Consumer Voice Analytics: Sentiment Tracking in E-Commerce

By

Madhumita Sai Srija Ganta
Sweta Hossain

Team 5

Introduction

In the rapidly evolving landscape of e-commerce, the ability to comprehend and respond to customer sentiments plays a pivotal role in achieving competitive advantage and fostering customer loyalty. Our project delves into the realm of consumer voice analytics, leveraging comprehensive customer review data to extract invaluable insights into consumer emotions, preferences, and purchasing patterns. This endeavor is driven by the recognition that in today's fiercely competitive environment, businesses must not only meet but exceed customer expectations to thrive.

By delving into the nuances of customer preferences and feedback through meticulous review analysis, businesses gain a nuanced understanding of their target audience. This understanding empowers them to tailor their marketing strategies more precisely, fine-tune existing products, and innovate new offerings that resonate deeply with consumer demands. Furthermore, the ability to track customer review trends over time equips companies with actionable intelligence for making strategic decisions, be it market expansion, diversification of product lines, or the strategic phasing out of underperforming products. In essence, this project embodies a strategic approach to leverage consumer voice analytics for driving business success, enhancing market competitiveness, and nurturing customer loyalty in the dynamic e-commerce landscape.

Project Goals

1. **Exploratory Data Analysis (EDA) to Preprocess and Understand the Data:**
    - ***Cleaning and Preprocessing:*** Identify and handle missing values, outliers, and inconsistencies in the dataset to ensure data integrity and reliability.
    - ***Feature Engineering:*** Extract relevant features from the raw data that can provide meaningful insights into customer behavior and preferences.
    - ***Initial Insights:*** Use visualizations, statistical summaries, and exploratory techniques to gain an initial understanding of the data distribution, trends, and patterns.
2. **Perform Sentiment Analysis:**
    - ***Text Preprocessing:*** Clean and preprocess review texts by removing stopwords, punctuation, and special characters.

- ***Sentiment Classification:*** Utilize Natural Language Processing (NLP) techniques such as sentiment lexicons or machine learning models to categorize review sentiments into positive, neutral, or negative classes.
- ***Sentiment Visualization:*** Visualize sentiment distributions and trends to understand the overall sentiment landscape of customer reviews.

3. **Analyze the Impact of Review Text Characteristics on Ratings**: Investigate the correlation between the length and detail of review texts and the ratings given.
4. **Conduct Trend Analysis Over Time:** Analyze changes in sentiments, ratings, and product popularity over time.

Technologies used:

1. **Pandas** for data manipulation and EDA.
2. **PySpark** for big data processing and transformation.
3. **Databricks** for collaborative analytics and ML on Spark clusters.
4. **TensorFlow** for NLP and sentiment analysis.
5. **Keras** for simplified deep learning model building.
6. **Scikit Learn** for various ML algorithms.
7. **Matplotlib** and **Seaborn** for data visualization

Models Used:

1. **TF-IDF** (Term Frequency-Inverse Document Frequency): The TF-IDF vectorizer converts text data from "reviewText" and "summary" into a feature matrix for sentiment analysis while capturing the importance of words based on their frequency and rarity across documents.
2. **XGBoost Regressor:** Utilized as a regressor, predicts sentiments (positive, neutral, negative) based on extracted features and is valued for its efficiency with large datasets and accurate regression predictions.
3. **Logistic regression:** This method is particularly useful for categorizing reviews into predefined categories based on extracted text features, providing a straightforward model to understand how textual features impact review sentiment.

4. **Random Forest Classifier:** Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode class (classification) or mean prediction (regression) of the individual trees.
5. **Regression Analysis:** It explores how review length/detail correlates with ratings, aiding in understanding how review characteristics influence customer satisfaction and ratings.
6. **Seasonal Decomposition of Time Series (STL):** It helps understand seasonal variations in ratings or sentiments by decomposing time series data into seasonal, trend, and residual components, providing insights into temporal patterns.

Exploratory Data Analysis:

The dataset central to this project comprises customer reviews curated from the "Amazon Fashion" category on the Amazon e-commerce platform. Each entry includes information such as overall rating (1 to 5 scale), purchase verification status, review date and time, reviewer and product identifiers, reviewer names, full review text, summary, Unix timestamp, review helpfulness votes, and product style details like color or size.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 883636 entries, 0 to 883635
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   overall         883636 non-null  float64
 1   verified        883636 non-null  bool
 2   reviewTime      883636 non-null  object
 3   reviewerID      883636 non-null  object
 4   asin            883636 non-null  object
 5   reviewerName    883544 non-null  object
 6   reviewText      882403 non-null  object
 7   summary         883103 non-null  object
 8   unixReviewTime  883636 non-null  int64
 9   vote            79900 non-null   object
 10  style           304569 non-null  object
 11  image           28807 non-null   object
dtypes: bool(1), float64(1), int64(1), object(9)
memory usage: 75.0+ MB
```

	overall
count	883636.00000
mean	3.90694
std	1.41828
min	1.00000
25%	3.00000
50%	5.00000
75%	5.00000
max	5.00000

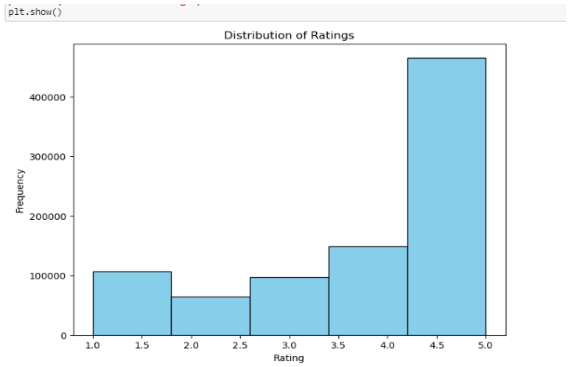
The dataset comprises 883,636 reviews with an average overall rating of 3.91 and a standard deviation of 1.42, indicating moderate variability around the mean rating. Ratings range from a minimum of 1 to a maximum of 5, with 25% of reviews rated 3 or lower, 50% rated 5 or lower (median), and 75% rated 5 or lower.

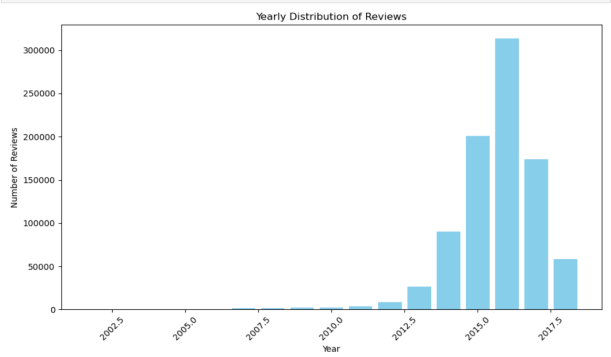
After preprocessing the text data, which involved breaking down the reviews into individual words (tokenization), reducing words to their root form (stemming), and eliminating common words that do not carry significant meaning (stopwords removal), a word cloud was generated.

This word cloud visually represents the frequency of words in the reviews, with larger words indicating higher frequency. The word cloud analysis reveals that the majority of reviews express positive sentiments. Customers seem to appreciate the quality of the products, indicating a good fit and satisfaction with their purchases. Additionally, buyers commonly mention that the items not only meet their expectations in terms of quality but also look appealing, enhancing their overall satisfaction with the products. This positive sentiment expressed across reviews suggests a favorable perception of the Amazon Fashion items within the dataset.

```
[('love', 54620.89533381057), ('fit', 37412.03276337915), ('great', 37243.63495861891), ('nice', 27963.999425694416), ('lik
e', 27065.359463195608), ('look', 26973.02149037537), ('size', 26726.740872783543), ('good', 26671.002912098716), ('small',
25355.060256519915), ('cute', 21340.14575611088)]
```

The top words extracted from the TF-IDF analysis reveal key themes in the processed text data. These words, including 'love', 'fit', 'great', 'nice', 'like', 'look', 'size', 'good', 'small', and 'cute', represent sentiments and aspects that customers commonly express in their reviews. The frequencies of these words range from approximately 21,340 occurrences for 'cute' to around 54,620 occurrences for 'love', indicating their prevalence in the dataset. Overall, these words predominantly convey positive experiences and perceptions, highlighting customers' satisfaction with various aspects of the Amazon Fashion products.





The distribution of ratings in the dataset indicates that the majority of items receive a rating of 5.0, reflecting a high level of satisfaction among customers. Conversely, there are relatively few items that receive low ratings, suggesting a generally positive perception of the Amazon Fashion products. Additionally, the year 2016 stands out as the year with the most reviews, totaling 313,629 reviews during that period. This significant volume of reviews in 2016 indicates a substantial level of customer engagement and feedback during that particular year, providing valuable insights into customer experiences and preferences during that time frame.

```
Top Reviewers with Names:
       reviewerID                reviewerName  num_reviews
0  A3G5KDMFNRUXHB  michelle williams brewster           40
1  A3JBQHQZEZPQK4                 Cookie C.            36
2  A1RRX286ZRI830                   Patrycja           32
3  A2PBHVTPTIIGKR               T.T. da BOSS           31
4  A2GP4EJIAA2OE0                         LJ           29
5  A1J7RPYGVGH5O3                shopaholic            27
6  A27WGWACHNQUR4                 Miss Lisa            26
7  A32M3PMH6DSLKD                    eunise            26
8   ALFRMOGTO1K4M               rosie thomas           25
9   AXRC1EH0U2WZ9                     sherry           24
```

The table represents the top reviewers along with their reviewer IDs, reviewer names, and the number of reviews they have contributed. Each row corresponds to a unique reviewer, identified by their reviewer ID, and their associated reviewer name. The 'num_reviews' column indicates how many reviews each reviewer has submitted. The top ten reviewers each have contributed more than 20 reviews total. They can offer valuable insights into customer opinions, preferences, and experiences with the Amazon Fashion products.

```
Products with Most Reviews:
            asin  num_reviews
3298   B000V0IBDM         4384
2453   B000KPIHQ4         4375
37176  B00I0VHS10         3889
71375  B00RLSCLJM         3638
2940   B000PHANNM         2572

Products with Highest Average Ratings:
            asin  avg_rating
0      0764443682         5.0
92972  B00Y88IVK6         5.0
92994  B00Y8MUW7M         5.0
92992  B00Y8LF6S8         5.0
92990  B00Y8FXRX0         5.0

Products with Most Variability in Ratings:
            asin  rating_std
181381  B01GREPK6Q    2.828427
39890   B00IUGHKBI    2.828427
39888   B00IUGHGLW    2.828427
144405  B01AYCIAHS    2.828427
10113   B0059R3Y9S    2.828427
```

```
Average Ratings - Top Reviewers: 4.108108108108108
Average Ratings - Regular Reviewers: 3.9068727783186543
Review Counts - Top Reviewers: 296
Review Counts - Regular Reviewers: 883340

Average Ratings - Verified Purchases: 3.9042740488404113
Average Ratings - Non-Verified Purchases: 3.9471576533119754
Review Counts - Verified Purchases: 828699
Review Counts - Non-Verified Purchases: 54937
```

Average ratings differ based on verified and non-verified purchases: verified purchases have an average rating of 3.90 out of 5, while non-verified ones rate slightly higher at 3.95 out of 5. Top reviewers, though fewer in number, give significantly higher ratings (4.11 out of 5) compared to regular reviewers (3.91 out of 5), indicating their influence on customer perceptions.

Sentiment Analysis:

XGBOOST:

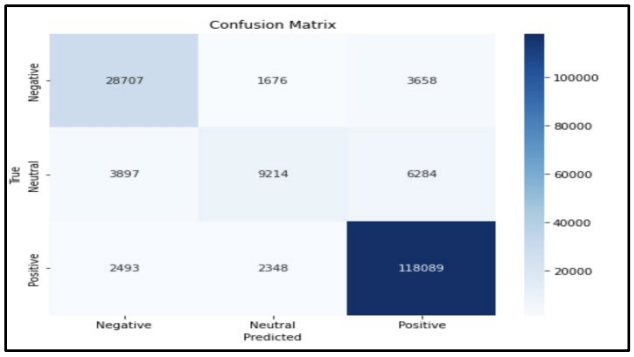
```
Accuracy: 0.8873240233579286
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.83      0.84     34673
           1       0.76      0.44      0.56     19333
           2       0.91      0.97      0.94    122722

    accuracy                           0.89    176728
   macro avg       0.84      0.75      0.78    176728
weighted avg       0.88      0.89      0.88    176728

Confusion Matrix:
[[ 28756   1295   4622]
 [  3448   8579   7306]
 [  1896   1346 119480]]
```

In this section of the report, we discuss the application and performance of the XGBoost classifier in the sentiment analysis task. XGBoost, which stands for eXtreme Gradient Boosting, is renowned for its high efficiency, scalability, and robust handling of various types of data, making it particularly suited for complex tasks such as text classification. The model was implemented on a dataset comprising textual reviews, categorized into three sentiment classes. The XGBoost classifier was trained using features transformed from text data into numerical format through the CountVectorizer method. This technique converts text data into a sparse matrix of token counts, providing a simple yet effective representation of text data for the model. The performance of the XGBoost model was evaluated using several metrics, including accuracy, precision, recall, and F1-score. The model achieved an accuracy of approximately **88.73%**, which is indicative of its robust ability to classify sentiments correctly across the dataset. There was some confusion between negative and neutral sentiments, as indicated by the misclassifications noted in the matrix. This suggests areas where the model might be improved, possibly through more nuanced feature engineering or by addressing class imbalance.

LOGISTIC REGRESSION:

This report section elaborates on the implementation and performance evaluation of a Logistic Regression model, specifically tailored for sentiment analysis within a big data context using PySpark. Logistic Regression is a probabilistic statistical model that predicts outcomes of categorical dependent variables based on one or more predictor variables. Its simplicity and efficiency in binary and multinomial classification make it particularly suitable for sentiment analysis tasks. The implementation involved creating a data processing pipeline in PySpark, which included tokenization of text data, hashing term frequency, and computing Inverse Document Frequency (IDF) to transform text into numerical features. Subsequently, the logistic regression model was configured for multinomial classification to predict three classes: negative, neutral, and positive sentiments. The model was trained on a substantial dataset, split into 80% for training and 20% for testing. After training, the model achieved an overall **accuracy of 88%.**

The Logistic Regression model's performance in sentiment analysis proved robust, especially in distinguishing positive sentiments. The simplicity and interpretative nature of Logistic Regression allow for easy scalability and adjustments based on feature engineering, which could further refine accuracy, particularly in distinguishing between neutral and other sentiment classes.



RANDOM CLASSIFIER:

This section of the report examines the utilization and performance assessment of a Random Forest Classifier for sentiment analysis, implemented using PySpark's machine learning library. The Random Forest algorithm is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputting the class that is the mode of the classes predicted by individual trees. The data pipeline was established with several preprocessing stages including tokenization, stop word removal, hashing term frequency, and inverse document frequency (IDF) calculation. These steps are crucial for transforming raw text data into a format suitable for machine learning models. The Random Forest Classifier was then applied, set to handle multiclass classification, which is ideal for categorizing text into multiple sentiment categories (negative, neutral, positive). The model was trained on an 80-20 split of the dataset and achieved an **accuracy of approximately 83.67%**. This indicates a solid capability of the classifier to distinguish between different sentiments accurately. The strength of Random Forest lies in its ability to manage large data sets with multiple features, making it a robust choice for text data which typically involves high dimensionality.

NEURAL NETWORKS:

This segment of the report focuses on the application and evaluation of a Neural Network model for sentiment analysis, implemented using PySpark's machine learning libraries. Neural Networks are highly adaptable models that mimic human brain operations and are capable of recognizing underlying patterns and relationships in large datasets.The neural network model in this case was configured using a Multilayer Perceptron Classifier, an architecture that includes an input layer, several hidden layers, and an output layer. The input data was first processed through a pipeline involving tokenization, removal of stop words, term frequency hashing, and inverse document

frequency calculations. These steps converted text into a numerical representation suitable for the neural network. The model was trained on a dataset split into 80% for training and 20% for testing, achieving an overall **accuracy of 86.57%.** This performance indicates a strong capability of the neural network to classify sentiments effectively. Neural networks excel in handling nonlinear data and learning complex patterns, making them particularly useful for text data, which often contains intricate relationships to be deciphered for sentiment analysis.

COMPARATIVE ANALYSIS OF MODELS:

In this section of the report, we conduct a comparative analysis of four different machine learning models applied to the task of sentiment analysis on text data.

Model Performance Overview

- XGBoost:
    - Accuracy: 88.73%
    - XGBoost demonstrated the highest accuracy and performed exceptionally well across all classes, particularly for positive sentiments. It benefits from gradient boosting techniques that focus on reducing errors sequentially, making it highly effective for structured datasets.
- Logistic Regression:
    - Accuracy: 88%
    - This model offered strong performance with high interpretability. It excelled in positive sentiment predictions but faced challenges in distinguishing between neutral and other sentiments, possibly due to overlapping linguistic features.
- Random Forest:
    - Accuracy: 83.67%
    - Known for its robustness against overfitting, the Random Forest model showed good general performance. However, it lagged slightly behind other models in terms of overall accuracy, potentially due to the randomness in selecting features which can sometimes neglect important predictors.
- Neural Network (Multilayer Perceptron):
    - Accuracy: 86.57%

- The neural network was effective in modeling non-linear relationships inherent in text data. Its performance was competitive, showcasing the potential of more complex architectures for deep learning applications in text analysis.

Comparative Analysis of Machine Learning Models' Performance and Resource Utilization in a Parallel Computing Environment

This section of the report presents a detailed comparison of four machine learning models—Neural Networks, XGBoost, Logistic Regression, and Random Forest—focusing on their performance in terms of compute time and resource utilization as scale increases.

Model Execution Overview

- Neural Networks:
  - **Required 20 Spark jobs and took approximately 35 minutes to complete**. This model, involving a complex structure with multiple layers, typically demands significant computational resources, which explains the higher number of Spark jobs and longer execution time.
  - Show the most substantial increase in time with scaling due to the complexity of the model and the extensive computations required for each layer and neuron connections.
- XGBoost:
  - **Took 3 Spark jobs and around 4-5 minutes to compute**. XGBoost is known for its efficiency and speed, especially in parallel processing settings, making it significantly faster than the more computationally intensive Neural Networks.
  - Although efficient, the increase in data volume can lead to longer training times primarily due to the sequential nature of boosting, where each tree builds upon the previous one.
- Logistic Regression:
  - **Required only 1 Spark job and took about 1-2 minutes to complete**. As a simpler model compared to the others, Logistic Regression benefits from lower computational overhead, resulting in very swift execution times
  - Being the simplest model, it scales efficiently with minimal impact on execution time even as data volume increases.

- Random Forest Classifier:
    - **Needed 6 Spark jobs and took about 6 minutes to complete**. Being an ensemble method involving multiple decision trees, it is naturally more resource-intensive than Logistic Regression but less so than Neural Networks
    - Exhibits a moderate increase in time with scaling, proportional to the number of trees and the depth of each tree which needs to be calculated during the training phase.

Resource Utilization

In terms of CPU and memory usage:

- Neural Networks and Random Forest are more resource-intensive, consuming more memory and CPU time, particularly as the complexity and size of the data increase.
- XGBoost and Logistic Regression are more efficient in resource usage, with Logistic Regression being the most resource-efficient due to its less complex calculations.

BEST MODEL SELECTION:

Given the analysis of accuracy, scalability, and resource utilization for Neural Networks, XGBoost, Logistic Regression, and Random Forest in a parallel computing environment, XGBoost stands out as the optimal model. It strikes an excellent balance between high accuracy and computational efficiency, managing large datasets effectively without excessive resource consumption. XGBoost's performance in speed and scalability, coupled with its robust handling of various data complexities, makes it particularly suitable for tasks requiring both precision and fast processing, such as sentiment analysis. This combination of features positions XGBoost as the best choice among the evaluated models, adaptable to diverse computing environments while delivering reliable and efficient results.

Analyze the Impact of Review Text Characteristics on Ratings:

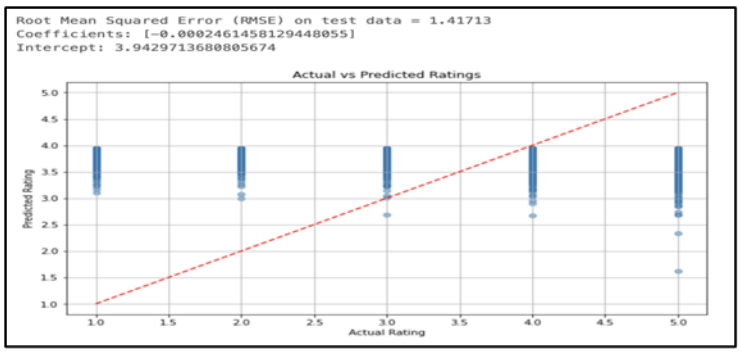
In this section of our report, we present a linear regression analysis that examines the relationship between the length of review texts and user ratings. The linear regression model was constructed using review text length as the independent variable and user ratings as the dependent variable. The model's parameters were estimated using a dataset where each entry consists of a user rating

and the corresponding length of the review text.

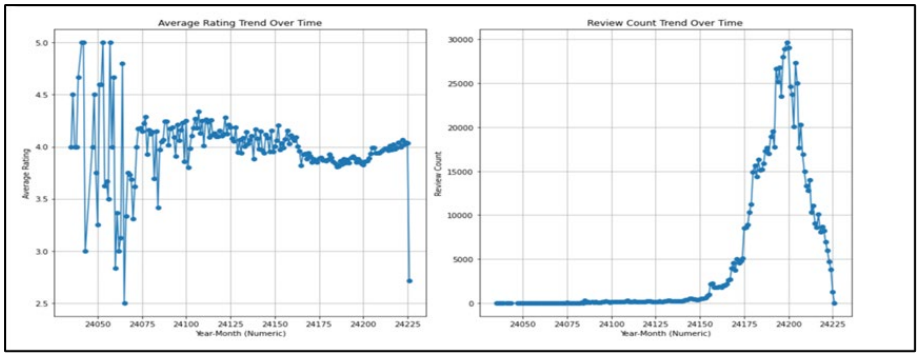
Results

- Root Mean Squared Error (RMSE): The RMSE on the test data was calculated at 1.41713, indicating the average difference between predicted ratings and actual ratings.
- Coefficients: The regression coefficient for review text length was -0.00246, suggesting a slight negative influence of review text length on ratings.
- Intercept: The model intercept was approximately 3.9429, which can be interpreted as the predicted rating when the review text length is zero.

The linear regression analysis indicates a moderate predictive ability of review text length on user ratings. **The slight negative correlation implies that longer reviews might be associated with marginally lower ratings, although the effect size is small.** The moderate RMSE value suggests that while the model's predictions are reasonably close to actual ratings, there is still a significant room for improvement, particularly in accurately predicting ratings at the extremes.



Conduct Trend Analysis Over Time



This section of the report analyzes trends in user ratings and review counts over a specified period. By examining these trends, we aim to gain insights into customer satisfaction, product reception, and market dynamics. The analysis was conducted using visualizations created from data aggregated on a monthly basis, showcasing the average ratings and the total number of reviews per month. These metrics were plotted to observe the changes over time, helping identify any significant trends, peaks, or troughs that could impact business strategies.

Analysis:

- Initially, the average ratings displayed significant fluctuations, ranging between 3.0 and 5.0. This variability could be indicative of varying customer expectations, initial product
- As time progressed, the average ratings stabilized around a 4.0 mark. This stabilization likely reflects improvements in product quality, consistency in customer service, or adaptation of the product to better meet consumer expectations.
- In the later periods, a gradual increase in average ratings was observed.
- The review counts initially were low, which is typical for new product introductions or when a product has limited market penetration.
- A sharp increase in review counts was noted midway through the timeline. This peak could be attributed to increased promotional activities, seasonal factors, or the product gaining popularity and thus receiving more consumer attention.
- There was a sharp decline in review counts following the peak period. This drop might be due to the natural life cycle of the product, possible market saturation, or changes in consumer preferences.

Conclusion:

This study emphasizes the crucial role of consumer voice analytics in e-commerce. By analyzing vast customer review data, it reveals insights into consumer preferences and behaviors, aiding strategic decision-making. Exploratory data analysis ensured dataset integrity, while sentiment analysis, powered by Natural Language Processing and Machine Learning models, accurately classified consumer sentiments, enhancing our understanding of the sentiment landscape.

Among the numerous algorithms tested, the XGBoost model stood out for its superior performance and remarkable mix of accuracy and efficiency. The logistic regression and neural network models also gave useful insights, while each had distinct areas for development.Our trend analysis highlighted significant patterns in consumer engagement over time, with notable fluctuations in average ratings and review counts. These trends are invaluable for predicting future market behaviors and adjusting business strategies accordingly.

Appendix for Code:

XGBoost:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)

# Convert text data into numerical features using CountVectorizer
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Initialize XGBoost classifier
xgb = XGBClassifier()

# Train the classifier
xgb.fit(X_train_vec, y_train)

# Make predictions on test data
y_pred = xgb.predict(X_test_vec)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Logistic Regression:

```
from pyspark.ml.feature import Tokenizer, HashingTF, IDF
from pyspark.ml import Pipeline

# Tokenize text
tokenizer = Tokenizer(inputCol="text", outputCol="words")

# Compute term frequency
hashingTF = HashingTF(inputCol="words",
outputCol="rawFeatures", numFeatures=10000)

# Compute the Inverse Document Frequency (IDF)
idf = IDF(inputCol="rawFeatures", outputCol="features")

# Create a pipeline to process the data
pipeline = Pipeline(stages=[tokenizer, hashingTF, idf])

# Fit the pipeline to the data
model = pipeline.fit(df1)

# Transform the data
df = model.transform(df1)
```

```
from pyspark.ml.classification import LogisticRegression

# Split the data into training and test sets
(train_set, test_set) = df.randomSplit([0.8, 0.2],
seed=1234)

# Initialize the logistic regression model for
multinomial (multiclass) classification
lr = LogisticRegression(featuresCol="features",
labelCol="label", maxIter=10, regParam=0.01,
family="multinomial")

# Train the model using the train set
lr_model = lr.fit(train_set)

# Make predictions on the test data
predictions = lr_model.transform(test_set)
```

Random Classifier:

```
# Split the data into training and testing sets
(trainingData, testData) = df.randomSplit([0.8, 0.2], seed=1234)

# Define stages for the pipeline
tokenizer = Tokenizer(inputCol="text", outputCol="words")
stop_words_remover = StopWordsRemover(inputCol=tokenizer.getOutputCol(), outputCol="filtered_words")
hashing_tf = HashingTF(inputCol=stop_words_remover.getOutputCol(), outputCol="raw_features")
idf = IDF(inputCol=hashing_tf.getOutputCol(), outputCol="features")

# Use RandomForestClassifier which supports multiclass classification
random_forest_classifier = RandomForestClassifier(featuresCol=idf.getOutputCol(), labelCol="label", numTrees=10)

# Create pipeline
pipeline = Pipeline(stages=[tokenizer, stop_words_remover, hashing_tf, idf, random_forest_classifier])

# Train the model
model = pipeline.fit(trainingData)

# Make predictions on the testing data
predictions = model.transform(testData)

# Evaluate the model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:", accuracy)
```

Neural Networks:

```
# Split data into training and testing sets
(trainingData, testData) = df.randomSplit([0.8, 0.2], seed=1234)

# Pipeline stages
tokenizer = Tokenizer(inputCol="text", outputCol="words")
remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
hashingTF = HashingTF(inputCol="filtered_words", outputCol="raw_features", numFeatures=10000)
idf = IDF(inputCol="raw_features", outputCol="features", minDocFreq=5)

# Neural network configuration
input_layer_size = 10000  # This should match the numFeatures in HashingTF
hidden_layers = [input_layer_size, 50, 3]  # Adjust hidden layer sizes and the output layer as needed
mlp = MultilayerPerceptronClassifier(layers=hidden_layers, blockSize=128, seed=1234, maxIter=100)

# Define the pipeline
pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, idf, mlp])

# Train the model
model = pipeline.fit(trainingData)

# Make predictions
predictions = model.transform(testData)

# Display example predictions
predictions.select("label", "prediction").show(10)

# Evaluate the model
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Accuracy: {accuracy}")

# Metrics and confusion matrix
predictions_and_labels = predictions.select("prediction", "label").rdd.map(lambda x: (float(x[0]), float(x[1])))
metrics = MulticlassMetrics(predictions_and_labels)
confusion_matrix = metrics.confusionMatrix().toArray()
print("Confusion Matrix:\n", confusion_matrix)

# Precision, Recall, and F1-Score
for label in sorted([0.0, 1.0, 2.0]):
    print(f"Class {int(label)} Precision: {metrics.precision(label)}")
    print(f"Class {int(label)} Recall: {metrics.recall(label)}")
    print(f"Class {int(label)} F1 Score: {metrics.fMeasure(label)}")
```

Analyze the Impact of Review Text Characteristics on Ratings:

```
# Assemble features to prepare for the model, ensuring data types are handled correctly
featureAssembler = VectorAssembler(inputCols=["reviewTextLength"], outputCol="features")
df_model = featureAssembler.transform(df1)

# Ensure all data types are correct for VectorAssembler, converting to float if necessary
df_model = df_model.select(col("features"), col("overall").cast("float").alias("label"))

# Split the data into training and testing sets
train_data, test_data = df_model.randomSplit([0.7, 0.3], seed=42)

# Define the Linear Regression model
lr = LinearRegression(featuresCol='features', labelCol='label')

# Train the model using the training data
lr_model = lr.fit(train_data)

# Make predictions on the test data
predictions = lr_model.transform(test_data)

# Evaluate the model
evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)

# Print the coefficients and intercept of the model
print("Coefficients: %s" % str(lr_model.coefficients))
print("Intercept: %s" % str(lr_model.intercept))

# Optionally, visualize the predictions vs actual ratings
predictions_pd = predictions.select("prediction", "label").toPandas()

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(predictions_pd['label'], predictions_pd['prediction'], alpha=0.5)
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.title('Actual vs Predicted Ratings')
plt.plot([1, 5], [1, 5], 'r--')  # Ideal line where predicted = actual
plt.grid(True)
plt.show()
```

Conduct Trend Analysis Over Time:

```
df1 = df1.withColumn("reviewDate", to_date(col("reviewDate"), "yyyy-MM-dd"))

# Group data by year and month to analyze trends over time
trends = df1.groupBy(year("reviewDate").alias("Year"), month("reviewDate").alias("Month")) \
            .agg(avg("overall").alias("AverageRating"),
                 count("reviewerID").alias("ReviewCount"),  # Use an existing column like 'reviewerID'
                 avg(length(col("reviewText"))).alias("AverageReviewTextLength"))

# Order the results by year and month
trends_ordered = trends.orderBy("Year", "Month")

# Show the trend results
trends_ordered.show()

# Optionally, convert to pandas for visualization
pandas_df = trends_ordered.toPandas()

# Plotting the trends using matplotlib
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 7))
plt.subplot(1, 2, 1)
plt.plot(pandas_df['Year'] * 12 + pandas_df['Month'], pandas_df['AverageRating'], marker='o')
plt.title('Average Rating Trend Over Time')
plt.xlabel('Year-Month (Numeric)')
plt.ylabel('Average Rating')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(pandas_df['Year'] * 12 + pandas_df['Month'], pandas_df['ReviewCount'], marker='o')
plt.title('Review Count Trend Over Time')
plt.xlabel('Year-Month (Numeric)')
plt.ylabel('Review Count')
plt.grid(True)

plt.tight_layout()
plt.show()

```