# CHAPTER 4
# ANALYSIS AND DESIGN

In this project, the author analyzes the sentiment classification of movie reviews by implementing the Random Forest and Logistic Regression algorithms. With these two algorithms, you can get results whether the reviews are more inclined to positive sentiment or negative sentiment. This project also calculates the accuracy of each algorithm and compares which algorithm has the highest accuracy. The algorithm that has the highest accuracy is a good algorithm for this project. Both algorithms use steps that are slightly different from each other, the steps are as follows:

## 4.1. Random Forest

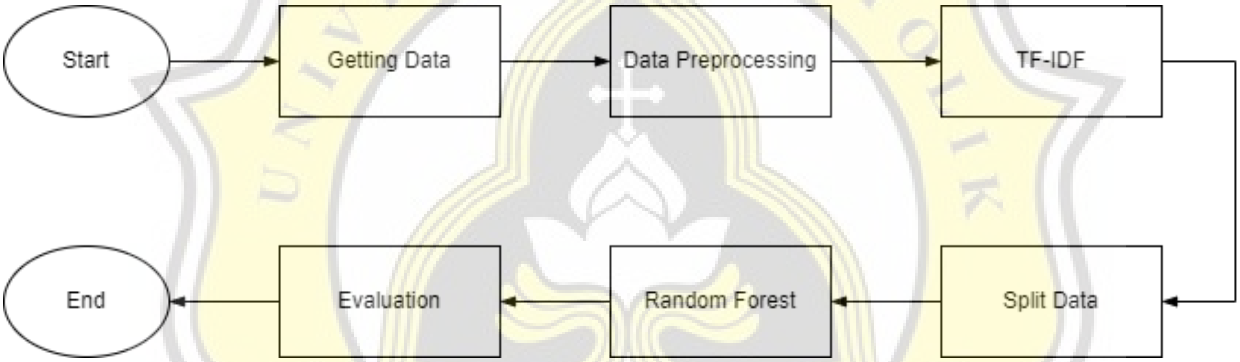


*Figure 4.1.1 Random Forest Workflow*

As seen in **Figure 4.1**, there are 6 steps for Random Forest. First getting data, then data preprocessing. Then do the TF-IDF calculation for each word. After that, the data is split into 70% for training and 30% for testing. Then implementation of Random Forest algorithm, and finally evaluation.

### 4.1.1. Getting Data

It can be seen in **Figure 4.1**, first getting data is done. The dataset used is IMDb Movie Reviews 2021, which was taken from Kaggle. In this dataset, 2.000 rows have 5 variables or features, namely id, title, review, rating, and author. Each review has a movie rating that is used to label the review later. The movie rating itself has a range from 1 to 10. The dataset table can be seen in **Table 4.1.1.1.**

*Table 4.1.1.1 Random Forest Dataset*

| ID | Title | Review | Rating | Author |
|---|---|---|---|---|
| 1 | Not Bad | I don't get the terrible reviews | 5 | margarida-44311 |
| 2 | What are all the bad reviews about is it a woke thing | The movie was great | 6 | joemay-2 |
| 3 | Great White=Jaws Lite | The CGI is not great | 4 | nebk |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | kuarinofu |
| 5 | Terrible story, dialogue, and CGI | I don’t like the terrible movie | 4 | Horror_Flick_Fanatic |
| ... | ..... | ..... | ..... | ..... |
| 4996 | What are all the bad reviews about is it a woke thing | Its poorly written, very poorly directed, poorly scripted | 8 | Jim_Screechy |
| 4997 | Utter Foolishness | This film is so pathetic and low budget with a script that has no plot | 3 | alinagoldenmeer |
| 4998 | Nicholas cage should be embarrassed | Over the top Cage in absurd gore movie | 1 | metahumanity_01 |
| 4999 | Don't waste your time | It is unusual for me to stop watching a movie half way, even if I really don't like it | 2 | walteriandolo |
| 5000 | Nic Cage loves pocket money | This is lazy and why movies and filmmaking is dying | 1 | paul_obe |

### *4.1.2. Data Preprocessing*

In **Figure 4.1** for Random Forest after getting the data, preprocessing the data is done. Each review is labeled positive and negative based on the rating. If the movie rating is >= 5 then it is labeled 1 which means it is positive, while the movie rating <= 5 is labeled 0 which means it is negative. In preprocessing the data, the author column is omitted because it does not have an effect on sentiment and adds a sentiment column that contains a label obtained from the rating. Here, the writer takes 20 samples from the dataset that is carried out by the preprocessing data, which can be seen in **Table 4.1.2.1**

*Table 4.1.2.1 Data Preprocessing*

| ID | Title | Review | Rating | Label |
|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 1 |
| 2 | Horse | The movie is great | 6 | 1 |
| 3 | Great White=Jaws Lite | The CGI is not great | 4 | 0 |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0 |
| 5 | Taxi | I do not like the terrible movie | 4 | 0 |
| 6 | A descent effort | It is great enough | 6 | 1 |
| 7 | Greenland | I like it | 7 | 1 |
| 8 | Mulan | The movie is good | 5 | 1 |
| 9 | A Quite Place II | It is bad | 3 | 0 |
| 10 | Spiderman | I do not like | 4 | 0 |
| 11 | Superman | The cgi is bad | 4 | 0 |
| 12 | Wonder Woman | It is great | 7 | 1 |

| ID | Title | Review | Rating | Label |
|---|---|---|---|---|
| 13 | Ava | It is good | 4 | 0 |
| 14 | Cinderella | The movie is nice | 7 | 1 |
| 15 | Combat | It is nice | 7 | 1 |
| 16 | Frozen | Not good | 4 | 0 |
| 17 | Tenet | Nice | 5 | 1 |
| 18 | Cruella | Like it | 6 | 1 |
| 19 | Fast & Furious | It is terrible | 4 | 0 |
| 20 | Twilight | Terrible movie | 7 | 1 |

### *4.1.3. TF-IDF (Term Frequency-Inverse Document Frequency)*

Then the TF-IDF calculation is carried out. TFI-DF is used to calculate the length of the text and produce results of greater relevance, also to convert text reviews into numeric or vector TFIDF and calculate the weight of each word feature. Here's the formula for calculating TF-IDF:

$$Tf = \frac{number\ of\ time\ term\ appears\ within\ document}{total\ number\ of\ document}$$

***Function 4.2.3: TF***

$$Idf = log\ \frac{total\ number\ of\ document}{number\ of\ words\ with\ term\ within\ document}$$

***Function 4.2.3.1: Idf***

$$Tf - Idf = Tf \times Idf$$

***Function 4.2.3.2: Tf-Idf***

TF-IDF presents vector text data based on a review. TF can be seen in **Function 4.2.3**, calculated by finding the number of occurrences of words in the data (per line) divided by the length of the sentence. IDF can be seen in **Function 4.2.3.1**, which serves to reduce the weight of a term if its occurrence is spread throughout the data, calculated by the log of the total data or document divided by the occurrence of words in the data. In Random Forest, the use of TF-IDF lowers the case for the review column and splits based on spaces named TempWord which can be seen in **Table 4.1.3.1.**

Then create a WordArray to count the number of the same words in 1 document, and a CountWord **Table 4.2.3.1** to count the number of occurrences of the same word in the data. The result is the implementation of the TF-IDF formula which can be seen in **Function 4.2.3.2**. The author will take 20 samples from preprocessed data to calculate TF-IDF. The author takes 20 data samples to clarify the calculation of TF-IDF, which can be seen in **Table 4.2.3.2**.

*Table 4.1.3.1 TempWord*

| **Iterasi** | **Lower case review column and split by space** |
|---|---|
| 1 | ‘i’, ‘do’, ‘not’, ‘get’, ‘the’, ‘terrible’, ‘reviews’ |
| 2 | ‘the’, movie, was, ‘great’ |
| 3 | ‘the’, ‘cgi’, ‘is’, ‘not’, ‘great’ |
| 4 | ‘the’, ‘movie’, ‘is’, ‘bad’ |
| 5 | ‘i’, “don’t”, ‘like’, ‘the’, ‘terrible’, ‘movie’ |
| 6 | ‘it’, ‘is’, ‘great’, ‘enough’ |

As can be seen in the table above, TempWord here lowers the case for the review column and splits based on spaces. The table is displayed according to the iteration or order of the data.

*Table 4.1.3.2 CountWord*

| Word | The number of the same word appears in the data | Word | The number of the same word appears in the data |
|---|---|---|---|
| i | 4 | great | 4 |
| do | 3 | cgi | 2 |
| not | 4 | bad | 3 |
| get | 1 | like | 4 |
| the | 8 | enough | 1 |
| terrible | 4 | it | 8 |
| reviews | 1 | good | 3 |
| movie | 6 | nice | 3 |
| is | 8 | | |

Next, the Count Word is calculated. Word count is obtained by counting the number of occurrences of the same word in the data. It can be seen in **Table 4.1.3.2.**

*Table 4.1.3.3 Calculation of TF-IDF*

| Word | Number of Word Appears | Total Number of Document | TF | IDF | TF-IDF (TF x IDF) |
|---|---|---|---|---|---|
| i | 4 | 20 | 4/20 = 0,2 | log(20/4) = 0,6990 | 0,1398 |
| do | 3 | 20 | 3/20 = 0,15 | log(20/3) = 0,8239 | 0,1236 |
| not | 4 | 20 | 4/20 = 0,2 | log(20/4) = 0,6990 | 0,1398 |
| get | 1 | 20 | 1/20 = 0,05 | log(20/1) = 1,3010 | 0,0652 |
| the | 8 | 20 | 8/20 = 0,4 | log(20/8) = 0,3979 | 0,1592 |
| terrible | 4 | 20 | 4/20 = 0,2 | log(20/4) = 0,6990 | 0,1398 |
| reviews | 1 | 20 | 1/20 = 0,05 | log(20/1) = 1,3010 | 0,0652 |

| Word | Number of Word Appears | Total Number of Document | TF | IDF | TF-IDF (TF x IDF) |
|---|---|---|---|---|---|
| movie | 6 | 20 | 6/20 = 0,3 | log(20/6) = 0,5229 | 0,1569 |
| is | 8 | 20 | 8/20 = 0,4 | log(20/8) = 0,3979 | 0,1592 |
| great | 4 | 20 | 4/20 = 0,2 | log(20/4) = 0,6990 | 0,1398 |
| cgi | 2 | 20 | 2/20 = 0,1 | log(20/2) = 1 | 0,1 |
| bad | 3 | 20 | 3/20 = 0,15 | log(20/3) = 0,8239 | 0.1236 |
| like | 4 | 20 | 4/20 = 0,2 | log(20/4) = 0,6990 | 0,1398 |
| enough | 1 | 20 | 1/20 = 0,05 | log(20/1) = 1,3010 | 0,0652 |
| it | 8 | 20 | 8/20 = 0,4 | log(20/8) = 0,3979 | 0,1592 |
| good | 3 | 20 | 3/20 = 0,15 | log(20/3) = 0,8239 | 0.1236 |
| nice | 3 | 20 | 3/20 = 0,15 | log(20/3) = 0,8239 | 0.1236 |

After the TempWord and CountWord are obtained, they are entered into the TF-IDF formula. In the **Table 4.1.3.4**, the TF-IDF calculation is based on the review column on the data. It can be seen that the word 'i' appears in the data 4 times with a total of 20 data. Then the TF is calculated by 4 divided by 20 so that the TF is 0.4. While the IDF log of 20 is divided by 4, so we get 0.6990. After obtaining the TF and IDF, the author can calculate the TF-IDF score, which is 0.4 multiplied by 0.6990 and the result is 0.1398.

*Table 4.1.3.5 TF-IDF Results*

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 1 | 0,11895 |
| 2 | What are all the bad reviews about is it  woke thing | It is great | 7 | 1 | 0,15273 |
| 3 | Cinderella | The movie is nice | 7 | 1 | 0,14733 |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0 | 0,14973 |
| 5 | Terrible story, dialogue, and CGI | I do not like the terrible movie | 4 | 0 | 0,12464 |
| 6 | A Descent Effort | It is great enough | 6 | 1 | 0,13085 |
| 7 | Greenland | I like it | 7 | 1 | 0,14627 |
| 8 | Mulan | The movie is good | 5 | 1 | 0,14973 |
| 9 | A Quite Place II | It is bad | 7 | 1 | 0,15273 |
| 10 | Spiderman | I do not like | 4 | 0 | 0,1344 |
| 11 | Superman | The cgi is bad | 4 | 0 | 0,1355 |
| 12 | Wonder Woman | The movie is great | 6 | 1 | 0,0996 |
| 13 | Ava | It is good | 4 | 0 | 0,14733 |
| 14 | Great White=Jaws Lite | The CGI is not great | 4 | 0 | 0,1396 |
| 15 | Combat | It is nice | 7 | 1 | 0,14733 |
| 16 | Frozen | Not good | 4 | 0 | 0,1317 |
| 17 | Tenet | Nice | 5 | 1 | 0,1236 |

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 18 | Cruella | Like it | 6 | 1 | 0,1495 |
| 19 | Fast & Furious | It is terrible | 4 | 0 | 0,15723 |
| 20 | Twilight | Terrible movie | 7 | 1 | 0,14835 |

In the **Table 4.1.3.6**, the results of the TF-IDF are obtained by adding up the results of the TF-IDF per word for 1 line review and then averaging.

For example for id 1: i', 'do', 'not', 'get', 'the', 'terrible', 'reviews'

Each of the words above has a TF-IDF result: 0.1398 , 0.1236, 0,1398, 0,0652, 0,1592, 0,1398, 0,1398

TF-IDF results for id 1 = (0.1398 + 0.1236 + 0.1398 +.0.0652 +0.1592 + 0.1398 + 0.1398) / 7

= 0,11895

This calculation applies to each ID.

### *4.1.4. Split Data*

After the TF-IDF calculation is obtained, the data is split, the data is divided into training data and data testing. Split data made 70% for training data and 30% for data testing. In split data, create 4 new arrays containing training data, training labels, testing data, testing labels. In the table below, the authors take 20 samples of data to be divided. 14 for training data and 6 for testing data. Training data and testing data will be displayed randomly.

*Table 4.1.4.1 Training Data*

| Rating | Score TF-IDF |
|---|---|
| 5 | 0,11895 |
| 6 | 0,15273 |
| 7 | 0,14733 |
| 4 | 0,14973 |
| 4 | 0,12464 |
| 6 | 0,13085 |
| 7 | 0,14627 |
| 5 | 0,14973 |
| 3 | 0,15273 |
| 4 | 0,1344 |
| 4 | 0,1355 |
| 6 | 0,0996 |
| 4 | 0,14733 |
| 4 | 0,1396 |

In the training data **Table 4.1.4.1,** the author uses 2 features, namely the TF-IDF rating and score. The TF-IDF score obtained in the table above is a calculation of 1 data or document. The author splits the data for training 70% of the dataset.

*Table 4.1.4.2 Training Label*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 5 | 0,11895 | 1 |
| 6 | 0,15273 | 1 |
| 7 | 0,14733 | 1 |
| 4 | 0,14973 | 0 |
| 4 | 0,12464 | 0 |
| 6 | 0,13085 | 1 |
| 7 | 0,14627 | 1 |
| 5 | 0,14973 | 1 |
| 3 | 0,15273 | 1 |
| 4 | 0,1344 | 0 |
| 4 | 0,1355 | 0 |
| 6 | 0,0996 | 1 |
| 4 | 0,14733 | 0 |
| 4 | 0,1396 | 0 |

After the training data is obtained, the training data is labeled with the name training label, which can be seen in **Table 4.1.4.2.** If the movie rating >= 5 then it is labeled 1 which means it is positive, while the movie rating <= 5 is labeled 0 which means it is negative.

*Table 4.1.4.3 Testing Data*

| Rating | Score TF-IDF |
|---|---|
| 7 | 0,14733 |
| 4 | 0,1317 |
| 5 | 0,1236 |
| 6 | 0,1495 |
| 4 | 0,15723 |
| 7 | 0,14835 |

*Table 4.1.4.4 Testing Label*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 7 | 0,14733 | 1 |
| 4 | 0,1317 | 0 |
| 5 | 0,1236 | 1 |
| 6 | 0,1495 | 1 |
| 4 | 0,15723 | 0 |
| 7 | 0,14835 | 1 |

In the testing data **Table 4.1.4.3**, 30% of the dataset is used. Just like the training data, the features in the testing data consist of 2 features, namely the TF-IDF rating and score. While in **Table 4.1.4.5**, testing data is labeled for each row.

### *4.1.5. Implementation of Random Forest Algorithm*

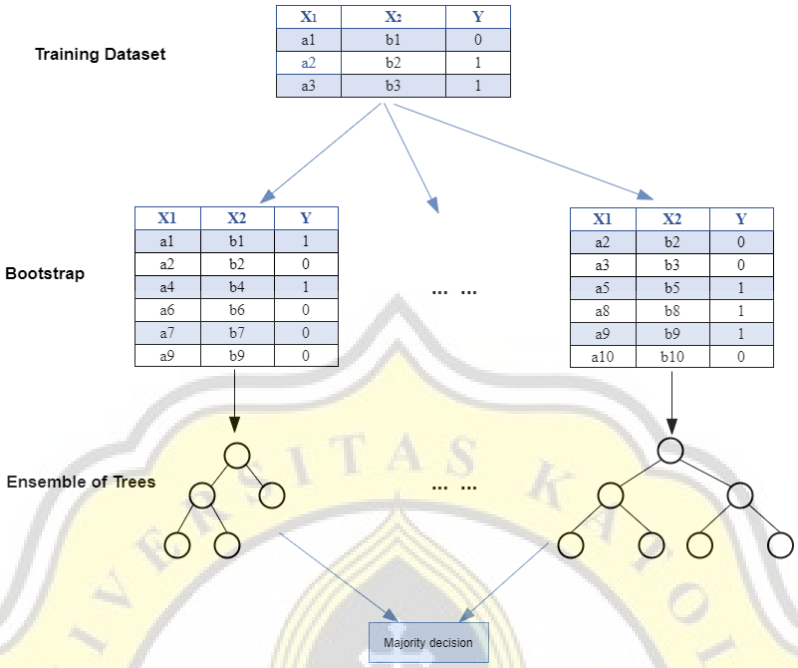


*Figure 4.1.5.1 Representation of Random Forest*

Based on **Figure 4.1.1** the next step is to implement the Random Forest algorithm. Its representation can be seen in **Figure 4.1.5.1** In the picture above, the training data has 2 variables, namely the TF-IDF rating and score. X1 for rating and X2 score for TF-IDF. While Y is a column label obtained from preprocessing data. Following are the steps for implementing the Random Forest algorithm:

1. First, a random sample is selected from 14 training data.

   In this project, a random sample selection was obtained by creating a bootstrap. The author took 20 samples randomly. Bootstrapping is done to train each tree in the Random Forest on subset observations or values. This bootstrap creation will calculate bootstrap data, bootstrap labels, OOB data, OOB labels, bootstrap indices, and OOB indices. Subsets or values that are not bootstrapped go into OOB. Bootstrap indices here take values randomly from the training data along with the training data which can be seen in **Table 4.1.5.1.**

*Table 4.1.5.1 Bootstrap Indices*

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 4 | 5 | 6 | 6 | 5 | 9 | 4 | 10 | 11 | 12 | 13 |

*Table 4.1.5.2 OOB Indices*

| | | | |
|---|---|---|---|
| 3 | 7 | 8 | 9 |

The value of bootstrap indices is taken randomly from the training data and throughout the training data. Whereas OOB indices take values that are not in bootstrap indices.

*Table 4.1.5.3 Bootstrap Data*

| **Rating** | **Score TF-IDF** |
|---|---|
| 5 | 0,11895 |
| 6 | 0,15273 |
| 4 | 0,1396 |
| 4 | 0,12464 |
| 6 | 0,13085 |
| 7 | 0,14627 |
| 7 | 0,14627 |
| 6 | 0,13085 |
| 4 | 0,1344 |
| 4 | 0,12464 |
| 4 | 0,1355 |
| 7 | 0,15273 |
| 6 | 0,0996 |
| 4 | 0,1396 |

In the bootstrap data **Table 4.1.5.3**, in this project, is an append of the training data, which means that the training data becomes the benchmark to get the bootstrap data value. The length of the bootstrap data is based on the bootstrap indices. The previous bootstrap indices were 14 lines long, so there are 14 lines in the bootstrap data. The values contained in the bootstrap data are based on the array order of the training data, so the bootstrap indices are known [0, 1, 2, 4, 5, 6, 6, 5, 4, 9, 10, 11, 12, 13] the first number is 0, then the first row of bootstrap data contains the first row of training data.

*Table 4.1.5.4 Bootstrap Label*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 5 | 0,11895 | 1 |
| 7 | 0,15378 | 1 |
| 4 | 0,1396 | 0 |
| 4 | 0,12464 | 0 |
| 6 | 0,13085 | 1 |
| 7 | 0,14627 | 1 |
| 7 | 0,14627 | 1 |
| 6 | 0,13085 | 1 |
| 4 | 0,1344 | 0 |
| 4 | 0,12464 | 0 |
| 4 | 0,1355 | 0 |
| 7 | 0,15273 | 1 |
| 6 | 0,15378 | 0 |
| 7 | 0,14733 | 1 |

The bootstrap data is labeled in a bootstrap label, based on the training label. Value 1 for positive and value 0 for negative. Based on the bootstrap data table, the bootstrap label can be seen in **Table 4.1.5.4.**

*Table 4.1.5.5 OOB Data*

| Rating | Score TF-IDF |
|---|---|
| 4 | 0,14973 |
| 5 | 0,14973 |
| 3 | 0,15273 |
| 4 | 0,1344 |

*Table 4.1.5.6 OOB Label*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 4 | 0,14973 | 0 |
| 5 | 0,14973 | 1 |
| 3 | 0,15273 | 0 |
| 4 | 0,1344 | 0 |

The previous OOB indices were 4 lines long, so there are 4 lines in the OOB data. The values contained in the OOB data are based on the array order of the training data, so the OOB indices are known [3, 7, 8, 9], then the OOB data contains the fourth row of the training data. With this step 1 has been completed.

2. Second, training and building a decision tree for each sample from bootstrap data with steps as below:
   a. Split Point

      After calculating the bootstrap, a split point calculation is performed to find the child value with the highest Information Gain. This is done by choosing a value from the bootstrap randomly, each selected value is iterated and the Information Gain calculation is performed. The value with the highest Information Gain represents a node in the tree containing IDX features, value, left child node, and right child node. The split point count uses bootstrap data, bootstrap label, and max features parameters. Max features are features used in this project, there are 2, namely rating and results from TF-IDF. The split point uses the ls feature, which is 1 or 0, and the value of the IDX feature comes from the ls feature which is obtained randomly.

      The split point value is obtained from the IDX+1 feature (this IDX feature is obtained randomly from the ls feature). While the value is obtained from the IDX + 1 feature. There are 2 conditions to form a child node, namely:

      1) If the value <= the value of the split point it will form a left child node.
      2) If the value >= the value of the split point will form a right child node.

      The left and right children initially contain bootstrap data and an empty bootstrap label.

      Left child : bootstrap data [ ], bootstrap label [ ]

      Right child : bootstrap data [ ], bootstrap label [ ]

      The following is a calculation based on a sample from bootstrap data:

      - For feature IDX = 1 (out of 2 features, rating and TF-IDF, if Feature IDX = 1 which is the benchmark for TF-IDF), it has 2 loops:
        - Looping 1:

          Feature IDX = 1

          Value = Feature IDX 1+1 = 2 (takes the lowest 2 data from the training data) = 0,14733

          Split point = Feature IDX 1+1= 2 (take the top 2 data from the training data) = 0,11895

Value 0,14733 > Split point 0,11895, then form a right child node.

Feature IDX = 1

Value = 0,0996

Split point = 0,11895

Value 0,0996 < Split point 0,11895, then form a left child node.

*Table 4.1.5.7 Contents of Child in Feature IDX = 1 in Loop 1*

| Child | Input |
|---|---|
| Left child | bootstrap data: [6 0,0996]<br>bootstrap label: [1] |
| Right child | bootstrap data: [4 0,14733]<br>bootstrap label: [0] |

**Table 4.1.5.7** above shows the contents of the left child and right child based on Feature IDX = 1 when looping 1, with bootstrap data and bootstrap label input.

- Looping 2:

  Feature IDX = 1

  Value = 0,14733

  Split point = 0,15273

  Value 0,14733 < Split point 0,15273, then form a left child node.

  Feature IDX = 1

  Value = 0,0996

  Split point = 0,15273

  Value 0,0996 < Split point 0,15273, then form a left child node.

*Table 4.1.5.8 Contents of Child in Feature IDX = 1 in Loop 2*

| Child | Input |
|---|---|
| Left child | bootstrap data: [4 0,14733], [6 0,0996]<br>bootstrap label: [0, 1] |

**Table 4.1.5.8** above shows the contents of the left child based on Feature IDX = 1 when looping 2, with bootstrap data input and bootstrap label.

- For feature IDX = 0 (out of 2 features, rating and TF-IDF, if Feature IDX = 0 is the rating benchmark), it has 1 looping:

  Feature IDX = 0

  Value = 6

  Split point = 5

  Value 6 > Split point 5, then form a right child node.

*Table 4.1.5.9 Contentes of Child in Feature IDX = 0*

| Child | Input |
|---|---|
| Right child | bootstrap data: [6 0,0996]<br>bootstrap label: [1] |

**Table 4.1.5.9** above shows the contents of the left child and right child based on Feature IDX = 0, with input bootstrap data and bootstrap label.

b. Information Gain (IG) is calculated using entropy, takes the class list from the left and right children, and returns the information gain from a certain separation. Information Gain calculates how much information is obtained when splitting nodes at a certain value, also to determine the next branch.

From the results of the split point count, Information Gain is calculated for each loop of the IDX feature that was previously obtained. Here's the calculation:

- Looping 1:

  Feature IDX = 1

  Value = 0,14733

  Split point = 0,11895

  Value 0,14733 > Split point 0,11895, then form a right child node.

  Feature IDX = 1

  Value = 0,0996

  Split point = 0,11895

  Value 0,0996 < Split point 0,11895, then form a left child node.

  Parent: obtained by combining the left child and right child, then the parent [1, 0] can be seen in **Table 4.1.5.7.**

  Left child: [1]

  Right child: [0]

$$pParent = \frac{sum\ of\ number\ 1\ in\ parent}{length\ of\ parent}$$

***Function 4.2.5: Probability of Parent***

$$pParent = \frac{1}{2} = 0.5$$

$$H(X) = -p \log_2 p - (1-p) \log_2 (1-p)$$

***Function 4.2.5.1: Entropy***

$$IG(parent) = entropy(pParent)$$

***Function 4.2.5.2: Information Gain of Parent***

$$IG(parent) = -0.5 \log_2 0.5 - (1 - 0.5) \log_2 (1 - 0.5) = -0.1506$$

pParent = probability of parent

p = probability

IG (parent) = Information Gain of parent

Entropy (pParent) = entropy of parent probability

In calculating pParent using the entropy formula that can be seen in **Function 4.2.5**, the probability is taken from the parent. Likewise for pLeft and pRight, taking the probabilities of left and right.

$$pLeft = \frac{sum\ of\ number\ 1\ in\ left\ child}{length\ of\ left\ child}$$

*Function 4.2.5.3: Probability of Left Child*

$$pLeft = \frac{1}{1} = 1$$

$$IG(left) = entropy(pLeft)$$

*Function 4.2.5.4: Information Gain of Left Child*

$$IG(left) = -1 \log_2 1 - (1 - 1) \log_2 (1 - 1) = -0.3011$$

pLeft = probability of left child

p = probability

IG (left) = Information Gain of left child

Entropy (pLeft) = entropy of left child probability

After obtaining the left child's Information Gain, then calculate the IG of the right child.

$$pRight = \frac{sum\ of\ number\ 1\ in\ right\ child}{length\ of\ right\ child}$$

*Function 4.2.5.5: Probability of Right Child*

$$pRight = \frac{1}{1} = 1$$

$$IG(right) = entropy(pRight)$$

*Function 4.2.5.6: Information Gain of Right Child*

$$IG(right) = -1\log_2 1 - (1-1)\log_2(1-1) = -0.3011$$

pRight = probability of right child

p = probability

IG (right) = Information Gain of right child

Entropy (pRight) = entropy of right child probability

$$IG = IG(parent) - \frac{N_{left}}{N_p} \times IG(left) - \frac{N_{right}}{N_p} \times IG(right)$$

***Function 4.2.5.7: I nformation Gain***

$$IG = (-0.1506) - \left(\frac{1}{1} \times -0.3011\right) - \left(\frac{1}{1} \times -0.3011\right) = 0.4516$$

After getting the Information Gain value in loop 1, the information gain split is done to find the best Information Gain value. Best Info Gain is made to -999 first, and if the new IG calculation is greater than the best info gain, the previous best info gain will be replaced with the new value. Here's the calculation:

Best info gain = - 999

Information gain = 0.4516

Condition = information gain > best info gain = 0.4516 > -999

After that, immediately create a node as the root node to build a random tree:

Node 1 = Infomation Gain : 0.4516

Left child: [6 0,0996]

Right child: [4 0,14733]

Split point: 0,11895

Feature IDX: 1

Root Node 1 = Infomation Gain : -0.1506

Left child: [6 0,0996]

Right child: [4 0,14733]

Split point: 0,11895

Feature IDX: 1

- For the next loop, the IG is calculated in the same way as in loop 1.

  Looping 2:

  Feature IDX = 1

  Value = 0,14733

  Split point = 0,15273

  Value 0,14733 < Split point 0,15273, then form a left child node.

  Feature IDX = 1

  Value = 0,0996

  Split point = 0,15273

  Value 0,0996 < Split point 0,15273, then form a left child node.

  Parent: [0, 1]

  Left child: [0, 1]

  Right child: [ ]

  To calculate the IG, the author first calculates the probabilities of the parent in **Function 4.2.5**, left child in **Function 4.2.5.3**, and right child in **Function 4.2.5.5**. After that calculates the IG of each new calculates the entire IG.

$$pParent = \frac{1}{2} = 0.5$$

After obtaining the parent probability, the next step is to calculate the entropy in **Function 4.2.5.1** of pParent and then calculate the IG of the parent which can be seen in **Function 4.2.5.2**.

$$IG(parent) = -0.5 \log_2 0.5 - (1 - 0.5) \log_2 (1 - 0.5) = -0.1506$$

Next, calculate the IG of the left child in **Function 4.2.5.4**.

$$pLeft = \frac{1}{2} = 0.5$$

$$IG(left) = -0.5 \log_2 0.5 - (1 - 0.5) \log_2 (1 - 0.5) = -0.1506$$

Then calculate the overall Information Gain which can be seen in **Function 4.2.5.7**.

$$IG = (-0.1506) - \left(\frac{1}{2} \times -0.1506\right) - (0) = -0.0753$$

Best info gain = 0.4516

Information gain= -0.0753

Condition = information gain > best info gain = 0.4516 > -0.0753

Node 2 = Infomation Gain : 0.4516

Left child: [4 0,14733], [6 0,0996]

Right child: [ ]

Split point: 0,15273

Feature IDX: 1

Root Node 2 = Infomation Gain : 0.4516

Left child: [4 0,14733], [6 0,0996]

Right child: [ ]

Split point: 0,15273

Feature IDX: 1

- Looping 3:

  For feature IDX = 0, has 1 loop:

  Feature IDX = 0

  Value = 6

  Split point = 5

  Value 6 > Split point 5, then form a right child node.

  Parent:[1]

  Right child: [1]

  Same as before, calculate the IG of the parent first which can be seen in **Function 4.2.5.2**.

$$pParent = \frac{1}{2} = 0.5$$

$$IG(parent) = -0.5 \log_2 0.5 - (1 - 0.5) \log_2 (1 - 0.5) = -0.1506$$

  Then calculate the IG of the right child which can be seen in **Function 4.2.5.6**.

$$pRight = \frac{1}{2} = 0.5$$

$$IG(right) = -0.5 \log_2 0.5 - (1 - 0.5) \log_2 (1 - 0.5) = -0.1506$$

  Because the 3rd node has no left child, the left child is 0.

$$IG = (-0.1506) - (0) - \left(\frac{1}{2} \times -0.1506\right) = -0.0753$$

  Best info gain = 0.4516

  Information gain = -0.0753

  Condition = information gain > best info gain = 0.4516 > -0.0753

Node 3 = Infomation Gain : 0.4516

Left child: [ ]

Right child: [6 0,0996]

Split point: 5

Feature IDX: 0

Root Node 3 = Infomation Gain : 0.4516
Left child: [ ]
Right child: [6 0,0996]
Split point: 5
Feature IDX: 0

After getting the node, the author does Split Node to create a branch to the left child or right child. Split Node is obtained using node parameters, max features, min sample split, max depth, and depth. The max features in this project are 2, namely the TF-IDF rating and score. For max depth which is the maximum value of tree depth, in this project, the max depth is 10. Min sample split is the minimum number of samples needed to divide internal nodes, in this project the min sample split is 2. While depth is the depth of the tree that will be created later. For split nodes, there are 6 conditions:

1) If one child is empty, it will create an empty child.
   Empty child is obtained from left child [bootstrap label] + right child [bootstrap label].
2) If tree depth >= max depth, then the left node splits into the terminal node of the left child. Likewise for the right split.
3) If the length of the left child of bootstrap data <= min sample split, then the left split node is filled with the terminal node of the left child.
4) If the length of the left child of bootstrap data >= min sample split, it will calculate the split point count of the new left child and repeat steps a and b.
5) If the length of the right child of the bootstrap data <= min sample split, then the right split node is filled with the terminal node of the right child.
6) If the length of the right child of the bootstrap data <= min sample split, then the right split node is filled with the terminal node of the right child.

Previously, 3 nodes were obtained, then split the nodes:

1) Node 1 = Information Gain: 0.4516
Left child: Bootstrap data: [6 0,0996]
Bootstrap label: [1]
Right child: Bootstrap data: [4 0,14733]
Bootstrap label: [0]
Split point: 0,11895
Feature IDX: 1

Min sample split = 2

Node 1 is entered into condition no 3
Then the split node:
Length of left child 1 <= min sample split 2

Node 1 = Information Gain: 0.4516
Left child: Terminal node (left split) = [1], [0]
Right child: -
Split point: 0,11895
Feature IDX: 1

Delete the child node first, because it will be filled with a new value. Then the child are converted into terminal nodes (left split).

After the split node, the author determines the terminal node for each node. The terminal node will calculate how many numbers 1 and 0, the maximum number will be the final result of the terminal node.

Terminal node 1:

*Table 4.1.5.10 Terminal node 1*

| Node | Terminal Node |
|---|---|
| 1 | bootstrap label: [1] |

Terminal node 1 contains bootstrap label [1], in the table split node 1 terminal node is filled with label 1 first and then filled with label 0. Label 1 is entered first, then terminal node 1 is labeled 1.

After the terminal node is obtained, the next step is to calculate the OOB Score. OOB Score calculates miss labels from predict tree of each node and OOB data. Predict tree has not been obtained, so the author calculates the predicted tree first. Predict tree is obtained by comparing the testing data and split point tree based on the IDX Feature that has been obtained by each node previously. Because there are 6 testing data, then a comparison is made from each split point with each line of testing data. The calculation of the predicted tree and OOB Score is carried out for each node.

Predict Tree Node 1:

Feature IDX = 1 (take TF-IDF score)

- Comparison 1:

  Testing data = 0,14733

*Table 4.1.5.11 Testing Data*

| Rating | Score TF-IDF |
|---|---|
| 7 | 0,14733 |
| 4 | 0,1317 |
| 5 | 0,1236 |
| 6 | 0,1495 |
| 4 | 0,15723 |
| 7 | 0,14835 |

Split point: 0,11895

Condition: testing data (Feature IDX) <= tree ((split point) (FeatureIDX)) return to left split, and vice versa.

0.14733 >= 0.11895 then it return into the right split.

- Comparison 2:

  Testing data = 0,1317

  Split point: 0,11895

  0,1317 >= 0,11895 then it return into the right split.

- Comparison 3:

  Testing data = 0,1236

  Split point: 0,11895

  0,1236 >= 0,11895 then it return into the right split.

- Comparison 4:

  Testing data = 0,1495

  Split point: 0,11895

  0,1495 >= 0,11895 then it return into the right split.

- Comparison 5:

  Testing data = 0,15723

  Split point: 0,11895

  0,15723 >= 0,11895 then it return into the right split.

- Comparison 6:

  Testing data = 0,14835

  Split point: 0,11895

  0,14835 >= 0,11895 then it return into the right split.

From 6 comparisons at node 1, the Predict Tree label is obtained from each comparison based on split point = 5. The value of 5 is a rating, in preprocessing data if rating >= 5 then it is labeled 1 or positive. Then the label Predict Tree Node 2 = [1, 1, 1, 1, 1, 1].

After getting the Predict Tree, the next step is to calculate the OOB Score.

The calculation of the OOB Score is obtained by calculating the missing label from the split point root node and OOB data (if the labels are not the same, then the missing label is counted 1).

*Table 4.1.5.12 OOB Data*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 4 | 0,14973 | 0 |
| 5 | 0,14973 | 1 |
| 3 | 0,15273 | 0 |
| 4 | 0,1344 | 0 |

*Table 4.1.5.13 Split Point from Node 1*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 5 | 0,11895 | 1 |

OOB Score node 1: compares the split node labels from node 1 and the testing data (per row).

*Table 4.1.5.14 OOB Score from Node 1*

| | |
|---|---|
| Label 1 | 0 |
| | 1 |
| | 0 |
| | 0 |

Miss label: 3 (there is only 1 label from the split point which is the same as the label from testing data, while there are 3 that are not the same, then the missing label is 3).

So the OOB Score: 3/6 = 0,5.

2) Node 2 = Information Gain: 0.4516

Left child: Bootstrap data: [4 0,14733], [6 0,0996]

Bootstrap label: [0, 1]

Right child: [ ]

Split point: 0,15273

Feature IDX: 1

Node 2 is entered into condition no 1

Then the split node:

| | | |
|---|---|---|
| Node 2 = | Information Gain: | 0.4516 |
| | Left child: | Terminal node (empty child) |
| | Right child: | Terminal node (empty child) |
| | Split point: | 0,15273 |
| | Feature IDX: | 1 |

Delete the child node first, because it will be filled with a new value. Because one child is empty, it will create a terminal node (empty child).

Terminal node 2:

*Table 4.1.5.15 Terminal node 2*

| Node | Terminal Node |
|---|---|
| 2 | bootstrap label: [0] |

Terminal node 2 contains bootstrap label [0], in the split table node 2 because it does not have a right child, it creates a terminal node (empty child) which contains the label of the left child. The terminal node is filled with label 0 first, then label 1. Label 0 is entered first, then terminal node 2 is labeled 0.

Predict Tree Node 2:

- Comparison 1:

  Feture IDX = 1 (takes TF-IDF score)

  Testing data = 0,14733

  Split point = 0,15273

  0,14733 <= 0,15273 then it return into the left split.

- Comparison 2:

  Testing data = 0,1317

  Split point = 0,15273

  0,1317 <= 0,15273 then it return into the left split.

- Comparison 3:

  Testing data = 0,1236

  Split point = 0,15273

  0,1236 <= 0,15273 then it return into the left split.

- Comparison 4:

  Testing data = 0,1495

  Split point = 0,15273

  0,1495 >= 0,15273 then it return into the right split.

- Comparison 5:

  Testing data = 0,15723

  Split point = 0,15273

  0,15723 >= 0,15273 then it return into the right split.

- Comparison 6:

  Testing data = 0,14835

  Split point = 0,15273

  0,14835 <= 0,15273 then it return into the left split.

From 6 comparisons at node 2, the Predict Tree label is obtained from each comparison based on split point = 5. The value of 5 is a rating, in preprocessing data if rating >= 5 then it is labeled 1 or positive. Then the label Predict Tree Node 2 = [1, 1, 1, 1, 1, 1].

Same as in node 1, after calculating the Predict Tree, the OOB Score is calculated.

*Table 4.1.5.16 Split Point from Node 2*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 7 | 0,15273 | 1 |

OOB Score Node 2:

*Table 4.1.5.17 OOB Score from Node 2*

| | |
|---|---|
| Label 1 | 0 |
| | 1 |
| | 0 |
| | 0 |

Miss label: 3

So the OOB Score: 3/6 = 0,5.

3) Node 3 = Information Gain: 0.4516
Left child: [ ]
Right child: Bootstrap data: [6 0,0996]
Bootstrap label: [1]
Split point: 5
Feature IDX: 0

Node 3 is entered into condition no 1

Then the split node:

Node 3 = Information Gain: 0.4516
Left child: Terminal node (empty child)
Right child: Terminal node (empty child)
Split point: 5
Feature IDX: 1

Delete the child node first, because it will be filled with a new value. Because one child is empty, it will create a terminal node (empty child).

Terminal node 3:

*Table 4.1.5.18 Terminal Node 3*

| Node | Terminal Node |
|---|---|
| 3 | bootstrap label: [1] |

Terminal node 3 contains a bootstrap label [1], in the split table node 3 because does not have a left child value, it creates a terminal node (empty child). The empty child is the result of a right child which has label 1. Then terminal node 3 is labeled 1.

Predict Tree Node 3:

- Comparison 1:

  Feture IDX = 0 (takes the TF-IDF score)

  Testing data = 7

  Split point = 5

  7 >= 5 then it return into the right split.
- Comparison 2:

  Testing data = 4

  Split point = 5

  4 <= 5 then it return into the left split.
- Comparison 3:

  Testing data = 5

  Split point = 5

  5 <= 5 then it return into the left split.
- Comparison 4:

  Testing data = 6

  Split point = 5

  6 >= 5 then it return into the right split.

- Comparison 5:

  Testing data = 4

  Split point = 5

  4 <= 5 then it return into the left split.

- Comparison 6:

  Testing data = 7

  Split point = 5

  7 >= 5 then it return into the right split.

From 6 comparisons at node 3, the Predict Tree label is obtained from each comparison based on split point = 5. The value of 5 is a rating, in preprocessing data if rating >= 5 then it is labeled 1 or positive. Then the label Predict Tree Node 3 = [1, 1, 1, 1, 1, 1].

After getting the Predict Tree, the OOB Score is calculated.

*Table 4.1.5.19 Split Point from Node 3*

| Rating | Score TF-IDF | Label |
|---|---|---|
| 5 | 0,11895 | 1 |

*Table 4.1.5.20 OOB Score from Node 3*

| | |
|---|---|
| Label 1 | 0 |
| | 1 |
| | 0 |
| | 0 |

Miss label: 3

So the OOB Score: 3/6 = 0.5.

From the three nodes above, tree_ls is formed, which combines the three nodes.

*Table 4.1.5.21 Tree_ls*

| Node | Rating | Score TF-IDF | Label |
|---|---|---|---|
| 1 | 5 | 0,11895 | 1 |
| 2 | 7 | 0,15273 | 1 |
| 3 | 5 | 0,11895 | 1 |

c. The next step is to find predict Random Forest (RF), with parameter tree_ls and testing data. In predict RF there are 2 predictions, namely Ensemble Prediction and Final Prediction. Ensemble Prediction is a combination of the results of the previous Predict Tree. While Final Prediction counts the number of labels 1 or 0 in the Ensemble Prediction. The final prediction results are based on the majority voting of the Final Prediction.

   Ensemble Prediction = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

   Final Prediction = sum of numbers 1 = 18

   Majority Vote = 1

   The majority voting is 1, which means that the sentiment analysis on the movie review by implementing the Random Forest algorithm produces sentiments that tend to be positive (1).

3. Finally, evaluate the Random Forest algorithm. Here the author evaluates the model by calculating the accuracy. Which can be seen in **Function 4.2.5.8**, by matching the predicted results from the predictions with the testing label (looking for how many results are the same) then summed and divided by the length of the testing label.

$$Accuracy = \frac{number\ of\ match\ label\ from\ predicted\ label\ with\ the\ testing\ label}{length\ of\ testing\ label}$$

*Function 4.2.5.8: Accuracy*

## 4.2. Logistic Regression

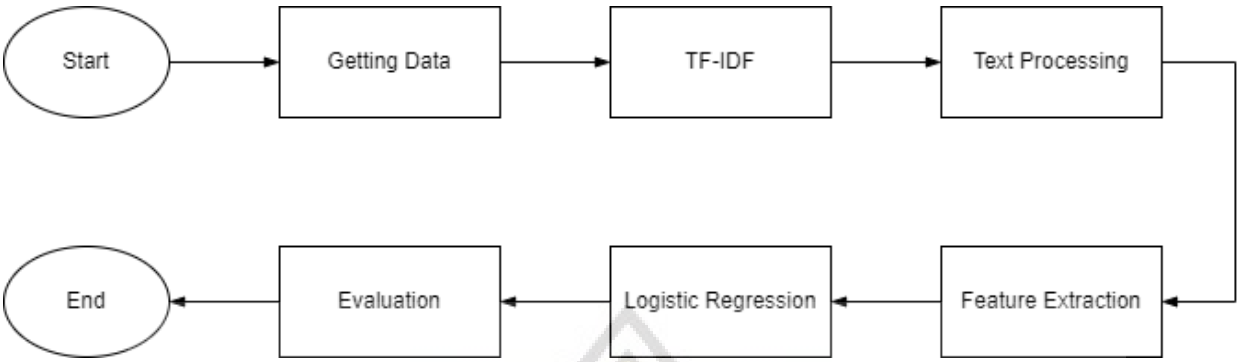


*Figure 4.2.1 Logistic Regression Design Scheme*

As seen in **Figure 4.2**, there are 6 steps for Random Forest. First getting data, then the TF-IDF calculation is done for each word. Then text processing, labeling, and data split are done here. After that, make a matrix for the feature extraction. Then implementation of Random Forest algorithm, and finally evaluation.

### *4.2.1. Getting Data*

According to **Figure 4.2.1,** first getting data is done. The data used in the form of the same dataset as Random Forest can be seen in **Table 4.2.1.1.**

*Table 4.2.1.1 Logistic Regression Dataset*

| ID | Title | Review | Rating | Author |
|---|---|---|---|---|
| 1 | Not Bad | I don't get the terrible reviews | 5 | margarida-44311 |
| 2 | What are all the bad reviews about is it a woke thing | The movie was great | 6 | joemay-2 |
| 3 | Great White=Jaws Lite | The CGI is not great | 4 | nebk |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | kuarinofu |

| ID | Title | Review | Rating | Author |
|---|---|---|---|---|
| 5 | Terrible story, dialogue, and CGI | I don’t like the terrible movie | 4 | Horror_Flick_Fanatic |
| ... | ..... | ..... | ..... | ..... |
| 4996 | What are all the bad reviews about is it a woke thing | Its poorly written, very poorly directed, poorly scripted | 8 | Jim_Screechy |
| 4997 | Utter Foolishness | This film is so pathetic and low budget with a script that has no plot | 3 | alinagoldenmeer |
| 4998 | Nicholas cage should be embarrassed | Over the top Cage in absurd gore movie | 1 | metahumanity_01 |
| 4999 | Don't waste your time | It is unusual for me to stop watching a movie half way, even if I really don't like it | 2 | walteriandolo |
| 5000 | Nic Cage loves pocket money | This is lazy and why movies and filmmaking is dying | 1 | paul_obe |

### 4.2.2. *TF-IDF (Term Frequency-Inverse Document Frequency)*

Then the TF-IDF calculation is carried out in the same way as in the Random Forest which can be seen in **Table 4.2.2.1**, as many as 20 data samples.

*Table 4.2.2.1 TF-IDF Logistic Results*

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 0,11895 |
| 2 | What are all the bad reviews about is it woke thing | It is great | 7 | 0,15273 |
| 3 | Cinderella | The movie is nice | 7 | 0,14733 |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0,14973 |
| 5 | Terrible story, dialogue, and CGI | I do not like the terrible movie | 4 | 0,12464 |
| 6 | A Descent Effort | It is great enough | 6 | 0,13085 |
| 7 | Greenland | I like it | 7 | 0,14627 |
| 8 | Mulan | The movie is good | 5 | 0,14973 |
| 9 | A Quite Place II | It is bad | 7 | 0,15273 |
| 10 | Spiderman | I do not like | 4 | 0,1344 |
| 11 | Superman | The cgi is bad | 4 | 0,1355 |
| 12 | Wonder Woman | The movie is great | 6 | 0,0996 |
| 13 | Ava | It is good | 4 | 0,14733 |
| 14 | Great White=Jaws Lite | The CGI is not great | 4 | 0,1396 |
| 15 | Combat | It is nice | 7 | 0,14733 |
| 16 | Frozen | Not good | 4 | 0,1317 |
| 17 | Tenet | Nice | 5 | 0,1236 |
| 18 | Cruella | Like it | 6 | 0,1495 |

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 19 | Fast & Furious | It is terrible | 4 | 0,15723 |
| 20 | Twilight | Terrible movie | 7 | 0,14835 |

### 4.2.3. Text Processing

Next is text processing, labeling based on the rating on the dataset. If the rating is >= 5 then it is labeled 1 which means it is positive, while the movie rating <= 5 is labeled 0 which means it is negative. The labeling is directly added in the dataset column. Here the labeling is classified as positive and negative and is named Positive Data and Negative Data.

*Table 4.2.3.1 Positive Data*

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 1 | 0,11895 |
| 2 | What are all the bad reviews about is it woke thing | It is great | 7 | 1 | 0,15273 |
| 3 | Cinderella | The movie is nice | 7 | 1 | 0,14733 |
| 6 | A Descent Effort | It is great enough | 6 | 1 | 0,13085 |
| 7 | Greenland | I like it | 7 | 1 | 0,14627 |
| 8 | Mulan | The movie is good | 5 | 1 | 0,14973 |
| 9 | A Quite Place II | It is bad | 7 | 1 | 0,15273 |
| 12 | Wonder Woman | The movie is great | 6 | 1 | 0,0996 |
| 15 | Combat | It is nice | 7 | 1 | 0,14733 |
| 17 | Tenet | Nice | 5 | 1 | 0,1236 |
| 18 | Cruella | Like it | 6 | 1 | 0,1495 |

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 20 | Twilight | Terrible movie | 7 | 1 | 0,14835 |

*Table 4.2.3.2 Negative Data*

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0 | 0,14973 |
| 5 | Terrible story, dialogue, and CGI | I do not like the terrible movie | 4 | 0 | 0,12464 |
| 10 | Spiderman | I do not like | 4 | 0 | 0,1344 |
| 11 | Superman | The cgi is bad | 4 | 0 | 0,1355 |
| 13 | Ava | It is good | 4 | 0 | 0,14733 |
| 14 | Great White=Jaws Lite | The CGI is not great | 4 | 0 | 0,1396 |
| 16 | Frozen | Not good | 4 | 0 | 0,1317 |
| 19 | Fast & Furious | It is terrible | 4 | 0 | 0,15723 |

In text processing, the author column is omitted because it is not needed in this project. Then in text processing, a data split is done. The data is divided into training data and testing data. Split data made 70% for training data from positive data and negative data and 30% for testing data from positive data and negative data. In split, data create 4 new arrays containing training data, training labels, testing data, testing labels. Training and testing data were selected randomly based on positive and negative data. Here's the training and testing:

*Table 0.3 Training Data Positive*

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 0,11895 |
| 2 | What are all the bad reviews about is it woke thing | It is great | 7 | 0,15273 |
| 3 | Cinderella | The movie is nice | 7 | 0,14733 |
| 6 | A Descent Effort | It is great enough | 6 | 0,13085 |
| 7 | Greenland | I like it | 7 | 0,14627 |
| 8 | Mulan | The movie is good | 5 | 0,14973 |
| 9 | A Quite Place II | It is bad | 7 | 0,15273 |
| 12 | Wonder Woman | The movie is great | 6 | 0,0996 |

*Table 0.4 Training Data Negative*

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0,14973 |
| 5 | Terrible story, dialogue, and CGI | I do not like the terrible movie | 4 | 0,12464 |
| 10 | Spiderman | I do not like | 4 | 0,1344 |
| 11 | Superman | The cgi is bad | 4 | 0,1355 |
| 13 | Ava | It is good | 4 | 0,14733 |

*Table 0.5 Training Label*

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 1 | Not Bad | I do not get the terrible reviews | 5 | 1 | 0,11895 |
| 2 | What are all the bad reviews about is it woke thing | It is great | 7 | 1 | 0,15273 |
| 3 | Cinderella | The movie is nice | 7 | 1 | 0,14733 |
| 6 | A Descent Effort | It is great enough | 6 | 1 | 0,13085 |
| 7 | Greenland | I like it | 7 | 1 | 0,14627 |
| 8 | Mulan | The movie is good | 5 | 1 | 0,14973 |
| 9 | A Quite Place II | It is bad | 7 | 1 | 0,15273 |
| 12 | Wonder Woman | The movie is great | 6 | 1 | 0,0996 |
| 4 | Bare-bones killer shark film | The movie is bad | 4 | 0 | 0,14973 |
| 5 | Terrible story, dialogue, and CGI | I do not like the terrible movie | 4 | 0 | 0,12464 |
| 10 | Spiderman | I do not like | 4 | 0 | 0,1344 |
| 11 | Superman | The cgi is bad | 4 | 0 | 0,1355 |
| 13 | Ava | It is good | 4 | 0 | 0,14733 |

From the positive training data and negative training data, they are combined to be given a label which becomes a training label which can be seen in **Table 4.2.3.5.**

*Table 0.6 Testing Data Positive*

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 15 | Combat | It is nice | 7 | 0,14733 |
| 17 | Tenet | Nice | 5 | 0,1236 |
| 18 | Cruella | Like it | 6 | 0,1495 |
| 20 | Twilight | Terrible movie | 7 | 0,14835 |

*Table 0.7 Testing Data Negative*

| ID | Title | Review | Rating | TF-IDF |
|---|---|---|---|---|
| 14 | Great White=Jaws Lite | The CGI is not great | 4 | 0,14733 |
| 16 | Frozen | Not good | 4 | 0,1236 |
| 19 | Fast & Furious | It is terrible | 4 | 0,1495 |

*Table 0.8 Testing Label*

| ID | Title | Review | Rating | Label | TF-IDF |
|---|---|---|---|---|---|
| 15 | Combat | It is nice | 7 | 1 | 0,14733 |
| 17 | Tenet | Nice | 5 | 1 | 0,1236 |
| 18 | Cruella | Like it | 6 | 1 | 0,1495 |
| 20 | Twilight | Terrible movie | 7 | 1 | 0,14835 |
| 14 | Great White=Jaws Lite | The CGI is not great | 4 | 0 | 0,14733 |
| 16 | Frozen | Not good | 4 | 0 | 0,1236 |
| 19 | Fast & Furious | It is terrible | 4 | 0 | 0,1495 |

From testing positive data and testing negative data, they are combined to be given a label which becomes a testing label which can be seen in **Table 4.2.3.8.**

Furthermore, based on positive-negative training data and positive-negative data testing, the calculation of positive training counts, negative training counts, positive testing counts, and negative testing counts is carried out. The calculation grouped positive negative training and positive-negative testing itself. The positive training count counts the amount of data that is labeled positively on the training data, while the negative training count counts the amount of data that is labeled negatively on the training data. Positive count testing counts the number of positive data on testing data and negative count testing counts on the number of negative data on testing data. Here are the results of count training and count testing:

Count Training Positif = 8

Count Training Negatif = 5

Count Testing Positif = 4

Count Testing Negatif = 3

### 4.2.4. *Feature Extraction*

In feature extraction, Word Dict calculations are performed, namely grouping positive data and negative data. To count each the same number of words in positive reviews and negative reviews. Data is lower case and split by space. The first dictionary is created, namely, posWord which contains the number of occurrences of words in positive data, and the second negWord dictionary contains the number of occurrences of words in negative data. For more details, see the table below:

*Table 4.2.4.1 Word Dict*

| Word | The number of the same word appears in the data | Total Words on Positive Data (1) | Total Words on Negative Data (0) |
|---|---|---|---|
| i | 4 | 2 | 2 |
| do | 3 | 1 | 2 |
| not | 4 | 1 | 3 |
| get | 1 | 1 | - |
| the | 8 | 4 | 4 |
| terrible | 4 | 2 | 2 |
| reviews | 1 | 1 | - |
| movie | 6 | 4 | 2 |
| is | 8 | 7 | 5 |
| great | 4 | 2 | 1 |
| cgi | 2 | - | 2 |
| bad | 3 | 1 | 2 |
| like | 4 | 2 | 2 |
| enough | 1 | 1 | - |
| it | 8 | 6 | 2 |
| good | 3 | 1 | 2 |
| nice | 3 | 2 | - |

In the table above, is the number of each word that appears in the positive data and negative data. The number of occurrences of the word is stored as a dictionary type. For example, [nice, 1] = 3, [good, 1] = 2, [nice, 0] = 2.

Furthermore, the calculation of feature extraction is carried out. Because computers do not deal with text, and only understand the language of numbers. So the author converts the data into vectors so that it can be entered into the logistic regression algorithm. The data is converted into a 3-dimensional (X) vector containing, rating, TF-IDF score, posWord dictionary (number of positive words in positive data for each id), and negWord dictionary (number of negative words in negative data for each id).

For example id 1: 'i do not get the terrible reviews'

| | i | do | not | get | the | terrible | reviews |
|---|---|---|---|---|---|---|---|
| Number of word in positive data | 2 | 1 | 1 | 1 | 4 | 2 | 1 |
| Number of word in negative data | 2 | 2 | 3 | 0 | 4 | 2 | 0 |

Total occurrence of words in positive data = 13

Total occurrence of words in negative data = 13

The matrix has a rating, TF-IDF in it, the number of words in the positive data, the number of words in the negative data.

Matrix id 1 = [5 0.11895 13 13]

This calculation is repeated for each id based on the training data. Then the results of the X matrix from the training data:

X = [[5 0.11895 13 13], [7 0,15273 15 8], [7 0,14 733 17 11], [6 0,13085 16 8], [7 0,14627 10 6], [8 0,149733 16 13], [7 0,15273 14 9], [6 0,0996 17 12], [4 0,14973 16 13], [4 0,12464 16 17], [4 0,1344 6 9], [4 0,1355 12 13], [4 0,14733 14 9]]

### 4.2.5. *Implementation of Logistic Regression Algorithm*

The next step according to the design is the implementation of the Logistic Regression algorithm. The intercept, xData, yData, and weight calculations are used for the next step. Then the calculations for intercept, xData, yData, and weight are as follows:

a. The intercept is initialized to 1 throughout the training data, the length of the training data is 13 data. So it is made 13 to the right.

   Intercept = [1 1 1 1 1 1 1 1 1 1 1 1 1]

b. xData combines intercept vertically and X.

   xData = [[1 5 0.11895 13 13]
   [1 7 0,15273 15 8]
   [1 7 0,14733 17 11]
   [1 6 0,13085 16 8]
   [1 7 0,14627 10 6]
   [1 8 0,14973 16 13]
   [1 7 0,15273 14 9]
   [1 6 0,0996 17 12]
   [1 4 0,14973 16 13]
   [1 4 0,12464 16 17]
   [1 4 0,1344 6 9]
   [1 4 0,1355 12 13]
   [1 4 0,14733 14 9]]

c. Weight is set to 0 (as many as 5) in length according to the number of contents of the matrix in 1 row and will be updated later.

   Weight = [0 0 0 0 0]

d. yData is obtained from training labels.

yData = [1 1 1 1 1 1 1 1 0 0 0 0 0]

In the next step, the author creates a function for the training model that requires parameters LR (Learning Rate), iteration, intercept, xData, weight, and yData. This function is used to calculate sigmoid, loss, and gradient descent. Later it will also update the weight to fit into the data. The new weight is used to predict the label. This function is used for each iteration, here is the implementation of 1 iteration:

1st iteration:

First calculate the Sigmoid:

The sigmoid function returns a value from 0 to 1. If the output is greater than or equal to 0.5, it is classified as positive data. Otherwise, it is classified as negative data. Sigmoid depiction can be seen in **Figure 4.2.5.1.**

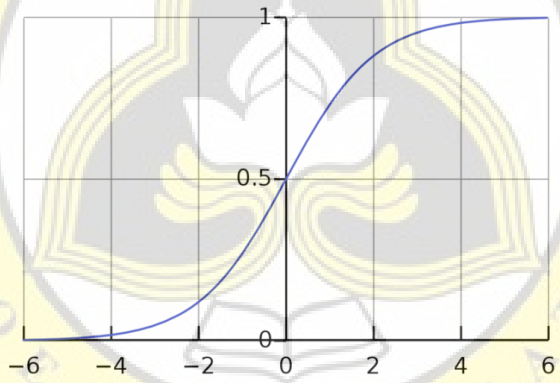


*Figure 0.1 Curve of Sigmoid Function*

In the graph above, it will return a value between 0 to 1. If the value of x (z) goes to positive infinity then the predicted value of y will be 1 and if it goes to negative infinity then the predicted value of y becomes 0.

Find the z-value first:

$$z = x \times w$$

***Function 4.2.5: Z***

X = xData

W = weight

Then the z value:

Z = [0 0 0 0 0 0 0 0 0 0 0 0 0]

After getting the z value, implement it to the sigmoid formula:

$$sigmoid = \frac{1}{1 + e^{-z}}$$

***Function 4.2.5.1: Sigmoid***

Sigmoid calculation:

$$sigmoid = \frac{1}{1 + e^{-(0)}} = 0.5$$

Sigmoid = [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0. 5 0.5]

The next step is to calculate the loss value, find the optimal parameters, and minimize errors from the model to get the best predictions. The function for count loss can be seen in **Function 4.2.5.2**.

Loss :

$$loss = [((-y \log h)) - ((1 - y) \log 1 - h))]$$

***Function 4.2.5.2: loss***

h = sigmoid

y = yData

Calculate the log h along the sigmoid, then the result is:

log h = log0.5

= [-0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718]

After getting log h, implementation to loss formula:

Loss = ([-1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0] * [-0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718 -0.69314718]) – ([1-1 1-1 1-1 1-1 1-1 1-1 1-1 1-1 1-0 1-0 1-0 1- 0 1-0] * [1-(-0.69314718) 1-( -0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-( -0.69314718) 1-( -0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-(-0.69314718) 1-(-0.69314718)])

= ([0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718]) – ([0 0 0 0 0 0 0 0 1 1 1 1 1] * [1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718])

= ([0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718]) – ([0 0 0 0 0 0 0 0 1.69314718 1.69314718 1.69314718 1.69314718 1.69314718])

= [0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 0.69314718 -1 -1 -1 -1 -1]

Then look for gradient descent which updates the weights by minimizing loss. The gradient descent formula can be seen in **Function 4.2.4**, the gradient descent formula is obtained from the loss derivative.

Gradient Descent :

$$gradient\ descent = \frac{(h - y).X^T}{len(y)}$$

***Function 4.2.5.3: Gradient Descent***

h = sigmoid

y = yData

len(y) = lenght of yData

$X^T$ = xData transpose

Calculate the value of (h-y) first:

h-y = ([0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]) – ([1 1 1 1 1 1 1 1 0 0 0 0 0])

= [-0. 5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 0.5 0.5 0.5 0.5 0.5]

Then transpose matrix xData:

$X^T$ = [[1 1 1 1 1 1 1 1 1 1 1 1 1]

[5 7 7 6 7 8 7 6 4 4 4 4 4]

[0.11895 0,15273 0,14733 0,13085 0,14627 0,14973 0,15273 0,0996 0,14973 0,12464 0,1344 0,1355 0,14733]

[13 15 17 16 10 16 14 17 16 16 6 12 14]

After that, just implement the gradient descent formula which can be seen in **Function 4.2.5.3**.

Gradient Descent = $\frac{[-1.2 \quad -16.5 \quad -0.203295 \quad -27 \quad -9]}{13}$

= [-0.9231 -1.2692 -0.0156 -2.0769 -0.6923]

Then gradient descent is obtained, the weight update is performed. The new weight is used to predict the label. This update weight is also calculated for each iteration.

Update Weight :

$$nWeight = oWeight - lr\,.gradient\ descent$$

***Function 4.2.5.4: Update Weight***

nWeight = new weight

oWeight = old weight

lr = learning rate

So the update weight calculation:

oWeight = [0 0 0 0 0]

Lr = 0.1

Gradient Descent = [-0.9231 -1.2692 -0.0156 -2.0769 -0.6923]

Lr * Gradient Descent = [-0.09231 -0.12692 -0.00156 -0.20769 -0.06923]

Update Weight = ([0 0 0 0 0]) – ([-0.09231 -0.12692 -0.00156 -0.20769 -0.06923])

= [0.09231 0.12692 0.00156 0.20769 0.06923]

The next step is to predict the class label. Calculate the new sigmoid first using the new weight. The sigmoid function can be seen in **Function 4.2.5.1**.

New Sigmoid:

Z = xData * nWeight

X = xData

xData = [[1 5 0.11895 13 13]

[1 7 0,15273 15 8]

[1 7 0,14733 17 11]

[1 6 0,13085 16 8]

[1 7 0,14627 10 6]

[1 8 0,14973 16 13]

[1 7 0,15273 14 9]

[1 6 0,0996 17 12]

[1 4 0,14973 16 13]

[1 4 0,12464 16 17]

[1 4 0,1344 6 9]

[1 4 0,1355 12 13]

[1 4 0,14733 14 9]]

New Weight = [0.09231 0.12692 0.00156 0.20769 0.06923]

Z = [[1 5 0.11895 13 13]

[1 7 0,15273 15 8]

[1 7 0,14733 17 11]

[1 6 0,13085 16 8]

[1 7 0,14627 10 6] * [0.09231 0.12692 0.00156 0.20769 0.06923]

[1 8 0,14973 16 13]

[1 7 0,15273 14 9]

[1 6 0,0996 17 12]

[1 4 0,14973 16 13]

[1 4 0,12464 16 17]

[1 4 0,1344 6 9]

[1 4 0,1355 12 13]

[1 4 0,14733 14 9]]

Z = [[4.32706 4.65018 5.27324 4.73092 3.47326 5.33094 4.51172 5.21548
4.82326 5.10014 2.46941 3.99248 4.13095]]

The sigmoid calculation for the first value of z:

$$s = \frac{1}{1 + e^{-(4.32706)}} = 0.98696582$$

Then the result of the sigmoid (result before):

Result Before = [[0.98696582 0.99053064 0.99489918 0.99125873
0.99125873 0.96991728 0.99518379 0.98913968
0.99459752 0.9920236 0.99394104 0.92196933
0.98188048 0.98418648]]

After that, find the result after from the result before with a threshold. If result before >= threshold then True, if result before <= threshold then False.

Threshold = 0.5 (dibuat 0.5)

Result After = [[True True True True True True True True True True
True True True True]]

The next step is to make a label prediction, the value is made 0 first as long as the result before.

Label Prediction = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0]]

Then compare the Result After with the predicted label. If the result after 'True', then it is labeled 1, if 'False' then it is labeled 0.

So the final result of the label prediction:

Prediction Label = [[1 1 1 1 1 1 1 1 1 1 1 1 1 1]]

The prediction label is all 1, then the sentiment analysis on the movie review using the Logistic Regression algorithm is more inclined to positive sentiment, the same as Random Forest.

The last step is the evaluation of the model, calculating accuracy by checking the similarity of yData and predicting the label of the percentage of the same number divided by the length of yData.

$$Accuracy = \frac{number\ of\ match\ label\ from\ predicted\ label\ with\ the\ testing\ label}{length\ of\ testing\ label}$$

*Function 4.2.5.5: Accuracy*

So all steps for Random Forest and Logistic Regression algorithm have been completed. From the implementation of the two algorithms, it is found that both have sentiments that tend to be positive. Then to evaluate the model, calculate the accuracy of the Random Forest and Logistic Regression algorithms, perform an accuracy comparison. The algorithm that has the greatest accuracy is the best algorithm and is suitable for this case.