# CSCI 556 Data Analysis & Visualization

## Statistical Machine Learning

Instructor: Dr. Jinoh Kim

## Statistical machine learning

- Powerful automated techniques for predictive modeling (regression and classification) methods fall under the umbrella of statistical machine learning (ML)
- *K*-Nearest Neighbors (KNN): classify a record in accordance with how similar records are classified
- Decision trees: learn rules about the relationships between predictor variables and outcome variables
- Ensemble learning based on decision trees

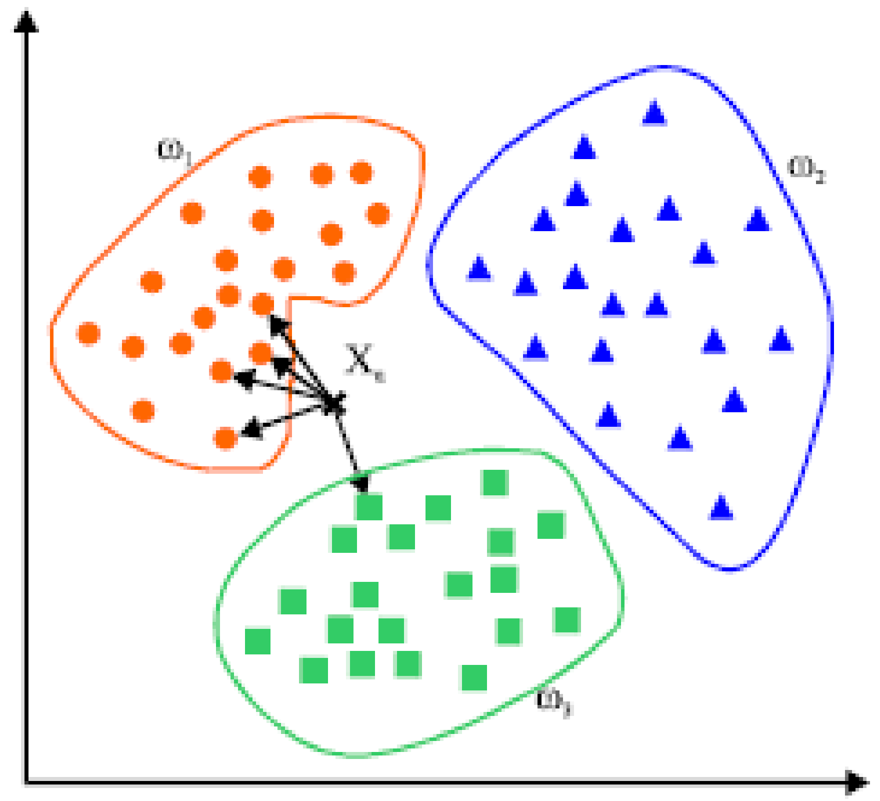
# *K*-Nearest Neighbors (KNN)

- One of the simpler prediction/classification techniques without the assumption of a model to be fit
- Basic idea:

  For each record,

  1. Find *K* records that have similar features (i.e., similar predictor values)
  2. For classification: Find out what the majority class is among those similar records, and assign that class to the new record
  3. For prediction (also called *KNN regression*): Find the average among those similar records, and predict that average for the new record

# KNN example



Source: mathworks.com

- Sample X has five closest neighbors.
- What class would it assigned to X?

# KNN prediction/classification

- KNN prediction results depend on:
  - how the features are scaled => standardization/normalization
  - how similarity is measured => distance metrics
  - how big *K* is set => choosing *K*
  - Also, all predictors must be in numeric form => one-hot encoding

## Distance metrics

- Determines similarity (nearness)
- Measures how far two records $(x_1, x_2, \dots, xp)$ and $(u_1, u_2, \dots, u_p)$ are from one another
- Euclidean distance: corresponds to the straight-line distance between two points

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}$$

- Manhattan distance: distance between two points traversed in a single direction at a time (e.g., traveling along rectangular city blocks)

$$| x_1 - u_1 | + | x_2 - u_2 | + \cdots + | x_p - u_p |$$

# Euclidean vs. Manhattan distance

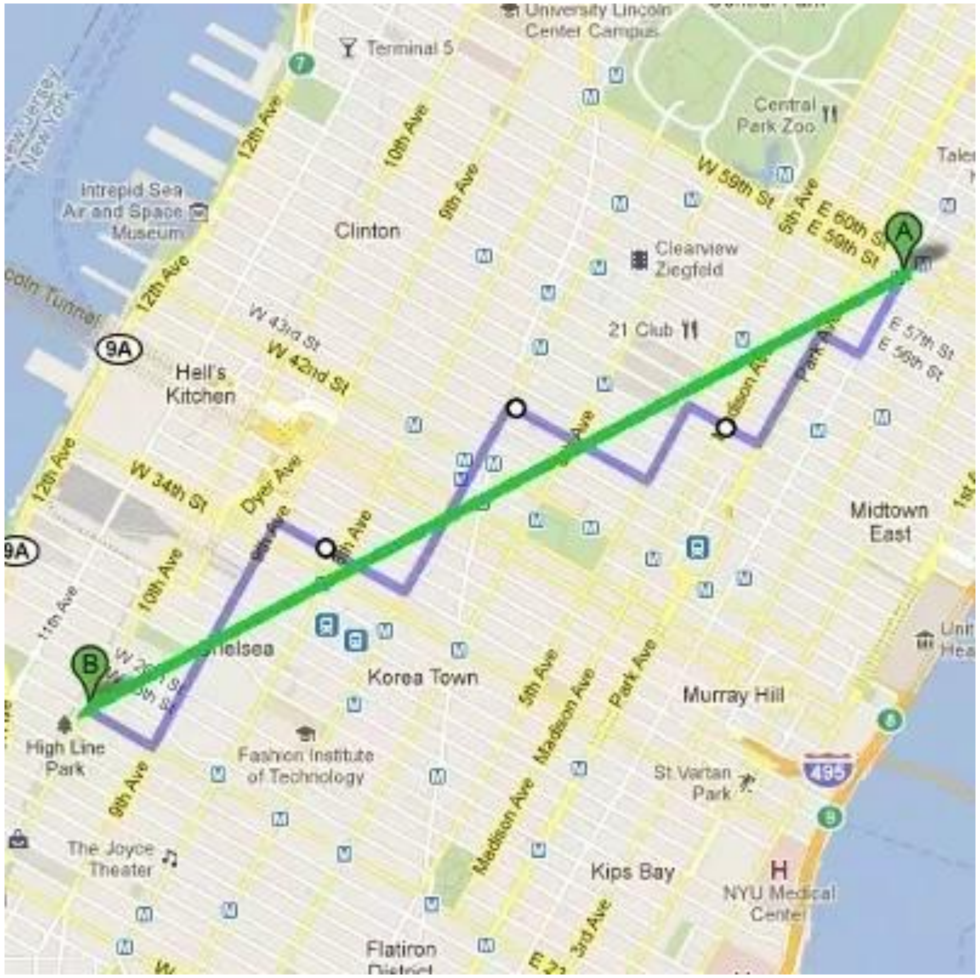


Image from quora.com

## Standardization (normalization)

- Want to ensure that a variable does not overly influence a model simply due to the scale of its original measurement
- Standardization (normalization) puts all variables on similar scales by subtracting the mean and dividing by the standard deviation
- Z-score: Measurements are stated in terms of "standard deviations away from the mean"

$$z = \frac{x - \bar{x}}{s}$$

, where $x$=input, $\bar{x}$ =mean, $s$=standard deviation, and $z$ is the normalized value (i.e., $x_{new}$)

- Min-max scaling: scales the range to [0, 1] by referring to the min and max value in the dataset

$$x_{new} = \frac{x - xmin}{x_{max} - xm_{in}}$$

# Normalization example

- Data X = <2,8,9,4,6,7,8,3,4,9,6>
- Z-score: Measurements are stated in terms of "standard deviations away from the mean"
  - Mean $\bar{x}$ =6, standard deviation s=2.582
  - For each value in X, z-score can be calculated

$$z = \frac{x - \bar{x}}{s}$$

- Min-max scaling: scales the range to [0, 1] by referring to the min and max value in the dataset
  - $x_{min}$=2, $x_{max}$=9
  - For each value in X, minmax scaling can be done

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

# Choosing K

- Choice of *K* is very important to the performance of KNN
- Simplest choice: 1-nearest neighbor classifier (K=1)
- No general rule about the best *K* — it depends greatly on the nature of the data
- Higher values of *K* provide smoothing that reduces the risk of overfitting in the training data
  - If *K* is too high, we may oversmooth the data and miss out on KNN's ability to capture the local structure in the data
- The *K* that best balances between overfitting and oversmoothing is typically determined by accuracy metrics (with holdout or validation data)

# Example: different *K*s

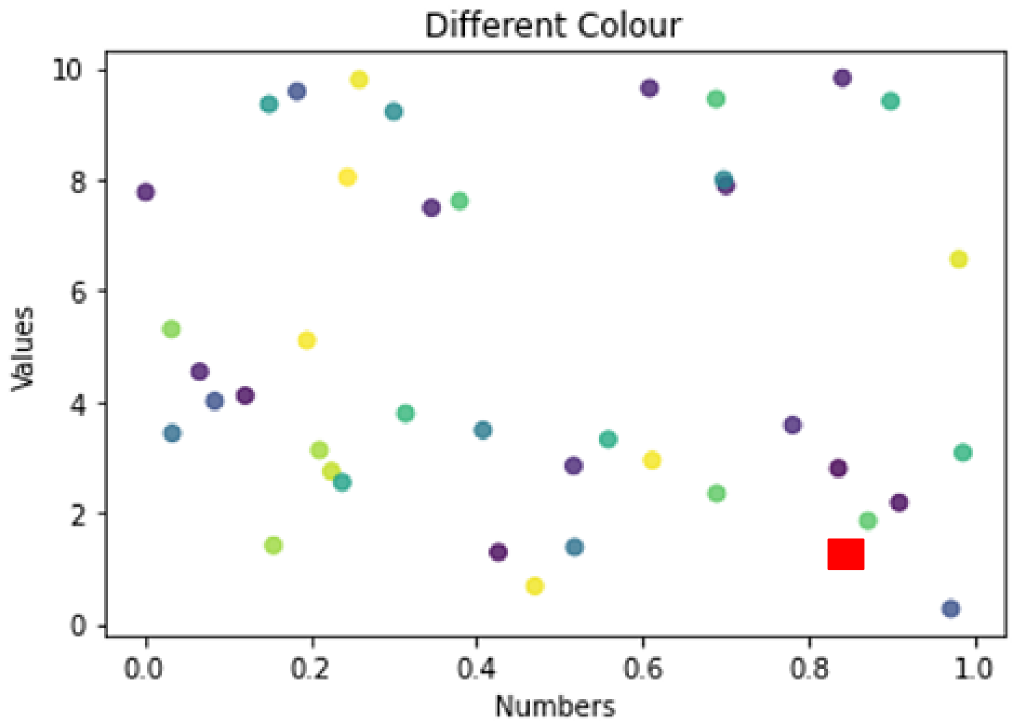
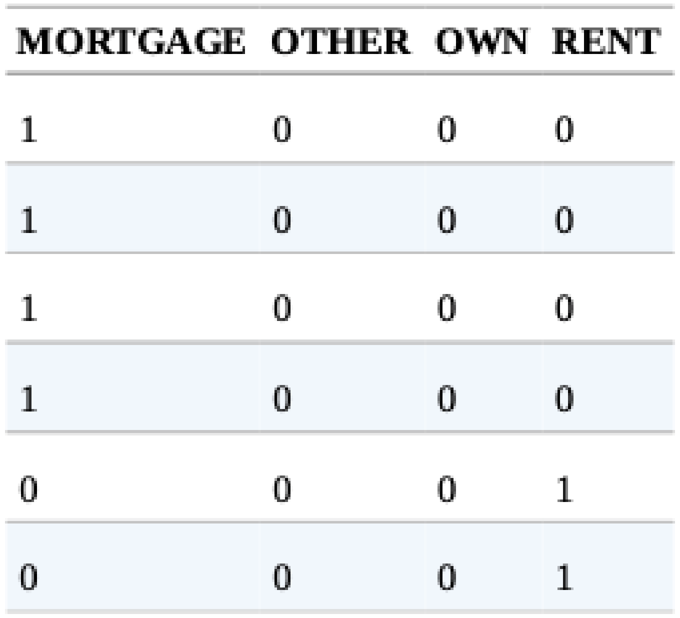


Image from https://www.includehelp.com/python/scatter-plot.aspx

- For class assignment to target (red square),
  - What if K=1? and K=3?

# One hot encoding

- Most ML models require categorical type of variables to be converted to a series of binary dummy variables conveying the same information
- One hot encoding creates creates a binary column for each category and returns a sparse matrix
- Example: categorical variable "home occupant status" has a value out of "mortgage", "rents", "other", or "own" (with no mortgage)
- One hot encoder creates four binary variables, with one 1 and three 0s



| MORTGAGE | OTHER | OWN | RENT |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 |

## KNN as a feature engine

- KNN can be used to add “local knowledge” in a staged process with other classification techniques:
  1. KNN is run on the data, and for each record, a classification (or quasi-probability of a class) is derived
  2. That result is added as a new feature to the record, and another classification method is then run on the data. The original predictor variables are thus used twice

## Example: KNN as a feature engine

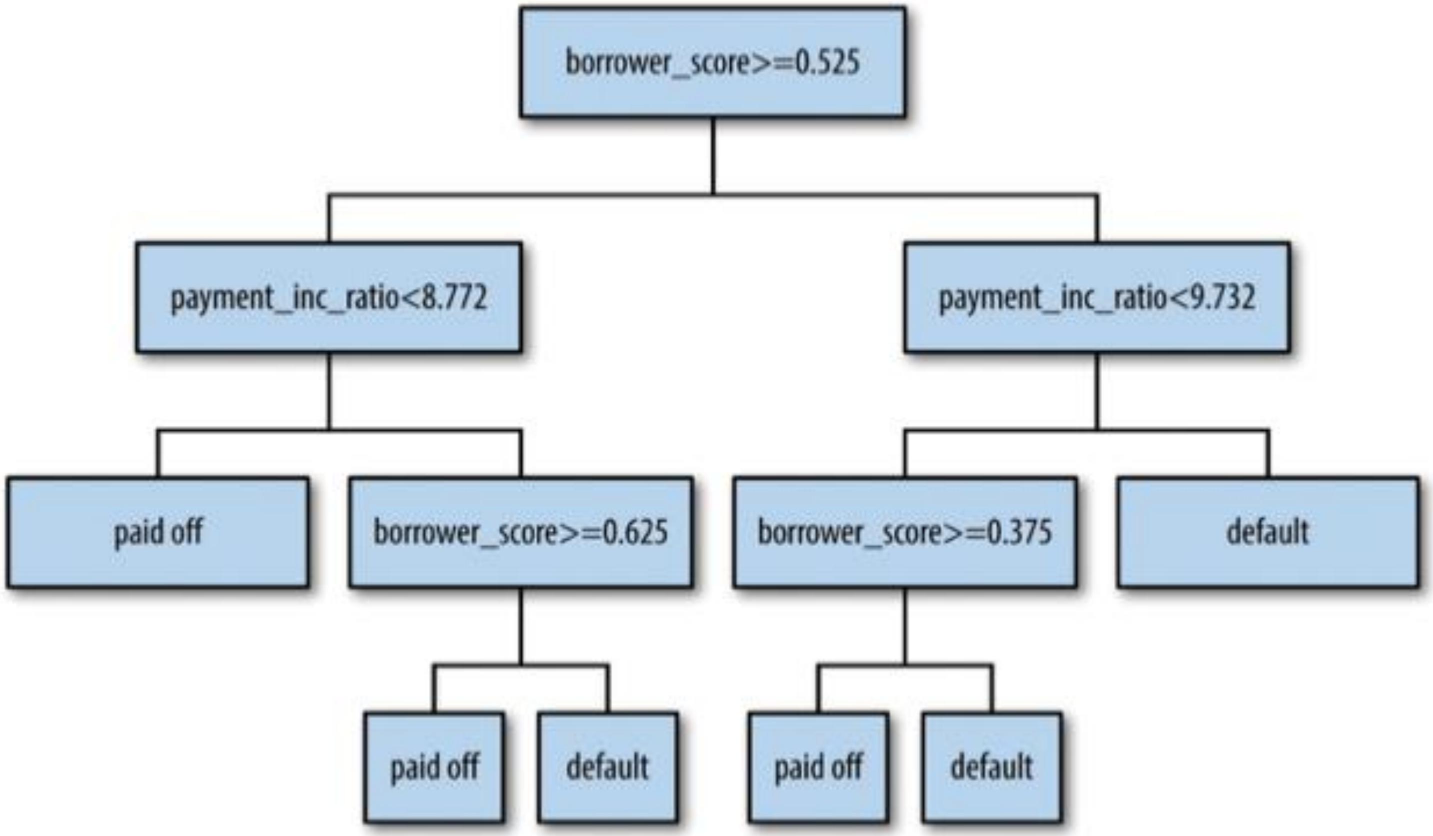
- In pricing a home for sale, a realtor will base the price on similar homes recently sold, known as "comps"
- Using KNN, we can estimate the price of the home by looking at the sale prices of similar homes (e.g., based on location, total square feet, type of structure, lot size, and number of bedrooms and bathrooms)
- For the estimation, the average of the *K*-Nearest Neighbors can be used instead of a majority vote (known as *KNN regression*)
- The estimated price can be added to the existing data (e.g., for subsequent classification)

## Tree models

- Also called decision trees or Classification and Regression Trees (CART)
- Forms the basis for the most widely used and powerful predictive modeling tools in data science for both regression and classification
- A tree model is a set of “if-then-else” rules that are easy to understand and to implement.
- Trees have the ability to discover hidden patterns corresponding to complex interactions in the data (unlike regression and logistic regression)
- Simple tree models can be expressed in terms of predictor relationships that are easily interpretable (unlike KNN or naive Bayes)

## Example tree model

- Two variables: payment_inc_ratio and borrower_score
- Outcome = {paid_off, default}

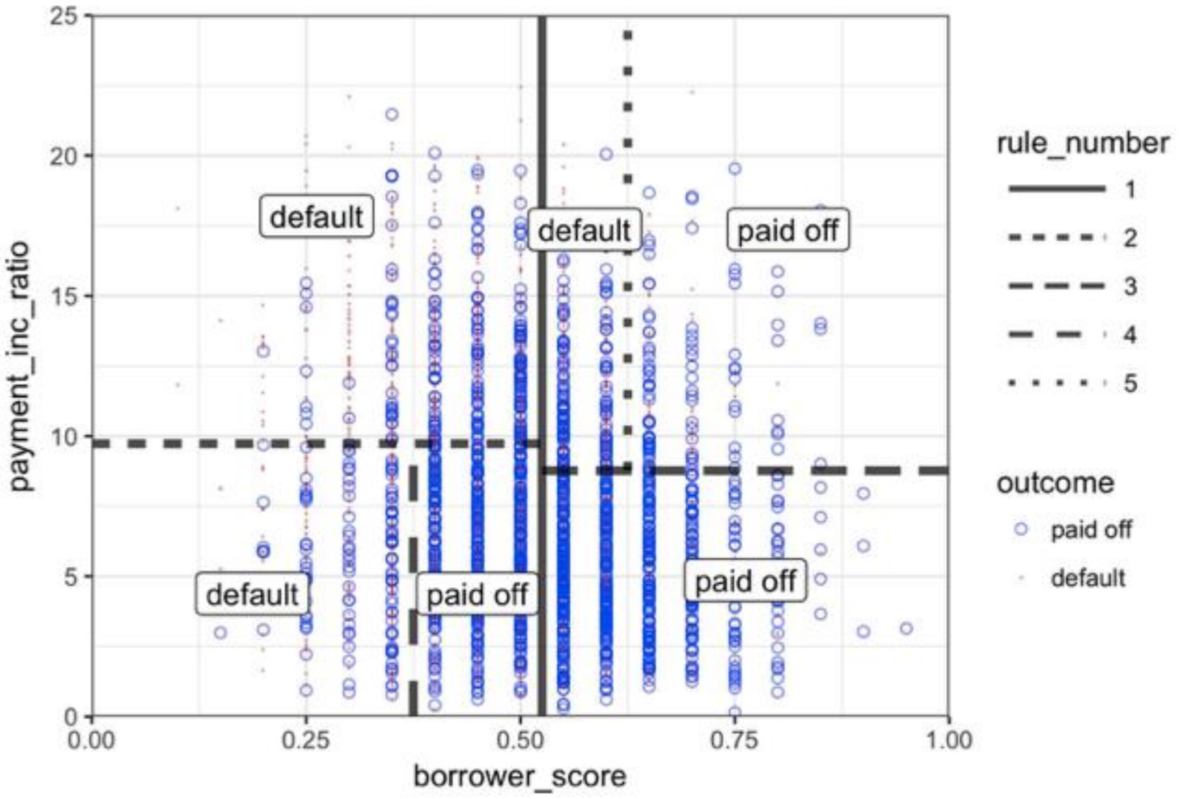


## Constructing decision trees

- Strategy: top-down learning using recursive *divide-and-conquer* process
  - First: select attribute for root node
    Create branch for each possible attribute value
  - Then: split instances into subsets
    One for each branch extending from the node
  - Finally: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances have the same class

# Recursive partitioning

- The data is repeatedly partitioned using predictor values that do the best job of separating the data into relatively homogeneous partitions

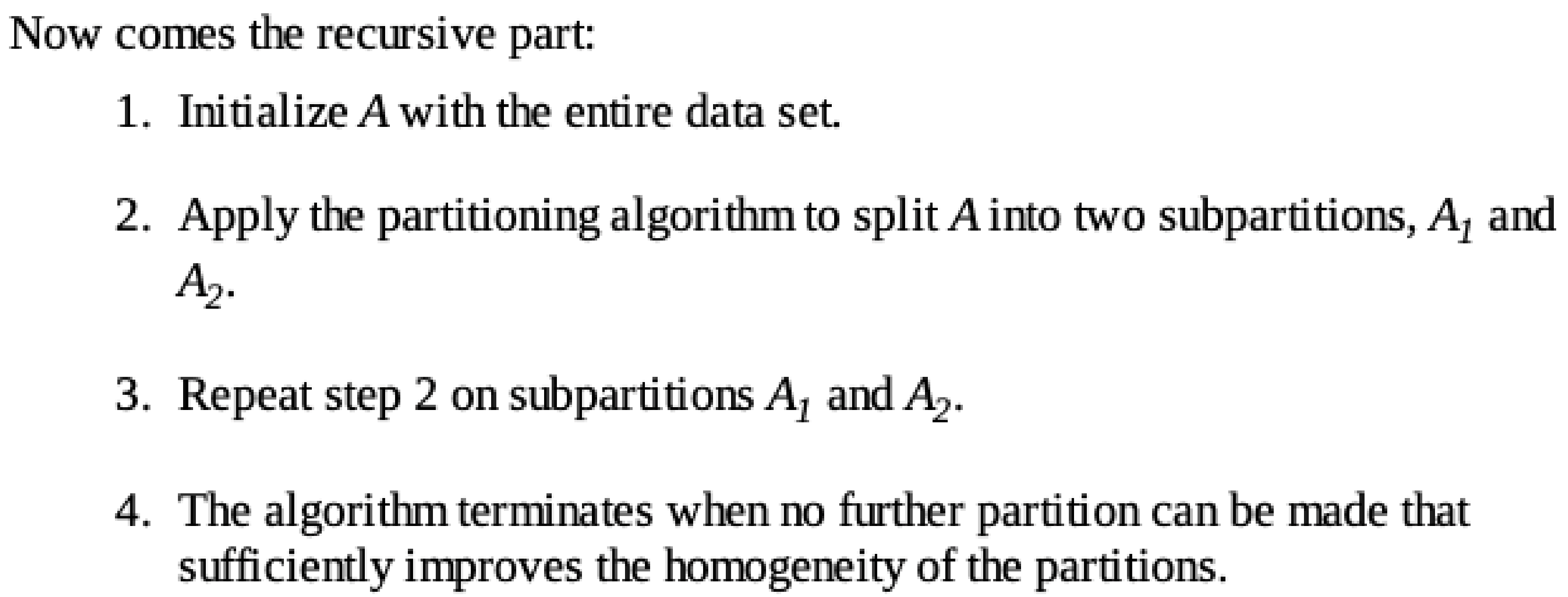


# Recursive partitioning algorithm

Suppose we have a response variable *Y* and a set of *P* predictor variables $X_j$ for $j = 1, \cdots, P$. For a partition *A* of records, recursive partitioning will find the best way to partition *A* into two subpartitions:

1. For each predictor variable $X_j$,
   a. For each value $s_j$ of $X_j$:
      i. Split the records in *A* with $X_j$ values $< s_j$ as one partition, and the remaining records where $X_j \geq s_j$ as another partition.
      ii. Measure the homogeneity of classes within each subpartition of *A*.
   b. Select the value of $s_j$ that produces maximum within-partition homogeneity of class.
2. Select the variable $X_j$ and the split value $s_j$ that produces maximum within-partition homogeneity of class.

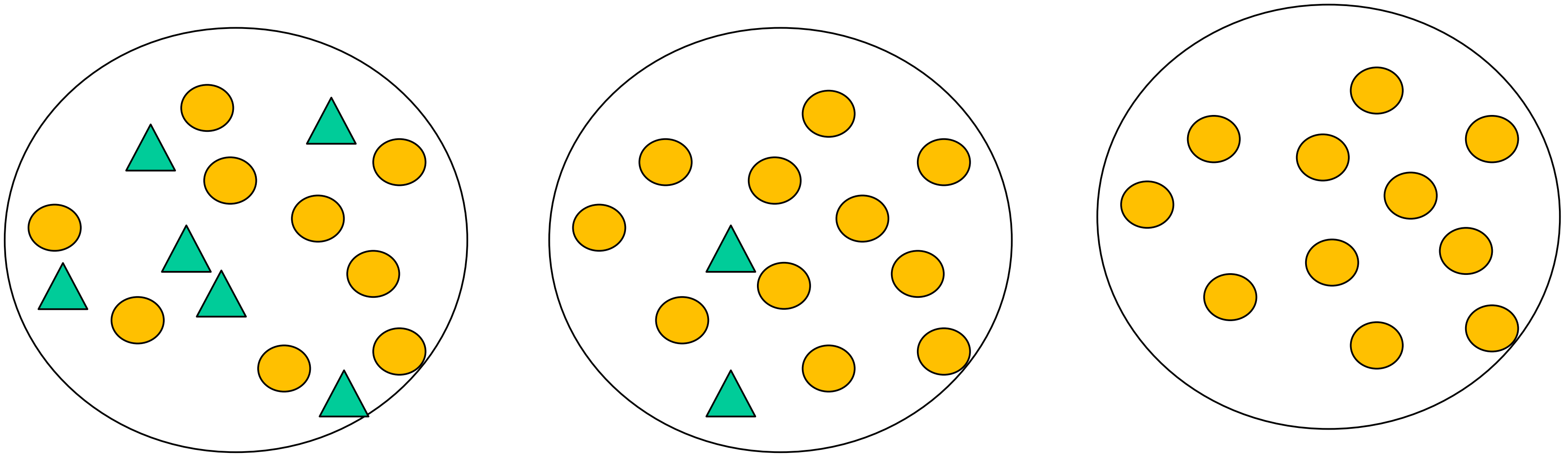
# Recursive partitioning algorithm (2)



Now comes the recursive part:

1. Initialize $A$ with the entire data set.
2. Apply the partitioning algorithm to split $A$ into two subpartitions, $A_1$ and $A_2$.
3. Repeat step 2 on subpartitions $A_1$ and $A_2$.
4. The algorithm terminates when no further partition can be made that sufficiently improves the homogeneity of the partitions.

## Impurity



Highly impure

Less impure

Minimum impurity

## Measuring homogeneity (impurity)

- Tree models recursively create partitions (sets of records), *A*, that predict an outcome of *Y* = 0 or *Y* = 1
- Recursive partitioning algorithms needs a way to measure homogeneity (or equivalently, the impurity of a partition)
- Common measures for impurity: Gini and entropy
- Gini impurity:

$$Gini = 1 - \sum_j p_j^2$$

- Entropy:

$$Entropy = -\sum_j p_j \log_2 p_j$$

- The partition yielding the lowest impurity is selected (and 0 is the most pure score possible)

# Gini impurity example

- Gini impurity:

$$Gini = 1 - \sum_j p_j^2$$

- Partition with 4 red and 0 blue balls:

  Gini = 1-(prob(red)$^2$ + prob(blue)$^2$)

  = 1-(1$^2$+0$^2$) = 0

- Partition with 2 red and 2 blue balls:

  Gini = 1-(prob(red)$^2$ + prob(blue)$^2$)

  = 1-(0.5$^2$+0.5$^2$) = 0.5

- Partition with 3 red and 1 blue balls:

  Gini = 1-(prob(red)$^2$ + prob(blue)$^2$)

  = 1-(0.75$^2$+0.25$^2$) = 0.375

# Entropy example

- Entropy (*E*):

$$Entropy = -\sum_j p_j \log_2 p_j$$

- Partition with 4 red and 0 blue balls:

  $E = -(1 log_2 1) = 0$

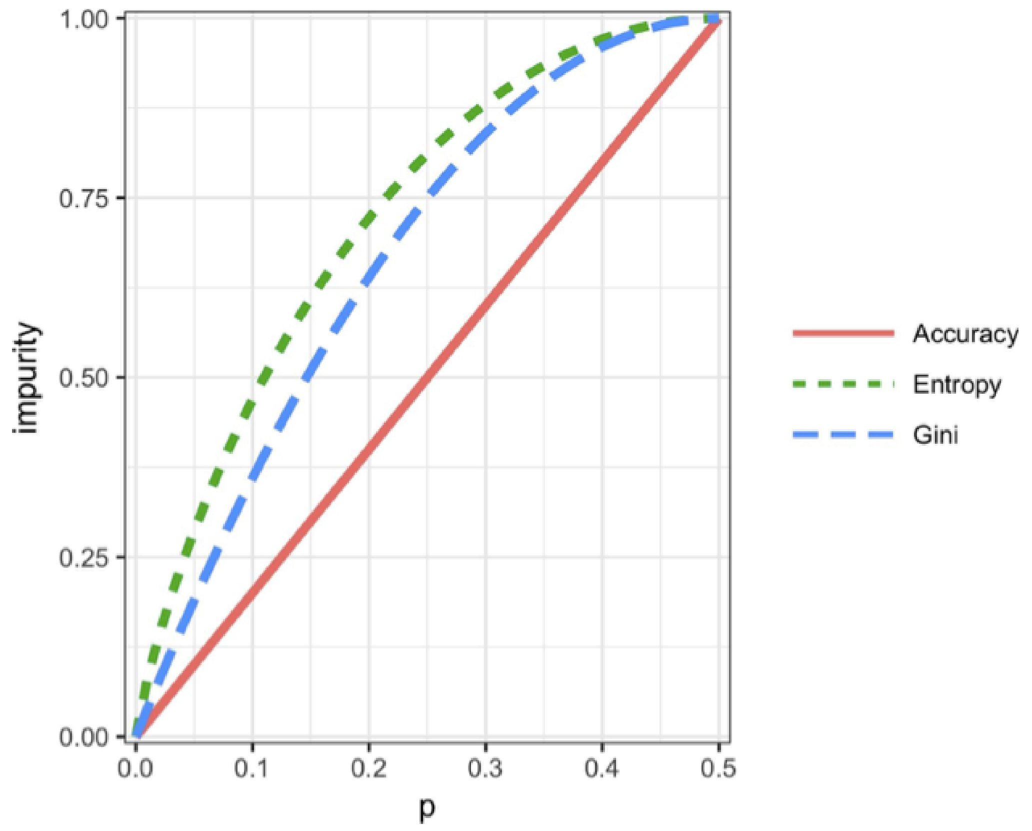
- Partition with 2 red and 2 blue balls:

  $E = -((0.5 log_2 0.5) + (0.5 log_2 0.5)) = 1$

- Partition with 3 red and 1 blue balls:

  $E = -((0.75 log_2 0.75) + (0.25 log_2 0.25)) = 0.81125$

# Gini impurity vs. entropy



- Gini impurity and entropy measures are similar, with entropy giving higher impurity scores

# Other measures for tree splitting

- The "information" measure (denoted *Info(a,b)*) relates to the amount of information obtained by making a decision, defined as:

$$\mathrm{Info}(a,b) = \mathrm{Entropy}(\frac{a}{a+b}, \frac{b}{a+b})$$

$$Entropy = -\sum_{j} p_j \log_2 p_j$$

- Example: *Info([2,3]) = entropy(2/5, 3/5)*

  *= -2/5 log(2/5) – 3/5 log(3/5) = 0.971 (bits)*

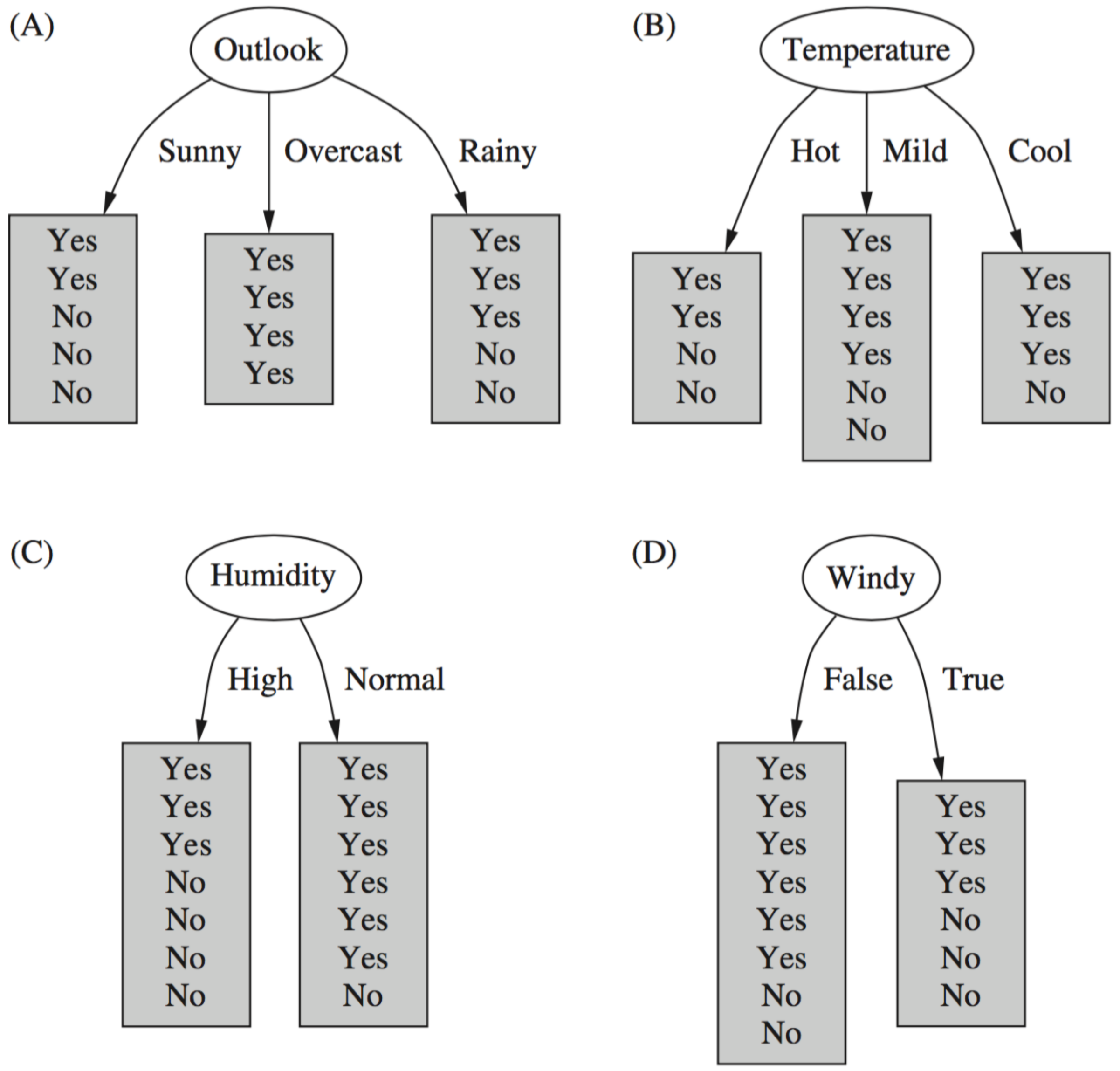
- Information gain: a difference between information before splitting and after splitting
- Gain ratio: modification of the information gain, which takes number and size of branches into account when choosing an attribute

## Example: Weather data

- Four attributes: Outlook, Temp, Humidity, Windy
- Outcome: Play (Y/N)

| Outlook | Temp | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | Hot | High | False | No |
| Sunny | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Rainy | Mild | High | False | Yes |
| Rainy | Cool | Normal | False | Yes |
| Rainy | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Sunny | Mild | High | False | No |
| Sunny | Cool | Normal | False | Yes |
| Rainy | Mild | Normal | False | Yes |
| Sunny | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Rainy | Mild | High | True | No |

# Which attribute to select?



# Calculating information values

- *Outlook = Sunny* :

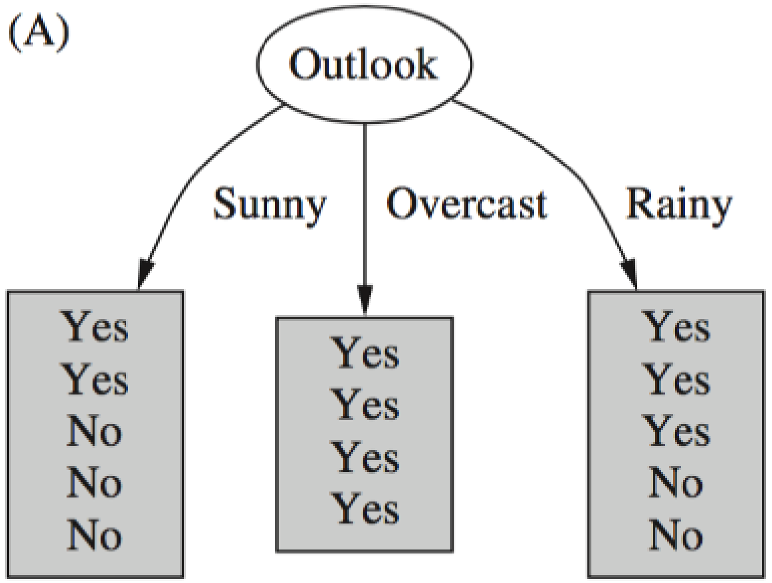
$$\text{Info}([2,3]) = 0.971 \text{ bits}$$

- *Outlook = Overcast* :

$$\text{Info}([4,0]) = 0.0 \text{ bits}$$

- *Outlook = Rainy* :

$$\text{Info}([3,2]) = 0.971 \text{ bits}$$



- Expected information for attribute:

$$\text{Info}([2,3],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971$$
$$= 0.693 \text{ bits}$$

# Calculating information gain

- Information gain = Info before splitting – Info after splitting

  Gain(*Outlook*) = Info([9,5]) – info([2,3],[4,0],[3,2])
  = 0.940 – 0.693
  = 0.247 bits

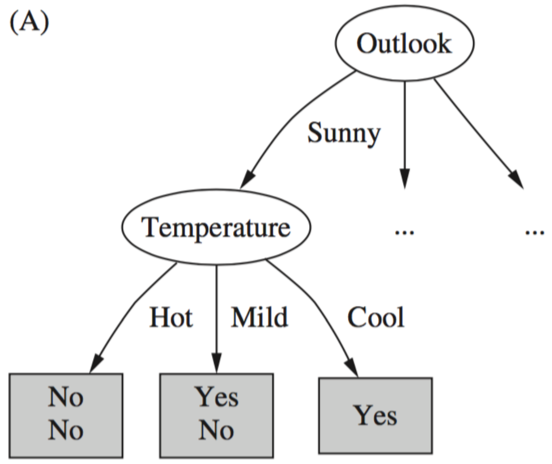
- Information gain for attributes from weather data:

| | |
|---|---|
| Gain(*Outlook*) | = 0.247 bits |
| Gain(*Temperature*) | = 0.029 bits |
| Gain(*Humidity*) | = 0.152 bits |
| Gain(*Windy*) | = 0.048 bits |

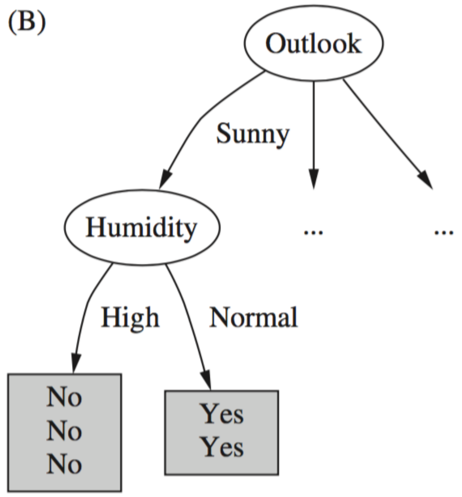
- Best if we split the Outlook feature

# Continuing to split (after choosing Outlook)

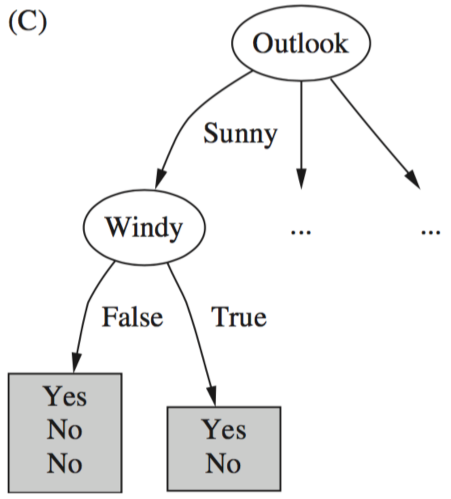
(A)



(B)



(C)



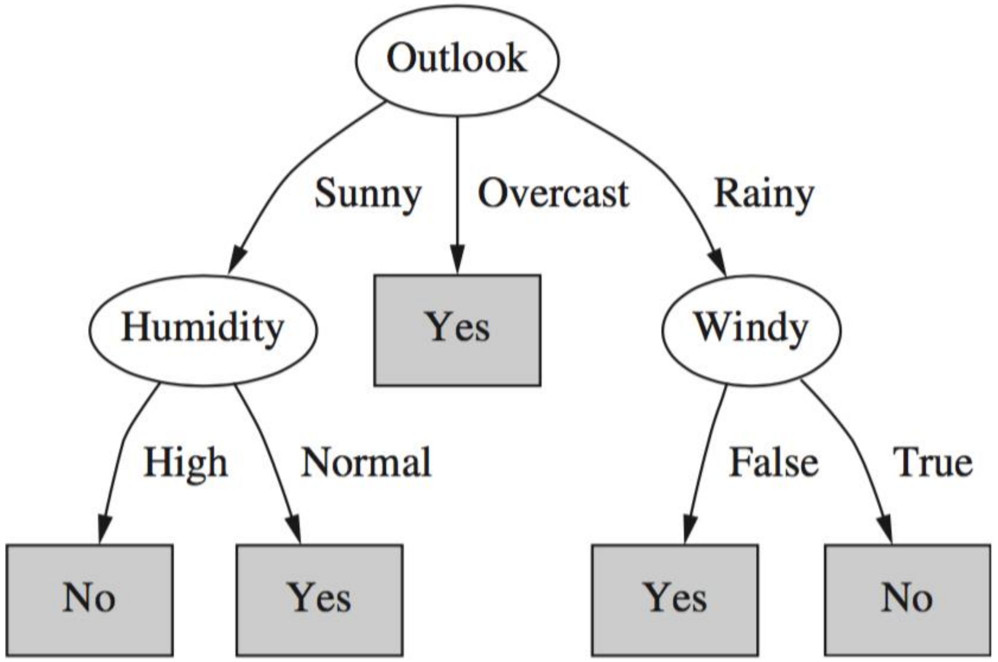
Calculating info gain:

Gain(*Temperature* )= 0.571 bits

Gain(*Humidity* ) = 0.971 bits

Gain(*Windy* ) = 0.020 bits

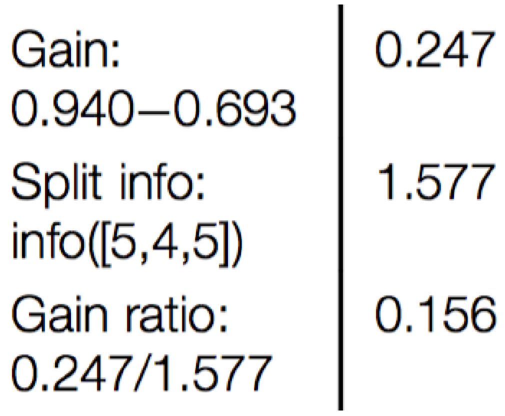
# Final decision tree



- Splitting stops when data cannot be split any further

# Gain ratio

- Sometimes, gain ratio is used, which is a modification of the information gain, which takes number and size of branches into account when choosing an attribute
- Intrinsic information: entropy of the distribution of instances into branches
- Gain ratio is defined as the information gain of the attribute divided by its intrinsic information
- Gain ratio for Outlook = info([2,3],[4,0],[3,2])/Info(5,4,5)
- Value of attribute should decrease as intrinsic information gets larger
- Example (*Outlook* at root node):



| | |
|---|---|
| Gain: 0.940–0.693 | 0.247 |
| Split info: info([5,4,5]) | 1.577 |
| Gain ratio: 0.247/1.577 | 0.156 |

# All gain ratios for Weather data

| Outlook | | Temperature | |
|---|---|---|---|
| Info: | 0.693 | Info: | 0.911 |
| Gain: 0.940-0.693 | 0.247 | Gain: 0.940-0.911 | 0.029 |
| Split info: info([5,4,5]) | 1.577 | Split info: info([4,6,4]) | 1.557 |
| Gain ratio: 0.247/1.577 | 0.157 | Gain ratio: 0.029/1.557 | 0.019 |
| **Humidity** | | **Windy** | |
| Info: | 0.788 | Info: | 0.892 |
| Gain: 0.940-0.788 | 0.152 | Gain: 0.940-0.892 | 0.048 |
| Split info: info([7,7]) | 1.000 | Split info: info([8,6]) | 0.985 |
| Gain ratio: 0.152/1 | 0.152 | Gain ratio: 0.048/0.985 | 0.049 |

## Stopping the tree from growing

- When to stop growing a tree at a stage that will generalize to new data?
- Two common ways to stop splitting:
  1. Avoid splitting a partition if a resulting sub-partition is too small, or if a terminal leaf is too small.
  2. Don’t split a partition if the new partition does not “significantly” reduce the impurity.

## How trees are used

- Two appealing aspects:
  - Tree models provide a visual tool for exploring the data, to gain an idea of what variables are important and how they relate to one another; Trees can capture nonlinear relationships among predictor variables.
  - Tree models provide a set of rules that can be effectively communicated to non-specialists.
- Trees can be used to predict a continuous value (i.e., regression) with the predictive measure of the square root of the mean squared error (RMSE)

## Ensemble approach

- Averaging (or taking majority votes) of multiple models – Wisdom of crowds
- Turns out to be more accurate than just selecting one model
- Two main variants: bagging and boosting
- Simplified version of ensembles:
    1. Develop a predictive model and record the predictions for a given data set
    2. Repeat for multiple models, on the same data
    3. For each record to be predicted, take an average (or a weighted average, or a majority vote) of the predictions

# Bagging

- Bagging = bootstrap aggregating
- Each model is fit to a bootstrap resample
- Algorithm:

1. Initialize $M$, the number of models to be fit, and $n$, the number of records to choose ($n < N$). Set the iteration $m = 1$.
2. Take a bootstrap resample (i.e., with replacement) of $n$ records from the training data to form a subsample $Y_m$ and $\mathbf{X}_m$ (the bag).
3. Train a model using $Y_m$ and $\mathbf{X}_m$ to create a set of decision rules $\hat{f}_m(\mathbf{X})$.
4. Increment the model counter $m = m + 1$. If $m <= M$, go to step 1.

- In the case where $\hat{f}_m$ predicts the probability Y=1, the bagged estimate is given by:

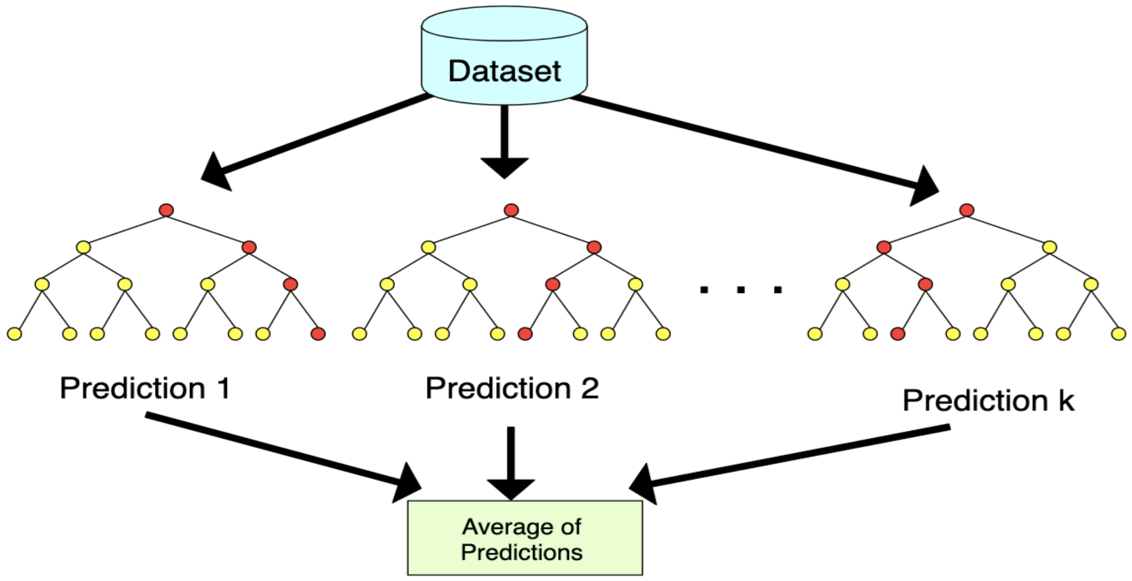
$$\hat{f} = \frac{1}{M}\left(\hat{f}_1(\mathbf{X}) + \hat{f}_2(\mathbf{X}) + \cdots \hat{f}_M(\mathbf{X})\right)$$

# Random forest

- Based on applying bagging to decision trees
- One extension: in addition to sampling the records, the algorithm also samples the variables

Procedure (simplified):

1. Sample from the training data
2. Train trees based on the samples, and prediction is recorded for samples
3. Predict by averaging all predictions



# Random forest algorithm

1. Take a bootstrap (with replacement) subsample from the *records*.

2. For the first split, sample $p < P$ *variables* at random without replacement.

3. For each of the sampled variables $X_{j(1)}, X_{j(2)}, \ldots, X_{j(p)}$, apply the splitting algorithm:

   a. For each value $s_{j(k)}$ of $X_{j(k)}$:

      i. Split the records in partition *A* with $X_{j(k)} < s_{j(k)}$ as one partition, and the remaining records where $X_{j(k)} \geq s_{j(k)}$ as another partition.

      ii. Measure the homogeneity of classes within each subpartition of *A*.

   b. Select the value of $s_{j(k)}$ that produces maximum within-partition homogeneity of class.

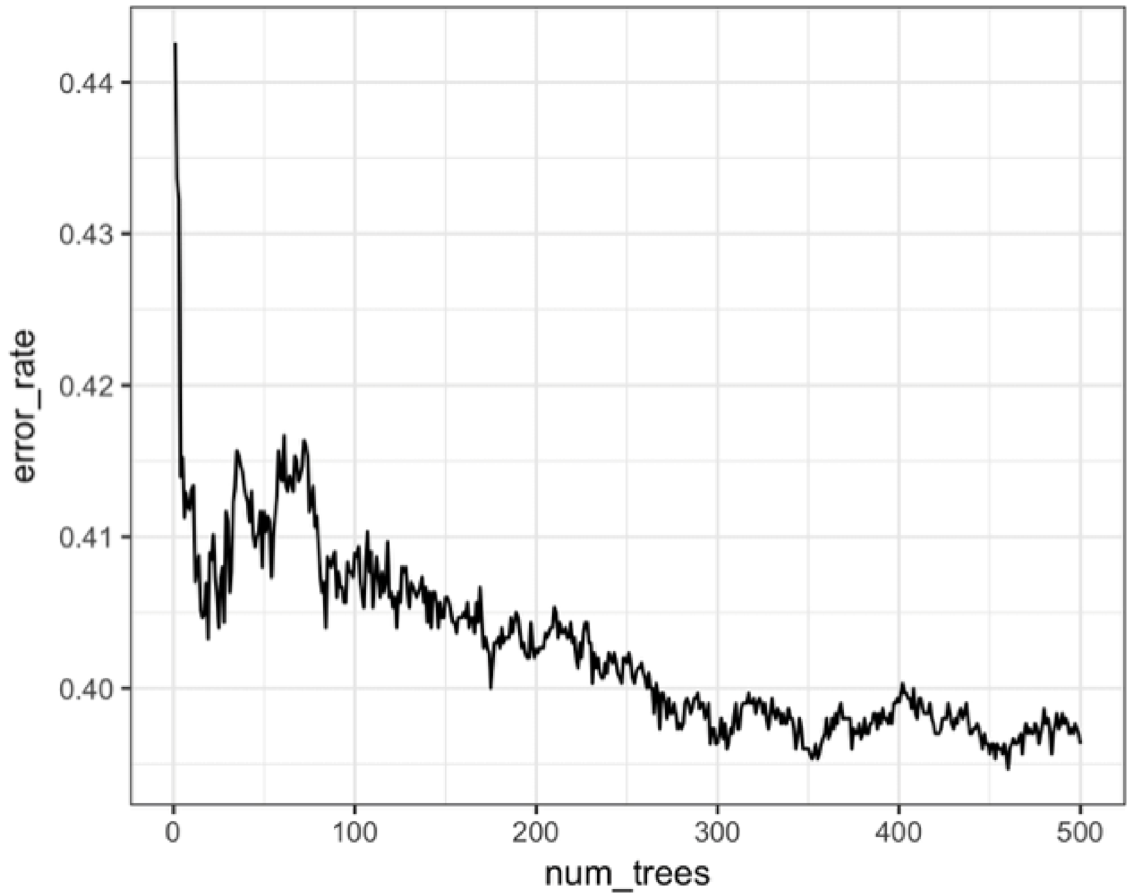
4. Select the variable $X_{j(k)}$ and the split value $s_{j(k)}$ that produces maximum within-partition homogeneity of class.

5. Proceed to the next split and repeat the previous steps, starting with step 2.

6. Continue with additional splits following the same procedure until the tree is grown.

7. Go back to step 1, take another bootstrap subsample, and start the process over again.

# Example: how many trees?

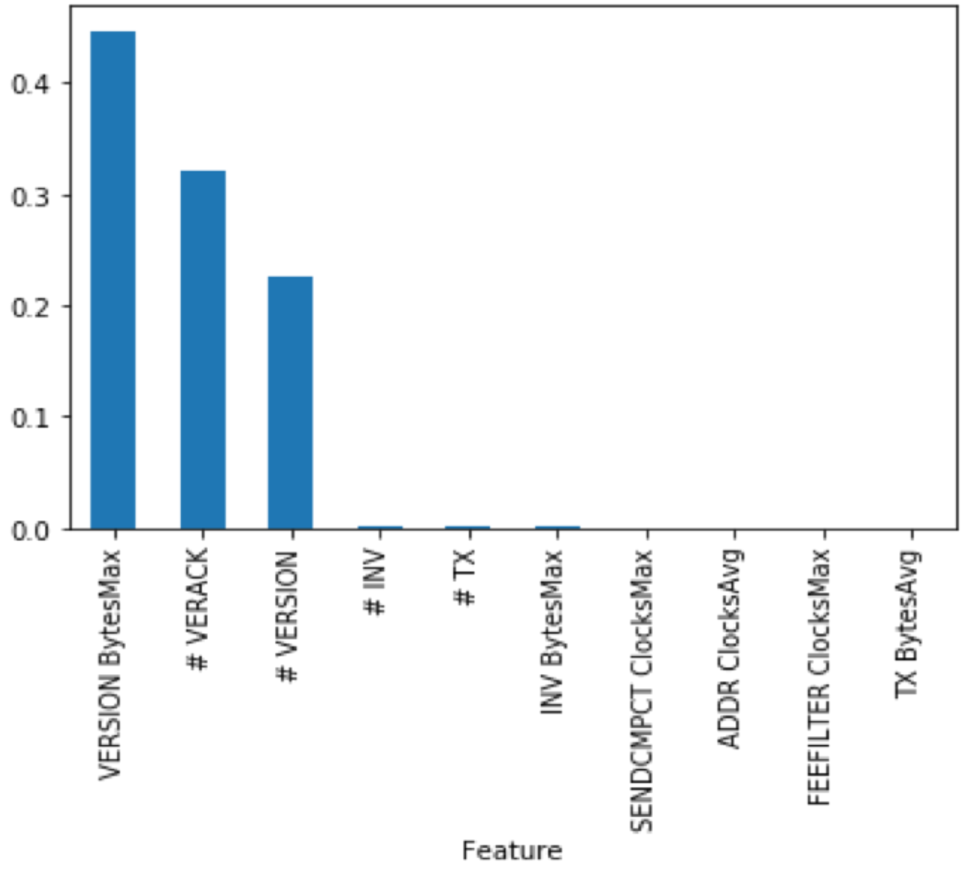


- Improvement in accuracy of the random forest with the addition of more trees

# More about random forest

- Random forest produces more accurate predictions than a simple tree, but the simple tree's intuitive decision rules are lost

- Random forest has the ability to automatically determine which predictors are important

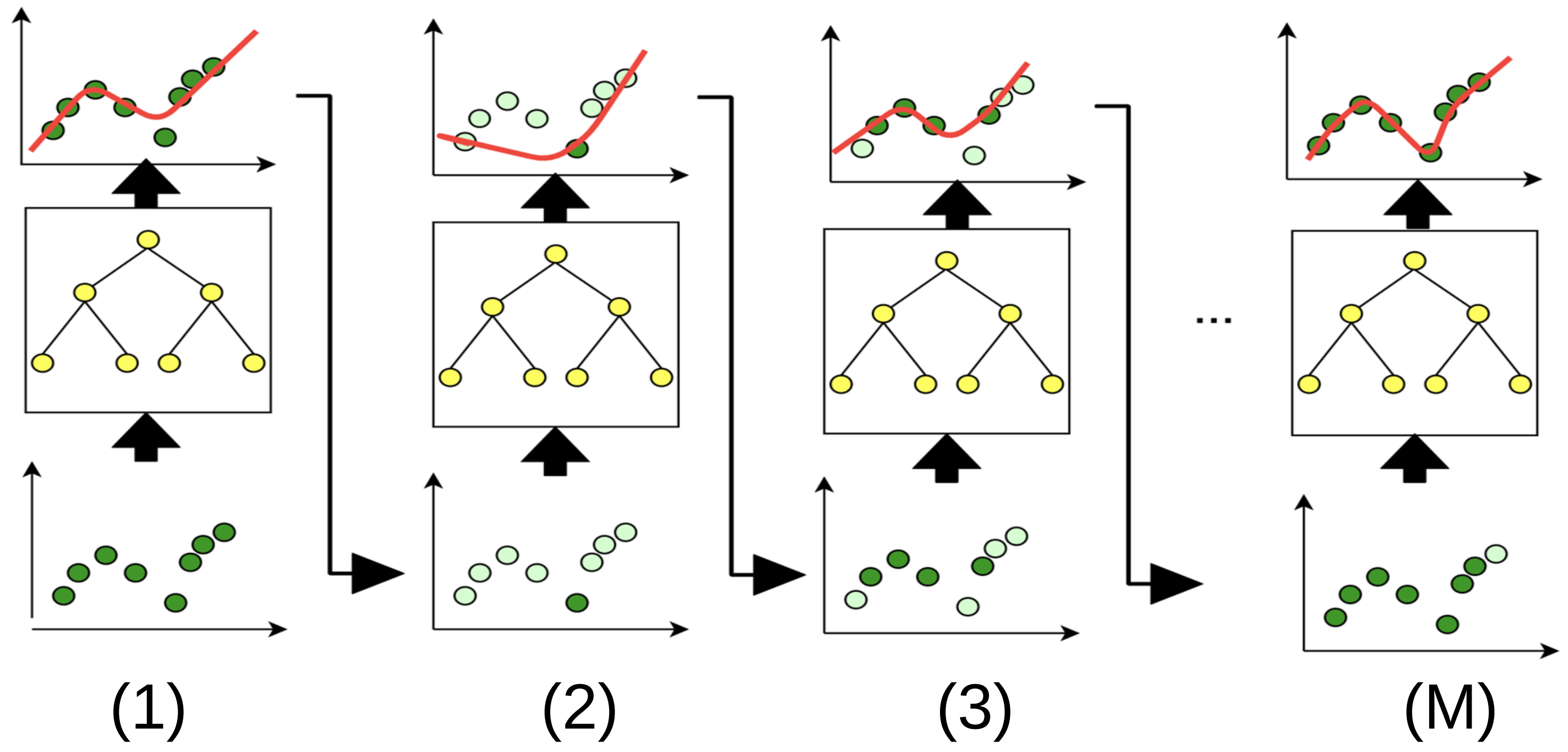


## Boosting

- Like bagging, boosting is most commonly used with decision trees
- While bagging can be done with relatively little tuning, boosting requires much greater care in its application
- Boosting takes the concept of the residuals (the error term in linear regression)
- Variants: Adaboost, gradient boosting, stochastic gradient boosting

# Boosting algorithm (simplified)

1. Set equal weight to each observations
2. If there is error in step 1, focus on observations had error and implement learning model
3. Repeat step 2 until the limit of of base learning algorithm is reached or higher accuracy is achieved.



# Adaboost algorithm

1. Initialize $M$, the maximum number of models to be fit, and set the iteration counter $m = 1$. Initialize the observation weights $w_i = 1 / N$ for $i = 1, 2, \ldots, N$. Initialize the ensemble model $\hat{F}_0 = 0$.

2. Train a model using $\hat{f}_m$ using the observation weights $w_1, w_2, \ldots, w_N$ that minimizes the weighted error $e_m$ defined by summing the weights for the misclassified observations.

3. Add the model to the ensemble: $\hat{F}_m = \hat{F}_{m-1} + \alpha_m \hat{f}_m$ where $\alpha_m = \frac{\log 1 - e_m}{e_m}$.

4. Update the weights $w_1, w_2, \ldots, w_N$ so that the weights are increased for the observations that were misclassfied. The size of the increase depends on $\alpha_m$ with larger values of $\alpha_m$ leading to bigger weights.

5. Increment the model counter $m = m + 1$. If $m \leq M$, go to step 1.

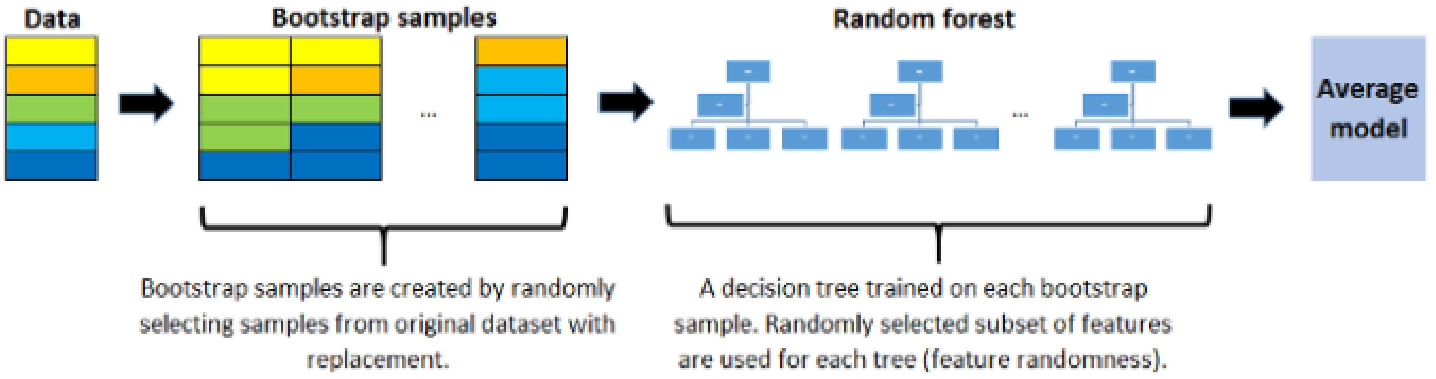
- The boosted estimated is given by:

$$\hat{F} = \alpha_1 \hat{f}_1 + \alpha_2 \hat{f}_2 + \cdots + \alpha_M \hat{f}_M$$

- The factor $\alpha_m$ ensures that models with lower error ($e_m$) have a bigger weight: $\alpha_m = \frac{\log 1 - em}{e_m}$

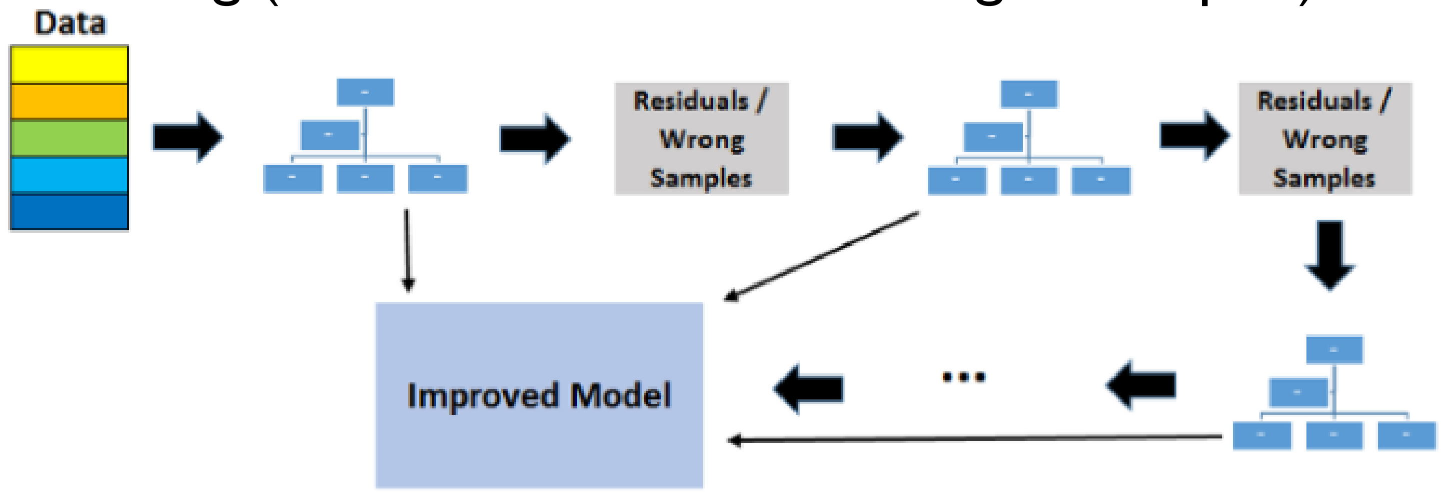
# Random forests vs. gradient boosting

## Random forests


Data
Bootstrap samples
...
Random forest
...
Average model
Bootstrap samples are created by randomly selecting samples from original dataset with replacement.
A decision tree trained on each bootstrap sample. Randomly selected subset of features are used for each tree (feature randomness).

How random forests work

## Gradient boosting (one of well-known boosting techniques)



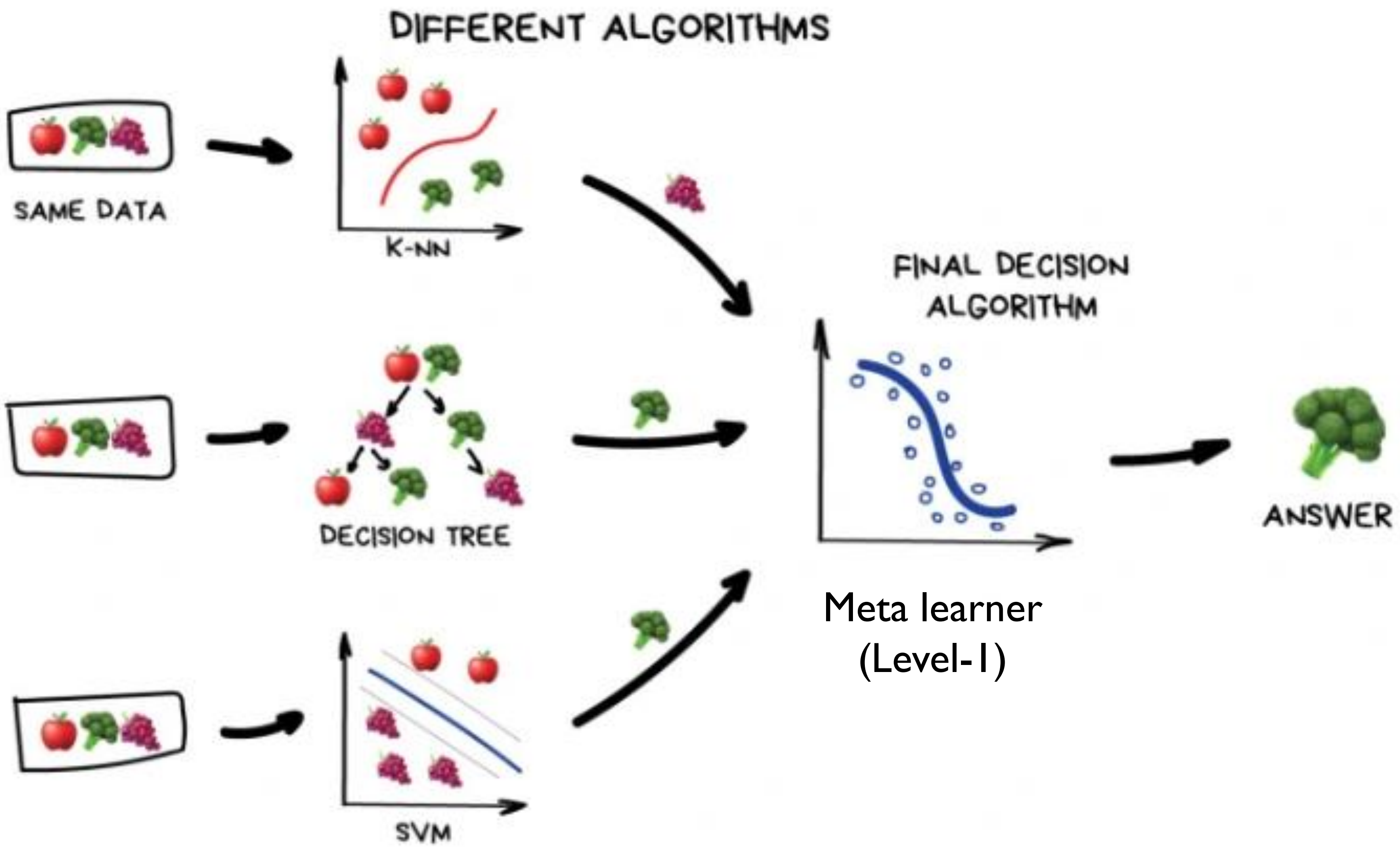
How gradient boosted decision trees work

Source: ai.plainenglish.io

# Stacking

- Question: how to build a *heterogeneous* ensemble consisting of different types of models (e.g., decision tree and neural network)
  - Problem: models can be vastly different in accuracy
- Idea: to combine predictions of base learners, do *not* just vote, instead, use *meta learner*
  - In stacking, the base learners are also called *level-0 models*
  - Meta learner is called *level-1 model*
  - Predictions of base learners are input to meta learner
- Base learners are usually different learning schemes
- Stacking for classification:
  - An instance first fed into the level-0 models
  - The level-0 guesses are fed into the level-1 model
  - The level-1 model combines and makes the final prediction

# Stacking example



Meta learner
(Level-1)

Base learners
(Level-0)

Image from mksaad.wordpress.com

## Summary

- *K*-Nearest Neighbors (KNN)
- Decision trees
- Ensemble learning: bagging, boosting, stacking