Ensemble technique is one of the most fundamental algorithms for classification and regression in the Machine Learning world.

We regularly come across many game shows on television and you must have noticed an option of “Audience Poll”. Most of the times a contestant goes with the option which has the highest vote from the audience and most of the times they win. We can generalize this in real life as well where taking opinions from a majority of people is much more preferred than the opinion of a single person. Ensemble technique has a similar underlying idea where we aggregate predictions from a group of predictors, which may be classifiers or regressors, and most of the times the prediction is better than the one obtained using a single predictor. Such algorithms are called Ensemble methods and such predictors are called Ensembles.

Let’s suppose we have ‘n’ predictors: Z1, Z2, Z3, ......., Zn with a standard deviation of σ

Variance(z) = σ^2

If we use single predictors Z1, Z2, Z3, ......., Zn the variance associated with each will be σ2 but the expected value will be the average of all the predictors.

Let’s consider the average of the predictors: µ = (Z1 + Z2 + Z3+.......+ Zn)/n

if we use µ as the predictor then the expected value still remains the same but see the variance now:

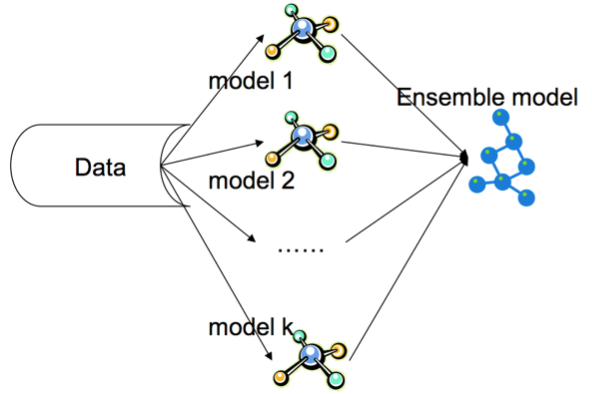
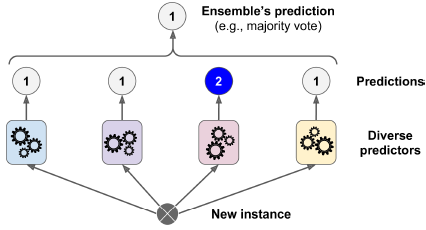
variance(µ) = σ^2/n

So, the expected value remained ‘µ’ but variance decreases when we use average of all the predictors.

This is why taking mean is preferred over using single predictors.

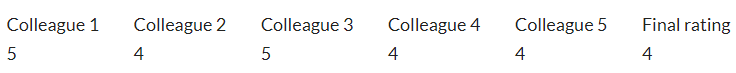
Ensemble methods take multiple small models and combine their predictions to obtain a more powerful predictive power.

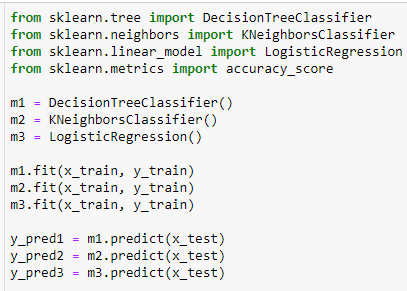
There are few very popular Ensemble techniques which we will talk about in detail such as Bagging, Boosting, stacking etc.

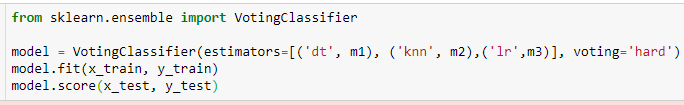


Simple Ensemble Techniques:

**Max Voting:** The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.







0.7430894308943089

**Averaging**

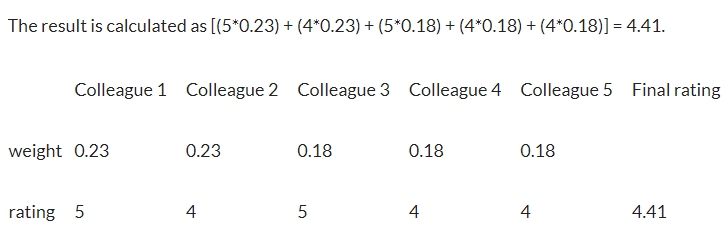
Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

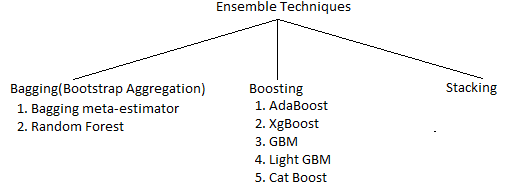


**Weighted Average**

This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.

finalpred = (y\_pred1\*0.3+y\_pred2\*0.3+y\_pred3\*0.4)





**Bagging(bootstrap aggregation)**

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Here’s a question: If you create all the models on the same set of data and combine it, will it be useful? There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.

Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

In real life scenarios we don’t have multiple different training sets on which we can train our model separately and at the end combine their result. Here, bootstrapping comes into picture. Bootstrapping is a technique of sampling different sets of data from a given training set by using replacement. After bootstrapping the training dataset, we train model on all the different sets and aggregate the result. This technique is known as Bootstrap Aggregation or Bagging.

**Bagging** is the type of ensemble technique in which a single training algorithm is used on different subsets of the training data where the subset sampling is done with replacement (bootstrap). Once the algorithm is trained on all the subsets, then bagging makes the prediction by aggregating all the predictions made by the algorithm on different subsets. In case of regression, bagging prediction is simply the mean of all the predictions and in the case of classifier, bagging prediction is the most frequent prediction (majority vote) among all the predictions.

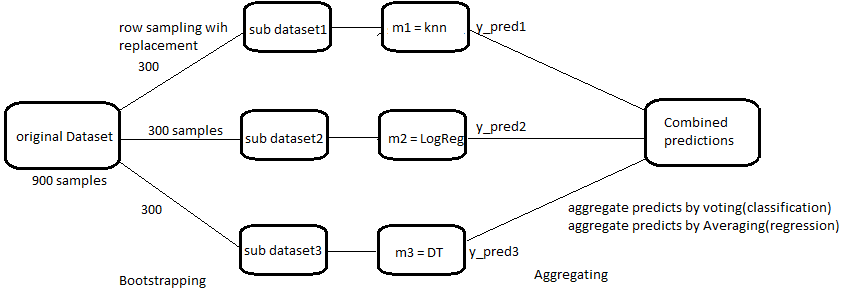
Bagging is also known as parallel model since we run all models parallely and combine there results at the end.

Multiple subsets are created from the original dataset, selecting observations with replacement.

A base model (weak model) is created on each of these subsets.

The models run in parallel and are independent of each other.

The final predictions are determined by combining the predictions from all the models.



**Advantages of a Bagging Model**

* Bagging significantly decreases the variance without increasing bias.
* Bagging methods work so well because of diversity in the training data since the sampling is done by bootstrapping.
* Also, if the training set is very huge, it can save computational time by training model on relatively smaller data set and still can increase the accuracy of the model.
* Works well with small datasets as well.

**Disadvantage of a Bagging Model**[¶](http://localhost:8888/notebooks/MachineLearningOfficial/6_1_EnsembleLearning_And_RandomForest/EnsembleLearningAndRandomForest.ipynb#Disadvantage-of-a-Bagging-Model)

The main disadvantage of Bagging is that it improves the accuracy of the model on the expense of interpretability i.e. if a single tree was being used as the base model, then it would have a more attractive and easily interpretable diagram, but with use of bagging this interpretability gets lost.

**Difference between kfold and bagging**

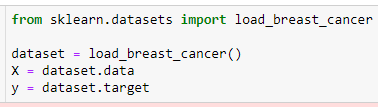
* In kfold we will not get repetition(overlap) of data
* but in bagging we will get repetitive(overlap) of data

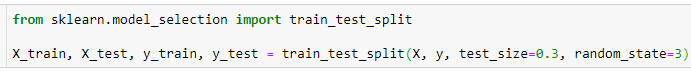
**Pasting**

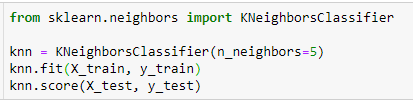
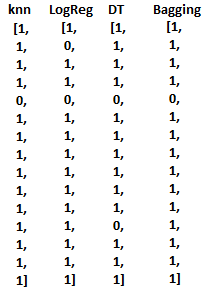
Pasting is an ensemble technique similar to bagging with the only difference being that there is no replacement done while sampling the training dataset. This causes less diversity in the sampled datasets and data ends up being correlated. That's why bagging is more preferred than pasting in real scenarios.

**Out-of-Bag Evaluation**

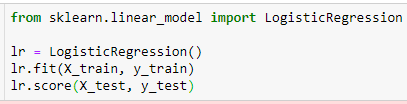
In bagging, when different samples are collected, no sample contains all the data but a fraction of the original dataset. There might be some data which are never sampled at all. The remaining data which are not sampled are called out of bag instances. Since the model never trains over these data, they can be used for evaluating the accuracy of the model by using these data for prediction. We do not need validation set or cross validation and can use out of bag instances for that purpose.



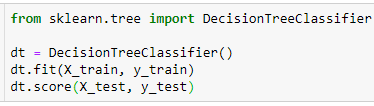




0.9239766081871345

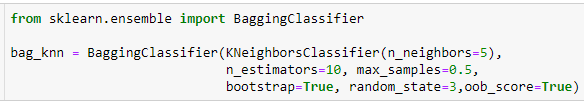


0.9298245614035088



0.9473684210526315

let's using bagging over our KNN classifier and see if our score improves:

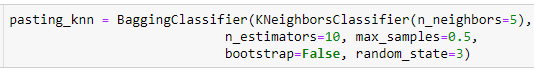




0.9415204678362573

**Note: our score significantly improves with use of bagging.**

let's not use bootstrap and see the model accuracy! Remember this is "Pasting"





0.9473684210526315

**Parameters used in the  algorithms:**

**base\_estimator:**

It defines the base estimator to fit on random subsets of the dataset.

When nothing is specified, the base estimator is a decision tree.

**n\_estimators:**

It is the number of base estimators to be created.

The number of estimators should be carefully tuned as a large number would take a very long time to run, while a very small number might not provide the best results.

**max\_samples:**

This parameter controls the size of the subsets.

It is the maximum number of samples to train each base estimator.

**max\_features:**

Controls the number of features to draw from the whole dataset.

It defines the maximum number of features required to train each base estimator.

**n\_jobs:**

The number of jobs to run in parallel.

Set this value equal to the cores in your system.

If -1, the number of jobs is set to the number of cores.

**random\_state:**

It specifies the method of random split. When random state value is same for two models, the random selection is same for both models.

This parameter is useful when you want to compare different models.

**Random Forest:**

Decision trees are one of such models which have low bias but high variance. We have studied that decision trees tend to over fit the data. So bagging technique becomes a very good solution for decreasing the variance in a decision tree. Instead of using a bagging model with underlying model as a decision tree, we can also use Random forest which is more convenient and well optimized for decision trees. The main issue with bagging is that there is not much independence among the sampled datasets i.e. there is correlation.

The advantage of random forests over bagging models is that the random forests makes a tweak in the working algorithm of bagging model to decrease the correlation in trees. The idea is to introduce more randomness while creating trees which will help in reducing correlation.

Let’s understand how algorithm works for a random forest model

1. Just like in bagging, different samples are collected from the training dataset using bootstrapping.

2. On each sample we train our tree model and we allow the trees to grow with high depths.

Now, the difference with in random forest is how the trees are formed. In bootstrapping we allow all the sample data to be used for splitting the nodes but not with random forests. When building a decision tree, each time a split is to happen, a random sample of ‘m’ predictors are chosen from the total ‘p’ predictors. Only those ‘m’ predictors are allowed to be used for the split.

**Why is that?**

Suppose in those ‘p’ predictors, 1 predictor is very strong. Now each sample this predictor will remain the strongest. So, whenever trees will be built for these sampled data, this predictor will be chosen by all the trees for splitting and thus will result in similar kind of tree formation for each bootstrap model. This introduces correlation in the dataset and averaging correlated dataset results do not lead low variance. That’s why in random forest the choice for selecting node for split is limited and it introduces randomness in the formation of the trees as well.

Most of the predictors are not allowed to be considered for split.

Generally, value of ‘m’ is taken as m ≈√p , where ‘p’ is the number of predictors in the sample.

When m=p , the random forest model becomes bagging model.

\*This method is also referred as “Feature Sampling”

3. Once the trees are formed, prediction is made by the random forest by aggregating the predictions of all the model. For regression model, the mean of all the predictions is the final prediction and for classification mode, the mode of all the predictions is considered the final predictions.

**Working of a Random Forest Model**

From the given dataset different samples are created by bootstrapping and these samples are used to train different decision trees. Once the training is complete, prediction is made using all the different models.

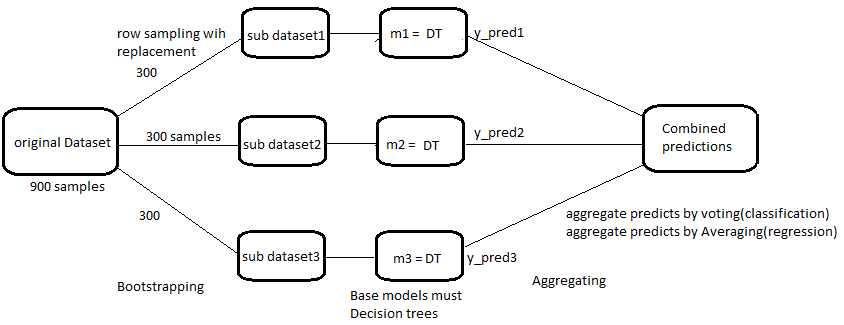
**Steps:**

Random subsets are created from the original dataset (bootstrapping).

At each node in the decision tree, only a random set of features are considered to decide the best split.

A decision tree model is fitted on each of the subsets.

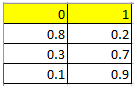
The final prediction is calculated by averaging the predictions from all decision trees.



**Predicting Outcome**

Random forest makes the prediction by taking the mode of all the predictions made by all the models, since this is the case of classification. This process is also known as “Majority voting”. We can also use prediction probability to make the final prediction. We can use the predict\_proba method, which will predict a probability from 0 to 1 that a given class is the right one for a row. For a problem with output being only 0 and 1, we'll get a matrix with as many rows as there is in the data and 2 columns. predict\_proba will return something like this:

Each row corresponds to a prediction. The first column is the probability that the prediction is a 0, the second column is the probability that the prediction is a 1. Each row adds up to 1.

If we just take the second column, we get the average value that the classifier would predict for that row. If there's a .9 probability that the correct classification is 1, we can use the .9 as the value the classifier is predicting. This will give us a continuous output in a single vector instead of just 0 or 1. We can then add all of the vectors we get through this method together and divide by the number of vectors to get the mean prediction by all the members of the ensemble. We can then round off to get 0 or 1 predictions. Similarly, in case of regression Random forest makes the prediction by taking the mean of all the predictions made by different models.

**Advantages and Disadvantages of Random Forest:**

1) It can be used for both regression and classification problems.

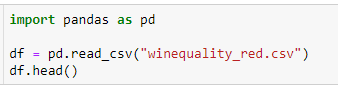
2) Since base model is a tree, handling of missing values is easy.

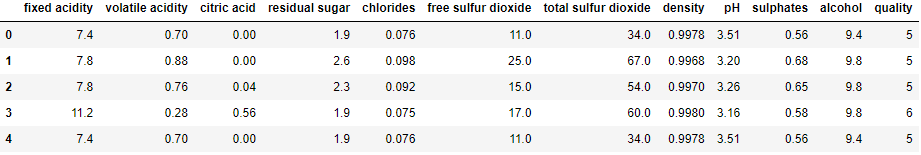
3) It gives very accurate result with very low variance.

4) Results of a random forest are very hard to interpret in comparison with decision trees.

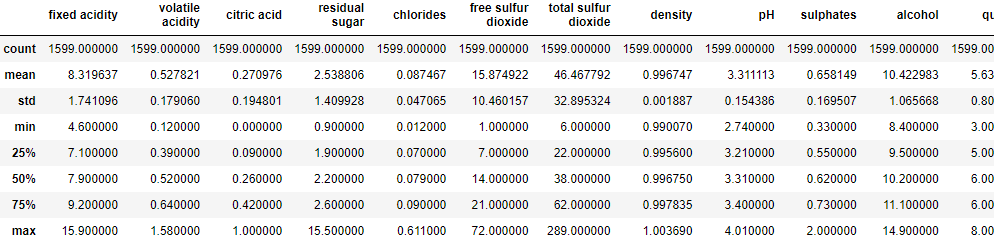
5) High computational time than other respective models.

Random Forest should be used where accuracy is up utmost priority and interpretability is not very important. Also, computational time is less expensive than the desired outcome.

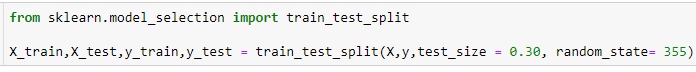


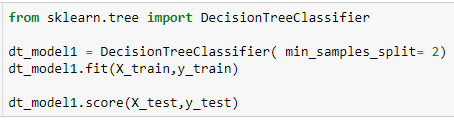




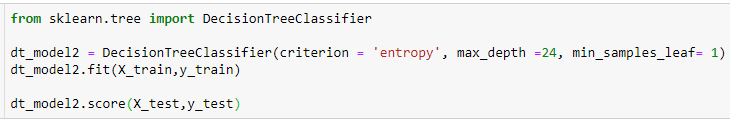






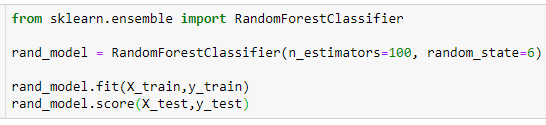


0.6753246753246753



0.658008658008658

Random state, if given none then score will vary every time you run the RandomForestClassifier. If we assign a value to it, then result will remain constant.



0.7445887445887446

We can see that two individual decision trees have both less score than a single random forest classifier.

So, using random forest classifier has increased the predictive power of our model.

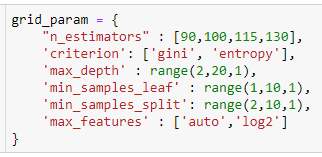
Great, let's do some hyper parameter tuning and see if we can increase our accuracy more.

Random forest hyper parameters are a combination of best hyper parameters of both decision tree and Bagging classifier.

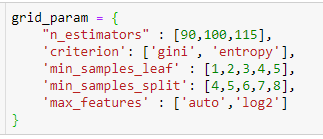
Let's now try to tune some hyper parameters using the GridSearchCV algorithm. We have studied about CrossValidation in upcoming lecture.

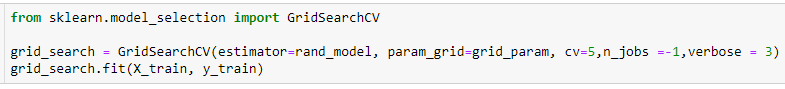
GridSearchCV is a method used to tune our hyper parameters. We can pass different values of hyper parameters as parameters for grid search. It does a exhaustive generation of combination of different parameters passed. Using cross validation score, Grid Search returns the combination of hyper parameters for which the model is performing the best.

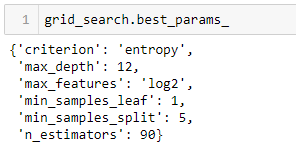
Note that it is common that a small subset of those parameters can have a large impact on the predictive or computation performance of the model while others can be left to their default values.



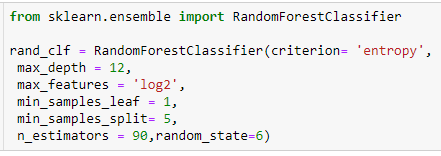
Let's do some more tweak in the hyper parameters and try gridSearch on it.







We will pass these parameters into our random forest classifier.





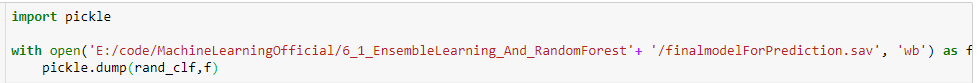


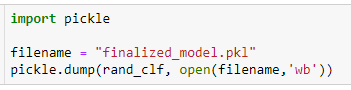


0.7316017316017316

Our accuracy has improved and score is better than the last grid search. So, we can say that giving all the hyperparameters in the gridSearch doesn't guarantee the best result. We have to do hit and trial with parameters to get the perfect score.

You are welcome to try tweaking the parameters more and try an improve the accuracy more.





### load the model



array([5], dtype=int64)

**Parameters**

**n\_estimators:**

It defines the number of decision trees to be created in a random forest.

Generally, a higher number makes the predictions stronger and more stable, but a very large number can result in higher training time.

**criterion:**

It defines the function that is to be used for splitting.

The function measures the quality of a split for each feature and chooses the best split.

**max\_features :**

It defines the maximum number of features allowed for the split in each decision tree.

Increasing max features usually improve performance but a very high number can decrease the diversity of each tree.

**max\_depth:**

Random forest has multiple decision trees. This parameter defines the maximum depth of the trees.

**min\_samples\_split:**

Used to define the minimum number of samples required in a leaf node before a split is attempted.

If the number of samples is less than the required number, the node is not split.

**min\_samples\_leaf:**

This defines the minimum number of samples required to be at a leaf node.

Smaller leaf size makes the model more prone to capturing noise in train data.

**max\_leaf\_nodes:**

This parameter specifies the maximum number of leaf nodes for each tree.

The tree stops splitting when the number of leaf nodes becomes equal to the max leaf node.

**n\_jobs:**

This indicates the number of jobs to run in parallel.

Set value to -1 if you want it to run on all cores in the system.

**random\_state:**

This parameter is used to define the random selection.

It is used for comparison between various models.

**Note**: The decision trees in random forest can be built on a subset of data and features. Particularly, the sklearn model of random forest uses all features for decision tree and a subset of features are randomly selected for splitting at each node.

To sum up, Random forest **r**andomly selects data points and features, and builds multiple trees (Forest) .

**Diff between GridSearchCV and RandomizedSearchCV**

in grid it consider all the ranges

it will go for sequential combination

in random it does not consider all the ranges

it will go for random combination