Programming Assignment - 1

Drithi Davuluri (B21AI055) G Mukund (B21CS092)
February 20, 2024

Task -1

1. *POS tagger and NER models*

1.1 Spacy

- Employed SpaCy's en_core_web_sm model for English language processing.
- Created 2 columns to store the results of POS tags and NER of SpaCy.
- We generate the POS tags for each token in the sentence word in sentence and stores them as a list of tuples.
- We generate named entities for each entity (ent) in the sentence (doc) and stores them as a list of tuples.
- We do the same for all the sentences and hence get 2 columns namely pos_spacy and ner_spacy.

1.2 NLTK

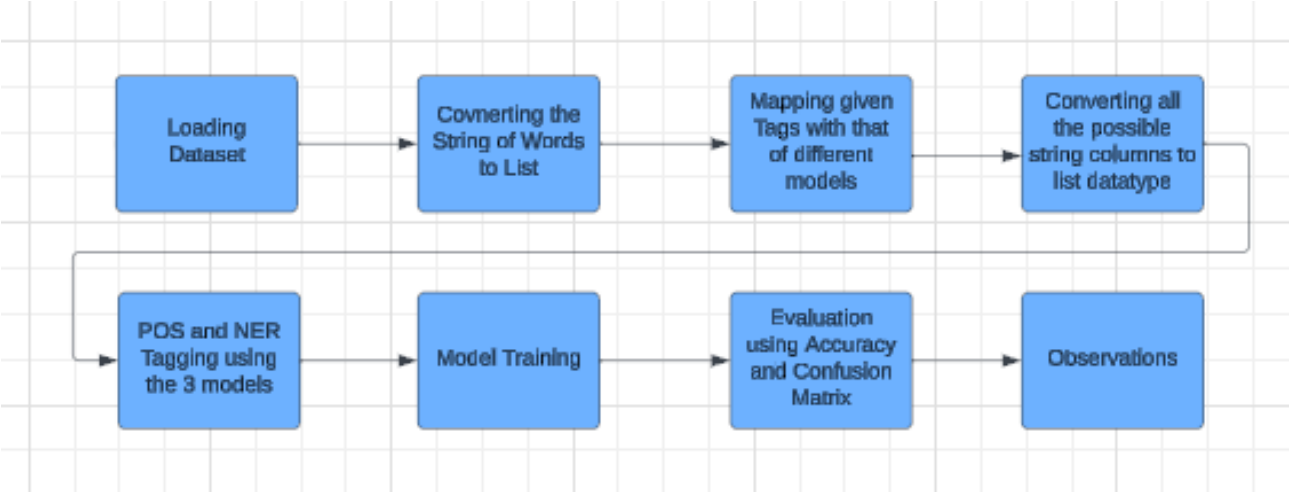
- Utilized NLTK's `pos_tag` function for POS tagging and `ne_chunk` for Named Entity Recognition (NER) processing.
- Defined functions `pos_tag_nltk` and `ner_nltk_func` to generate POS tags and NER tags respectively.
- Created columns `pos_nltk` and `ner_nltk` to store the results of NLTK's POS tagging and NER.
- Iterated through each sentence in the DataFrame, applying the defined functions to generate POS and NER tags.
- Stored the results in the corresponding columns `pos_nltk` and `ner_nltk` for each sentence, resulting in POS and NER tagged data.

1.3 StanfordNER

- Configured the Stanford NER model with the path to the model and the JAR file.
- Created functions `pos_tag_stanford_ner` and `ner_stanford_ner` to perform POS tagging and NER using the Stanford NER Tagger.

- Assigned the results of POS tagging and NER to the columns `pos_stanford` and `ner_stanford` respectively in the DataFrame.
- Iterated through each sentence in the DataFrame, applying the Stanford NER Tagger to generate POS and NER tags.
- Saved the tagged results in the respective columns `pos_stanford` and `ner_stanford` for each sentence, providing POS and NER tagged data.

2. *Text Processing Pipeline*



2.1 Text Preprocessing

1. *Data Transformation*: Converted the 'Word' column from a string representation of a list to a list of words for effective word manipulation.

2. *Sentence Formation*: Reconstructed sentences by joining the words extracted from the 'Word' column, ensuring coherent sentence structures.

3. *Tag Mapping*: Defined dictionaries to map entity tags in the dataset to standard tags used by different NLP libraries, ensuring tag consistency across models.

4. *Column Initialization*: Initialized additional columns ('spacy_tag', 'nltk_tag', 'stanford_ner') to store mapped tags for each model.

5. *Tag Mapping Application*: Iteratively applied the defined mapping dictionaries to map entity tags to their corresponding standard tags for each model. This involved converting string representations to lists, mapping tags, and updating the DataFrame with mapped tags.

2.2 POS, NER Tagging for all the 3 models

Spacy:

- Utilized SpaCy's en_core_web_sm model for English language processing.
- Created columns pos_spacy and ner_spacy to store POS tags and NER results respectively.
- Employed SpaCy's tokenization to generate POS tags for each token in the sentence.
- Utilized SpaCy's entity recognition to identify named entities in the sentence.
- Stored the POS tags and NER results as lists of tuples in the respective columns.

NLTK:

- Utilized NLTK's pos_tag function for POS tagging and ne_chunk for NER.
- Defined functions pos_tag_nltk and ner_nltk_func for generating POS and NER tags.
- Created columns pos_nltk and ner_nltk to store the tagged results.
- Applied the defined functions to each sentence in the DataFrame to obtain POS and NER tagged data.

Stanford NER:

- Configured Stanford NER with the model path and JAR file.
- Implemented functions pos_tag_stanford_ner and ner_stanford_ner for POS tagging and NER.
- Stored POS tags and NER results in columns pos_stanford and ner_stanford.
- Applied Stanford NER Tagger to each sentence, generating POS and NER tags.
- Saved the tagged data in the respective columns for further analysis.

2.3 Model Evaluation

NLTK:

Data Transformation:

- Converted 'pos_nltk', 'pos_spacy', and 'POS' columns from string representations of lists to lists of tuples.
- Ensured uniform data format across columns for further analysis.

Filtering Irregular Entries:

- Removed rows with unequal lengths of POS lists in 'POS' and 'pos_nltk' columns.
- Maintained data consistency by filtering irregular entries.

Evaluation Metrics:

- Extracted ground truth and predicted data.
- Flattened nested lists for evaluation.
- Calculated accuracy score to assess NLTK POS tagger performance against ground truth. Final accuracy score of nltk was 96.

Spacy:

Data Transformation:

- Converted 'pos_nltk', 'pos_spacy', and 'POS' columns from string representations of lists to lists of tuples.
- Ensured uniform data format across columns for further analysis.

Filtering Irregular Entries:

- Removed rows with unequal lengths of POS lists in 'POS' and 'pos_nltk' columns.
- Maintained data consistency by filtering irregular entries.

Evaluation Metrics:

- Extracted ground truth and predicted data.
- Flattened nested lists for evaluation.
- Calculated accuracy score to assess NLTK POS tagger performance against ground truth. Final accuracy score of nltk was 96.

3. *Model Predictions-*

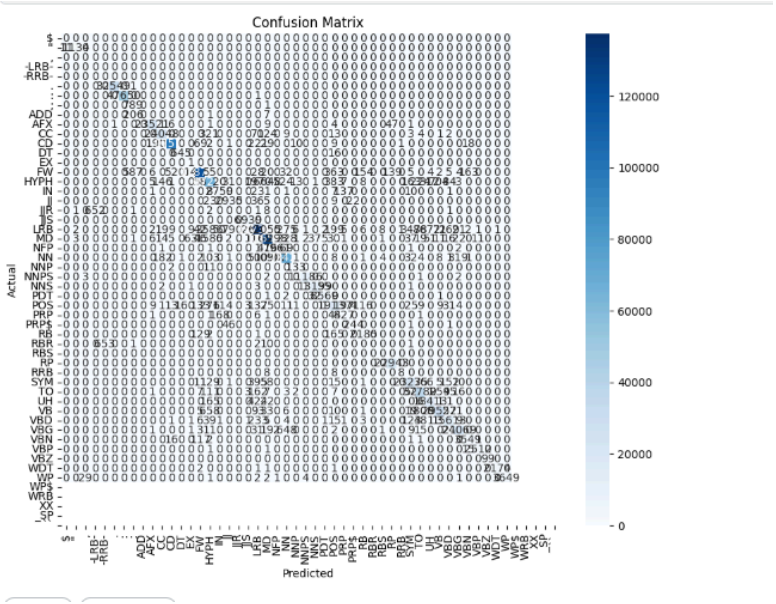
NLTK: The final accuracy of the model is 96.249 %.
SpaCy : The final accuracy of the model is 95.804 %.

- The NLTK tagger achieves slightly higher accuracy than spaCy on this dataset - 96.25% vs 95.80%. This indicates NLTK may be doing a better job overall at predicting the correct POS tags.

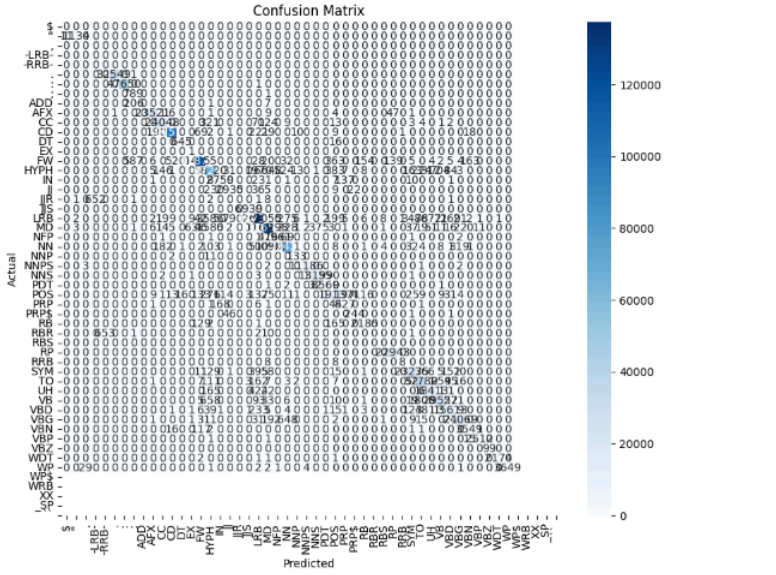
- However, looking at the confusion matrices, NLTK seems to struggle more with certain tag pairs like NN/NNS (noun singular/plural) and VBP/VBZ (verb present tense singular/plural) based on the higher values off the diagonal.

Confusion Matrices-

1) Spacy



2) NLTK



4. *Comparative strengths-*

- The NLTK tagger achieves slightly higher accuracy than spaCy on this dataset - 96.25% vs 95.80%. This indicates NLTK may be doing a better job overall at predicting the correct POS tags.

- However, looking at the confusion matrices, NLTK seems to struggle more with certain tag pairs like NN/NNS (noun singular/plural) and VBP/VBZ (verb present tense singular/plural) based on the higher values off the diagonal.

Based on the other evaluation metrics obtained:

NLTK Model:
- Precision: 96.32%
- Recall: 96.25%
- F1-score: 96.26%

SpaCy Model:
- Precision: 96.27%
- Recall: 95.80%
- F1-score: 95.89%

Comparative Analysis:

1. Precision:
   - NLTK model has a slightly higher precision than SpaCy, indicating that it tends to make fewer false positive predictions.

2. Recall:
   - NLTK model also has a slightly higher recall than SpaCy, suggesting that it captures more of the actual positives.

3. F1-score:
   - NLTK model's F1-score is marginally higher than SpaCy, implying that it achieves a slightly better balance between precision and recall.

Overall, both models perform well, with NLTK showing a slightly better performance across all metrics. However, the differences are relatively small, indicating that both models are comparable in terms of their effectiveness for POS tagging tasks. Other factors such as ease of use, computational efficiency, and available features may also influence the choice between these two models for specific applications.

5. *Observations about errors-*

The confusion matrices reveal that spaCy better distinguishes between similar noun and verb tags indicating it leverages contextual information well. For example, spaCy has lower confusion between

singular and plural nouns (NN/NNS) and present tense verbs (VBP/VBZ) likely because it utilizes surrounding semantics.

However, NLTK demonstrates strengths in identifying certain POS tags like comparative adjectives (JJ/JJR) and prepositions (IN) where its hand-crafted rules offer control. This enables NLTK to correctly tag some irregular cases that depend less on context

SpaCy's neural network provides robustness in using context for disambiguation but lacks the flexibility of NLTK's feature engineering. An ideal system would incorporate both - spaCy's contextual modeling and NLTK's rules-based approach. While spaCy is convenient out-of-the-box, NLTK allows customization that could improve domain-specific accuracy.

6. *Mapping Tags-*

- Regarding the tag mapping strategy, we've meticulously outlined the process of aligning tags from various datasets with those used in SpaCy, NLTK, and Stanford NER models. This mapping is absolutely crucial as it ensures consistency and compatibility when dealing with data tagged using different standards or tools.

- Establishing a clear mapping between dataset tags and model tags is key to maintaining consistency in entity labeling across different datasets and models. This consistency is paramount for accurately training and evaluating natural language processing (NLP) models.

- One significant advantage of the mapping strategy is its contribution to enhancing the performance of NLP models. By enabling these models to recognize and categorize entities based on standardized tags, we ultimately achieve more accurate entity recognition and extraction across various NLP tasks, such as named entity recognition (NER) and information extraction.

- Through the establishment of a standardized tag mapping, we facilitate interoperability between different NLP tools and frameworks. This means that researchers and practitioners can seamlessly exchange data annotated with different tag schemes, promoting collaboration and knowledge sharing within the NLP community.

- Acknowledging the scalability and adaptability of the tag mapping approach is essential. It can be extended to accommodate additional tag schemes or customized annotations based on specific project requirements. This flexibility ensures that the mapping strategy remains relevant and effective across diverse NLP applications and domains.

Task - 2

1. Data Preparation

Downloaded 22 categories of datasets.

1.1 Initial sampling:

Uniform random sampling is used to select examples from the corpus. Specifically, up to 2000 examples are sampled from each json file in the dataset. This led to a skewed distribution, as follows-

```
overall
5.0    30708
4.0     6703
3.0     3363
1.0     1834
2.0     1392
```

The skewed distribution might lead to a sampling bias towards the majority '5.0' class.

1.2 Improved sampling:

To address the class imbalance, stratified sampling is used to take equal samples per class.
400 examples are sampled for each of the 5 rating classes from each of the json files. This guarantees that skewed datasets won't bias the model and also provides a balanced dataset for more robust model training.

```
overall
5.0    8800
4.0    8283
3.0    7918
1.0    7165
2.0    7099
```

2. Text Preprocessing

Goal: Prepare the text data for building NLP classification models
Text Columns: 'reviewText', 'summary'

Preprocessing Pipeline:

Preprocessing Steps:

- Lowercasing: Convert text to lowercase to reduce variance.
- Remove special characters: Regex is used to remove non-alphabetical characters. This reduces noisy symbols.
- Tokenization: Text is split into words/tokens using NLTK's word_tokenize. This is needed before removing stopwords.
- Stopword Removal: NLTK's stopwords list is used to remove common irrelevant words like 'the', 'is', etc. Custom stopwords like 'not', ‘very' are retained.
  The reasons to retain important stopwords during text preprocessing for sentiment analysis are:
  - To preserve negation context - e.g. "not", "no"
  - To capture sentiment intensity - e.g. "very", "extremely"
  - To understand time context - e.g. "now", "today"
  - To analyze comparative sentiment - e.g. "better", "worse"
  - To get frequency context - e.g. "many", "few"
  - To identify contrasting sentiment - e.g. "but", "however"
- Stemming: Porter stemmer reduces words to their root form. For e.g. 'making' is converted to 'make'. This reduces sparsity.

New Columns: 'processed_reviewText' and 'processed_summary' contain the preprocessed text.
Overall, preprocessing reduces noise and sparsity.

3. Set of terms that best describe the corpus

3.1 TF-IDF Analysis for Unigrams:

Utilized the TfidfVectorizer from scikit-learn to calculate TF-IDF scores for unigrams in the processed review text. Extracted the top 10 terms with the highest TF-IDF scores for each unique rating (1.0, 2.0, 3.0, 4.0, 5.0).

rating 5.0: ['great', 'love', 'good', 'work', 'veri', 'product', 'use', 'one', 'like', 'but']
rating 4.0: ['good', 'but', 'great', 'work', 'use', 'like', 'not', 'veri', 'game', 'one']
rating 3.0: ['but', 'not', 'ok', 'good', 'work', 'use', 'like', 'game', 'one', 'veri']
rating 1.0: ['not', 'but', 'work', 'use', 'one', 'like', 'veri', 'product', 'game', 'get']
rating 2.0: ['not', 'but', 'use', 'like', 'veri', 'one', 'work', 'game', 'get', 'good']

3.2 TF-IDF Analysis for Bigrams:

Extended the analysis to include bigrams (pairs of adjacent words) by setting ngram_range=(2, 2) in the TfidfVectorizer. Identified the top 10 bigrams with the highest TF-IDF scores for each rating to capture meaningful word combinations.

rating 5.0: ['work great', 'great product', 'veri good', 'good product', 'gift card', 'great price', 'work well', 'great gift', 'veri nice', 'excel product']
rating 4.0: ['work great', 'work well', 'veri good', 'good product', 'leaf blower', 'pull rod', 'rod brush', 'but not', 'drill attach', 'rod get']
rating 3.0: ['dryer vent', 'but not', 'lint catcher', 'clean lint', 'lose rod', 'brush broke', 'climb attic', 'make sure', 'put togeth', 'bit more']
rating 1.0: ['not work', 'wast money', 'would not', 'veri disappoint', 'not buy', 'not recommend', 'not worth', 'didnt work', 'not good', 'but not']
rating 2.0: ['but not', 'not good', 'would not', 'not veri', 'not work', 'work well', 'dont like', 'not like', 'not worth', 'not buy']

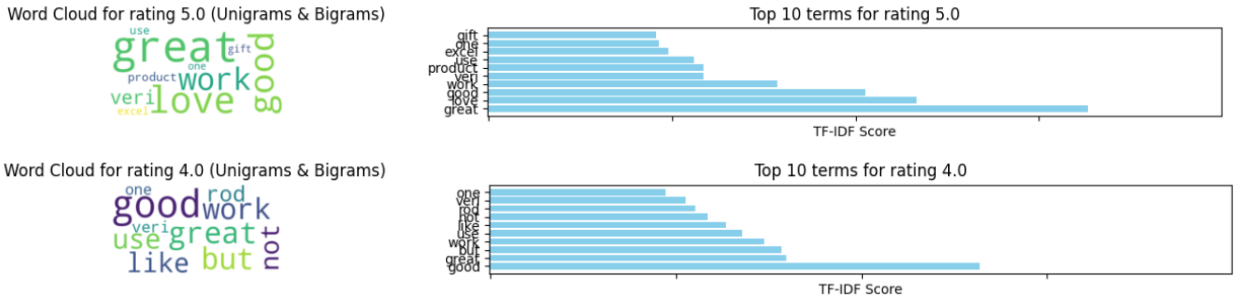
3.3 Combined Unigrams and Bigrams Analysis:

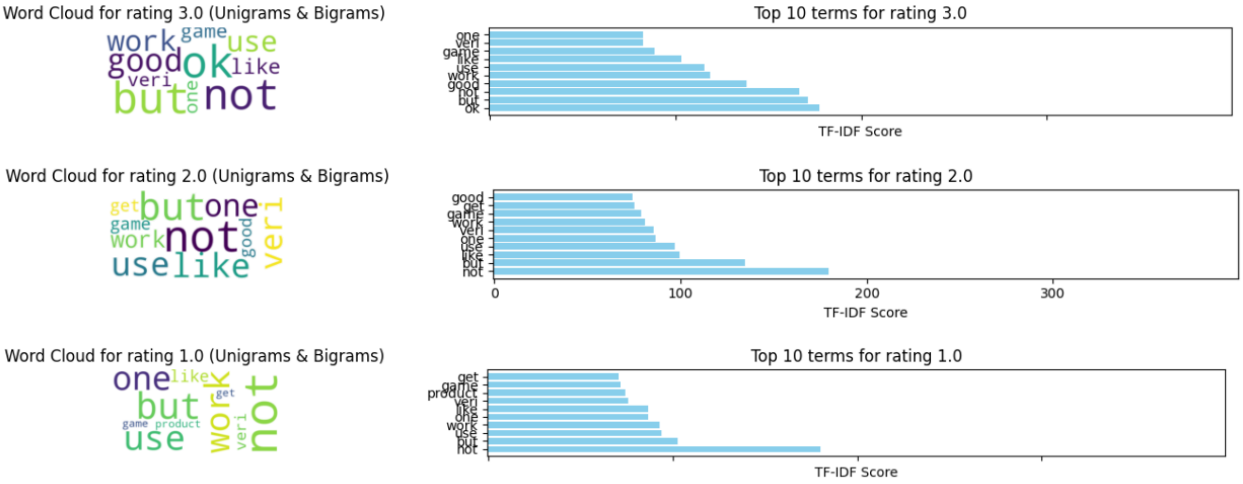
Performed TF-IDF analysis on the combination of unigrams and bigrams by setting ngram_range = (1, 2). Obtained the top 10 terms (unigrams and bigrams) with the highest TF-IDF scores for each rating, providing a comprehensive view.

rating 5.0: ['great', 'love', 'good', 'work', 'veri', 'product', 'use', 'excel', 'one', 'gift']
rating 4.0: ['good', 'great', 'but', 'work', 'use', 'like', 'not', 'rod', 'veri', 'one']
rating 3.0: ['ok', 'but', 'not', 'good', 'work', 'use', 'like', 'game', 'veri', 'one']
rating 1.0: ['not', 'but', 'use', 'work', 'one', 'like', 'veri', 'product', 'game', 'get']
rating 2.0: ['not', 'but', 'like', 'use', 'one', 'veri', 'work', 'game', 'get', 'good']

3.4 Visualization

Implemented word clouds and bar charts for effective visualization of the most significant terms.





3.5 Analysis

The analysis reveals distinct patterns in customer sentiment. Higher ratings (5.0, 4.0) are characterized by positive terms, like 'great,' 'love,' and 'good', emphasizing satisfaction and positive experiences, while lower ratings (1.0, 2.0) feature negative terms like 'not,' 'disappoint,' and 'not worth' indicating dissatisfaction. Bigrams offer additional context, highlighting specific concerns and product-related challenges. The combined analysis provides a nuanced understanding of sentiment, enabling businesses to focus on both overall satisfaction and targeted improvements for enhanced customer experience.

4. *Train-Validation split*

Split the dataset into training and validation sets using train_test_split with a test size of 20%.
Train set size: 31412 samples
Validation set size: 7853 samples

5. Multi-Class Classification

5.1 Naive Bayes classifier

- Gaussian Naive Bayes
    - Assumes continuous feature values
    - Models using Gaussian normal distributions
    - Mean and variance calculated from training data
    - Predicts class based on Bayes theorem
- Multinomial Naive Bayes

- Assumes discrete feature values
- Models using multinomial distributions
- Calculates class probabilities using feature frequencies
- Predicts class with highest probability based on Bayes

5.1.1 Bag of Words (BoW) Approach with ReviewText

Utilized CountVectorizer to transform the processed review text into a bag of words representation. Then we applied both the Gaussian Naive Bayes and the Multinomial Naive Bayes.

5.1.2 Bag of Words (BoW) Approach with summary

Utilized CountVectorizer to transform the processed summary into a bag of words representation. Then we applied both the Gaussian Naive Bayes and the multinomial Naive Bayes.

5.1.3 TF-IDF Approach with review + summary

TfidfVectorizer is used to convert text to weighted term frequency vectors. It transforms text into numerical feature vectors with TF-IDF weights. The review and summary TF-IDF vectors are concatenated vertically using hstack. This combines features from both texts into one consolidated vector. GaussianNB and MultinomialNB models are trained on the vectors.

5.1.4 Result

Assessed model performance using accuracy, precision, recall, F1 score, and confusion matrix.

GaussianNB

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	34.9 %	33.8 %	42. 6 %
Precision	37.8 %	39.5%	42.8 %
Recall	34.9 %	33.8 %	42.6 %
F1 Score	30.9 %	28.27%	40.9 %
Confusion	[[456 124 77 66 653]	[[443 55 62 61 755]	[[589 171 118 107 391]

Matrix	[314 262 128 82 656] [229 127 250 132 847] [164 106 113 284 976] [107 47 57 107 1489]]	[245 158 116 101 822] [137 70 205 128 1045] [65 46 72 211 1249] [33 33 39 60 1642]]	[374 364 188 136 380] [267 182 377 187 572] [170 125 170 677 501] [115 81 98 167 1346]]

MultinomialNB

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	48.27 %	57.01 %	63.7 %
Precision	48.21 %	56. 74 %	64.05 %
Recall	48.27 %	57.01 %	63.72 %
F1 Score	48.10 %	56.82 %	63.68 %
Confusion Matrix	[[799 317 126 74 60] [379 593 263 142 65] [218 388 601 259 119] [119 187 315 635 387] [107 81 139 317 1163]]	[[840 258 139 69 70] [333 618 283 121 87] [151 241 824 226 143] [78 99 239 923 304] [83 56 104 292 1272]]	[[938 212 138 65 23] [285 676 319 124 38] [116 200 940 270 59] [45 64 219 1108 207] [36 32 83 314 1342]]

5.2 Decision Tree

- Entropy Criteria
  - Created a Decision Tree Classifier using the entropy criterion.
  - Entropy measures the impurity of a node in a decision tree.
  - Aims to maximize information gain at each split for better classification.
- Gini Criteria
  - Constructed another Decision Tree Classifier using the Gini impurity criterion.
  - Gini impurity measures the level of impurity or disorder in a node.
  - Seeks to minimize Gini impurity at each node for optimal splits.

5.2.1 Bag of Words (BoW) Approach with ReviewText

Utilized CountVectorizer to transform the processed review text into a bag of words representation. Then we applied both the Entropy Criteria and the Gini Criteria.

5.2.2 Bag of Words (BoW) Approach with summary

Utilized CountVectorizer to transform the processed summary into a bag of words representation. Then we applied both the Entropy Criteria and the Gini Criteria.

5.2.3 TF-IDF Approach with review + summary

Tfidf Vectorizer is used to convert text to weighted term frequency vectors. It transforms text into numerical feature vectors with TF-IDF weights. The review and summary TF-IDF vectors are concatenated vertically using hstack. This combines features from both texts into one consolidated vector. Entropy Criteria and the Gini criteria decision tree models are trained on the vectors.

5.2.4 Result

Assessed model performance using accuracy, precision, recall, F1 score, and confusion matrix.

Entropy Criteria

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	44.09 %	56.18 %	53.64 %
Precision	44.03 %	56.33 %	53.65 %
Recall	44.09 %	56.18 %	53.64 %
F1 Score	44.06 %	55.97 %	53.64 %
Confusion Matrix	[[653 303 191 127 102] [309 518 278 183 154] [210 276 633 275 191] [128 194 292 653 376] [96 139 194 372 1006]]	[[912 215 118 64 67] [412 653 203 104 70] [245 251 745 205 139] [136 83 228 875 321] [114 63 121 282 1227]]	[[723 321 145 85 102] [292 658 241 145 106] [167 240 803 235 140] [117 139 263 840 284] [101 110 133 274 1189]]

Gini criteria

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	44.67 %	56.16 %	54.67 %
Precision	44.60 %	56.28 %	54.66 %
Recall	44.67 %	56.16 %	54.67 %
F1 Score	44.63 %	55.97 %	54.66 %
Confusion Matrix	[[671 299 171 132 103] [292 519 305 186 140] [215 292 634 257 187] [145 185 285 663 365] [96 132 180 378 1021]]	[[917 230 102 56 71] [398 668 219 95 62] [239 252 749 212 133] [132 87 244 856 324] [110 67 118 291 1221]]	[[776 267 159 89 85] [292 658 255 140 97] [168 238 816 238 125] [116 148 252 835 292] [68 123 131 276 1209]]

5.3 Random Forest

- 20 Trees
  - Constructed a Random Forest Classifier with 20 decision trees.
  - Each decision tree was built on a subset of the training data with replacement.
- 50 Trees
  - Created another Random Forest Classifier with 50 decision trees.
  - Increased the number of trees to explore model robustness and reduce overfitting.
- 100 Trees
  - Developed a third Random Forest Classifier with 100 decision trees.
  - Further increased the number of trees for improved ensemble learning.

5.3.1 Bag of Words (BoW) Approach with ReviewText

Utilized CountVectorizer to transform the processed review text into a bag of words representation. Then we applied Random forest with varying numbers of trees.

5.3.2 Bag of Words (BoW) Approach with summary

Utilized CountVectorizer to transform the processed summary into a bag of words representation. Then we applied Random forest with varying numbers of trees.

5.3.3 TF-IDF Approach with review + summary

Tfidf Vectorizer is used to convert text to weighted term frequency vectors. It transforms text into numerical feature vectors with TF-IDF weights. The review and summary TF-IDF vectors are concatenated vertically using hstack. This combines features from both texts into one consolidated vector. Random forest with varying numbers of trees are trained on these texts.

5.3.4 Result

Assessed model performance using accuracy, precision, recall, F1 score, and confusion matrix.

20 Trees

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	49.07 %	57.53 %	58.98 %
Precision	48.7 %	57.41 %	59.15 %
Recall	49.07 %	57.53 %	58.98 %
F1 Score	48.72 %	57.26 %	58.78 %
Confusion Matrix	[[785 291 133 76 91] [371 537 312 124 98] [249 271 657 248 160] [117 196 269 663 398] [80 94 166 255 1212]]	[[893 255 108 55 65] [370 691 208 96 77] [203 261 769 203 149] [104 109 207 855 368] [83 83 96 235 1310]]	[[979 234 78 48 37] [407 639 241 93 62] [226 233 835 194 97] [107 154 255 856 271] [82 89 102 211 1323]]

50 Trees

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	51.62 %	58.16 %	61.33 %
Precision	51.62 %	58.05 %	61.20 %

Recall	51.19 %	58.16 %	61.33 %
F1 Score	51.13 %	57.88 %	60.99 %
Confusion Matrix	[[862 274 130 46 64] [378 559 318 89 98] [203 250 736 230 166] [94 164 292 655 438] [81 91 138 255 1242]]	[[908 244 105 51 68] [364 703 206 91 78] [190 259 779 201 156] [96 96 215 868 368] [82 79 103 233 1310]]	[[1013 211 66 47 39] [384 672 243 87 56] [201 236 868 180 100] [87 129 233 881 313] [64 67 76 217 1383]]

100 Trees

	BoW+Review	***BoW+Summary***	***TF-IDF+Summary+Review***
Accuracy	52.56 %	57.90 %	62.54 %
Precision	51.94 %	57.75 %	62.37 %
Recall	52.56 %	57.90 %	62.54 %
F1 Score	51.92 %	57.61 %	62.09 %
Confusion Matrix	[[905 246 112 48 65] [396 533 331 89 93] [207 235 750 230 163] [84 117 325 670 447] [64 78 122 273 1270]]	[[904 250 102 54 66] [377 688 209 88 80] [197 253 784 200 151] [100 94 224 860 365] [83 70 99 244 1311]]	[[1042 190 69 32 43] [387 681 235 74 65] [197 222 883 179 104] [73 115 232 889 334] [66 57 71 196 1417]]

5.4 Conclusion

- For both Naive Bayes and Decision Tree models, using TF-IDF vectors of combined review + summary text provided the best performance across all evaluation metrics.
- Increasing model complexity generally helped - Gaussian NB outperformed Multinomial NB and Random Forest with more trees performed better than lower.
- On the combined TF-IDF features, Gaussian NB achieved the highest accuracy of 63.7%. Multinomial NB was close second at 63.7%.
- For Random Forest, accuracy improved from 58.98% with 20 trees to 62.54% with 100 trees on the TF-IDF vectors.

- Precision, recall, F1 scores all showed similar trends, with TF-IDF features on complex models performing the best.
- Confusion matrices revealed some class pairs with high misclassification due to semantic similarities.

6. Binary-Class Classification

Transformed the original multi-class classification problem into a binary classification task with the rating of 1.0 and 2.0 as negative and 3.0, 4.0 and 5.0 as positive. Focused on distinguishing between negative and positive sentiments.

6.1 Naive Bayes classifier

6.1.1 Bag of Words (BoW) Approach

Gaussian Naive Bayes Accuracy (Binary Classification): 43.49%
Multinomial Naive Bayes Accuracy (Binary Classification): 81.15 %

Gaussian Naive Bayes Accuracy (Binary Classification): 0.4349929963071438
Precision (Binary Classification): 0.802020202020202
Recall (Binary Classification): 0.1576961271102284
F1 Score (Binary Classification): 0.26356846473029044
Confusion Matrix:
[[2622 196]
[4241 794]]

Multinomial Naive Bayes Accuracy (Binary Classification): 0.8115369922322679
Precision (Binary Classification): 0.839541547277937
Recall (Binary Classification): 0.8728897715988083
F1 Score (Binary Classification): 0.8558909444985394
Confusion Matrix:
[[1978 840]
[640 4395]]

6.1.2 TF-IDF Approach

Gaussian Naive Bayes Accuracy with TF-IDF (Binary Classification): 54.90 %
Multinomial Naive Bayes Accuracy with TF-IDF (Binary Classification): 83.47 %

Gaussian Naive Bayes Accuracy with TF-IDF (Binary Classification): 0.5490895199286897

Precision (Binary Classification): 0.7936320754716981
Recall (Binary Classification): 0.4009930486593843
F1 Score (Binary Classification): 0.5327879667502309
Confusion Matrix:
[[2293 525]
 [3016 2019]]

Multinomial Naive Bayes Accuracy with TF-IDF (Binary Classification): 0.8347128485928944
Precision (Binary Classification): 0.8219886265724625
Recall (Binary Classification): 0.9473684210526315
F1 Score (Binary Classification): 0.8802362059420558
Confusion Matrix:
[[1785 1033]
 [265 4770]]

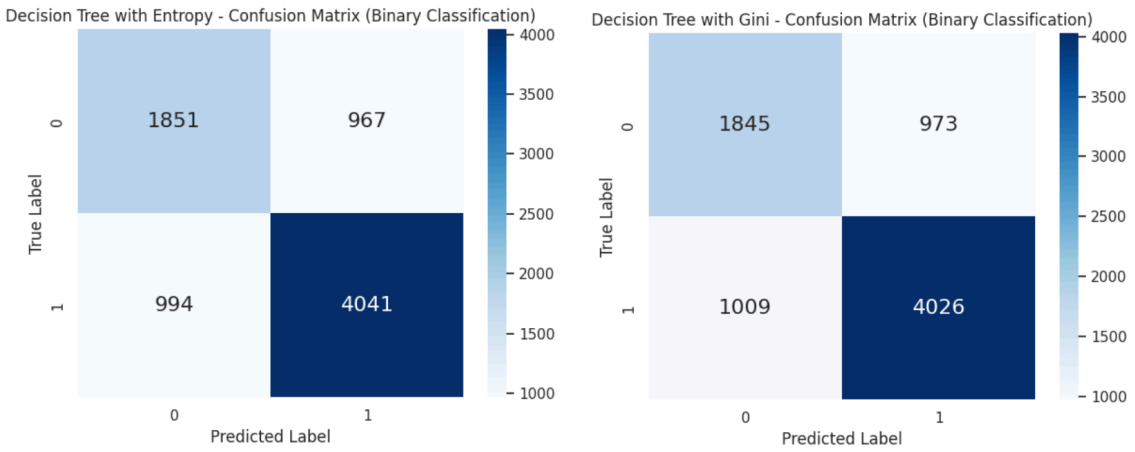
6.2 Decision Tree

6.2.1 Bag of Words (BoW) Approach

Decision Tree Accuracy with Entropy (Binary Classification): 75.02 %
Decision Tree Accuracy with Gini (Binary Classification): 74.76 %

6.2.2 TF-IDF Approach

Decision Tree Accuracy with Entropy (Binary Classification): 78.49 %
Decision Tree Accuracy with Gini (Binary Classification): 78.78 %



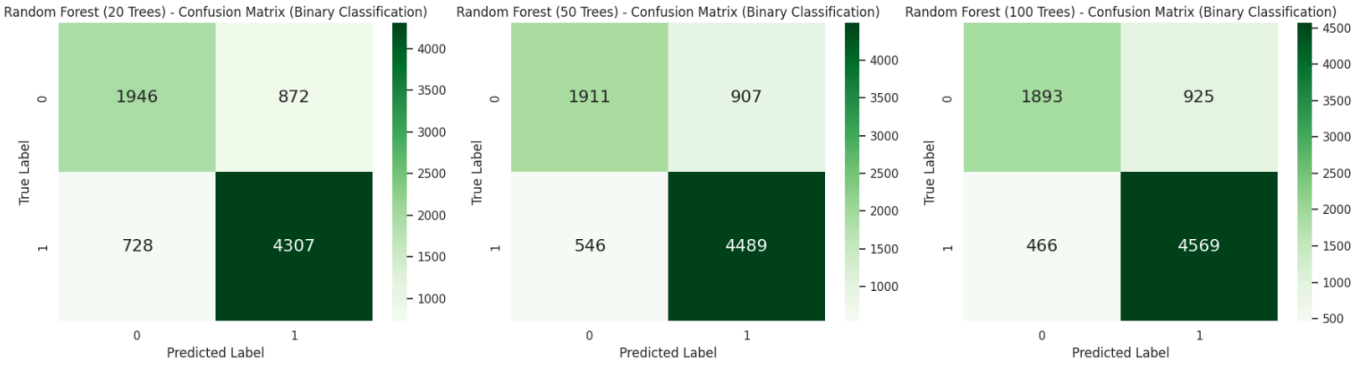
6.3 Random Forest

6.3.1 Bag of Words (BoW) Approach

Random Forest Accuracy with 20 Trees (Binary Classification): 79.62 %
Random Forest Accuracy with 50 Trees (Binary Classification): 81.49 %
Random Forest Accuracy with 100 Trees (Binary Classification): 82.28 %

6.3.2 TF-IDF Approach

Random Forest Accuracy with 20 Trees (Binary Classification): 80.03 %
Random Forest Accuracy with 50 Trees (Binary Classification): 81.80 %
Random Forest Accuracy with 100 Trees (Binary Classification): 82.49 %

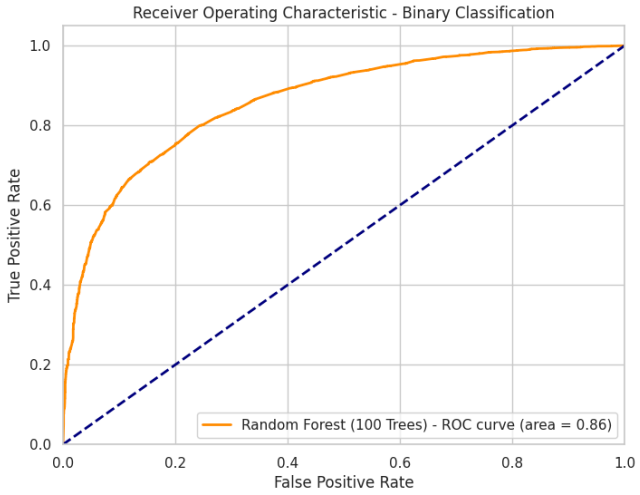


6.4 Conclusion

- Multinomial Naive Bayes consistently outperformed Gaussian NB, achieving over 80% accuracy.
- TF-IDF weighting provided significant improvements over bag-of-words for both Gaussian and Multinomial NB models.
- Decision trees also performed very well, with about 75% accuracy using bag-of-words. TF-IDF features improved it further to ~79%.
- Random forest achieved the best overall accuracy of 82-83% using TF-IDF weighted text features. Increasing ensemble size improved performance.
- Precision, recall and F1-score followed similar trends, with TF-IDF vectors on ensemble models performing the best.
- All models showed much better performance on the simplified binary task compared to the original multi-class problem.

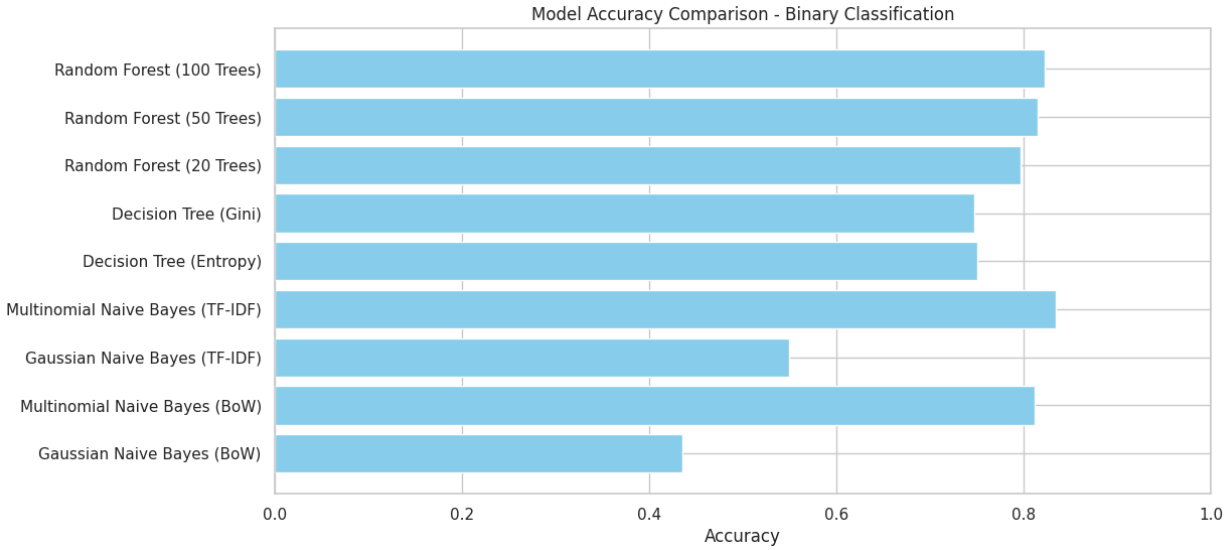
7. ROC Curve

Evaluation of Binary Classification Model: Utilized the ROC curve to assess the performance of the Random Forest model (100 Trees) in the binary classification task. Focused on distinguishing between false positive rate (FPR) and true positive rate (TPR) across different classification thresholds.



8. Model Accuracy Comparison

Compiled a comparison of accuracy scores across different binary classification models. Focused on assessing the performance of each model in distinguishing between two classes (positive and negative sentiments).



9. Hyperparameters

- Naive Bayes:
    - GaussianNB: No major hyperparameters to tune
    - MultinomialNB: Alpha smoothing hyperparameter for laplace/lidstone smoothing to handle zero probabilities. Default alpha=1.0 worked well.

- Decision Tree:
    - Criterion: 'entropy' and 'gini' tried. Gini performed slightly better.
    - Max depth: Not specified, allowed full growth. Pruning may improve.
    - Min samples split: Default of 2, could try higher values.

- Random Forest:
    - n_estimators: Number of trees varied from 20, 50, 100. More trees improved performance.
    - Max features: Default of 'auto' used. Could try tuning.
    - Max depth: Not specified, could restrict tree depth as hyperparameter.

Binary Classification Models: Additionally, class weights could be tried to handle class imbalance. But default equal weights worked reasonably well.

8. Erroneous results

8.1 Naive Bayes:

Assumes feature independence which is not valid for text data. Features have semantic relationships.
Cannot handle overlapping/related features well. For example "not good" vs "good".

8.2 Decision Tree:

Prone to overfitting text data easily, even with depth control.
High variance - small changes in data can result in very different trees.
Doesn't generalize well for test data with different feature distribution.

8.3 Random Forest:

Individual trees can still overfit local patterns in text data.
Difficult to interpret and analyze error sources in ensemble models.
Outliers and noise can cause isolated trees to learn erroneous patterns.

8.4 General Challenges:

Text data has very high dimensionality. Models struggle to learn useful representations.
Semantic similarities between some classes makes discrimination difficult.
Reviews require deeper language understanding beyond word frequencies.
Data imbalance can make models biased if not handled properly.

8.5 Conclusion:

Advanced NLP techniques are needed to capture semantic relationships in text.
Regularization, pruning and smoothing are essential to prevent overfitting.
Balancing datasets and tweaking model priors can help address class imbalances.
Ensembles handle overfitting better than single models, like trees.