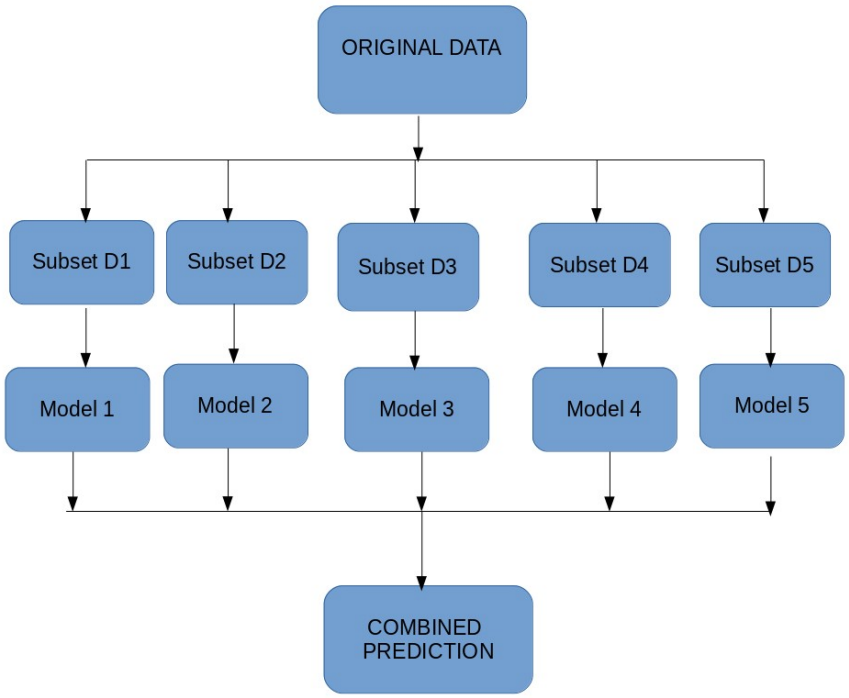
Bagging

The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result.
If we create all the models on the same set of data and combine it, it will not be useful as there is a high chance that these models will give the same result since they are getting the same input. One of the technique to overcome this problem is bootstrapping.

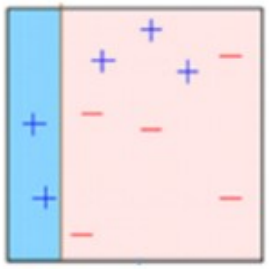
Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.
1.Multiple subsets are created from the original dataset, selecting observations with replacement.
2.A base model (weak model) is created on each of these subsets.
3.The models run in parallel and are independent of each other.
4.The final predictions are determined by combining the predictions from all the models.

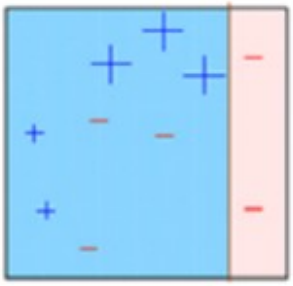


Boosting

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model. Let's understand the way boosting works in the below steps. 1.A subset is created from the original dataset. 2.Initially, all data points are given equal weights. 3.A base model is created on this subset. 4.This model is used to make predictions on the whole dataset.

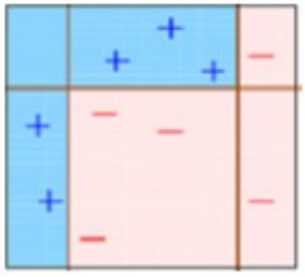


5.Errors are calculated using the actual values and predicted values. 6.The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights) 7.Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



8.Similarly, multiple models are created, each correcting the errors of the previous model. 9.The final model (strong learner) is the weighted mean of all the models (weak learners).

Thus, the boosting algorithm combines a number of weak learners to form a 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.



Algorithms based on Bagging and Boosting

Boosting algorithms:

•Ada Boost
•Gradient boosting
•XG boost

Ada Boost

Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

Below are the steps for performing the AdaBoost algorithm:
1.Initially, all observations in the dataset are given equal weights.
2.A model is built on a subset of data.
3.Using this model, predictions are made on the whole dataset.
4.Errors are calculated by comparing the predictions and actual values.
5.While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6.Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
7.This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

In [13]:
```
# Load libraries
import pandas as pd
import numpy as np

# load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None)

#split dataset in features and target variable
X = pima.iloc[:, 0:8].values
y = pima.iloc[:, 8].values

#Test-train split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [14]:
```
#Ada Boost
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
model.fit(X_train, y_train)
model.score(X_test,y_test)
```

Out[14]:
```
0.7489177489177489
```

Gradient Boosting (GBM)

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree.

We will use a simple example to understand the GBM algorithm. We have to predict the age of a group of people using the below data:

ID	Married	Gender	Current City	Monthly Income	Age (target)
1	Y	M	A	51,000	35
2	N	F	B	25,000	24
3	Y	M	A	74,000	38
4	N	F	A	29,000	30
5	N	F	B	37,000	33

1.The mean age is assumed to be the predicted value for all observations in the dataset.
2.The errors are calculated using this mean prediction and actual values of age.

ID	Married	Gender	Current City	Monthly Income	Age (target)	Mean Age (prediction 1)	Residual 1
1	Y	M	A	51,000	35	32	3
2	N	F	B	25,000	24	32	-8
3	Y	M	A	74,000	38	32	6
4	N	F	A	29,000	30	32	-2
5	N	F	B	37,000	33	32	1

3.A tree model is created using the errors calculated above as target variable. Our objective is to find the best split to minimize the error.
4.The predictions by this model are combined with the predictions 1.

ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)
1	35	32	3	3	35
2	24	32	-8	-5	27
3	38	32	6	3	35
4	30	32	-2	-5	27
5	33	32	1	3	35

5.This value calculated above is the new prediction.

In [15]:
```
#GBM
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model.fit(X_train, y_train)
model.score(X_test,y_test)
```

Out[15]:
```
0.7662337662337663
```

XGBoost

XGBoost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. XGBoost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. XGBoost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as **'regularized boosting'** technique.

Let us see how XGBoost is comparatively better than other techniques:
1.***Regularization***:
◦Standard GBM implementation has no regularisation like XGBoost.
◦Thus XGBoost also helps to reduce overfitting.

2.***Parallel Processing***:
◦XGBoost implements parallel processing and is faster than GBM .
◦XGBoost also supports implementation on Hadoop.

3.***High Flexibility***:
◦XGBoost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.

4.***Handling Missing Values***:
◦XGBoost has an in-built routine to handle missing values.

5.***Tree Pruning***:
◦XGBoost makes splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.

6.***Built-in Cross-Validation***:
◦XGBoost allows a user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

#XG Boost import xgboost as xgb model=xgb.XGBClassifier(random_state=1,learning_rate=0.01) model.fit(X_train, y_train) model.score(X_test,y_test)

Questionnaire