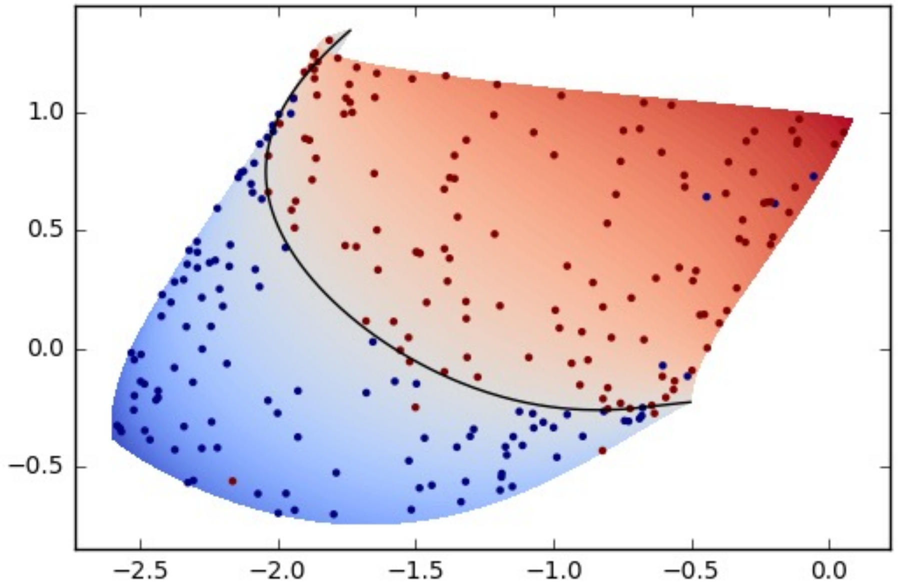
Lecture 20: Ensemble Methods



Gavin Kerrigan

Spring 2023

Some slides adapted from Padhraic Smyth, Alex Ihler

Announcements

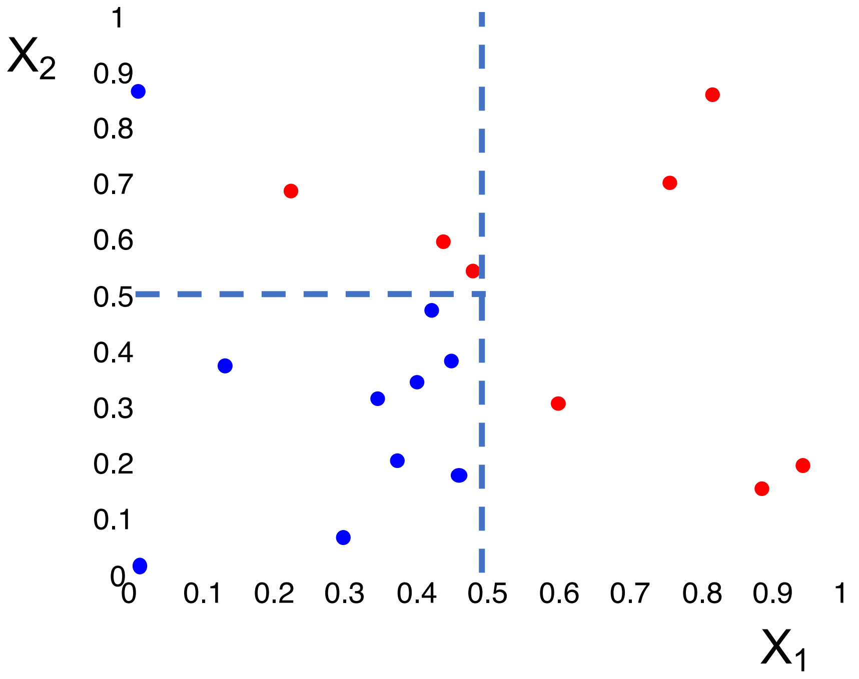
- Discussion sections tomorrow:
  - Tutorial on PyTorch
  - ~~Facebook’s~~ Meta’s Python library for deep learning
  - Recommend installing before discussion
  - https://pytorch.org/get-started/locally/

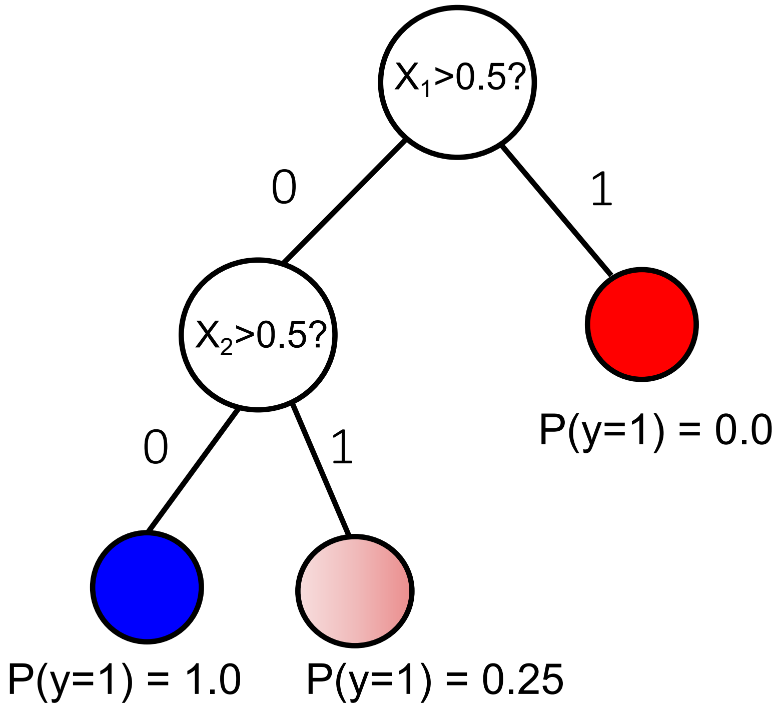
Wrapup on Decision Trees

Combining Classifiers

Bagging and Random Forests

Decision Tree Model in 2 Dimensions





Algorithm BuildTree: Greedy training of a decision tree classifier

Input: Labeled dataset $D = \{ (x_i, y_i) \}, i = 1,.. n$
Output: A decision tree with parameters θ

```
Compute P = ClassProbabilityVector(D)

if LeafCondition(D, P, node) then
    node = leaf node

else
    $t_j$ = FindBestSplit(D)
    $D_L = \{ (x_i, y_i) : x_{ij} \leq t \}$
    $D_R = \{ (x_i, y_i) : x_{ij} > t \}$
    Set left and right children to trees from BuildTree($D_L$) and BuildTree($D_R$)

end if
```

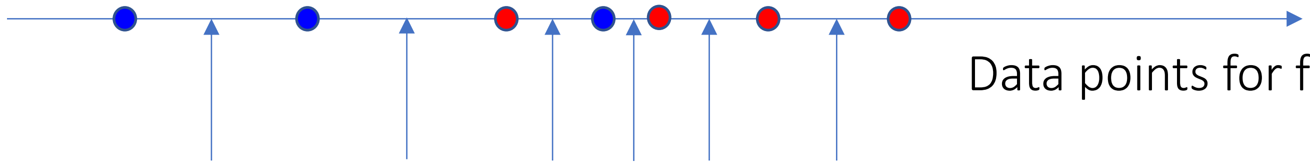
FindBestSplit(D)

Given labeled data D = $\{(\mathbf{x}_i, y_i)\}$ at some node find the best split (e.g., the root node with n datapoints – but same idea for any node)

```
For each feature $x_j$, $j = 1, \ldots d$
    Sort the n values for that feature

    For each possible threshold t (n-1 of them)
        Update the class probabilities
        Compute Gini(t) given class probabilities
    end

    Select $t_j$ with $G_j = \min \{ \text{Gini}(t) \}$ for feature $x_j$

Select threshold $t_j$ and feature $x_j$ with minimum $G_j$ over features
```

Possible Threshold Values

Two classes, red and blue



Data points for feature x

Note that the Gini index does not change between data points since it only depends on numbers of each label to the left and right of it

So we only need to evaluate thresholds between data points:
e.g., can pick thresholds halfway between each pair of data points

Time Complexity of FindBestSplit(D)

Given labeled data D = $\{(\mathbf{x}_i, y_i)\}$ at some node find the best split (e.g., the root node with n datapoints – but same idea for any node)

Assume n and d >> K (number of classes)

```
For each feature $x_j$, j = 1, … d                                          O(d)
        Sort the n values for that feature                                  O(n logn)

        For each possible threshold t (n-1 of them)                         O(n)
                Update the class probabilities
                Compute Gini(t) given class probabilities
        end

        Select $t_j$ with $G_j$ = min { Gini(t) } for feature $x_j$         O(n)

Select threshold $t_*$ and feature $x_*$ with minimum $G_j$ over features   O(d)
```

Overall Time Complexity

Complexity at root node (from previous slide)
O(d (n log(n) + n) + d) = O(d n log(n))

Complexity of growing a balanced tree of depth L?

Worst case = O(L d n log(n))
(if there are nodes at each level with O(n) datapoints that require resorting)

In practice, however, we can store the sorted results from the root node and use indices to do threshold search on subsets (internal nodes)

So the time complexity of training a decision tree classifier is usually reported as O(d n log(n))

Prediction (Test) Complexity of Decision Tree Classifiers

Assume: n examples, d real-valued features, binary splits, max depth L

Time complexity of prediction with 1 example?

= Depth of tree = O(L) (worst case)

Space complexity at prediction time?

= O(number of nodes)

where number of nodes = number of leaves + num of internal nodes

$= 2^L + (2^L - 1)$

$\Rightarrow O(2^{L+1})$

(this is for a tree with all paths of length L)

Summary of Decision Tree Classifiers

- Decision tree classifiers
  - Flexible functional form (but linear/axis-parallel)
  - At each internal node, split on one variable
  - At leaves, produces vector of class probabilities

- Learning decision trees
  - Score all splits & pick best
    - Criterion used for splitting? Gini index
  - Stopping criteria

- Complexity depends on number of nodes/depth

Questions?

Wrapup on Decision Trees

Combining Classifiers

Bagging and Random Forests

Classifier Notation

Assume binary classification (2 classes) for simplicity of notation
(methods can be generalized to K > 2 classes)

We will use $f(\mathbf{x})$ to refer to some pretrained binary classifier, where $f(\mathbf{x})$ is the output of the classifier, given input features $\mathbf{x}$

The model has been trained so it has some fixed parameter values θ
We will use $f(\mathbf{x})$ in this section instead of $f(\mathbf{x} \mid \theta)$, for simplicity of notation

We interpret $f(\mathbf{x})$ as $P(y = 1|\mathbf{x})$, assuming $f(\mathbf{x})$ is in the range $[0, 1]$

$f(\mathbf{x})$ could be any such classifier

- logistic model
- neural network
- decision tree
- kNN (where f(x) = fraction of k nearest neighbors that belong to class 1)

Classifier Notation

$\hat{y}(\mathbf{x})$ is the class label predicted by the classifier given features $\mathbf{x}$

For binary classification (2 class labels):

- if $f(\mathbf{x}) > 0.5$ then $\hat{y}(\mathbf{x}) = 1$
- if $f(\mathbf{x}) < 0.5$ then $\hat{y}(\mathbf{x}) = 0$

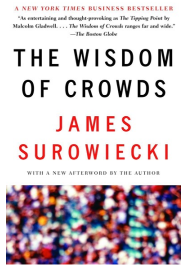
A classifier in general can produce both:

1. a "soft" probability prediction $f(\mathbf{x})$
2. a "hard" label prediction $\hat{y}(\mathbf{x})$

More generally for $K \geq 2$ classes: $\hat{y}(\mathbf{x}) \;=\; \arg\max_k \{f_1(\mathbf{x}), \ldots, f_K(\mathbf{x})\}$

Ensemble Methods: Combining Classifiers

Results from cognitive science ("wisdom of crowds"):
combinations of human predictions: better than any single human


A NEW YORK TIMES BUSINESS BESTSELLER
THE WISDOM OF CROWDS
JAMES SUROWIECKI
WITH A NEW AFTERWORD BY THE AUTHOR

Example: Galton's Ox (1906)
-- At a county fair: guess the weight of an Ox to win a prize
-- Average of predictions from crowd much more accurate than any individual
(within 1lb of true weight!)

So, as with human predictions it can be useful to combine classifiers

Intuition:
if classifiers have different "expertise" it may be worth combining them
(e.g., if they make different errors)

Terminology:
Ensemble methods = algorithms/methods for combining classifiers
Ensembles = sets of combined classifiers

Notation for Multiple Classifiers

Lets say we have M pre-trained classifiers

Class probabilities (for class 1) from each classifier:

$$f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}), \ldots, f_M(\mathbf{x})$$

"Hard" label predictions from each classifier:

$$\hat{y}_1(\mathbf{x}), \ldots, \hat{y}_m(\mathbf{x}), \ldots, \hat{y}_M(\mathbf{x})$$

(subscript indicates the mth classifier, $m = 1, \ldots, M$)

The M classifiers can be any type of classifier, e.g.,
e.g., M=8 with 3 neural networks, 4 decision trees, a logistic classifier, etc

Simple Ensemble Methods: Voting

Say we have M pretrained models (e.g., all trained on same training dataset D)

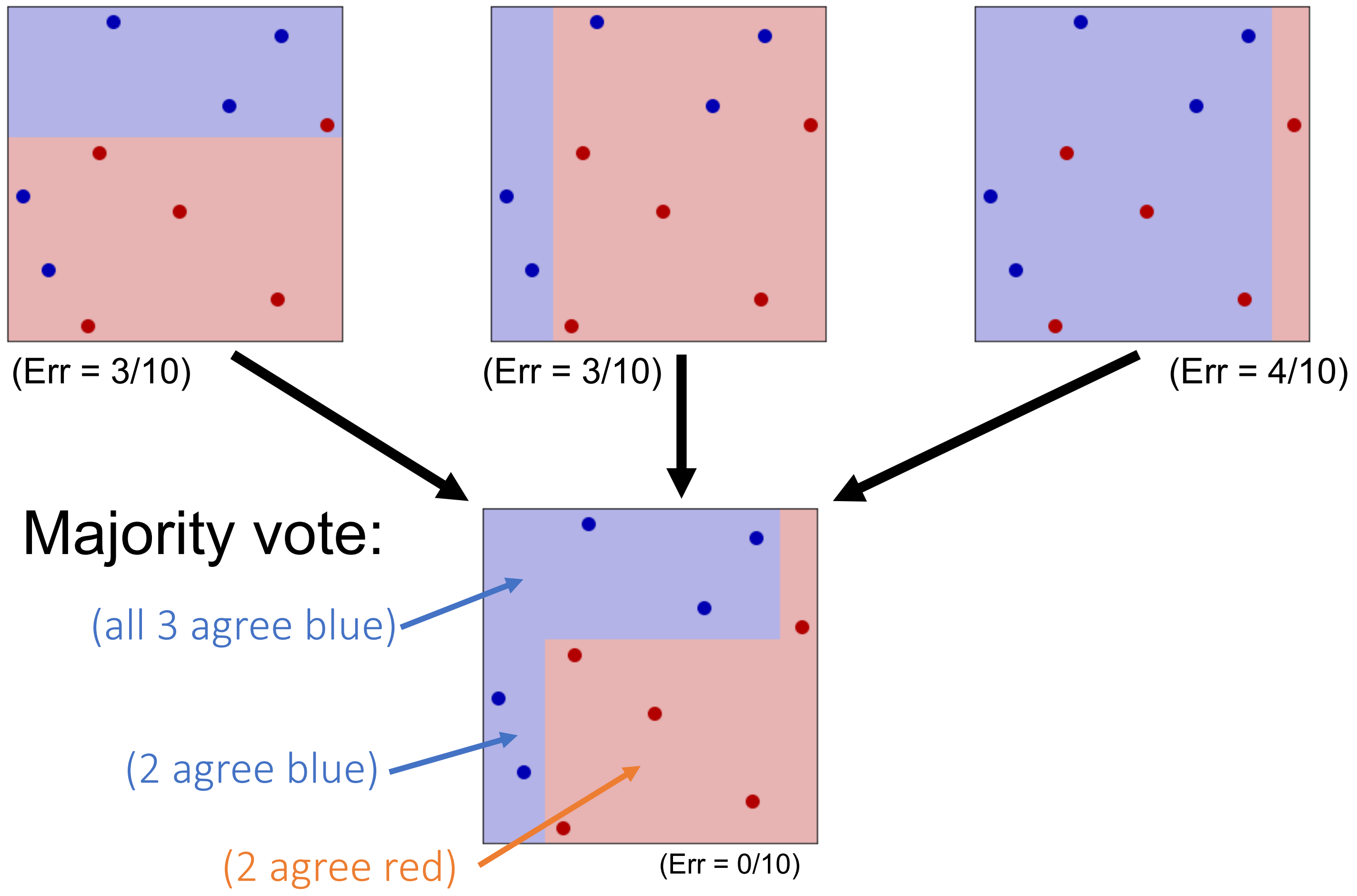
For a new test datapoint x, at prediction time:

Compute: $\hat{y}_1(\mathbf{x}), \ldots, \hat{y}_m(\mathbf{x}), \ldots, \hat{y}_M(\mathbf{x})$

i.e., pass **x** through each classifier and predict a label with each

Now let the combined prediction = the majority vote of the classifiers
(best to have M be an odd number for K=2 classes)

Simple Ensemble Methods: Voting



Simple Ensemble Methods: Averaging

Alternative to a simple majority vote: *average* the predicted probabilities

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{M}\sum_{m=1}^{M} f_m(\mathbf{x}) \\ &= \frac{1}{M}\sum_{m=1}^{M} P_m(y=1|\mathbf{x}) \end{aligned}$$

Because we are taking an average of numbers between 0 and 1, the average, *f*(**x**), will lie between 0 and 1

=> we can interpret *f*(**x**) as a probability P(y=1|x)
and threshold at 0.5 to make a label prediction

Ensemble Methods: Weighted Averaging

$$\begin{aligned} f(\mathbf{x}) &= \sum_{m=1}^{M} w_m f_m(\mathbf{x}) \\ &= \sum_{m=1}^{M} w_m P_m(y = 1|\mathbf{x}) \end{aligned}$$

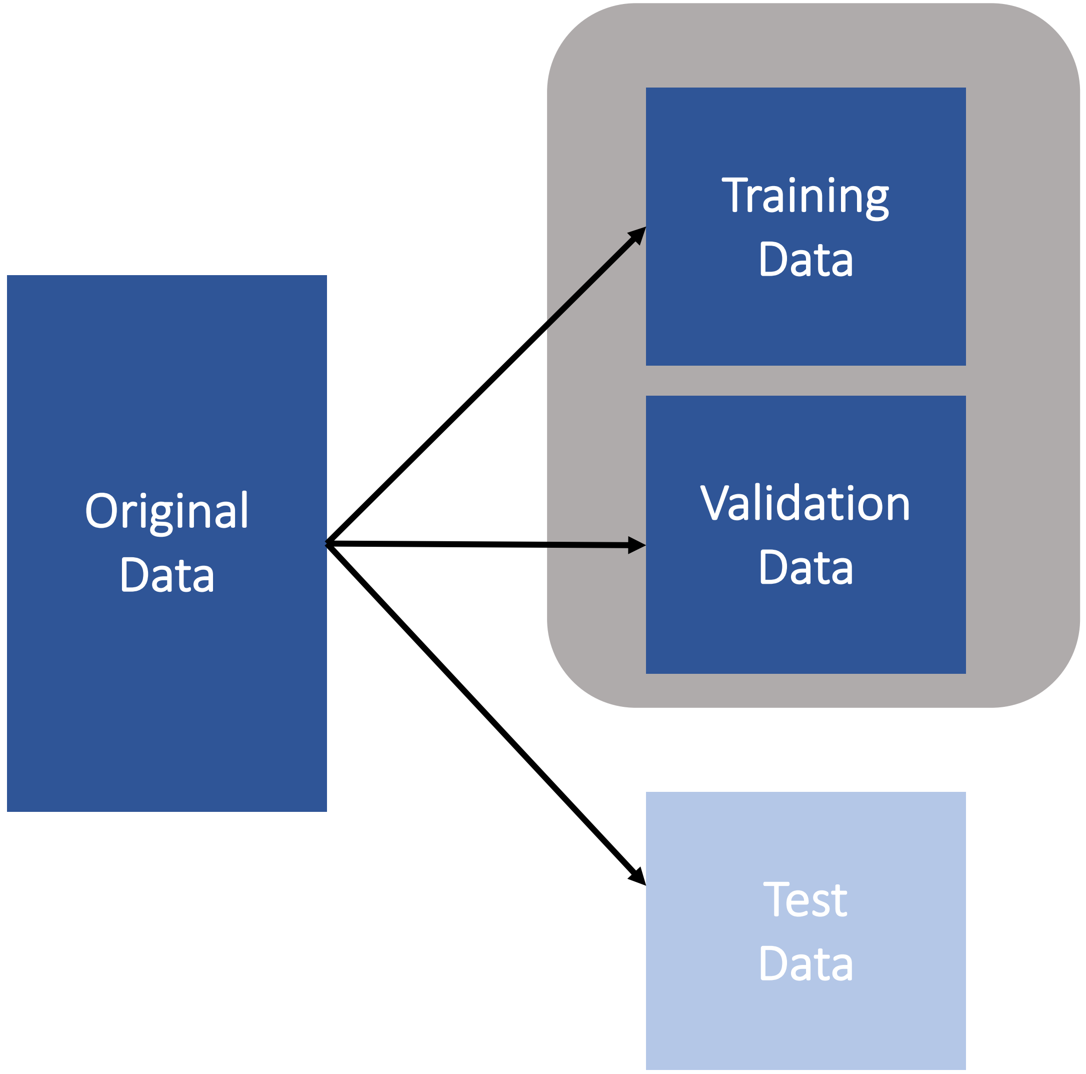
where weights w_m are between 0 and 1, and $\sum_m w_m = 1$

Because the weights are constrained to be non-negative and sum to 1, the weighted sum, *f*(**x**), will be between 0 and 1
……so we can again interpret *f*(**x**) as P(y=1|x)

Note: unweighted averaging is a special case with weights = 1/M

How should we learn the weights?

Stacking Method for Combining Models



Data used for final model evaluation

1. Train the M models on the training data

2. Learn the M weights on the validation data (e.g., using a logistic model + cross entropy)

Why not just learn the weights on the training data?

Validation Data

Feature vectors	Labels
$\mathbf{x}_1$	y_1
$\mathbf{x}_2$	y_2
…	…
…	…
…	…
$\mathbf{x}_n$	y_n

Predictions of M pretrained models on Validation Data

Predictions of Model 1	Predictions of Model 2	…	Predictions of Model M	Labels
$f_1(\mathbf{x}_1)$	$f_2(\mathbf{x}_1)$	…	$f_2(\mathbf{x}_1)$	y_1
$f_1(\mathbf{x}_2)$	$f_2(\mathbf{x}_2)$	…	$f_2(\mathbf{x}_2)$	y_2
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
$f_1(\mathbf{x}_n)$	$f_2(\mathbf{x}_n)$	…	$f_M(\mathbf{x}_n)$	y_n

Stacking uses the dataset above to learn the model below

$$f(\mathbf{x}) = \sum_{m=1}^{M} w_m f_m(\mathbf{x})$$

Weights are fit using validation data

Models pretrained on training data

Netflix Prize Competition

Netflix Competition: 2006-2009
$1 million prize

Welcome!

The Netflix Prize seeks to substantially improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences. Improve it enough and you win one (or more) Prizes. Winning the Netflix Prize improves our ability to connect people to the movies they love.

Read the Rules to see what is required to win the Prizes. If you are interested in joining the quest, you should register a team.

You should also read the frequently-asked questions about the Prize. And check out how various teams are doing on the Leaderboard.

Good luck and thanks for helping!

© 1997-2006 Netflix, Inc. All rights reserved.

Awarding of $1 million in 2009
…winning entry based on ensembles with stacking

NETFLIX
2009
DATE: 09·21·09
PAY TO THE ORDER OF: BellKor's Pragmatic Chaos
$ 1,000,000 00
AMOUNT: ONE MILLION 00/100
FOR: The Netflix Prize

Summary of Ensemble (Combining) Methods

M different classifiers (e.g., all trained on same data)

Combining Methods so far…

- Voting: take the majority vote of the classifiers
  - Simple, easy to understand
- Averaging: take the average of the M class probabilities
  - Can take model confidence into account
- Weighted averaging (Stacking)
  - Learn weights for the models on a validation dataset
  - Can upweight the more accurate models, downweight others

Questions?

Wrapup on Decision Trees

Combining Classifiers

Bagging and Random Forests

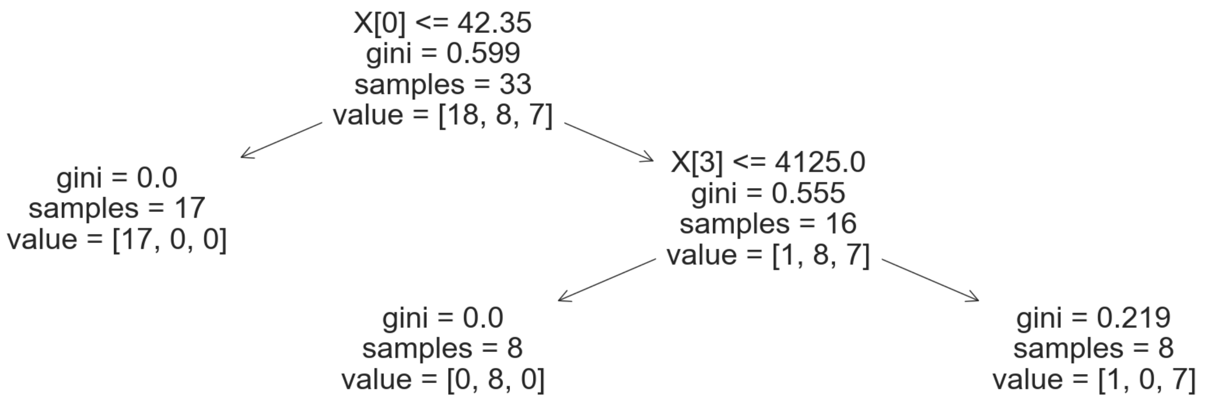
Classifier Variance

Variance in building classifiers:
 -> how much a model will change given different datasets of size n

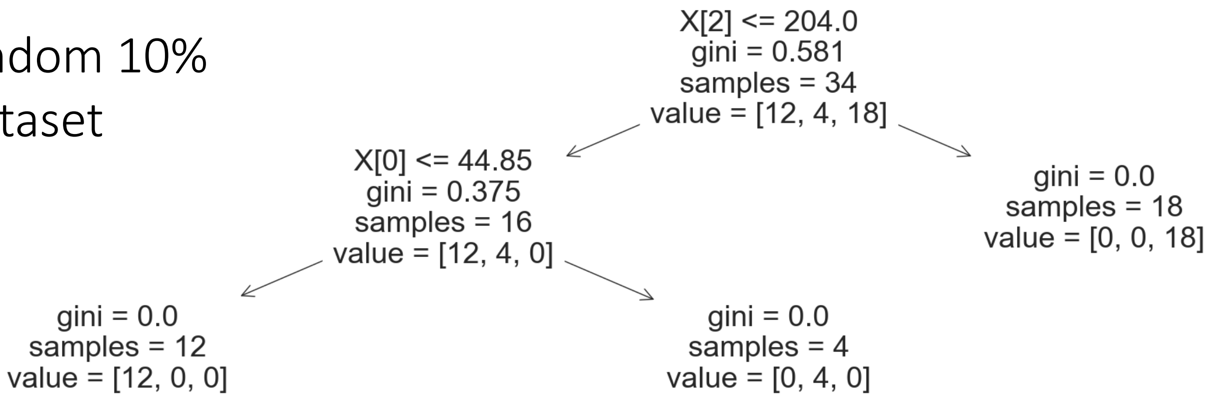
e.g., given a dataset D:
 -> split it randomly into equal-sizes subsets (e.g., in half)
 -> a model with high variance can have quite different structure or parameters, when fit on the 2 subsets

Variance of Decision Trees

Tree 1
trained on random 10% of penguin dataset



Tree 2
trained on random 10% of penguin dataset



Classifier Variance

Decision trees in particular tend to have high variance

Linear models (like logistic models) tend to have low variance

High variance contributes to test error
(recall bias-variance from earlier lectures)

.....so we would like to reduce variance if we can

One way to reduce variance is to average over different models, e.g., average over different trees

Averaging over Decision Trees

How can we build multiple different trees for the same dataset?

General idea:

- Given a training dataset of size n
- Randomly generate M random subsets of the data
- Build a tree on each of the M subsets
- Combine the trees (e.g., voting or averaging)

How should we create our random subsets in this process?

Bootstrap Samples

Given a dataset D of n training datapoints, consisting of pairs (x_i, y_i)

A bootstrap sample D_m, of size n, is generated as follows:

```
$D_m$ = empty set
For j = 1:n
        -> randomly pick a pair ($x_i$, $y_i$) from the training data
        -> add this pair to the bootstrap sample $D_m$
```

This produces a "bootstrap" dataset D_m, of size n

Note that some datapoints can appear multiple times in
i.e., bootstrap sampling is sampling with replacement

Example of Bootstrap Sampling

Original Data D

x1	x2	y
1.4	2.2	0
2.6	7.3	1
0.3	6.1	1
9.2	5.5	1
4.8	0.3	0

D_1 = Bootstrap Sample 1

x1	x2	y
1.4	2.2	0
1.4	2.2	0
9.2	5.5	1
9.2	5.5	1
4.8	0.3	0

D_2 = Bootstrap Sample 2

x1	x2	y
2.6	7.3	1
1.4	2.2	0
2.6	7.3	1
1.4	2.2	0
4.8	0.3	0

D_3 = Bootstrap Sample 3

x1	x2	y
4.8	0.3	0
0.3	6.1	1
0.3	6.1	1
9.2	5.5	1
0.3	6.1	1

D_4 = Bootstrap Sample 4

x1	x2	y
9.2	5.5	1
9.2	5.5	1
2.6	7.3	1
1.4	2.2	0
0.3	6.1	1

Sidenote: Generating Random Samples

How do we randomly sample examples between i=1 and i=n in our dataset?

Most programming languages produce pseudo-random numbers:
-> given a seed value they produce very long sequences of values
that don't have any discernible pattern (looks random)

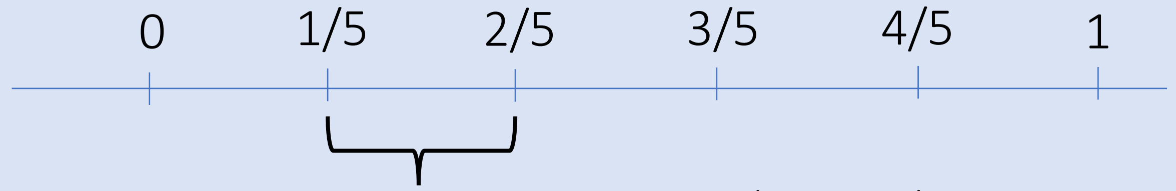
These are divided by their max value to produce sequences of numbers between 0 and 1 that look random ("uniformly distributed")

This is the basic rand() function supported in most programming languages

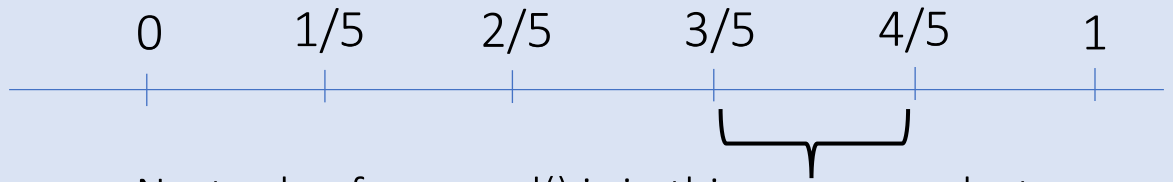
Sidenote: Generating Random Samples

Can we use rand() to randomly select from the training examples?

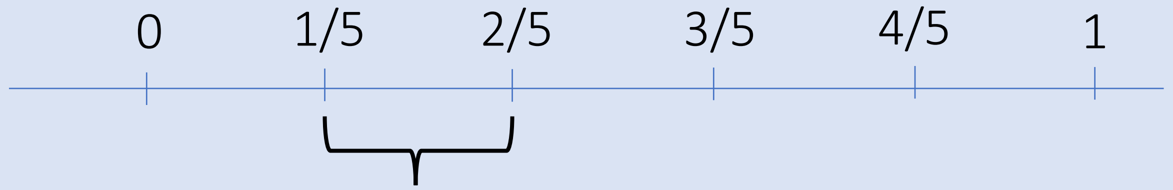
Example with n = 5



If rand() produces a number between 1/5 and 2/5, select example i=2



Next value from rand() is in this range -> select example i=4



Next value from rand() is again in this range -> select example i=2

Bagged Trees (“Bagging”)

Professor Leo Breiman (1928-2005)
UC Berkeley

Bagging (1994) = “Bootstrap Aggregation”

Training Procedure:

-> Generate M bootstrap samples D_1, ... D_M

-> Fit a Decision Tree to each, resulting in $f_1(\mathbf{x})$, ..., $f_M(\mathbf{x})$
(using standard decision tree learning)

Prediction Procedure, given a new **x**

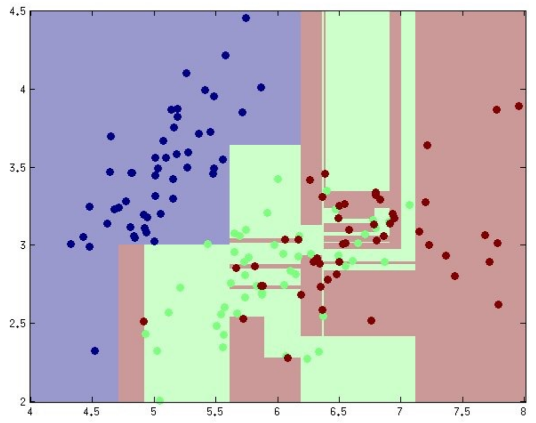
-> run **x** through each of the M trees

-> combine the M predictions to get an overall prediction $f(\mathbf{x})$
e.g., take the average (unweighted) of the class probabilities

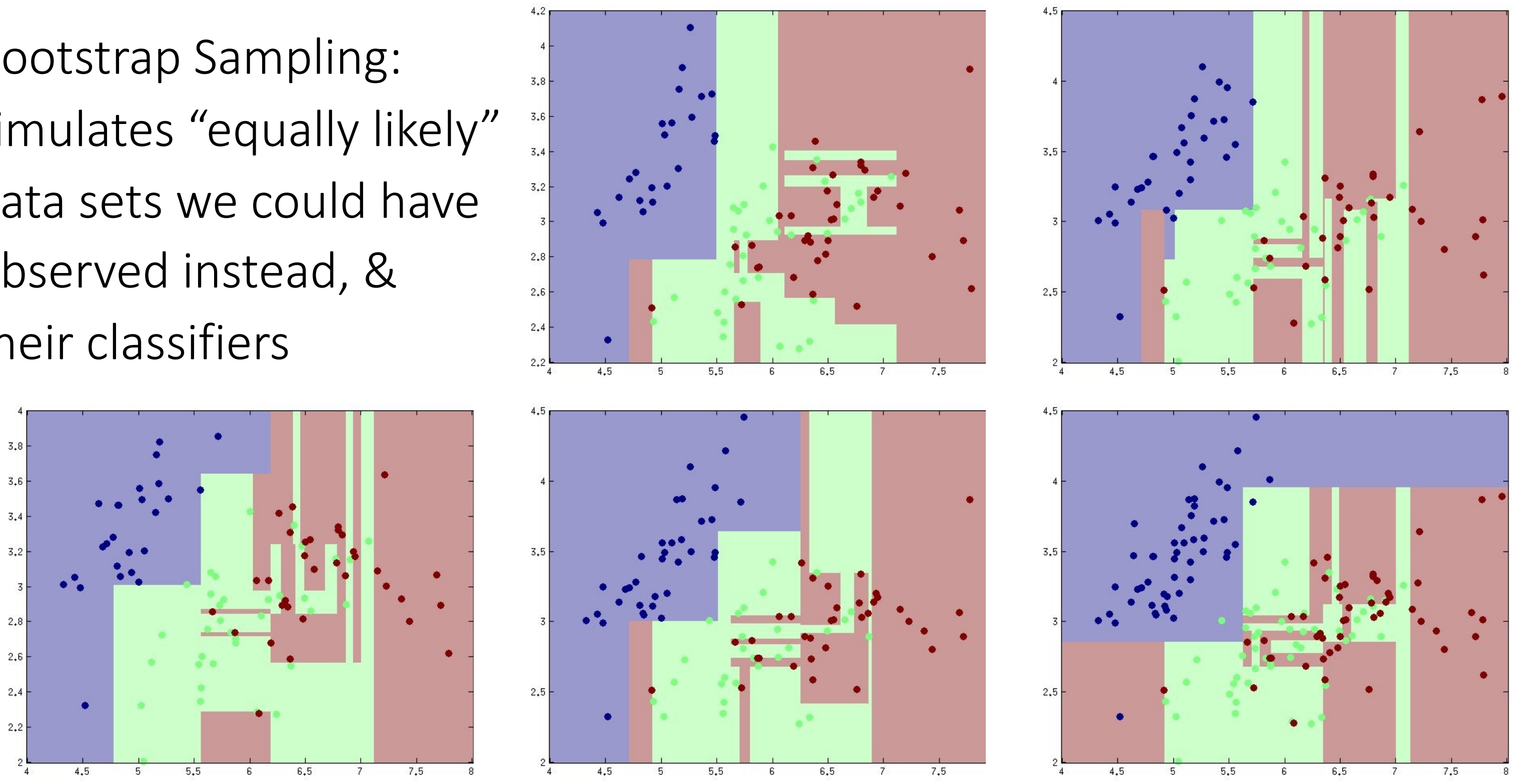
(Statistical theory tells us that this will reduce variance compared to a single tree and (in principle) be more accurate than a single tree)

Bagged decision trees

- Randomly resample data
- Learn a decision tree for each
  - No max depth = very flexible class of functions
  - Trees are low bias, but high variance

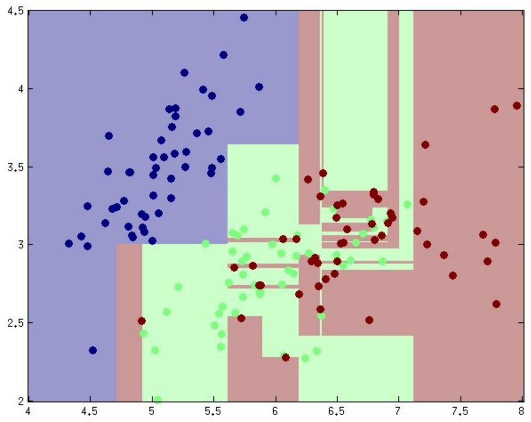

Single tree on full data set

Bootstrap Sampling: simulates "equally likely" data sets we could have observed instead, & their classifiers



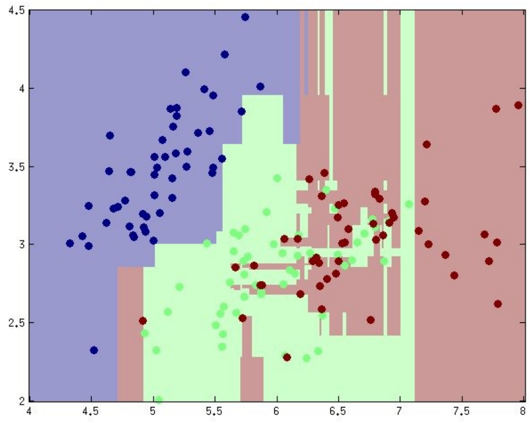
Bagged decision trees

- Average (no weighting) over trees
- Reduces memorization effect
  - Not every predictor sees each data point
  - Lowers effective “complexity” of the overall average
  - Usually, better generalization performance
  - Intuition: reduces variance while keeping bias low

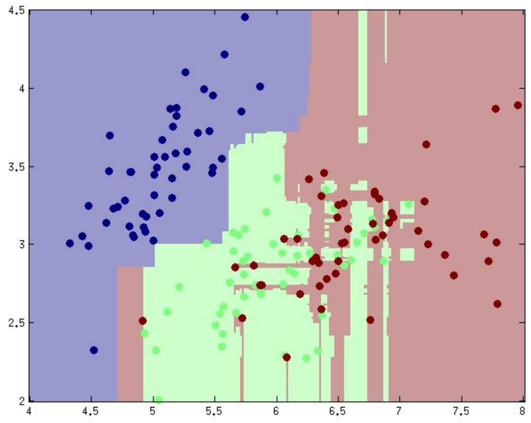


Single tree on full data set

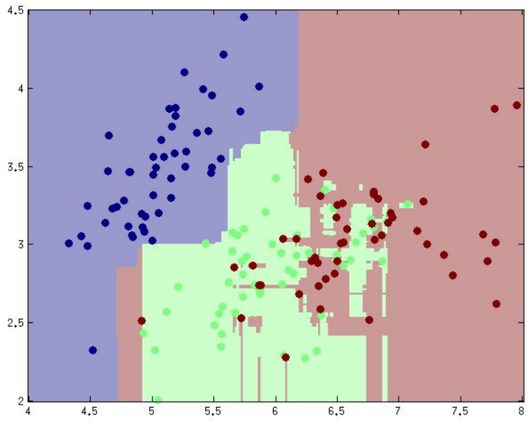
Avg of 5 trees



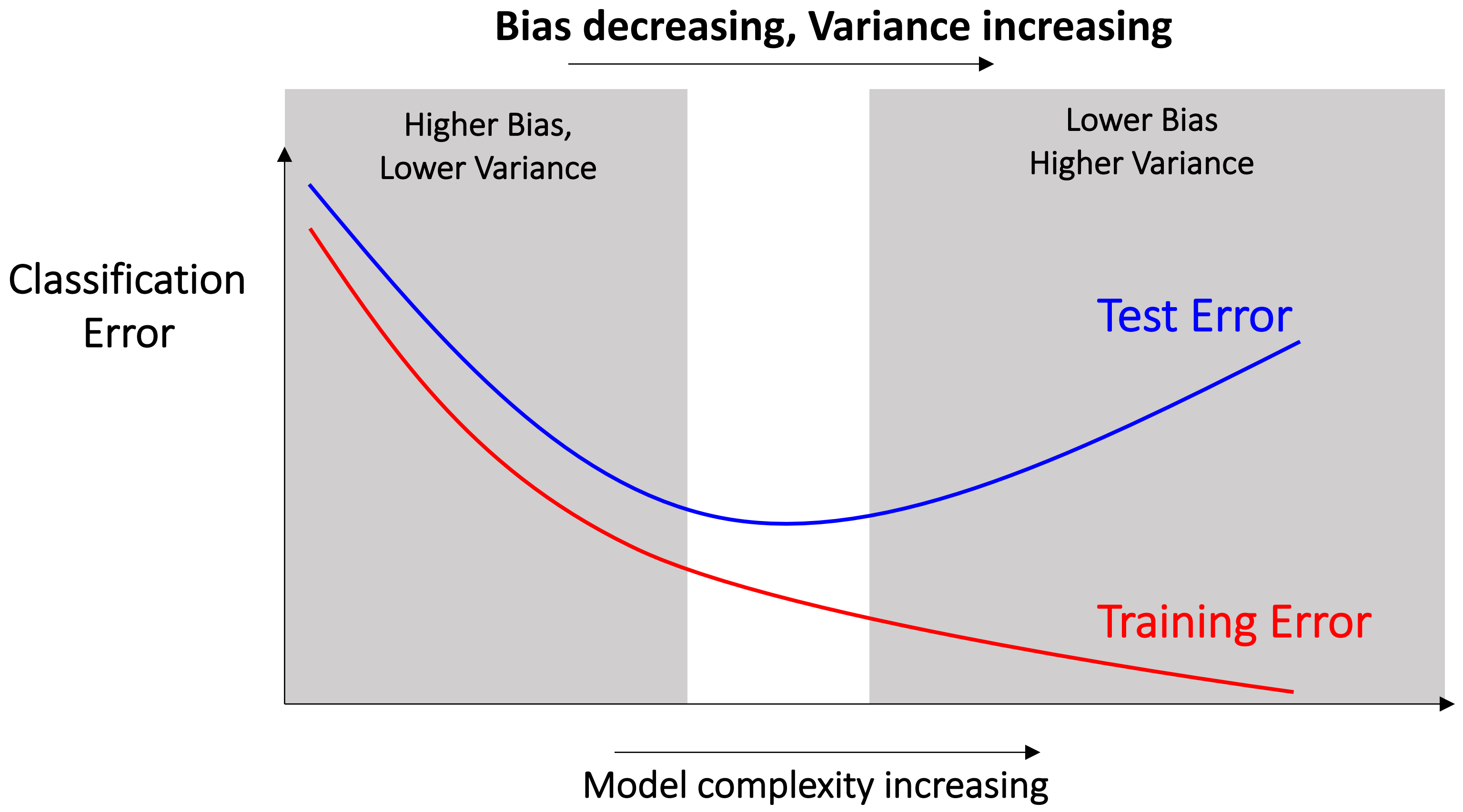
Avg of 25 trees



Avg of 100 trees



Recall the Bias-Variance Tradeoff



A Limitation….

- Problem with bagging applied to decision trees
  - With large datasets, we can often learn (roughly) the same tree
  - Averaging over these doesn't help much!

- Introduce extra variation in trees
  - At each step of training, only allow a (random) subset of features
  - Enforces diversity ("best" feature not available)
  - Keeps bias low (every feature available eventually)

Random Forests

Professor Leo Breiman
UC Berkeley

Random forests add another level of randomization to bagging for trees

Training Procedure:
-> Generate M bootstrap samples D_1, …. D_M
-> Fit a Decision Tree to each
… but when searching for the best split at any node
only search over R randomly selected features*
-> results in M trees, $f_1(x)$, ….., $f_M(x)$

Additional randomization step

Prediction Procedure, given new **x** (same as for bagging)
-> run **x** through each of the M trees
-> combine the M predictions to get an overall prediction f(**x**)
e.g., take the average (unweighted) of the class probabilities

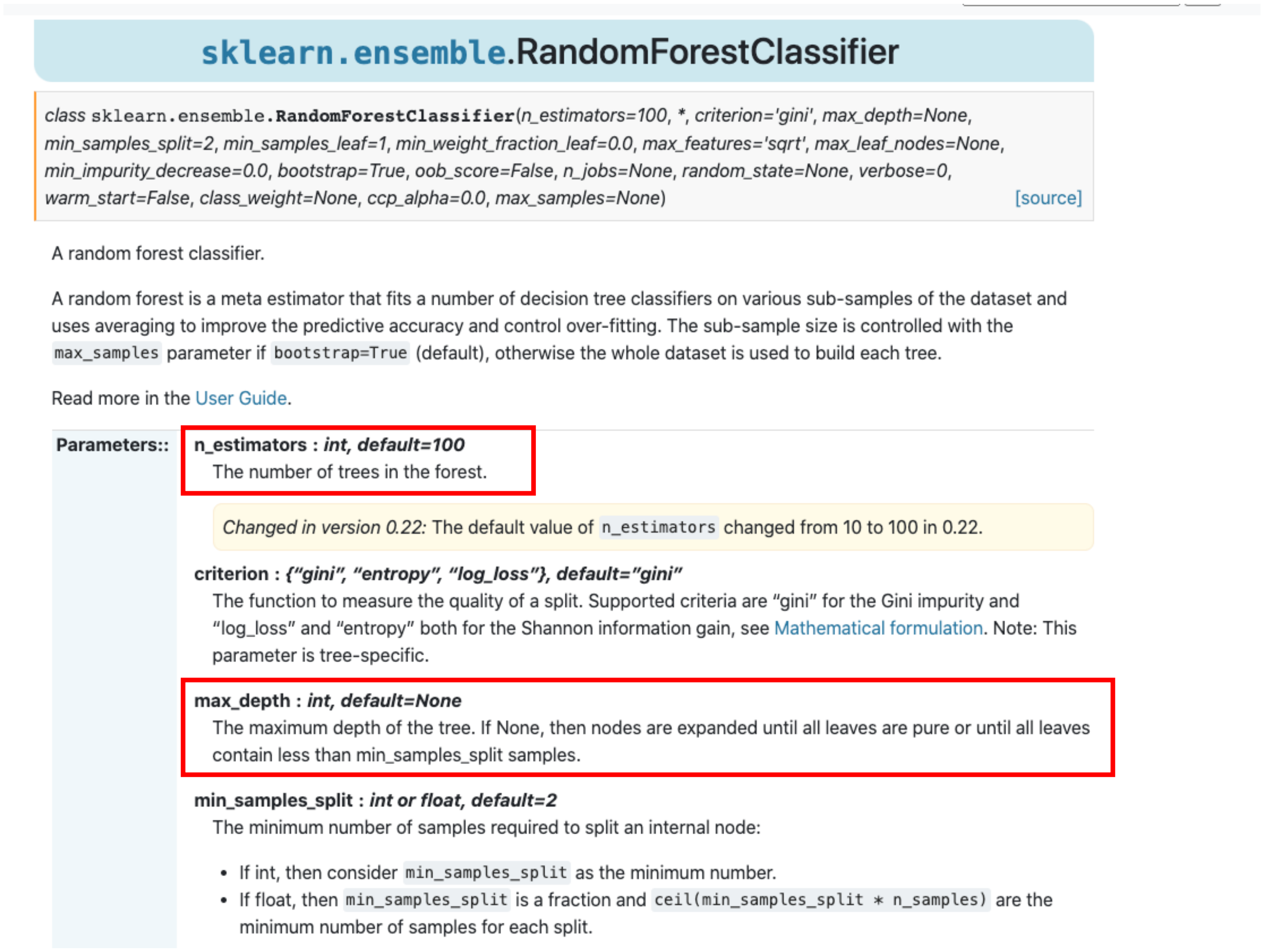
*To clarify: every time we search for a best split at any node in any tree, the algorithm randomly selects (at each node, in every tree) a random set of R < d features to search over, instead of searching over all d features.

Hyperparameters for Random Forests

Number of trees grown M = number of bootstrap samples
(default in scikit-learn: M = 100)

Number of features R randomly selected for search at each node
(default in scikit-learn: R = sqrt(d))

Max-depth of each tree (or other stopping criteria):
(default in scikit-learn is no max-depth,
keep splitting until nodes are all one class)



sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Read more in the User Guide.

Parameters::

n_estimators : *int, default=100*
The number of trees in the forest.

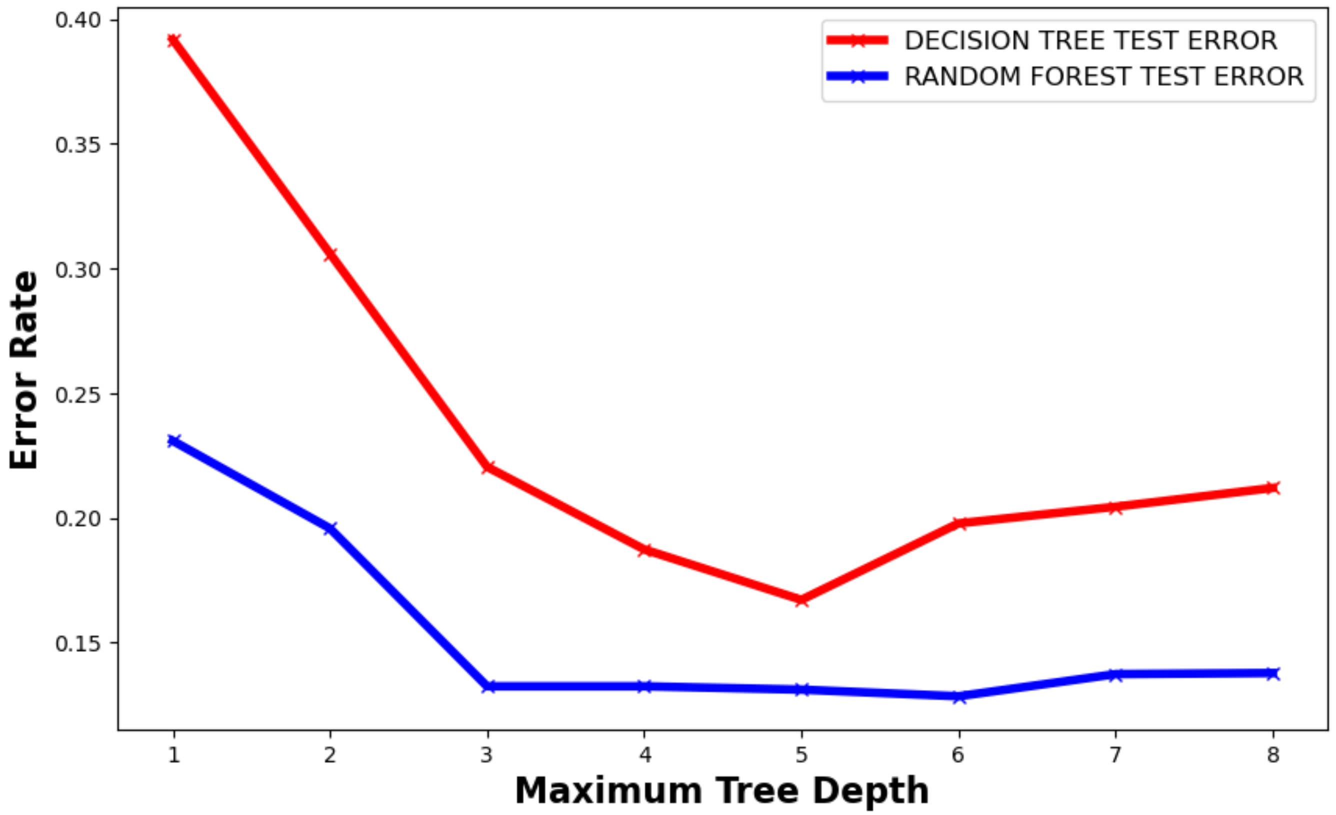
Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

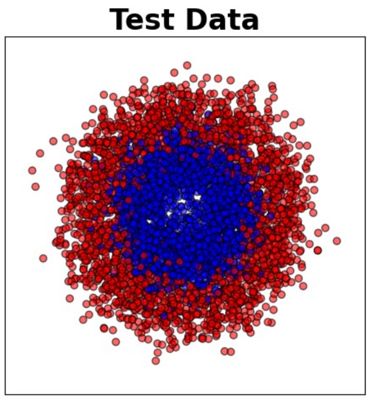
criterion : *{"gini", "entropy", "log_loss"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation. Note: This parameter is tree-specific.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

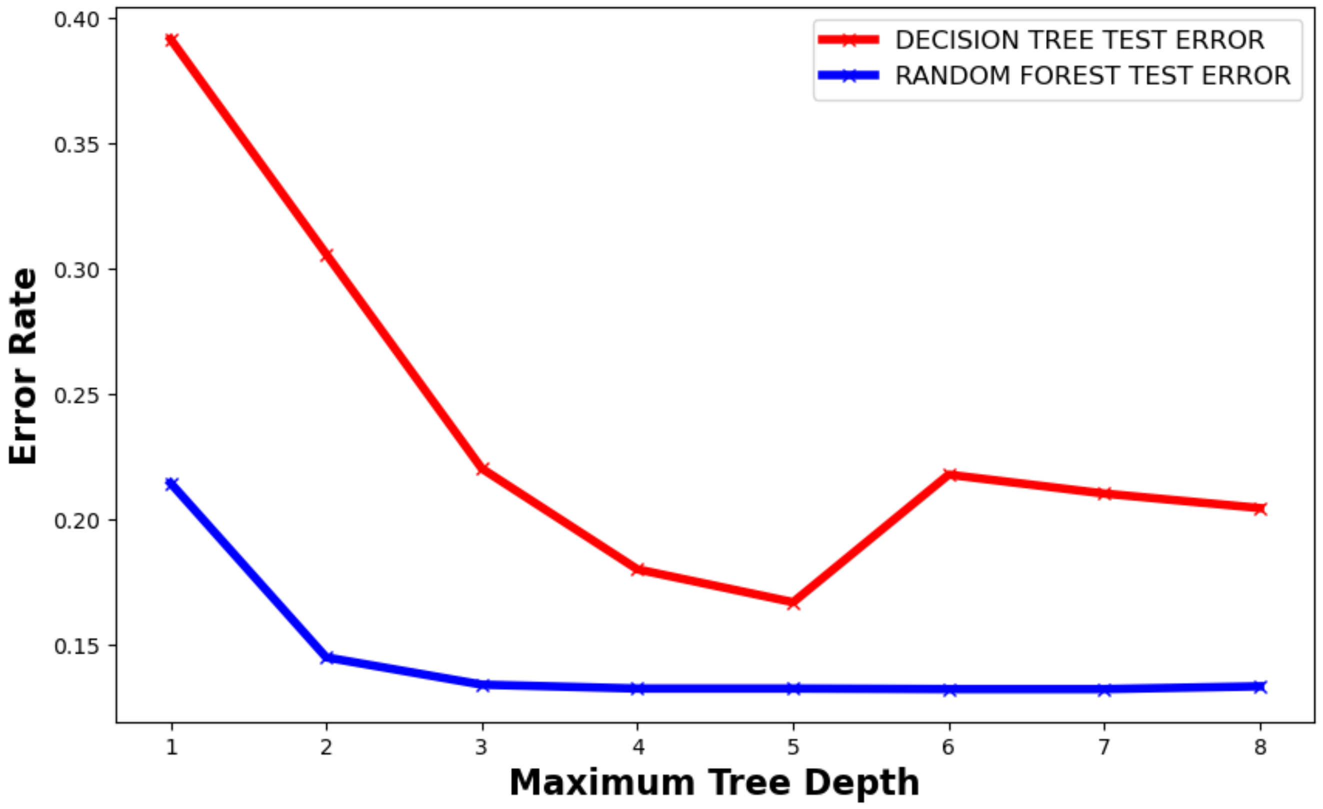


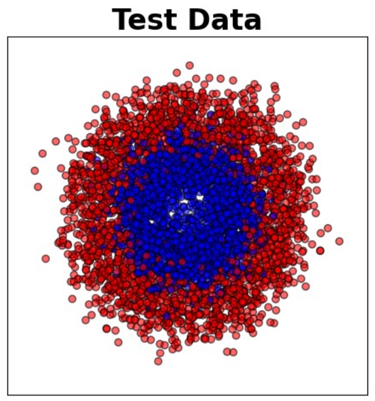


Models trained on 100 datapoints

X-axis = max tree depth for both decision trees and random forest

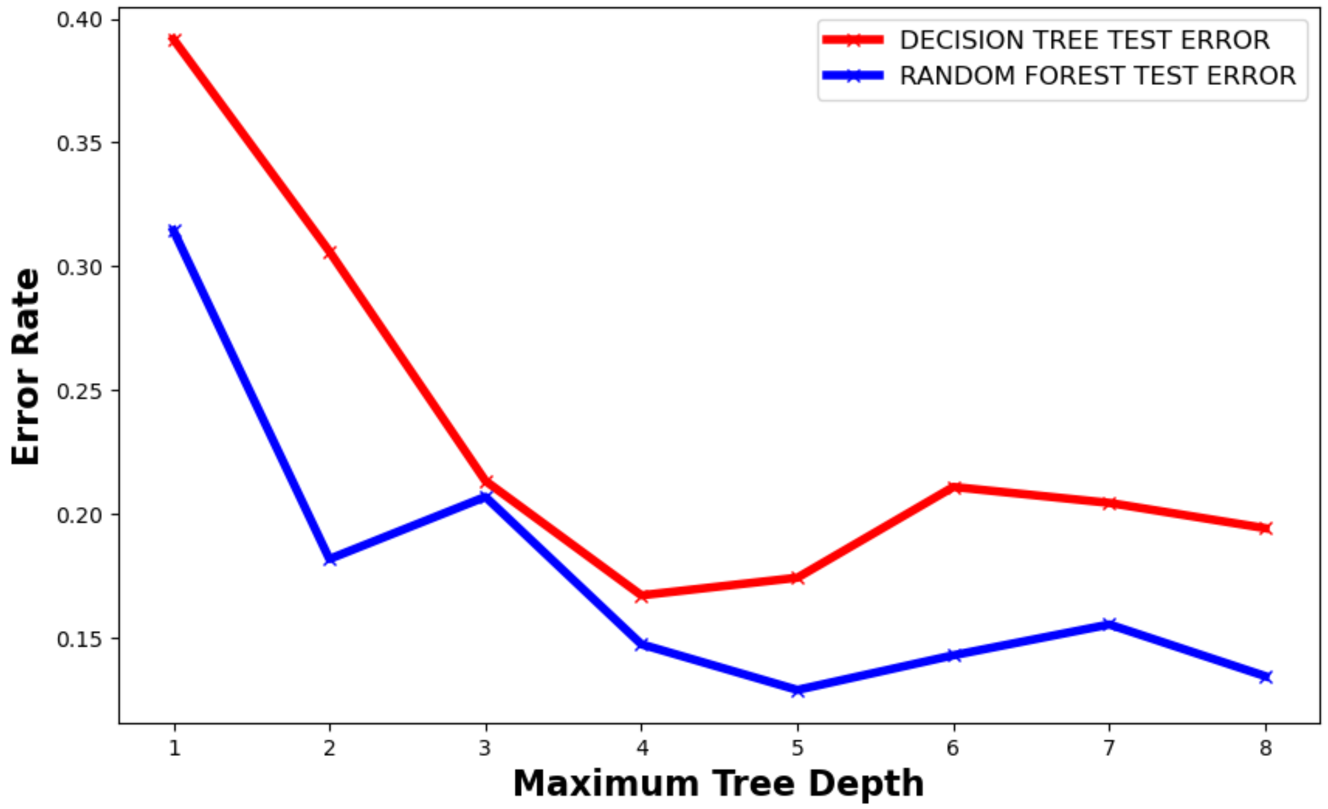
Number of trees in random forest = 100
Default settings in scikit-learn otherwise

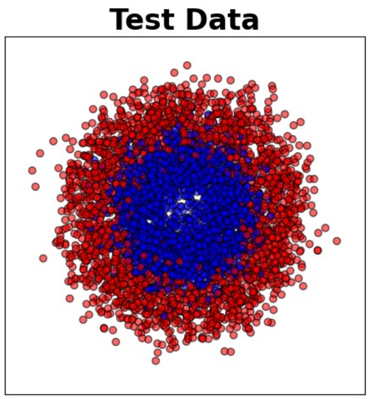




Same as previous slide but with
Number of trees in random forest = 1000

Heuristically:
-> random forests tend not to overfit
-> typically not much benefit seen beyond M = 100 trees





Same as previous slide but with
Number of trees in random forest = 20)

Summary of Random Forests

- Strengths
  - Typically more accurate than decision trees (reduced variance)
  - Can produce flexible boundaries
  - Scale-invariant (like decision trees)
  - Less likely to overfit compared to other models
  - Widely used in practice (especially for “tabular” data)

- Weaknesses
  - Hard to interpret/understand a “forest of trees”
  - Decision boundaries are still piecewise axis-parallel
  - Does not produce hidden representations
    - Less effective than neural networks for images, speech, text data

Questions?

Wrapup

- Decision Trees:
  - Training time complexity O(d n log(n))
  - Prediction: time complexity O(L), space complexity $O(2^{L+1})$
    - L is the maximum tree depth
- We can combine classifiers via…
  - Voting
  - Averaging
  - Weighted averaging (”stacking”)
- Bagging and Random Forests
  - Create bootstraps of your dataset and fit many models
  - Key Idea: want *different* models in an ensemble