GBDi
Government Big Data Institute
สถาบันส่งเสริมการวิเคราะห์และบริหารข้อมูลขนาดใหญ่ภาครัฐ (สวข.)

depa

Introduction to Supervised Learning:

Basic Classification Models

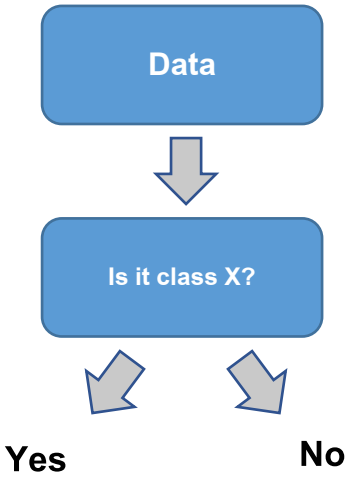
Thanakorn Thaminkaew

Data Scientist

Original materials by Dr. Duangjai Jitkongchuen, Papoj Thamjaroenporn, and Patipan Prasertsom

Classification

- In machine learning, classification problem is a problem of predicting the correct label (class) that a given data belongs to.
- There are many types of classification algorithms, such as
  - Regression-based classifiers: ex. logistic regressions
  - Tree-based classifiers: ex. decision tree, random forest, gradient tree boosting
  - Probability-based classifiers: ex. Bayes classifier
  - Similarity-based classifier: ex. k-Nearest Neighbor classifier
  - etc.
- Today, we will touch on some of the well-known classifiers



k-Nearest Neighbors classifier (kNN)

RECAP

k-Nearest Neighbors classifier (kNN)

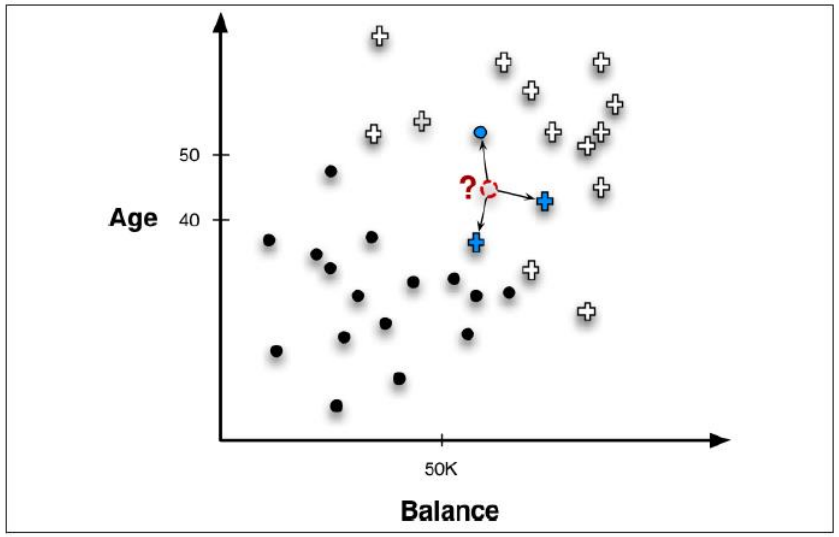
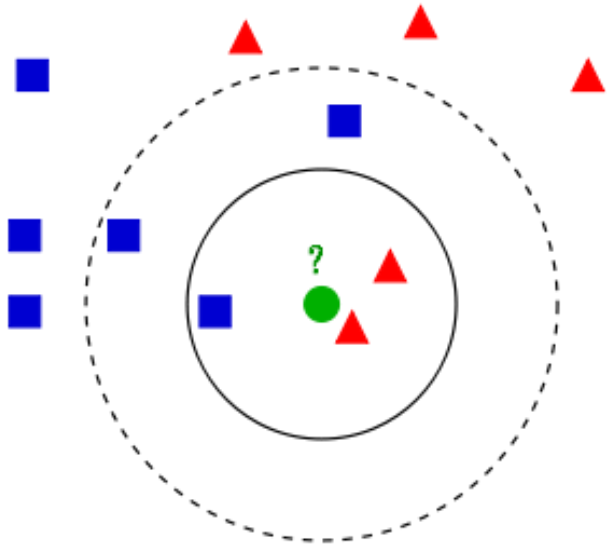


Figure 6-2. Nearest neighbor classification. The point to be classified, labeled with a question mark, would be classified + because the majority of its nearest (three) neighbors are +.

- Observes k closest data point and decide the class of data points by voting.
- Usually, k is chosen as an odd number (why?)



- The choice of ***k*** matters!
- Different number of k can result in different predictions
- Feature scale matters!

Figures from Data Science for Business by Foster Provost & Tom Fawcett and Antti Ajanki AnAj (https://commons.wikimedia.org/wiki/File:Dot_Product.svg)

k-Nearest Neighbors classifier (kNN)

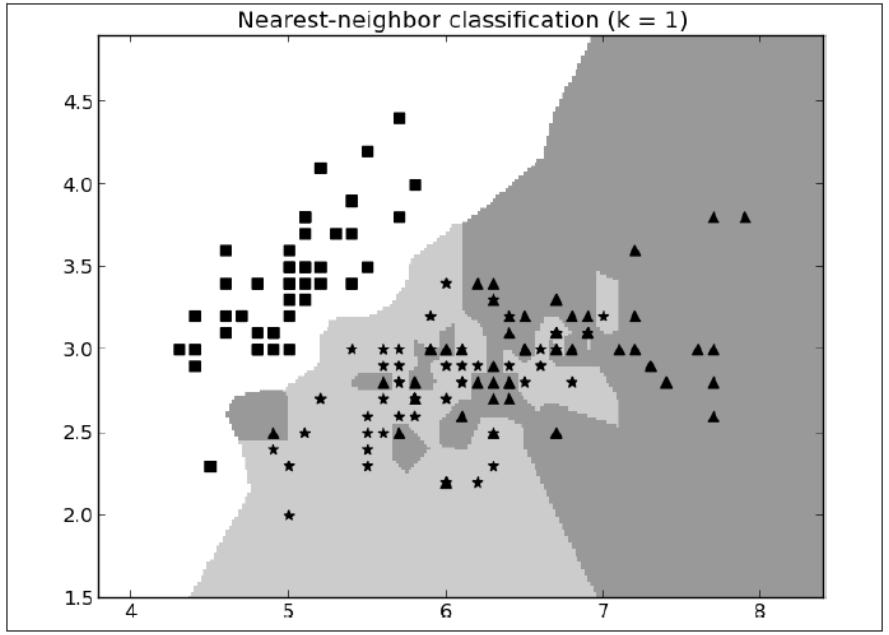


Figure 6-4. Classification boundaries created on a three-class problem created by 1-NN (single nearest neighbor).

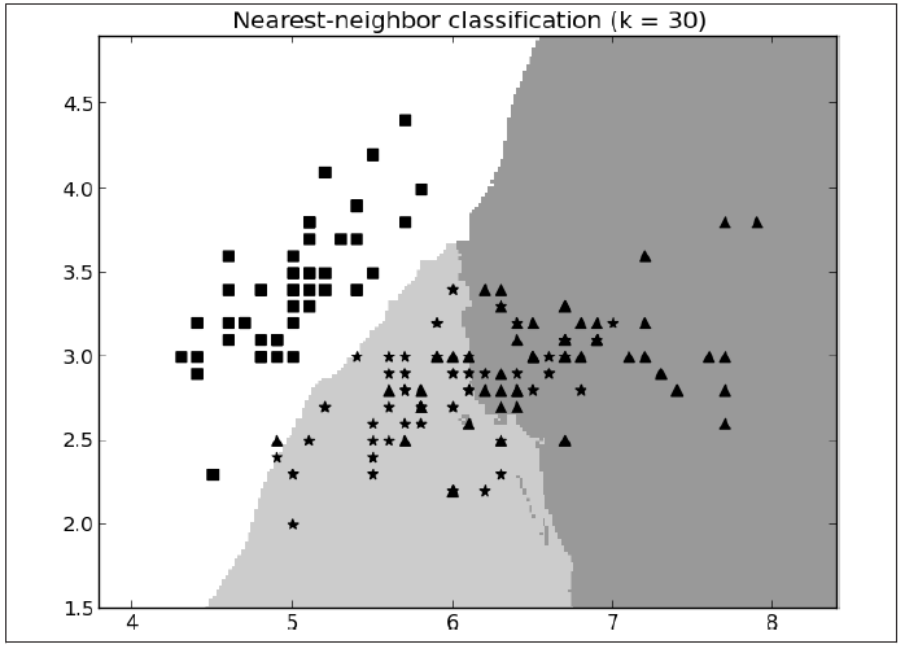


Figure 6-5. Classification boundaries created on a three-class problem created by 30-NN (averaging 30 nearest neighbors).

kNN models with a small *k*

- A finer granularity separation boundaries
- More susceptible to the presents of outlier.

kNN models with a large *k*

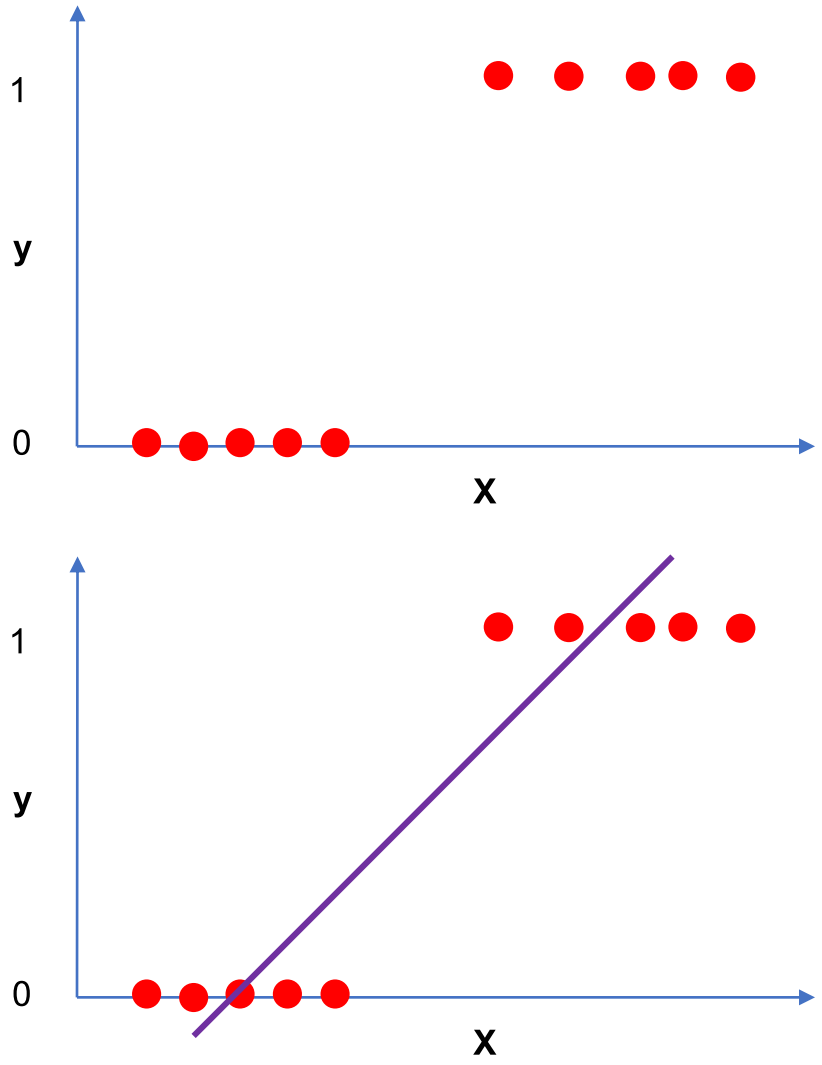
- More tolerant to noise
- Smooth but coarser separation boundaries

Figures from Data Science for Business by Foster Provost & Tom Fawcett

Logistic Regression

Logistic Regression

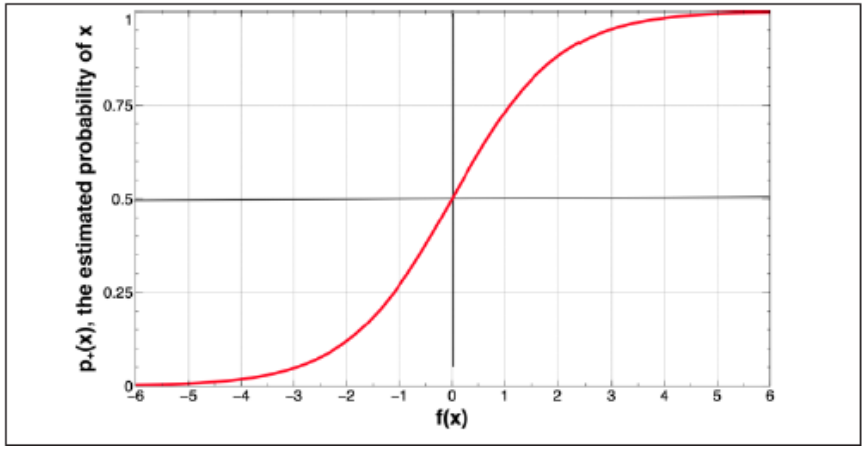
- Logistic regression is a binary (0/1) classification model, meaning it predict whether a data is of a certain class or not.
- **Motivation**: Let's say our data has classes (y=0 and y=1) as shown on the right. Can we use linear regression to predict them?
- While this looks like it can *somewhat* do the job, it is not appropriate
- We do not want the predicted output to be infinitely high/low. We simply want it to tell us whether the data is of our target type .
- Even better if it tells us how *confident* it is that the data is of our target type.



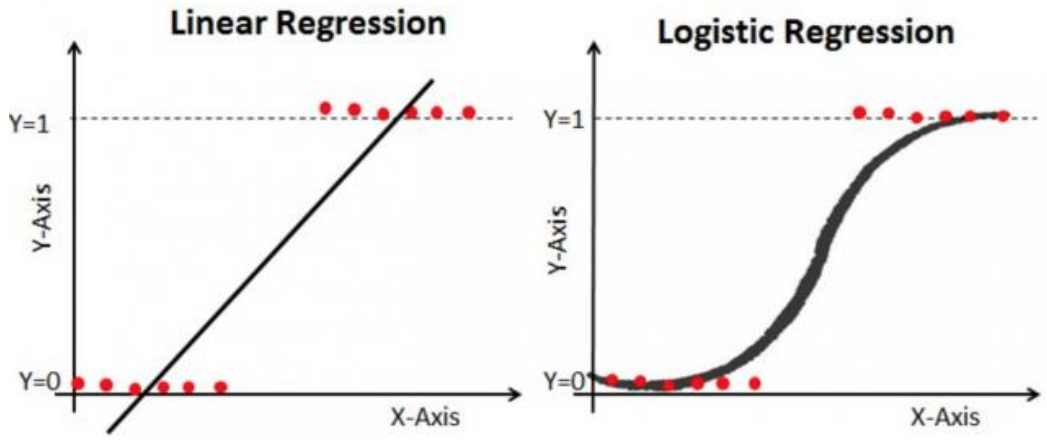
Logistic Regression

- A *sigmoid* function has a value ranging from 0 to 1 for its entire input range

$$y = \frac{1}{1 + e^{-x}}$$



- This makes it's a more appropriate choice to use compared to using purely linear equation



Figures from ODSC - Open Data Science (https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7) and Data Science for Business by Foster Provost & Tom Fawcett

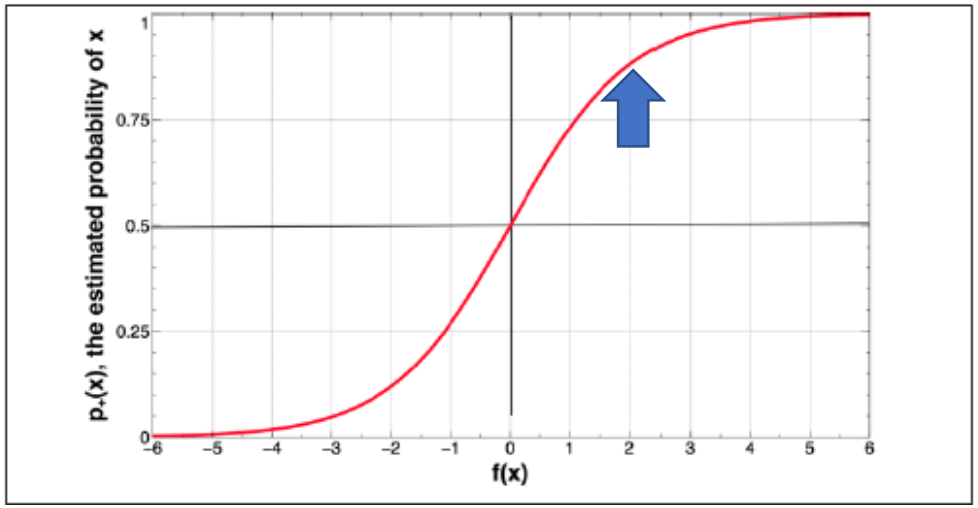
Logistic Regression

- Specifically, a logistic regression model uses a sigmoid function in conjunction with the linear equation.

$$y = \frac{1}{1 + e^{-f(x)}}$$

where

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$



- Fitting a logistic equation is akin to fitting a linear equation model which will then predict where we are on the input of the sigmoid function.

- This is similar to predicting how confidence we are of our prediction

Figure from ODSC - Open Data Science (https://medium.com/@ODSC/logistic-regression-with-python-ede39f8573c7)

Logistic Loss Function

- Since the goal for a logistic regression is to classify the data, its loss function needs to minimize the misclassification
  - Needs to have high value when predict incorrectly and low when predict correctly
- For example, we want a function that behaves like
  - $-\log\left(\hat{y}^{(i)}\right)$ when $y^{(i)} = 1$
  - $-\log\left(1-\hat{y}^{(i)}\right)$ when $y^{(i)} = 0$

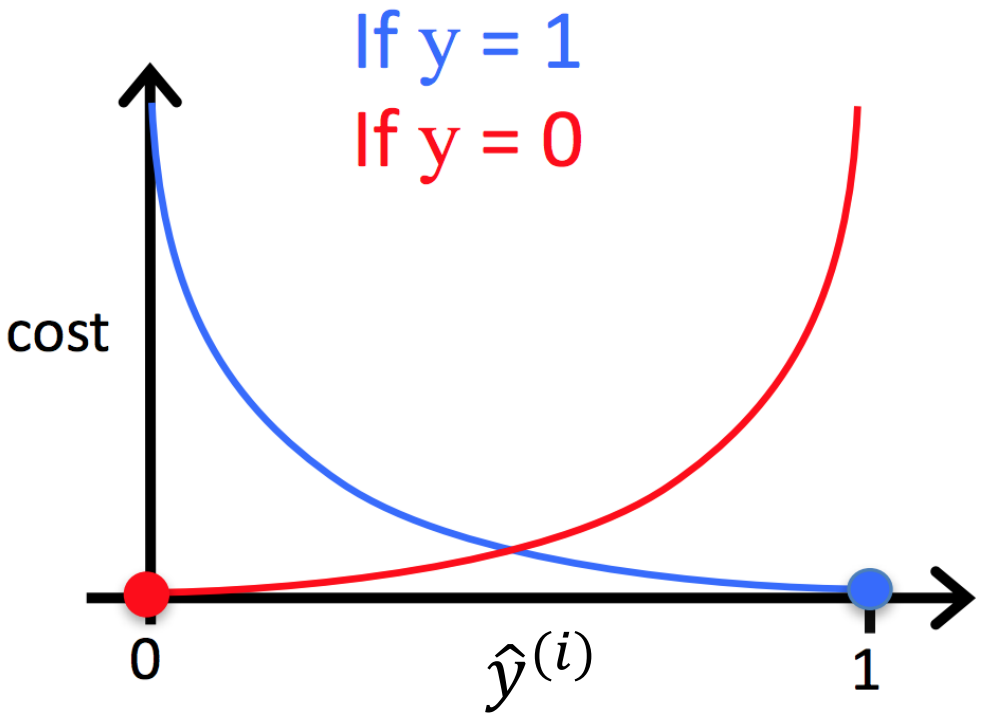


Figure from TwinPenguins (https://datascience.stackexchange.com/questions/40982/logistic-regression-cost-function)

Logistic Loss Function

- The logistic loss function is

$$Logistic\ Loss\ = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right)$$

where

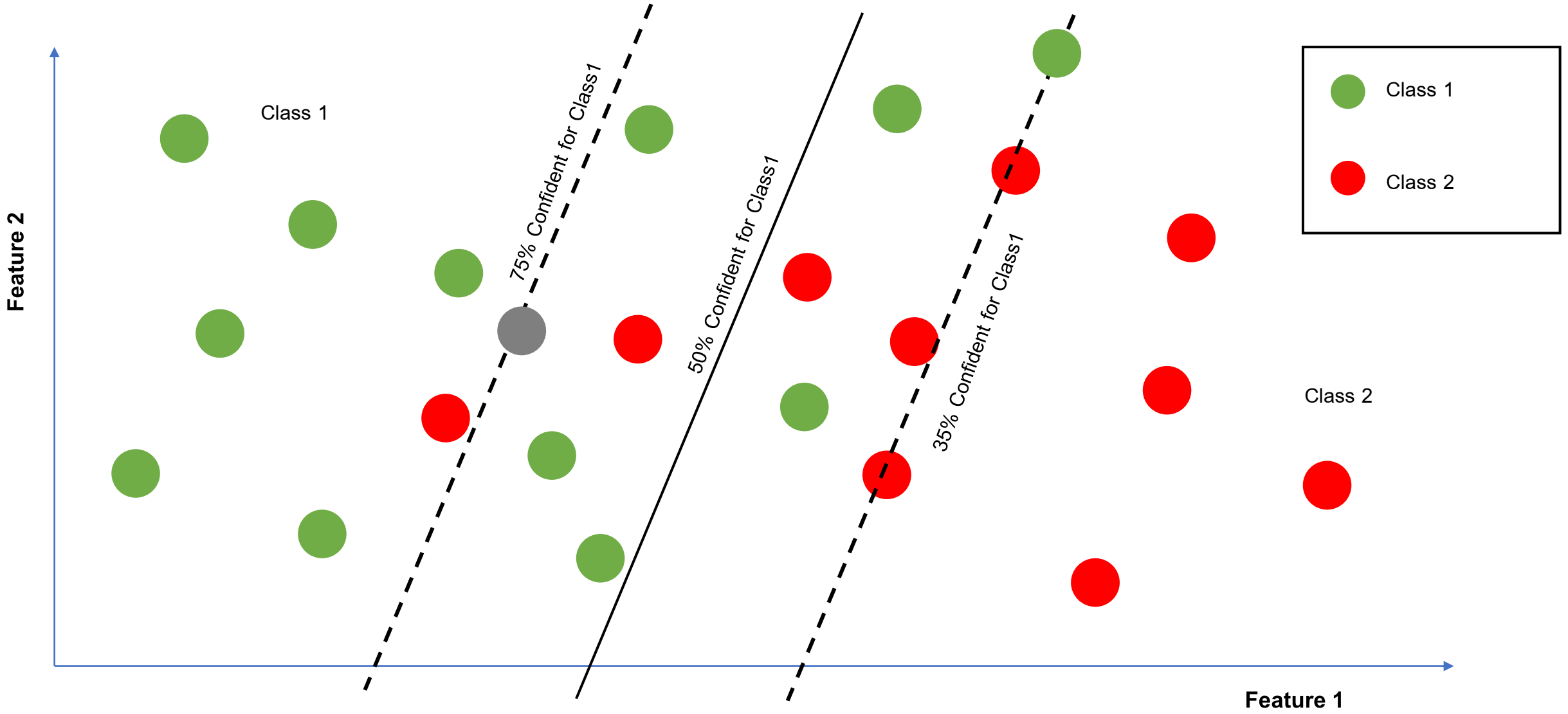
$$\hat{y}^{(i)} = \frac{1}{1+e^{-f\left(x^{(i)}\right)}}$$

and

$$f\left(x^{(i)}\right) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_m x_m^{(i)}$$

- Optimization technique like gradient descent can then be applied to find the optimal set of parameters
- Also note that Logistic regression has linear classification boundary

Logistic Regression: Separation Boundary and Threshold



Logistic Regression

- One of the biggest advantage of logistic regression model is its explainability
- Similar to linear regression, the coefficients obtained for each feature in the final equation of the logistic regression are indicators of the direction (positive/negative) and strength of the relationships of each feature and the prediction target
    - Furthermore, this allows us to explain the expected consequence when changing the value of a feature.
- The coefficients obtained can also be used to perform feature selection by weeding out features with small coefficients.
- Also just like linear regression, regularization techniques can also be applied to improve the model's performance (in fact, regularization is applied by default in sklearn library)

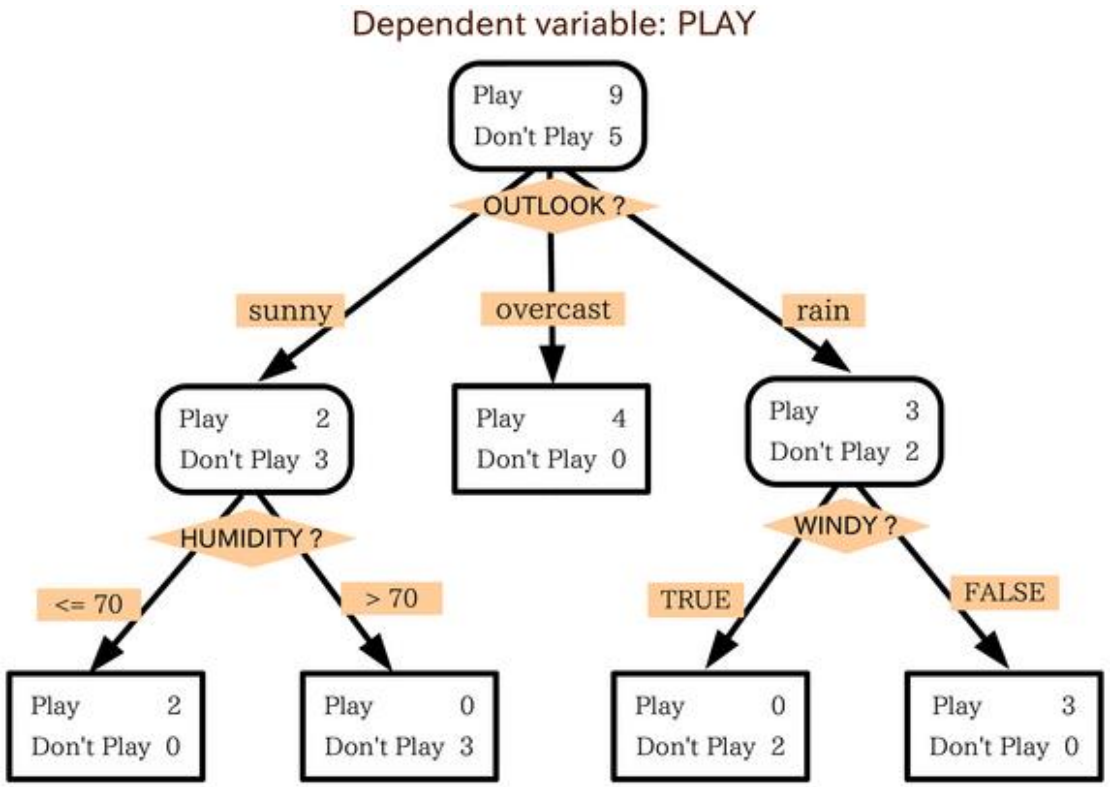
Python Tutorial

- Colab!
- 3.1 Logistic Regression (titanic)
- 3.2 Logistic Regression (wine)

Decision Tree

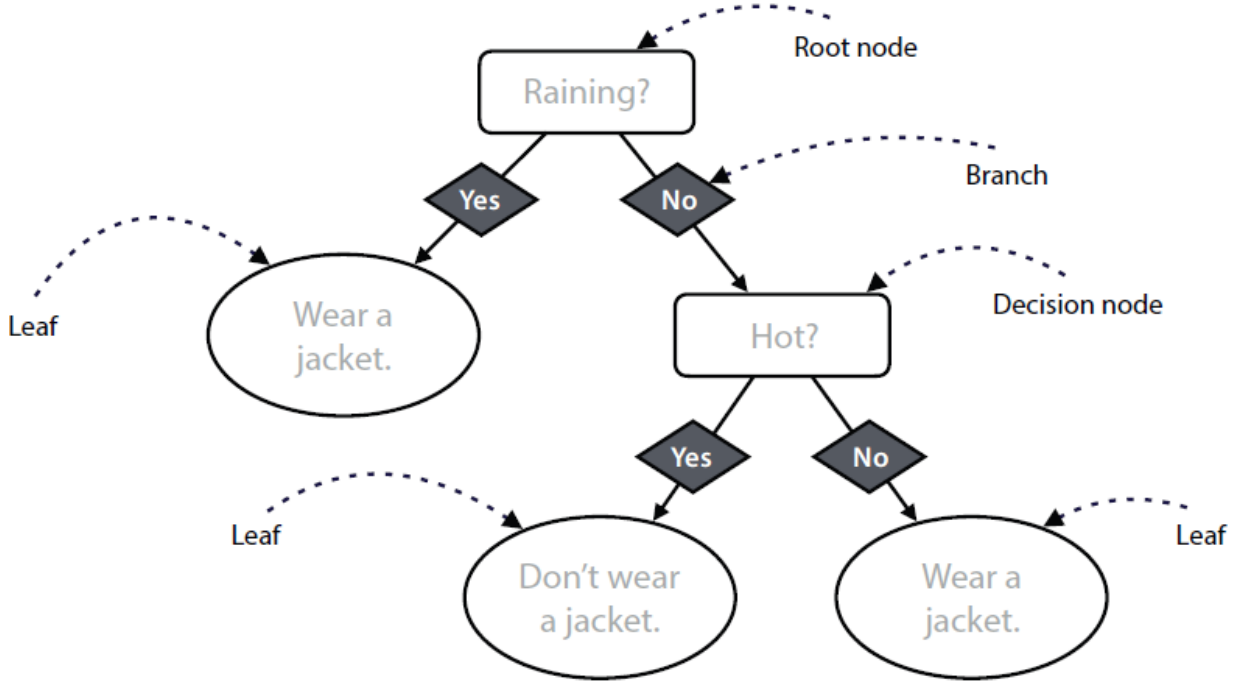
Decision Tree

- A decision tree is a tree-like structure where the internal nodes represents series of “decisions” which eventually leads to an outcome (represented by the terminal nodes)
- Each node within a tree (apart from the final level) is a *segmentation criteria*.
- The leaf node (final level) represent the result of the classification. The label (classification decision) for data in each leaf node is made based on the content of the node’s data



Decision Tree

- **decision tree** A machine learning model based on yes-or-no questions and represented by a binary tree. The tree has a root node, decision nodes, leaf nodes, and branches.
- **root node** The topmost node of the tree. It contains the first yes-or-no question. For convenience, we refer to it as the root.
- **decision node** Each yes-or-no question in our model is represented by a decision node, with two branches emanating from it
- **leaf node** A node that has no branches emanating from it. These represent the decisions we make after traversing the tree. For convenience, we refer to them as leaves.
- **branch** The two edges emanating from each decision node, corresponding to the “yes” and “no” answers to the question in the node.
- **depth** The number of levels in the decision tree. Alternatively, it is the number of branches on the longest path from the root node to a leaf node.



Decision Tree Example: Data and Attributes

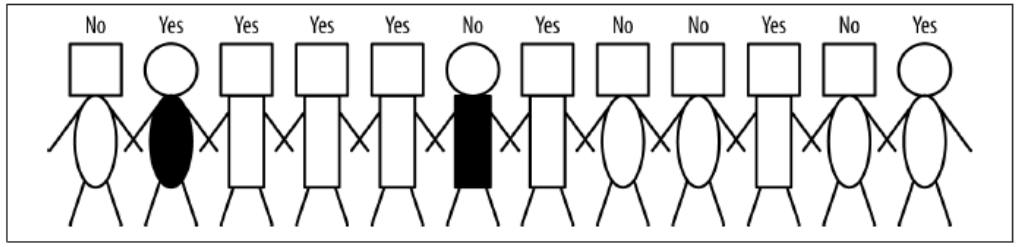


Figure 3-2. A set of people to be classified. The label over each head represents the value of the target variable (write-off or not). Colors and shapes represent different predictor attributes.

What are attributes that describes this data (people)?

Attributes:

- head-shape: square, circular
- body-shape: rectangular, oval
- body-color: gray, white

Target variable:

- write-off: Yes, No

What is the most informative attribute that can be used to segment this data (people)?

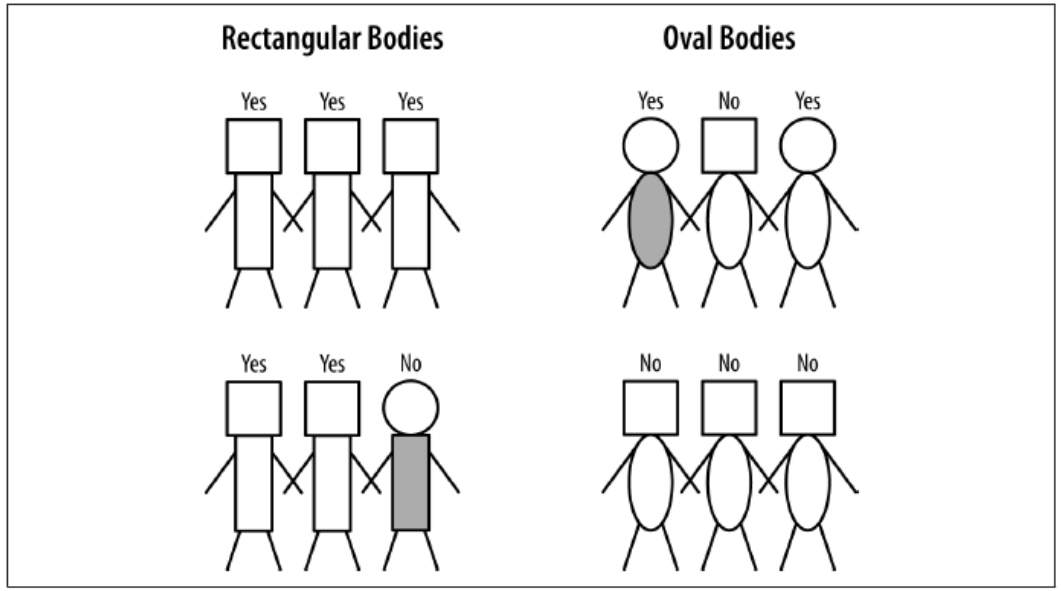


Figure 3-11. First partitioning: splitting on body shape (rectangular versus oval).

Decision Tree Example: Further Segmentation

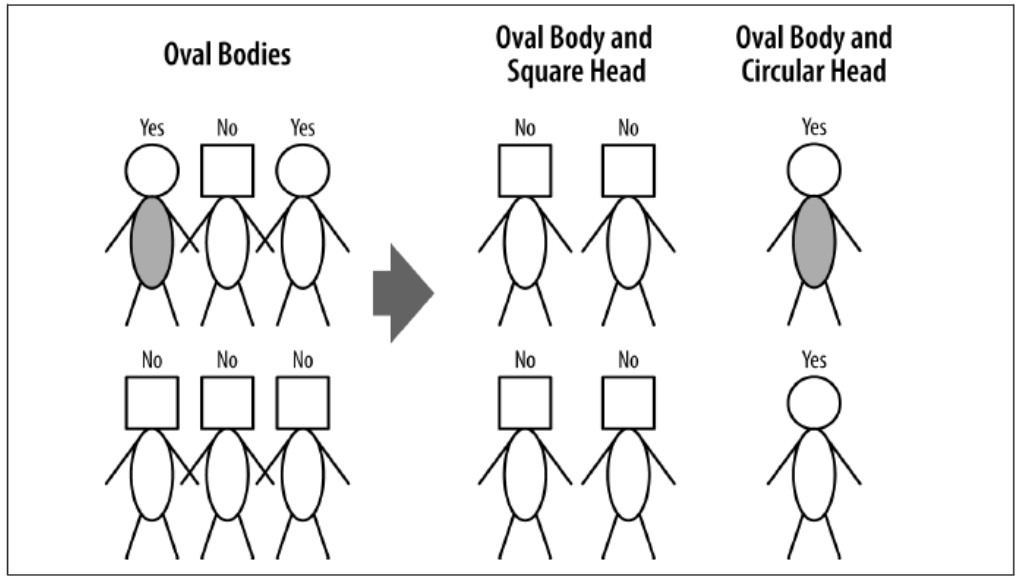


Figure 3-12. Second partitioning: the oval body people sub-grouped by head type.

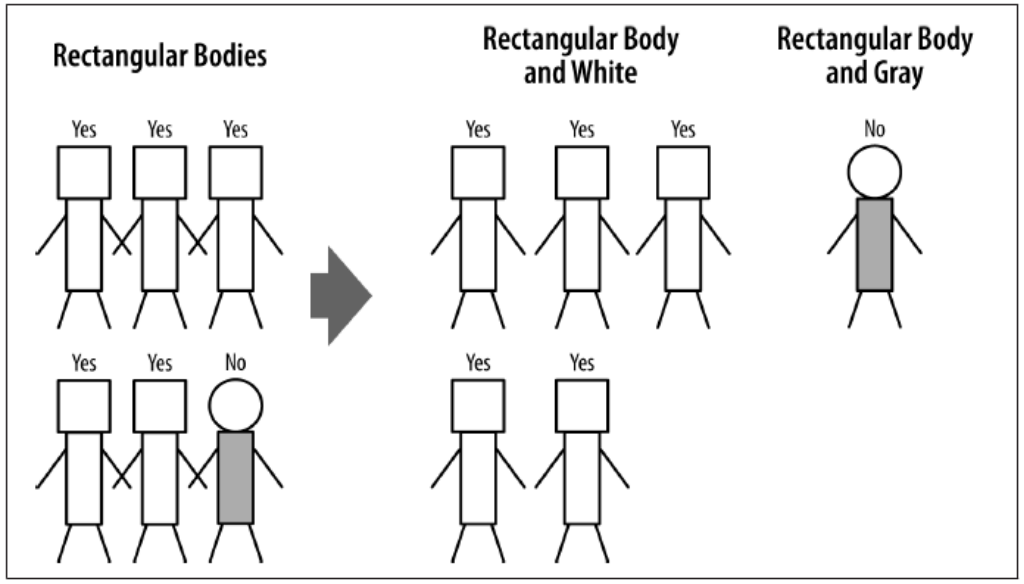


Figure 3-13. Third partitioning: the rectangular body people subgrouped by body color.

Decision Tree Example: Overall Decision Tree

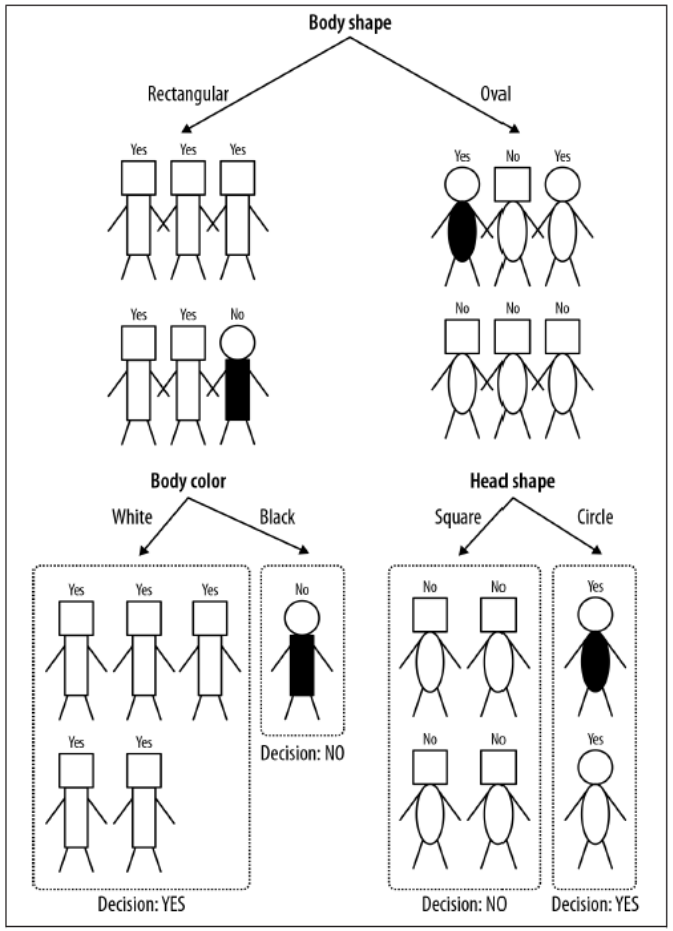


Figure 3-14. The classification tree resulting from the splits done in Figure 3-11 to Figure 3-13.

Decision Tree: Gini impurity index

Gini impurity measures the **probability of misclassifying a randomly chosen element** in the dataset if it were labeled randomly according to the class distribution in the node. It ranges from 0 (pure node) to 1 (impure node). A Gini impurity of 0 indicates that all the elements in the node belong to the same class.

We conclude that the second split is better, because it has a lower average Gini index.

gini impurity index In a set with m elements and n classes, with a_i elements belonging to the i-th class, the Gini impurity index is

$$Gini = 1 - p_1^2 - p_2^2 - \cdots - p_n^2,$$

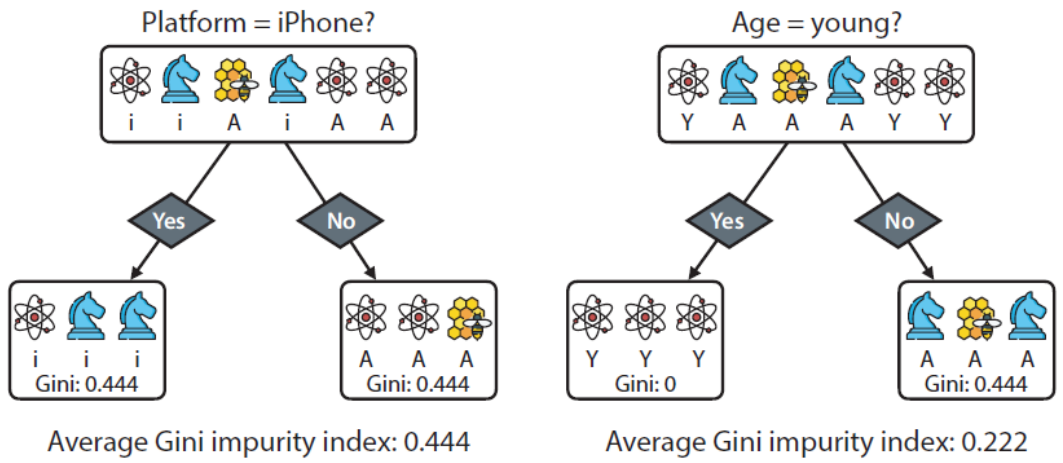
where $p_i = \frac{a_i}{m}$. This can be interpreted as the probability that if we pick two random elements out of the set, they belong to different classes.

Classifier 1 (by platform):

$$\text{Average Gini index} = \frac{1}{2}(0.444+0.444) = 0.444$$

Classifier 2 (by age):

$$\text{Average Gini index} = \frac{1}{2}(0.444+0) = 0.222$$



Decision Tree: Entropy

Entropy measures **the amount of uncertainty or randomness** in the dataset. It calculates the impurity by considering the class distribution in the node. Entropy ranges from 0 (pure node) to log(base 2) of the number of classes (impure node). An entropy of 0 indicates that all the elements in the node belong to the same class.

We conclude that the second split is better, because it has a lower average entropy.

AAAAAAAA

Bucket 1

Low Entropy

AAAABBCD

Bucket 2

Medium Entropy

AABBCCDD

Bucket 3

High Entropy

entropy In a set with m elements and n classes, with a_i elements belonging to the i-th class, the entropy is

$$Entropy = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \cdots - p_n \log_2(p_n),$$

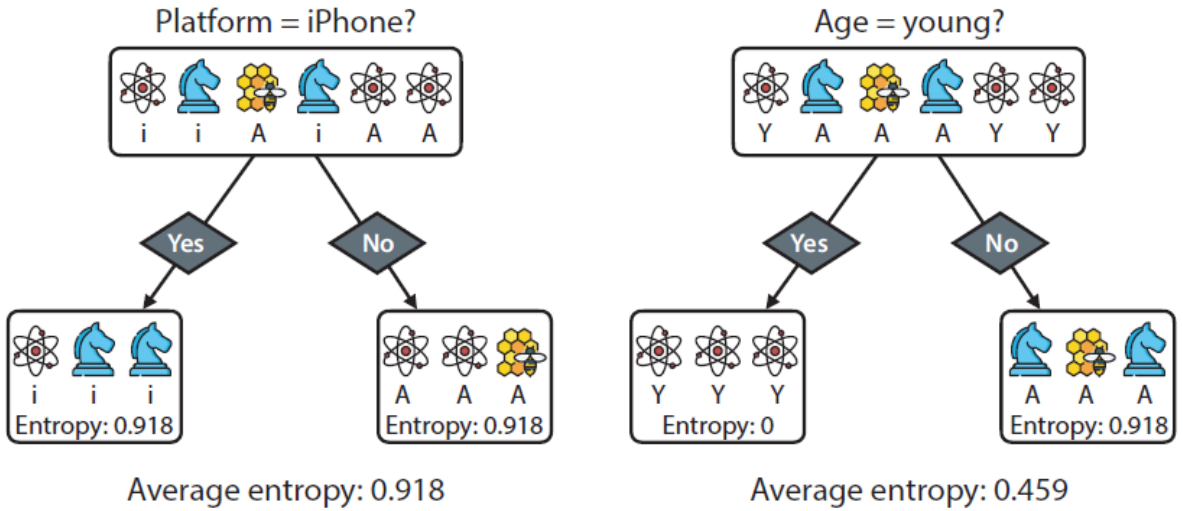
$$\text{where } p_i = \frac{a_i}{m}.$$

Classifier 1 (by platform):

$$\text{Average entropy} = \frac{1}{2}(0.918 + 0.918) = 0.918$$

Classifier 2 (by age):

$$\text{Average entropy} = \frac{1}{2}(0.918+0) = 0.459$$



Selecting the Best Split: Information Gain

- The most common splitting criterion is called an *information gain (IG)*
- We would like the split to be *informative*, meaning that it provide us with more information about our classification target. Therefore, we would like for the overall level of entropy to decrease after the segmentation
- We call the reduction in entropy level (the level of disorder) an *information gain (IG)*
  - i.e. Information Gain = total entropy before split - total entropy after split
- In other words ,

$$IG = entropy(parent) - p(c_1) \times entropy\,(c_1) - p(c_2) \times entropy(c_2) - \cdots$$

$$= entropy(parent) - \sum_i p(c_i) \times entropy\,(c_i)$$

- We pick the segmentation criteria to be the one that has the highest information gain.

Information Entropy

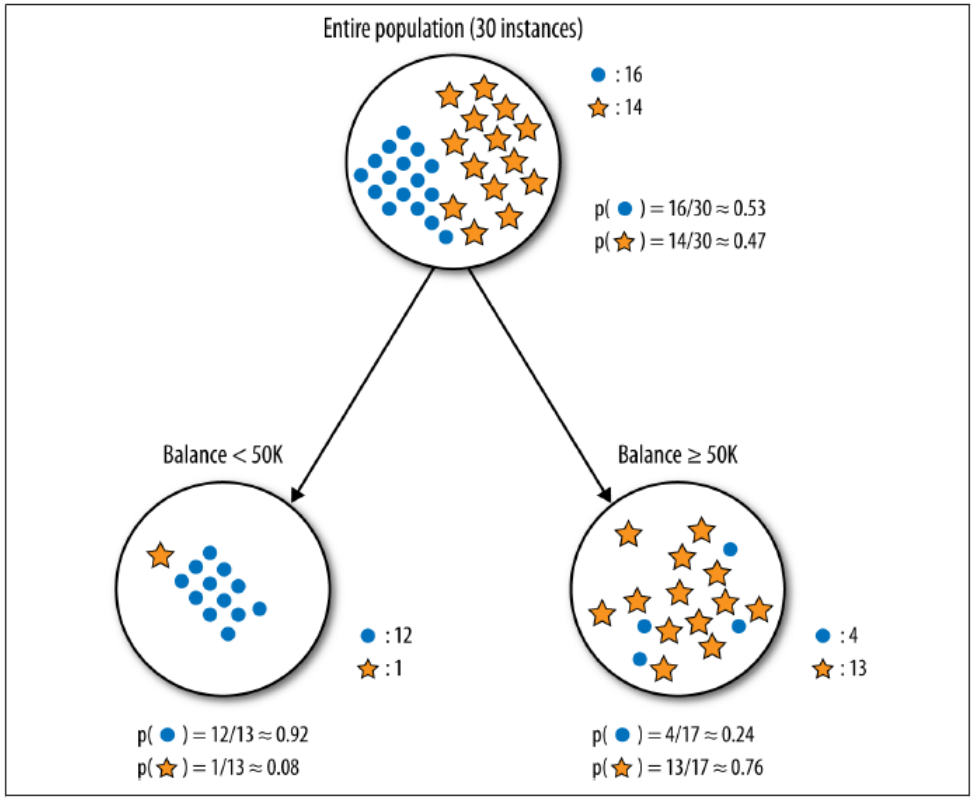


Figure 3-4. Splitting the "write-off" sample into two segments, based on splitting the Balance attribute (account balance) at 50K.

- The entropy of the *parent* (entire population) is

$$entropy(parent) = -p(\bullet) \times \log_2 p(\bullet) - p(\star) \times \log_2 p(\star)$$
$$\approx -0.53 \times -0.9 - 0.47 \times -1.1$$
$$\approx 0.99 \text{ (very impure)}$$

- The entropy of the *left* child is:

$$entropy(Balance < 50K) = -p(\bullet) \times \log_2 p(\bullet) - p(\star) \times \log_2 p(\star)$$
$$\approx -0.92 \times (-0.12) - 0.08 \times (-3.7)$$
$$\approx 0.39$$

- The entropy of the *right* child is:

$$entropy(Balance \geq 50K) = -p(\bullet) \times \log_2 p(\bullet) - p(\star) \times \log_2 p(\star)$$
$$\approx -0.24 \times (-2.1) - 0.76 \times (-0.39)$$
$$\approx 0.79$$

Selecting the Best Split: Information Gain

- The entropy of the *parent* is 0.99
- The entropy of the *left* child is 0.39
- The entropy of the *right* child is 0.79
- The Information Gain (IG) is:

$$IG = entropy(parent)$$
$$-[p(Balance < 50K) \times entropy(Balance < 50K)$$
$$+ p(Balance \geq 50K) \times entropy(Balance \geq 50K)]$$
$$\approx 0.99 - (0.43 \times 0.39 + 0.57 \times 0.79)$$
$$\approx 0.37$$

Decision trees for regression

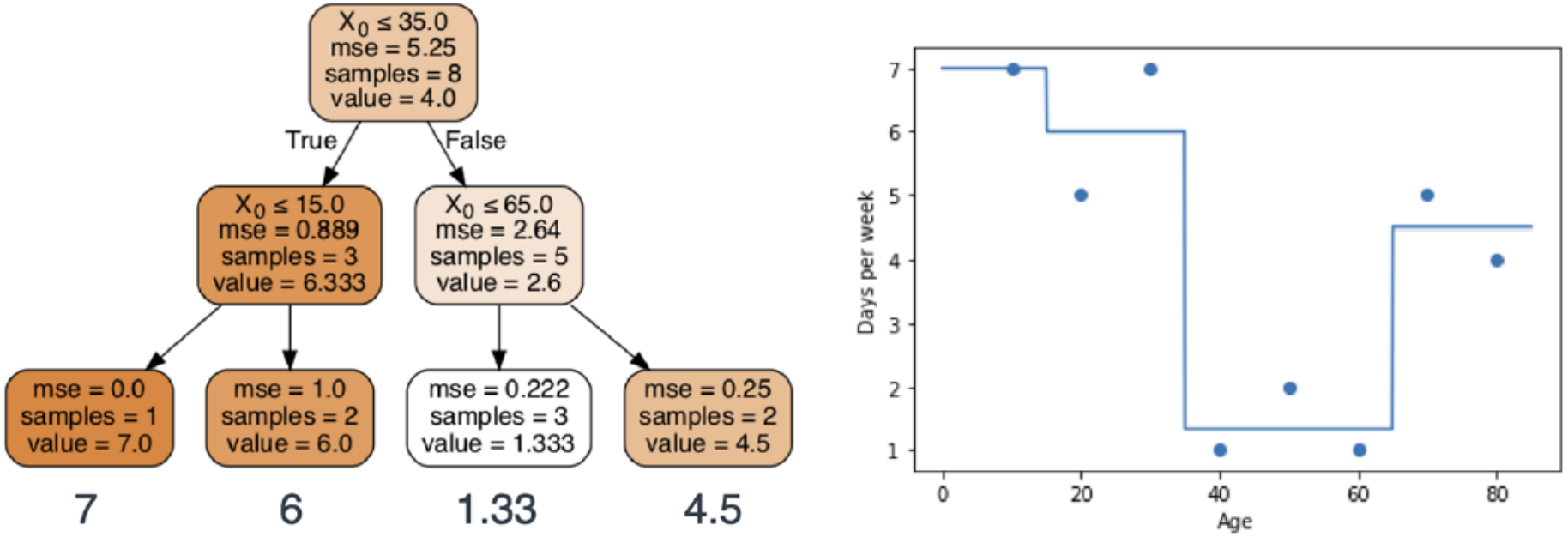
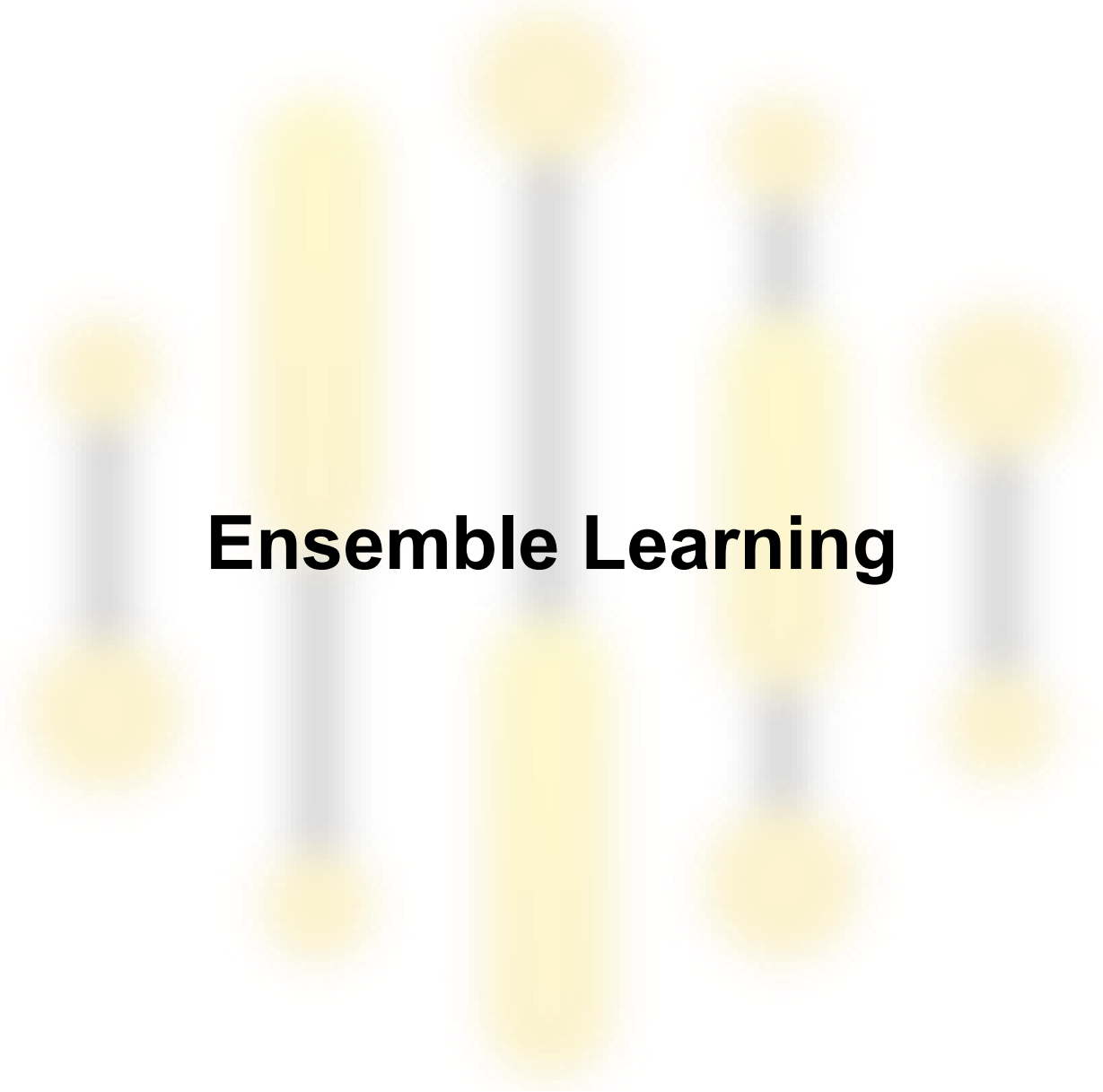


Figure 9.26 Left: The resulting decision tree obtained in Scikit-Learn. This tree has three decision nodes and four leaves. Right: The plot of the predictions made by this decision tree. Note that the cutoffs are at ages 35, 15, and 65, corresponding to the decision nodes in the tree. The predictions are 7, 6, 1.33, and 4.5, corresponding to the leaves in the tree.

Python Tutorial

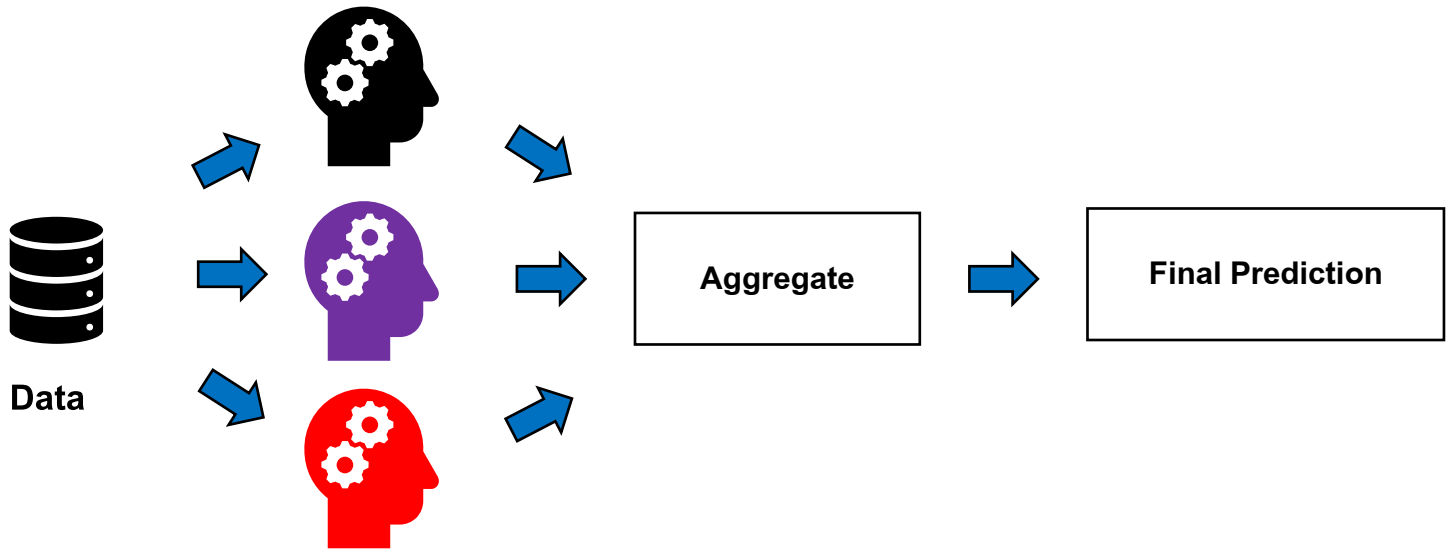
- Colab!
- 3.3 Decision Tree (titanic)
- 3.4 Decision Tree (wine)



Ensemble Learning

Ensemble Learning

- Ensemble learning is a process which utilizes multiple machine learning models to learn a particular task, then combines their results to make the final prediction.
- For example, an ensemble classifier may take result of its multiple classifiers then deciding the final prediction via voting or some other means of aggregations
- Random Forest and XGBoost models are examples of popular ensembles models



Majority Voting

- Somewhat surprisingly, this voting classifier often achieves a higher accuracy than the best classifier in the ensemble. In fact, even if each classifier is a *weak learner* (meaning it does only slightly better than random guessing), the ensemble can still be a *strong learner* (achieving high accuracy), provided there are a sufficient number of weak learners in the ensemble, and they are sufficiently diverse.

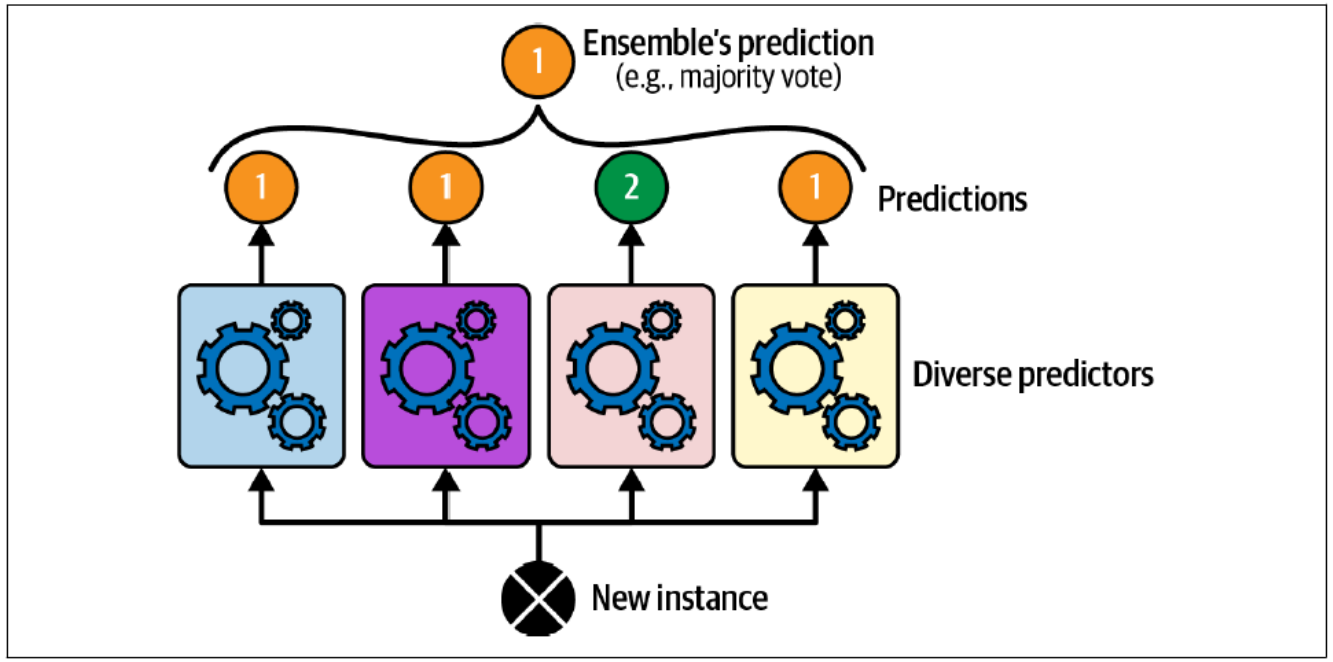


Figure 7-2. Hard voting classifier predictions

Bagging

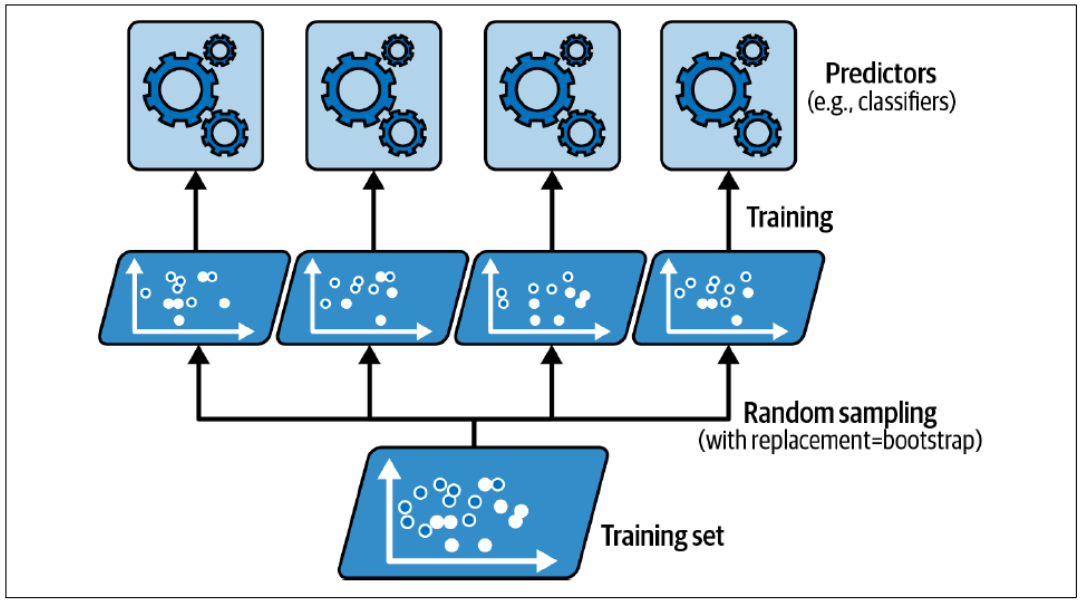


Figure 7-4. Bagging and pasting involve training several predictors on different random samples of the training set

- From the original data, repeatedly perform bootstrapping (taking random sampling with replacement) and create multiple sets of bootstrap samples
- Train multiple distinct classifiers on each set of bootstrap samples
- Uses all classifiers to predict a new input by voting / averaging
- Helps making the model more generalized

Bagging

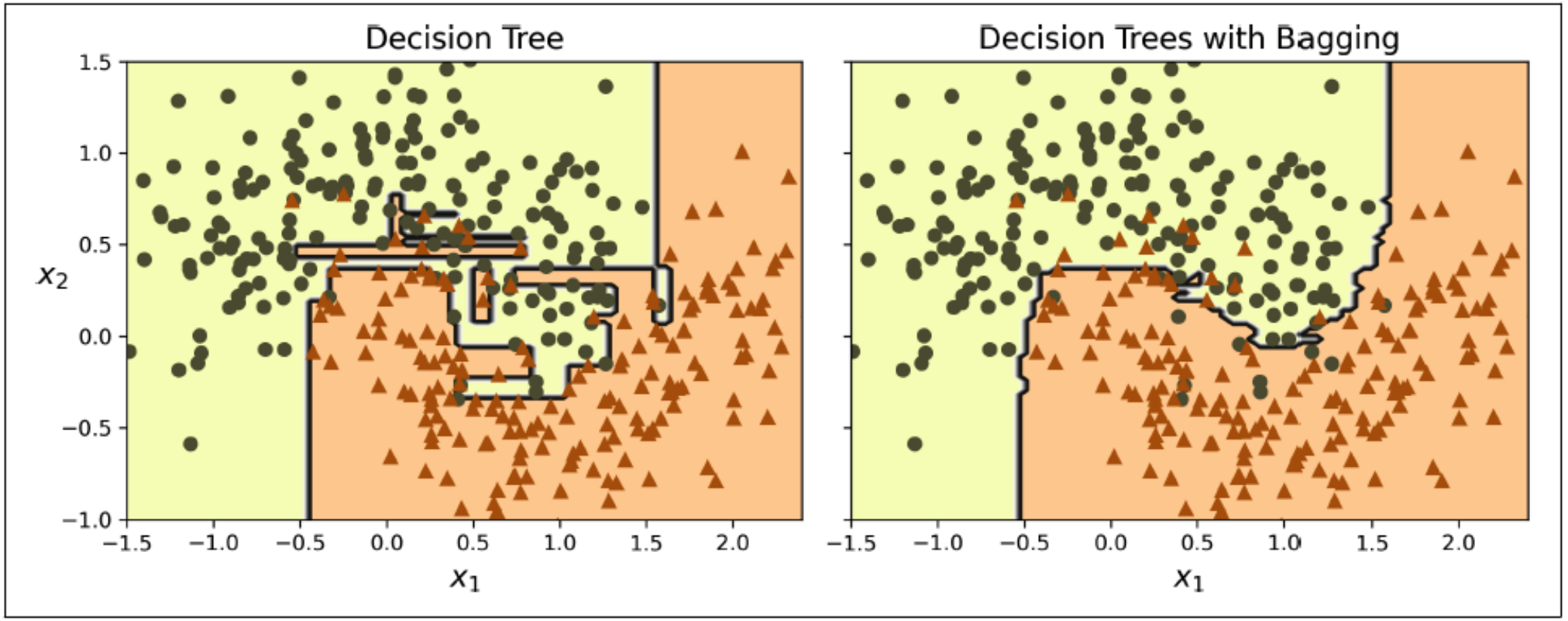


Figure 7-5. A single decision tree (left) versus a bagging ensemble of 500 trees (right)

Random forest

- An ensemble model consists of multiple decision trees, each of which are trained by samples from bootstrap sampling with only a random subset features available.
  - With large number of trees, the model is less likely to overfit.
  - With only a random subset of features available for each tree, it is less likely for trees' results to be highly correlated.
- Each of decision trees are built and train in a greedy fashion using conditional entropy
- The final prediction is decided by average predictions of all trees.

Content from KMUTT Big Data Experience Center's lecture on Ensemble Learning and
Jason Brownlee (https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/)

Random forest

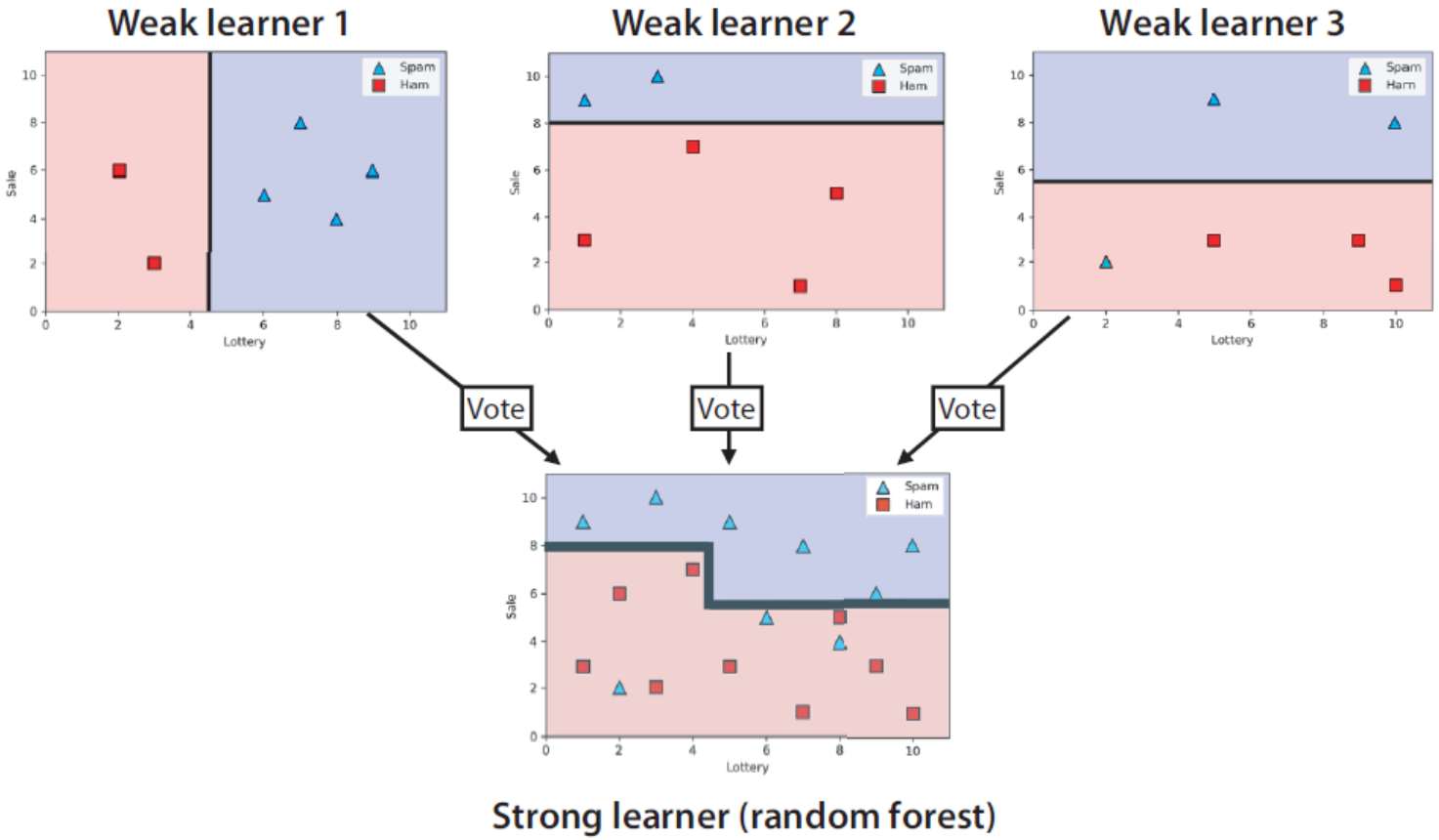


Figure 12.6 The way to obtain the predictions of the random forest is by combining the predictions of the three weak learners. On the top, we can see the three boundaries of the decision trees from figure 12.5. On the bottom, we can see how the three decision trees vote to obtain the boundary of the corresponding random forest.

Boosting

- Aims to improve the performance of the model by allowing it to fit the training data better
- Unlike bagging, it creates a sequence of classifiers and gives higher influence to a more accurate classifiers
- For each iteration, it makes examples currently misclassified more important (or less, in some cases) by giving larger weights in in the construction of the next classifiers.
- The final prediction is decided by combining the results of classifiers by weighted vote (weight given by classifier accuracy)

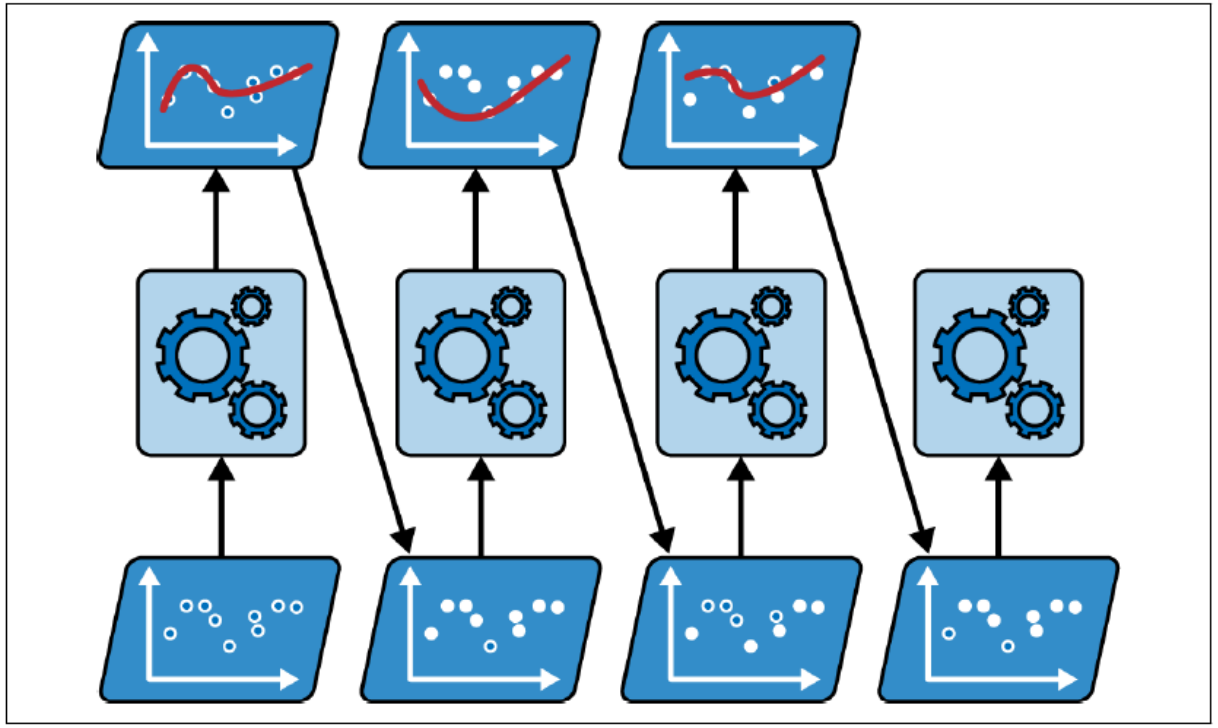
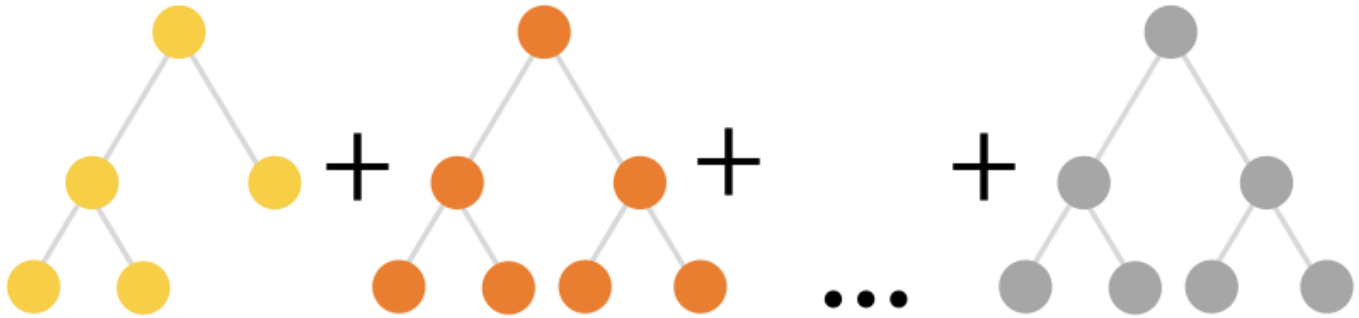


Figure 7-7. AdaBoost sequential training with instance weight updates

Gradient Tree Boosting

- The idea of gradient boosting is a combination of gradient descent and boosting
- Recall that in boosting, we aims to create a **strong** classifier (highly correlated to the target variable) by creating a sequence of **weak** classifiers, each compensating the shortcoming of prior weak learners.
- In Gradient Boosting, these shortcomings are identified by gradients and the adjustment is made to the model.
- Gradient tree boosting applies gradient boosting techniques on tree-based model



Content from KMUTT Big Data Experience Center's lecture on Ensemble Learning
Figure from https://catboost.ai/news/catboost-enables-fast-gradient-boosting-on-decision-trees-using-gpus

Gradient Tree Boosting

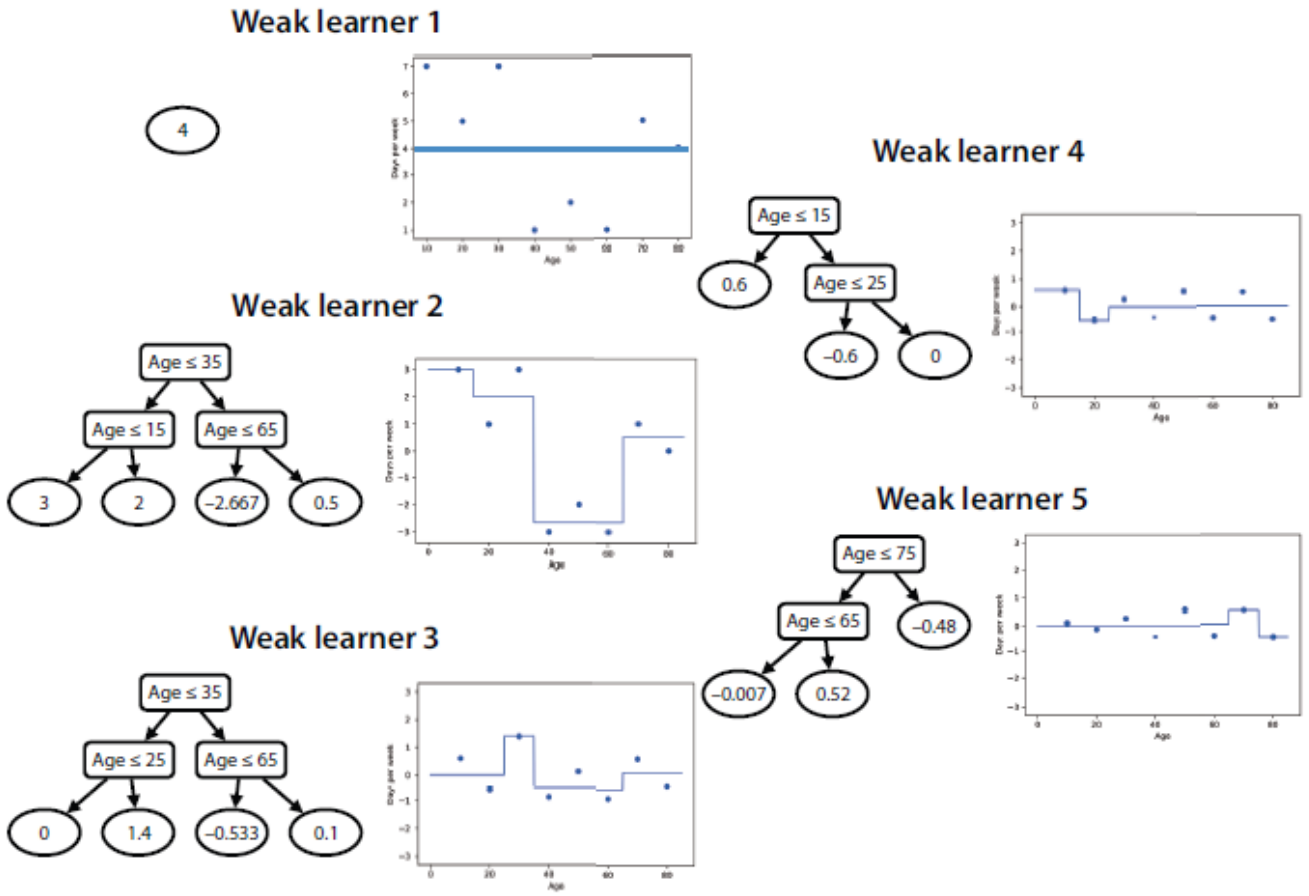


Figure 12.22 The five weak learners in the gradient boosting model. The first one is a decision tree of depth 0 that always predicts the average of the labels. Each successive weak learner is a decision tree of depth at most 2, which fits the residuals from the prediction given by the previous weak learners. Note that the predictions of the weak learners get smaller, because the residuals get smaller when the predictions of the strong learner get closer to the labels.

Extreme Gradient Boosting (XGBoost)

- A scalable and portable and accurate implementation of gradient boosting machines
- One of the most popular models. Perform well across variety of applications.
- Tree pruning: A way to reduce overfitting by simplifying the weak learners

Model Feature

- **Gradient Boosting** algorithm also called gradient boosting machine including the learning rate.
- **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
- **Regularized Gradient Boosting** with both L1 and L2 regularization.

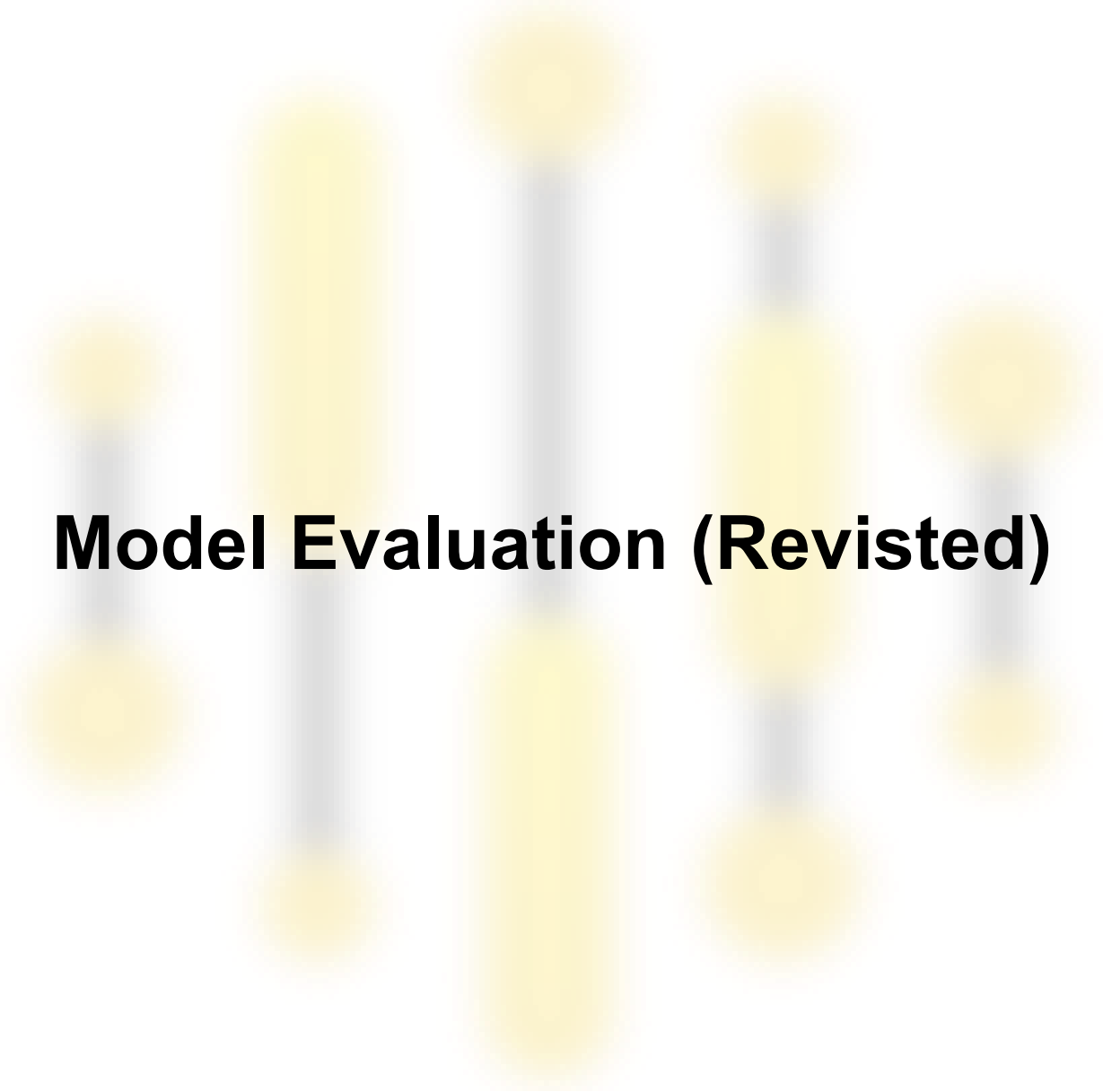
Extreme Gradient Boosting (XGBoost)

System Feature

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make best use of hardware.

Algorithm Feature

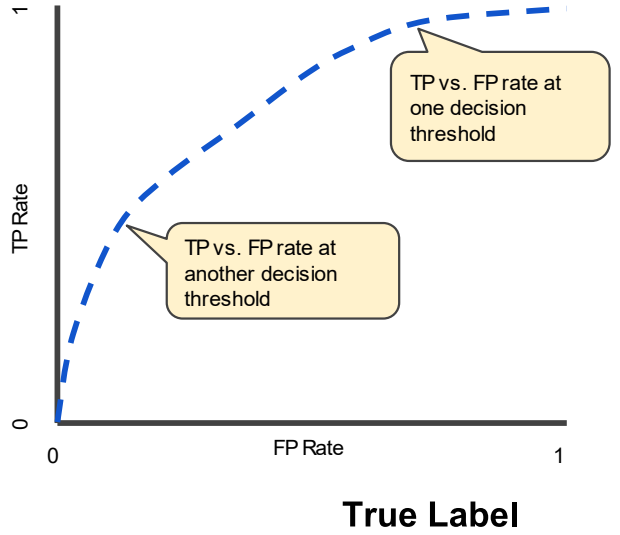
- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.



Model Evaluation (Revisted)

Model Selections: ROC curve and AUC

- The ROC (Receiver Operating Characteristic) curve is a plot that shows performance of a model at different classification threshold
- ROC Curves plots TPR vs. FPR, essentially comparing the "hit rate" and the "false alarm rate"

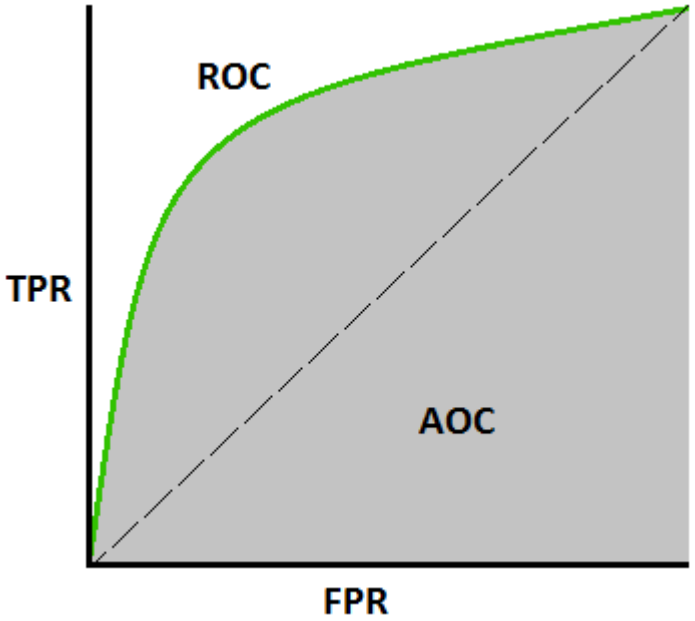


$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$False\ Positive\ Rate = \frac{FP}{FP + TN}$$

Predicted Label	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

Model Selections: ROC curve and AUC



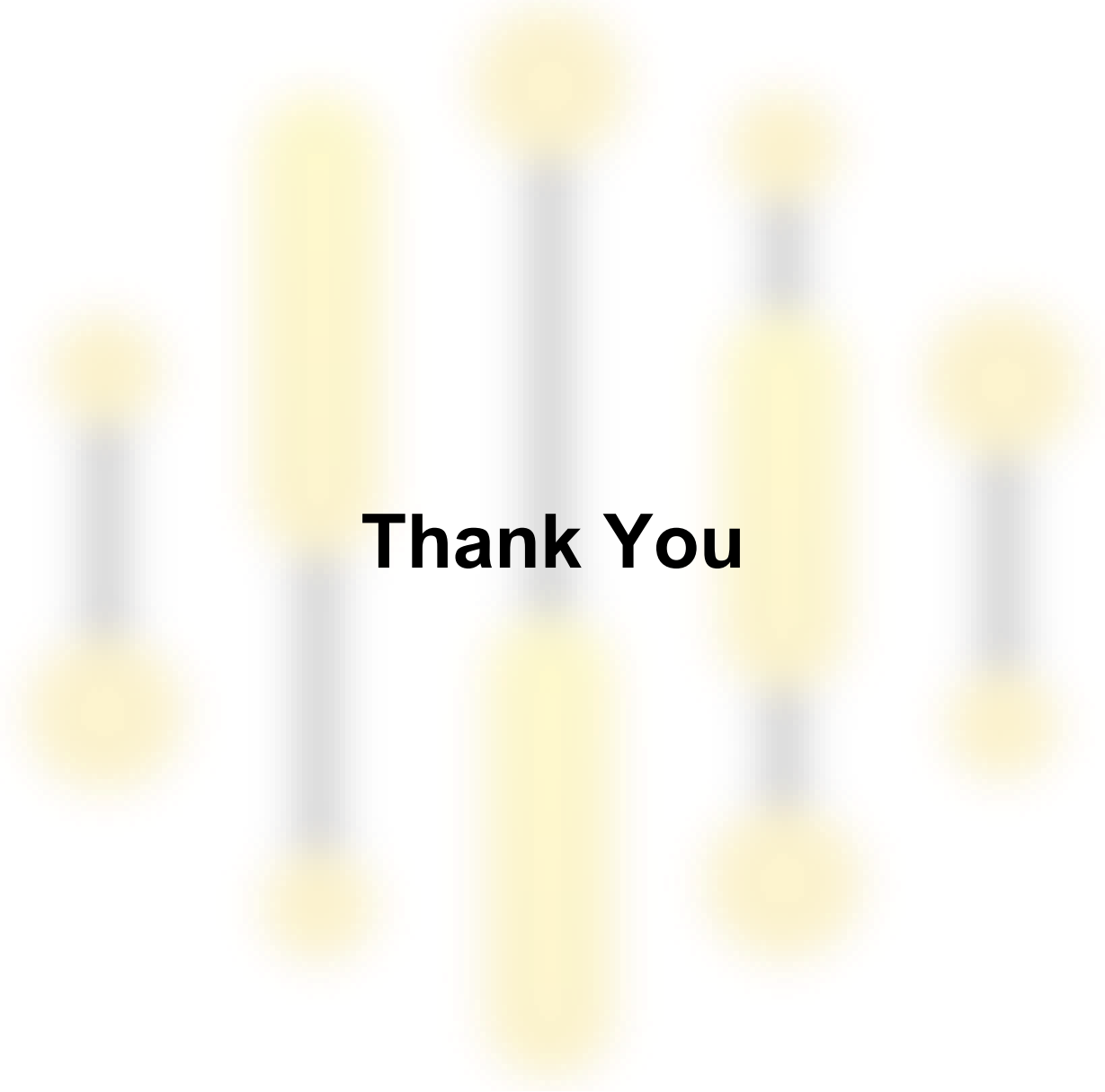
- AUC (Area Under the ROC Curve) provides a metric for considering the models performance across all classification threshold.

- *Side note: so far, we have been using a classification threshold of 0.5. However, it is also possible (and sometimes beneficial) for a classifier to vary.*

Contents from Google's Machine Learning Crash Course (https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc)
Figure from Sarang Narkhede (https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5)

Python Tutorial

- Colab!
- 3.5 Random Forest and XGBoost (Telco-Customer-Churn)



Thank You

GBDi
Government Big Data Institute
สถาบันส่งเสริมการวิเคราะห์และบริหารข้อมูลขนาดใหญ่ภาครัฐ (สวช.)

Facebook

Twitter

Blockdit

govbigdata

Follow us on

GBDi

gbdi.depa.or.th

YouTube

Government Big Data Institute (GBDi)

Line Official

@gbdi

BIG DATA
THAILAND

Follow us on

BIG DATA
THAILAND

Website

Facebook

Blockdit

Twitter