**Classification - Machine Learning**

We will learn Classification algorithms, types of classification algorithms, support vector machines(SVM), Naive Bayes, Decision Tree and Random Forest Classifier .

Objectives

Let us look at some of the objectives covered under this section of Machine Learning tutorial.

* Define Classification and list its algorithms
* Describe Logistic Regression and Sigmoid Probability
* Explain K-Nearest Neighbors and KNN classification Understand Support Vector Machines, Polynomial Kernel, and Kernel Trick
* Analyze Kernel Support Vector Machines with an example
* Implement the Naïve Bayes Classifier
* Demonstrate Decision Tree Classifier
* Describe Random Forest Classifier

Classification: Meaning

Classification is a type of supervised learning. It specifies the class to which data elements belong to and is best used when the output has finite and discrete values. It predicts a class for an input variable as well.

There are 2 types of Classification:

* Binomial
* Multi-Class

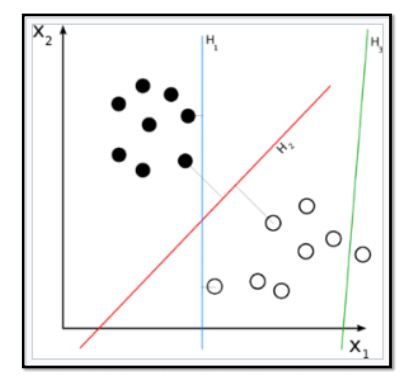
Classification: Use Cases

Some of the key areas where classification cases are being used:

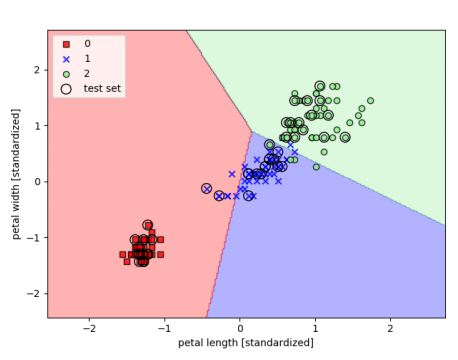
* To find whether an email received is a spam or ham
* To identify customer segments
* To find if a bank loan is granted
* To identify if a kid will pass or fail in an examination

Classification: Example

Social media sentiment analysis has two potential outcomes, positive or negative, as displayed by the chart given below.



* This chart shows the classification of the Iris flower dataset into its three sub-species indicated by codes 0, 1, and 2.



* The test set dots represent the assignment of new test data points to one class or the other based on the trained classifier model.

Types of Classification Algorithms

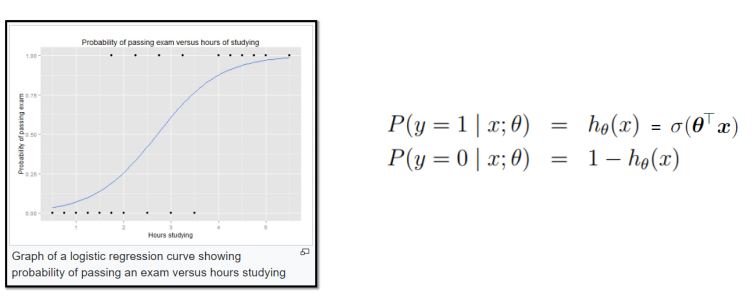
Let’s have a quick look into the types of Classification Algorithm below.

* Linear Models
  + Logistic Regression
  + Support Vector Machines
* Nonlinear models
  + K-nearest Neighbors (KNN)
  + Kernel Support Vector Machines (SVM)
  + Naïve Bayes
  + Decision Tree Classification
  + Random Forest Classification

Logistic Regression: Meaning

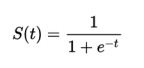
Let us understand the Logistic Regression model below.

* This refers to a regression model that is used for classification.
* This method is widely used for binary classification problems. It can also be extended to multi-class classification problems.
* Here, the dependent variable is categorical: y ϵ {0, 1}
* A binary dependent variable can have only two values, like 0 or 1, win or lose, pass or fail, healthy or sick, etc
* In this case, you model the probability distribution of output y as 1 or 0. This is called the sigmoid probability (σ).
* If σ(θ Tx) > 0.5, set y = 1, else set y = 0
* Unlike Linear Regression (and its Normal Equation solution), there is no closed form solution for finding optimal weights of Logistic Regression. Instead, you must solve this with maximum likelihood estimation (a probability model to detect the maximum likelihood of something happening).
* It can be used to calculate the probability of a given outcome in a binary model, like the probability of being classified as sick or passing an exam.

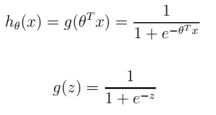


Sigmoid Probability

* The probability in the logistic regression is often represented by the Sigmoid function (also called the logistic function or the S-curve):



* In this equation, t represents data values \* the number of hours studied and S(t) represents the probability of passing the exam.
* Assume sigmoid function:

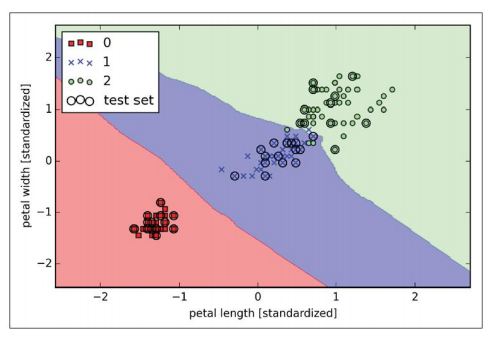


* g(z) tends toward 1 as z -> infinity , and g(z) tends toward 0 as z -> infinity

K-nearest Neighbors (KNN)

K-nearest Neighbors algorithm is used to assign a data point to clusters based on similarity measurement. It uses a supervised method for classification.

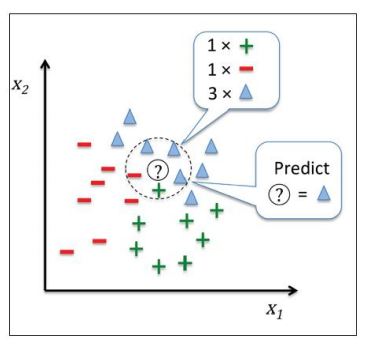
The steps to writing a k-means algorithm are as given below:



* Choose the number of k and a distance metric. (k = 5 is common)
* Find k-nearest neighbors of the sample that you want to classify
* Assign the class label by majority vote.

KNN Classification

A new input point is classified in the category such that it has the most number of neighbors from that category. For example:



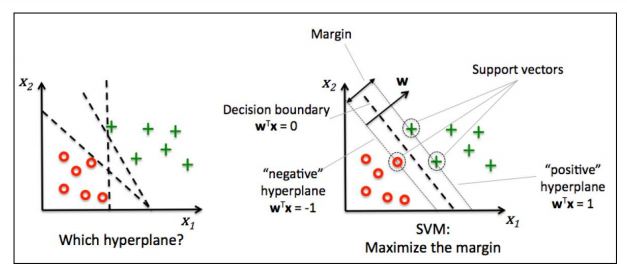
* Classify a patient as high risk or low risk.
* Mark email as spam or ham.

Keen on learning about Classification Algorithms in Machine Learning? [Click here!](https://www.simplilearn.com/big-data-and-analytics/machine-learning-certification-training-course?source=GhPreviewCTAText#/course-preview)

Support Vector Machine (SVM)

Let us understand Support Vector Machine (SVM) in detail below.

* SVMs are classification algorithms used to assign data to various classes.
* They involve detecting hyperplanes which segregate data into classes.
* SVMs are very versatile and are also capable of performing linear or nonlinear classification, regression, and outlier detection.
* Once ideal hyperplanes are discovered, new data points can be easily classified.

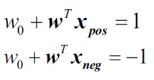


* The optimization objective is to find “maximum margin hyperplane” that is farthest from the closest points in the two classes (these points are called support vectors).
* In the given figure, the middle line represents the hyperplane.

SVM Example

Let’s look at this image below and have an idea about SVM in general.

* Hyperplanes with larger margins have lower generalization error.
* The positive and negative hyperplanes are represented by:



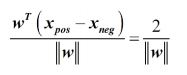
* Classification of any new input sample xtest :
  + If w0 + wTxtest > 1, the sample xtest is said to be in the class toward the right of the positive hyperplane.
  + If w0 + wTxtest < -1, the sample xtest is said to be in the class toward the left of the negative hyperplane.
* When you subtract the two equations, you get:



* Length of vector w is (L2 norm length):



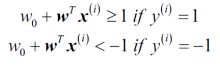
* You normalize with the length of w to arrive at:



SVM: Hard Margin Classification

Given below are some points to understand Hard Margin Classification.

* The left side of equation SVM-1 given above can be interpreted as the distance between the positive (+ve) and negative (-ve) hyperplanes; in other words, it is the margin that can be maximized.
* Hence the objective of the function is to maximize with the constraint that the samples are classified correctly, which is represented as :



* This means that you are minimizing ‖w‖.
* This also means that all positive samples are on one side of the positive hyperplane and all negative samples are on the other side of the negative hyperplane. This can be written concisely as :

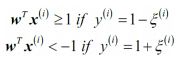


* Minimizing ‖w‖ is the same as minimizing. This figure is better as it is differentiable even at w = 0.
* The approach listed above is called “hard margin linear SVM classifier.”

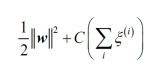
SVM: Soft Margin Classification

Given below are some points to understand Soft Margin Classification.

* To allow for linear constraints to be relaxed for nonlinearly separable data, a slack variable is introduced. (i) measures how much ith instance is allowed to violate the margin.
* The slack variable is simply added to the linear constraints.



* Subject to the above constraints, the new objective to be minimized becomes:

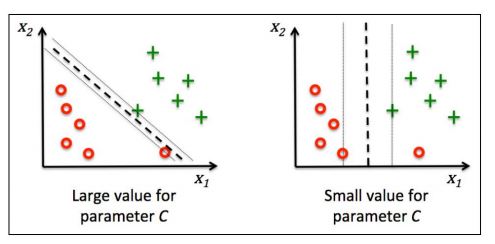


* You have two conflicting objectives now—minimizing slack variable to reduce margin violations and minimizing to increase the margin.
* The hyperparameter C allows us to define this trade-off.
* Large values of C correspond to larger error penalties (so smaller margins), whereas smaller values of C allow for higher misclassification errors and larger margins.

[](https://www.simplilearn.com/big-data-and-analytics/machine-learning-certification-training-course?source=GhPreviewCTABannert#/course-preview)

SVM: Regularization

The concept of C is the reverse of regularization. Higher C means lower regularization, which increases bias and lowers the variance (causing overfitting).



IRIS Data Set

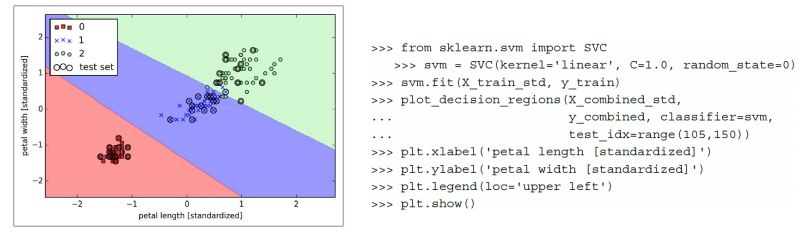
The Iris dataset contains measurements of 150 IRIS flowers from three different species:

* Setosa
* Versicolor
* Viriginica

Each row represents one sample. Flower measurements in centimeters are stored as columns. These are called features.

IRIS Data Set: SVM

Let’s train an SVM model using sci-kit-learn for the Iris dataset:

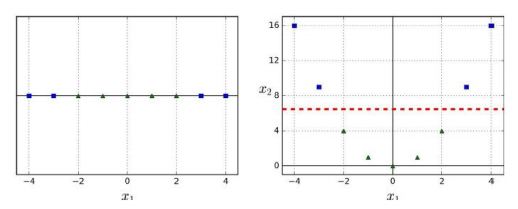


Nonlinear SVM Classification

There are two ways to solve nonlinear SVMs:

* by adding polynomial features
* by adding similarity features

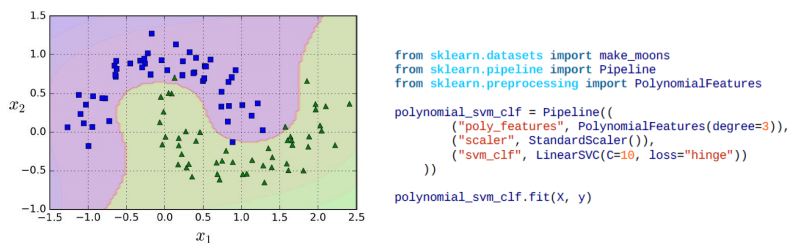
Polynomial features can be added to datasets; in some cases, this can create a linearly separable dataset.



* In the figure on the left, there is only 1 feature x1.
* This dataset is not linearly separable.
* If you add x2 = (x1)2 (figure on the right), the data becomes linearly separable.

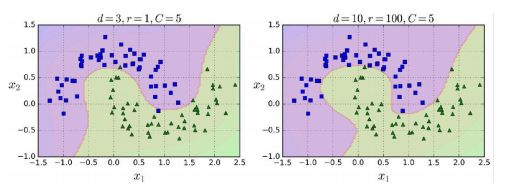
Polynomial Kernel

In sci-kit-learn, one can use a Pipeline class for creating polynomial features. Classification results for the Moons dataset are shown in the figure.

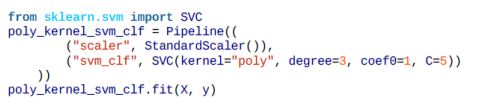


Polynomial Kernel with Kernel Trick

Let us look at the image below and understand Kernel Trick in detail.



* For large dimensional datasets, adding too many polynomial features can slow down the model.
* You can apply a kernel trick with the effect of polynomial features without actually adding them.
* The code is shown (SVC class) below trains an SVM classifier using a 3rd-degree polynomial kernel but with a kernel trick.

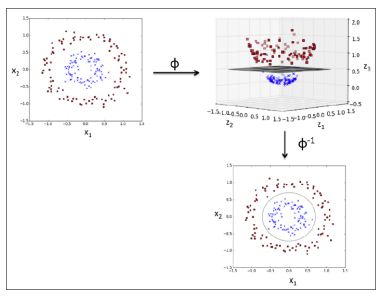


* The hyperparameter coefθ controls the influence of high-degree polynomials.

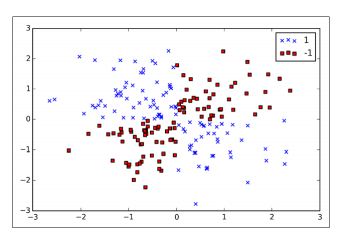
Kernel SVM

Let us understand in detail about Kernel SVM.

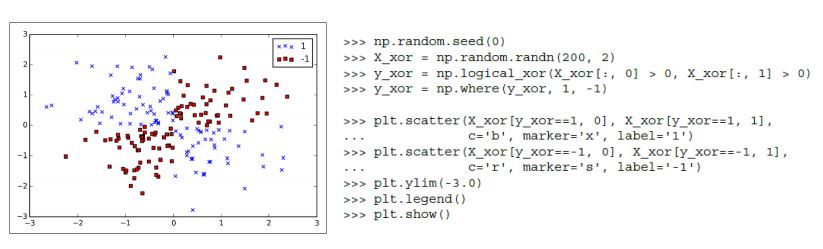
* Kernel SVMs are used for classification of nonlinear data.
* In the chart, nonlinear data is projected into a higher dimensional space via a mapping function where it becomes linearly separable.



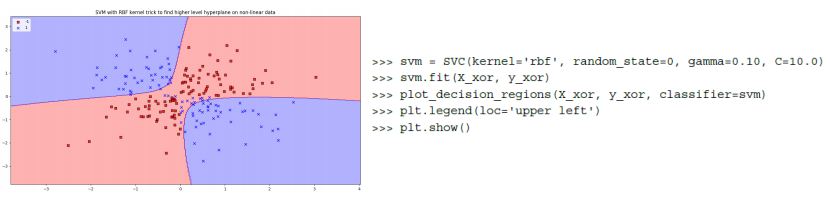
* In the higher dimension, a linear separating hyperplane can be derived and used for classification.
* A reverse projection of the higher dimension back to original feature space takes it back to nonlinear shape.
* As mentioned previously, SVMs can be kernelized to solve nonlinear classification problems.
* You can create a sample dataset for XOR gate (nonlinear problem) from NumPy. 100 samples will be assigned the class sample 1, and 100 samples will be assigned the class label -1.



As you can see, this data is not linearly separable.



You now use the kernel trick to classify XOR dataset created earlier.



Naïve Bayes Classifier

What is Naive Bayes Classifier?

Have you ever wondered how your mail provider implements spam filtering or how online news channels perform news text classification or even how companies perform sentiment analysis of their audience on social media? All of this and more are done through a machine learning algorithm called Naive Bayes Classifier.

Naive Bayes

Named after Thomas Bayes from the 1700s who first coined this in the Western literature. Naive Bayes classifier works on the principle of conditional probability as given by the Bayes theorem.

Advantages of Naive Bayes Classifier

Listed below are six benefits of Naive Bayes Classifier.

* Very simple and easy to implement
* Needs less training data
* Handles both continuous and discrete data
* Highly scalable with the number of predictors and data points
* As it is fast, it can be used in real-time predictions
* Not sensitive to irrelevant features

Bayes Theorem

We will understand Bayes Theorem in detail from the points mentioned below.

* According to the Bayes model, the conditional probability P(Y|X) can be calculated as:

P(Y|X) = P(X|Y)P(Y) / P(X)

* This means you have to estimate a very large number of P(X|Y) probabilities for a relatively small vector space X.
* For example, for a Boolean Y and 30 possible Boolean attributes in the X vector, you will have to estimate 3 billion probabilities P(X|Y).
* To make it practical, a Naïve Bayes classifier is used, which assumes conditional independence of P(X) to each other, with a given value of Y.
* This reduces the number of probability estimates to 2\*30=60 in the above example.

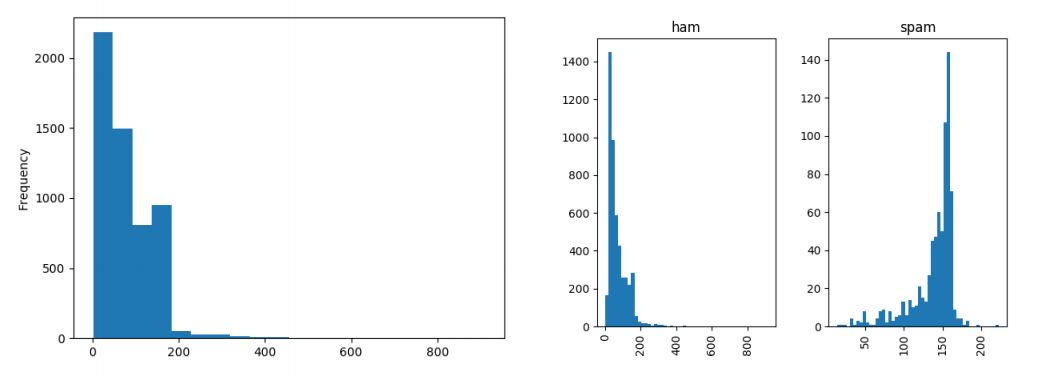
Naïve Bayes Classifier for SMS Spam Detection

Consider a labeled SMS database having 5574 messages. It has messages as given below:



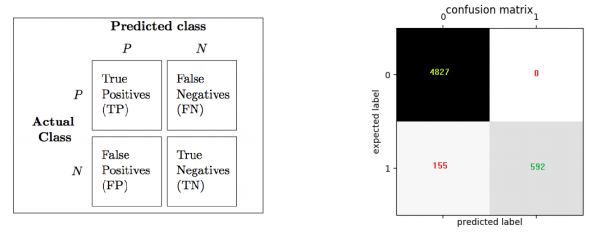
* Each message is marked as spam or ham in the data set.
* Let’s train a model with Naïve Bayes algorithm to detect spam from ham.

The message lengths and their frequency (in the training dataset) are as shown below:

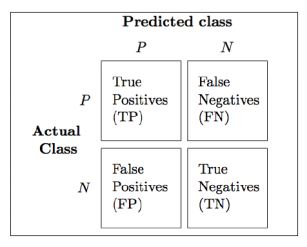


Analyze the logic you use to train an algorithm to detect spam:

* Split each message into individual words/tokens (bag of words).
* Lemmatize the data (each word takes its base form, like “walking” or “walked” is replaced with “walk”).
* Convert data to vectors using scikit-learn module CountVectorizer.
* Run TFIDF to remove common words like “is,” “are,” “and.”
* Now apply scikit-learn module for Naïve Bayes MultinomialNB to get the Spam Detector.
* This spam detector can then be used to classify a random new message as spam or ham.
* Next, the accuracy of the spam detector is checked using the Confusion Matrix.
* For the SMS spam example above, the confusion matrix is shown on the right.
* Accuracy Rate = Correct / Total = (4827 + 592)/5574 = 97.21% Error Rate = Wrong / Total = (155 + 0)/5574 = 2.78%



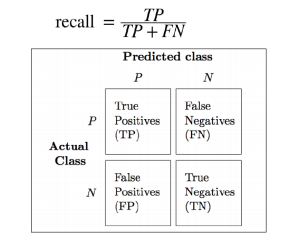
Although confusion Matrix is useful, some more precise metrics are provided by Precision and Recall.



* Precision refers to the accuracy of positive predictions.



* Recall refers to the ratio of positive instances that are correctly detected by the classifier (also known as True positive rate or TPR).



Precision/Recall Trade-off

To detect age-appropriate videos for kids, you need high precision (low recall) to ensure that only safe videos make the cut (even though a few safe videos may be left out).

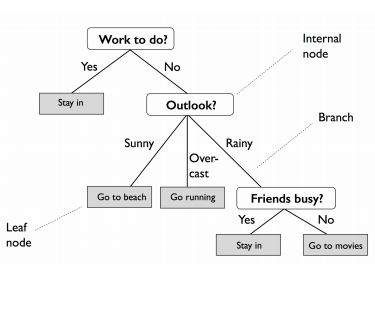
The high recall is needed (low precision is acceptable) in-store surveillance to catch shoplifters; a few false alarms are acceptable, but all shoplifters must be caught.

Learn about Naive Bayes in detail. [Click here!](https://www.simplilearn.com/big-data-and-analytics/machine-learning-certification-training-course?source=GhPreviewCTAText#/course-preview)

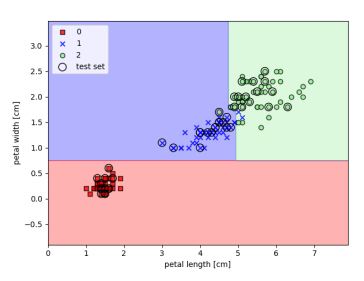
Decision Tree Classifier

Some aspects of the Decision Tree Classifier mentioned below are.

* Decision Trees (DT) can be used both for classification and regression.
* The advantage of decision trees is that they require very little data preparation.
* They do not require feature scaling or centering at all.
* They are also the fundamental components of Random Forests, one of the most powerful ML algorithms.
* Unlike Random Forests and Neural Networks (which do black-box modeling), Decision Trees are white box models, which means that inner workings of these models are clearly understood.
* In the case of classification, the data is segregated based on a series of questions.
* Any new data point is assigned to the selected leaf node.

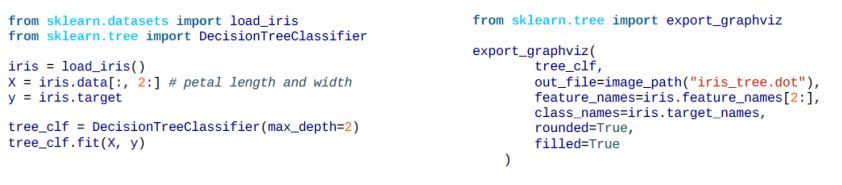


* Start at the tree root and split the data on the feature using the decision algorithm, resulting in the largest information gain (IG).
* This splitting procedure is then repeated in an iterative process at each child node until the leaves are pure.
* This means that the samples at each node belonging to the same class.
* In practice, you can set a limit on the depth of the tree to prevent overfitting.
* The purity is compromised here as the final leaves may still have some impurity.
* The figure shows the classification of the Iris dataset.



IRIS Decision Tree

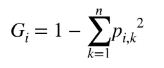
Let’s build a Decision Tree using scikit-learn for the Iris flower dataset and also visualize it using export\_graphviz API.



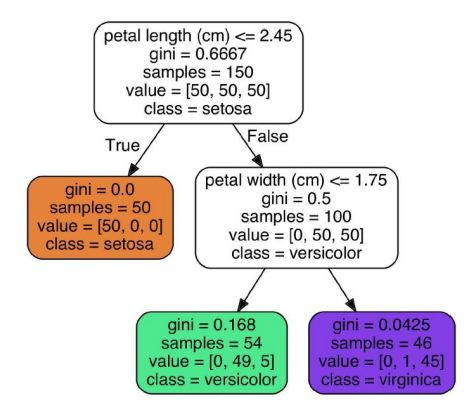
* The output of export\_graphviz can be converted into png format:



* Sample attribute stands for the number of training instances the node applies to.
* Value attribute stands for the number of training instances of each class the node applies to.
* Gini impurity measures the node’s impurity. A node is “pure” (gini=0) if all training instances it applies to belong to the same class.



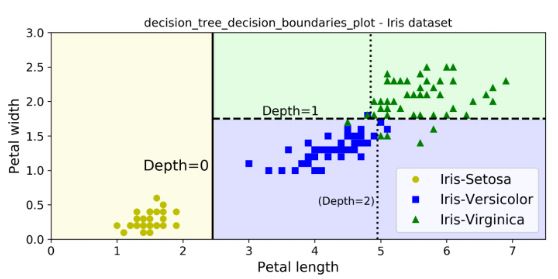
* For example, for Versicolor (green color node), the Gini is 1-(0/54)2 -(49/54)2 -(5/54) 2 ≈ 0.168



Decision Boundaries

Let us learn to create decision boundaries below.

* For the first node (depth 0), the solid line splits the data (Iris-Setosa on left). Gini is 0 for Setosa node, so no further split is possible. The second node (depth 1) splits the data into Versicolor and Virginica.
* If max\_depth were set as 3, a third split would happen (vertical dotted line).

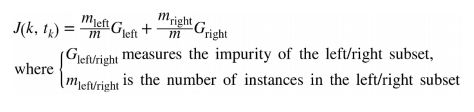


* For a sample with petal length 5 cm and petal width 1.5 cm, the tree traverses to depth 2 left node, so the probability predictions for this sample are 0% for Iris-Setosa (0/54), 90.7% for Iris-Versicolor (49/54), and 9.3% for Iris-Virginica (5/54)

CART Training Algorithm

Scikit-learn uses Classification and Regression Trees (CART) algorithm to train Decision Trees. CART algorithm:

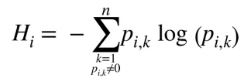
* Split the data into two subsets using a single feature k and threshold tk (example, petal length < “2.45 cm”). This is done recursively for each node.
* k and tk are chosen such that they produce the purest subsets (weighted by their size). The objective is to minimize the cost function as given below:



* The algorithm stops executing if one of the following situations occurs:
  + max\_depth is reached
  + No further splits are found for each node
  + Other hyperparameters may be used to stop the tree:
    - min\_samples\_split
    - min\_samples\_leaf
    - min\_weight\_fraction\_leaf
    - max\_leaf\_nodes

Gini Impurity or Entropy

Entropy is one more measure of impurity and can be used in place of Gini.



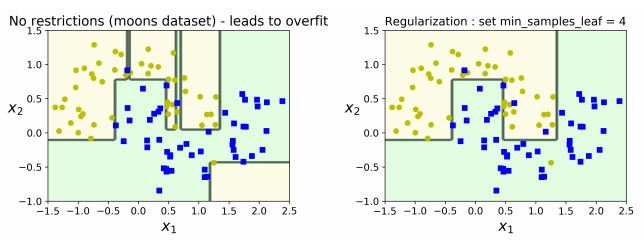
It is a degree of uncertainty, and Information Gain is the reduction that occurs in entropy as one traverses down the tree. Entropy is zero for a DT node when the node contains instances of only one class. Entropy for depth 2 left node in the example given above is:



Gini and Entropy both lead to similar trees.

DT: Regularization

The following figure shows two decision trees on the moons dataset.

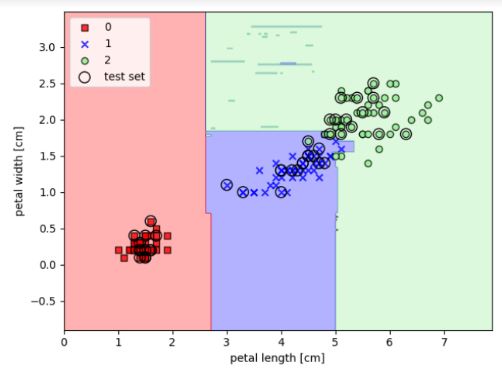


* The decision tree on the right is restricted by min\_samples\_leaf = 4.
* The model on the left is overfitting, while the model on the right generalizes better.

Random Forest Classifier

Let us have an understanding of Random Forest Classifier below.

* A random forest can be considered an ensemble of decision trees (Ensemble learning).
* Random Forest algorithm:
  + Draw a random bootstrap sample of size n (randomly choose n samples from the training set).
  + Grow a decision tree from the bootstrap sample. At each node, randomly select d features.
  + Split the node using the feature that provides the best split according to the objective function, for instance by maximizing the information gain.
  + Repeat the steps 1 to 2 k times. (k is the number of trees you want to create, using a subset of samples)
  + Aggregate the prediction by each tree for a new data point to assign the class label by majority vote (pick the group selected by the most number of trees and assign new data point to that group).
* Random Forests are opaque, which means it is difficult to visualize their inner workings.



* However, the advantages outweigh their limitations since you do not have to worry about hyperparameters except k, which stands for the number of decision trees to be created from a subset of samples.
* RF is quite robust to noise from the individual decision trees. Hence, you need not prune individual decision trees.
* The larger the number of decision trees, the more accurate the Random Forest prediction is. (This, however, comes with higher computation cost).

Key Takeaways

Let us quickly run through what we have learned so far in this Classification tutorial.

* Classification algorithms are supervised learning methods to split data into classes. They can work on Linear Data as well as Nonlinear Data.
* Logistic Regression can classify data based on weighted parameters and sigmoid conversion to calculate the probability of classes. K-nearest Neighbors (KNN) algorithm uses similar features to classify data.
* Support Vector Machines (SVMs) classify data by detecting the maximum margin hyperplane between data classes.
* Naïve Bayes, a simplified Bayes Model, can help classify data using conditional probability models.
* Decision Trees are powerful classifiers and use tree splitting logic until pure or somewhat pure leaf node classes are attained.
* Random Forests apply Ensemble Learning to Decision Trees for more accurate classification predictions.

Conclusion

This completes ‘Classification’ tutorial. In the next tutorial, we will learn ['Unsupervised Learning with Clustering.'](https://www.simplilearn.com/unsupervised-learning-with-clustering-machine-learning-tutorial)

In this article we will look at logistic classifier and how regularization affects the performance of the classifier.

Logistic regression classifier and how regularization affects the performance of the classifier.

Training a machine learning algorithms involves optimization techniques.However apart from providing good accuracy on training and validation data sets ,it is required the machine learning to have good generalization accuracy.The machine learning algorithms should perform well on unseen examples as well.The model is trained by optimizing its performance over some training dataset however its performance is determined on its ability to perform on unseen datasets.

The term over fitting is often used when machine learning algorithm has high accuracy on training data set but poor generalization accuracy.

Problem of Overfitting

Over-fitting generally occurs when a model is excessively complex.A model that has been overfit will generally have poor generalization capabilities , as it can perform errors due to minor fluctuations in data dues to noise and other parameters which were not modeled during the training process.

Just because a model agrees with training data does not mean it will perform well on unseen examples and is not necessarily a good model.In case of overfitting the model just tries to learn pecularities of the training set and does not work well on unseen examples that differ from the training set.

In general over-fitting can be associated with complexity.In multiclass logistic classifier for MNIST digit classification where are 7850 free parameters that are optimized. This is a large number of parameters.

Training,Validation and Testing Datasets

Typically we can get some idea as to if over-fitting has been performed on not by periodically testing the model on unseen test data set.Thus during the training process we make use of 3 types of data sets.

Training data set that is primarily used for learning.

The validation data set which is a part of training data set but not used during the learning and

test data set that contains a large variations of typically input data that represents all possible unseen examples.

The validation data set is the unseen data set against when we can test the model to check for over-fitting and generalization performance.The learning algorithm has not seen the samples from the validation data set.Validation data sets is used to test the various choices of the model parameters .Thus validation data set which is again considered a part of training dataset is used to periodically evaluate the model parameters.Based on the performance of the model we decide if over-fitting has occurred,learning process has been saturated and possibly adaptively tune the model.

The difference between validation data set and test data set is that validation data set is not as challenging as the test data set.It contains unseen examples but may not contain a large variation or all possible end cases or difficult examples.

The test data is primarily used to check if the model truly performs as expected agains a large variation in data.That is why the test data set is supposed to contains samples with large variations in data and difficult and end cases ,which are not included in the training data .If we use test data directly for validation purpose we may bias the classifier to overfit or choose parameters according to the test data ,which is again undesirable for the sake of generality

Validation Curves

If the we find that validation accuracy or training accuracy is increasing while the accuracy on test data is decreasing we can assume that this is due to over-fitting.Intuitively, over-fitting occurs when the model or the algorithm fits the data too well.

In some causes we may observe that accuracy on the training data is reducing while that on the test data is constant .This also may imply over-fitting. Since model is trying to fit the training data better by updating the model ,however updating the model does not affect its performance on the test data set.

Avoid Overfitting

In the article we look at logistic regression classifier and how to handle the cases of overfitting

Increasing size of dataset

One of the ways to combat over-fitting is to increase the training data size.Let take the case of MNIST data set trained with 5000 and 50000 examples,using similar training process and parameters.

Below are observed errors on the training,validation and testing dataset

at the 250-th iteration the errors are

training accuracy is 12.34 %

testing accuracy is 12.0 %

validation accuracy is 11.29 %

we can observe that training and validation errors steadily decrease during the initial part of the learning process.However after 100-th iteration we can observe that rate at which training error improves is larger that that of validation errors.

The validation error essentially remains constant.At 120-th iteration we see again some improvement in validation curves ,but still rate of error on training dataset is higher.After the 140-th iteration it is validation curves essentially represents a straight line indicating no learning though the training error keeps on reducing.This may be an indication of over-fitting.

Now lets perform the same testing with the complete set of 50000 samples.Of cource the absolute error will be better when we include a larger training set as the classifier would learn to handle larger variations in data.But what is essential is to observe the rate of change of errors and not the absolute errors.

we can see at 250-iteration the errors are

training accuracy is 11.34 %

testing accuracy is 10.0 %

validation accuracy is 10.29 %

This is almost the same as the earlier training process.This implies that training of data from 5000 samples dataset for 250 iterations gives a similar performance a using a larger data set.This may lead us to conclude that learning algorithm has learn efficiently from data.However what is essential to observe that there is no saturation of error curves.The error curves are dropping at a larger rate at 300 in the present example than the earlier one.Thus if we had let the training continue the error rates would have dropped even more before saturating.Especially we would have obtained lower error on test and validation data sets.

Thus increasing the training sample size has avoided over-fitting at 300-th iteration of the training process and will lead to better generalization performance.

Early stopping

Another way to combat over-fitting is to perform early stopping.

Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit. Early stopping rules have been employed in many different machine learning methods, with varying amounts of theoretical foundation

Early stopping based on cross-validation combats over-fitting by monitoring the model’s performance on a validation set.The error on the validation set is used as a proxy for the generalization error in determining when over-fitting has begun.

Many ad-hoc rules for deciding when over-fitting has truly begun are used in different implementation but few how them have any theoretical foundation.

One of rules may be that if we see that the error on the training data is decreasing while error on the validation dataset remains constant or begins to decrease,this may indicate that we need to stop the training process or it might lead to over-fitting.

One of the commonly used criteria for early stopping is rate of change fo validation error.If the validation error does not change significantly in successive iterations ,we reduces the number of iterations for which gradient based learning is performed.

As mentioned earlier A validation set is a set of examples that we never use for gradient descent, but which is also not a part of the test set.The validation examples are considered to be representative of future test examples.If the model’s performance ceases to improve sufficiently on the validation set, or even degrades with further optimization, then the some heuristic's can be employed to cease further optimization.

In present article we specify a strategy based on a geometrically increasing amount of patience which are used in deeplearning.net tutorials in stochastic gradient learning based frameworks.We will look at stochastic gradient descent algorithms in the future artciles .The gradient based learning algorithms executes the learning algorithms N times.

In mathematics, a geometric progression, also known as a geometric sequence, is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the common ratio.

The heuristic used is the geometric progression of validation error.Let us consider the common ratio of 0.9.if best validation error be 0.5 and next one is less than 0.9\*0.5=0.45 the we consider that signifiant reduction in error has occurred.

if error has reduced significantly then we increase the number of iteration the algorithm is supposed to run by a geometric factor and if there is no significant increase in error we reduce the number of iteration to be run by a geometric factor.If the max number of iterations have been reached then we say that learning process has been completed and retain the parameters of the best learning iteration.

Using this criteria the for training samples with 5000 samples,the learning stops at 134-th iteration with

training accuracy is 12.82 %

testing accuracy is 11.4 %

validation accuracy is 12.09 %

we can see that at even 250-th iteration we had not obtained a significant improvement in the performance

training accuracy is 12.34 %

testing accuracy is 12.0 %

validation accuracy is 11.29 %

Thus early stopping helps prevent over fitting.

The code for validation heuristics is as follows

Hide Copy Code

if error3 < self.best\_validation\_loss:

self.best\_validation\_loss=error3;

if error3 < self.best\_validation\_loss \*self.improvement\_threshold:

self.patience = max(self.patience, self.iter \* self.patience\_increase)

self.best\_validation\_loss = error3

self.best\_iter = self.iter

else:

self.patience = min(self.patience,self.iter+self.iter/self.patience\_increase)

error3 is the current validation error in the above code. self.improvement\_threshold is error improvement factor typically takes value between 0.9-1 self.patience\_increase is geometric iteration factor,typically a value greater than 2. The number of iterations is reduced by this factor every time validation heuristic is not met.

Regularization

A way to combat over-fitting is through regularization. Regularization techniques can be viewed as imposing certain prior distribution on the model parameters.Mathematically the regularization process implies performing constrained optimization.

L2 regularization implies imposing Gaussian prior on weights while L1 prior implies imposing Laplacian prior on weights.

There are several regularization techniques .However in this article we will look at Lp regularization and its effect on Logistic classification process.

Lp regularization involve adding an extra term to the cost function, which penalizes certain parameter configurations

If the Original loss function is defined as

L(θ,D)=−∑i=1NlogP(Y=yi|X=xi,θ)

The regularized loss function is given by

E(θ,D)=L(θ,D)+λR(θ)

In the general in the case of (L\_{p}) regularization

R′(θ)=λ||θ||pp=λ(∑j=0|θ||θj|p)

where ||θ||pp is the Lp norm of θ and λ is a parameter which controls the relative importance of the regularization parameter. Generally L1 or L2 norms are used .

Intuitively adding the regularization term will penalizes large values of parameters which decrease the amount of non-linearity that the network models.Adding a regularization term will have the effect of simplifying the models and improve the performance in presence of over fitting. Thus performing minimization of loss functions in presence of regularization term will provide us the simplest model that can fit the training data.

In the earlier article "MultiClass Logistic Regression in Python" the optimum parameters of the classifier were determined by minimizing the cost function.We had computed the gradient of the cost function wrt to the parameters.

due to addition of regularization term to the cost function,the gradient cost function will have additional terms corresponding to gradient of L2 norm.

The first effect of the regularization term is on the cost function.The cost function will have a prior likelihood on account of the regularization term.

Hide Copy Code

def negative\_log\_likelihood(self,params):

# args contains the training data

x,y=self.args;

self.update\_params(params);

sigmoid\_activation = pyvision.softmax(self.W,self.b,x);

index=[range(0,np.shape(sigmoid\_activation)[0]),y];

p=sigmoid\_activation[index]

l=-np.mean(np.log(p));

if self.Regularization==2:

l=l+0.5\*self.eta\*np.sum(self.W\*\*2);

if self.Regularization==1:

l=l+self.eta\*np.sum(np.abs(self.W));

return l;

The next changes is in the gradient computation.

Hide Copy Code

""" function to compute the gradient of parameters for a single data sample """

def compute\_gradients(self,out,y,x):

out=(np.reshape(out,(np.shape(out)[0],1)));

out[y]=out[y]-1;

W=out\*x.T;

if self.Regularization==2:

W=W+self.eta\*self.W;

if self.Regularization==1:

W=W+self.eta\*np.sign(W);

res=np.vstack((W.T,out.flatten()))

return res;

where (i) indicates the class for which the derivative is being computed.

L2 regularization based optimization is simple since the additional cost function added is continous and differentiable. For.For L1 regularization we use the basic sub-gradient method to compute the derivatives.

First we look at L2 regularization process.

L2 Regularization

The regularization is affected by regularization constant.Often the process is to determine the constant empirically by running the training with various values

.

A large values of constant will lead to flattening of error curves very soon an generally model will exhibit a lower accuracy.However a small values will lead to large number of iternations being performed and may lead to over-fitting.

These can be seen from below curves for L2 regularization

regularization parameter=0.9

regularization parameter=0.1

we can see that with parameter value of 0.9 learning saturates at around 18% error at 40 iterations while for the value 0,1 learning saturates at iteration 120 with error of 15%.Thus regularization process in generally prevents the learning algorithm to respond to outliers in data by controlling the weight parameters.

Thus we can observe inherent trade off.Regularization prevents the model to respond to variation in data,which in turn leads to lower accuracy of the model in learning all possible variations of data accurately.This is the case of bias variance trade off.

## What is Boosting?

Definition: The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.

Let’s understand this definition in detail by solving a problem of spam email identification:

How would you classify an email as SPAM or not? Like everyone else, our initial approach would be to identify ‘spam’ and ‘not spam’ emails using following criteria. If:

1. Email has only one image file (promotional image), It’s a SPAM
2. Email has only link(s), It’s a SPAM
3. Email body consist of sentence like “You won a prize money of $ xxxxxx”, It’s a SPAM
4. Email from our official domain “[Analyticsvidhya.com](http://analyticsvidhya.com/)” , Not a SPAM
5. Email from known source, Not a SPAM

Above, we’ve defined multiple rules to classify an email into ‘spam’ or ‘not spam’. But, do you think these rules individually are strong enough to successfully classify an email? No.

Individually, these rules are not powerful enough to classify an email into ‘spam’ or ‘not spam’. Therefore, these rules are called as **weak learner**.

To convert weak learner to strong learner, we’ll combine the prediction of each weak learner using methods like:  
•   Using average/ weighted average  
•   Considering prediction has higher vote

For example:  Above, we have defined 5 weak learners. Out of these 5, 3 are voted as ‘SPAM’ and 2 are voted as ‘Not a SPAM’. In this case, by default, we’ll consider an email as SPAM because we have higher(3) vote for ‘SPAM’.

## How Boosting Algorithms works?

Now we know that, boosting combines weak learner a.k.a. base learner to form a strong rule. An immediate question which should pop in your mind is, ‘How boosting identify weak rules?‘

To find weak rule, we apply base learning (ML) algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Here’s another question which might haunt you, ‘How do we choose different distribution for each round?’

For choosing the right distribution, here are the following steps:

Step 1:  The base learner takes all the distributions and assign equal weight or attention to each observation.

Step 2: If there is any prediction error caused by first base learning algorithm, then we pay higher attention to observations having prediction error. Then, we apply the next base learning algorithm.

Step 3: Iterate Step 2 till the limit of base learning algorithm is reached or higher accuracy is achieved.

Finally, it combines the outputs from weak learner and creates  a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are mis-classiﬁed or have higher errors by preceding weak rules.

## Types of Boosting Algorithms

Underlying engine used for boosting algorithms can be anything.  It can be decision stamp, margin-maximizing classification algorithm etc. There are many boosting algorithms which use other types of engine such as:

1. AdaBoost (**Ada**ptive **Boost**ing)
2. Gradient Tree Boosting
3. XGBoost

In this article, we will focus on AdaBoost and Gradient Boosting followed by their respective python codes and will focus on XGboost in upcoming article.

ypes of Boosting Algorithms

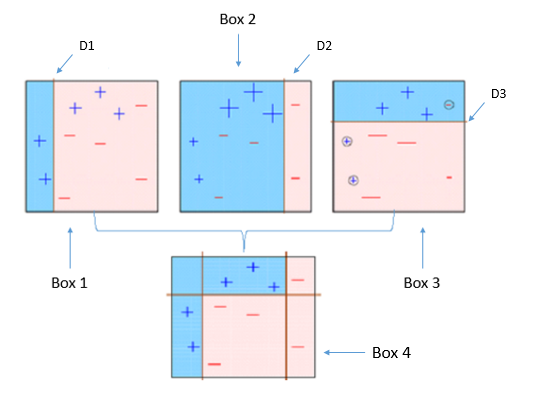
Underlying engine used for boosting algorithms can be anything. It can be decision stamp, margin-maximizing classification algorithm etc. There are many boosting algorithms which use other types of engine such as:

AdaBoost (Adaptive Boosting)

Gradient Tree Boosting

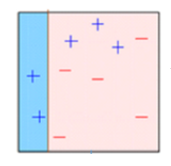
XGBoost

Boosting Algorithm: AdaBoost

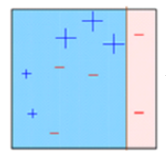


This diagram aptly explains Ada-boost. Let’s understand it closely:

Box 1: You can see that we have assigned equal weights to each data point and applied a decision stump to classify them as + (plus) or – (minus). The decision stump (D1) has generated vertical line at left side to classify the data points. We see that, this vertical line has incorrectly predicted three + (plus) as – (minus). In such case, we’ll assign higher weights to these three + (plus) and apply another decision stump.

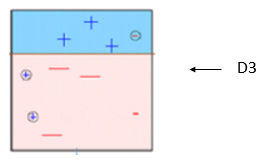


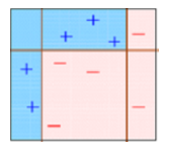
Box 2: Here, you can see that the size of three incorrectly predicted + (plus) is bigger as compared to rest of the data points. In this case, the second decision stump (D2) will try to predict them correctly. Now, a vertical line (D2) at right side of this box has classified three mis-classified + (plus) correctly. But again, it has caused mis-classification errors. This time with three -(minus). Again, we will assign



higher weight to three – (minus) and apply another decis ion stump.

Box 3: Here, three – (minus) are given higher weights. A decision stump (D3) is applied to predict these mis-classified observation correctly. This time a horizontal line is generated to classify + (plus) and – (minus) based on higher weight of mis-classified observation



Box 4: Here, we have combined D1, D2 and D3 to form a strong prediction having complex rule as compared to individual weak learner. You can see that this algorithm has classified these observation quite well as compared to any of individual weak learner. 

AdaBoost (Adaptive Boosting) : It works on similar method as discussed above. It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.

Mostly, we use decision stamps with AdaBoost. But, we can use any machine learning algorithms as base learner if it accepts weight on training data set. We can use AdaBoost algorithms for both classification and regression problem.

ou can tune the parameters to optimize the performance of algorithms, I’ve mentioned below the key parameters for tuning:

n\_estimators: It controls the number of weak learners.

learning\_rate:Controls the contribution of weak learners in the final combination. There is a trade-off between learning\_rate and n\_estimators.

base\_estimators: It helps to specify different ML algorithm.

You can also tune the parameters of base learners to optimize its performance.

Boosting Algorithm: Gradient Boosting

In gradient boosting, it trains many model sequentially. Each new model gradually minimizes the loss function (y = ax + b + e, e needs special attention as it is an error term) of the whole system using Gradient Descent method. The learning procedure consecutively fit new models to provide a more accurate estimate of the response variable.

The principle idea behind this algorithm is to construct new base learners which can be maximally correlated with negative gradient of the loss function, associated with the whole ensemble. You can refer article “Learn Gradient Boosting Algorithm” to understand this concept using an example.

In Python Sklearn library, we use Gradient Tree Boosting or GBRT. It is a generalization of boosting to arbitrary differentiable loss functions. It can be used for both regression and classification problems.

Python Code

from sklearn.ensemble import GradientBoostingClassifier #For Classification

from sklearn.ensemble import GradientBoostingRegressor #For Regression

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1.0, max\_depth=1)

clf.fit(X\_train, y\_train)

n\_estimators: It controls the number of weak learners.

learning\_rate:Controls the contribution of weak learners in the final combination. There is a trade-off between learning\_rate and n\_estimators.

max\_depth: maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

You can tune loss function for better performance.

End Note

In this article, we looked at boosting, one of the method of ensemble modeling to enhance the prediction power. Here, we have discussed the science behind boosting and its two types: AdaBoost and Gradient Boost. We also studied their respective python codes.

In my next article, I will discuss about another type of boosting algorithms which is now a days secret of wining data science competitions “XGBoost”.