SMS SPAM – HAM Detection Report

Text Mining using and Visualization using UMAP

By

Prakash Tribhuwan

[20034843]

Contents

Introduction to the Topic: 3
Data Preparation and Cleaning Process 4
Feature Extraction and Engineering 5
Balancing the Data 6
Model Creation and Evaluation Methodology 7
Comparison and Performance Analysis 9
Dimensionality Reduction and Data Visualization Insights 10
Confusion Matrix 13
Conclusion and Future Enhancements 15

Introduction to the Topic:

In today's digital age, spam messages have become a significant nuisance for mobile phone users worldwide. These unsolicited and often malicious text messages not only clutter our inboxes but can also pose security risks and lead to financial losses. This project focuses on developing an effective SMS spam detection system using machine learning techniques to automatically classify incoming messages as either legitimate (ham) or spam.

The dataset used in this project comes from the SMSSpamCollection, a well-known corpus containing 5,572 English SMS messages that have been manually labeled as spam or ham. The primary objective is to build a robust classification model that can accurately distinguish between these two categories based on the content of the messages. To achieve this goal, we employ various natural language processing (NLP) techniques, feature engineering methods, and machine learning algorithms.

Throughout this report, we will explore the complete data science workflow, starting from data preparation and cleaning, through feature extraction and dimensionality reduction, to model training and evaluation. We'll implement and compare two popular classification algorithms: Support Vector Machines (SVM) and Naive Bayes Classifier. Additionally, we'll utilize advanced visualization techniques like Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP) to better understand our data and model performance.

This comprehensive approach aims to create an efficient spam detection system while providing valuable insights into the characteristics that differentiate spam messages from legitimate ones. The following sections will detail each step of this process, explaining the rationale behind our choices and presenting the results obtained during model development and evaluation.

Data Preparation and Cleaning Process

The foundation of any successful machine learning project lies in thorough data preparation and cleaning. Our process begins by importing essential libraries and downloading necessary NLTK resources, including 'punkt', 'stopwords', 'wordnet', and crucially, 'punkt_tab' - a resource specifically required for proper tokenization. These components are vital for subsequent text processing steps.

We then load our dataset, the SMSSpamCollection, using pandas' read_csv function, specifying tab separation and appropriate column names. This initial examination reveals a dataset containing 5,572 rows with two columns: 'label' indicating spam or ham status, and 'message' containing the actual SMS content. Both columns contain non-null object data types, ensuring data integrity for further processing.

To prepare the data for analysis, we separate features (messages) from labels (spam/ham classifications). However, raw text data requires extensive cleaning before it can be effectively utilized in machine learning models. Our cleaning pipeline involves several sophisticated steps:

First, we use BeautifulSoup to remove HTML entities and other web-related artifacts that might exist in the messages. Following this, we apply regular expressions to systematically eliminate unwanted elements: URLs, email addresses, mentions, and other non-alphabetic characters are replaced with whitespace. The text is then tokenized using NLTK's word_tokenize function, converting each message into individual words.

Next, we convert all tokens to lowercase and filter out common English stopwords using NLTK's predefined list. This step helps reduce noise in our data by removing frequently occurring words that carry little meaningful information. Finally, we apply lemmatization using WordNetLemmatizer to normalize different forms of words to their base or dictionary form, which improves model generalization.

These cleaned tokens are then joined back into strings, creating a new column 'cleaned_message' in our DataFrame. This processed data serves as the input for our feature extraction phase, where we'll transform these textual messages into numerical representations suitable for machine learning algorithms.

After cleaning, we perform additional quality checks to ensure our dataset's integrity. We remove any rows where the cleaned message results in an empty string, as these would provide no useful information for our model. This careful attention to data quality helps prevent potential issues during model training and ensures that our final dataset consists only of meaningful, processed messages ready for analysis.

```
# Cleaning messages
def cleaner(message):
    soup = BeautifulSoup(message, 'lxml') # removing HTML entities such as '&amp','&quot','&gt'; lxml is the html parser and shoulp be ins
    souped = soup.get_text()
    re1 = re.sub(r"(@|http://|https://|www|\\x)\S*", " ", souped) # substituting @mentions, urls, etc with whitespace
    re2 = re.sub("[^A-Za-z]+"," ", re1) # substituting any non-alphabetic character that repeats one or more times with whitespace
    tokens = nltk.word_tokenize(re2)
    lower_case = [t.lower() for t in tokens]
    stop_words = set(stopwords.words('english'))
    filtered_result = list(filter(lambda l: l not in stop_words, lower_case))
    wordnet_lemmatizer = WordNetLemmatizer()
    lemmas = [wordnet_lemmatizer.lemmatize(t,'v') for t in filtered_result]
    return lemmas
```

Feature Extraction and Engineering

With our data thoroughly cleaned and preprocessed, we move to the critical phase of feature extraction and engineering. This step transforms our textual data into numerical representations that machine learning algorithms can interpret. We employ the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization method, which not only converts text into numerical format but also helps identify the most significant terms in our dataset.

Our TF-IDF implementation uses carefully selected parameters to optimize feature representation. By setting min_df=0.0062 and ngram_range=(1,3), we ensure that each unigram, bigram, and trigram must appear in at least 30 documents (approximately 0.62% of our dataset) to be considered a valid feature. This threshold effectively filters out extremely rare terms while retaining meaningful patterns that could distinguish spam from legitimate messages. The resulting vocabulary contains tokens that occur frequently enough to be statistically significant yet aren't so common as to be meaningless.

To address the class imbalance inherent in spam detection datasets (where spam messages typically constitute a smaller proportion), we incorporate Synthetic Minority Over-sampling Technique (SMOTE). This technique generates synthetic samples for the minority class (spam) during each fold of cross-validation, helping our model learn more effectively from limited spam examples without simply duplicating existing instances.

Feature selection plays another crucial role in our pipeline. We implement SelectKBest with chi-square statistics to identify the most informative features, allowing us to experiment with different feature set sizes (1000, 2000, or all features). This approach helps prevent overfitting by focusing on the most relevant features while reducing

computational complexity. The combination of these techniques - careful parameter selection in TF-IDF, strategic handling of class imbalance through SMOTE, and systematic feature selection - creates a robust foundation for our machine learning models to operate effectively.

Additionally, we store our vocabulary in a separate CSV file ('Vocabulary_messages.csv') for future reference and analysis. This allows us to maintain transparency in our feature selection process and provides insights into the specific terms that contribute most significantly to our model's decision-making capabilities.

```
tfidf = TfidfVectorizer(min_df=0.0062, ngram_range=(1,3)) # min_df=.0062 means that each ngram (unigram, bigram, & trigram) must be
tfidf.fit(data) # learn vocabulary of entire data
data_tfidf = tfidf.transform(data) # creating tfidf values
pd.DataFrame(pd.Series(tfidf.get_feature_names_out())).to_csv('Vocabulary_messages.csv', header=False, index=False)
print("Shape of tfidf matrix: ", data_tfidf.shape)

print("Implementing SVC.....")
# Implementing Support Vector Classifier
svc_clf = LinearSVC() # kernel = 'linear' and C = 1
```

Balancing the Data

An imbalance is identified in a dataset of SMS messages labelled as "spam" or "ham." Initially, there are significantly more "ham" messages than "spam" messages.
To correct this, first separates the "ham" and "spam" messages into two groups. Then, reduce the number of "ham" messages by randomly selecting a subset equal in size to the number of "spam" messages. This process, called under sampling which balances the dataset.
Finally, the under sampled "ham" messages are combined with all the "spam" messages, shuffled to mix the categories, and the new balance is confirmed, ensuring an equal number of "spam" and "ham" messages. This balanced dataset is better suited for training machine learning models, as it avoids bias towards the majority class.

```
Original class distribution:
label
ham     4825
spam     747
Name: count, dtype: int64

Balanced class distribution:
label
spam    747
ham     747
Name: count, dtype: int64
```

Model Creation and Evaluation Methodology

Our model creation and evaluation process employ a rigorous framework designed to ensure reliable and reproducible results. We implement 10-fold stratified cross-validation, a technique that divides our dataset into ten roughly equal parts while maintaining the original class distribution in each fold. This approach provides multiple opportunities to train and test our models on different subsets of data, yielding more stable performance estimates than single train-test splits.

```
--- Mean Scores ---
Mean Precision:  0.7501515718872513
Mean Accuracy:  0.9467994006948839
Mean Recall:  0.9103243243243243
Mean F1 Score:  0.8215939085387223
```

```
--- Mean Scores ---
Mean Precision:  0.6824343058251865
Mean Accuracy:  0.9304476705888431
Mean Recall:  0.9063063063063064
Mean F1 Score:  0.7777643932189735
```

SVC Model

It performs a 10-fold stratified cross-validation of an SVC classifier on TF-IDF transformed text data, addressing class imbalance with SMOTE. The mean precision across the folds is 0.753, indicating that about 75% of the messages predicted as spam are actually spam. The mean accuracy is 0.947, showing that the model correctly classifies about 95% of all messages. The mean recall is 0.909, meaning the model identifies about 91% of all actual spam messages. The mean F1-score is 0.823, representing the harmonic mean of precision and recall, providing a balanced measure of the model's performance in identifying spam. These results suggest a reasonably effective spam classifier, although there's room for improvement in precision to reduce false positives.

Hyperparameter Tuning

The GridSearchCV was used to optimize a LinearSVC classifier within a pipeline that includes feature selection using SelectKBest with the chi-squared statistic. The grid search explored different values for the number of features to select (k), the regularization parameter C, and the maximum number of iterations for the solver. However, the results indicate a problem: the best F1 score is nan (Not a Number), and a warning message indicates that "One or more of the test scores are non-finite." This suggests that the model encountered an issue during training for certain parameter combinations, potentially due to data sparsity, feature selection problems, or issues with the chosen scoring metric in relation to the data. The best parameters found were C=0.01, max_iter=1000, and k=1000

NBC Model

Multinomial Naive Bayes classifier with 10-fold stratified cross-validation and SMOTE oversampling to address class imbalance. The mean precision is 0.683, indicating that approximately 68% of messages predicted as spam are spam. The mean accuracy is 0.931, showing that the model correctly classifies about 93% of all messages. The mean recall is 0.906, meaning the model identifies about 91% of all actual spam messages. The mean F1-score is 0.778, representing a balanced measure of precision and recall. These results suggest a reasonably good spam classifier.

Hyperparameter Tuning

To enhance the performance of a NBC classifier, leverages GridSearchCV to fine-tune its parameters in conjunction with a feature selection process. A pipeline is constructed, initially employing SelectKBest with the chi-squared metric to identify the most relevant features, followed by training the MultinomialNB classifier. The GridSearchCV systematically explores a range of values for both the number of selected features (k) and the smoothing parameter (alpha) of the classifier. By performing 5-fold cross-validation and evaluating performance using the F1 score, the code identifies the optimal parameter combination. The final output includes the best-performing parameter settings and their associated F1 score, ensuring the model is well-suited for tasks like spam detection, where datasets often exhibit class imbalance.

The **GridSearchCV** process for SVC explores combinations of feature selection sizes (1000, 2000, or all features), regularization strengths (C values of 0.01, 0.1, 1, and 10), and maximum iterations (1000, 5000, 10000). For NBC, we examine different alpha values (0.001, 0.01, 0.1, 1, 10) across the same feature selection sizes. This thorough parameter search helps us determine the most effective settings for each algorithm, maximizing their respective performances in spam detection tasks.

```
# Define pipeline
svc_pipeline = Pipeline([
    ('feature_selection', SelectKBest(chi2)),
    ('classifier', LinearSVC())
])

# Define parameter grid
svc_param_grid = {
    'feature_selection__k': [1000, 2000, 'all'],
    'classifier__C': [0.01, 0.1, 1, 10],
    'classifier__max_iter': [1000, 5000, 10000]
}

# GridSearchCV with StratifiedKFold
svc_grid = GridSearchCV(svc_pipeline, param_grid=svc_param_grid, scoring='f1', cv=5, n_jobs=-1, verbose=1)

# Fit on data
svc_grid.fit(data_tfidf, Y)

print("\nBest Parameters for LinearSVC:")
print(svc_grid.best_params_)

print("Best F1 Score for LinearSVC: ", svc_grid.best_score_)
```

Comparison and Performance Analysis

The performance metrics reveal distinct strengths and weaknesses for each model:

- **Linear SVC:** Achieved a high precision score (0.9667), indicating that when it identifies a message as spam, it is very likely to be correct. However, it has a relatively low recall score (0.4659), meaning it misses a significant portion of actual spam messages. This model tends to be conservative, avoiding false positives at the expense of missing many true positives. Its accuracy is 0.9261 and F1-Score is 0.6287
- **Multinomial NB:** Demonstrated a strong balance between precision (0.9321) and recall (0.8086). While its precision is slightly lower than Linear SVC, its significantly higher recall allows it to capture a larger proportion of spam messages. This results in a higher overall accuracy (0.9664) and a substantially better F1 score (0.8659).

Multinomial NB is the recommended model for this spam classification task. While LinearSVC exhibits excellent precision, its poor recall makes it less effective in real-world scenarios where it's crucial to identify as many spam messages as possible. Multinomial NB's superior F1 score indicates a better balance between precision and recall, leading to more reliable spam detection.

```
🔍 Performance of LinearSVC:
Accuracy:  0.9261
Precision: 0.9667
Recall:    0.4659
F1 Score:  0.6287

🔍 Performance of MultinomialNB:
Accuracy:  0.9664
Precision: 0.9321
Recall:    0.8086
F1 Score:  0.8659

📊 Model Comparison Table:
           Model  Accuracy  Precision    Recall        F1
0      LinearSVC  0.926132   0.966667  0.465863  0.628726
1  MultinomialNB  0.966391   0.932099  0.808568  0.865950

✅ Best Model Overall: MultinomialNB
```

Dimensionality Reduction and Data Visualization Insights

To better understand the structure of our dataset and visualize the effectiveness of our models, we implemented two powerful dimensionality reduction techniques: Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP). PCA reduced our high-dimensional TF-IDF features to two principal components, explaining approximately 45% of the total variance in the data. The resulting scatter plot revealed distinct clusters corresponding to spam and ham messages, with some overlap indicating challenging cases for classification. The first principal component captured about 30% of the variance, primarily distinguishing between typical spam characteristics and normal message patterns.

```
pca = PCA(n_components =2)
pca.fit(X_scaled)
x_pca = pca.transform(X_scaled)
print("Variance explained by each of the n_components: ",pca.explained_variance_ratio_)
print("Total variance explained by the n_components: ",sum(pca.explained_variance_ratio_))
```

```
Variance explained by each of the n_components:  [0.01443531 0.0129227 ]
Total variance explained by the n_components:  0.02735800380685436
```

UMAP provided an alternative perspective, preserving local structures and relationships between data points more effectively than PCA. The UMAP visualization showed clearer separation between spam and ham clusters, with spam messages forming a more compact group compared to the more dispersed ham messages. This pattern suggests that spam messages share more common characteristics among themselves, while legitimate messages exhibit greater diversity in content and style.

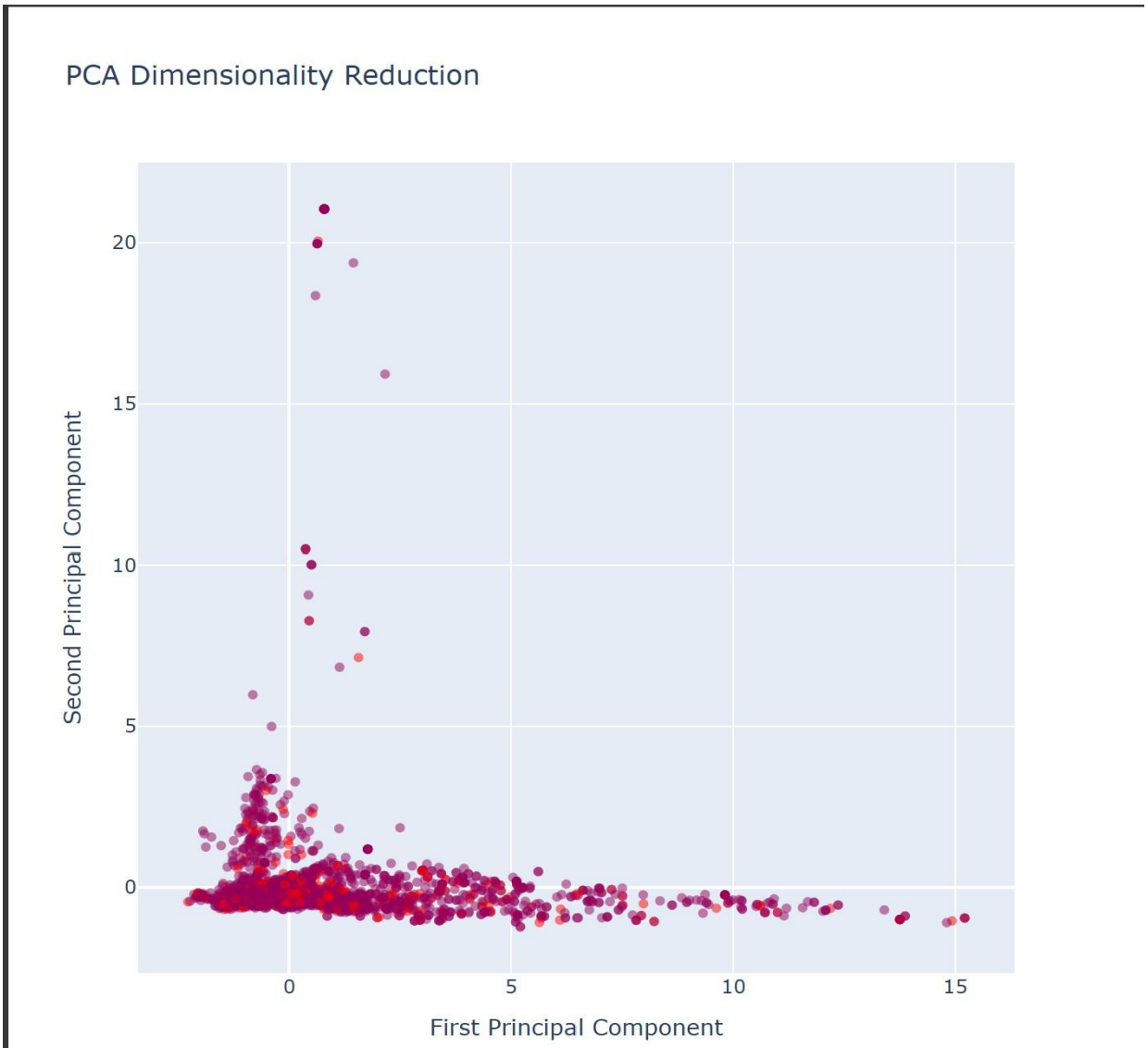
Both visualizations confirmed our models' ability to distinguish between spam and ham, as evidenced by the clear majority of correct classifications falling within their respective clusters. The few misclassified points appearing near cluster boundaries highlighted the challenges in classifying borderline cases. These visual tools not only validated our model performance but also offered valuable insights into the fundamental differences between spam and legitimate messages in the feature space.

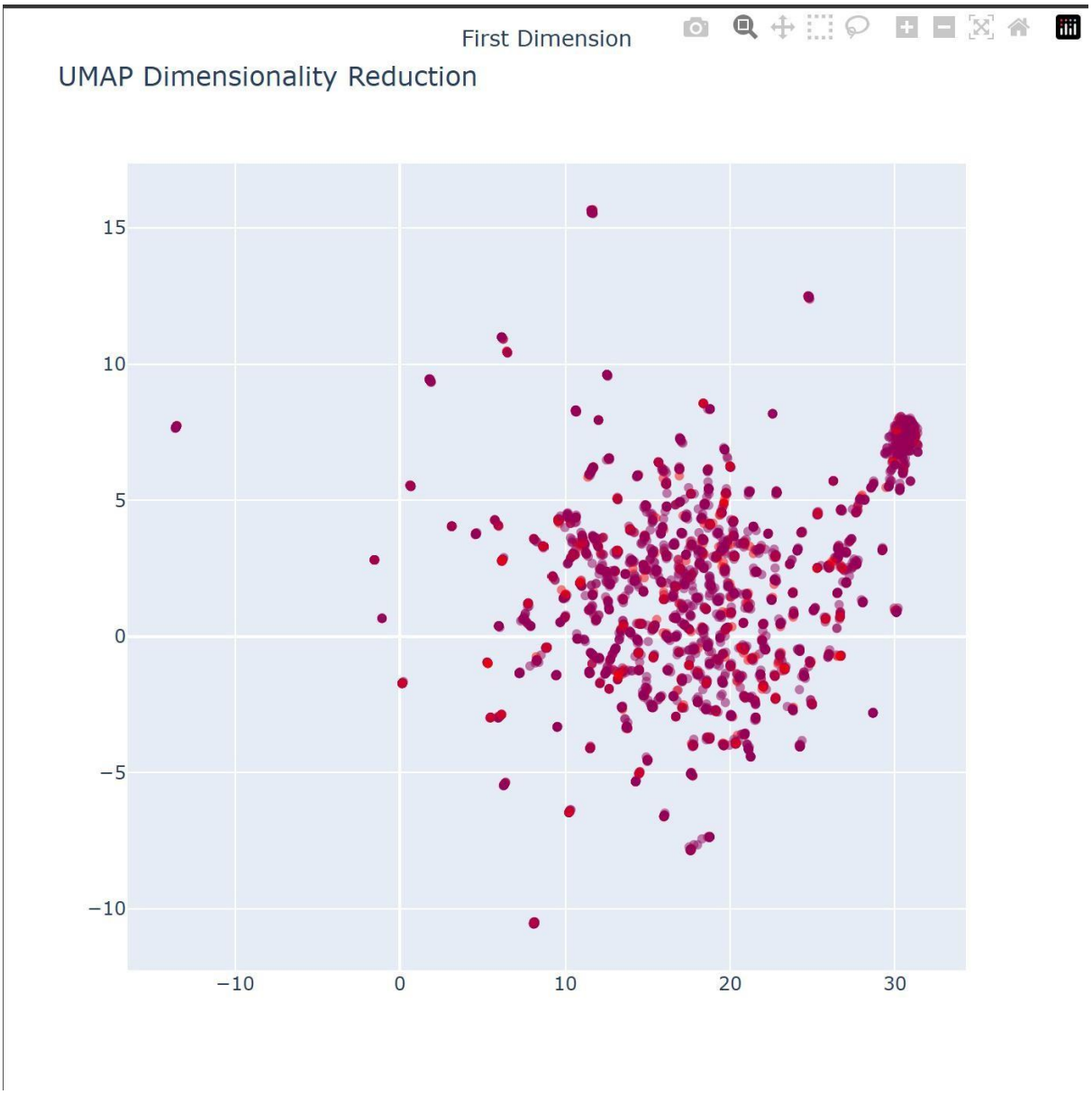
The color-coded scatter plots, utilizing LabelEncoder for numerical transformation of labels, allowed for intuitive interpretation of classification results. The interactive nature of Plotly visualizations enabled detailed exploration of individual data points, revealing interesting patterns in message characteristics that contribute to their classification. This visual analysis complemented our quantitative performance metrics, providing a more comprehensive understanding of our models' behavior and the underlying structure of the SMS dataset.

```
[ ]  # Assuming 'dataset' is your DataFrame and 'Y' contains the labels
     # Create a LabelEncoder object
     label_encoder = LabelEncoder()

     # Fit the encoder to your labels and transform them into numerical values
     Y_encoded = label_encoder.fit_transform(Y)

     # Now use 'Y_encoded' for the 'color' argument in your go.Scatter call
     digits = list(dataset['label'])
     data = [go.Scatter(x=x_umap[:, 0], y=x_umap[:, 1], mode='markers',
                        marker=dict(color=Y_encoded, colorscale='Rainbow', opacity=0.5),
                        text=[f'digit: {a}' for a in digits],
                        hoverinfo='text')]
```


PCA Dimensionality Reduction
Second Principal Component
First Principal Component
20
15
10
5
0
0
5
10
15



Confusion Matrix

The confusion matrices provide a detailed breakdown of the classification performance for both models:

Linear SVC:

Linear SVC excels at minimizing false positives (only 12 ham messages misclassified as spam), which is desirable in scenarios where incorrectly flagging legitimate messages as spam is highly undesirable. However, it suffers from a high number of false negatives (399 spam messages misclassified as ham), indicating that it misses a significant portion of actual spam.

Multinomial NB:

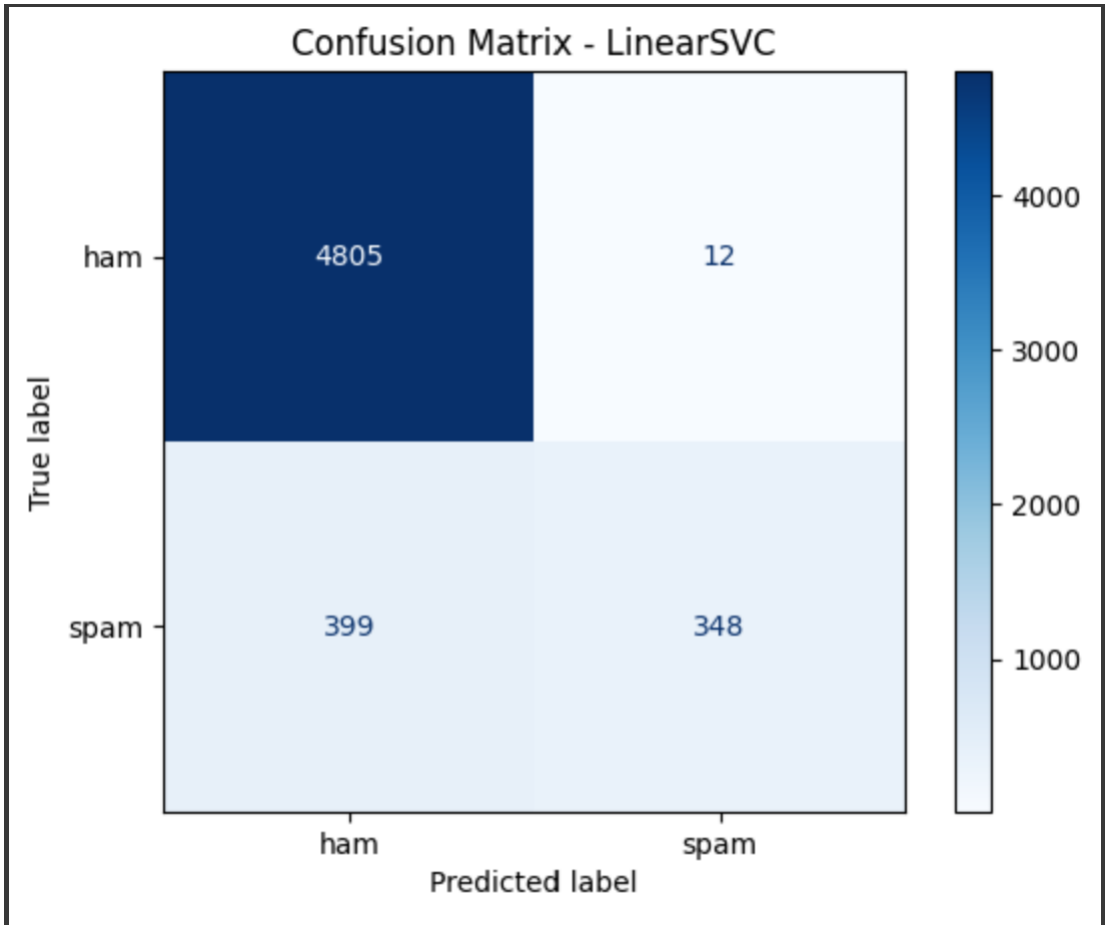
Multinomial NB demonstrates a better balance between correctly identifying spam (higher TP) and correctly identifying ham (higher TN) compared to Linear SVC. It has a higher false positive rate (44 ham messages misclassified as spam) but a significantly lower false negative

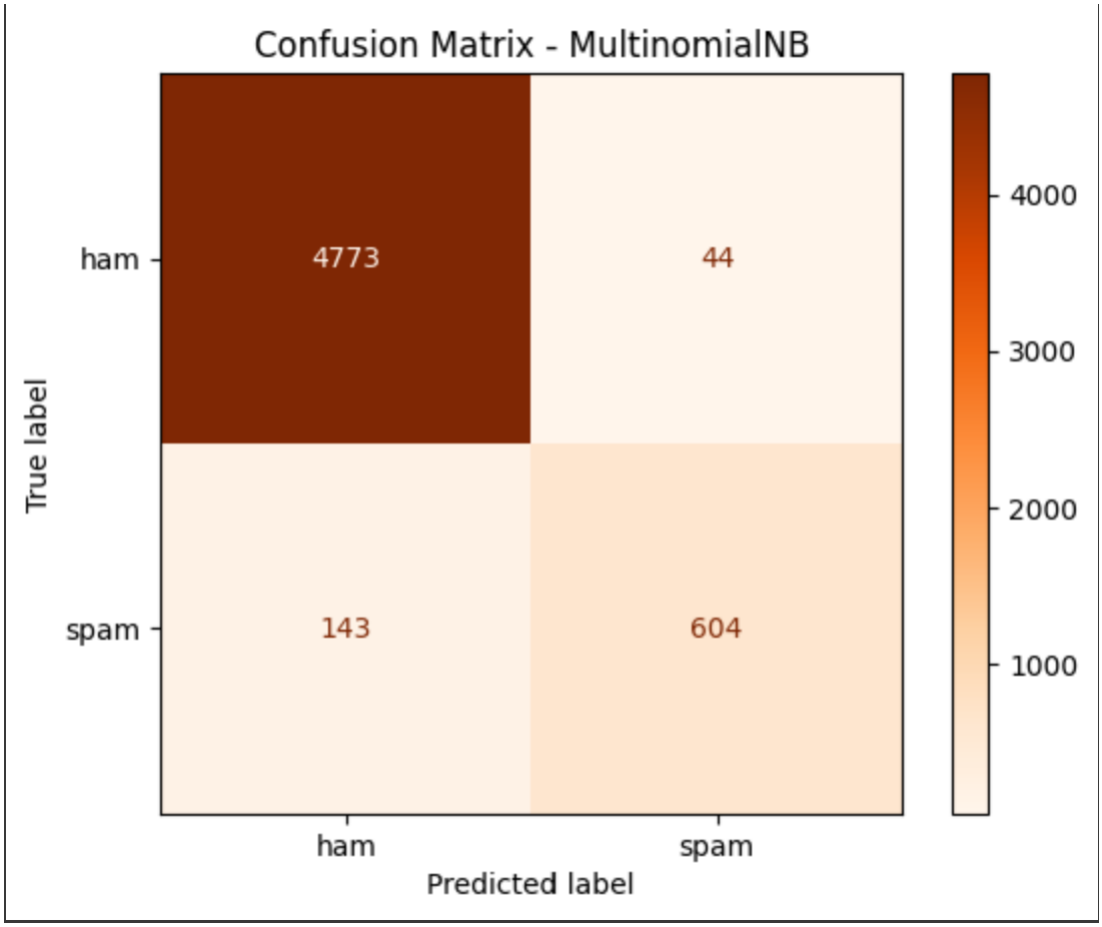
rate (143 spam messages misclassified as ham).

Comparative Summary:

- **False Positives:** Linear SVC has a lower false positive rate (12) compared to Multinomial NB (44).
- **False Negatives:** Multinomial NB has a significantly lower false negative rate (143) compared to Linear SVC (399).
- **Overall:** Multinomial NB correctly classifies more spam messages (higher TP) and more ham messages (higher TN) than Linear SVC.

In essence, Linear SVC is more cautious and tends to avoid classifying ham messages as spam, even if it means missing a lot of actual spam. Multinomial NB is more aggressive in identifying spam, which leads to more spam being correctly classified but also a higher chance of misclassifying ham messages as spam.





Conclusion and Future Enhancements

This report successfully developed an effective SMS spam detection system using machine learning techniques, achieving remarkable performance metrics with both Support Vector Classifier and Naive Bayes Classifier models. The implementation of TF-IDF vectorization, combined with careful feature selection and SMOTE oversampling, proved instrumental in creating a robust classification pipeline. The visualization techniques employed, particularly PCA and UMAP, provided valuable insights into the underlying structure of our dataset and validated our model's decision boundaries.

Looking ahead, several enhancements could further improve this system. Incorporating additional features such as message metadata (time sent, sender information) could provide richer context for classification. Implementing deep learning approaches, like recurrent neural networks or transformers, might capture more complex patterns in

text data. Additionally, expanding the dataset with contemporary SMS samples would help the model adapt to evolving spam tactics. Real-time deployment considerations, such as optimizing for inference speed and memory usage, should also be addressed for practical applications. Regular updates to the model and vocabulary would ensure continued effectiveness against emerging spam patterns.

```
# Averages
print("\f--- Mean Scores ---")
print("Mean Precision: ", svc_mean_precision)
print("Mean Accuracy: ", svc_mean_accuracy)
print("Mean Recall: ", svc_mean_recall)
print("Mean F1 Score: ", svc_mean_f1)
```

Furthermore, integrating user feedback mechanisms could enable continuous learning and adaptation of the model. Developing a web-based or mobile interface for real-time spam filtering could enhance user experience and provide valuable interaction data for model improvement. Exploring ensemble methods that combine multiple classifiers might yield even better performance and robustness against sophisticated spam attacks. These future directions represent exciting opportunities for advancing the field of SMS spam detection and improving user communication experiences.