Capstone Project – 2.3

Customer's Review/ratings classification

Table of Contents

1. Problem Statement

You are working in an e-commerce company, and your company has put forward a task to analyze customer reviews for various products. You are supposed to create a report that classifies the products based on customer reviews.

Dataset Information:
The Reviews.csv dataset contains 60145 rows and 10 columns.

Feature Name	Description
Id	Record ID
ProductId	Product ID
UserId	User ID who posted the review
ProfileName	Profile name of the User
HelpfullnessNumerator	Numerator of the helpfulness of the review
HelpfullnessDenominator	Denominator of the helpfulness of the review
Score	Product Rating
Time	Review time in timestamp
Summary	Summary of the review
Text	Actual text of the review

---

2. Project Objective

1. Find various trends and patterns in the review data, and create useful insights that best describe the product quality.
2. Classify each review based on the sentiment associated with the same.

However, it includes different steps to explore the trends and obtain something useful from the dataset, which are:

- Exploratory Data Analysis
- Data Pre-processing & cleaning the data
- Dealing with Null entries and creating equally distributed sample
- Further, Feature Engineering to Extract and select features accordingly

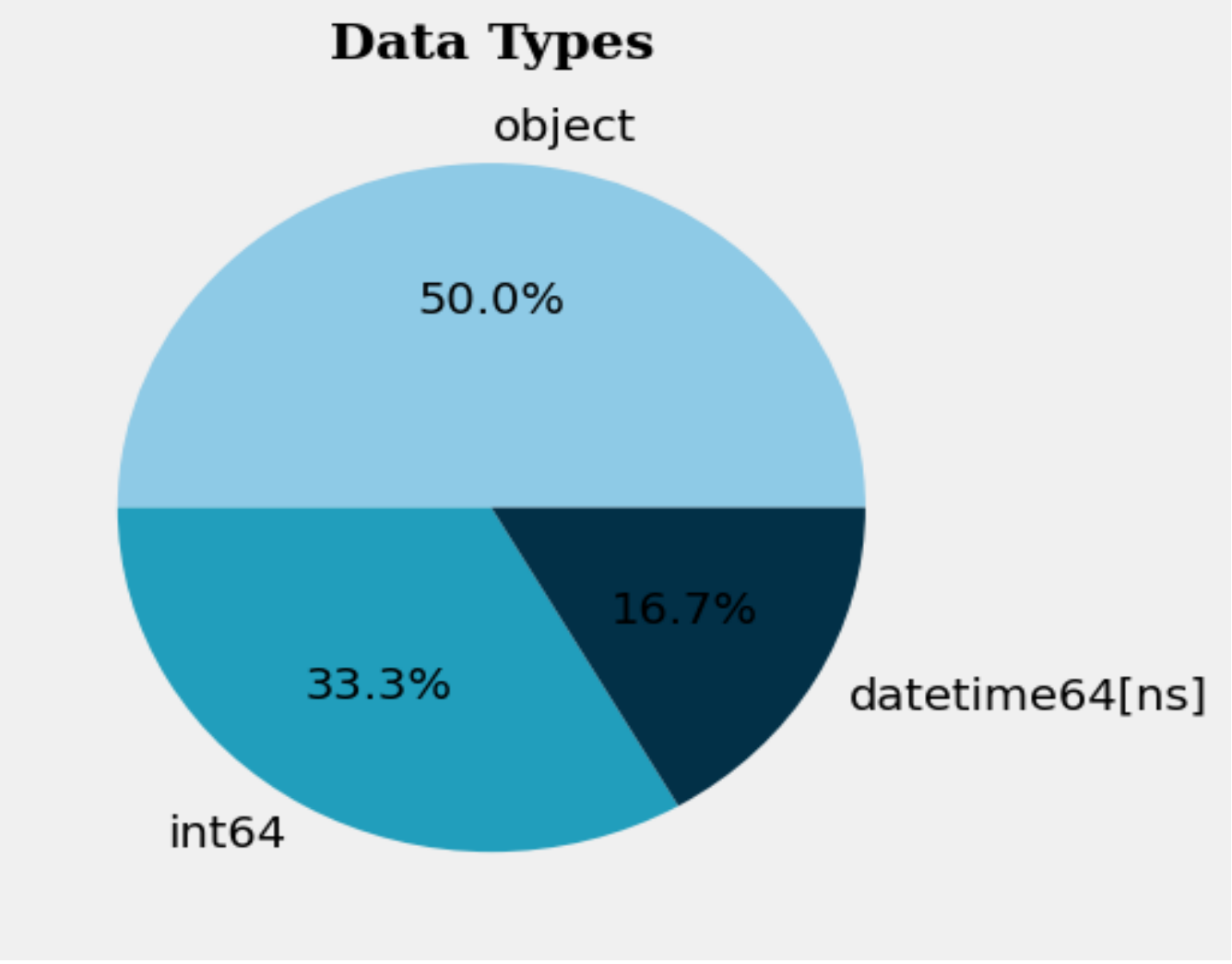
3. Data Description

The shape of the dataset as per the problem statement 568454 entries with 8 columns/features having, Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, and Text. This is reduced by the number of features and proceedings with only 5 features, which are:

Range Index: 568454 entries, 0 to 568453
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	Time	568454	non-null int64
1	ProductId	568454	non-null object
2	Summary	568427	non-null object
3	Text	568454	non-null object
4	Score	568454	non-null int64

Data types distribution of the overall dataset we have,

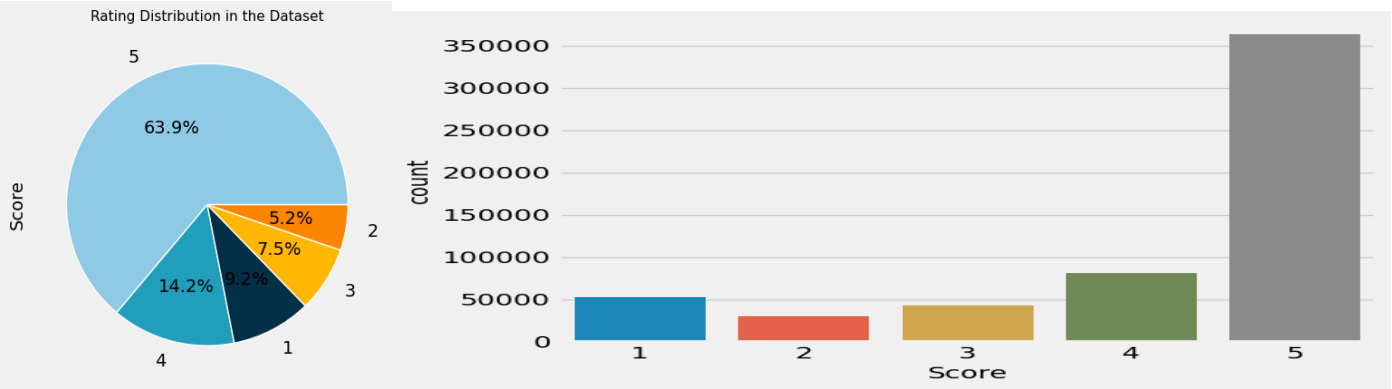


3.1. Rating Distribution

Exploring further, I have observed that the dataset is not evenly distributed according to the customer's rating/score, and Rating '5' have more than enough entries in all. I will resample the given data as per the problem statement having 60145 entries with an even number of entries on the basis of the customer's rating. However, the following observation is below:

- Number entries for Rating-'1' in the dataset: (52268, 6)
- Number entries for Rating-'2' in the dataset: (29744, 6)
- Number entries for Rating-'3' in the dataset: (42638, 6)
- Number entries for Rating-'4' in the dataset: (80655, 6)
- Number entries for Rating-'5' in the dataset: (363122, 6)

Plots obtained:



Clearly, we can say that the dataset is unbalanced.

---

4. Data Pre-processing Steps and Inspiration

4.1. Extracting the root word:

How often a word is used is key information in natural language processing. It is therefore important to reduce words to their root form. An example would be the usage of the word 'learn'. If we differentiate this base form from a modified version like 'learning' then we might lose relational context between two documents that have used either word.

I am using Lemmatization to reduce tokens to their base word. This technique takes into account context similarity according to part-of-speech anatomy. Stemming is another common approach, although stemming only performs truncation and would not be able to reduce 'taught' to 'teach'.

We will be using the '*WordNetLemmatizer*' from the Natural Language Toolkit (or NLTK). Lemmatization only applies to each word but it is dependent on sentence structure to understand context. We, therefore, need to have part-of-speech tags associated with each word. Our output is derived from applying the `get_wordnet_pos` function to our `clean_text ` column.

The `get_wordnet_pos` works as follows:
- Each review is broken down into a list of sentences
- Punctuations that only group words or separate sentences (hyphens, therefore, are excluded) are removed (replaced by whitespace) using RegEx
- Every sentence is further broken down into words (tokens)

Each of the sentences then becomes an ordered bag of words. Every word is then tagged to a part of speech. This word-tag tuple pair is then fed one at a time to the `lemmatize word` function, which works as follows:
- Only modifiable words – nouns, verbs, adjectives, and adverbs – can be reduced to roots
- These words are lemmatized and appended to the `root` list
- Words that are not modifiable are added as they are to the `root` list

The output lists are linked together as a string using whitespace. In the end, each `clean_text ` review will retain its text form but with each word simplified as much as possible.

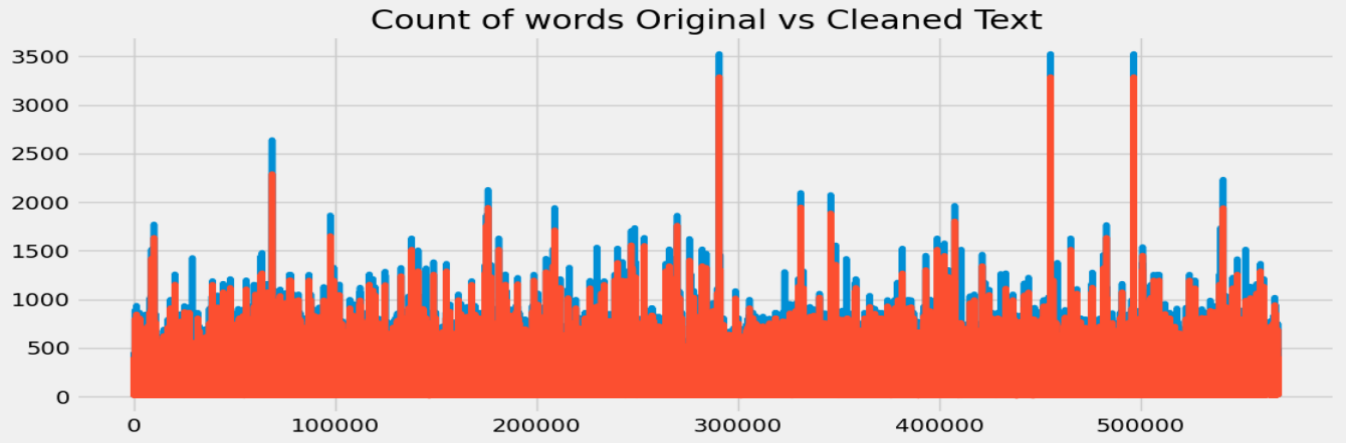
4.2. Removing Punctuation:
The `clean_text` reviews are further cleaned by dropping punctuations. Using regular expressions, only whitespaces, and alphanumeric characters are kept.

4.3. Converting to lowercase:
Every letter is also converted to lowercase. This makes it so that 'Good' will not be distinguishable from 'good'.

4.4. Removing stop words:
Stop words consist of the most commonly used words that include pronouns (e.g., us, she, their), articles (e.g., the), and prepositions (e.g., under, from, off). These words are not helpful in distinguishing a document from another and are therefore dropped.

4.5. Removing Extra Spaces:
Again, we make use of regular expressions to ensure we never get more than a single whitespace to separate words in our sentences.

After applying all the pre-processing steps, I made a separate column to store it by 'clean_text'. And, I have reduced the overall word count by 10.24%. This can also be observed in the below plot:



4.6. Creating sentiment column:

Sentiment lets us understand the overall feedback of the customer which clarifies product quality as per the usage of the products. I have created a column 'sentiment' which categorizes the score value as Very-unsatisfied (1), Unsatisfied (2), Neutral (3), Satisfied (4), and Very-satisfied (5) based on the given score by using the function 'categorise_sentiment'.

4.7. Feature Engineering (Feature Extraction & Selection):

A. Tokenization

The entries for the `clean_text` column are extracted to make up our 'corpora', which is simply a collection of all our documents. Each review is then transformed into an ordered list of words. This is the process of 'tokenization' – the document is broken down into individual words or tokens.

B. Creating the Vocabulary

The `vocabulary` is the key-value pairs of all the unique tokens from every product review. Each token is assigned a lookup ID.

C. Phrase Modelling

Since the order of words matters in most NLP models, it is often helpful to group neighbouring words that appear to convey one meaning as though they are a single word, like 'smart TV'.

To be considered a 'phrase', the number of times that two words should appear next to each other is set to at least `300`. The threshold then takes that minimum and compares it to the total number of token instances in the corpora. The higher the threshold, the more often two words must appear adjacent to be grouped into a phrase. By performing the below-mentioned pre-processing steps:

a. Uni-grams
b. Bi-grams
c. Tri-grams

D. Bag of Words Model

The classical approach in expressing text as a set of features is getting the token frequency. Each entry to the data frame is a document while each column corresponds to every unique token in the entire corpora. The row will identify how many times a word appears in the document.

E. TF-IDF Model

The Term Frequency-Inverse Document Frequency (TF-IDF) approach assigns continuous values instead of simple integers for the token frequency. Words that appear frequently overall tend to not establish saliency in a document and are thus weighted lower. Words that are unique to some documents tend to help distinguish them from the rest and are thus weighted higher. The `tfidf` weighting is based on our `bow` variable.

However, we have performed this to obtain a data frame, which is stored in 'tfidf_mat' using 'create_tfidf_matrix' function to proceed further with the machine-learning model for classifying the rating scores with all the features extracted from tokenization.

F. Word Embedding

The downside of count-based techniques is that without regard to word sequence and sentence structure, the semantics get lost. The 'Word2Vec' technique, on the other hand, actually embeds meaning in vectors by quantifying how often a word appears within the vicinity of a given set of other words.

A context window the span of `context_size` slides across every document one token at a time. In each step, the center word is described by its adjacent words, and the probability that the token appears together with the others is expressed in the `feature_size` dimensions. Since the minimum word requirement is set to `1`, every token in the corpora is embedded in the Word2Vec model.

I have performed this to create a word embedding vector with a maximum of 500 features, which is stored in 'model_word2vec' variable, and also to find the similar words available in the data set, for which I have created a function 'similar_word', which works as follows:

- The function takes a list of words.
- Finds all possible words and returns the list having 5 similar words.
- Then, print the passed words with their similar words in the key-value pair.

Also, I have created to model to perform the NER, find the entity of the words, and create a meaningful semantic with their relevant words, which can be helpful for recommendation systems, finding relevant statements, news, etc.

4.8. Underrepresentation vs. Overrepresentation:

Performing over-representation is possible by bootstrapping the minority classes to match the size of the majority classes. This can be done using K-Nearest Neighbours (KNN) or via Support Vector Machine (SVM) by clustering a given class first before generating random samples within the decision boundaries of the class. A popular module called `SMOTE`, or (Synthetic Minority Over-sampling Technique), does exactly this. However, since the imbalance in our classes is massive, and because we have 100 dimensions for each one of our almost 1.7 million observations, this approach is extremely computationally expensive.

Because our dataset is huge, we can afford to perform sampling in every class and still have a significant amount of data for the model. This way, we can then opt to 'Underrepresent' the majority class according to our minority class.

---

5. Choosing the Algorithm for the Project

As we have obtained two different vectors, by performing TF-IDF and Word2Vec and need to choose the best algorithm for the model building. I have created a function to fit the 5 different models to get an

optimal-performing model. We have created three different functions, which are, ‘read_data’ returns dependent and independent variables, so, further using ‘prepare_data’, it can be further split into trains and test set using train test split, and then, function ‘run_classification_models’ will be used to fit the vector and target column, to check the accuracy with test and train set.

Here, I have chosen 6 different classification algorithms mentioned below and algorithms are,

* Logistic Regression
* Linear SVC
* Multinomial Naive Bayes or GaussianNB
* Decision Tree Classifier
* Random Forest Classifier
* Gradient Boosting Classifier

Later, I decided to not fit the ‘Gradient Boosting Classifier’ as the training model with this algorithm is very time-consuming and not worth it, because, Random Forest & Decision Tree already give better accuracy with the train set and acceptable accuracy with the train set. Also, attached the screenshot for it below:

OUTCOMES:

Using TF-IDF Dataset

```
%%time
if __name__ == '__main__':
    X, y = tfidf_mat.T,model_Word2Vec.score
    X_train, X_test, y_train, y_test = prepare_data(X, y)
    run_classification_models(X_train, X_test, y_train, y_test)
```

```
Logistic Regression Accuracy for Train set: 71.247%
Logistic Regression F1 Score for Train set: 0.712

Logistic Regression Accuracy for Test set: 55.092%
Logistic Regression F1 Score for Test set: 0.551

Linear SVC Accuracy for Train set: 87.060%
Linear SVC F1 Score for Train set: 0.871

Linear SVC Accuracy for Test set: 52.490%
Linear SVC F1 Score for Test set: 0.525

Multinomial Naive Bayes Accuracy for Train set: 73.073%
Multinomial Naive Bayes F1 Score for Train set: 0.731

Multinomial Naive Bayes Accuracy for Test set: 51.359%
Multinomial Naive Bayes F1 Score for Test set: 0.514

Decision Tree Classifier Accuracy for Train set: 99.981%
Decision Tree Classifier F1 Score for Train set: 1.000

Decision Tree Classifier Accuracy for Test set: 43.744%
Decision Tree Classifier F1 Score for Test set: 0.437

Random Forest Classifier Accuracy for Train set: 99.981%
...
Random Forest Classifier F1 Score for Test set: 0.508
```

Using Word2Vec Dataset

```
%%time
if __name__ == '__main__':
    X, y = read_data(model_Word2Vec)
    X_train, X_test, y_train, y_test = prepare_data(X, y)
    run_classification_models(X_train, X_test, y_train, y_test)
```

```
Logistic Regression Accuracy for Train set: 52.789%
Logistic Regression F1 Score for Train set: 0.528

Logistic Regression Accuracy for Test set: 50.387%
Logistic Regression F1 Score for Test set: 0.504

Linear SVC Accuracy for Train set: 46.589%
Linear SVC F1 Score for Train set: 0.466

Linear SVC Accuracy for Test set: 44.335%
Linear SVC F1 Score for Test set: 0.443

Multinomial Naive Bayes raised an error, fitting Gaussian Naive Bayes instead...
Gaussian Naive Bayes Accuracy for Train set: 39.656%
Gaussian Naive Bayes F1 Score for Train set: 0.397

Gaussian Naive Bayes Accuracy for Test set: 39.263%
Gaussian Naive Bayes F1 Score for Test set: 0.393

Decision Tree Classifier Accuracy for Train set: 99.981%
Decision Tree Classifier F1 Score for Train set: 1.000

Decision Tree Classifier Accuracy for Test set: 42.797%
Decision Tree Classifier F1 Score for Test set: 0.428
...
Random Forest Classifier F1 Score for Test set: 0.496
```

Note: I have made a Word2Vec-based vector by having the mean of the weights of their features, so, the ‘model_word2vec’ feature has negative values, due to which MultinomialNB gives an error, so, I decided to fit GaussianNB instead.

On the basis of the accuracy, I have decided to move forward with Random Forest Classifier for the final model-building process.

6. Motivation and Reasons for Choosing the Algorithm

As the problem was given to classify the review made by the customer as per their ratings ranging from 1 to 5. I have chosen 6 classification algorithms, i.e,`LogisticRegression`, `LinearSVC`, `MultinomialNB` or `GaussianNB`, `DecisionTreeClassifier`, `RandomForestClassifier`, and `GradientBoostingClassifier` in which `DecisionTreeClassifier`, `RandomForestClassifier` are performed much better with train set than others, and all of the models performing average with the test set, reason are mentioned below:

1. Accuracy: One of the most important factors when selecting a classifier algorithm is its accuracy in predicting the correct class labels for new, unseen data. Different algorithms may be better suited to different types of data or classification problems, but, DecisionTreeClassifier and RandomForestClassifier performed well & obtain 99.981% and 99.973% respectively, for both of our obtained datasets.

2. Complexity: Another factor to consider is the complexity of the algorithm, both in terms of the number of computational resources required to train and test the model, as well as the complexity of the resulting model itself. A simpler model may be easier to interpret and more efficient to compute but may sacrifice some accuracy.

3. Scalability: The classifier algorithm should be able to scale to handle large datasets or high-dimensional feature spaces while maintaining reasonable computational efficiency.

4. Interpretability: In some applications, it may be important to have a classifier algorithm that produces results that are easily interpretable by humans, so that they can be used to guide decision-making or to gain insights into the underlying data.

* **Logistic Regression:**

1. I tried different hyper-parameter tunning, penalty: {'l1', 'l2'}, where the model started to lower the accuracy. It seems the hyperparameter(λ) is very large which causes the model to underfit

2. At last, after trying `class_weight`, `solver` and `random_state`, still model not performed better, so tried to train it without `class_weight` and got 10.8% for *tfidf_df* & 13.2% for *model_Word2Vec* rise in train_set accuracy.

* **Linear SVC:**

1. This model also not performed well with different attributes, model started to show convergence errors but it got resolved by specifying `max_iter` and `penalty` but it tends to time complexity to run the model.

2. `class_weight` and `random_state` helps to achieve 2% more accuracy which is the result 87% & 47% for the datasets.

* **Multinomial Naive Bayes:**

1. This model has the worst accuracy in all and gets 'ValueError' as *model_Word2Vec* has negative values so fitted `GaussianNB` instead.

2. used different alpha values ranging (from 0.1 to 0.7), accuracy not increased at all.

*** Decision Tree Classifier:**

1. Model gives 99% accuracy on train_data with `class_weight` `criterion: entropy` `random_state`

*** Random Forest Classifier:**

1. This model also gives 99% accuracy on train_data with some tunning, i.e, `class_weight` `criterion: entropy` `random_state`

2. `n_estimators` used and trained with 25, 50, and 100 but the model gives accuracy with estimator:25

Note:

1. The `class_weight` attribute is provided with a dictionary that represents the associated weight of each class – the majority class is given as 1 and the rest are given the multiplying factor at which they would level with the largest class.
2. The criteria chosen is `entropy` which is similar to `gini` but instead of splitting nodes until there are pure classes, the nodes are split until the classes within have equal probability.

---

7. Assumptions

After running all the different models, I have observed that models are performing enough to get approx. prediction and most of all are performing well, the reason being the number of classes we have, as I am presuming that most of the words are almost similar, the models trained with this and predicted nearby classes, but not accurate as per the true target label.

Now, I am proceeding with the Random Forest classifier for the final model building. Post, I will also be moving forward to fit a Bi-LSTM model using Pad sequence and Word embedding method with TensorFlow.

So, I need to prefer to proceed with three classes to obtain a good model, thereafter, we can proceed with the `Deep Neural Network - Word Embedding` having all 5 classes to obtain a good model in all.

So, the final models to proceed are as follows:

a. Random Forest Classifier
b. Bi-LSTM with Word Embedding

---

8. Model Evaluation and Techniques

In our study, we will make use of four metrics to measure the model performance:

* Accuracy
* F1 Score
* Confusion Matrix
* AUC Score

Accuracy will identify how many reviews are correctly labeled by the model. There are five ratings and thus five classes. No review can have two or more ratings and so the probability that a correct prediction is made from pure guesswork is `30%`.

The **F1 score** is taking precision and recall into consideration. Taking into account false positives and false negatives for each class is especially important in inherently imbalanced datasets.

The confusion matrix is a table that shows the true positive, false positive, true negative, and false negative values for each class. It provides an overview of the model's performance across all classes.

AUC can be computed using the trapezoidal rule[3]. In general, an AUC of 0.5 suggests no discrimination (i.e., ability to diagnose patients with and without the disease or condition based on the test), 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding. I am using this technique to evaluate the prediction made by the Bi-LSTM model, because, In the model, I am using a 'softmax' layer, which will return a list or array having probability values.

However, the baseline scores are for when a model only randomly guesses the output labels – in this case, when every prediction is the same class. The scores are also based on an evenly distributed dataset.

---

9. Inferences from the Same

9.1. Inferences obtain by performing EDA (Exploratory Data Analysis) on Products and customer rating:

I have created a function that can provide a pie chart with the customer sentiments distribution of a particular product and the count of users lying under each sentiment, by using the function, 'pie_chart' and 'bar_chart' which is taking the product-based data using the function 'create_product_data'.

* Here, we can say most of the products have more than 80% rating as 'Very-satisfied'.
* Almost 41% of total products are rated only once by the user, which is 30394 out of 74243.
* And 87% of total products are rated 10 times or less by the user, which is 64630 out of 74243.
* Only 10% of overall products are more frequently used & rated by the users.
* 'B007JFMH8M' product is mostly rated and the most popular in all. Also, 30394 products are the least popular and are rated only once by the users.
* So, we can say that 30394 products are the least purchased products or least rated and less popular among all.
* With this analysis we can also categorize the product's quality as per their sentiment/score.

Above analysis clarify that the distribution of rating count for each products is not even or we can say skewed.

9.1. Inferences about the model and their scores

* Keeping all the features in using TF-IDF models worth the train set accuracy, however, the test set accuracy is not getting increased with several models I have tried.
* As per the better train & test accuracy obtained with Random Forest Classifiers with Word2Vec dataset significantly better than the TF-IDF, because, the Word2Vec model uses only 500 features instead of 58 thousand.
* Moving forward with the Bi-LSTM is said to be worthful, as the probability results over the test set are better than all other models.
* The train loss and validation loss graph not even appeared to overfit, even after 25 epochs, accuracy keeps increasing and loss was decreased.
* However, it is clear that the number of classes is the reason for such test accuracy. I can say pre-processing the data again having three labels will allow us to achieve better test accuracy.

---

10. Future Possibilities of the Project

Though we have observed satisfactory results in our model compared to the baseline, there are several limitations in the way the model handles data. These could serve as areas of improvement. First, despite a rich vocabulary, the model will not be able to handle words that it has not encountered during training. In fact, if an unknown word appears in a review, the word is dropped from the dimension-averaging step since has not been referenced in our `word_vec_df`. However, I have also tried to create proceed along with TF-IDF, so, I have created a vector with all the features to overcome the possible issues.

Because each word is simplified by lemmatization during pre-processing, then alternate forms of a token shouldn't necessarily be a concern. However, the model cannot identify if a word is misspelled and will identify one simply as a new word. Incorporating a spellchecker would add to the computational cost and will certainly add to the model's complexity.

Finally, as is usually the case in NLP, sarcasm or text that is intended to be ironic is interpreted by what is literally in the text and not by its underlying context. Because sarcasm is usually detected by readers through the mood and sentiment of the document, it takes adding another layer of NLP just to approximate whether the review is sarcastic or not in order to properly work with such text. This supplement layer will not only utilize tagged sarcastic text as supervised labels but must also consider the review's given product rating in its judgment to detect sarcasm.

However, The Bi-LSTM model has performed significantly better results as it has 92% accuracy and is better compared to other models, we have trained. The Bi-LSTM model shows an AUC score of 80% and word_vec model also performed well to find similar words available in the reviews, which can be helpful to find out the similar reviews made by the customer.

Overall, Bi-LSTM and word_vec models are capable to generate the satisfactory result. We can also use these models to obtain customer's sentiment over the product.

11. Conclusion

A lot of Natural Language Processing techniques were covered in the study. Just some of the concepts explored include topic modeling – where similar/neighboring texts were clustered together according to the topic. Though the Word2Vec phase was central to our final model, the pre-processing steps were perhaps just as crucial. Prior to tokenization, each document had to be decoded from UTF and encoded to ASCII, and converted to lowercase. The texts were stripped of accents, stop words, and punctuation, and multiple whitespaces were dropped. Words were simplified to their root words in order to compact the vocabulary as much as possible. Tokens that were often used together were also singularized through phrase modeling.

Beyond word use and word frequency, our model actually extracts and quantifies *context*. Every token in all the reviews is understood by its neighboring words and embedded in a given number of dimensions. All the interactions of a word with all the other words it has been associated with are expressed in vectors. And all the words in a given review are averaged according to each of the dimensions to create its `500` features. So, the essence of a review by its words makes up the final data frame.

What we have is a multi-class model where each of the five classes corresponds to a review's star rating. This is then a discrete approach where each class is independent of each other. In a situation where a 5-star rating is misinterpreted by the model as a 1-star review, then the model has simply misclassified – it is agnostic to how far off `1` and `5` are. This is in contrast with a *continuous* approach whereas a misclassification of a 5-star review as a 1-star review would be more penalizing. Our model then is reliant on the distinction of each kind of review.

It is more concerned with asking "**What makes a 5-star review different from a 4-star review?**" than asking "**Is this review more approving than criticizing?**" having, a model trained with only three classes will be much better than we have for now.

12. References

1. https://towardsdatascience.com/pretrained-word-embeddings-using-spacy-and-keras-textvectorization-ef75ecd56360
2. https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools
3. https://www.python-graph-gallery.com/pie-plot-matplotlib-basic
4. https://plotly.com/python/pie-charts/
5. https://plotly.com/python/bar-charts/
6. https://towardsdatascience.com/4-simple-tips-for-plotting-multiple-graphs-in-python-38df2112965c
7. https://professorkazarinoff.github.io/Problem-Solving-101-with-Python/05-Plotting-with-Matplotlib/05.08-Pie-Charts/
8. https://plotly.com/python/figure-labels/
9. https://towardsdatascience.com/multi-class-text-classification-model-comparison-and-selection-5eb066197568
10. https://towardsdatascience.com/the-most-common-evaluation-metrics-in-nlp-ced6a763ac8b

11. https://github.com/paradoxpj/twt-sentimental-analysis/blob/main/Mental%20Health%20Sentiment%20Analysis%20using%20Twitter.ipynb
12. https://github.com/roshancyriacmathew/Deep-Learning-on-Amazon-Alexa-Reviews/blob/main/Amazon%20Alexa%20Review%20Analysis%202%20%20_%20Live%20-%20%20LSTM.ipynb