Predicting the Success Hit of Video Games

A Case Study on Hits of Video Games using Machine Learning

Abstract

In this context we will present the notion of Machine Learning model in real life. We will introduce to the dataset on which we will work to understand the data and how it will help to predict the success or failure of any game. Then we will move towards data analysis and will get answers for the reason behind the success of any game depending on the previous data we have. Moreover, we will predict the success of games by applying machine learning models to the dataset and finding the accuracy of our model.

Somesh Sharma
someshsharma@jklu.edu.in

Department of Computer Science and Engineering
Institute of Engineering
JK Lakshmipat University, Jaipur, India

CONTEXT

1 Introduction

In this report we have used the vgsales dataset from Kaggle, to view data set click here. We will predict the video game hits i.e. weather a game will be sold over one million units. There are thousands of game releases in a month. But, many few of them make it to top games. This is because they have got the idea that what is the demand of public which type of games they actually want and this is the reason they are able to make their way to top by analysing the data they have. There are some ways to analyse and predict the outcomes of games such as analysing previous sales, the genre and company name, market budget for such game, peek of games, change in the prices, and reviews of gamers.

1.1 Analysis of previous data

Taking a look at the dataset and try to predict how next game will perform. Assuming if the previous versions of games are sold well, then might be continuation of series will be successful or might be better than previous one. Game Retailers as well as manufacturers are qualified to predict game sales as long as they have necessary data.

1.2 Genre/Developer's Name

Not all games have previous versions but every game fits with its genre and developer's name. If the games previous versions have big sales, then we are 100% sure that the upcoming version will attract more fans. Similarly, if a developer have a bestseller portfolio, we can expect the similar results or better results as compared to previous versions. The best way would be to use your own data of sales and check the sales of genre as game release is near.

1.3 Marketing budget

Sometimes an idea at the marketing budget helps to evaluate the selling rate of a game. Let's take an example of GTA 5 whose development costs amount of 115 million dollars. In 2015, August Rockstar Games sold 54 million units of GTA 5. Additional advertisements are the sign that developers and publishers wants to reach more players.

1.4 Understanding games publications

Every video game industry helps in assessing the sales rate of a game. Those people who contribute to them and take part in various activities such as trade fairs, beta tests and shows also helps them to improve the game. So, it always pays off to depend on their predictions too. Not always the amount of advertising helps the game to be a hit. Focusing on articles and reviews both positive and negative helps to overcome obstacle.

1.5 Customer's Expectation

The numbers of fans is another reason for game success. We can analyse this by collecting the information like what fans are posting and expecting from you from different platforms like twitch, facebook and many other.

1.6 Change in Price

This factor helps to grab the sales potential of games by its price including pre-releases and it grows when the release dates come closer. Moreover, if the review gets poor the price of game drops before release

Hence, there are many more factors which affect the games hit but in this report we will cover this topic not on the basis of myths and unknown sources. We will get answers using technical aspects.

2 Learning Outcomes

In this document you discovered the Adaboosting Boosting ensemble method for machine learning. You learned about:

- Understanding the objective of business, how we will use the data provided to us.
- Finding the maximum sources we have to collect data.
- Analyze which data is useful for us to meet our objective.
- Requirement of preprocessing and how to clean data that it should perfectly fit on our training model.
- Working of any base model and how it helps to predict any classification or regression.
- Role of a boosting algorithm in Machine Learning and how do we implement boosting algorithm.
- Analyzing either our model is working in best case scenario or not.

3 Suggested Questions

- Why we need to predict the hit of a video game?
- Which Rating type of games lead the market?
- How hit of a game is defined?
- How does Critic score defines the popularity of video games?
- How correlations effect the prediction in Machine Learning?
- Why boosting is required in Machine Learning?

Before starting the analysis take a look at the data. We need to work analyse it properly because there are many null values present in data. This data includes feature such as Sales, Name, Rating, Critic Score, User Score, etc. We need to analyse the relation between these columns.

3.1 Why we need to predict the hit of a video game?

The need of prediction gives the sense of control. As we predict what is going to happen we have changes to take things under control. We can have chance to manage the situation we are in. The required objective of prediction is to find the way toward perfect decision which can affect in positive aspect and if not, is should maintain the workflow of any organisation but never have a negative impact.

Taking any step in context to video game industry or in any other platform, prediction is required in everywhere for the success. Whenever a company starts manufacturing a game they have a prediction of how much a single game can benefit them depending on the multiple features as we discussed in previous section. But on talking about the biases we cannot reach to the decision because we do not know how precise or accurate we are. Here comes the role of machine learning which helps the organisation to get to a decision with percentage of precision and accuracy depending on previous data we have. This helps organization that which type of games and with what ratings are popular in which regions.

3.2 How hit of a game is defined?

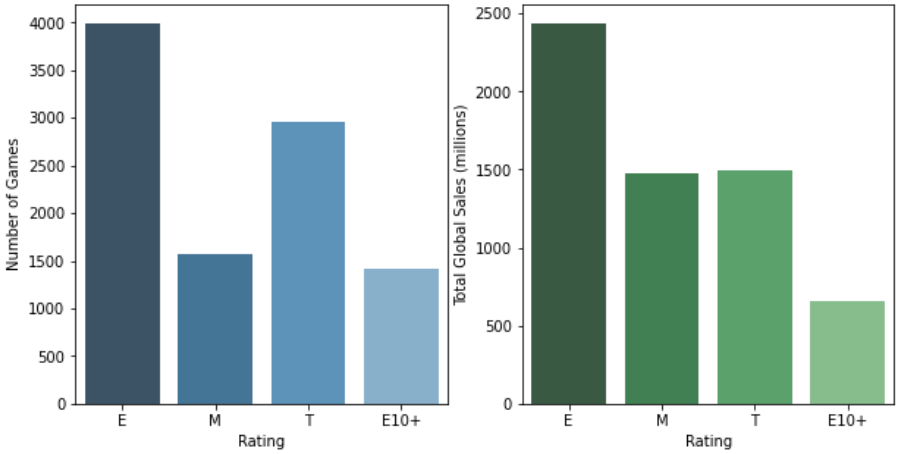
Usually a game is said to be successful when they have achieved or certified their position in top awards like D.I.C.E. Awards, Electronic Gaming Awards and many more. Moreover, we can define hit of a game by the sale of units of video games. We will represent every million unit of sales as 1. If a game is sold over 1 millions of units it will represent in category hit.

3.3 Which Rating type of games lead the market?

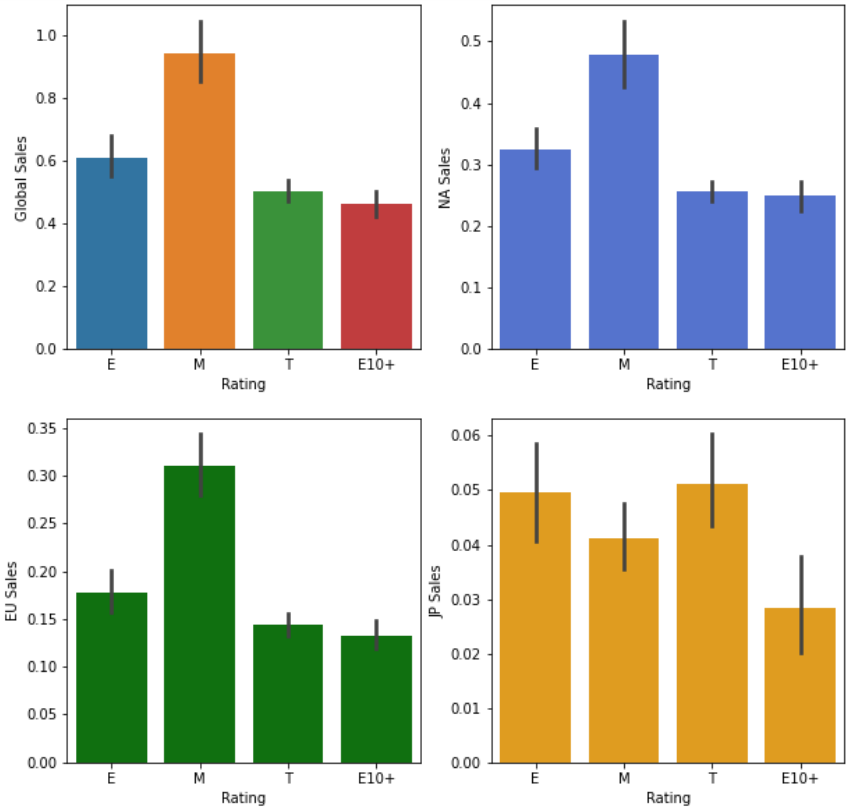
Relation between Rating and Sales of Video Games

From above figure we can observe that most of the manufactured games lies under **Rating 'E'** and are the most sold games in millions. On other hand games of Rating 'E10+' is least manufactured and least sold. One thing we can notice is that games which lies under Rating 'M' are manufactured less than Rating 'T' but have competed well in sales with the same. To see the respective input of the data **click here**.

While analysing the dataset we have found that some rating counts were below 10 so we reject those values as it will affect the visualisation and distribution of data. Here are the top four ratings which we will consider in the dataset named 'E', 'M', 'T', 'E10+'.



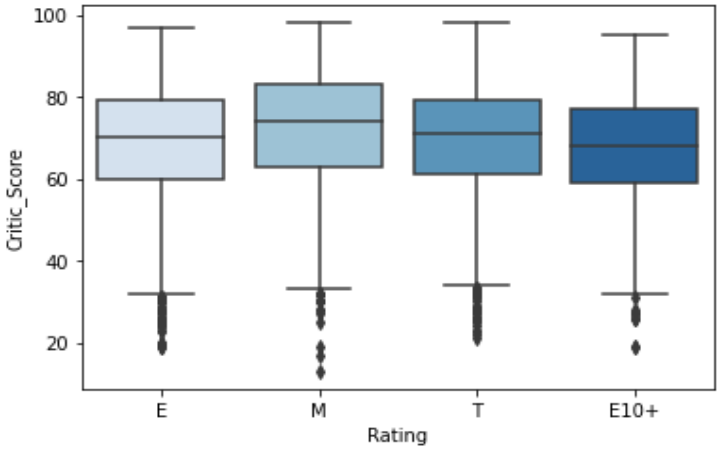
Now let's take a look at the same relation but with different region. As compared with global sales only japan sales are different from other sales because in this region most sold game units are of Rating 'T' while on other regions it is M. Hence we can say that in japan games with Rating 'E', 'M', 'T' are consider to be most popular games. So from the data we have we can say that different Rating types are popular in different regions. This is an important feature to notice as it will impact on the sales of manufacturers and publishers.



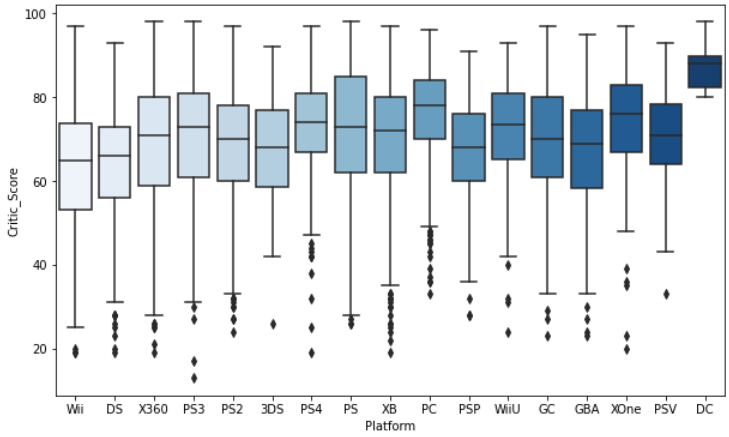
3.4 How does critic score defines the popularity of video games?

Understanding the impact of critic score with other features.

First, let's have a look on the graph to check either critic score is biased or not on the basis of Platform or Rating of Games.



From this figure we can compare the biasness of critic score with respect to Rating and we can see that it's not biased. Moving towards the next feature we will look at the biasness of critic score with the Platform.



This figure seems quite interesting as the critic score is biased only for Platform DC (DreamCast). We need to notice that lot of games in Dreamcast do not have reviews on them. Here we can say that might be this platform is not much popular for games.

Let's have a look to games and publisher which have lower critic score. These games were not attractive. This factor is important because these games will not lie in the category of hit games.

Name	Publisher
Nickelodeon Party Blast	Infogamers
Chicken Shoot	Zoo Digital Publishing
Rugby 15	Bigben Interactive
Leisure Suit Larry: Box Office Bust	CodeMasters
Anubis 2	Metro 3D
Ride to Hell	Deep Silver
Balls of Fury	Zoo Digital Publishing

Click here for better view.

Critic vs Genre: Racing and Action games have the highest critics and are the most sold games having highest rating.

Critic vs Platform: PS (Play Station) and X360(Microsoft) are the top platform for the game sales.

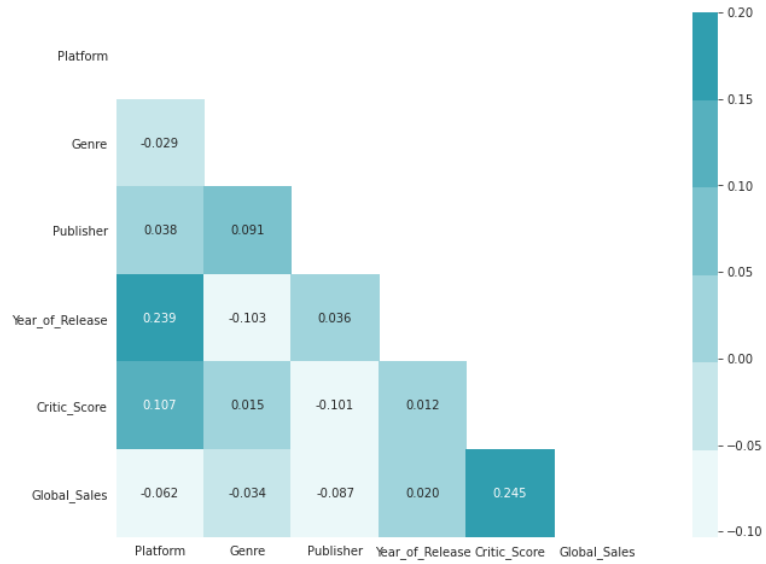
Critic vs Publisher: Microsoft Game Studios, Warner Bros., Nintendo are the top game publishers.

Here according to the dataset the top critic score values lies under the hit categories. For example, if an action or racing game is of platform play station or X360 and published by Microsoft Game Studios or Warner Bros. it will lie under the hit category. Here we finally know that how a high critic value is a factor of a game's hit.

3.5 How correlations effect the prediction in Machine Learning?

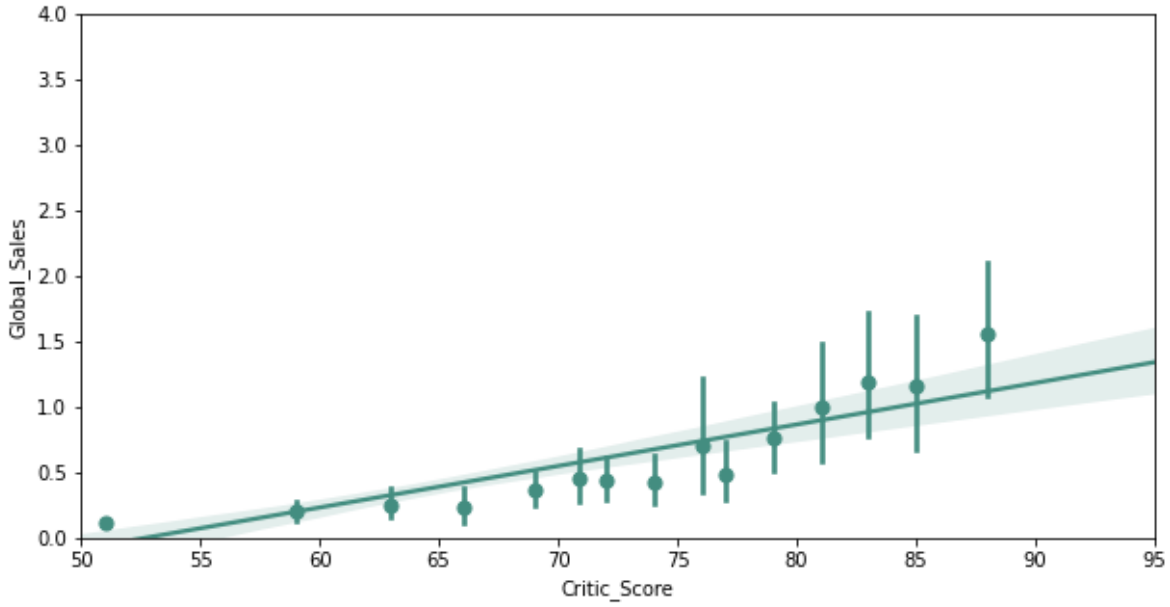
Before entering in Machine learning model we need to find the correlation between multiple features. The features with the highest correlation will help us to define if the game is hit or not.

For plotting a correlation matrix we have separated the categorical features and converted them into numerical code.



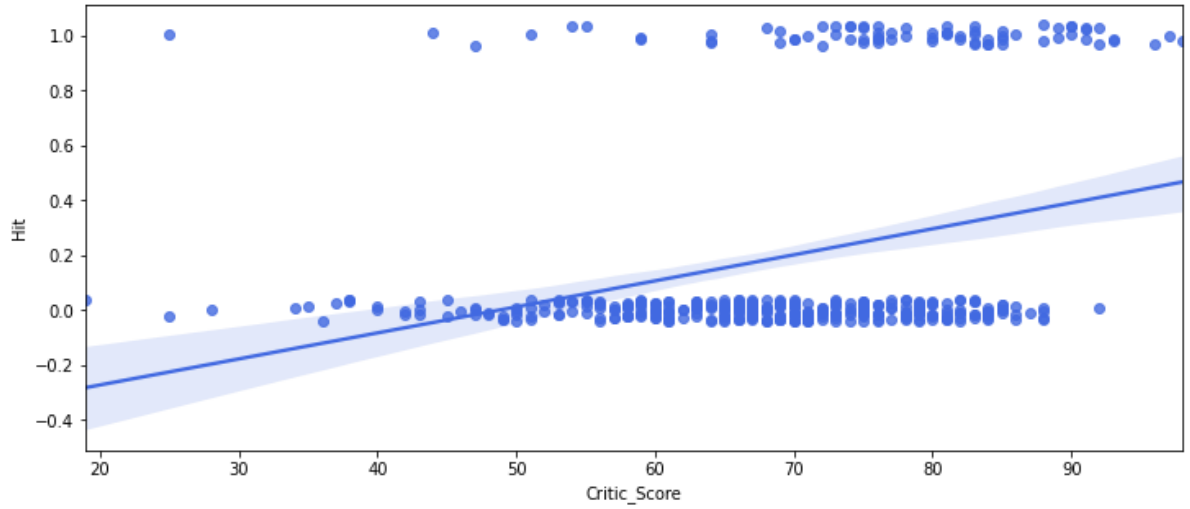
After plotting the correlation matrix we can see that **Year of Release** is highly correlated to the **Platform** and **Global Sales** to the **Critic Score**. We will now closely look at these features.

Now let's have a close look at the graph of Global Sales vs Critic Scores.



This graph is representing the critic_score of year of release greater than 2014. We can notice that from around 75 slope started getting steeper. This tells us that every additional point gives a higher impact. For example, starting from 65 increase of 8 points gives the higher value of 200k of sales.

Now we have defined our target variable, as we mentioned that a Game lies in Hit if it completes its 1 million unit of sales. So, we have defined a function which will add the Hit column and will be our target variable for prediction.



As we know that the higher the critic score more hit a games are. Here we can notice the importance of correlation matrix if the relation value lies on 0 or below 0 it clearly means that the features are not related to each other and higher the value of correlation matrix it tells that they are connected to each other. Now we have finally defined our target variable we can move towards prediction model.

3.6 Why boosting is required in Machine Learning?

To know why boosting is important let's make prediction using a base model i.e. some classifier algorithm which will be used further by boosting algorithm. To do this we have encoded the categorical variables into binary form which we have encoded to numbers in previous section.

	Year_of_Release	Critic_Score	Hit	Platform_0	Platform_1	Platform_2	Platform_3	Platform_4	Platform_5	Platform_6	Platform_7	Platform_8
0	2006.0	76.0	1	0	0	0	1	0	0	0	0	0
1	2008.0	82.0	1	0	0	0	1	0	0	0	0	0
2	2009.0	80.0	1	0	0	0	1	0	0	0	0	0
3	2006.0	89.0	1	0	1	0	0	0	0	0	0	0
4	2006.0	58.0	1	0	0	0	1	0	0	0	0	0

5 rows × 334 columns

Dummy variables are quantative variables which only contains two values either 0 or 1. For example, to understand the relationship between income of republican and democrat, we can define expression as:

$$I = a + a1X1 + a2X2$$

Where X1 and X2 represents republican and democrat respectively and 'a' is some constant.

Here, X1 = 1 if Republican, else 0

Similarly, X2 = 1 if Democrat, else 0

We always need to define the dummies in systematic order unless it will lead to the multicollinearity so if we have k categorical values we always need to define k-1 dummies else the kth dummy will be useless and holds no new information.

Next, we have divided our dataset into training and test. We use 70% of dataset for training so that we can leave least possible learners behind. And another 30% to test our accuracy on dataset. We have use decision tree with max depth of 1 because in this situation we are just categorizing in 2 categories. Decision tree with 1 depth also have two node i.e. either 0 or 1.

After running the algorithm here the output of confusion matrix:

	precision	recall	f1-score	support
0	0.83	1.00	0.91	1998
1	0.00	0.00	0.00	397
accuracy			0.83	2395
macro avg	0.42	0.50	0.45	2395
weighted avg	0.70	0.83	0.76	2395

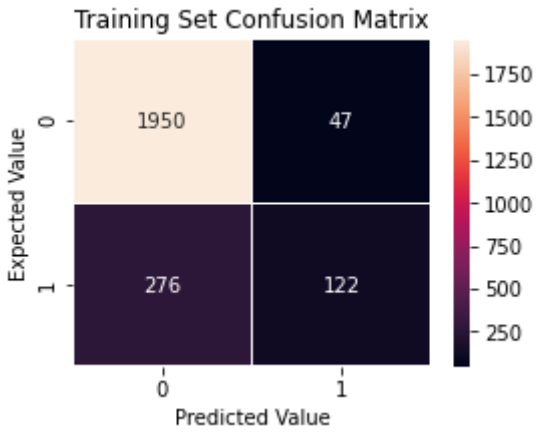
Here precision is 0.83 which is pretty good and we can say that this model relates to the low false positive rate. As we notice recall's value all the values in dataset are labelled.

Here we can notice that the precision is 83% here we can apply boosting algorithm to increase the precision of our model. The role of boosting algorithm in machine learning is comparatively same as a loop in any program. Boosting works by combining the leak learner and forming a strong learner by increasing the weight of incorrectly classified elements and decreasing the weight of correctly classified elements. To understand the working of algorithm please refer to the appendix.

After applying the boosting algorithm we get the following output:

	precision	recall	f1-score	support
0	0.88	0.98	0.92	1997
1	0.72	0.31	0.43	398
accuracy			0.87	2395
macro avg	0.80	0.64	0.68	2395
weighted avg	0.85	0.87	0.84	2395

As we can see, the precision of this model is upgraded from 83% to 88%. Here we can understand the role of boosting in Machine learning. The boosting algorithms are independent of the type of prediction we are doing either its classification or regression problem. Now let's have a look at the confusion matrix of our final model.



The figure represent the confusion matrix of training dataset which was 30% part of the actual dataset. 1950 records are true positive which means they are positive points (Hits) which were correctly classified and 122 records were negative points (Not Hits) which were correctly classified. On the other side there are 323 records which are incorrectly classified on the basis of Hit and Not Hit.

4 Appendix (A)
Explaining the algorithm

As we have mentioned the context of boosting algorithm which we have used for classification. Here's a small example to understand the working of adaboosting algorithm in field of classification prediction.

Weak Learner

In ensemble modelling theory, we can say that weak learners/ base models are used as building blocks for designing a complex models by combining weak learners. Moreover, these base models do not perform so well by themselves either because they have high bias or because they have too much variance. The idea of any ensemble models is to keep trying reducing the bias or variance of these weak learners by combining them so that it can form a strong learner or we can say an ensemble model which achieves better performance.

Combining Weak Learner

As we need to set up an ensemble model, firstly, we need to set our base model. Mostly, a single base model is used so that we have homogeneous set of weak learners that can be trained in different forms. Then, the ensemble model we get is said to be homogeneous. Moreover, there are also some methods exist which uses different type of base models: so that some heterogeneous weak learners are combined to form a heterogeneous ensembles model.

Algorithm

While considering a binary classification, we can identify that the AdaBoost algorithm is the process that proceeds as follow. Firstly, it updates the existing observations weight depending on precious weak learners in the dataset and train some new weak learner with the focus on the observations which was misclassified in the previous ensemble model. Next, it adds these weak learners to improve performances to this base model. The more better a weak learner perform the more it contribute for the strong learner.

From the beginning of the algorithm, all observations have assigned same weight 1/N. Then we repeat the following steps L times which is the L learners in the sequence of dataset:

1. Fit all best possible weak learners with the current observations weights we have assigned.
2. Now compute the value of the coefficient which is some kind of scalar evaluation metric for the weak learner. It required to be taken into the ensemble model.
3. Here we need to update the strong learner by adding new weak learner multiplied by its updated coefficient.
4. Finally, we compute new observations weights that tells which observations we would like to focus in our next iteration.

By repeating the above steps we have successfully build our sequential ensemble model. We need to notice that adaptive boosting tries to solve at each iteration exactly like optimisation problem does.

Tracking

Now, let's track this algorithm with the help of an example. Consider following dataset:

x1	x2	Decision
2	3	TRUE
2.1	2	TRUE
4.5	6	TRUE
4	3.5	FALSE
3.5	1	FALSE
5	7	TRUE
5	3	FALSE
6	5.5	TRUE
8	6	FALSE
8	2	FALSE

In decision trees non-linear are the decision rules which are nested. Moreover, decision trees with 1 - depth are single level decision trees also called decision stumps. Now, let's consider the decision block in the above table is the output of some features (input) in any base machine learning model. Now, set true equals to 1 and false equals to –1.
The main motto in AdaBoost is to increase the weight of unclassified (incorrectly classified) ones and to decrease the weight value of classified ones.
Initially, we distribute weights in uniform distribution. i.e. **w = I/n, where n = no. of data instance.**

x1	x2	Decision	weight	weight_actual
2	3	1	0.1	0.1
2.1	2	1	0.1	0.1
4.5	6	1	0.1	0.1
4	3.5	-1	0.1	-0.1
3.5	1	-1	0.1	-0.1
5	7	1	0.1	0.1
5	3	-1	0.1	-0.1
6	5.5	1	0.1	0.1
8	6	-1	0.1	-0.1
8	2	-1	0.1	-0.1

weighted_actual stores weight multiplied to actual value in each line. Now, we need to use weighted_actual as targe. We have set the actual values as +1 or –1 but decision stump returns decimal values. So, **prediction** for 1st row where x1 = 2, on iteration with other rows if x1 > 2 it will be

sign –1 and if x1 <= 2 then it will be sign +1. Moreover, if prediction is correct we will mark it as 0 and if it is wrong we will mark it as 1. See below table for next observation.

x1	x2	Decision	weight	weight_actual	prediction	loss	weight * loss
2	3	1	0.1	0.1	1	0	0
2.1	2	1	0.1	0.1	1	0	0
4.5	6	1	0.1	0.1	-1	1	0.1
4	3.5	-1	0.1	-0.1	-1	0	0
3.5	1	-1	0.1	-0.1	-1	0	0
5	7	1	0.1	0.1	-1	1	0.1
5	3	-1	0.1	-0.1	-1	0	0
6	5.5	1	0.1	0.1	-1	1	0.1
8	6	-1	0.1	-0.1	-1	0	0
8	2	-1	0.1	-0.1	-1	0	0

Sum of weight multiplied to loss is the total error. In this case E = 0.3.
Now we will define α variable, where $\boldsymbol{\alpha = \frac{1}{2}\log((1 - E)/E)}$
Here, α = 0.42
Now here, α is used to update the weight in the next round.
$\boldsymbol{W_{i+1} = W_i * e^{-\alpha}}$, where W_{i+1} is the updated weight.
Here, sum of updated weight should match to 1. So, we need to normalize these weight values. Normalized weight can be computed by dividing each weight value to the sum of weights.

x1	x2	Decision	weight	weight_actual	prediction	w_(i+1)	norm(w_(i+1))
2	3	1	0.1	0.1	1	0.065	0.071
2.1	2	1	0.1	0.1	1	0.065	0.071
4.5	6	1	0.1	0.1	-1	0.153	0.167
4	3.5	-1	0.1	-0.1	-1	0.065	0.071
3.5	1	-1	0.1	-0.1	-1	0.065	0.071
5	7	1	0.1	0.1	-1	0.153	0.167
5	3	-1	0.1	-0.1	-1	0.065	0.071
6	5.5	1	0.1	0.1	-1	0.153	0.167
8	6	-1	0.1	-0.1	-1	0.065	0.071
8	2	-1	0.1	-0.1	-1	0.065	0.071

Iteration 2

Now update the normalized weight to weight column. Again, build the decision stumps. Use x1 or x2 features where weighted_actual is the target column. Note weighed_actual is weight multiplied to actual.

x1	x2	Decision	weight	weighted_actual	loss	Weight*loss
2	3	1	0.071	0.071	1	0.071
2.1	2	1	0.071	0.071	1	0.071
4.5	6	1	0.167	0.167	0	0
4	3.5	-1	0.071	-0.071	0	0
3.5	1	-1	0.071	-0.071	0	0
5	7	1	0.167	0.167	0	0
5	3	-1	0.071	-0.071	0	0
6	5.5	1	0.167	0.167	0	0
8	6	-1	0.071	-0.071	1	0.071
8	2	-1	0.071	-0.071	0	0

Again repeat the same steps.

For Iteration 2 E = 0.21, α = 0.65
Here, weight of the following iteration will be:

x1	x2	Decision	weight	prediction	w_(i+1)	norm(w_(i+1))
2	3	1	0.071	-1	0.137	0.167
2.1	2	1	0.071	-1	0.137	0.167
4.5	6	1	0.167	1	0.087	0.106
4	3.5	-1	0.071	-1	0.037	0.045
3.5	1	-1	0.071	-1	0.037	0.045
5	7	1	0.167	1	0.087	0.106
5	3	-1	0.071	-1	0.037	0.045
6	5.5	1	0.167	1	0.087	0.106
8	6	-1	0.071	1	0.137	0.167
8	2	-1	0.071	-1	0.037	0.045

Iteration 3

x1	x2	Decision	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.167	1	0	0.000	0.114	0.122
2.1	2	1	0.167	1	0	0.000	0.114	0.122
4.5	6	1	0.106	-1	1	0.106	0.155	0.167
4	3.5	-1	0.045	-1	0	0.000	0.031	0.033
3.5	1	-1	0.045	-1	0	0.000	0.031	0.033
5	7	1	0.106	-1	1	0.106	0.155	0.167
5	3	-1	0.045	-1	0	0.000	0.031	0.033
6	5.5	1	0.106	-1	1	0.106	0.155	0.167
8	6	-1	0.167	-1	0	0.000	0.114	0.122
8	2	-1	0.045	-1	0	0.000	0.031	0.033

E = 0.31, α = 0.38

Iteration 4

x1	x2	Decision	weight	prediction	loss	w * loss	w_(i+1)	norm(w_(i+1))
2	3	1	0.122	1	0	0.000	0.041	0.068
2.1	2	1	0.122	1	0	0.000	0.041	0.068
4.5	6	1	0.167	1	0	0.000	0.056	0.093
4	3.5	-1	0.033	1	1	0.033	0.100	0.167
3.5	1	-1	0.033	1	1	0.033	0.100	0.167
5	7	1	0.167	1	0	0.000	0.056	0.093
5	3	-1	0.033	1	1	0.033	0.100	0.167
6	5.5	1	0.167	1	0	0.000	0.056	0.093
8	6	-1	0.122	-1	0	0.000	0.041	0.068
8	2	-1	0.033	-1	0	0.000	0.011	0.019

E = 0.10, α = 1.10

Prediction
Cumulative sum of all round's alpha times prediction gives the final prediction.

round 1 alpha	round 2 alpha	round 3 alpha	round 4 alpha
0.42	0.65	0.38	1.1
round 1 prediction	round 2 prediction	round 3 prediction	round 4 prediction
1	-1	1	1
1	-1	1	1
-1	1	-1	1
-1	-1	-1	1
-1	-1	-1	1
-1	1	-1	1
-1	-1	-1	1
-1	1	-1	1

Here, 0.42 x 1 + 0.65 x (-1) + 0.38 x 1 + 1.1 x 1 = 1.25
And when we apply sign function, Sign (1.25) = +1, also known as true which is correctly classified.

5 Appendix (B)

5.1 Python Code for implementation

1. Data Analysis of Rating with respect to Sales

As we need to know the aspect of Column Rating we will select only those columns which have no null values.

```
[ ] dfrat = df[df.Rating.notnull()]
    dfrat.head()
```

	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score
0	Wii Sports	Wii	2006.0	Sports	Nintendo	41.36	28.96	3.77	8.45	82.53	76.0	51.0	8
2	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.68	12.76	3.79	3.29	35.52	82.0	73.0	8.3
3	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.61	10.93	3.28	2.95	32.77	80.0	73.0	8

Here, we grouped the dataset on the basis of Rating and we can notice that there are some Ratings which have very low counts.

```
[ ] dfrat.groupby(df['Rating']).count()
```

Rating	Name	Platform	Year_of_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
AO	1	1	1	1	1	1	1	1	1	1
E	3991	3991	3922	3991	3989	3991	3991	3991	3991	3991
E10+	1420	1420	1393	1420	1418	1420	1420	1420	1420	1420
EC	8	8	8	8	8	8	8	8	8	8
K-A	3	3	3	3	3	3	3	3	3	3
M	1563	1563	1536	1563	1562	1563	1563	1563	1563	1563
RP	3	3	1	3	3	3	3	3	3	3
T	2961	2961	2905	2961	2959	2961	2961	2961	2961	2961

We will discard the Rating with low counts so that our data should remain normalized and we can only focus on top values.

```
[ ] masking = np.logical_not(dfrat['Rating'].isin(['AO','EC','K-A', 'RP']))
    dfrat = dfrat[masking]
    dfrat['Rating'].unique()
```

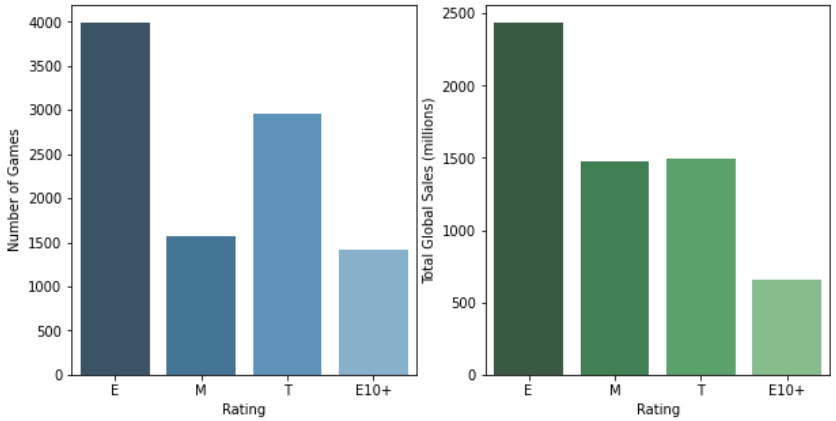
```
array(['E', 'M', 'T', 'E10+'], dtype=object)
```

```
df_group = dfrat[['NA_Sales','EU_Sales','JP_Sales','Global_Sales','Other_Sales']].groupby(dfrat['Rating']).sum()
df_group
```

Rating	NA_Sales	EU_Sales	JP_Sales	Global_Sales	Other_Sales
E	1293.26	710.25	198.11	2436.90	234.19
E10+	353.32	188.52	40.20	655.81	73.56
M	748.48	483.97	64.24	1473.84	177.10
T	759.75	427.03	151.40	1494.40	155.17

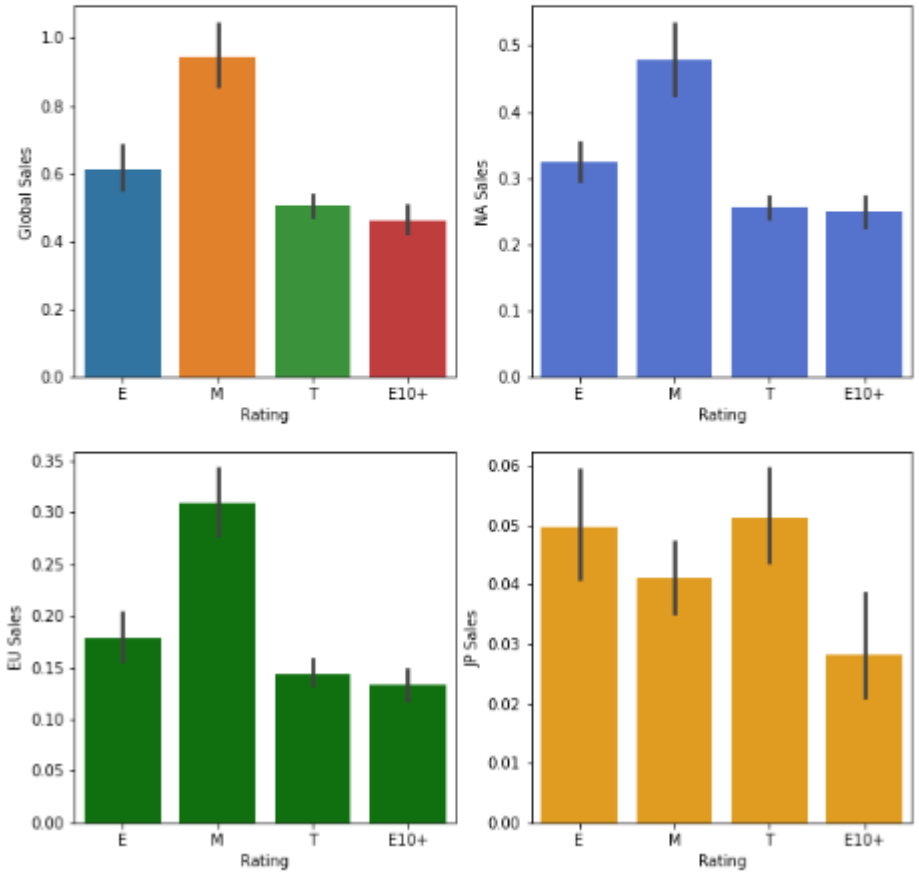
Here we have visualized the number of games vs Rating and which rating sales are more popular.

```
f, (ax1, ax2) = plt.subplots(1,2, figsize=(10,5))
sns.countplot(x="Rating", data=dfrat, palette="Blues_d", ax=ax1)
ax1.set_ylabel("Number of Games")
sns.barplot(x="Rating", y="Global_Sales",data=dfrat, palette="Greens_d", estimator=sum, ax=ax2, ci=None)
ax2.set_ylabel("Total Global Sales (millions)")
plt.show()
```



Now we compared the Rating on the basis of Region Sales.

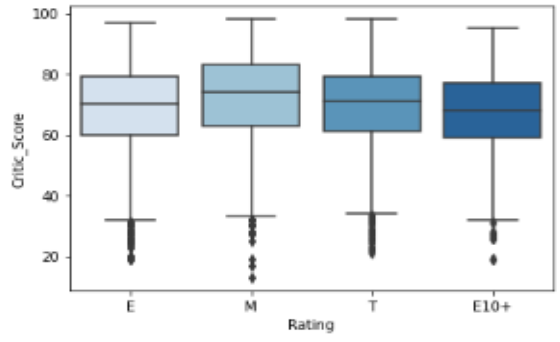
```
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(10,10))
sns.barplot(x="Rating", y="Global_Sales", data=dfrat, ax=ax1)
ax1.set_ylabel("Global Sales")
sns.barplot(x="Rating", y="NA_Sales", color="royalblue", data=dfrat, ax=ax2)
ax2.set_ylabel("NA Sales")
sns.barplot(x="Rating", y="EU_Sales", color="green", data=dfrat, ax=ax3)
ax3.set_ylabel("EU Sales")
sns.barplot(x="Rating", y="JP_Sales", color="orange", data=dfrat, ax=ax4)
ax4.set_ylabel("JP Sales")
plt.show()
```



2. Data Analysis of Critic Score and visualizations.

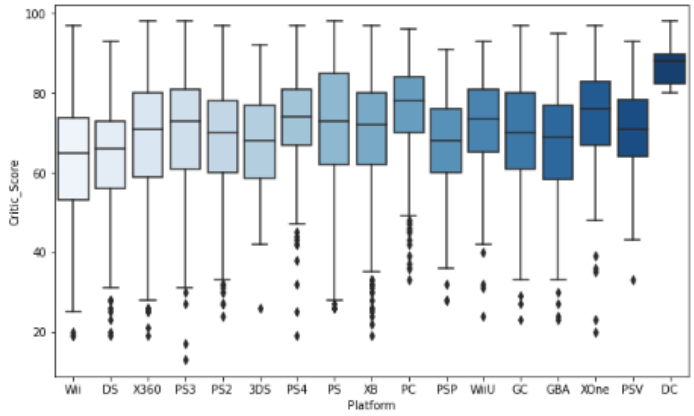
We have checked that are Critic Score biased for Ratings

```
sns.boxplot(data=dfrat, x="Rating", y="Critic_Score", palette="Blues")
plt.show()
```



Now we have checked the biasness of Critic Score on the platforms which we have for video games.

```
fig, ax = plt.subplots(figsize=(10,6))
sns.boxplot(ax=ax, data=dfrat, x="Platform", y="Critic_Score", palette="Blues")
plt.show()
```



Then we have find out the Names of games and those publishers who have very low critic score.

```
dfrat.query("Critic_Score < 20").Name
```

```
7219                  Nickelodeon Party Blast
9106                            Chicken Shoot
10016                               Rugby 15
10663    Leisure Suit Larry: Box Office Bust
12132                             Anubis II
12823                          Ride to Hell
13690                          Ride to Hell
15345                         Balls of Fury
Name: Name, dtype: object
```

```
dfrat.query("Critic_Score < 20").Publisher
```

```
7219                 Infogrames
9106     Zoo Digital Publishing
10016        Bigben Interactive
10663               Codemasters
12132                  Metro 3D
12823               Deep Silver
13690               Deep Silver
15345    Zoo Digital Publishing
Name: Publisher, dtype: object
```

Then we have analysed the Critic score on the basis of score with respect to Genre, Platform and publisher.

```
def score_grp(score):
    if score > 89:
        return '90-100'
    elif score > 79:
        return '80-89'
    elif score > 69:
        return '70-79'
    elif score > 59:
        return '60-69'
    elif score > 49:
        return '50-59'
    else:
        return '0-49'
```

```
dftemp = df.dropna(subset=['Critic_Score']).reset_index(drop=True)
dftemp['Score_Group'] = dftemp['Critic_Score'].apply(lambda x: score_grp(x))
```

```
def in_top(x):
    if x in pack:
        return x
    else:
        pass
def width(x):
    if x == 'Platform':
        return 14.4
    elif x == 'Publisher':
        return 11.3
    elif x == 'Genre':
        return 13.6

def height(x):
    if x == 'Genre':
        return 8
    else:
        return 9
```

```
cols = ['Genre', 'Platform', 'Publisher']
for col in cols:
    pack = []
    top = dftemp[['Name', col]].groupby([col]).count().sort_values('Name', ascending=False).reset_index()[:15]
    for x in top[col]:
        pack.append(x)
    dftemp[col] = dftemp[col].apply(lambda x: in_top(x))
    dfh_platform = dftemp[[col, 'Score_Group', 'Global_Sales']].groupby([col, 'Score_Group']).median().reset_index().pivot(col, "Score_Group", "Global_Sales")
    plt.figure(figsize=(width(col), height(col)))
    cmap=sns.cubehelix_palette(8, start=.5, rot=-.75)
    sns.heatmap(dfh_platform, annot=True, fmt=".2g", linewidths=.5, cmap=cmap).set_title((' \n'+col+' vs. critic score \n'), fontsize=18)
    plt.ylabel('', fontsize=14)
    plt.xlabel('Score group \n', fontsize=14)
    pack = []
```

After this we have plotted correlation matrix so that we can focus on our important features.

```
cols = ['Platform', 'Genre', 'Publisher', 'Developer', 'Rating']
for col in cols:
    uniques = df[col].value_counts().keys()
    uniques_dict = {}
    ct = 0
    for i in uniques:
        uniques_dict[i] = ct
        ct += 1

    for k, v in uniques_dict.items():
        df.loc[df[col] == k, col] = v
```

```
dff = df[['Platform','Genre','Publisher','Year_of_Release','Critic_Score','Global_Sales']]
dff = dff.dropna().reset_index(drop=True)
dff = dff.astype('float64')
```

```
mask2 = np.zeros_like(dff.corr())
mask2[np.triu_indices_from(mask2)] = True
cmap = sns.light_palette((210, 90, 60), input="husl")
with sns.axes_style("white"):
    fig, ax = plt.subplots(1,1, figsize=(15,8))
    ax = sns.heatmap(dff.corr(), mask=mask2, vmax=0.2, square=True, annot=True, fmt=".3f", cmap=cmap)
```

Here, we found that Critic-Score and Global-Sales were highly correlated. Also, Critic_Score were highly correlated to our target variable ‘Hit’. Then we plotted the regression plots between them. Last step was to create a prediction model so we created a base model of Decision Tree.

As we are following the objective for classification problem. We encoded our categorical records in binary form.

```
df_copy = pd.get_dummies(df2)
```

```
df_copy.head()
```

	Year_of_Release	Critic_Score	Hit	Platform_3DS	Platform_DC	Platform_DS	Platform_GBA	Platform_GC	Platform_PC	Platform_PS	Platform_PS2
0	2006.0	76.0	1	0	0	0	0	0	0	0	0
1	2008.0	82.0	1	0	0	0	0	0	0	0	0
2	2009.0	80.0	1	0	0	0	0	0	0	0	0
3	2006.0	89.0	1	0	0	1	0	0	0	0	0
4	2006.0	58.0	1	0	0	0	0	0	0	0	0

3. Prediction Model which include Decision Tree as Base Model and Adaboosting for Boosting Algorithm.

Now, we have assigned the features and labels to their respective variables so that we can split our dataset into training and testing. And finally fit the training dataset to our base model.

```
df_ml = df_copy
y = df_ml['Hit'].values
df_ml = df_ml.drop(['Hit'],axis=1)
X = df_ml.values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=2)
```

```
clf=DecisionTreeClassifier(max_depth=1, random_state=101, max_features=None, min_samples_leaf=15)
```

```
clf.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=15, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=101, splitter='best')
```

Now on the basis of training dataset we have predicted the classification on training set. Also, we have checked the accuracy by comparing it with original dataset.

```
y_pred_clf=clf.predict(X_test)
```

```
print(classification_report(y_test, y_pred_clf))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91      1997
           1       0.00      0.00      0.00       398

    accuracy                           0.83      2395
   macro avg       0.42      0.50      0.45      2395
weighted avg       0.70      0.83      0.76      2395

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
  _warn_prf(average, modifier, msg_start, len(result))
```

Now we have created another instance to implement boosting algorithm that is Adaboost and provided the base model as decision tree.

```
clff=AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50, algorithm='SAMME')
```

```
clff.fit(X_train,y_train)
```

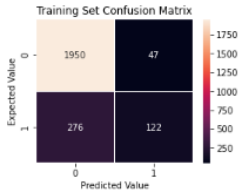
```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=1,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=None,
                                                         splitter='best'),
                   learning_rate=1.0, n_estimators=50, random_state=None)
```

Here we repeated the same step by fitting the training model and predicting the target variable on test dataset. Also, we have checked the accuracy of the model and implemented the confusion matrix to know how many target variables are correctly classified or not.

```
y_pred = clff.predict(X_test)
print(clff.score(X_test,y_test))
```

```
0.8651356993736952
```

```
fig, ax = plt.subplots(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, linewidths=.5, ax=ax, fmt="d").set(xlabel='Predicted Value', ylabel='Expected Value')
plt.title('Training Set Confusion Matrix')
plt.show()
```




```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.98      0.92      1997
           1       0.72      0.31      0.43       398

    accuracy                           0.87      2395
   macro avg       0.80      0.64      0.68      2395
weighted avg       0.85      0.87      0.84      2395
```

6 Reference

- Explaining Adaboosting, Robert E. Schapire. Click Here
- Supervised Machine Learning Algorithm, June 2017, JET Akinsola. Research Gate
- Ensemble Methods, Foundations and Algorithms, Ralf Herbrich and Thore Graepel, 2012. Click Here