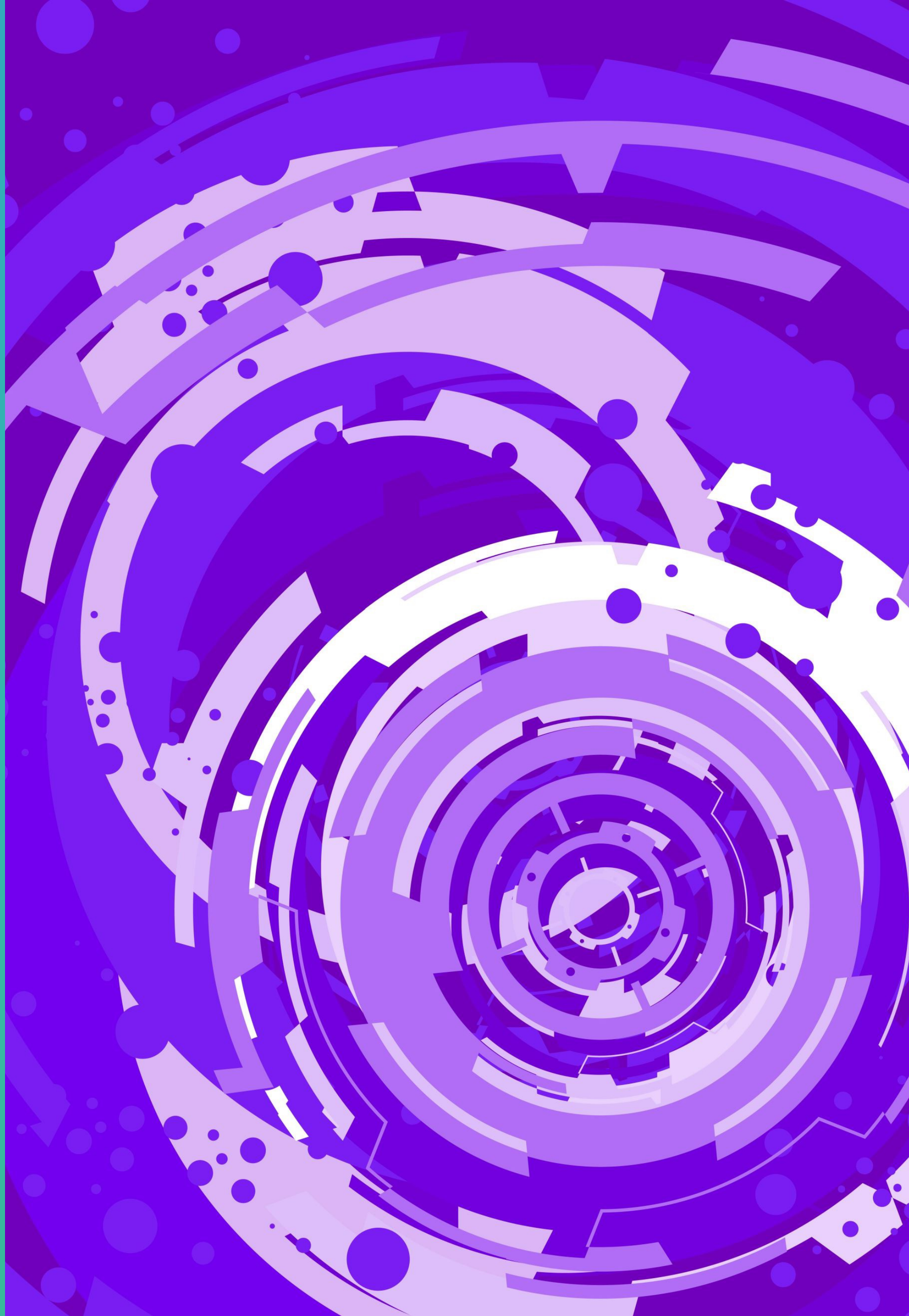
URBAN GEOGRAPHIC INFORMATION SYSTEM

Python ML Classification

Chun-Hsiang Chan

Department of Geography,

National Taiwan Normal University



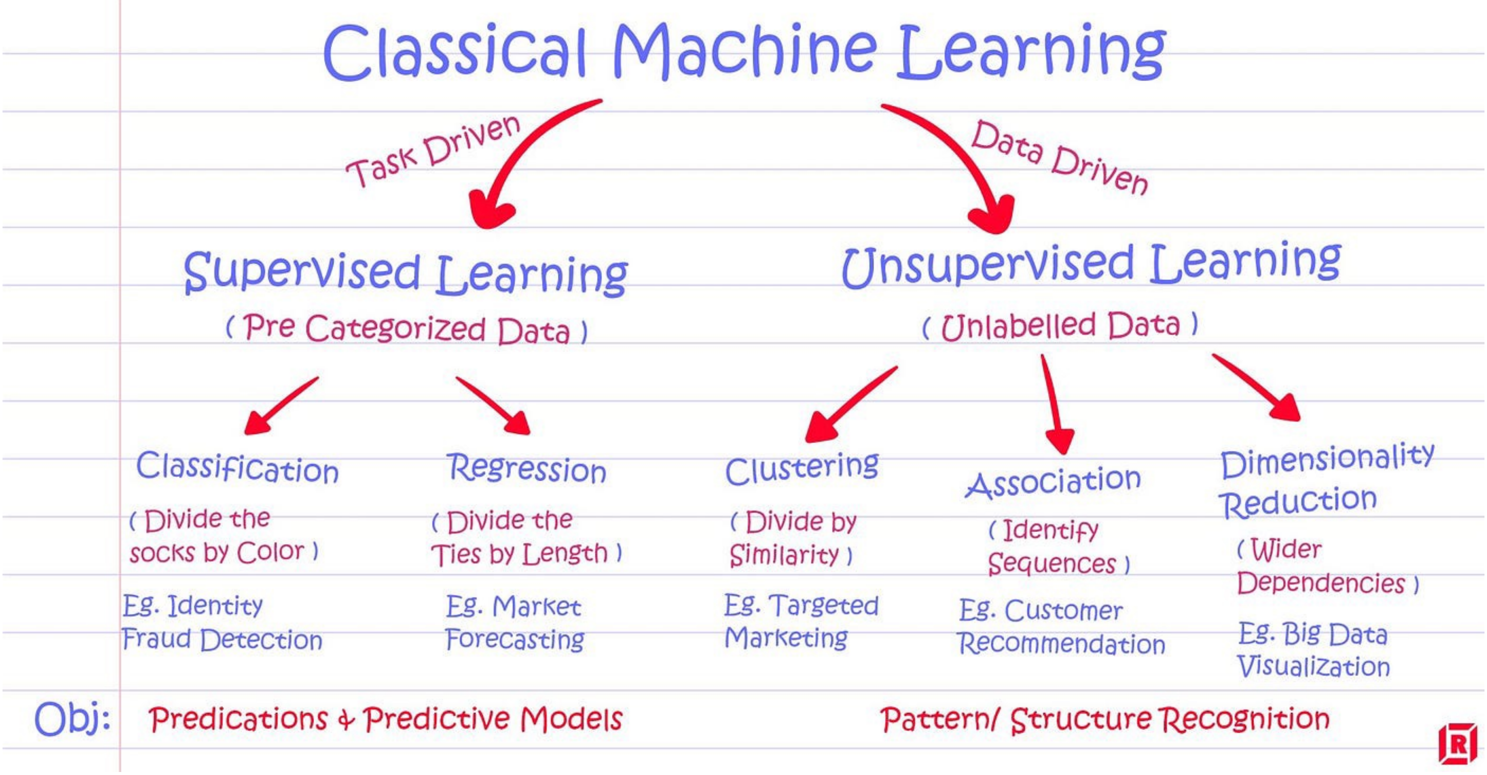
Outline

- Concept of Classification
- Evaluation Metrics
- Model Optimization
- Overfitting vs Underfitting
- Bagging
- Boosting
- K-Nearest Neighbors
- Decision Tree
- Random Forest
- AdaBoost
- Gradient Boost
- XGBoost
- Naive Bayes
- Linear SVM
- Remarks

Concept of Classification

Supervised vs unsupervised learning

- Regression and classification are designed for continuous and categorical (discrete) datasets, respectively.



Source: https://medium.com/@dkatzman_3920/supervised-vs-unsupervised-learning-and-use-cases-for-each-8b9cc3ebd301

Concept of Classification

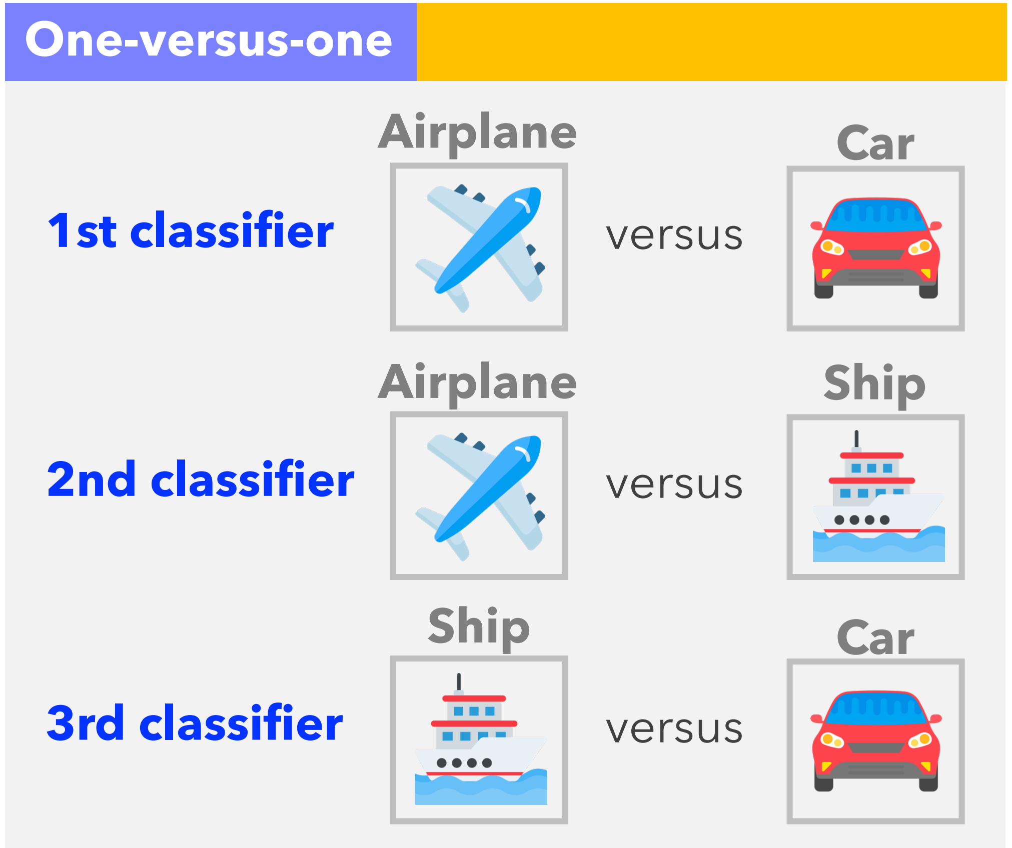
Classifier Types

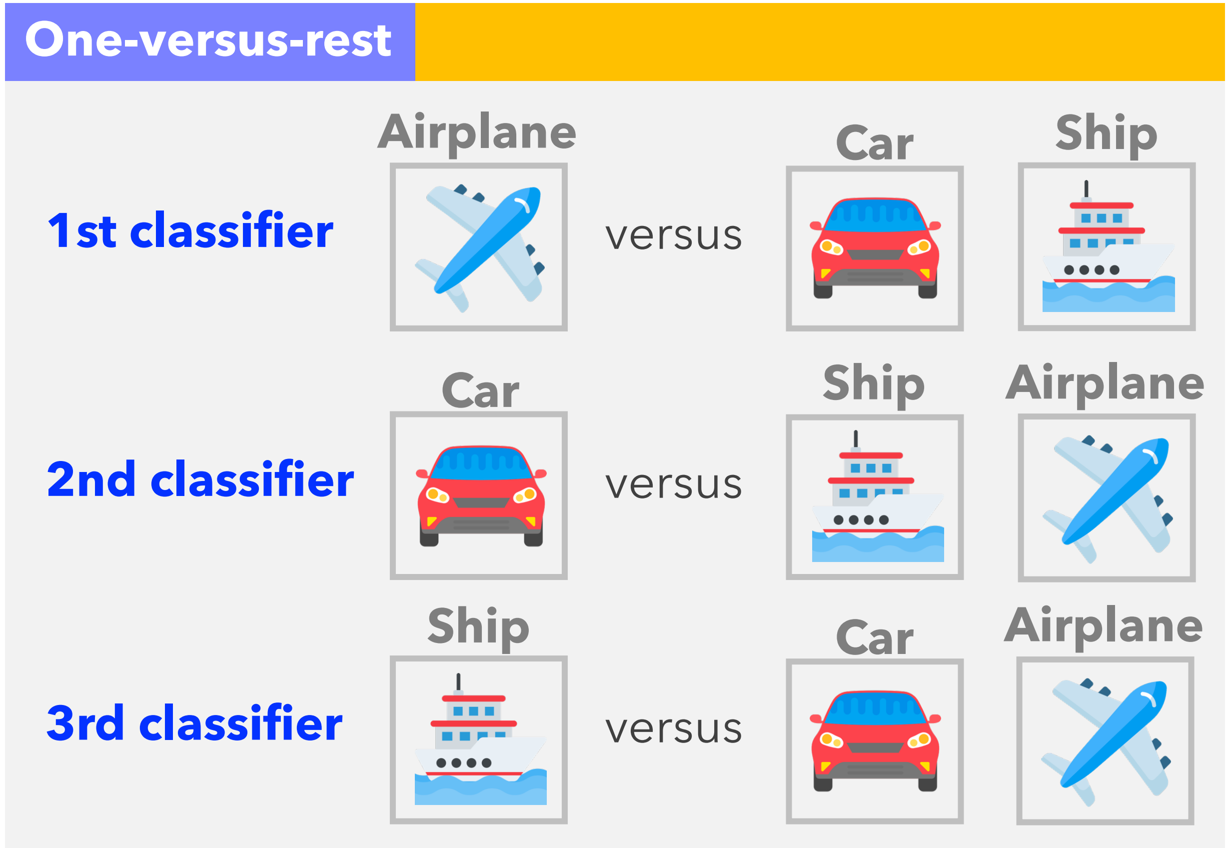
- Basically, all machine learning classifiers consist of binary, multi-class, and multi-label classifications.
- **Binary Classifier:**

Concept of Classification

Classifier Types

- **Multi-class classification:** classify airplanes, cars, and ships.



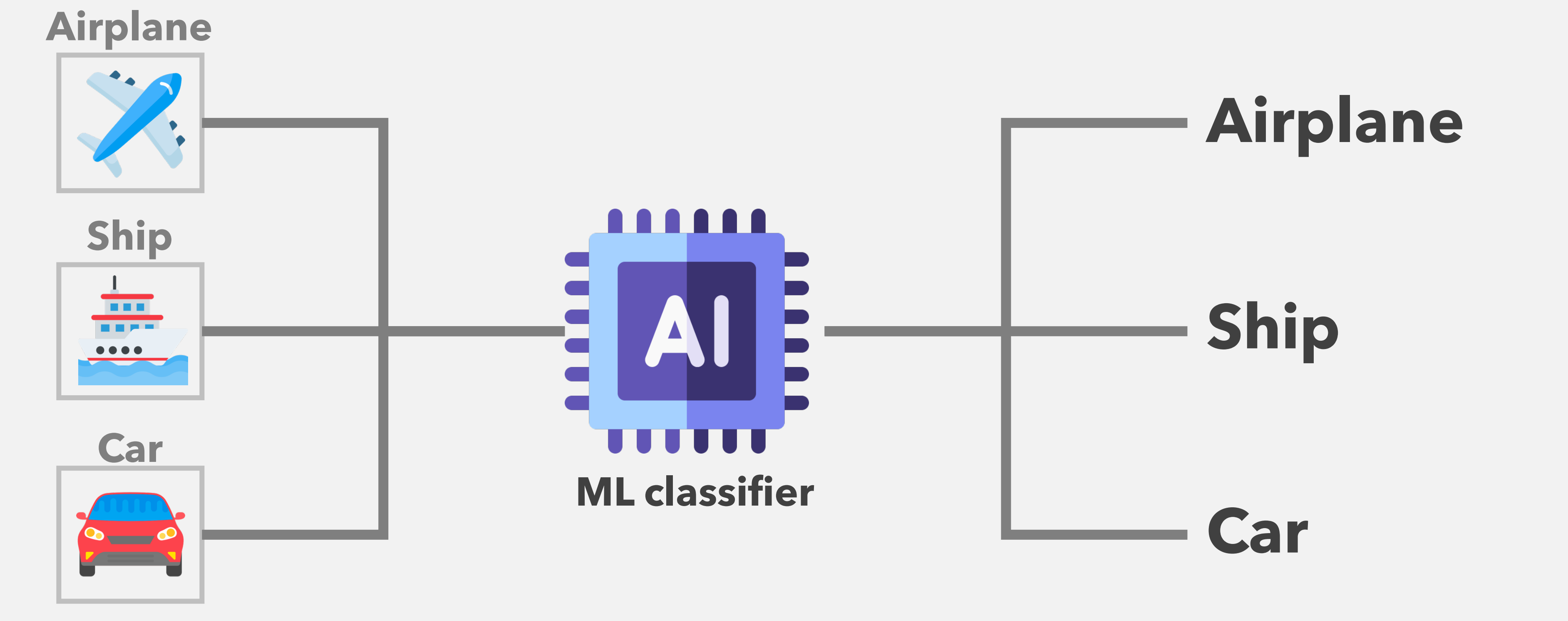


Concept of Classification

Classifier Types

- **Multi-label classification:**

One Model classifies all categories ...



Concept of Classification

Data Imbalance Issue

- Normally, it is too difficult to collect all categories with a huge sample size; therefore, some categories might have only a few samples, leading to a bad prediction result.
- Think about why the ML models cannot predict the small-sample-size category well?
- If so, how can we overcome this dilemma?
- Please list your solutions…

Concept of Classification

- **Possible solutions:**
  - **Cluster-based oversampling.**
  - **Random undersampling:** random elimination of examples from the majority class.
  - **SMOTE oversampling:** random replication of examples from the minority class.

SMOTE

```
class imblearn.over_sampling.SMOTE(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None)
```

[source]

Class to perform over-sampling using SMOTE.

This object is an implementation of SMOTE - Synthetic Minority Over-sampling Technique as presented in [1].

Read more in the User Guide.

Evaluation Metrics

- Evaluation metrics are vital elements to quantitatively assess the performance of ML model training and testing processes.
- The most common way to demonstrate the model performance is a confusion matrix.

Confusion Matrix		Actual Values		
		Positive (Yes)	**Negative (No)**	
Predicted Values	**Positive (Yes)**	True Positive (TP)	False Positive (FP)	**Type I Error**
	Negative (No)	False Negative (FN)	True Negative (TN)	
		Type II Error		

Evaluation Metrics

Items	Formula	Meanings
Accuracy	$\frac{TP+TN}{TP+TN+FN+FP}$	Out of the prediction made by the model, what percentage is correct.
Precision	$\frac{TP}{TP+FP}$	Out of all the YES predications, how many of them were correct?
Recall (Sensitivity)	$\frac{TP}{TP+FN}$	How good was the model at predicting real YES events?
Recall (Specificity)	$\frac{TN}{TN+FP}$	How good was the model at predicting real NO events?
F1 score	$\frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}$	Dealing with the imbalanced dataset, we have to use F1 score, indicating the harmonic mean of the precision and recall.

Source: https://www.datacamp.com/blog/classification-machine-learning

Evaluation Metrics

AUC-ROC Curve

AUC: area under curve
ROC: receiver operating characteristic curve

- AUC-ROC generates probability values instead of binary 0/1 values. It should be used when your dataset is roughly balanced.
- Using ROC for imbalanced datasets lead to incorrect interpretation.
- ROC curves provide a good overview of the trade-off between the TP rate and FP rate for binary classifiers using different probability thresholds.

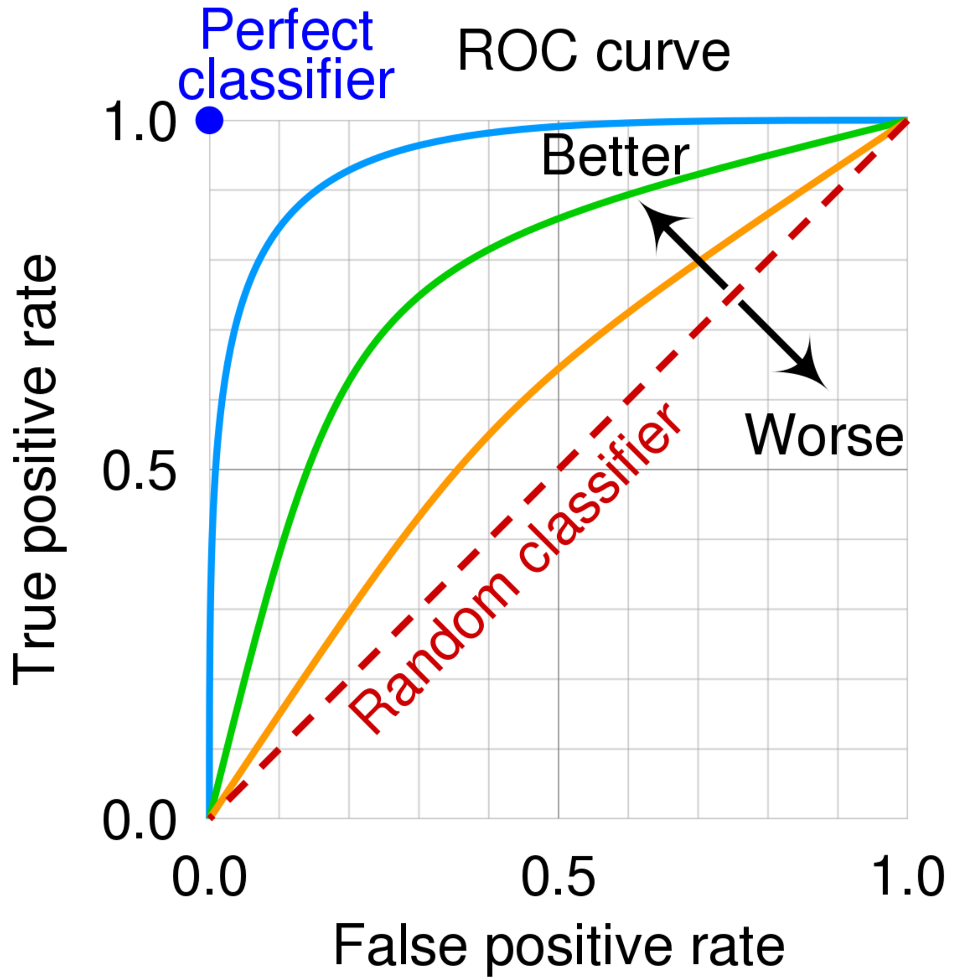
Source: https://www.datacamp.com/blog/classification-machine-learning

Evaluation Metrics

AUC-ROC Curve

AUC Value	Meanings
<0.5	Poor classifier
0.5~0.7	Random classifier
0.7~0.8	Good classifier
0.8~	Strong classifier
1	Perfect classifier

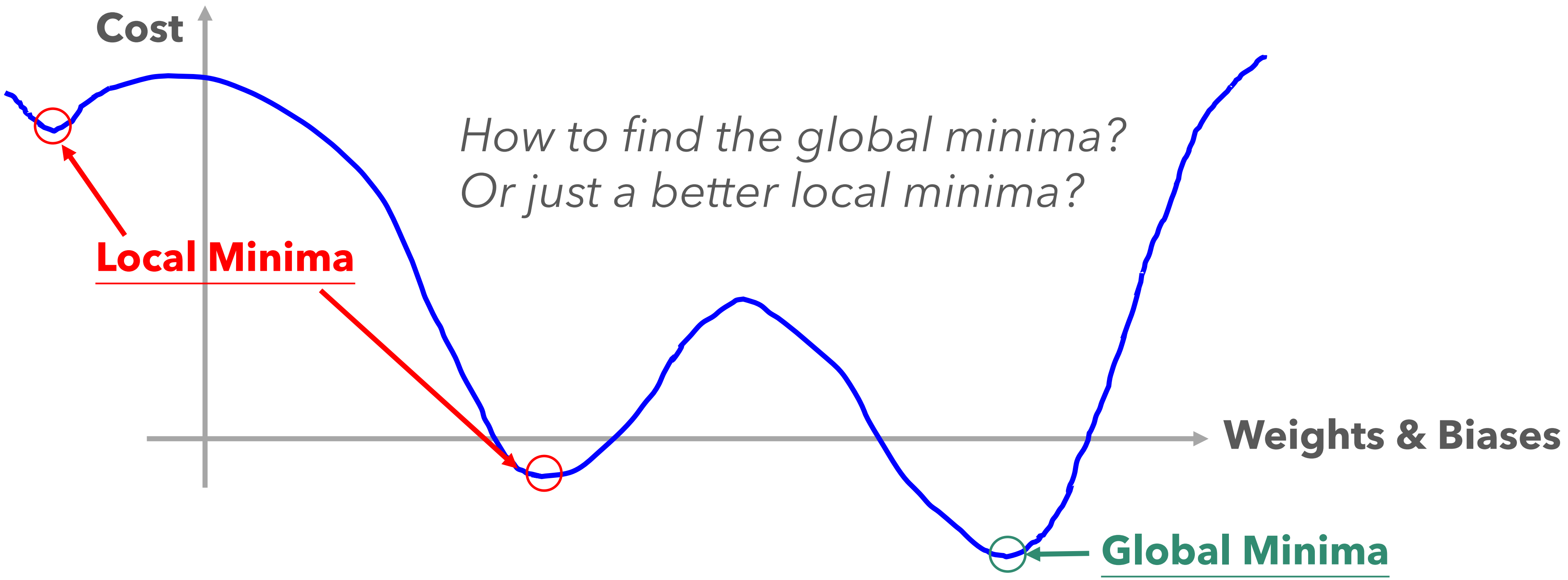
Source: https://www.datacamp.com/blog/classification-machine-learning



Source: https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02

Model Optimization

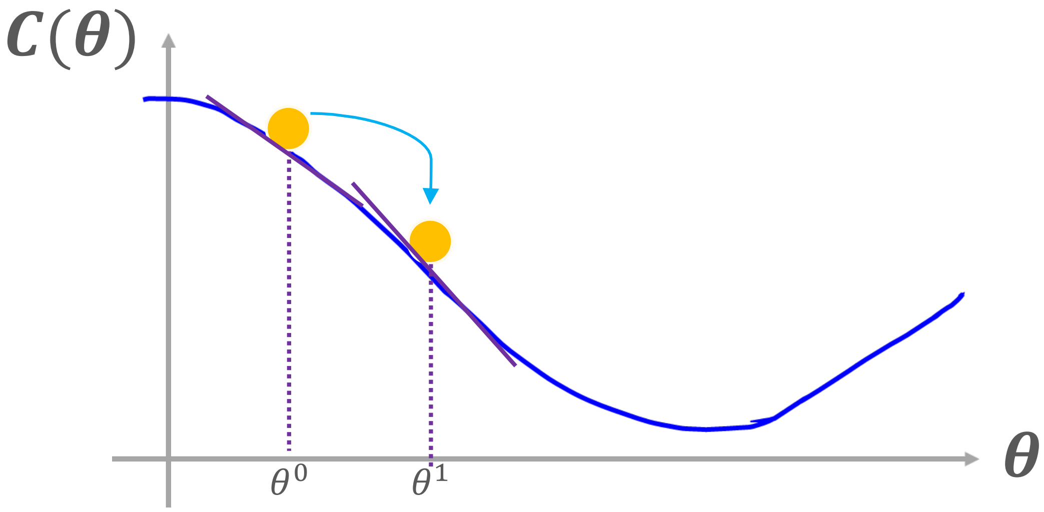
- To reach the best model performance, it is important to optimize (tune) the hyperparameters of your model.



Model Optimization

- **Gradient Descent:**

Which way?
How does it work?



Assume that θ has only one variable…

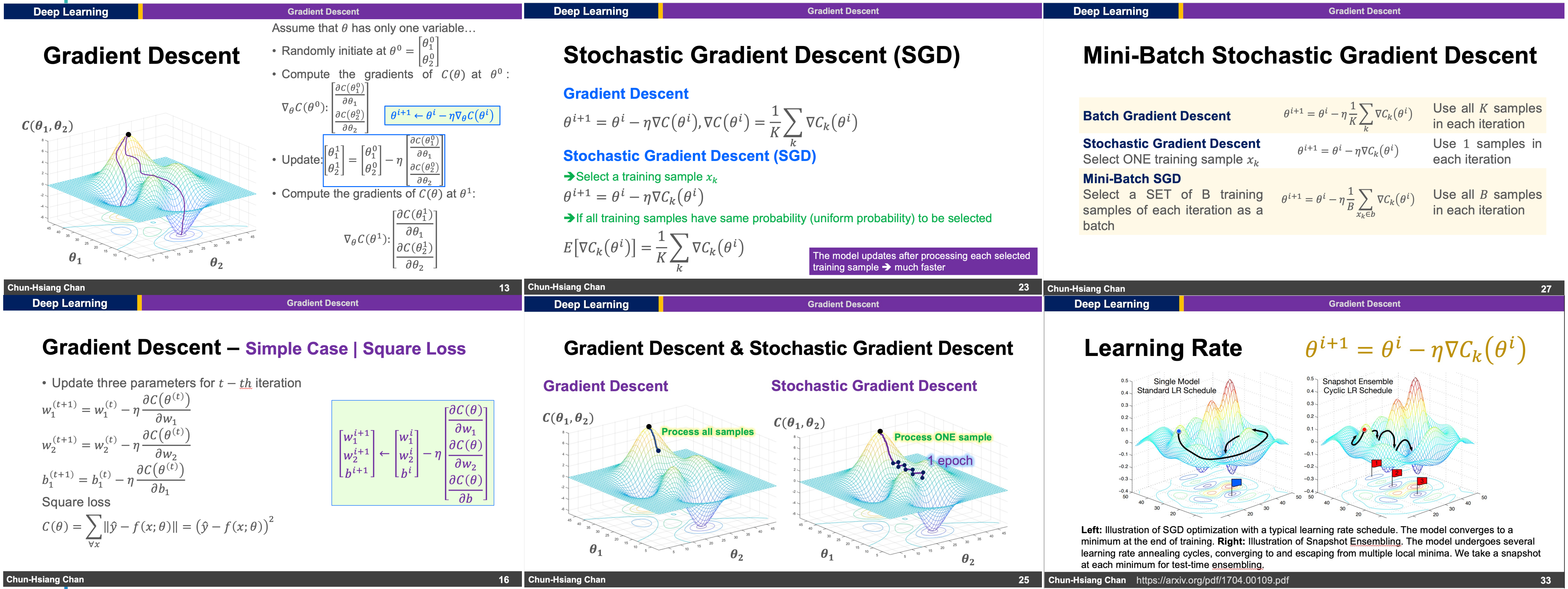
- Randomly initiate at θ^0
- Compute $\frac{dC(\theta^0)}{d\theta}$: $\theta^1 \leftarrow \theta^0 - \eta\frac{\partial(\theta^0)}{\partial\theta}$
- Compute $\frac{dC(\theta^1)}{d\theta}$: $\theta^2 \leftarrow \theta^1 - \eta\frac{\partial(\theta^1)}{\partial\theta}$
- …

➔ $\theta^{i+1} \leftarrow \theta^i - \eta\nabla_\theta C(\theta^i)$

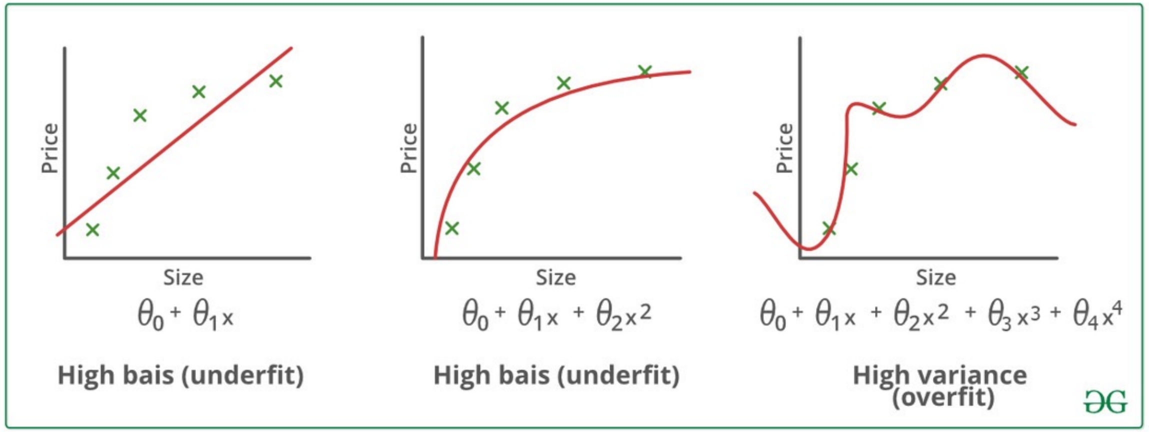
where η is the learning rate

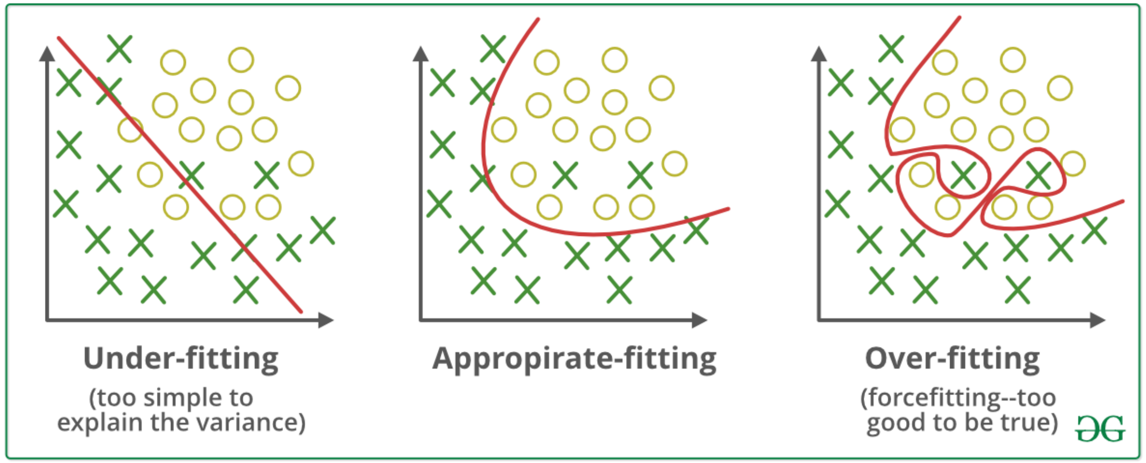
Model Optimization

- **For more information**: plz see **here** **[YouTube Link]**.



Overfitting vs Underfitting





Techniques to reduce overfitting

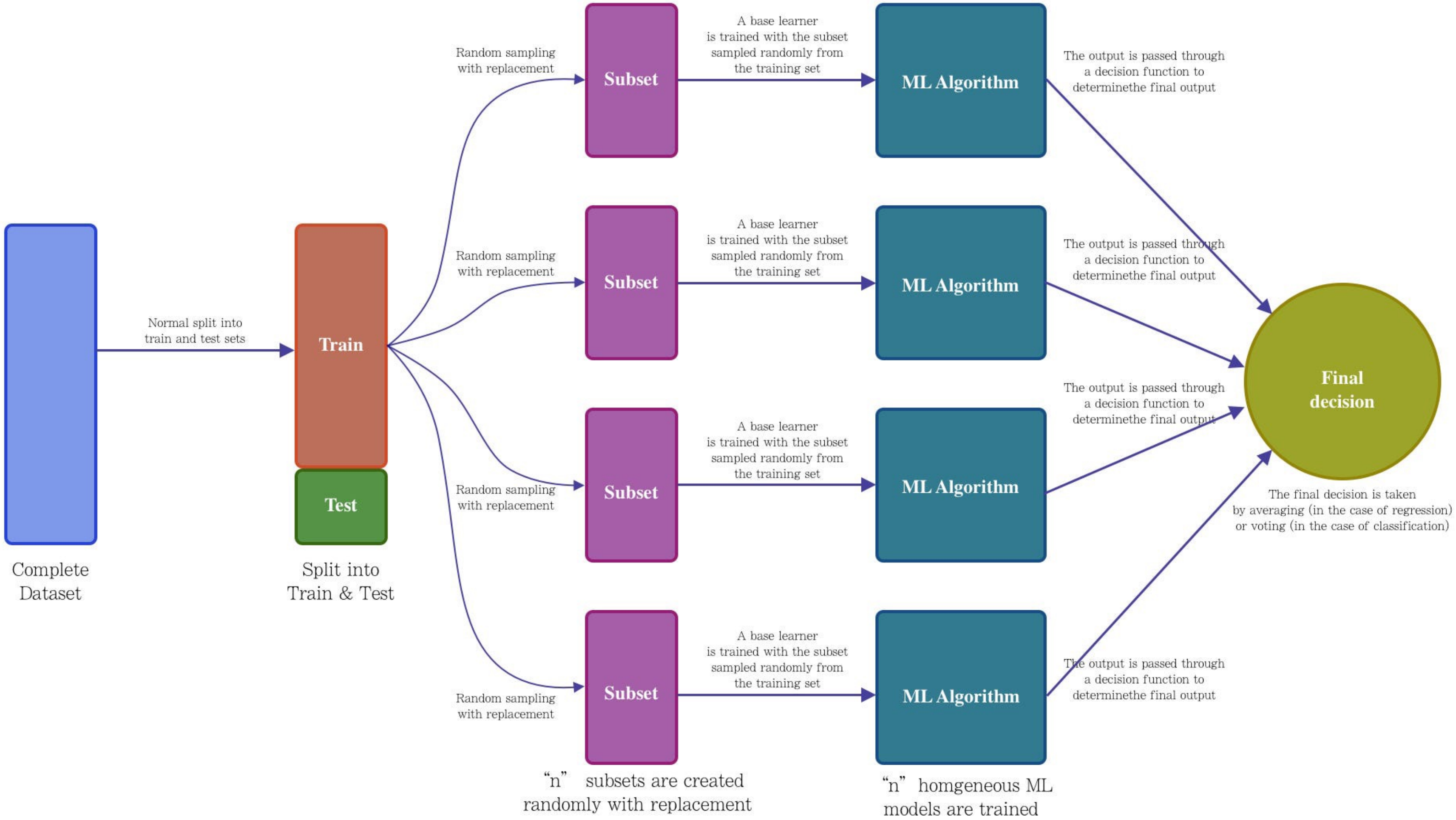
1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (keep an eye on the loss over the training period; as soon as loss increases, stop training).
4. Ridge Regularization and Lasso Regularization
5. Use dropout for neural networks to tackle overfitting.

Bagging (Bootstrap Aggregation)

- The two main components of the **bagging** technique are the *random sampling with replacement* (**bootstrapping**) and the *set of homogeneous* machine learning algorithms (**ensemble learning**).
- The **bagging** process is quite easy to understand. First, it extracts "*n*" subsets from the training set, and then these subsets are used to train "*n*" base learners of the same type.
- For making a prediction, each one of the "*n*" learners is fed with the test sample, and the output of each learner is averaged (in case of regression) or voted (majority vote).

Source: https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422

Bagging (Bootstrap Aggregation)



Source: https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422

Boosting

- Boosting creates an ensemble model by combining several weak decision trees sequentially. It assigns weights to the output of individual trees. Then it gives incorrect classifications from the first decision tree a higher weight and input to the next tree. After numerous cycles, the boosting method combines these weak rules into a single powerful prediction rule.
- **Step 1:** The boosting algorithm assigns equal weight to each data sample. It feeds the data to the first machine model, called the base algorithm. The base algorithm makes predictions for each data sample.

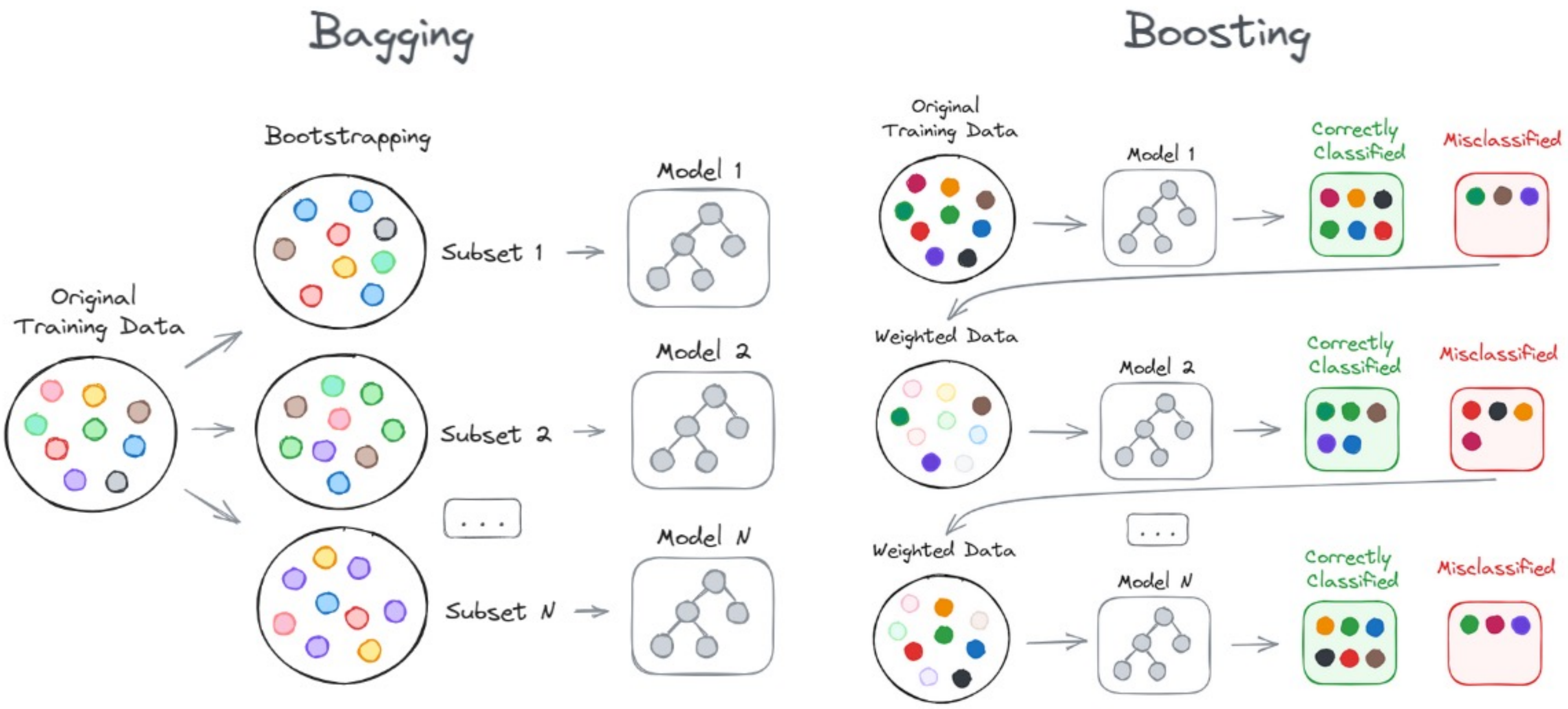
Source: https://aws.amazon.com/what-is/boosting/

Boosting

- **Step 2:** The boosting algorithm assesses model predictions and increases the weight of samples with a more significant error. It also assigns a weight based on model performance. A model that outputs excellent predictions will have a high amount of influence over the final decision.
- **Step 3:** The algorithm passes the weighted data to the next decision tree.
- **Step 4:** The algorithm repeats steps 2 and 3 until instances of training errors are below a certain threshold.

Source: https://aws.amazon.com/what-is/boosting/

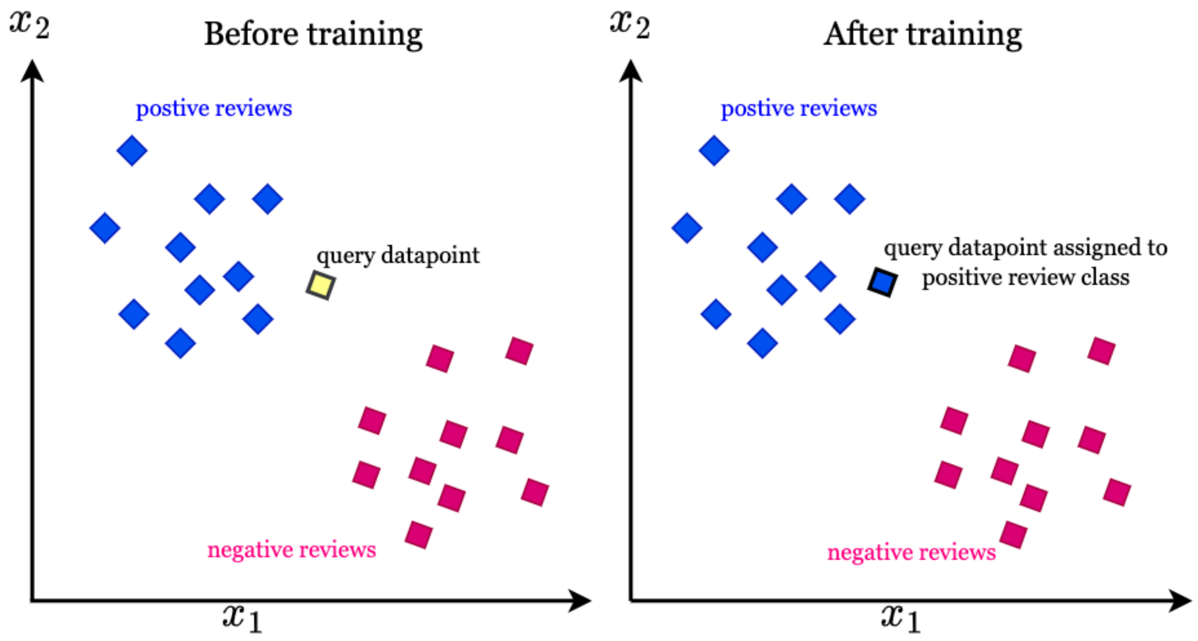
Bagging vs Boosting



Source: https://pub.towardsai.net/bagging-vs-boosting-the-power-of-ensemble-methods-in-machine-learning-6404e33524e6

k-Nearest Neighbors

- The k-nearest neighbors classify new data points to a particular category based on their similarity with the other data points in that category.
- Distance measurement:
    - Manhattan Distance
    - Minkowski Distance
    - Cosine Similarity
    - …



Source: https://intuitivetutorial.com/2023/04/07/k-nearest-neighbors-algorithm/

Decision Tree

- The decision tree is the most basic machine learning classifier.
- Before introducing the decision tree concept, we first need to know "entropy" and "information gain", used to determine the tree node.
- (Information) Entropy:

$$Entropy = -\sum_{i=1}^{n} P(x_i)\log P(x_i)$$

- Information Gain:

$$Information\ gain = Entropy(S) - \sum_{i=1}^{n} w_i\, Entropy(i)$$

Decision Tree

Idx	Outlook	Temp	Humidity	Windy	Play
0	Sunny	Hot	High	0	0
1	Sunny	Hot	High	1	0
2	Overcast	Hot	High	0	1
3	Rainy	Mild	High	0	1
4	Rainy	Cool	Normal	0	1
5	Rainy	Cool	Normal	1	0
6	Overcast	Cool	Normal	1	1
7	Sunny	Mild	High	0	0
8	Sunny	Cool	Normal	0	1
9	Rainy	Mild	Normal	0	1
10	Sunny	Mild	Normal	1	1
11	Overcast	Mild	High	1	1
12	Overcast	Hot	Normal	0	1
13	Rainy	Mild	High	1	0

$$E(S) = 0.94$$

$$E(outlook = sunny) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$

$$E(outlook = overcast) = -\frac{4}{4}\log_2\frac{4}{4} = 0$$

$$E(outlook = rainy) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971$$

$$w_{sunny} = \frac{5}{14}; w_{overcast} = \frac{4}{14}; w_{rainy} = \frac{5}{14}$$

$$Info\ Gain(outlook)$$

$$= 0.94 - \left[\frac{5}{14} * E(sunny) + \frac{4}{14}E(overcast) + \frac{5}{15}E(rainy)\right] = 0.247$$

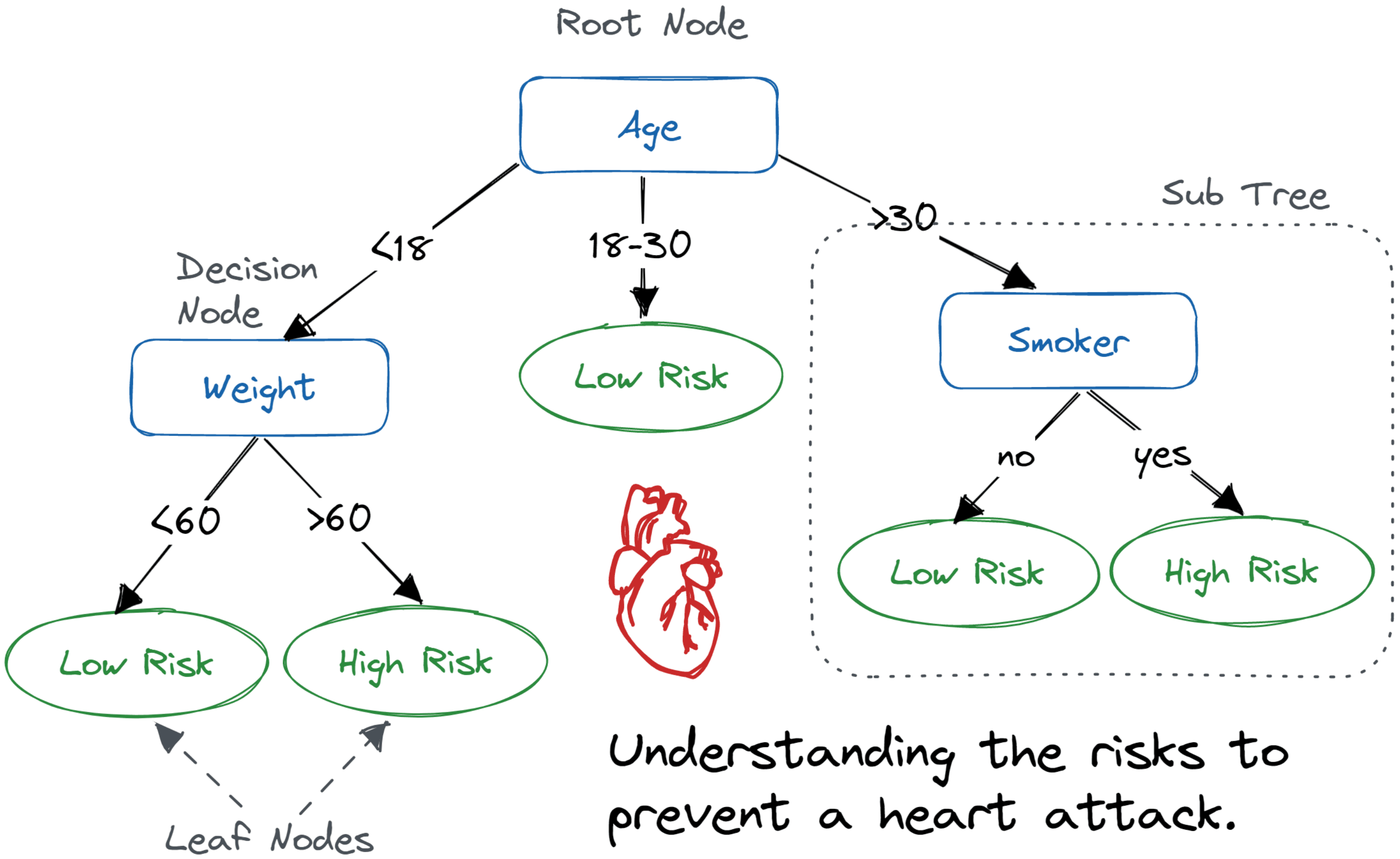
$$Info\ Gain(temp) = 0.029$$

$$Info\ Gain(humidity) = 0.152$$

$$Info\ Gain(windy) = 0.048$$

Select the feature with the highest information gain.

Decision Tree



Source: https://www.datacamp.com/tutorial/decision-tree-classification-python

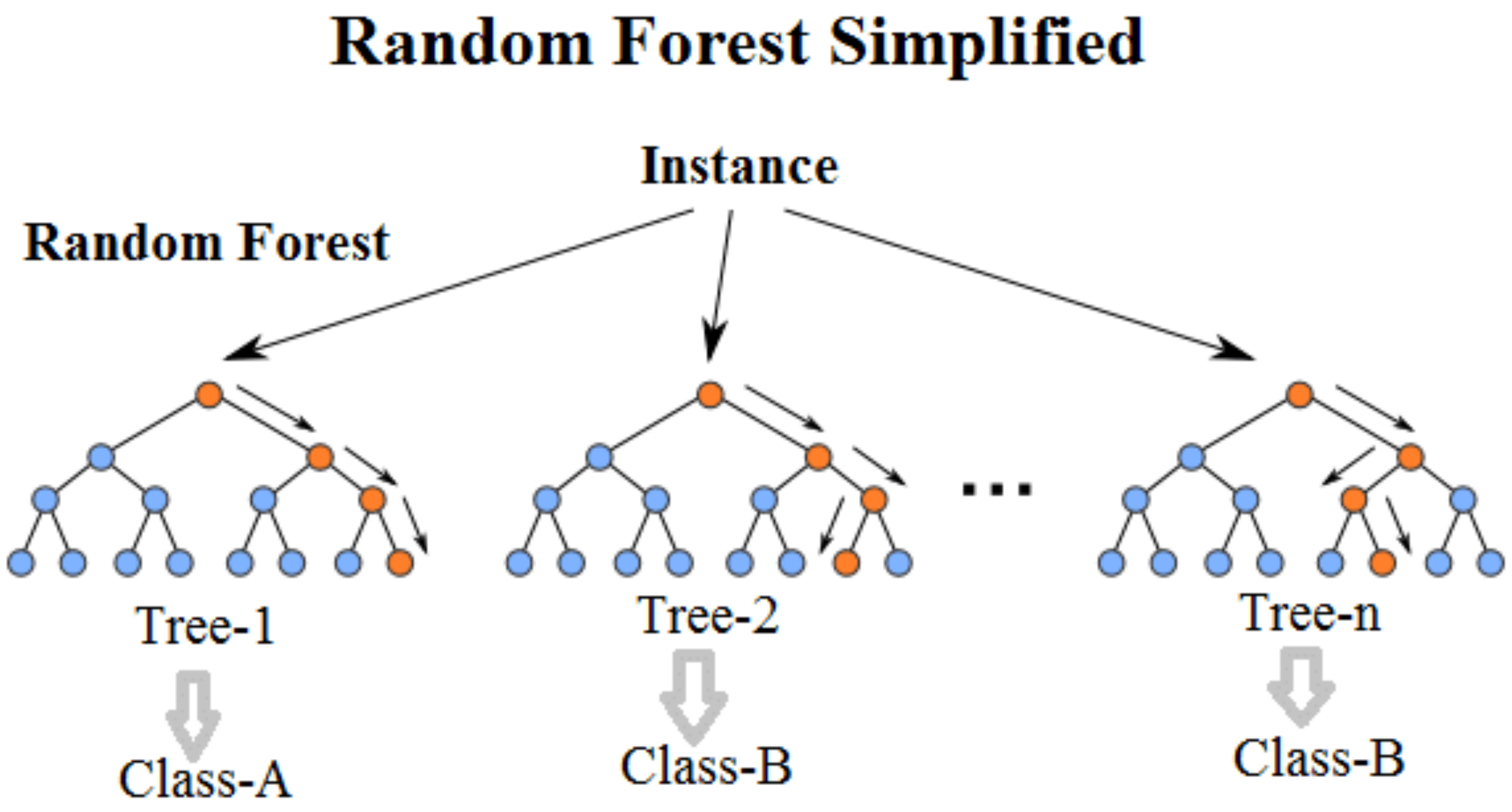
Random Forest

Random Forest = Bagging + Decision Tree

1. Randomly select n samples for a sample set (bagging).
2. Building a decision tree depends on n samples; for each node, randomly select d features for splitting with information gain.
3. Repeat steps 1 and 2 for k times.

- Integrate all prediction results from all trees to determine the classification result with the majority vote.

Random Forest



Source: https://medium.com/chung-yi/ml%E5%85%A5%E9%96%80-%E5%8D%81%E4%B8%83-%E9%9A%A8%E6%A9%9F%E6%A3%AE%E6%9E%97-random-forest-6afc24871857

AdaBoost

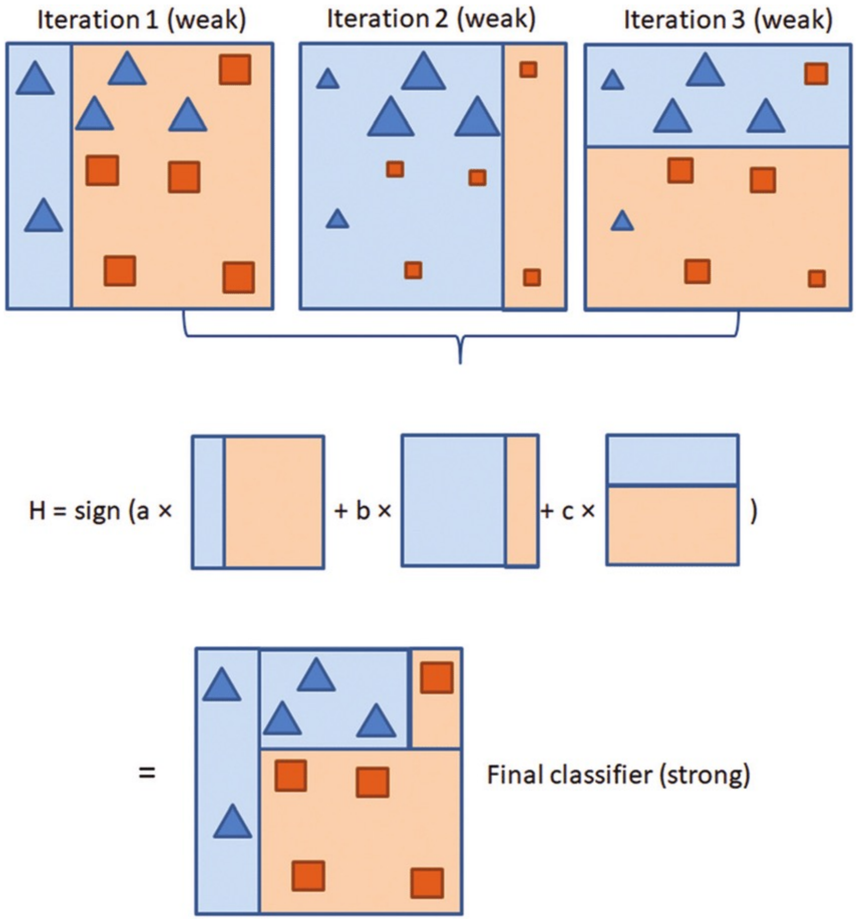
- Adaptive Boosting (AdaBoost) was one of the earliest boosting models developed. It adapts and tries to self-correct in every iteration of the boosting process.
- AdaBoost initially gives the same weight to each dataset. Then, it automatically adjusts the weights of the data points after every decision tree. It gives more weight to incorrectly classified items to correct them for the next round. It repeats the process until the residual error, or the difference between actual and predicted values, falls below an acceptable threshold.

AdaBoost

- Assign every observation, x_i, an initial weight value, $w_i = \frac{1}{n}$, where n is the total number of observations.
- Train a "weak" model (most often a decision tree).
- For each observation:
  - If predicted incorrectly, w_i is increased
  - If predicted correctly, w_i is decreased
- Train a new weak model where observations with greater weights are given more priority.
- Repeat step 3 and 4 until observations perfectly predicted are present number of trees are trained.

Source: https://www.reddit.com/r/learnmachinelearning/comments/8velf3/adaboost_weight_initialization/

AdaBoost



Source:
https://www.researchgate.net/publication/351719735_Exploring_AdaBoost_and_Random_Forests_machine_learning_approaches_for_infrared_pathology_on_unbalanced_data_sets/figures?lo=1&utm_source=google&utm_medium=organic

Input :

- A training set $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m))$.

Initialization :

- Maximum number of iterations T;
- initialize the weight distribution $\forall i \in \{1, \dots, m\}, D^{(1)}(i) = \frac{1}{m}$.

for $t = 1, \dots, T$ **do**

- Learn a classifier $f_t : \mathbb{R}^d \to \{-1, +1\}$ using distribution $D^{(t)}$
- Set $\epsilon_t = \sum_{i: f_t(\mathbf{x}_i) \neq y_i} D^{(t)}(i)$
- Choose $a_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
- Update the weight distribution over examples

$$\forall i \in \{1, \dots, m\}, D^{(t+1)}(i) = \frac{D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}}{Z^{(t)}}$$

where $Z^{(t)} = \sum_{i=1}^{m} D^{(t)}(i) e^{-a_t y_i f_t(\mathbf{x}_i)}$ is a normalization factor such that $D^{(t+1)}$ remains a distribution.

Output : The voted classifier $\forall \mathbf{x}, F(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} a_t f_t(\mathbf{x})\right)$

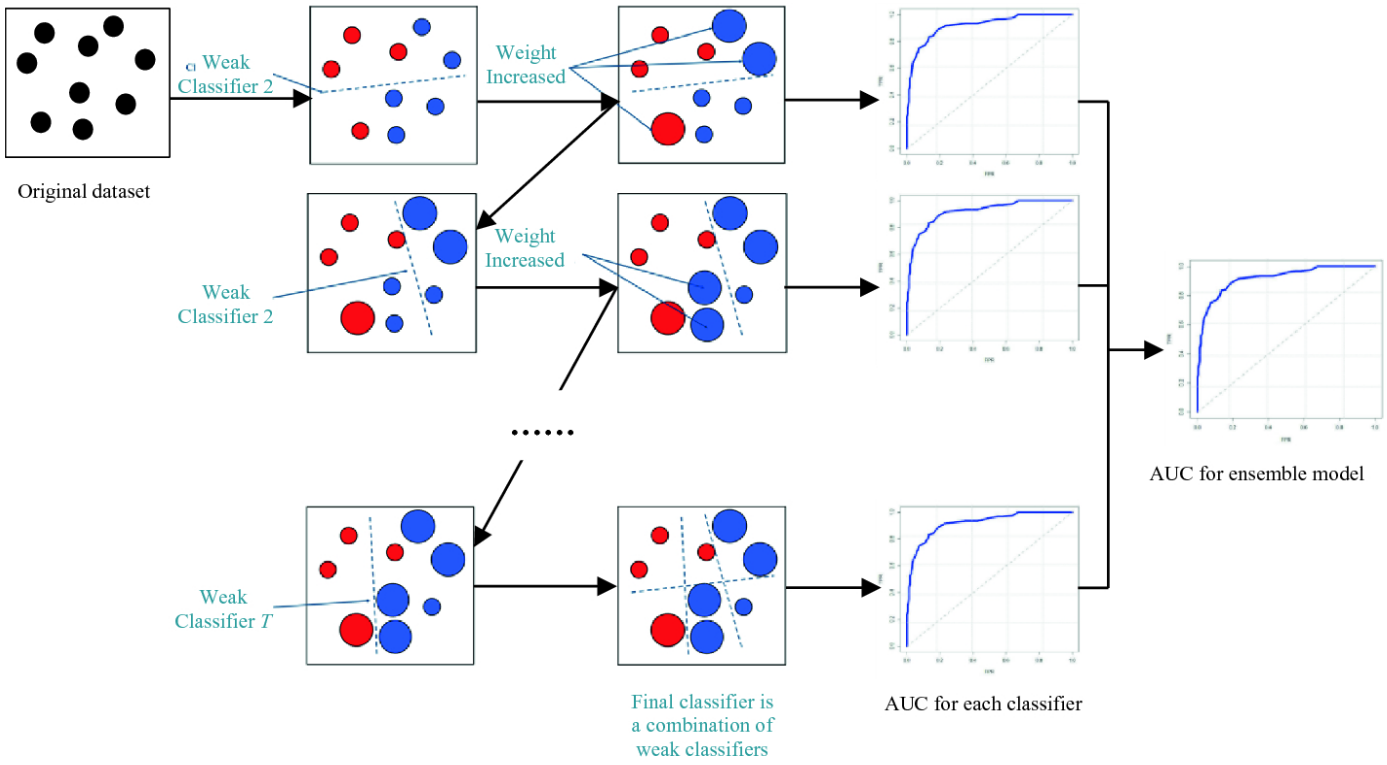
Source: https://ama.liglab.fr/~amini/AdaBoost/

Gradient Boosting

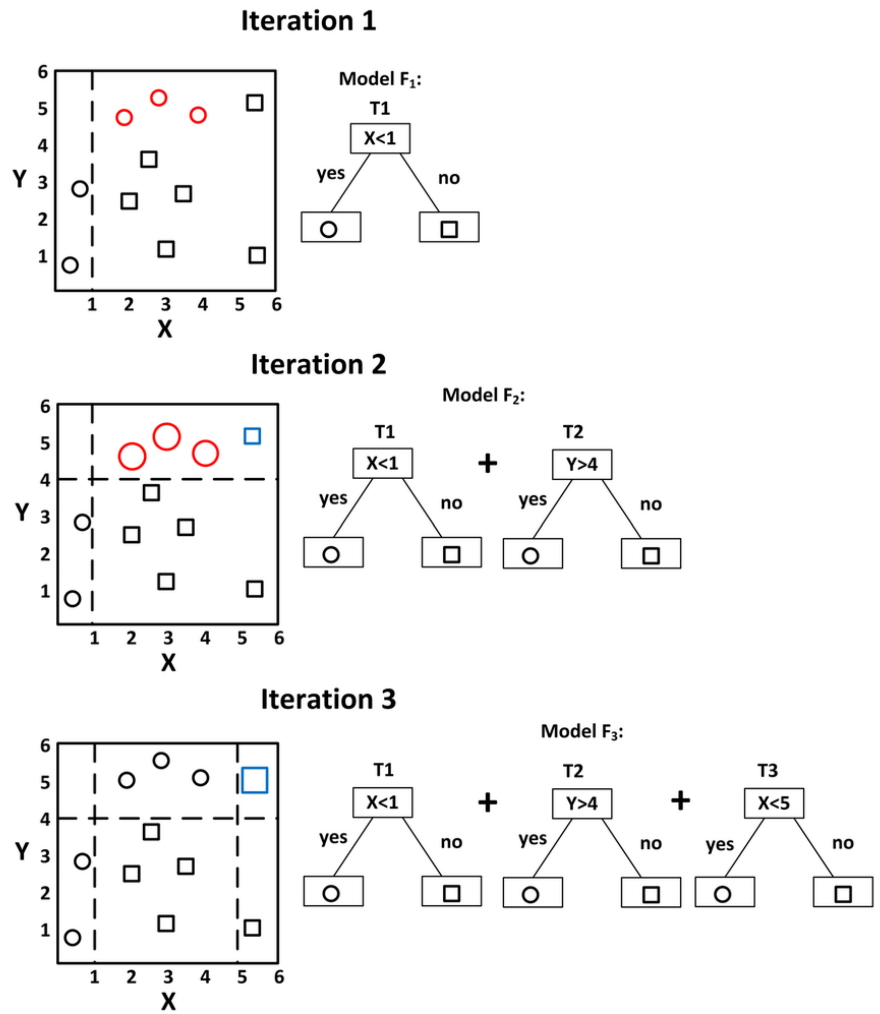
- Gradient Boosting (GB) is similar to AdaBoost in that it, too, is a sequential training technique. The difference between AdaBoost and GB is that GB does not give incorrectly classified items more weight.
- Instead, GB software optimizes the loss function by generating base learners sequentially so that the present base learner is always more effective than the previous one.
- This method attempts to generate accurate results initially instead of correcting errors throughout the process, like AdaBoost.
- For this reason, GB software can lead to more accurate results. Gradient Boosting can help with both classification and regression-based problems.

Source: https://aws.amazon.com/what-is/boosting/

Gradient Boosting



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fdatascience.eu%2Fmachine-learning%2Fgradient-boosting-what-you-need-to-know%2F&psig=AOvVaw3QgdPcdywxRhcvGsTqI3qn&ust=1701741598913000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhxqFwoTCODL3s3X9IIDFQAAAAAdAAAAABAS



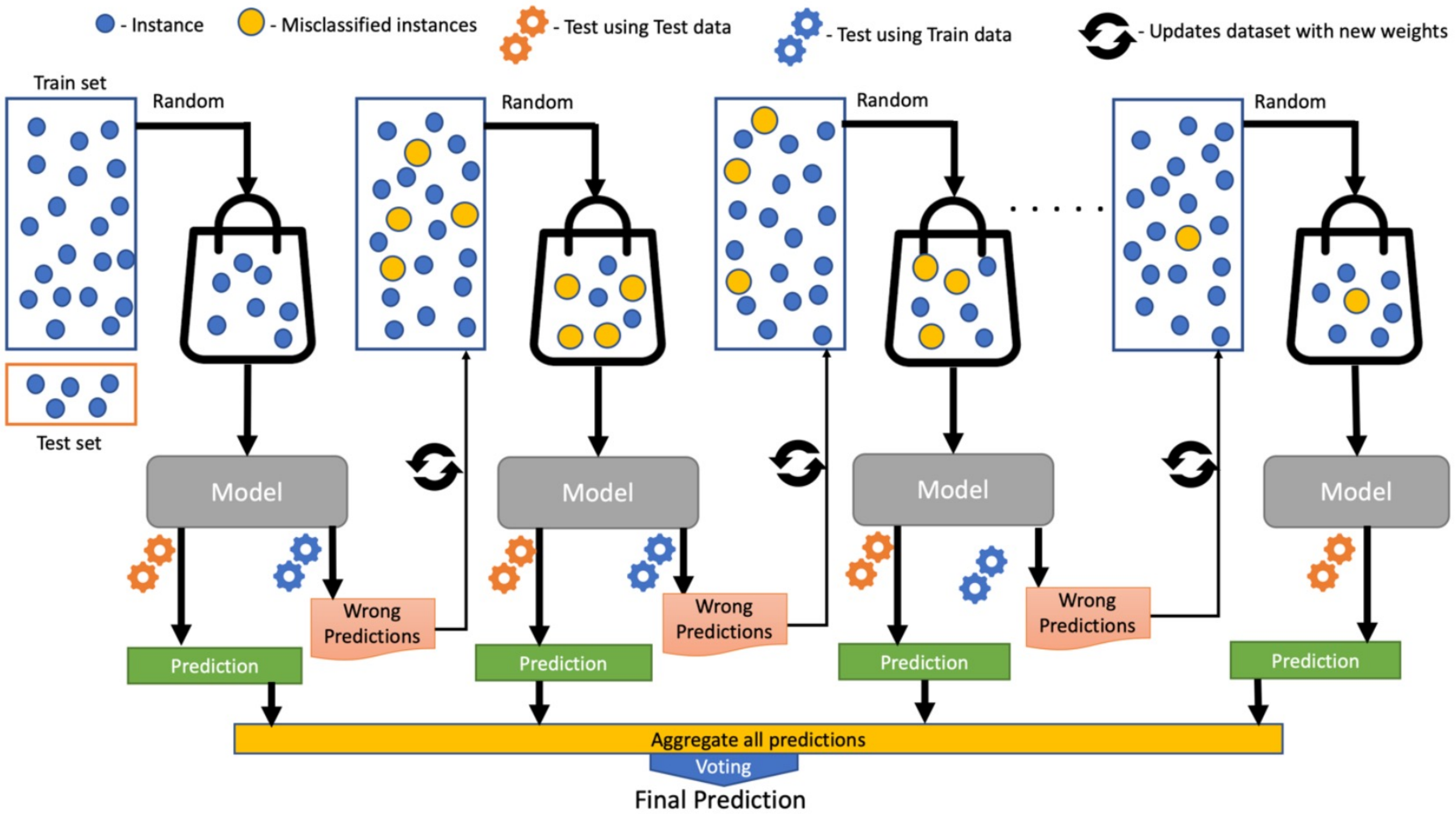
Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FA-simple-example-of-visualizing-gradient-boosting_fig5_326379229&psig=AOvVaw3QgdPcdywxRhcvGsTqI3qn&ust=1701741598913000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhxqFwoTCODL3s3X9IIDFQAAAAAdAAAAABAJ

EXtreme Gradient Boosting (XGBoost)

- Extreme Gradient Boosting (XGBoost) improves gradient boosting for computational speed and scale in several ways.
- XGBoost uses multiple cores on the CPU so that learning can occur in parallel during training. It is a boosting algorithm that can handle extensive datasets, making it attractive for big data applications.
- The key features of XGBoost are parallelization, distributed computing, cache optimization, and out-of-core processing.

Source: https://aws.amazon.com/what-is/boosting/

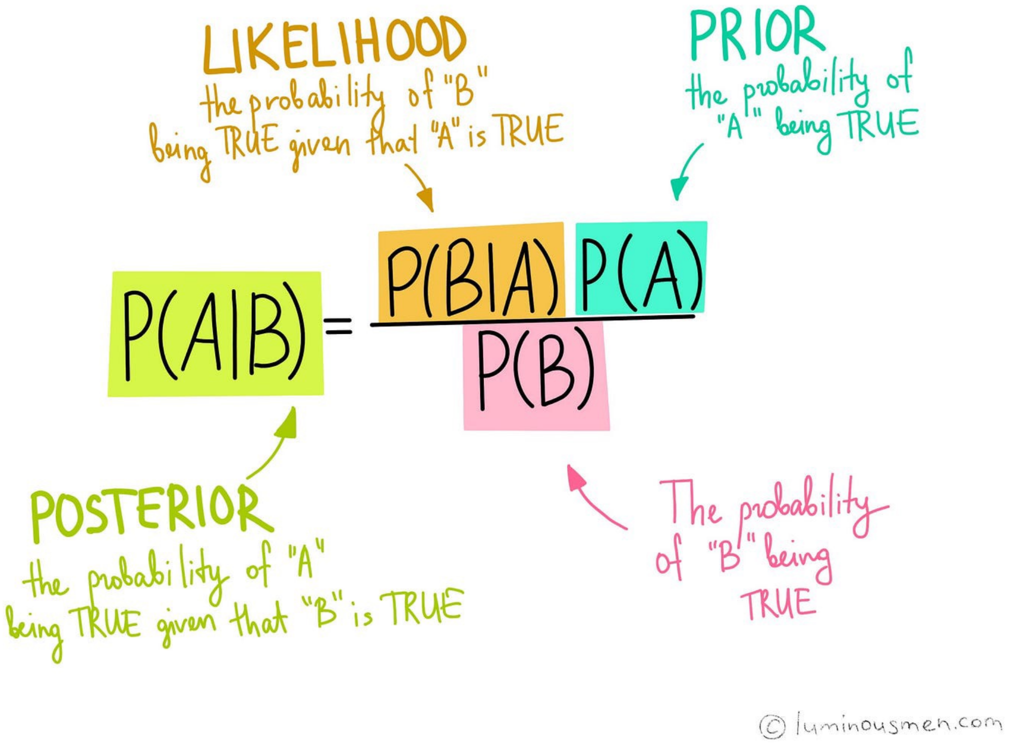
EXtreme Gradient Boosting (XGBoost)



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fdzone.com%2Farticles%2Fxgboost-a-deep-dive-into-boosting&psig=AOvVaw2Mi-Y3kQBhz_4VradEMbkF&ust=1701740885693000&source=images&cd=vfe&opi=89978449&ved=0CBQQjhxqFwoTCIDH_vnU9IIDFQAAAAAdAAAAABAa

Naïve Bayes

- It is a classification technique based on Bayes' Theorem with an independence assumption among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



Source: https://www.google.com/url?sa=i&url=https%3A%2F%2Fheartbeat.comet.ml%2Funderstanding-the-mathematics-behind-naive-bayes-ab6ee85f50d0&psig=AOvVaw1FrZcrDX7LPw6F_TsGyxns&ust=1701751731448000&source=images&cd=vfe&opi=89978449&ved=0CBIQjRxqFwoTCKDToa399IIDFQAAAAAdAAAAABAS
Source: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Naïve Bayes

- **Convert the data set into a frequency table**In this first step data set is converted into a frequency table
- **Create Likelihood table by finding the probabilities**Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.
- **Use Naive Bayesian equation to calculate the posterior probability**Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction.

Naïve Bayes

- Maximize Posterior Probability

$$y^* = \underset{y_i \in y}{\operatorname{argm}ax}\, P(y_i | x_1, x_2, \dots, x_n)$$

- That is

$$y^* = \underset{y_i \in y}{\operatorname{argm}ax} \frac{P(x_1, x_2, \dots, x_n | y_i) P(y_i)}{P(x_1, x_2, \dots, x_n)} = \underset{y_i \in y}{\operatorname{argm}ax}\, P(x_1, x_2, \dots, x_n | y_i) P(y_i)$$

- Assume all features are independent, then

$$P(x_1, x_2, \dots, x_n | y_i) = \prod_{j}^{n} P(x_j | y_i)$$

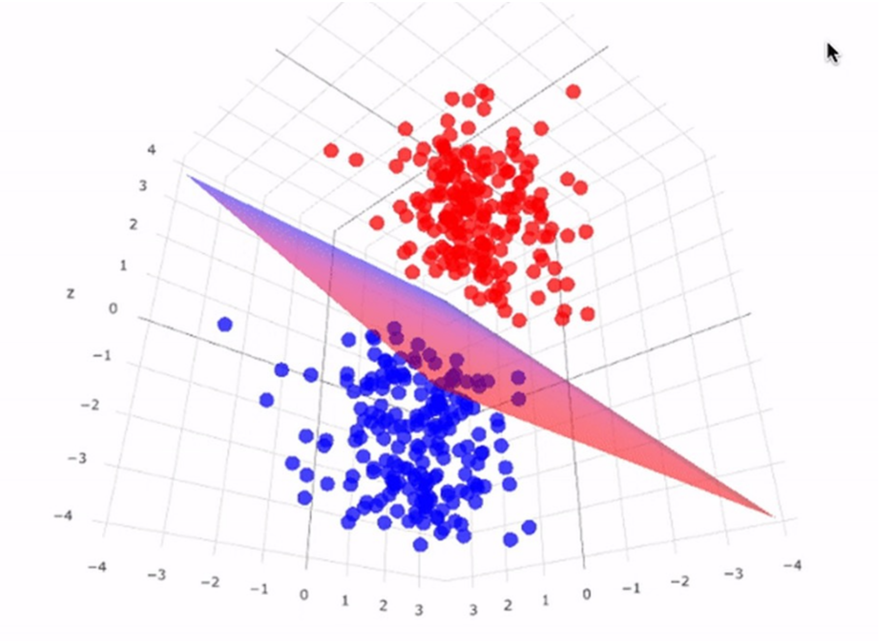
- Therefore, posterior probability maximization could be

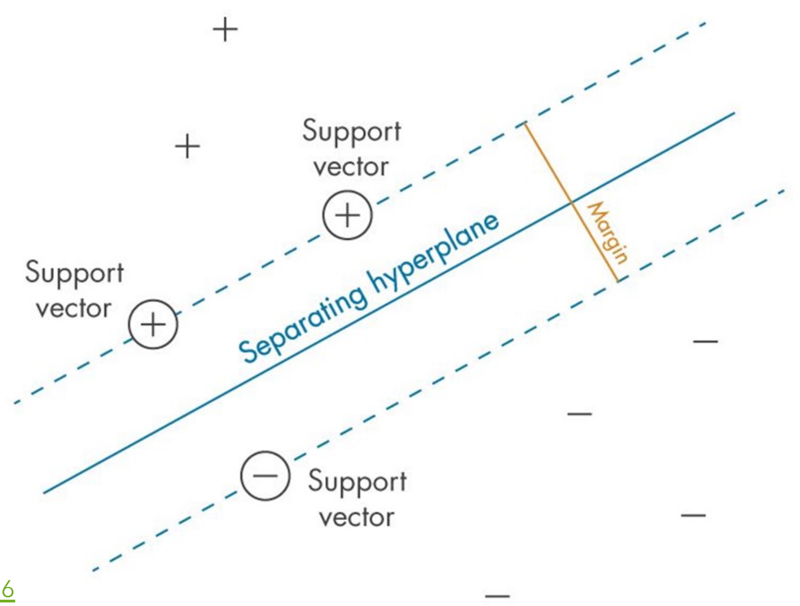
$$y^* = \underset{y_i \in y}{\operatorname{argm}ax}\, P(y_i) \prod_{j}^{n} P(x_j | y_i)$$

Source: https://roger010620.medium.com/%E8%B2%9D%E6%B0%8F%E5%88%86%E9%A1%9E%E5%99%A8-naive-bayes-classifier-%E5%90%ABpython%E5%AF%A6%E4%BD%9C-66701688db02

Linear SVM

- Support Vector Machines (SVMs) are a type of supervised machine learning algorithm used for classification and regression tasks. They are widely used in various fields, including pattern recognition, image analysis, and natural language processing.
- SVMs work by finding the optimal hyperplane that separates data points into different classes.

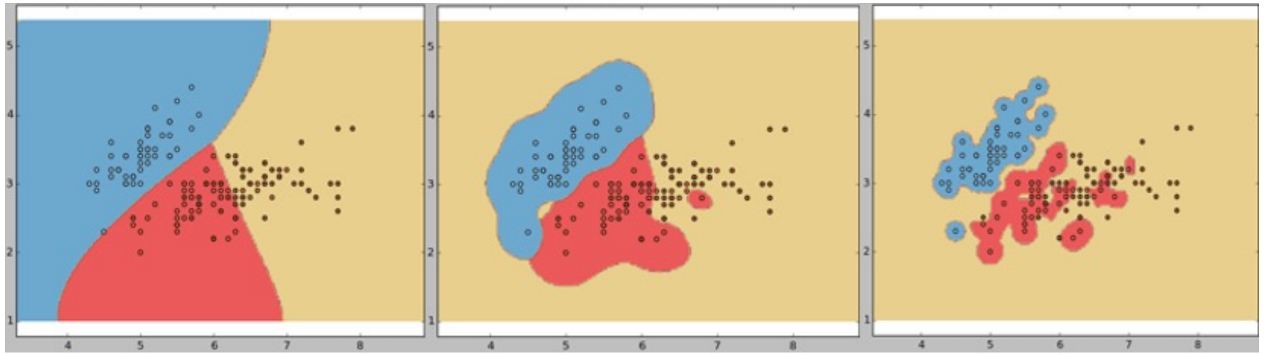


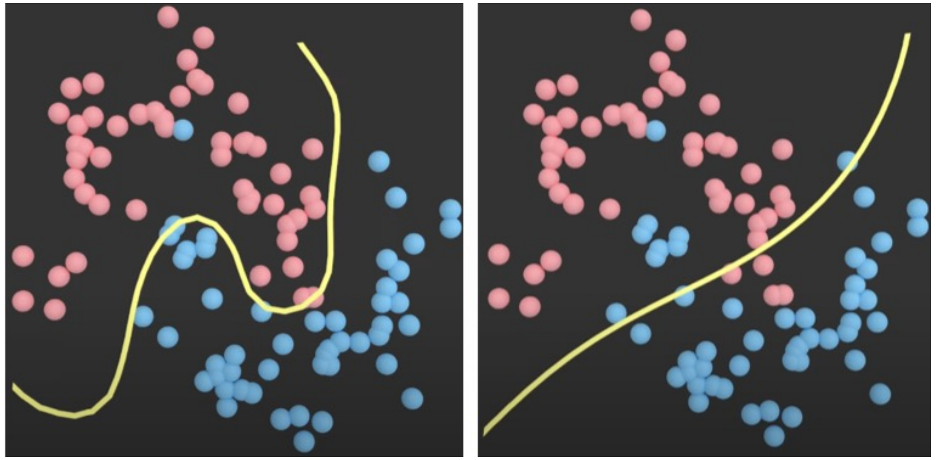


Source: https://medium.com/low-code-for-advanced-data-science/support-vector-machines-svm-an-intuitive-explanation-b084d6238106

Linear SVM

- A hyperplane is a decision boundary that separates data points into different classes in a high-dimensional space. In two-dimensional space, a hyperplane is simply a line that separates the data points into two classes. In three-dimensional space, a hyperplane is a plane that separates the data points into two classes. Similarly, in N-dimensional space, a hyperplane has (N-1)-dimensions.





Remarks

- Every machine learning classifier has its advantage in dealing with specific problems; therefore, it is difficult to say which model is the best one.
- In general, we will enroll all classifiers into the analysis and select the best one.
- But in fact, some of them usually perform very well in most cases: XGBoost and Random Forest.

Question Time

- **Assignment:**
- **Download today's lab practice and upload to moodle.**
- **Thx**

References:
https://scikit-learn.org/stable/modules/clustering.html#overview-of-clustering-methods

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw
Web: toodou.github.io

SCAN ME