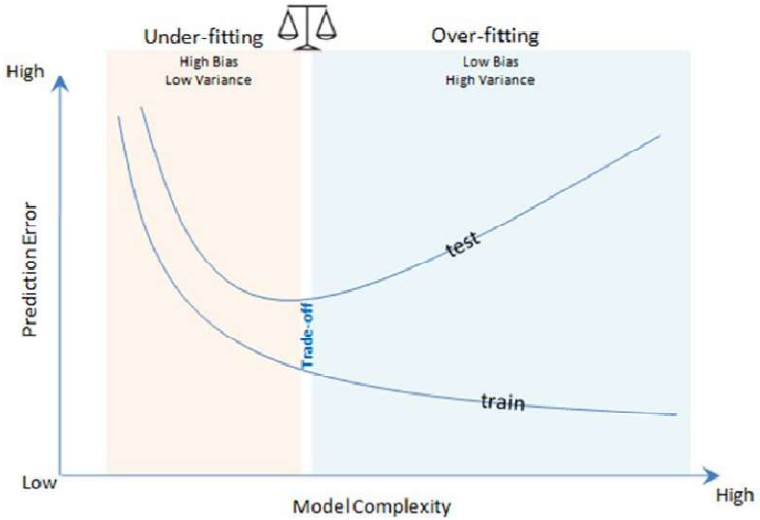
Bias and Variance

A fundamental problem with supervised learning is the bias–variance tradeoff. Ideally,a model should have two key characteristics:

1. It should be sensitive enough to accurately capture the keypatterns in the training dataset.
2. It should be generalized enough to work well on any unseendatasets.

Unfortunately, while trying to achieve the aforementioned first point, there is an sample chance of overfitting to noisy or unrepresentative training data points, leading toa failure of generalizing the model.

On the other hand, trying to generalize a model mayresult in failing to capture important regularities.



Bias-variance trade-off

Bias

If model accuracy is low on the training dataset as well as the test dataset, the modelis said to be underfitting or has a high bias. This means the model is not fitting thetraining dataset points well in regression or the decision boundary is not separatingthe classes well in classification.

Two key reasons for bias are:

1) not including the rightfeatures, and

2) not picking the correct order of polynomial degree for model fitting.

To solve the underfitting issue or to reduced bias, try including more meaningfulfeatures and try to increase the model complexity by trying higher order polynomialfittings.

Variance

If a model is giving high accuracy on the training dataset, but on the test dataset theaccuracy drops drastically, then the model is said to be overfitting or has high variance.

The key reason for overfitting is using a higher order polynomial degree (may not berequired), which will fit the decision boundary tool well to all data points including thenoise of the train dataset instead of the underlying relationship. This will lead to a highaccuracy (actual vs. predicted) in the train dataset and when applied to the test dataset,the prediction error will be high.

Guidelines to solve the overfitting issue:

- ✓ Try to reduce the number of features, that is, keep only the meaningful features or try regularization methods that will keep all the features but reduce the magnitude of the feature parameter.

- ✓ Dimension reduction can eliminate noisy features, in turn reducing the model variance.

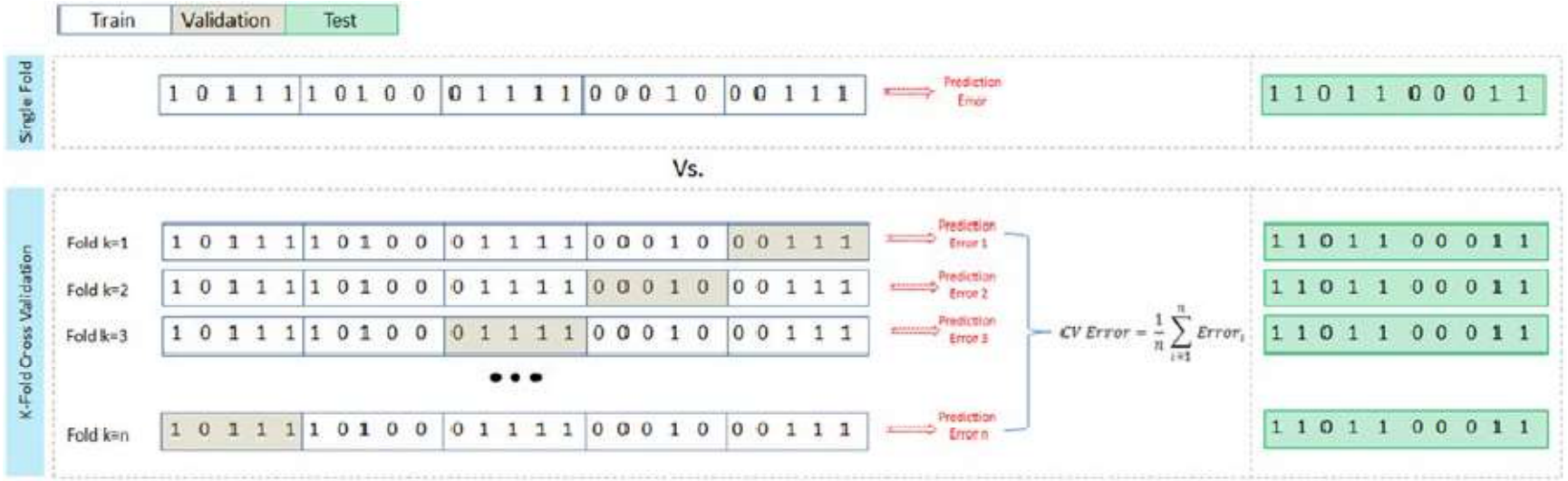
- ✓ Bringing more data points to make training dataset large will also reduce variance.

- ✓ Choosing the right model parameters can help to reduce the bias and variance, for example:
  - Using right regularization parameters can decrease variance inregression-based models.
  - For a decision tree, reducing the depth of the decision tree willreduce the variance.

K-Fold Cross Validation

K-fold cross-validation splits the training dataset into k folds without replacement—anygiven data point will only be part of one of the subsets, where k-1 folds are used for themodel training and one fold is used for testing. The procedure is repeated k times so thatwe obtain k models and performance estimates.

We then calculate the average performance of the models based on the individualfolds, to obtain a performance estimate that is less sensitive to the subpartitioning of thetraining data compared with the holdout or single fold method.



K-fold cross-validation

How is k-fold cross validation performed?

The general strategy is quite straight forward and the following steps can be used:

1. First, shuffle the dataset and split into k number of subsamples. (It is important to try to make the subsamples equal in size and ensure k is less than or equal to the number of elements in the dataset).
2. In the first iteration, the first subset is used as the test data while all the other subsets are considered as the training data.
3. Train the model with the training data and evaluate it using the test subset. Keep the evaluation score or error rate, and get rid of the model.
4. Now, in the next iteration, select a different subset as the test data set, and make everything else (including the test set we used in the previous iteration) part of the training data.
5. Re-train the model with the training data and test it using the new test data set, keep the evaluation score and discard the model.
6. Continue iterating the above k times. Each data subsamples will be used in each iteration until all data is considered. You will end up with a k number of evaluation scores.
7. The total error rate is the average of all these individual evaluation scores.

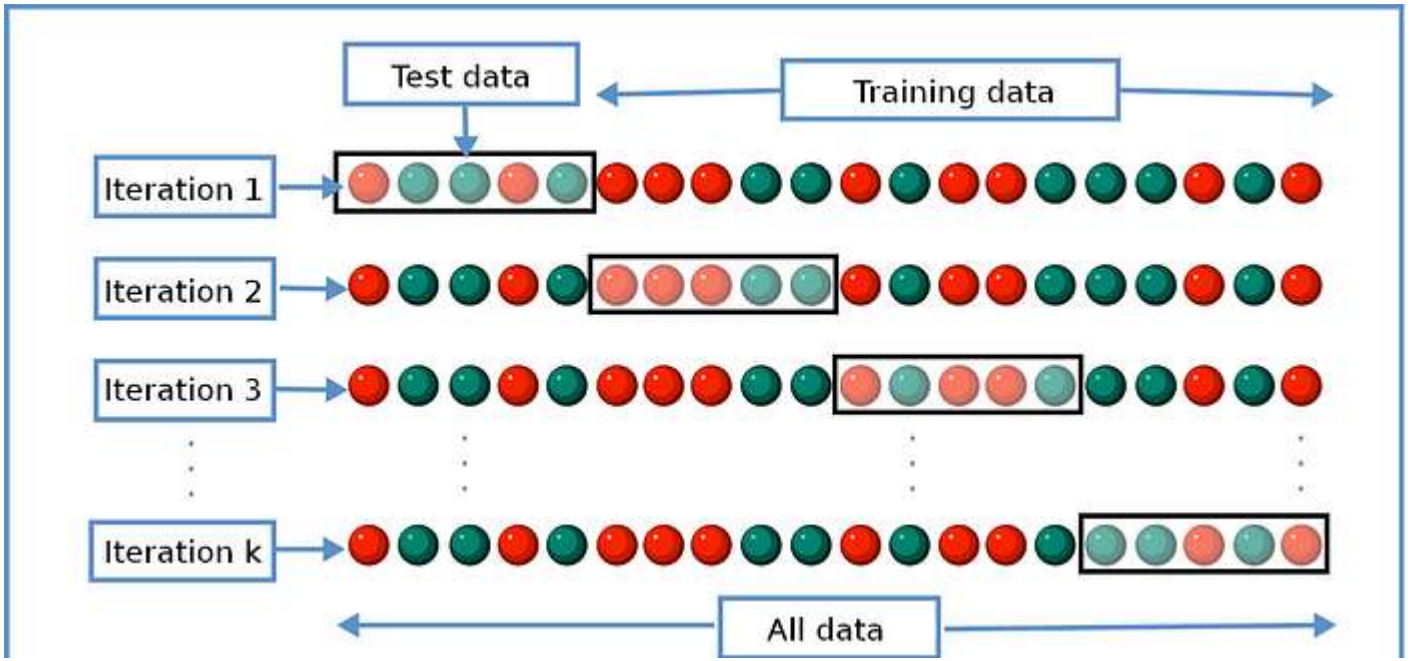


Diagram of k-fold cross validation By Gufosowa — Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=82298768

How to determine the best value for 'k' in K-Fold Cross Validation?

Chosing a good value for k is important. A poor value for k can result in a poor evaluation of the model's abilities. In other words, it can cause the measured ability of the model to be overestimated (high bias) or change widely depending on the training data used (high variance).

Generally, there are three ways to select k:

- Let k = 5, or k =10. Through experimentation, it has been found that selecting k to be 5 or 10 results in sufficiently good results.

- Let k = n, where n is the size of the dataset. This ensures each sample is used in the test data set.
- Another way is to chose k so that every split data sample is sufficiently large, ensuring they are statistically represented in the larger dataset.

Types of cross validation

Cross validation can be divided into two major categories:

- **Exhaustive**, where the method learn and test on every single possibility of dividing the dataset into training and testing subsets.
- **Non-exhaustive** cross validation methods where all ways of splitting the sample are not computed.

Exhaustive cross-validation

Leave-p-out cross validation is a method of exhaustive cross validation. Here, p number of observations (or elements in the sample dataset) are left out as the training dataset, everything else is considered as part of the training data. For more clarity, if you look at the above image, p is equal to 5, as shown by the 5 circles in the 'test data'.

Leave-one-out cross validation a special form of leave-p-out exhuastive cross validation method, where p = 1. This is also a specific case for k-fold cross validation, where k = N(number of elements in the sample dataset).

Non-exhaustive cross-validation

K-fold cross validation where k is not equal to N, Stratified cross validation and repeated random sub-sampling validation are non-exhaustive cross validation methods.

Stratified cross validation: partitions are selected such that each partition contains roughly the same amount of elements for each class label. For example, in binary classification, every split has elements of which roughly 50% belongs to class 0 and 50% that belongs to class 1.

Stratified k-Fold is a variation of the standard **k-Fold CV** technique which is designed to be effective in such cases of target imbalance.

Sometimes we may face a large imbalance of the target value in the dataset. For example, in a dataset concerning wristwatch prices, there might be a larger number of wristwatch having a high price. In the case of classification, in cats and dogs dataset there might be a large shift towards the dog class.

It works as follows. Stratified k-Fold splits the dataset on k folds such that each fold contains approximately the same percentage of samples of each target class as the complete set. In the case of regression, Stratified k-Fold makes sure that the mean target value is approximately equal in all the folds.

The algorithm of Stratified k-Fold technique:

1. Pick a number of folds – k
2. Split the dataset into k folds. Each fold must contain approximately the same percentage of samples of each target class as the complete set
3. Choose k – 1 folds which will be the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration a new model must be trained
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

Here class proportions are preserved in each fold, leading to better bias and variance estimates.

Repeated random sub-sampling validation (Monte Carlo cross validation): Data is split into multiple random subsets and the model is trained and evaluated for each split. The results are averaged over the splits. Unlike the k-fold cross validation, proportions of the training and test

set size are not dependent on the size of the data set, which is an advantage. However, a disadvantage is that some data elements will never be selected as a part of the test set, while some may be selected multiple times. When the amount of random splits are increased and approach infinity, the results tend to be similar to that of leave-p-out cross validation.

Ensemble Methods

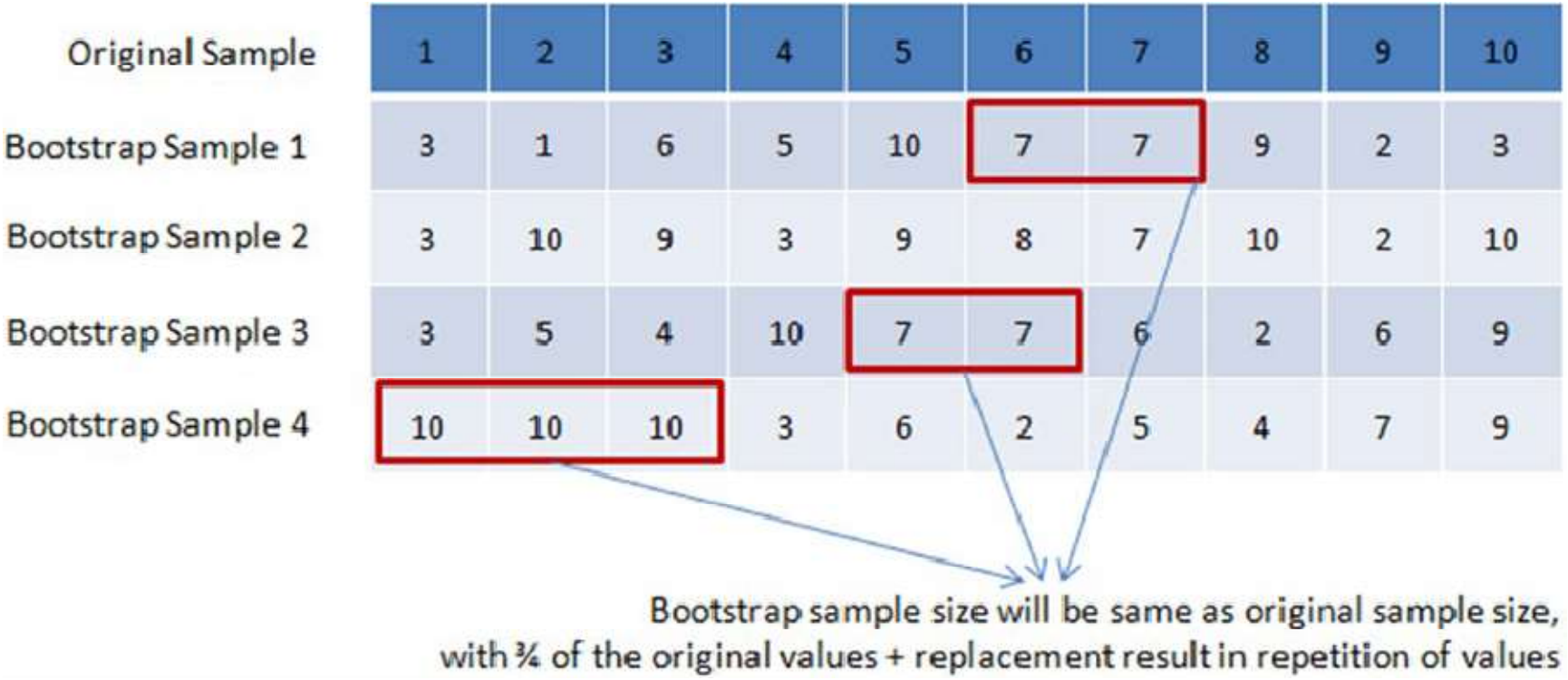
Ensemble methods enable combining multiple model scores into a single score to createa robust generalized model.

At a high level, there are two types of ensemble methods:

1. Combine multiple models of similar type.
    - Bagging (bootstrap aggregation)
    - Boosting
2. Combine multiple models of various types.
    - Vote classification
    - Blending or stacking

Bagging

Bootstrap aggregation (also known as bagging) was proposed by Leo Breiman in 1994;it is a model aggregation technique to reduce model variance. The training data is splitinto multiple samples with a replacement called bootstrap samples. Bootstrap samplesize will be the same as the original sample size, with 3/4 of the original values andreplacement resulting in repetition of values.

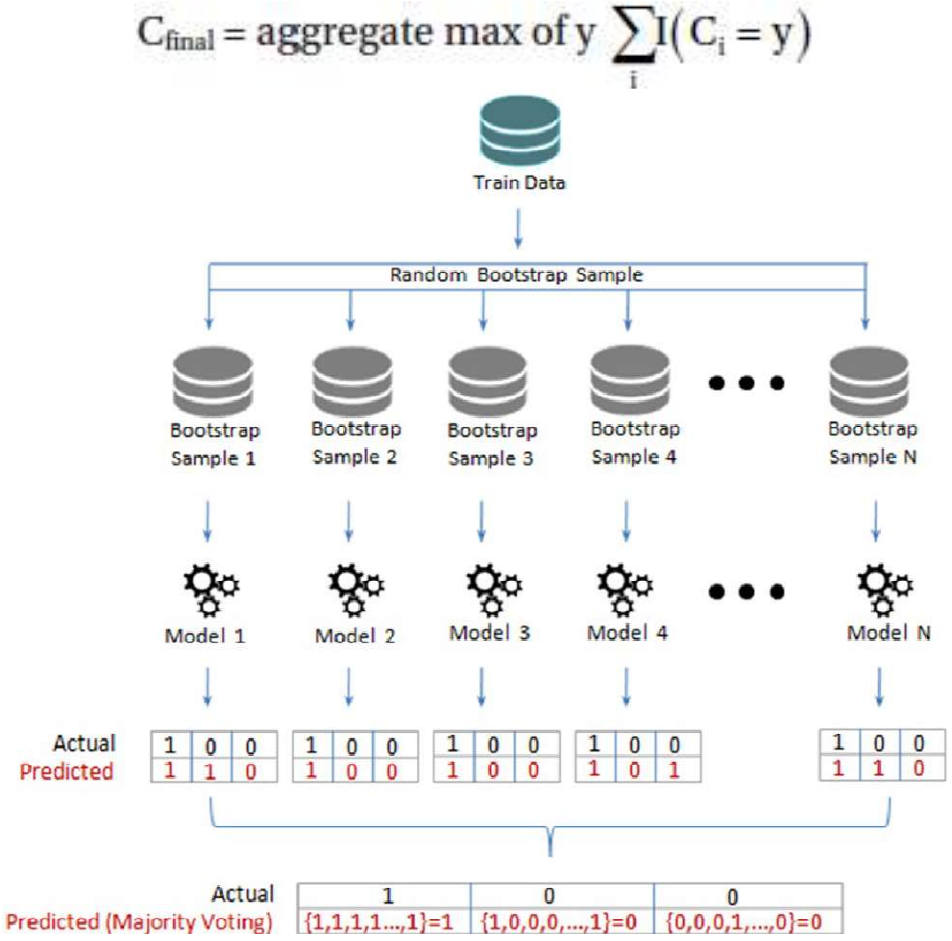


Bootstrapping

Independent models on each of the bootstrap samples are built, and the averageof the predictions for regression or majority vote for classification is used to create thefinal model.

The following figure shows the bagging process flow.

If N is the number of bootstrap samplescreated out of the original training set, for i = 1 to N, train a base ML model Ci.



Bagging

Bagging—Essential Tuning Parameters

Let's look at the key tuning parameters to get better model results:

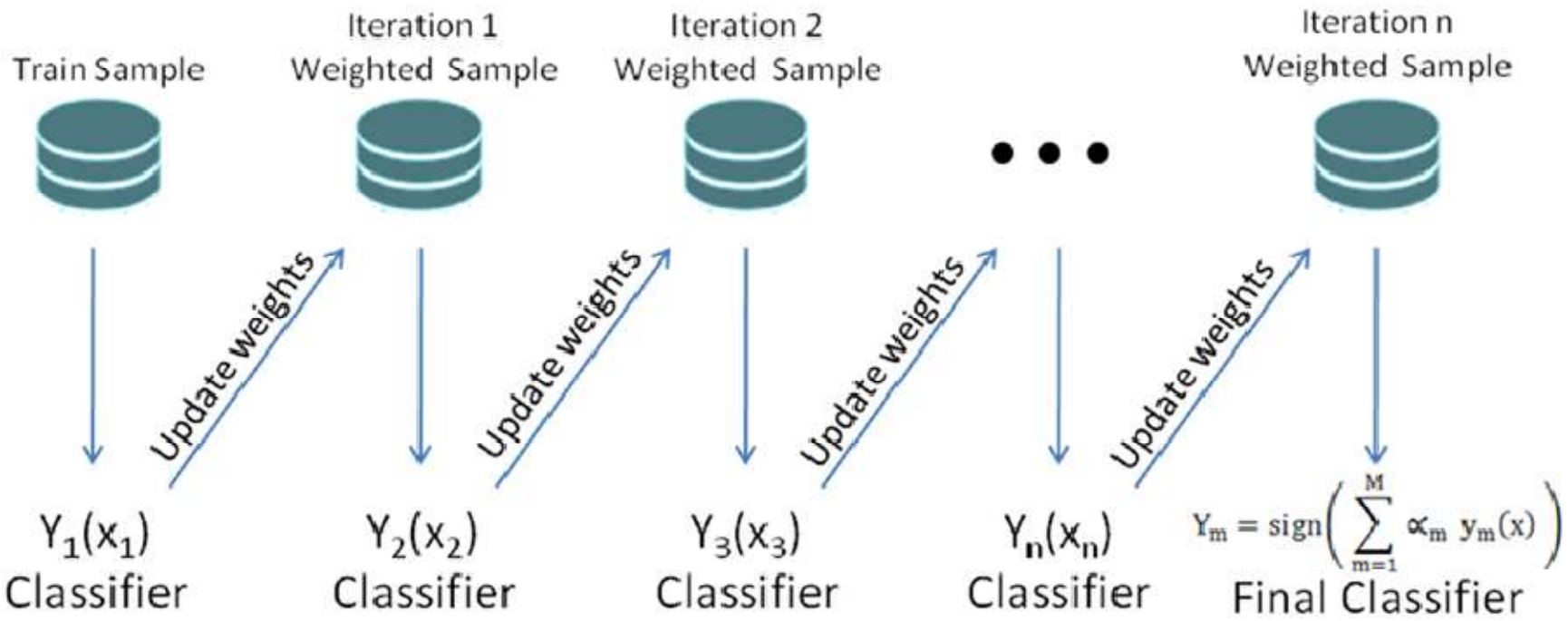
- *n_estimators:* This is the number of trees—the larger the better. Notethat beyond a certain point, the results will not improve significantly.

- *max_features:* This is the random subset of features to be usedfor splitting a node—the lower the better to reduce variance (butincreases bias). Ideally, for a regression problem it should be equalto n_features (total number of features) and for classification, square root of n_features.

- *n_jobs:* Number of cores to be used for parallel construction of tress.
If set to -1, all available cores in the system are used, or you canspecify the number.

Boosting

Freud and Schapire in 1995 introduced the concept of boosting with the well-known**AdaBoost** algorithm (adaptive boosting). The core concept of boosting is that ratherthan an independent individual hypothesis, combining hypotheses in a sequentialorder increases the accuracy. Essentially, boosting algorithms convert the weaklearners into strong learners. Boosting algorithms are well designed to address biasproblems (shown in the following figure).



AdaBoosting

At a high level the AdaBoosting process can be divided into three steps:

- Assign uniform weights for all data points $W_0(x) = 1 / N$, where N isthe total number of training data points.

- At each iteration, fit a classifier $y_m(x_n)$ to the training data and updateweights to minimize the weighted error function.
  - The weight is calculated as: $W_n^{(m+1)} = W_n^{(m)} \exp\{ \alpha_m y_m (x_n) \neq t_n \}$.
  - The hypothesis weight or the loss function isgiven by:

    $$\alpha_m = \tfrac{1}{2} \log \{ 1 - \varepsilon_m / \varepsilon_m \},$$

    and, the term rate is given by:

    $$\epsilon_m = \frac{\sum_{n=1}^{N} W_n^{(m)} I\left(y_m(x_n) \neq t_n\right)}{\sum_{n=1}^{N} W_n^{(m)}},$$

    where, $(y_m(x_n) \neq t_n)$ has values 0 / 1, i.e. 0 if x_n correctly classified, else 1.

- The final model is given by:

$$Y_m = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x) \right)$$

Gradient Boosting

Due to the stagewiseadditivity, the loss function can be represented in a form suitablefor optimization. This gave birth to a class of generalized boosting algorithms known asgeneralized boosting machine (GBM). Gradient boosting is an example implementationof GBM and it can work with different loss functions such as regression, classification,risk modeling, etc. As the name suggests, it is a boosting algorithm that identifiesshortcomings of a weak learner by gradients (AdaBoost uses high-weight data points),hence the name gradient boosting.

- Iteratively fit a classifier $y_m(x_n)$ to the training data. The initial modelwill be with a constant value $y_0(x)$ arg min $\delta\sum_{i=1}^{N}L(y_m,\delta)$.
- Calculate the loss (i.e., the predicted value vs. actual value) for eachmodel fit iteration $g_m(x)$ or compute the negative gradient and useit to fit a new base learner function $h_m(x)$, and find the best gradientdescent step-size:

$$\delta_m = \arg\min_{\delta}\sum_{i=1}^{n} L\left(y_m, y_{m-1}(x)+\delta h_m(x)\right).$$

- Update the function estimate: $y_m(x) = y_{m-1}(x) + \delta\, h_m(x)$, and output, $y_m(x)$.

Gradient boosting corrects the erroneous boosting iteration's negative impact in subsequent iterations.

Boosting—Essential Tuning Parameters

Model complexity and overfitting can be controlled by using correct values for twocategories of parameters:

1. **Tree structure**

n_estimators: This is the number of weak learners to be built.

max_depth: This is the maximum depth of the individualestimators. The best value depends on the interaction of the inputvariables.

min_samples_leaf: This will be helpful to ensure a sufficientnumber of samples results in leaf.

subsample: This is the fraction of the sample to be used for fittingindividual models (default=1). Typically .8 (80%) is used tointroduce a random selection of samples, which in turn increasesthe robustness against overfitting.

2. **Regularization parameter**

learning_rate: This controls the magnitude of change inestimators. The lower learning rate is better, which requires highern_estimators (that is the trade-off).

Xgboost (eXtreme Gradient Boosting)

In March 2014, Tianqui Chen built xgboost in C++ as part of the distributed (deep)ML community, and it has an interface for Python. It is an extended, more regularizedversion of a gradient boosting algorithm. This is one of the most well-performing, largescale,scalable ML algorithms, which has been playing a major role in winning solutionsin Kaggle (forum for predictive modeling and analytics competition) data sciencecompetition.

XGBoost objective functionobj(θ):

$$\text{obj}(\theta) = \sum_{i}^{n} l(y_i - \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Regularization term is given by:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} W_j^2$$

Controls the overall number of leaves created + Scores of the overall number of leaves created

Gradient descent technique is used for optimizing the objective function, andmore mathematics about the algorithms can be found at the site http://xgboost.readthedocs.io/en/latest/.

Some of the key advantages of the xgboost algorithm are:

- It implements parallel processing.
- It has a built-in standard to handle missing values, which means theuser can specify a particular value different than other observations(such as -1 or -999) and pass it as a parameter.
- It will split the tree up to maximum depth, unlike gradient boostingwhere it stops splitting the node on encountering a negative loss inthe split.

XGboost has a bundle of parameters, and at a high level we can group them intothree categories. Let's look at the most important within these categories.

1. General parameters
    a. *nthread*: Number of parallel threads; if not given a value allcores will be used.
    b. *Booster*: This is the type of model to be run, with gbtree(tree-based model) being the default. "gblinear" to be used forlinear models.

2. Boosting parameters
    a. *eta*: This is the learning rate or step size shrinkage to preventoverfitting; default is 0.3 and it can range between 0 and 1.

b. *max_depth*: Maximum depth of tree, with the default being 6

c. *min_child_weight*: The minimum sum of weights of allobservations required in a child. Start with the 1/square rootof event rate.

d. *colsample_bytree*: A fraction of columns to be randomlysampled for each tree, with a default value of 1.

e. *Subsample*: A fraction of observations to be randomlysampled for each tree, with a default value of 1. Lowering thisvalue makes the algorithm conservative to avoid overfitting.

f. *lambda*: L2 regularization term on weights, with a defaultvalue of 1

g. *alpha*: L1 regularization term on weight

3. Task parameters

a. *Objective*: This defines the loss function to be minimized, withdefault value “reg: linear.” For binary classification it should be“binary: logistic” and for multiclass “multi:softprob” to get theprobability value and “multi: softmax” to get predicted class. Formulticlass, num_class (number of unique classes) is to be specified.

b. *eval_metric*: Metric to be used for validating modelperformance

sklearn has a wrapper for xgboost (XGBClassifier).

Differences Between Bagging and Boosting

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with replacement and random sampling methods from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.

S.NO	Bagging	Boosting
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost and Xgboost uses Boosting techniques

Bibliography:

1. https://www.bmc.com/blogs/bias-variance-machine-learning/
2. https://www.simplilearn.com/tutorials/machine-learning-tutorial/bias-and-variance
3. https://www.analyticsvidhya.com/blog/2020/08/bias-and-variance-tradeoff-machine-learning/
4. https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
5. https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/
6. https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/
7. https://www.kaggle.com/code/prashant111/bagging-vs-boosting
8. https://www.datatrained.com/post/bagging-and-boosting/
9. https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205
10. https://analyticsindiamag.com/primer-ensemble-learning-bagging-boosting/
11. https://towardsdatascience.com/what-is-k-fold-cross-validation-5a7bb241d82f
12. https://www.javatpoint.com/cross-validation-in-machine-learning
13. https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/
14. https://towardsdatascience.com/what-is-k-fold-cross-validation-5a7bb241d82f
15. https://www.geeksforgeeks.org/cross-validation-machine-learning/
16. https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right
17. Swamynathan, M. (2019). Mastering machine learning with python in six steps: A practical implementation guide to predictive data analytics using python. Apress.
18. "Machine Learning Fundamentals: Bias and Variance" from https://www.youtube.com/watch?v=EuBBz3bI-aA