upGrad

#LifeKoKaroLift

Post-Graduate Diploma in ML/AI

upGrad

Course : Machine Learning

Lecture On : Boosting - Part 1

Instructor : Manish Kumar

upGrad

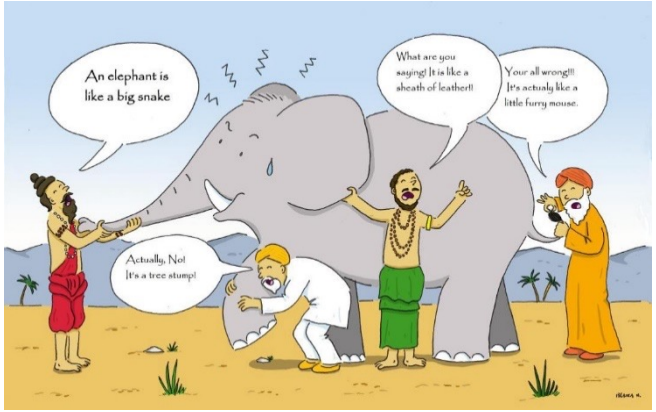
Modules Included

- Ensemble methods
- Bagging
- Boosting
- Ada-Boost

Ensemble methods

upGrad

Ensemble learning is a machine learning paradigm where multiple models are trained to solve the same problem and combined to get better results.



***sequential* ensemble** methods where 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 weight.

***parallel* ensemble methods** where 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.

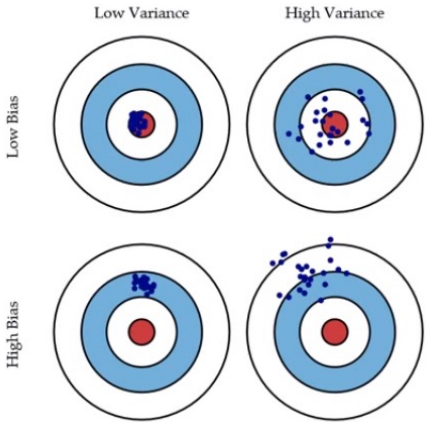
Bias and Variance in regression models

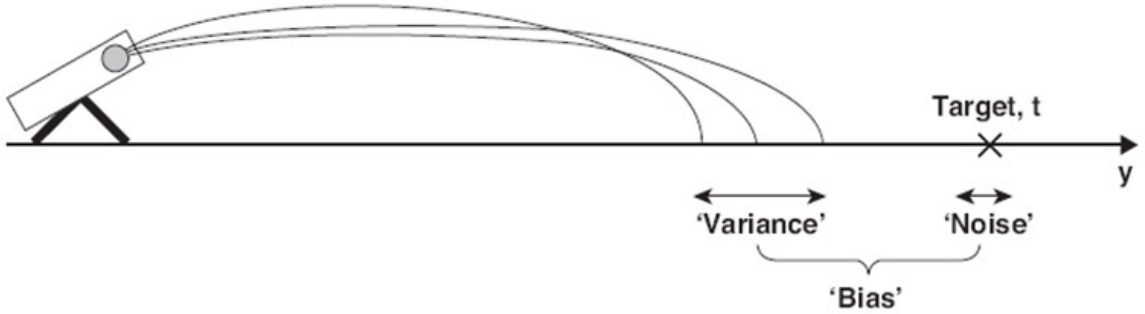
upGrad

The prediction error for any machine learning algorithm can be broken down into three parts:

- Bias Error
- Variance Error
- Irreducible Error

The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.



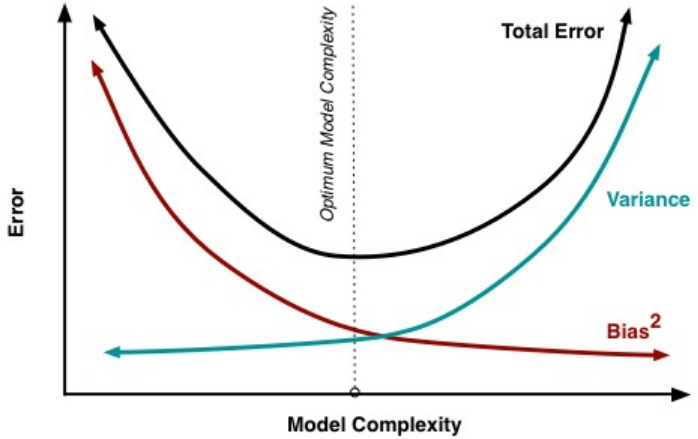


Bias-Variance Trade-Off

upGrad

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.



The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Random Forest

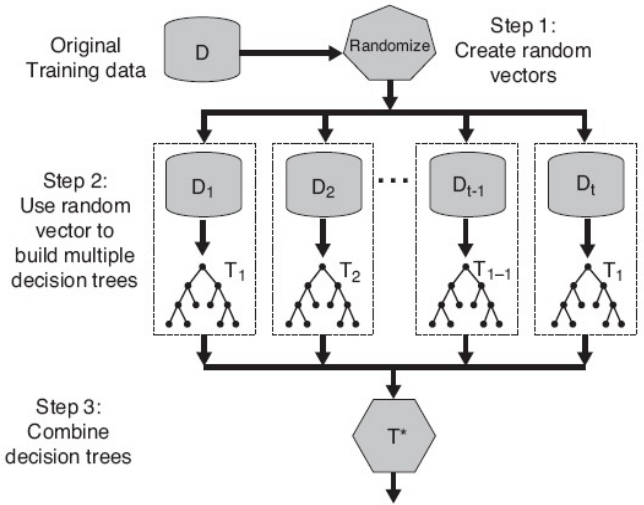
upGrad

Bagging (Bootstrap Aggregation)

- **Decision trees are very sensitive to the data they are trained on.**
- **small changes to the training set can result in significantly different tree structures.**

Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees.

Feature Randomness — In a normal decision tree, while splitting a node, we consider all the features and pick the one that produces the most separation. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.



Random Forest

upGrad

initial dataset

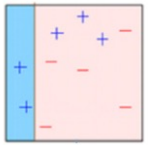
bootstrap samples + selected features

deep trees fitted on each bootstrap sample and considering only selected features

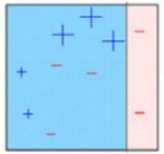
random forest (kind of average of the trees)

Boosting

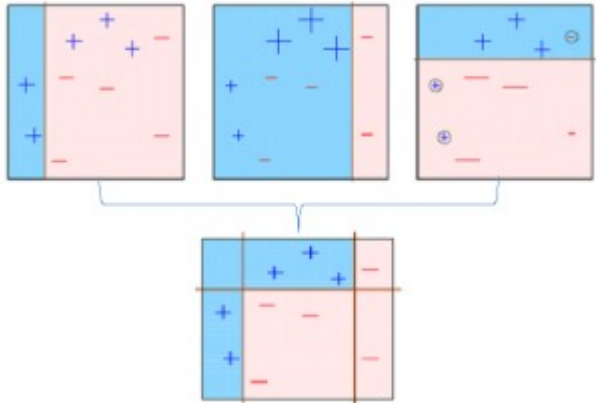
upGrad



Base model is created on a subset

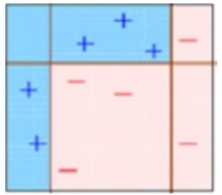


The observations which are incorrectly predicted, are given higher weights and an another model is created and predictions are made on the dataset



multiple models are created, each correcting the errors of the previous model.

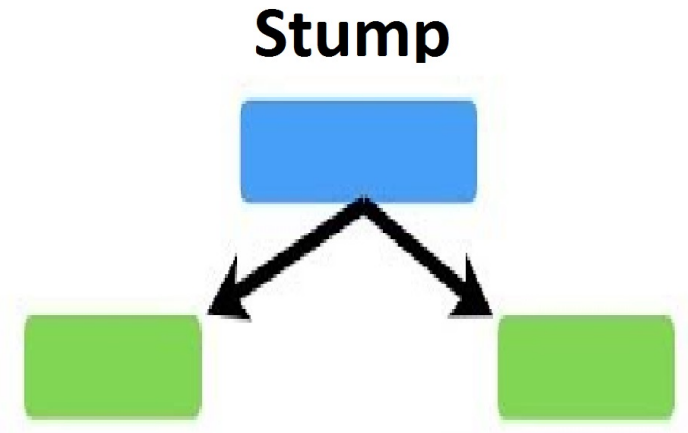
The final model (strong learner)



. The individual models would not perform well on the entire dataset, but they work well for some part of the dataset. Thus, each model actually boosts the performance of the ensemble.

Boosting, that often considers homogeneous weak learners, learns them sequentially in a very adaptive way (a base model depends on the previous ones) and combines them following a deterministic strategy

Being **mainly focused at reducing bias**, the base models that are often considered for boosting are models with low variance but high bias. For example, if we want to use trees as our base models, we will choose most of the time shallow decision trees with only a few depths.



Source: Google

Ada Boost

upGrad

Glucose	Insulin	BMI	Age	Diabetes
89	94	28.1	21	-1
137	168	43.1	33	1
78	88	31	26	-1
197	543	30.5	53	1
189	846	30.1	59	1
166	175	25.8	51	1
118	230	45.8	31	1
103	83	43.3	33	-1
115	96	34.6	32	-1
126	235	39.3	27	-1

Problem Statement : to predict whether or not a patient has diabetes based on the given features

Glucose	Insulin	BMI	Age	Diabetes	Sample weight
89	94	28.1	21	-1	0.1
137	168	43.1	33	1	0.1
78	88	31	26	-1	0.1
197	543	30.5	53	1	0.1
189	846	30.1	59	1	0.1
166	175	25.8	51	1	0.1
118	230	45.8	31	1	0.1
103	83	43.3	33	-1	0.1
115	96	34.6	32	-1	0.1
126	235	39.3	27	-1	0.1

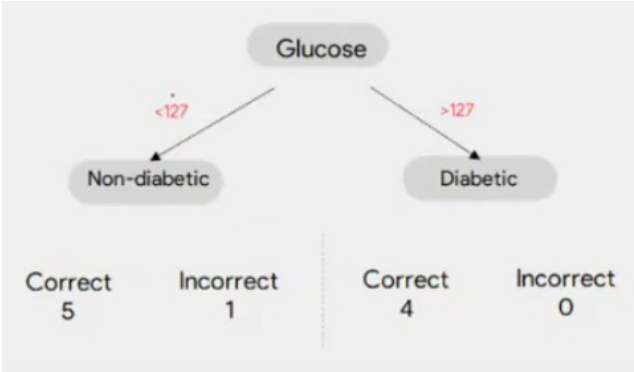
Sample Weight = 1/10 =0.1

Step 1 Initialize the sample weights

At the beginning, each sample is associated with a weight that indicates how important it is with regards to the classification

Ada Boost

upGrad



Glucose	Insulin	BMI	Age	Diabetes	Prediction	Sample weight
89	94	28.1	21	-1	-1	0.1
137	168	43.1	33	1	1	0.1
78	88	31	26	-1	-1	0.1
197	543	30.5	53	1	1	0.1
189	846	30.1	59	1	1	0.1
166	175	25.8	51	1	1	0.1
118	230	45.8	31	1	-1	0.1
103	83	43.3	33	-1	-1	0.1
115	96	34.6	32	-1	-1	0.1
126	235	39.3	27	-1	-1	0.1

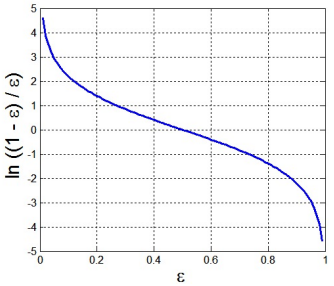
Error rate:
(*i* = index of classifier, *j*=index of instance)

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be

Weight of a classifier C_i's vote is

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Glucose	Insulin	BMI	Age	Diabetes	Prediction	Sample weight	New weights	Norm weights
89	94	28.1	21	-1	-1	0.1	0.03	0.06
137	168	43.1	33	1	1	0.1	0.03	0.06
78	88	31	26	-1	-1	0.1	0.03	0.06
197	543	30.5	53	1	1	0.1	0.03	0.06
189	846	30.1	59	1	1	0.1	0.03	0.06
166	175	25.8	51	1	1	0.1	0.03	0.06
118	230	45.8	31	1	-1	0.1	0.30	0.50
103	83	43.3	33	-1	-1	0.1	0.03	0.06
115	96	34.6	32	-1	-1	0.1	0.03	0.06
126	235	39.3	27	-1	-1	0.1	0.03	0.06

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \begin{cases} \exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

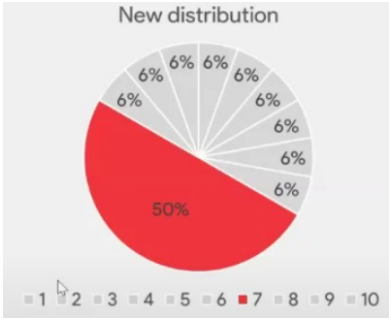
where Z_i is the normalization factor

Ada Boost

upGrad

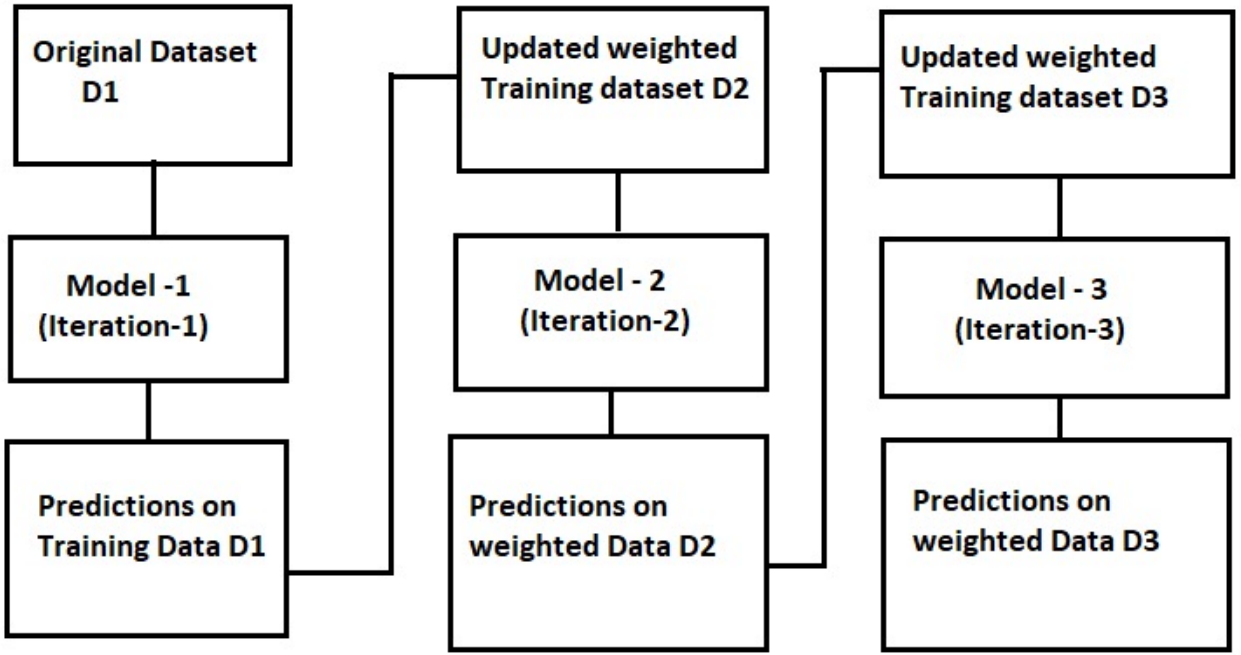
Glucose	Insulin	BMI	Age	Sample weight
118	230	45.8	31	0.1
189	846	30.1	59	0.1
115	96	34.6	32	0.1
197	543	30.5	53	0.1
118	230	45.8	31	0.1
103	83	43.3	33	0.1
126	235	39.3	27	0.1
118	230	45.8	31	0.1
118	230	45.8	31	0.1
89	94	28.1	21	0.1

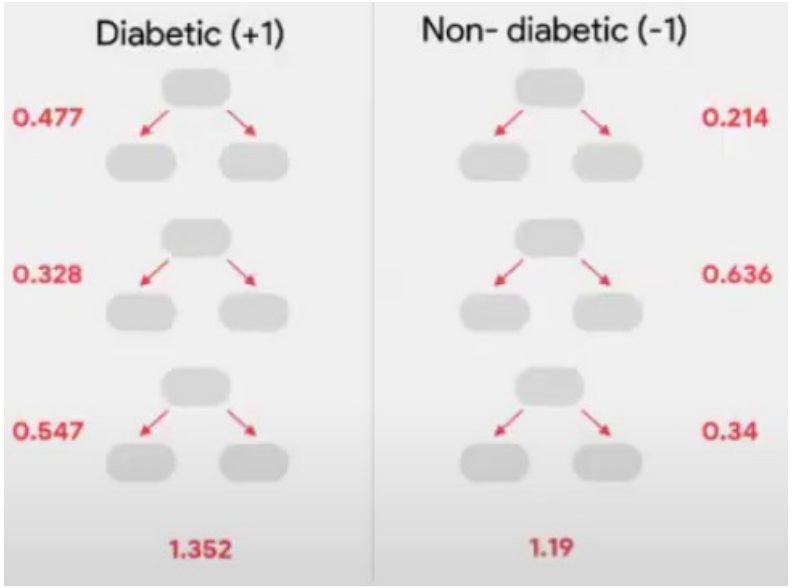
Glucose	Insulin	BMI	Age	Norm weights
89	94	28.1	21	0.06
137	168	43.1	33	0.06
78	88	31	26	0.06
197	543	30.5	53	0.06
189	846	30.1	59	0.06
166	175	25.8	51	0.06
118	230	45.8	31	0.50
103	83	43.3	33	0.06
115	96	34.6	32	0.06
126	235	39.3	27	0.06



Ada Boost

upGrad





Testing:

For each class c, sum the weights of each classifier that assigned class c to X (unseen data)

The class with the highest sum is the **WINNER!**

$$C^*(x_{test}) = \arg\max_{y} \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

Ada Boost

upGrad

class sklearn.ensemble.**AdaBoostClassifier**(*base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*) [source]

Parameters:

base_estimator : *object, default=None*
The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

n_estimators : *int, default=50*
The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate : *float, default=1.*
Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters.

algorithm : *{'SAMME', 'SAMME.R'}, default='SAMME.R'*
If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

random_state : *int, RandomState instance or None, default=None*
Controls the random seed given at each `base_estimator` at each boosting iteration. Thus, it is only used when `base_estimator` exposes a `random_state`. Pass an int for reproducible output across multiple function calls. See Glossary.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

upGrad

#LifeKoKaroLift

Thank You!