Data Analytics (PETR 6397)

Support Vector Machines, Decision Trees, and Random Forests

Dr. Ahmad Sakhaee-Pour

Petroleum Engineering Department

University of Houston

Spring 2025

Discussed topics

- Support Vector Machines
- Decision Trees
- Ensemble Learning
- Random Forests

Support Vector Machine (SVM)

- SVM is a supervised algorithm for classification and regression
- Its objective is to maximize the margin between support vectors through a separating hyperplane
- The objective of SVM is NOT minimizing the cost function used in many other ML algorithms

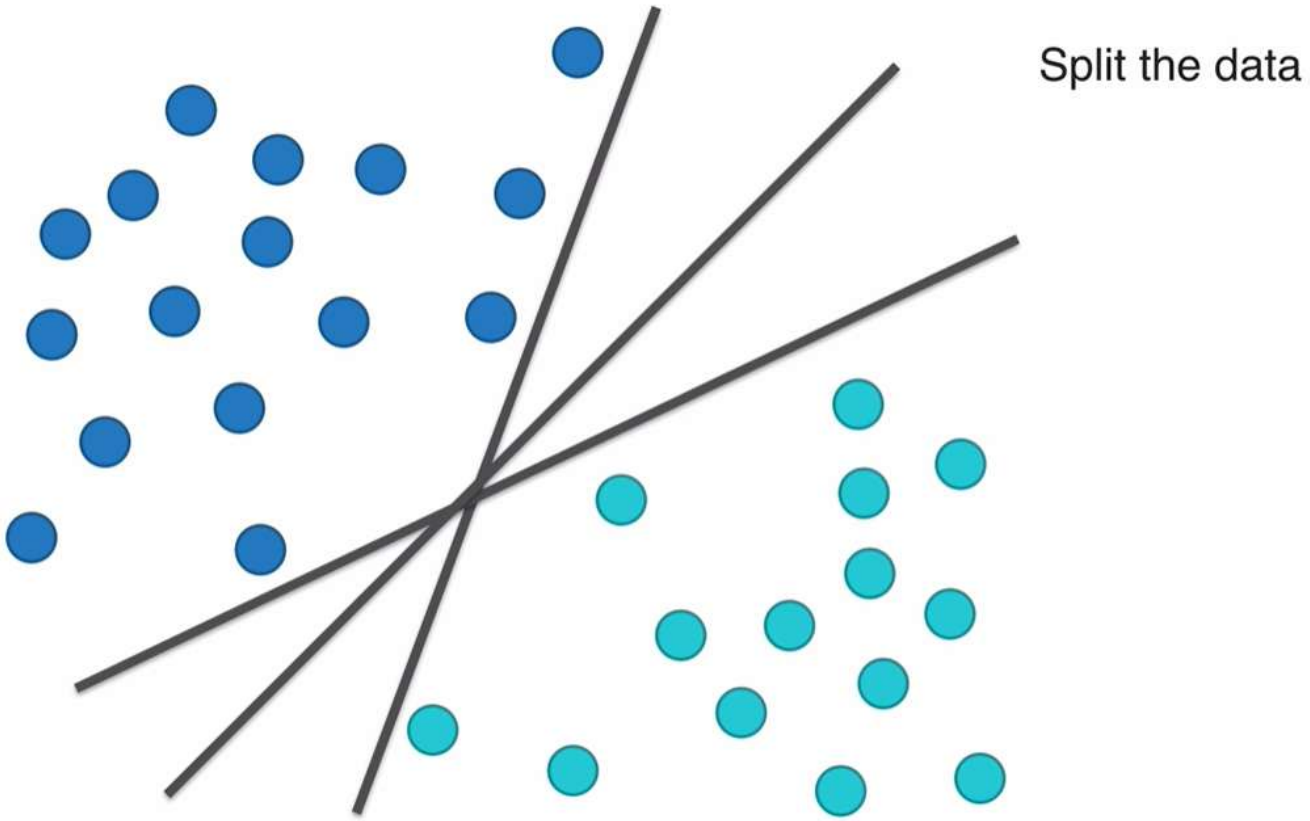
Fundamental properties of SVM

- It performs linear and non-linear classification and regression

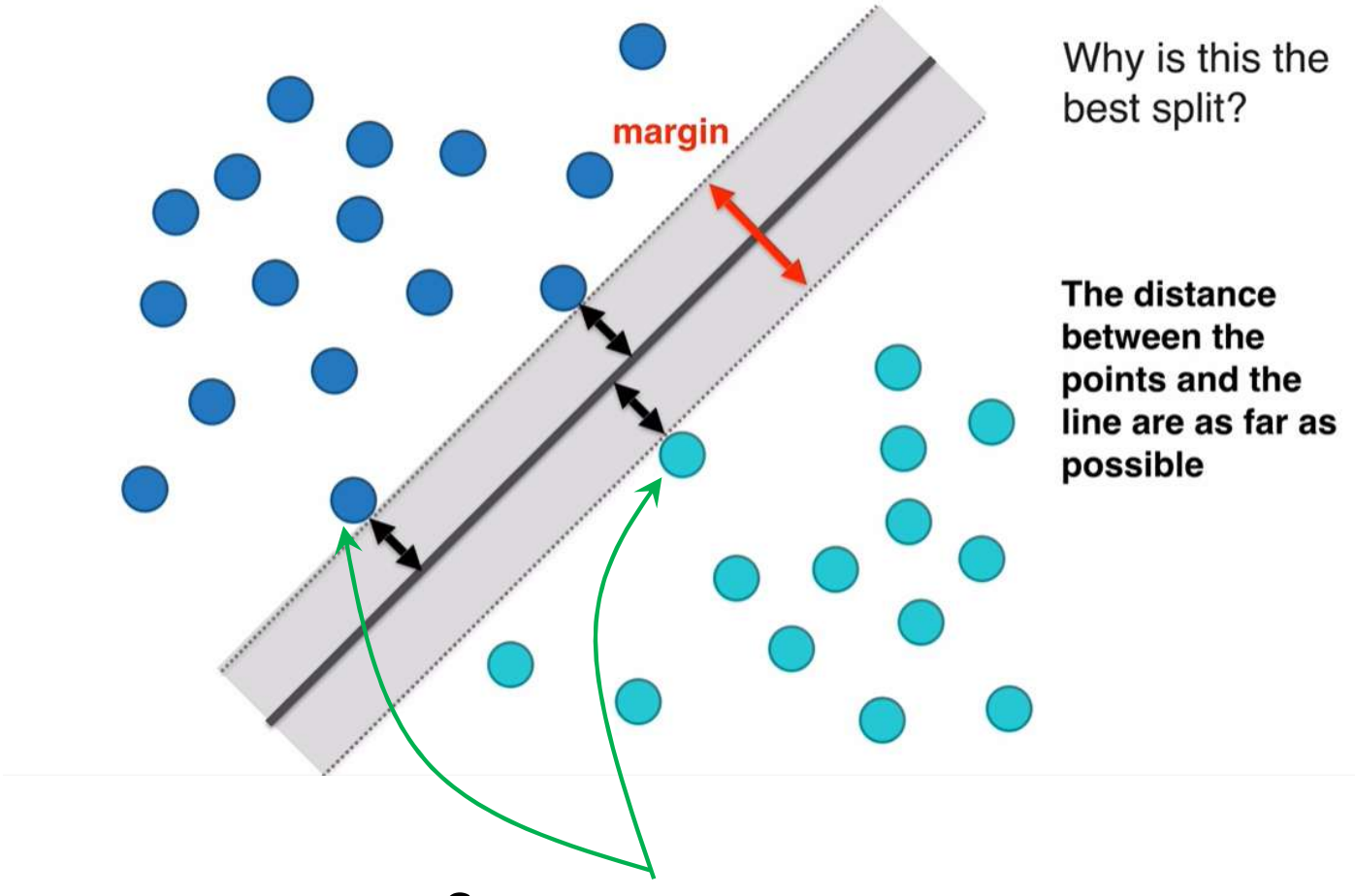
- SVM is called large margin classification because it fits the widest possible street between the classes

- They do not scale well (a major shortcoming when dealing with large datasets)

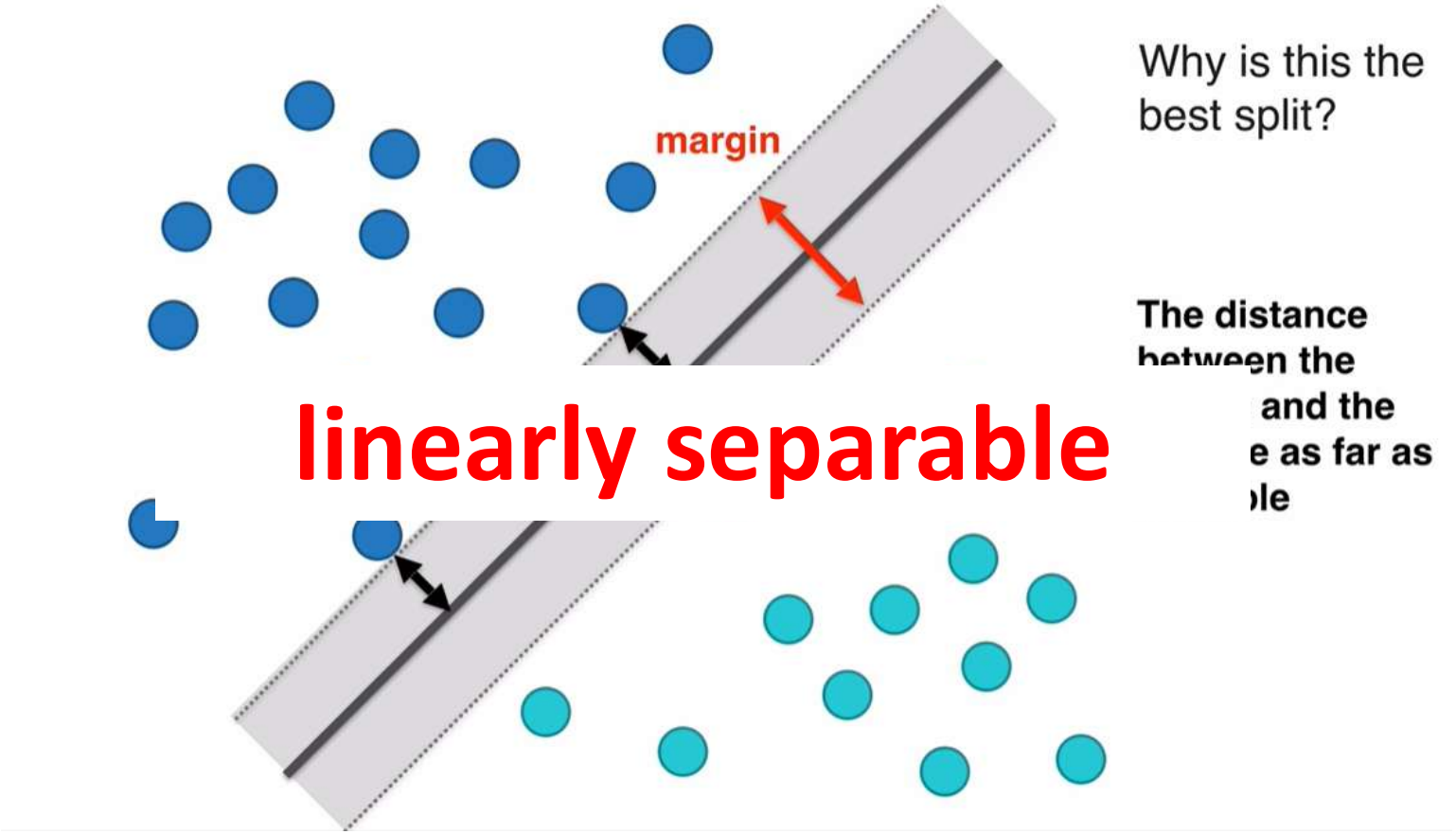
There are different ways to split (divide) the data



SVM goal is to maximize the margin between the support vectors



Original SVM works when data are linearly separable



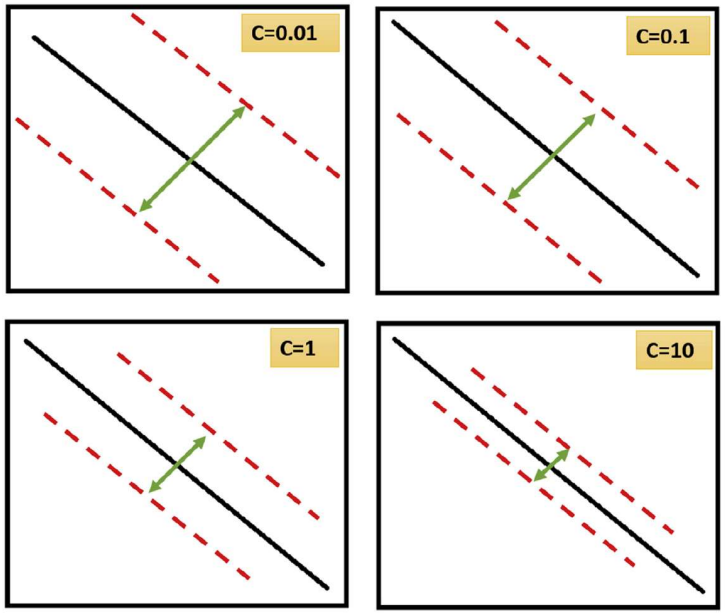
Hard margin versus soft margin

- Hard margin: all instances are off the street
- Hard margin works well if data are linearly separable
- Hard margin is sensitive to outliers (consider a scenario where one sample from one group is in the middle of another)

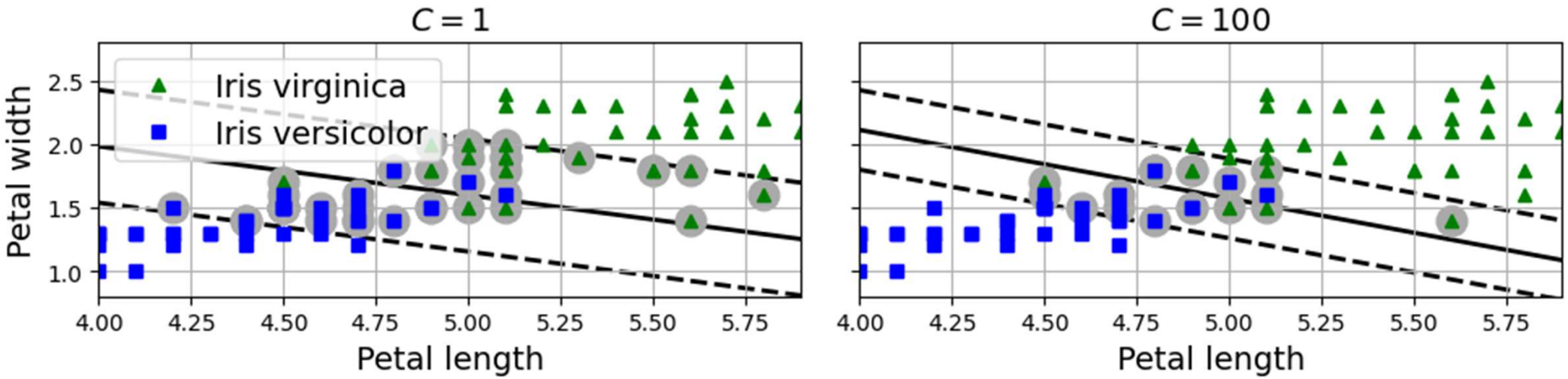
- Soft margin: we allow some samples to violate the margin but increase the street as wide as possible
- Soft margin addresses the hard margin shortcoming of dealing with non-linear data and outliers

Degree of tolerance of soft margin is captured by C (Cost)

- Smaller C:
  - Larger margin
  - Smaller penalty when misclassified
  - Lower training accuracy
  - More general
- Larger C:
  - Smaller margin
  - Larger penalty when misclassified
  - Higher training accuracy
  - Less general



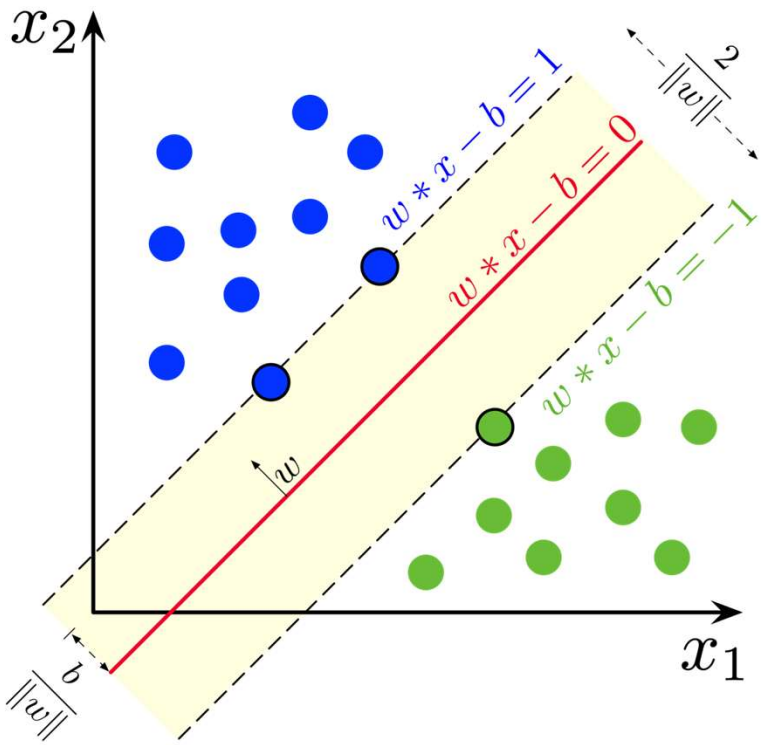
Which scenario is better?



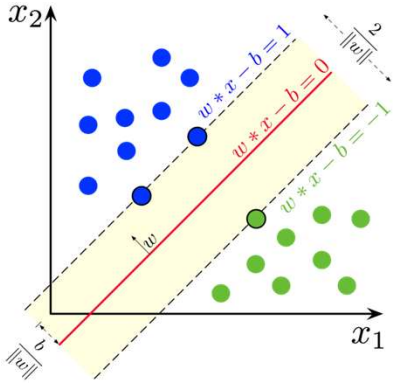
How does Linear SVM work?

- **w** is the vector normal to the plane
- **w**=(w1,w2,…,wn)
- b is the bias

- The goal is to maximize the margin ($2/\|\mathbf{w}\|$). So we minimize $\|\mathbf{w}\|$ based on hard margin and soft margin (next slide)



Objectives of linear SVM classifiers



- Hard margin:

  Minimize $\frac{1}{2}\ \mathbf{w}^{\mathrm{T}}\mathbf{w}$

  Subject to $t^{(i)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{w} + \mathbf{b}\right) \geq 1$ for $i = 1,2, \ldots, m$

- Soft margin:

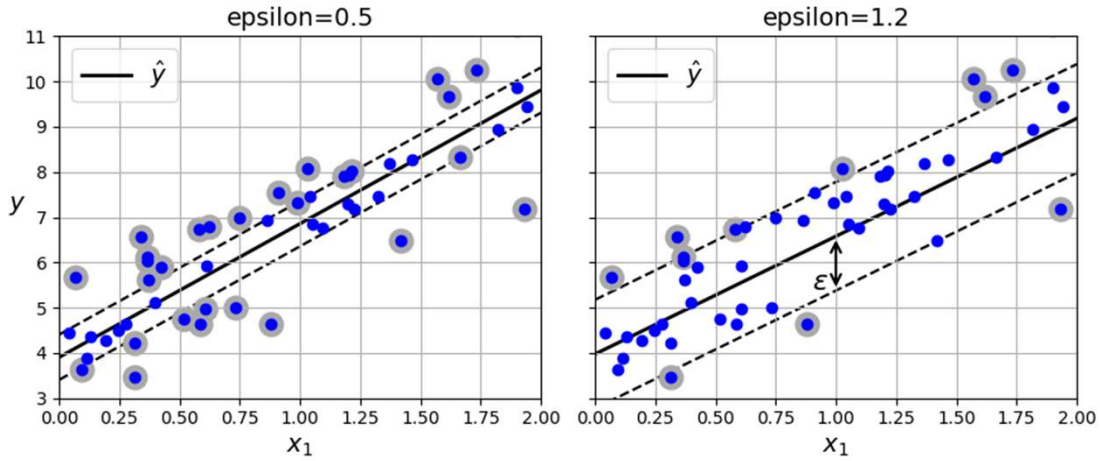
  Minimize $\frac{1}{2}\ \mathbf{w}^{\mathrm{T}}\mathbf{w} + C \sum_{i=1}^{m} \zeta^{i}$

  Subject to $t^{(i)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{w} + \mathbf{b}\right) \geq 1 - \zeta^{i}$ for $i = 1,2, \ldots, m$

- $t^{(i)}$ is +1 if the sample is above the plane
- $t^{(i)}$ is -1 if the sample is below the plane

SVM regression

- In regression, SVM fits as many instances as possible on the street while limiting margin violations
- The width is controlled by epsilon



- Reducing epsilon increases the number of support vectors
- Adding more training examples within the margin will not impact the prediction (model is not sensitive to epsilon)

Non-linear SVM

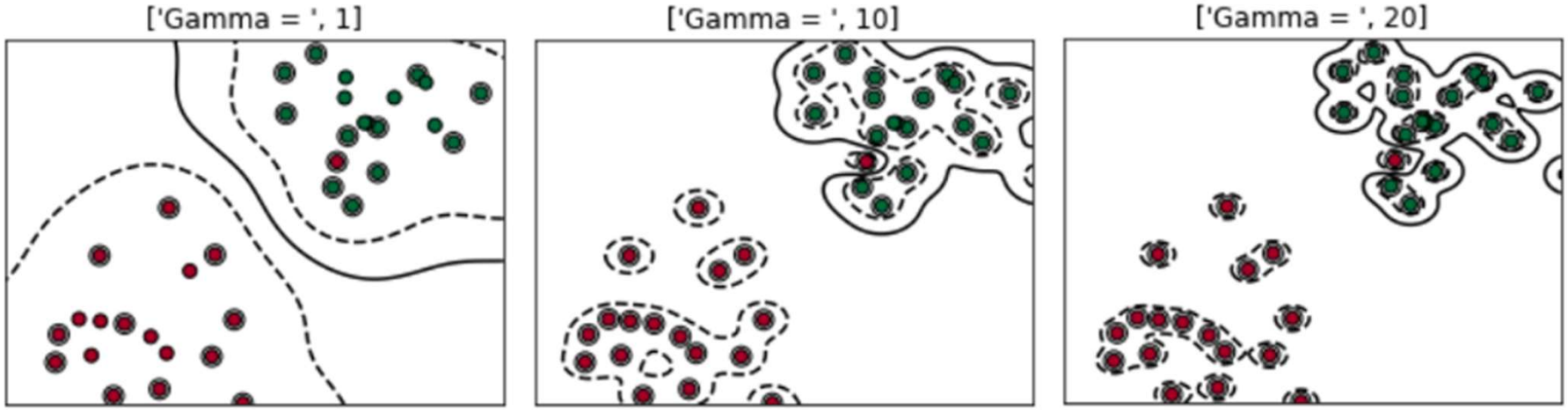
- We can add more features (such as polynomials) and use SVM for non-linear problems. We used this trick for regression in this course.

- We can also apply SVM using kernels (Polynomial kernel, Gaussian RBF kernel)

- Vapnik (and his colleagues) proposed linear SVM in 1963 and extended it to non-linear scenarios in 1992.

Non-linear SVM with Radial Basis Function (RBF) kernel

- Gamma: the inverse of the radius of influence of samples selected as support vectors. Low gamma means a large radius of influence.
  - Low gamma creates a simple model
  - High gamma creates a complex model that may overfit and be sensitive to noise



Decision Trees

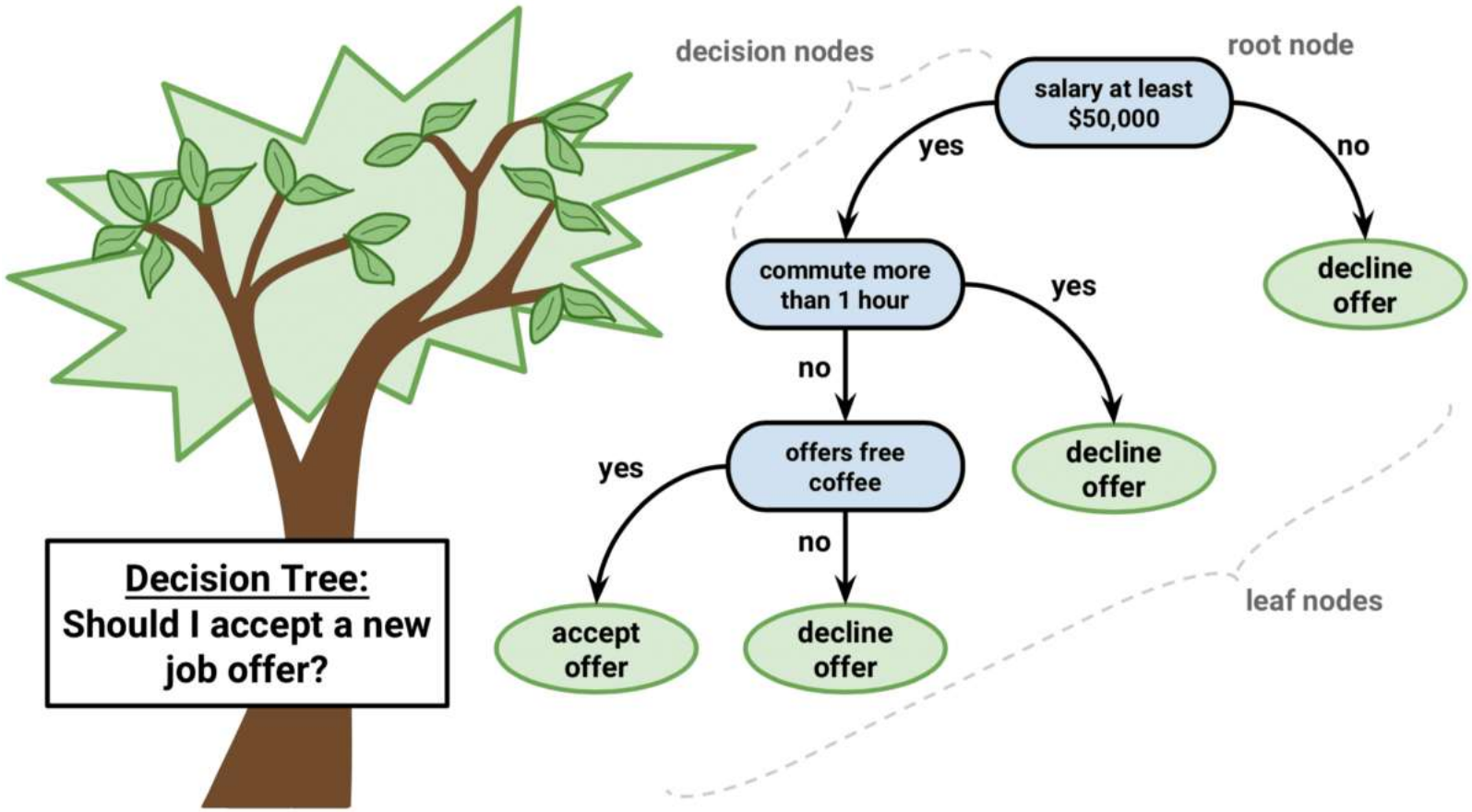
Decision Tree

- Decision tree is also a supervised algorithm for classification and regression
- It is not very powerful
- But it is easy to understand and interpret
- It is expanded with random forest, bagging, and boosting to increase its capabilities

Interesting properties of decision trees

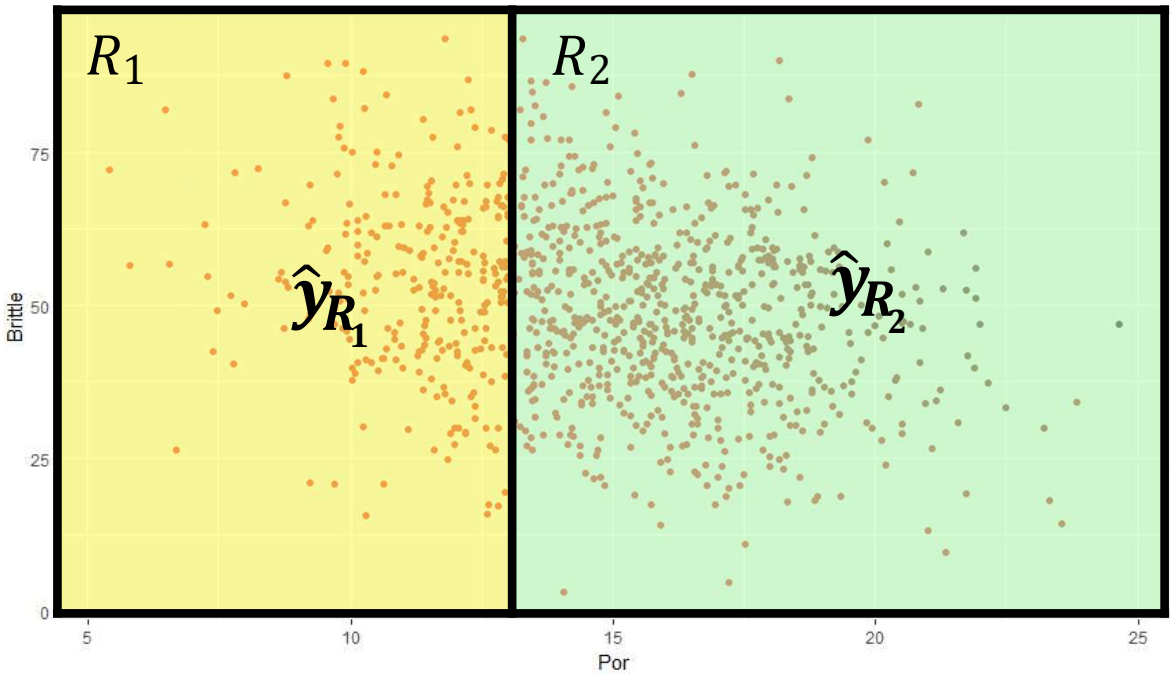
- They require little or no data preparation
- They do not require feature scaling
- Easy to explain
- Similar to how humans make decision
- They are white box models
- Lower predictive accuracy than other machine learning methods

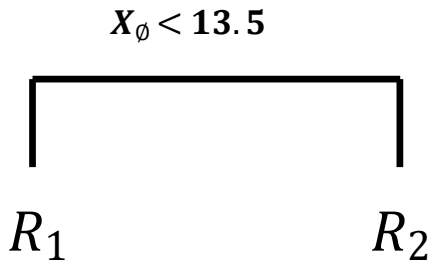
Decision Tree is similar to how we make a decision



Decision Tree for dividing samples based on porosity (1)

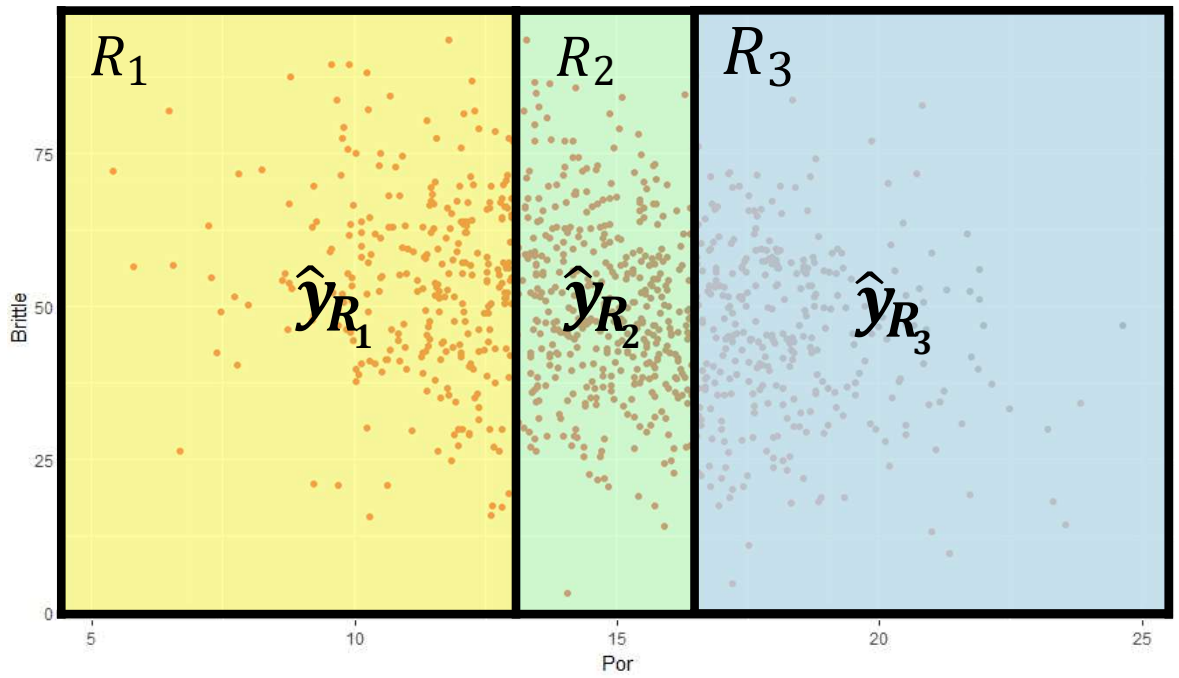
- How do we construct the Regions R_1, R_2, …, R_J?

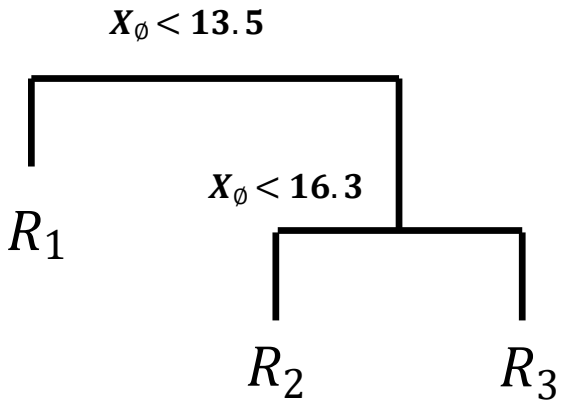




Decision Tree for dividing samples based on porosity (2)

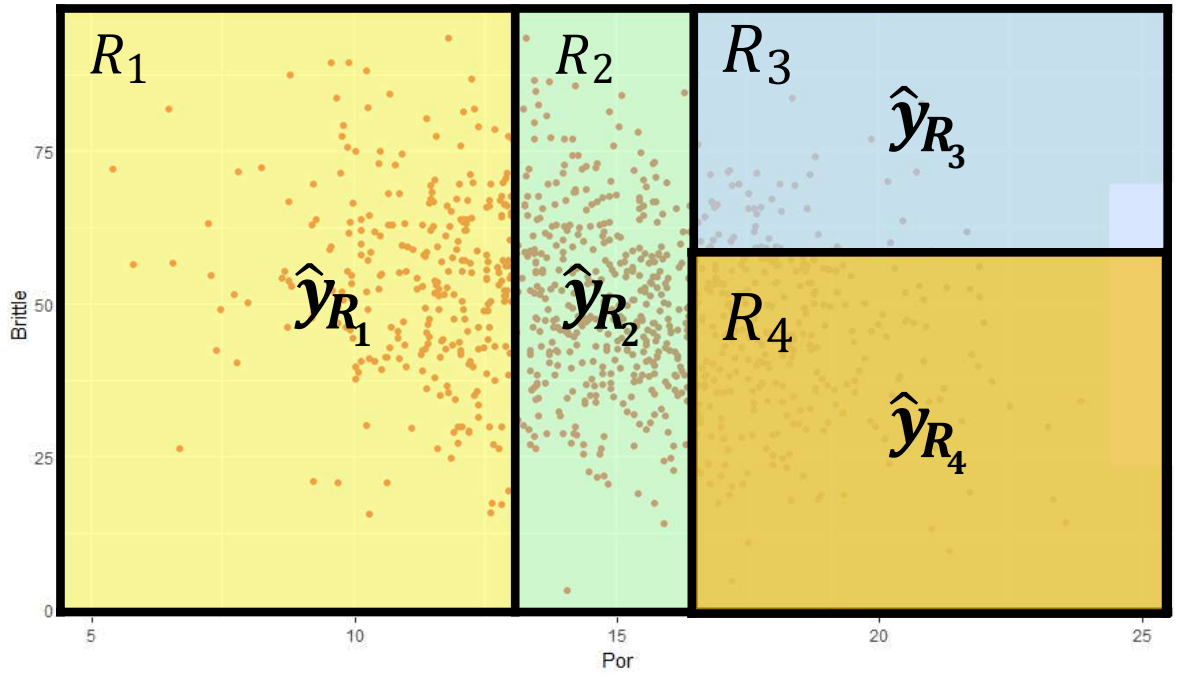
- How do we construct the Regions R_1, R_2, …, R_J?

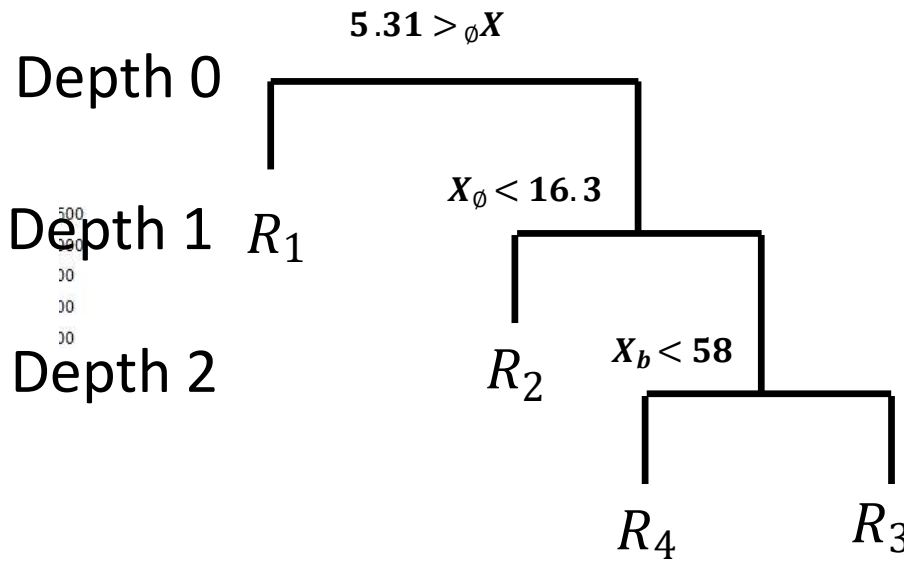




Decision Tree for dividing samples based on porosity (3)

- How do we construct the Regions R_1, R_2, …, R_J?





Depth 0,1,.. shows the penetration into the model

We have different types of nodes in the tree

First node is the root node

Q: what is child node? leaf node? split node?

Termination of decision tree

- When do we stop splitting?
  - We could continue until each datum has its own box!
    - This would be over fit!
  - Typical approach applies a minimum training data in each box criteria
    - It stops when all boxes have reached the minimum
  - We could continue until we cannot reduce Residual Sum of Squares (RSS)
    - But the current split could lead to an even better split → short sighted

We prefer a simplified decision tree

- The splitting process is deterministic, and it continues until there is no impurity (obvious overfit)
- Decision trees, if allowed to grow complicated, will generally overfit
- It is better to simplify the tree to a smaller tree with fewer splits
    - lower model variance
    - better interpretation
    - lower model bias
- Hyperparameter tuning is concerned with finding the optimal values using cross validation (an example is discussed in the second code in this lecture).

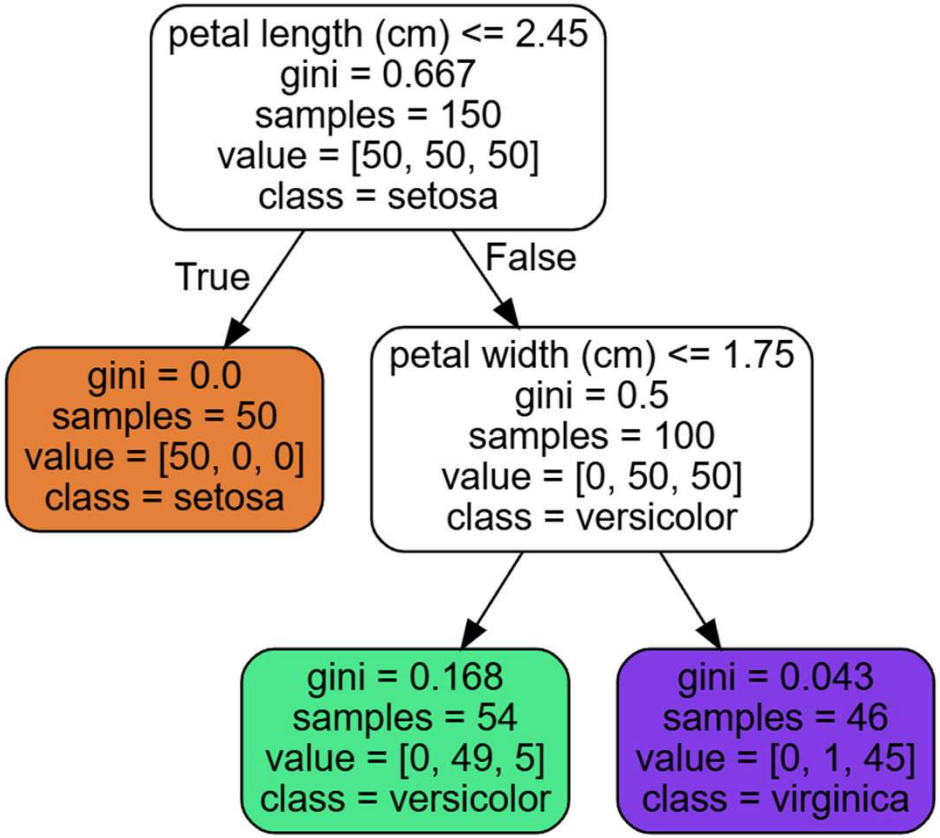
Gini impurity

- Gini impurity shows how often an element would be incorrectly labeled if it was randomly labeled
- gini=0 if all samples belong to a single class (a node is pure)

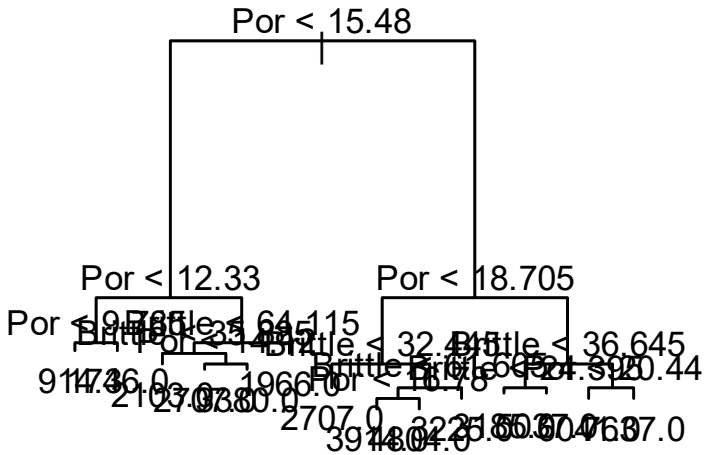
$$G_i = 1 - \sum \left(p_{i,k}\right)^2$$

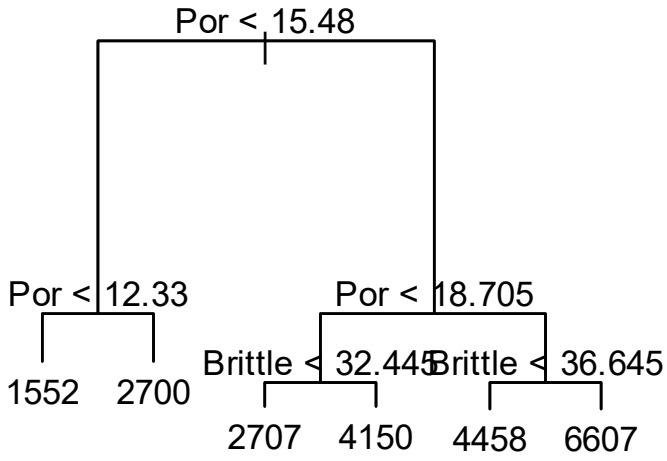
- G_i: gini impurity of the ith node
- $p_{i,k}$: the ratio of class *k* instances among all data in the ith node

Calculate the gini impurities



Is it possible to have more than two child nodes?





Cost function of the decision tree

- Classification and regression tree (CART) searches for pair (k,t_k) to generate the purest subsets possible by minimizing the following cost

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

G: Gini impurity

$m_{left(or\ right)}$: Number of instances in left or right

- It keeps splitting the data until some criterion stops it

Stopping criterion

A. Classification and Regression Tree (CART) stops if it cannot find a split to lower the impurity (this rarely happens in practice)

B. CART stops splitting when it reaches preset criteria:

B1. It reaches the maximum depth of the model (we set max_depth)

B2. It reaches the max_leaf_nodes

B3. It reaches max_features

….

Note: CART is a greedy algorithm

Entropy is also used in the cost function

- Inspired by the entropy property in thermodynamics (or information theory):

$$H_i = -\sum p_{i,k} \log(p_{i,k})$$

- Entropy and gini impurity usually generate similar decision trees
- When they differ, entropy generates a more balanced decision tree
- Gini is slightly faster (notice the log function in the entropy)

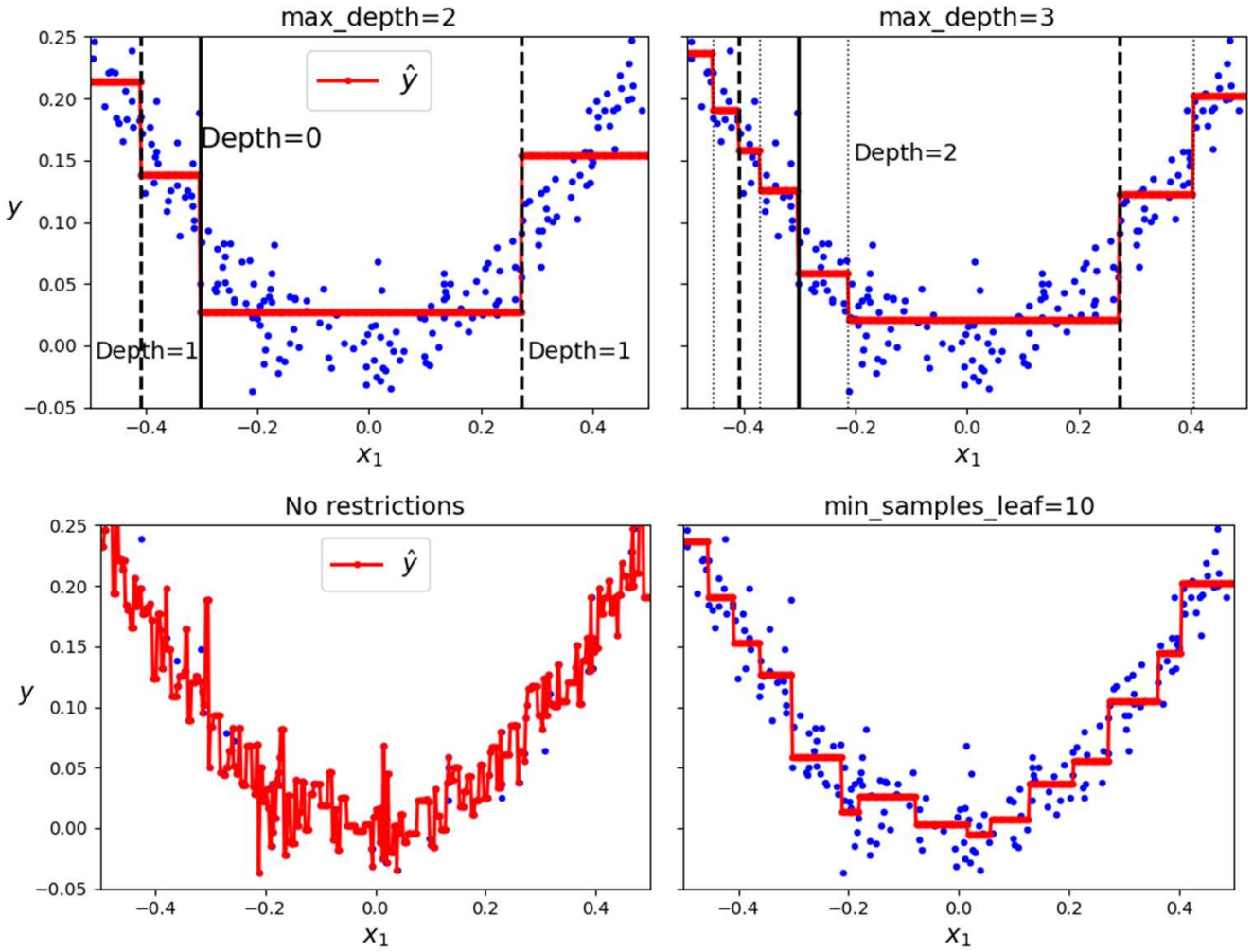
Decision tree also applies to regression but requires a relevant cost function

- It splits the data by minimizing the MSE (instead of gini)

$$J(k, t_k) = \frac{m_{left}}{m} \mathrm{MSE}_{\mathrm{left}} + \frac{m_{right}}{m} \mathrm{MSE}_{\mathrm{right}}$$

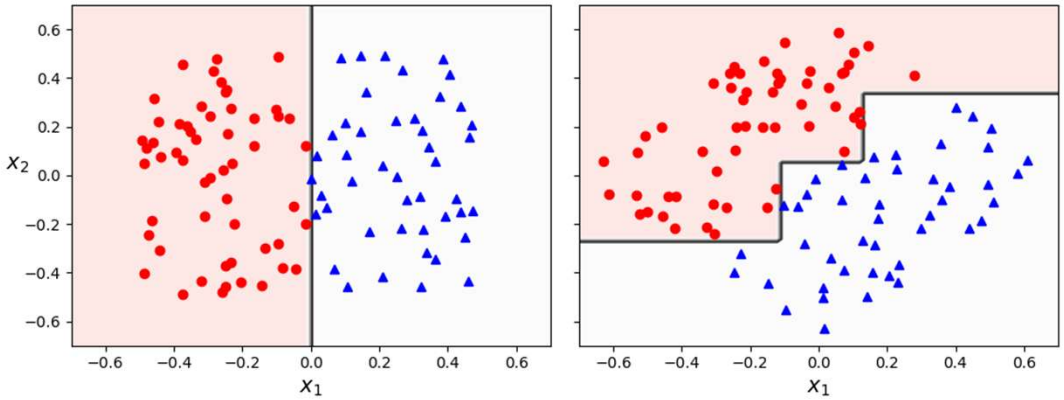
$\mathrm{MSE}_{\mathrm{node}}$ is the mean squared error that captures the difference between the predicted value and the **AVERAGE of actual values** in each node

Example of overfitting with no regularization



Two major shortcomings of decision tree

- They are very sensitive to data orientation because of the orthogonal boundaries



- Small changes in hyperparameters produce different results

Effects of the stochasticity on decision tree

- The training algorithm is stochastic because it randomly selects the features to evaluate at each node

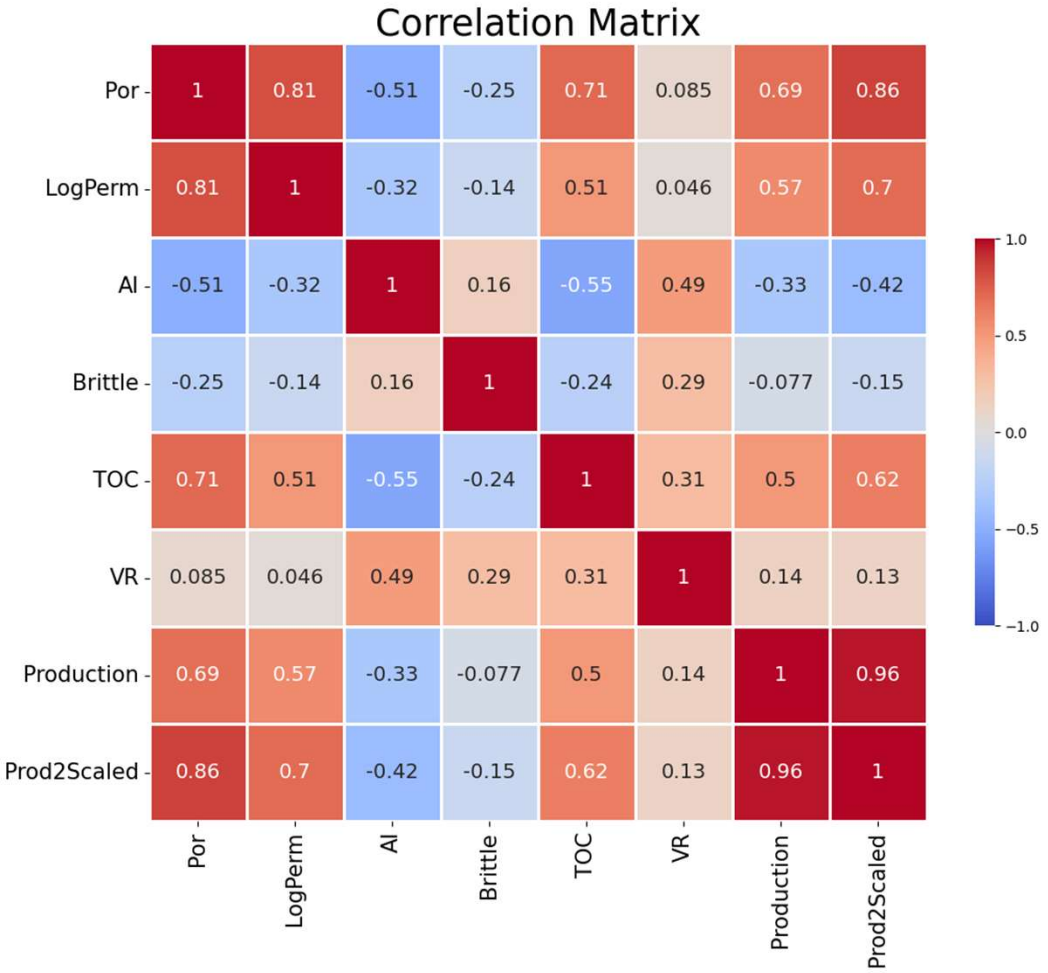
- This means that we may get very different decision trees for the same dataset (unless you set the random_state)

- Remember that the decision tree is a greedy algorithm

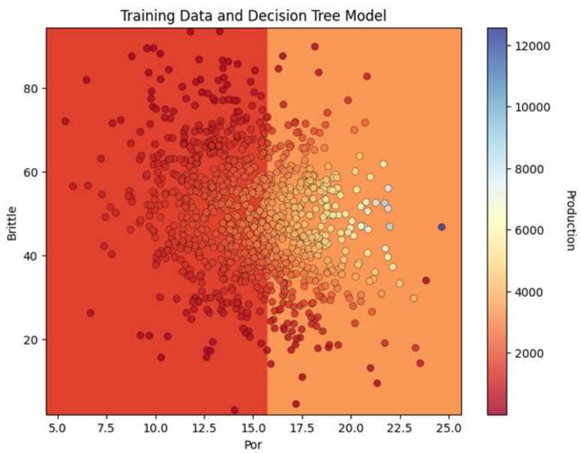
- Q: how do we address the negative impact of stochasticity on the decision tree?

Code 1

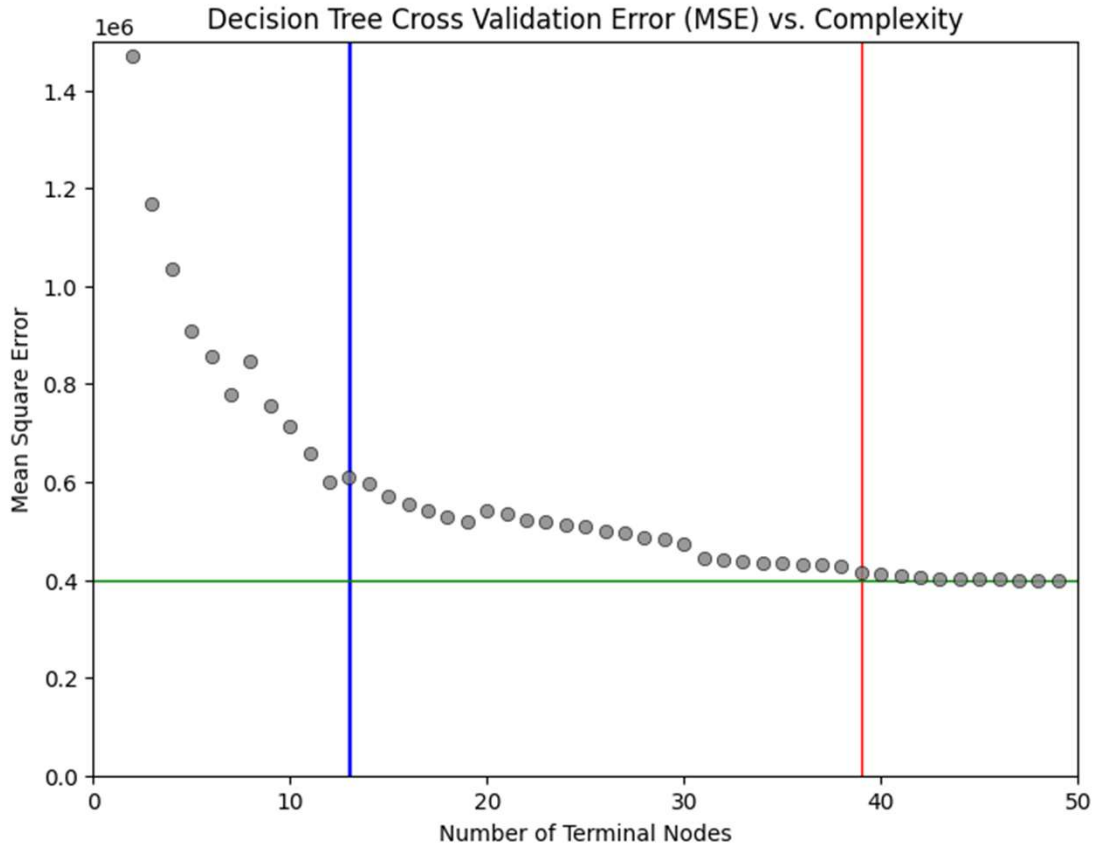
- Run “Decision_Tree” after uploading the data (unconv_MV_v2)
- How would you interpret the correlation matrix



Evolution of the decision tree with increasing the number of nodes



Determine and justify the best number of terminal nodes



- Ensemble Learning and Random Forests

Fundamental idea of ensemble learning

- Main idea: a group of people is smarter than a single person (wisdom of the crowd)
- The same applies to machine learning: a group of classifiers (or regressors) is better than a single classifier
- Ensemble is simply a group of classifiers (or regressors)
- Ensemble learning (sometimes called ensemble method) is based on ensemble models
- Q: how do we choose from the predictions?

Voting classifiers

- Hard voting: choose the outcome with the highest vote (similar to electing people based on the majority vote)
- The outcome is a strong learner even if we use weak learners, especially if there are many

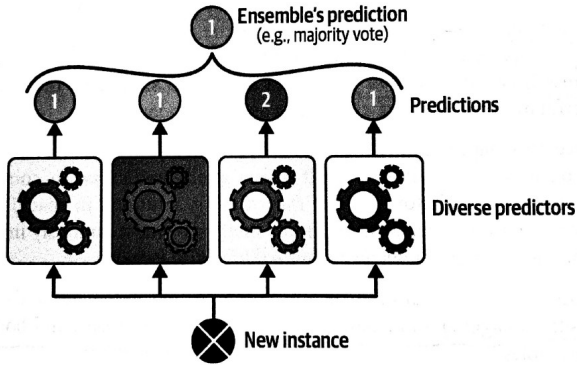


Fig. 7.2 of the reference

- The ensemble (group of classifiers) performs better than its best classifier, again similar to how a society with many people behave
- Soft voting: gives more weight to the models with higher scores (performs better than hard voting)

Methods to improve the ensemble performance

- In society, it would be better to have diverse ideas from different people
- Example of hard coding: In solving data analytics problems, it is better to form a diverse team (scientists, engineers,…)
- Example of soft voting: In solving data analytics problems, it is better to form a diverse team with good scientists and good engineers
- Ensemble methods work best where the predictors are independent

Methods to create diverse classifiers

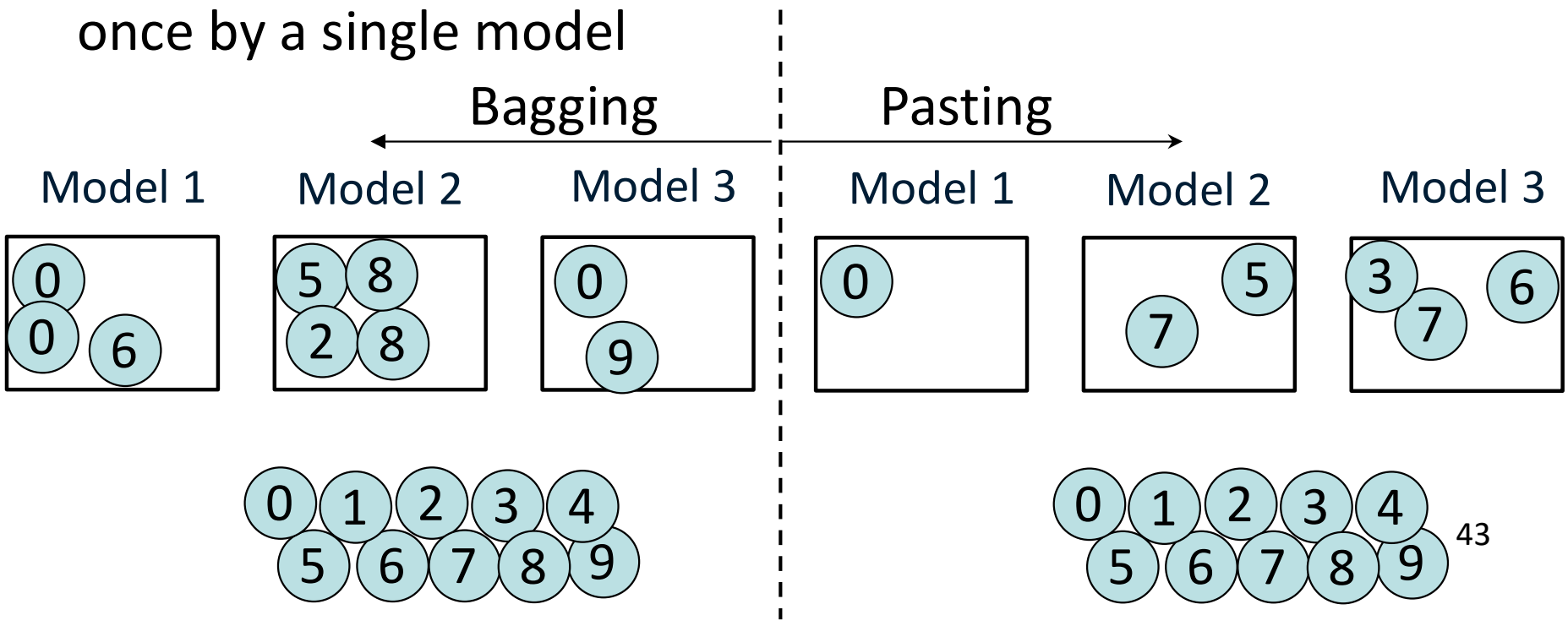
A. Use different algorithms (SVM, Decision Tree,…)

B. Use different subsets of data (Bagging, Pasting)

C. Apply both

Bagging versus Pasting

- In Bootstrap aggregating (Bagging), we sample with replacement
- In Pasting, we sample without replacement
- Bagging allows a training instance to be sampled more than once by a single model

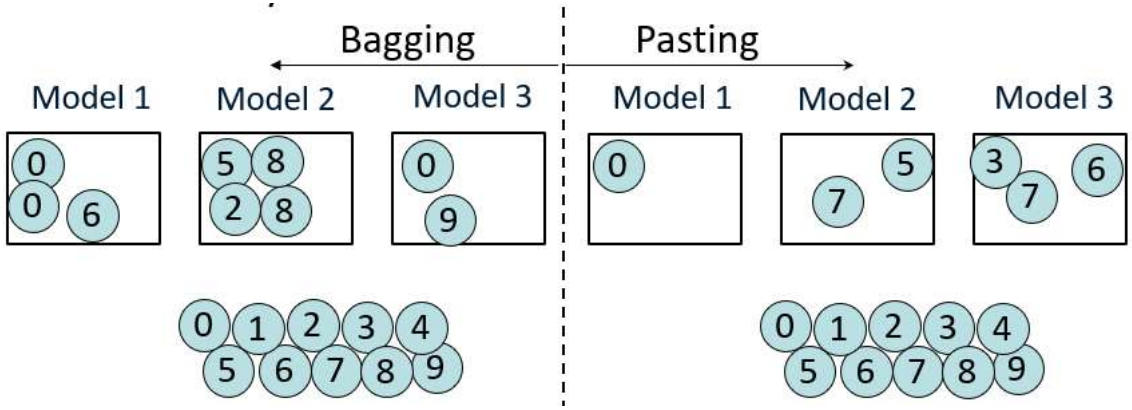


Aggregating the results

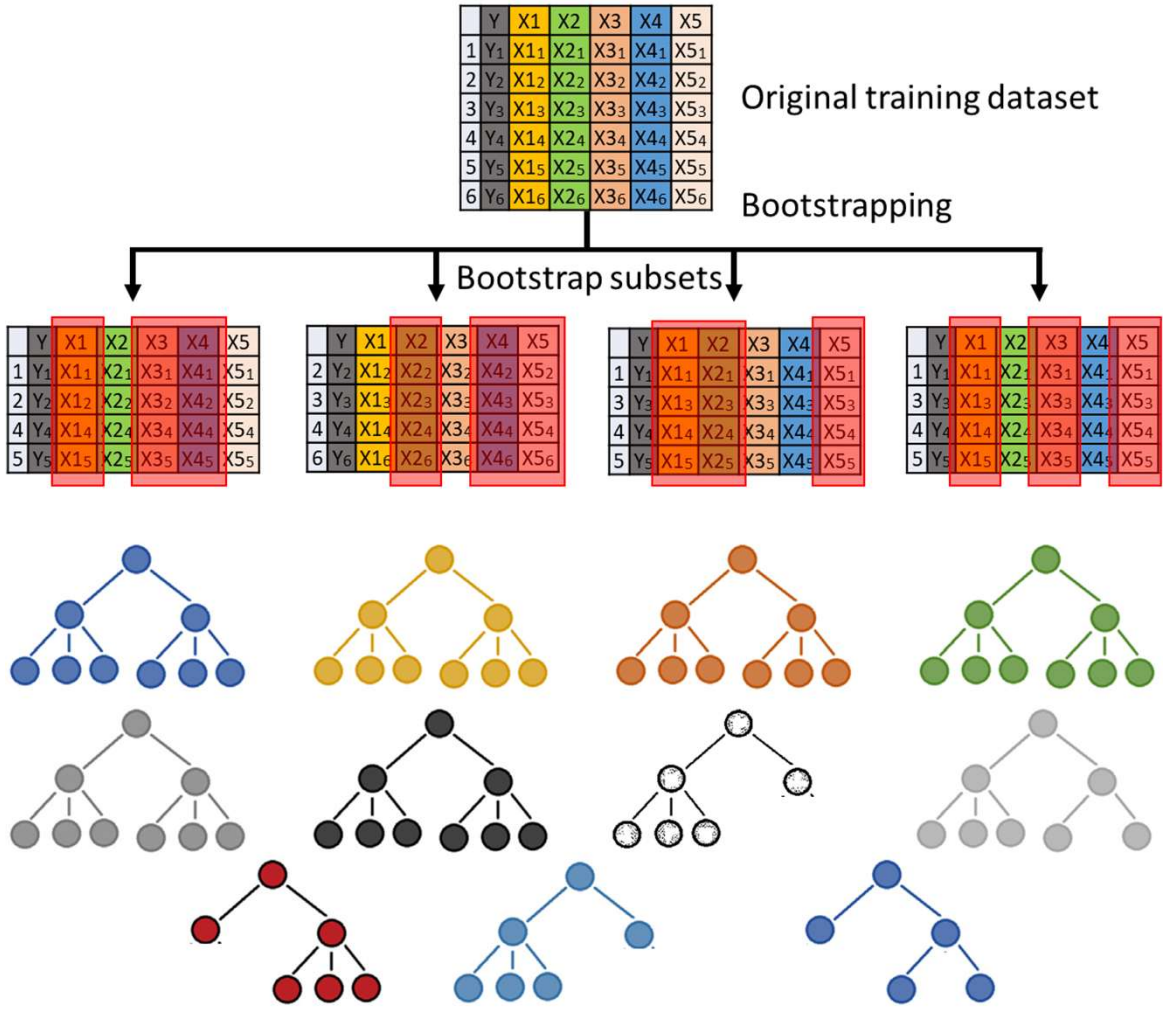
- Classification (usually) uses the most frequent answer
- Regression (usually) averages the results
- Aggregated results generalize better
- Again, remember that the ensemble model usually performs better than a single predictor (classifier or regressor)

Performance of bagging vs pasting

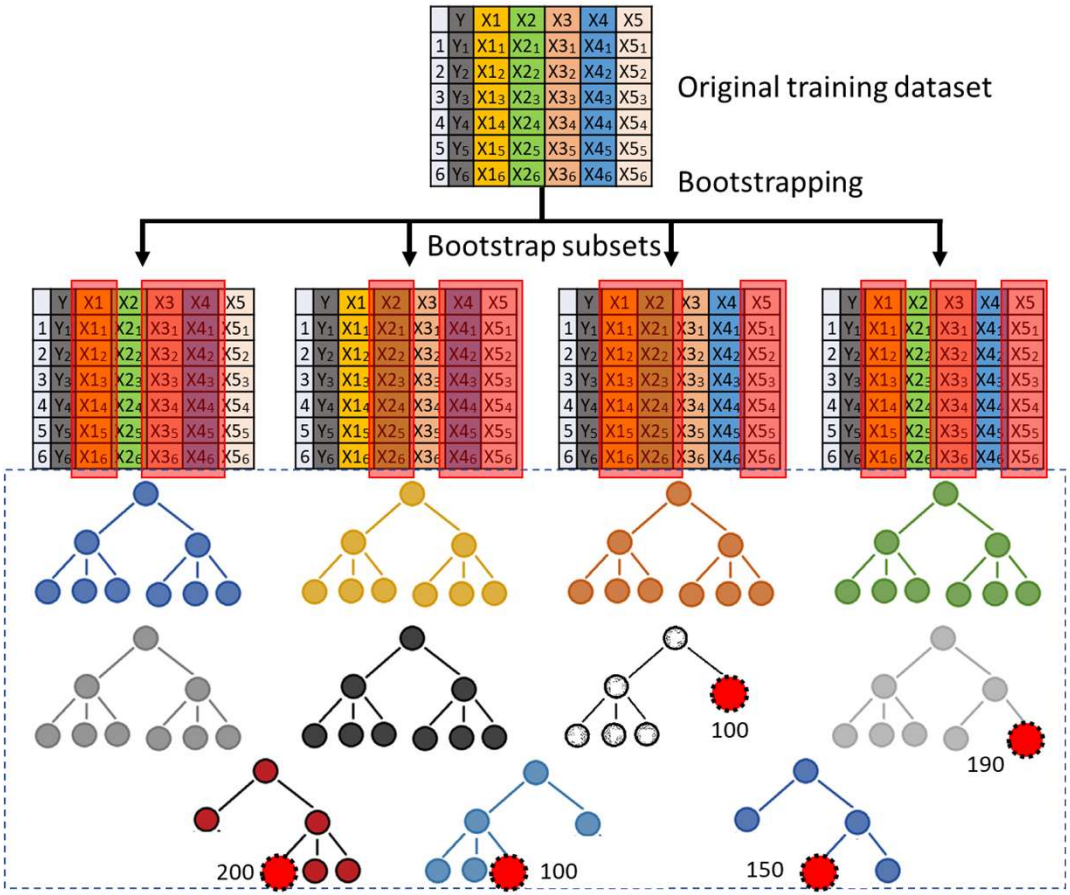
- Bagging is usually preferred in practice because it performs better with more diversity and lower variance
- Q1 (answer after reading the book): what are the effects of bagging and pasting on the ensemble bias?
- Q2 (answer after reading the book): which fraction of the training instances are not sampled for each predictor in bagging?



Random Forest is an ensemble of decision trees used for classification or regression



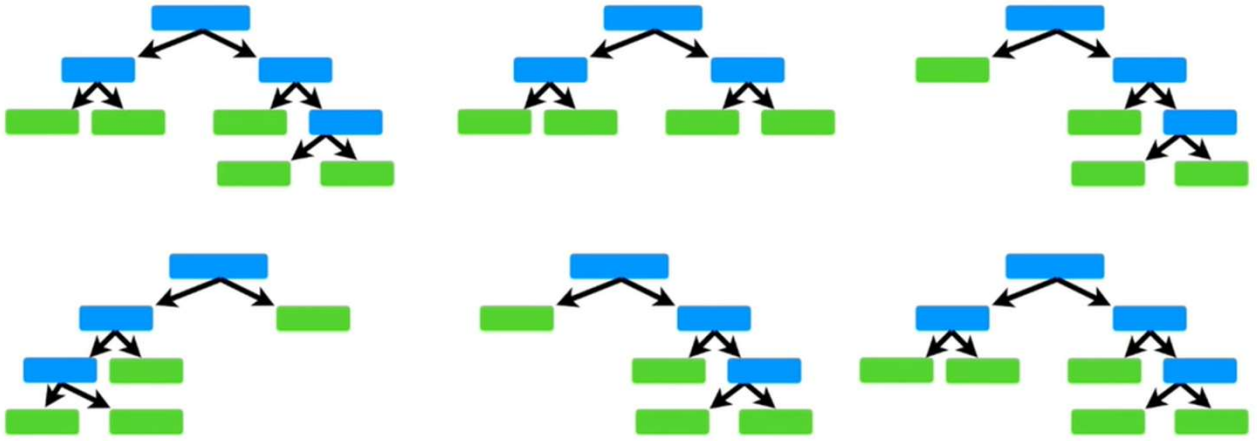
Example of random forest regression



- What is the predicted value in this example?

Random Forest vs Decision Tree Classification

- Random forest reduces variance with minimal increase in bias
- It is difficult to interpret random forest
- There is a lack of transparency in the random forest



Extra Trees

- Extra trees (or extremely randomized trees) is a supervised algorithm similar to random forest
- Extra trees algorithm uses the entire original sample (it is not based on bagging)
- Extra trees are used for classification and regression

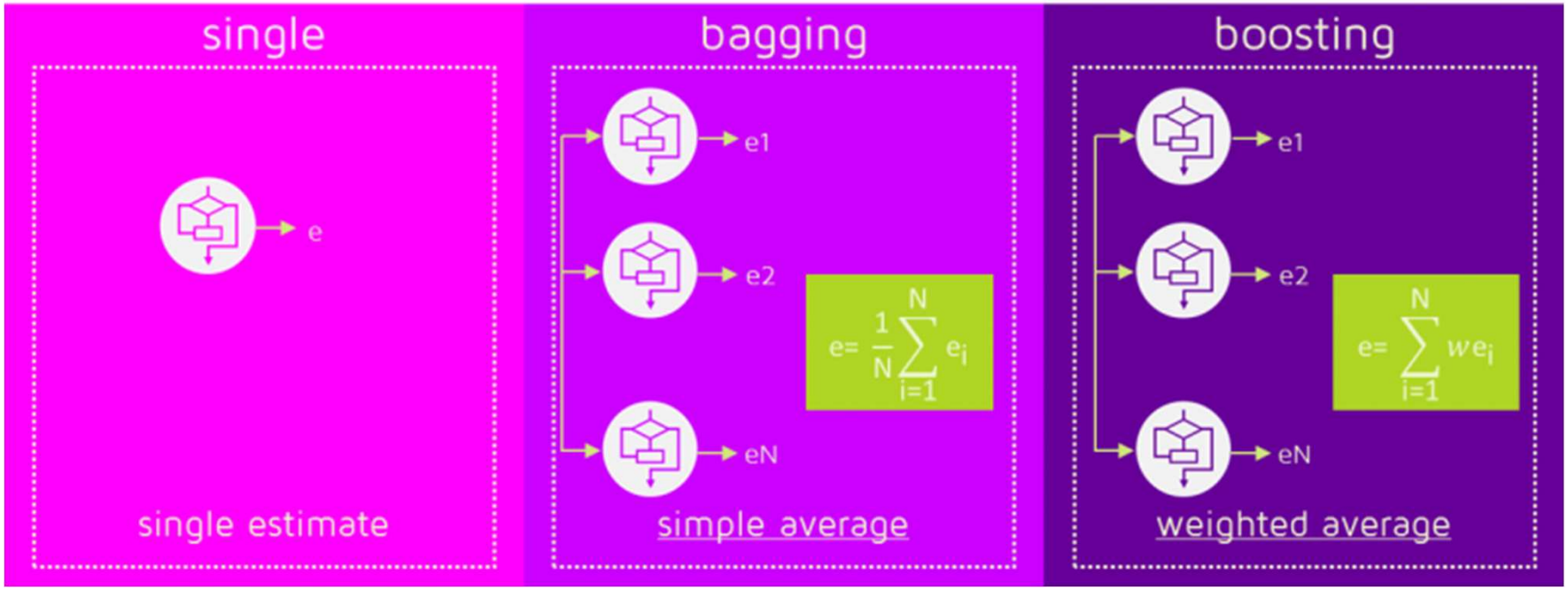
Main features of Decision Tree, Random forest, and Extra tree

Metrics	Decision tree	Random Forest (RF)	Extra trees (ET)
Number of trees	1	Many	Many
Bootstrapping	N/A	Yes	No (option for bootstrapping is available in scikit-learn)
# of features for the split at each decision	All features	Random subset of Features	Random subset of features
Splitting criteria	Best split	Best split	Random split

Boosting

- Boosting is a process where we combine several learners to create a stronger learner
- Adaptive Boosting (AdaBoost) improves the performance of a predecessor that underperforms
- Gradient Boosting minimizes a loss function by optimizing the weights (placing more weights on instances with erroneous predictions)

Gradient Boosting vs. Bagging



Gradient Boosting

- In gradient boosting, after building the weak learners, the predictions are compared with actual values
- The difference between prediction and actual values represents the error rate
- The error rate is used to calculate the gradient, which is the partial derivative of the loss function
- The gradient is used to find the direction that the model parameters would have to change to reduce the error in the next round of training.

Boosting Workflow

- Boosting applies multiple weak learners to build a stronger learner
  - A weak learner offers predictions slightly better than random selection
  - A weak learner builds a simple model with a high error rate, the model can be quite inaccurate, but moves in the correct direction
  - Calculate the error from the simple model
  - Fit another model to the error
  - Calculate the error from the addition of the first and second model
  - Repeat until the desired accuracy is obtained or some other stopping criteria

eXtreme Gradient Boosting (XGBoost)

- XGBoost is a more regularized form of Gradient Boosting
- XGBoost uses $\ell 1$ and $\ell 2$ regularization to improve model generalization and overfitting reduction
- XGBoost is usually faster than gradient boosting due to the parallelization
- XGBoost can handle missing values within a data set (data preparation is not as time-consuming)

When to use XGBoost?

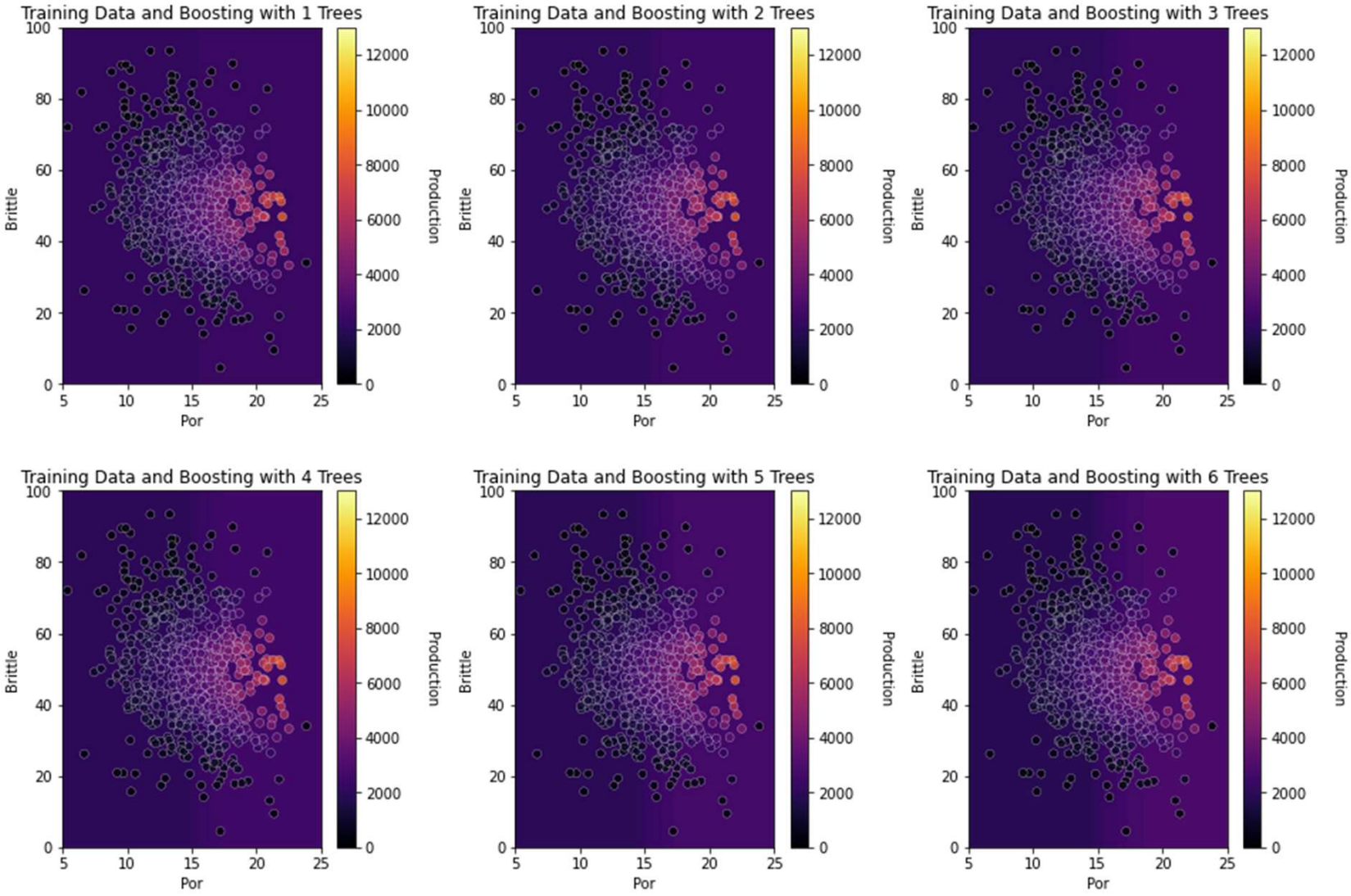
1. When there is a larger number of training samples: Ideally, more than 1000 samples and less than 100 features or we can say when the number of features < number of training samples.

2. When there is a mixture of categorical and numeric features or just numeric features.

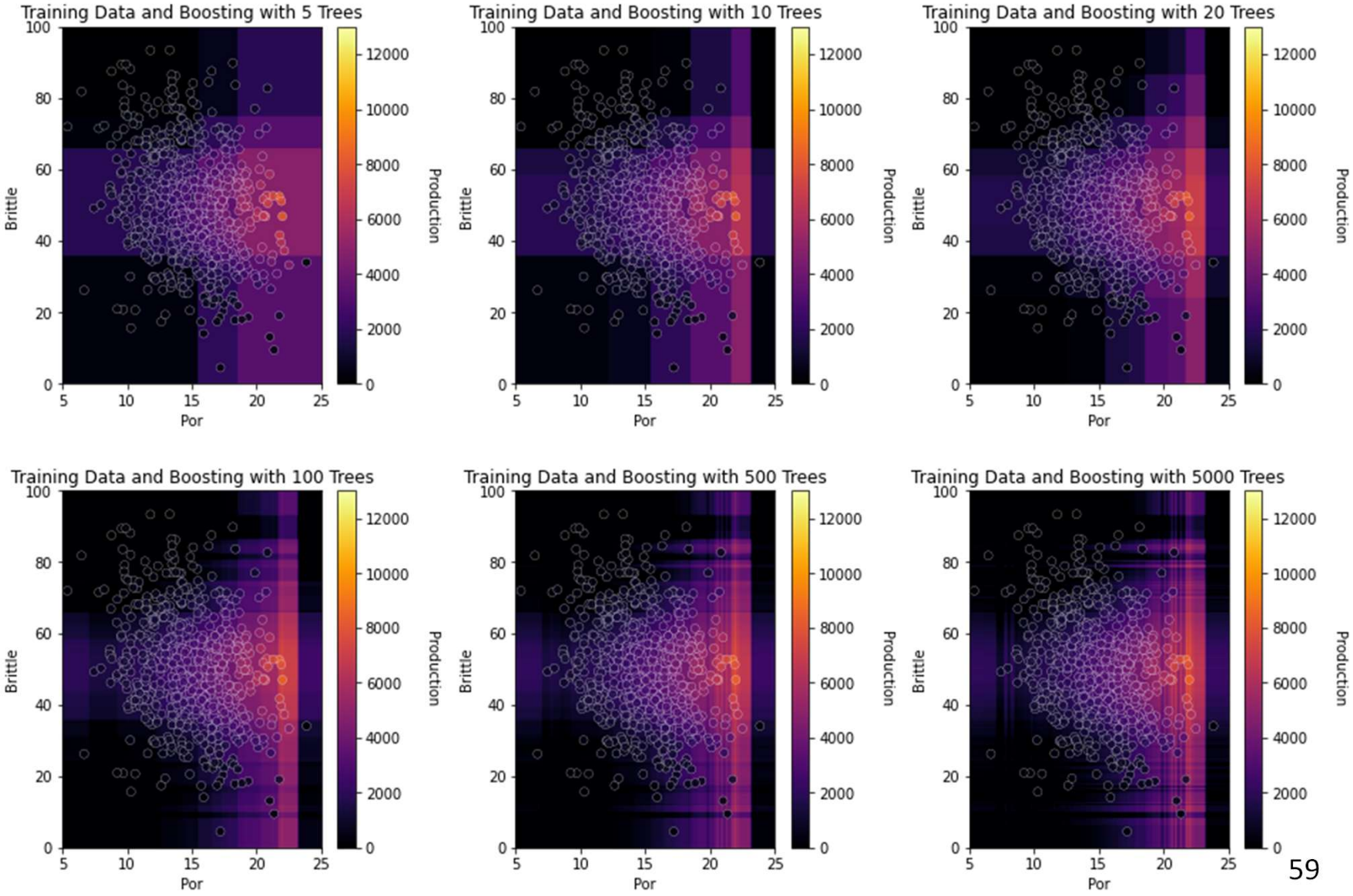
Code 2

- Run “GradientBoosting_and_XGBoost”

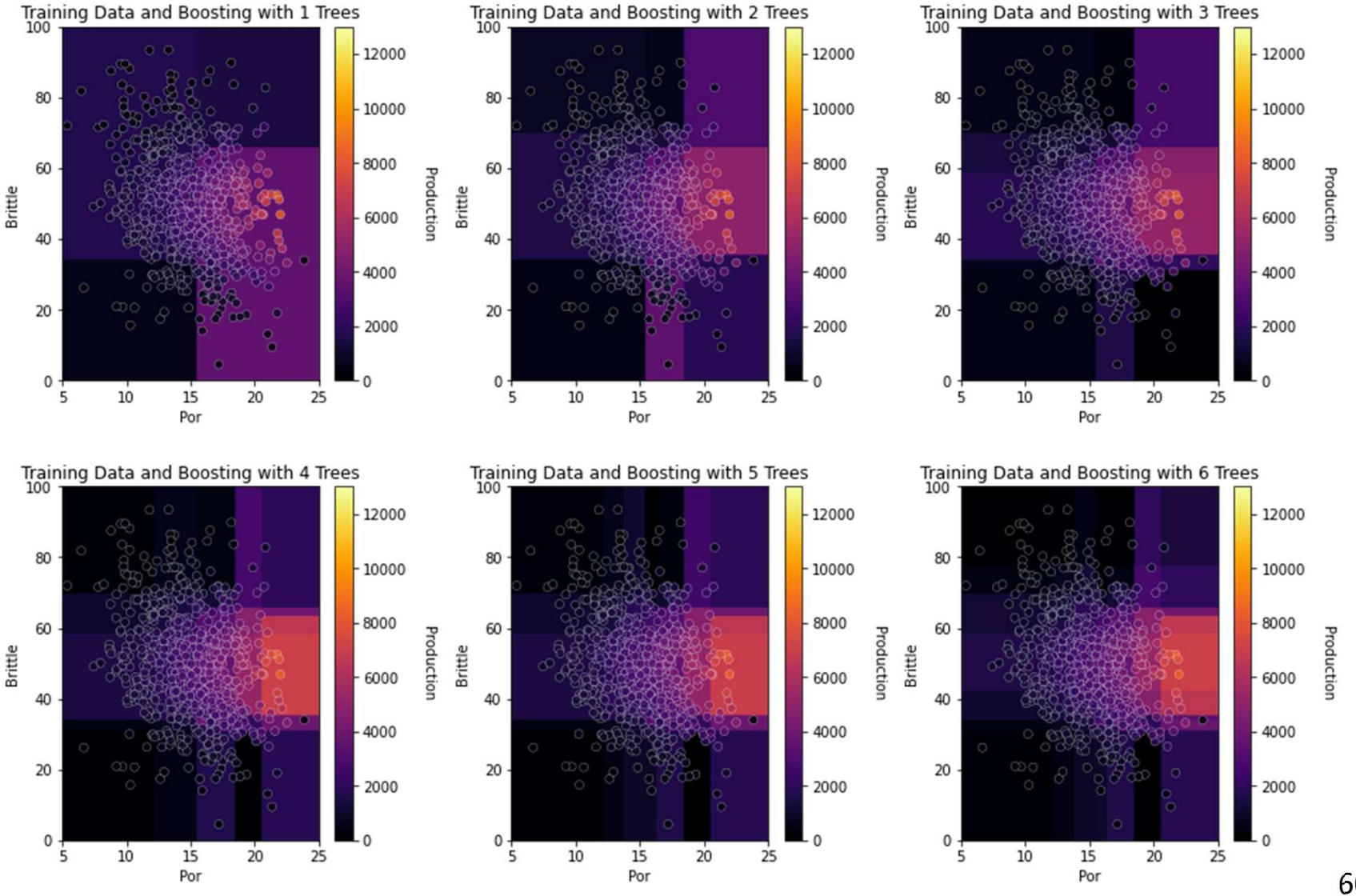
Significant misfit when we use up to 6 decision trees, **but each makes one decision**. **Each tree is just a "correction" to the previous one**. The combination of trees allows make more complex predictions. **Depth of each tree=1**



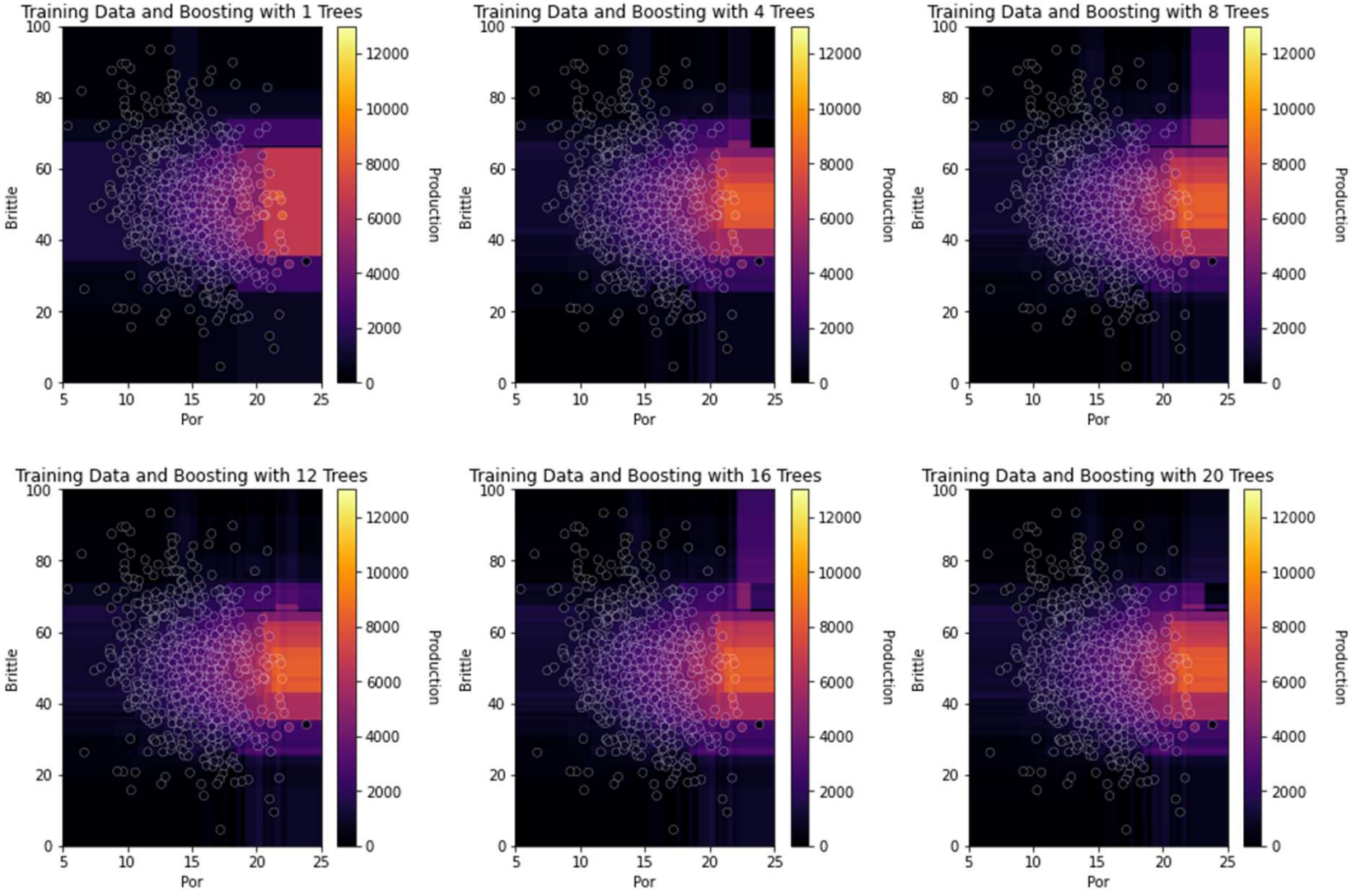
More trees but each has **depth = 1**



More patterns appear when **depth = 2** even with limited trees



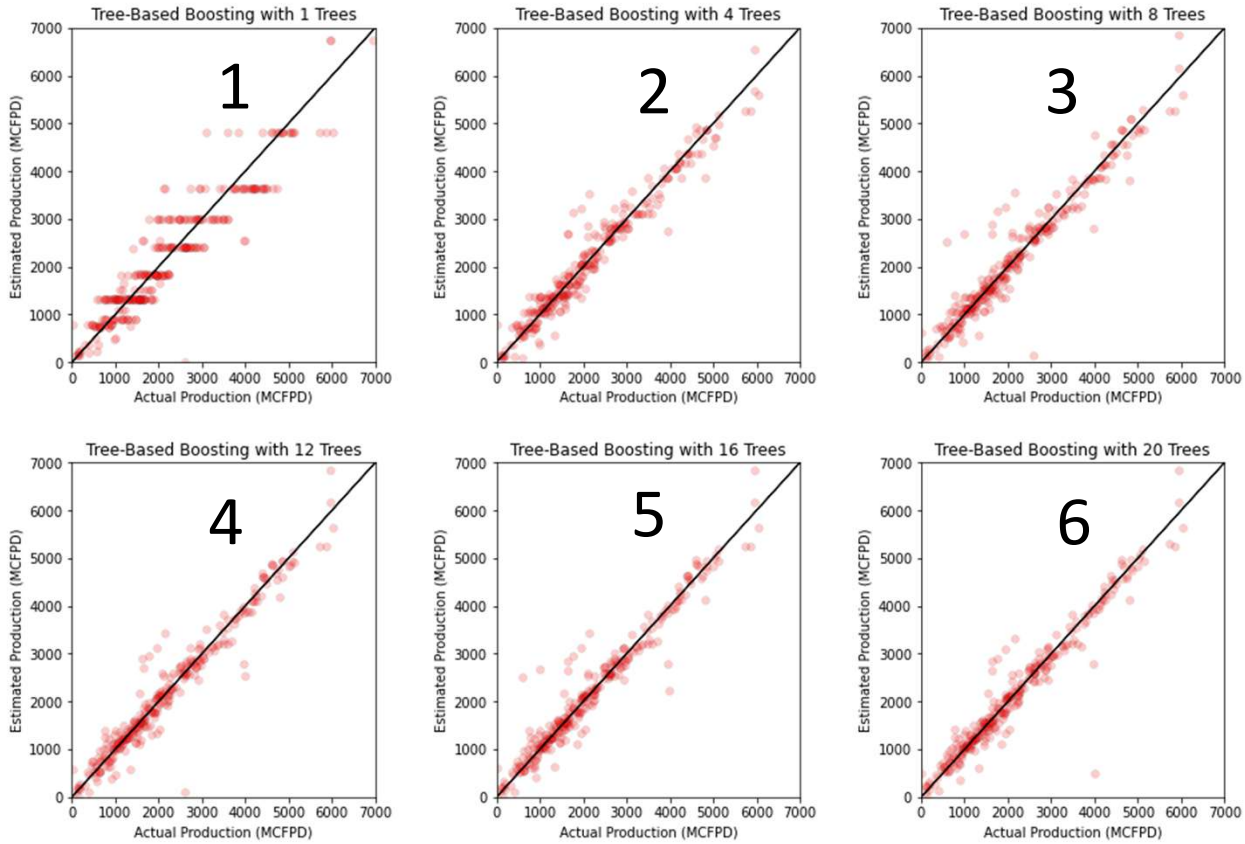
More trees with depth = 5



Finding the best solution

- The results look more interesting as we increase the number of trees and their depths
- We analyze the performance on test data to find the best scenario (cross validation)

Identify the best model



1, Mean Squared Error on Training = 345995.84 , Variance Explained = 0.85 Cor = 0.92

2, Mean Squared Error on Training = 209960.36 , Variance Explained = 0.91 Cor = 0.95

3, Mean Squared Error on Training = 188190.61 , Variance Explained = 0.92 Cor = 0.96

4, Mean Squared Error on Training = 164960.31 , Variance Explained = 0.93 Cor = 0.96

5, Mean Squared Error on Training = 166082.49 , Variance Explained = 0.93 Cor = 0.96

6, Mean Squared Error on Training = 176161.67 , Variance Explained = 0.92 Cor = 0.96

XGBoost

- TO-DO go through the rest of code and analyze the results