38.1 Ensembles

Ensemble - Definition

A model which uses multiple models to obtain a better predictive performance than the performance obtained from any one of the constituent models is called an Ensemble model.

There are 4 types of mostly used ensemble models. They are

a) Bagging
b) Boosting
c) Stacking
d) Cascading

Using these ensembles, we can build very powerful and high-performance models.

The key aspect of all the ensemble models is “The more different the constituent models are, the better we can combine them.”

38.2 Bootstrapped Aggregation (Bagging) Intuition

Let us assume we are given a dataset 'D' of 'n' data points. Let it be denoted as 'D_n'.

Let us pick 'k' samples of size 'm' each and let those samples be denoted as 'D_m^1', 'D_m^2', 'D_m^3',....,'D_m^k' respectively.

D_m^1→ Sample with 'm' data points used to build a model 'M_1'.

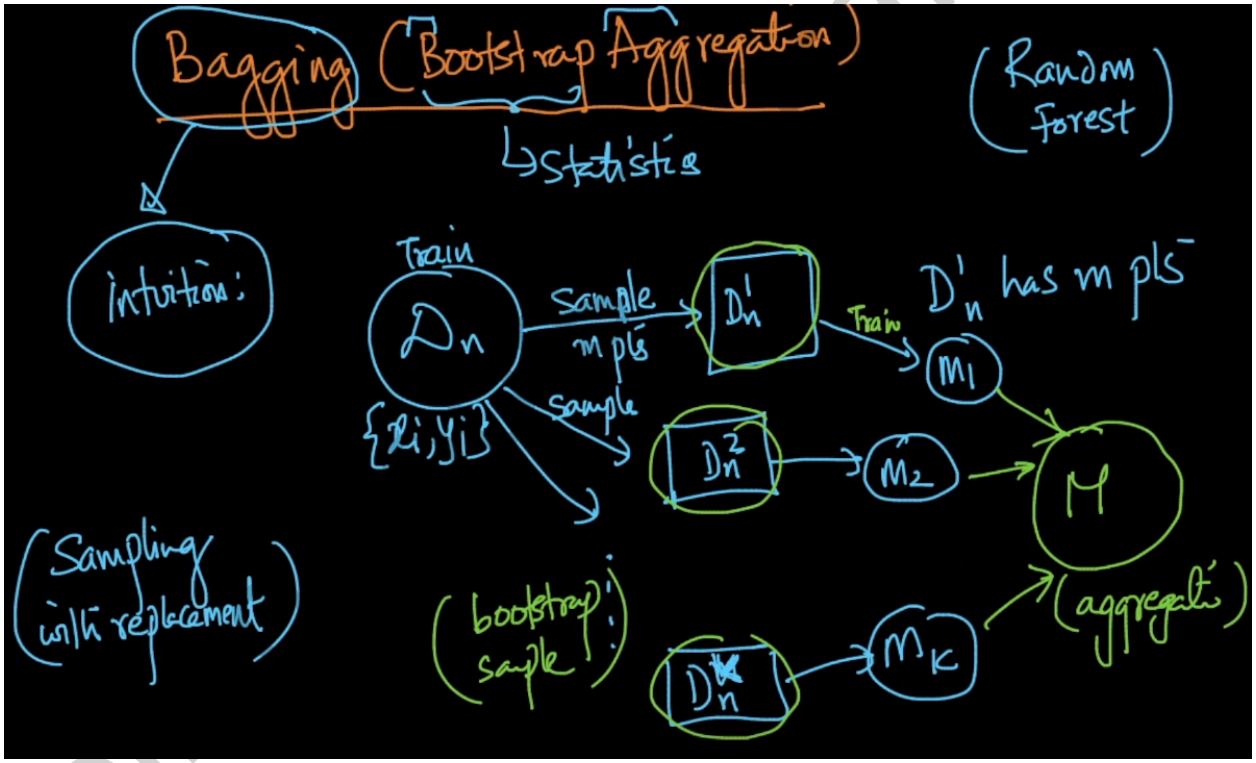
D_m^2→ Sample with 'm' data points used to build a model 'M_2'.

D_m^3→ Sample with 'm' data points used to build a model 'M_3'.

.

.

D_m^k→ Sample with 'm' data points used to build a model 'M_k'.



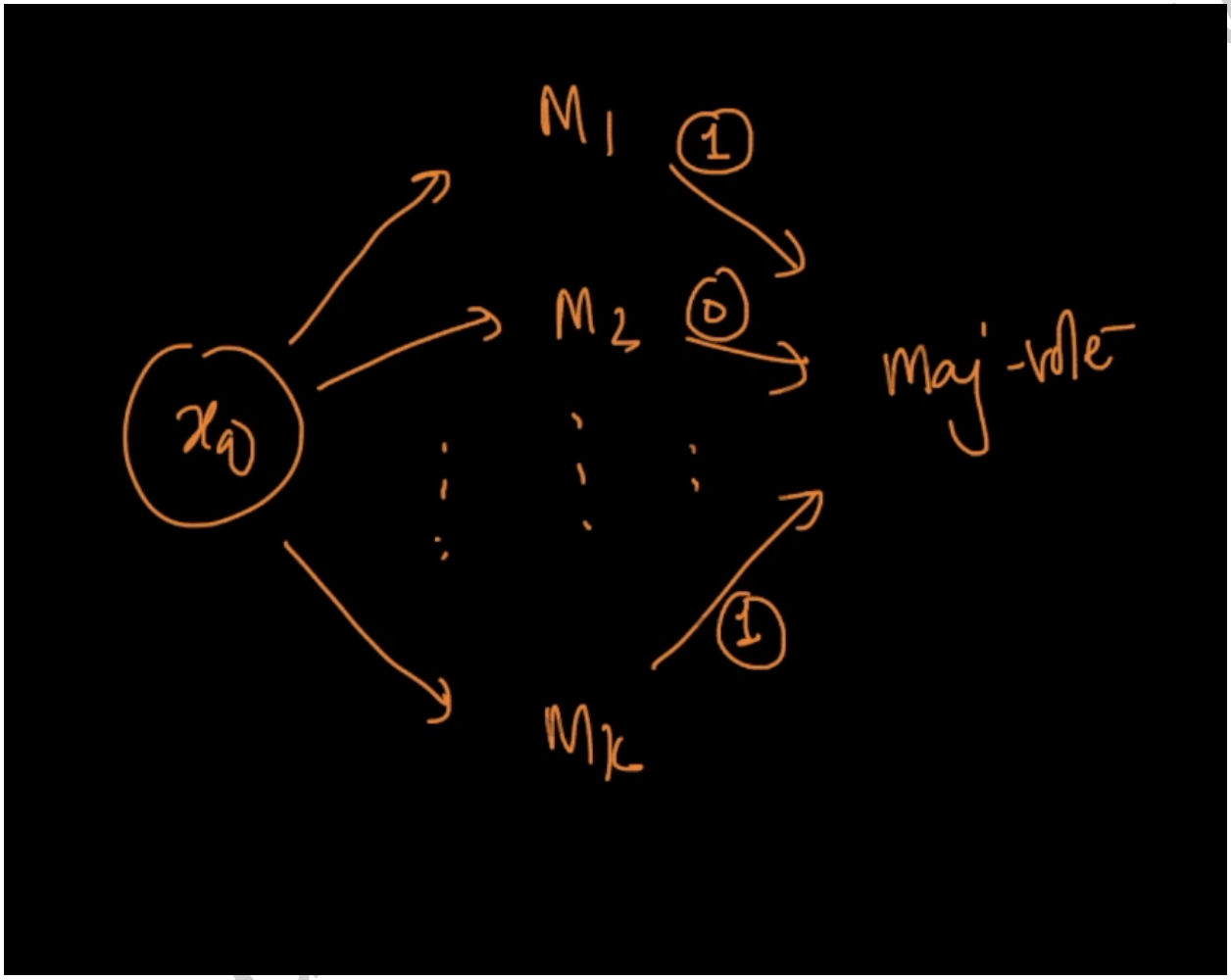
Here each model (ie., M_1, M_2, M_3, …., M_k) is built using random samples of size 'm' each(m<n). Here it is sampling with replacement.

For every model 'M_i' is built using 'D_m^i' of size 'm'. It means each model 'M_i' has seen a different subset of data.

All the samples created so far are called **Bootstrap samples**. Now all these models trained on the bootstrap samples must be combined into a larger model. This process is known as **Aggregation**.

A typical aggregation operation used is the **majority vote (in the case of classification)** and **computing the mean/median(in the case of regression)**.

In the case of classification, each bootstrap sample of data is used in training the model and the majority vote is taken.



So if a query point 'x_q' is given, it is passed through all the constituent models, and each model classifies this query point into one of the classes, and the majority vote will be taken among these predicted class labels and the result is assigned to this query point 'x_q'.

In the case of Regression, a query point 'x_q' is passed through all the constituent models, and each model computes the output for this point. The mean/median of all these outputs is computed, and is assigned as the output value to the point 'x_q'.

Here is a key point in Bootstrapped Aggregation. If there is a change in the train dataset 'D_n', then as the samples are picked randomly, the change in the samples will be less, thereby leading to the least amount of changes in the models trained by these bootstrap samples, and the final aggregation result is least affected.

This implies that Bagging can reduce variance in a model. Because of the way the model was designed, Bagging can reduce the variance in a model without impacting the bias. So the generalization error could be reduced as it is dependent on both the bias and the variance using the below equation

Generalization Error = Bias^2 + Variance + Irreducible Error

If each of these base models (ie., M_1, M_2, M_3, …., M_k) is a low bias, high variance model, then with the use of Bagging, we will continue to have low bias, but with a reduction in the variance. This reduction in the variance is because of the combination of **sampling** and **aggregation**.

Here even if there is a huge difference in 'D_n' as we are randomly splitting the 'D_n' into samples, only a few models get affected, thereby the variance gets affected.

So ultimately bagging says that if we take a bunch of high variance, low bias models and combine them using bagging (ie., bootstrap sampling + aggregation), then it gives a low bias, reduced variance model.

One of the examples with Low Bias, High Variance models is the Decision Tree with Large Depth. One of the very popular bagging techniques using the decision trees with large depth as the base models, and combining them is called **Random Forest**.

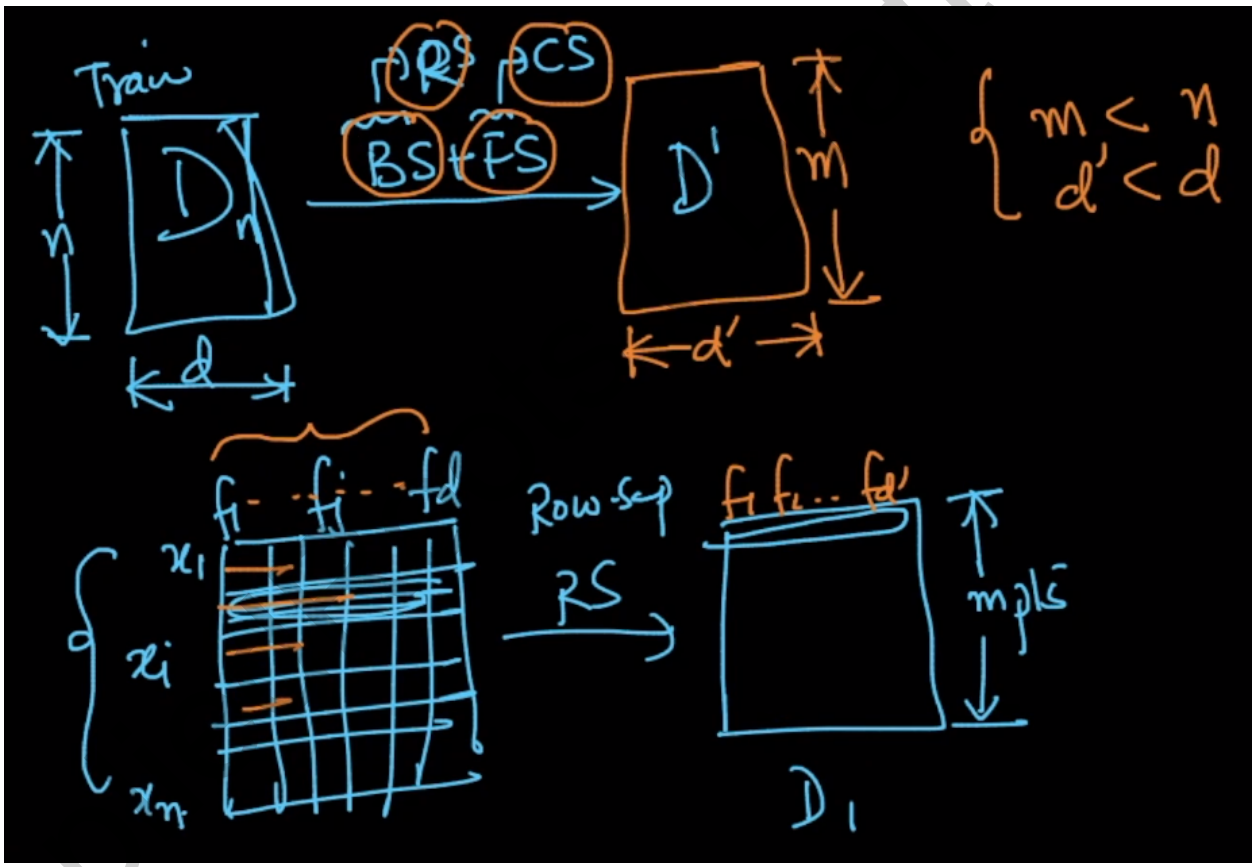
38.3 Random Forest and their construction

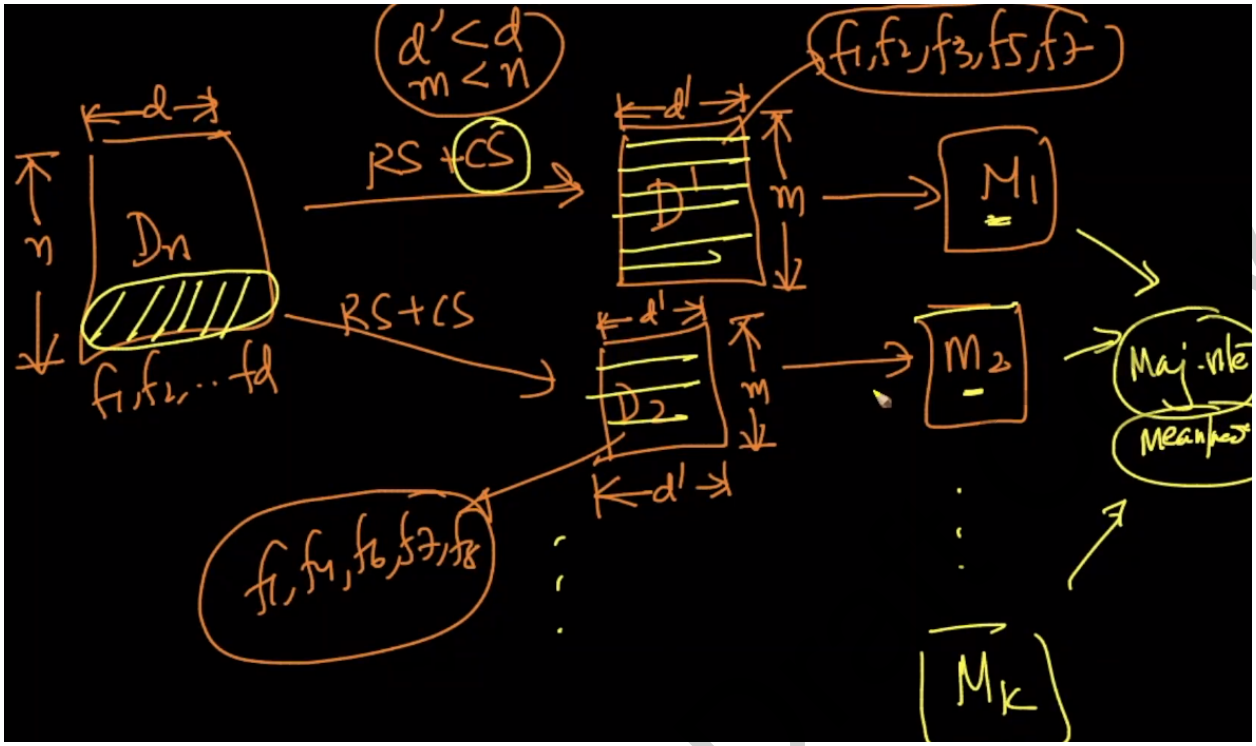
The word ‘Random’ in Random Forest means random sampling with replacement. The word ‘Forest’ means a group of trees. So

Random Forest = Decision Trees (as Base Learners) +
Bagging on top of the Base Learners +
Column Sampling (also called Feature Bagging)

Here Bootstrap sampling is also called Row Sampling. Feature Sampling is also called Column Sampling.

Let us assume we have a dataset ‘D_n’ with ‘n’ data points and ‘d’ dimensions.

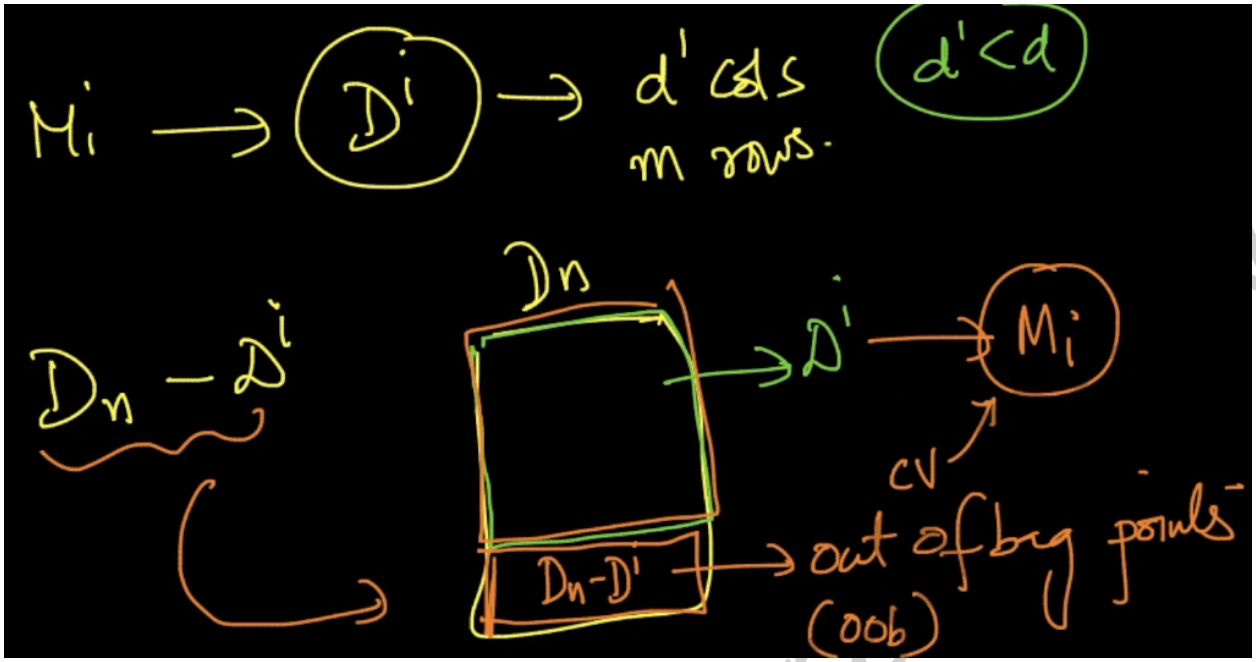




We have to pick a sample of 'm' points(m<n) separately and of 'd^{l}' dimensions(d^{l}<d), and use it in training the intermediate models.

In Random Forest, all the intermediate models M_i's are the decision trees of reasonable depth. The data points and the features in each intermediate model are not the same. As the datasets used for training these intermediate models are different, the models also will be different.

The Base Learners in Bagging should be the Low Bias, High Variance models. Hence we are choosing Decision Trees as the base learners in Random Forest. So in Random Forest, due to Bootstrap Sampling and Aggregation, the variance gets reduced. Each intermediate dataset contains different data points and also different features.



Out of the dataset 'D_n', when random sampling is used and '$D_m^{|}$' dataset is extracted, this intermediate dataset can be used to train the base learning model. And the remaining data points (ie., D_n-$D_m^{|}$) are used for cross-validation on the intermediate model and result in the out-of-bag error.

$D_m^{|}$ → In Bag Samples

D_n - $D_m^{|}$ → Out of Bag Samples

38.4 Bias-Variance Tradeoff

Random Forests tend to have Low Bias because their Base Learners are the models with Low Bias to start with. The bias of the overall model is the same as the bias of the individual base learners.

Model (M) = Aggregation(M_1, M_2, M_3, …., M_K)

If the number of Base learners increases, the variance decreases.

If the number of Base learners decreases, the variance increases.

So 'K' is a hyperparameter that has to be tuned carefully.

We have another 2 hyperparameters. They are Column Sampling Ratio and Row Sampling Ratio.

Column Sampling Ratio(CSR) = d^l/d

Row Sampling Ratio(RSR) = m/n

As the row sampling ratio and the column sampling ratio decrease, the in-bag data sets become more and more uncorrelated, and by using such datasets for training the base learners, the variance of the aggregate model will decrease.

So we have 3 hyperparameters here to tune(ie., 'K', 'RSR', and 'CSR'). But in general, 'CSR' and 'RSR' are kept constant, and only 'K' is used in cross-validation.

Q) How does the variance of the overall model decrease with an increase in 'K'?

Ans) When the number of base learners is small(say K=1), then we have one base model which is highly overfitted. So the overall model with K=1 is overfitted.

As 'K' increases, we have many models built on various slices of data each learning a different aspect of the data. So the overall model after the majority vote is less likely to overfit than a single model, as each model is slightly different from the others.

38.5 Bagging - Train and Run time complexity

In Random Forest with ‘K’ base learners

Train Time: O(n*log(n)*K*d)

But when we have multi-cores we can build each learner on each core. We call it **Trivially Parallelizable**.

When we have large amounts of data with a reasonable number of features, we can build trees faster. But if the dimensionality is high, then it becomes difficult.

Runtime: O(depth*K)

The depth in the base learners of the Random Forest model should be large(roughly 10 to 20).

Space Complexity: O(Decision Trees structure* K)

It means we are storing the entire decision tree structure of all the ‘K’ base learners.

Note

As the video lecture 38.6 is on the code discussion, we aren’t giving any notes on it. We suggest you go through the video lecture, and if you have any queries, please feel free to post them in the comments section.

38.7 - Extremely Randomized Trees

In Random Forests, we have seen aggregation and randomization in Row and Column Sampling. But in extremely randomized trees, we have one more level of randomization. That is randomization in selecting the numerical feature values as the threshold for performing a split, in the base learners.

In Decision Trees and Random Forests, in order to perform a split on the basis of the values of the numerical features, we first sort the values and consider each value as the threshold, and compute the Information gain. That particular threshold at which we get the maximum information gain is used in performing a split at that level in the decision tree.

In Extremely randomized trees, instead of considering all the values as the threshold, only a few values are picked randomly as thresholds, and information gain is computed for each threshold. Whichever threshold is responsible for the maximum information gain, would be chosen as the optimal value to perform a split.

Random Forest = Row Sampling + Column Sampling + Aggregation

Extremely Randomized Trees =
Row Sampling + Column Sampling + Aggregation + Randomization in selecting the thresholds from real values features.

Even in Random Forests, we are using Randomization in row and column samplings, as a way to reduce variance. In extremely randomized trees, we use randomization in row and column samplings and selecting a threshold value. Extremely Randomized trees tend to reduce variance better than random forests, but the bias might slightly increase. Extremely randomized trees are much more useful when we have real-valued features.

Q) How do extremely randomized trees reduce more variance when compared to random forests?

Ans) The base learners in Random Forests have higher chances of overfitting(ie., high variance) when compared to the base learners of extremely randomized trees, as the extremely randomized trees are using only a subset of possible values.

Both random forests and extreme trees use majority voting to reduce the overfitting, but the final model of extreme trees tends to be less overfit, as their base learners are relatively less overfitting when compared to the base learners of random forests.

Note

Extremely randomized trees can also handle categorical features as we can pick the random splits from a set of discrete values that the feature can take.

If a categorical feature has very few categories, we need not worry about the time complexity of evaluating all the possible values for a split. In such a case, extremely randomized trees work the same way as random forests.

Extremely randomized trees are very helpful when we have many real-valued features (or) categorical features with many categories.

38.8 Random Forest: Cases

Random Forest = Decision Trees (as base learners) + Row Sampling + Column Sampling + Aggregation

High Dimensionality

The decision trees cannot handle high dimensionality. Also, the decision trees are not recommended for use, when there are categorical features with a huge number of categories.

All the cases of the decision trees are applicable for random forests also, except a few. In all those cases where Decision Trees are not recommended for use, the Random Forests are also not recommended for use.

Bias-Variance Tradeoff

The Bias-Variance Tradeoff of Random Forest is not the same as that of the decision trees. In decision trees, the bias-variance tradeoff depends on the depth of the tree, whereas in random forests, the bias-variance tradeoff depends on the number of base learners.

Feature Importance

In Decision Trees, a feature 'f_i' is said to be important on the basis of the overall reduction in entropy (or) Gini impurity because of this feature at various levels in the decision tree.

In Random Forest, a feature 'f_i' is said to be important on the basis of the overall reduction in entropy (or) Gini impurity at various levels of each of the base learners. (ie., if a feature is important in most of the base learners, then it is important in the whole random forest model).

Except in these two cases, in the rest of the cases, both Random Forests and Decision Trees work the same.

38.9 - Boosting Intuition

In the case of Bagging, we have seen

Bagging = (High Variance, Low Bias Base models) + Randomization(in Column and row sampling and picking the threshold in ERT) + Aggregation

Here in the case of Boosting,

Boosting = (Low Variance, High Bias Base models) + Additive Combining (Reducing Bias while keeping the variance low)

Our main intention is to reduce the generalization error either using Bagging or Boosting. Same as Bagging, Boosting also allows us to build both Regression and Classification models.

Core idea of Boosting

Let us assume, we have the training data set $D_{Train} = \{x_i, y_i\}_{i=1}^{n}$.

Here the base learners are the High Bias and Low Variance models. One example of such a model is a Decision Tree with shallow depth. As these base learners have a high bias, the training error will be high.

Stage-0

Train the base learner 'M_0' with 'D_{Train}'. For every point 'x_i', compute $error_i = y_i - h_0(x_i)$

Stage-1

Now consider a base learner 'M_1' and the training data for it will be $\{x_i, error_i\}_{i=1}^{n}$. Let the output of the model 'M_1' be $h_1(x)$.

Let $F_1(x)$ be the model at the end of stage 1.

$\mathbf{F_1(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x)}$ → (weighted sum of 2 base models $h_1(x)$ and $h_0(x)$)

Stage-2

Here we have to train the model 'M_2' with $\{x_i, error_i\}_{i=1}^{n}$. Here

$\mathbf{error_i = y_i - F_1(x_i)}$

The model at the end of stage 2 is

$F_2(x) = \alpha_0 h_0(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x)$ → (weighted sum of 3 base models $h_2(x)$, $h_1(x)$ and $h_0(x)$)

Similarly, at the end of stage 'K', the final model is

$F_k(x) = \Sigma_{i=0}^{k} \alpha_i h_i(x)$

This model $F_k(x)$ is the additive weighted model.

$h_i(x)$ is trained to fit the residual error at the end of the previous stage.

So every base learner $h_i(x)$ is trained on $\{x_i, error_i\}_{i=1}^{n}$ to fit the residual error at the end of the stage (i-1).

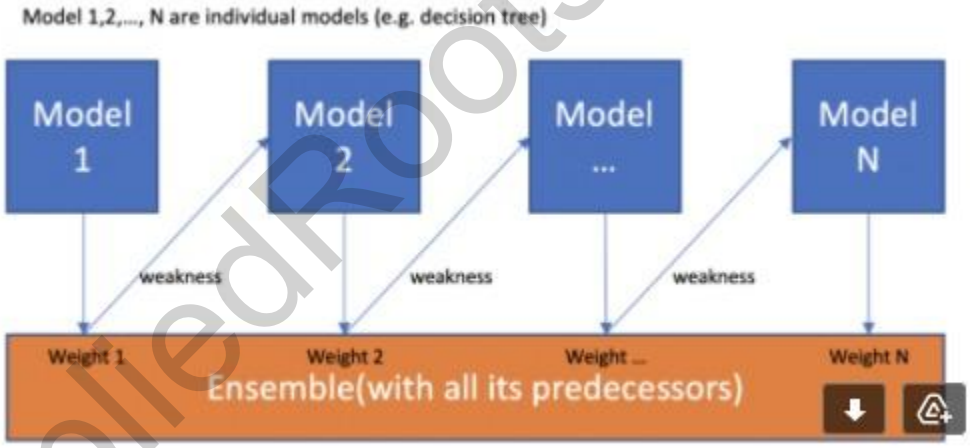
The final model $F_k(x)$ ends up having a low residual error. As the residual error reduces, the bias of the model also decreases. Hence the objective of Boosting is fulfilled.

The most commonly used Boosting algorithms are

a) Gradient Boosting Decision Tree (GBDT)
b) Ada Boosting (Also known as Adaptive Boosting).

Ada boost is mostly used when we are working with the images data.

Note



We cannot train all the base learners in Boosting in parallel, like in Bagging. We are trying to predict y_i's in the first model, and each subsequent model is trying to fix the errors by combining the previous models.

Boosting could easily overfit. Hence we use either of the two tricks to avoid overfitting.

a) Using High Bias base learners
b) Applying Regularization by shrinkage

Boosting could overfit if the number of base learners is high. We can avoid overfitting even when 'K'(number of base learners) is large by reducing α_i's. (where α_i's are the weights given to each model)

38.10 - Residuals, Loss Functions, and Gradients

In Boosting, we have seen the final model with K+1 base learners, and the final model at the end of stage 'K' is given as

$$\mathbf{F_K(x) = \Sigma_{i=0}^{K} \alpha_i h_i(x)}$$

The residual at the end of stage 'K' is given as

$$\mathbf{error_i = y_i - F_K(x)}$$

To solve any problem of Machine Learning, we have to minimize the loss function using an optimization problem. The loss function is dependent on the class labels y_i's and the final model we had at the end of stage 'K'. (ie., $F_K(x)$)

Let us assume we are working on a regression problem using Boosting. Then the loss function is given as

$$\mathbf{L(y_i, F_K(x_i)) = (y_i - F_K(x_i))^2}$$

If the loss function 'L' is of the form $(y_i - z_i)^2$, then

$$\mathbf{\partial L/\partial zi = \partial/\partial zi\ (y_i - z_i)^2 = 2*(y_i - z_i)*(-1) = -2*(y_i - z_i)} \text{ (where } z_i = F_K(x_i))$$

So now the value of $\mathbf{\partial L/\partial F_K(x_i)}$ is given as

$$\mathbf{\partial L/\partial F_K(x_i) = -2*(y_i - F_K(x_i))}$$

Let us send the negative sign to the left-hand side, then the equation becomes

$$\mathbf{-\partial L/\partial F_K(x_i) = 2*(y_i - F_K(x_i))} \rightarrow \mathbf{(1)}$$

$\partial L/\partial F_K(x_i) \rightarrow$ Gradient

$(y_i - F_K(x_i)) \rightarrow$ Residual/Error

So from equation (1), if we exclude the number '2', we can say that Negative Gradient is nearly equal to the residual.

(ie., $\mathbf{-\partial L/\partial F_K(x_i) \approx (y_i - F_K(x_i))}$). Hence the negative gradient is also called **pseudo-residual**.

So when we train a base learner at the end of the stage 'i', then

$error_i = (y_i - F_{i-1}(x))$

So in the optimization, instead of using the residual, we can use the negative gradient(which is the pseudo residual). The main reason for using pseudo-residual, in place of residual is, if we use residual we can minimize any loss function, as long as it is differentiable. Because of this, the Boosting algorithms are so powerful.

Models like Random Forests could not minimize all types of losses(because it can only minimize the entropy loss), whereas Boosting models like Gradient Boosting, can minimize any loss, just because we are using the pseudo residuals, in place of normal residuals.

38.11 - Gradient Boosting

Let the training data be $\{(x_i, y_i)\}_{i=1}^n$.

'M' → number of iterations (ie., the number of base learners).

Let L(y,F(x)) → Differential loss function

Steps of Algorithm

1) Initialize the model with a constant value.

   $\mathbf{F_0(x) = arg\text{-}min_\gamma \Sigma_{i=1}^n L(y_i, \gamma)}$

   Here we have to find a constant 'γ' that could minimize $\mathbf{\Sigma_{i=1}^n L(y_i, \gamma)}$. Here '$\gamma$' that minimizes $\mathbf{F_0(x)}$ is the mean of y_i's.(In case of regression)

2) For m = 1 to M,

   a) Compute the pseudo residuals,

   $$\mathbf{r_{im} = -[\partial L(y_i, F(x_i))/\partial F(x_i)]}_{F(x) = Fm\text{-}1(x)} \text{ (for i = 1,2,3,...,n)}$$

   $\mathbf{r_{im}}$ → Pseudo residual for the m^{th} stage of the i^{th} point.

   b) Fit a base learner $h_m(x)$ to pseudo residuals (ie., train it using the training set $\mathbf{\{(x_i, r_{im})\}_{i=1}^n}$

   c) Compute multiplier 'γ_m' by solving the following one dimensional optimization problem.

   $$\mathbf{\gamma_{im} = arg\text{-}min_\gamma \Sigma_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))}$$

   d) Update the model

   $$\mathbf{F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)}$$

3) Output $F_M(x)$

Note

One of the great properties of Gradient Boosting is that we can pick any differentiable loss function(L) and any base model $h_i(x)$ and still be able to optimize the loss function using Gradient Boosting. To achieve this,we need to introduce the loss function in the construction of 'γ_{im}'.

The beauty of Gradient boosting is that the base model could be any model(Decision Trees, Logistic Regression or SVM) and is independent of the loss function. Shallow Decision Trees with high bias are a very popular option making them GBDTs. The loss function is independent of the base model we use, and it should be differentiable.

38.12 - Regularization by Shrinkage

The formulation of the final model in Gradient Boosting is given as

$\mathbf{F_M(x) = h_0(x) + \Sigma_{m=1}^{M} \gamma_m h_m(x)}$ (where M → Number of base models)

As the number of base models increase, variance tends to increase, bias tends to decrease and leads to overfitting.

Shrinkage

Earlier we have seen the model at the end of stage 'M' is given as

$\mathbf{F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)}$

In the case of Shrinkage, the above formula is slightly modified. A new parameter '**v**' gets multiplied to '$\mathbf{\gamma_m h_m(x)}$' in the above equation and the new equation becomes

$\mathbf{F_m(x) = F_{m-1}(x) + v\gamma_m h_m(x)}$ (where 0<v<=1)

In shrinkage, we have the learning rate which is denoted as 'v' as a hyperparameter.

If 'v' increases, then the model overfits. It is better to use Grid Search Cross-Validation and find the optimal values of 'M' and 'v' for building GBDT.

Here shrinkage(v) is a hyperparameter that helps us avoid overfitting. It is one of the hyperparameters that we can tune to avoid overfitting of Gradient Boosting models, which are prone to overfitting. It reduces the output generated by each of the base models through multiplication as 0<v<=1, thereby avoiding the base models to overfit on the train data.

Note

'γ' is the weight we give to the M^{th} model and this allows us to give weight to each base model based on how useful that base model is in reducing the overall loss. This can be thought of as weights similar to those weights we give to each feature in the case of Logistic Regression. The weights here are given to the base models, but not to the individual features. The weights (γ_m) are determined by solving an optimization problem at each iteration of Gradient Boosting.

38.13 - Train and Run time complexity

Train Time:

For one Decision Tree: O(n*log(n)*d)

For 'M' base learners: O(n*log(n)*d*M)

GBDT is not easily parallelizable like Random Forest. GBDT takes more time to train, than Random Forest, though the time complexity is the same.

Runtime:

For one Decision Tree: O(depth)

For 'M' base learners(for $h_m(x)$): O(depth*M + m)

Space Complexity:

O(storing each tree + γ_m)

Note

As the video lecture 38.14 is on the documentations of Gradient Boosting Classifier and XGBoost models, we are not providing any notes for it. Please go through the video lecture, and if you have any queries, please feel free to post them in the comments section under the video.

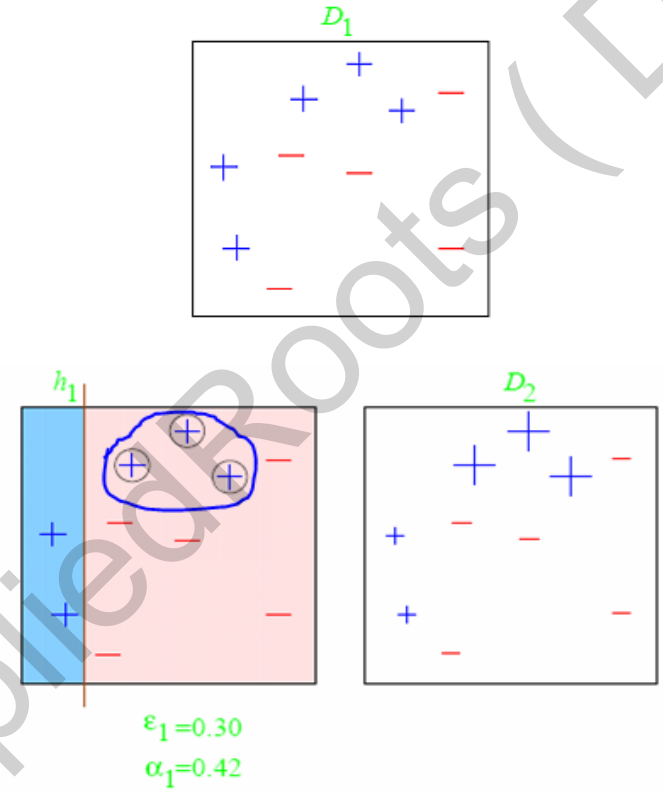
38.15 - AdaBoost: Geometric Intuition

AdaBoost is one of the Boosting algorithms. It is similar to Gradient Boosting, but there are a few differences between them. AdaBoost is typically mostly used in image processing applications and computer vision based tasks.

Please go through the link below for the example that was discussed here.

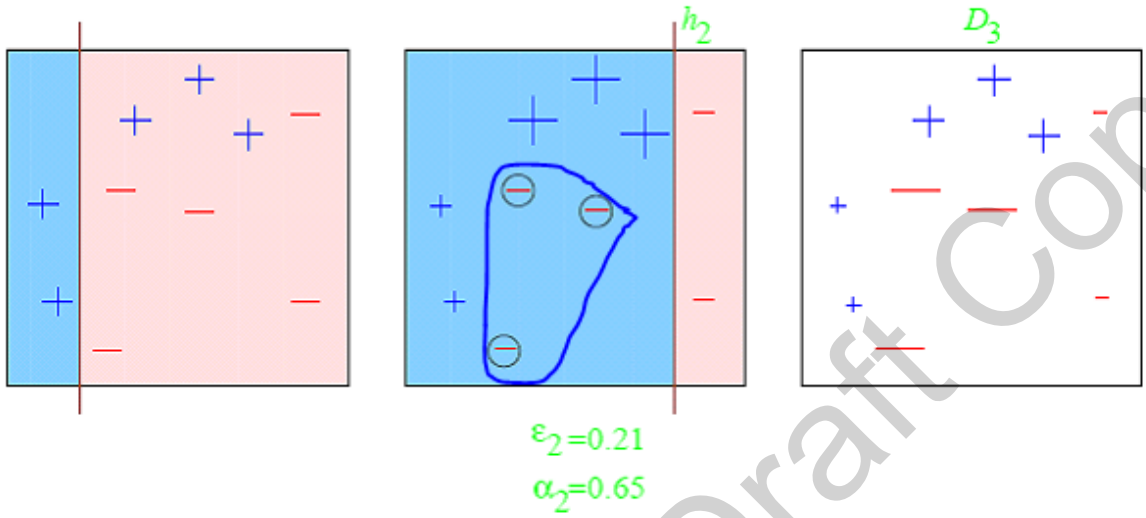
https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=lectures.boosting

In Adaboost, same as the other Boosting models, the base learners are of high bias and low variance. So if we assume the decision trees with a depth of 1 as the base learners, then the decision surface would be a straight line parallel to the 'Y' axis. So when the input data(let's say D_1) is given to the base learner, we do get a few misclassifications.



When we look at the classification made in 'h_1', 3 positive points are misclassified. We shall now either perform upsampling of the misclassified points (or) assign more weightage to the misclassified points and then train

the next base learner model using this updated/modified dataset. This time, it gives another decision surface that is a little better when compared to the decision surface obtained in the previous base learner.



In the predictions obtained from 'h_2', we could see all the positive points are classified correctly, but there are 3 negative points that are misclassified. So now we have to add more weightage to these 3 negative points and again pass this modified data as input to the next base learner. This way it continues until the 'K' stages and will get the best model at the end (with minimal amount of misclassifications).

This model is called AdaBoost because, at every stage we are adapting to the errors that were obtained in the previous stage, and are giving more importance to the points that are misclassified.

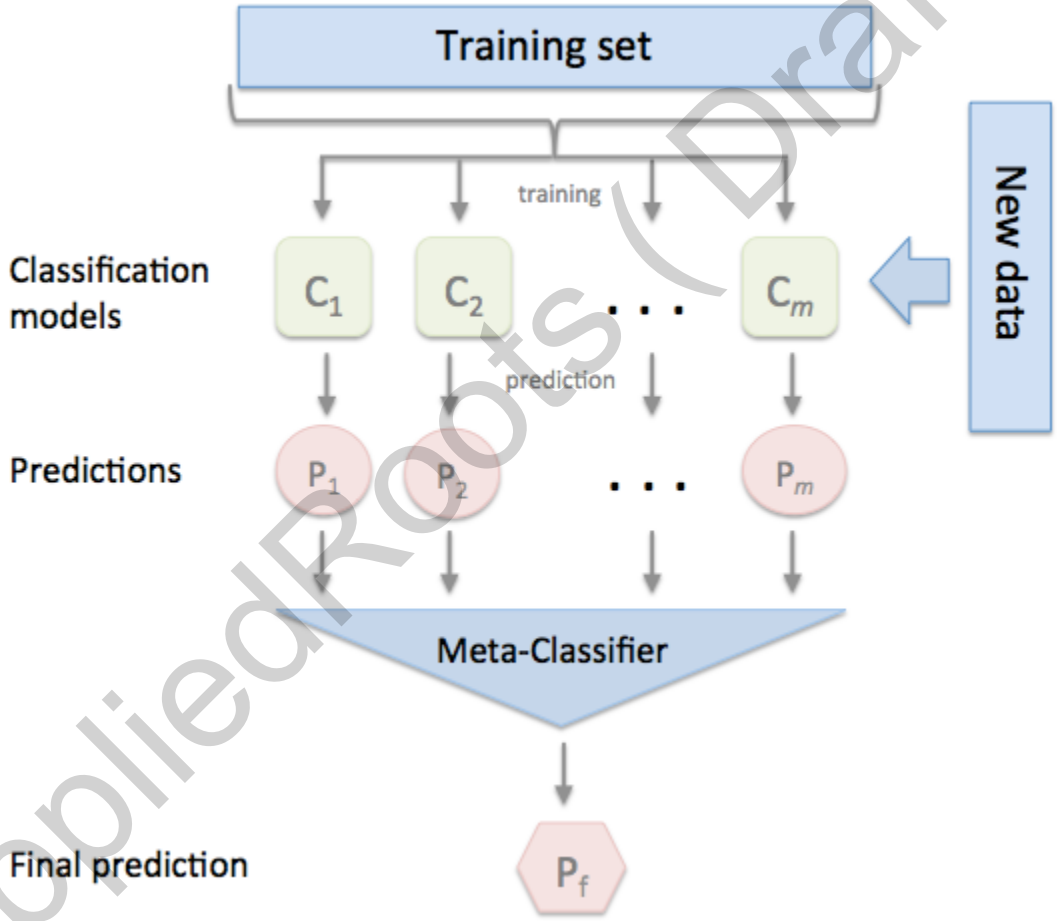
The core idea behind Adaboost is to adapt to the changes in the previous stage and give more weightage to the misclassifications in the next stage.

38.16 - Stacking Models

Stacking is an ensemble model that combines multiple classification models via meta-classifier. In Boosting we have seen that each base learner was built by taking the output of the base learner in the previous stage, into consideration. But here the individual classifiers are trained parallely and are independent of each other. No model has nothing to do with the output of the other model.

Please go through the below webpage for an example of Stacking Classifier.

http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/



In this web page, we can see there are ‘m’ different classification models. The Stacking Classifier is almost similar to the Bagging model, but with a few changes.

In Bagging models, all the base learners are of high variance and low bias. But here in Stacking, we generally go with the best fit models as the individual classifiers(ie., no high variance and no high bias).

Let us assume there are ‘n’ data points in our dataset and there are ‘m’ individual classifiers used in stacking, then if we denote our dataset as $D_n = \{x_i, y_i\}_{i=1}^{n}$, then

We have to first pass the training data as input to each of these ‘m’ individual classifiers. For each point (x_i, y_i), let the output obtained from each of these individual classifiers be denoted as $h_{1i}(x)$, $h_{2i}(x)$, $h_{3i}(x)$,..., $h_{mi}(x)$. So now, when a point (x_i, y_i) passes through ‘m’ individual classifiers, it gets ‘m’ outputs. Similarly all the ‘n’ data points are passed through the ‘m’ classifiers, and each point gets ‘m’ outputs. So the total number of outputs will be **n*m**.

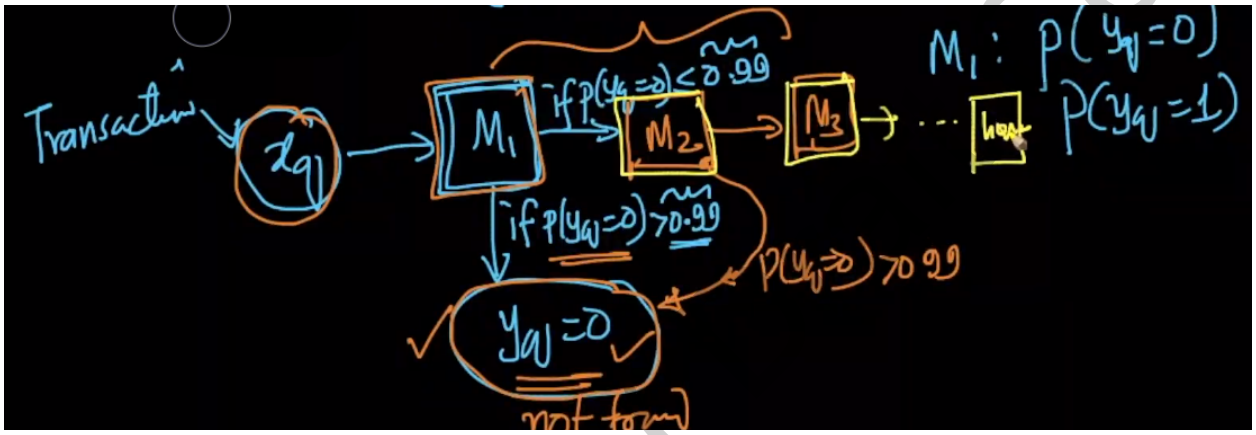
Now the meta-classifier mentioned in this example could be again any one classification model(like Logistic Regression, SVM, DT, etc). So now the same data points should be passed to the meta-classifier, but not directly. The outputs obtained from the ‘m’ classifiers for each data point are concatenated into an m-dimensional vector form. So in this way, we should concatenate the ‘m’ outputs of all the ‘n’ data points, and get the result in the form of an nXm matrix.

This nXm matrix along with the class labels y_i’s will be passed as an input to the meta-classifier and the output of the meta-classifier will be the final predicted class label for a given query point.

38.17 - Cascading Classifiers

So far we have learned about Bagging, Boosting and Stacking ensembles. Let us now learn about another ensemble technique called Cascading.

Let us consider the problem of predicting whether a given credit card transaction is fraudulent. Let us denote the class labels of the fraudulent cases as 1, and those of the non-fraudulent cases as 0.



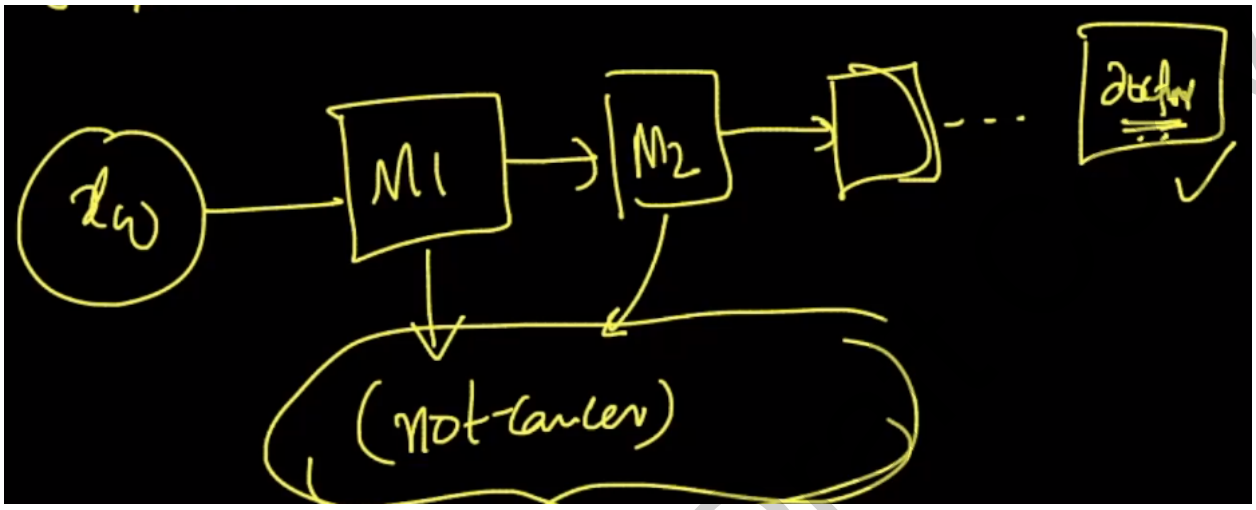
For any query point 'x_q', our model 'M_1' predicts the value $P(y_q=1)$ and $P(y_q=0)$. As this is a highly critical study and we don't want to take a risk, let us assume, we have set the threshold of $P(y_q=0)>0.99$. So for a given transaction, only if the value of $P(y_q=0)>0.99$, then only we confirm it as a non-fraudulent case, otherwise, we confirm it as a fraudulent case.

But in case, if we do not just want to make a decision on the basis of the prediction of one model, if we want to make judgement only after multiple models give the same result, then in such cases, let us assume we shall add the models 'M_2' ad 'M_3'.

So if 'M_1' yields $P(y_q=0)>0.99$, then we confirm it as a non-fraudulent case. But if $P(y_q=0)<=0.99$, then instead of directly confirming it as a fraudulent case, we can again pass the query point through multiple models. If all those models give $P(y_q=0)>0.99$, then we can confirm it as a non-fraudulent case. Otherwise, we confirm it as a fraudulent case.

Cascading models are typically used when the cost of making a mistake is high. Even if a small mistake of 0.1% could severely impact the business, in such cases, we employ the ensemble models.

One more case study where we can employ the cascading models is in the problem of predicting whether a given patient has cancer or not, on the basis of the data from his/her medical tests. Here even a small mistake of 0.1% could severely affect the life of the patient.



Working of a Cascading Model

Whenever we pass the whole training dataset 'D_{Train}' as an input to the model 'M_1', the model predicts $P(y_q=0)>0.99$ for a few points and let all these points combinedly be denoted as D^{I}.

So now when we have to pass the data as an input to the model 'M_2', then instead of passing the whole 'D_{Train}', we have to exclude the data points of D^{I} and pass the remaining data points. (ie., $D_{M1} = D_{Train}-D^{I}$)

So again the model 'M_2' predicts $P(y_q=0)>0.99$ for a few points and let all these points combinedly be denoted as D^{II}. So now we pass the data points $D_{M2}(D_{M1}-D^{II})$ as input to the model 'M_2' and so on. All those points which got the predictions as P(yq=0)>0.99 are excluded(because we already have made a clear decision on them as non-fraudulent cases) and the remaining points are passed as input to the next models in the sequence.

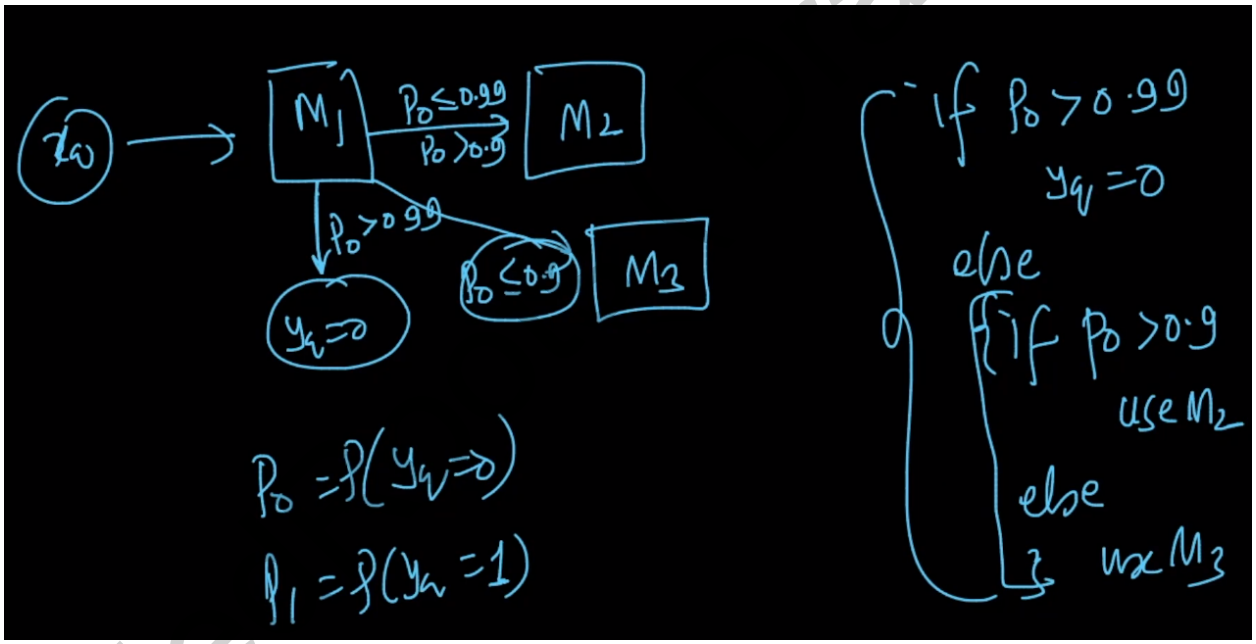
Note

We pass only the data points x_i's as input to each of the individual models in the cascading models. In the beginning, if each point in 'D_{Train}' has 'd' dimensions (ie., $x_i \in R^d$), then there would be no changes in the

dimensions of the data for the other models in the sequence. **We do not carry the output of previous models to the next models as input**. Please do not get confused here. If our initial dataset has 1000 dimensions, then appending the output(ie., predicted y_i's) of 'M_1' to the 1000-dimensional data, and sending the new 1001-dimensional data as input to 'M_2' is not correct. We do not carry forward the outputs of previous models, as the inputs to the next models in cascading classifiers.

Note

Also, it is not mandatory to have only one model at each stage in cascading. We can have multiple models at one stage, depending on a few if-else conditions we can choose which points to be sent to which model as inputs. Below is an example of it.



In this example, out of all the points which got the predictions $P(y_q=0)<=0.99$ by the model 'M_1', we are again dividing them into two groups(one with $P(y_q=0.9)<=0.9$ and the other with $P(y_q=0)>0.9$). Those points with $P(y_q=0)<=0.9$ are sent as input to the model 'M_3' and those points with $P(y_q=0)>0.9$ are sent as input to the model 'M_2'.