Data 607 - Project Report

Precision in Perception: Sentiment Analysis for Apple Vision Pro through YouTube Comment Text Classification

607: Statistical & Machine Learning
University of Calgary
2024

Bo Li (30212597)

Brian Ho (30222881)

James Huvenaars (30031411)

Ethan Scott (30117295)

TABLE OF CONTENTS

1. INTRODUCTION

1.1. MOTIVATION

In today's digital age, where our phones serve as gateways to a plethora of social media platforms, the landscape of marketing has evolved dramatically. TikTok, Instagram, Facebook, and YouTube stand as bustling hubs where brands vie for attention amidst a sea of content. With a global social media user base expected to surge beyond 5.85 billion by 2027 (Wong, 2023), the significance of these platforms for marketers cannot be overstated. Yet, amidst this vast expanse of digital chatter lies a wealth of data waiting to be harnessed. Effective social media management tools hold the key to unlocking this potential, enabling brands to decipher audience sentiment and inform strategic decisions with clarity and foresight.

1.2. PROBLEMS

For corporations, quantifying the impact of their marketing investments poses a formidable challenge. How can one precisely gauge the influence of past road shows on newly acquired customers? Is today's order a direct result of a comment left by a viewer during last quarter's product launch? Beyond mere dissemination, can social media serve as a dual-purpose platform, not only spreading information but also gathering insights into customer sentiments? Such data not only inform the refinement of marketing approaches but also signal adjustments necessary for future product launches.

1.3. OBJECTIVES

In this project, our focus centers on unravelling the sentiments expressed within YouTube comments, particularly those surrounding the Apple Vision Pro review. By employing classification techniques, we aim to discern whether these comments lean towards the "Positive," "Neutral," or "Negative" spectrum, offering valuable insights into potential sales estimations and audience perceptions. Through meticulous analysis, we aim to empower marketers with the knowledge needed to navigate the complex digital engagement landscape effectively.

2. METHODOLOGY

2.1 DATA

Our dataset comprises comments scraped from the most viewed YouTube video discussing the Apple Vision Pro review hosted by prominent YouTuber Marques Brownlee (“Apple Vision Pro Review: Tomorrow’s Ideas... Today’s Tech!” n.d.). With over 79 million views and 12,000 comments, this dataset offers a rich tapestry of opinions and perspectives. Leveraging the YouTube Data API, we accessed publicly available comments to construct our dataset, which includes

variables such as commenter handle, comment text, timestamp, likes, reply count, and sentiment analysis polarity.

The variables measured in the original dataset and their corresponding descriptions are below:

Table 1. Original Dataset:

Variable Name	Description
Name	The handler of the person who made the comment
Comment	The actual comment from the viewer
Time	The actual time the comment was left
Likes	How many likes did the comment get?
Reply Count	Is there a reply to the comment? (binary: "1", "0")
Analysis_Polarity	As "Y," The sentiment of the comment ('Negative,' 'Positive,' 'Neutral')

Table 2. Examples of "Comment":

Variable Name	Frequency of the word
"the"	13859
"to"	7638
"a"	7495
"it"	7414
'i"	6748

2.2 APPROACH

Based on the provided dataset, our central goal is identifying the most effective model for classifying audience sentiment towards Apple's latest product, the Apple Vision Pro, based on the sentiment of their comments and replies. While some developed Python libraries like TextBlob provide a convenient solution for sentiment analysis and classification, they usually use a pre-trained lexicon-based approach for sentiment analysis (Shah, 2020). Building a custom model allows us to tailor the sentiment analysis process specifically to the characteristics of our data that can improve the accuracy and relevance of sentiment classification. Thus, our approach entails evaluating various candidate models constructed using different statistical methods. By building custom models tailored to the characteristics of our data, we aim to enhance our understanding of audience attitudes and intentions, thereby informing product refinement and optimization strategies. we will focus on the following questions to guide our project:

1. What's the data look like?
2. How do we use the data?
3. What models can be used to analysis the data?
4. Can we use this model to predict future sentiment?

2.3 WORKFLOW

1. **Collect Data**
2. **Data Exploration**
3. **Data Processing**
4. **Conditions Checking**
5. **Model Development**
    - 5.1.Gaussian Naive Base Model
    - 5.2.Logistic Regression Model
    - 5.3.Stochastic Gradient Descent (SGD) Model
    - 5.4.Random Forest
    - 5.5.XGBoost
    - 5.6.K-Nearest Neighors(KNN)
    - 5.7.Support Vector Machine (SVM)
6. **Prediction and Result Interpretations**
    - 6.1.Accuracy Score
    - 6.2.ROC & AUC
7. **Conclusion and Reflection**

3. Collect Data

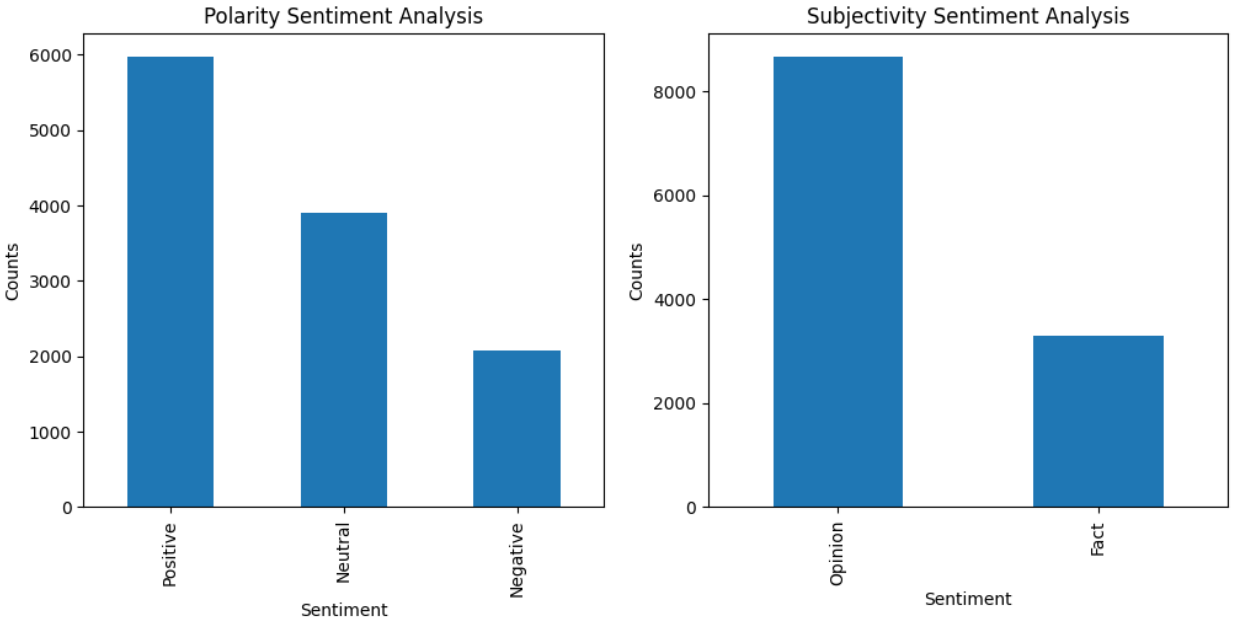
```
# Scrape Or Download Comments Using Python Through The Youtube Data API

api_key = "AIzaSyB1_1Nu0qpmlD8WQmG_FkTwOka3I4gFyXs"


youtube = build('youtube', 'v3', developerKey=api_key)

ID = "86Gy035z_KA"

box = []


def scrape_comments_with_replies():
    data = youtube.commentThreads().list(part='snippet', videoId=ID, maxRe

    for i in data["items"]:

        name = i["snippet"]['topLevelComment']["snippet"]["authorDisplayNa
        comment = i["snippet"]['topLevelComment']["snippet"]["textDisplay"
        published_at = i["snippet"]['topLevelComment']["snippet"]['publish
        likes = i["snippet"]['topLevelComment']["snippet"]['likeCount']
        replies = i["snippet"]['totalReplyCount']

        box.append([name, comment, published_at, likes, replies])

        totalReplyCount = i["snippet"]['totalReplyCount']
```

Leveraging the YouTube Data API, we accessed publicly available comments to construct our dataset. YouTube Data API | Google Developers, 2019)

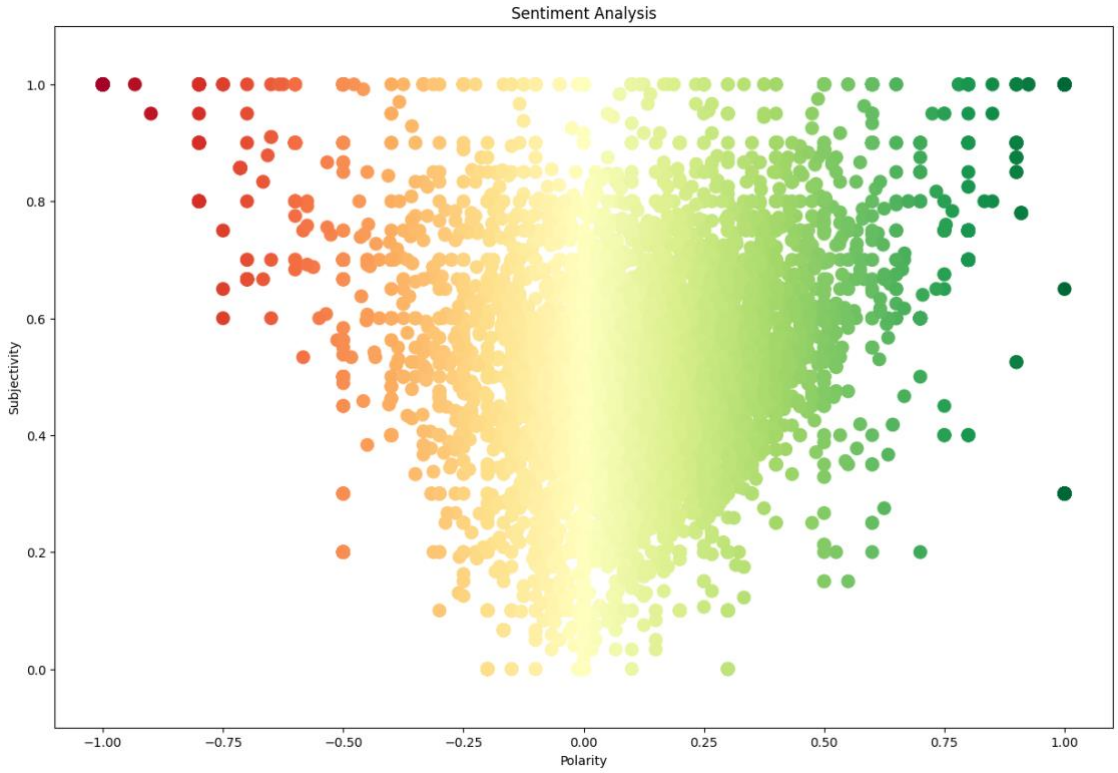
4. DATA EXPLORATION

Upon scraping the comments, we initially utilized TextBlob Library, a model for text processing, to perform basic sentiment analysis, providing us with foundational insights into comment polarity. This initial analysis served as the groundwork for our subsequent exploration and model construction.

During our initial exploration of the dataset, we observed that approximately half of the comments exhibited a positive sentiment, while slightly more than a third were neutral, and just under a third leaned negative. Additionally, common words in the dataset largely comprised connector words, offering limited substantive insights into the comments' content.



Subsequently, we refined our analysis by filtering out connector words, revealing a more nuanced view of the comment composition. While many words remained neutral, terms such as "Apple," "headset," and "vision" emerged prominently. Intriguing words like "expensive" and "interesting" surfaced, offering potential indicators of audience perceptions.



These initial insights serve as pivotal benchmarks for evaluating our model's effectiveness. The distribution of positive comments is particularly important, as our model's success hinges on surpassing this baseline distribution. Achieving accuracy greater than fifty percent is imperative; otherwise, the model would merely default to labelling all comments as positive to achieve a higher correctness rate.

5. DATA PROCESSING

5.1 Data Cleaning and Formatting

We began by analyzing the frequency of words in the comment column of our dataset. We initialized an empty list to store individual words from each comment. We iterated through each comment, splitting it into words and extending the word list with these words. After processing all comments, we created a word dictionary that counts the occurrences of each word in the word list and filtered the dictionary to retain only words that occur at least twice (minimum frequency) while no more than 1500 times (maximum frequency). By doing so, we eliminated infrequent words that might not contribute much to the analysis and removed highly frequent words that could be common stop words.

If we do a comparison using Word Clouds to show between the original whole dataset and the one after trimming using frequency as criteria, you can see the difference below:

```
[ ]  MIN_FREQ = 2
     MAX_FREQ = 1500

     word_counts = {key: word_counts[key] for key in word_counts if word_counts
     len(word_counts)

     7163
```

To create the dataset for model training and testing, we first created an index dictionary where each word from the filtered word dictionary was mapped to an integer index. This index was used to represent each word as the features in a feature matrix with dimensions 11956x7164 (number of comments, number of features/filtered words). Next, we iterated over each comment and split it into individual words. We then checked if each word was present in the index dictionary. If the word existed, we updated the corresponding entry in the feature matrix by incrementing the count of that word at its respective index. After this process, the feature matrix contained the count of each filtered word for each comment in the dataset.

5.2 Data Dimension Reduction and Train-Test splitting

Due to the dataset's significantly high number of features, Principal Component Analysis (PCA) was performed for dimensional reduction. Through several trials, we set the number of principal components as 333, where these components capture over 98% of the total variance.

Subsequently, the feature matrix obtained from PCA transformation was split into training and testing sets, with a test size of 20%. Before proceeding with model training, the data was standardized using `StandardScaler` to ensure that each feature had a mean of 0 and a standard deviation of 1. This was performed separately for the training and testing sets to prevent data leakage. On the other hand, label encoding was utilized to convert the categorical target variable `Analysis_Polarity` into numerical labels. The labels were encoded such that 'Negative' corresponded to 0, 'Neutral' to 1, and 'Positive' to 2

pca.eigenvalues_summary

component	eigenvalue	% of variance	% of variance (cumulative)
0	22.451	6.68%	6.68%
1	2.781	0.83%	7.51%
2	2.422	0.72%	8.23%
3	2.330	0.69%	8.92%
4	2.133	0.63%	9.56%
...	...	...	...
328	0.458	0.14%	99.13%
329	0.452	0.13%	99.27%
330	0.451	0.13%	99.40%
331	0.440	0.13%	99.53%
332	0.436	0.13%	99.66%

6. CONDITIONS CHECKING

6.1 Independence

After applying PCA, the condition for Independence will be applied and this condition is required during the model development process.

6.2 Normality

As dataset size with 11966 rows and 7163 columns. The condition for normality can also be assumed. This assumption is also needed during the model development process.

7. MODEL DEVELOPMENT

Following data preprocessing, we proceeded with model building, utilizing seven distinct machine learning algorithms: Naive Bayes, Logistic Regression, Stochastic Gradient Descent, Random Forest, XGBoost, K-Nearest Neighbors, and Support Vector Machine (SVM). Hyperparameter tuning and cross validation were performed to optimize predictive accuracy and robustness across all models.

7.1 Gaussian Naive Base

Naive Bayes is a probabilistic classification model based on Bayes' theorem with an assumption of independence among features. It's particularly efficient for text classification tasks due to its simplicity and scalability, making it suitable for large datasets with high-dimensional feature spaces. Naive Bayes often performs surprisingly well in real-world applications, especially when the independence assumption holds reasonably well.

Fine Tuning Parameters:

For Gaussian Naive Base model, there is no parameters for fine tuning.

Accuracy Score for Gaussian NB:

Classification Accuracy of Naive Bayes: 0.5033

7.2 Logistic Regression

Logistic Regression is a linear model used for classification tasks, estimating the probability of an outcome using a logistic function. It's interpretable, providing insights into the relationship between the independent variables and the probability of the outcome. Logistic Regression is robust to noise and can handle both categorical and numerical features, making it widely used in various fields such as healthcare, finance, and marketing for its simplicity and effectiveness.

Fine Tuning Parameters:

```
param_grid = {
    'max_iter': [10000],
     "C": [0.1, 0.5, 1, 10, 15, 20, 50, 100],
    'class_weight': ['dict', 'balanced'],  # Weight function used in prediction
    'penalty': ['l2', 'l1', 'elasticnet'],  # Penalty term
}

grid_search = GridSearchCV(LogisticRegression(), param_grid=param_grid, scoring="accuracy", cv=5)
grid_search.fit(X_train, y_train)
```

Best Parameters and Accuracy Score for Logistic Regression:

Classification Accuracy of LR: 0.7099

Best parameters for this model: {'C': 20, 'class_weight': 'balanced', 'max_iter': 10000, 'penalty': 'l2'}

7.3 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm used for training machine learning models, particularly useful for large-scale datasets. It updates the model parameters iteratively by computing the gradient of the loss function on a subset of the training data. SGD is computationally efficient and can be parallelized, making it suitable for models with a large number of parameters and training instances. However, it may require careful tuning of hyperparameters and regularization to prevent overfitting.

Fine Tuning Parameters:

```
param_grid = {
    'max_iter': [10000],
    'alpha': [0.0001, 0.001, 0.01, 0.1, 1],  # Regularization parameter
    'loss': ['hinge', 'log_loss', 'modified_huber', 'squared_hinge'],  # Loss function
    'penalty': ['l2', 'l1', 'elasticnet'],  # Penalty term
}

grid_search = GridSearchCV(SGDClassifier(), param_grid=param_grid, scoring="accuracy", cv=5)
grid_search.fit(X_train, y_train)
```

Best Parameters and Accuracy Score for SGD:

Classification Accuracy of SGD: 0.7012

Best parameters for this model: {'alpha': 0.001, 'loss': 'log_loss', 'max_iter': 10000, 'penalty': 'elasticnet'}

7.4 Random Forest

Random Forest is an ensemble learning method that builds multiple decision trees during training and combines their predictions through averaging or voting. It's known for its robustness against overfitting and ability to handle both categorical and numerical data well. Random Forest can capture complex relationships between features and target variables, making it suitable for various classification and regression tasks. Additionally, it provides insights into feature importance, aiding in feature selection and interpretation.

Fine Tuning Parameters:

```
param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node
}
grid_search = GridSearchCV(RandomForestClassifier(), param_grid=param_grid, scoring="accuracy", cv=5)
```

Best Parameters and Accuracy Score for Random Forest:

Classification Accuracy of Random Forest method: 0.6611

Best parameters for this model: {'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 200}

7.5 XGBoost (Extreme Gradient Boosting)

XGBoost (Extreme Gradient Boosting) is a scalable and efficient implementation of gradient boosting, which sequentially builds an ensemble of weak learners and minimizes errors by adjusting the weights of misclassified instances. It's highly customizable, allowing users to specify different objectives and evaluation criteria. XGBoost is known for its high predictive accuracy and speed, making it a popular choice for winning solutions in data science competitions and industry applications.

Fine Tuning Parameters:

```
param_grid = {
    'n_neighbors': [3, 5, 10, 20],  # Number of neighbors to use for kneighbors queries
    'weights': ['uniform', 'distance'],  # Weight function used in prediction
    'metric': ['euclidean', 'manhattan'],  # Distance metric
}
grid_search = GridSearchCV(XGBClassifier(), param_grid=param_grid, scoring="accuracy", cv=5)
pipe = grid_search.fit(X_train, y_train)
```

Best Parameters and Accuracy Score for XGB:

Classification Accuracy of XGB method: 0.6966

Best parameters for this model: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 200}

7.6 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple and intuitive classification algorithm that classifies a data point based on the class labels of its nearest neighbors in the feature space. It doesn't require training a model during the training phase, making it computationally inexpensive. However, KNN can be sensitive to the choice of distance metric and the number of neighbors (k), and it may suffer from the curse of dimensionality for high-dimensional data.

Fine Tuning Parameters:

```
param_grid = {
    'weights': ['uniform', 'distance'], # Weight function used in prediction
    'algorithm': ['auto', 'ball_tree', 'kd_tree'],
    'p': [1, 2], # This parameter defines the power parameter for the Minkowski distance metric.
    'n_neighbors': [5, 10, 20, 40, 80, 150] # Number of neighbors to use for kneighbors queries
}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, scoring="accuracy", cv=5)
pipe = grid_search.fit(X_train, y_train)
```

Best Parameters and Accuracy Score for KNN:

Classification Accuracy of KNN method: 0.5077

Best parameters for this model: {'algorithm': 'kd_tree', 'n_neighbors': 5, 'p': 2, 'weights': 'distance'}

7.7 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. It constructs a hyperplane in a high-dimensional space to separate data points into different classes with a maximal margin, aiming to maximize the margin between classes while minimizing the classification error. SVM is effective in handling both linear and nonlinear classification tasks through the use of kernel functions, making it versatile for various applications. Additionally, SVM is robust to overfitting and performs well in high-dimensional spaces, although it may require careful selection of hyperparameters and kernel functions.

Fine Tuning Parameters:

```
param_grid = {
    "C": [0.1, 0.5, 1, 10, 15, 20, 50, 100], #This parameter controls the regularization strength in SVM
    "kernel": ('linear', 'rbf', 'sigmoid'), # decision boundary and the flexibility of the model
    "gamma": [1, 0.1, 0.01, 0.001, 0.0001] #the width of the Gaussian kernel and influences the influence of each training example
}
grid_search = GridSearchCV(SVC(), param_grid=param_grid, scoring="accuracy", cv=5)
pipe = grid_search.fit(X_train, y_train)
```

Best Parameters and Accuracy Score for SVM:

Classification Accuracy of KNN method: 0.7391

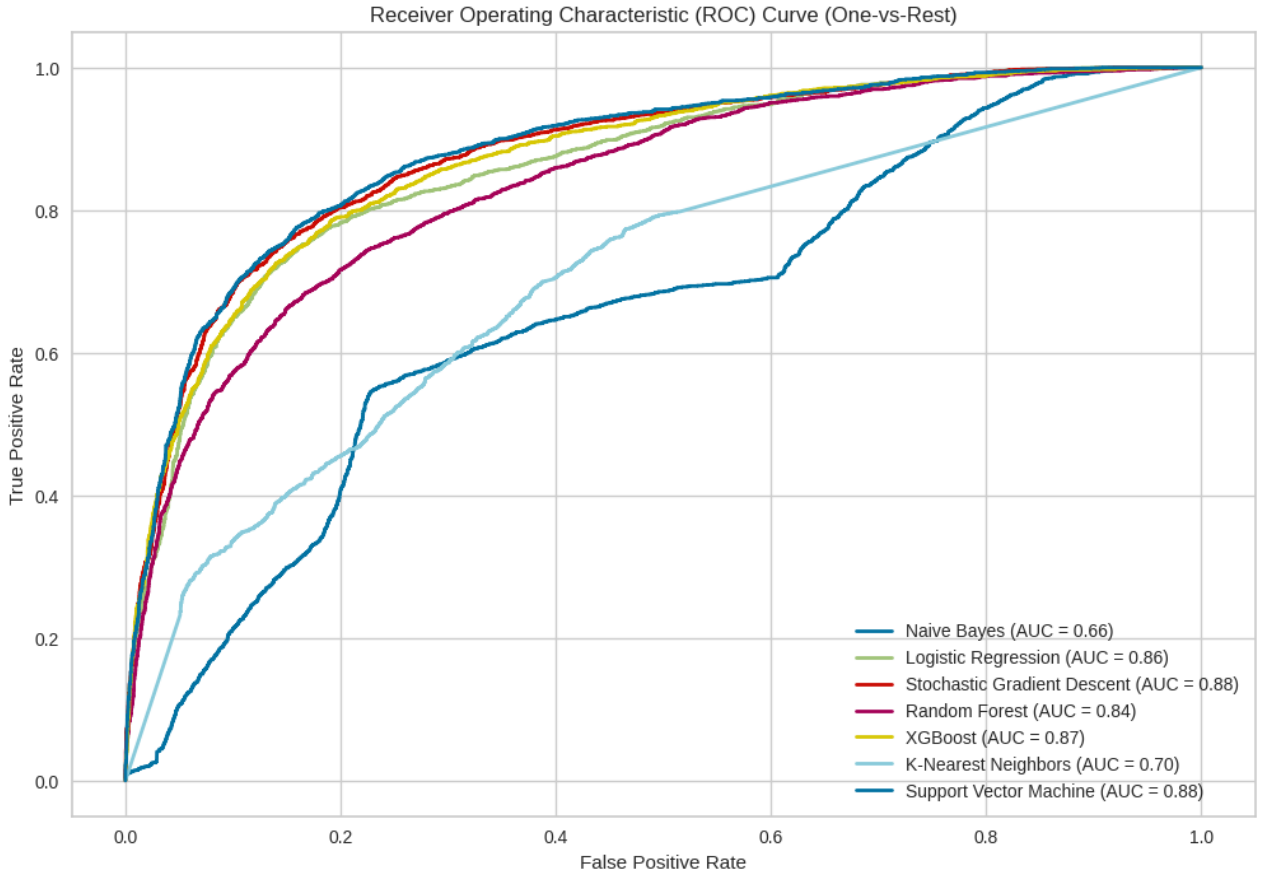
Best parameters for this model: {C = 1, kernel = 'linear', gamma = 1}

8. RESULTS

Throughout model development, we assessed performance using two primary metrics: accuracy score and Area Under the Curve (AUC). These metrics provided insights into each model's ability to classify sentiment in YouTube comments accurately. Models demonstrating higher accuracy scores and AUC values were prioritized for further evaluation and refinement.

By systematically evaluating and comparing the performance of each model, we aimed to identify the most effective classifier for our sentiment analysis task. This approach enabled us to select a model capable of accurately discerning and categorizing sentiment nuances within the dataset, facilitating informed decision-making and strategic planning.

The findings of our models are below:



Model Name	Accuracy Score	AUC
Naive Bayes	0.5033	0.65
Logistic Regression	0.7099	0.86
Stochastic Gradient Descent	0.7012	0.88
Random Forest	0.6611	0.84
XGBoost	0.6966	0.87
K-Nearest Neighbors	0.5077	0.80
Support Vector Machine	0.7391	0.88

9. CONCLUSION AND DISCUSSION

In conclusion, our investigation into sentiment analysis of YouTube comments regarding the Apple Vision Pro review has provided us with significant findings. Our thorough examination of different models has determined that Support Vector Machine (SVM) stands out as the most reliable classifier. It achieved an accuracy score of 73.91% and an impressive AUC of 0.88, showcasing its effectiveness in accurately discerning sentiment within the dataset.

This outcome validates our initial aim of identifying the best model for gauging audience sentiment towards Apple's product. By opting for SVM, we achieved precise sentiment classification and laid a solid foundation for marketers and businesses to make informed decisions and plan strategically in the social media landscape.

Furthermore, the insights gained from this project are not limited to this specific case. They offer valuable lessons that can be applied to similar topics and datasets across various industries. By adopting a robust methodology and gaining insights from our analysis, we contribute to a better understanding of sentiment analysis techniques and their practical applications in extracting meaningful insights from user-generated content on social media platforms.

REFERENCES

Shah, P. (2020, June 27). Sentiment Analysis using TextBlob - Towards Data Science. Medium; Towards Data Science. https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524

Wong, B. (2023, May 18). Top Social Media Statistics and Trends of 2024 (C. Bottorff, Ed.). Forbes. https://www.forbes.com/advisor/business/social-media-statistics/

"Apple Vision pro Review: Tomorrow's Ideas... Today's Tech!" n.d. Www.youtube.com. Accessed March 16, 2024. https://www.youtube.com/watch?v=86Gy035z_KA&t=487s.

YouTube Data API | Google Developers. (2019). Google Developers. https://developers.google.com/youtube/v3

End of Final Project Report