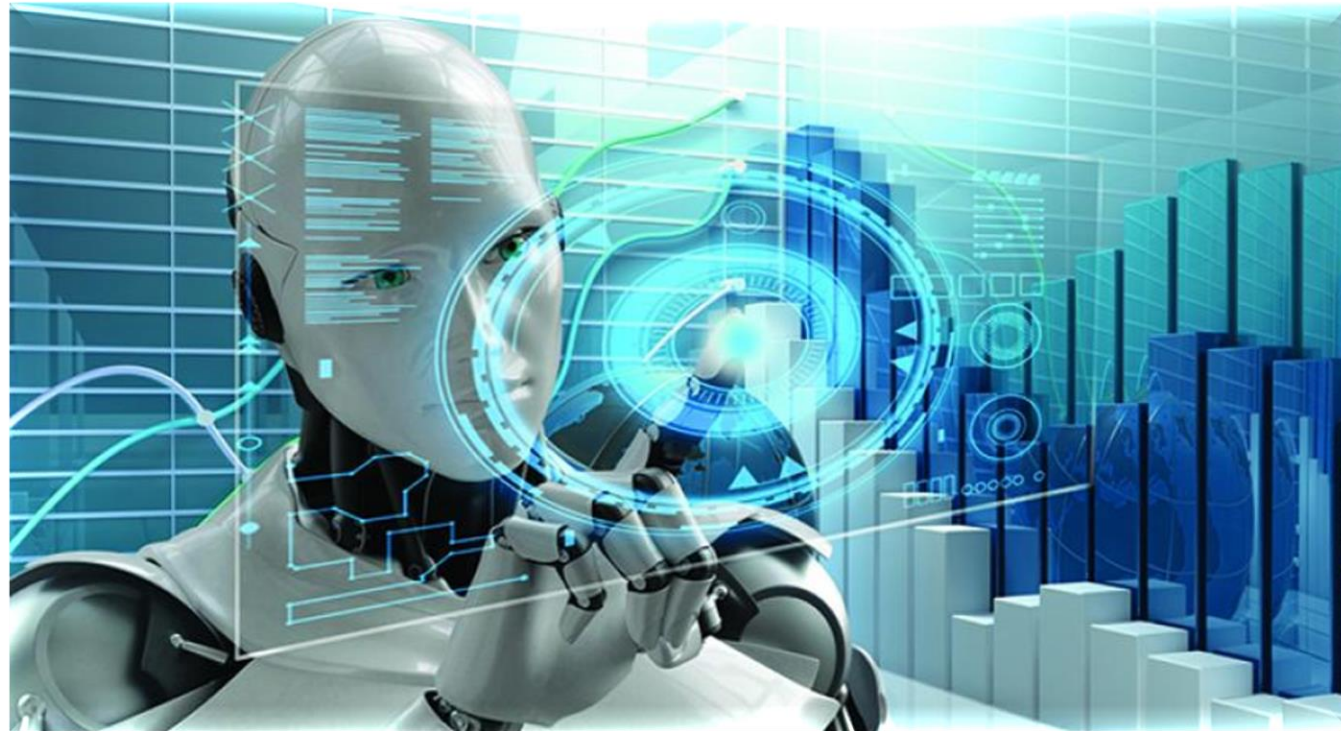


INTRODUCTION TO MACHINE LEARNING

DATA 602 Lecture 7



What is Ensemble Machine Learning?

- **Ensemble learning** combines the outputs of multiple learners to increase the prediction power of a system
- **Ensemble learning** models are designed to increase accuracy and improve computation time
- **Ensemble methods** are meta-algorithms that combine several machine learning techniques into one predictive model in order to
 - Decrease variance (**bagging**)
 - Reduce bias (**boosting**), and/or
 - Improve predictions (**stacking**)

Advantages

- **Ensemble learning** helps improve predictions by combining several models
- This approach allows the production of better predictive performance compared to a single model
- **Ensemble methods** can handle mixed types of target variables and predictors with very little feature processing (i.e. missing values)
- There are no complex hyperparameters to adjust
- It is possible to visualize the prediction process through trees

Differences between Regression and Classification

- In the case of **regression**, **ensemble models** improve predictions by replicating trees and **averaging** results
- In the case of **classification**, **ensemble models** serve as **voting** systems: the most frequent response is selected as the output for all replications
- In the case of **regression**, the **standard deviation** provides confidence measurement about the prediction
- In the case of **classification**, the **percentage of trees predicting a certain class** is indicative of the level of confidence in the prediction
 - However, the percentage cannot be used as a probability estimate because it is the outcome of a **voting** system

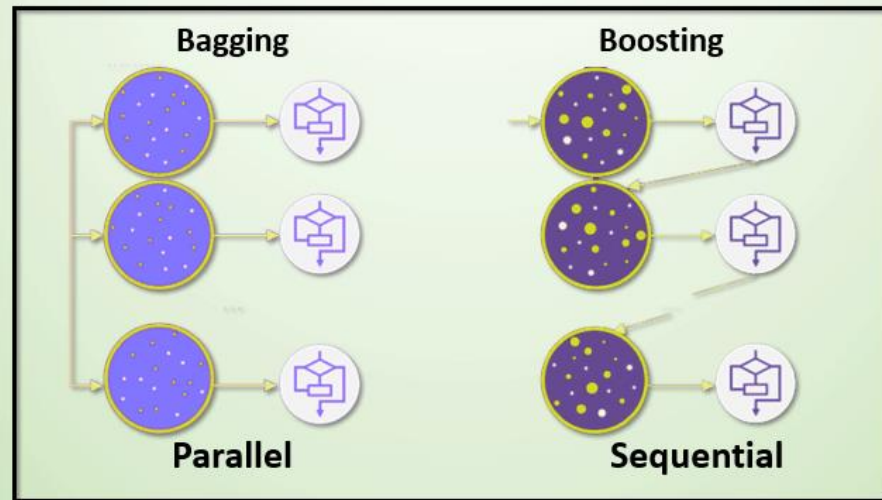
Ensemble methods can be divided into two groups:

- **Sequential** ensemble methods
 - The **base learners** are generated sequentially (e.g. **AdaBoost**)
 - The basic motivation of sequential methods is to **exploit the dependence between the base learners**
 - The overall performance can be boosted by **weighing** previously mislabeled examples with higher weights
- **Parallel** ensemble methods
 - The **base learners** are generated in parallel (e.g. **Random Forest**)
 - The basic motivation of parallel methods is to **exploit independence between the base learners** since the error can be reduced dramatically by **averaging**

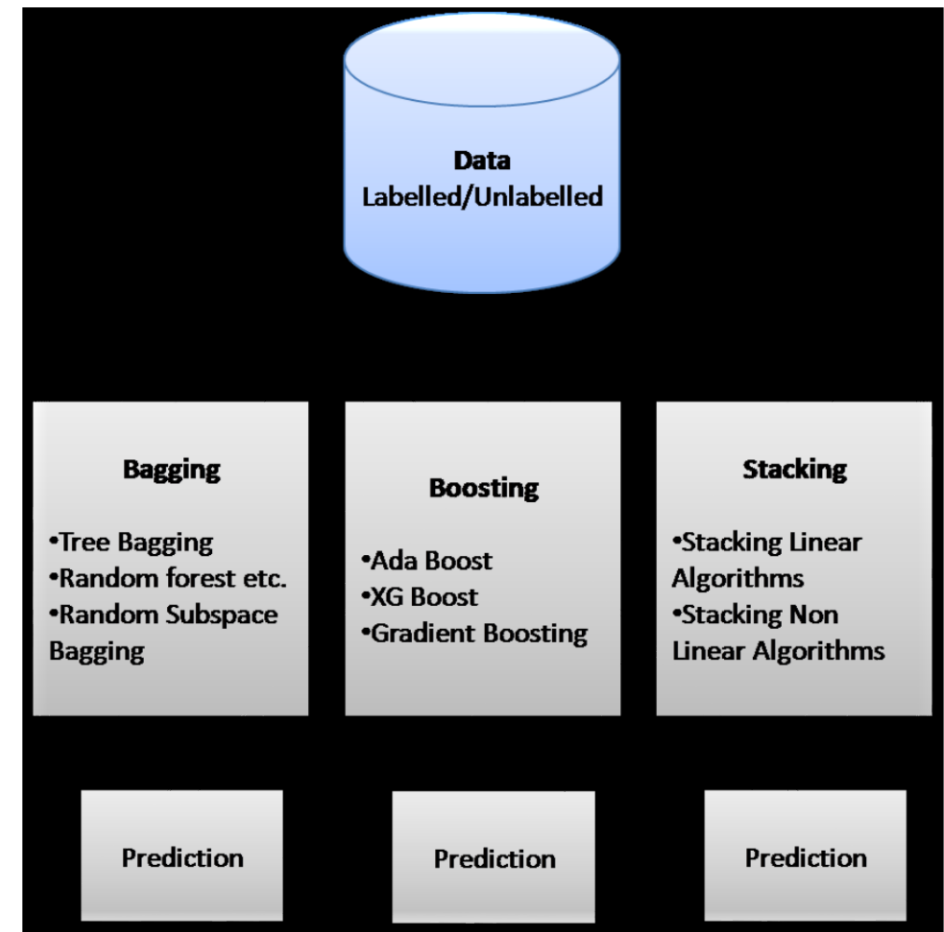
There are two types of **ensemble methods**

- **Those that combine models of similar types: bagging** (bootstrap aggregation) and **boosting**
- **Those that combine multiple models of various types: voting** and **blending** or **stacking**

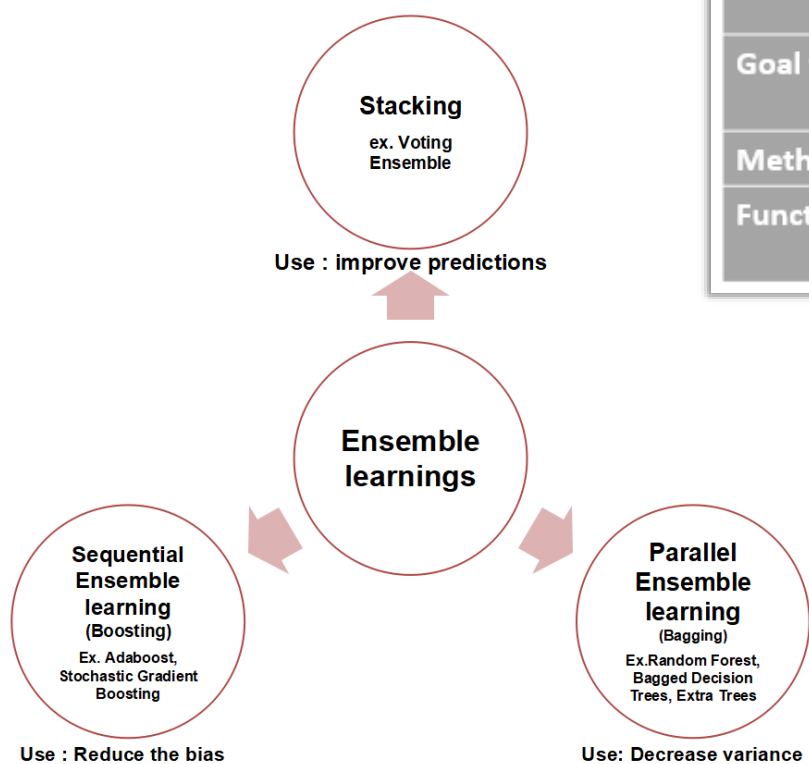
Bagging and Boosting



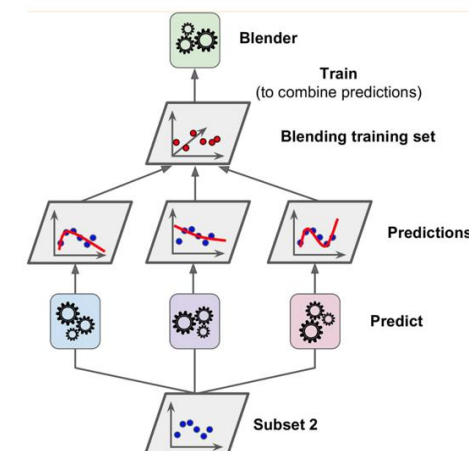
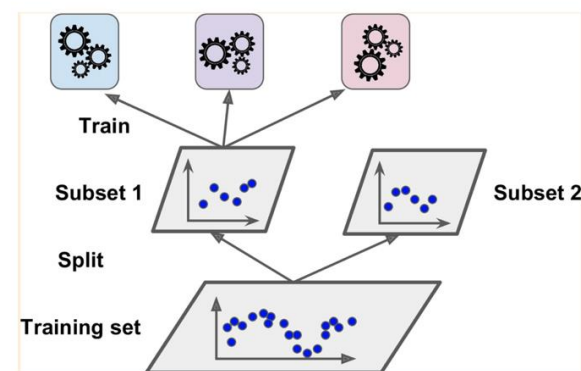
www.educba.com



Ensemble Learning Algorithms



	Bagging	Boosting	Stacking
Partitioning of the data into subsets	Random	Giving <u>mis-classified</u> samples higher preference	Various
Goal to achieve	Minimize variance	Increase predictive force	Both
Methods where this is used	Random subspace	Gradient descent	Blending
Function to combine single models	(Weighted) average	Weighted majority vote	Logistic regression

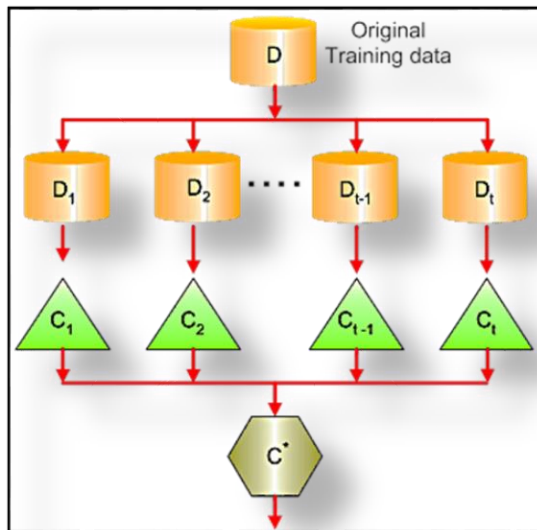




Bagging

Bagging was proposed by Leo Breiman in 1994

- **Bagging** stands for **bootstrap aggregation**
- The **training data** are **split** into **multiple samples with replacement**
- It is a model aggregation technique **to reduce model variance**
- One way to reduce the variance of an estimate is to **average together multiple estimates**
- **Bagging** uses **bootstrap sampling** to obtain the data subsets for training the base learners
- To **aggregate** the outputs of base learners, bagging uses **voting for classification** and **averaging for regression**
- **Bagging** is effective when the models created are different from each other
- **Independent models** on each of the bootstrap samples are built and the **average** of the predictions for regression or **majority vote** for classification is used to create the final model



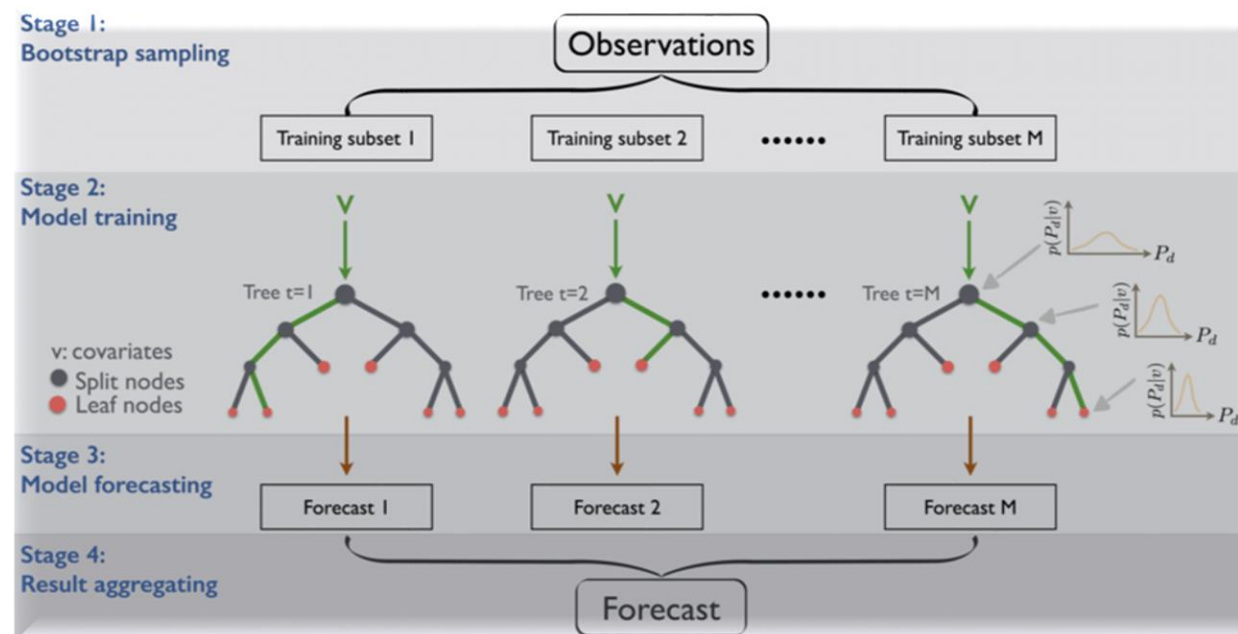
1. You start with the original dataset
2. You create multiple datasets
3. You build multiple classifiers
4. You combine the classifiers (average or vote)

$C_{\text{final}} = \text{aggregate max of } y \sum_i I(C_i = y)$
 with N as the number of bootstrap samples for $i = 1$ to N and model C_i

To **improve the bagging model results**, you need to adjust three main tuning parameters:

- `n_estimators`: It is the number of trees—the larger, the better—although results do not improve significantly beyond a certain point
- `max_features`: this is the random subset of features to be used for splitting a node—the lower, the better to reduce variance, but this may increase bias. In a regression, `max_features` should be equal to `n_features`
- `n_jobs`: number of cores to be used for parallel construction of trees. If set to -1, all available cores in the system are used, or you can specify the number

- **Bootstrap sampling** is a method that relies on **random sampling with replacement**
- This is different from **pasting**, which is **sampling without replacement**
- **Bootstrapping** makes it possible to sample examples from a set to create a new one multiple times
- It makes it possible to assign measures of accuracy (bias, error) to estimates
- Inference about a sampled population can be modelled by **resampling** the sample and performing inference about the re-sampled data
- Results improve when trees are independent from one another (i.e. they are not correlated)
- **Bootstrapping** reduces variance by inducing variations in otherwise similar predictors



- **Random Forests (RF)** and **Extremely Randomized Trees** are two examples of **bagging** algorithms:
 - In **Random Forests**, each tree in the ensemble is built from **a sample drawn with replacement** (i.e. a **bootstrap** sample) from the training set
 - In addition, instead of using all the features, a **random subset of features** is selected, further randomizing the tree
 - As a result, the **bias** of the forest **increases slightly**. However, due to the **averaging of less correlated trees**, its **variance decreases**, which results in an overall better model
 - **Random Forest** models require **little pre-processing** and **few hyperparameters**
 - **Random Forest** models fit complex target functions with little risk of overfitting
 - With **Random Forest**, each tree is **independent** from others and can be built in **parallel** to others (i.e. they are uncorrelated)
- In an **Extremely Randomized Trees** algorithm, the splitting thresholds are randomized
 - This usually allows a further **reduction of the model variance**, at the expense of a **slightly greater increase in bias**



Boosting



The **Boosting algorithm** was introduced by **Freud and Shapire in 1995** with **Adaptive Boosting (AdaBoost)**

- **Boosting** refers to a family of algorithms that can **convert weak learners to strong learners**
- **Boosting** builds models **sequentially** and training each model involves using information from previous ones
- One way to reduce the variance of an estimate is to **average together multiple estimates**
- The main principle of boosting is to **fit a sequence of weak learners**– models that are only slightly better than random guessing, such as small decision trees– **to weighted versions of the data**
- More weight is given to examples that were misclassified by earlier rounds
- The predictions are then combined through a **weighted majority vote (classification)** or a **weighted sum (regression)** to produce the final prediction
- The principal difference between **boosting** and the **committee methods, such as bagging**, is that **base learners** are trained in **sequence** on a **weighted** version of the data
- The main algorithms using boosting are
 - **AdaBoost**, or Adaptive Boosting, used high-weight data points
 - **Gradient Tree Boosting** is a generalization of boosting to arbitrary differentiable loss functions
 - It can be used for both **regression** and **classification** problems
 - **Gradient Boosting** builds the model in a sequential way
 - It identifies **shortcomings of a weak learner** by **gradients**

The picture can't be displayed.



1. Assign uniform weights for all data points $W_0(x) = \frac{1}{N}$ where N is the number of training data points
2. At each iteration, fit a classifier $y_m(x_n)$ to the training data and update weights to minimize the weighted error function
3. The final model is given by $Y_m = \text{sign}[\sum_{m=1}^M \alpha_m y_m(x)]$

The weight is calculated as $W_n^{m+1} = W_n^m \exp(\alpha_m y_m(x_n)) \neq tn$

- **AdaBoost** tweaks the **instance weights at every interaction**
 - Models that produce more prediction errors are weighted less
 - It is easier to reduce the cost function of the learning function by working on the examples that weigh more
 - In case of misclassification, each example has its weight increased and, in the next iteration, examples with higher weights influence the model such that

$$h(x) = \text{sign}[\sum_{m=1}^M \alpha_m h_m(x)]$$

- With **AdaBoost**, the prediction function $h(x)$ is a **vector of signs** (positive or negative) that point out classes in a **binary direction**
- The weight is determined by the **misclassified errors** from the previous models and $\alpha_m = \frac{1}{2} \log\left[\frac{(1 - \text{err}_m)}{\text{err}_m}\right]$

Gradient Boosting

- **Gradient Boosting** works by **sequentially adding** the previous predictors' underfitted predictions to the ensemble, ensuring the errors made previously are corrected
- **Gradient Boosting** tries to **fit the new predictor to the residual errors** made by the previous predictor
- Here is how the **Gradient Boosting** model works:
 - A model is built on a subset of data
 - Using this model, predictions are made on the whole dataset
 - Errors are calculated by comparing the predictions and actual values
 - A new model is created using the errors calculated as target variable. The objective is to find the best split to minimize the error
 - The predictions made by this new model are combined with the predictions of the previous one
 - New errors are calculated using this predicted value and actual value
 - This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached

Gradient Boosting

- **Gradient Boosting** works with ranking, classification, and regression with each problem using a different cost function
- **Gradient Boosting** helps identify the set of weights w_m that reduce the cost functions
- **Gradient Boosting** relies on
 - **Weights optimized by gradient descent**
 - **Shrinkage** that works as **learning rate**
 - **Sub-sampling** that emulates the **pasting** approach (i.e. sampling without replacement a proportion of the training set)
 - **Trees of fixed size**, which reduces complexity
- The formula for **gradient boosting** is

$$f(x) = \sum_{m=1}^M v * hm(x; w_m)$$

with v as the shrinkage factor, M as the number of total models, h as the final functions as the sum of a series of M models

- The weight w_m is determined by gradient descent optimization, not by the misclassification error from previous models

Gradient Boosting's essential tuning parameters are

- `n_estimators`: number of weak learners to be built
- `max_depth`: the maximum depth of the individual estimators
- `min_samples_leaf`: this ensures a sufficient number of samples results in a leaf
- `subsample`: this is the fraction of the sample to be used for fitting individual models (default = 1). Typically, .8 (80%) is used to introduce a random selection of samples, which in turn increases the robustness against overfitting
- `learning_rate`: it is a regularization parameter which controls the magnitude of change in estimators. A lower learning rate is desirable, which requires higher `n_estimators`

- **Extreme Gradient Boosting** or **XGBoost** is a variant of **Gradient Boosting** was introduced by Tianqui Chen in 2014
 - It includes a variety of regularization, which reduces overfitting and improves overall performance
 - It implements parallel processing being much faster than Gradient Boosting
 - It allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model. XGBoost allows a user to run a **cross-validation** at each iteration of the boosting process
 - It has built-in routine to handle missing values
 - It will split the tree up to a maximum depth, unlike gradient boosting where it stops splitting the node on encountering a negative loss in the split
- **XGBoost** has a bundle of tuning parameters
 - **General parameters:** nthread (number of parallel threads), Booster
 - **Boosting parameters:** eta (learning rate; 0.3 is the default), max_depth, min_child_weight (min sum of weights of all observations required in a child), colsample_bytree (fraction of columns to be randomly sampled for each tree), subsample, lambda (L2 regularization with default of 1), alpha (L1 regularization on weight)
 - **Task parameters:** objective (the loss function to be minimized), eval_metric (metrics to validate the model)



Stacking



Stacking was introduced by David Wolpert in 1992

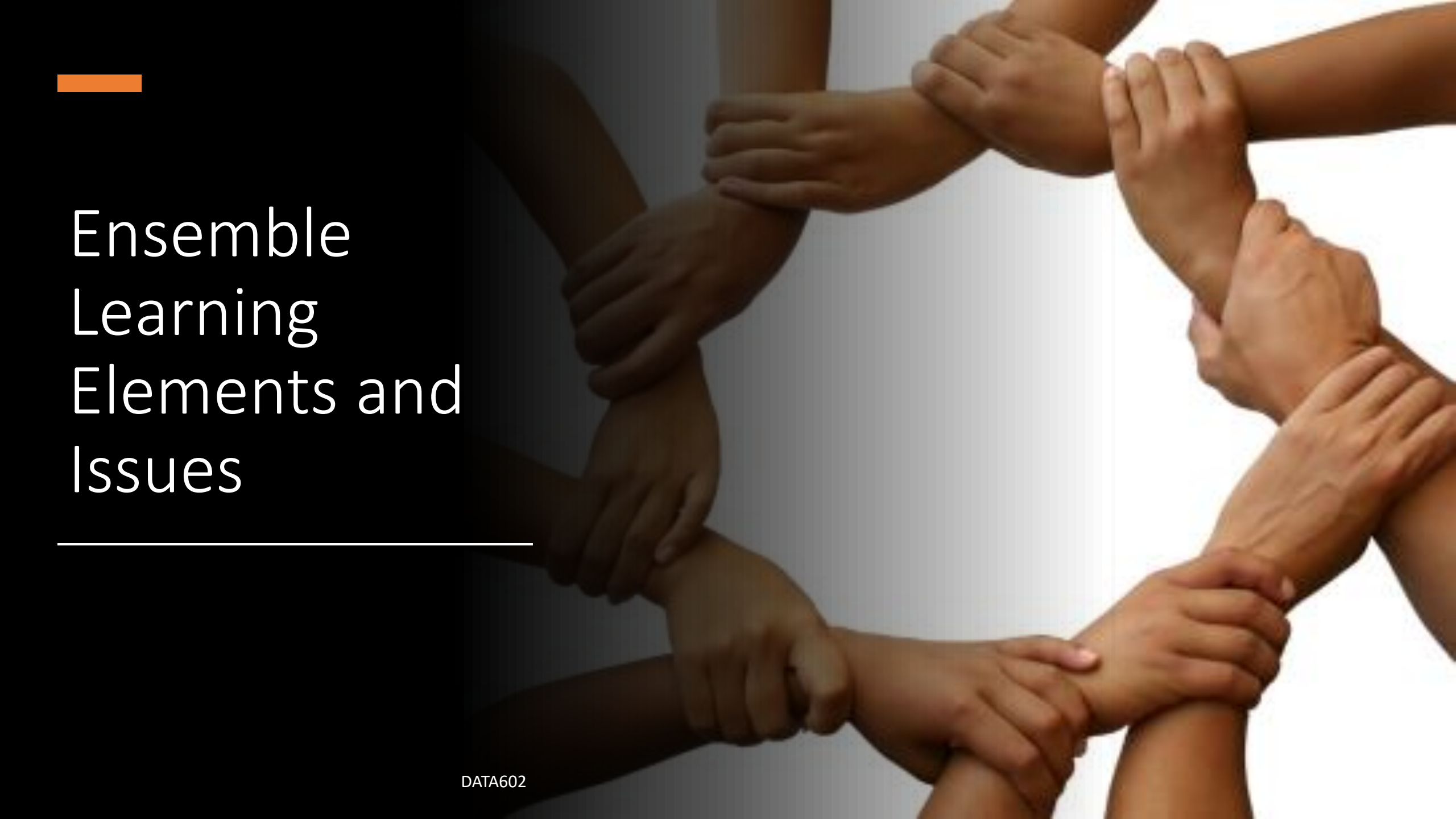
- **Stacking** is an ensemble learning technique that **combines multiple classification or regression models** via a **meta-classifier** or a **meta-regressor**
- **Stacking** is an ensemble learning technique that **uses predictions from multiple models** (for example, decision tree, K-Nearest Neighbors, or Support Vector Machine) to **build a new model**. This model is used for **making predictions** on the test set
- The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features
- The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous
- **Stacking** is a commonly used technique for winning the Kaggle data science competition



Blending

Blending

- **Blending** follows the same approach as **stacking** but uses only a **holdout** (validation) set from the train set to make predictions
- In other words, unlike **stacking**, the predictions are made on the **holdout** set only
- **A hold-out set** is designed to train the blender
 - The training dataset is divided into a pair of subsets. The first one trains the first layer predictors which produce predictions for the second set known as held-out set
- **The training set is divided into three subsets** rather than two
 - The first is to train layer one
 - The second creates the set needed for training layer two (using the predictions from layer one)
 - The third does the same for layer three using the predictions from layer two
 - You go through the three layers in sequence to predict a new instance



Ensemble Learning Elements and Issues

- The **Gini impurity measure** is one of the methods used in decision tree algorithms to identify the optimal split from a root node and subsequent splits
- **Gini impurity** determines the probability of misclassifying an observation
- **Gini impurity** is the probability of *incorrectly* classifying a randomly chosen element in the dataset if it were randomly labeled *according to the class distribution* in the dataset
- If we have C total classes and $p(i)$ is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as

$$G = \sum_{i=1}^C p(i) * (1 - p(i))$$

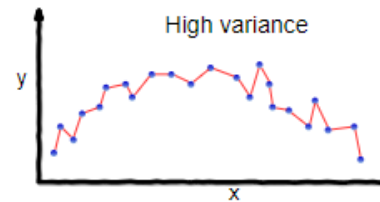
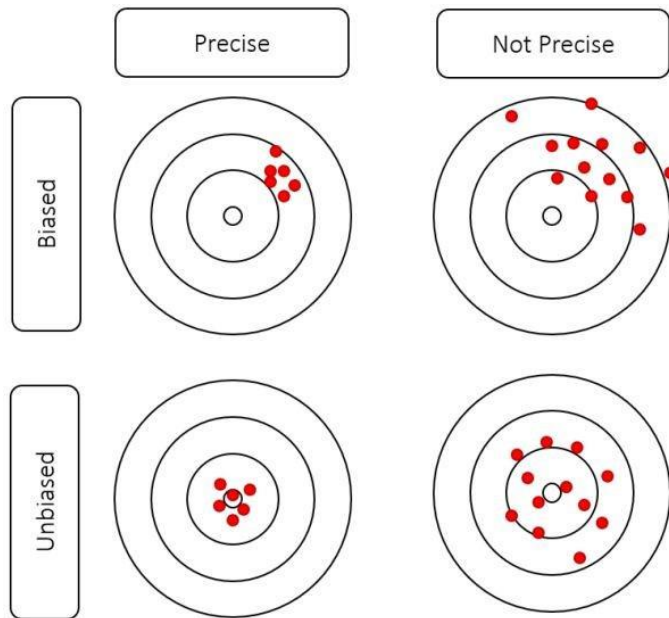
- For the example above, if we have $C = 2$ and $p(1) = p(2) = 0.5$, then $G = 0.5$
- **A Gini impurity of 0 is the lowest and best possible impurity outcome.** It can only be achieved when all elements belong to the same class
- We determine the quality of the split by **weighting the impurity of each branch based how many elements each branch has**
- When training a decision tree, the best split is chosen by **maximizing the Gini gain**, which is calculated by subtracting the weighted impurities of the branches from the original impurity

- Scikit-learn's **Random Forest** model has a **feature_importance_** attribute that gives the value of **Gini impurity** reduction caused by each feature across all levels normalized across trees
- The line of code is as follows: `feature_importance.sort_values(by='importance', ascending=False)`
- The only hyperparameter of interest here is the number of base learners
- **Random Forests** seldom overfit, usually, they saturate with an increasing number of base learners, increasing computational overhead without deteriorating performance
- The relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable
- Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples
- The **expected fraction of the samples** they contribute to can be used as an estimate of the **relative importance of the features**
- In scikit-learn, the fraction of samples a feature is combined with the decrease in impurity from splitting them to create a normalized estimate of the predictive power of that feature
- By **averaging** the estimates of predictive ability over several randomized trees, we can **reduce the variance** of such an estimate and use it for feature selection. This is known as the **mean decrease in impurity**, or **MDI**

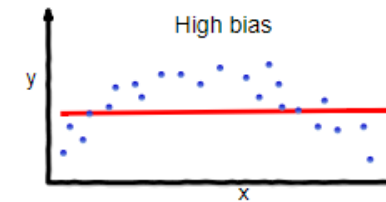
- **Majority voting** is known as **hard voting**
- The **argmax of the sum of predicted probabilities** is known as **soft voting**
 - Parameter weights can be used to assign specific weight to classifiers
 - The predicted class probabilities for each classifier are multiplied by the classifier weight and averaged
 - The final class label is derived from the highest average probability class label
 - Assume we assign an equal weight of 1 to all classifiers. Based on soft voting, the predicted class label is 1, as it has the highest average probability

Are Ensemble Learning Methods Better?

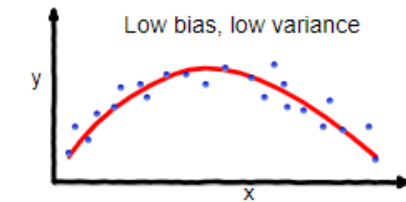
- **Ensemble methods** combine several **tree base algorithms** to construct better **predictive performance** than a single **tree base algorithm**
- The main principle behind the **ensemble model** is that a group of **weak learners** come together to form a **strong learner**, thus increasing the **accuracy** of the model
- When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are **noise, variance, and bias**
- **Ensemble** helps to reduce these factors (**except noise**, which is **irreducible error**)



overfitting



underfitting



Good balance

Bias Error determines how much, on an average, the predicted value is different from the actual value

Variance error measures differences in predictions from same observations. A model with high variance will overfit on training data, which is not desirable

- The role of **bagging models** is to create **independent prediction** models by generating trees that are **not correlated**
 - The process generate **independent** trees using **bootstrapping** in a **parallel** design
- With **boosting**, the predictors are chained together in a **serial** design
 - Weak learners are **boosted** at each step toward the goals of a working predicting ensemble without looking back once the outcomes are aggregated
 - **Boosting** relies on **biased** models that can predict **simple targets**