Introduction to Ensemble Learning

07 October 2022 10:03

Ensemble:

- Ensemble = "Group"

Ensemble Method:

- Which involves the Group of Predictive models
- To achieve a better accuracy and model stability.
- Ensemble methods are known to impart supreme boost to tree based models.

Why?

- Like every other model, a tree based algorithm also suffers from the plague of **bias** and **variance**.
- Prone to **overfitting**.

How?

- Normally, as we increase the complexity of your model, in this case decision tree, you will see a reduction in training error due to lower bias in the model.
- As we continue to make your model more complex, we end up over-fitting your model and your model will start suffering from high variance.

What should be done?

- A champion model should maintain a balance between these two types of errors. This is known as the **trade-off management of bias-variance errors**.

Solution:

- Ensemble learning is one way to tackle bias-variance trade-off.

There are **various ways to ensemble** *weak learners* to come up with *strong learners*:

1. Bagging (Parallel ensemble learning)
2. Boosting (Sequential ensemble learning)
3. Stacking

Note:

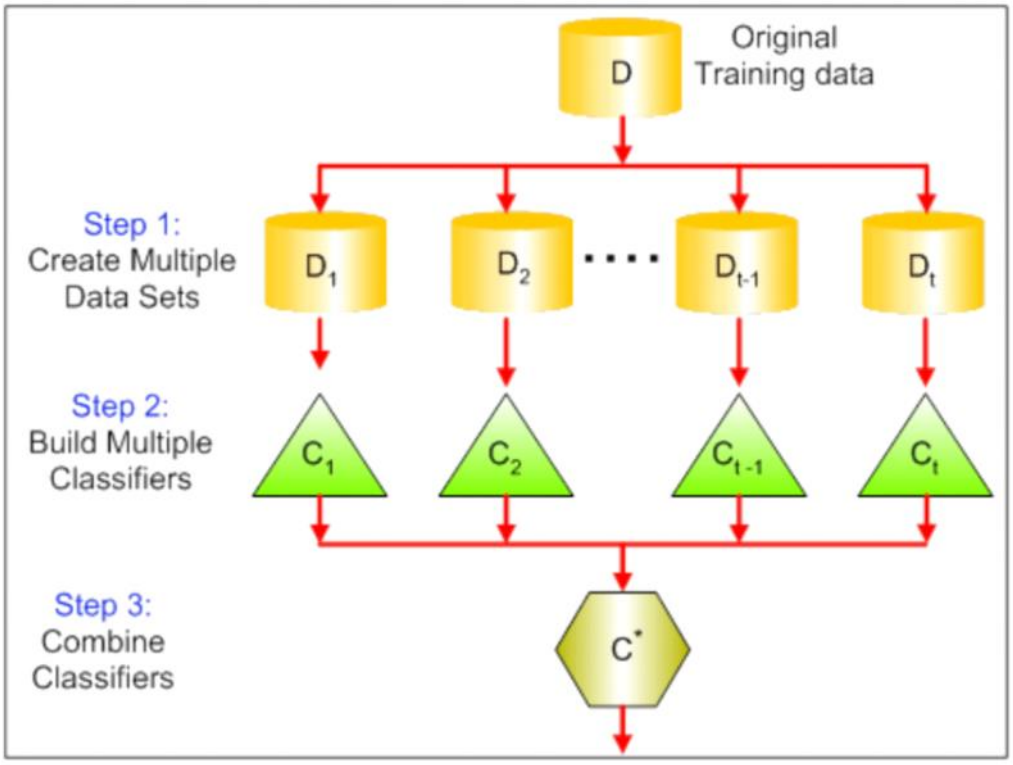
- Generally, If variance is high, prefer Bagging
- If bias and accuracy is high, prefer Sequetial.

1. Bagging

10 October 2022 15:30

Bagging is an ensemble technique used **to reduce the variance of our predictions** by combining the result of multiple classifiers modeled on different sub-samples of the same data set.

The following figure will make it clearer:



The steps followed in bagging are:

1. **Create Multiple DataSets:**
   - Sampling is done with replacement on the original data and new datasets are formed.
2. **Build Multiple Classifiers:**
   - Classifiers are built on each data set.
   - Generally the same classifier is modeled on each data set and predictions are made.
3. **Combine Classifiers:**
   - The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.
   - The combined values are generally more robust than a single model.

Note:

- Here the number of models built is not a hyper-parameters.

- Higher number of models are always better or may give similar performance than lower numbers.

Important:

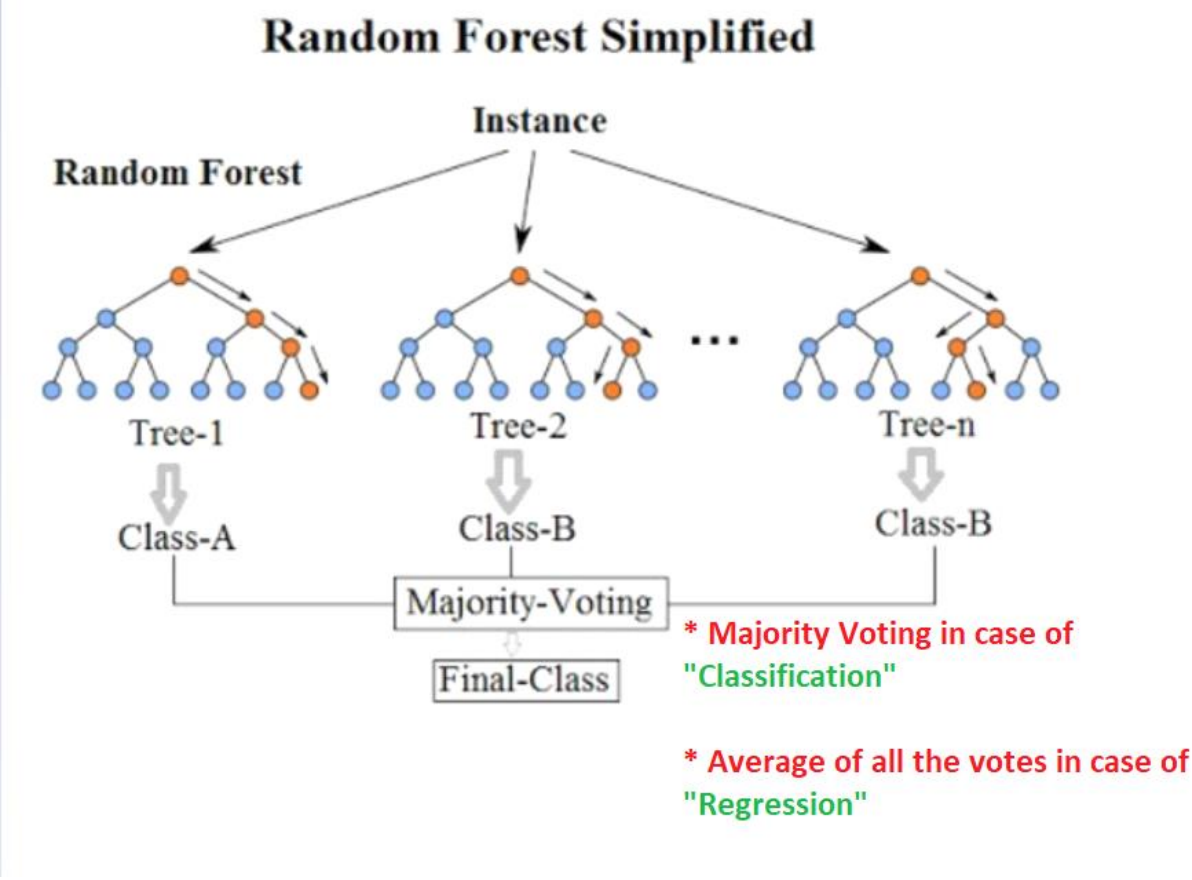
- It can be theoretically shown that the variance of the combined predictions are reduced to 1/n (n: number of classifiers) of the original variance, under some assumptions. (Think Central Limit Theorem)

There are various implementations of bagging models. Random forest is one of them and we'll discuss it next.

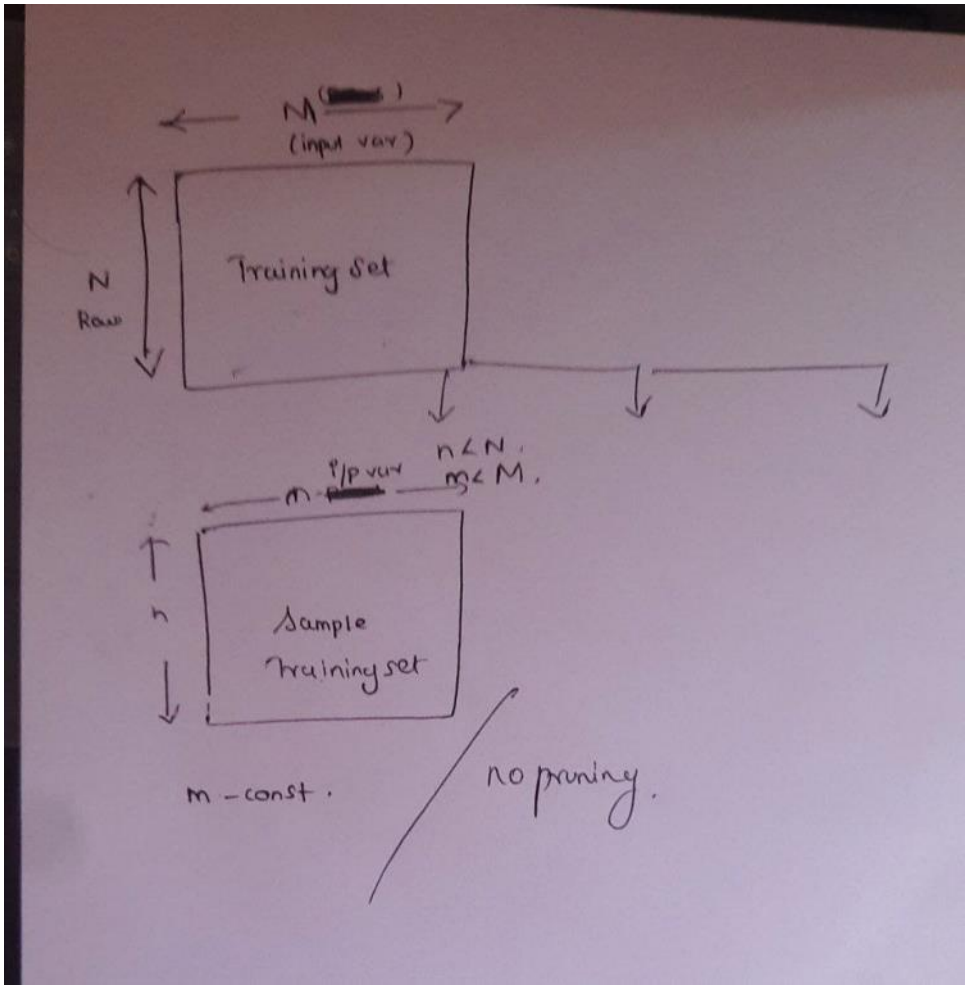
Random Forest

10 October 2022 15:30

- In Random Forest, we grow **multiple tree**s as ~~opposed to~~ a **single tree in CART model** (Classification and regression Tree)**.**
- We construct trees from the subsets of the original dataset.
- These **subsets** can have a **fraction of the columns as well as rows**.
- **To classify a new object based on attributes,** each tree gives a classification and we say that the tree "**votes**" **for that class**.
- The forest chooses the **classification having the most votes** (over all the trees in the forest) and in case of **regression, it takes the average** of outputs by different trees.



Working of Random Forest:



For Rows,

- Assume number of rows in the training set is N.
- Then, a sample of n < N rows is taken at random but *with replacement.*
- This sample will be the training set for growing the tree.

For Columns (or Input variable)

- If there are M input variables, a number m < M is specified such that at each node, m variables are selected at random out of the M.
- The best split on these m is used to split the node.
- The value of m is held constant while we grow the forest.

Importantly,

- Each tree is grown to the largest extent possible and there is **no pruning**.
- Predict new data by aggregating the predictions of the n tree trees (i.e., majority votes for classification, average for regression).

Advantages

- This algorithm can solve both type of problems i.e. **classification and regression** and does a decent estimation at both fronts.
- RF has the power of handling **large datasets with higher dimensionality**. It can handle thousands of input variables and identify most significant variables so it is considered as **one of the dimensionality reduction methods.**
- Further, the model outputs Importance of variable, which can be a very handy feature (on some random data set).

- It has an effective method for **estimating missing data and maintains accuracy** when a large proportion of the **data is missing**.
- It has methods for **balancing errors** in data sets **where classes are imbalanced.**
- The capabilities of the above can be **extended to unlabeled data**, leading to **unsupervised clustering, data views and outlier detection.**
- Random Forest involves *sampling of the input data with replacement* called as **bootstrap sampling.**
    - Here one third (say) of the data is **not used for training** and can be **used to testing.** These are called **the out of bag samples**.
    - Error estimated on these out of bag samples is known as **out of bag error**.
    - **Out-of-bag estimate** is as accurate as using a test set of the same size as the training set.
    - Therefore, using the out-of-bag error estimate **removes the need for a set aside test set.**

Disadvantages

- It surely does a good job at classification **but not as good as for regression problem** as it does not give precise continuous nature predictions.
- In case of regression, it **doesn't predict beyond the range in the training data**, and that they may **over-fit data** sets that are particularly noisy.
- Random Forest can feel **like a black box approach** for statistical modelers – you have very **little control** on what the model does. You can at best – try different parameters and random seeds!

2. Boosting

10 October 2022 15:30

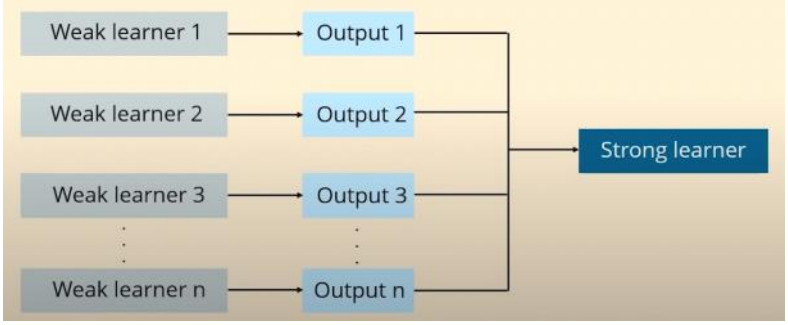
Before the define the boosting, let know some concept about weak learner and strong learner.

Weak learner/ Strong Learner:

- To predict the output, various rules have been provided
- E.g. To predict it as cat or dog we provide the set of rules such as,
  - The image has pointy ears - Cat
  - The image has cat shaped eyes - Cat
  - The image has bigger limbs - Dogs
- Now, the each of this individual rule that help to predict the output
- But still sometime we may get wrong prediction individually

Hence,

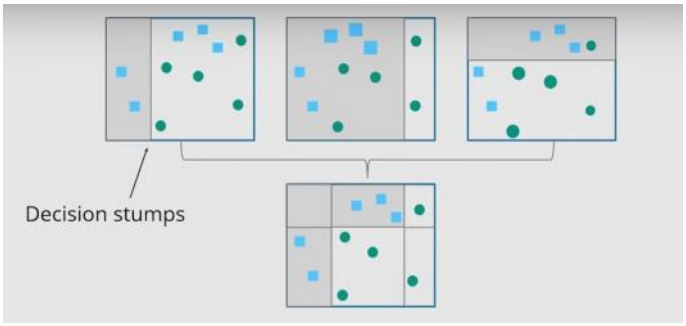
- This individual rules that helps to predict called as **"Weak Learner"**
- The aggregation of all prediction of the weak learners are called as **"Strong Learner"**



Boosting

Boosting is the set of machine learning algorithm that used to combine the weak learners to form strong learners in order to increase the accuracy of the model.

Working of Boosting:



1. The base algorithm reads the data and assign equal weight to each sample observation. (as seen in 1st diagram)
2. False Prediction are assigned to the next base learner with a higher weightage on these incorrect prediction. (as seen in 2nd diagram)
3. Step 2 is repeated until the algorithm can correctly classify the output. (as seen in 4th diagram)

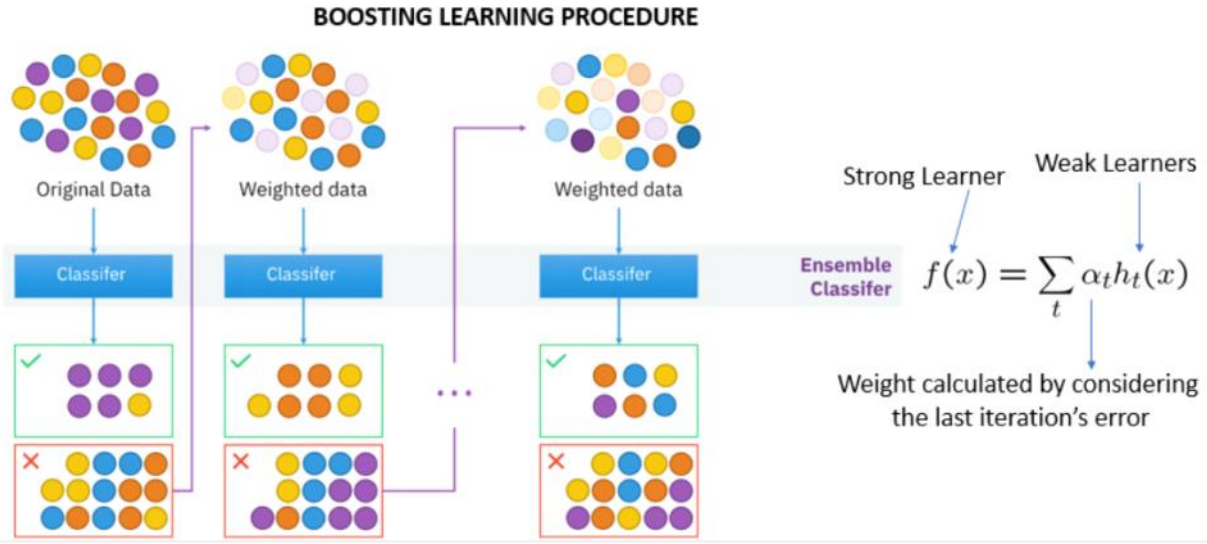
Types of Boosting:

1. **Gradient Boosting Machine**
2. **XGBoost**
3. **AdaBoost**

4. **LightGBM**
5. **CatBoost**

Definition from Colab:

Boosting fit a sequence of weak learners − models that are only slightly better than random guessing, such as small decision trees − to weighted versions of the data. More weight is given to examples that were misclassified by earlier rounds.

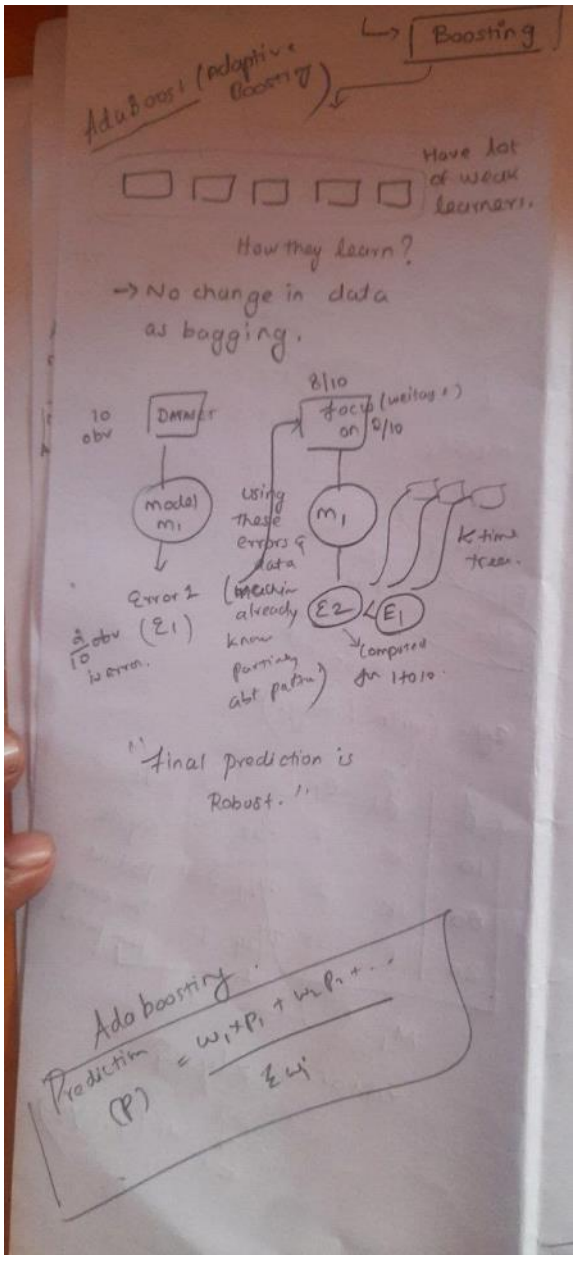


AdaBoost

12 October 2022 15:22

- AdaBoost or Adaptive Boosting
- Having a lot of weak learners
- How they will learn?
- As the data has been changed for modelling in Bagging. Boosting doesn't follow it.

Example:



Step 01:

- Let say we have 10 observations in our dataset
- We try to model(m1) using an algorithm.
- We will get the error e1 in the 1st iteration.
    - Assume that we get 2/10 observation as error e1.

Step 02:

- Using the error e1 and the dataset, machine will learn the particular pattern of the dataset from the 1st iteration itself.
- Now the same process is repeated, but machine will focus on the error e1 by applying some weightage to 2/10.
- Data set is again processed with the model m1 and we get error e2
    - It is important to note that, error e2 always less than that of error e1 **(e2<e1)**.

Step 03:

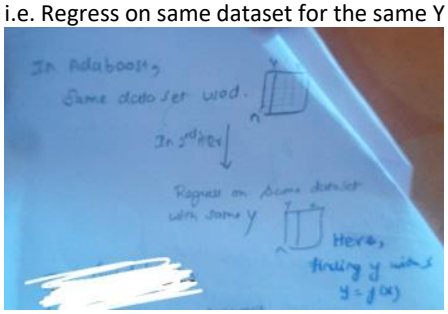
- Same step 02 is repeated for k times till getting the final prediction **(P)**
- **Final prediction (P) = (w1p1+w2p2+….wipk) / sum of wi**

Gradient Boosting Machine

10 October 2022 15:31

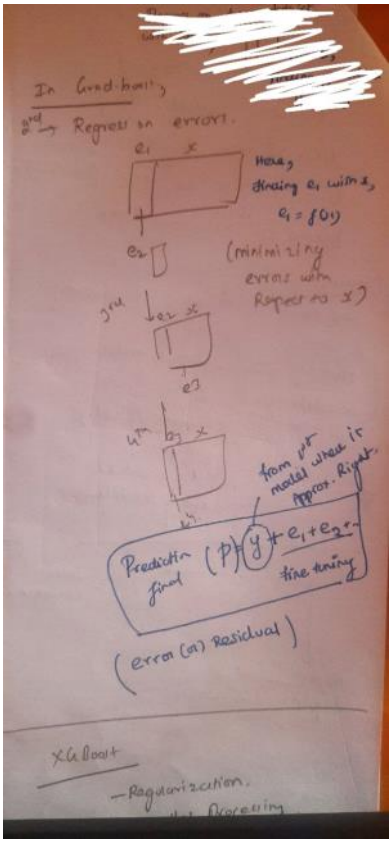
In AdaBoost,

- Same dataset has been used in all iteration.
- i.e. Regress on same dataset for the same Y



While in Gradient Boosting,

- From the 2nd iteration, Machine will regress on same dataset by adding error e1 as target variable.
- i.e. Regress on dataset for the error
- By which machine tend to minimizing the errors with respect to X.
- Final Prediction will be (P) = (Y + e1 + e2 + ….ek)
- Note: Error or residual



Definition:

> "Gradient Boosting Machine (GBM) builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function."

How does it work?

- Let's take a dataset $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\}$
- Choose a loss function, let's say MSE.
- Fit a naive model on the dataset, a simple tree or just take $\bar{y}$, call this model $F_0(x)$

First interation

- Get residuals of all predictions, $r_{i1}(x) = y_i - F_0(x_i)$
- Fit a model (can be regression tree) on residuals $\{(x_1, r_{11}), (x_2, r_{21}), (x_3, r_{31}), \ldots, (x_{n1}, r_{n1})\}$, call this model $h_1(x)$
- New predictor is $F_1(x) = F_0(x) + \gamma_1 h_1(x)$. Find γ_1 which minimizes MSE.

Second iteration

- Get residuals of all predictions, $r_{i2}(x) = y_i - F_1(x)$
- Fit a model (can be regression tree) on residuals $\{(x_1, r_{12}), (x_2, r_{22}), (x_3, r_{32}), \ldots, (x_{n2}, r_{n2})\}$, call this model $h_2(x)$
- New predictor is $F_2(x) = F_1(x) + \gamma_2 h_2(x)$. Find γ_2 which minimizes MSE.

and so on...

and so on...

- Get residuals of all predictions, $r_{im}(x) = y_i - F_{m-1}(x)$
- Fit a model (can be regression tree) on residuals $\{(x_1, r_{1m}), (x_2, r_{2m}), (x_3, r_{3m}), \ldots, (x_{nm}, r_{nm})\}$, call this model $h_m(x)$
- **Final predictor** is $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$. Find γ_m which minimizes MSE.

XG Boost

10 October 2022 15:31

Extreme Gradient Boosting (XGBoost) is just an extension of gradient boosting with the following added advantages:

1. **Regularization:**
    - Standard GBM implementation has no regularization like XGBoost, therefore it also **helps to reduce overfitting**. In fact, XGBoost is also known as ‘regularized boosting‘ technique.
2. **Parallel Processing:**
    - XGBoost implements parallel processing and is blazingly faster as compared to GBM.
    - But hang on, we know that **boosting is sequential process** so how can it be parallelized? We know that each tree can be built only after the previous one, but to make a tree it uses all the cores of the system. XGBoost also **supports implementation on Hadoop**.
3. **High Flexibility:**
    - XGBoost allow users to define custom optimization objectives and evaluation criteria. This adds a whole new dimension to the model and there is no limit to what we can do.
4. **Handling Missing Values:**
    - XGBoost has an in-built routine to handle missing values.
    - User is required to supply a different value than other observations and pass that as a parameter.
    - XGBoost tries different things as it **encounters a missing value on each node and learns which path to take for missing values in future.**
5. **Tree Pruning:**
    - **A GBM would stop splitting a node when it encounters a negative loss in the spli**t.
    - Thus it is more of a greedy algorithm.
    - XGBoost on the other hand make **splits upto the max_depth** specified and then start **pruning** the tree backwards and remove splits beyond **which there is no positive gain**.
    - Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.
6. **Built-in Cross-Validation:**

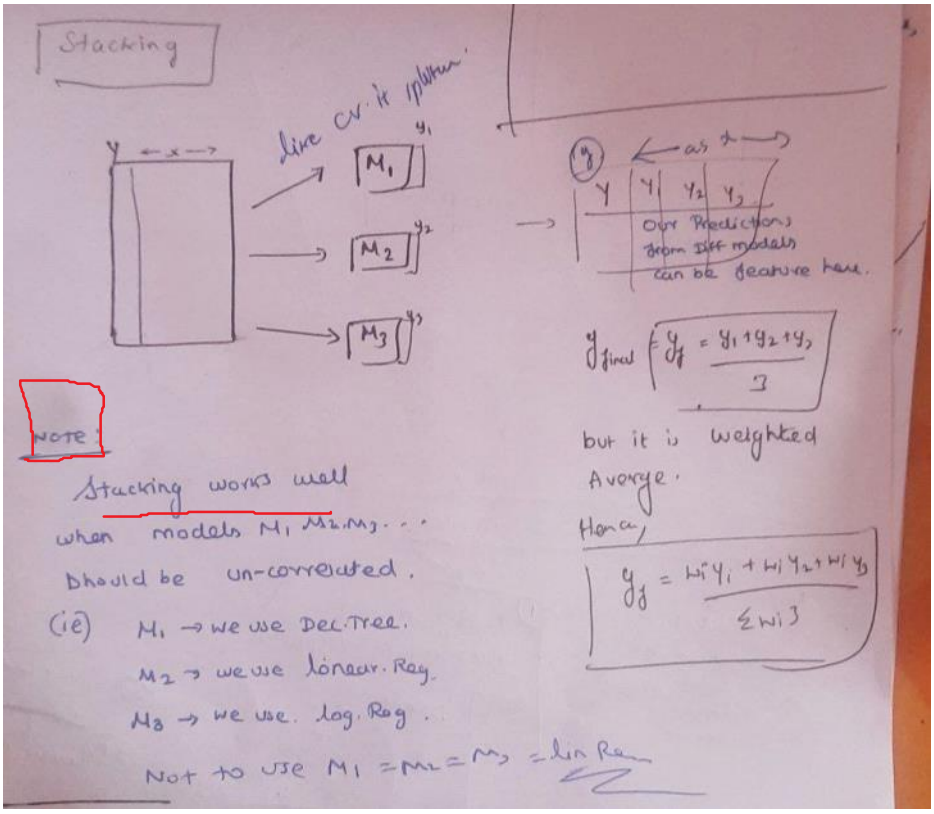
- XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
- This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

7. **Continue on Existing Model:**
    - User can start training an XGBoost model from its last iteration of previous run.
    - This can be of significant advantage in certain specific applications. GBM implementation of sklearn also has this feature so they are even on this point.

3. Stacking

10 October 2022 15:31

- The dataset can be classify into different model classifier as M1, M2, M3 up to Mk.
- For each model classifier, we will get different Target variable or predictions as Y1, Y2, Y3.
- In final step,
    - All the predictions from the weak learner Y1, Y2, Y3 will be created as a single dataset with Y from the source dataset
    - Now, our prediction from the different models Y1, Y2, Y3 can be feature here.
    - Y final will be weighted average of the all, i.e.,
    - **Y final = (w1Y1+w2Y2+….+wiYk) / sum of wi**



Definition:

Stacking or Stacked Generalization is an ensemble technique.

- It uses a **meta-learning algorithm** to learn how to best combine the predictions from two or more base machine learning algorithms.
- The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a **classification or regression task** and make predictions that have better performance than any single model in the ensemble.
- Given multiple machine learning models that are skilful on a problem, but in different ways, how do you choose which model to use (trust)?
- The approach to this question is to use another machine learning model that learns when to use or trust each model in the ensemble.

- **Unlike bagging, in stacking, the models are typically different** (e.g. not all decision trees) and fit on the same dataset (e.g. instead of samples of the training dataset).
- Unlike boosting, in stacking, **a single model** is used to learn how to best combine the predictions from the contributing models (e.g. instead of a sequence of models that correct the predictions of prior models).
- The architecture of a stacking model involves two or more base models, often referred to as level-0 models, and a meta-model that combines the predictions of the base models, referred to as a level-1 model.

Level-0 Models (Base-Models):

- Models fit on the training data and whose predictions are compiled.

Level-1 Model (Meta-Model):

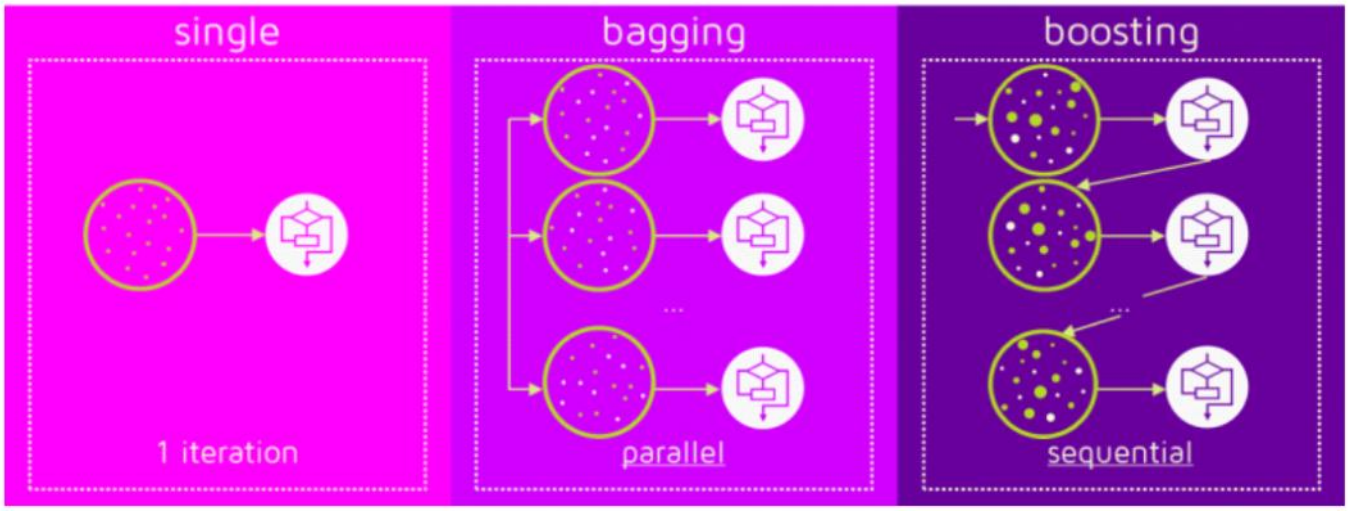
- Model that learns how to best combine the predictions of the base models.
- The meta-model is trained on the predictions made by base models on out-of-sample data.
- That is, data not used to train the base models is fed to the base models, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the meta-model.

The **outputs** from the base models used as input to the meta-model may be

1. real value in the **case of regression**, and
2. probability values, probability like values, or class labels in the **case of classification.**

Bagging vs. Boosting

10 October 2022 15:32



Implementation

10 October 2022 15:32

1. Importing Libraries

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt

     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.model_selection import GridSearchCV
     from sklearn.metrics import roc_auc_score, confusion_matrix, accuracy_score

     from pylab import rcParams
     rcParams['figure.figsize'] = 8, 8
```

2. Importing Dataset

3. Data Understanding

```
data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.114697	0.796303	-0.149553	-0.823011	0.878763	-0.553152	0.939259	-0.108502	0.1
1	-0.039318	0.495784	-0.810884	0.546693	1.986257	4.386342	-1.344891	-1.743736	-0.5
2	2.275706	-1.531508	-1.021969	-1.602152	-1.220329	-0.462376	-1.196485	-0.147058	-0.9
3	1.940137	-0.357671	-1.210551	0.382523	0.050823	-0.171322	-0.109124	-0.002115	0.8
4	1.081395	-0.502615	1.075887	-0.543359	-1.472946	-1.065484	-0.443231	-0.143374	1.6

V22	V23	V24	V25	V26	V27	V28	V29	Target
-0.807853	-0.055940	-1.025281	-0.369557	0.204653	0.242724	0.085713	0.89	0
-0.072200	-0.197573	1.014807	1.011293	-0.167684	0.113136	0.256836	85.00	0
-0.103533	0.150945	-0.811083	-0.197913	-0.128446	0.014197	-0.051289	42.70	0
0.650355	0.034206	0.739535	0.223605	-0.195509	-0.012791	-0.056841	29.99	0
0.821209	-0.137223	0.986259	0.563228	-0.574206	0.089673	0.052036	68.00	0

```
[ ] # Checking the distribution of two classes in the target variable
    data.Target.value_counts()
```

```
0    56864
1       98
Name: Target, dtype: int64
```

Clearly the dataset is heavily imbalanced!

```
[ ] # Checking the shape of our data
    data.shape
```

```
(56962, 30)
```

4. Creating Variables X and Y

```
[ ] # Creating the dataset with all independent variables
    X = data.iloc[:,:-1]

    # Creating the dataset with the dependent variable
    Y = data.iloc[:,-1]
```

- Split the dataset into train and test using stratified sampling on our dependent variable.
- Using a stratified sampling ensures the distribution of dependent variable remains same across train and test datasets

5. Creating Train and test split dataset

```
[ ] # Splitting the dataset into the Training set and Test set
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0, stratify = Y)
```

to split the dataset into train and test sets in a way that preserves the same proportions of examples in each class as observed in the original dataset.

Note: For Imbalanced Dataset

- We use "Stratify"
- Also for evaluating the metrics, we prefer the AUC than using Accuracy

```
[ ] print("The shape of train dataset :")
    print(X_train.shape)

    print("\n The shape of test dataset :")
    print(X_test.shape)
```

```
The shape of train dataset :
(45569, 29)

 The shape of test dataset :
(11393, 29)
```

```
print("Distribution of classes of dependent variable in train :")
print(Y_train.value_counts())

print("\n Distribution of classes of dependent variable in test :")
print(Y_test.value_counts())
```

```
Distribution of classes of dependent variable in train :
0    45491
1       78
Name: Target, dtype: int64

 Distribution of classes of dependent variable in test :
0    11373
1       20
Name: Target, dtype: int64
```

6. Hyper Parameter tuning

```
# Hyperparameter tuning

classifier = RandomForestClassifier() # For GBM, use GradientBoostingClassifier()
grid_values = {'n_estimators':[50, 80,  100], 'max_depth':[3, 5, 7]}
classifier = GridSearchCV(classifier, param_grid = grid_values, scoring = 'roc_auc', cv=5)

# Fit the object to train dataset
classifier.fit(X_train, Y_train)
```

Same procedure for GBM

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 5, 7], 'n_estimators': [50, 80, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

7. Predicting both train and test dataset using the fitting model

```
[ ] train_preds =  classifier.predict(X_train)
    test_preds  = classifier.predict(X_test)
```

8. Evaluation metrics

```
[ ] # Obtain accuracy on train set
    accuracy_score(Y_train,train_preds)
```

```
0.9995391603941276
```

```
[ ] # Obtain accuracy on test set
    accuracy_score(Y_test,test_preds)

    0.998683402089002
```

Accuracy is not a good metric to evaluate our model since the dataset is heavily imbalanced. Thus we will use roc_auc score

```
[ ] # Calculate roc_auc score on train set
    roc_auc_score(Y_train,train_preds)

    0.8973809415105495
```

```
[ ] # Calculate roc_auc score on test set
    roc_auc_score(Y_test,test_preds)

    0.7996922535830476
```

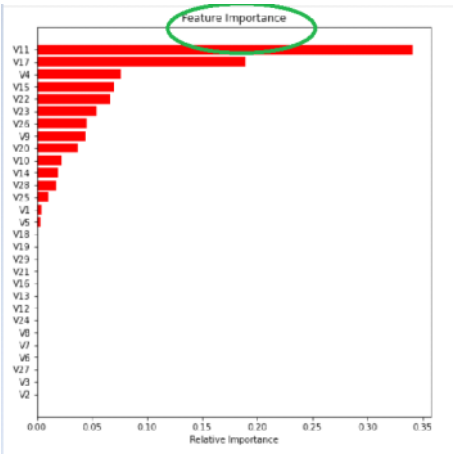
```
[ ] # Obtain the confusion matrix on train set
    confusion_matrix(Y_train,train_preds)

    array([[45486,     5],
           [   16,    62]])
```

```
    # Obtain the confusion matrix on test set
    confusion_matrix(Y_test,test_preds)

    array([[11366,     7],
           [    8,    12]])
```

9. Plot for Feature Importance

```
[ ] features = X_train.columns
    importances = classifier.feature_importances_
    indices = np.argsort(importances)
```

```
    plt.title('Feature Importance')
    plt.barh(range(len(indices)), importances[indices], color='red', align='center')
    plt.yticks(range(len(indices)), [features[i] for i in indices])
    plt.xlabel('Relative Importance')

    plt.show()
```



- Feature (variable) importance indicates how much each feature contributes to the model prediction. Basically, it determines the degree of usefulness of a specific variable for a current model and prediction

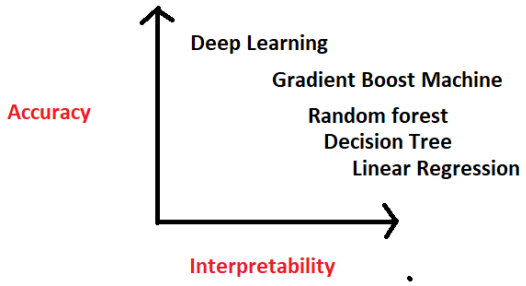
Note

12 October 2022 15:56

I.Q.

How you choose mode? (2ways)

1. I will choose according to the preference between Accuracy and Interpretation.



2. Based on the Pattern of the Dataset
   - If it is linear, then choose linear regression, ridge, lasso
   - If it not linear, many categorical then, Decision tree

How the Number of Trees/ model will be choose?

- As it is "art of modeling"
- What value of k
- What value of depth
- All should be art of modelling

1 Which of the following is true about "max_depth" hyperparameter in Gradient Boosting? 1) Lower is better parameter in case of same validation accuracy 2) Higher is better parameter in case of same validation accuracy 3) Increase the value of max_depth may overfit the data 4) Increase the value of max_depth may underfit the data

Your Answer 1 and 3

5 In Random forest you can generate hundreds of trees (say T1, T2 .....Tn) and then aggregate the results of these tree. Which of the following is true about individual(Tk) tree in Random Forest?

Your Answer Individual tree is built on a subset of the features
Individual tree is built on a subset of observations

- Random forest is based on bagging concept, that consider faction of sample and faction of feature for building the individual trees.

8 Suppose you are using a bagging based algorithm say a RandomForest in model building. Which of the following can be true? 1) Number of tree should be as large as possible 2) You will have interpretability after using RandomForest

Your Answer 1

- Since Random Forest aggregate the result of different weak learners, If It is possible we would want more number of trees in model building. Random Forest is a black box model you will lose interpretability after using it.