Advanced Machine Learning Method & Tools

Analytiks

Advanced Machine Learning Method & Tools

Topics

- Pipelines
- Cross Validation
- Reducing Prediction Errors
- Model Optimization
- Ensemble Models

Pipelines

Scikit-Learn Pipelines

- Tool for managing machine learning workflows
- String a number of data preprocessing tasks (e.g. scaling, PCA) together with estimators
- Useful for:
    - Creating a coherent and easy-to-follow workflow
    - Enforcing order in pre-processing steps
    - Make pre-processing reproducible across different data and combinations
    - Automate standard workflows
    - Easily check model's reaction to differently processed data
    - Prevents leaking data from your training dataset to your test dataset
    - Allows pre-processing to be constrained on each fold of cross validation

LAB: Pipelines

Cross-Validation

Cross Validation

- Cross validation checks how well a model generalizes to new data
- Divide our data into a large training set and a smaller validation set, then train on the training set and use the validation set to measure our accuracy - generally called the “Holdout Method”
- K-Fold - a popular method of performing cross validation
  - Randomly divide the data into several equal subsets
  - Iteratively create and test predictive models such that each of the subsets is withheld and used for model testing one time while the remaining subsets are used to train the model.
  - “K” is the number of iterations used.

Model Optimization: Cross Validation

EXERCISE: Cross Validation w/ Pipeline

Reducing Prediction Errors

Prediction Errors

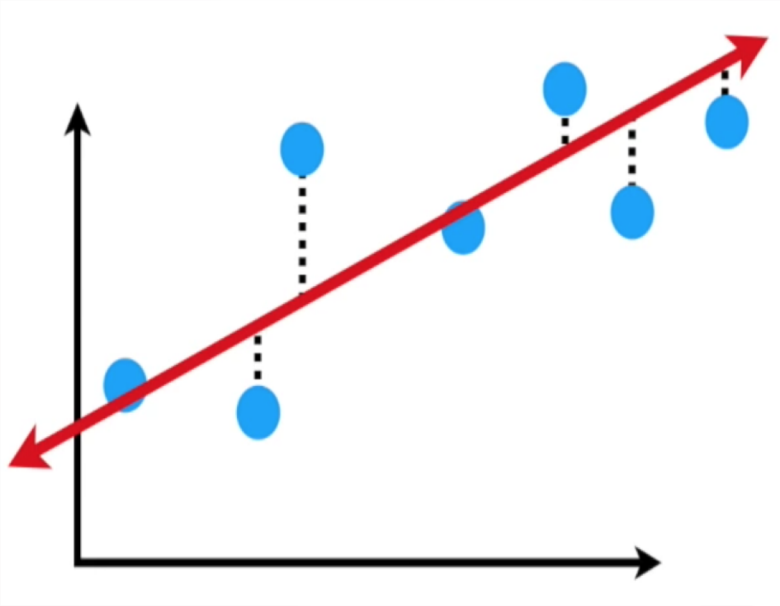
- Irreducible Error / Noise
- Bias
- Variance

Prediction Errors: Irreducible Error / Noise

- Cannot be reduced regardless of what algorithm is used.
- It is the error likely caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

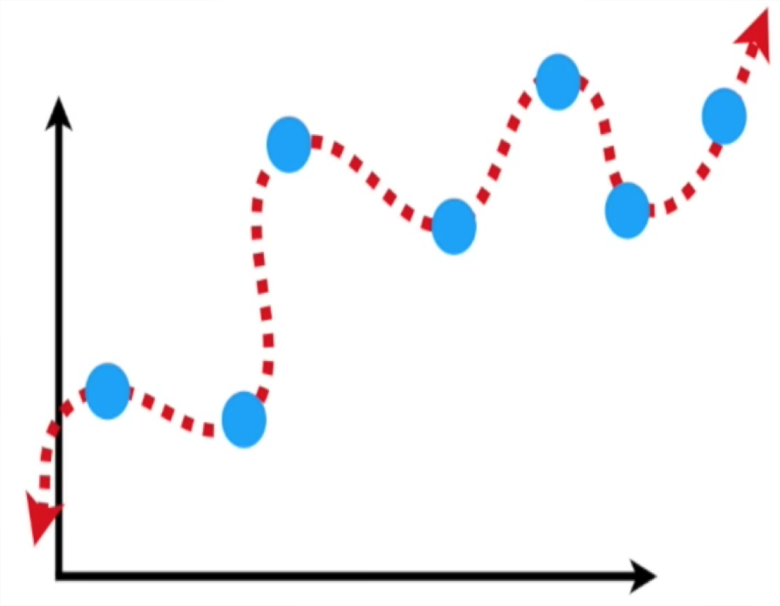
Prediction Errors: Bias

- Simplifying assumptions made by a model to make the target function easier to learn.
- Parametric algorithms generally have a high bias making them fast to learn and easier to understand but generally less flexible with a lower predictive power on complex problems
- Low Bias: Makes less assumptions about the form of the target function.
- High Bias: Makes more assumptions about the form of the target function.
- Examples of Low-Bias ML algorithms:
    - Decision Trees,
    - k-Nearest Neighbors
- Examples of High Bias ML algorithms include:
    - Linear Regression
    - Logistic Regression.



Prediction Errors: Variance

- Amount that the estimate of the target function will change if different training data was used.
- ML algorithms with high variance are strongly influenced by the specifics of the training data. (“Memorizes the Training Data” vs Learning the underlying patterns)
- Low Variance: Makes small changes to the estimate of the target function with changes to the training dataset.
- High Variance: Makes large changes to the estimate of the target function with changes to the training dataset.
- Examples of Low Variance ML algorithms:
    - Linear Regression
    - Logistic Regression
- Examples of high-variance machine learning algorithms include:
    - Decision Trees
    - k-Nearest Neighbors



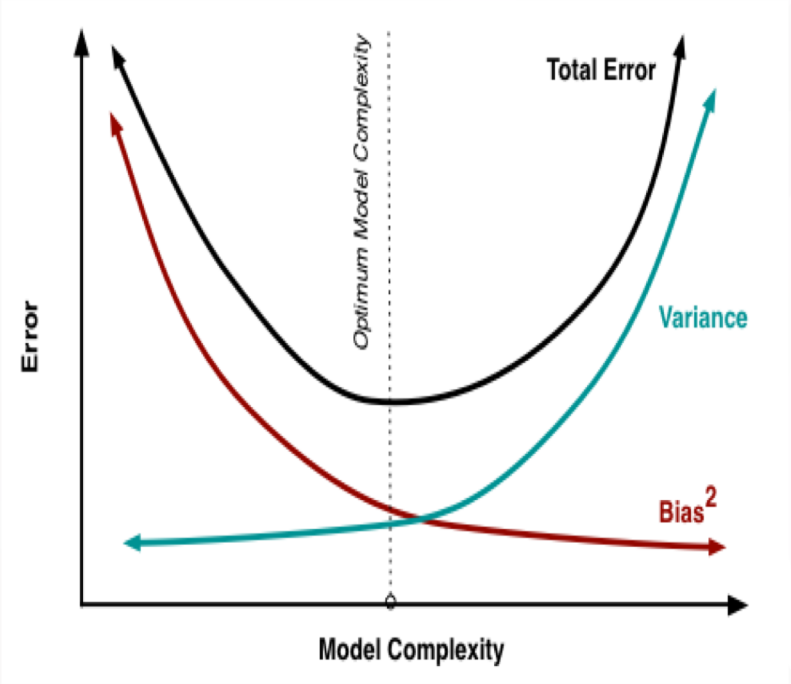
Prediction Errors: Bias-Variance Tradeoff

- The goal of any supervised machine learning algorithm is to achieve low bias and low variance

- General Trends:
  - Parametric (Linear) ML algorithms - often have a high bias but a low variance.
  - Non-Parametric (Non-Linear) ML algorithms - often have a low bias but a high variance.

- The trade-off:
  - Increasing the bias will decrease the variance.
  - Increasing the variance will decrease the bias.

- Optimizing ML algorithms parameters is used to balance out bias and variance.



Model Optimization

Model Optimization

Overview

Optimization is defined as the process of finding the conditions that give the minimum or maximum value of a function, where the function represents the effort required or the desired benefit.

- Hyperparameter Optimization
  - Grid Search
  - Random Search

Model Optimization: Hyper-Parameter Optimization

- HYPERPARAMETERS are the higher level properties of the model that could not be directly learned by training the model.

- These can be decided by setting different values, training different models, and choosing the values that test better.

- Examples:
  - Number of leaves or depth of a tree
  - Number of neighbors (k) in K nearest neighbors
  - Number of clusters in a k-means clustering

Model Optimization: Hyper-Parameter Optimization

- Hyperparameter Optimization or Tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm.

- The “best’ model is usually arrived at by tuning the parameters of an estimator / model.

- Common Algorithms:
  - Grid Search
  - Random Search

- A search consists of:
  - an estimator (regressor or classifier)
  - a parameter space;
  - a method for searching or
  - sampling candidates;
  - a cross-validation scheme
  - a score function.

Model Optimization: Hyper-Parameter Optimization

Grid Search	Random Search

• Exhaustively searches through a manually specified subset of the hyperparameter space of a learning algorithm.
• Also called Cartesian Search or Parameter Sweep
• Typically measure its performance by cross validation.
	• The strategy of Random Search is to sample solutions from across the entire search space using a uniform probability distribution.
	  ○ Each future sample is independent of the samples that come before it.
	• Some benefits over Grid Search:
	  ○ A budget can be chosen independent of the number of parameters and possible values.
	  ○ Adding parameters that do not influence performance does not decrease efficiency

LAB: Hyper-Parameter Optimization

Ensemble Models

Ensemble Models

Overview

- The goal of ensemble learning is to combine predictions of several models/estimators to:
  - Improve generalizability
  - Increase Robustness
- Often leads to a better predictive performance than a single learner
- Well-suited when small differences in the training data produce very different classifiers (e. g. decision trees)
- Drawbacks: increases computation time, reduces interpretability

Ensemble Models

Overview

- Reduce Bias - a combination of multiple models may learn a more expressive model than a single model
- Reduce Variance - less dependent on noise of a single training set

Ensemble Models

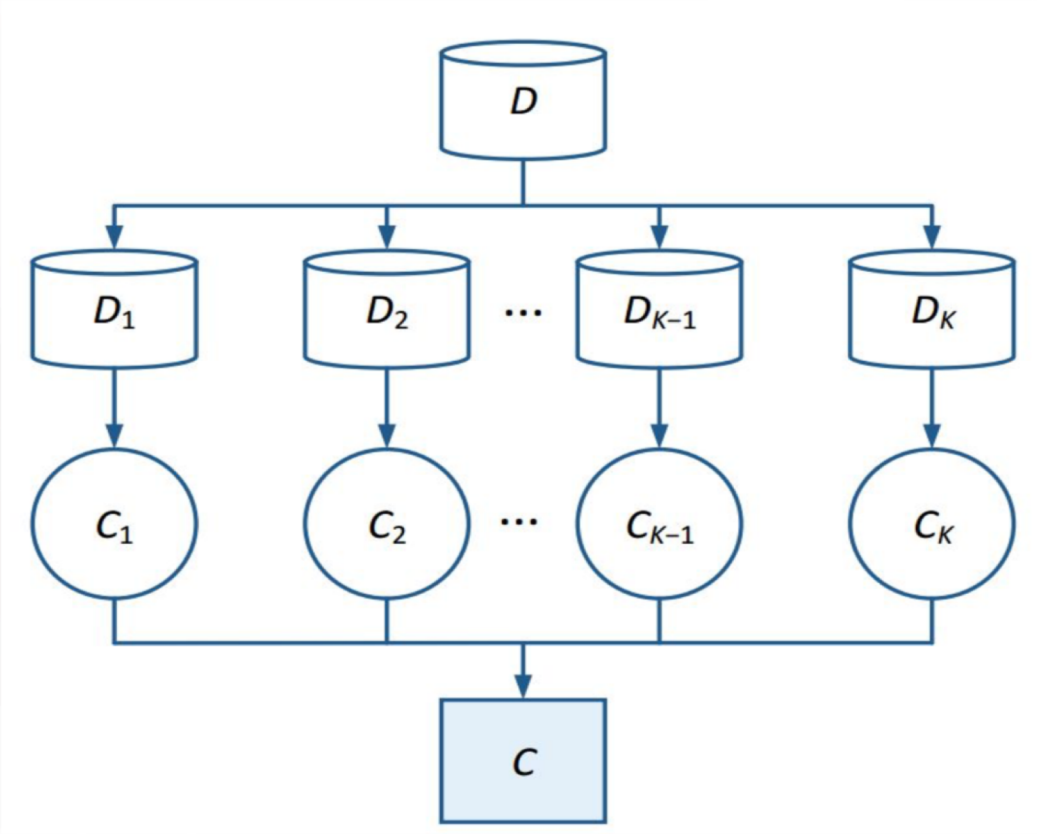
Algorithms

- Bagging
- Boosting
- Voting

Ensemble Models: Bagging

- Bagging = Bootstrap Aggregation
- The Idea: Apply similar learners on small sample populations, then take either:
  - mean of predicted values (for regression)
  - majority vote
  - weighted vote based on probability estimates
- Reduces Variance, in general
- Improves performance for unstable classifiers which vary significantly with small changes in the data set (i.e. Decision Trees)

Ensemble Models: Process



Original Dataset

Step 1:
Create Separate Datasets

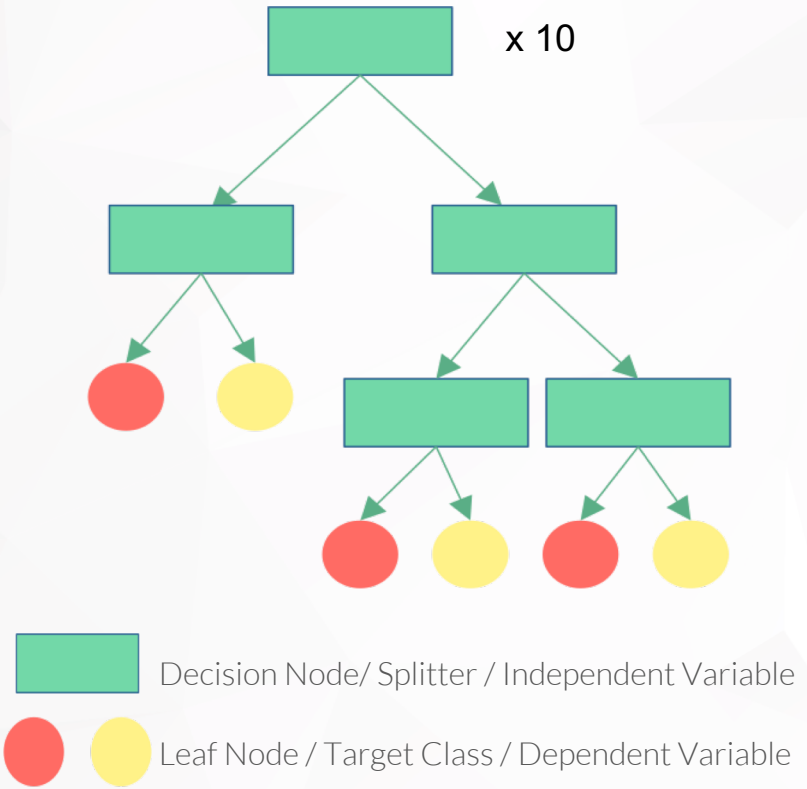
Step 2:
Train multiple Classifiers

Step 3:
Combine Classifiers

Bagging: Random Forest - Overview

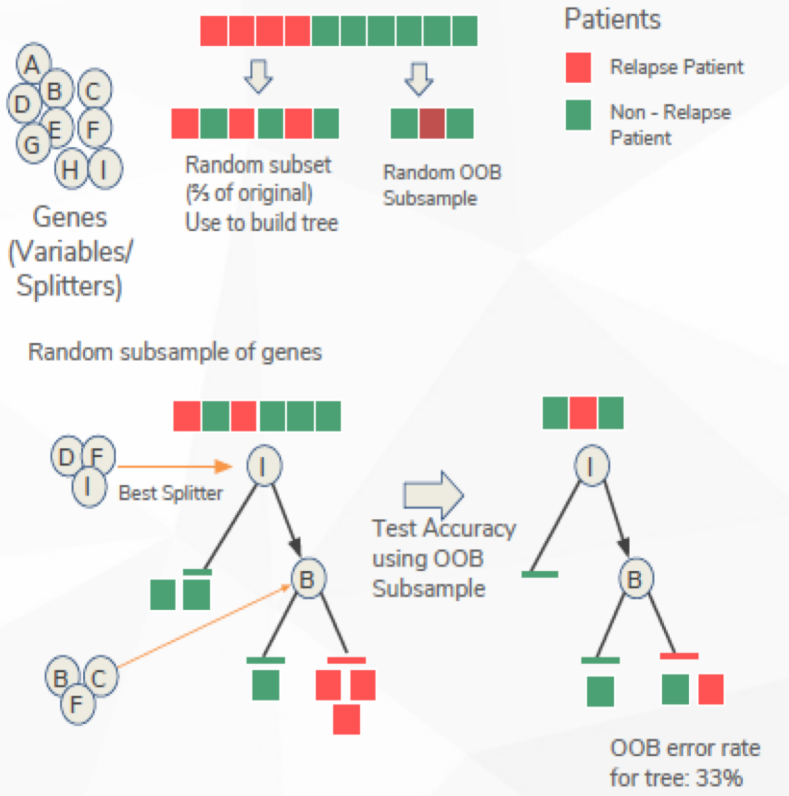
Ensemble learning method for classification and regression that operate by constructing a lot of decision trees at training time and outputting the class that is the mode/majority of the classes output by individual trees

- Use when you can't think of any other algorithm
- Considered to be a panacea of all data science problems.
- A way of averaging multiple deep decision trees, trained on different parts of the same training set
- Goal: overcome overfitting problem of individual decision tree

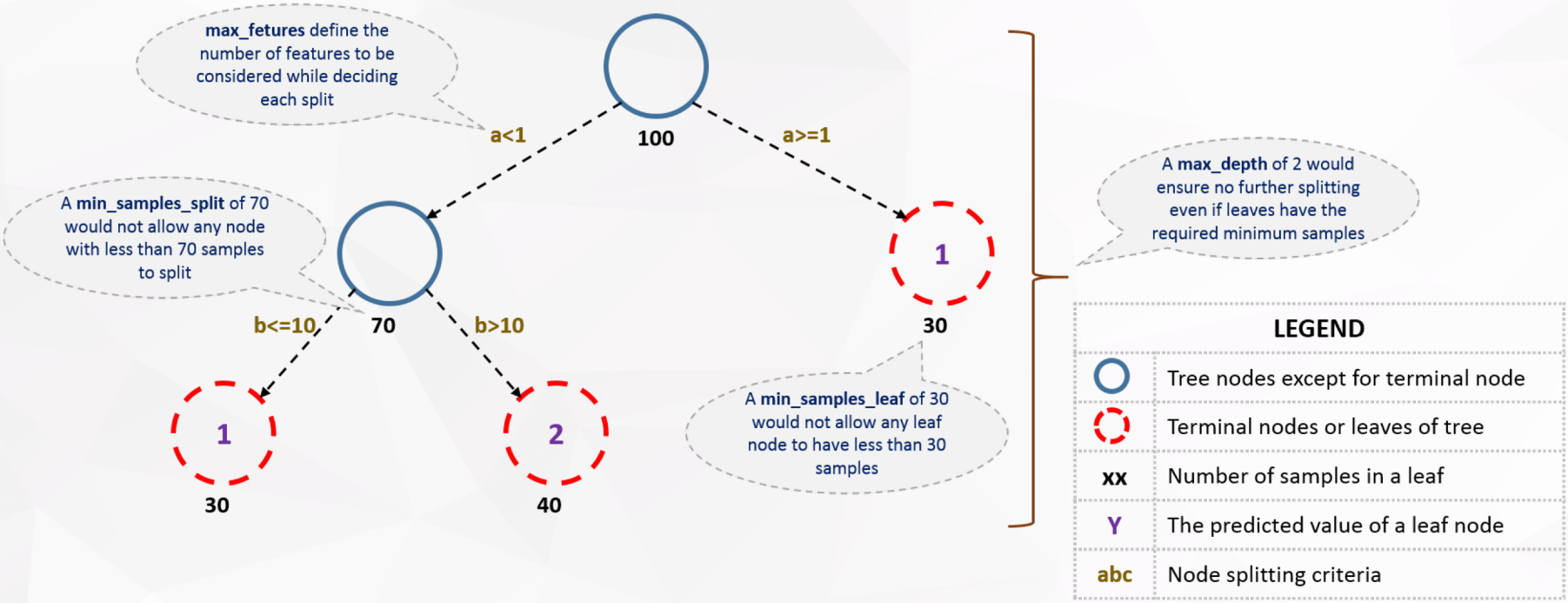


Bagging: Random Forest - Process

- Random record selection (with resampling)
- Random variable selection
- For each tree, using the leftover data, calculate the misclassification rate or OOB error rate. Aggregate error from all trees to determine the overall OOB error rate
  - Out of Bag (OOB) - records not selected for building trees (leftover data)
- The forest chooses the classification having the most votes over all the trees in the forest



Bagging: Random Forest



Bagging: Random Forest - Parameters

- max_features
  - maximum number of features Random Forest is allowed to try in individual tree
  - sqrt, 0.2, Auto/None
  - > max_features, slower but reduces the diversity of each tree

- n_estimators
  - number of trees built before voting.
  - > # trees better performance but slower

- min_sample_leaf
  - smaller leaf makes the model more prone to capturing noise in train data
  - Used to control overfitting as higher depth will allow model to learn relations very specific to a particular sample.

- max_depth
  - The maximum depth of a tree
  - Limiting depth may help resolve overfitting

Bagging: Random Forest - Pros and Cons

+	-
• Can be used for classification and regression • Can handle large data set with higher dimensionality • Effective method for estimating missing data • It has methods for balancing errors in data sets where classes are imbalanced. • Random Forest involves sampling of the input data with replacement called as bootstrap sampling. Here one third of the data is not used for training and can be used for testing	• Does a good job at classification but NOT as good for regression problem • Not good at generalizing to cases with completely new data • Random Forest can feel like a black box approach for statistical modelers • If a variable is a categorical variable with multiple levels, random forests are biased towards the variable having multiple levels

LAB:
Random Forest

Analytiks

Ensemble Models: Boosting

The idea:

- Later classifiers focus on examples that were misclassified by earlier classifiers;
- Combine several weak models to produce a powerful ensemble
- Iterative technique which adjusts the weights of an observation based on the last classification.
- Reduces Bias in general
- Sample Algorithms:
  - AdaBoost
  - Gradient Boosting

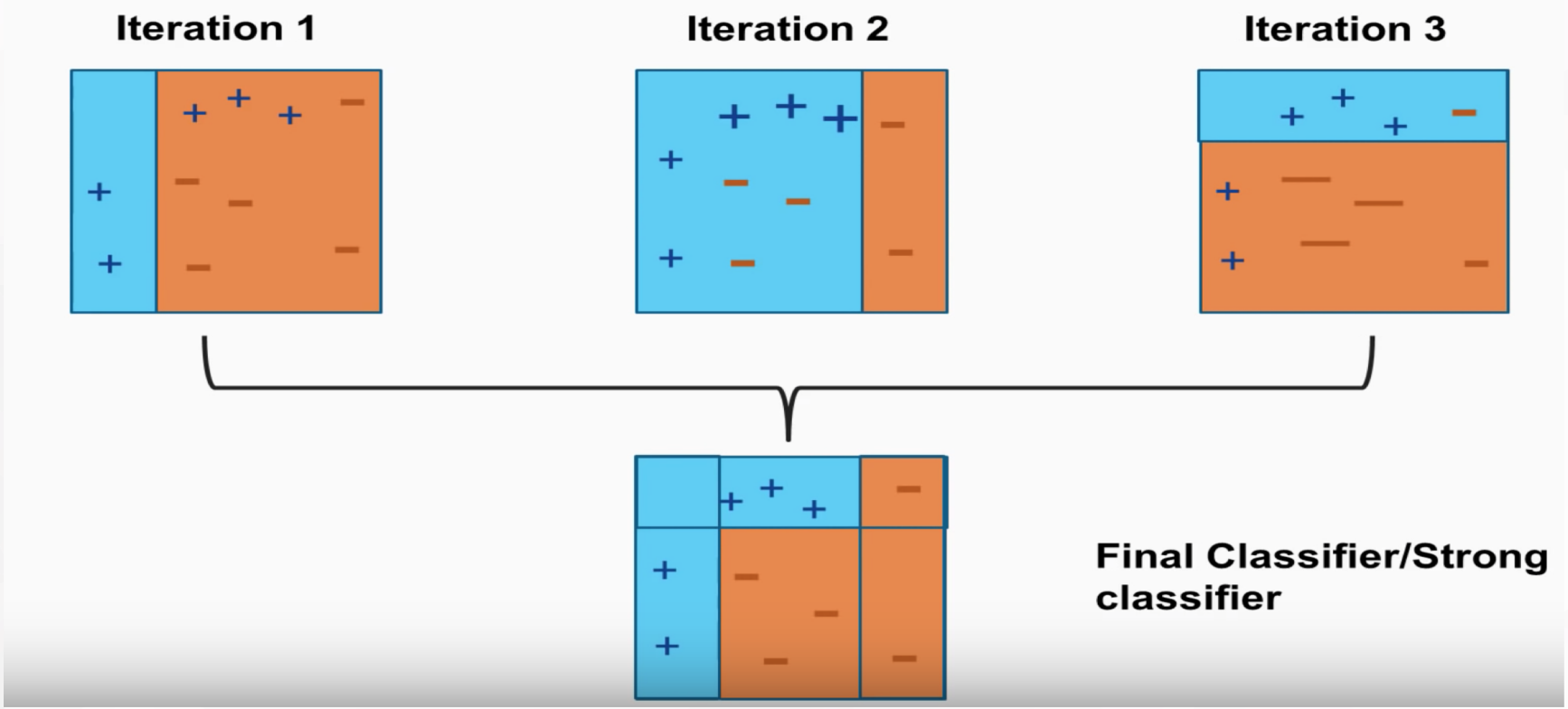
Boosting: ADABOOST - Overview

- ADABOOST = Adaptive Boosting
- Arguably the best known of all ensemble-based algorithms
- Extends boosting to multi-class and regression problems

Boosting: ADABOOST - Process

1. Weigh all training samples equally
1. Train model on train set
1. Compute error of model on training set
1. Increase weights on train cases model gets wrong
1. Train new model on re-weighted train set
1. Re-compute errors on weighted train set
1. Increase weights again on cases model gets wrong
1. Repeat until “tired” (100+ iterations)
1. Final Model: weighted prediction of each model

Boosting: ADABOOST - Process



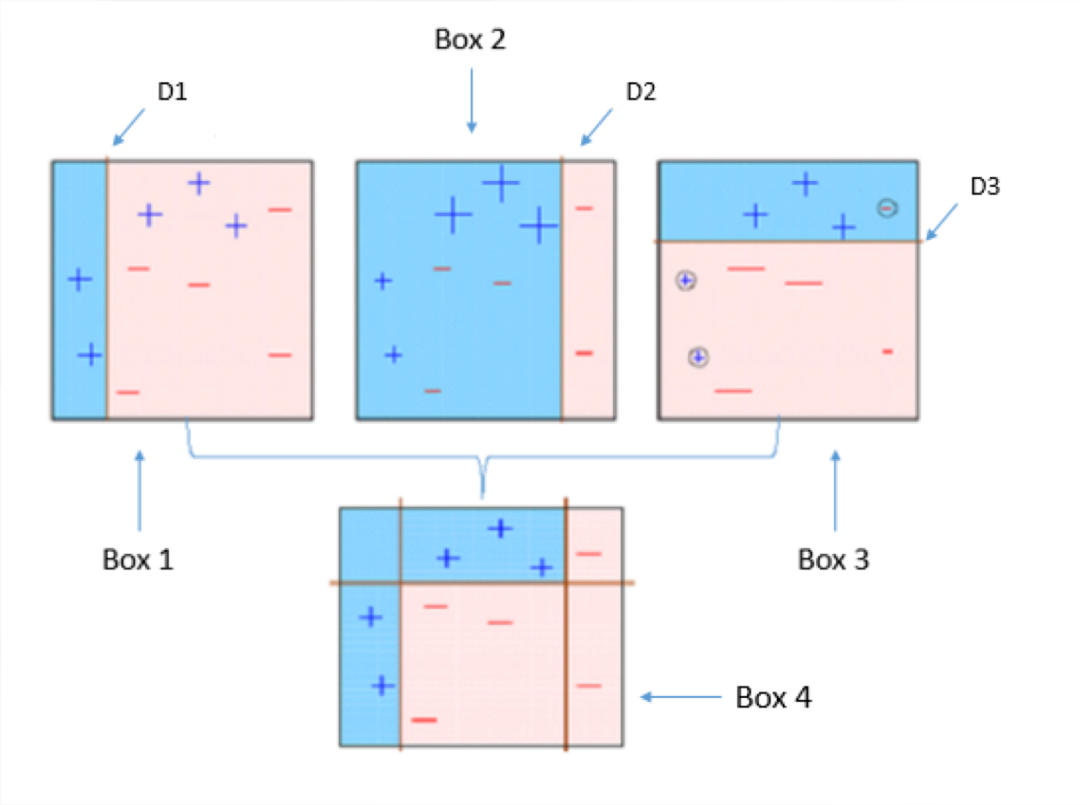
Boosting: Gradient Boosting

- Can handle Classification and Regression
- Combines the concept of Gradient Descent and Boosting
- Intuition is same as AdaBoost but weight of error is replaced by a Loss function that is minimized
- Loss function value increases with how bad the classifier/regressor is
- Involves three elements:
  - A loss function to be optimized.
  - A weak learner to make predictions.
  - An additive model to add weak learners to minimize the loss function.

Boosting: Gradient Boosting - Process

1. Assume mean is the prediction of all variables.
1. Calculate errors of each observation from the mean (latest prediction).
1. Find the variable that can split the errors perfectly and find the value for the split.
   This is assumed to be the latest prediction.
1. Calculate errors of each observation from the mean of both the sides of split (latest prediction).
1. Repeat the step 3 and 4 till the objective function maximizes/minimizes.
1. Take a weighted mean of all the classifiers to come up with the final model.

Ensemble Models: Gradient Boosting



Boosting: Gradient Boosting - Parameters

- min_samples_split
  - Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
  - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
  - Too high values can lead to under-fitting hence, it should be tuned using CV.
- min_samples_leaf
  - Defines the minimum samples (or observations) required in a terminal node or leaf.
  - Used to control over-fitting similar to min_samples_split.
  - Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
- max_depth
  - The maximum depth of a tree.
  - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- max_leaf_nodes
  - The maximum number of terminal nodes or leaves in a tree
  - If this is defined, GBM will ignore max_depth.
- max_features
  - The number of features to consider while searching for a best split.
  - As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
  - Higher values can lead to over-fitting but depends on case to case.

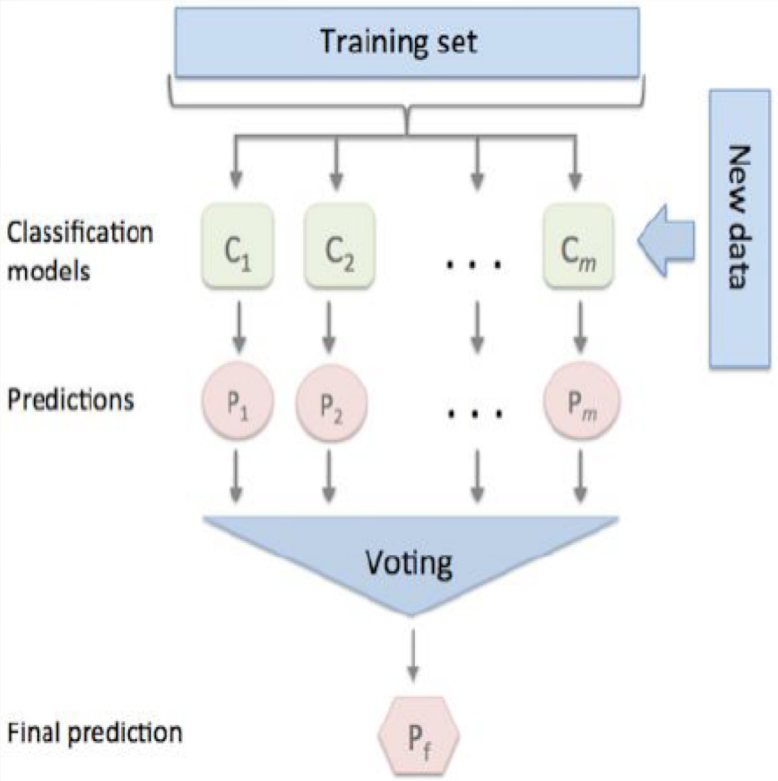
LAB: GBM

Ensemble Models: Voting

- Build multiple models (usually of different types) then use simple statistics to combine predictions

- Hard Voting / Majority Class
  - Predicted class label is the label that most classifiers predict (majority / mode)

- Soft Voting / Weighted
  - Predicted class label is the argmax of the sum of the predicted probabilities
  - Only recommended if the classifiers are well-calibrated
  - Can use uniform weights or specified weights



LAB:
Voting