Tree based approaches for Prediction

Alistair Johnson, DPhil
Miguel Armengol, PhD Candidate

Learning goals of this workshop

- Understand the simplest models, decision trees, and the most complex, gradient boosting

- Understand the intuition behind improvements in tree models

- Be able to apply these models yourself!

Topics covered

- **Background**: Iris dataset
- **Background**: Regression
- Trees
- Bagging
- Boosting
- Bootstrap aggregation
- Gradient boosting

Background

Fisher’s Iris

Origins of the Iris dataset

- Darwin compared hybrids with purebreds
  - Inbred << Hybrids
- Asked his cousin Galton for help
  - **σ** !
- “I doubt after making many trials, whether it is possible to derive useful conclusions from these few observations. We ought to have measurement of **at least fifty plants in each case**, in order to be in a position to deduce fair results.”

Toadflax

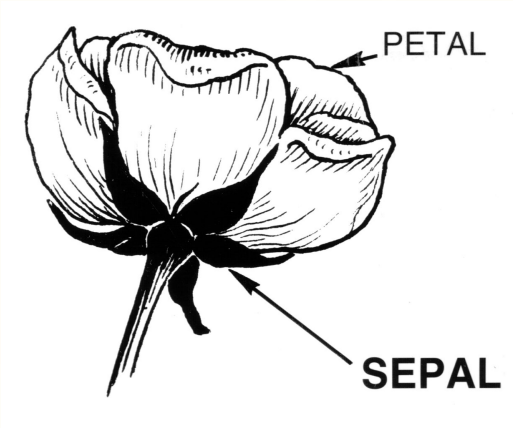
Fisher’s Iris dataset

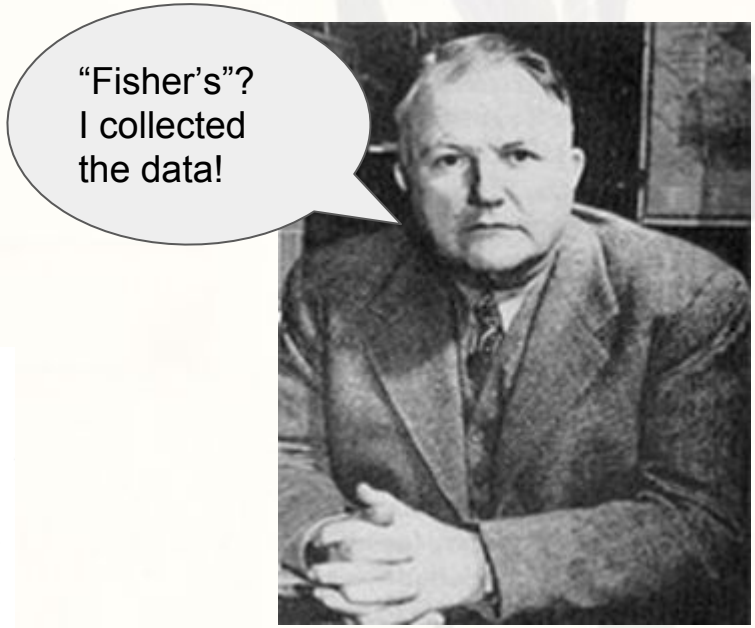
Introduced in 1936

Dataset used to demonstrate classification

Petal/sepal measurements

50 flowers, 3 species





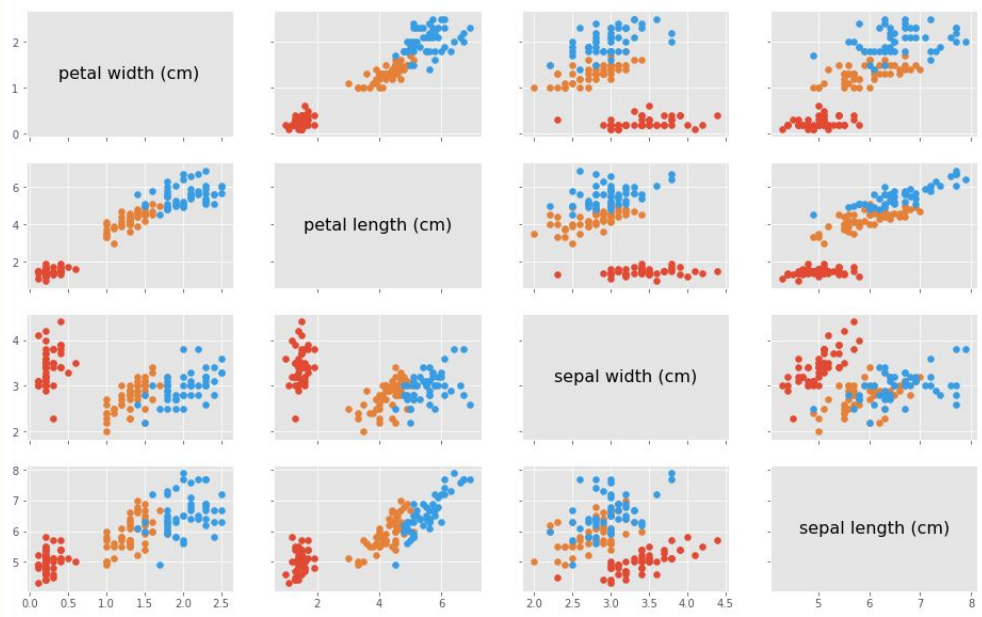
Edgar Anderson

Iris dataset

setosa

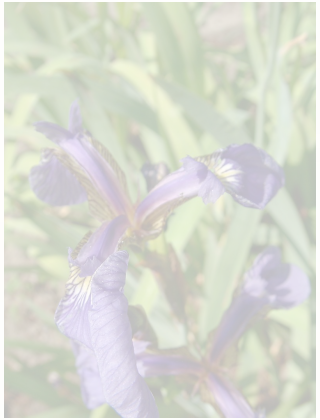
versicolor

virginica



Note: Setosa (Red) is very easy to distinguish

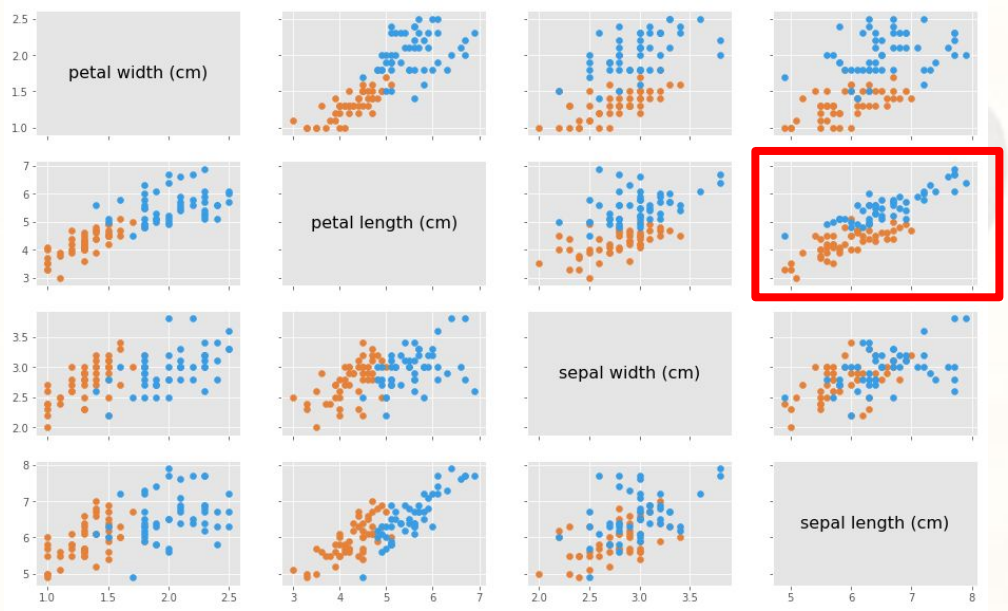
Iris dataset - ignore setosa for simplicity



setosa

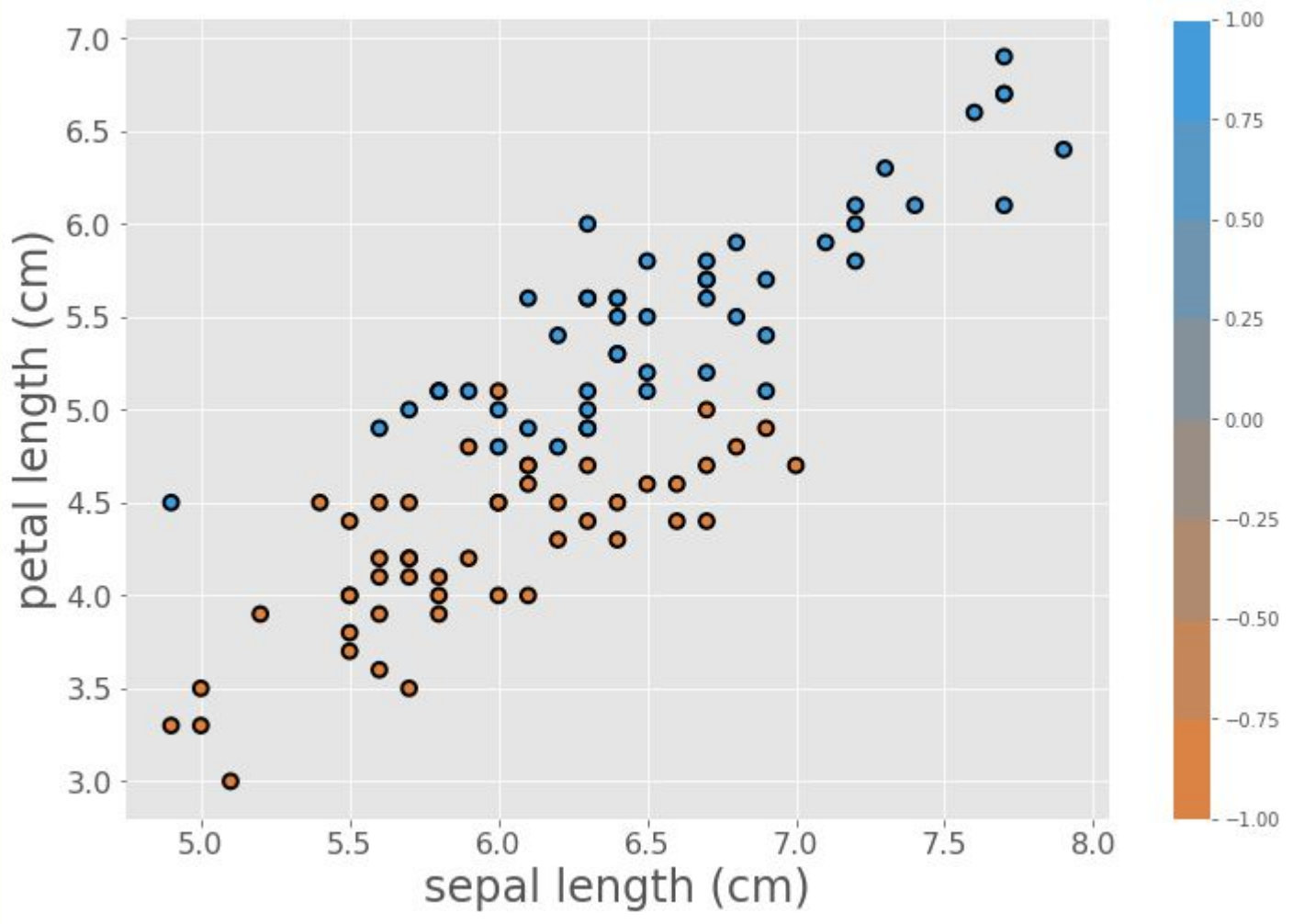
versicolor

virginica



Let’s look at petal length vs. sepal length

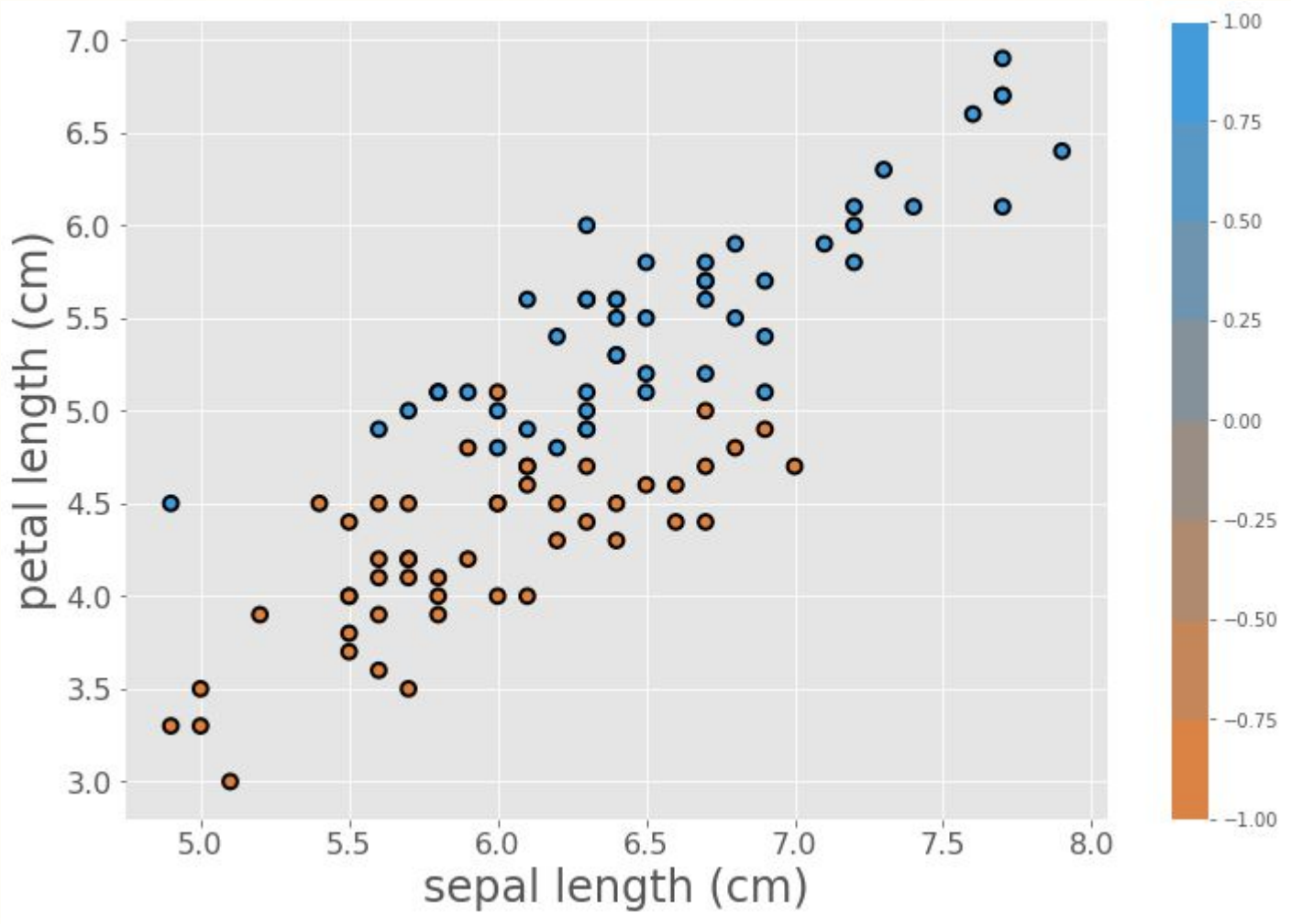
Data


petal length (cm)
sepal length (cm)
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
5.0
5.5
6.0
6.5
7.0
7.5
8.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Data

We will try to classify this data with two models:

- Regression
- Decision tree



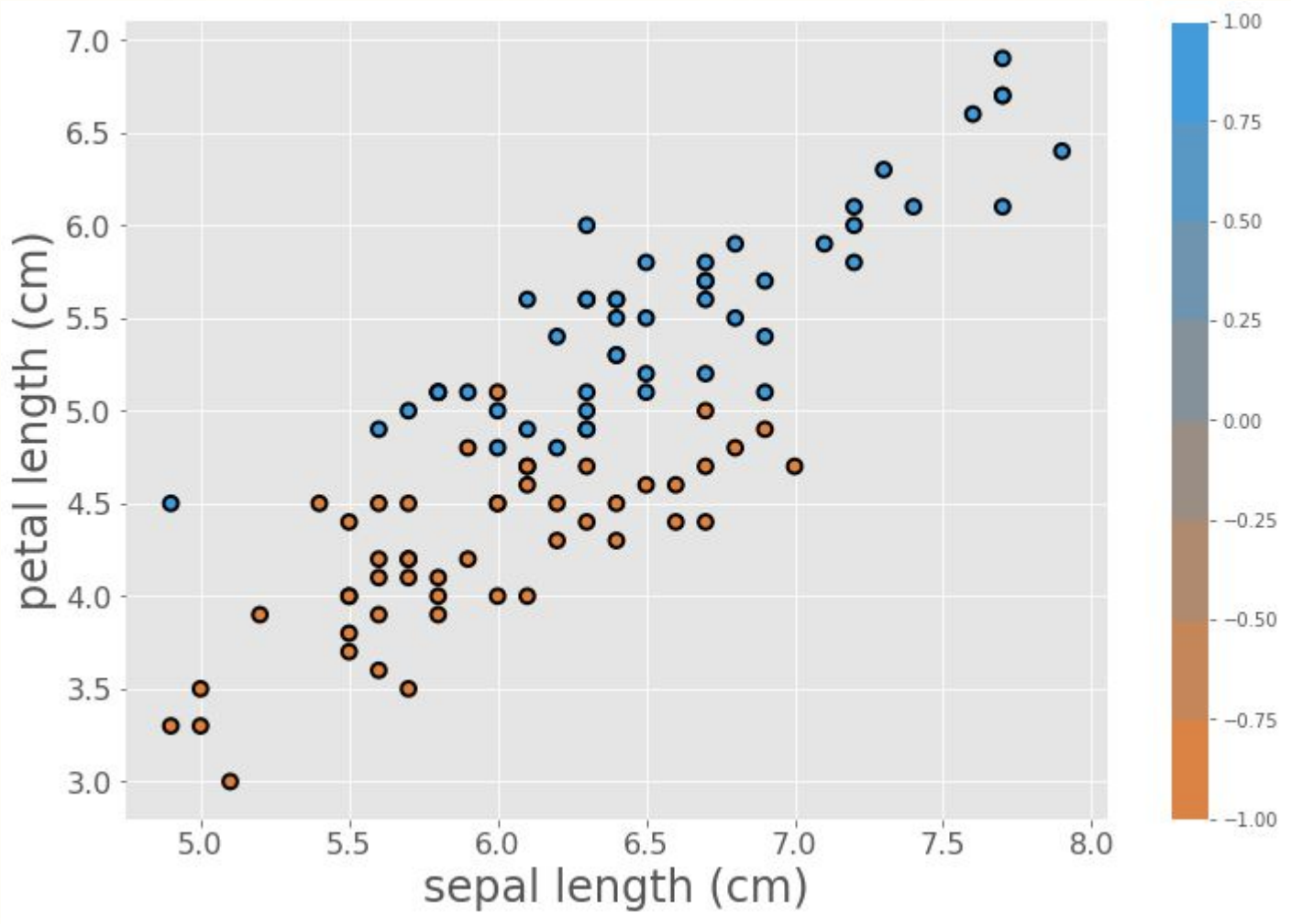
Regression

- Map one (or more) variables to a target

$y = mx + b$

OR

$y = b_1x_1 + b_0$



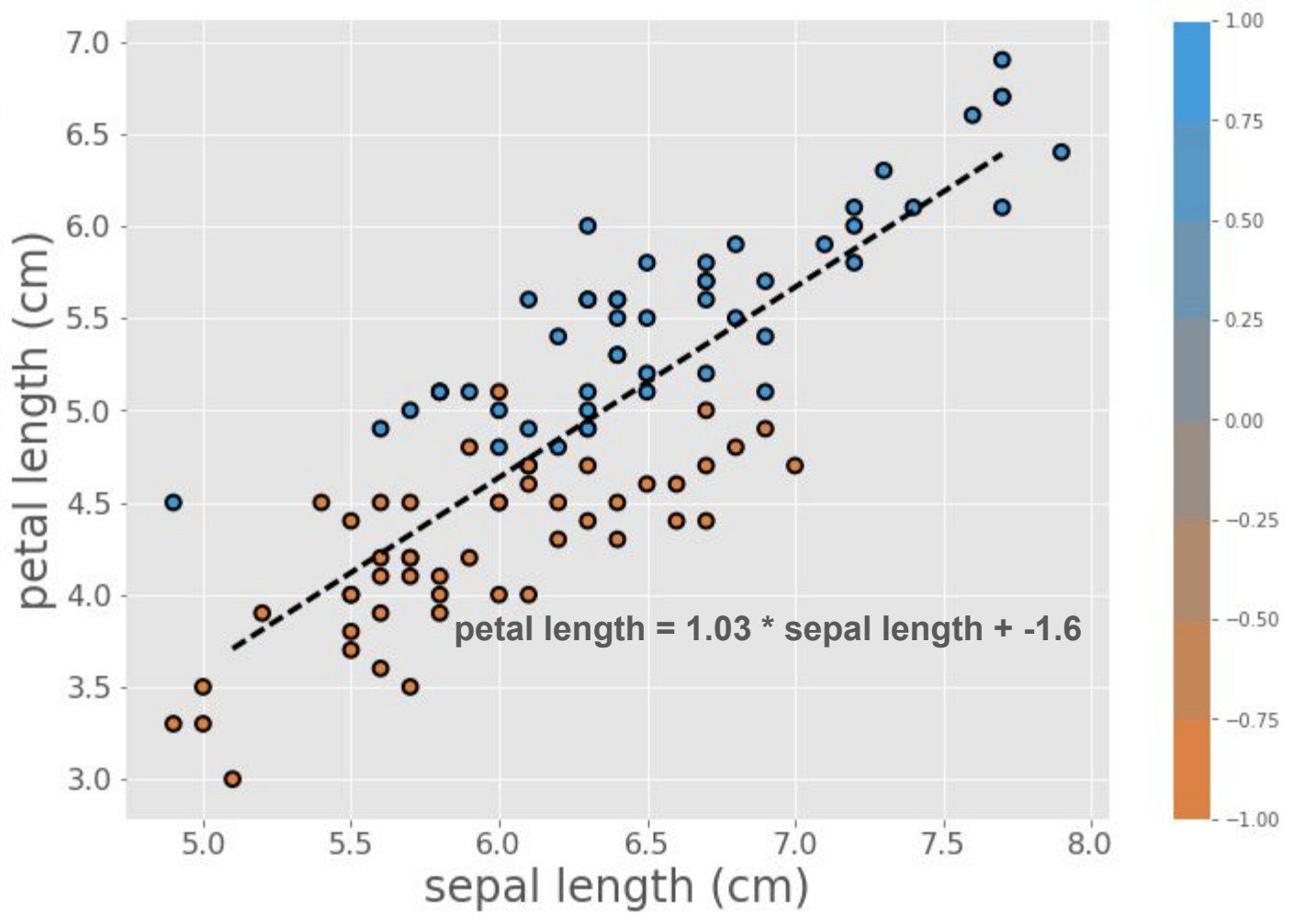
Regression

- Map one (or more) variables to a target

$y = mx + b$

OR

$y = b_1x_1 + b_0$



Regression

- Map one (or more) variables to a target

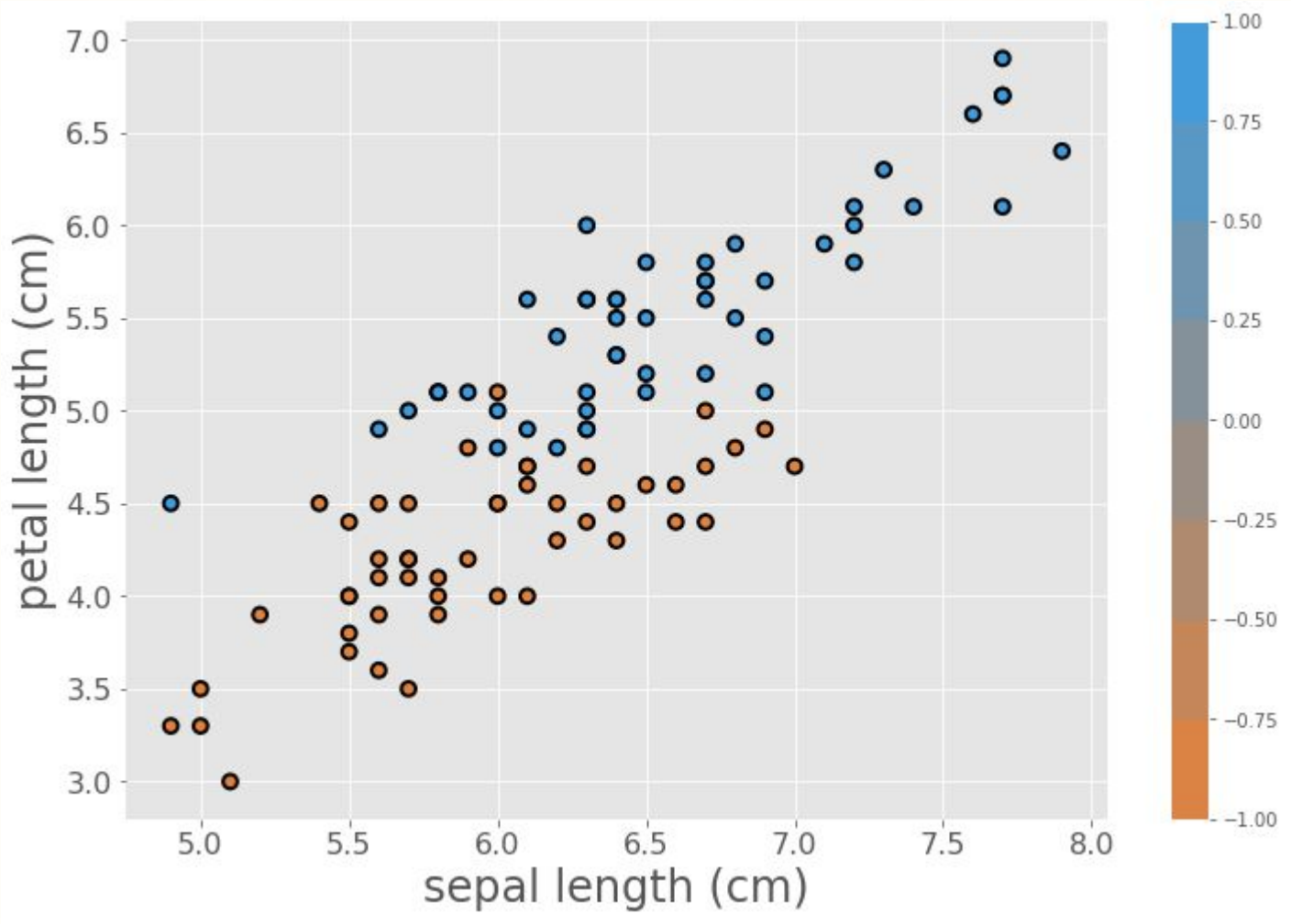
$y = mx + b$

OR

$y = b_1x_1 + b_0$

$y = b_2x_2 + b_1x_1 + b_0$

Iris Class {-1, 1}

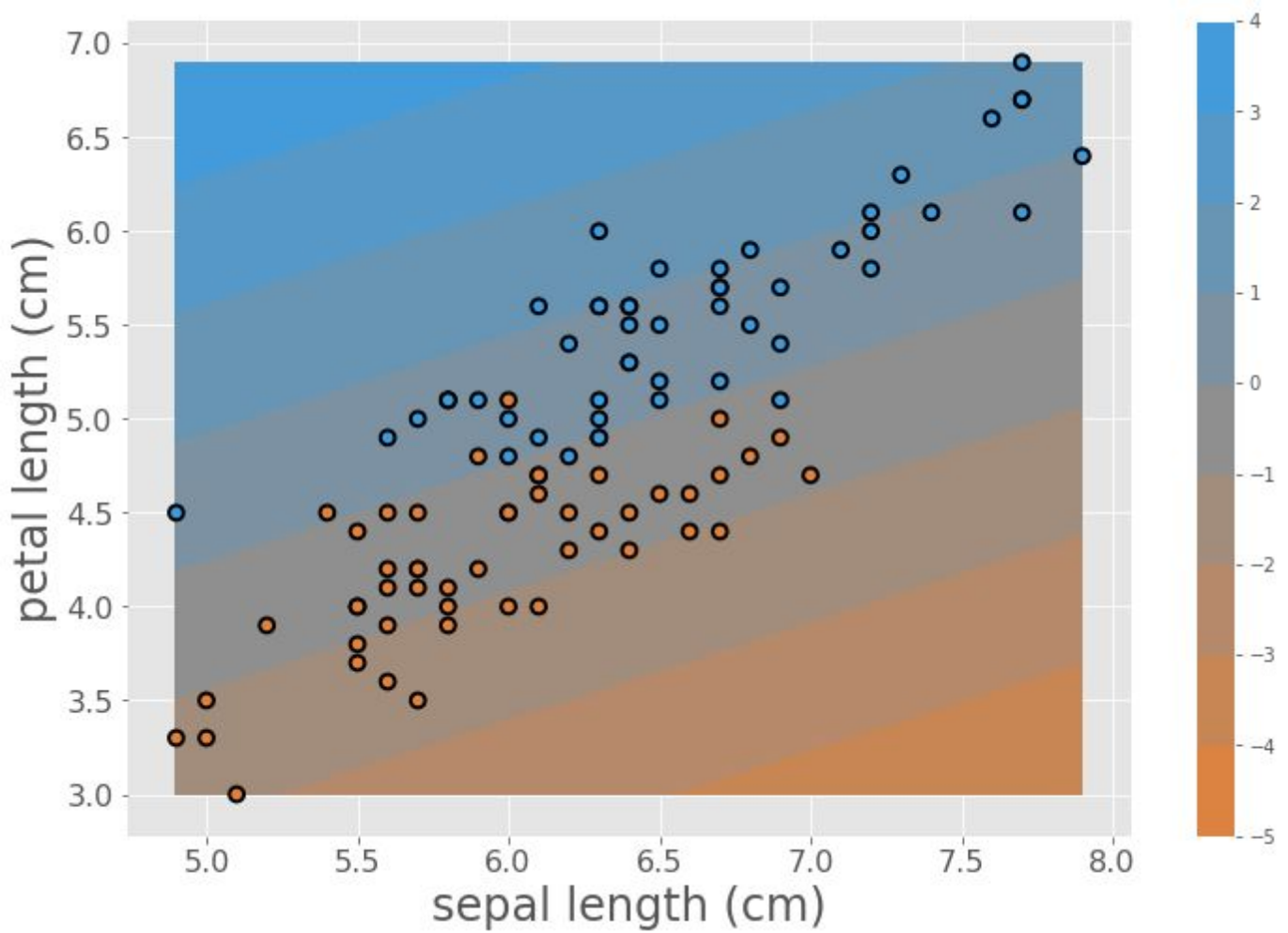


Regression

- Map one (or more) variables to a target

$y = b_2x_2 + b_1x_1 + b_0$

Iris Class {-1, 1}



Regression

$y = b_2x_2 + b_1x_1 + b_0$

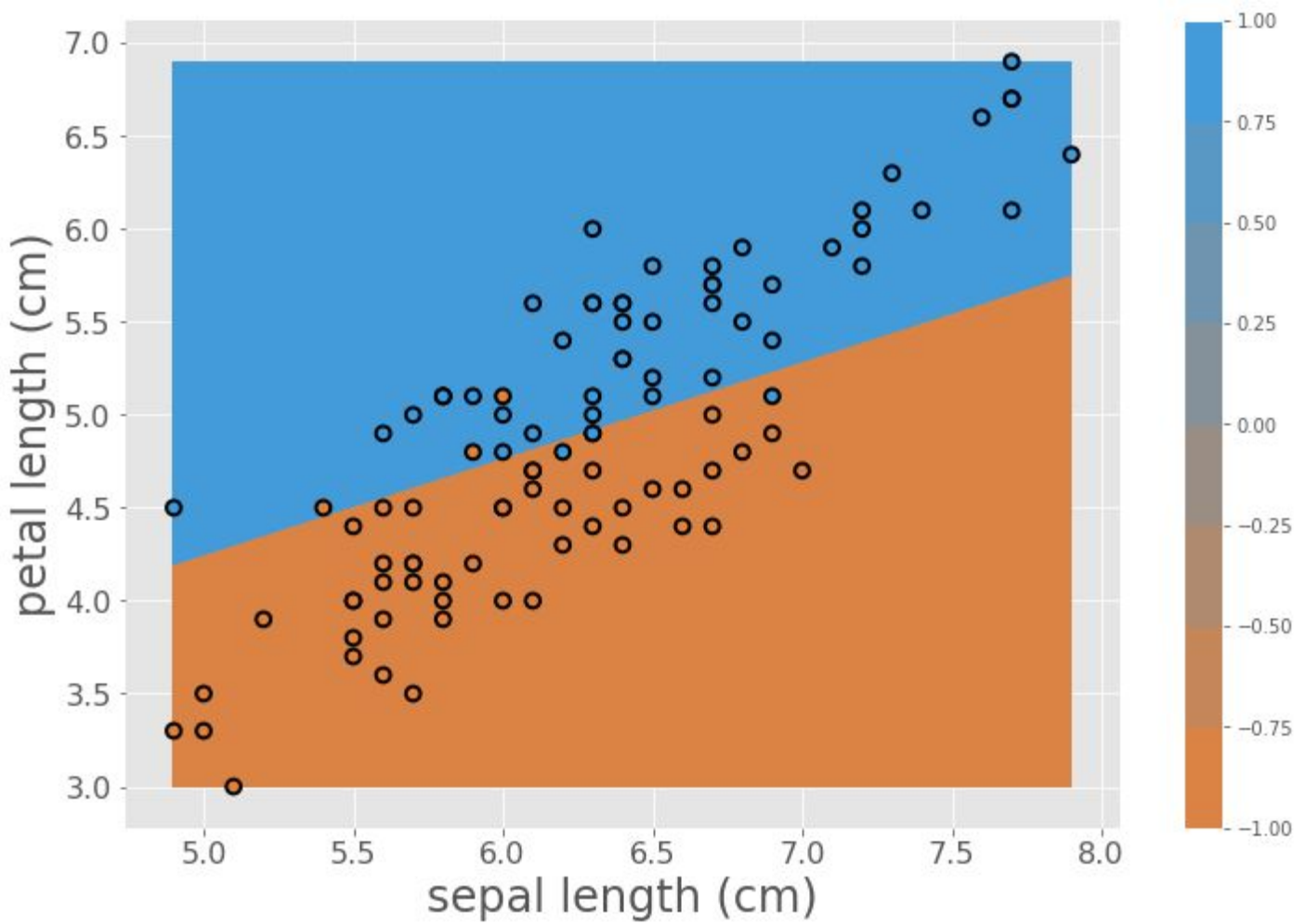
… if y > 0

-> blue

… else

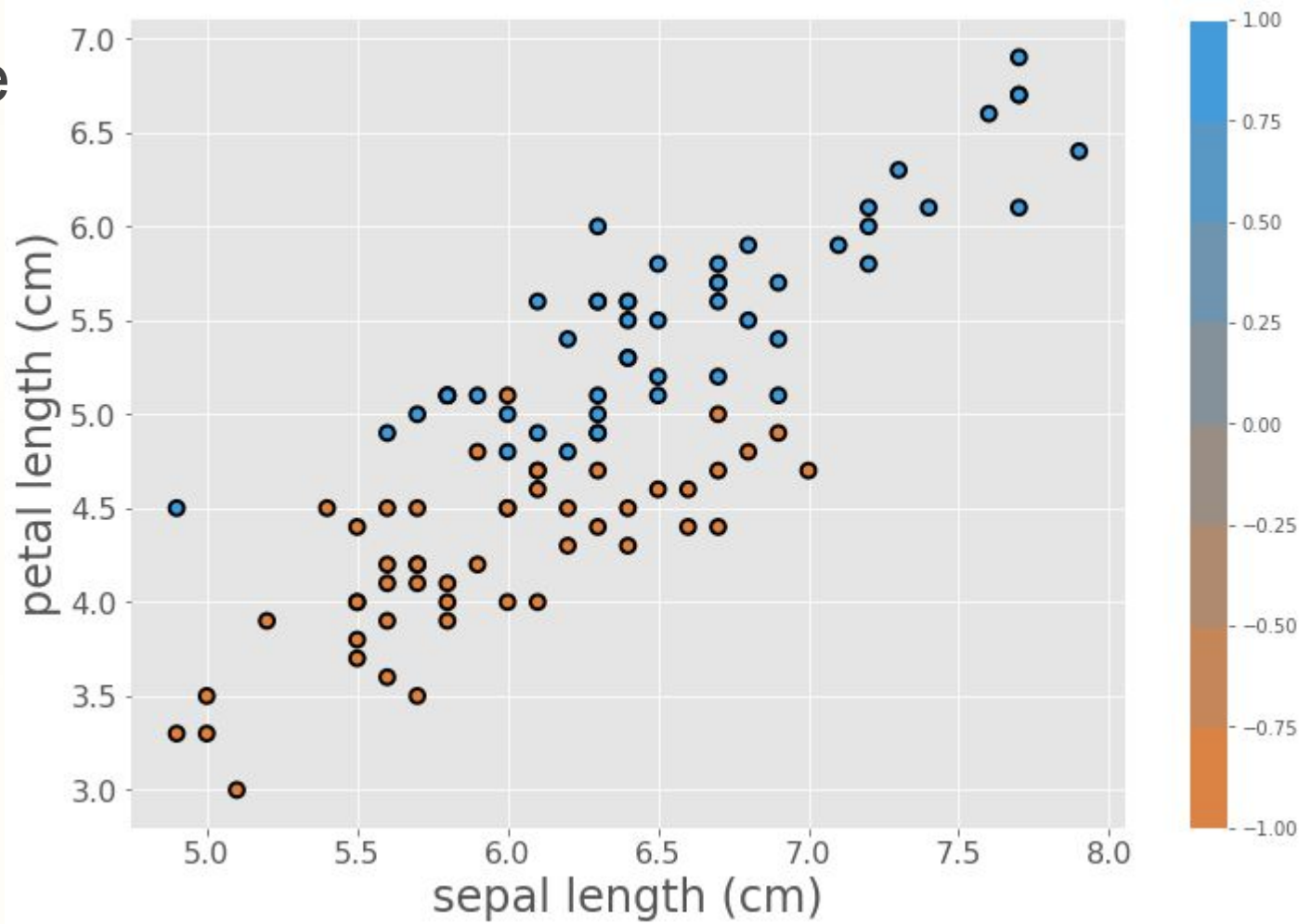
-> orange

<u>linear boundary</u>



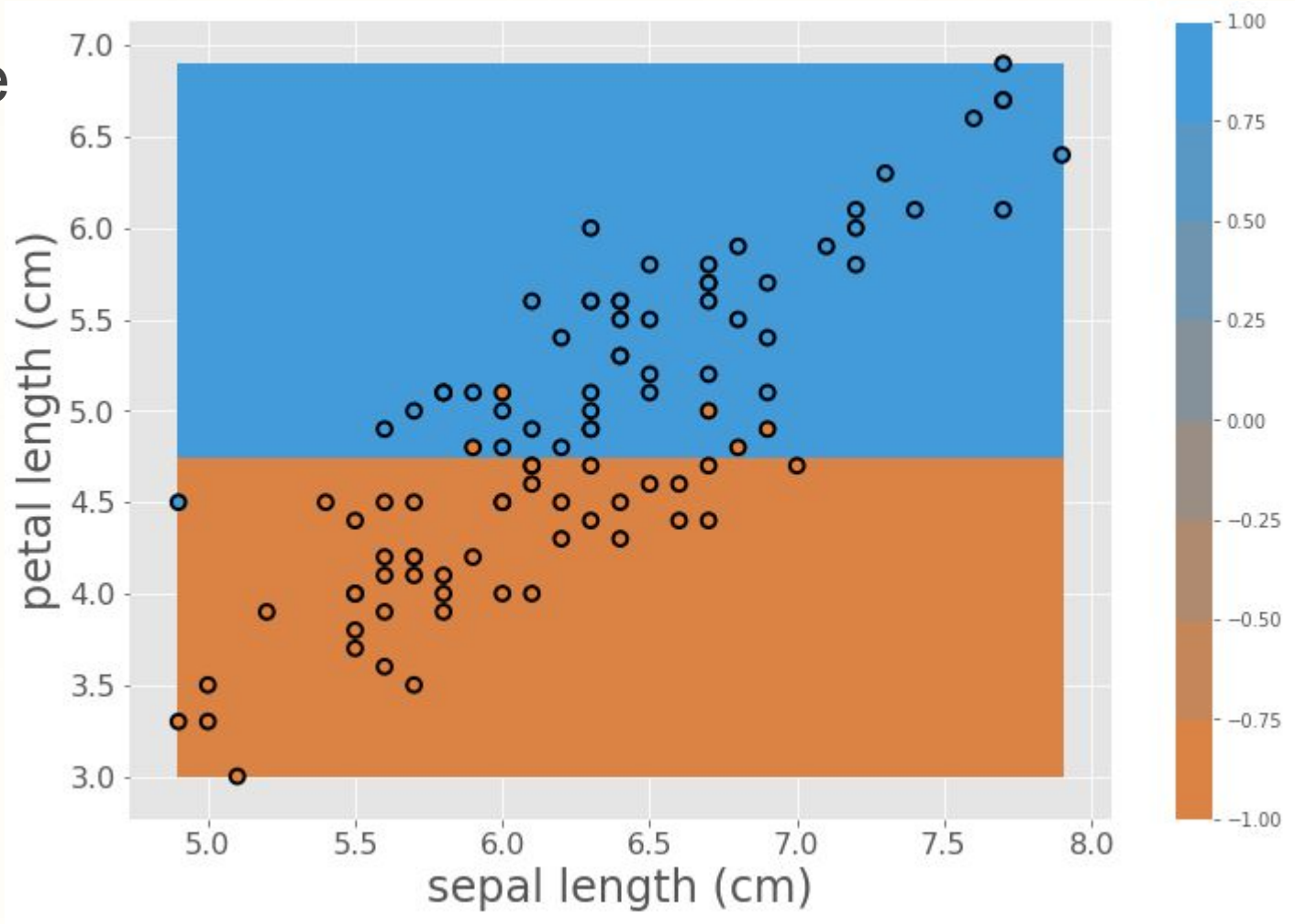
Decision tree

Split data into two groups



Decision tree

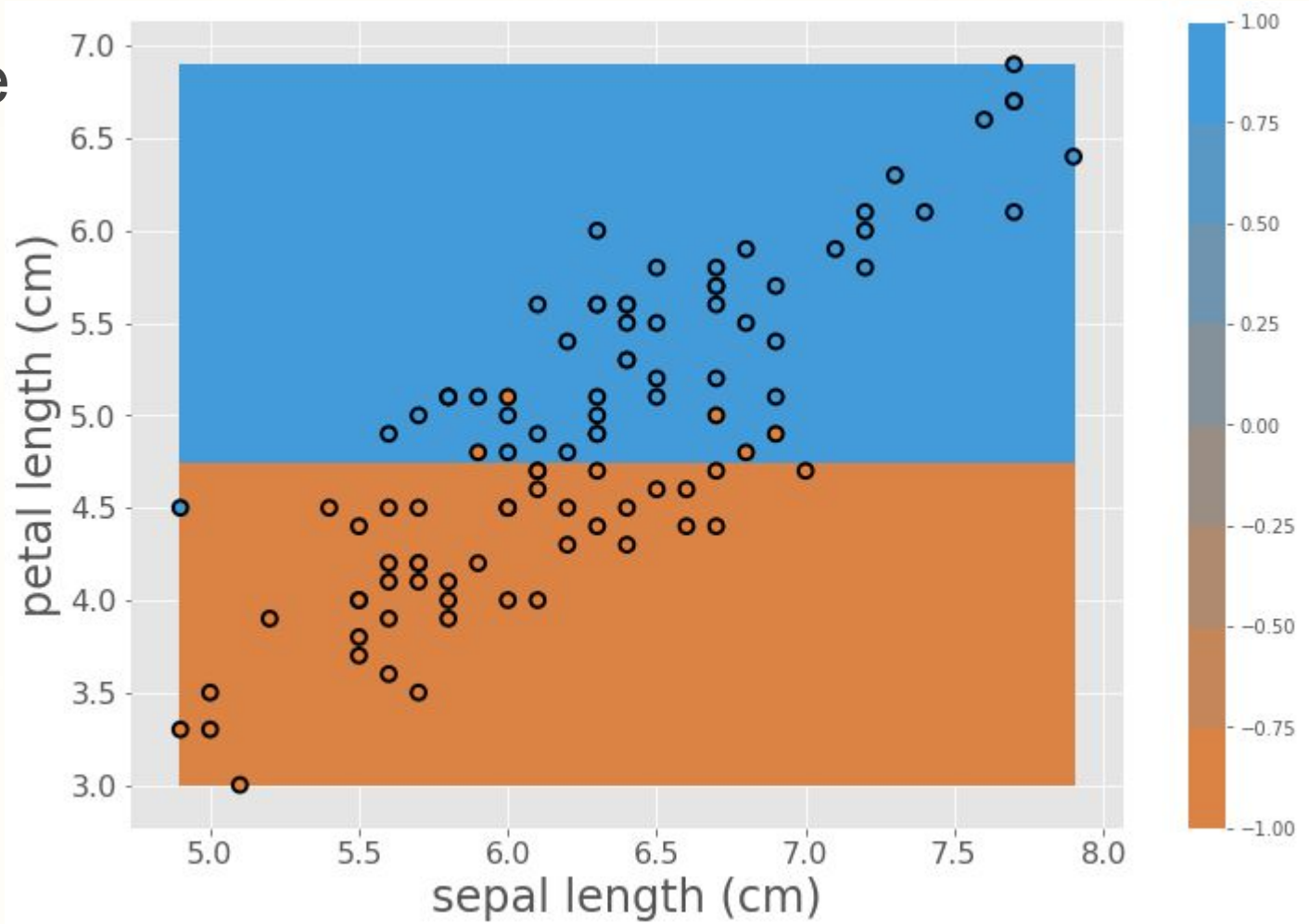
Split data into two groups



Decision tree

Split data into two groups

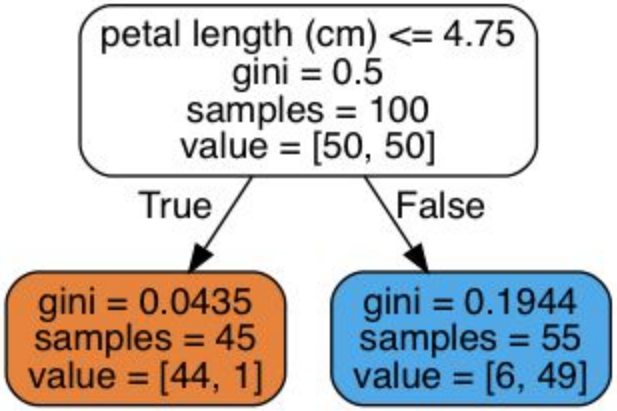
Classify each group using the majority class

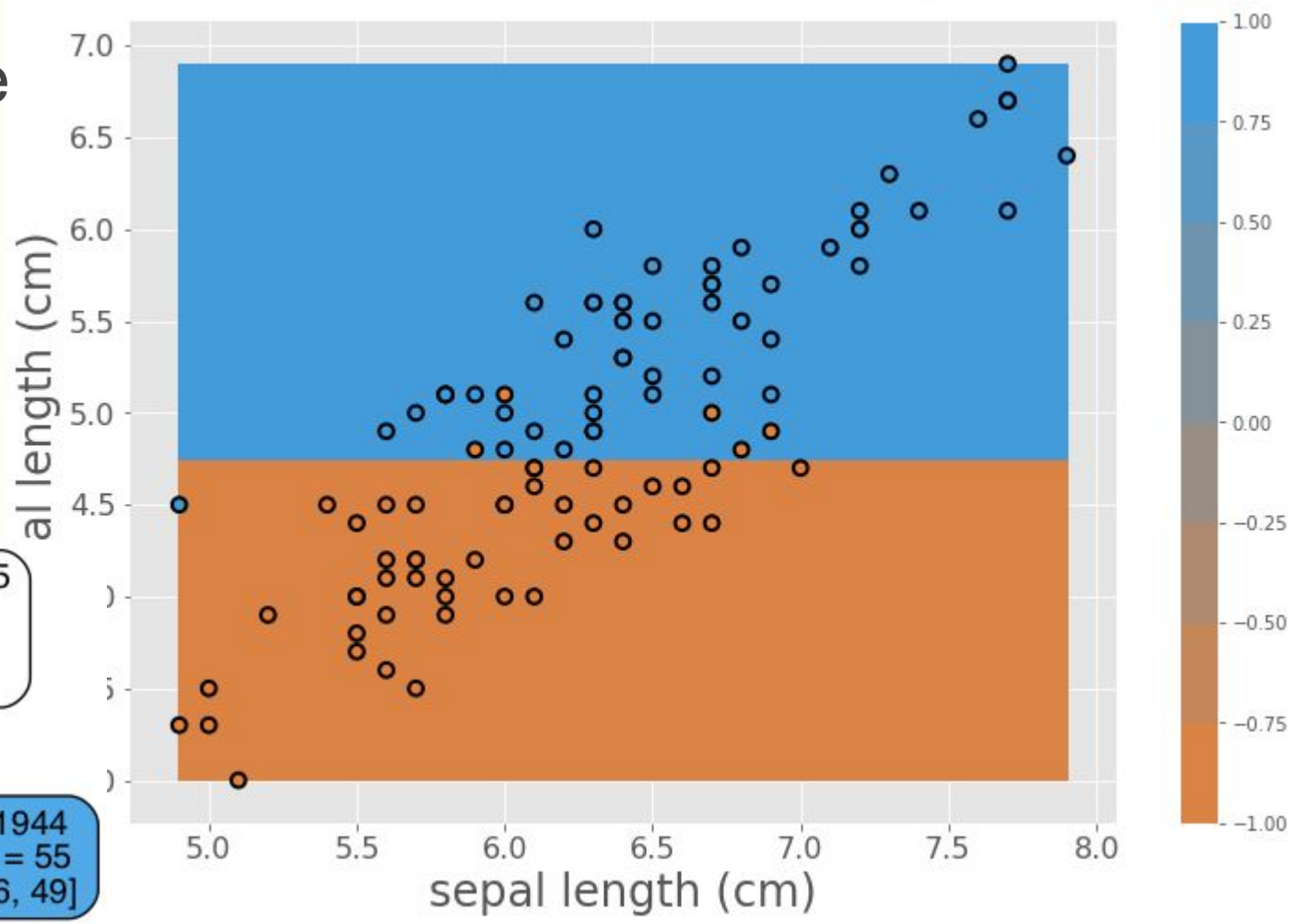


Decision tree

Split data into two groups

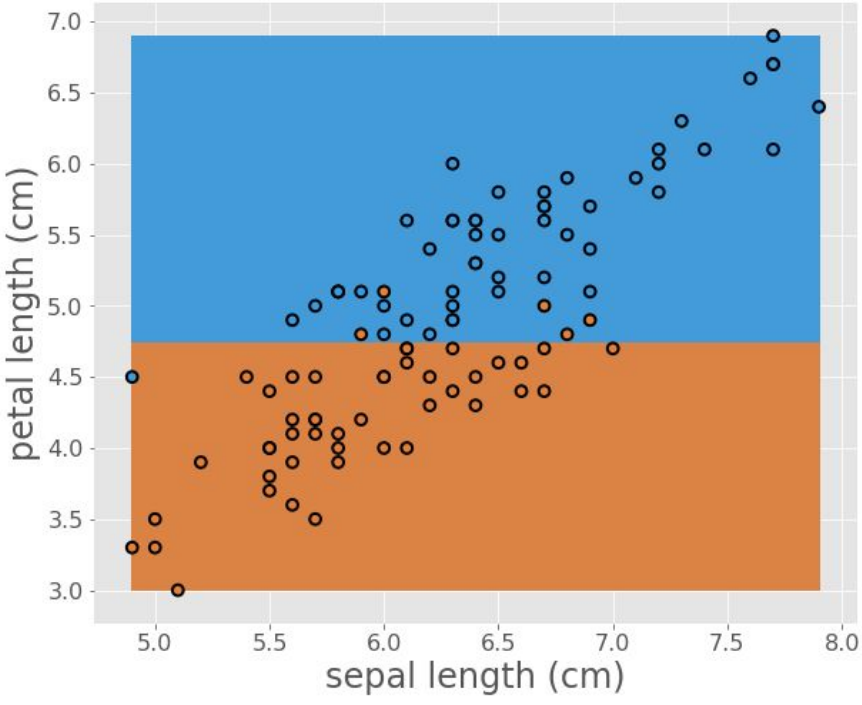
Classify each group using the majority class



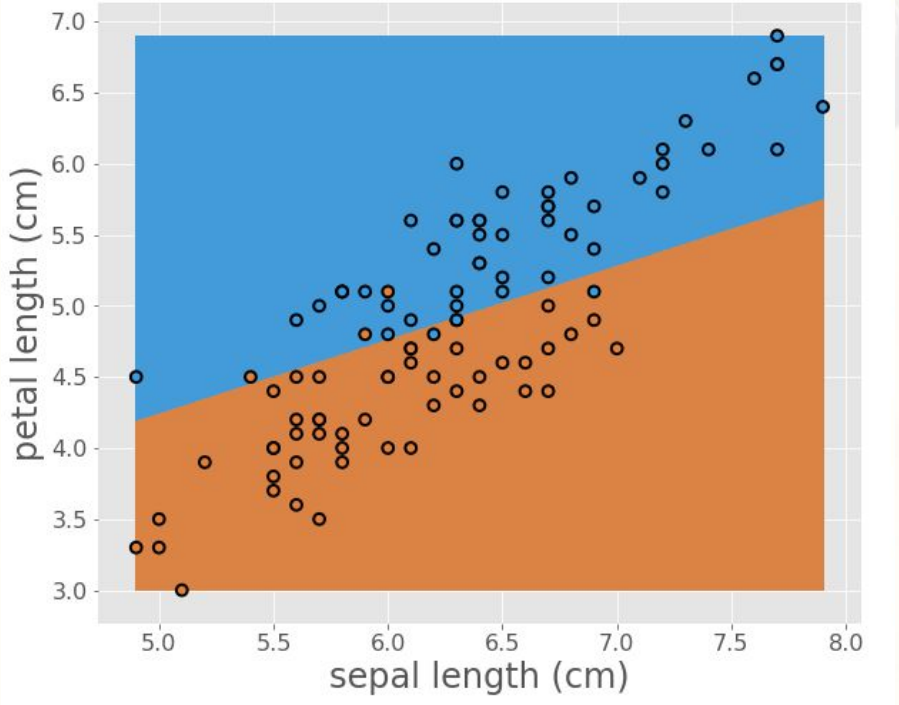


Models

Decision tree

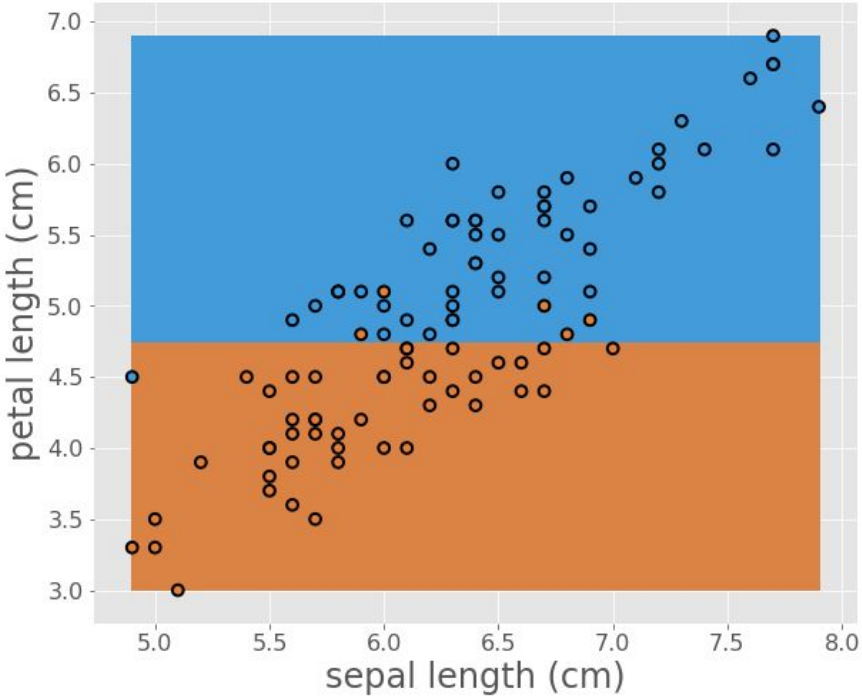


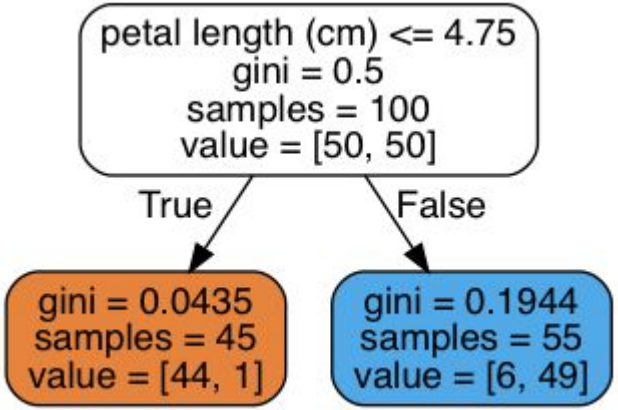
Linear regression (Logistic Regression is similar)



Models

Decision tree



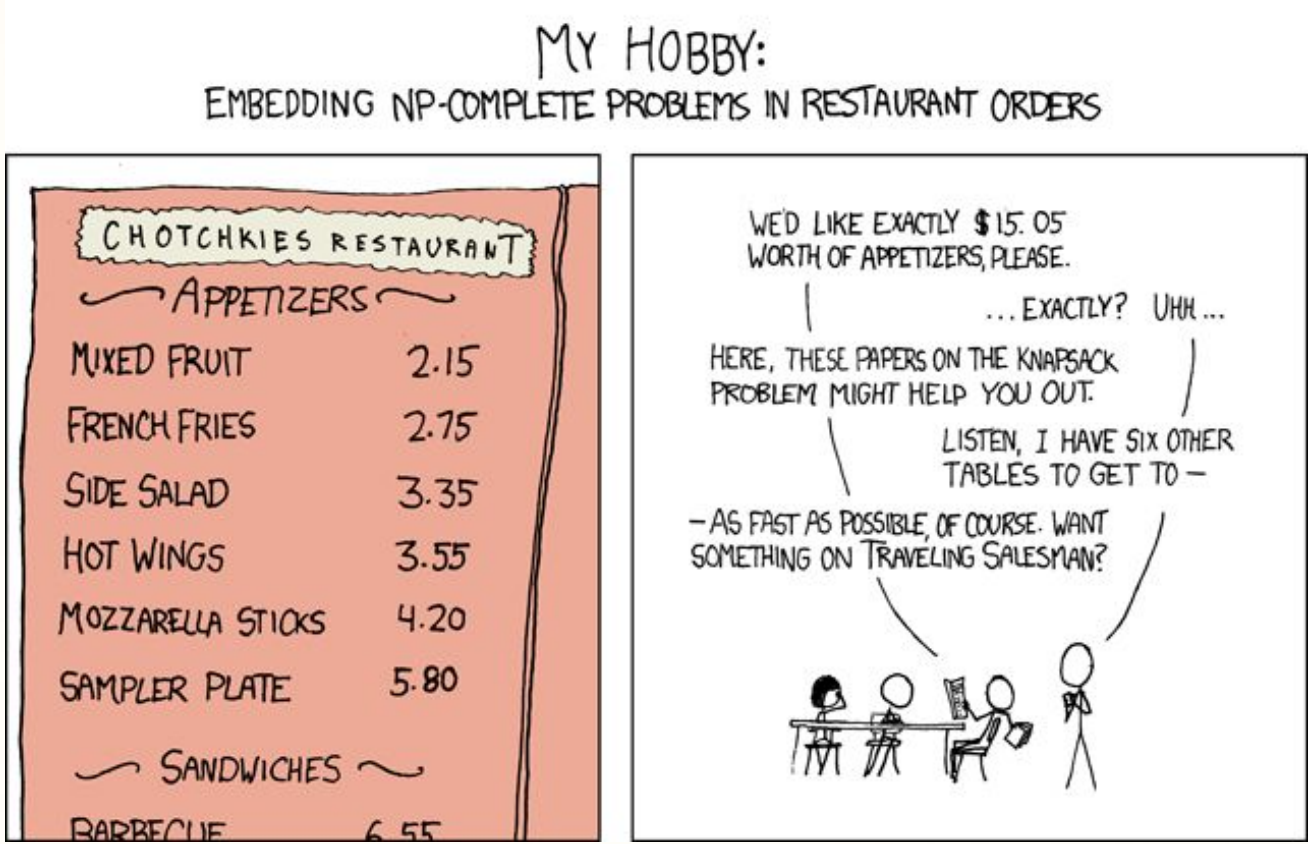


How do we build a decision tree?

Decision trees

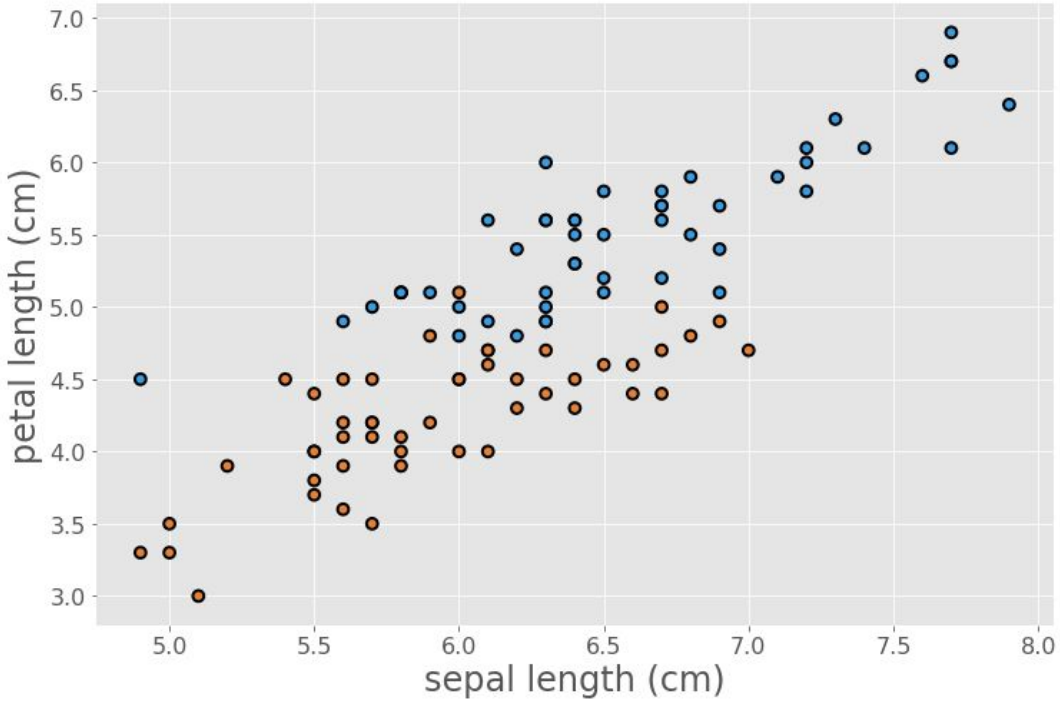
- Picking the “best” decision tree is **computationally infeasible**

- So we pick a **greedy** algorithm



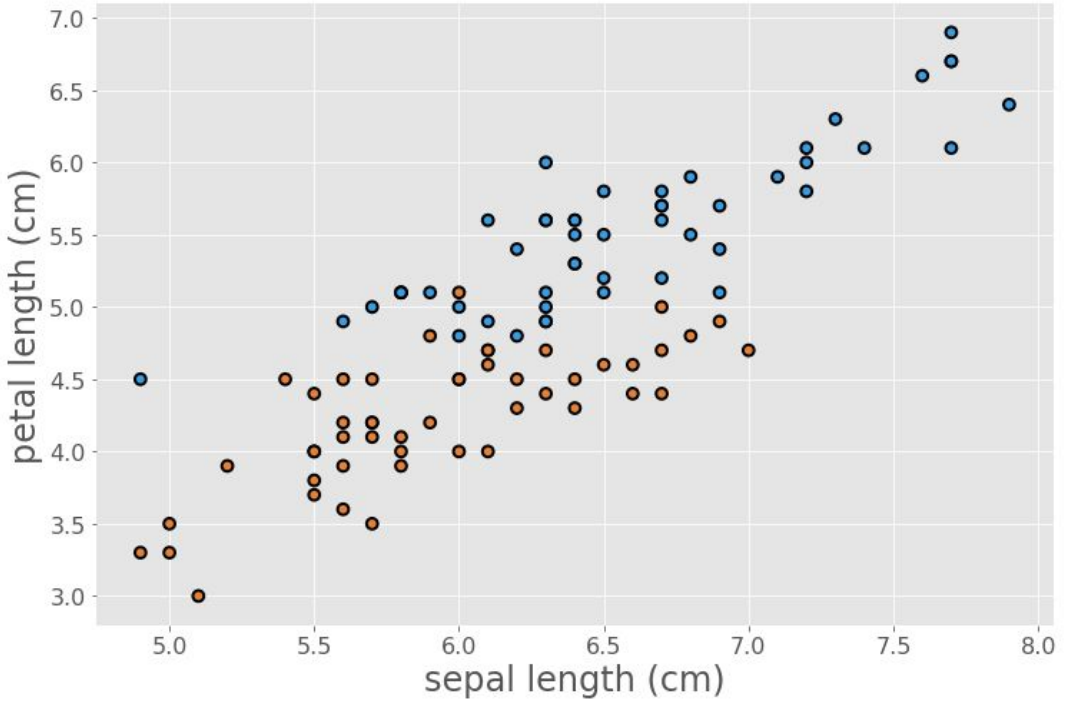
Decision tree algorithm

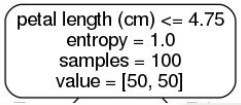
1. Pick the single feature split that best separates the data



Decision tree algorithm

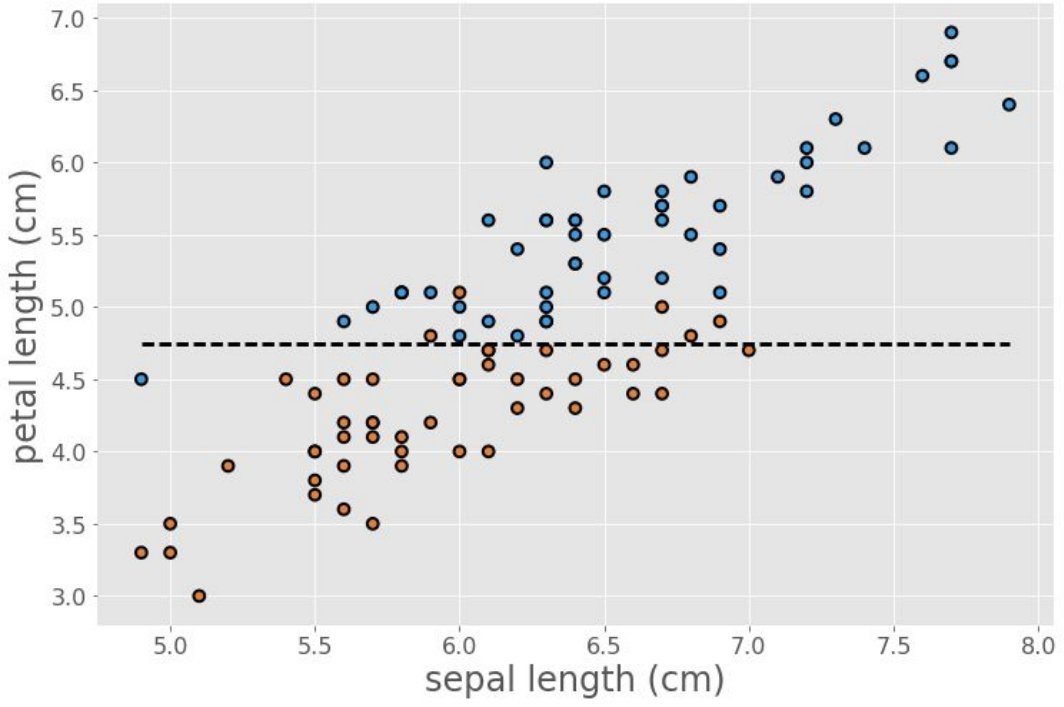
1. Pick the single feature split that best separates the data
    a. **<u>Look across all possible thresholds across all possible features</u>**

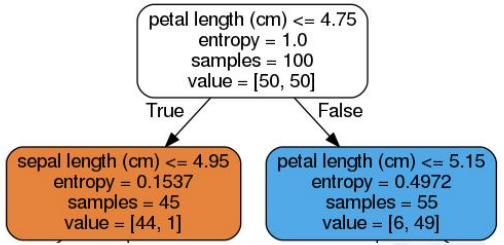




Decision tree algorithm

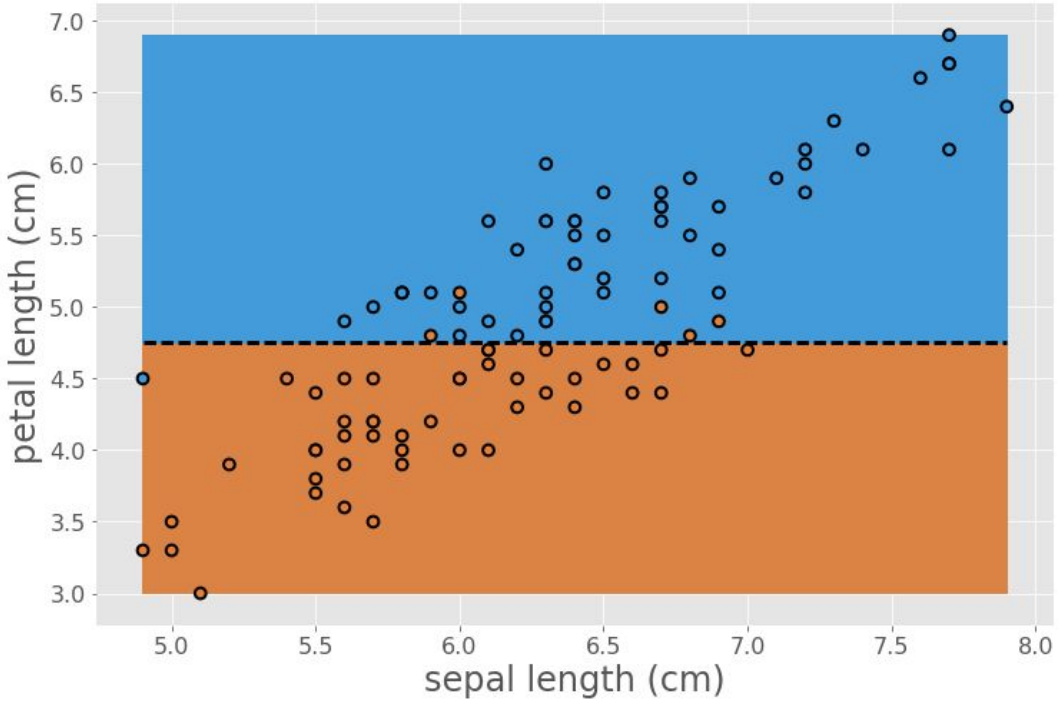
1. Pick the single feature split that best separates the data
2. Split the data on that feature

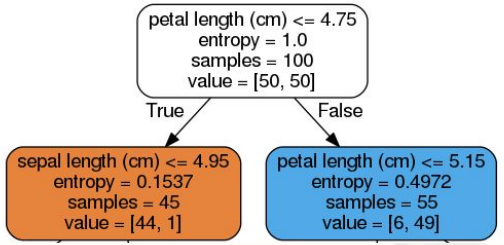




Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

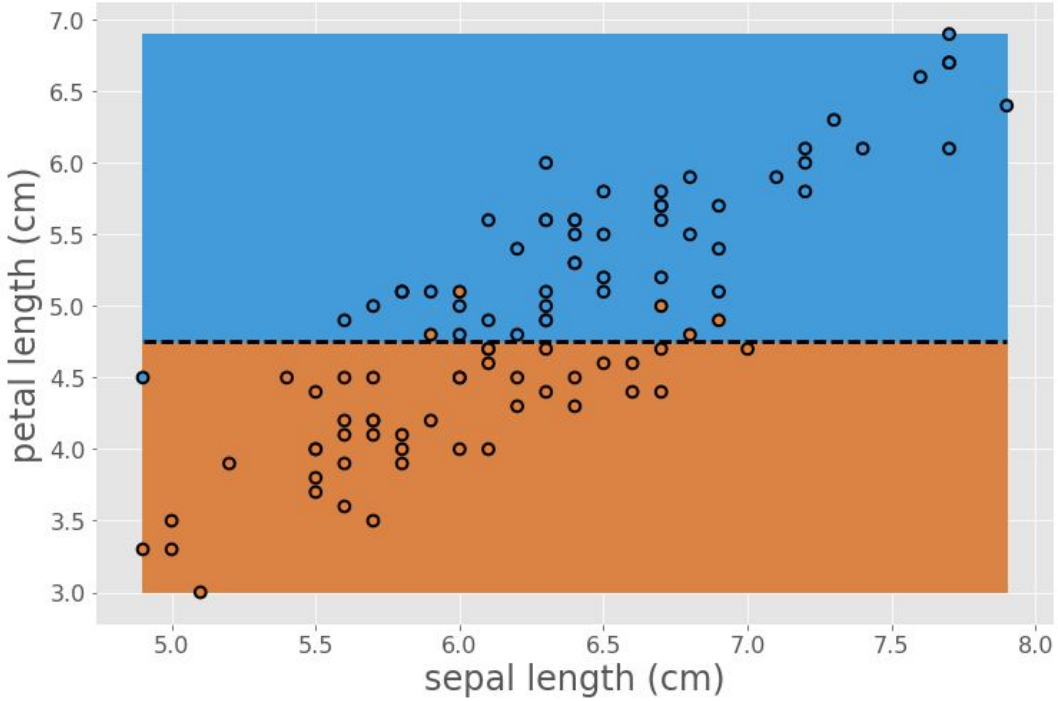


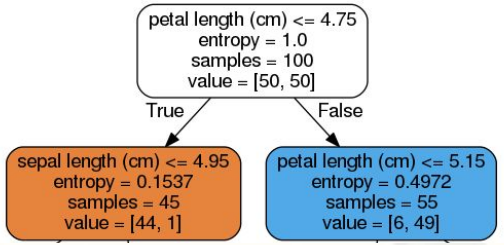


Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

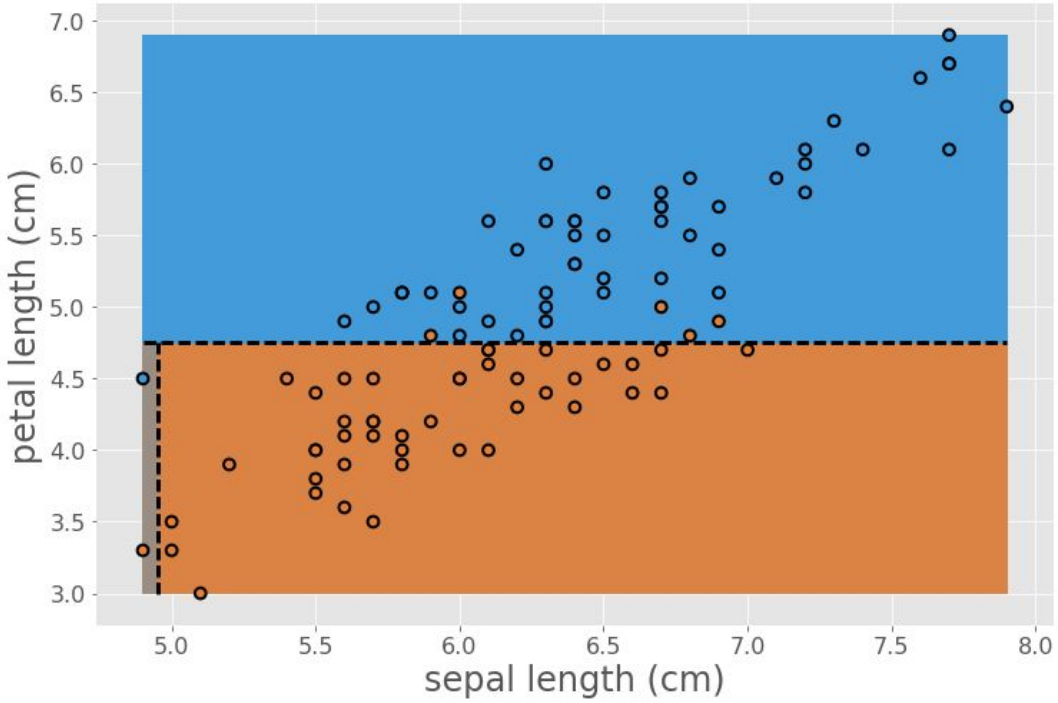


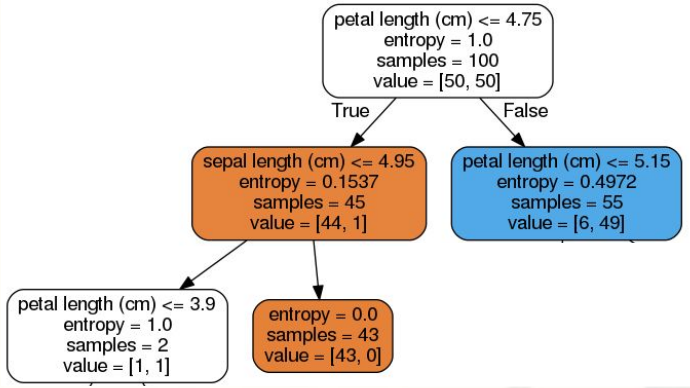


Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

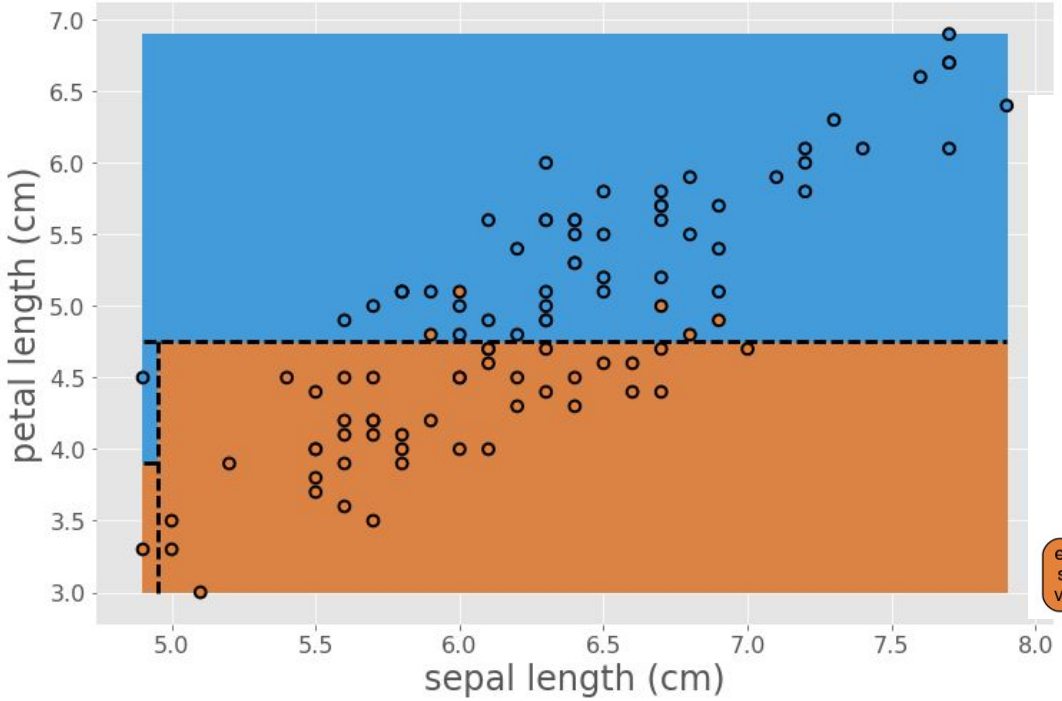


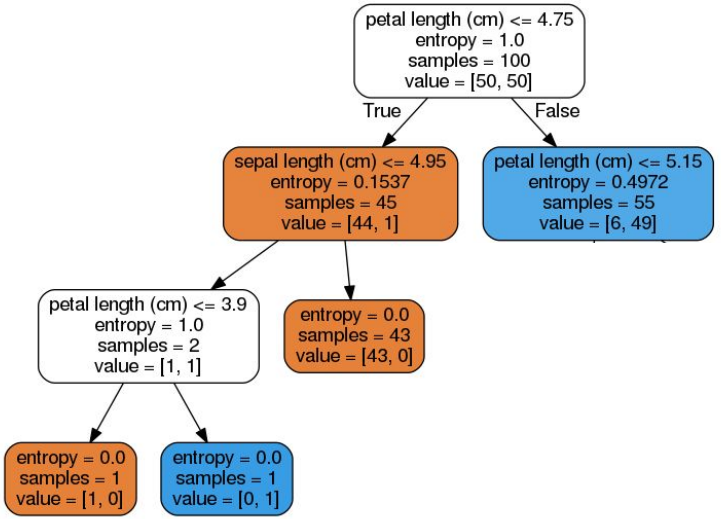


Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

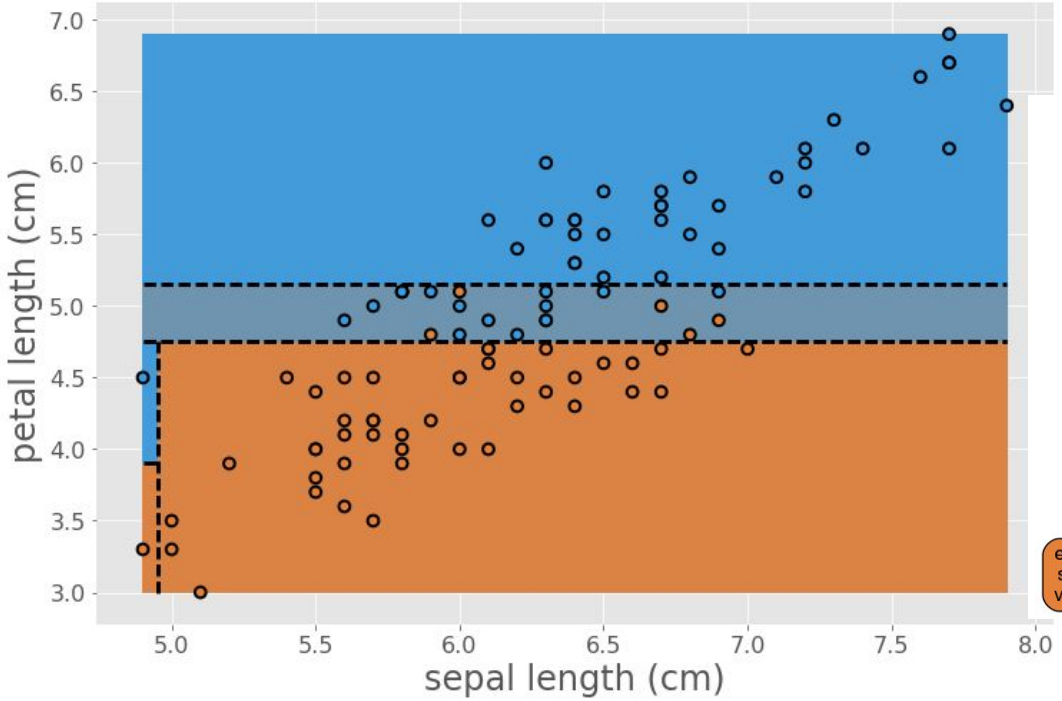


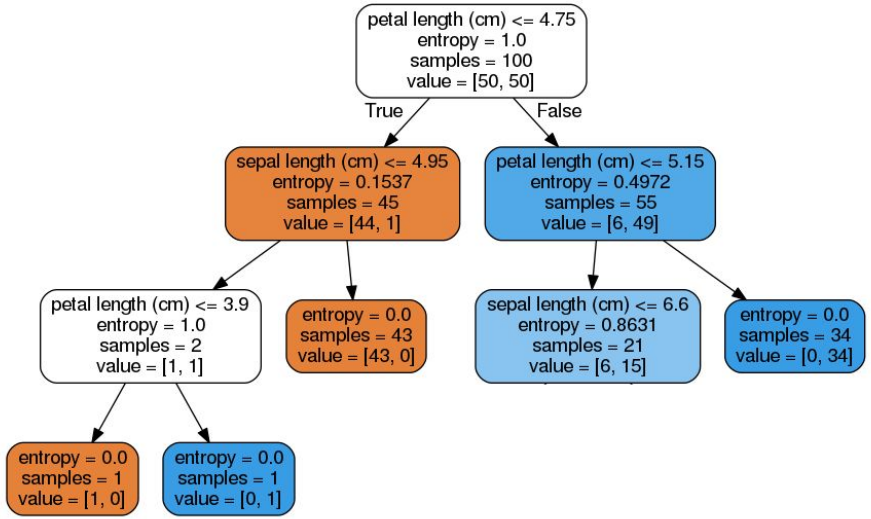


Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

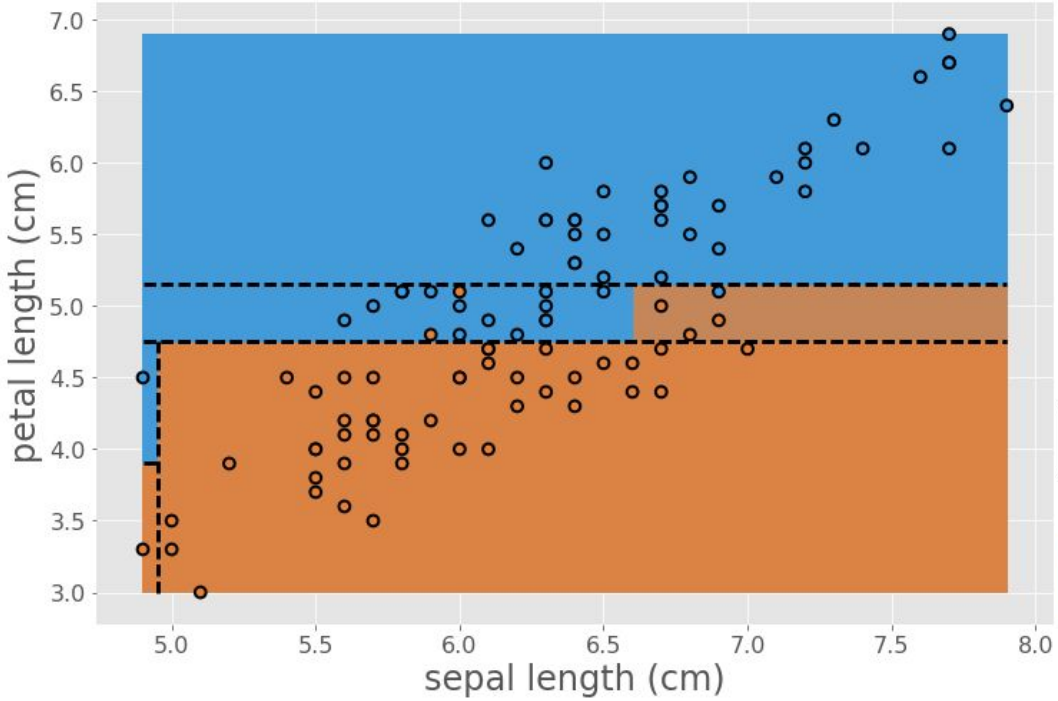


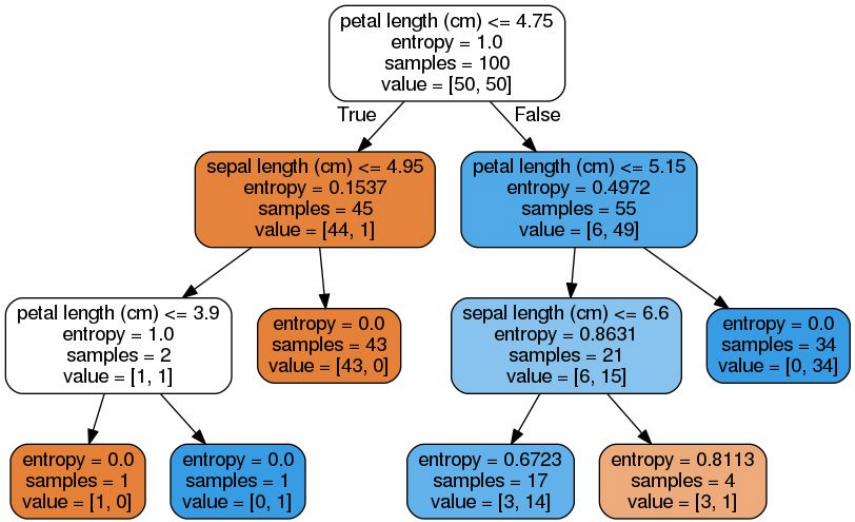


Decision tree algorithm

1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

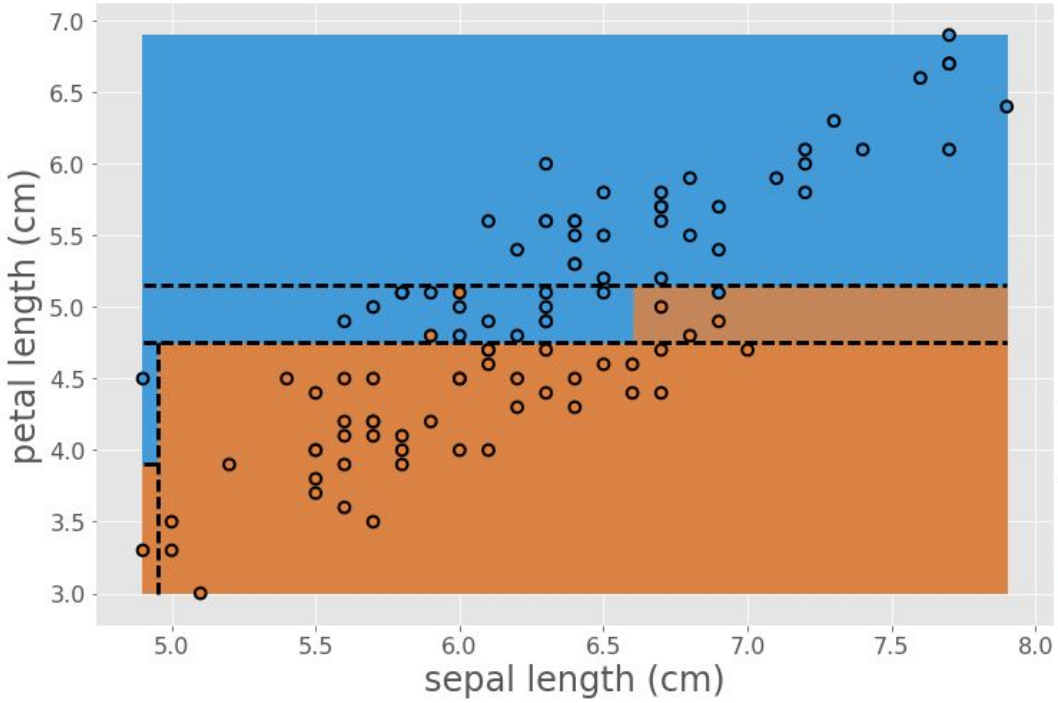




Decision tree algorithm

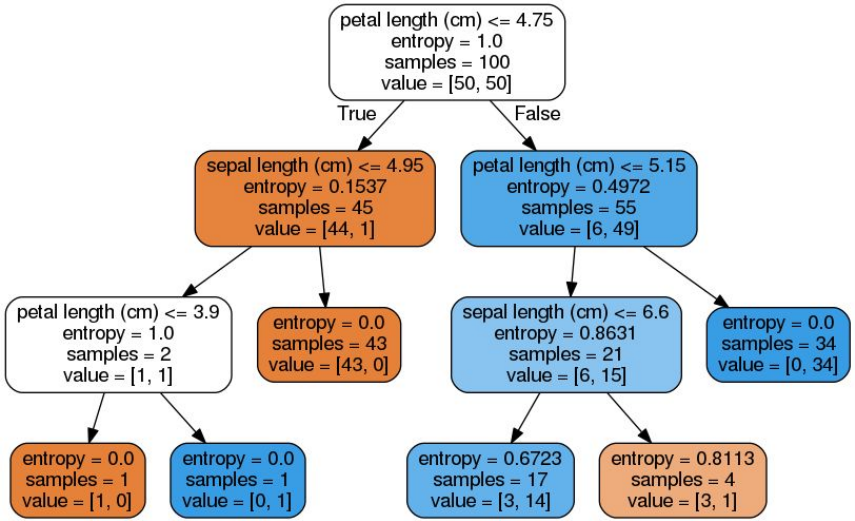
1. Pick the single feature split that best separates the data
2. Split the data on that feature

Repeat in all subgroups

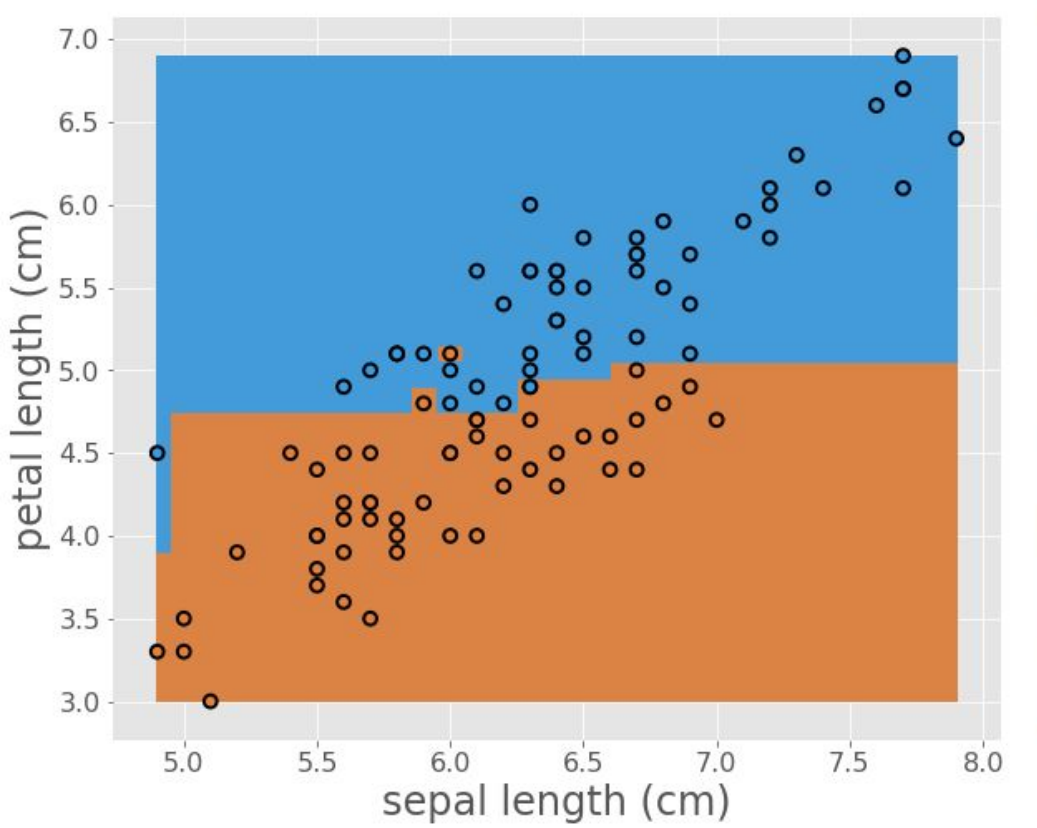


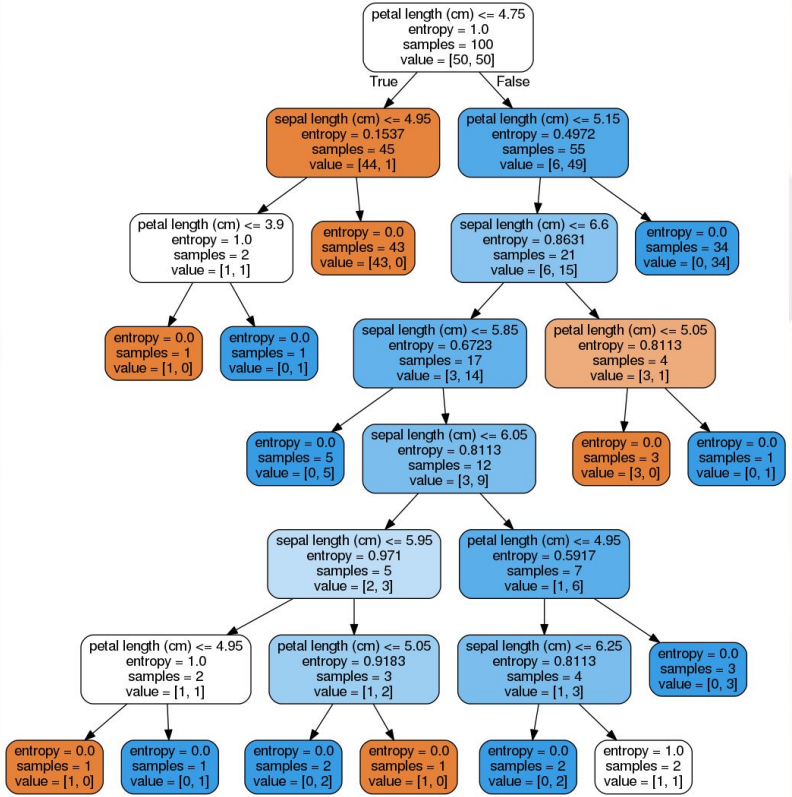
STOP

When we reach *max_depth*
When all leaves are pure



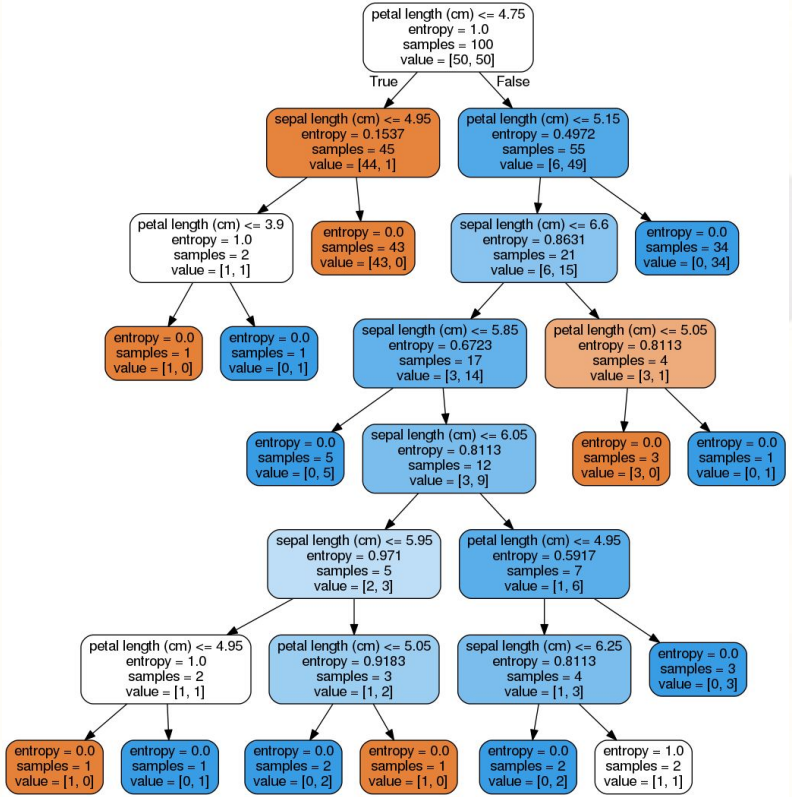
Decision tree



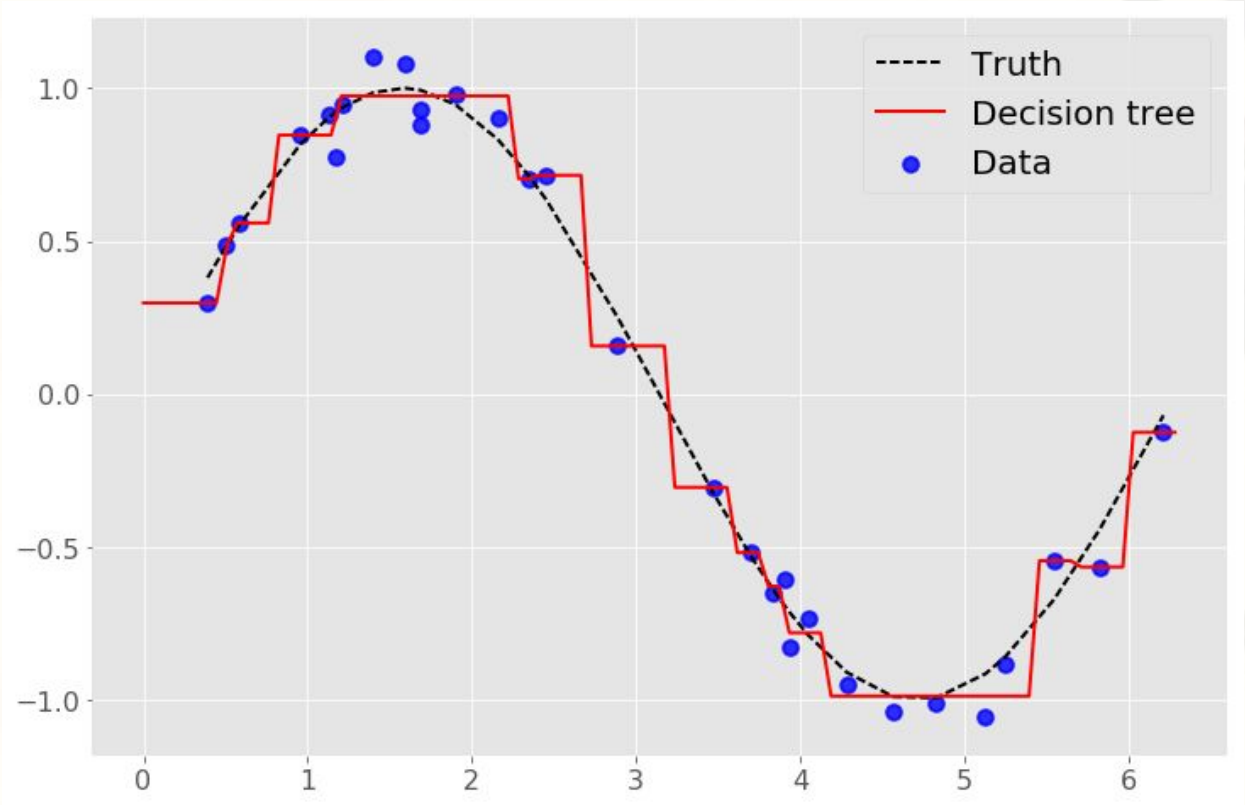


Bells and whistles

- Splitting criteria
  - Gini, Entropy
- Limit the tree size
  - Specify tree depth, prune the tree
- Number of samples
  - Minimum in a leaf
  - Minimum to split a node
- Software package
  - CART (usually you are using this)
  - ID3
  - C4.5
  - C5.0 ($$$)



Decision tree - regression



Application example

Original Contribution

February 2, 2005

Risk Stratification for In-Hospital Mortality in Acutely Decompensated Heart Failure

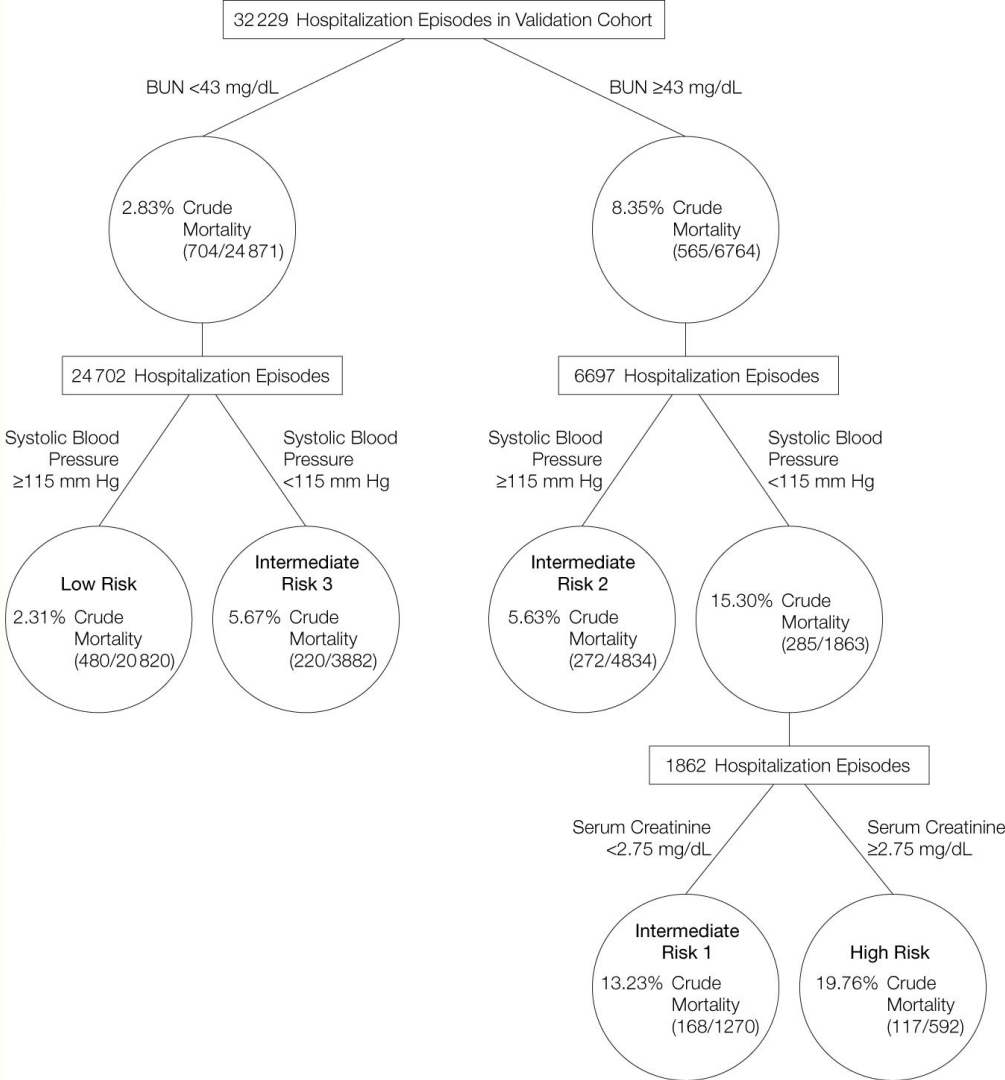
Classification and Regression Tree Analysis

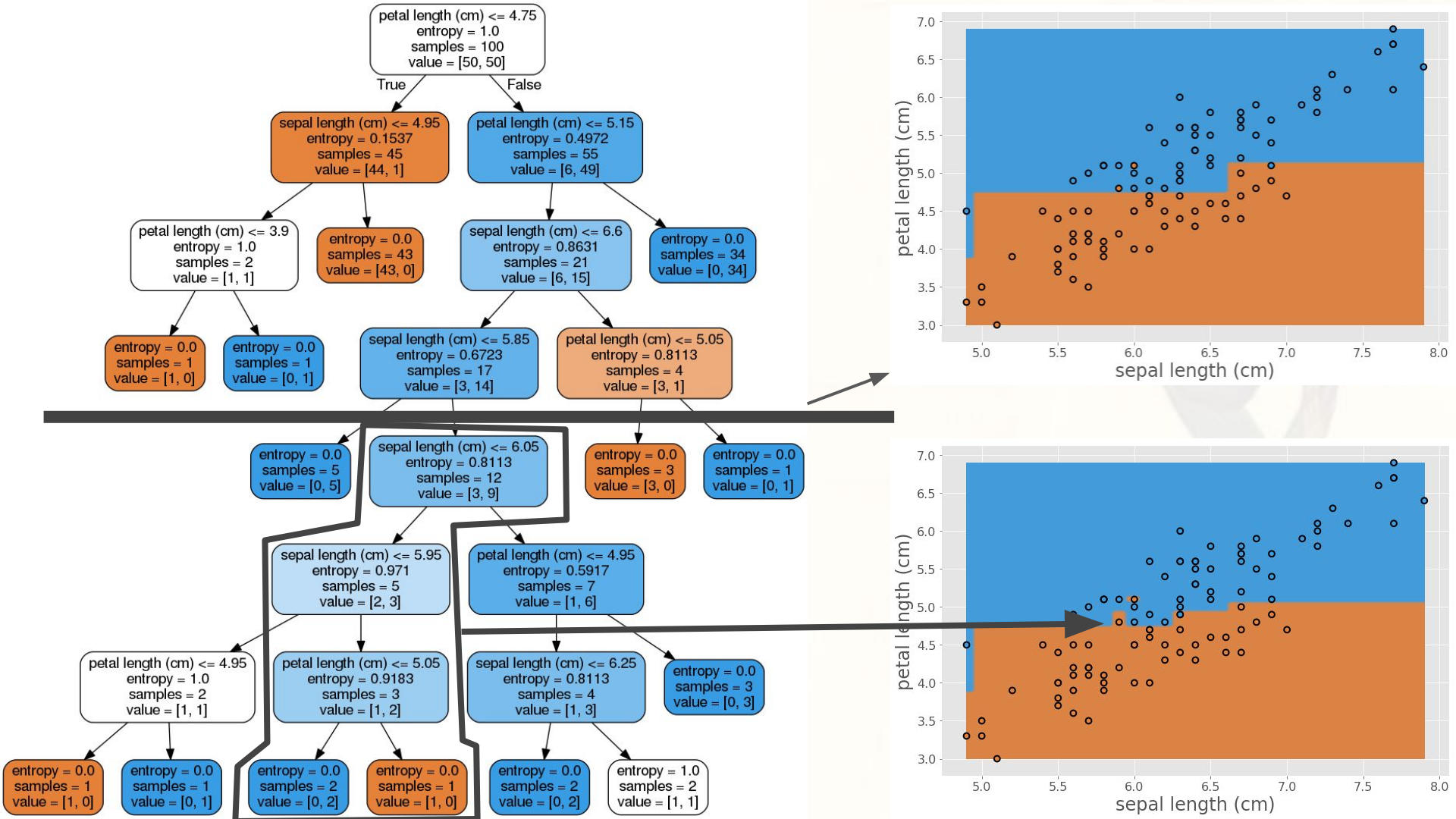
Gregg C. Fonarow, MD; Kirkwood F. Adams, MD; William T. Abraham, MD; et al

» Author Affiliations | Article Information

JAMA. 2005;293(5):572-580. doi:10.1001/jama.293.5.572

- Authors wanted rules
- Handle skewed data
- Ignore missing data when splitting



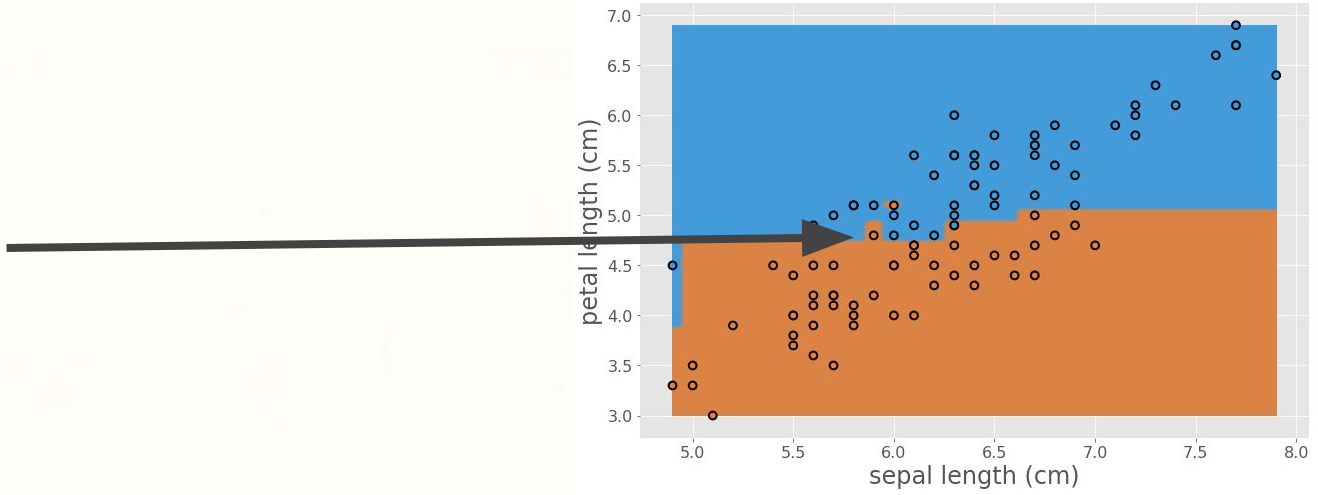

petal length (cm) <= 4.75
entropy = 1.0
samples = 100
value = [50, 50]
True
False
sepal length (cm) <= 4.95
entropy = 0.1537
samples = 45
value = [44, 1]
petal length (cm) <= 5.15
entropy = 0.4972
samples = 55
value = [6, 49]
petal length (cm) <= 3.9
entropy = 1.0
samples = 2
value = [1, 1]
entropy = 0.0
samples = 43
value = [43, 0]
sepal length (cm) <= 6.6
entropy = 0.8631
samples = 21
value = [6, 15]
entropy = 0.0
samples = 34
value = [0, 34]
entropy = 0.0
samples = 1
value = [1, 0]
entropy = 0.0
samples = 1
value = [0, 1]
sepal length (cm) <= 5.85
entropy = 0.6723
samples = 17
value = [3, 14]
petal length (cm) <= 5.05
entropy = 0.8113
samples = 4
value = [3, 1]
entropy = 0.0
samples = 5
value = [0, 5]
sepal length (cm) <= 6.05
entropy = 0.8113
samples = 12
value = [3, 9]
entropy = 0.0
samples = 3
value = [3, 0]
entropy = 0.0
samples = 1
value = [0, 1]
sepal length (cm) <= 5.95
entropy = 0.971
samples = 5
value = [2, 3]
petal length (cm) <= 4.95
entropy = 0.5917
samples = 7
value = [1, 6]
petal length (cm) <= 4.95
entropy = 1.0
samples = 2
value = [1, 1]
petal length (cm) <= 5.05
entropy = 0.9183
samples = 3
value = [1, 2]
sepal length (cm) <= 6.25
entropy = 0.8113
samples = 4
value = [1, 3]
entropy = 0.0
samples = 3
value = [0, 3]
entropy = 0.0
samples = 1
value = [1, 0]
entropy = 0.0
samples = 1
value = [0, 1]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 1
value = [1, 0]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 1.0
samples = 2
value = [1, 1]
petal length (cm)
sepal length (cm)
petal length (cm)
sepal length (cm)

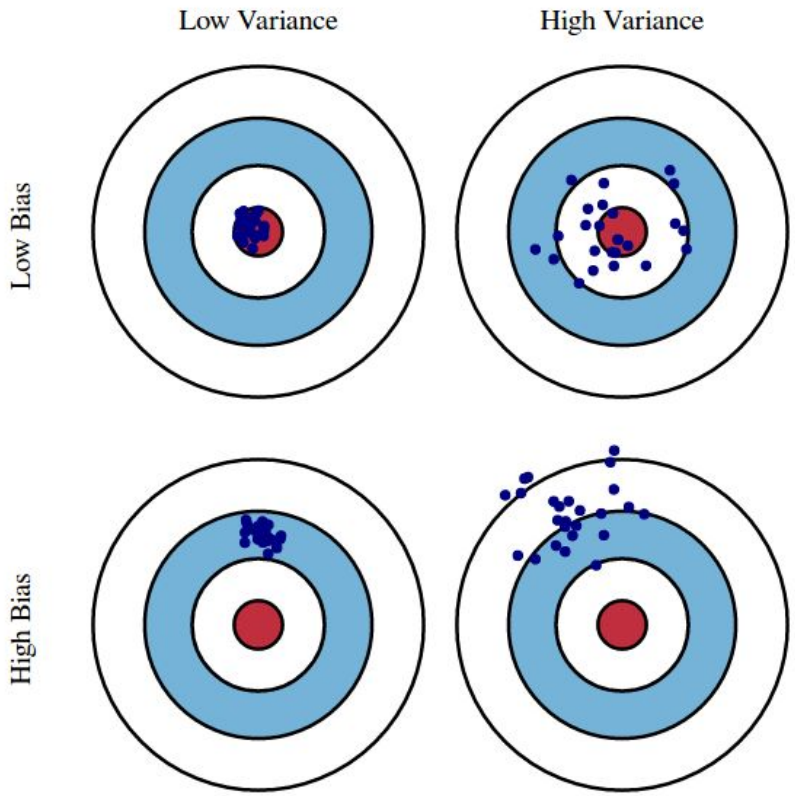
Decision trees are flexible

Flexibility allows for very nuanced boundaries

Sometimes good, sometimes bad

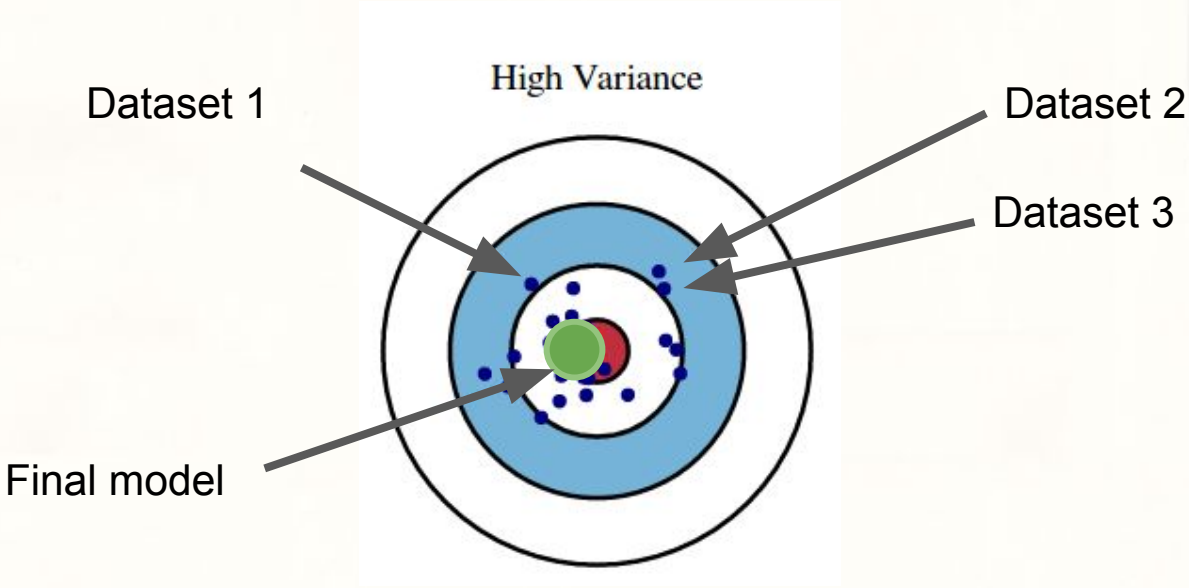
Using statistics jargon, decision trees exhibit **high variance**





$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



We need to **create new datasets** to build models

Boosting (AdaBoost)

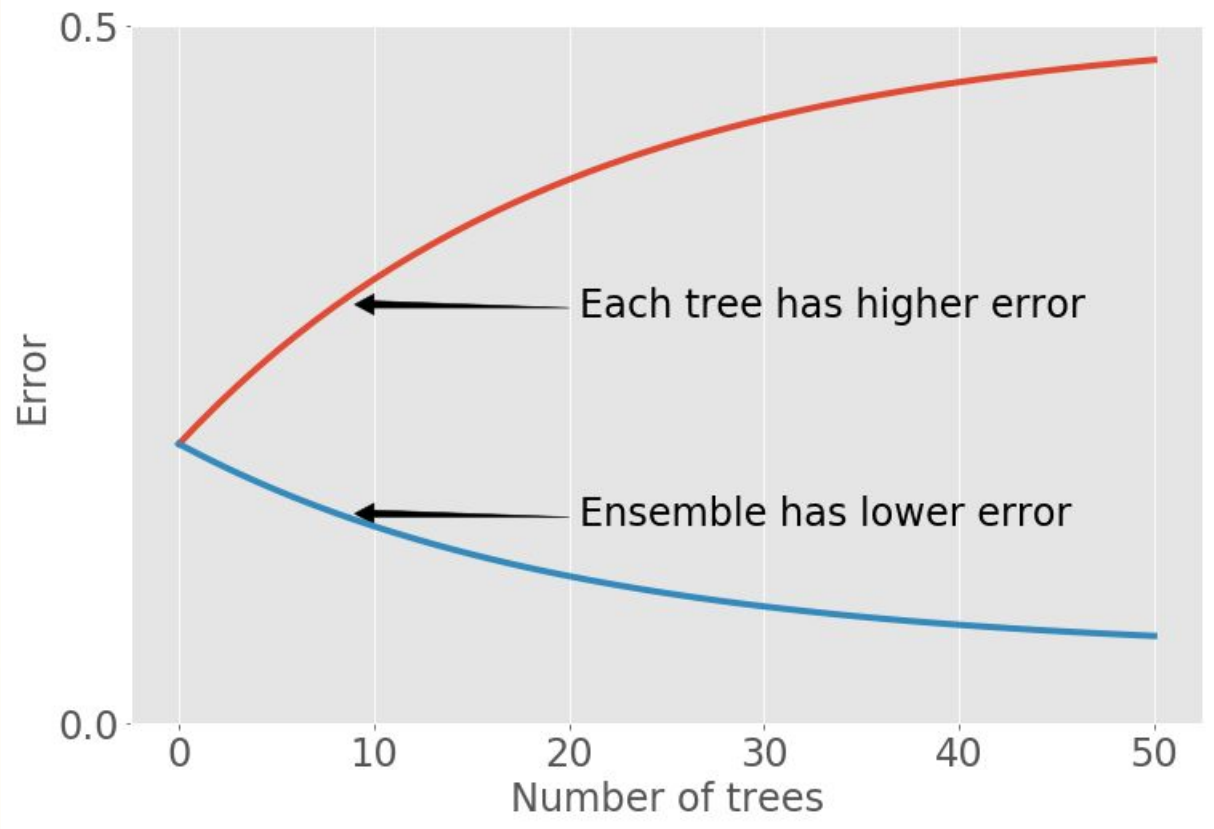
Boosting

- Proposed by Freund and Schapire
- Build decision trees **sequentially**
- Subsequent decision trees are trained on a **modified dataset**

Boosting

Each tree is shown the **most difficult examples**

Each individual tree does worse, but the overall average does better



AdaBoost algorithm

1. Build a tree using all the data
2. Find the samples where the tree misclassified the data
3. Weight these samples higher
4. Build a new tree using the re-weighted data
5. Repeat 2-4 until termination

AdaBoost algorithm

1. Build a tree using all the data
2. Find the samples where the tree misclassified the data
3. Weight these samples higher
4. Build a new tree using the re-weighted data
5. Repeat 2-4 until termination

Example using decision tree of depth 1

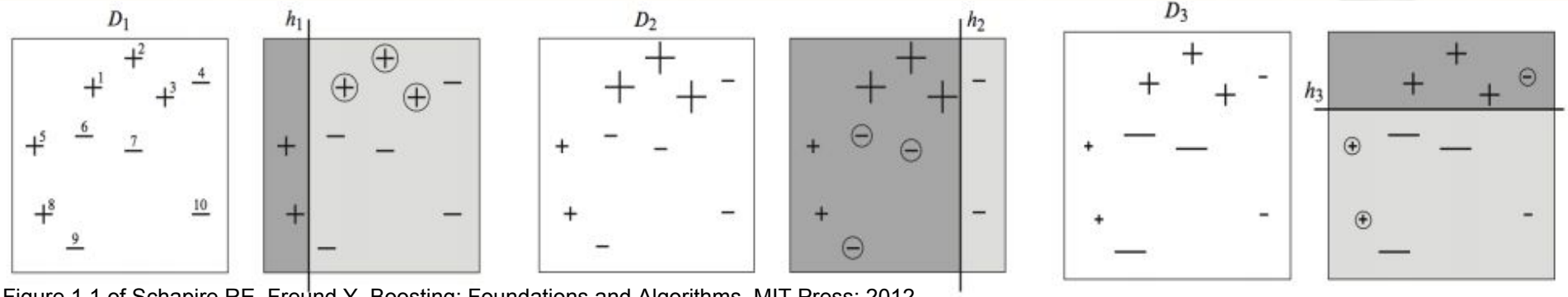


Figure 1.1 of Schapire RE, Freund Y. Boosting: Foundations and Algorithms. MIT Press; 2012

AdaBoost algorithm

- In each iteration, hard to classify samples are **weighted higher**
  - The algorithm sequentially focuses on hard cases
- For each tree, high accuracy trees are **weighted higher**
  - Otherwise, far right tree has equal weight to first tree

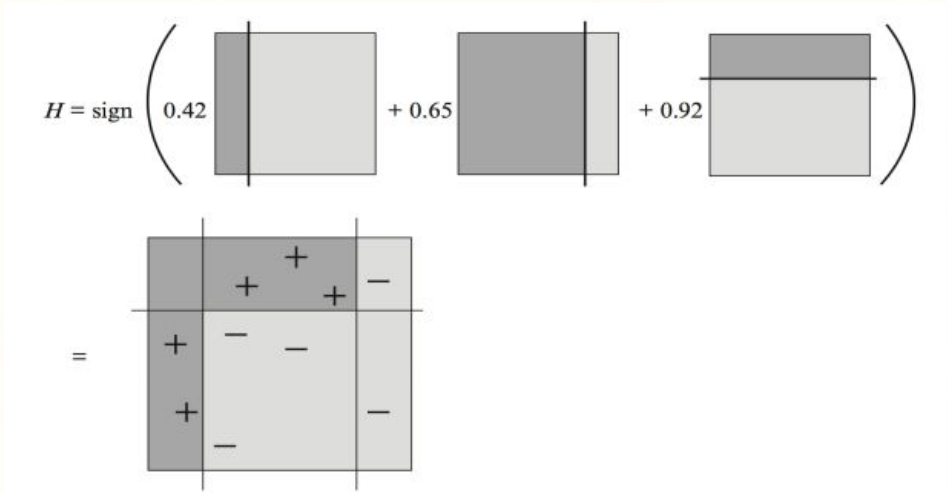
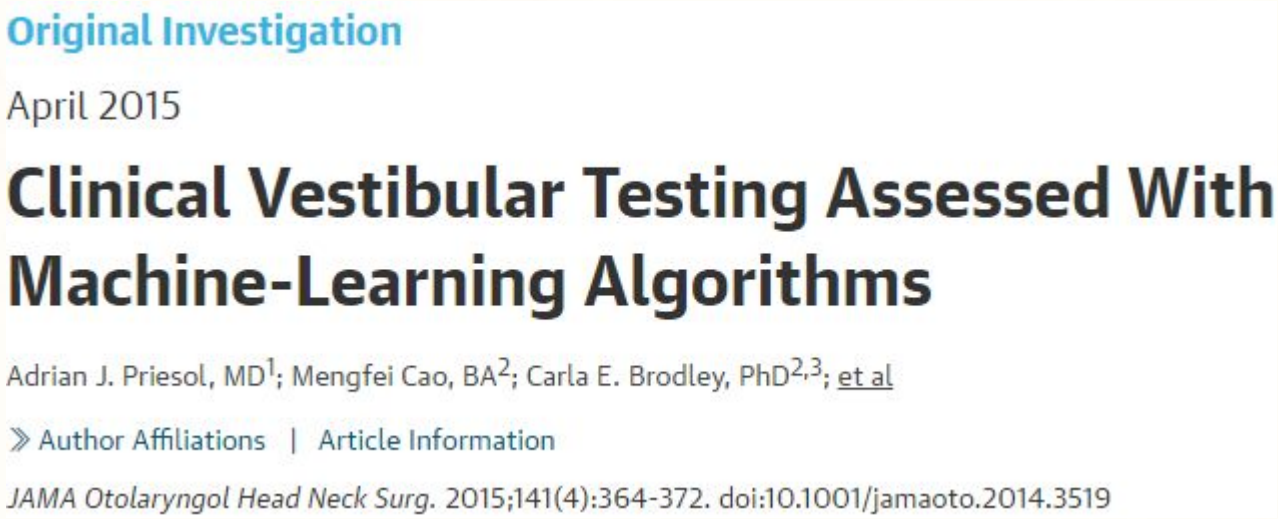


Figure 1.2 of Schapire RE, Freund Y. Boosting: Foundations and Algorithms. MIT Press; 2012

Applications of AdaBoost



Original Investigation

April 2015

Clinical Vestibular Testing Assessed With Machine-Learning Algorithms

Adrian J. Priesol, MD[1]; Mengfei Cao, BA[2]; Carla E. Brodley, PhD[2,3]; et al

» Author Affiliations | Article Information

JAMA Otolaryngol Head Neck Surg. 2015;141(4):364-372. doi:10.1001/jamaoto.2014.3519

"... applied several of the more common methods (decision trees, random forests, logistic regression, AdaBoost, and support vector machines [SVMs])"

"An SVM with a radial basis kernel … performed best"

History of Boosting

1999: Bauer and Kohavi found ...

- average 27% relative improvement of AdaBoost vs tree
- boosting algorithms, in contrast to Bagging, reduce variance AND bias

Freund & Schapire won the Gödel prize in 2003

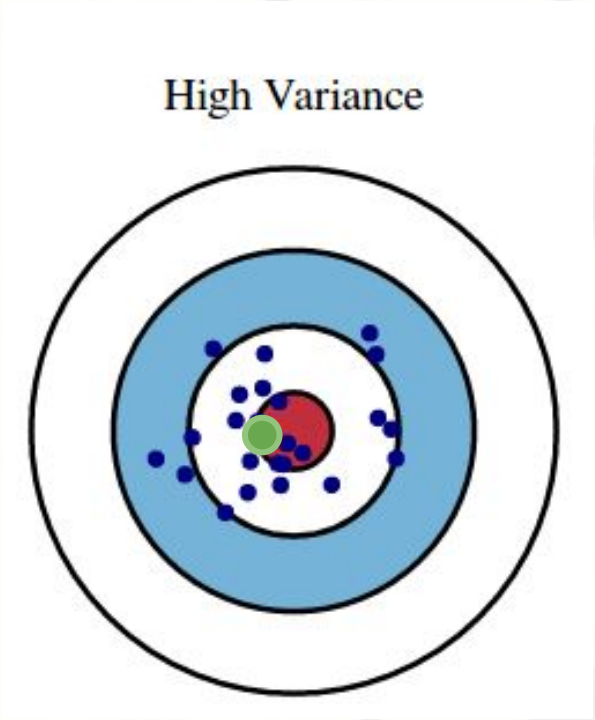
Leo Breiman

Bauer E, Kohavi R. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. Journal of Machine Learning. 1999;36:105–139.

Bagging

Building an ensemble

- Combine **weak learners** into a **single strong learner**
- Requires weak learner to be **high variance**



Bagging

- **Boosting**: create a “new” dataset, build a new tree
- Each new dataset is **carefully chosen**

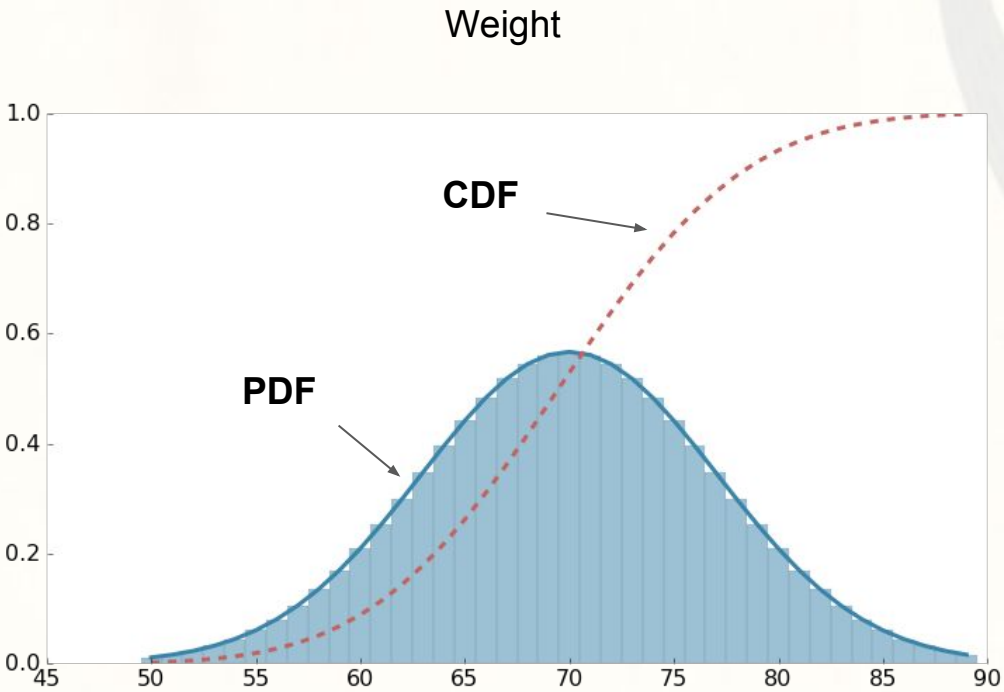
- **Bagging**: create a “new” dataset, build a new tree
- Each new dataset is **randomly generated** (bootstrap)

Bootstrapping

- We can generate new datasets **<u>from our current dataset</u>**
- Sounds impossible - “pulling yourself up with your bootstraps”

Baron von Munchhausen

Bootstrapping



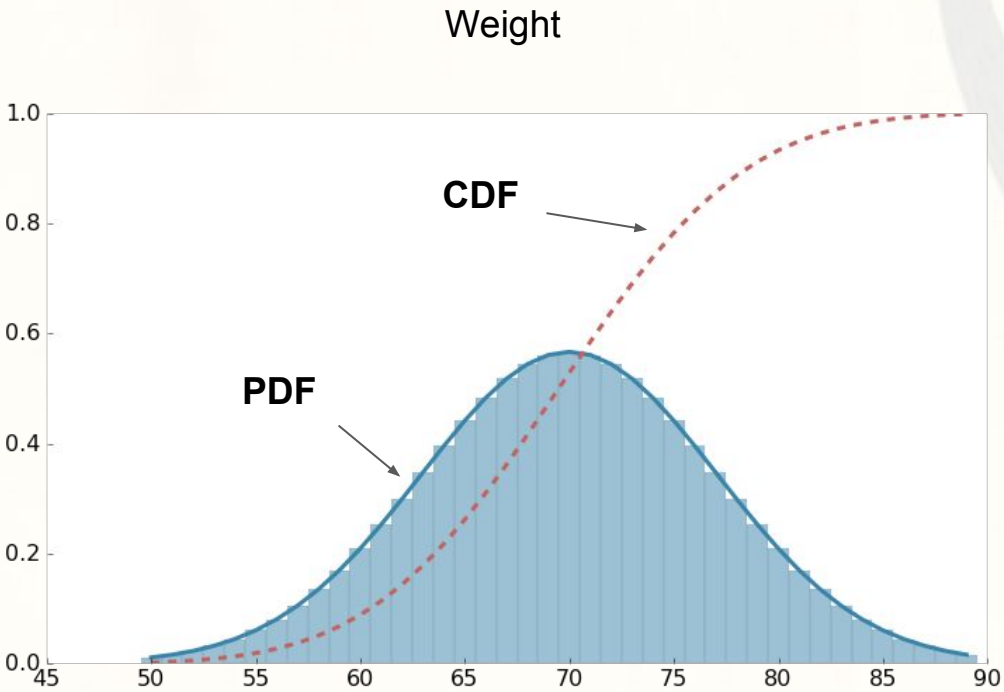
Bootstrapping

When we collect data:

We are sampling from the "population" PDF

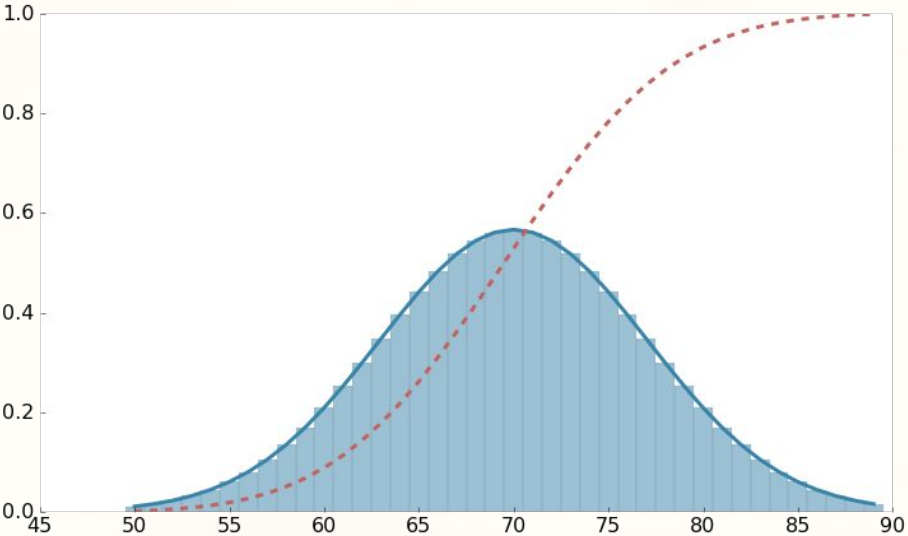
So if we want new trees..

We want repeated samples from the population PDF

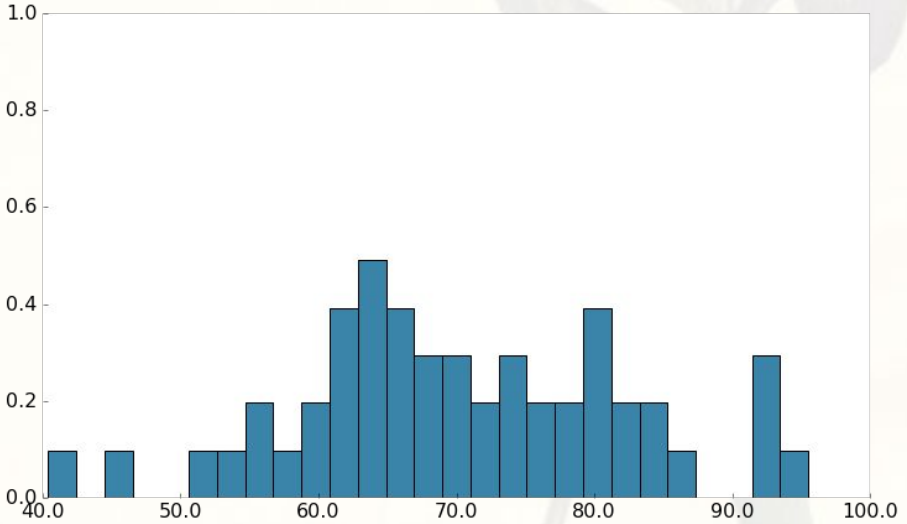


Bootstrapping

What we want

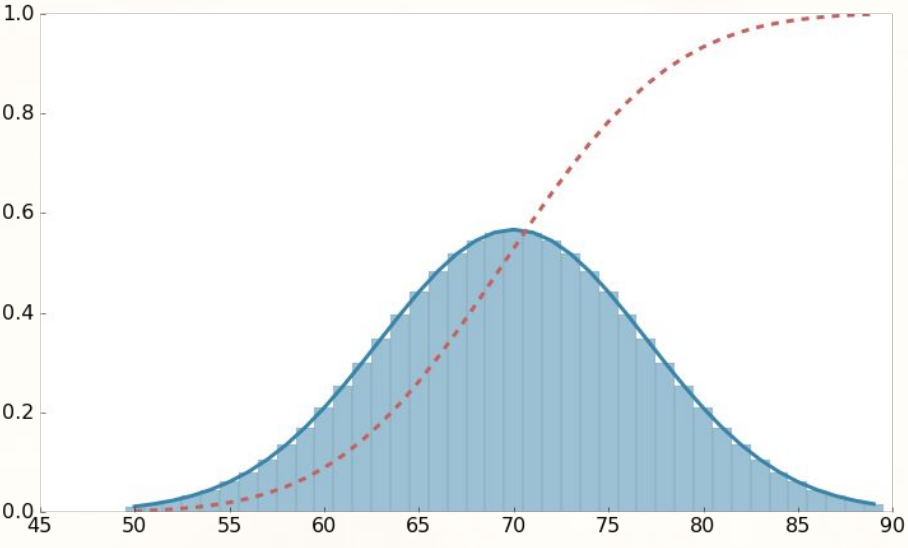


What we have (N=50)

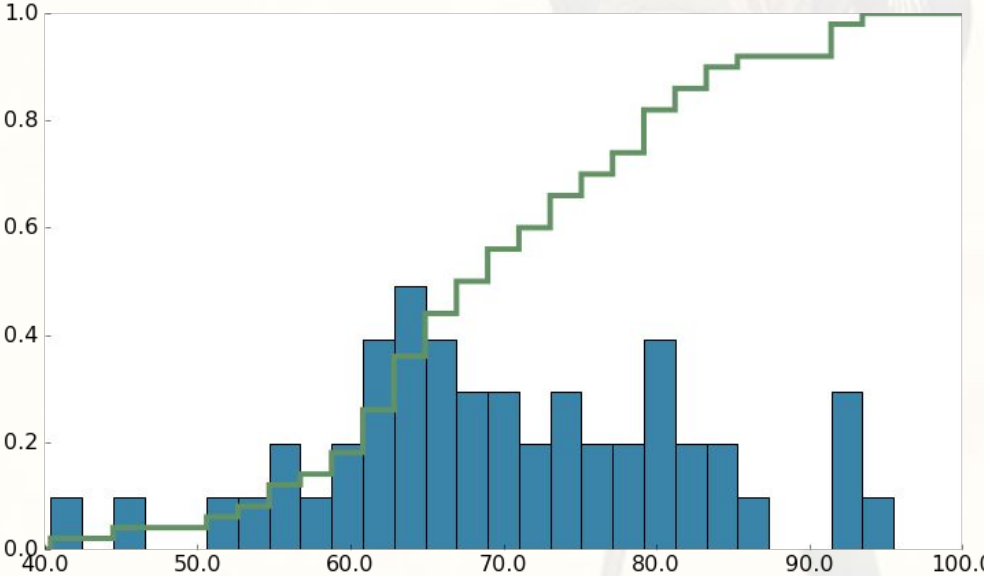


Bootstrapping

What we want

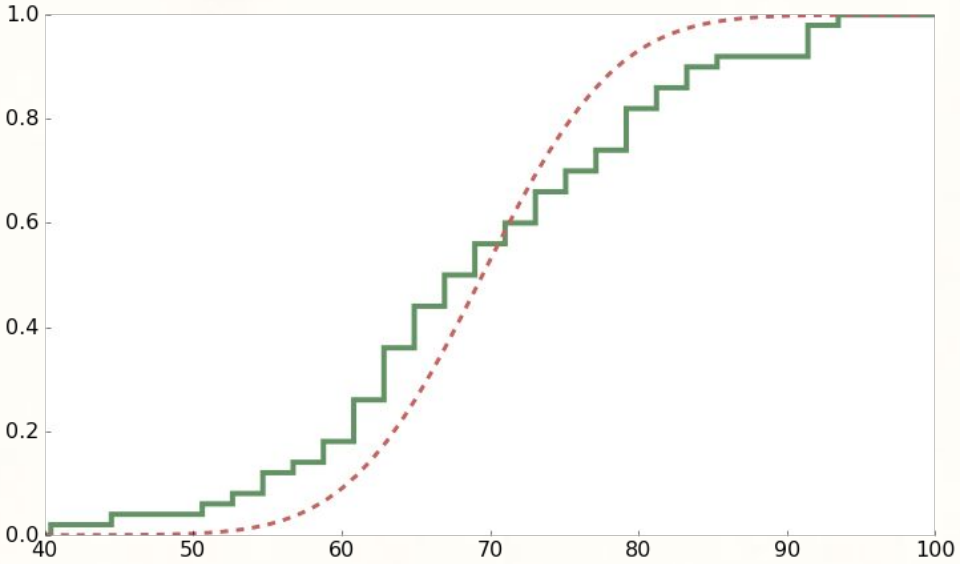


What we have (N=50) .. + the CDF

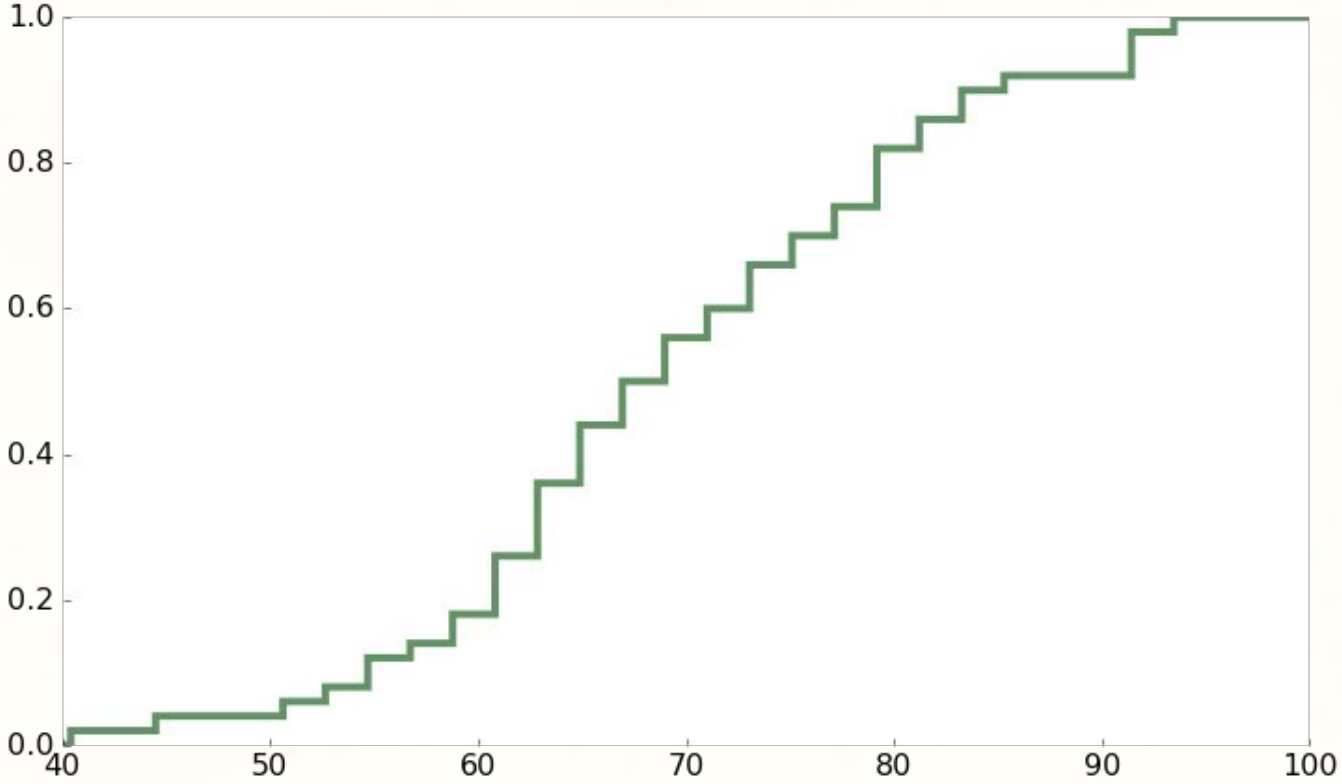


Bootstrapping

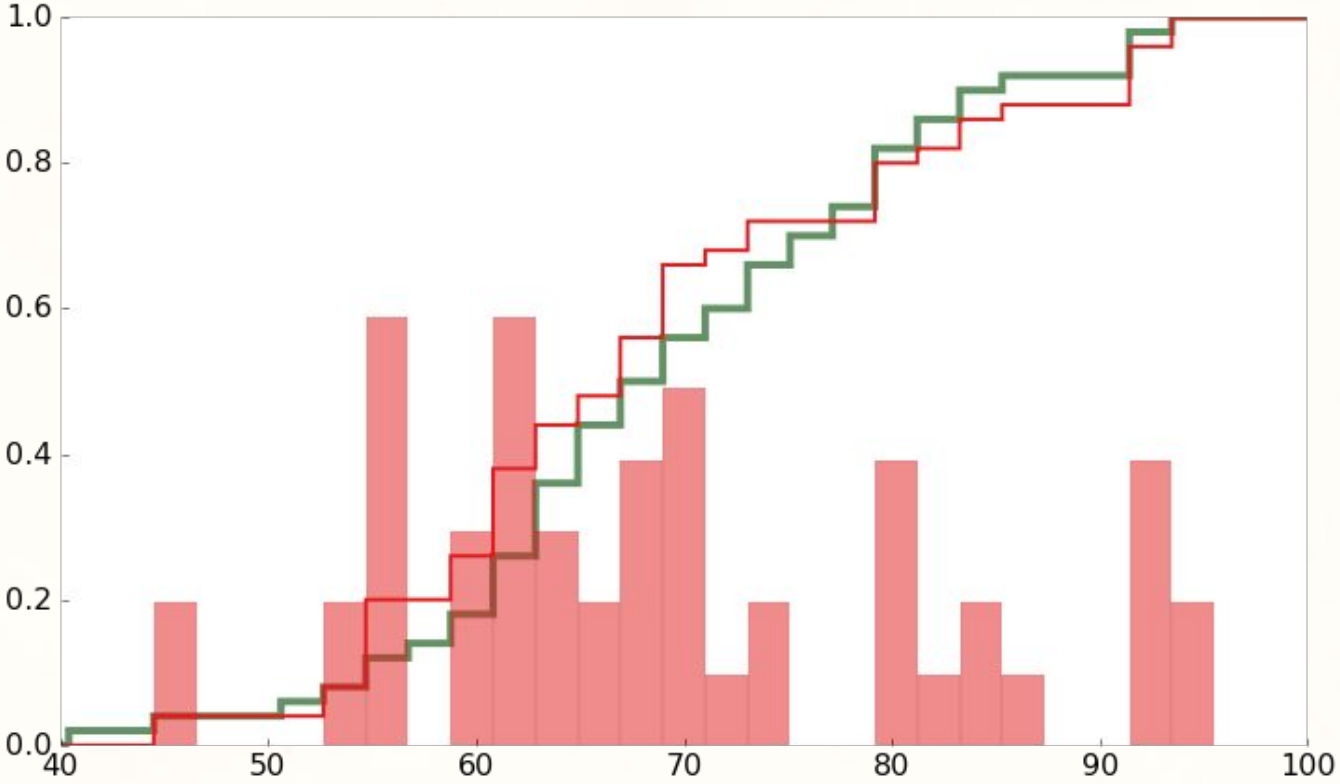
The empirical CDF (purple) is a good approximation to the true CDF (red)



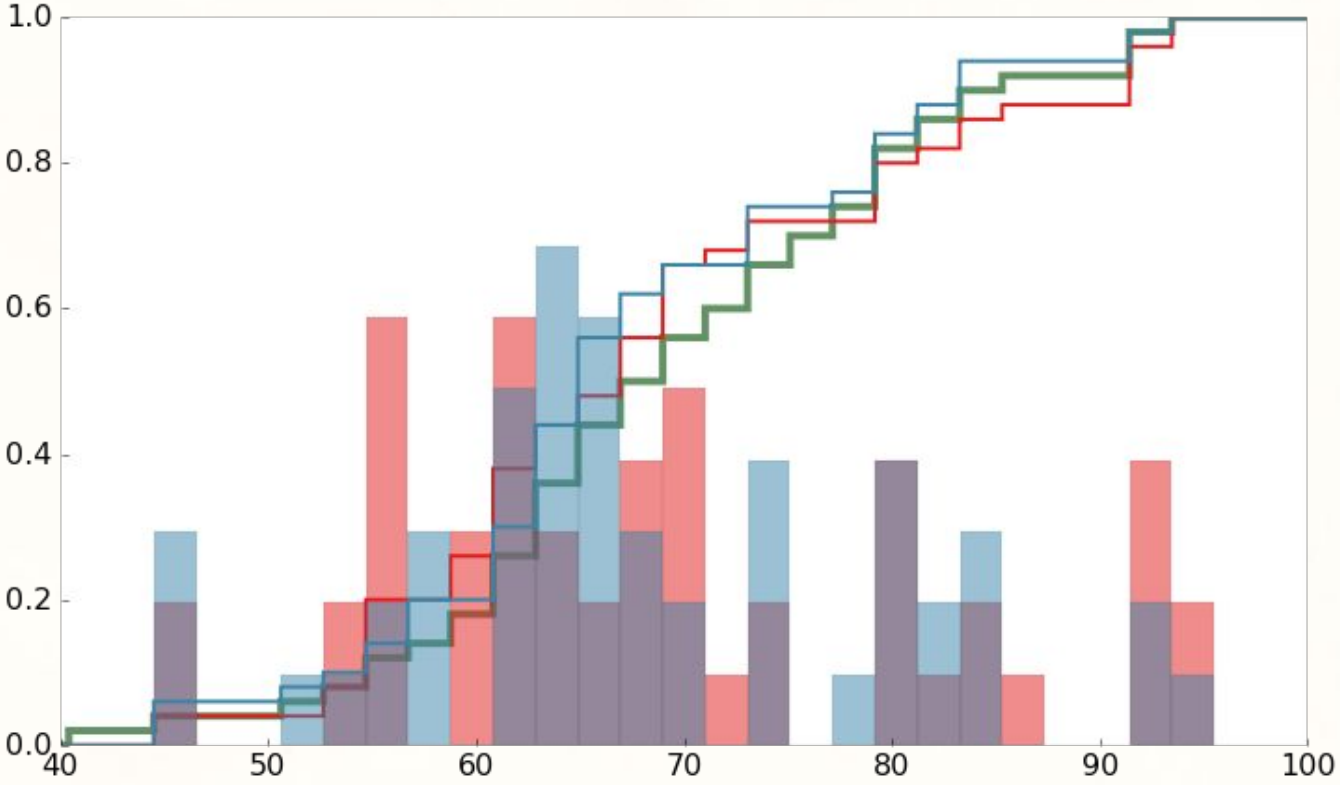
Bootstrapping in action



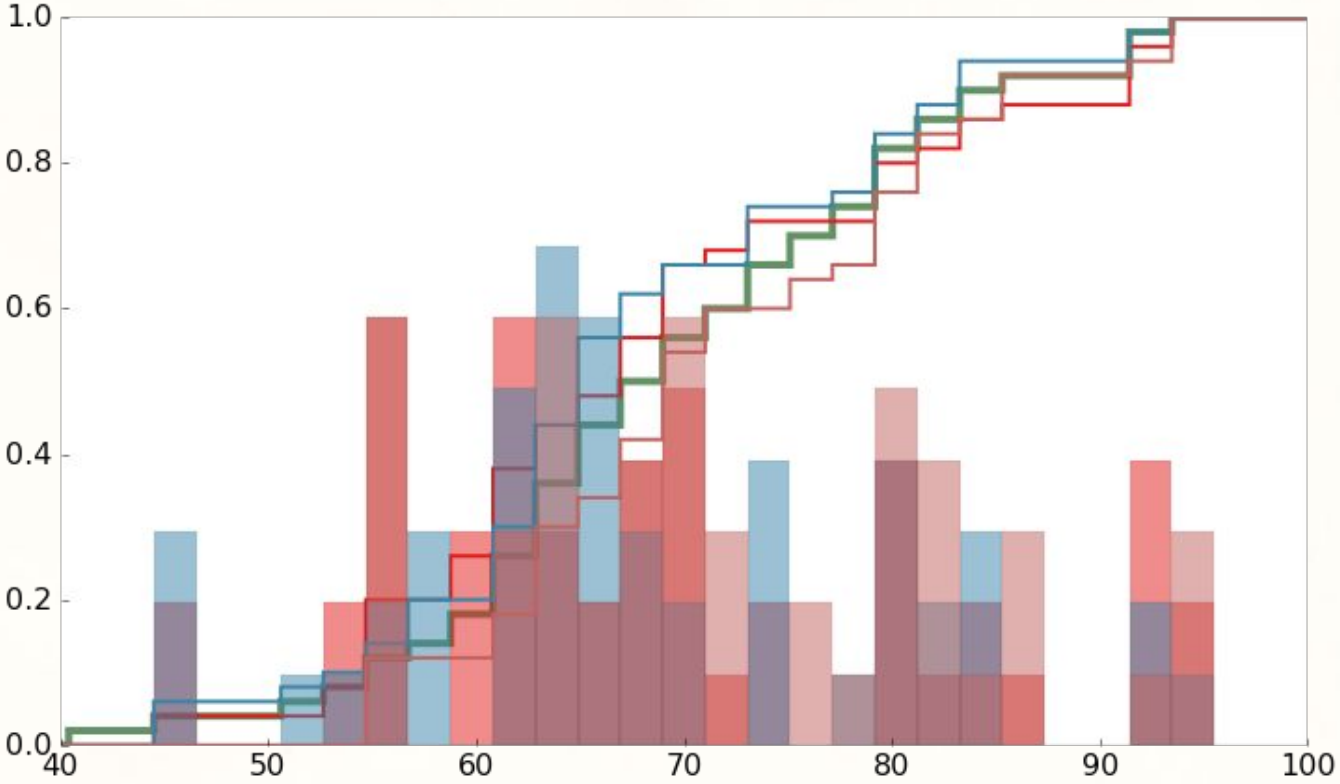
Bootstrapping in action



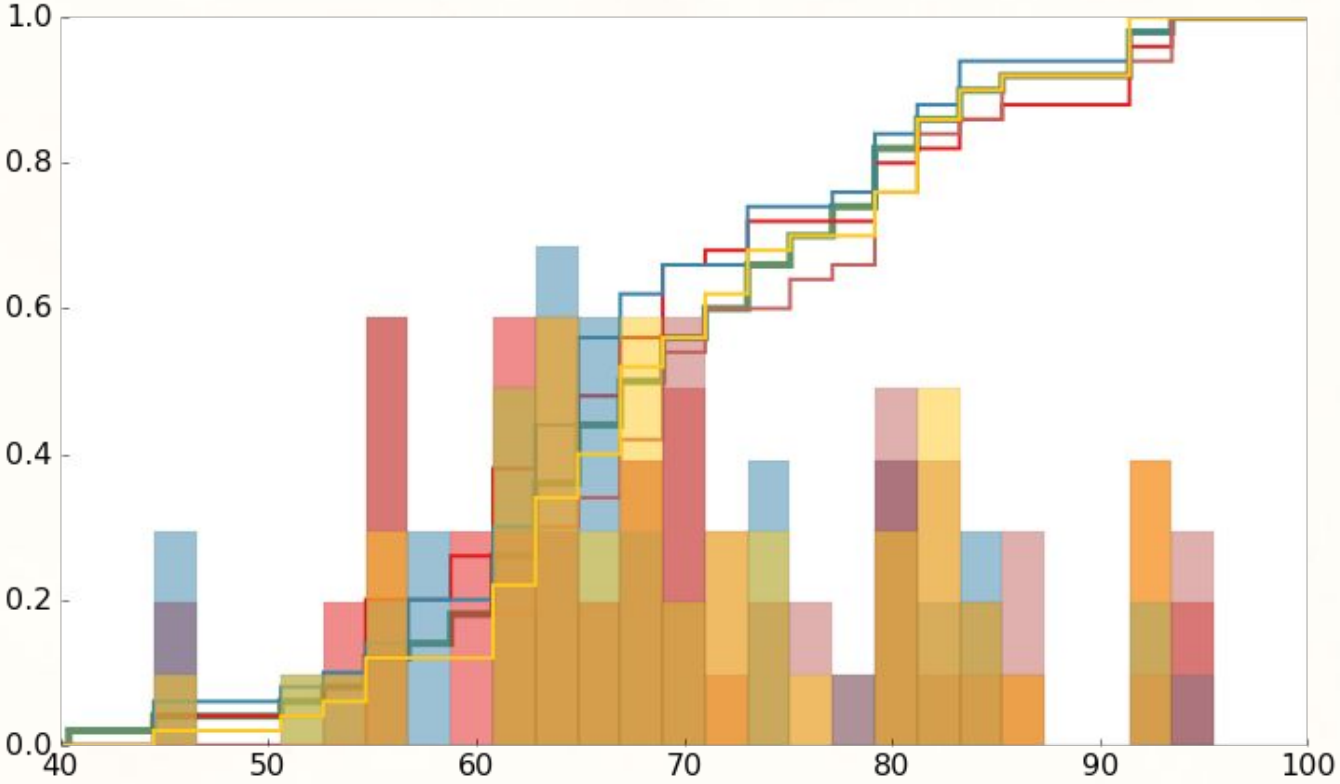
Bootstrapping in action



Bootstrapping in action



Bootstrapping in action



Bootstrapping: bonus

- If you have ever read “confidence intervals were generated using bootstrapping” - this is exactly the same method!
- Bootstrap resampling the data, calculating a statistic, then calculating the 5th and 95th percentile is a very common way to derive 95% confidence intervals

Recap

- We wanted to build many decision trees to average their predictions
- We needed many datasets to do this
- We found out that we can “simulate” datasets from our current data

-> We can build many decision trees from bootstrap samples of our data

“Bootstrap aggregation” -> **bagging**

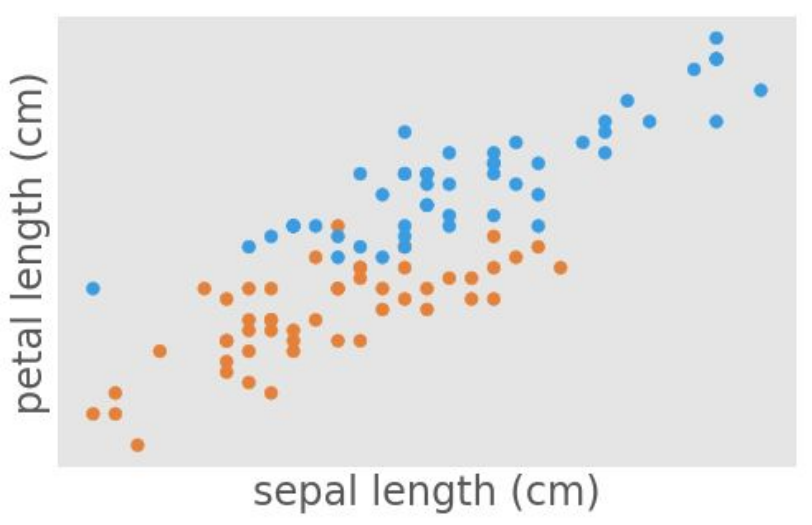
Bootstrap aggregation

1. Create a bootstrap sample of data
2. Build a decision tree on this data
3. Repeat 1-2 until the termination criteria

Bootstrap sample +
Decision surface

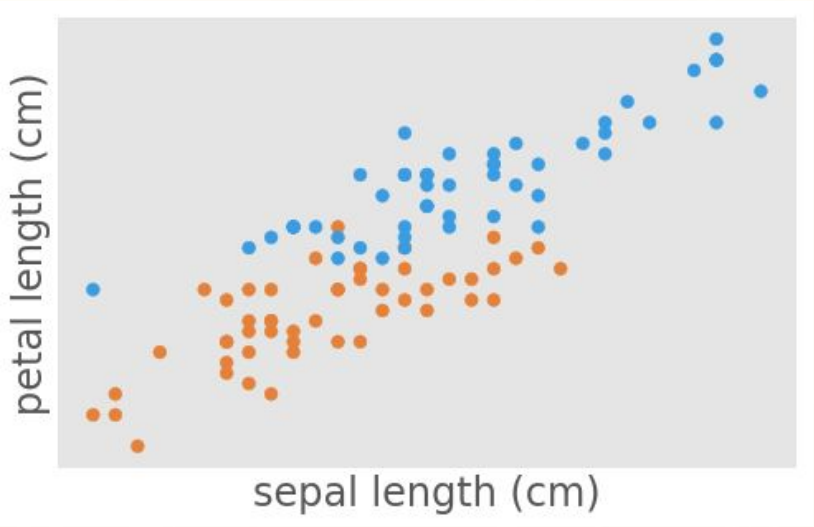
Bootstrap aggregation

Original data

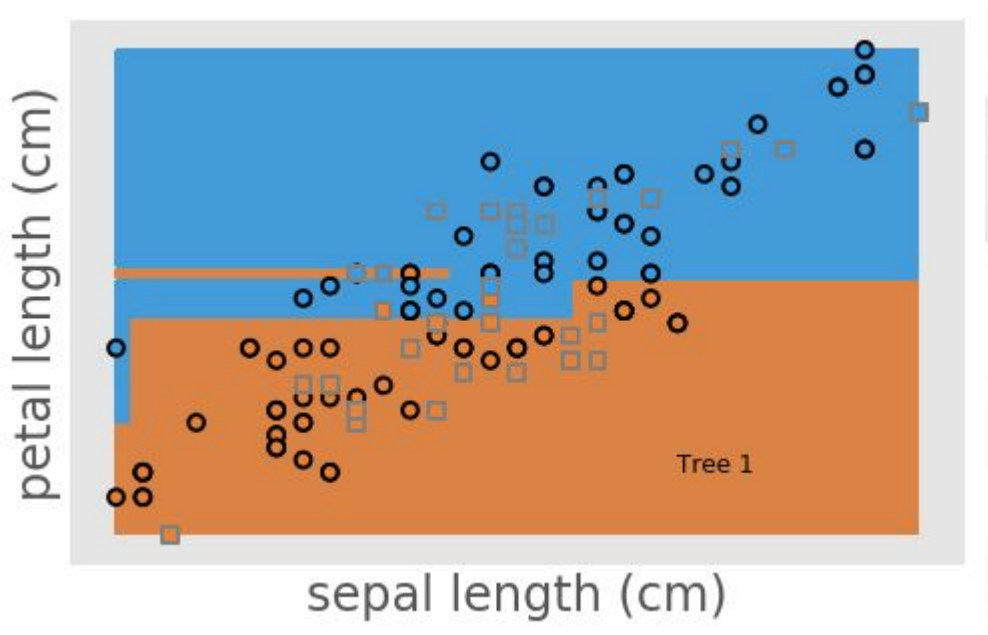


Bootstrap aggregation

Original data

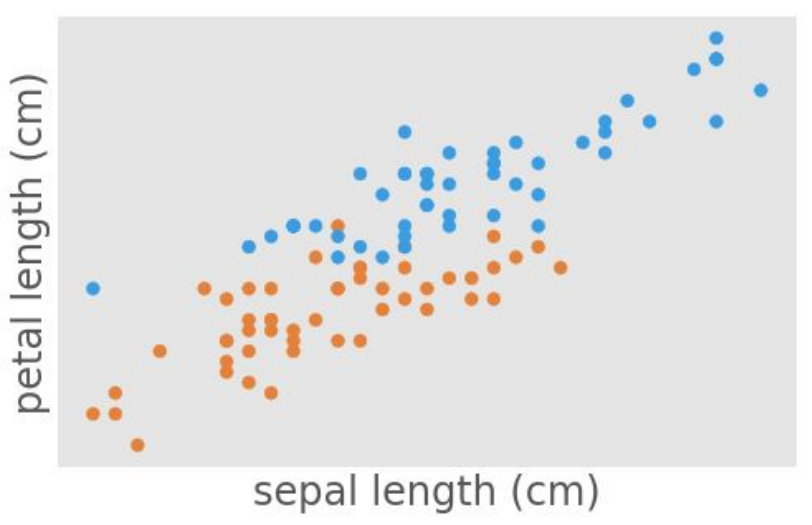


Bootstrap sample +
Decision surface

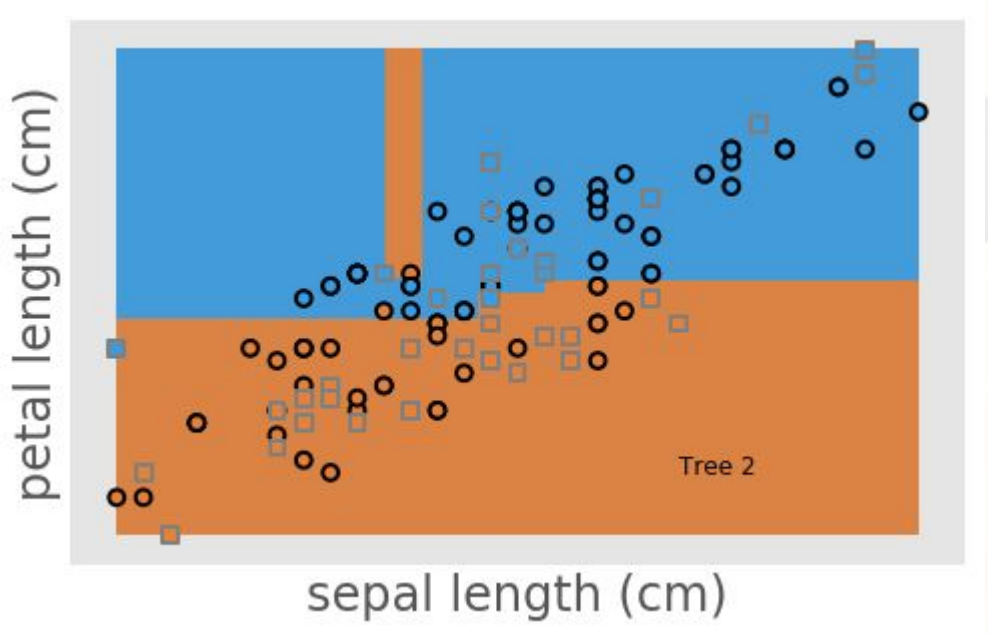


Bootstrap aggregation

Original data

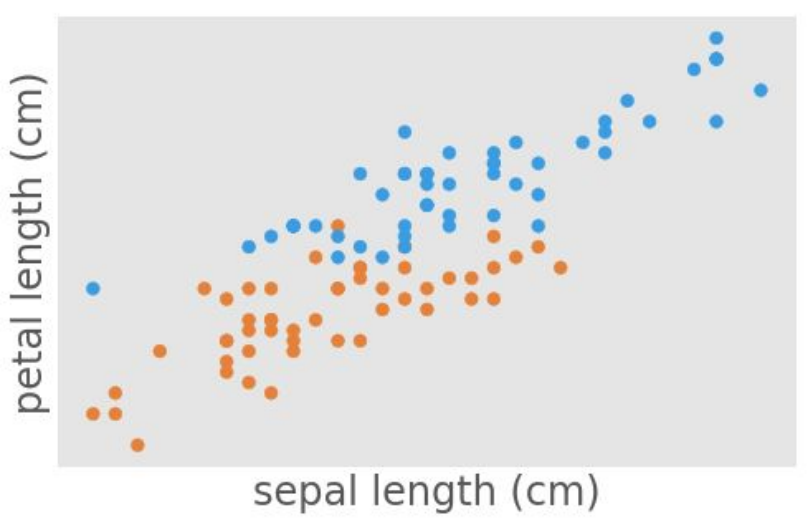


Bootstrap sample +
Decision surface

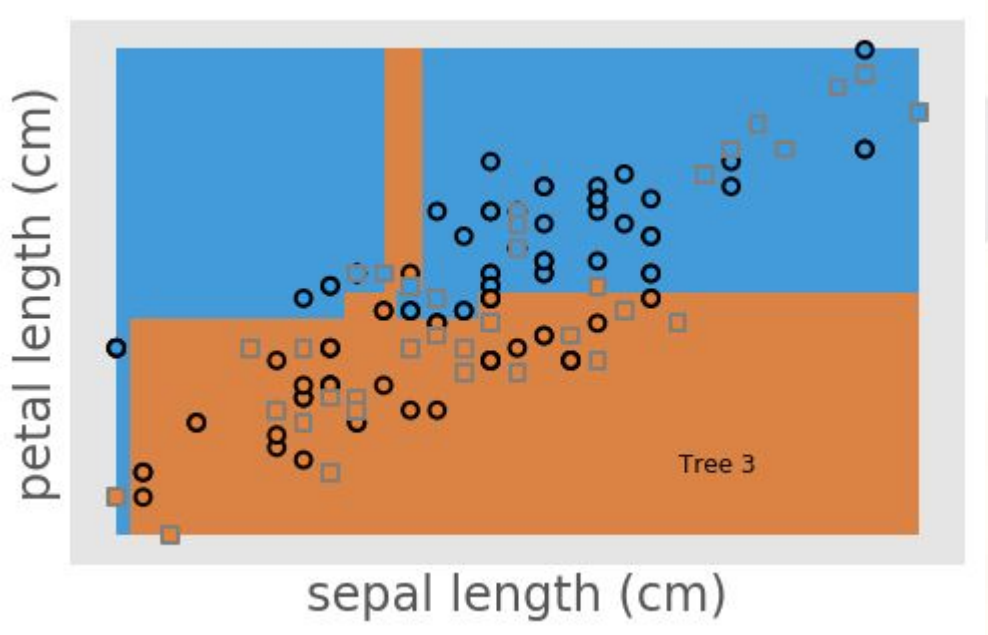


Bootstrap aggregation

Original data

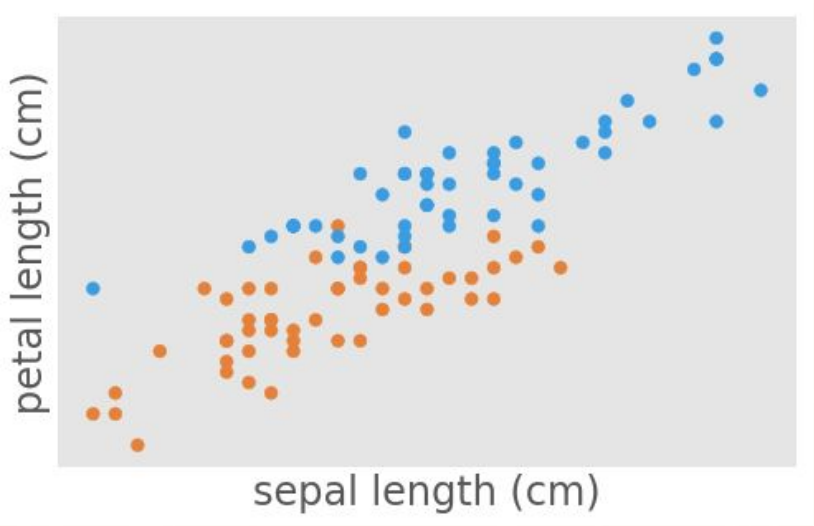


Bootstrap sample +
Decision surface

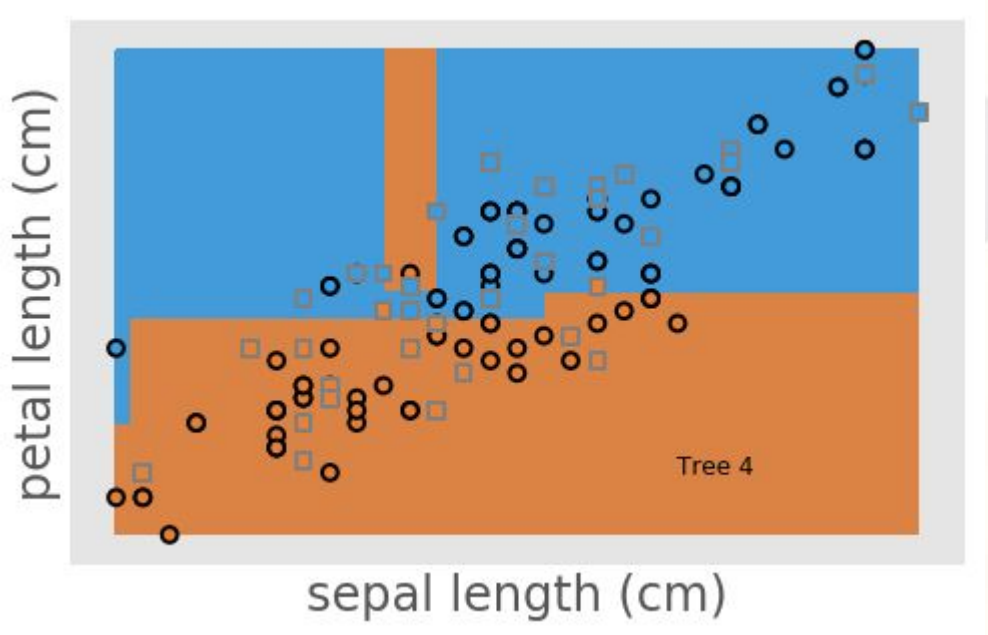


Bootstrap aggregation

Original data

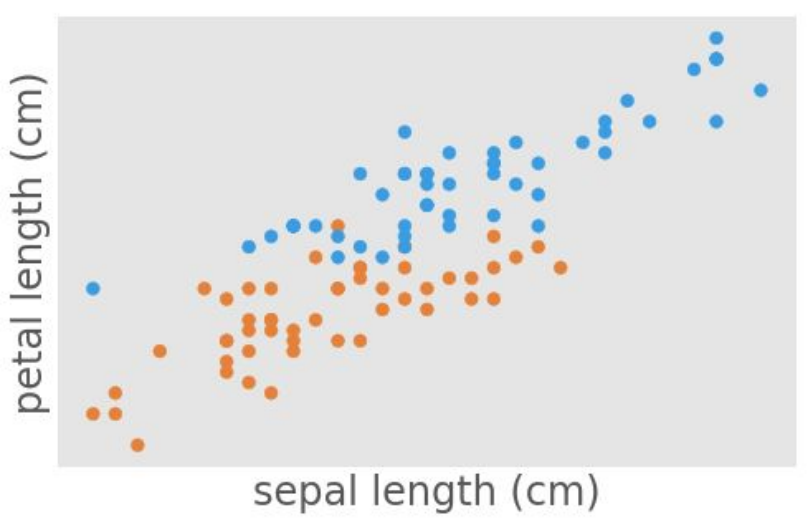


Bootstrap sample +
Decision surface

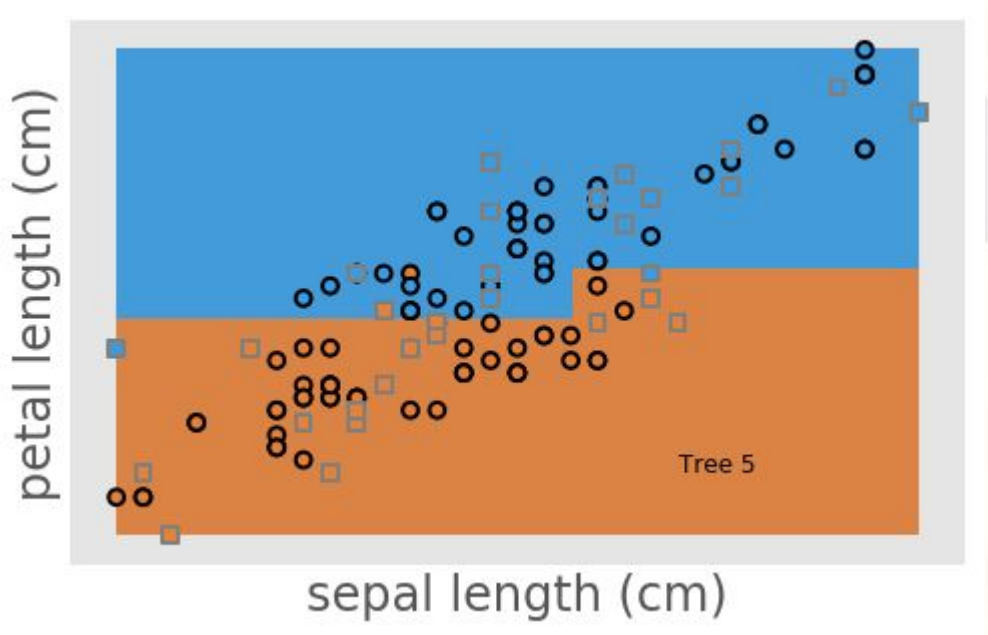


Bootstrap aggregation

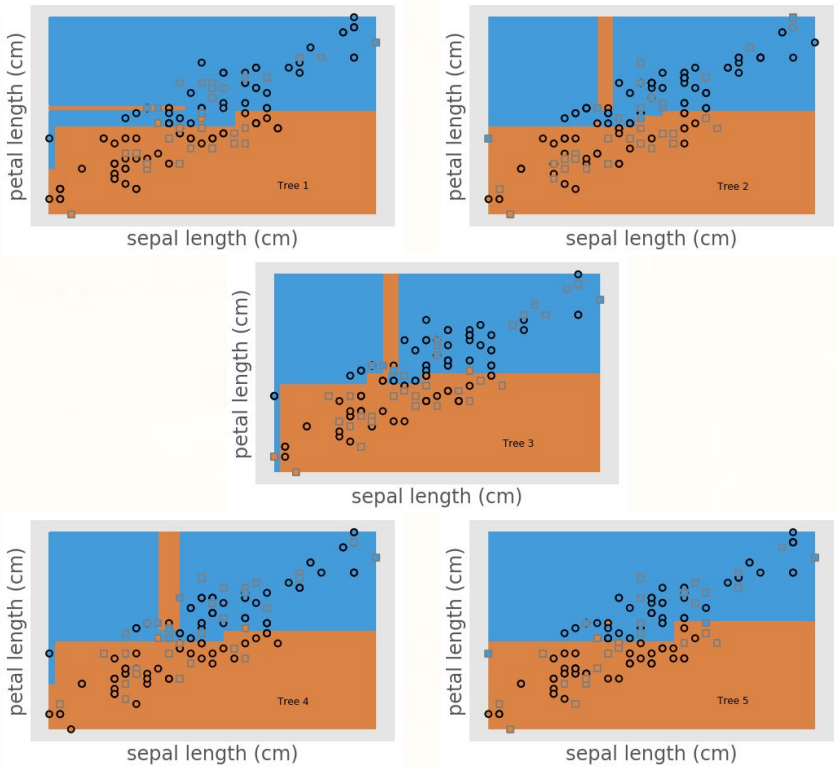
Original data



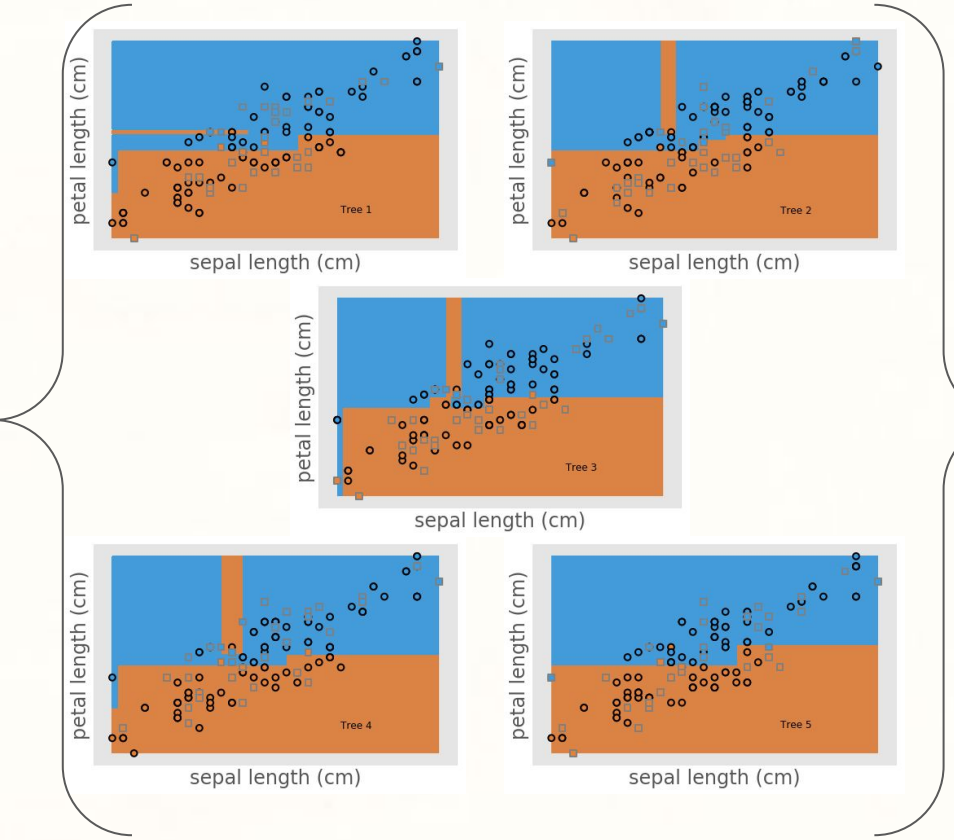
Bootstrap sample +
Decision surface

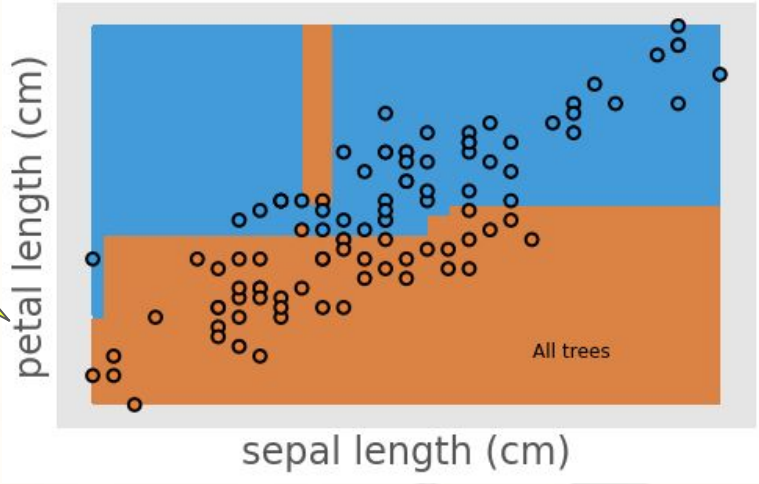


Bootstrap aggregation



Bootstrap aggregation





Bootstrap aggregation

1. Create a bootstrap sample of data
2. Build a decision tree on this data
3. Repeat 1-2 until the termination criteria
4. Average the result of all the trees

Random Forest™

Random Forest

- Random Forest = bagging + subsample *features*
- … that’s it

Random forest

1. Pick a random subset of features (up to M)
   a. (up to M, e.g. M=2 means pick 2 features)
2. Create a bootstrap sample of data
3. Build a decision tree on this data
4. Repeat 1-3 until the termination

Sepal length
Sepal width
Petal length
Petal width

Random forest

1. **Pick a random subset of features (up to M)**
2. Create a bootstrap sample of data
3. Build a decision tree on this data
4. Repeat 1-3 until the termination

Sepal length
Sepal width
Petal length
Petal width

Random forest

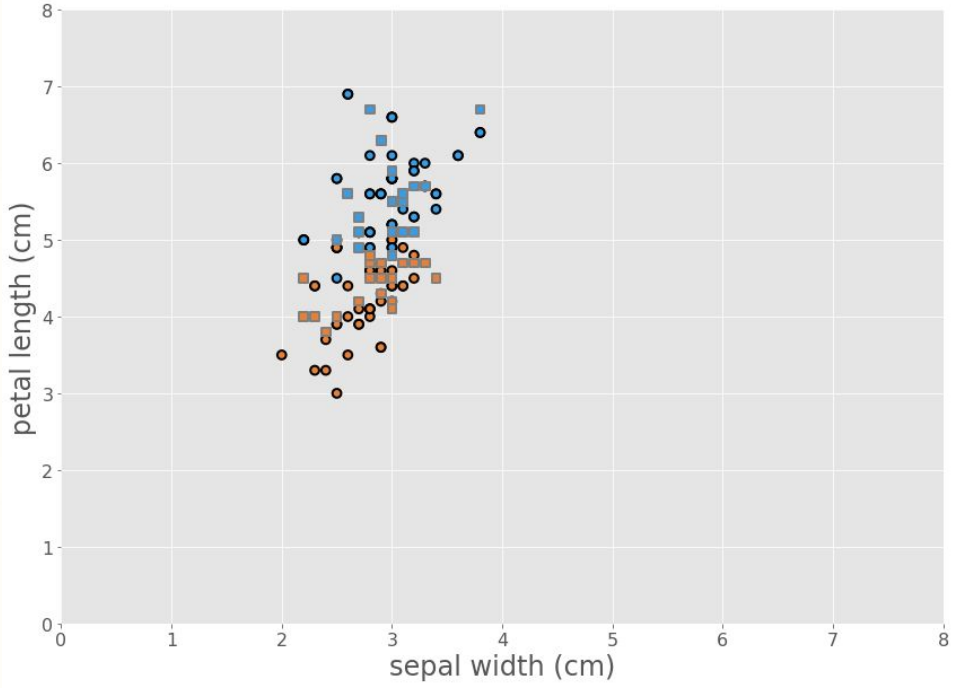
1. **Pick a random subset of features (up to M)**
2. Create a bootstrap sample of data
3. Build a decision tree on this data
4. Repeat 1-3 until the termination

~~Sepal length~~
Sepal width
Petal length
~~Petal width~~

Random forest

1. Pick a random subset of features (up to M)
2. **Create a bootstrap sample of data**
3. Build a decision tree on this data
4. Repeat 1-3 until the termination

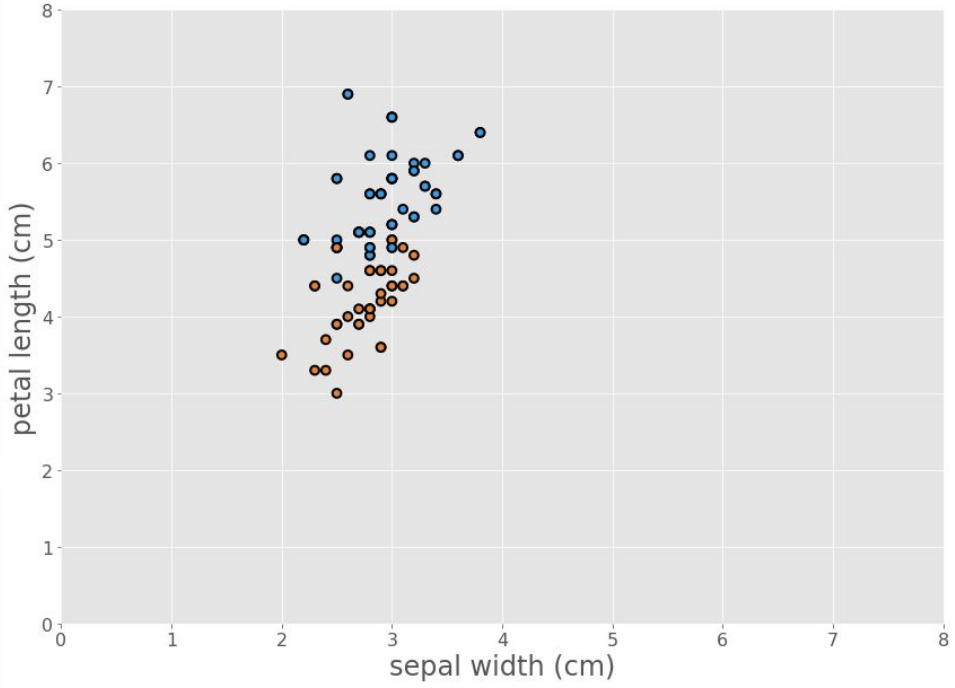
~~Sepal length~~
Sepal width
Petal length
~~Petal width~~



Random forest

1. Pick a random subset of features (up to M)
2. **Create a bootstrap sample of data**
3. Build a decision tree on this data
4. Repeat 1-3 until the termination

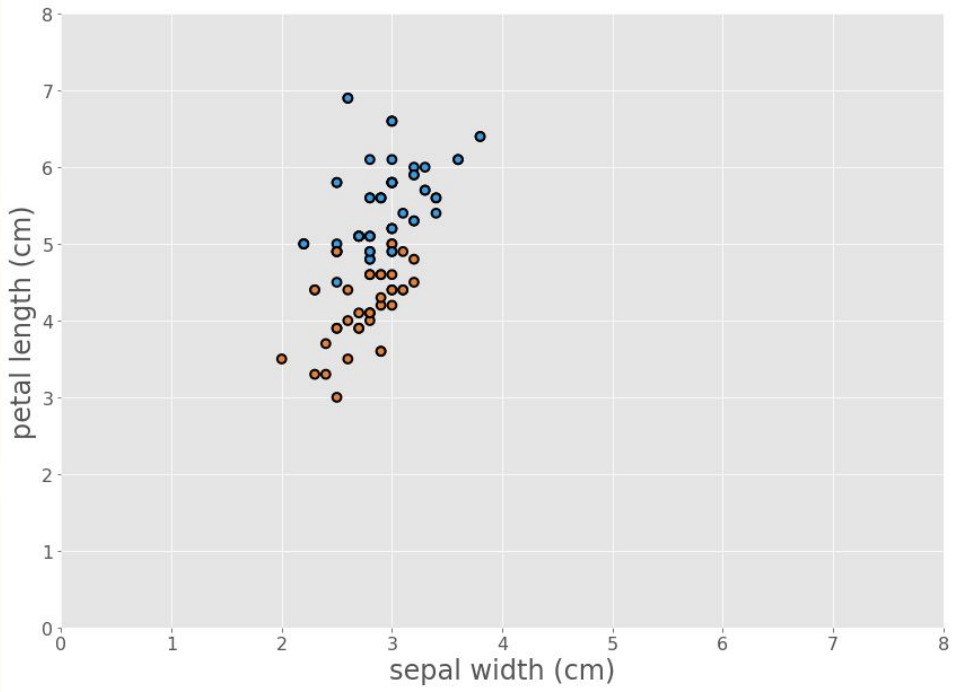
~~Sepal length~~
Sepal width
Petal length
~~Petal width~~



Random forest

1. Pick a random subset of features (up to M)
2. Create a bootstrap sample of data
3. **Build a decision tree on this data**
4. Repeat 1-3 until the termination

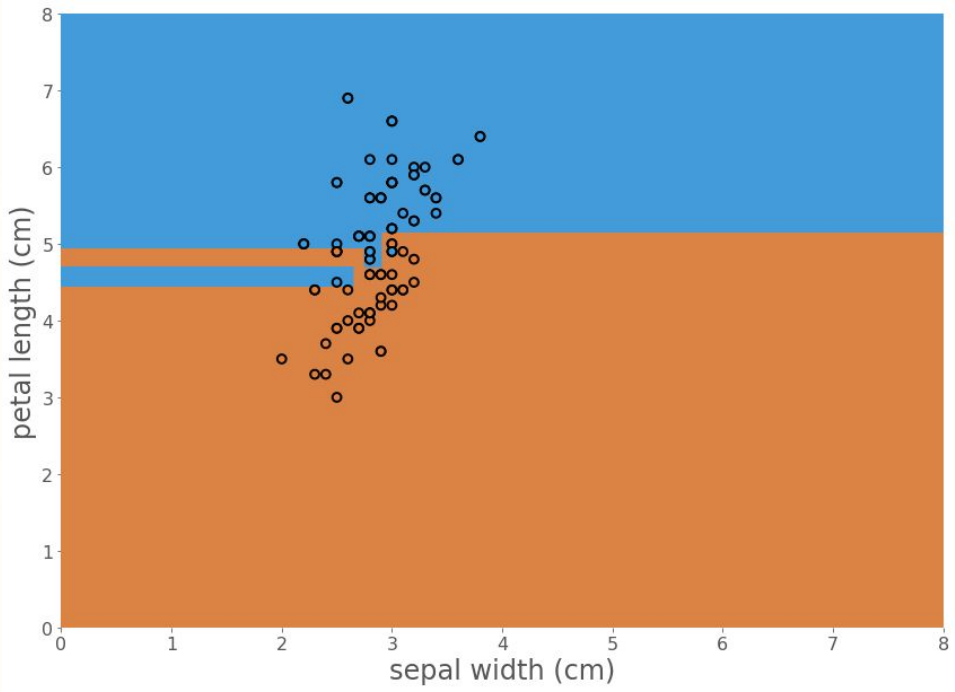
~~Sepal length~~
Sepal width
Petal length
~~Petal width~~



Random forest

1. Pick a random subset of features (up to M)
2. Create a bootstrap sample of data
3. **Build a decision tree on this data**
4. Repeat 1-3 until the termination

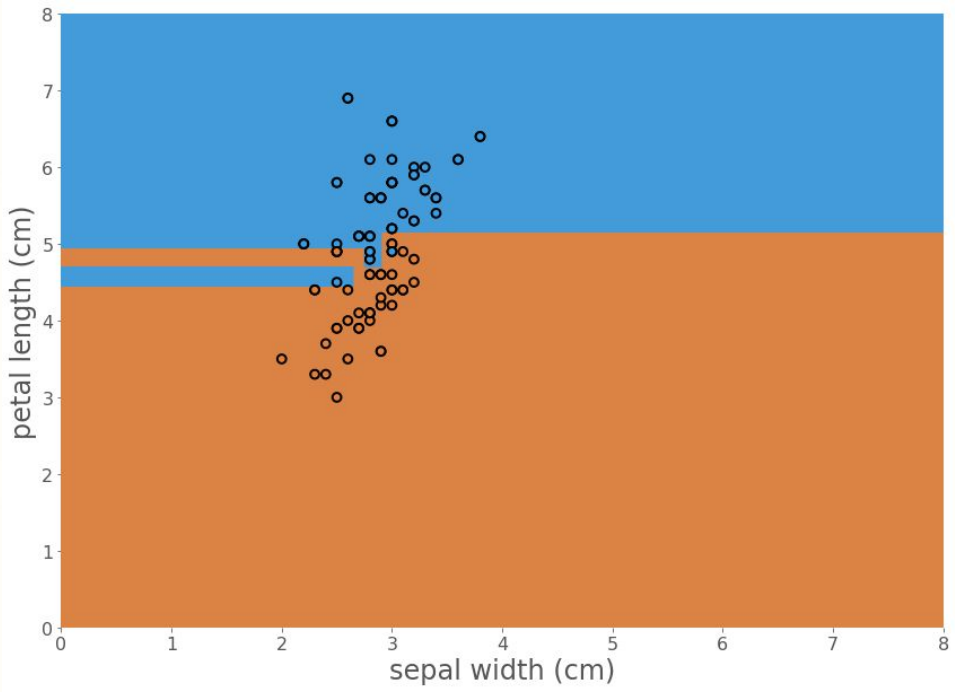
~~Sepal length~~
Sepal width
Petal length
~~Petal width~~



Random forest

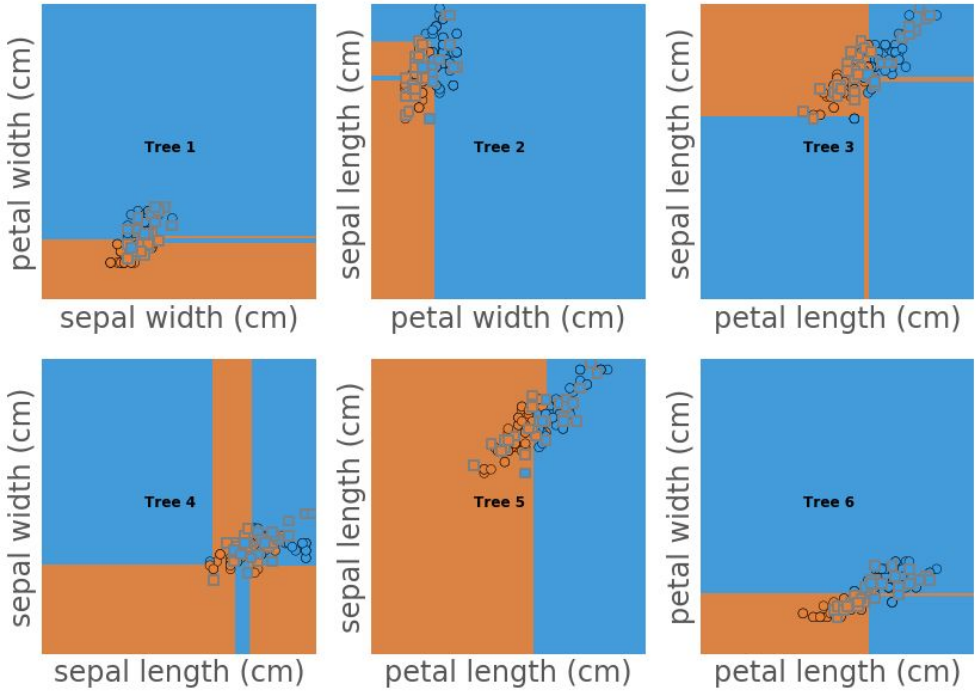
1. Pick a random subset of features (up to M)
2. Bootstrap sample the data
3. Train a decision tree on this subset
4. **Repeat 1-3 until termination**

~~Sepal length~~
Sepal width
Petal length
~~Petal width~~



Random forest

1. Pick a random subset of features (up to M)
2. Bootstrap sample the data
3. Train a decision tree on this subset
4. **Repeat 1-3 until termination**



History of Trees

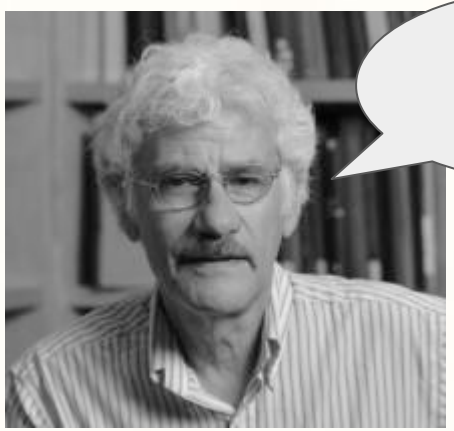
- **1996**: AdaBoost [1]
- **1999**: Formulate AdaBoost as gradient descent using exponential loss [2]
- **2001**: Random forests [3]

Leo Breiman

[1] Freund, Y. and Schapire, R.E., July 1996. Experiments with a new boosting algorithm. ICML (Vol. 96, pp. 148-156).
[2] Mason, L., Baxter, J., Bartlett, P.L. and Frean, M.R., December 1999. Boosting Algorithms as Gradient Descent. In NIPS (pp. 512-518).
[3] Breiman, L.. Random forests. Machine learning, 45(1), pp.5-32.

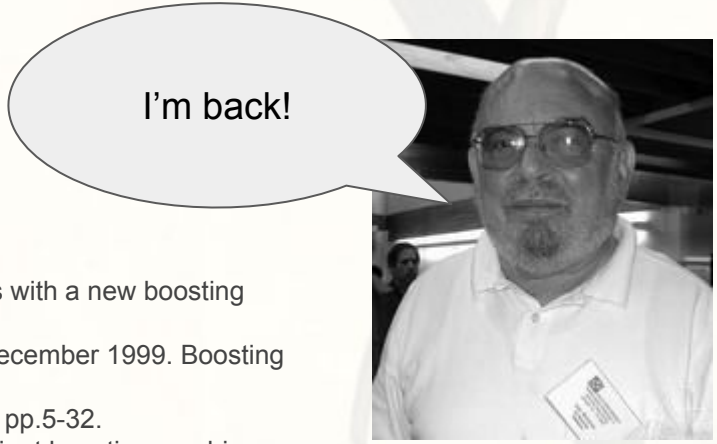
History of Trees

- **1996**: AdaBoost [1]
- **1999**: Formulate AdaBoost as gradient descent using exponential loss [2]
- **2001**: Random forests [3]
- **2001**: Generalized form of “Gradient Boosting” [4]



Jerome H Friedman

[1] Freund, Y. and Schapire, R.E., July 1996. Experiments with a new boosting algorithm. ICML (Vol. 96, pp. 148-156).
[2] Mason, L., Baxter, J., Bartlett, P.L. and Frean, M.R., December 1999. Boosting Algorithms as Gradient Descent. In NIPS (pp. 512-518).
[3] Breiman, L.. Random forests. Machine learning, 45(1), pp.5-32.
[4] Friedman, J.H., Greedy function approximation: a gradient boosting machine. Annals of statistics, pp.1189-1232.



Leo Breiman

Gradient Boosting

Boosting

- Recall boosting: iteratively improve the model by identifying shortcomings

- AdaBoost
  - Shortcomings identified by **re-weighting data observations**
- Gradient Boosting
  - Shortcomings identified by **gradient of the residuals**, or “where should I go next”

- Both models shift focus to **difficult to predict** observations

Gradient boosting

- Let’s say I have a guess (*g*) at the target (*y*)
  - $g_1 = 0.9$, $y_1 = 1.0$
  - $g_2 = 1.6$, $y_2 = 1.4$
- How do I improve this guess *without changing* ***g***?

Gradient boosting

- Let's say I have a guess (*g*) at the target (*y*)
  - $g_1 = 0.9$, $y_1 = 1.0$
  - $g_2 = 1.6$, $y_2 = 1.4$
- How do I improve this guess *without changing g*?
  - I can add numbers to make it match y
  - $y_1 = g_1 + 0.1$
  - $y_2 = g_2 + -0.2$

Gradient boosting

- Let's say I have a guess (*g*) at the target (*y*)
  - $g_1 = 0.9$, $y_1 = 1.0$
  - $g_2 = 1.6$, $y_2 = 1.4$
- How do I improve this guess *without changing g*?
  - I can add numbers to make it match y
  - Let's call these numbers "h"
  - $y_1 = g_1 + h_1$
  - $y_2 = g_2 + h_2$

Gradient boosting

- Let's say I have a guess (*g*) at the target (*y*)
  - $g_1 = 0.9$, $y_1 = 1.0$
  - $g_2 = 1.6$, $y_2 = 1.4$
- How do I improve this guess *without changing g*?
  - I can add numbers to make it match y
  - Let's call these numbers "h"
  - $y_1 = g_1 + h_1$
  - $y_2 = g_2 + h_2$
- OR… equivalently
  - $h_1 = y_1 - g_1$
  - $h_2 = y_2 - g_2$

Gradient boosting

- Let's say I have a guess (*g*) at the target (*y*)
  - $g_1 = 0.9, y_1 = 1.0$
  - $g_2 = 1.6, y_2 = 1.4$
- How do I improve this guess *without changing g*?
  - I can add numbers to make it match y
  - Let's call these numbers "h"
  - $y_1 = g_1 + h_1$
  - $y_2 = g_2 + h_2$
- OR… equivalently
  - $h_1 = y_1 - g_1$
  - $h_2 = y_2 - g_2$

Because my guess is based on data (features "x"), we can write:

- $h_1(x) = y_1(x) - g_1(x)$
- $h_2(x) = y_2(x) - g_2(x)$

Gradient boosting

- $h_1 = y_1 - g_1$
- $h_2 = y_2 - g_2$
  - These are **residuals**

- $h(x_1) = y_1 - g(x_1)$
- $h(x_2) = y_2 - g(x_2)$

New guess

Original guess

Gradient boosting

- $h_1 = y_1 - g_1$
- $h_2 = y_2 - g_2$
  - These are **residuals**

- $h(x_1) = y_1 - g(x_1)$
- $h(x_2) = y_2 - g(x_2)$

New guess

Original guess

h(x) == NEW TREE
g(x) == CURRENT MODEL

Updating our model based on residuals **is equivalent to gradient descent!**

Cauchy A. Methodes générales pour la résolution des systemes d’équations simultanées. C.R. Acad Sci Par 1847, 25:536–538.

Gradient boosting

- Why do I care about gradient descent?

Cauchy A. Methodes générales pour la résolution des systemes d’équations simultanées. C.R. Acad Sci Par 1847, 25:536–538.

Gradient boosting

- Why do I care about gradient descent?

- Our example only worked for regression
- Gradient descent is ***general***
- A lot of research to take advantage of!

Cauchy A. Methodes générales pour la résolution des systemes d'équations simultanées. C.R. Acad Sci Par 1847, 25:536–538.

Gradient boosting

- Why do I care about gradient descent?

- Our example only worked for regression
- Gradient descent is ***general***
- A lot of research to take advantage of!

Cauchy A. Methodes générales pour la résolution des systemes d'équations simultanées. C.R. Acad Sci Par 1847, 25:536–538.

Advantages of Gradient Boosting

- Before (AdaBoost):
  - Ad-hoc approach - reweight data each time you build a new tree
- Now (Gradient Boosting):
  - We define a **loss function**, and then we do **gradient descent**

Using gradient boosting...

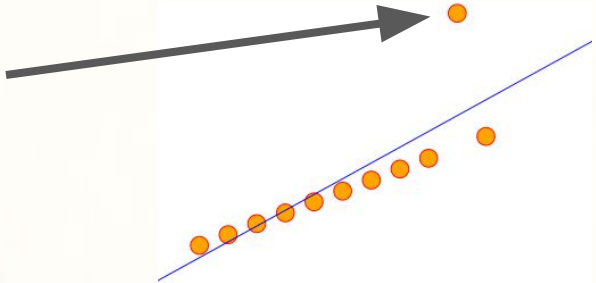
- We can use **any loss function**
- We can **regularize our model**

Advantage #1: Defining our own loss function

- Many classifiers use a sum of squared errors loss function…
- This loss function **<u>focuses on outliers</u>**

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
$L = (y - F)^2/2$	0.005	0.02	0.125	5.445

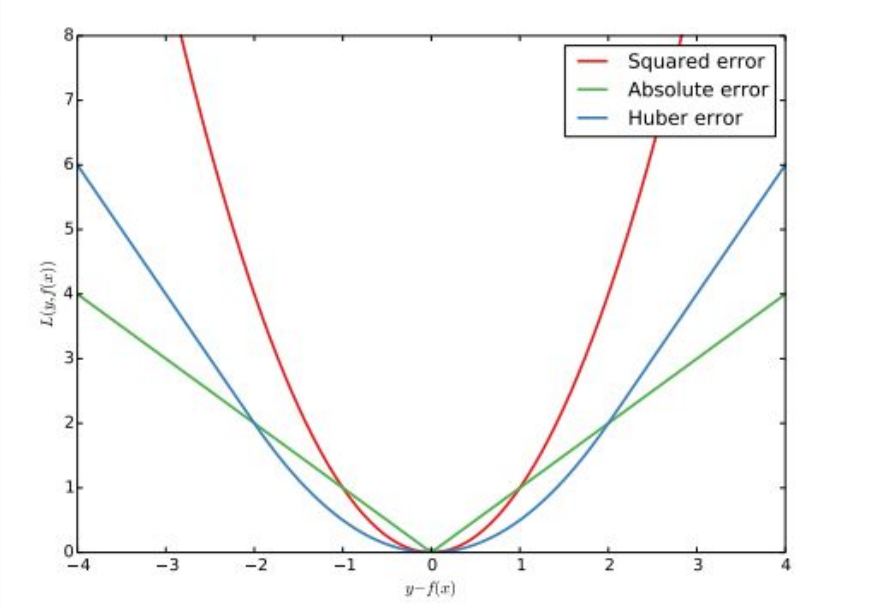
This outlier has “moved” our line up, higher than it should be!

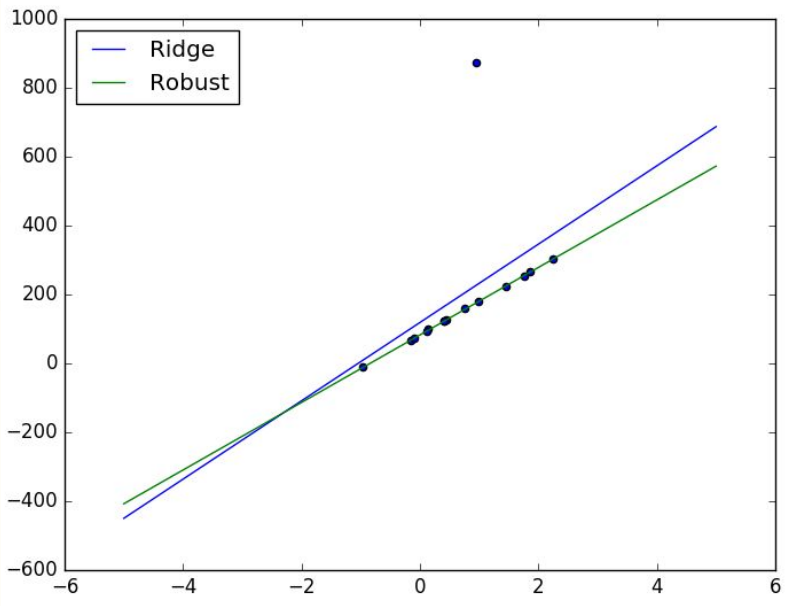


Li, C., 2014. A Gentle Introduction to Gradient Boosting.
URL: http://www.ccs.neu.edu/home/vip/teach/MLcourse/4_boosting/slides/gradient_boosting.pdf

Advantage #1: Defining our own loss function

- Other loss functions are **less sensitive to outliers**

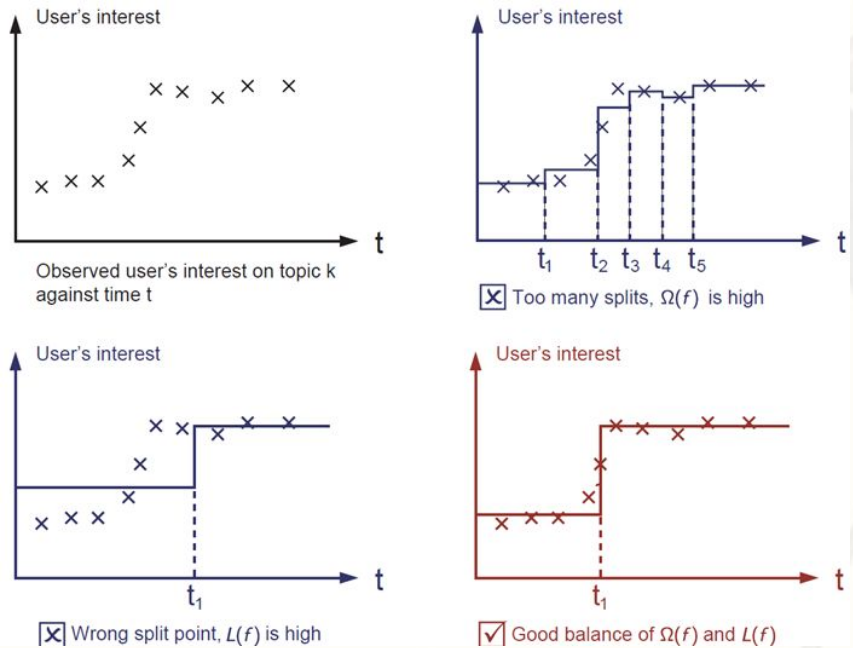




Peter Prettenhofer, Gilles Louppe. Gradient Boosted Regression Trees.
http://www.slideshare.net/DataRobot/gradient-boosted-regression-trees-in-scikitlearn

Advantage #2: Regularization

- **Regularization** - explicitly balancing performance with complexity
- **L** - performance
- **Ω** - complexity

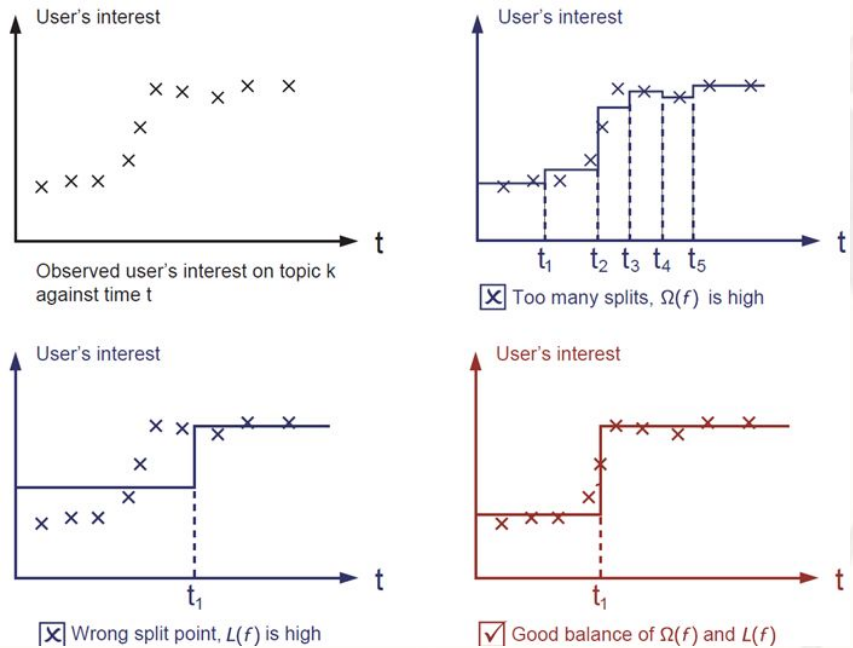


Advantage #2: Regularization

- **Regularization** - explicitly balancing performance with complexity
- **L** - performance
- **Ω** - complexity

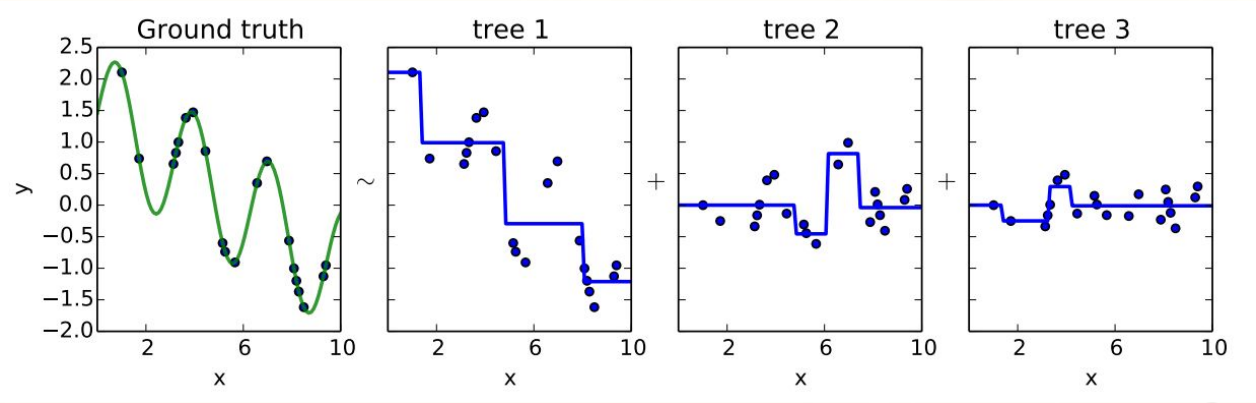
Before, we did this **heuristically**

Now, we can do it **numerically**

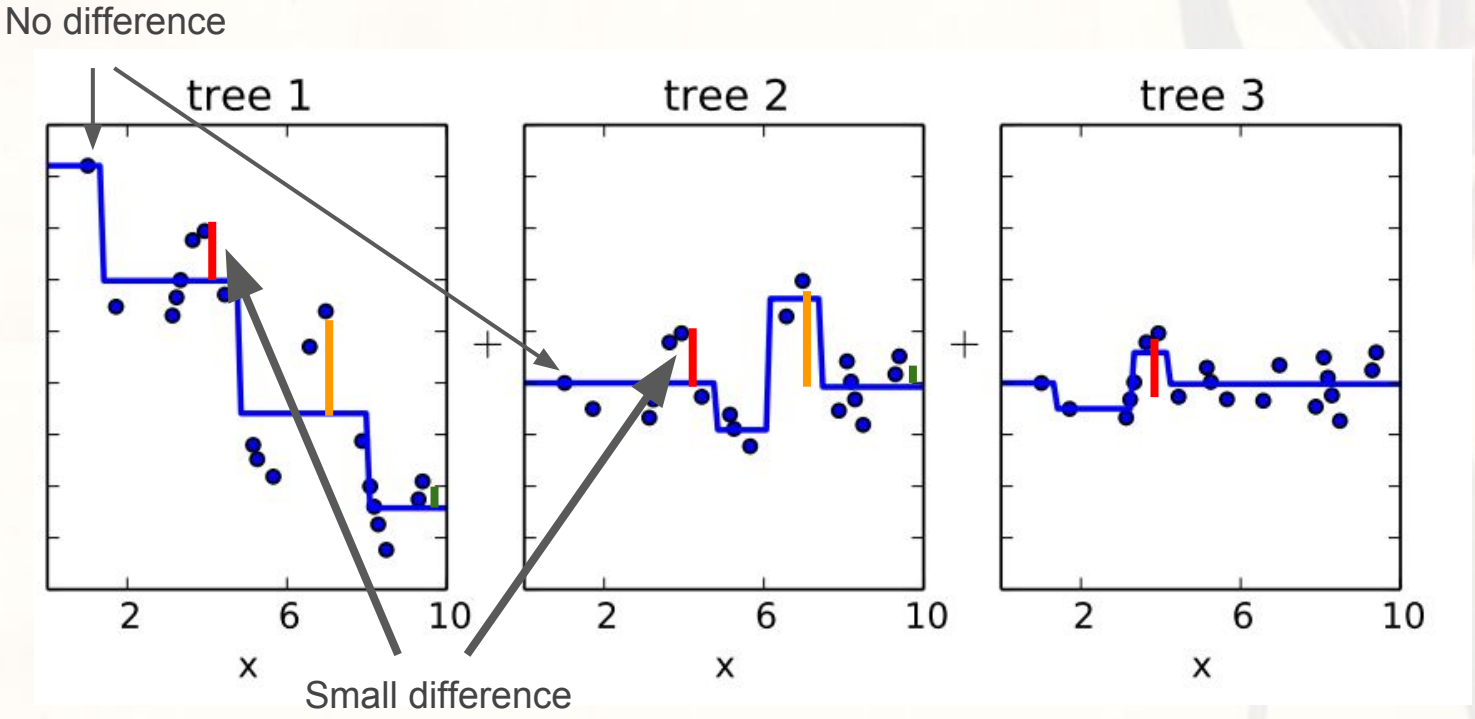


Algorithm for Gradient Boosting

1. Build a tree
2. Calculate negative gradients of the current model
3. Fit a new tree to those negative gradients
4. Update: model = model + new tree
5. Repeat 2-4 until termination



Algorithm for Gradient Boosting - detail



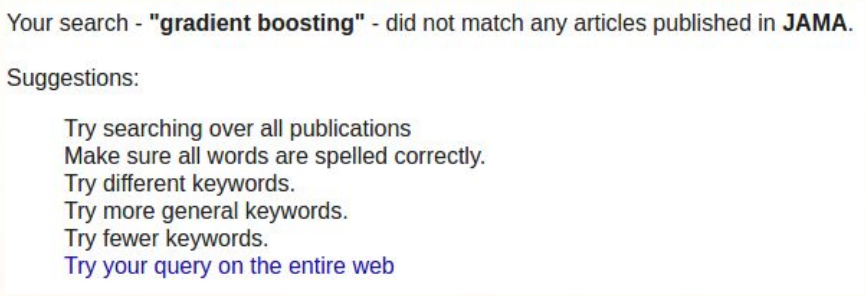
Summary

Techniques reviewed

- Decision tree
  - Good interpretability, poor performance unless task is easy
- AdaBoost
  - Loses some interpretability, good performance
- Bagging, Random Forest
  - Robust classifier, works well on many problems
- Gradient Boosting
  - Solid foundation for trading off performance/complexity, excellent performance

Popularity of techniques

How many results for JAMA articles (including subjournals e.g. JAMA Oncology):

- “Decision tree” - 176
- “Adaboost” - 1 (JAMA Otolaryngology)
- “Random forest” - 12
- 

Further reading

Schapire RE, Freund Y. Boosting: Foundations and Algorithms. MIT Press; 2012

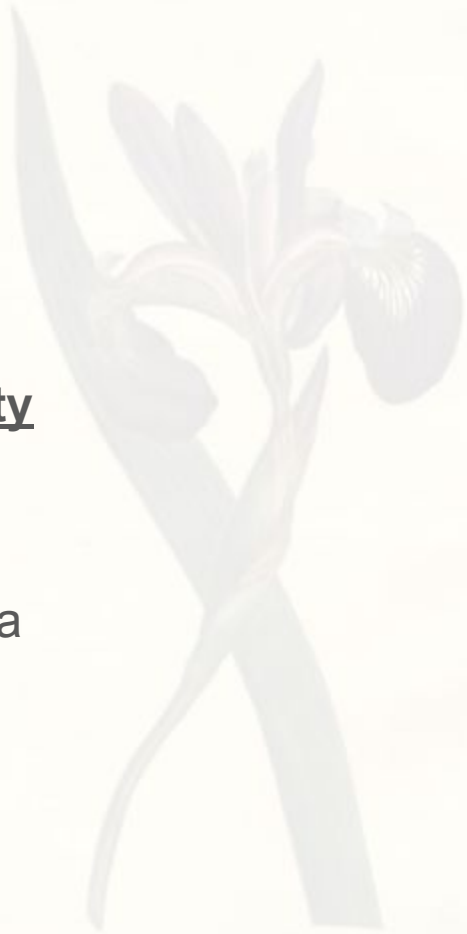
… contains worked out examples and many extensions of boosting

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Springer, Berlin: Springer series in statistics, 2001.

… fantastic book explaining many machine learning models

If you only remember three things...

- Decision trees perform poorly, but provide interpretability
- Forest models **have high accuracy** but **lose interpretability**
- Gradient boosting is the method of choice for structured data



Workshop!

- Run through everything you've learned
- Ask for help when you need it

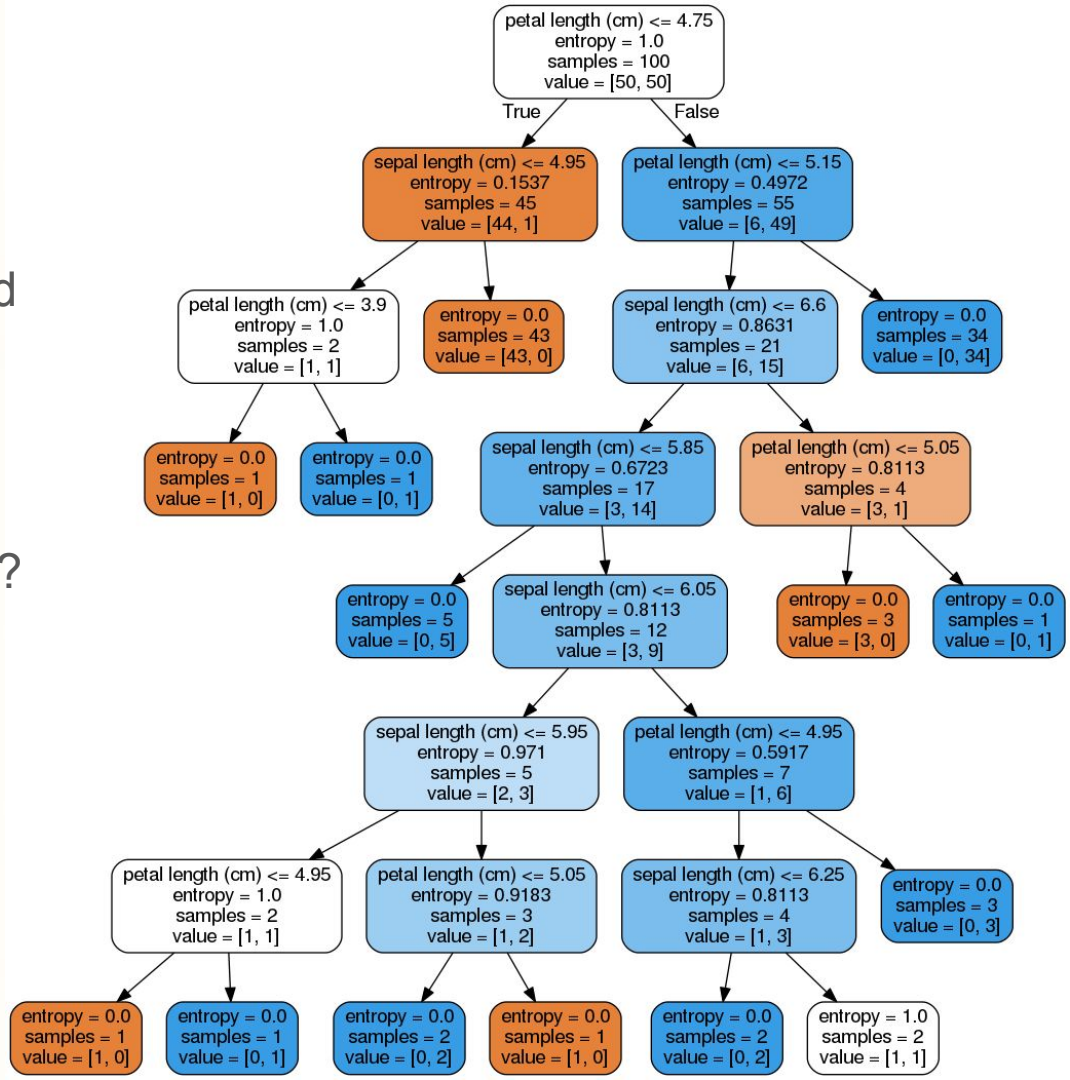
Appendix

Interpretation: feature importance

Feature importance

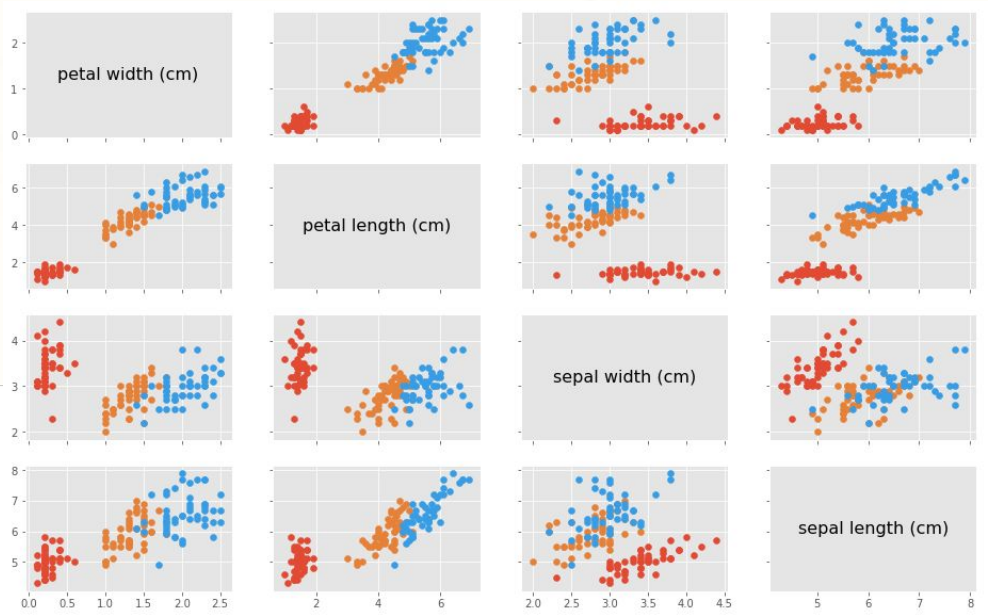
Another measure: how frequently did the feature show up?

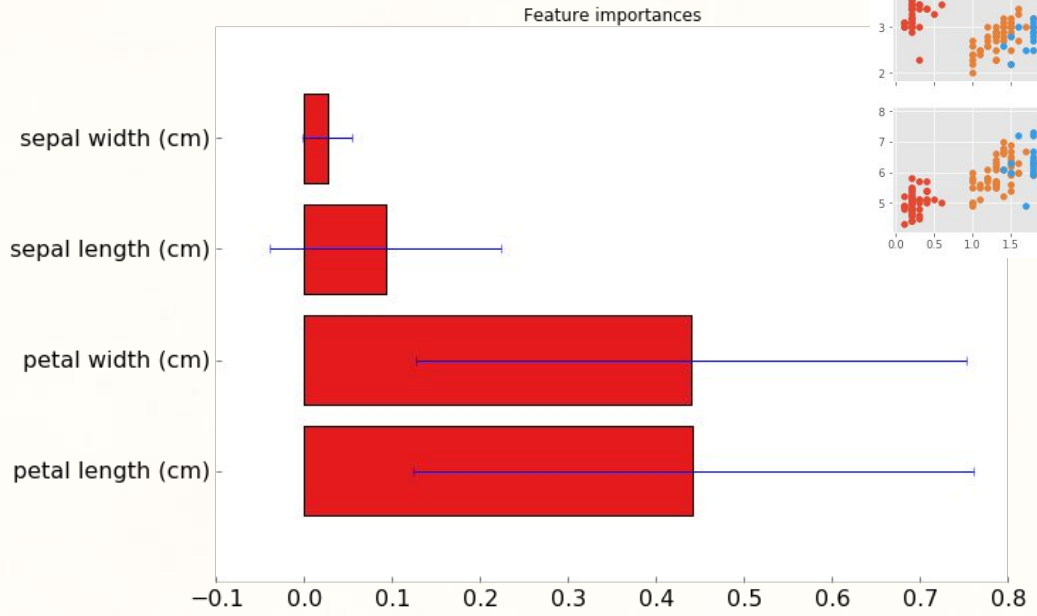
What was the average impurity gain?



Surrogate importance

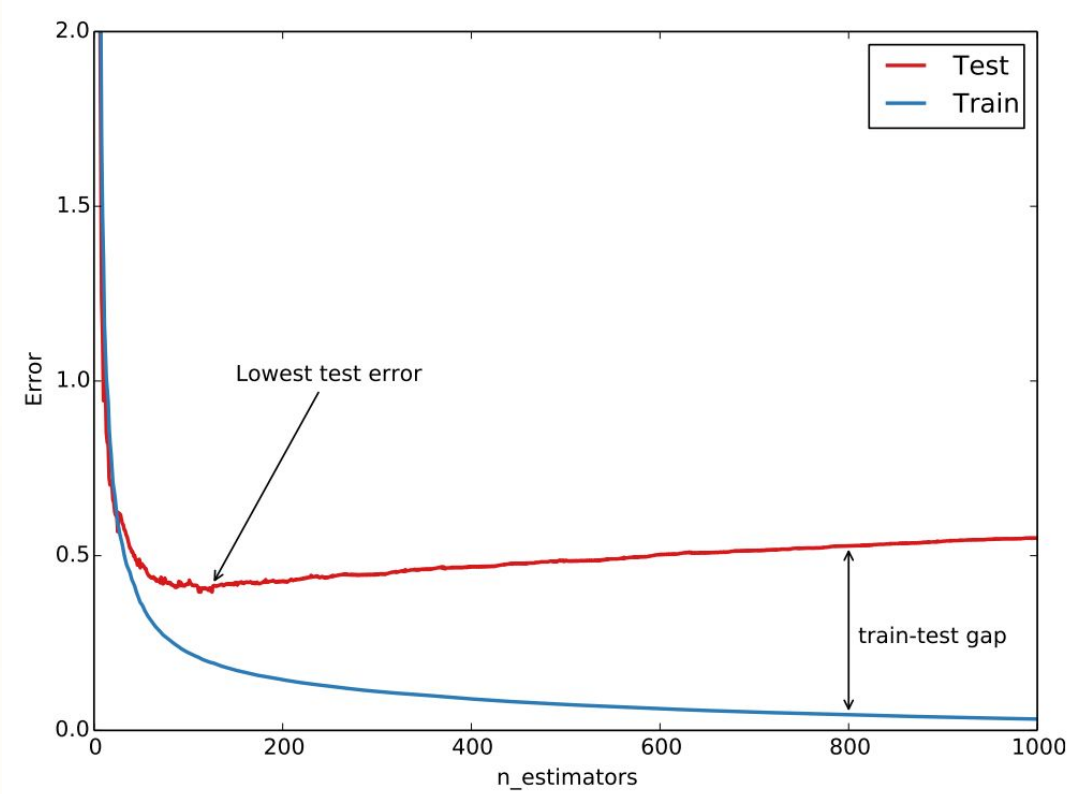
- Note: We can use “out of bag” samples in RF to get feature importance





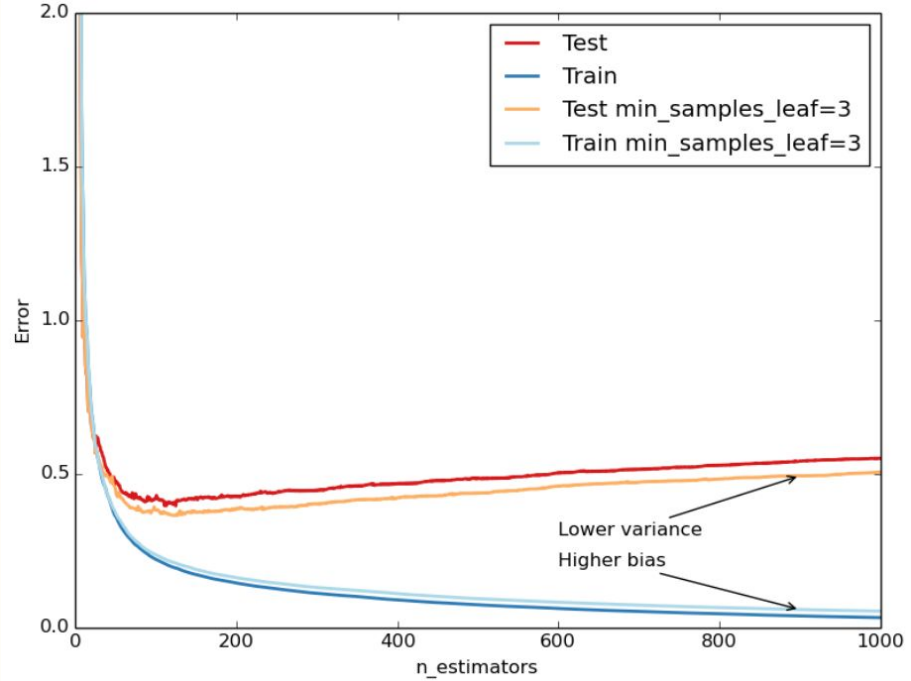
Bells and whistles for Gradient Boosting

- Tree depth
- Shrinkage
- Stochastic gradient descent

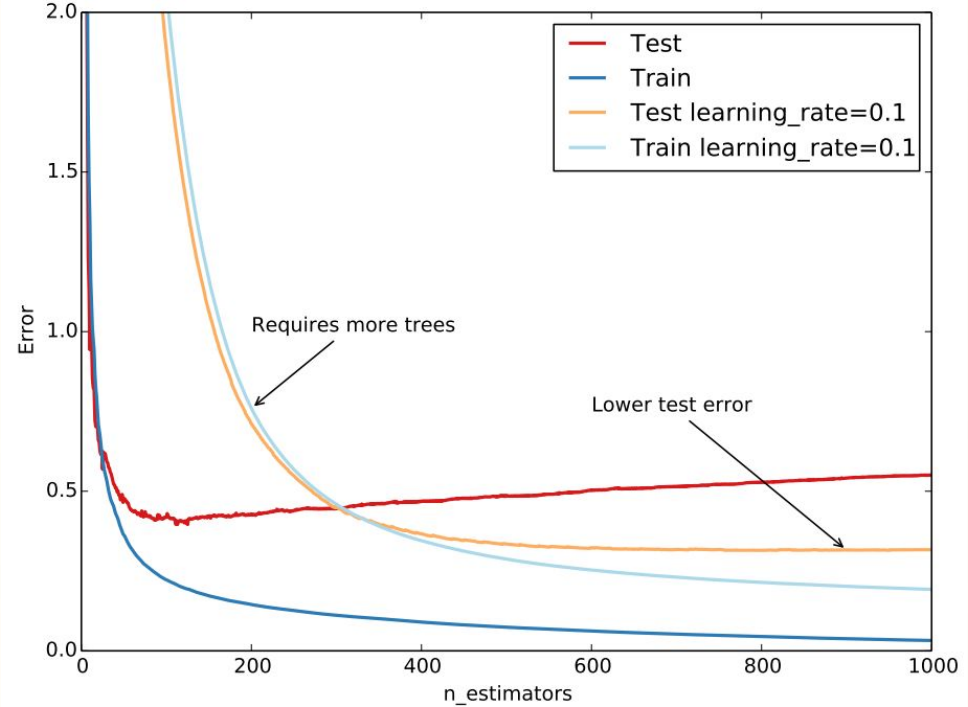


Tree depth (heuristic)

- Max depth of individual trees (feature interactions)
- Minimum samples per leaf node

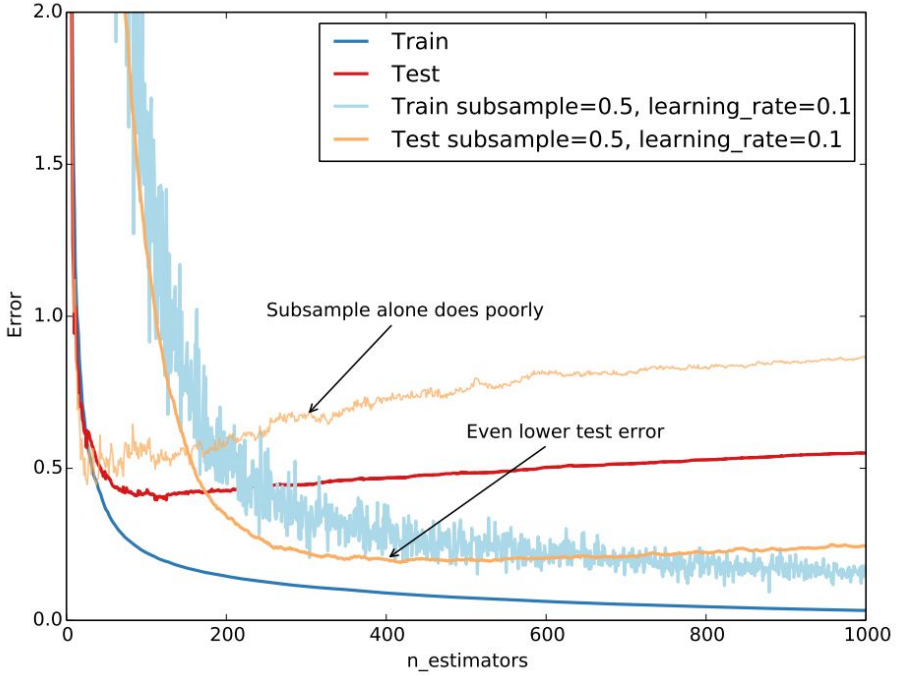


Shrinkage (numeric)



Stochastic Gradient Descent

- Use random data for each tree, rather than the full dataset
- ** Also use random subset of features
- Converges slower but performs better



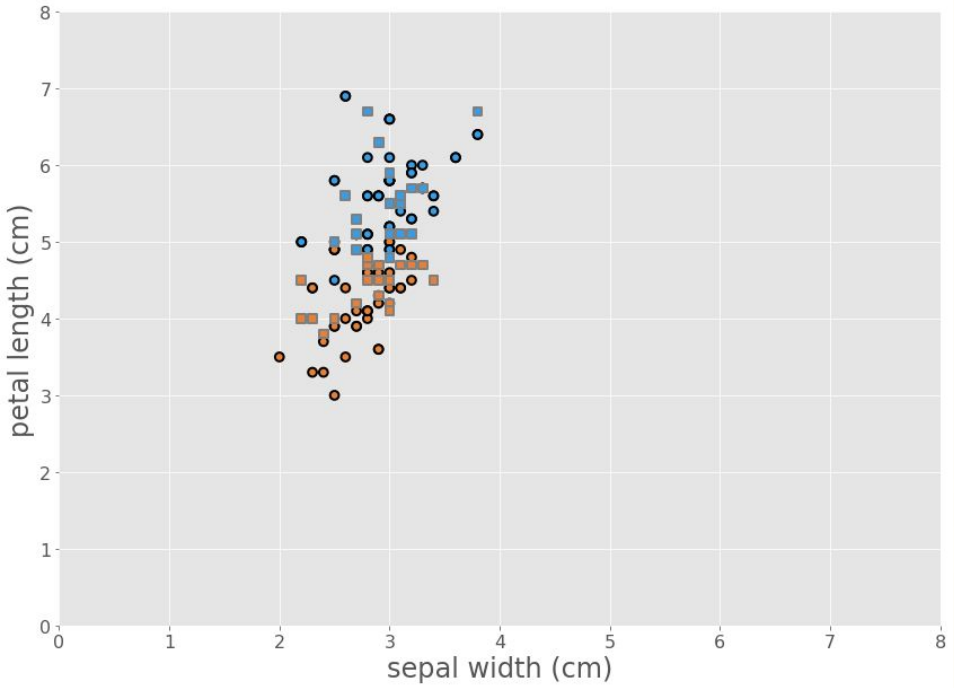
Random Forest

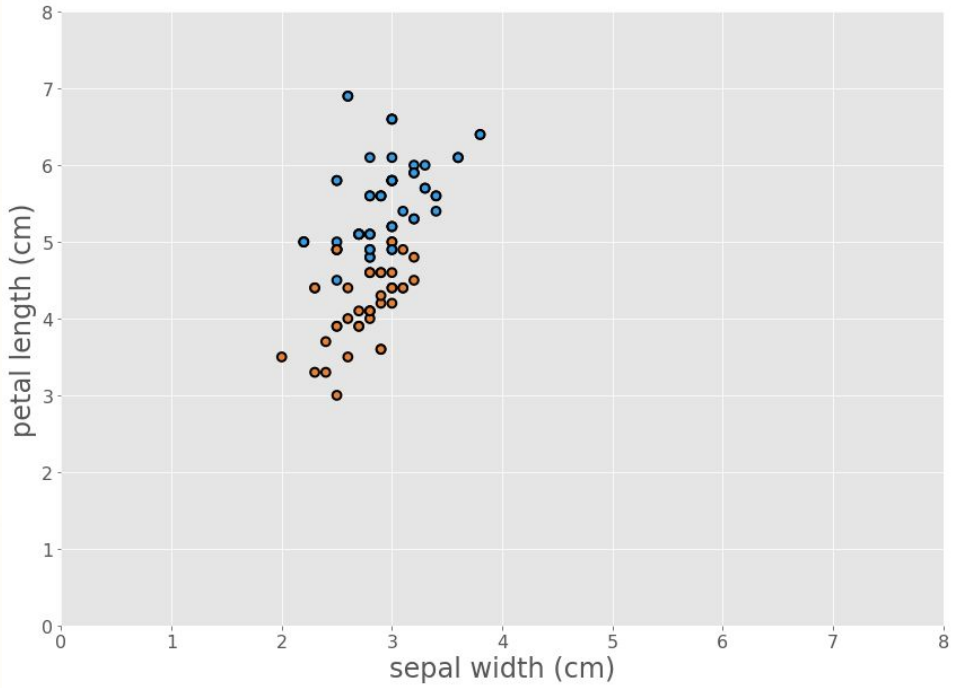
Random forest bonuses

- Out of bag predictions
- Feature importance

Out of bag predictions

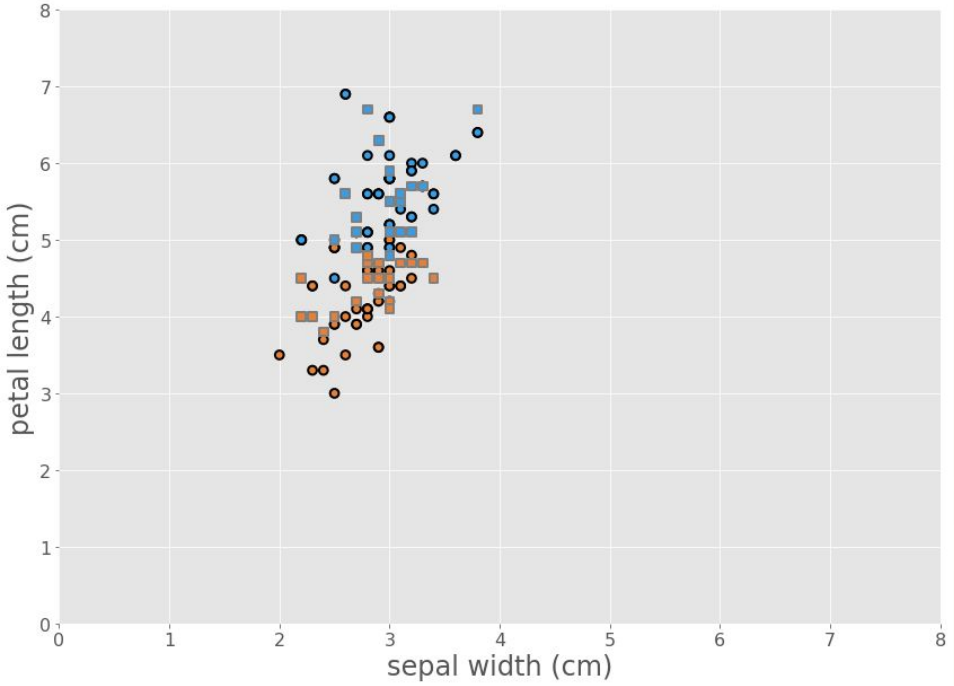
Recall that we exclude samples when building each tree





Out of bag predictions

Recall that we exclude samples when building each tree

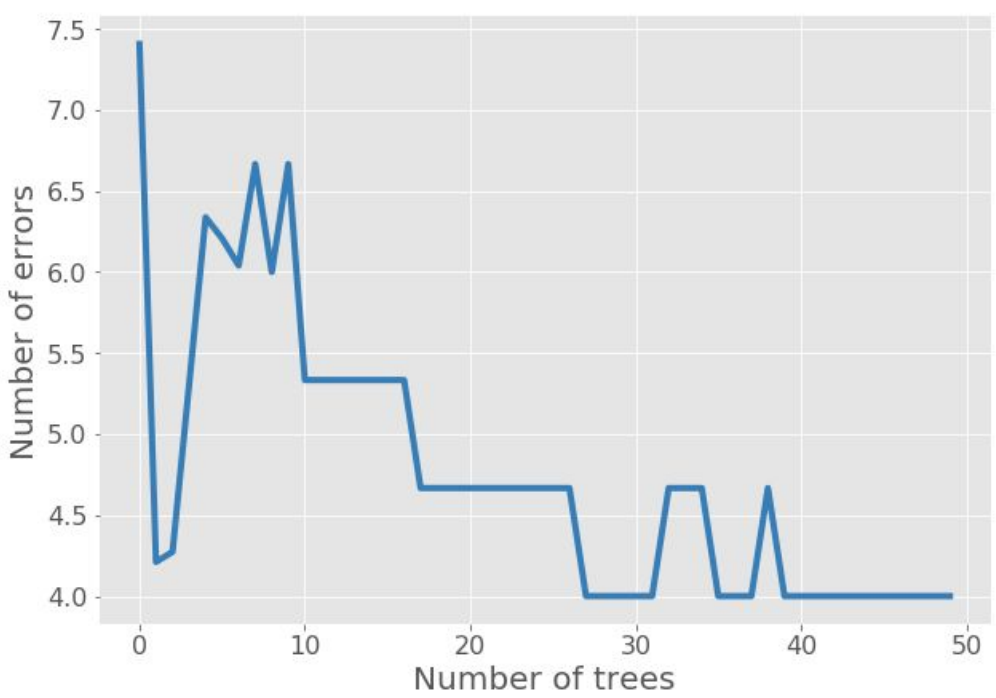


We can use these samples as a **validation set**

This means that we can get a performance measure of our model **for free!**

Out of bag predictions

Recall that we exclude samples when building each tree



We can use these samples as a **validation set**

This means that we can get a performance measure of our model **for free!**

Missing data in Decision trees

Majority: all goes to the node which already has the biggest number of instances (CART, is not the primary rule)

Proportion: distribute to all children, but with diminished weights, proportional with the number of instances from each child node (C45 and others)

Random: distribute randomly to only one single child node, eventually according with a categorical distribution (faster running time than other rules)

Surrogates: build, sort and use surrogates to distribute instances to a child node. Surrogates are input features which resemble best how the test feature sent data instances to the left or right child node (CART, if that fails, majority rule is used)

The background is an Iris!

Curtis's botanical magazine

Iris Virginica