**Sentiment Analysis on Amazon Product Reviews**

**Procedure followed:**

Importing and loading Dataset:

Firstly, import all necessary python libraries and load the dataset into jupyter file using pandas.

Exploratory Data Analysis:

* By using info() methods in pandas. we are getting know that both review and sentiment columns has object datatype and total entries are 10000.
* shape() method gives the (10000,2) that means 10000 rows and 2 columns are there in dataset.
* describe() method gives count of each column and how many unique entire are existed in the dataset. In review column, Total count is 10000 and unique reviews were 10000. In label column, Total count is 10000 and unique classes are two (pos and neg).
* checking balance of the dataset is vital task in classification problem, here the dataset is balanced.so, no need of any subsampling techniques.

Text Pre-processing:

* In Sentiment Analysis problem we must have to clean the text and pre-process the text for further tasks.
* After that we have to use text pre-processing techniques like Tokenization, Lemmatization and remove stop words from the reviews.
* After all this process we are obtained cleaned text. Then we have to changes the class lables from categorical to binary digits.

Vectorizing:

* We can’t directly provide the textual data to the Machine model. So, we have to use Vectorization techniques.
* Here, we are going use Count Vectorizer to convert text data to DTM (document term matrix)

Model Building:

Multinomial Naïve Bayes Model:

* By using Sklearn library we can import the MultinomialNB.
* Training the model with training set of X and y.
* After training of the model, we have to evaluate the model performance by using Metrics (Accuracy, Precision, Recall) with test set.
* Accuracy: 0.83
* Precision: 0.83, Recall: 0.83, f1-score: 0.83
* After implementing Hyperparameter tuning: GridSearchCV techniques

grid.best\_estimator\_: MultinomialNB(alpha=1, class\_prior=None, fit\_prior=True)

grid.best\_params\_: {'alpha': 1}

* Accuracy: 0.83
* Precision: 0.83, Recall: 0.83, f1-score: 0.83

Decision Tree:

* By using Sklearn library. we can import the DecisionTreeClassifier.
* Training the model with training set of X and y.
* After training of the model, we have to evaluate the model performance by using Metrics (Accuracy, Precision, Recall) with test set.
* Accuracy: 0.7146
* Precision: 0.72, Recall: 0.72, f1-score: 0.72
* After implementing Hyperparameter tuning: GridSearchCV techniques

grid.best\_estimator\_: DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None, max\_features=None, max\_leaf\_nodes=29,min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=None,

splitter='best')

**grid.best\_params**\_: {'criterion': 'gini', 'max\_leaf\_nodes': 29, 'min\_samples\_split': 2}

* Accuracy: 0.73
* Precision: 0.74, Recall: 0.74, f1-score: 0.74

Support Vector Machine:

* By using Sklearn library we can import the SVC.
* Training the model with training set of X and y.
* In Support Vector Machine, there are kernels: linear, rbf, poly, sigmoid.
* After training of the model, we have to evaluate the model performance by using Metrics (Accuracy, Precision, Recall) with test set.
* Accuracy: 0.5096
* Precision: 0.29, Recall: 0.51, f1-score: 0.34
* After Implementing Hyperparameter tuning: GridSearchCV,

grid1.best\_params\_: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

grid1.best\_estimator\_: SVC(C=10, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma=0.001, kernel='rbf',

max\_iter=-1, probability=False, random\_state=None, shrinking=True,

tol=0.001, verbose=False)

* Accuracy: 0.83
* Precision: 0.84, Recall: 0.84, f1-score: 0.84

Conclusion:

SVM with {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'} Over-performs MultiNomialNB, Decision Tree Model.

Recurrent Neural Networks (LSTM):

1. Firstly, import required libraries and load the dataset using Pandas library.
2. Text Pre-processing steps using Keras framework. Tokenization, text to sequence and

Padding the sequence.

1. Splitting the pre-processed dataset to training and validation sets.
2. Initiating keras’s Sequential model.
3. Adding Embedding layer with input dim=10000, output dim=50 and input length = 500 to the sequential model.
4. Adding Convolution 1D layer with 128 filters and kernel size is 8 and rectified linear unit (relu) activation function to the sequential model.
5. Adding Dropout layer with dropout rate 0.2 to avoid overfitting to the model. Dropout is a Regularization technique which is used to remove 20% of neurons which will lead to overfitting the model.
6. Repeat the same layer with same parameters for one more time.
7. Adding Bidirectional LSTM with 128 units, return\_sequences=True, dropout=0.2 and recurrent dropout=0.2
8. Adding Dense layer with 256 units and rectified linear unit (relu) activation function to the sequential model.
9. Again, adding Dropout layer with dropout rate 0.2 to avoid overfitting to the model.
10. Output layer with only one unit and sigmoid activation for obtaining probability of class label.
11. Here, we are going to provide binary\_crossentropy as loss function, Adam as optimizer and accuracy as metrics for the model.
12. Adding Early stopping as Call back for the model to interactively train the model.
13. At last we are going to train the model with training set, validation set and 10 epochs.

Train on 7000 samples, validate on 3000 samples

Epoch 1/10

7000/7000 [==============================] - 74s 11ms/step - loss: 0.6782 - accuracy: 0.5490 - val\_loss: 0.5312 - val\_accuracy: 0.7390

Epoch 2/10

7000/7000 [==============================] - 65s 9ms/step - loss: 0.3678 - accuracy: 0.8420 - val\_loss: 0.3813 - val\_accuracy: 0.8393

Epoch 3/10

7000/7000 [==============================] - 65s 9ms/step - loss: 0.1588 - accuracy: 0.9443 - val\_loss: 0.4271 - val\_accuracy: 0.8387

Epoch 4/10

7000/7000 [==============================] - 67s 10ms/step - loss: 0.0733 - accuracy: 0.9786 - val\_loss: 0.6877 - val\_accuracy: 0.8367

Epoch 5/10

7000/7000 [==============================] - 67s 10ms/step - loss: 0.0461 - accuracy: 0.9850 - val\_loss: 0.7730 - val\_accuracy: 0.8217

Epoch 6/10

7000/7000 [==============================] - 66s 9ms/step - loss: 0.0313 - accuracy: 0.9896 - val\_loss: 0.6837 - val\_accuracy: 0.8287

Epoch 7/10

7000/7000 [==============================] - 67s 10ms/step - loss: 0.0167 - accuracy: 0.9946 - val\_loss: 0.9672 - val\_accuracy: 0.8257

Epoch 8/10

7000/7000 [==============================] - 66s 9ms/step - loss: 0.0121 - accuracy: 0.9960 - val\_loss: 0.8076 - val\_accuracy: 0.8237

Epoch 9/10

7000/7000 [==============================] - 67s 10ms/step - loss: 0.0095 - accuracy: 0.9969 - val\_loss: 1.0589 - val\_accuracy: 0.8330

Epoch 10/10

7000/7000 [==============================] - 65s 9ms/step - loss: 0.0110 - accuracy: 0.9961 - val\_loss: 0.9148 - val\_accuracy: 0.8303

<keras.callbacks.callbacks.History at 0x7fdb8203c978>