**MIS798 Special Study**

**Classifying Amazon products as Defective / Non-defective Based on Customers' reviews using Natural Language Processing**

**Report By: Submitted To:**

Parul Jain (RedId : 825191465) Prof. David Golberg

Nishu Singh (RedId: 826991926)

Umadevi Betageri (RedId: 827194401)

**Abstract**

Natural language processing (NLP) is a branch of artificial intelligence that focuses on analyzing and understanding human language. It can be used to understand customer reviews on Amazon by analyzing the text of the review and extracting important information and insights. One way to use NLP in this context is to train a machine learning algorithm on a large dataset of customer reviews. The algorithm can be trained to identify common patterns and trends in the text of the reviews.

This information can be used by Amazon to improve the quality of its products and customer satisfaction. For example, if a large number of customers are mentioning a particular issue with a product, Amazon can use this information to identify and address the problem. Additionally, by understanding the sentiment of customer reviews, Amazon can track its overall reputation and identify areas where it can improve. Overall, using NLP can help Amazon gain valuable insights from customer reviews and improve its business. This research aims to classify Amazon products as defective or non-defective based on customer reviews using a machine learning algorithm.

**Introduction**

Classification is a technique used in machine learning to identify which category or class an item belongs to. In the context of Amazon reviews, classification could be used to predict whether a review is positive or negative. This can be useful for identifying the overall sentiment of customer reviews and understanding how customers feel about a particular product. The algorithm can be trained on a large dataset of customer reviews, where each review is labeled as either defective or non-defective. The training data can be collected from Amazon's website, where customers provide feedback on the products they have purchased.

Once the model is trained, it can be used to classify new customer reviews as defective or non-defective. To do this, the algorithm will analyze the text of the review using Natural Language processing and identify features that are indicative of a defective product. For example, if a review contains words like "broken," "defective," or "faulty," the algorithm may be more likely to classify it as defective. On the other hand, if a review contains words like "good," "great," or "excellent," the algorithm may be more likely to classify it as non-defective.

The algorithm can then make a prediction for each new review, and the results can be used to identify defective products and take appropriate action, such as recalling the product or offering a refund to the customer. By using machine learning, this approach can quickly and accurately classify large numbers of reviews, allowing Amazon to improve the quality of its products and customer satisfaction.

**Dataset Description**

The Datasets used in the project contain the reviews for 4 appliances from Amazon’s Appliances category - Blenders, Coffee Makers, Slow Cookers and Toaster Ovens. Apart from the original columns, an additional ‘Defect’ column is created to classify the data as defective. This column was manually added by Prof. Goldberg and his team for academic projects and was shared with us to do the research. All the datasets have the same columns and data structure therefore the datasets are merged together to give a final dataframe with 51k rows.

| **Column** | **Dtype** | **Description** |
| --- | --- | --- |
| DataSet Entry ID | int64 | Unique Id for every review |
| Text | object | Review Text |
| Tagger Pid | object | ID of the person who reviewed the product |
| Date | object | Review entry Date |
| Defect | object | No Defect, Performance Defect, Safety Hazard |
| Components Mentioned | object | Components of the appliance |
| Comment | object | Comment about the components |
| Authority? | bool | Boolean value with True/False |

*Table 1: Final Dataset Columns*

**Test data description**

A new dataset was created to test the best performing model on the unseen data. A set of reviews were fetched from the API [1]. To measure the performance of the model on this dataset, 300 reviews were manually labeled. To label the data as defective, only performance defects were taken into consideration and any issue concerning delivery or safety was ignored. The original data only had 3 fields - Rating, Product\_id and review\_text. The manually created column was named ‘Output’. The Table below shows the structure of the final test data.

| **Column** | **Dtype** | **Description** |
| --- | --- | --- |
| Rating | int64 | Rating out of 5 |
| Product\_ID | object | Unique ID of product that is reviewed |
| Review\_Text | object | Review |
| Output | object | Defect, No Defect |

*Table 2: Test Dataset Columns*

**Methodologies**

**Data Preprocessing**

Unstructured text data is one of the most important sources of information in the modern world. Text analytics applications include collecting thoughts from social media data, discovering customer satisfaction from product evaluations or feedback, and more. To find insights from text data i.e., reviews in our case, we need to preprocess the data before we train the classification or regression models. Text preprocessing includes removing stop words, numbers, URLs’, punctuations etc.,

Since “Defect” is a dependent feature in our dataset and our focus is to classify the products as defective or non-defective, we kept only “No Defect” and “Performance Defect” data by removing “Safety Hazard” reviews.

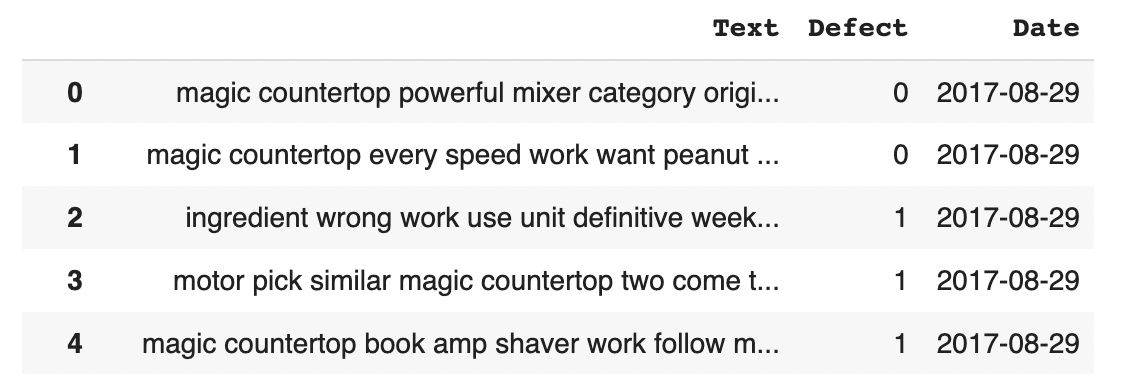
In the next step we created some python functions which take raw text as input and prepare data for vector creations. The functions are as follows:

* **remove\_url**: It will remove URLs’ from the text.
* **remove\_numbers**: Removes any numbers from the text.
* **remove\_non\_ascii**: Removes non-ASCII characters from list of tokenized words.
* **to\_lowercase:** Convert all characters to lowercase from a list of tokenized words.
* **remove\_punctuation**: Remove punctuation from list of tokenized words.
* **remove\_stopwords:** Remove stop words from list of tokenized words.
* **lemmatize\_text**: Does lemmatization of the tokenized words.
* **remove\_oneandtwo\_letter\_word**: Removes one letter word from list of tokenized words



*Figure 1: Flowchart for Data Preprocessing steps*

The flowchart above shows the steps involved in data preprocessing. Initially we removed URLs and numbers from the review text and tokenized text into words. Later we removed non-ASCII characters, punctuations, stopwords, one and two letter words from tokenized words. Then we converted all letters of tokenized words to lowercase letters, performed lemmatization and joined the lemmatized words. Finally, we removed unwanted columns, dropped null values and assigned 0 to “No Defect” and 1 to “Defect” reviews. Our preprocessed data looks as follows.

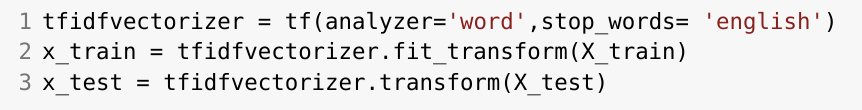


*Figure 2: Sample data after preprocessing*

**Creating vectors**

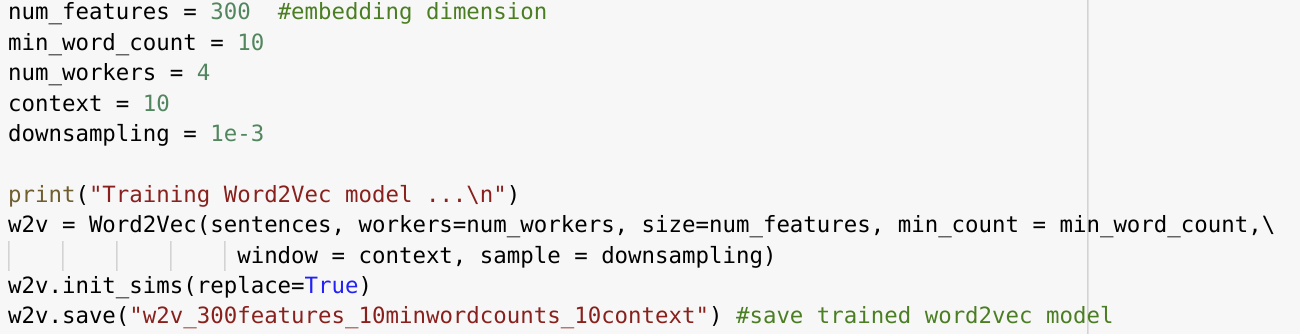
*Vectorization* is a technique used to turn text input into numerical data. There are various techniques to create vectors, we’ve used TF-IDF(Term frequency-inverse document frequency) and the word2vec.

The **TfidfVectorizer** method is a highly popular approach that converts text into meaningful numerical representations that may be used to fit machine prediction algorithms. This is phase 2 of our project. In this phase, we read the data that has been saved in the data preprocessing phase and split the data into train and test sets for input which is “Text” and output variable which is “Defect”. Then applied the fit\_transform function on train input data and the transform function on test input data.



*Figure 3: Code for Tf-idf Vectorizer*

**Word2Vec** This method transforms words into corresponding vectors using deep learning and neural network-based approaches such that the vectors that are semantically related are close to one another in n-dimensional space. In this project, we selected 300 dimensions.

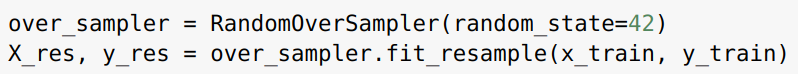


*Figure 4: Code for Word2Vec Vectorizer*

**Data Balancing Techniques**

As the dataset we are using for this project is imbalanced. We have applied three data balancing techniques to balance the dataset.

* Using Random over sampler: The technique is used to upsample the minority class.



* Using SMOTE (Synthetic Minority Oversampling Technique): This technique is used to add duplicate records of the minority class. SMOTE creates new instances by synthesizing the data already available.



* Using ADASYN: This technique is similar to SMOTE. Despite all of the sample's linear correlation to the parent, they all have a little bit more variation, or are a little bit dispersed.



**Model Building**

After creating vectors, our next task is to build a model. We have used the following machine learning algorithms to train the models:

1. Multinomial Naive Bayes

2. Logistic Regression

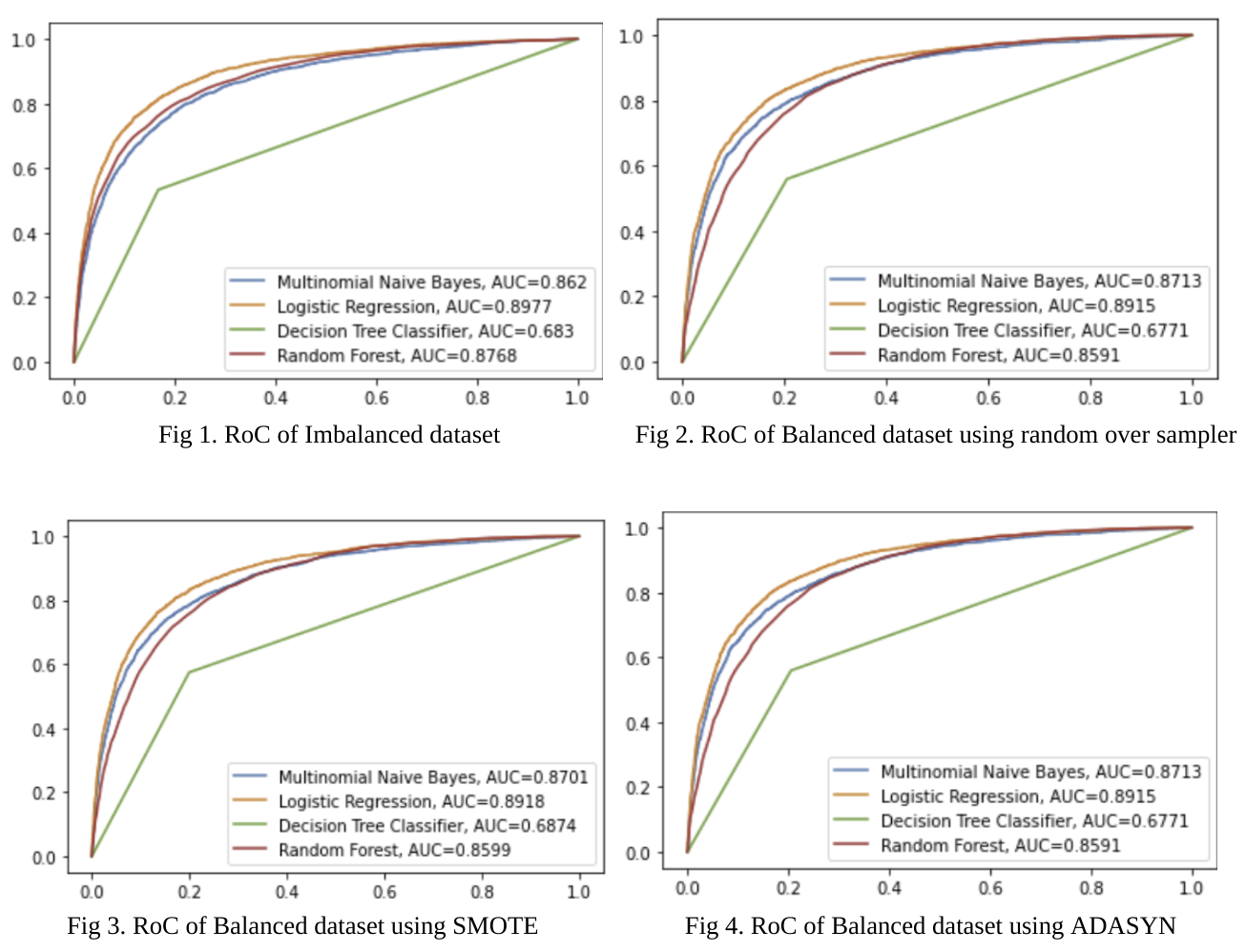
3. Decision Tree Classifier

4. Random Forest Classifier

We built a model by using an imbalanced dataset and a balanced dataset. Using an Imbalanced dataset, the best-performing model is by using Logistic Regression with an accuracy of 85%. The accuracy of models varies depending on the balancing techniques. However, a model trained using Logistic Regression is best performing among all balancing techniques.

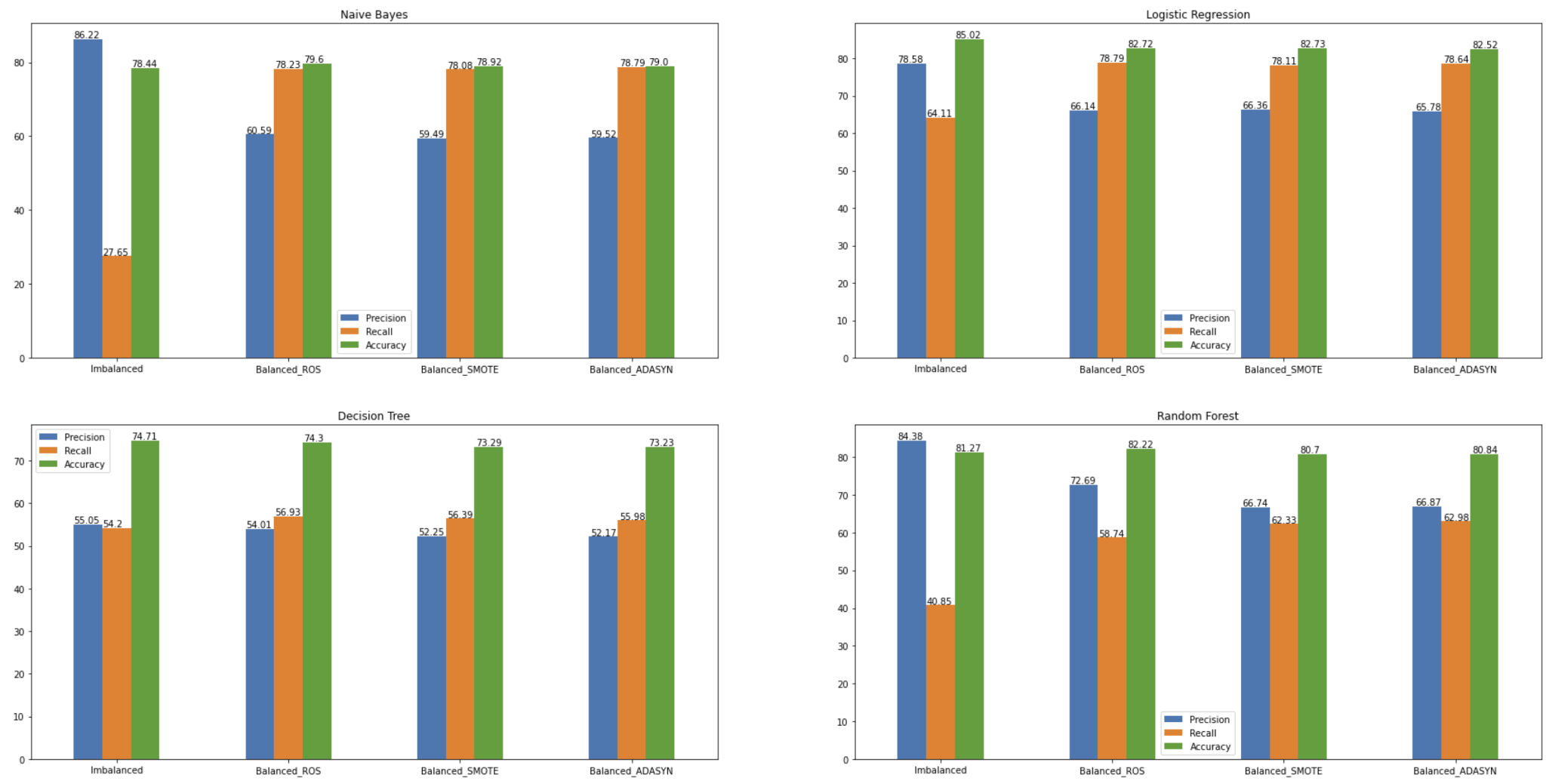
**Model Performance and Visualization**

We’ve created a receiver operating characteristic curve (ROC) curve to measure the model performance. The RoC curve is a graph that displays how well a classification model performs across all classification thresholds.



*Figure 5: ROC for different models with balanced and imbalance Data*

We created a bar plot to analyze which balancing techniques work well with the model. According to figure 5, we are getting better results using the “Random over Sampler”.



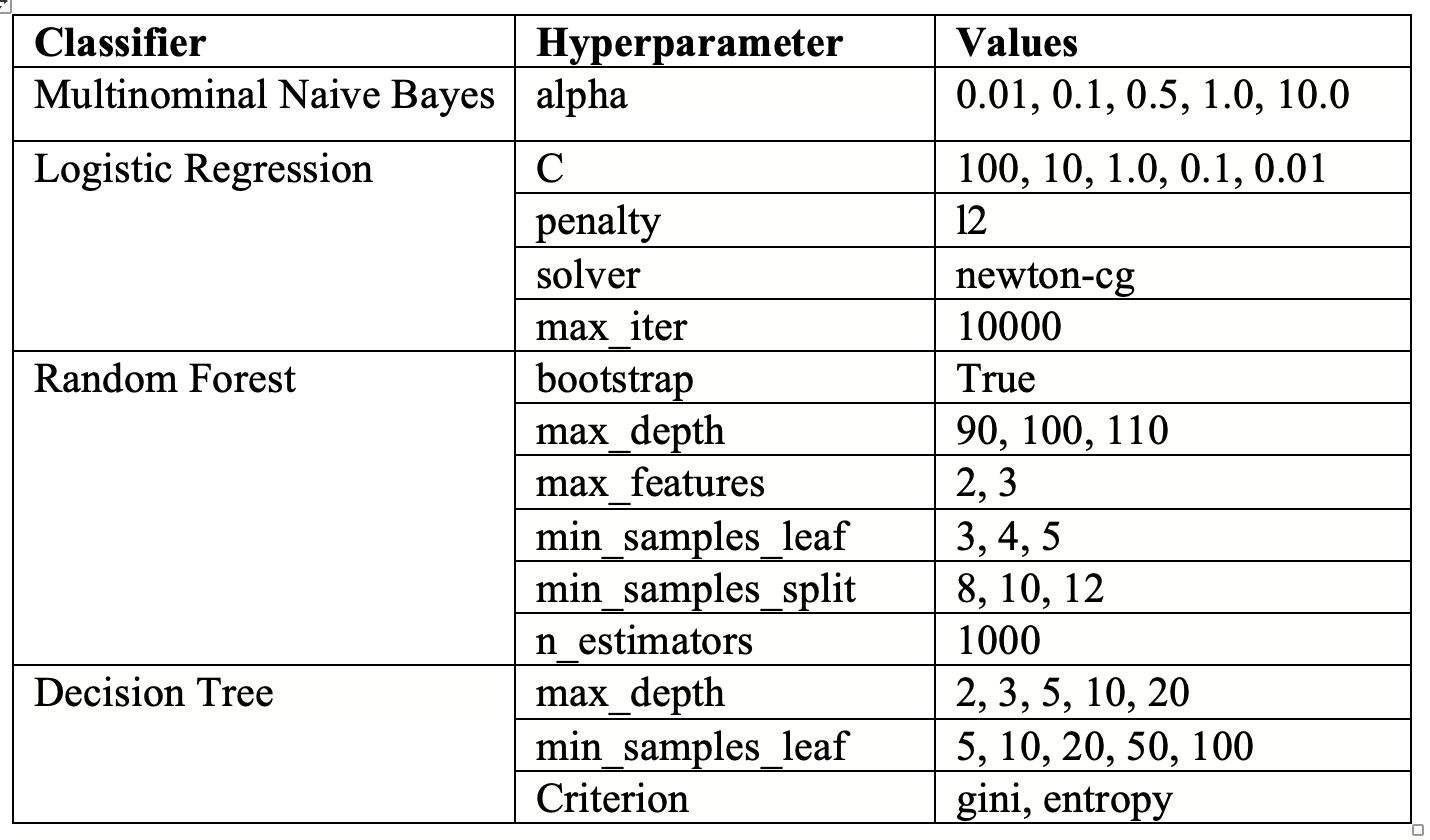
*Figure 6: Accuracy, Precision and Recall for Models with Balanced and Imbalanced Data*

**Hyperparameter Tuning**

Finding the optimal combination of hyperparameters to improve the model's performance is known as hyperparameter tuning (or hyperparameter optimization). Most of the time our predicted model parameters will yield worse outcomes if our hyperparameters aren't properly tuned to minimize the loss function. This indicates that our model has some flaws. Hence, to get optimal results of a machine learning model, hyperparameter tuning is required. It works by conducting multiple trials within a single training procedure. Every trial involves the complete execution of our training application with the values of the selected hyperparameters set within the predetermined bounds. Once this procedure is complete, we get the set of hyperparameter values for which the model performs at its best.

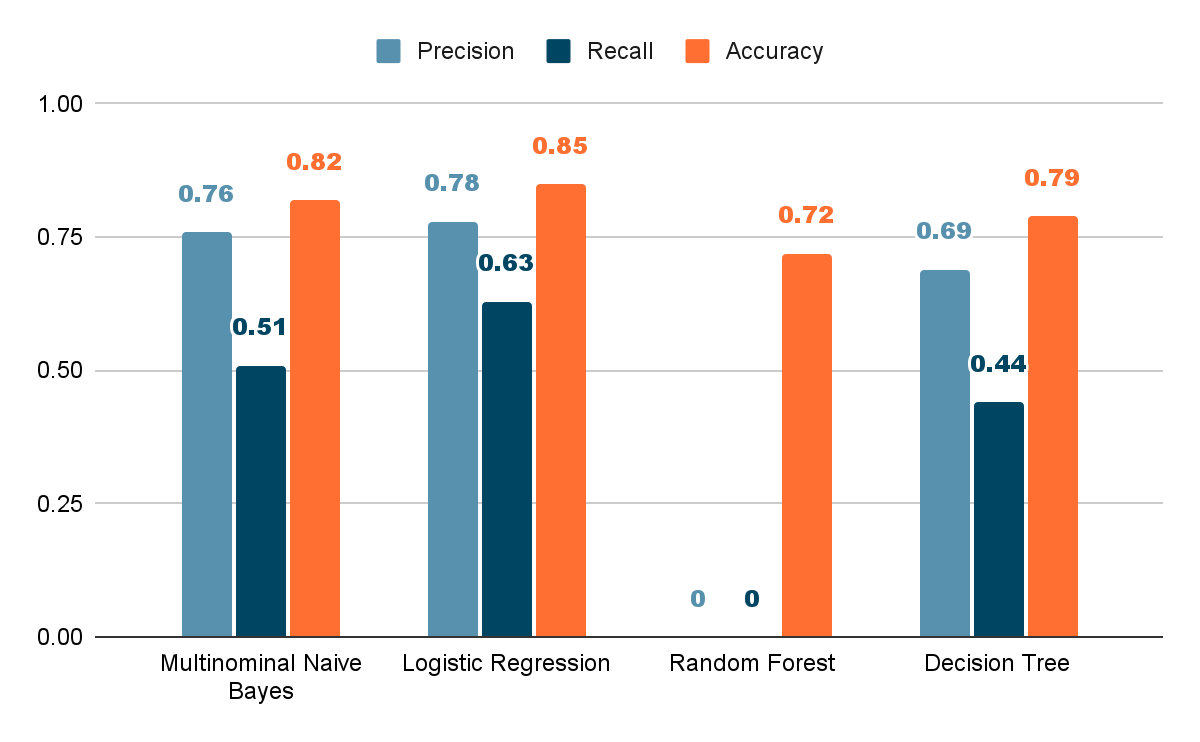
There are several methods of tuning hyperparameters. Random Search and Grid Search are widely used hyperparameter tuning techniques. In random Random Search we construct a grid of potential hyperparameter values. Each iteration tries a different random combination of these hyperparameters, evaluates the results, and then returns the set of hyperparameters that produced the best results. Whereas, in Grid Search we create a grid of potential hyperparameter values. Each iteration tries a set of hyperparameters in a particular sequence. It records the model performance while fitting the model with every conceivable set of hyperparameters. The best model with the best hyperparameters is then returned.

In this project, to further improve the model performance, we used the grid search method for tuning hyperparameters for the models mentioned above after creating the vectors. We tried with the hyperparameters shown in Table-1 for Multinomial Naive Bayes, Logistic Regression, Decision Tree and Random Forest classifiers. The function designed for grid search technique will try all the values provided for hyperparameters and returns the best one.

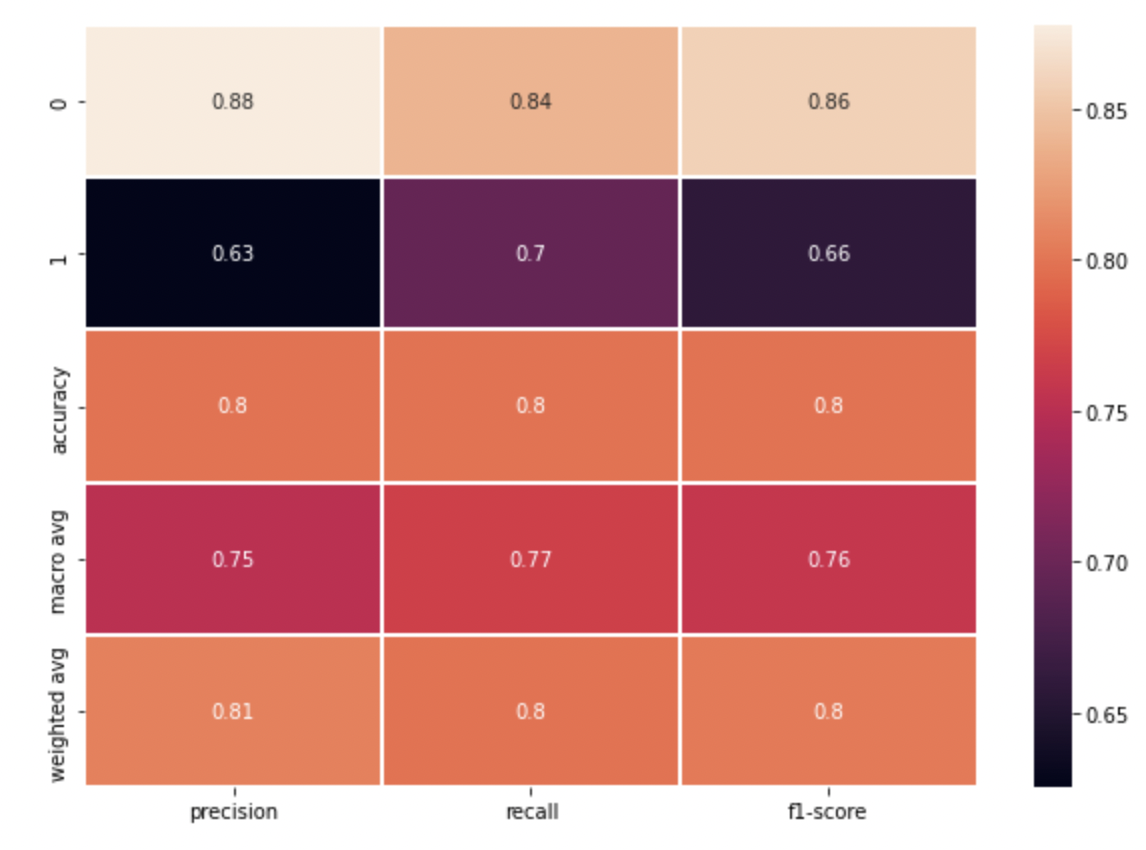


*Table 3: Best performing hyperparameters*

We performed hyperparameter tuning on both balanced and imbalanced datasets and the results are shown in figure 6 and 7. For the random forest classifier precision and recall are 0s. It can be concluded that, despite hyperparameter tuning there is no improvement in the results of classifiers as compared to previous results.



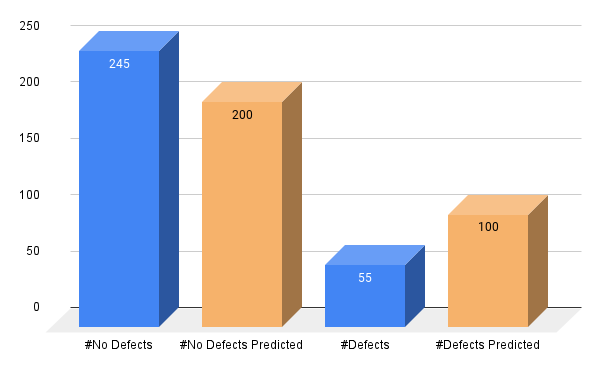
*Figure 7: Results of Classifiers on imbalanced dataset after hyperparameter tuning*



*Figure 8: Result of Logistic Regression model on balanced dataset after hyperparameter tuning*

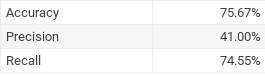
**Results and Conclusion**

Finally after doing all the aforementioned phases, therefore, decided to work further on the model which is trained using a logistic regression on the balanced dataset (by Random over sampler). So, here we are explaining the result of the model on the test data. The test data is the data for which the labels were manually generated for 300 rows of the appliances dataset. In the “Testing model on unseen data” code file, we have run the saved model which is done in previous steps.



*Figure 9: Model Predictions on Test Data*

Figure 6, represents the performance of the model on test data, our model has predicted 200 No Defects out of 245 actual No Defects and Defects predicted 100 out of which 55 are actual Defects. The performance metrics are listed below:



*Figure 10: Performance Metrics*

**References**

[1] <http://deepyeti.ucsd.edu/jianmo/amazon/index.html>

[2] <https://nijianmo.github.io/amazon/index.html>

[3] <https://imbalanced-learn.org/stable/user_guide.html#user-guide>

[4] <https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>