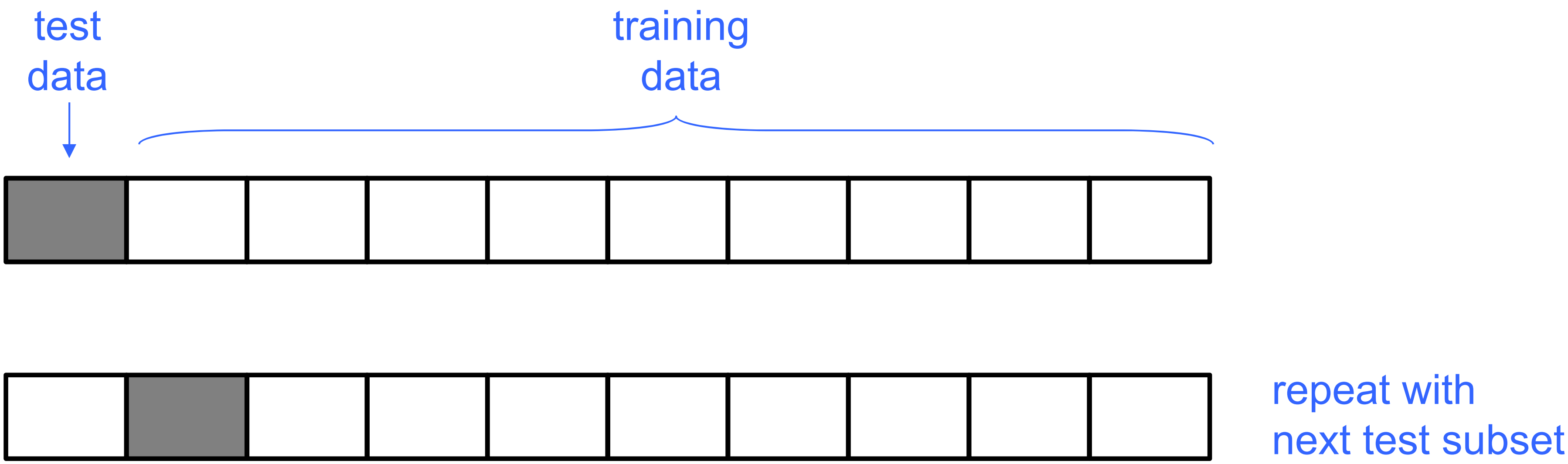


2. Test model on validation dataset



k-fold cross validation (CV)

Divide dataset into k parts (preferably randomly)



... repeat with each test subset

k-fold CV inference algorithm

Algorithm

divide dataset into k parts i = 1...k
for each i
    test dataset = part i
    training dataset = remaining data
    find f using training dataset
    use f to predict for test dataset
    e_i = prediction error for part i
CV_error = mean(e)

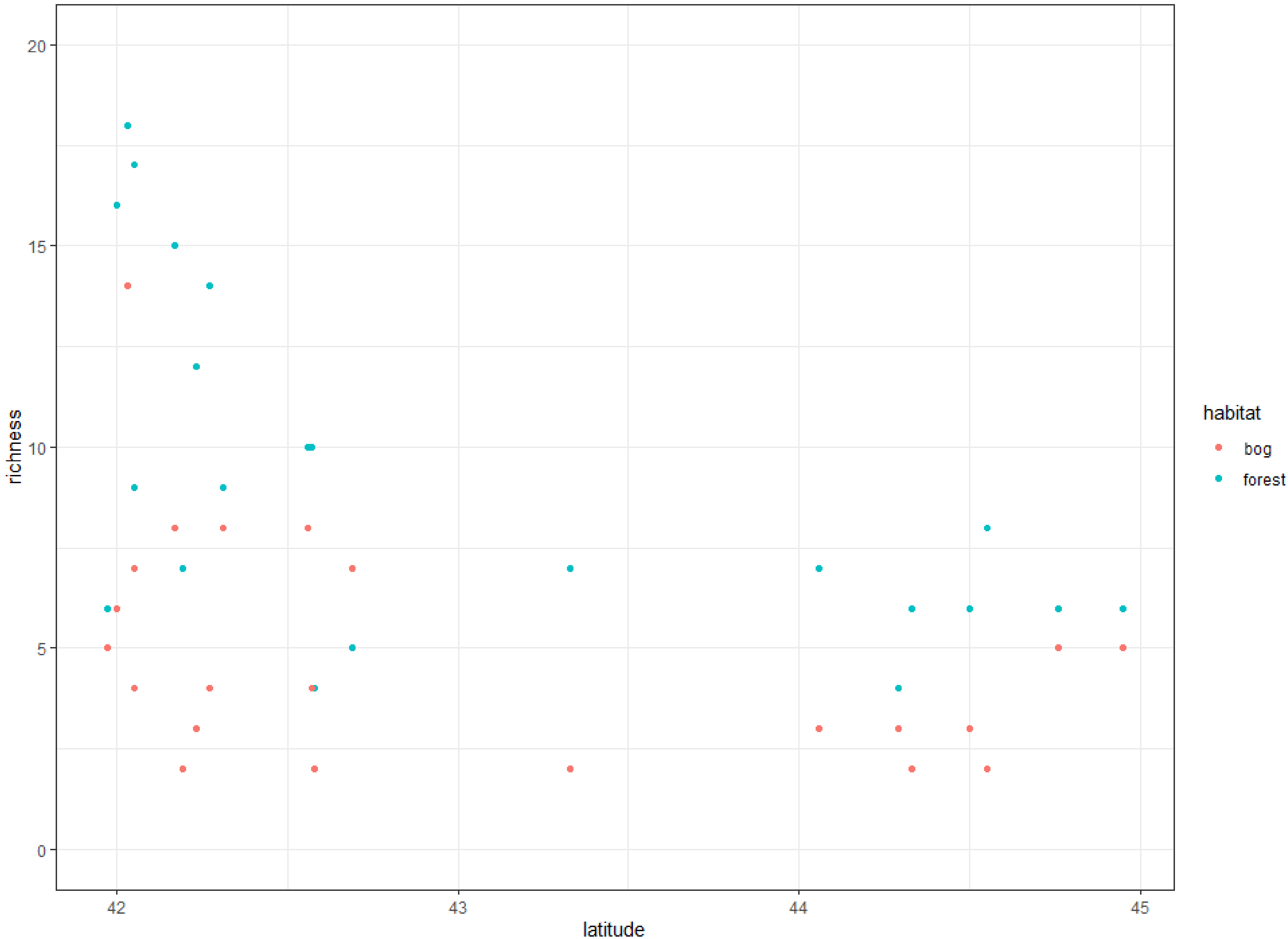
Typical values for k: 5, 10, n

Two machine learning tasks

- Regression:
  - numerical response variable
  - predict a numerical value given x
  - e.g. number of species given latitude
- Classification:
  - categorical response variable
  - predict the category given x
  - e.g. is it a bird, deer, tree, or mountain lion?
  - e.g. is it dead or alive?; present or absent?

Ants data

```
> head(ants)
  site habitat latitude elevation richness
1  TPB  forest    41.97       389        6
2  HBC  forest    42.00         8       16
3  CKB  forest    42.03       152       18
4  SKP  forest    42.05         1       17
5   CB  forest    42.05       210        9
6   RP  forest    42.17        78       15
```



richness
20
15
10
5
0
42
43
44
45
latitude
habitat
bog
forest

Basic full ML setup

- 3 algorithms:
  - model: flexible function $\hat{f}(x)$;
    e.g. polynomial linear model

    $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ ... + \beta_m x^m$ m=order

Basic full ML setup

- 3 algorithms:
  - model: flexible function $\hat{f}(x)$;
    e.g. polynomial linear model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_m x^m \qquad \text{m=order}$$

How is this an algorithm?

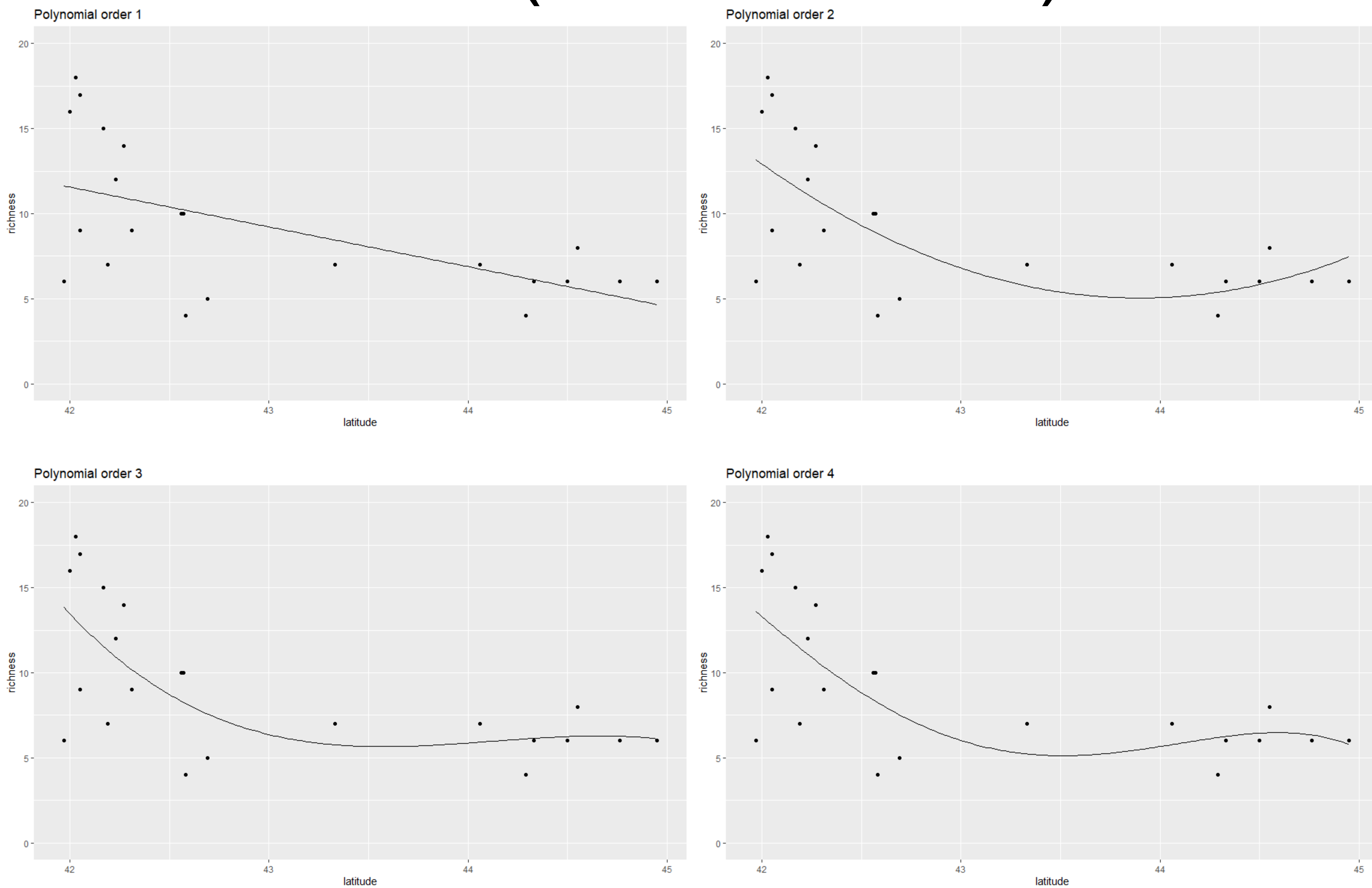
R

```r
poly2 <- function(x, beta_0, beta_1, beta_2) {
    y <- beta_0 + beta_1 * x +  beta_2 * x^2
    return(y)
}
```

Python

```python
def poly2(x, beta_0, beta_1, beta_2):
    y = beta_0 + beta_1 * x + beta_2 * x**2
    return y
```

Ants (forest habitat)



Basic full ML setup

- 3 algorithms:
  - model: flexible function $\hat{f}(x)$;
    e.g. polynomial linear model

    $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ ... + \beta_m x^m$ m=order
  - training: optimize an objective function
    e.g. least squares
  - minimize $SSQ = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ for training data

```
lm(richness ~ poly(latitude, order), data=forest_ants)
```

Basic full ML setup

- 3 algorithms:
  - model: flexible function $\hat{f}(x)$;
    e.g. polynomial linear model

    $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_m x^m$ m=order
  - training: optimize an objective function
    e.g. least squares
  - minimize $SSQ = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ for training data
  - inference: measure error by k-fold cross validation; tuning parameter (order of poly)

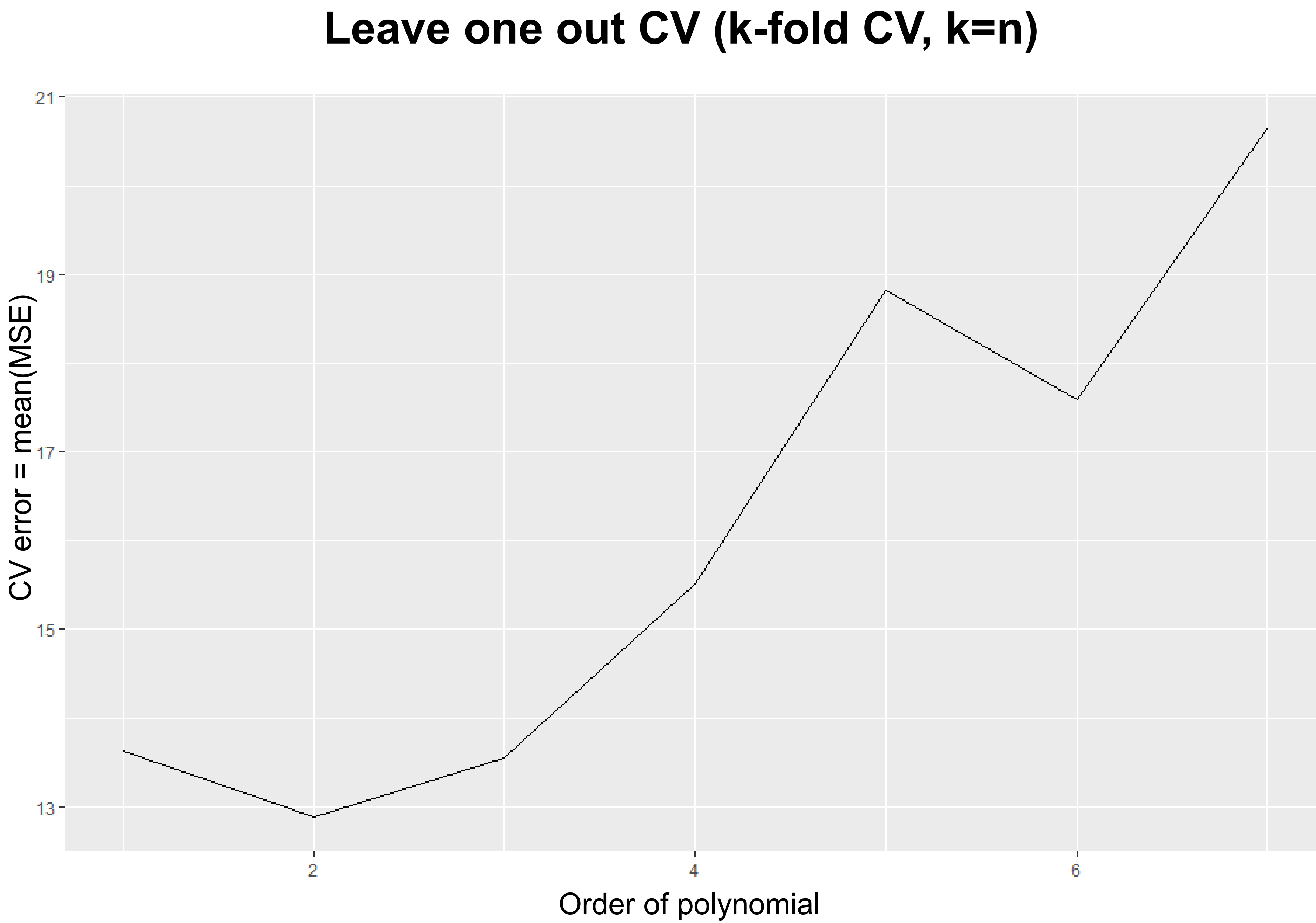
Leave-one-out cross validation

- LOOCV
- special case of k-fold CV: k = n

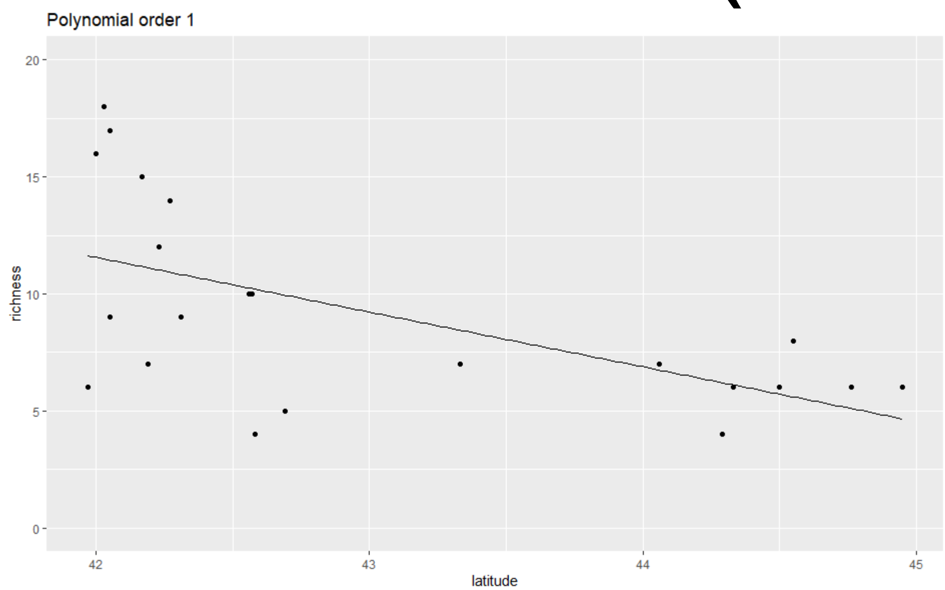
Algorithm

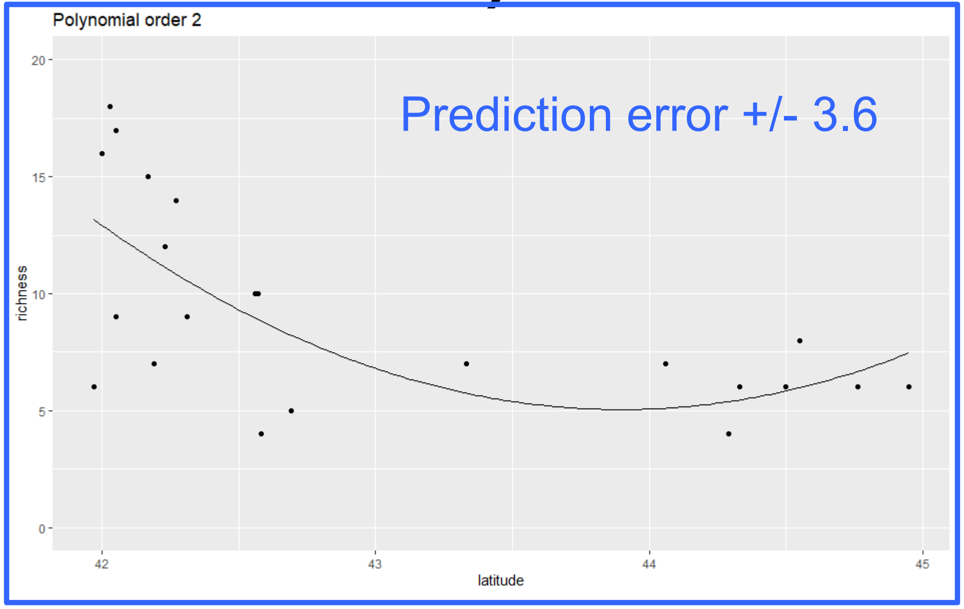
```
for each data point
        fit model without point
        predict for that point
        measure prediction error (compare to observed)
CV_error = mean error across points
```

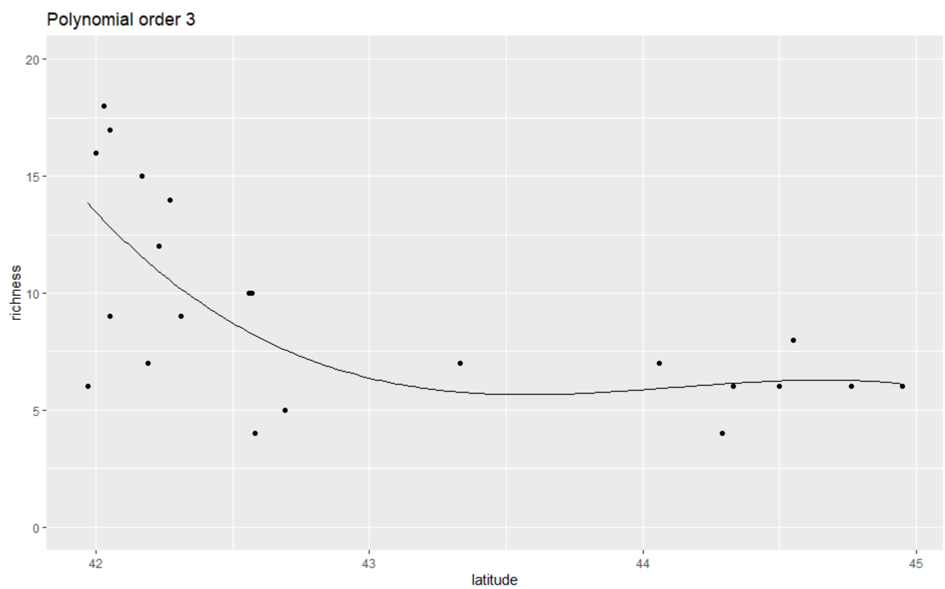
(AIC and Bayesian WAIC are equivalent as $n \to \infty$)

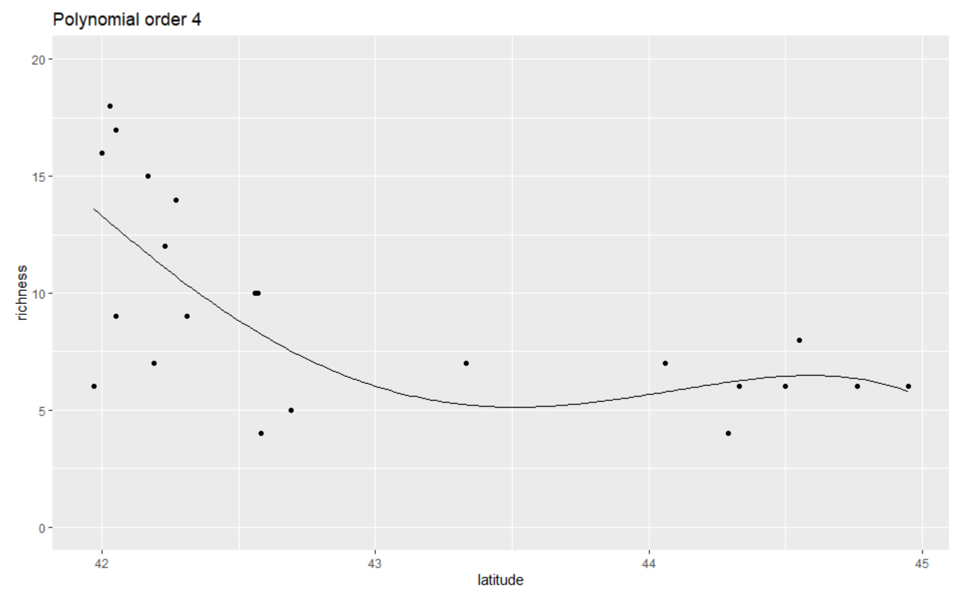

Leave one out CV (k-fold CV, k=n)
21
19
17
15
13
CV error = mean(MSE)
2
4
6
Order of polynomial

Ants (forest habitat)





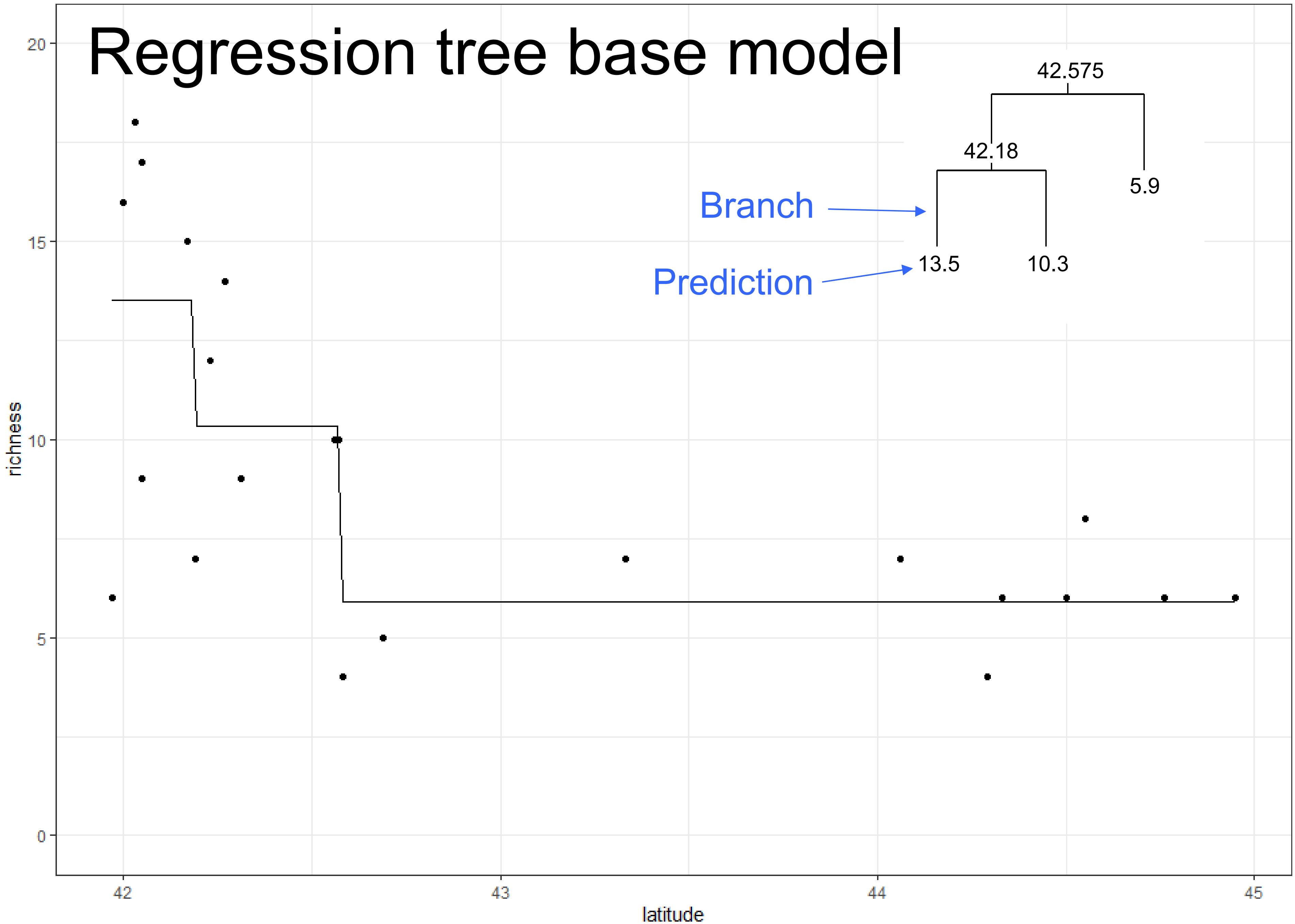




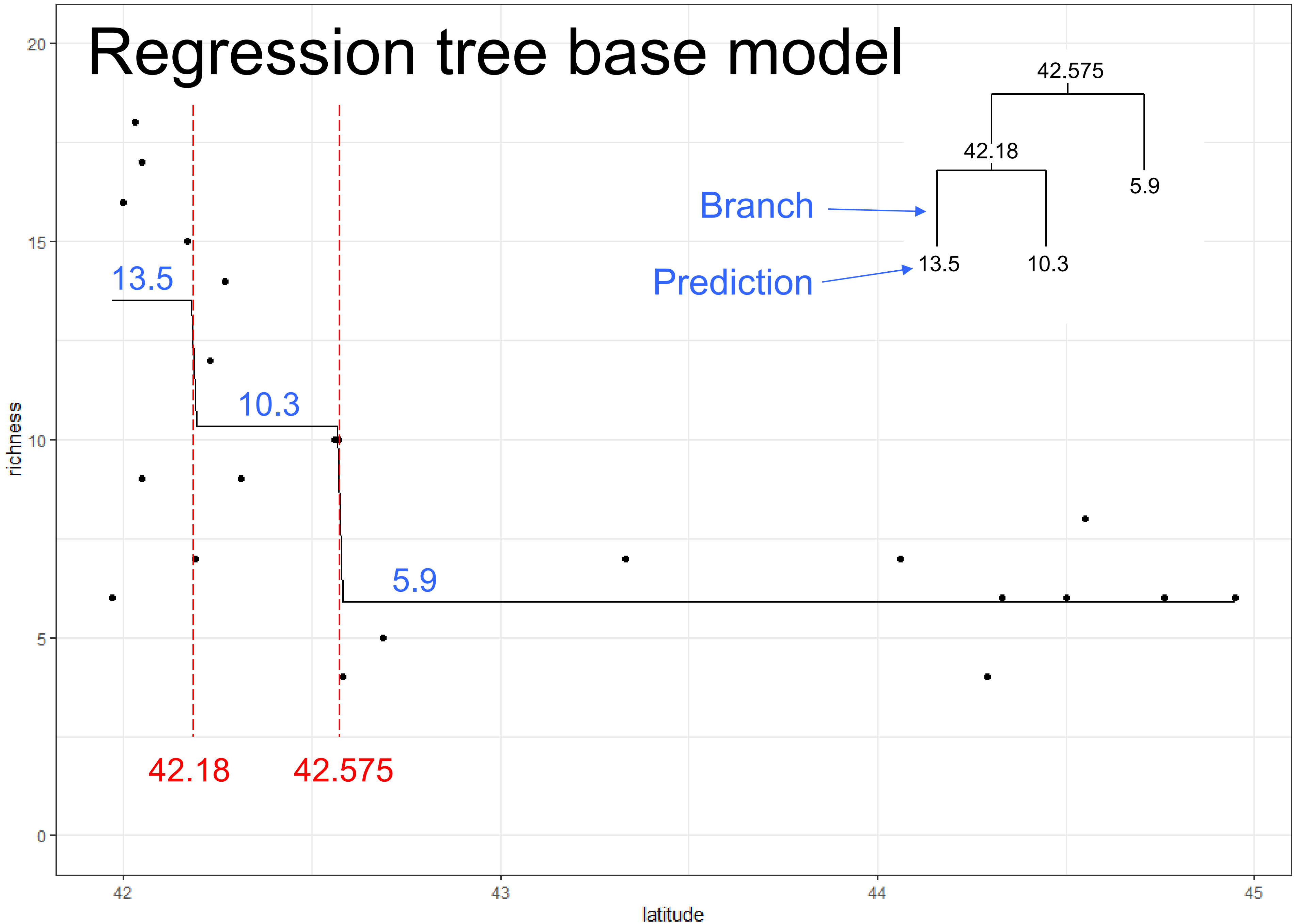
Basic model algorithms

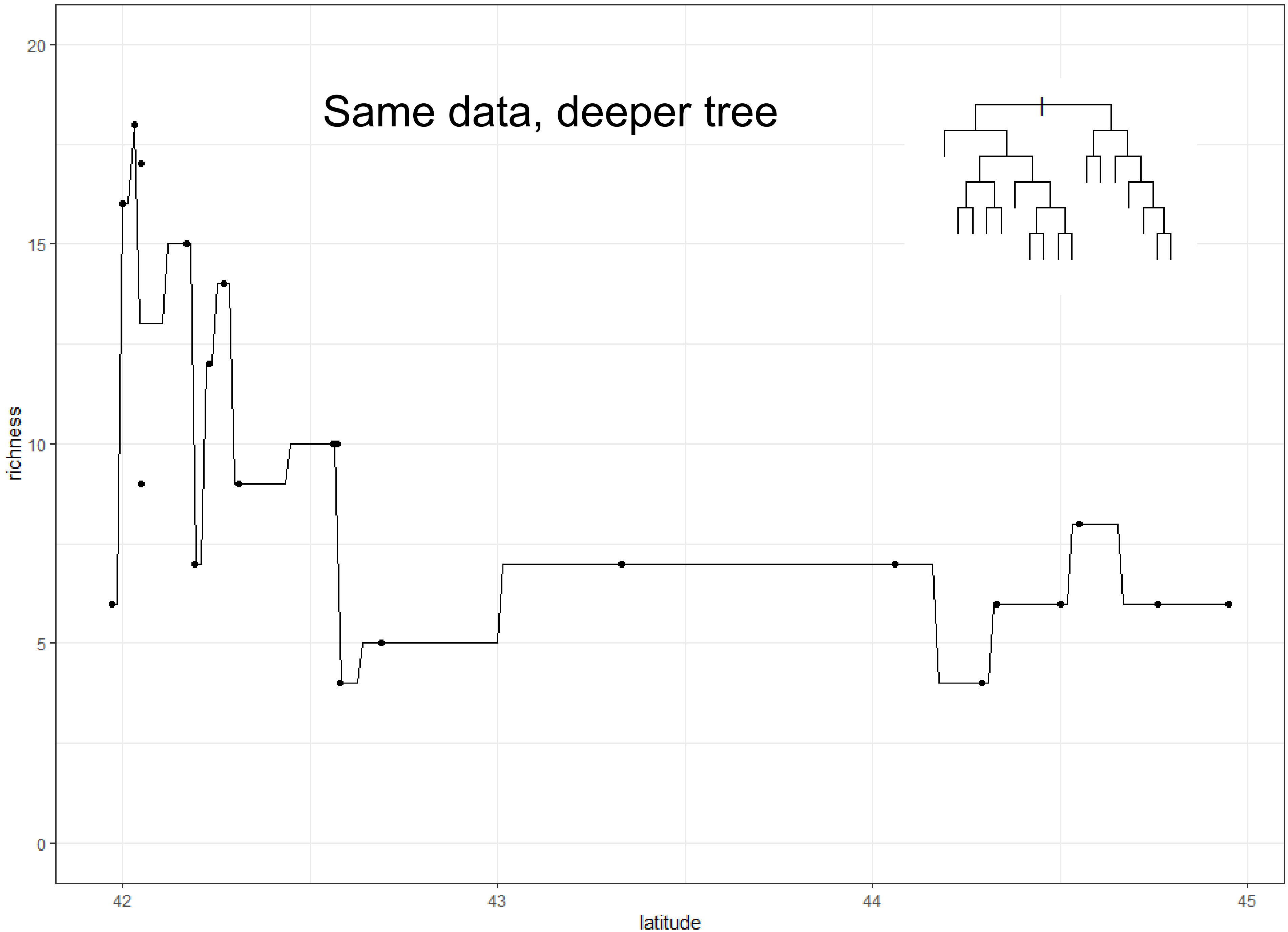
- Ensemble methods
  - Bagging
  - Random forest
  - Boosting
- Neural networks (various architectures)
  - feed forward
  - CNN
  - transformer

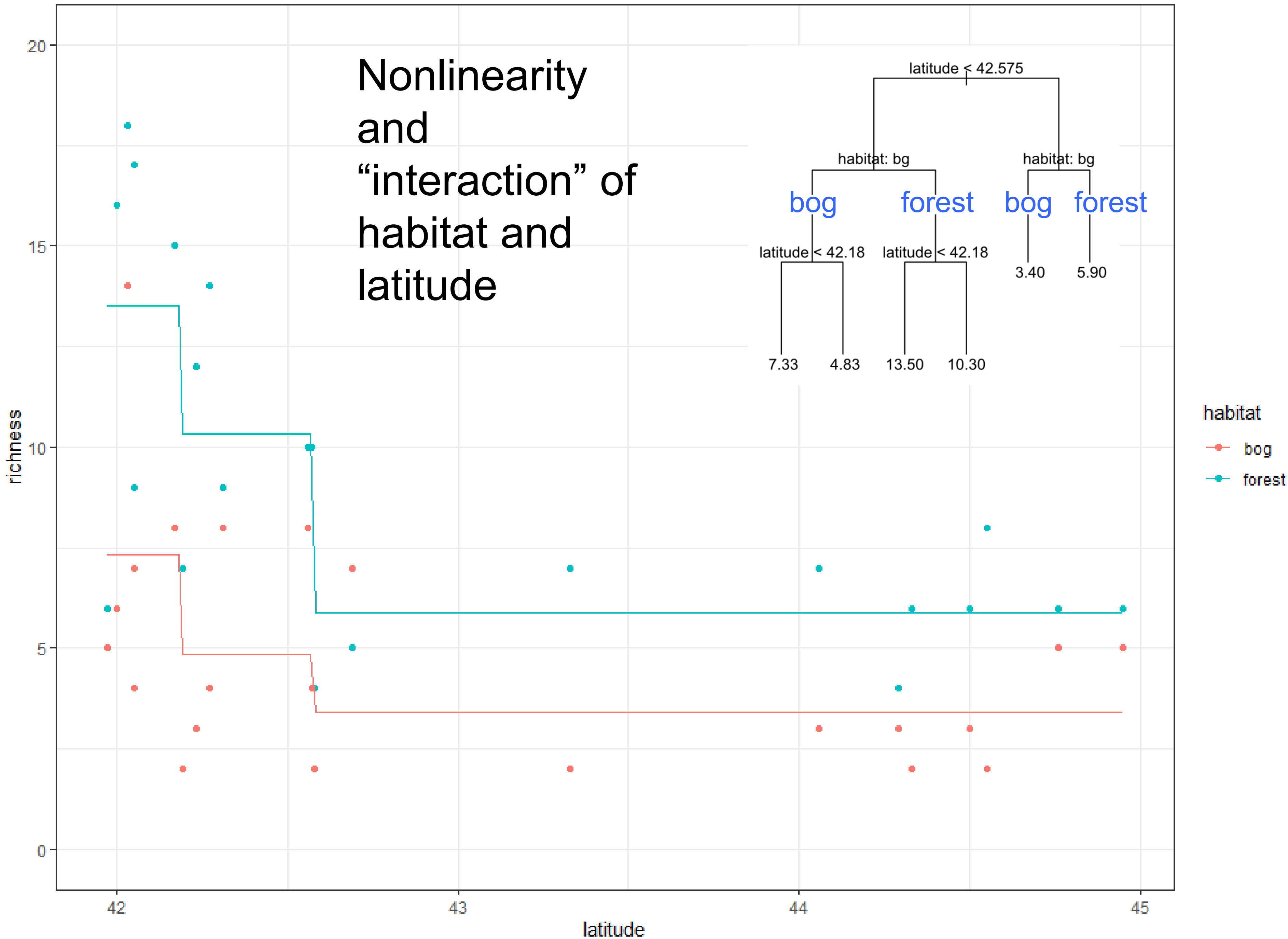
Regression tree base model



Regression tree base model




Same data, deeper tree
richness
latitude
0
5
10
15
20
42
43
44
45


Nonlinearity and “interaction” of habitat and latitude
latitude < 42.575
habitat: bg
habitat: bg
bog
forest
bog
forest
latitude < 42.18
latitude < 42.18
3.40
5.90
7.33
4.83
13.50
10.30
richness
latitude
habitat
bog
forest
20
15
10
5
0
42
43
44
45

Ensemble methods

- Train many models
- Average the models to predict
- Averaging reduces variance

Variance of a single model

$$\mathrm{Var}(\bar{y}) = \frac{\sigma_y^2}{n}$$

Number of models in ensemble

Variance of mean across models

Bagging

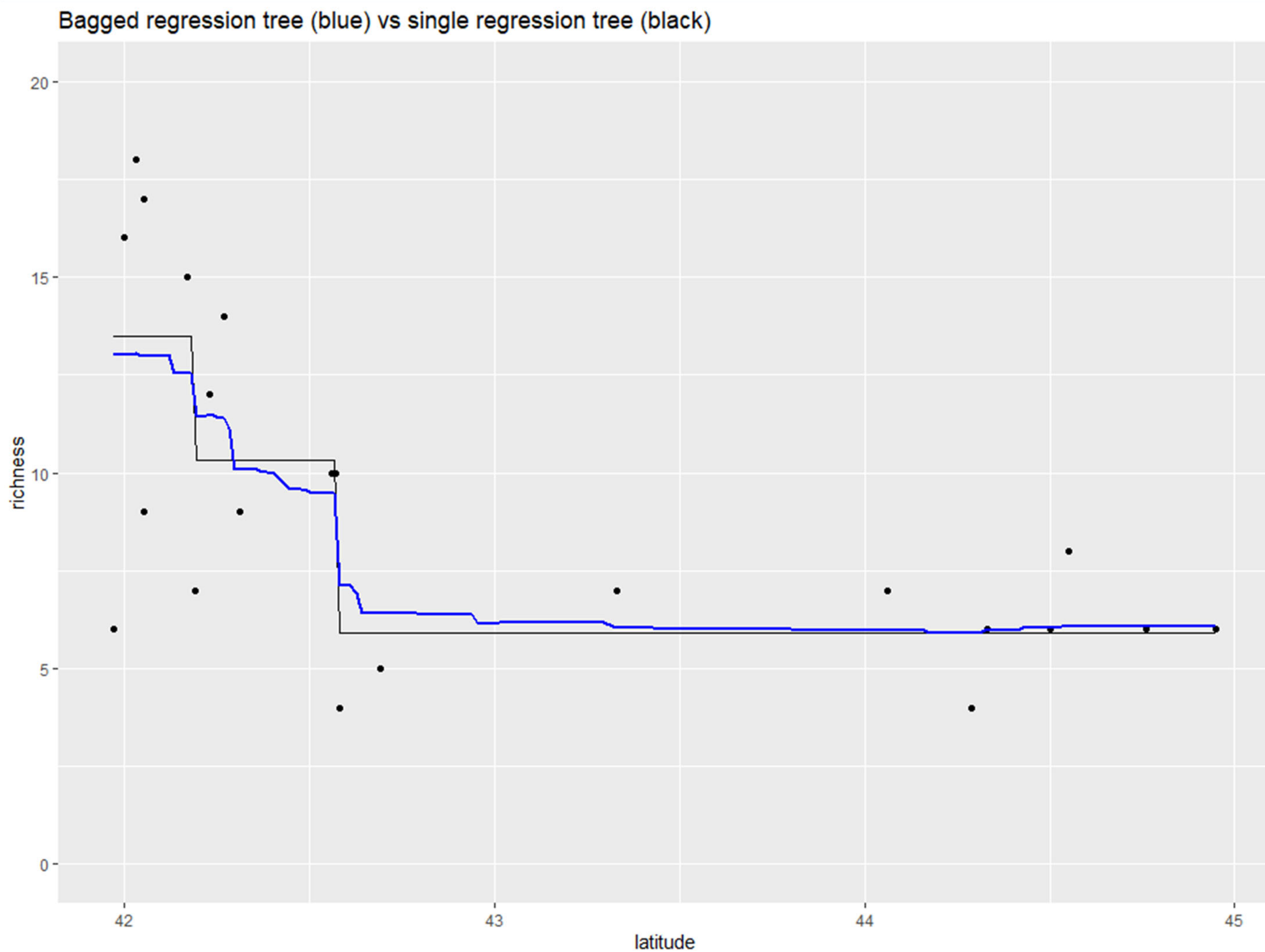
Bootstrap aggregation

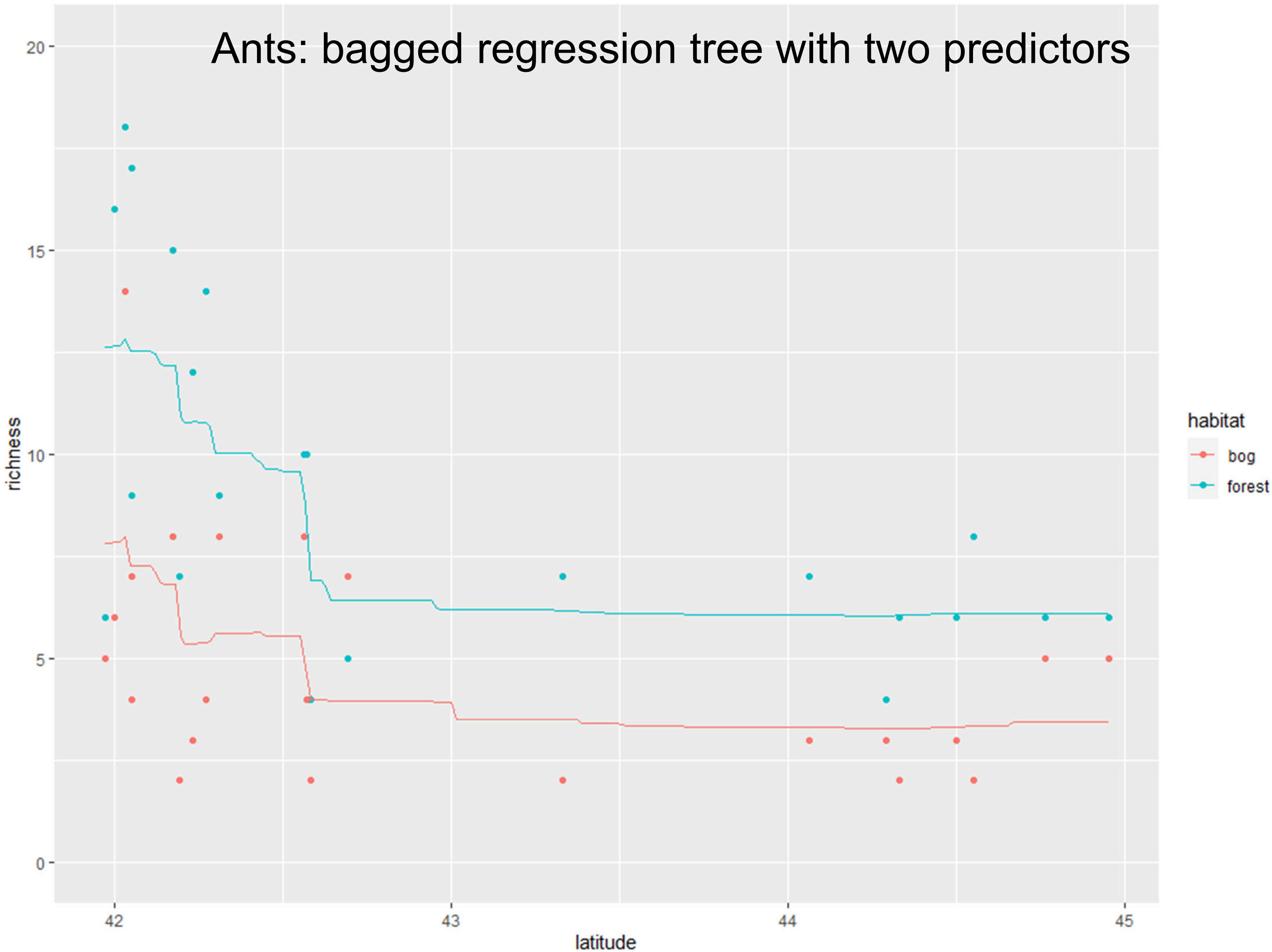
- Bootstrap
  - form new datasets by resampling from the data
- Aggregate
  - average over bootstrapped model fits

Bagging algorithm

for many repetitions
    resample the data with replacement
    train the base model
    record prediction
final prediction = mean of predictions

Base model: can be any type of model


Bagged regression tree (blue) vs single regression tree (black)
richness
latitude
0
5
10
15
20
42
43
44
45


Ants: bagged regression tree with two predictors
richness
latitude
20
15
10
5
0
42
43
44
45
habitat
bog
forest

Random forest

Builds on bagging algorithm

Algorithm

for many repetitions

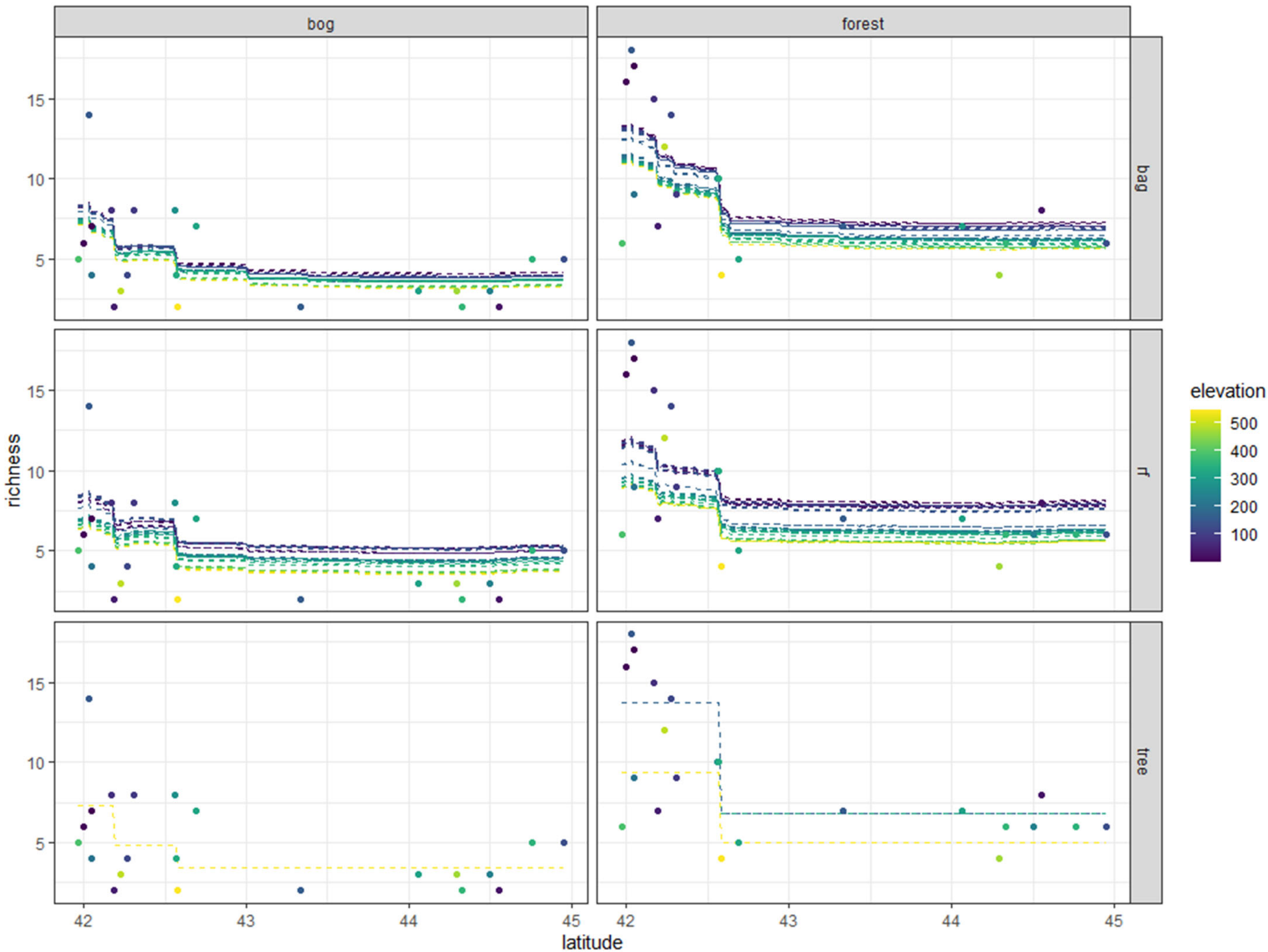
    randomly select m predictor variables

    resample the data (rows) with replacement

    train the tree model

    record prediction

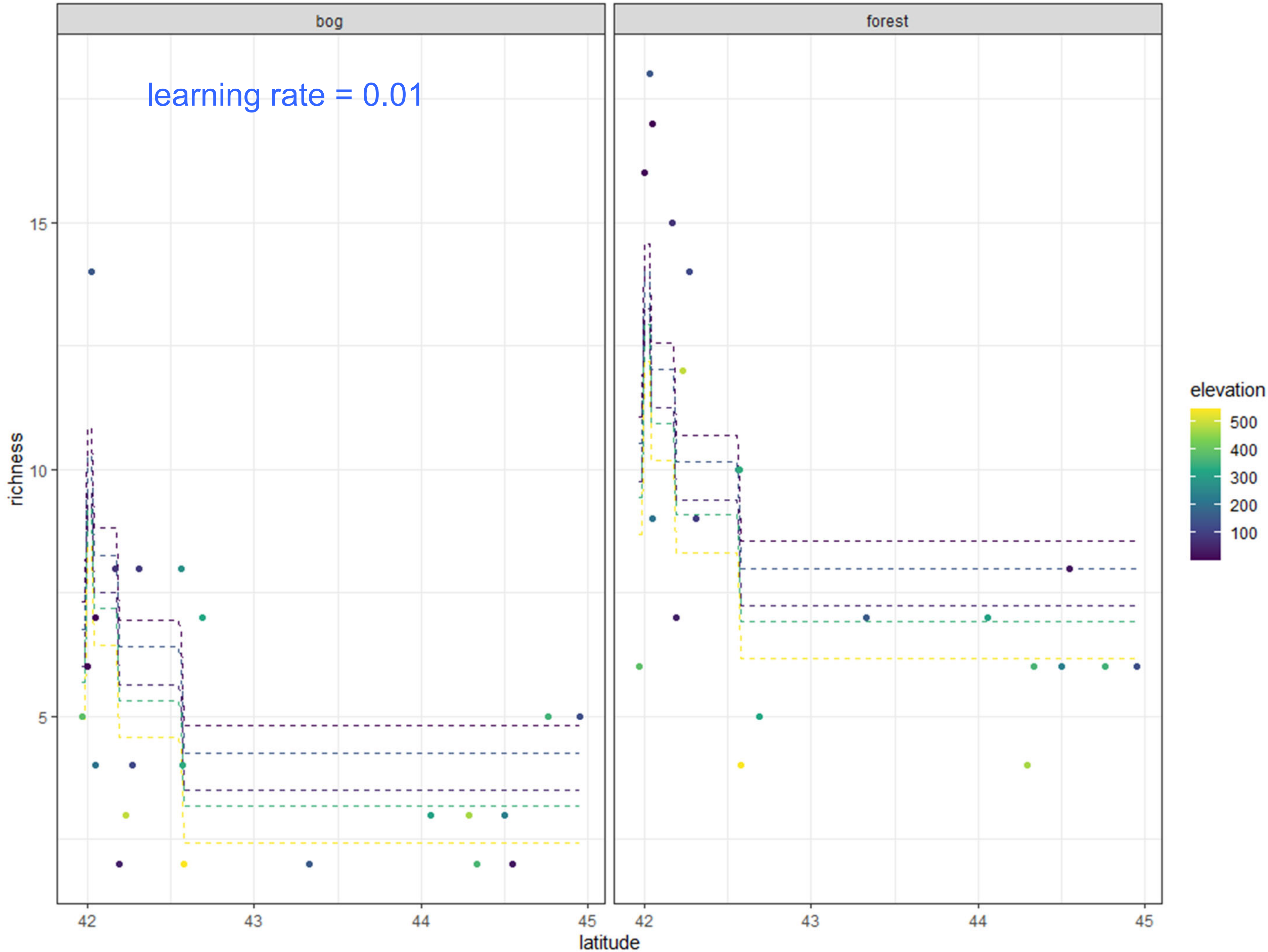
final prediction = mean of predictions



bog
forest
bag
rf
tree
richness
latitude
elevation
500
400
300
200
100
15
10
5
42
43
44
45

Boosting algorithm

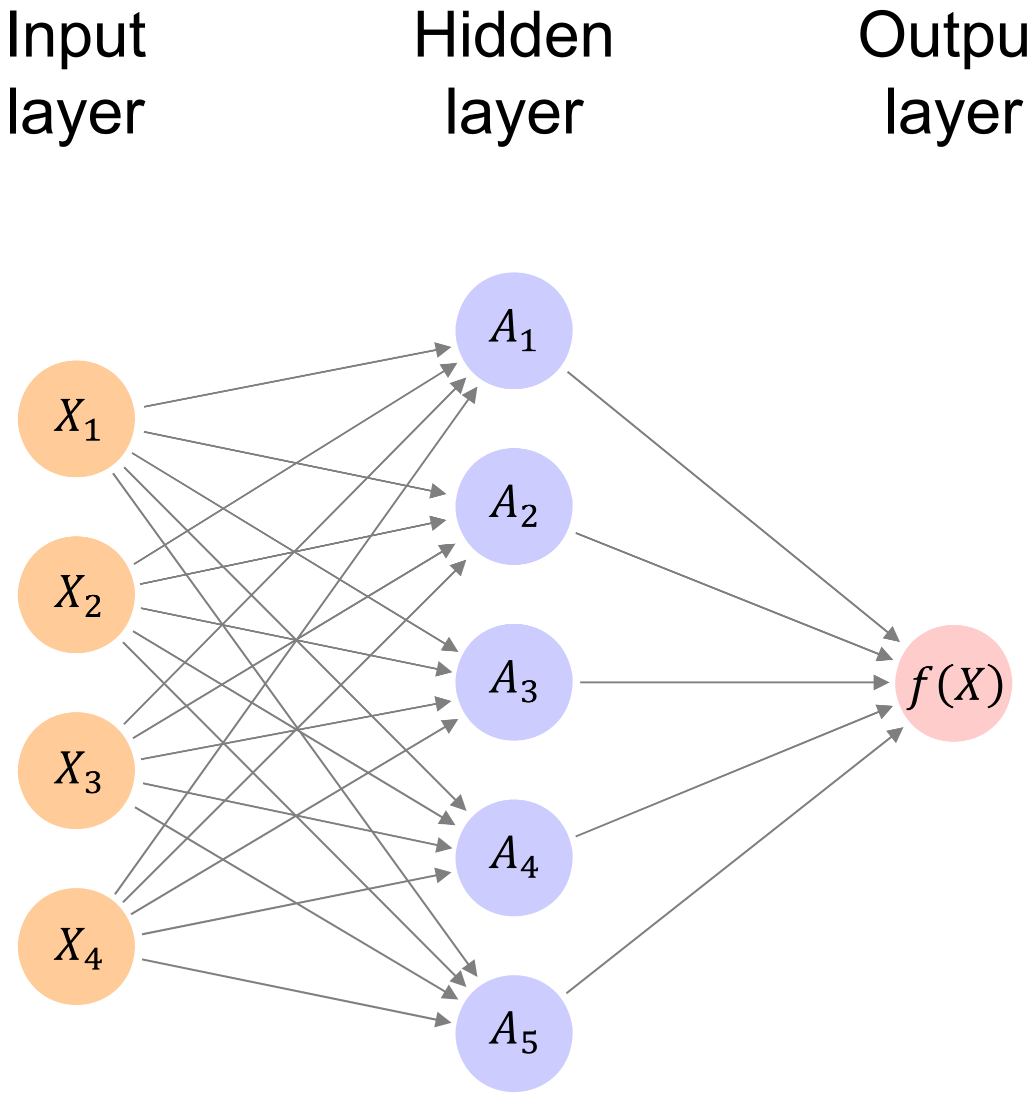
- Too complex for this quick overview
- Basic idea: learn slowly by building up an ensemble iteratively from many models, each with small weight
- Key tuning parameter: learning rate
- Can include aspects of bagging and RF
- Training algorithm: gradient descent
- State of the art: xgboost


bog
forest
learning rate = 0.01
richness
15
10
5
42
43
44
45
latitude
elevation
500
400
300
200
100

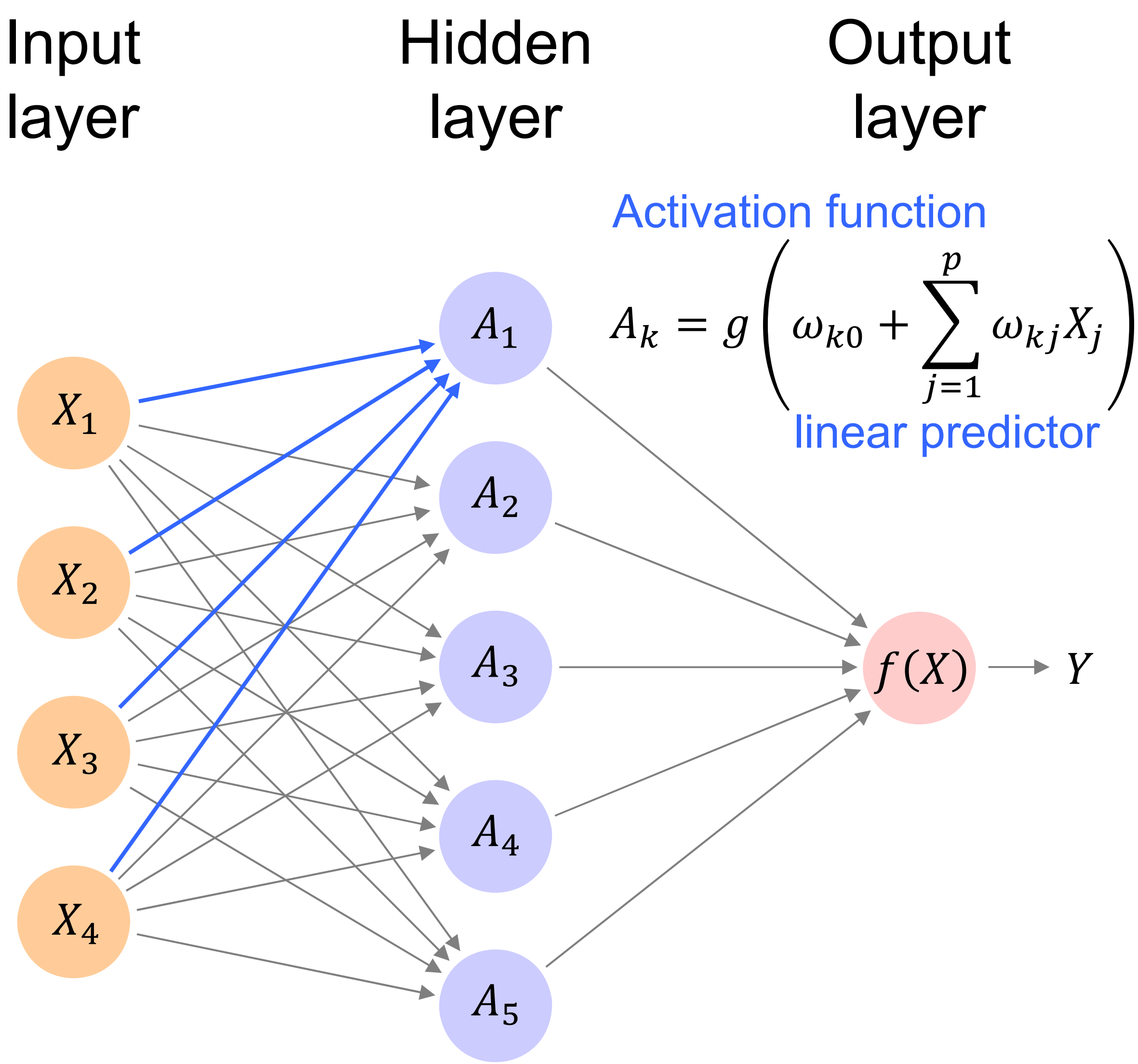
Neural Networks

- R and Python interfaces to:
  - keras/tensorflow - google
  - torch/pytorch - facebook

Single layer NN

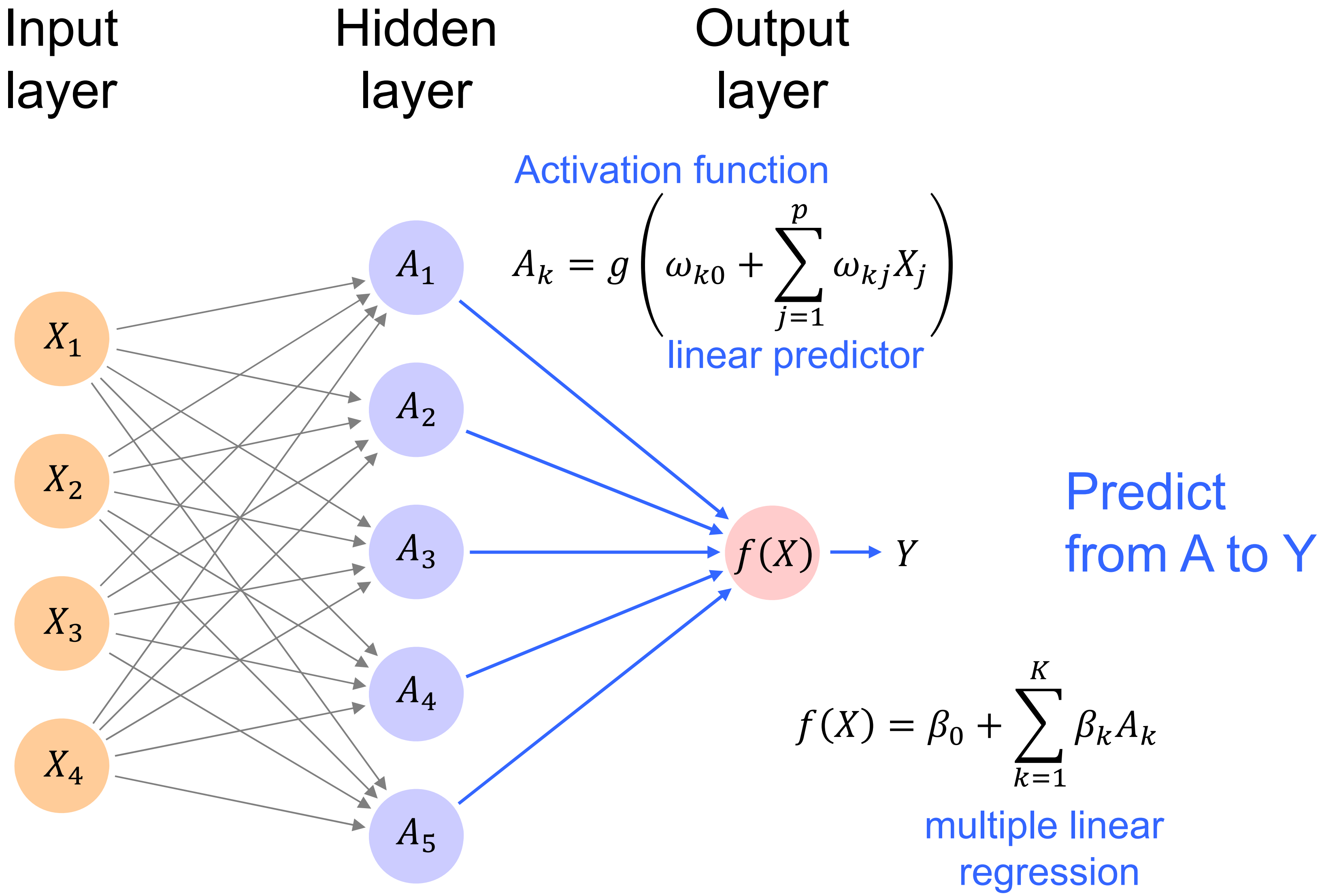


Single layer NN



Transform from X to A

Single layer NN



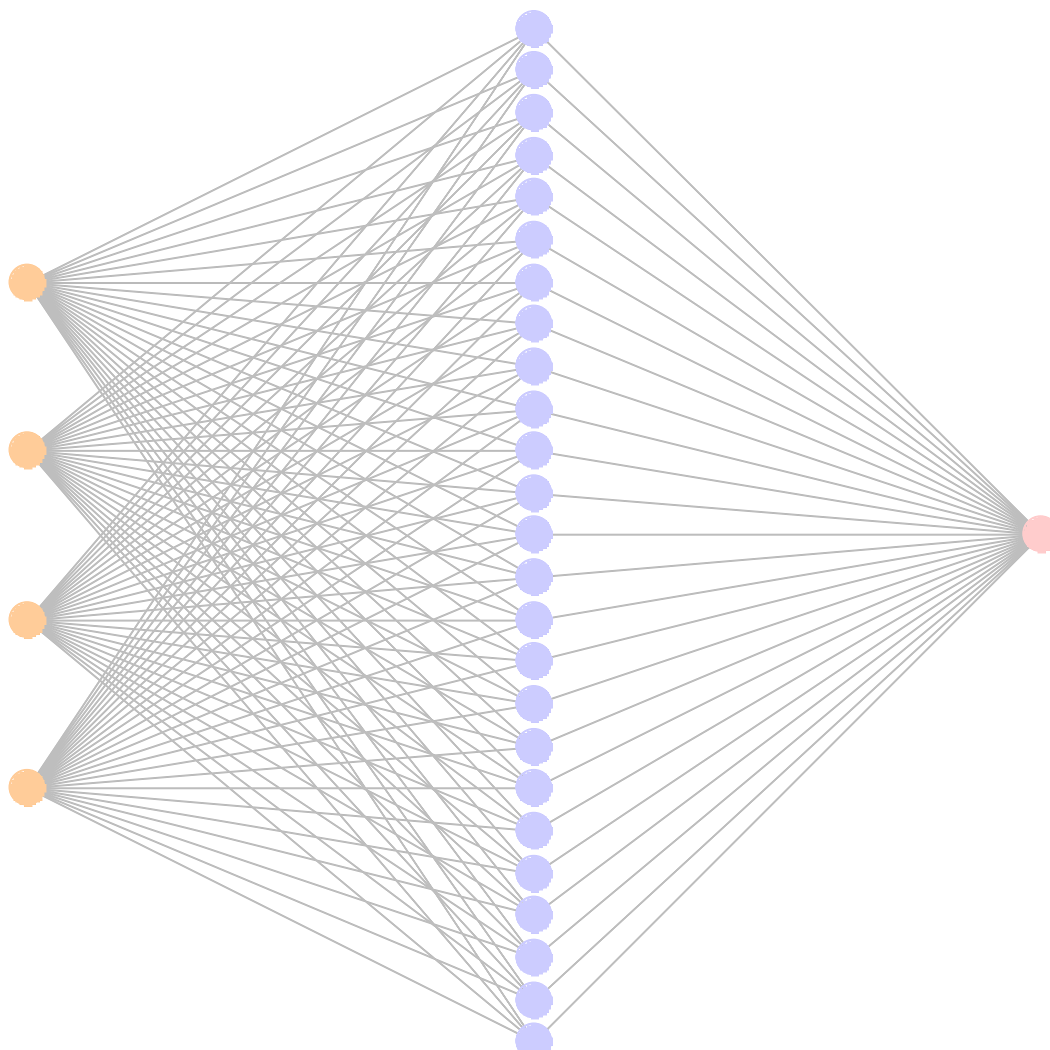
Training algorithm

- Stochastic gradient descent
- Model and training algorithms incorporate many previous strategies, e.g.
  - random subsets of data (like bagging)
  - random subsets of nodes (like RF)
- Thus, parameters are regularized
  - any one parameter has small effect

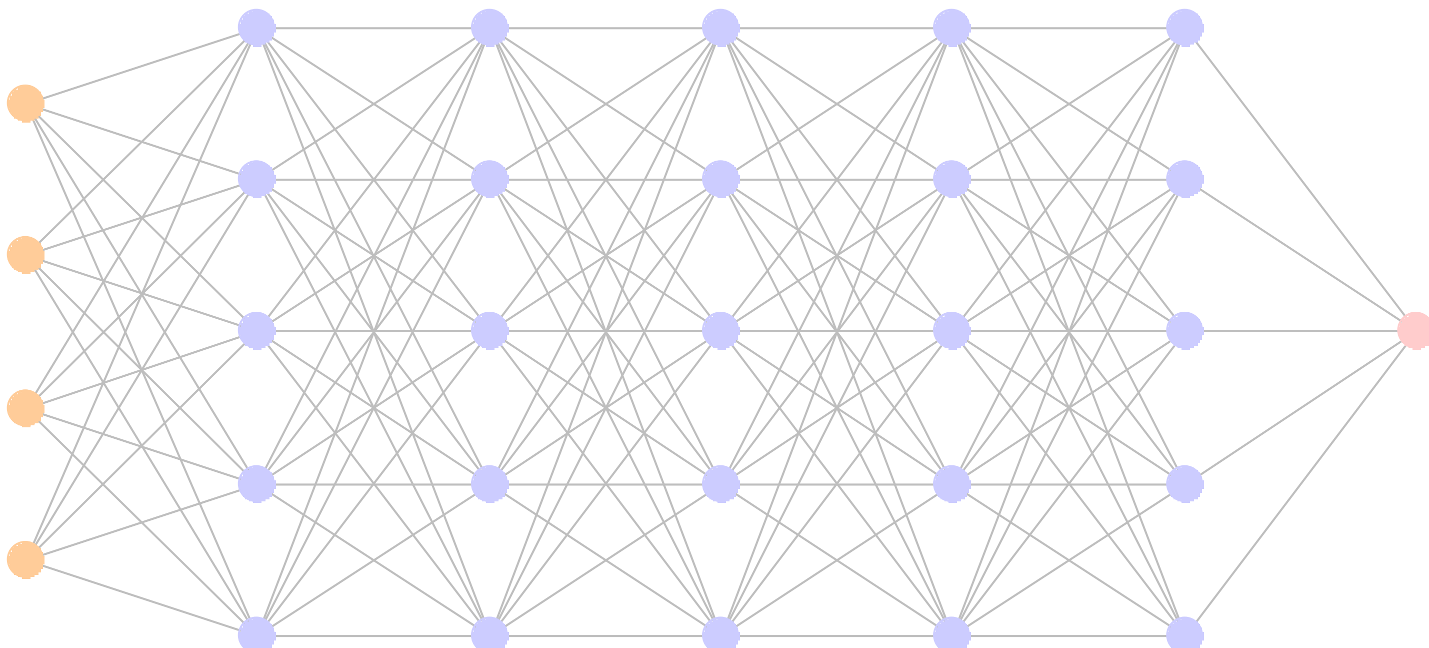
Deep learning

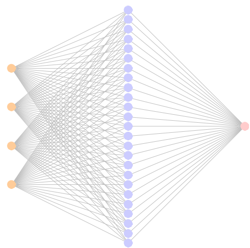
- Multilayer neural networks
- Model algorithm
  - expressiveness
    - ability to approximate complex nonlinearity
  - architecture: width versus depth
  - deep networks approximate nonlinearity efficiently

Wide: 25 hidden units

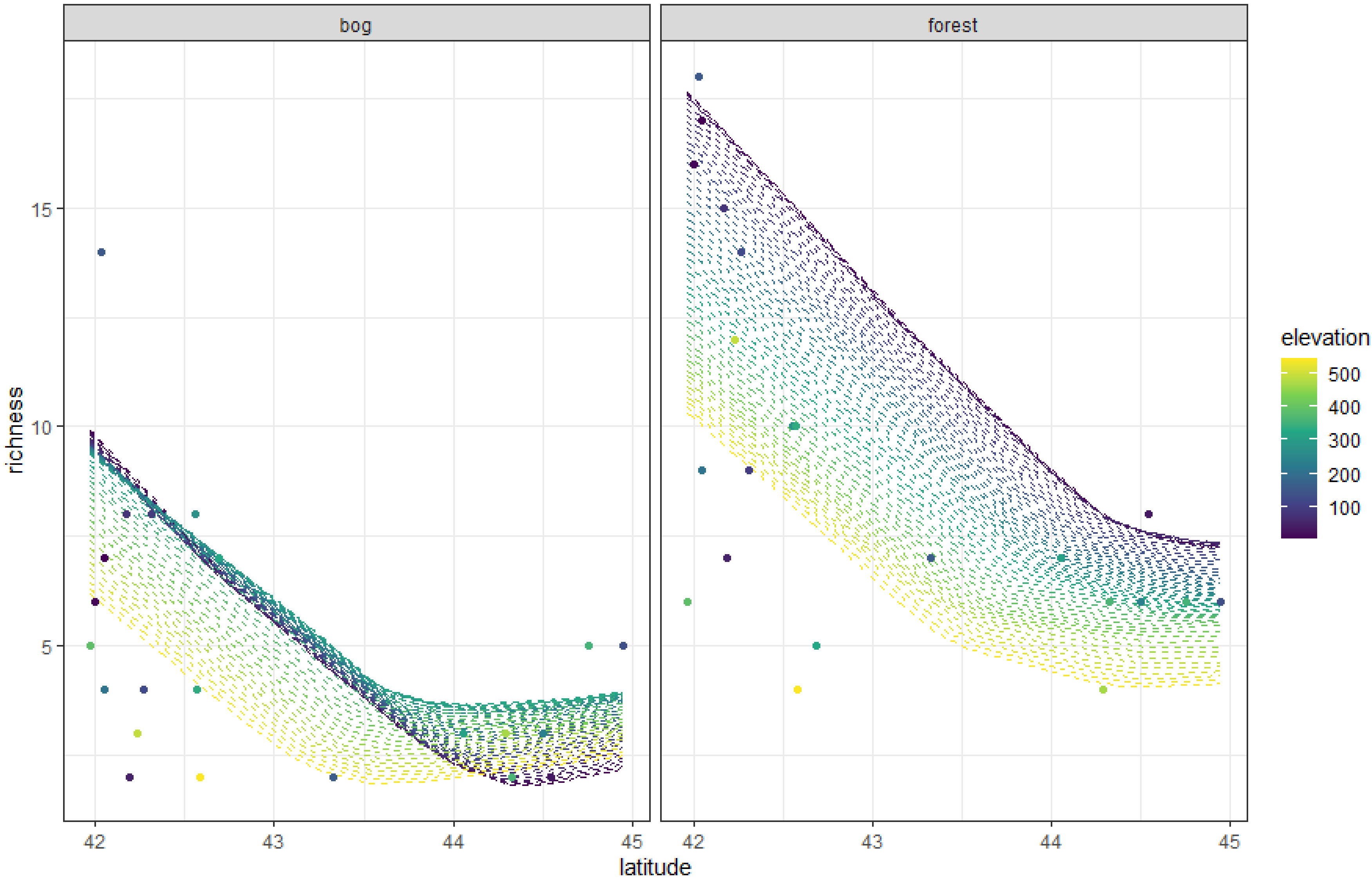


Deep: 25 hidden units

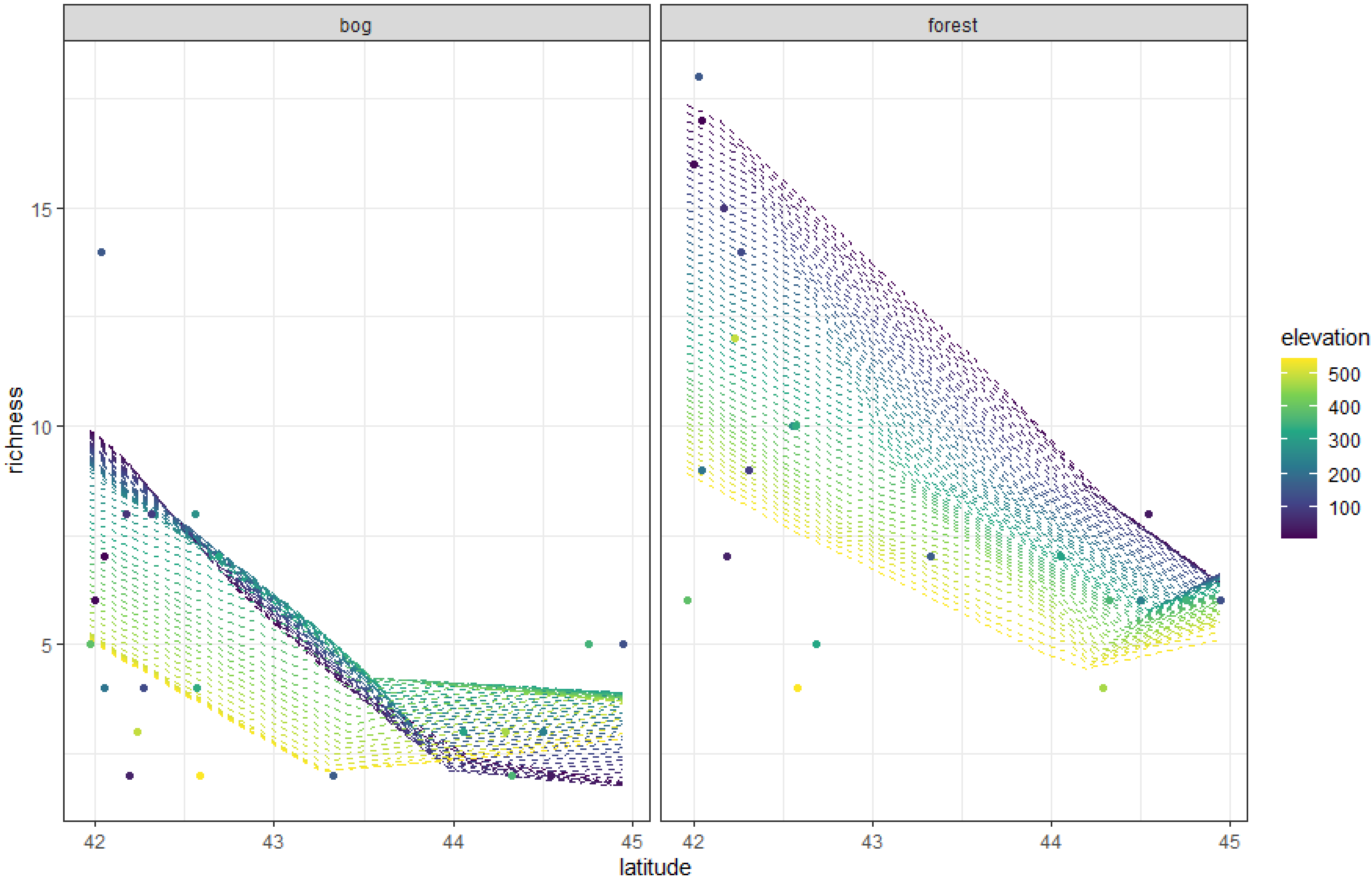




Ants data: wide



Ants data: deep

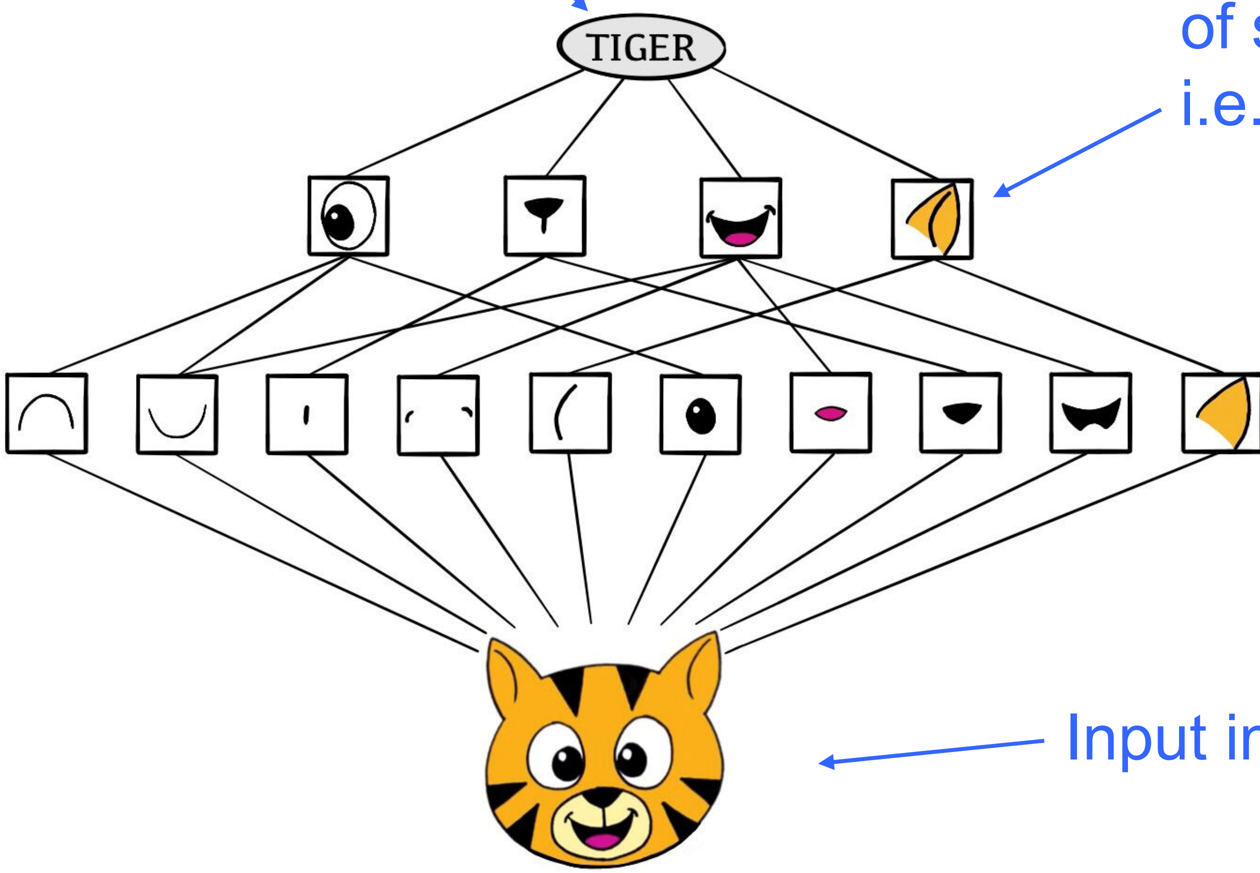


Convolutional NNs

Output is high-level concept

Conv layer 2 learns combinations of small features i.e. objects

Conv layer 1 learns small features e.g. edges



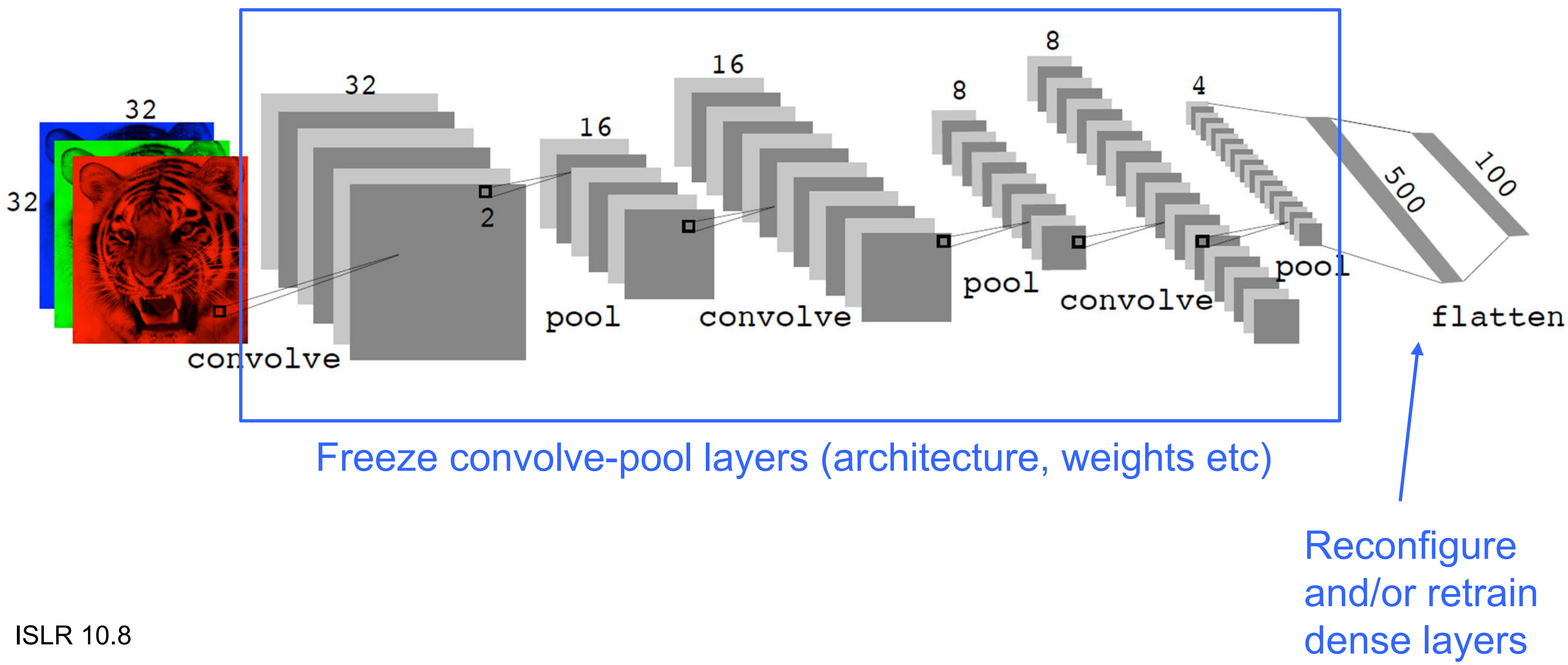
Input image

ISLR 10.6

Transfer learning

Pretrained model on related big data

Retrain last 1-2 layers on specialized little data



ISLR 10.8

Rapid innovation

- Architectures
- Algorithms
- Recent example: transformer
  – > GPT

Outlook

Automated data collection

+

machine learning

=

revolutionary