

**Design and Implementation of Analytics System**

*Detecting Spam Reviews on E-commerce Using ML*

*Abhishek Reddy Tetali*

*Krishna Chaitanya Reddy Atla*

**School of Graduate Professional Studies**

Data Analaytics

DAAN 888 – Design and Implementation of Analytics System

Fall, 2024

# Document Control

## Work carried out by:

|  |  |  |
| --- | --- | --- |
| **Name** | **Email Address** | **Task description** |
| *Abhishek Reddy Tetali* | [aft5636@psu.edu](mailto:aft5636@psu.edu) |  |
| *Krishna Chaitanya Reddy Atla* | kma6396@psu.edu |  |
|  |  |  |

## Revision Sheet

|  |  |  |
| --- | --- | --- |
| Release No. | Date | Revision Description |
| 1 | 9/8/2024 | Assignment 1 Submission - Problem Statement Definition |
| 2 | 9/22/2024 | Assignment 2 Submission - Data Collection |
| 3 | 10/06/2024 | Assignment 3 Submission |
| 4 | 10/20/2024 | Assignment 4 Submission |
| 5 | 11/3/2024 | Assignment 5 Submission |
| 6 | 11/17/2024 | Assignment 6 Submission |
| 7 | 12/5/2024 | Final Submission |

**TABLE OF CONTENTS**

1. General Guidelines
2. Week 2 Predictive / Descriptive Analytics System Group-Based Assignment
3. WEEK 4 Predictive/Descriptive Analytics System Group-Based Assignment
4. WEEK 6 Predictive Analytics System Group-Based Assignment
5. WEEK 8 Predictive/descriptive Analytics System Group-Based Assignment
6. WEEK 10 Predictive/descriptive Analytics System Group-Based Assignment
7. WEEK 12 Predictive/descriptive Analytics System Group-Based Assignment
8. WEEK 13 Predictive/descriptive Analytics System Group-Based Assignment
9. WEEK 13 Predictive/descriptive Analytics System Group-Based Assignment

**General Guidelines**

1. To complete all the homework assignments for this course please use this template document.
2. Each assignment has to be submitted by Sunday 11:59 PM EST.
3. Each figure should be followed by a brief description about the figure.
4. The figures can be hand drawn and scanned in some circumstances, but the hand drawn figure should be clear and legible to obtain full credits. Unclear hand drawn figures will receive partial credits. For constructing figures and diagrams it is advised to use tools.
5. Figures and tables should have appropriate captions. For documenting and referencing styles please follow the APA or MLA writing style.
6. Please make sure that you provide a reference section.
7. Any material text or figure taken from books, journals or the Internet should be referenced. If you have a sentence or a figure that does not belong (authorship) to you, they need to be clearly referenced. If you fail to do so your report will be considered as a case of plagiarism. It is your duty to make sure that your report is free from any activity related to plagiarism. In case you are suspected of attempting plagiarism then you will be responsible for the cause. The penalty for plagiarism will be “0” awarded to your report. So, it is good to keep simple, always have the principle to acknowledge people for their contributions.

Please go through the following instructions before submitting the report

#### **Academic Integrity**

Academic integrity — scholarship free of fraud and deception — is an important educational objective of Penn State. Academic dishonesty can lead to a failing grade or referral to the [Office of Student Conduct](http://www.sa.psu.edu/ja/).

Academic dishonesty includes, but is not limited to:

* cheating
* plagiarism
* fabrication of information or citations
* facilitating acts of academic dishonesty by others
* unauthorized prior possession of examinations
* submitting the work of another person or work previously used without informing the instructor and securing written approval
* tampering with the academic work of other students
* Use of the AI tools including ChatGPT to generate contents

#### How Academic Integrity Violations Are Handled

In cases where academic integrity is questioned, [procedure requires an instructor to notify a student](http://www.psu.edu/oue/aappm/G-9-academic-integrity.html) of suspected dishonesty before filing a charge and recommended sanction with the college. Procedures allow a student to accept or contest a charge. If a student chooses to contest a charge, the case will then be managed by the respective college or campus Academic Integrity Committee. If a disciplinary sanction also is recommended, the case will be referred to the [Office of Student Conduct](http://www.sa.psu.edu/ja/title=).

All Penn State colleges abide by this Penn State policy, but review procedures may vary by college when academic dishonesty is suspected. Information about Penn State's academic integrity policy and college review procedures is included in the information that students receive upon enrolling in a course.

Additionally, Penn State students are expected to act with civility and personal integrity; respect other students' dignity, rights, and property; and help create and maintain an environment in which all can succeed through the fruits of their own efforts. An environment of academic integrity is requisite to respect for oneself and others, and a civil community.

#### For More Information on Academic Integrity at Penn State

Please see the [Academic Integrity Chart](http://www.campuses.psu.edu/CAO.pdf)  for specific college contact information or visit one of the following URLs:

* Penn State Senate [Policy on Academic Integrity](http://www.psu.edu/dept/oue/aappm/G-9.html)
* [iStudy for Success!](http://istudy.psu.edu/tutorials/) — learn about plagiarism, copyright, and academic integrity through an educational module
* [Turnitin](http://tlt.its.psu.edu/turnitin) a web-based plagiarism detection and prevention system

# Week 2 Predictive / Descriptive Analytics System Group-Based Assignment

**Purpose:**

To describe the purpose/objectives of the proposed predictive analytics system (project)

**Tasks:**

1. After the meeting with the instructor in week 1 the group will put together a proposal for designing a predictive analytics system.
2. Clearly list the tasks that will be completed in the following weeks. I should have provided directions in regard to completing the tasks for different weeks during the week 1 meeting with the team.

**Objectives / Goals:** The objective is to assess how well different machine learning models perform on text categorization tasks, with an emphasis on spam detection. in online reviews. By experimenting with different pre-processing techniques and model architectures, this research seeks to determine the most effective approach for accurate spam classification.

**Business queries:**

1. To compare the performance of Perceptron, Multinomial Naive Bayes, Linear Support Vector Classifier models, and Logistic Regression XgBoost on original and pre-processed datasets.
2. To analyze the impact of lemmatization and data cleaning on the model's accuracy, precision, recall, and F1-score.
3. To identify the most effective model and pre-processing technique for spam detection.
4. To discuss the ethical considerations related to the use of machine learning models in spam detection.

**Overview of the Proposed Solution:**

**Preparing the Datasets (Week 3 – Week 6):**

* Data Exploration
* Data Cleaning

**Model Building and Training (Week 7 – Week 13):**

* Implementing ML Models like Logistic regression, Multinomial Naïve Bayes, Linear SVC, XGBoost and understand which suits best.
* Compare the results for the above models using visualizations.

**Data Collection:**

The Data related to our projct can be found in any e-commerece websites. We thought of searching Kaggle and we founda great data realted to Amazon Product reviews for different categories of products.

The link for the dataset is as follows: “<https://www.kaggle.com/datasets/naveedhn/amazon-product-review-spam-and-non-spam>”

# WEEK 4 Predictive/Descriptive Analytics System Group-Based Assignment

**Purpose:**

To layout the plan for collecting data for the predictive/descriptive analytic system.

**Tasks:**

1. After the class presentation in week 3 the group will put together a plan for collecting data for the proposed analytics system.
2. List the different sources for data collection. Also update the progress on the data collection (if any).
3. Clearly list the tasks that will be completed in the following weeks. I should have provided directions in regard to completing the tasks for different weeks during the week 3 meeting with the team.

**Data Understanding, Cleaning, and Preparation for Exploratory analysis:**

A recently released dataset is a valuable resource for studying spam detection in online reviews. This dataset includes Amazon product reviews labelled as spam or not spam, with a total of 15.4 million reviewers having left 26.7 million reviews. Each review is assigned a binary class, with "0" indicating that the review is not spam and "1" indicating that it is spam. This large dataset provides an excellent foundation for constructing and testing machine learning models for identifying and filtering spam reviews.

**Data Understanding –**

The dataset contains a significant class between spam and non-spam reviews. Out of the 1,997,140 reviews:

* Spam reviews (class 1: 1,662,754 (approximately 83.21% of the dataset)
* Non-spam reviews (class 0: 334,386 (approximately 16.79% of the dataset)

### Review of columns

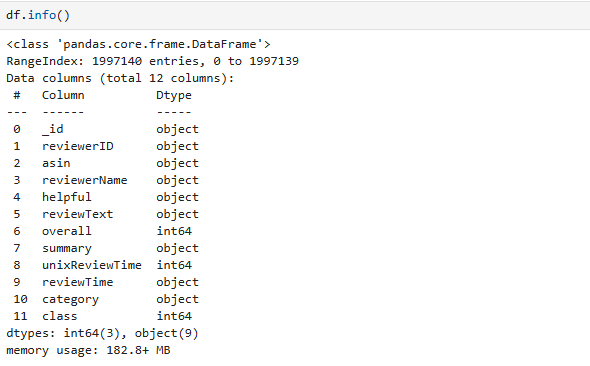
|  |  |  |
| --- | --- | --- |
| **Variable** | **Description (from OLBG)** | **Explanation** |
| \_id | Unique identifier for each review | This variable ensures that each review can be distinctly referenced and managed within the dataset. |
| reviewerID | ID of the reviewer | Used to track reviews made by individual users, helping in identifying patterns in user behavior. |
| asin | ID of the product being reviewed | Stands for Amazon Standard Identification Number, uniquely identifying the product in question. |
| reviewerName | Name of the reviewer | Provides the name of the person who wrote the review, useful for sentiment analysis and tracking |
| helpful | A tuple indicating the number of helpful votes and total votes for the review | Shows how many users found the review helpful, critical for assessing the review's credibility. |
| reviewText | The text of the review | |  | | --- | | The main content of the review, essential for natural language processing tasks like sentiment analysis. | |
| overall | The overall rating given by the reviewer | Indicates the reviewer’s overall sentiment towards the product, often on a scale from 1 to 5. |
| summary | A succinct synopsis of the evaluation | Provides a quick overview of the review content, useful for quick insights and analysis. |
| unixReviewTime | Unix timestamp of the review | The review's time in Unix timestamp format, useful for time-series analysis and sorting. |
| reviewTime | Date and time of the review | The human-readable date and time when the review was posted, aiding in temporal analysis. |
| category | Category of the product being reviewed | Helps in filtering and analysing reviews by product type, such as electronics, books, etc. |
| class | Binary label indicating whether the review is spam (1) or not spam (0) | determines whether a review is considered legitimate or spam, which is crucial for the identification of spam. |
| Year | Year of the review | Indicates the year when the review was posted, useful for longitudinal studies and trends analysis |
| Month | Month of the review | Indicates the month when the review was posted, adding granularity to temporal analysis. |
| helpful\_percentage | Percentage of helpful votes out of total votes for the review | Shows the proportion of users who found the review helpful, providing an immediate measure of usefulness. |

###### Table 2: Review of columns

Data type and measurement types of the variables:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Data type** | **Measurement type** |
| \_id | Integer/String | Nominal |
| reviewerID | String | Nominal |
| asin | String | Nominal |
| reviewerName | String | Nominal |
| helpful | Tuple (Integers) | Ratio |
| reviewText | String | Nominal |
| overall | Integer | Ordinal |
| summary | String | Nominal |
| unixReviewTime | Integer(Unix Timestamp) | Interval |
| reviewTime | Date Time | Interval |
| category | String | Nominal |
| class | Integer | Nominal |
| Year | Integer | Interval |
| Month | Integer | Ordinal |
| helpful\_percentage | Float (Percentage) | Ratio |

As most of variable in this dataset represent categories or labels they are nominal.



# WEEK 6 Predictive Analytics System Group-Based Assignment

**Purpose:**

To layout the plan for preparing data for conducting predictive/descriptive analytics.

**Tasks:**

1. After in class presentation in week 5 the group will put together a plan for preparing the data for performing analytics.
2. Submit a report with details on data collection. Your data collection step should have been completed by now.
3. List all the steps for preparing the data for predictive/descriptive modeling. I should have provided directions in regard to completing the tasks for preparing the data during the week 5 meeting with the team.

### Data Visualizations:

A red and blue rectangular object with white text

Description automatically generated

##### Figure 1 : Spam Ham class

Spam Reviews (Class 1): This dataset contains 1,662,754 spam reviews. These evaluations are marked as spam for a variety of reasons, including promotion, fraud, or dishonesty.  
The dataset includes 334,386 ham reviews classified as Class 0. Ham reviews, also known as non-spam or genuine reviews, are submitted by customers who have actually used the product or service. These evaluations contribute to the platform's legitimacy and credibility by providing valuable insights to other users seeking to make informed decisions.

A close up of words

Description automatically generated

##### Figure 2 : Word Cloud

* "bought": This word appears prominently, suggesting that many reviews mention purchasing items.
* "play", "game", "toy": These words are significantly large, indicating that the reviews frequently discuss items related to play, games, and toys.
* "year", "old": These terms often appear together, likely in the context of age-related comments, such as "year old" referring to the age of a child.

**Positive and Negative word Clouds**

A close-up of words

Description automatically generated

##### Figure 3 Positive and Negative word Clouds

How did we get positive and negative word clouds:

* We classified the positive and negative reviews based on overall rating with ratings >= 4 vs.<= 2 as positive and negative respectively.

**Positive Words Word Cloud:**

* "love": This word is one of the largest, indicating that it is frequently used in positive reviews.
* "play", "game": These terms are also large, suggesting that positive reviews often talk about the playability and enjoyment of games.
* "year", "old": These words appear together, likely in the context of describing age suitability for children.

**Negative Words Word Cloud:**

* "disappointed", "money": These words indicate common complaints about dissatisfaction and value for money.
* "product", "received": Frequent mentions of "product" and "received" might point to problems with the items themselves or with the delivery process.
* "not", "waste": These negative sentiments highlight dissatisfaction and regret.

**Overall Ratings Distribution Graph: -**

A graph of blue bars

Description automatically generated with medium confidence

##### Figure 4 : Overall Ratings Distribution Graph

The review ratings show a skewed distribution with a majority concentrated in the higher ratings (4 and 5). Out of total reviews, 1,275,445 are 5-star and 387,309 are 4-star, indicating that most customers were highly satisfied. In contrast, only 128,156 reviews are 1-star, and 77,132 are 2-star, reflecting limited strong dissatisfaction. This pattern suggests that the products or services generally meet or exceed customer expectations, with a predominance of positive experiences.

**Number of Reviews Over Time**

A graph showing the number of reviews over time

Description automatically generated

##### Figure 5 : Number of Reviews Over Time

The provided line graph depicts the number of reviews submitted between 1999 and 2014. The number of reviews is displayed on the y-axis, while the x-axis displays the years. Every dot on the graph denotes the quantity of reviews in a specific year.

**Analysis of the Graph:**

Early Years The low volume of reviews in the early years can be attributed to the infancy of e-commerce and online review systems. Consumers were less accustomed to sharing their opinions online.  
  
Adoption Phase: The gradual increase from 2007 to 2010 corresponds to the expansion of e-commerce and the rise of review platforms such as Amazon, Yelp, and others. During this stage, more consumers began to understand the importance of sharing and reading reviews.

A graph of different colored bars

Description automatically generated with medium confidence

##### Figure 6 : Purchase Analysis - Month wise

A graph with orange bars

Description automatically generated

Fig 7 : Distribution of Helpful Reviews

The majority of evaluations have a useful percentage of 0.00, which indicates that only non-helpful or no helpful votes were cast for them. The extremely tall bar at the 0.00 mark indicates this.This implies that a significant number of evaluations may not be considered beneficial by other users, or they may have been downvoted for being unhelpful.

Notable Increase at 1.00

# WEEK 8 Predictive/descriptive Analytics System Group-Based Assignment

**Purpose:**

To layout the plan for cleaning the data for conducting predictive / descriptive analytics.

**Tasks:**

1. After the class presentation in week 7 the team will complete the steps for data preparation.
2. List all the steps for variable (feature) selection and variable transformation. I should have provided directions in regard to completing the tasks for variable selection and transformation during the week 7 meeting with the team.

## Data Pre-Processing:

### Text Cleaning:

* **Removal of Punctuation:** Removing punctuation marks to ensure that the text is standardized and does not contain unnecessary characters.
* **Lowercasing:** Converting all text to lowercase in order to preserve consistency and prevent handling the same word differently depending on capitalization.
* **Removal of Stopwords:** Eliminating common stopwords (e.g., "and," "the," "is") that do not contribute significant meaning to the text.
* **Removal of Special Characters:** Stripping out any special characters that may not be relevant to the analysis.

### Lemmatization:

Conversion to Base Forms: Converting words to their base or root forms (e.g., "running" to "run") to reduce dimensionality and standardize terms.

### Creation of Different Datasets:

1. **Original Data:**

Variables: X\_train1, y\_train1, X\_test1, y\_test1

Description: The original dataset without any preprocessing.

1. **Data with Lemmatization:**

Variables: X\_train2, y\_train2, X\_test2, y\_test2

Description: The dataset where only lemmatization has been applied to standardize words to their root forms.

1. **Cleaned Data:**

Variables: X\_train3, y\_train3, X\_test3, y\_test3

Description: The dataset after text cleaning (removal of punctuation, lowercasing, removal of stopwords, removal of special characters).

1. **Cleaned and Lemmatized Data:**

Variables: X\_train4, y\_train4, X\_test4, y\_test4

Description: The dataset that has undergone both text cleaning and lemmatization to ensure both standardization and dimensionality reduction.

# WEEK 10 Predictive/descriptive Analytics System Group-Based Assignment

**Purpose:**

To detail the variable selection and Transformation steps

**Tasks:**

1. After the class presentation in week 9 the team will complete the steps for variable selection and Transformation.
2. List all the steps for predictive analytics modeling. I should have provided the directions in regard to completing the tasks for predictive/descriptive analytics modeling during the week 9 meeting with the team.

**Data Transformation:**

**Vectorization:**

Text data, such as product reviews, is inherently unstructured and cannot be directly used as input for machine learning algorithms. Most machine learning models require numerical input, which means we must convert text into a numerical representation—a process called vectorization.

**Why is Vectorization Needed? Model Compatibility:**

Machine learning models like Logistic Regression, XGBoost, and others operate on numerical data. Without vectorization, these models cannot process textual data. Feature Extraction:

Vectorization transforms words into numerical features that represent their importance in the text data. This allows the model to identify patterns and relationships between words and classes (e.g., spam or non-spam). Handling Vocabulary:

The process creates a vocabulary of unique words from the dataset and assigns numerical values based on their frequency or significance, ensuring consistent representation. Improved Accuracy:

By providing numerical representations, vectorization enables models to effectively learn and make accurate predictions on text data.

We used TF-IDF Vectorization to transform the text data into numerical vectors. Here's an overview of the steps we performed:

Define a TF-IDF Vectorizer:

The TfidfVectorizer from scikit-learn was initialized with the following parameters: analyzer='word': Indicates we are analyzing words (rather than characters or n-grams). min\_df=0.0: No minimum document frequency threshold was applied. max\_features=2000: Limits the vocabulary size to the top 2,000 most important words, balancing performance and complexity. Prepare the Data:

We applied TF-IDF vectorization on four different variations of the review text: Original Reviews (reviews): The raw text data as it appears in the dataset. Lemmatized Reviews (reviews\_lemm): Text processed to convert words to their base or dictionary form (lemmatization). Cleaned Reviews (reviews\_cleaned\_text): Text where unnecessary characters, numbers, and stopwords were removed. Cleaned and Lemmatized Reviews (reviews\_cleand\_text\_lem): Text that was both cleaned and lemmatized for optimal processing. Transform the Data:

For each variation, we generated a numerical matrix (X1, X2, X3, X4) where rows represent reviews and columns represent the words in the vocabulary. Each cell in the matrix contains the TF-IDF score for a word in a specific review.

Why TF-IDF Was Chosen Relevance Over Frequency: Unlike BoW, TF-IDF accounts for how important a word is in the dataset by reducing the weight of common but uninformative words. Improved Classification Performance: TF-IDF emphasizes unique, meaningful words that are likely to improve model accuracy. Scalability: Limiting the vocabulary to 2,000 features helps handle large datasets efficiently without overloading memory.

Resulting Vectorized Data The vectorized data matrices (X1, X2, X3, X4) are sparse matrices, meaning most cells are zero (as not all words appear in all reviews). These matrices serve as input to machine learning models for training and evaluation.

X1 (Original Reviews): Captures the raw textual information. X2 (Lemmatized Reviews): Reduces redundancy by grouping similar word forms (e.g., "running," "runs," "ran" → "run"). X3 (Cleaned Reviews): Focuses on meaningful content by removing noise. X4 (Cleaned and Lemmatized Reviews): Combines cleaning and lemmatization for enhanced preprocessing.

`

# WEEK 12 Predictive / Descriptive Analytics System Group-Based Assignment

**Purpose:**

To detail the modelling, evaluation and validation of the Analytics system

**Tasks:**

1. After in class presentation in week 11 the team will complete the steps for modelling, evaluating and validation of the Analytics system.
2. Submit the report detailing the modelling, evaluating and validation of the Analytics system.

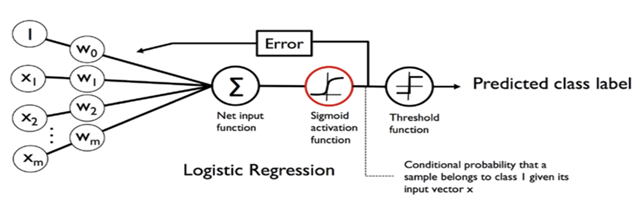
**Model Implementation plan:**

This section discusses the classification models utilized for detecting spam reviews in e-commerce. The primary goal is to identify reviews that exhibit spam characteristics using machine learning techniques.

### **Logistic Regression**

For this study, logistic regression was chosen because of its effectiveness and interpretability in situations involving binary categorization. Given its linear nature, it is effective for problems where a linear boundary can represent the relationship between features and the outcome (Jindal et al., 2007). Additionally, Logistic Regression provides probabilities for predictions, which is valuable for assessing the likelihood of a review being classified as spam.

The Logistic Regression model was initialized and configured with default settings to establish a baseline performance. The training involved learning the link between the review characteristics and their classification labels by fitting the model on the labeled training data. The model was used to predict the test set after training. The accuracy and performance of the model were then assessed using a variety of criteria. effectiveness in distinguishing spam reviews from non-spam ones.



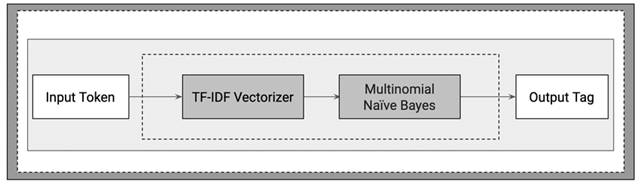
##### Fig 8: Architecture of a Logistic Regression Model

(Source : <https://www.researchgate.net/figure/Architecture-of-a-Logistic-Regression-Model-56_fig7_334575492>)

### **Multinomial Naive Bayes**

For this project, Multinomial Naive Bayes was selected because it excels at text classification jobs when the features are word counts or phrase frequencies. It is a good option for spam identification in reviews because of its potency with high-dimensional data and its robustness with categorical data (Cheng et al., 2023). This approach simplifies computation and frequently performs well in practice for text classification since it makes the assumption that characteristics are conditionally independent given the class.

Default hyperparameters were used to initialize the Multinomial Naive Bayes model. The model was trained by including it into the training data, from which it acquired knowledge of the feature probability distribution for every class. Subsequently, the model was employed on the test set to categorize reviews as spam or not. Metrics like accuracy, precision, recall, and F1-score were evaluated as part of the performance evaluation process to find out how successfully the model distinguished between reviews that were spam and those that weren't.



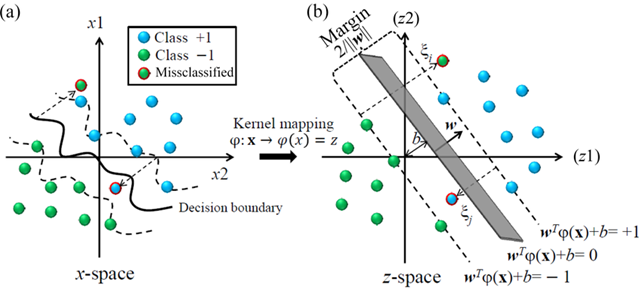
##### Fig 9: Naïve Bayes Model Architecture

(Source : <https://www.researchgate.net/figure/Naive-Bayes-Model-Architecture_fig3_356421477>)

### **Linear Support Vector Classification**

LinearSVC was selected due to its efficiency in handling large-scale text classification tasks and its ability to create a hyperplane that maximally separates classes. LinearSVC is known for its strong performance in binary classification problems and is particularly effective in high-dimensional spaces, such as those encountered in text data. Its capability to handle large datasets with many features and its robustness in distinguishing between classes made it a suitable choice for spam detection (Drucker et al., 1999).

Initializing the model with default settings allowed for the implementation of the Linear Support Vector Classification model. The model was fitted to the training set in order to learn how to identify the best hyperplane for dividing reviews into spam and non-spam. The model was used to predict the test set's class labels once it had been trained. Metrics including recall, accuracy and precision, and F1-score were used to evaluate the model's performance in accurately classifying spam reviews.



##### Fig 10 : Graphical presentation of the support vector classifier

(Source : <https://www.researchgate.net/figure/Graphical-presentation-of-the-support-vector-machine-classifier-with-a-non-linear-kernel_fig1_299529384>)

### **Perceptron**

The Perceptron model was selected due to its ease of use and efficiency in handling binary classification problems. As a foundational algorithm in machine learning, the Perceptron is well-suited for scenarios where the decision boundary is linear. Its ability to adaptively adjust weights during training and its straightforward implementation make it a good fit for spam detection tasks, where the goal is to differentiate between spam and non-spam reviews based on their features (Shahariar et al., 2019).

The Perceptron model was implemented by initializing it with standard parameters. Training involved feeding the model with labeled review data, allowing it to iteratively update its weights based on classification errors. The model was used to categorize reviews in the test dataset once it had been trained. The model's performance was evaluated using measures including accuracy, precision, recall, and F1-score to make sure effectively distinguished between spam and non-spam reviews.



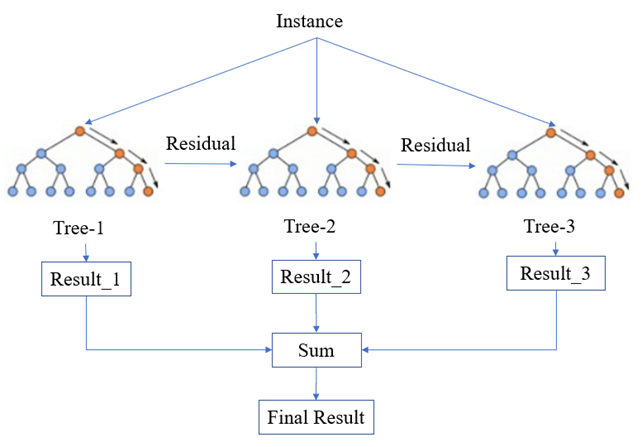
##### Fig 11 : Perceptron Model Architecture

(Source : <https://www.geeksforgeeks.org/sklearn-classification-using-perceptron/>)

### **XGBoost Classifier**

The XGBoost Classifier was selected for its robustness and superior performance in classification tasks, especially those involving complex patterns in the data. Known for its gradient boosting framework, XGBoost is very good at managing big datasets and identifying complex feature correlations. (Hameed et al., 2023). Its ability to manage overfitting through regularization and its efficiency in model training make it highly suitable for spam detection, where distinguishing between spam and non-spam reviews can be challenging due to the diversity and subtlety of the text data.

The XGBoost Classifier was implemented by configuring its hyperparameters to optimize performance. The model was trained on the preprocessed review data, leveraging its gradient boosting capabilities to iteratively improve its predictions. Feature importance scores were evaluated to understand which features contributed most to the classification. After training, the model was tested using a different dataset to evaluate its efficacy and accuracy. Its performance was assessed using evaluation measures, including accuracy, precision, recall, and F1-score, to make sure it achieved the intended categorization goals.



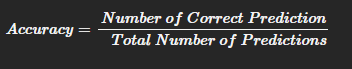
##### Fig 12 : Simplified structure of XGBoost

(Source : <https://www.researchgate.net/figure/Simplified-structure-of-XGBoost_fig2_348025909>)

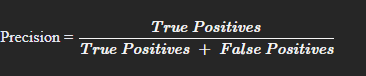
## **Model Evaluation Techniques**

Various evaluation metrics were utilized to assess the spam detection models' performance. These metrics include precision, recall, F1-score, and accuracy, each providing valuable insights into the model's effectiveness (Vujovic et al., 2021). The models were evaluated on multiple datasets: the original data, data with lemmatization, cleaned data, and cleaned and lemmatized data. This comprehensive evaluation approach ensures a thorough understanding of how each model performs under different preprocessing conditions.

**Accuracy:** The percentage of correctly identified cases is measured by accuracy. (spam and non-spam) out of the total cases. It provides an overall indication of the model's performance.

**Formula:** 

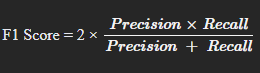
**Precision:** The precision measures the percentage of correct optimistic forecasts among all the model's optimistic predictions. It shows the proportion of spam reviews that were found to be spam.

**Formula:** 

**Recall:** Recall quantifies the proportion of accurate positive forecasts among all actual positive occurrences. It displays the model's capacity to recognize every pertinent spam review.

**Formula:** 

**F1-Score:** The F1-score, which is the harmonic mean of recall and accuracy, is a statistic that offers a balance between the two. It is especially helpful in cases when the distribution of classes is not uniform.

**Formula:** 

By evaluating the models using these metrics on different versions of the dataset, the effectiveness of the preprocessing techniques and the overall model performance can be thoroughly assessed. This approach helps in identifying the best-performing model and understanding how preprocessing affects the detection of spam reviews.

# WEEK 13 Predictive/descriptive Analytics System Group-Based Assignment

**Purpose:**

To complete all the modules of the Predictive/Descriptive Analytics System

**Tasks:**

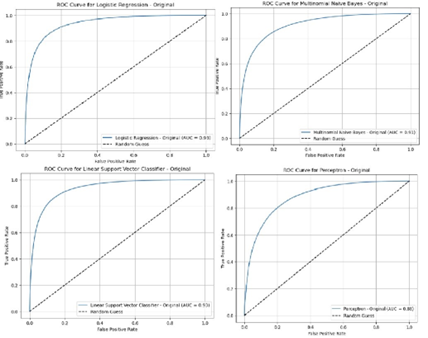
1. Submit the final project report of your team’s Predictive /Descriptive Analytics system by the end of week 13. In this week the team is supposed to complete the visualization of the Analytics system.
2. Project report feedback and more instructions about the final submission of the project report, datasets, scripts etc. will be provided by the instructor over the e-mail.

**Results & Discussion:**

The Results chapter will focus on presenting and analysing the effectiveness of the classification methods applied to the identification of spam reviews. Important performance indicators including accuracy, F1-score, recall, and precision, will be summarized and compared across different datasets, including the original, lemmatized, cleaned, and cleaned with lemmatization data. A comparative analysis will highlight the best-performing models and how different preprocessing methods affect how well they perform. Confusion matrices will be used to illustrate these findings and provide insights into the effectiveness of each approach.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 90.99% | 90.51% | 90.99% | 90.54% |
| **Multinomial Naive Bayes** | 86.69% | 87.21% | 86.69% | 83.18% |
| **XGBoost** | 90.49% | 89.93% | 90.49% | 89.91% |
| **Perceptron** | 87.63% | 87.12% | 87.63% | 87.33% |
| **Liner SVC** | 90.29% | 89.74% | 90.29% | 89.48% |

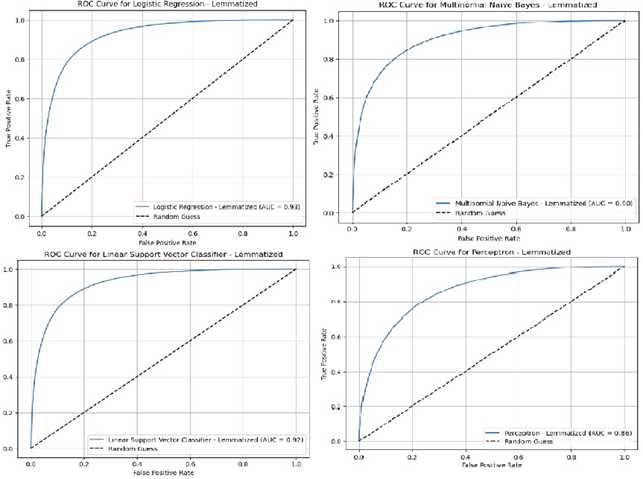
###### Table 1: Model Performance on Original Data (These results are average of five iterations)



The results in Table 1 illustrate the performance of various models on the original dataset. **Logistic Regression** achieved the highest accuracy at **90.99%**, closely followed by **XGBoost** with an accuracy of **90.49%**, demonstrating their strong capability to handle unprocessed data. **Linear SVC** also performed well, with an accuracy of **90.29%**, making it a reliable choice. In comparison, **Multinomial Naive Bayes** had the lowest accuracy at **86.69%**, which may be attributed to its simplistic assumptions about the data. The **Perceptron model**, with an accuracy of **87.63%**, showed moderate performance but delivered a competitive F1-Score. Overall, Logistic Regression and XGBoost stand out as the most effective models for the original dataset, consistently achieving high metrics across all evaluation criteria.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 90.49% | 89.93% | 90.49% | 89.90% |
| **Multinomial Naive Bayes** | 86.91% | 87.29% | 86.91% | 83.63% |
| **Linear SVC** | 90.45% | 89.90% | 90.45% | 89.77% |
| **Perceptron** | 87.16% | 86.17% | 87.16% | 86.46% |
| **XGBoost** | 89.47% | 88.92% | 89.47% | 88.25% |

###### Table 2: Model Performance on Lemmatized Data



The results in Table 2 highlight the performance of various models on the lemmatized dataset. **Logistic Regression** and **Linear SVC** achieved the highest accuracy, with scores of **90.49%** and **90.45%**, respectively, indicating their robustness in handling the processed data. **XGBoost**, while slightly trailing with an accuracy of **89.47%**, still demonstrated strong performance across all metrics, making it a competitive alternative. **Multinomial Naive Bayes** had the lowest accuracy at **86.91%**, reflecting its limitations when applied to this dataset. The **Perceptron model** performed moderately well, with an accuracy of **87.16%**. Overall, the results suggest that Logistic Regression and Linear SVC are the most reliable models for the lemmatized dataset, leveraging the advantages of the simplified and standardized text features.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 90.36% | 89.78% | 90.36% | 89.77% |
| **Multinomial Naive Bayes** | 86.83% | 87.16% | 86.83% | 83.50% |
| **Linear SVC** | 90.28% | 89.70% | 90.28% | 89.58% |
| **Perceptron** | 84.46% | 85.43% | 84.46% | 84.88% |
| **XGBoost** | 89.41% | 88.85% | 89.41% | 88.17% |

###### Table 3: Model Performance on Cleaned Data

### 

The performance evaluation in Table 3 demonstrates the effectiveness of various models on the cleaned dataset. **Logistic Regression** and **Linear SVC** again emerged as the top performers, with accuracies of **90.36%** and **90.28%**, respectively, highlighting their ability to handle the preprocessed data effectively. **XGBoost** followed closely with an accuracy of **89.41%**, showcasing its reliability across different datasets. **Multinomial Naive Bayes** exhibited lower performance, achieving an accuracy of **86.83%**, which may stem from its simplifying assumptions about word independence. The **Perceptron model**, while exhibiting relatively lower accuracy at **84.46%**, demonstrated competitive precision and F1-Score, suggesting it can still be a viable option. Overall, the results indicate that Logistic Regression and Linear SVC consistently perform best on cleaned data, maintaining a strong balance across all metrics.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| **Logistic Regression** | 90.49% | 89.93% | 90.49% | 89.91% |
| **Multinomial Naive Bayes** | 86.89% | 87.25% | 86.89% | 83.60% |
| **Linear SVC** | 90.46% | 89.91% | 90.46% | 89.77% |
| **Perceptron** | 84.76% | 85.46% | 84.76% | 85.07% |
| **XGBoost** | **91.02%** | **90.53%** | **91.02%** | **90.50%** |

###### Table 4: Model Performance on Cleaned and Lemmatized Data

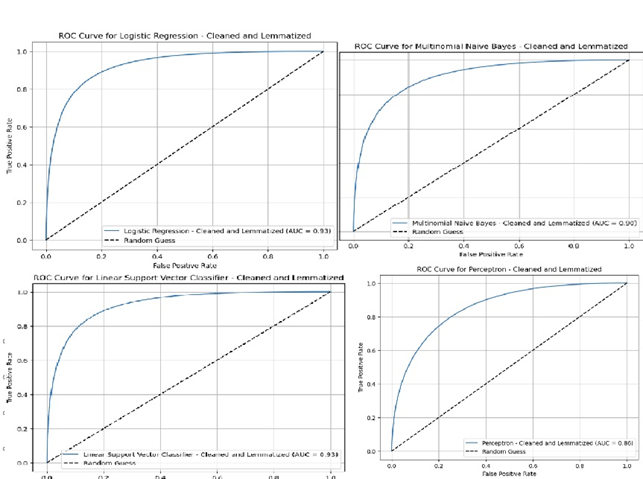


Table 4 highlights the performance of various models on the cleaned and lemmatized dataset, demonstrating the impact of preprocessing on model effectiveness. Among the models, **XGBoost** stands out as the best performer, achieving the highest accuracy (**91.02%**) and recall (**91.02%**) while maintaining strong precision (**90.53%**) and F1-Score (**90.50%**). These results indicate that XGBoost excels at accurately identifying both spam and non-spam reviews, making it the most reliable model for spam detection. **Logistic Regression** and **Linear SVC** also performed well, with accuracies of **90.49%** and **90.46%**, respectively, but fell slightly short in recall and F1-Score compared to XGBoost. **Multinomial Naive Bayes** and **Perceptron** showed lower overall performance, with accuracies of **86.89%** and **84.76%**, respectively, reflecting their limitations in leveraging the cleaned and lemmatized features effectively. Overall, XGBoost's superior recall and balanced metrics make it the most suitable choice for spam review detection on the preprocessed dataset.

### **Why XGBoost with Cleaned and Lemmatized Data is the Best Model for Spam Review Detection**

The performance of **XGBoost** on the cleaned and lemmatized dataset is outstanding, as evidenced by its metrics: **91.02% Accuracy**, **90.53% Precision**, **91.02% Recall**, and **90.50% F1-Score**. These results demonstrate that XGBoost not only performs well overall but is particularly well-suited for the specific application of **spam review detection**. Its combination of high recall, precision, and F1-Score ensures robust classification of spam reviews with minimal errors.

**High recall (91.02%)** ensures that XGBoost is particularly adept at correctly identifying spam reviews (true positives). For spam review detection, prioritizing high recall is critical because failing to detect spam (false negatives) is more detrimental than incorrectly flagging legitimate reviews (false positives). Among all the models tested, XGBoost achieved the highest recall, making it the most reliable choice for minimizing undetected spam reviews.

At the same time, XGBoost achieves a balanced **precision (90.53%)**, reducing the likelihood of false positives. Precision measures the proportion of correctly predicted spam reviews out of all predicted spam reviews. This balance ensures that while XGBoost successfully identifies most spam, it does not overwhelm users or systems by incorrectly classifying too many legitimate reviews as spam.

The **high F1-Score (90.50%)** achieved by XGBoost further demonstrates its robustness. The F1-Score balances precision and recall, providing a comprehensive measure of the model's effectiveness. XGBoost's ability to excel in both precision and recall while minimizing trade-offs ensures its suitability for spam review detection.

The use of **cleaned and lemmatized data** enhances XGBoost’s performance by improving the quality of the input data. Cleaning removes noise, such as irrelevant characters and stopwords, while lemmatization standardizes words to their base forms, reducing redundancy. These preprocessing steps help XGBoost, a gradient-boosted decision tree model, efficiently leverage the weighted importance of meaningful words, ultimately leading to superior spam detection.

In the context of spam review detection, prioritizing true positives while maintaining balanced precision and recall is essential. XGBoost achieves this balance effectively, outperforming models like Logistic Regression and Linear SVC. Its superior performance across all metrics highlights its robustness and reliability, making it the best model for this application.

**Applying under sampling and training Multi Linear Regression and XGBoost models on cleaned and lemmatized data**

To address the issue of class imbalance in our dataset, we employed the undersampling technique. Class imbalance occurs when one class significantly outnumbers the other, which can lead to biased machine learning models that predominantly favor the majority class. By balancing the dataset, we aim to ensure fair and effective model training.

We began by analyzing the initial distribution of the target variable (class) to identify the minority and majority classes. This step was crucial to understanding the extent of imbalance in the data and provided a baseline for comparison after the balancing process. The minority class was separated as the smaller subset of data, while the majority class contained significantly more instances.

To balance the dataset, we randomly undersampled the majority class, reducing its size to match that of the minority class. By using random sampling with a fixed seed for reproducibility, we ensured that the subset of majority class instances was representative of the overall class. The minority class was then combined with the undersampled majority class, creating a balanced dataset where both classes had an equal number of instances.

Once the classes were combined, we shuffled the resulting dataset to mix instances from both classes. Shuffling prevents any ordering bias in the data and ensures that machine learning models trained on the dataset receive randomized input. Finally, we verified the success of the balancing process by analyzing the new class distribution, confirming that both classes were equally represented.

The process of undersampling was essential for mitigating the bias introduced by the original imbalance. By balancing the dataset, we improved the fairness and effectiveness of machine learning models, allowing them to learn equally from both classes. This ensures that the models are better equipped to correctly classify instances of both the majority and minority classes, enhancing their overall performance and reliability.



**Results of Multi Linear Regression and XGBoost after class balancing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** |
| XGBoost | 0.85646718 | 0.85647599 | 0.85646718 | 0.8564663 |
| Multi Linear Regression | 0.8330307878642882 | 0.8330629819206286 | 0.8330307878642882 | 0.8330267283639575 |

The performance evaluation of two models, **XGBoost** and **Multi Linear Regression**, reveals that XGBoost outperforms its counterpart across all key metrics: **Accuracy**, **Precision**, **Recall**, and **F1-Score**. XGBoost achieved an accuracy of 85.65%, precision of 85.65%, recall of 85.65%, and an F1-Score of 85.65%, demonstrating its consistent and reliable performance. These metrics highlight XGBoost’s superior ability to classify both spam and non-spam reviews accurately, minimize false positives, and correctly identify most spam reviews, reducing false negatives.

In comparison, Multi Linear Regression achieved an accuracy of 83.30%, precision of 83.31%, recall of 83.30%, and an F1-Score of 83.30%. While these metrics indicate reasonably good performance, they are lower than those of XGBoost, reflecting its slightly reduced ability to balance precision and recall effectively. This makes Multi Linear Regression less reliable for tasks where minimizing false negatives is critical.

XGBoost’s higher recall is particularly important in the context of spam review detection, where missing spam reviews (false negatives) can have greater consequences than false positives. Additionally, its superior F1-Score emphasizes its balanced performance, making it better suited for real-world applications. Overall, XGBoost’s strong metrics across the board make it the preferred model for accurate and efficient spam review detection.

**Deriving the XGBoost Equation and Identifying Key Factors in Spam Reviews**

To analyze the importance of various features in the XGBoost model, the feature importance values were extracted using the feature\_importances\_ attribute of the trained model. These importance scores represent how significantly each feature (word) contributes to the model's decision-making process. Simultaneously, the feature names corresponding to these importance values were retrieved from the vectorizer using the get\_feature\_names\_out() method.

The extracted feature importance scores and their corresponding feature names were then organized into a structured format using a DataFrame. This DataFrame includes two columns: one for the feature (word) and another for its respective importance score. The features were subsequently sorted in descending order of importance, allowing for easy identification of the most influential words in the context of spam review detection. This process highlights the key factors that the model relies on to differentiate between spam and non-spam reviews.

**Results**

The analysis of feature importance reveals key insights into the factors influencing the classification of spam reviews. Positive coefficients, such as those for words like **"love" (9.77)**, **"great" (8.99)**, **"perfect" (8.15)**, and **"highly" (7.78)**, indicate that these terms are strongly associated with legitimate, non-spam reviews. These words likely convey positive sentiments that genuine reviews typically exhibit. Conversely, negative coefficients, such as **"disappointing" (-10.30)**, **"disappointed" (-10.22)**, **"waste" (-9.60)**, and **"poorly" (-8.75)**, are indicative of spam reviews. These terms often express dissatisfaction or exaggeration, which are characteristic of spam content designed to mislead or manipulate.

The clear distinction in the polarity of these coefficients demonstrates the model's ability to differentiate between genuine and spam reviews based on textual patterns. The words with the highest positive and negative importance scores serve as critical indicators for the model, highlighting its reliance on sentiment cues and keyword frequencies to classify reviews effectively. This understanding not only validates the model's interpretability but also provides actionable insights into the linguistic markers of spam reviews.

# WEEK 14 Predictive / Descriptive Analytics System Group-Based Assignment

**Purpose:**

To provide a demonstration of your team’s Predictive /Descriptive Analytics System

**Tasks:**

1. The team should prepare to demonstrate the designed Predictive/Descriptive Analytics System in this capstone course.
2. The demonstration should be presented in class. The power point presentation should be uploaded through the box/Canvas.
3. Each team should submit the data sets and the scripts into their respective box folder.
4. Every team will have an opportunity to go through the demonstration of the Predictive/Descriptive Analytics system designed by other teams.
5. More instructions will be provided by the instructor over e-mail.

**Appendix**

#preprocessing import pandas as pd import numpy as np import matplotlib.pyplot as plt from wordcloud import WordCloud from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.model\_selection import train\_test\_split import re import string from nltk.corpus import stopwords import nltk from nltk.stem import WordNetLemmatizer nltk.download('stopwords') stop\_words = set(stopwords.words('english')) from scipy.sparse import issparse

#models from sklearn.naive\_bayes import MultinomialNB from sklearn.linear\_model import LogisticRegression from sklearn.multiclass import OneVsRestClassifier from sklearn.svm import LinearSVC from sklearn.linear\_model import Perceptron from sklearn.ensemble import RandomForestClassifier from xgboost import XGBClassifier from sklearn.model\_selection import StratifiedKFold from sklearn.metrics import roc\_curve, auc, precision\_score, recall\_score, f1\_score, accuracy\_score, confusion\_matrix import time #evaluation from sklearn.metrics import ConfusionMatrixDisplay,accuracy\_score,confusion\_matrix,precision\_score, recall\_score, f1\_score

from nltk.tokenize import word\_tokenize from nltk.corpus import stopwords from nltk.stem import WordNetLemmatizer import nltk

# **Download resources if not already downloaded**

nltk.download('punkt') nltk.download('stopwords') nltk.download('wordnet')

df = pd.read\_json('data.json', lines=True)

df.columns

df.shape

df.info()

# **Count the number of spam and not spam reviews**

spam\_counts = df['class'].value\_counts()

# **Plotting**

plt.figure(figsize=(8, 6)) spam\_counts.plot(kind='bar', color=['blue', 'red']) plt.title('Distribution of Spam vs. Not Spam Reviews') plt.xlabel('Class') plt.ylabel('Count') plt.xticks([0, 1], ['Not Spam', 'Spam'], rotation=0) plt.show()

# **Combine all review text into one string**

review\_text = ' '.join(df['reviewText'].astype(str))

# **Generate word cloud**

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(review\_text)

# **Plotting**

plt.figure(figsize=(10, 8)) plt.imshow(wordcloud, interpolation='bilinear') plt.title('Word Cloud for Review Text') plt.axis('off') plt.show()

# **Separate positive and negative reviews based on overall rating**

positive\_reviews = ' '.join(df[df['overall'] >= 4]['reviewText'].astype(str)) negative\_reviews = ' '.join(df[df['overall'] <= 2]['reviewText'].astype(str))

# **Generate word clouds for positive and negative reviews**

positive\_wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(positive\_reviews) negative\_wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(negative\_reviews)

# **Plotting**

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1) plt.imshow(positive\_wordcloud, interpolation='bilinear') plt.title('Positive Words Word Cloud') plt.axis('off')

plt.subplot(1, 2, 2) plt.imshow(negative\_wordcloud, interpolation='bilinear') plt.title('Negative Words Word Cloud') plt.axis('off')

plt.show()

# **Convert reviewTime to datetime**

df['reviewTime'] = pd.to\_datetime(df['reviewTime'])

# **Extract year and month from reviewTime**

df['Year'] = df['reviewTime'].dt.year df['Month'] = df['reviewTime'].dt.month

# **Group by year and month, count the number of reviews**

review\_counts = df.groupby(['Year', 'Month']).size().reset\_index(name='Number of Reviews')

# **Plotting**

plt.figure(figsize=(12, 6)) plt.plot(review\_counts['Year'], review\_counts['Number of Reviews'], marker='o', linestyle='-') plt.title('Number of Reviews Over Time') plt.xlabel('Year') plt.ylabel('Number of Reviews') plt.grid(True) plt.xticks(review\_counts['Year'].unique()) plt.show()

# **Count the number of reviews for each overall rating**

overall\_counts = df['overall'].value\_counts().sort\_index()

# **Plotting**

plt.figure(figsize=(8, 6)) overall\_counts.plot(kind='bar', color='skyblue') plt.title('Overall Ratings Distribution') plt.xlabel('Overall Rating') plt.ylabel('Number of Reviews') plt.xticks(rotation=0) plt.show()

# **Group by Year and count the number of purchases**

purchase\_yearly = df.groupby(df['reviewTime'].dt.year)['asin'].count()

# **Group by Month and count the number of purchases**

purchase\_monthly = df.groupby(df['reviewTime'].dt.month)['asin'].count()

# **Plotting**

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1) purchase\_yearly.plot(kind='bar', color='lightgreen') plt.title('Purchase Analysis - Year wise') plt.xlabel('Year') plt.ylabel('Number of Purchases') plt.xticks(rotation=45) plt.grid(True)

plt.subplot(1, 2, 2) purchase\_monthly.plot(kind='bar', color='lightcoral') plt.title('Purchase Analysis - Month wise') plt.xlabel('Month') plt.ylabel('Number of Purchases') plt.xticks(rotation=0) plt.grid(True)

plt.tight\_layout() plt.show()

stop\_words = set(stopwords.words('english')) lemmatizer = WordNetLemmatizer()

def clean\_text(text): text = text.translate(str.maketrans('', '', string.punctuation)) text = text.lower() text = re.sub(r'\W', ' ', text) text = re.sub(r'\d+', '', text) text = ' '.join(word for word in text.split() if word not in stop\_words) return text

#Lemmatize words def Lemm(text): words = word\_tokenize(text) # Tokenize the text words = [word for word in words if word not in stop\_words] # Remove stop words words = [lemmatizer.lemmatize(word) for word in words] # Lemmatize the words return ' '.join(words) # Join words back into a string

df['cleaned\_text'] = df['reviewText'].apply(clean\_text) df['text\_lemm'] = df['reviewText'].apply(Lemm) df['cleaned\_text\_lemm'] = df['text\_lemm'].apply(clean\_text)

vectorization = TfidfVectorizer(analyzer = 'word', min\_df=0.0, max\_features=2000) reviews = df['reviewText'].to\_list() reviews\_lemm = df['cleaned\_text\_lemm'].to\_list() reviews\_cleaned\_text = df['cleaned\_text'].to\_list() reviews\_cleand\_text\_lem = df['text\_lemm'].to\_list()

X1 = vectorization.fit\_transform(reviews) X2 = vectorization.fit\_transform(reviews\_lemm) X3 = vectorization.fit\_transform(reviews\_cleaned\_text) X4 = vectorization.fit\_transform(reviews\_cleand\_text\_lem)

def split\_80\_10\_10\_multiple(X, y, iterations=5): splits = [] for split\_num in range(1, iterations+1): X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.2, stratify=y) X\_test, X\_eval, y\_test, y\_eval = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, stratify=y\_temp) splits.append((X\_train, X\_test, X\_eval, y\_train, y\_test, y\_eval))

return(splits)

splits1 = split\_80\_10\_10\_multiple(X1, df['class']) # Original dataset splits2 = split\_80\_10\_10\_multiple(X2, df['class']) # Lemmatized dataset splits3 = split\_80\_10\_10\_multiple(X3, df['class']) # Cleaned dataset splits4 = split\_80\_10\_10\_multiple(X4, df['class']) # Cleaned & Lemmatized dataset

for i in range(5): X\_train, X\_test, X\_eval, y\_train, y\_test, y\_eval = splits1[i] globals()[f'X\_train1\_split{i+1}'] = X\_train globals()[f'X\_test1\_split{i+1}'] = X\_test globals()[f'X\_eval1\_split{i+1}'] = X\_eval globals()[f'y\_train1\_split{i+1}'] = y\_train globals()[f'y\_test1\_split{i+1}'] = y\_test globals()[f'y\_eval1\_split{i+1}'] = y\_eval

for i in range(5): X\_train, X\_test, X\_eval, y\_train, y\_test, y\_eval = splits2[i] globals()[f'X\_train2\_split{i+1}'] = X\_train globals()[f'X\_test2\_split{i+1}'] = X\_test globals()[f'X\_eval2\_split{i+1}'] = X\_eval globals()[f'y\_train2\_split{i+1}'] = y\_train globals()[f'y\_test2\_split{i+1}'] = y\_test globals()[f'y\_eval2\_split{i+1}'] = y\_eval

for i in range(5): X\_train, X\_test, X\_eval, y\_train, y\_test, y\_eval = splits3[i] globals()[f'X\_train3\_split{i+1}'] = X\_train globals()[f'X\_test3\_split{i+1}'] = X\_test globals()[f'X\_eval3\_split{i+1}'] = X\_eval globals()[f'y\_train3\_split{i+1}'] = y\_train globals()[f'y\_test3\_split{i+1}'] = y\_test globals()[f'y\_eval3\_split{i+1}'] = y\_eval

for i in range(5): X\_train, X\_test, X\_eval, y\_train, y\_test, y\_eval = splits4[i] globals()[f'X\_train4\_split{i+1}'] = X\_train globals()[f'X\_test4\_split{i+1}'] = X\_test globals()[f'X\_eval4\_split{i+1}'] = X\_eval globals()[f'y\_train4\_split{i+1}'] = y\_train globals()[f'y\_test4\_split{i+1}'] = y\_test globals()[f'y\_eval4\_split{i+1}'] = y\_eval

def plot\_roc\_curve(model, X\_test, y\_test, model\_name): """ Plot ROC curve for the given model and test data. """ if hasattr(model, "predict\_proba"): y\_probs = model.predict\_proba(X\_test)[:, 1] else: y\_probs = model.decision\_function(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test, y\_probs)  
roc\_auc = auc(fpr, tpr)  
  
plt.figure(figsize=(8, 6))  
plt.plot(fpr, tpr, label=f"{model\_name} (AUC = {roc\_auc:.2f})")  
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")  
plt.title(f"ROC Curve for {model\_name}")  
plt.xlabel("False Positive Rate")  
plt.ylabel("True Positive Rate")  
plt.legend()  
plt.grid()  
plt.show()

def adaptive\_k\_fold\_evaluation(model, X, y, k\_min=5, k\_max=10, max\_duration=60): """ Perform k-fold evaluation, adapting k (number of folds) based on model duration. """ for k in range(k\_min, k\_max + 1): skf = StratifiedKFold(n\_splits=k, shuffle=True, random\_state=42) precision\_scores = [] recall\_scores = [] f1\_scores = [] fold\_durations = []

print(f"Running {k}-fold evaluation...")  
 for train\_index, test\_index in skf.split(X, y):  
 X\_train, X\_test = X[train\_index], X[test\_index]  
 y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]  
  
 start\_time = time.time()  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 duration = time.time() - start\_time  
  
 fold\_durations.append(duration)  
 precision\_scores.append(precision\_score(y\_test, y\_pred, average='weighted'))  
 recall\_scores.append(recall\_score(y\_test, y\_pred, average='weighted'))  
 f1\_scores.append(f1\_score(y\_test, y\_pred, average='weighted'))  
  
 avg\_duration = np.mean(fold\_durations)  
 if avg\_duration \* k > max\_duration:  
 print(f"Stopping at {k - 1} folds due to time constraints (Avg. Duration: {avg\_duration:.2f}s).")  
 break  
  
return {  
 "Folds": k - 1,  
 "Precision": np.mean(precision\_scores),  
 "Recall": np.mean(recall\_scores),  
 "F1-Score": np.mean(f1\_scores),  
 "Avg Duration per Fold": avg\_duration  
}

def train\_evaluate\_visualize(model, X\_train, y\_train, X\_test, y\_test, k\_min=5, k\_max=10, max\_duration=60, model\_name="Model"): """ Train, evaluate, perform adaptive k-fold evaluation, and visualize ROC curve for the given model and data. """ # Convert target labels to dense if needed if hasattr(y\_train, "toarray"): y\_train = y\_train.toarray().ravel() # Ensure 1D if hasattr(y\_test, "toarray"): y\_test = y\_test.toarray().ravel() # Ensure 1D

# Train and evaluate  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
  
# Metrics  
accuracy = accuracy\_score(y\_test, y\_pred)  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
cm = confusion\_matrix(y\_test, y\_pred)  
  
# Print evaluation metrics  
print(f"Results for {model\_name}:")  
print(f"Accuracy: {accuracy:.4f}")  
print(f"Precision: {precision:.4f}")  
print(f"Recall: {recall:.4f}")  
print(f"F1-score: {f1:.4f}")  
print("Confusion Matrix:")  
print(cm)  
  
# Adaptive K-Fold Evaluation  
k\_fold\_results = adaptive\_k\_fold\_evaluation(model, X\_train, y\_train, k\_min, k\_max, max\_duration)  
print(f"K-Fold Results ({k\_fold\_results['Folds']} folds):")  
print(f"Precision: {k\_fold\_results['Precision']:.4f}, Recall: {k\_fold\_results['Recall']:.4f}, F1-Score: {k\_fold\_results['F1-Score']:.4f}")  
print(f"Avg Duration per Fold: {k\_fold\_results['Avg Duration per Fold']:.2f}s")  
  
# ROC Curve  
plot\_roc\_curve(model, X\_test, y\_test, model\_name)  
  
return accuracy, precision, recall, f1, cm

models = [ ("Logistic Regression", LogisticRegression(max\_iter=400)), ("Multinomial Naive Bayes", MultinomialNB()), ("Linear Support Vector Classifier", LinearSVC()), ("Perceptron", Perceptron()), ("XGBoost", XGBClassifier()) ]

datasets = []

# **For the "Original" dataset**

for i in range(5): datasets.append(( f"Original - Split {i+1}", globals()[f'X\_train1\_split{i+1}'], globals()[f'y\_train1\_split{i+1}'], globals()[f'X\_test1\_split{i+1}'], globals()[f'y\_test1\_split{i+1}'], globals()[f'X\_eval1\_split{i+1}'], globals()[f'y\_eval1\_split{i+1}'] ))

# **For the "Lemmatized" dataset**

for i in range(5): datasets.append(( f"Lemmatized - Split {i+1}", globals()[f'X\_train2\_split{i+1}'], globals()[f'y\_train2\_split{i+1}'], globals()[f'X\_test2\_split{i+1}'], globals()[f'y\_test2\_split{i+1}'], globals()[f'X\_eval2\_split{i+1}'], globals()[f'y\_eval2\_split{i+1}'] ))

# **For the "Cleaned" dataset**

for i in range(5): datasets.append(( f"Cleaned - Split {i+1}", globals()[f'X\_train3\_split{i+1}'], globals()[f'y\_train3\_split{i+1}'], globals()[f'X\_test3\_split{i+1}'], globals()[f'y\_test3\_split{i+1}'], globals()[f'X\_eval3\_split{i+1}'], globals()[f'y\_eval3\_split{i+1}'] ))

# **For the "Cleaned and Lemmatized" dataset**

for i in range(5): datasets.append(( f"Cleaned and Lemmatized - Split {i+1}", globals()[f'X\_train4\_split{i+1}'], globals()[f'y\_train4\_split{i+1}'], globals()[f'X\_test4\_split{i+1}'], globals()[f'y\_test4\_split{i+1}'], globals()[f'X\_eval4\_split{i+1}'], globals()[f'y\_eval4\_split{i+1}'] ))

for model\_name, model in models: # For each dataset with 5 splits for dataset\_name, X\_train, y\_train, X\_test, y\_test, X\_eval, y\_eval in datasets: print(f"\nModel: {model\_name}, Dataset: {dataset\_name}")

# Train, evaluate, and visualize the results for each model and dataset split  
 train\_evaluate\_visualize(model, X\_train, y\_train, X\_test, y\_test, k\_min=5, k\_max=10, max\_duration=60, model\_name=f"{model\_name} - {dataset\_name}")

for model\_name, model in models: # For each dataset with 5 splits for dataset\_name, X\_train, y\_train, X\_test, y\_test, X\_eval, y\_eval in datasets: print(f"\nModel: {model\_name}, Dataset: {dataset\_name}")

# Train, evaluate, and visualize the results for each model and dataset split  
 train\_evaluate\_visualize(model, X\_train, y\_train, X\_test, y\_test, k\_min=5, k\_max=10, max\_duration=60, model\_name=f"{model\_name} - {dataset\_name}")

df.head()

def adaptive\_k\_fold\_evaluation(model, X, y, k\_min=5, k\_max=10, max\_duration=60): """ Perform adaptive K-Fold evaluation based on max\_duration. """ # Convert y to a NumPy array to avoid indexing issues y = np.array(y)

metrics = []  
total\_time = 0  
  
for k in range(k\_min, k\_max + 1):  
 skf = StratifiedKFold(n\_splits=k, shuffle=True, random\_state=42)  
 fold\_metrics = []  
 fold\_durations = []  
  
 print(f"\nRunning {k}-fold evaluation...")  
 start\_time = time.time()  
  
 for train\_idx, test\_idx in skf.split(X, y):  
 # Use indices to split X and y  
 X\_train, X\_test = X[train\_idx], X[test\_idx]  
 y\_train, y\_test = y[train\_idx], y[test\_idx]  
  
 # Train the model  
 model.fit(X\_train, y\_train)  
  
 # Predict  
 y\_pred = model.predict(X\_test)  
  
 # Metrics  
 accuracy = accuracy\_score(y\_test, y\_pred)  
 precision = precision\_score(y\_test, y\_pred, average="weighted")  
 recall = recall\_score(y\_test, y\_pred, average="weighted")  
 f1 = f1\_score(y\_test, y\_pred, average="weighted")  
  
 fold\_metrics.append((accuracy, precision, recall, f1))  
  
 # Time tracking  
 duration = time.time() - start\_time  
 fold\_durations.append(duration)  
  
 total\_time += np.mean(fold\_durations)  
  
 if total\_time > max\_duration:  
 print(f"Stopping at {k - 1} folds due to time constraints.")  
 break  
  
 metrics.append(np.mean(fold\_metrics, axis=0)) # Average across folds  
  
return {  
 "Folds": k - 1,  
 "Metrics": metrics,  
 "Average Metrics": np.mean(metrics, axis=0),  
 "Total Duration": total\_time,  
}

from sklearn.metrics import roc\_curve, auc from sklearn.model\_selection import StratifiedKFold import time import numpy as np import pandas as pd import matplotlib.pyplot as plt

def extract\_equation\_and\_interpret(model, vectorizer, model\_name): """ Extract model coefficients or feature importances and map them to original words. """ if model\_name == "Logistic Regression": # Coefficients for logistic regression coefficients = model.coef\_[0] feature\_names = vectorizer.get\_feature\_names\_out() feature\_importance = pd.DataFrame({ "Word": feature\_names, "Coefficient": coefficients }).sort\_values(by="Coefficient", ascending=False) print(f"\nTop 10 Words for {model\_name}:\n", feature\_importance.head(10)) print(f"\nBottom 10 Words for {model\_name}:\n", feature\_importance.tail(10)) return feature\_importance

elif model\_name == "XGBoost":  
 # Feature importances for XGBoost  
 feature\_importance = model.feature\_importances\_  
 feature\_names = vectorizer.get\_feature\_names\_out()  
 feature\_importance\_df = pd.DataFrame({  
 "Word": feature\_names,  
 "Importance": feature\_importance  
 }).sort\_values(by="Importance", ascending=False)  
 print(f"\nTop 10 Words for {model\_name}:\n", feature\_importance\_df.head(10))  
 return feature\_importance\_df  
  
else:  
 print("Model does not support feature importance extraction.")  
 return None

def train\_evaluate\_with\_equations(models, X, y, vectorizer, iterations=5, k\_min=5, k\_max=10, max\_duration=60): """ Train and evaluate models with adaptive K-Fold evaluation, including model equations and word mapping. """ final\_results = [] for model\_name, model in models: print(f"\nTraining and Evaluating: {model\_name}") iteration\_metrics = []

for i in range(iterations):  
 # Train-test split  
 X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42 + i)  
 X\_test, X\_eval, y\_test, y\_eval = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, stratify=y\_temp, random\_state=42 + i)  
  
 # Train the model  
 model.fit(X\_train, y\_train)  
  
 # Predict discrete labels  
 y\_pred = model.predict(X\_test) # Ensure this gives discrete labels  
  
 # Standard evaluation metrics  
 accuracy = accuracy\_score(y\_test, y\_pred)  
 precision = precision\_score(y\_test, y\_pred, average="weighted")  
 recall = recall\_score(y\_test, y\_pred, average="weighted")  
 f1 = f1\_score(y\_test, y\_pred, average="weighted")  
 cm = confusion\_matrix(y\_test, y\_pred)  
  
 print(f"\nIteration {i + 1} Results:")  
 print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}")  
 print(f"Confusion Matrix:\n{cm}")  
  
 # K-Fold evaluation  
 k\_fold\_results = adaptive\_k\_fold\_evaluation(model, X\_train, y\_train, k\_min, k\_max, max\_duration)  
 print(f"\nK-Fold Results ({k\_fold\_results['Folds']} folds):")  
 print(f"Average Metrics: {k\_fold\_results['Average Metrics']}")  
 print(f"Total Duration: {k\_fold\_results['Total Duration']:.2f}s")  
  
 # Save iteration results  
 iteration\_metrics.append({  
 "Iteration": i + 1,  
 "Accuracy": accuracy,  
 "Precision": precision,  
 "Recall": recall,  
 "F1-Score": f1,  
 "Confusion Matrix": cm,  
 "K-Fold Average Metrics": k\_fold\_results['Average Metrics']  
 })  
  
 # Extract and map model equation or feature importance  
 feature\_importance = extract\_equation\_and\_interpret(model, vectorizer, model\_name)  
  
 # Aggregate metrics over iterations  
 avg\_metrics = {  
 "Model": model\_name,  
 "Average Accuracy": np.mean([m["Accuracy"] for m in iteration\_metrics]),  
 "Average Precision": np.mean([m["Precision"] for m in iteration\_metrics]),  
 "Average Recall": np.mean([m["Recall"] for m in iteration\_metrics]),  
 "Average F1-Score": np.mean([m["F1-Score"] for m in iteration\_metrics]),  
 "Average K-Fold Metrics": np.mean([m["K-Fold Average Metrics"] for m in iteration\_metrics], axis=0),  
 "Feature Importance": feature\_importance,  
 }  
 final\_results.append(avg\_metrics)  
  
 print(f"\nAverage Results for {model\_name}:\n{avg\_metrics}\n{'=' \* 50}")  
  
return final\_results

from collections import Counter import pandas as pd

# **df['cleaned\_text\_lemm'] -> Text column**

# **df['class'] -> Target column**

# **Check initial class distribution**

print("Class distribution before undersampling:", Counter(df['class']))

# **Separate majority and minority classes**

minority\_class = df[df['class'] == df['class'].value\_counts().idxmin()] majority\_class = df[df['class'] == df['class'].value\_counts().idxmax()]

# **Undersample majority class**

majority\_class\_undersampled = majority\_class.sample(n=len(minority\_class), random\_state=42)

# **Combine minority class with undersampled majority class**

df\_balanced = pd.concat([minority\_class, majority\_class\_undersampled])

# **Shuffle the dataset**

df\_balanced = df\_balanced.sample(frac=1, random\_state=42).reset\_index(drop=True)

# **Check class distribution after undersampling**

print("Class distribution after undersampling:", Counter(df\_balanced['class']))

df\_balanced.head()

# **Define models**

models = [ ("Logistic Regression", LogisticRegression(max\_iter=500)), ("XGBoost", XGBClassifier(use\_label\_encoder=False, eval\_metric="logloss")), ]

# **Vectorize the data**

vectorizer = TfidfVectorizer(analyzer="word", max\_features=5000) X = vectorizer.fit\_transform(df\_balanced['cleaned\_text\_lemm']) y = df\_balanced['class']

# **Train and Evaluate**

final\_results = train\_evaluate\_with\_equations(models, X, y, vectorizer, iterations=5, k\_min=5, k\_max=10, max\_duration=60)